Financial Instruments Toolbox™
User's Guide

R2013**b**
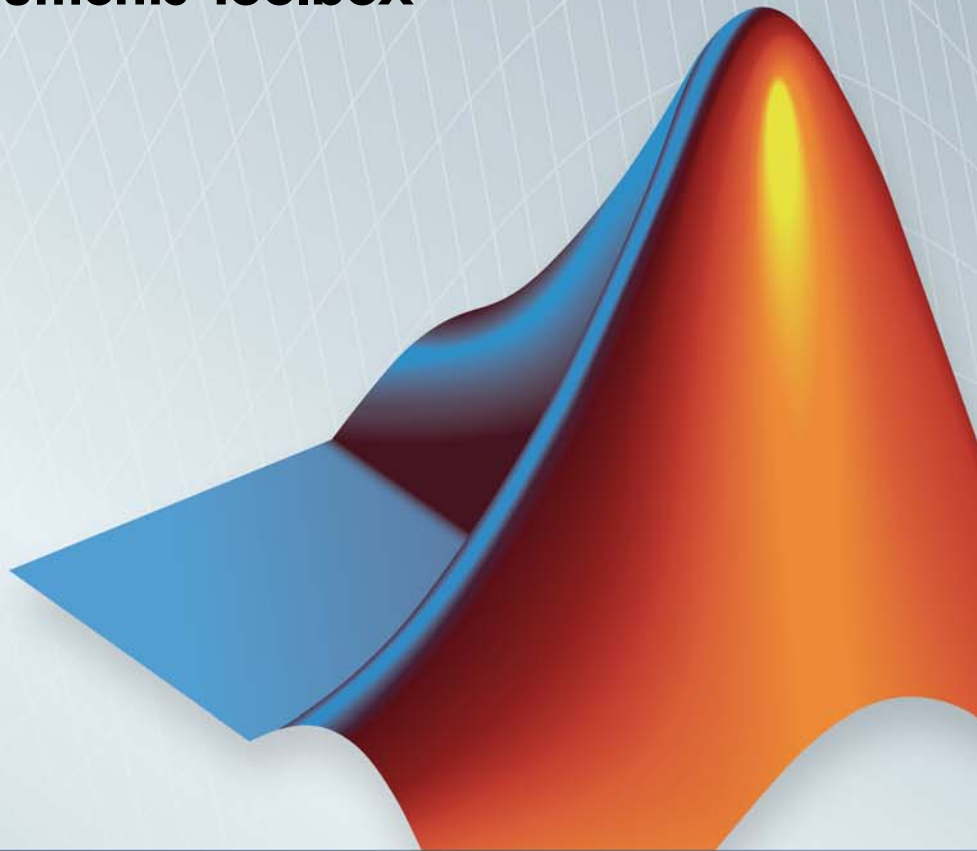
MATLAB®

MathWorks®

How to Contact MathWorks

www.mathworks.com	Web
comp.soft-sys.matlab	Newsgroup
www.mathworks.com/contact_TS.html	Technical Support

suggest@mathworks.com	Product enhancement suggestions
bugs@mathworks.com	Bug reports
doc@mathworks.com	Documentation error reports
service@mathworks.com	Order status, license renewals, passcodes
info@mathworks.com	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

Financial Instruments Toolbox™ User's Guide

Trademarks

Patents

Revision History

Contents

3

Hedging Portfolios

4

Mortgage-Backed Securities

5

Debt Instruments

6

Derivative Securities

7

Credit Derivatives

8

Interest-Rate Curve Objects

9

Numerix Workflows

10

Class Reference

A

Functions — Alphabetical List

11

Derivatives Pricing Options

B

Bibliography

C

<div align="right">

Glossary

</div>

<div align="right">

Index

</div>

1

Getting Started

Financial Instruments Toolbox Product Description

Design, price, and hedge complex financial instruments

Financial Instruments Toolbox™ provides functions for pricing, modeling, and analyzing fixed-income, credit, and equity instrument portfolios. You can use the toolbox to perform cash flow modeling and yield curve fitting analysis, compute prices and sensitivities, view price evolutions, and perform hedging analyses using common equity and fixed-income modeling methods. The toolbox lets you create new financial instrument types and fit yield curves to market data using parametric fitting models and bootstrapping.

Financial Instruments Toolbox includes functions for pricing and analyzing fixed-income and equity instruments. Fixed-income modeling tools let you calculate price, yield, spread, and sensitivity values for several types of securities and derivatives, including mortgage-backed securities, treasury bills, bonds, bonds with embedded options, swaps, caps, floors, and floating-rate notes. For equities, the toolbox lets you compute price, implied volatility, and greek values of vanilla equity options and of several exotic equity derivatives such as Bermuda, basket, barrier, digital, and rainbow options.

Key Features

- Yield curve fitting with bootstrapping and parametric fitting models, and term-structure analysis

- Black Scholes, Black, Garman-Kohlhagen, Roll-Geske-Whaley, Bjerksund-Stensland, Nengjiu Ju, Stulz, and Longstaff-Schwartz models

- Fixed-income and equity derivative calculations for price, yield, discount rate, cash-flow schedule, spread, implied volatility, option adjusted spread (OAS), and greeks

- Tree models: CRR, EQP, LR, ITT, HJM, BDT, BK, and HW

- Interest-rate instruments: bonds, stepped-coupon bonds, futures, vanilla options, Bermuda options, bonds with embedded options, vanilla swaps, forward swaps, amortizing swaps, swaptions, caps, floors, range notes, floating-rate notes, and collared floating-rate notes

- Equity instruments: stocks, vanilla options, Bermuda options, Asian options, lookback options, barrier options, digital options, rainbow options, basket options, compound options, and chooser options

- Credit instruments: mortgage pools, balloon mortgages, and credit default swaps

Interest-Rate-Based Derivatives

The toolbox provides functionality that supports the creation and management of these interest-rate-based instruments:

- Bonds
- Bond options (puts and calls)
- Bond with embedded options
- Caps
- Fixed-rate notes
- Floating-rate notes
- Floors
- Swaps
- Swaption

Additionally, the toolbox provides functions to create *arbitrary cash flow instruments*. The toolbox provides pricing and sensitivity routines for these instruments. For more information, see "Pricing Using Interest-Rate Term Structure" on page 2-39 ,"Pricing Using Interest-Rate Tree Models" on page 2-67, and"Interest-Rate Derivatives Using Closed-Form Solutions" on page 2-90.

Equity-Based Derivatives

The toolbox also provides functions to create and manage various equity-based derivatives, including the following:

- Asian options
- Barrier options
- Basket options
- Compound options
- Digital options
- Lookback options
- Rainbow options
- Vanilla stock options (put and call options)

The toolbox also provides pricing and sensitivity routines for these instruments. (See "Pricing Equity Derivatives Using Trees" on page 3-66, "Equity Derivatives Using Closed-Form Solutions" on page 3-85, and "Basket Option" on page 3-25.)

Expected Users

In general, this guide assumes experience working with financial derivatives and some familiarity with the underlying models.

In designing Financial Instruments Toolbox documentation, we assume your title is similar to one of these:

- Analyst, quantitative analyst
- Risk manager
- Portfolio manager
- Fund manager, asset manager
- Financial engineer
- Trader
- Student, professor, or other academic

We also assume your background, education, training, and responsibilities match some aspects of this profile:

- Finance, economics, perhaps accounting
- Engineering, mathematics, physics, other quantitative sciences
- Bachelor's degree minimum; MS or MBA likely; Ph.D. perhaps; CFA
- Comfortable with probability theory, statistics, and algebra
- Understand linear or matrix algebra, calculus, and differential equations
- Previously done traditional programming (C, Fortran, etc.)
- Responsible for instruments or analyses involving large sums of money
- Perhaps new to MATLAB®

Portfolio Creation

Introduction

The instadd function creates a set of instruments (portfolio) or adds instruments to an existing instrument collection. The TypeString argument specifies the type of the investment instrument. For interest-rate-based derivatives, the types are: Bond, OptBond, CashFlow, Fixed, Float, Cap, Floor, and Swap. For equity derivatives, the types are Asian, Barrier, Compound, Lookback, and OptStock.

The input arguments following TypeString are specific to the type of investment instrument. Thus, the TypeString argument determines how the remainder of the input arguments is interpreted. For example, instadd with the type string Bond creates a portfolio of bond instruments.

```
InstSet = instadd('Bond', CouponRate, Settle, Maturity, Period,
Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
StartDate, Face)
```

Interest-Rate-Based Derivatives

In addition to the bond instrument already described, the toolbox can create portfolios containing the following set of interest-rate-based derivatives:

- Bond option

  ```
  InstSet = instadd('OptBond', BondIndex, OptSpec, Strike, ExerciseDates, AmericanOpt)
  ```

- Arbitrary cash flow instrument

  ```
  InstSet = instadd('CashFlow', CFlowAmounts, CFlowDates, Settle, Basis)
  ```

- Fixed-rate note instrument

  ```
  InstSet = instadd('Fixed', CouponRate, Settle, Maturity, FixedReset, Basis, Principal)
  ```

- Floating-rate note instrument

  ```
  InstSet = instadd('Float', Spread, Settle, Maturity, FloatReset, Basis, Principal)
  ```

- Cap instrument

  ```
  InstSet = instadd('Cap', Strike, Settle, Maturity, CapReset, Basis, Principal)
  ```

- Floor instrument

  ```
  InstSet = instadd('Floor', Strike, Settle, Maturity, FloorReset, Basis, Principal)
  ```

- Swap instrument

  ```
  InstSet = instadd('Swap', LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType)
  ```

- Swaption instrument

  ```
  InstSet = instadd('Swaption', OptSpec, Strike, ExerciseDates, Spread, ...
  Settle, Maturity, AmericanOpt, SwapReset, Basis, Principal)
  ```

- Bond with embedded option instrument

  ```
  InstSet = instadd('OptEmBond', CouponRate, Settle, Maturity, OptSpec, Strike, ...
  ExerciseDates, 'AmericanOpt', AmericanOpt, 'Period', Period,'Basis', Basis, ...
  'EndMonthRule', EndMonthRule,'Face',Face,'IssueDate', IssueDate, 'FirstCouponDate', ...
  FirstCouponDate, 'LastCouponDate', LastCouponDate,'StartDate', StartDate)
  ```

Equity Derivatives

The toolbox can create portfolios containing the following set of equity derivatives:

- Asian instrument

  ```
  InstSet = instadd('Asian', OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, ...
  AvgType, AvgPrice, AvgDate)
  ```

- Barrier instrument

```
InstSet = instadd('Barrier', OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, ...
BarrierType, Barrier, Rebate)
```

- Compound instrument

```
InstSet = instadd('Compound', UOptSpec, UStrike, USettle, UExerciseDates, UAmericanOpt, ...
COptSpec, CStrike, CSettle, CExerciseDates, CAmericanOpt)
```

- Lookback instrument

```
InstSet = instadd('Lookback', OptSpec, Strike, Settle, ExerciseDates, AmericanOpt)
```

- Stock option instrument

```
InstSet = instadd('OptStock', OptSpec, Strike, Settle, Maturity, AmericanOpt)
```

Adding Instruments to an Existing Portfolio

To use the instadd function to add additional instruments to an existing instrument portfolio, provide the name of an existing portfolio as the first argument to the instadd function.

Consider, for example, a portfolio containing two cap instruments only:

```
Strike = [0.06; 0.07];
Settle = '08-Feb-2000';
Maturity = '15-Jan-2003';

Port_1 = instadd('Cap', Strike, Settle, Maturity);
```

These commands create a portfolio containing two cap instruments with the same settlement and maturity dates, but with different strikes. In general, the input arguments describing an instrument can be either a scalar, or a number of instruments (NumInst)-by-1 vector in which each element corresponds to an instrument. Using a scalar assigns the same value to all instruments passed in the call to instadd.

Use the instdisp command to display the contents of the instrument set:

```
instdisp(Port_1)

Index Type Strike Settle     Maturity    CapReset Basis Principal
```

```
1    Cap  0.06   08-Feb-2000 15-Jan-2003 1        0     100
2    Cap  0.07   08-Feb-2000 15-Jan-2003 1        0     100
```

Now add a single bond instrument to Port_1. The bond has a 4.0% coupon and the same settlement and maturity dates as the cap instruments.

```
CouponRate = 0.04;
Port_1 = instadd(Port_1, 'Bond', CouponRate, Settle, Maturity);
```

Use instdisp again to see the resulting instrument set:

```
instdisp(Port_1)


Index Type Strike Settle        Maturity    CapReset Basis Principal
1    Cap  0.06   08-Feb-2000   15-Jan-2003 1        0     100
2    Cap  0.07   08-Feb-2000   15-Jan-2003 1        0     100

Index Type CouponRate Settle        Maturity    Period Basis EndMonthRule IssueDate ... Face
3    Bond 0.04        08-Feb-2000   15-Jan-2003 2      0     1            NaN       ... 100
```

Pricing a Portfolio Using the Black-Derman-Toy Model

This example illustrates how the Financial Instruments Toolbox™ can be used to create a Black-Derman-Toy (BDT) tree and price a portfolio of instruments using the BDT model.

Create the Interest Rate Term Structure

The structure RateSpec is an interest rate term structure that defines the initial forward-rate specification from which the tree rates are derived. Use the information of annualized zero coupon rates in the table below to populate the RateSpec structure.

```
  From             To           Rate
01 Jan 2005    01 Jan 2006      0.0275
01 Jan 2005    01 Jan 2007      0.0312
01 Jan 2005    01 Jan 2008      0.0363
01 Jan 2005    01 Jan 2009      0.0415
01 Jan 2005    01 Jan 2010      0.0458


StartDates = ['01 Jan 2005'];

EndDates =   ['01 Jan 2006';
              '01 Jan 2007';
              '01 Jan 2008';
              '01 Jan 2009';
              '01 Jan 2010'];

ValuationDate = ['01 Jan 2005'];
Rates = [0.0275; 0.0312; 0.0363; 0.0415; 0.0458];
Compounding = 1;

RateSpec = intenvset('Compounding',Compounding,'StartDates', StartDates,...
                     'EndDates', EndDates, 'Rates', Rates,'ValuationDate',


RateSpec =

         FinObj: 'RateSpec'
    Compounding: 1
```

```
                  Disc: [5x1 double]
                 Rates: [5x1 double]
              EndTimes: [5x1 double]
            StartTimes: [5x1 double]
              EndDates: [5x1 double]
            StartDates: 732313
        ValuationDate: 732313
                 Basis: 0
          EndMonthRule: 1
```

Specify the Volatility Model

Create the structure VolSpec that specifies the volatility process with the following data.

```
Volatility = [0.005; 0.0055; 0.006; 0.0065; 0.007];
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Volatility)


BDTVolSpec =

               FinObj: 'BDTVolSpec'
        ValuationDate: 732313
              VolDates: [5x1 double]
              VolCurve: [5x1 double]
       VolInterpMethod: 'linear'
```

Specify the Time Structure of the Tree

The structure TimeSpec specifies the time structure for an interest rate tree. This structure defines the mapping between the observation times at each level of the tree and the corresponding dates.

```
Maturity = EndDates;
BDTTimeSpec = bdttimespec(ValuationDate, Maturity, Compounding)


BDTTimeSpec =
```

```
           FinObj: 'BDTTimeSpec'
   ValuationDate: 732313
        Maturity: [5x1 double]
     Compounding: 1
           Basis: 0
     EndMonthRule: 1
```

Create the BDT Tree

Use the previously computed values for RateSpec, VolSpec and TimeSpec to create the BDT tree.

```
BDTTree = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec)
```

```
BDTTree =

      FinObj: 'BDTFwdTree'
     VolSpec: [1x1 struct]
    TimeSpec: [1x1 struct]
    RateSpec: [1x1 struct]
        tObs: [0 1 2 3 4]
        dObs: [732313 732678 733043 733408 733774]
        TFwd: {[5x1 double]  [4x1 double]  [3x1 double]  [2x1 double]  [4]}
      CFlowT: {[5x1 double]  [4x1 double]  [3x1 double]  [2x1 double]  [5]}
     FwdTree: {1x5 cell}
```

Observe the Interest Rate Tree

Visualize the interest rate evolution along the tree by looking at the output structure BDTTree. BDTTree returns an inverse discount tree, which you can convert into an interest rate tree with the cvtree function.

```
BDTTreeR = cvtree(BDTTree);
```

Look at the upper branch and lower branch paths of the tree:

```
%Rate at root node:
RateRoot     = treepath(BDTTreeR.RateTree, [0])

%Rates along upper branch:
RatePathUp   = treepath(BDTTreeR.RateTree, [1 1 1 1])

%Rates along lower branch:
RatePathDown = treepath(BDTTreeR.RateTree, [2 2 2 2])


RateRoot =

    0.0275


RatePathUp =

    0.0275
    0.0347
    0.0460
    0.0560
    0.0612


RatePathDown =

    0.0275
    0.0351
    0.0472
    0.0585
    0.0653
```

You can also display a graphical representation of the tree to examine interactively the rates on the nodes of the tree until maturity. The function treeviewer displays the structure of the rate tree in the left pane. The tree visualization in the right pane is blank, but by selecting Diagram and clicking on the nodes you can examine the rates along the paths.

```
treeviewer(BDTTreeR)
```
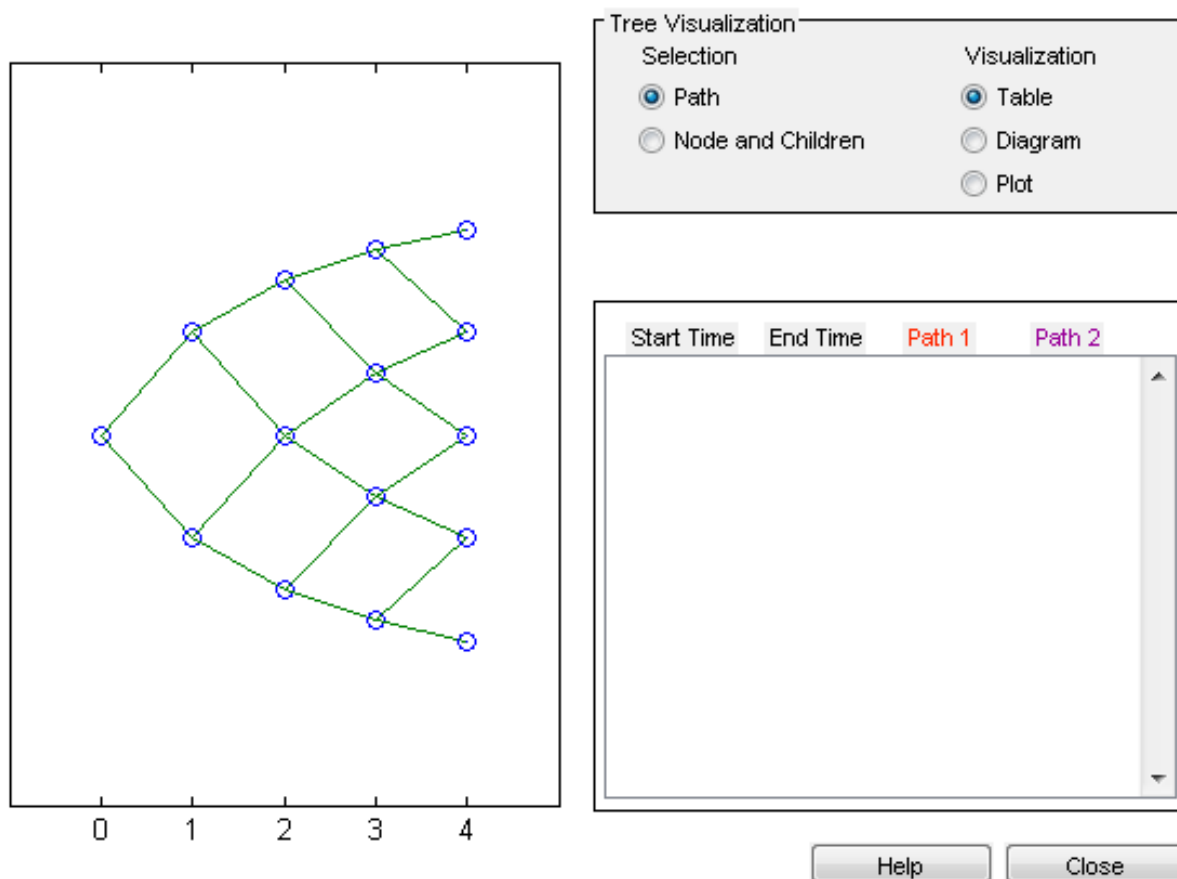
Create an Instrument Portfolio

Create a portfolio consisting of two bond instruments and a option on the 5% Bond.

```
% Bonds
CouponRate = [0.04;0.05];
Settle = '01 Jan 2005';
Maturity = ['01 Jan 2009';'01 Jan 2010'];
Period = 1;
```

```
% Option
OptSpec = {'call'};
Strike = 98;
ExerciseDates = ['01 Jan 2010'];
AmericanOpt = 1;

InstSet = instadd('Bond',CouponRate, Settle,  Maturity, Period);
InstSet = instadd(InstSet,'OptBond', 2, OptSpec, Strike, ExerciseDates, Ame
```

Examine the set of instruments contained in the variable InstSet.

```
instdisp(InstSet)

Index Type CouponRate Settle        Maturity      Period Basis EndMonthRu
1     Bond 0.04       01-Jan-2005   01-Jan-2009   1      0     1
2     Bond 0.05       01-Jan-2005   01-Jan-2010   1      0     1

Index Type    UnderInd OptSpec Strike ExerciseDates  AmericanOpt
3     OptBond 2        call    98     01-Jan-2010    1
```

Price the Portfolio Using a BDT Tree

Calculate the price of each instrument in the instrument set.

```
Price = bdtprice(BDTTree, InstSet)


Price =

  99.6374
 102.2460
   4.2460
```

The prices in the output vector Price correspond to the prices at observation time zero (tObs = 0), which is defined as the Valuation Date of the interest-rate tree.

In the Price vector, the first element, 99.6374, represents the price of the first instrument (4% Bond); the second element, 102.2460, represents the price of the second instrument (5% Bond), and 4.2460 represents the price of the Option.

Portfolio Management

Instrument Constructors

The toolbox provides constructors for the most common financial instruments. A *constructor* is a function that builds a structure dedicated to a certain type of object; in this toolbox, an *object* is a type of market instrument.

The instruments and their constructors in this toolbox are listed below.

Instrument	Constructor
Asian option	instasian
Barrier option	instbarrier
Bond	instbond
Bond option	instoptbnd
Arbitrary cash flow	instcf
Compound option	instcompound
Fixed-rate note	instfixed
Floating-rate note	instfloat
Cap	instcap
Floor	instfloor
Lookback option	instlookback
Stock option	instoptstock
Swap	instswap
Swaption	instswaption

Each instrument has parameters (fields) that describe the instrument. The toolbox functions let you do the following:

- Create an instrument or portfolio of instruments.
- Enumerate stored instrument types and information fields.
- Enumerate instrument field data.
- Search and select instruments.

The instrument structure consists of various fields according to instrument type. A *field* is an element of data associated with the instrument. For example, a bond instrument contains the fields `CouponRate`, `Settle`, `Maturity`, and so on. Additionally, each instrument has a field that identifies the investment type (bond, cap, floor, and so on).

In reality, the set of parameters for each instrument is not fixed. You have the ability to add additional parameters. These additional fields are ignored by the toolbox functions. They may be used to attach additional information to each instrument, such as an internal code describing the bond.

Parameters not specified when *creating* an instrument default to `NaN`, which, in general, means that the functions using the instrument set (such as `intenvprice` or `hjmprice`) will use default values. At the time of *pricing*, an error occurs if any of the required fields is missing, such as `Strike` in a cap or `CouponRate` in a bond.

Creating Instruments or Properties

Use the `instaddfield` function to create a kind of instrument or to add new properties to the instruments in an existing instrument collection.

To create a kind of instrument with `instaddfield`, you must specify three arguments:

- `Type`
- `FieldName`
- `Data`

`Type` defines the type of the new instrument, for example, `Future`. `FieldName` names the fields uniquely associated with the new type of instrument. `Data` contains the data for the fields of the new instrument.

An optional fourth argument is `ClassList`. `ClassList` specifies the data types of the contents of each unique field for the new instrument.

Use either syntax to create a kind of instrument using `instaddfield`:

```
InstSet = instaddfield('FieldName', FieldList, 'Data', DataList,...
'Type', TypeString)
InstSet = instaddfield('FieldName', FieldList, 'FieldClass',...
ClassList, 'Data' , DataList, 'Type', TypeString)
```

To add new instruments to an existing set, use:

```
InstSetNew = instaddfield(InstSetOld, 'FieldName', FieldList,...
'Data', DataList, 'Type', TypeString)
```

As an example, consider a futures contract with a delivery date of July 15, 2000, and a quoted price of \$104.40. Since Financial Instruments Toolbox software does not directly support this instrument, you must create it using the function `instaddfield`. Use these parameters to create instruments:

- Type: `Future`
- Field names: `Delivery` and `Price`
- Data: Delivery is July 15, 2000, and price is \$104.40.

Enter the data into MATLAB software:

```
Type = 'Future';
FieldName = {'Delivery', 'Price'};
Data = {'Jul-15-2000', 104.4};
```

Finally, create the portfolio with a single instrument:

```
Port = instaddfield('Type', Type, 'FieldName', FieldName,...
'Data', Data);
```

Now use the function `instdisp` to examine the resulting single-instrument portfolio:

```
instdisp(Port)

Index Type    Delivery    Price
1      Future Jul-15-2000 104.4
```

Because your portfolio `Port` has the same structure as those created using the function `instadd`, you can combine portfolios created using `instadd` with portfolios created using `instaddfield`. For example, you can now add two cap instruments to `Port` with `instadd`.

```
Strike = [0.06; 0.07];
Settle = '08-Feb-2000';
Maturity = '15-Jan-2003';

Port = instadd(Port, 'Cap', Strike, Settle, Maturity);
```

View the resulting portfolio using `instdisp`.

```
instdisp(Port)

Index   Type   Delivery      Price
1        Future 15-Jul-2000   104.4

Index Type Strike Settle      Maturity   CapReset Basis Principal
2     Cap  0.06   08-Feb-2000 15-Jan-2003 1        0     100
3     Cap  0.07   08-Feb-2000 15-Jan-2003 1        0     100
```

Searching or Subsetting a Portfolio

Financial Instruments Toolbox software provides functions that enable you to:

- Find specific instruments within a portfolio.

- Create a subset portfolio consisting of instruments selected from a larger portfolio.

The `instfind` function finds instruments with a specific parameter value; it returns an instrument index (position) in a large instrument set. The `instselect` function, on the other hand, subsets a large instrument set into

a portfolio of instruments with designated parameter values; it returns an instrument set (portfolio) rather than an index.

instfind

The general syntax for `instfind` is

```
IndexMatch = instfind(InstSet, 'FieldName', FieldList, 'Data',...
DataList, 'Index', IndexSet, 'Type', TypeList)
```

`InstSet` is the instrument set to search. Within `InstSet` instruments categorized by type, each type can have different data fields. The stored data field is a row vector or string for each instrument.

The `FieldList`, `DataList`, and `TypeList` arguments indicate values to search for in the `FieldName`, `Data`, and `Type` data fields of the instrument set. `FieldList` is a cell array of field name(s) specific to the instruments. `DataList` is a cell array or matrix of acceptable values for the parameter(s) specified in `FieldList`. `FieldName` and `Data` (consequently, `FieldList` and `DataList`) parameters must appear together or not at all.

`IndexSet` is a vector of integer index(es) designating positions of instruments in the instrument set to check for matches; the default is all indices available in the instrument set. `TypeList` is a string or cell array of strings restricting instruments to match one of the `TypeList` types; the default is all types in the instrument set.

`IndexMatch` is a vector of positions of instruments matching the input criteria. Instruments are returned in `IndexMatch` if all the `FieldName`, `Data`, `Index`, and `Type` conditions are met. An instrument meets an individual field condition if the stored `FieldName` data matches any of the rows listed in the `DataList` for that `FieldName`.

instfind Examples. The examples use the provided MAT-file `deriv.mat`.

The MAT-file contains an instrument set, `HJMInstSet`, that contains eight instruments of seven types.

```
load deriv.mat
instdisp(HJMInstSet)


Index Type CouponRate Settle      Maturity     Period Basis ... Name      Quantity
1    Bond 0.04       01-Jan-2000 01-Jan-2003   1      NaN   ... 4% bond    100
2    Bond 0.04       01-Jan-2000 01-Jan-2004   2      NaN   ... 4% bond    50


Index Type    UnderInd OptSpec Strike ExerciseDates  AmericanOpt Name       Quantity
3    OptBond 2        call    101    01-Jan-2003    NaN         Option 101 -50


Index Type  CouponRate Settle      Maturity     FixedReset Basis Principal Name     Quantity
4    Fixed 0.04       01-Jan-2000 01-Jan-2003   1          NaN   NaN       4% Fixed 80


Index Type  Spread Settle     Maturity   FloatReset  Basis Principal Name       Quantity
5    Float 20     01-Jan-2000 01-Jan-2003 1          NaN   NaN       20BP Float 8


Index Type Strike Settle      Maturity     CapReset Basis Principal Name     Quantity
6    Cap  0.03   01-Jan-2000  01-Jan-2004   1        NaN   NaN       3% Cap   30


Index Type  Strike Settle      Maturity    FloorReset Basis Principal Name      Quantity
7    Floor 0.03 01-Jan-2000 01-Jan-2004   1          NaN   NaN       3% Floor  40


Index Type LegRate    Settle      Maturity     LegReset Basis Principal LegType Name     Quantity

8    Swap [0.06 20] 01-Jan-2000  01-Jan-2003  [1  1]   NaN   NaN       [NaN]   6%/20BP Swap 10
```

Find all instruments with a maturity date of January 01, 2003.

```
Mat2003 = ...
instfind(HJMInstSet,'FieldName','Maturity','Data','01-Jan-2003')

Mat2003 =

    1
    4
    5
    8
```

Find all cap and floor instruments with a maturity date of January 01, 2004.

```
CapFloor = instfind(HJMInstSet,...
'FieldName','Maturity','Data','01-Jan-2004', 'Type',...
{'Cap';'Floor'})

CapFloor =

     6
     7
```

Find all instruments where the portfolio is long or short a quantity of 50.

```
Pos50 = instfind(HJMInstSet,'FieldName',...
'Quantity','Data',{'50';'-50'})

Pos50 =

     2
     3
```

instselect

The syntax for instselect is the same syntax as for instfind. instselect returns a full portfolio instead of indexes into the original portfolio. Compare the values returned by both functions by calling them equivalently.

Previously you used instfind to find all instruments in HJMInstSet with a maturity date of January 01, 2003.

```
Mat2003 = ...
instfind(HJMInstSet,'FieldName','Maturity','Data','01-Jan-2003')

Mat2003 =

     1
     4
     5
     8
```

Now use the same instrument set as a starting point, but execute the `instselect` function instead, to produce a new instrument set matching the identical search criteria.

```
Select2003 = ...
instselect(HJMInstSet,'FieldName','Maturity','Data',...
'01-Jan-2003')
```

```
instdisp(Select2003)
```

Index	Type	CouponRate	Settle	Maturity	Period	Basis	EndMonthRule	IssueDate	FirstCouponDate	Last
1	Bond	0.04	01-Jan-2000	01-Jan-2003	1	NaN	NaN	NaN	NaN	NaN

Index	Type	CouponRate	Settle	Maturity	FixedReset	Basis	Principal	Name	Quantity
2	Fixed	0.04	01-Jan-2000	01-Jan-2003	1	NaN	NaN	4% Fixed	80

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	Name	Quantity
3	Float	20	01-Jan-2000	01-Jan-2003	1	NaN	NaN	20BP Float	8

Index	Type	LegRate	Settle	Maturity	LegReset	Basis	Principal	LegType	Name	Quantity
4	Swap	[0.06 20]	01-Jan-2000	01-Jan-2003	[1 1]	NaN	NaN	[NaN]	6%/20BP Swap	10

instselect Examples. These examples use the portfolio `ExampleInst` provided with the MAT-file `InstSetExamples.mat`.

```
load InstSetExamples.mat
instdisp(ExampleInst)
```

Index	Type	Strike	Price	Opt	Contracts
1	Option	95	12.2	Call	0
2	Option	100	9.2	Call	0
3	Option	105	6.8	Call	1000

Index	Type	Delivery	F	Contracts
4	Futures	01-Jul-1999	104.4	-1000

Index	Type	Strike	Price	Opt	Contracts
5	Option	105	7.4	Put	-1000
6	Option	95	2.9	Put	0

Index	Type	Price	Maturity	Contracts

```
7      TBill 99    01-Jul-1999    6
```

The instrument set contains 3 instrument types: `Option`, `Futures`, and `TBill`. Use `instselect` to make a new instrument set containing only options struck at `95`. In other words, select all instruments containing the field `Strike` *and* with the data value for that field equal to `95`.

```
InstSet = instselect(ExampleInst,'FieldName','Strike','Data',95);
```

```
instdisp(InstSet)
```

```
Index Type    Strike Price Opt  Contracts
1     Option  95     12.2  Call    0
2     Option  95      2.9  Put     0
```

You can use all the various forms of `instselect` and `instfind` to locate specific instruments within this instrument set.

2

Interest-Rate Derivatives

Supported Interest-Rate Instruments

Bond

A *bond* is a long-term debt security with a preset interest-rate and maturity. At maturity you must pay the principal and interest.

The price or value of a bond is determined by discounting the expected cash flows of the bond to the present, using the appropriate discount rate. The following equation represents the relationship of the expected cash flows and discount rate:

$$B_0 = \frac{C}{2}\left[\frac{1-\left(1+\frac{r}{2}\right)^{-2t}}{\frac{r}{2}}\right] + \frac{F}{\left(1+\frac{r}{2}\right)^{2t}}$$

where:

B_0 is the bond value.

C is the annual coupon payment.

F is the face value of the bond.

r is the required return on the bond.

t is the number of years remaining until maturity.

Financial Instruments Toolbox supports the following for pricing and specifying a bond.

Function	Purpose
bondbybdt	Price a bond using a BDT interest-rate tree.
bondbyhw	Price a bond using an HW interest-rate tree.
bondbybk	Price a bond using a BK interest-rate tree.
bondbyhjm	Price a bond using an HJM interest-rate tree.
bondbyzero	Price a bond using a set of zero curves.
instbond	Construct a bond instrument.

Stepped Coupon Bonds

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond. For more information on options features (call and puts), see "Stepped Coupon Bonds with Calls and Puts" on page 2-7. The following functions have a modified CouponRate argument to support a new variable coupon schedule allowing pricing of stepped coupon bonds.

Function	Purpose
bondbyzero	Price bonds using a term structure model.
bondbybdt	Price bonds using a BDT tree model.
bondbyhjm	Price bonds using an HJM tree model.
bondbyhw	Price bonds using an HW tree model.
bondbybk	Price bonds using a BK tree model.
instbond	Construct a bond instrument.
instoptbnd	Construct a bond option instrument.
instdisp	Display instruments stored in a variable.

Sinking Fund Bonds

A sinking fund bond is a coupon bond with a sinking fund provision. This provision obligates the issuer to amortize portions of the principal before maturity, affecting bond prices since the time of the principal repayment changes. This means that investors receive the coupon and a portion of the principal paid back over time. These types of bonds reduce credit risk, since it lowers the probability of investors not receiving their principal payment at maturity. For more information on options support for sinking fund bonds, see "Sinking Fund Bonds with an Embedded Option" on page 2-8. The following functions have a modified Face argument to support a variable face schedule for pricing bonds with a sinking provisions.

Function	Purpose
bondbyzero	Price bonds using a term structure model.
bondbybdt	Price bonds using a BDT tree model.

Function	Purpose
bondbyhjm	Price bonds using an HJM tree model.
bondbyhw	Price bonds using an HW tree model.
bondbybk	Price bonds using a BK tree model.
instbond	Construct a bond instrument.
instoptbnd	Construct a bond option instrument.
instdisp	Display instruments stored in a variable.

Bonds with an Amortization Schedule

A bond with an amortization schedule repays part of the principal (face value) along with the coupon payments. An amortizing bond is a special case of a sinking fund bond when there is no market purchase option and no call provision.. The following functions have a modified Face argument to support an amortization schedule.

Function	Purpose
bondbyzero	Price bonds using a term structure model.
bondbybdt	Price bonds using a BDT tree model.
bondbyhjm	Price bonds using an HJM tree model.
bondbyhw	Price bonds using an HW tree model.
bondbybk	Price bonds using a BK tree model.

Bond Options

Financial Instruments Toolbox software supports three types of put and call options on bonds:

- American option: An option that you exercise any time until its expiration date.

- European option: An option that you exercise only on its expiration date.

- Bermuda option: A Bermuda option resembles a hybrid of American and European options. You can exercise it on predetermined dates only, usually monthly.

Financial Instruments Toolbox supports the following for pricing and specifying a bond option.

Function	Purpose
optbndbybdt	Price a bond option price using a BDT interest-rate tree.
optbndbyhw	Price a bond option price using an HW interest-rate tree.
optbndbybk	Price a bond option price using a BK interest-rate tree.
optbndbyhjm	Price a bond option price using an HJM interest-rate tree.
instoptbnd	Construct a bond option instrument.

Bond with Embedded Options

A bond with embedded options allows the issuer to buy back or redeem the bond at a predetermined price at specified future dates. Financial Instruments Toolbox software supports American, European, and Bermuda callable and puttable bonds.

The pricing for a bond with embedded options is as follows:

- For a callable bond: *PriceCallableBond = BondPrice - BondCallOption*

- For a puttable bond: *PricePuttableBond = PriceBond + PricePutOption*

Financial Instruments Toolbox supports the following for pricing and specifying a bond with embedded options.

Function	Purpose
optembndbybdt	Price a bond with embedded options using a BDT interest-rate tree.
optembndbyhw	Price a bond with embedded options using an HW interest rate tree.
optembndbybk	Price a bond with embedded options using a BK interest-rate tree.
optembndbyhjm	Price a bond with embedded options using an HJM interest-rate tree.
instoptembnd	Construct a bond-with-embedded-options instrument.

Stepped Coupon Bonds with Calls and Puts

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. For more information on stepped coupon bonds, see "Stepped Coupon Bonds" on page 2-4. Stepped coupon bonds can have options features (call and puts). The following functions have a modified CouponRate argument to support a new variable coupon schedule allowing pricing stepped coupon bonds with callable and puttable features:

Function	Purpose
optembndbybdt	Price bonds with embedded options using a BDT model tree.
optembndbyhjm	Price bonds with embedded options using an HJM model tree.
optembndbybk	Price bonds with embedded options using a BK model tree.
optembndbyhw	Price bonds with embedded options using an HW model tree.
instbond	Construct a bond instrument.
instoptbnd	Construct a bond option instrument.

Function	Purpose
instoptembnd	Construct a bond with an embedded option instrument.
instdisp	Display instruments stored in a variable.

Sinking Fund Bonds with an Embedded Option

A sinking fund bond is a coupon bond with a sinking fund provision. For more information on sinking fund bonds, see "Sinking Fund Bonds" on page 2-4. The sinking fund bond can have a sinking fund option provision allowing the issuer to retire the sinking fund obligation either by purchasing the bonds to be redeemed from the market or by calling the bond via a sinking fund call, whichever is cheaper.

If interest rates are high, then the issuer buys back the required amount of bonds from the market since bonds will be cheap. But if interest rates are low (bond prices are high), then most likely the issuer buys the bonds at the call price. Unlike a call feature, however, if a bond has a sinking fund option provision, it is an obligation, not an option, for the issuer to buy back the increments of the issue as stated. Because of this, a sinking fund bond trades at a lower price than a nonsinking fund bond. The following functions have a modified Face argument to support a variable face schedule for pricing bonds with a sinking fund option provision.

Function	Purpose
optembndbybdt	Price bonds with embedded options using a BDT model tree.
optembndbyhjm	Price bonds with embedded options using an HJM model tree.
optembndbybk	Price bonds with embedded options using a BK model tree.
optembndbyhw	Price bonds with embedded options using an HW model tree.
instbond	Construct a bond instrument.

Function	Purpose
instoptbnd	Construct a bond option instrument.
instdisp	Display instruments stored in a variable.

Fixed-Rate Note

A *fixed-rate note* is a long-term debt security with a preset interest rate and maturity, by which the interest must be paid. The principal may or may not be paid at maturity. In Financial Instruments Toolbox software, the principal is always paid at maturity.

Function	Purpose
fixedbybdt	Price a fixed-rate note using a BDT interest-rate tree.
fixedbyhw	Price a fixed-rate note using an HW interest-rate tree.
fixedbybk	Price a fixed-rate note using a BK interest-rate tree.
fixedbyhjm	Price a fixed-rate note using an HJM interest-rate tree.
fixedbyzero	Price a fixed-rate note using a set of zero curves.
instfixed	Construct a fixed-rate instrument.

Floating-Rate Note

A *floating-rate note* is a security like a bond, but the interest rate of the note is reset periodically, relative to a reference index rate, to reflect fluctuations in market interest rates.

Function	Purpose
floatbybdt	Price a floating-rate note using a BDT interest-rate tree.
floatbyhw	Price a floating-rate note using an HW interest-rate tree.
floatbybk	Price a floating-rate note using a BK interest-rate tree.
floatbyhjm	Price a floating-rate note using an HJM interest-rate tree.
floatbyzero	Price a floating-rate note using a set of zero curves.
instfloat	Construct a floating-rate note instrument.

Floating-Rate Note with an Amortization Schedule

A floating-rate note with an amortization schedule repays part of the principal (face value) along with the coupon payments. The following functions have a Principal argument to support an amortization schedule.

Function	Purpose
floatbyzero	Price floating-rate note from set of zero curves.
floatbybdt	Price floating-rate note from Black-Derman-Toy interest-rate tree.
floatbyhjm	Price floating-rate note from Heath-Jarrow-Morton interest-rate tree.
floatbyhw	Price floating-rate note from Hull-White interest-rate tree.
floatbybk	Price floating-rate note from Black-Karasinski interest-rate tree.

Floating-Rate Note with Caps, Collars, and Floors

A floating-rate note with caps, collars, and floors This type of instrument can carry restrictions on the maximum (cap) or minimum (floor) coupon rate are paid. A cap is an unattractive feature for an investor, since they constrain the coupon rates from increasing. A floor, on the other hand, is an attractive feature, since it allows investors to get a minimum coupon rate when market rates decrease below a certain level. Also, a floating-rate note can have a collar which is a combination of a cap and a floor together. The following functions have a `CapRate` and `FloorRate` argument to support a capped, collared, or floored floating-rate note.

Function	Purpose
floatbybdt	Price a capped floating-rate note from a Black-Derman-Toy interest-rate tree.
floatbyhjm	Price a capped floating-rate note from a Heath-Jarrow-Morton interest-rate tree.
floatbyhw	Price a capped floating-rate note from a Hull-White interest-rate tree.
floatbybk	Price a capped floating-rate note from a Black-Karasinskiinterest-rate tree.
instfloat	Create a capped floating-rate note instrument.
instadd	Add a capped floating-rate note instrument to a portfolio.

Floating-Rate Note with Options

A floating-rate note with options. Just like in the bond market, floaters can have early redemption features. They give investors or issuers the option to retire the outstanding principal prior to maturity. The call option gives the right to retire the note prior to the maturity date (callable floater), and the put option gives the right to sell the note back at a specific price (puttable floater).

Function	Purpose
`optfloatbybdt`	Price an option for floating-rate note using a Black-Derman-Toy interest-rate tree.
`optfloatbyhjm`	Price an option for floating-rate note using a Heath-Jarrow-Morton interest-rate tree.
`optfloatbyhw`	Price an option for floating-rate note using a Hull-White interest-rate tree.
`optfloatbybk`	Price an option for floating-rate note using a Black-Karasinski interest-rate tree.
`instoptfloat`	Define the option instrument for floating-rate note.

Floating-Rate Note with Embedded Options

A floating-rate note with embedded options TBD

Function	Purpose
`optemfloatbybdt`	Price an embedded option for floating-rate note using a Black-Derman-Toy interest-rate tree.
`optemfloatbybk`	Price an embedded option for floating-rate note using a Black-Karasinski interest-rate tree.
`optemfloatbyhjm`	Price an embedded option for floating-rate note using a Heath-Jarrow-Morton interest-rate tree.
`optemfloatbyhw`	Price an embedded option for floating-rate note using a Hull-White interest-rate tree.
`instoptemfloat`	Define the floating-rate note with embedded option instrument.

Cap

A *cap* is a contract that includes a guarantee that sets the maximum interest rate to be paid by the holder, based on an otherwise floating interest rate. The payoff for a cap is:

$$\max(CurrentRate - CapRate, 0)$$

Function	Purpose
capbybdt	Price a cap instrument using a BDT interest-rate tree.
capbyhw	Price a cap instrument using an HW interest-rate tree.
capbybk	Price a cap instrument using a BK interest-rate tree.
capbyhjm	Price a cap instrument using an HJM interest-rate tree.
capbyblk	Price a cap instrument using the Black option pricing model.
instcap	Construct a cap instrument.

Floor

A *floor* is a contract that includes a guarantee setting the minimum interest rate to be received by the holder, based on an otherwise floating interest rate. The payoff for a floor is:

$$\max(FloorRate - CurrentRate, 0)$$

Function	Purpose
floorbybdt	Price a floor instrument using a BDT interest-rate tree.
floorbyhw	Price a floor instrument using an HW interest-rate tree.
floorbybk	Price a floor instrument using a BK interest-rate tree.
floorbyhjm	Price a floor instrument using an HJM interest-rate tree.
instfloor	Construct a floor instrument.

Range Note

A *range note* is a structured (market-linked) security whose coupon-rate is equal to the reference rate as long as the reference rate is within a certain range. If the reference rate is outside of the range, the coupon-rate is 0 for that period. This type of instrument entitles the holder to cash flows that depend on the level of some reference interest-rate that are floored to be positive and gives the holder of the note direct exposure to the reference rate. This type of instrument is useful for cases where you believe that interest rates will stay within a certain range. In return for the drawback that no interest will be paid for the time the range is left, a range note offers higher coupon rates than comparable standard products, like vanilla floating notes.

Function	Purpose
instrangefloat	Create a range note instrument.
rangefloatbybdt	Price range floating note using a BDT tree.
rangefloatbybk	Price range floating note using a BK tree.
rangefloatbyhjm	Price range floating note using an HJM tree.
rangefloatbyhw	Price range floating note using an HW tree.

Swap

A *swap* is contract between two parties obligating the parties to exchange future cash flows. This toolbox version handles only the vanilla swap, which is composed of a floating-rate leg and a fixed-rate leg.

Function	Purpose
swapbybdt	Price a swap instrument using a BDT interest-rate tree.
swapbyhw	Price a swap instrument using an HW interest-rate tree.
swapbybk	Price a swap instrument using a BK interest-rate tree.
swapbyhjm	Price a swap instrument using an HJM interest-rate tree.

Function	Purpose
swapbyzero	Price a swap instrument using a set of zero curves.
instswap	Construct a swap instrument.

Swap with an Amortization Schedule

A swap with an amortization schedule repays part of the principal (face value) along with the coupon payments. A swap with an amortization schedule is used to manage interest rate risk and serve as a cash flow management tool. For this particular type of swap, the notional amount decreases over time. This means that interest payments will decrease not only on the floating leg but also on the fixed leg. The following swap functions have a `Principal` argument to support an amortization schedule.

Function	Purpose
swapbyzero	Price swap instrument from set of zero curves.
swapbybdt	Price swap instrument from Black-Derman-Toy interest-rate tree.
swapbyhjm	Price swap instrument from Heath-Jarrow-Morton interest-rate tree.
swapbyhw	Price swap instrument from Hull-White interest-rate tree.
swapbybk	Price swap instrument from Black-Karasinski interest-rate tree.
instswap	Construct swap instrument.

Forward Swap

In a forward interest-rate swap, a fixed interest rate loan is exchanged for a floating interest rate loan at a future specified date The following functions have a `StartDate` argument to support the future date for the forward swap.

Function	Purpose
swapbyzero	Price a forward swap from a zero curve.
swapbybdt	Price a forward swap from a Black-Derman-Toy interest-rate tree.
swapbyhjm	Price a forward swap from a Heath-Jarrow-Morton interest-rate tree.
swapbyhw	Price a forward swap from a Hull-White interest-rate tree.
swapbybk	Price a forward swap from a Black-Karasinskiinterest-rate tree.
instswap	Create a forward swap instrument.
instadd	Add a capped floating-rate note instrument to a portfolio.

Swaption

A swaption is an option to enter into an interest-rate swap contract. A call swaption allows the option buyer to enter into an interest-rate swap where the buyer of the option pays the fixed-rate and receives the floating-rate. A put swaption allows the option buyer to enter into an interest-rate swap where the buyer of the option receives the fixed-rate and pays the floating-rate.

Function	Purpose
swaptionbybdt	Price a swaption instrument using a BDT interest-rate tree.
swaptionbyhw	Price a swaption instrument using an HW interest-rate tree.
swaptionbybk	Price a swaption instrument using a BK interest-rate tree.
swaptionbyhjm	Price a swaption instrument using an HJM interest-rate tree.

Function	Purpose
swaptionbyblk	Price swaptions using the Black model with a forward on a swap.
instswaption	Construct a swaption instrument.

Use swaptionbyblk to price a swaption using the Black model. The Black model is standard model used in the swaption market when pricing European swaptions. This type of model is widely used by when speed is important to quickly obtain a price at settlement date, even if the price is less accurate than other swaption pricing models based on interest-rate tree models.

Bond Futures

Bond futures are futures contracts where the commodity for delivery is a government bond. There are established global markets for government bond futures. Bond futures provide a liquid alternative for managing interest-rate risk.

In the U.S. market, the Chicago Mercantile Exchange (CME) offers futures on Treasury bonds and notes with maturities of 2, 5, 10, and 30 years. Typically, the following bond future contracts from the CME have maturities of 3, 6, 9, and 12 months:

- 30-year U.S. Treasury bond

- 10-year U.S. Treasury bond

- 5-year U.S. Treasury bond

- 2-year U.S. Treasury bond

The short position in a Treasury bond or note future contract must deliver to the long position in one of many possible existing Treasury bonds. For example, in a 30-year Treasury bond future, the short position must deliver a Treasury bond with at least 15 years to maturity. Because these bonds have different values, the bond future contract is standardized by computing a conversion factor. The conversion factor normalizes the price of a bond to a theoretical bond with a coupon of 6%. The price of a bond future contract is represented as:

$InvoicePrice = FutPrice \times CF + AI$

where:

FutPrice is the price of the bond future.

CF is the conversion factor for a bond to deliver in a futures contract.

AI is the accrued interest.

You can reference these conversion factors at U.S. Treasury Bond Futures Contract. The short position in a futures contract has the option of which bond to deliver and, in the U.S. bond market, when in the delivery month to deliver the bond. The short position typically chooses to deliver the bond known as the Cheapest to Deliver (CTD). The CTD bond most often delivers on the last delivery day of the month.

Financial Instruments Toolbox software supports the following bond futures:

- U.S. Treasury bonds and notes
- German Bobl, Bund, Buxl, and Schatz
- UK gilts
- Japanese government bonds (JGBs)

The functions supporting all bond futures are:

Function	Purpose
convfactor	Calculates bond conversion factors for U.S. Treasury bonds, German Bobl, Bund, Buxl, and Schatz, U.K. gilts, and JGBs.
bndfutprice	Prices bond future given repo rates.
bndfutimprepo	Calculates implied repo rates for a bond future given price.

The functions supporting U.S. Treasury bond futures are:

Function	Purpose
tfutbyprice	Calculates future prices of Treasury bonds given the spot price.
tfutbyyield	Calculates future prices of Treasury bonds given current yield.
tfutimprepo	Calculates implied repo rates for the Treasury bond future given price.
tfutpricebyrepo	Calculates Treasury bond futures price given the implied repo rates.
tfutyieldbyrepo	Calculates Treasury bond futures yield given the implied repo rates.

For more information on bond futures, see "Supported Bond Futures" on page 7-12.

Overview of Interest-Rate Tree Models

In this section...
"Interest-Rate Modeling" on page 2-20
"Rate and Price Trees" on page 2-21
"Viewing Rate or Price Movement" on page 2-22

Interest-Rate Modeling

Financial Instruments Toolbox software computes prices and sensitivities of interest-rate contingent claims based on several methods of modeling changes in interest rates over time:

- The interest-rate term structure

  This model uses sets of zero-coupon bonds to predict changes in interest rates.

- Heath-Jarrow-Morton (HJM) model

  The HJM model considers a given initial term structure of interest rates and a specification of the volatility of forward rates to build a tree representing the evolution of the interest rates, based on a statistical process.

- Black-Derman-Toy (BDT) model

  In the BDT model, all security prices and rates depend on the short rate (annualized 1-period interest rate). The model uses long rates and their volatilities to construct a tree of possible future short rates. The resulting tree can then be used to determine the value of interest-rate sensitive securities from this tree.

- Hull-White (HW) model

  The Hull-White model incorporates the initial term structure of interest rates and the volatility term structure to build a trinomial recombining tree of short rates. The resulting tree is used to value interest-rate dependent securities. The implementation of the HW model in Financial Instruments Toolbox software is limited to one factor.

- Black-Karasinski (BK) model

The BK model is a single-factor, log-normal version of the HW model.

For detailed information about interest-rate models, see:

- "Pricing Using Interest-Rate Term Structure" on page 2-39 for a discussion of price and sensitivity based on portfolios of zero-coupon bonds

- "Pricing Using Interest-Rate Tree Models" on page 2-67 for a discussion of price and sensitivity based on the HJM and BDT interest-rate models

---

Note Historically, the initial version of Financial Instruments Toolbox software provided only the HJM interest-rate model. A later version added the BDT model. The current version adds both the HW and BK models. This section provides extensive examples of using the HJM and BDT models to compute prices and sensitivities of interest-rate based financial derivatives.

The HW and BK tree structures are similar to the BDT tree structure. To avoid needless repetition throughout this section, documentation is provided only where significant deviations from the BDT structure exist. Specifically, "HW and BK Tree Structures" on page 2-63 explains the few noteworthy differences among the various formats.

Rate and Price Trees

The interest-rate or price trees supported in this toolbox can be either *binomial* (two branches per node) or *trinomial* (three branches per node). Typically, binomial trees assume that underlying interest rates or prices can only either increase or decrease at each node. Trinomial trees allow for a more complex movement of rates or prices. With trinomial trees the movement of rates or prices at each node is unrestricted (for example, up-up-up or unchanged-down-down).

Types of Trees

Financial Instruments Toolbox trees can be classified as *bushy* or *recombining*. A bushy tree is a tree in which the number of branches increases exponentially relative to observation times; branches never recombine. In this context, a recombining tree is the opposite of a bushy tree. A recombining tree has

branches that recombine over time. From any given node, the node reached by taking the path up-down is the same node reached by taking the path down-up. A bushy tree and a recombining binomial tree are illustrated next.
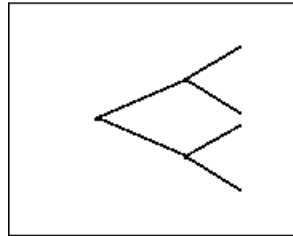
Bushy Tree
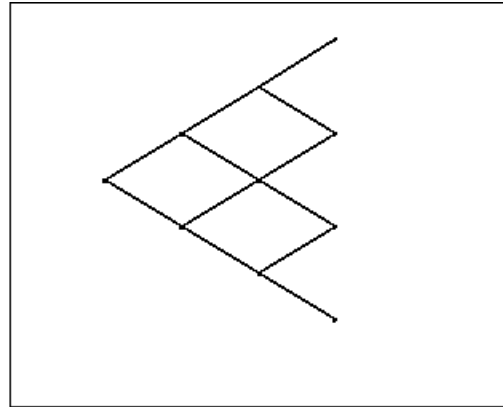
Recombining Binomial Tree

In this toolbox the Heath-Jarrow-Morton model works with bushy trees. The Black-Derman-Toy model, on the other hand, works with recombining binomial trees.

The other two interest rate models supported in this toolbox, Hull-White and Black-Karasinski, work with recombining trinomial trees.

Viewing Rate or Price Movement

This toolbox provides the data file `deriv.mat` that contains four interest-rate based trees:

- `HJMTree` — A bushy binomial tree
- `BDTTree` — A recombining binomial tree
- `HWTree` and `BKTree` — Recombining trinomial trees

The toolbox also provides the `treeviewer` function, which graphically displays the shape and data of price, interest rate, and cash flow trees. Viewed with `treeviewer`, the bushy shape of an HJM tree and the recombining shape of a BDT tree are apparent.
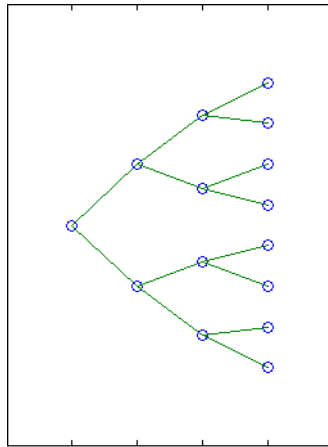
HJMTree (bushy)
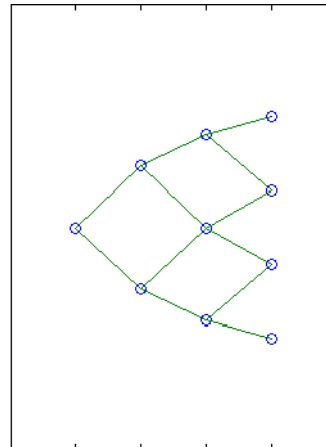
BDTTree (recombining)

With `treeviewer`, you can also see the recombining shape of HW and BK trinomial trees.
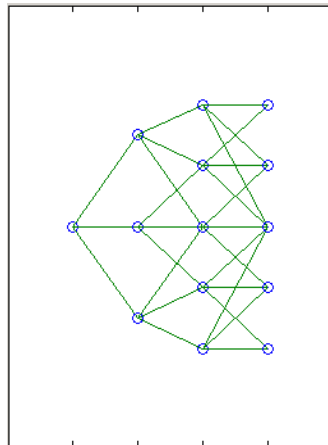
HWTree and BKTree (recombining)

Understanding the Interest-Rate Term Structure

In this section...

Introduction

The *interest-rate term structure* represents the evolution of interest rates through time. In MATLAB software, the interest-rate environment is encapsulated in a structure called `RateSpec` (*rate specification*). This structure holds all information required to completely identify the evolution of interest rates. Several functions included in Financial Instruments Toolbox software are dedicated to the creating and managing of the `RateSpec` structure. Many others take this structure as an input argument representing the evolution of interest rates.

Before looking further at the `RateSpec` structure, examine three functions that provide key functionality for working with interest rates: `disc2rate`, its opposite, `rate2disc`, and `ratetimes`. The first two functions map between discount factors and interest rates. The third function, `ratetimes`, calculates the effect of term changes on the interest rates.

Interest Rates Versus Discount Factors

Discount factors are coefficients commonly used to find the current value of future cash flows. As such, there is a direct mapping between the rate applicable to a period of time, and the corresponding discount factor. The function `disc2rate` converts discount factors for a given term (period) into interest rates. The function `rate2disc` does the opposite; it converts interest rates applicable to a given term (period) into the corresponding discount factors.

Calculating Discount Factors from Rates

As an example, consider these annualized zero-coupon bond rates.

From	To	Rate
15 Feb 2000	15 Aug 2000	0.05
15 Feb 2000	15 Feb 2001	0.056
15 Feb 2000	15 Aug 2001	0.06
15 Feb 2000	15 Feb 2002	0.065
15 Feb 2000	15 Aug 2002	0.075

To calculate the discount factors corresponding to these interest rates, call `rate2disc` using the syntax

```
Disc = rate2disc(Compounding, Rates, EndDates, StartDates,
ValuationDate)
```

where:

- `Compounding` represents the frequency at which the zero rates are compounded when annualized. For this example, assume this value to be 2.

- `Rates` is a vector of annualized percentage rates representing the interest rate applicable to each time interval.

- `EndDates` is a vector of dates representing the end of each interest-rate term (period).

- `StartDates` is a vector of dates representing the beginning of each interest-rate term.

- `ValuationDate` is the date of observation for which the discount factors are calculated. In this particular example, use February 15, 2000 as the beginning date for all interest-rate terms.

Next, set the variables in MATLAB.

```
StartDates = ['15-Feb-2000'];
EndDates  = ['15-Aug-2000'; '15-Feb-2001'; '15-Aug-2001';...
'15-Feb-2002'; '15-Aug-2002'];
Compounding = 2;
ValuationDate = ['15-Feb-2000'];
```

```
Rates = [0.05; 0.056; 0.06; 0.065; 0.075];
```

Finally, compute the discount factors.

```
Disc = rate2disc(Compounding, Rates, EndDates, StartDates,...
ValuationDate)

Disc =

    0.9756
    0.9463
    0.9151
    0.8799
    0.8319
```

By adding a fourth column to the rates table (see "Calculating Discount Factors from Rates" on page 2-24) to include the corresponding discounts, you can see the evolution of the discount factors.

From	To	Rate	Discount
15 Feb 2000	15 Aug 2000	0.05	0.9756
15 Feb 2000	15 Feb 2001	0.056	0.9463
15 Feb 2000	15 Aug 2001	0.06	0.9151
15 Feb 2000	15 Feb 2002	0.065	0.8799
15 Feb 2000	15 Aug 2002	0.075	0.8319

Optional Time Factor Outputs

The function rate2disc optionally returns two additional output arguments: EndTimes and StartTimes. These vectors of time factors represent the start dates and end dates in discount periodic units. The scale of these units is determined by the value of the input variable Compounding.

To examine the time factor outputs, find the corresponding values in the previous example.

```
[Disc, EndTimes, StartTimes] = rate2disc(Compounding, Rates,...
```

```
EndDates, StartDates, ValuationDate);
```

Arrange the two vectors into a single array for easier visualization.

```
Times = [StartTimes, EndTimes]

Times =

     0     1
     0     2
     0     3
     0     4
     0     5
```

Because the valuation date is equal to the start date for all periods, the `StartTimes` vector is composed of 0s. Also, since the value of `Compounding` is 2, the rates are compounded semiannually, which sets the units of periodic discount to 6 months. The vector `EndDates` is composed of dates separated by intervals of 6 months from the valuation date. This explains why the `EndTimes` vector is a progression of integers from 1 to 5.

Alternative Syntax (rate2disc)

The function `rate2disc` also accommodates an alternative syntax that uses periodic discount units instead of dates. Since the relationship between discount factors and interest rates is based on time periods and not on absolute dates, this form of `rate2disc` allows you to work directly with time periods. In this mode, the valuation date corresponds to 0, and the vectors `StartTimes` and `EndTimes` are used as input arguments instead of their date equivalents, `StartDates` and `EndDates`. This syntax for `rate2disc` is:

```
Disc = rate2disc(Compounding, Rates, EndTimes, StartTimes)
```

Using as input the `StartTimes` and `EndTimes` vectors computed previously, you should obtain the previous results for the discount factors.

```
Disc = rate2disc(Compounding, Rates, EndTimes, StartTimes)

Disc =

    0.9756
```

```
        0.9463
        0.9151
        0.8799
        0.8319
```

Calculating Rates from Discounts

The function `disc2rate` is the complement to `rate2disc`. It finds the rates applicable to a set of compounding periods, given the discount factor in those periods. The syntax for calling this function is:

```
Rates = disc2rate(Compounding, Disc, EndDates, StartDates,
ValuationDate)
```

Each argument to this function has the same meaning as in `rate2disc`. Use the results found in the previous example to return the rate values you started with.

```
Rates = disc2rate(Compounding, Disc, EndDates, StartDates,...
ValuationDate)

Rates =

    0.0500
    0.0560
    0.0600
    0.0650
    0.0750
```

Alternative Syntax (disc2rate)

As in the case of `rate2disc`, `disc2rate` optionally returns `StartTimes` and `EndTimes` vectors representing the start and end times measured in discount periodic units. Again, working with the same values as before, you should obtain the same numbers.

```
[Rates, EndTimes, StartTimes] = disc2rate(Compounding, Disc,...
EndDates, StartDates, ValuationDate);
```

Arrange the results in a matrix convenient to display.

```
Result = [StartTimes, EndTimes, Rates]

Result =

         0    1.0000    0.0500
         0    2.0000    0.0560
         0    3.0000    0.0600
         0    4.0000    0.0650
         0    5.0000    0.0750
```

As with `rate2disc`, the relationship between rates and discount factors is determined by time periods and not by absolute dates. Consequently, the alternate syntax for `disc2rate` uses time vectors instead of dates, and it assumes that the valuation date corresponds to time = 0. The time-based calling syntax is:

```
Rates = disc2rate(Compounding, Disc, EndTimes, StartTimes);
```

Using this syntax, you again obtain the original values for the interest rates.

```
Rates = disc2rate(Compounding, Disc, EndTimes, StartTimes)

Rates =

    0.0500
    0.0560
    0.0600
    0.0650
    0.0750
```

Interest-Rate Term Conversions

Interest rate evolution is typically represented by a set of interest rates, including the beginning and end of the periods the rates apply to. For zero rates, the start dates are typically at the valuation date, with the rates extending from that valuation date until their respective maturity dates.

Spot Curve to Forward Curve Conversion

Frequently, given a set of rates including their start and end dates, you may be interested in finding the rates applicable to different terms (periods). This

problem is addressed by the function `ratetimes`. This function interpolates the interest rates given a change in the original terms. The syntax for calling `ratetimes` is

```
[Rates, EndTimes, StartTimes] = ratetimes(Compounding, RefRates,
RefEndDates, RefStartDates, EndDates, StartDates, ValuationDate);
```

where:

- `Compounding` represents the frequency at which the zero rates are compounded when annualized.

- `RefRates` is a vector of initial interest rates representing the interest rates applicable to the initial time intervals.

- `RefEndDates` is a vector of dates representing the end of the interest rate terms (period) applicable to `RefRates`.

- `RefStartDates` is a vector of dates representing the beginning of the interest rate terms applicable to `RefRates`.

- `EndDates` represent the maturity dates for which the interest rates are interpolated.

- `StartDates` represent the starting dates for which the interest rates are interpolated.

- `ValuationDate` is the date of observation, from which the `StartTimes` and `EndTimes` are calculated. This date represents time = 0.

The input arguments to this function can be separated into two groups:

- The initial or reference interest rates, including the terms for which they are valid

- Terms for which the new interest rates are calculated

As an example, consider the rate table specified in "Calculating Discount Factors from Rates" on page 2-24.

From	To	Rate
15 Feb 2000	15 Aug 2000	0.05
15 Feb 2000	15 Feb 2001	0.056

From	To	Rate
15 Feb 2000	15 Aug 2001	0.06
15 Feb 2000	15 Feb 2002	0.065
15 Feb 2000	15 Aug 2002	0.075

Assuming that the valuation date is February 15, 2000, these rates represent zero-coupon bond rates with maturities specified in the second column. Use the function `ratetimes` to calculate the forward rates at the beginning of all periods implied in the table. Assume a compounding value of 2.

```
% Reference Rates.
RefStartDates = ['15-Feb-2000'];
RefEndDates  = ['15-Aug-2000'; '15-Feb-2001'; '15-Aug-2001';...
'15-Feb-2002'; '15-Aug-2002'];
Compounding = 2;
ValuationDate = ['15-Feb-2000'];
RefRates = [0.05; 0.056; 0.06; 0.065; 0.075];

% New Terms.
StartDates = ['15-Feb-2000'; '15-Aug-2000'; '15-Feb-2001';...
'15-Aug-2001'; '15-Feb-2002'];
EndDates =   ['15-Aug-2000'; '15-Feb-2001'; '15-Aug-2001';...
'15-Feb-2002'; '15-Aug-2002'];
% Find the new rates.
Rates = ratetimes(Compounding, RefRates, RefEndDates,...
RefStartDates, EndDates, StartDates, ValuationDate)

Rates =

    0.0500
    0.0620
    0.0680
    0.0801
    0.1155
```

Place these values in a table like the previous one. Observe the evolution of the forward rates based on the initial zero-coupon rates.

From	To	Rate
15 Feb 2000	15 Aug 2000	0.0500
15 Aug 2000	15 Feb 2001	0.0620
15 Feb 2001	15 Aug 2001	0.0680
15 Aug 2001	15 Feb 2002	0.0801
15 Feb 2002	15 Aug 2002	0.1155

Alternative Syntax (ratetimes)

The `ratetimes` function can provide the additional output arguments `StartTimes` and `EndTimes`, which represent the time factor equivalents to the `StartDates` and `EndDates` vectors. The `ratetimes` function uses time factors for interpolating the rates. These time factors are calculated from the start and end dates, and the valuation date, which are passed as input arguments. `ratetimes` can also use time factors directly, assuming time = 0 as the valuation date. This alternate syntax is:

```
[Rates, EndTimes, StartTimes] = ratetimes(Compounding, RefRates,
RefEndTimes, RefStartTimes, EndTimes, StartTimes);
```

Use this alternate version of `ratetimes` to find the forward rates again. In this case, you must first find the time factors of the reference curve. Use `date2time` for this.

```
RefEndTimes = date2time(ValuationDate, RefEndDates, Compounding)

RefEndTimes =

    1
    2
    3
    4
    5

RefStartTimes = date2time(ValuationDate, RefStartDates,...
Compounding)
```

```
RefStartTimes =

     0
```

These are the expected values, given semiannual discounts (as denoted by a value of 2 in the variable `Compounding`), end dates separated by 6-month periods, and the valuation date equal to the date marking beginning of the first period (time factor = 0).

Now call `ratetimes` with the alternate syntax.

```
[Rates, EndTimes, StartTimes] = ratetimes(Compounding,...
RefRates, RefEndTimes, RefStartTimes, EndTimes, StartTimes);

Rates =

    0.0500
    0.0620
    0.0680
    0.0801
    0.1155
```

`EndTimes` and `StartTimes` have, as expected, the same values they had as input arguments.

```
Times = [StartTimes, EndTimes]

Times =

     0     1
     1     2
     2     3
     3     4
     4     5
```

Modeling the Interest-Rate Term Structure

Financial Instruments Toolbox software includes a set of functions to encapsulate interest-rate term information into a single structure. These functions present a convenient way to package all information related to

interest-rate terms into a common format, and to resolve interdependencies when one or more of the parameters is modified. For information, see:

- "Creating or Modifying (intenvset)" on page 2-34 for a discussion of how to create or modify an interest-rate term structure (`RateSpec`) using the `intenvset` function
- "Obtaining Specific Properties (intenvget)" on page 2-36 for a discussion of how to extract specific properties from a `RateSpec`

Creating or Modifying (intenvset)

The main function to create or modify an interest-rate term structure `RateSpec` (rates specification) is `intenvset`. If the first argument to this function is a previously created `RateSpec`, the function modifies the existing rate specification and returns a new one. Otherwise, it creates a `RateSpec`.

When using `RateSpec` to specify the rate term structure to price instruments based on yields (zero coupon rates) or forward rates, specify zero rates or forward rates as the input argument. However, the `RateSpec` structure is not limited or specific to this problem domain. `RateSpec` is an encapsulation of rates-times relationships; `intenvset` acts as either a constructor or a modifier, and `intenvget` as an accessor. The interest rate models supported by the Financial Instruments Toolbox software work either with zero coupon rates or forward rates.

The other `intenvset` arguments are property-value pairs, indicating the new value for these properties. The properties that can be specified or modified are:

- `Basis`
- `Compounding`
- `Disc`
- `EndDates`
- `EndMonthRule`
- `Rates`
- `StartDates`
- `ValuationDate`

To learn about the properties `EndMonthRule` and `Basis`, type `help ftbEndMonthRule` and `help ftbBasis` or see the Financial Toolbox™ documentation.

Consider again the original table of interest rates (see "Calculating Discount Factors from Rates" on page 2-24).

From	To	Rate
15 Feb 2000	15 Aug 2000	0.05
15 Feb 2000	15 Feb 2001	0.056
15 Feb 2000	15 Aug 2001	0.06
15 Feb 2000	15 Feb 2002	0.065
15 Feb 2000	15 Aug 2002	0.075

Use the information in this table to populate the `RateSpec` structure.

```
StartDates = ['15-Feb-2000'];
EndDates =   ['15-Aug-2000';
             '15-Feb-2001';
             '15-Aug-2001';
             '15-Feb-2002';
             '15-Aug-2002'];
Compounding = 2;
ValuationDate = ['15-Feb-2000'];
Rates = [0.05; 0.056; 0.06; 0.065; 0.075];

rs = intenvset('Compounding',Compounding,'StartDates',...
StartDates, 'EndDates', EndDates, 'Rates', Rates,...
'ValuationDate', ValuationDate)

rs =

      FinObj: 'RateSpec'
  Compounding: 2
         Disc: [5x1 double]
        Rates: [5x1 double]
     EndTimes: [5x1 double]
```

```
   StartTimes: [5x1 double]
     EndDates: [5x1 double]
   StartDates: 730531
ValuationDate: 730531
        Basis: 0
 EndMonthRule: 1
```

Some of the properties filled in the structure were not passed explicitly in
the call to `RateSpec`. The values of the automatically completed properties
depend on the properties that are explicitly passed. Consider for example
the `StartTimes` and `EndTimes` vectors. Since the `StartDates` and `EndDates`
vectors are passed in, and the `ValuationDate`, `intenvset` has all the
information required to calculate `StartTimes` and `EndTimes`. Hence, these
two properties are read-only.

Obtaining Specific Properties (intenvget)

The complementary function to `intenvset` is `intenvget`, which gets function
specific properties from the interest-rate term structure. Its syntax is:

```
ParameterValue = intenvget(RateSpec, 'ParameterName')
```

To obtain the vector `EndTimes` from the `RateSpec` structure, enter:

```
EndTimes = intenvget(rs, 'EndTimes')

EndTimes =

     1
     2
     3
     4
     5
```

To obtain `Disc`, the values for the discount factors that were calculated
automatically by `intenvset`, type:

```
Disc = intenvget(rs, 'Disc')

Disc =
```

```
        0.9756
        0.9463
        0.9151
        0.8799
        0.8319
```

These discount factors correspond to the periods starting from `StartDates` and ending in `EndDates`.

Caution Although you can directly access these fields within the structure instead of using `intenvget`, it is advised not to do so. The format of the interest-rate term structure could change in future versions of the toolbox. Should that happen, any code accessing the `RateSpec` fields directly would stop working.

Now use the `RateSpec` structure with its functions to examine how changes in specific properties of the interest-rate term structure affect those depending on it. As an exercise, change the value of `Compounding` from 2 (semiannual) to 1 (annual).

```
rs = intenvset(rs, 'Compounding', 1);
```

Since `StartTimes` and `EndTimes` are measured in units of periodic discount, a change in `Compounding` from 2 to 1 redefines the basic unit from semiannual to annual. This means that a period of 6 months is represented with a value of 0.5, and a period of 1 year is represented by 1. To obtain the vectors `StartTimes` and `EndTimes`, enter:

```
StartTimes = intenvget(rs, 'StartTimes');
EndTimes = intenvget(rs, 'EndTimes');
Times = [StartTimes, EndTimes]

Times =

        0    0.5000
        0    1.0000
        0    1.5000
        0    2.0000
```

```
         0   2.5000
```

Since all the values in `StartDates` are the same as the valuation date, all `StartTimes` values are 0. On the other hand, the values in the `EndDates` vector are dates separated by 6-month periods. Since the redefined value of compounding is 1, `EndTimes` becomes a sequence of numbers separated by increments of 0.5.

Pricing Using Interest-Rate Term Structure

In this section...

Introduction

The instruments can be presented to the functions as a portfolio of different types of instruments or as groups of instruments of the same type. The current version of the toolbox can compute price and sensitivities for four instrument types using interest-rate curves:

- Bonds
- Fixed-rate notes
- Floating-rate notes
- Swaps
- OAS for callable and puttable bonds
- Agency OAS

In addition to these instruments, the toolbox also supports the calculation of price and sensitivities of arbitrary sets of cash flows.

Note that options and interest-rate floors and caps are absent from the above list of supported instruments. These instruments are not supported because their pricing and sensitivity function require a stochastic model for the evolution of interest rates. The interest-rate term structure used for pricing is treated as deterministic, and as such is not adequate for pricing these instruments.

Financial Instruments Toolbox software also contains functions that use the Heath-Jarrow-Morton (HJM) and Black-Derman-Toy (BDT) models to compute prices and sensitivities for financial instruments. These models support computations involving options and interest-rate floors and caps. See "Pricing Using Interest-Rate Tree Models" on page 2-67 for information on computing price and sensitivities of financial instruments using the HJM and BDT models.

Computing Instrument Prices

The main function used for pricing portfolios of instruments is `intenvprice`. This function works with the family of functions that calculate the prices of individual types of instruments. When called, `intenvprice` classifies the portfolio contained in `InstSet` by instrument type, and calls the appropriate pricing functions. The map between instrument types and the pricing function `intenvprice` calls is

`bondbyzero`:	Price a bond by a set of zero curves
`fixedbyzero`:	Price a fixed-rate note by a set of zero curves
`floatbyzero`:	Price a floating-rate note by a set of zero curves
`swapbyzero`:	Price a swap by a set of zero curves

You can use each of these functions individually to price an instrument. Consult the reference pages for specific information on using these functions.

`intenvprice` takes as input an interest-rate term structure created with `intenvset`, and a portfolio of interest-rate contingent derivatives instruments created with `instadd`.

The syntax for using `intenvprice` to price an entire portfolio is

```
Price = intenvprice(RateSpec, InstSet)
```

where:

- `RateSpec` is the interest-rate term structure.
- `InstSet` is the name of the portfolio.

Example: Pricing a Portfolio of Instruments

Consider this example of using the `intenvprice` function to price a portfolio of instruments supplied with Financial Instruments Toolbox software.

The provided MAT-file `deriv.mat` stores a portfolio as an instrument set variable `ZeroInstSet`. The MAT-file also contains the interest-rate term structure `ZeroRateSpec`. You can display the instruments with the function `instdisp`.

```
load deriv.mat;
instdisp(ZeroInstSet)

Index Type CouponRate Settle        Maturity    Period Basis...
1     Bond 0.04       01-Jan-2000   01-Jan-2003 1      NaN...
2     Bond 0.04       01-Jan-2000   01-Jan-2004 2      NaN...

Index Type  CouponRate Settle        Maturity    FixedReset Basis...
3     Fixed 0.04       01-Jan-2000 01-Jan-2003 1          NaN...

Index Type  Spread Settle        Maturity    FloatReset Basis...
4     Float  20     01-Jan-2000   01-Jan-2003 1          NaN...

Index Type LegRate    Settle      Maturity    LegReset Basis...
5     Swap [0.06 20] 01-Jan-2000  01-Jan-2003 [1  1]   NaN...
```

Use `intenvprice` to calculate the prices for the instruments contained in the portfolio `ZeroInstSet`.

```
format bank
Prices = intenvprice(ZeroRateSpec, ZeroInstSet)

Prices =

          98.72
          97.53
          98.72
         100.55
           3.69
```

The output `Prices` is a vector containing the prices of all the instruments in the portfolio in the order indicated by the `Index` column displayed by `instdisp`. Consequently, the first two elements in `Prices` correspond to the first two bonds; the third element corresponds to the fixed-rate note; the fourth to the floating-rate note; and the fifth element corresponds to the price of the swap.

Computing Instrument Sensitivities

In general, you can compute sensitivities either as dollar price changes or as percentage price changes. The toolbox reports all sensitivities as dollar sensitivities.

Using the interest-rate term structure, you can calculate two types of derivative price sensitivities, delta and gamma. *Delta* represents the dollar sensitivity of prices to shifts in the observed forward yield curve. *Gamma* represents the dollar sensitivity of delta to shifts in the observed forward yield curve.

The `intenvsens` function computes instrument sensitivities and instrument prices. If you need both the prices and sensitivity measures, use `intenvsens`. A separate call to `intenvprice` is not required.

Here is the syntax

```
[Delta, Gamma, Price] = intenvsens(RateSpec, InstSet)
```

where, as before:

- `RateSpec` is the interest-rate term structure.
- `InstSet` is the name of the portfolio.

Example: Sensitivities and Prices

Here is an example that uses `intenvsens` to calculate both sensitivities and prices.

```
format bank
load deriv.mat;
[Delta, Gamma, Price] = intenvsens(ZeroRateSpec, ZeroInstSet);
```

Display the results in a single matrix in bank format.

```
All = [Delta Gamma Price]

All =

        -272.64       1029.84         98.72
        -347.44       1622.65         97.53
        -272.64       1029.84         98.72
          -1.04          3.31        100.55
        -282.04       1059.62          3.69
```

To view the per-dollar sensitivity, divide the first two columns by the last one.

```
[Delta./Price, Gamma./Price, Price]

ans =

          -2.76         10.43         98.72
          -3.56         16.64         97.53
          -2.76         10.43         98.72
          -0.01          0.03        100.55
         -76.39        286.98          3.69
```

OAS for Callable and Puttable Bonds

Option Adjusted Spread (OAS) is a useful way to value and compare securities with embedded options, like callable or puttable bonds. Basically, when the constant or flat spread is added to the interest-rate curve/rates in the tree, the pricing model value equals the market price. Financial Instruments Toolbox supports pricing American, European and Bermuda callable and puttable bonds using different interest rate models. The pricing for a bond with embedded options is:

- For a callable bond, where the holder has bought a bond and sold a call option to the issuer:

  Price callable bond = Price Option free bond   Price call option

- For a puttable bond, where the holder has bought a bond and a put option:

```
Price puttable bond = Price Option free bond + Price put option
```

There are two additional sensitivities related to OAS for bonds with embedded options: Option Adjusted Duration and Option Adjusted Convexity. These are similar to the concepts of modified duration and convexity for option-free bonds. The measure Duration is a general term that describes how sensitive a bond's price is to a parallel shift in the yield curve. Modified Duration and Modified Convexity assume that the bond's cash flows do not change when the yield curve shifts. This is not true for OA Duration or OA Convexity because the cash flows may change due to the option risk component of the bond.

Function	Purpose
oasbybdt	Compute OAS using a BDT model.
oasbybk	Compute OAS using a BK model.
oasbyhjm	Compute OAS using an HJM model.
oasbyhw	Compute OAS using an HW model.

Agency OAS

Often bonds are issued with embedded options, which then makes standard price/yield or spread measures irrelevant. For example, a municipality concerned about the chance that interest rates may fall in the future might issue bonds with a provision that allows the bond to be repaid before the bond's maturity. This is a call option on the bond and must be incorporated into the valuation of the bond. Option-adjusted spread (OAS), which adjusts a bond spread for the value of the option, is the standard measure for valuing bonds with embedded options. Financial Instruments Toolbox software supports computing option-adjusted spreads for bonds with single embedded options using the agency model.

The Securities Industry and Financial Markets Association (SIFMA) has a simplified approach to compute OAS for agency issues (Government Sponsored Entities like Fannie Mae and Freddie Mac) termed "Agency OAS". In this approach, the bond has only one call date (European call) and uses Black's model (a variation on Black Scholes, http://en.wikipedia.org/wiki/Black_model) to value the bond option. The price of the bond is computed as follows:

$$\text{Price}_{\text{Callable}} = \text{Price}_{\text{NonCallable}} - \text{Price}_{\text{Option}}$$

where

$\text{Price}_{\text{Callable}}$ is the price of the callable bond.

$\text{Price}_{\text{NonCallable}}$ is the price of the noncallable bond, i.e., price of the bond using bndspread.

$\text{Price}_{\text{Option}}$ is the price of the option, i.e., price of the option using Black's model.

The Agency OAS is the spread, when used in the previous formula, yields the market price. Financial Instruments Toolbox software supports these functions:

Agency OAS Functions	Purpose
agencyoas	Compute the OAS of the callable bond using the Agency OAS model.
agencyprice	Price the callable bond OAS using Agency using the OAS model.

For more information on agency OAS, see "Agency Option-Adjusted Spreads" on page 6-2.

Understanding Interest-Rate Tree Models

Introduction

Financial Instruments Toolbox software supports the Black-Derman-Toy (BDT), Black-Karasinski (BK), Heath-Jarrow-Morton (HJM), and Hull-White (HW) interest-rate models. The Heath-Jarrow-Morton model is one of the most widely used models for pricing interest-rate derivatives. The model considers a given initial term structure of interest rates and a specification of the volatility of forward rates to build a tree representing the evolution of the interest rates, based on a statistical process. For further explanation, see the book *Modelling Fixed Income Securities and Interest Rate Options* by Robert A. Jarrow.

The Black-Derman-Toy model is another analytical model commonly used for pricing interest-rate derivatives. The model considers a given initial zero rate term structure of interest rates and a specification of the yield volatilities of long rates to build a tree representing the evolution of the interest rates. For further explanation, see the paper "A One Factor Model of Interest Rates and its Application to Treasury Bond Options" by Fischer Black, Emanuel Derman, and William Toy.

The Hull-White model incorporates the initial term structure of interest rates and the volatility term structure to build a trinomial recombining tree of short rates. The resulting tree is used to value interest rate dependent securities. The implementation of the Hull-White model in Financial Instruments Toolbox software is limited to one factor.

The Black-Karasinski model is a single factor, log-normal version of the Hull-White model.

For further information on the Hull-White and Black-Karasinski models, see the book *Options, Futures, and Other Derivatives* by John C. Hull.

Building a Tree of Forward Rates

The tree of forward rates is the fundamental unit representing the evolution of interest rates in a given period of time. This section explains how to create a forward-rate tree using Financial Instruments Toolbox software.

Note To avoid needless repetition, this document uses the HJM and BDT models to illustrate the creation and use of interest-rate trees. The HW and BK models are similar to the BDT model. Where specific differences exist, they are documented in "HW and BK Tree Structures" on page 2-63.

The MATLAB functions that create rate trees are `hjmtree` and `bdttree`. The `hjmtree` function creates the structure, `HJMTree`, containing time and forward-rate information for a bushy tree. The `bdttree` function creates a similar structure, `BDTTree`, for a recombining tree.

This structure is a self-contained unit that includes the tree of rates (found in the `FwdTree` field of the structure) and the volatility, rate, and time specifications used in building this tree.

These functions take three structures as input arguments:

- The volatility model `VolSpec`. (See "Specifying the Volatility Model (VolSpec)" on page 2-49.)

- The interest-rate term structure `RateSpec`. (See "Specifying the Interest-Rate Term Structure (RateSpec)" on page 2-52.)

- The tree time layout `TimeSpec`. (See "Specifying the Time Structure (TimeSpec)" on page 2-53.)

An easy way to visualize any trees you create is with the `treeviewer` function, which displays trees in a graphical manner. See "Graphical Representation of Trees" on page 2-113 for information about `treeviewer`.

Calling Sequence

The calling syntax for `hjmtree` is

```
HJMTree = hjmtree(VolSpec, RateSpec, TimeSpec)
```

Similarly, the calling syntax for `bdttree` is

```
BDTTree = bdttree(VolSpec, RateSpec, TimeSpec)
```

Each of these functions requires `VolSpec`, `RateSpec`, and `TimeSpec` input arguments:

- `VolSpec` is a structure that specifies the forward-rate volatility process. You create `VolSpec` using either of the functions `hjmvolspec` or `bdtvolspec`.

  The `hjmvolspec` function supports the specification of up to three factors. It handles these models for the volatility of the interest-rate term structure:

  - Constant
  - Stationary
  - Exponential
  - Vasicek
  - Proportional

  A one-factor model assumes that the interest term structure is affected by a single source of uncertainty. Incorporating multiple factors allows you to specify different types of shifts in the shape and location of the interest-rate structure. See `hjmvolspec` for details.

  The `bdtvolspec` function supports only a single volatility factor. The volatility remains constant between pairs of nodes on the tree. You supply the input volatility values in a vector of decimal values. See `bdtvolspec` for details.

- `RateSpec` is the interest-rate specification of the initial rate curve. You create this structure with the function `intenvset`. (See "Modeling the Interest-Rate Term Structure" on page 2-33.)

- `TimeSpec` is the tree time layout specification. You create this variable with the functions `hjmtimespec` or `bdttimespec`. It represents the mapping between level times and level dates for rate quoting. This structure indirectly determines the number of levels in the tree.

Specifying the Volatility Model (VolSpec)

Because HJM supports multifactor (up to 3) volatility models while BDT (also, BK and HW) supports only a single volatility factor, the `hjmvolspec` and `bdtvolspec` functions require different inputs and generate slightly different outputs. For examples, see "Creating an HJM Volatility Model" on page 2-49. For BDT examples see "Creating a BDT Volatility Model" on page 2-51.

Creating an HJM Volatility Model

The function `hjmvolspec` generates the structure `VolSpec`, which specifies

the volatility process $\sigma(t,T)$ used in the creation of the forward-rate trees. In this context capital T represents the starting time of the forward rate, and t represents the observation time. The volatility process can be constructed from a combination of factors specified sequentially in the call to function that creates it. Each factor specification starts with a string specifying the name of the factor, followed by the pertinent parameters.

HJM Volatility Specification Example. Consider an example that uses a single factor, specifically, a constant-sigma factor. The constant factor specification requires only one parameter, the value of σ. In this case, the value corresponds to 0.10.

```
HJMVolSpec = hjmvolspec('Constant', 0.10)

HJMVolSpec =

      FinObj: 'HJMVolSpec'
FactorModels: {'Constant'}
  FactorArgs: {{1x1 cell}}
  SigmaShift: 0
  NumFactors: 1
```

```
   NumBranch: 2
      PBranch: [0.5000 0.5000]
 Fact2Branch: [-1 1]
```

The `NumFactors` field of the `VolSpec` structure, `VolSpec.NumFactors =` 1, reveals that the number of factors used to generate `VolSpec` was one. The `FactorModels` field indicates that it is a `Constant` factor, and the `NumBranches` field indicates the number of branches. As a consequence, each node of the resulting tree has two branches, one going up, and the other going down.

Consider now a two-factor volatility process made from a proportional factor and an exponential factor.

```
% Exponential factor
Sigma_0 = 0.1;
Lambda = 1;
% Proportional factor
CurveProp = [0.11765; 0.08825; 0.06865];
CurveTerm = [   1  ;   2  ;    3   ];
% Build VolSpec
HJMVolSpec = hjmvolspec('Proportional', CurveProp, CurveTerm,...
1e6,'Exponential', Sigma_0, Lambda)

HJMVolSpec =

       FinObj: 'HJMVolSpec'
 FactorModels: {'Proportional'  'Exponential'}
   FactorArgs: {{1x3 cell}  {1x2 cell}}
   SigmaShift: 0
   NumFactors: 2
    NumBranch: 3
      PBranch: [0.2500 0.2500 0.5000]
  Fact2Branch: [2x3 double]
```

The output shows that the volatility specification was generated using two factors. The tree has 3 branches per node. Each branch has probabilities of 0.25, 0.25, and 0.5, going from top to bottom.

Creating a BDT Volatility Model

The function `bdtvolspec` generates the structure `VolSpec`, which specifies the volatility process. The function requires three input arguments:

- The valuation date `ValuationDate`

- The yield volatility end dates `VolDates`

- The yield volatility values `VolCurve`

An optional fourth argument `InterpMethod`, specifying the interpolation method, can be included.

The syntax used for calling `bdtvolspec` is:

```
BDTVolSpec = bdtvolspec(ValuationDate, VolDates, VolCurve,...
InterpMethod)
```

where:

- `ValuationDate` is the first observation date in the tree.

- `VolDates` is a vector of dates representing yield volatility end dates.

- `VolCurve` is a vector of yield volatility values.

- `InterpMethod` is the method of interpolation to use. The default is `linear`.

BDT Volatility Specification Example. Consider the following example:

```
ValuationDate = datenum('01-01-2000');
EndDates = datenum(['01-01-2001'; '01-01-2002'; '01-01-2003';
'01-01-2004'; '01-01-2005']);
Volatility = [.2; .19; .18; .17; .16];
```

Use `bdtvolspec` to create a volatility specification. Because no interpolation method is explicitly specified, the function uses the `linear` default.

```
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Volatility)

BDTVolSpec =
              FinObj: 'BDTVolSpec'
       ValuationDate: 730486
```

```
            VolDates: [5x1 double]
            VolCurve: [5x1 double]
     VolInterpMethod: 'linear'
```

Specifying the Interest-Rate Term Structure (RateSpec)

The structure `RateSpec` is an interest term structure that defines the initial forward-rate specification from which the tree rates are derived. "Modeling the Interest-Rate Term Structure" on page 2-33 explains how to create these structures using the function `intenvset`, given the interest rates, the starting and ending dates for each rate, and the compounding value.

Rate Specification Creation Example

Consider the following example:

```
Compounding = 1;
Rates = [0.02; 0.02; 0.02; 0.02];
StartDates = ['01-Jan-2000';
              '01-Jan-2001';
              '01-Jan-2002';
              '01-Jan-2003'];
EndDates =   ['01-Jan-2001';
              '01-Jan-2002';
              '01-Jan-2003';
              '01-Jan-2004'];
ValuationDate = '01-Jan-2000';

RateSpec = intenvset('Compounding',1,'Rates', Rates,...
'StartDates', StartDates, 'EndDates', EndDates,...
'ValuationDate', ValuationDate)

RateSpec =

        FinObj: 'RateSpec'
   Compounding: 1
          Disc: [4x1 double]
         Rates: [4x1 double]
      EndTimes: [4x1 double]
    StartTimes: [4x1 double]
      EndDates: [4x1 double]
```

```
   StartDates: [4x1 double]
ValuationDate: 730486
        Basis: 0
  EndMonthRule: 1
```

Use the function `datedisp` to examine the dates defined in the variable `RateSpec`. For example:

```
datedisp(RateSpec.ValuationDate)
01-Jan-2000
```

Specifying the Time Structure (TimeSpec)

The structure `TimeSpec` specifies the time structure for an interest-rate tree. This structure defines the mapping between the observation times at each level of the tree and the corresponding dates.

`TimeSpec` is built using either the `hjmtimespec` or `bdttimespec` function. These functions require three input arguments:

- The valuation date `ValuationDate`
- The maturity date `Maturity`
- The compounding rate `Compounding`

For example, the syntax used for calling `hjmtimespec` is

```
TimeSpec = hjmtimespec(ValuationDate, Maturity, Compounding)
```

where:

- `ValuationDate` is the first observation date in the tree.
- `Maturity` is a vector of dates representing the cash flow dates of the tree. Any instrument cash flows with these maturities fall on tree nodes.
- `Compounding` is the frequency at which the rates are compounded when annualized.

Creating a Time Specification

Calling the time specification creation functions with the same data used to create the interest-rate term structure, `RateSpec` builds the structure that specifies the time layout for the tree.

HJM Time Specification Example. Consider the following example:

```
Maturity = EndDates;
HJMTimeSpec = hjmtimespec(ValuationDate, Maturity, Compounding)


HJMTimeSpec =

        FinObj: 'HJMTimeSpec'
ValuationDate: 730486
     Maturity: [4x1 double]
  Compounding: 1
        Basis: O
 EndMonthRule: 1
```

Note that maturities specified when building `TimeSpec` need not coincide with the `EndDates` of the rate intervals in `RateSpec`. Since `TimeSpec` defines the time-date mapping of the tree, the rates in `RateSpec` are interpolated to obtain the initial rates with maturities equal to those in `TimeSpec`.

Creating a BDT Time Specification. Consider the following example:

```
Maturity = EndDates;
BDTTimeSpec = bdttimespec(ValuationDate, Maturity, Compounding)


BDTTimeSpec =

          FinObj: 'BDTTimeSpec'
   ValuationDate: 730486
        Maturity: [4x1 double]
     Compounding: 1
           Basis: O
    EndMonthRule: 1
```

Creating Trees

Use the `VolSpec`, `RateSpec`, and `TimeSpec` you have previously created as inputs to the functions used to create HJM and BDT trees.

Creating an HJM Tree

```
% Reset the volatility factor to the Constant case
HJMVolSpec = hjmvolspec('Constant', 0.10);

HJMTree = hjmtree(HJMVolSpec, RateSpec, HJMTimeSpec)

HJMTree =

   FinObj: 'HJMFwdTree'
  VolSpec: [1x1 struct]
 TimeSpec: [1x1 struct]
 RateSpec: [1x1 struct]
     tObs: [0 1 2 3]
     TFwd: {[4x1 double] [3x1 double] [2x1 double] [3]}
   CFlowT: {[4x1 double] [3x1 double] [2x1 double] [4]}
  FwdTree:{[4x1 double][3x1x2 double][2x2x2 double][1x4x2 double]}
```

Creating a BDT Tree

Now use the previously computed values for `VolSpec`, `RateSpec`, and `TimeSpec` as input to the function `bdttree` to create a BDT tree.

```
BDTTree = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec)

BDTTree =

   FinObj: 'BDTFwdTree'
  VolSpec: [1x1 struct]
 TimeSpec: [1x1 struct]
 RateSpec: [1x1 struct]
     tObs: [0 1.00 2.00 3.00]
     TFwd: {[4x1 double]  [3x1 double]  [2x1 double]  [3.00]}
   CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4.00]}
  FwdTree: {[1.02] [1.02 1.02] [1.01 1.02 1.03] [1.01 1.02 1.02 1.03]}
```

Examining Trees

When working with the models, Financial Instruments Toolbox software uses trees to represent forward rates, prices, and so on. At the highest level, these trees have structures wrapped around them. The structures encapsulate information required to interpret completely the information contained in a tree.

Consider this example, which uses the interest rate and portfolio data in the MAT-file `deriv.mat` included in the toolbox.

Load the data into the MATLAB workspace.

```
load deriv.mat
```

Display the list of the variables loaded from the MAT-file.

```
whos
```

```
  Name             Size          Bytes  Class     Attributes

  BDTInstSet       1x1           15956  struct
  BDTTree          1x1            5138  struct
  BKInstSet        1x1           15946  struct
  BKTree           1x1            5904  struct
  CRRInstSet       1x1           12434  struct
  CRRTree          1x1            5058  struct
  EQPInstSet       1x1           12434  struct
  EQPTree          1x1            5058  struct
  HJMInstSet       1x1           15948  struct
  HJMTree          1x1            5838  struct
  HWInstSet        1x1           15946  struct
  HWTree           1x1            5904  struct
  ITTInstSet       1x1           12438  struct
  ITTTree          1x1            8862  struct
  ZeroInstSet      1x1           10282  struct
  ZeroRateSpec     1x1            1580  struct
```

HJM Tree Structure

You can now examine in some detail the contents of the `HJMTree` structure contained in this file.

```
HJMTree

HJMTree =

   FinObj: 'HJMFwdTree'
  VolSpec: [1x1 struct]
 TimeSpec: [1x1 struct]
 RateSpec: [1x1 struct]
     tObs: [0 1 2 3]
     TFwd: {[4x1 double]  [3x1 double]  [2x1 double]  [3]}
   CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
  FwdTree:{[4x1 double][3x1x2 double][2x2x2 double][1x4x2 double]}
```

`FwdTree` contains the actual forward-rate tree. MATLAB software represents it as a cell array with each cell array element containing a tree level.

The other fields contain other information relevant to interpreting the values in `FwdTree`. The most important are `VolSpec`, `TimeSpec`, and `RateSpec`, which contain the volatility, time structure, and rate structure information respectively.

First Node. Observe the forward rates in `FwdTree`. The first node represents the valuation date, `tObs = 0`.

```
HJMTree.FwdTree{1}

ans =

   1.0356
   1.0468
   1.0523
   1.0563
```

Look closely at the `RateSpec` structure used in generating this tree to see where these values originate. Arrange the values in a single array.

```
[HJMTree.RateSpec.StartTimes HJMTree.RateSpec.EndTimes...
HJMTree.RateSpec.Rates]

ans =

         0    1.0000    0.0356
    1.0000    2.0000    0.0468
    2.0000    3.0000    0.0523
    3.0000    4.0000    0.0563
```

If you find the corresponding inverse discounts of the interest rates in the third column, you have the values at the first node of the tree. You can turn interest rates into inverse discounts using the function `rate2disc`.

```
Disc = rate2disc(HJMTree.TimeSpec.Compounding,...
HJMTree.RateSpec.Rates, HJMTree.RateSpec.EndTimes,...
HJMTree.RateSpec.StartTimes);
FRates = 1./Disc

FRates =
    1.0356
    1.0468
    1.0523
    1.0563
```

Second Node. The second node represents the first-rate observation time, `tObs = 1`. This node displays two states: one representing the branch going up and the other representing the branch going down.

Note that `HJMTree.VolSpec.NumBranch = 2`.

```
HJMTree.VolSpec
```

```
ans =

          FinObj: 'HJMVolSpec'
    FactorModels: {'Constant'}
      FactorArgs: {{1x1 cell}}
      SigmaShift: 0
      NumFactors: 1
       NumBranch: 2
          PBranch: [0.5000 0.5000]
      Fact2Branch: [-1 1]
```

Examine the rates of the node corresponding to the up branch.

```
HJMTree.FwdTree{2}(:,:,1)
```

```
ans =

    1.0364
    1.0420
    1.0461
```

Now examine the corresponding down branch.

```
HJMTree.FwdTree{2}(:,:,2)
```

```
ans =

    1.0574
    1.0631
    1.0672
```

Third Node. The third node represents the second observation time, `tObs` = 2. This node contains a total of four states, two representing the branches going up and the other two representing the branches going down. Examine the rates of the node corresponding to the up states.

```
HJMTree.FwdTree{3}(:,:,1)
```

```
ans =
```

```
    1.0317    1.0526
    1.0358    1.0568
```

Next examine the corresponding down states.

```
HJMTree.FwdTree{3}(:,:,2)

ans =

    1.0526    1.0738
    1.0568    1.0781
```

Isolating a Specific Node. Starting at the third level, indexing within the tree cell array becomes complex, and isolating a specific node can be difficult. The function bushpath isolates a specific node by specifying the path to the node as a vector of branches taken to reach that node. As an example, consider the node reached by starting from the root node, taking the branch up, then the branch down, and then another branch down. Given that the tree has only two branches per node, branches going up correspond to a 1, and branches going down correspond to a 2. The path up-down-down becomes the vector [1 2 2].

```
FRates = bushpath(HJMTree.FwdTree, [1 2 2])

FRates =

    1.0356
    1.0364
    1.0526
    1.0674
```

bushpath returns the spot rates for all the nodes touched by the path specified in the input argument, the first one corresponding to the root node, and the last one corresponding to the target node.

Isolating the same node using direct indexing obtains

```
HJMTree.FwdTree{4}(:, 3, 2)

ans =
```

```
     1.0674
```

As expected, this single value corresponds to the last element of the rates returned by `bushpath`.

You can use these techniques with any type of tree generated with Financial Instruments Toolbox software, such as forward-rate trees or price trees.

BDT Tree Structure

You can now examine in some detail the contents of the `BDTTree` structure.

```
BDTTree
```

```
BDTTree =

      FinObj: 'BDTFwdTree'
     VolSpec: [1x1 struct]
    TimeSpec: [1x1 struct]
    RateSpec: [1x1 struct]
        tObs: [0 1.00 2.00 3.00]
        TFwd: {[4x1 double]  [3x1 double]  [2x1 double]  [3.00]}
      CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4.00]}
     FwdTree: {[1.10]  [1.10 1.14]  [1.10 1.14 1.19]  [1.09 1.12 1.16 1.22]}
```

`FwdTree` contains the actual rate tree. MATLAB software represents it as a cell array with each cell array element containing a tree level.

The other fields contain other information relevant to interpreting the values in `FwdTree`. The most important are `VolSpec`, `TimeSpec`, and `RateSpec`, which contain the volatility, time structure, and rate structure information respectively.

Look at the `RateSpec` structure used in generating this tree to see where these values originate. Arrange the values in a single array.

```
[BDTTree.RateSpec.StartTimes BDTTree.RateSpec.EndTimes...
BDTTree.RateSpec.Rates]
```

```
ans =
```

```
0    1.0000    0.1000
0    2.0000    0.1100
0    3.0000    0.1200
0    4.0000    0.1250
```

Look at the rates in FwdTree. The first node represents the valuation date, tObs = 0. The second node represents tObs = 1. Examine the rates at the second, third, and fourth nodes.

```
BDTTree.FwdTree{2}

ans =

    1.0979    1.1432
```

The second node represents the first observation time, tObs = 1. This node contains a total of two states, one representing the branch going up (1.0979) and the other representing the branch going down (1.1432).

Note The convention in this document is to display *prices* going up on the upper branch. Consequently, when displaying *rates*, rates are falling on the upper branch and increasing on the lower branch.

```
BDTTree.FwdTree{3}

ans =

   1.0976    1.1377    1.1942
```

The third node represents the second observation time, tObs = 2. This node contains a total of three states, one representing the branch going up (1.0976), one representing the branch in the middle (1.1377) and the other representing the branch going down (1.1942).

```
BDTTree.FwdTree{4}

ans =
```

```
        1.0872      1.1183      1.1606      1.2179
```

The fourth node represents the third observation time, `tObs = 3`. This node contains a total of four states, one representing the branch going up (`1.0872`), two representing the branches in the middle (`1.1183` and `1.1606`), and the other representing the branch going down (`1.2179`).

Isolating a Specific Node. The function `treepath` isolates a specific node by specifying the path to the node as a vector of branches taken to reach that node. As an example, consider the node reached by starting from the root node, taking the branch up, then the branch down, and finally another branch down. Given that the tree has only two branches per node, branches going up correspond to a 1, and branches going down correspond to a 2. The path up-down-down becomes the vector [1 2 2].

```
FRates = treepath(BDTTree.FwdTree, [1 2 2])

FRates =

    1.1000
    1.0979
    1.1377
    1.1606
```

`treepath` returns the short rates for all the nodes touched by the path specified in the input argument, the first one corresponding to the root node, and the last one corresponding to the target node.

HW and BK Tree Structures

The HW and BK tree structures are similar to the BDT tree structure. You can see this if you examine the sample HW tree contained in the file `deriv.mat`.

```
load deriv.mat;
HWTree

HWTree =

     FinObj: 'HWFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
```

```
    RateSpec: [1x1 struct]
       tObs: [0 1.00 2.00 3.00]
       dObs: [731947.00 732313.00 732678.00 733043.00]
     CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4.00]}
      Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
    Connect: {[2.00]  [2.00 3.00 4.00]  [2.00 2.00 3.00 4.00 4.00]}
    FwdTree: {[1.03]  [1.05 1.04 1.02]  [1.08 1.07 1.05 1.03 1.01]  [1.09 1.08 1.06 1.04 1.02]
```

All fields of this structure are similar to their BDT counterparts. There are
two additional fields not present in BDT: `Probs` and `Connect`. The `Probs` field
represents the occurrence probabilities at each branch of each node in the
tree. The `Connect` field describes the connectivity of the nodes of a given tree
level to nodes to the next tree level.

Probs Field. While BDT and one-factor HJM models have equal probabilities
for each branch at a node, HW and BK do not. For HW and BK trees, the
`Probs` field indicates the likelihood that a particular branch will be taken in
moving from one node to another node on the next level.

The `Probs` field consists of a cell array with 1 cell per tree level. Each cell
is a 3-by-`NUMNODES` array with the top row representing the probability of
an up movement, the middle row representing the probability of a middle
movement, and the last row the probability of a down movement.

As an illustration, consider the first two elements of the `Probs` field of the
structure, corresponding to the first (root) and second levels of the tree.

`HWTree.Probs{1}`

```
0.16666666666667
0.66666666666667
0.16666666666667
```

`HWTree.Probs{2}`

```
0.12361333418768    0.16666666666667    0.21877591615172
0.65761074966060    0.66666666666667    0.65761074966060
0.21877591615172    0.16666666666667    0.12361333418768
```

Reading from top to bottom, the values in `HWTree.Probs{1}` correspond to the
up, middle, and down probabilities at the root node.

`HWTree.Probs{2}` is a 3-by-3 matrix of values. The first column represents the top node, the second column represents the middle node, and the last column represents the bottom node. As with the root node, the first, second, and third rows hold the values for up, middle, and down branching off each node.

As expected, the sum of all the probabilities at any node equals 1.

```
sum(HWTree.Probs{2})
```

```
1.0000    1.0000    1.0000
```

Connect Field. The other field that distinguishes HW and BK tree structures from the BDT tree structure is `Connect`. This field describes how each node in a given level connects to the nodes of the next level. The need for this field arises from the possibility of nonstandard branching in a tree.

The `Connect` field of the HW tree structure consists of a cell array with 1 cell per tree level.

```
HWTree.Connect
```

```
ans =

    [2]    [1x3 double]    [1x5 double]
```

Each cell contains a 1-by-`NUMNODES` vector. Each value in the vector relates to a node in the corresponding tree level and represents the index of the node in the next tree level that the middle branch of the node connects to.

If you subtract 1 from the values contained in `Connect`, you reveal the index of the nodes in the next level that the up branch connects to. If you add 1 to the values, you reveal the index of the corresponding down branch.

As an illustration, consider `HWTree.Connect{1}`:

```
HWTree.Connect{1}
```

```
ans =

     2
```

This indicates that the middle branch of the root node connects to the second (from the top) node of the next level, as expected. If you subtract 1 from this value, you obtain 1, which tells you that the up branch goes to the top node. If you add 1, you obtain 3, which points to the last node of the second level of the tree.

Now consider level 3 in this example:

```
HWTree.Connect{3}
```

```
2     2     3     4     4
```

On this level, there is nonstandard branching. This can be easily recognized because the middle branch of two nodes is connected to the same node on the next level.

To visualize this, consider the following illustration of the tree.
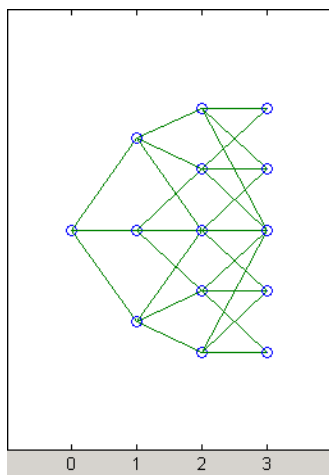
Here it becomes apparent that there is nonstandard branching at the third level of the tree, on the top and bottom nodes. The first and second nodes connect to the same trio of nodes on the next level. Similar branching occurs at the bottom and next-to-bottom nodes of the tree.

Pricing Using Interest-Rate Tree Models

In this section...
"Introduction" on page 2-67
"Computing Instrument Prices" on page 2-67
"Computing Instrument Sensitivities" on page 2-76
"Calibrating Hull-White Model Using Market Data" on page 2-79

Introduction

For purposes of illustration, this section relies on the HJM and BDT models. The HW and BK functions that perform price and sensitivity computations are not explicitly shown here. Functions that use the HW and BK models operate similarly to the BDT model.

Computing Instrument Prices

The portfolio pricing functions `hjmprice` and `bdtprice` calculate the price of any set of supported instruments, based on an interest-rate tree. The functions are capable of pricing these instrument types:

- Bonds
- Bond options
- Bond with embedded options
- Arbitrary cash flows
- Fixed-rate notes
- Floating-rate notes
- Floating-rate notes with options or embedded options
- Caps
- Floors
- Range Notes
- Swaps

- Swaptions

For example, the syntax for calling `hjmprice` is:

```
[Price, PriceTree] = hjmprice(HJMTree, InstSet, Options)
```

Similarly, the calling syntax for `bdtprice` is:

```
[Price, PriceTree] = bdtprice(BDTTree, InstSet, Options)
```

Each function requires two input arguments: the interest-rate tree and the set of instruments, `InstSet`. An optional argument `Options` further controls the pricing and the output displayed. (See Appendix B, "Derivatives Pricing Options" for information about the `Options` argument.)

`HJMTree` is the Heath-Jarrow-Morton tree sampling of a forward-rate process, created using `hjmtree`. `BDTTree` is the Black-Derman-Toy tree sampling of an interest-rate process, created using `bdttree`. See "Building a Tree of Forward Rates" on page 2-47 to learn how to create these structures.

`InstSet` is the set of instruments to be priced. This structure represents the set of instruments to be priced independently using the model.

`Options` is an options structure created with the function `derivset`. This structure defines how the tree is used to find the price of instruments in the portfolio, and how much additional information is displayed in the command window when calling the pricing function. If this input argument is not specified in the call to the pricing function, a default Options structure is used. The pricing options structure is described in "Pricing Options Structure" on page B-2.

The portfolio pricing functions classify the instruments and call the appropriate instrument-specific pricing function for each of the instrument types. The HJM instrument-specific pricing functions are `bondbyhjm`, `cfbyhjm`, `fixedbyhjm`, `floatbyhjm`, `optbndbyhjm`, `rangefloatbyhjm`, `swapbyhjm`, and `swaptionbyhjm`. A similarly named set of functions exists for BDT models. You can also use these functions directly to calculate the price of sets of instruments of the same type.

HJM Pricing Example

Consider the following example, which uses the portfolio and interest-rate data in the MAT-file `deriv.mat` included in the toolbox. Load the data into the MATLAB workspace.

```
load deriv.mat
```

Use the MATLAB `whos` command to display a list of the variables loaded from the MAT-file.

```
whos
```

```
Name              Size            Bytes  Class    Attributes

  BDTInstSet      1x1             15956  struct
  BDTTree         1x1              5138  struct
  BKInstSet       1x1             15946  struct
  BKTree          1x1              5904  struct
  CRRInstSet      1x1             12434  struct
  CRRTree         1x1              5058  struct
  EQPInstSet      1x1             12434  struct
  EQPTree         1x1              5058  struct
  HJMInstSet      1x1             15948  struct
  HJMTree         1x1              5838  struct
  HWInstSet       1x1             15946  struct
  HWTree          1x1              5904  struct
  ITTInstSet      1x1             12438  struct
  ITTTree         1x1              8862  struct
  ZeroInstSet     1x1             10282  struct
  ZeroRateSpec    1x1              1580  struct
```

`HJMTree` and `HJMInstSet` are the input arguments required to call the function `hjmprice`.

Use the function `instdisp` to examine the set of instruments contained in the variable `HJMInstSet`.

```
instdisp(HJMInstSet)
```

Index	Type	CouponRate	Settle	Maturity	Period	Basis	EndMonthRule	IssueDate	FirstCouponDate	LastCouponDate	StartDate	Fac
1	Bond	0.04	01-Jan-2000	01-Jan-2003	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Bond	0.04	01-Jan-2000	01-Jan-2004	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Index	Type	UnderInd	OptSpec	Strike	ExerciseDates	AmericanOpt	Name	Quantity
3	OptBond	2	call	101	01-Jan-2003	NaN	Option 101	-50

Index	Type	CouponRate	Settle	Maturity	FixedReset	Basis	Principal	Name	Quantity
4	Fixed	0.04	01-Jan-2000	01-Jan-2003	1	NaN	NaN	4% Fixed	80

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	Name	Quantity
5	Float	20	01-Jan-2000	01-Jan-2003	1	NaN	NaN	20BP Float	8

Index	Type	Strike	Settle	Maturity	CapReset	Basis	Principal	Name	Quantity
6	Cap	0.03	01-Jan-2000	01-Jan-2004	1	NaN	NaN	3% Cap	30

Index	Type	Strike	Settle	Maturity	FloorReset	Basis	Principal	Name	Quantity
7	Floor	0.03	01-Jan-2000	01-Jan-2004	1	NaN	NaN	3% Floor	40

Index	Type	LegRate	Settle	Maturity	LegReset	Basis	Principal	LegType	Name	Quantity
8	Swap	[0.06 20]	01-Jan-2000	01-Jan-2003	[1 1]	NaN	NaN	[NaN]	6%/20BP Swap	10

Index	Type	CouponRate	Settle	Maturity	Period	Basis	...	Name	Quantity
1	Bond	0.04	01-Jan-2000	01-Jan-2003	1	NaN	...	4% bond	100
2	Bond	0.04	01-Jan-2000	01-Jan-2004	2	NaN	...	4% bond	50

Note that there are eight instruments in this portfolio set: two bonds, one bond option, one fixed-rate note, one floating-rate note, one cap, one floor, and one swap. Each instrument has a corresponding index that identifies the instrument prices in the price vector returned by hjmprice.

Now use hjmprice to calculate the price of each instrument in the instrument set.

```
Price = hjmprice(HJMTree, HJMInstSet)
```

```
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

   98.7159
   97.5280
    0.0486
   98.7159
  100.5529
    6.2831
    0.0486
    3.6923
```

Note The warning shown above appears because some of the cash flows for the second bond do not fall exactly on a tree node.

BDT Pricing Example

Load the MAT-file deriv.mat into the MATLAB workspace.

```
load deriv.mat
```

Use the MATLAB whos command to display a list of the variables loaded from the MAT-file.

```
whos
```

```
  Name          Size         Bytes  Class      Attributes

  BDTInstSet    1x1          15956  struct
  BDTTree       1x1           5138  struct
  BKInstSet     1x1          15946  struct
  BKTree        1x1           5904  struct
  CRRInstSet    1x1          12434  struct
  CRRTree       1x1           5058  struct
  EQPInstSet    1x1          12434  struct
  EQPTree       1x1           5058  struct
```

```
HJMInstSet      1x1              15948   struct
HJMTree         1x1               5838   struct
HWInstSet       1x1              15946   struct
HWTree          1x1               5904   struct
ITTInstSet      1x1              12438   struct
ITTTree         1x1               8862   struct
ZeroInstSet     1x1              10282   struct
ZeroRateSpec    1x1               1580   struct
```

BDTTree and BDTInstSet are the input arguments required to call the function bdtprice.

Use the function instdisp to examine the set of instruments contained in the variable BDTInstSet.

```
instdisp(BDTInstSet)
```

Index	Type	CouponRate	Settle	Maturity	Period	Basis	EndMonthRule	IssueDate	FirstCouponDate	LastCouponDate	StartDate	Fac
1	Bond	0.1	01-Jan-2000	01-Jan-2003	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Bond	0.1	01-Jan-2000	01-Jan-2004	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Index	Type	UnderInd	OptSpec	Strike	ExerciseDates	AmericanOpt	Name	Quantity
3	OptBond	1	call	95	01-Jan-2002	NaN	Option 95	-50

Index	Type	CouponRate	Settle	Maturity	FixedReset	Basis	Principal	Name	Quantity
4	Fixed	0.1	01-Jan-2000	01-Jan-2003	1	NaN	NaN	10% Fixed	80

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	Name	Quantity
5	Float	20	01-Jan-2000	01-Jan-2003	1	NaN	NaN	20BP Float	8

Index	Type	Strike	Settle	Maturity	CapReset	Basis	Principal	Name	Quantity
6	Cap	0.15	01-Jan-2000	01-Jan-2004	1	NaN	NaN	15% Cap	30

Index	Type	Strike	Settle	Maturity	FloorReset	Basis	Principal	Name	Quantity
7	Floor	0.09	01-Jan-2000	01-Jan-2004	1	NaN	NaN	9% Floor	40

Index	Type	LegRate	Settle	Maturity	LegReset	Basis	Principal	LegType	Name	Quantity
8	Swap	[0.15 10]	01-Jan-2000	01-Jan-2003	[1 1]	NaN	NaN	[NaN]	15%/10BP Swap	10

Note that there are eight instruments in this portfolio set: two bonds, one bond option, one fixed-rate note, one floating-rate note, one cap, one floor, and one swap. Each instrument has a corresponding index that identifies the instrument prices in the price vector returned by bdtprice.

Now use bdtprice to calculate the price of each instrument in the instrument set.

```
Price = bdtprice(BDTTree, BDTInstSet)

Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

   95.5030
   93.9079
    1.7657
   95.5030
  100.4865
    1.4863
    0.0245
    7.4222
```

Price Vector Output

The prices in the output vector Price correspond to the prices at observation time zero (tObs = 0), which is defined as the valuation date of the interest-rate tree. The instrument indexing within Price is the same as the indexing within InstSet.

In the HJM example, the prices in the Price vector correspond to the instruments in this order.

```
InstNames = instget(HJMInstSet, 'FieldName','Name')

InstNames =

4% bond
4% bond
Option 101
```

```
4% Fixed
20BP Float
3% Cap
3% Floor
6%/20BP Swap
```

Consequently, in the `Price` vector, the fourth element, 98.7159, represents the price of the fourth instrument (4% fixed-rate note); the sixth element, 6.2831, represents the price of the sixth instrument (3% cap).

In the BDT example, the prices in the `Price` vector correspond to the instruments in this order.

```
InstNames = instget(BDTInstSet, 'FieldName','Name')

InstNames =

10% Bond
10% Bond
Option 95
10% Fixed
20BP Float
15% Cap
9% Floor
15%/10BP Swap
```

Consequently, in the `Price` vector, the fourth element, 95.5030, represents the price of the fourth instrument (10% fixed-rate note); the sixth element, 1.4863, represents the price of the sixth instrument (15% cap).

Price Tree Structure Output

If you call a pricing function with two output arguments, for example,

```
[Price, PriceTree] = hjmprice(HJMTree, HJMInstSet)
```

you generate a price tree along with the price information.

The optional output price tree structure `PriceTree` holds all the pricing information.

HJM Price Tree. In the HJM example, the first field of this structure, FinObj, indicates that this structure represents a price tree. The second field, PBush, is the tree holding the price of the instruments in each node of the tree. The third field, AIBush, is the tree holding the accrued interest of the instruments in each node of the tree. Finally, the fourth field, tObs, represents the observation time of each level of PBush and AIBush, with units in terms of compounding periods.

In this example, the price tree looks like

```
PriceTree =

FinObj: 'HJMPriceTree'
 PBush: {[8x1 double]  [8x1x2 double]  ...[8x8 double]}
AIBush: {[8x1 double]  [8x1x2 double] ... [8x8 double]}
  tObs: [0 1 2 3 4]
```

Both PBush and AIBush are 1-by-5 cell arrays, consistent with the five observation times of tObs. The data display has been shortened here to fit on a single line.

Using the command line interface, you can directly examine PriceTree.PBush, the field within the PriceTree structure that contains the price tree with the price vectors at every state. The first node represents tObs = 0, corresponding to the valuation date.

```
PriceTree.PBush{1}

ans =

   98.7159
   97.5280
    0.0486
   98.7159
  100.5529
    6.2831
    0.0486
    3.6923
```

With this interface, you can observe the prices for *all* instruments in the portfolio at *a specific time*.

2-75

BDT Price Tree. The BDT output price tree structure `PriceTree` holds all the pricing information. The first field of this structure, `FinObj`, indicates that this structure represents a price tree. The second field, `PTree`, is the tree holding the price of the instruments in each node of the tree. The third field, `AITree`, is the tree holding the accrued interest of the instruments in each node of the tree. The fourth field, `tObs`, represents the observation time of each level of `PTree` and `AITree`, with units in terms of compounding periods.

You can directly examine the field within the `PriceTree` structure, which contains the price tree with the price vectors at every state. The first node represents `tObs = 0,` corresponding to the valuation date.

```
[Price, PriceTree] = bdtprice(BDTTree, BDTInstSet)

PriceTree.PTree{1}

ans =

    95.5030
    93.9079
     1.7657
    95.5030
   100.4865
     1.4863
     0.0245
     7.4222
```

Computing Instrument Sensitivities

Sensitivities can be reported either as dollar price changes or percentage price changes. The delta, gamma, and vega sensitivities that the toolbox computes are dollar sensitivities.

The functions `hjmsens` and `bdtsens` compute the delta, gamma, and vega sensitivities of instruments using an interest-rate tree. They also optionally return the calculated price for each instrument. The sensitivity functions require the same two input arguments used by the pricing functions (`HJMTree` and `HJMInstSet` for HJM; `BDTTree` and `BDTInstSet` for BDT).

Sensitivity functions calculate the dollar value of delta and gamma by shifting the observed forward yield curve by 100 basis points in each direction, and

the dollar value of vega by shifting the volatility process by 1%. To obtain the per-dollar value of the sensitivities, divide the dollar sensitivity by the price of the corresponding instrument.

HJM Sensitivities Example

The calling syntax for the function is:

```
[Delta, Gamma, Vega, Price] = hjmsens(HJMTree, HJMInstSet)
```

Use the previous example data to calculate the price of instruments.

```
load deriv.mat
[Delta, Gamma, Vega, Price] = hjmsens(HJMTree, HJMInstSet);

Warning: Not all cash flows are aligned with the tree. Result will
be approximated.
```

Note The warning appears because some of the cash flows for the second bond do not fall exactly on a tree node.

You can conveniently examine the sensitivities and the prices by arranging them into a single matrix.

```
All = [Delta, Gamma, Vega, Price]

All =

       -272.65       1029.90          0.00         98.72
       -347.43       1622.69         -0.04         97.53
         -8.08        643.40         34.07          0.05
       -272.65       1029.90          0.00         98.72
         -1.04          3.31             0        100.55
        294.97       6852.56         93.69          6.28
        -47.16       8459.99         93.69          0.05
       -282.05       1059.68          0.00          3.69
```

As with the prices, each row of the sensitivity vectors corresponds to the similarly indexed instrument in HJMInstSet. To view the *per-dollar*

sensitivities, divide each dollar sensitivity by the corresponding instrument price.

BDT Sensitivities Example

The calling syntax for the function is:

```
[Delta, Gamma, Vega, Price] = bdtsens(BDTTree, BDTInstSet);
```

Arrange the sensitivities and prices into a single matrix.

```
All = [Delta, Gamma, Vega, Price]

All =

    -232.67      803.71      -0.00       95.50
    -281.05     1181.93      -0.01       93.91
     -50.54      246.02       5.31        1.77
    -232.67      803.71          0       95.50
       0.84        2.45          0      100.49
      78.38      748.98      13.54        1.49
      -4.36      382.06       2.50        0.02
    -253.23      863.81          0        7.42
```

To view the *per-dollar sensitivities*, divide each dollar sensitivity by the corresponding instrument price.

```
All = [Delta ./ Price, Gamma ./ Price, Vega ./ Price, Price]

All =

      -2.44        8.42      -0.00       95.50
      -2.99       12.59      -0.00       93.91
     -28.63      139.34       3.01        1.77
      -2.44        8.42          0       95.50
       0.01        0.02          0      100.49
      52.73      503.92       9.11        1.49
    -177.89    15577.42     101.87        0.02
     -34.12      116.38          0        7.42
```

Calibrating Hull-White Model Using Market Data

The pricing of interest rate derivative securities relies on models that describe the underlying process. These interest rate models depend on one or more parameters that you must determine by matching the model predictions to the existing data available in the market. In the Hull-White model, there are two parameters related to the short rate process: mean reversion and volatility. Calibration is used to determine these parameters, such that the model can reproduce, as close as possible, the prices of caps or floors observed in the market. The calibration routines find the parameters that minimize the difference between the model price predictions and the market prices for caps and floors.

For a Hull-White model, the minimization is two dimensional, with respect to mean reversion (α) and volatility (σ). That is, calibrating the Hull-White model minimizes the difference between the model prices and market prices for caps and floors:

$$\frac{(\text{ModelPrice}(\alpha\ ,\ \sigma) - \text{MarketPrice})}{(\text{MarketPrice})}$$

Hull-White Model Calibration Example

Use market data to identify the implied volatility (σ) and mean reversion (α) coefficients needed to build a Hull-White tree to price an instrument. The ideal case is to use the volatilities of the caps or floors used to calculate `Alpha` (α) and `Sigma` (σ). This will most likely not be the case, so market data must be interpolated to obtain the required values.

Consider a cap with these parameters:

```
Settle = ' Jan-21-2008';
Maturity = 'Mar-21-2011';
Strike = 0.0690;
Reset = 4;
Principal = 1000;
Basis = 0;
```

The caplets for this example would fall in:

```
capletDates = cfdates(Settle, Maturity, Reset, Basis);
```

2-79

```
datestr(capletDates')

ans =

21-Mar-2008
21-Jun-2008
21-Sep-2008
21-Dec-2008
21-Mar-2009
21-Jun-2009
21-Sep-2009
21-Dec-2009
21-Mar-2010
21-Jun-2010
21-Sep-2010
21-Dec-2010
21-Mar-2011
```

In the best case, look up the market volatilities for caplets with a `Strike` = 0.0690, and maturities in each reset date listed, but the likelihood of finding these exact instruments is low. As a consequence, use data that is available in the market and interpolate to find appropriate values for the caplets.

Based on the market data, you have the cap information for different dates and strikes. Assume that instead of having the data for `Strike` = 0.0690, you have the data for `Strike1` = 0.0590 and `Strike2` = 0.0790

Maturity	Strike1 = 0.0590	Strike2 = 0.0790
21-Mar-2008	0.1533	0. 1526
21-Jun-2008	0.1731	0. 1730
21-Sep-2008	0. 1727	0. 1726
21-Dec-2008	0. 1752	0. 1747
21-Mar-2009	0. 1809	0. 1808
21-Jun-2009	0. 1809	0. 1792
21-Sep-2009	0. 1805	0. 1797
21-Dec-2009	0. 1802	0. 1794

Maturity	Strike1 = 0.0590	Strike2 = 0.0790
21-Mar-2010	0. 1802	0. 1733
21-Jun-2010	0. 1757	0. 1751
21-Sep-2010	0. 1755	0. 1750
21-Dec-2010	0. 1755	0. 1745
21-Mar-2011	0. 1726	0. 1719

The nature of this data lends itself to matrix nomenclature, which is perfect for MATLAB. `hwcalbycap` requires that the dates, the strikes, and the actual volatility be separated into three variables: `MarketStrike`, `MarketMat`, and `MarketVol`.

```
MarketStrike = [0.0590; 0.0790];
MarketMat = {'21-Mar-2008';
'21-Jun-2008';
'21-Sep-2008';
'21-Dec-2008';
'21-Mar-2009';
'21-Jun-2009';
'21-Sep-2009';
'21-Dec-2009';
'21-Mar-2010';
'21-Jun-2010';
'21-Sep-2010';
'21-Dec-2010';
'21-Mar-2011'};


MarketVol = [0.1533 0.1731 0.1727 0.1752 0.1809 0.1800 0.1805 0.1802 0.1735 0.1757 ...
            0.1755 0.1755 0.1726;   % First column in table corresponding to Strike1
            0.1526 0.1730 0.1726 0.1747 0.1808 0.1792 0.1797 0.1794 0.1733 0.1751 ...
            0.1750 0.1745 0.1719];  % Second column in table corresponding to Strike2
```

Complete the input arguments using this data for `RateSpec`:

```
Rates= [0.0627;
0.0657;
0.0691;
```

```
          0.0717;
          0.0739;
          0.0755;
          0.0765;
          0.0772;
          0.0779;
          0.0783;
          0.0786;
          0.0789;
          0.0792;
          0.0793];

ValuationDate = '21-Jan-2008';
EndDates = {'21-Mar-2008';'21-Jun-2008';'21-Sep-2008';'21-Dec-2008';...
            '21-Mar-2009';'21-Jun-2009';'21-Sep-2009';'21-Dec-2009';....
            '21-Mar-2010';'21-Jun-2010';'21-Sep-2010';'21-Dec-2010';....
            '21-Mar-2011';'21-Jun-2011'};
Compounding = 4;
Basis = 0;

RateSpec = intenvset('ValuationDate', ValuationDate, ...
'StartDates', ValuationDate, 'EndDates', EndDates, ...
'Rates', Rates, 'Compounding', Compounding, 'Basis', Basis);

RateSpec =

              FinObj: 'RateSpec'
         Compounding: 4
                Disc: [14x1 double]
               Rates: [14x1 double]
            EndTimes: [14x1 double]
          StartTimes: [14x1 double]
            EndDates: [14x1 double]
          StartDates: 733428
       ValuationDate: 733428
               Basis: 0
         EndMonthRule: 1
```

Call the calibration routine to find values for volatility parameters Alpha and Sigma. Use hwcalbycap to calculate the values of Alpha and Sigma based on market data. Internally, hwcalbycap calls the Optimization Toolbox™ function lsqnonlin. You can customize lsqnonlin by passing an optimization options structure created by optimoptions and then this can be passed to hwcalbycap using the name-value pair argument for OptimOptions. For example, optimoptions defines the target objective function tolerance as 100*eps and then calls hwcalbycap:

```
o=optimoptions('lsqnonlin','TolFun',100*eps);

[Alpha, Sigma] = hwcalbycap(RateSpec, MarketStrike, MarketMat, MarketVol,...
Strike, Settle, Maturity, 'Reset', Reset, 'Principal', Principal, 'Basis',...
Basis, 'OptimOptions', o)

Local minimum possible.

lsqnonlin stopped because the size of the current step is less than
the default value of the step size tolerance.

Warning: LSQNONLIN did not converge to an optimal solution. It exited with exitflag = 2.

> In hwcalbycapfloor at 93
  In hwcalbycap at 75

Alpha =

   1.0000e-06


Sigma =

    0.0127
```

The previous warning indicates that the conversion was not optimal. The search algorithm used by the Optimization Toolbox™ function lsqnonlin did not find a solution that conforms to all the constraints. To discern whether the solution is acceptable, look at the results of the optimization by specifying a third output (OptimOut) for hwcalbycap:

```
[Alpha, Sigma, OptimOut] = hwcalbycap(RateSpec, MarketStrike, MarketMat,...
```

```
MarketVol, Strike, Settle, Maturity, 'Reset', Reset, 'Principal', Principal,...
'Basis', Basis, 'OptimOptions', o);
```

The `OptimOut.residual` field of the `OptimOut` structure is the optimization residual. This value contains the difference between the Black caplets and those calculated during the optimization. You can use the `OptimOut.residual` value to calculate the percentual difference (error) compared to Black caplet prices and then decide whether the residual is acceptable. There is almost always some residual, so decide if parametrizing the market with a single value of `Alpha` and `Sigma` is acceptable.

Price caplets using market data and Black's formula to obtain reference caplet values. To determine the effectiveness of the optimization, calculate reference caplet values using Black's formula and the market data. Note, you must first interpolate the market data to obtain the caplets for calculation:

```
MarketMatNum = datenum(MarketMat);
[Mats, Strikes] = meshgrid(MarketMatNum, MarketStrike);
FlatVol = interp2(Mats, Strikes, MarketVol, datenum(Maturity), Strike, 'spline');
```

Compute the price of the cap using the Black model:

```
[CapPrice, Caplets] = capbyblk(RateSpec, Strike, Settle, Maturity, FlatVol,...
'Reset', Reset, 'Basis', Basis, 'Principal', Principal);
Caplets = Caplets(2:end)';

Caplets =

    0.3210
    1.6355
    2.4863
    3.1903
    3.4110
    3.2685
    3.2385
    3.4803
    3.2419
    3.1949
    3.2991
    3.3750
```

Compare optimized values and Black values and display graphically.
After calculating the reference values for the caplets, compare the values,
analytically and graphically, to determine whether the calculated single
values of `Alpha` and `Sigma` provide an adequate approximation:

```
OptimCaplets = Caplets+OptimOut.residual;


disp(' ');
disp('    Black76   Calibrated Caplets');
disp([Caplets                OptimCaplets])

plot(MarketMatNum(2:end), Caplets, 'or', MarketMatNum(2:end), OptimCaplets, '*b');
datetick('x', 2)
xlabel('Caplet Maturity');
ylabel('Caplet Price');
title('Black and Calibrated Caplets');
h = legend('Black Caplets', 'Calibrated Caplets');
set(h, 'color', [0.9 0.9 0.9]);
set(h, 'Location', 'SouthEast');
set(gcf, 'NumberTitle', 'off')
grid on

 Black76   Calibrated Caplets
    0.3210    0.3636
    1.6355    1.6603
    2.4863    2.4974
    3.1903    3.1874
    3.4110    3.4040
    3.2685    3.2639
    3.2385    3.2364
    3.4803    3.4683
    3.2419    3.2408
    3.1949    3.1957
    3.2991    3.2960
    3.3750    3.3663
```
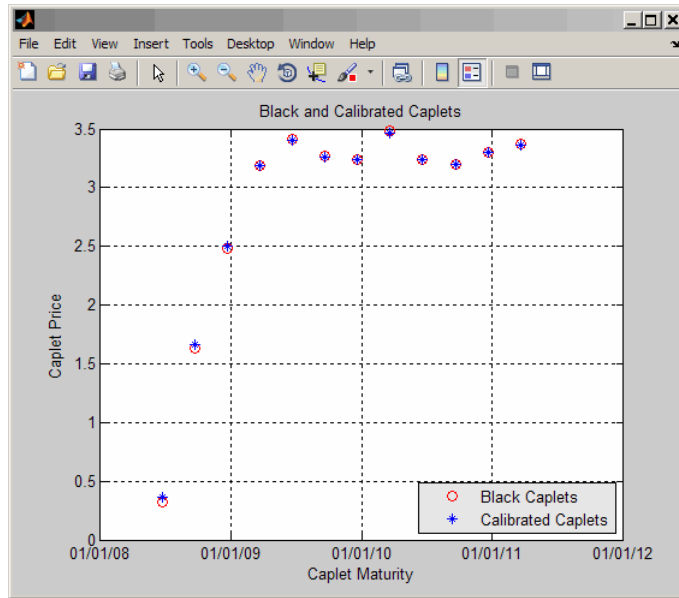
Compare cap prices using the Black, HW analytical, and HW tree models. Using the calculated caplet values, compare the prices of the corresponding cap using the Black model, Hull-White analytical, and Hull-White tree models. To calculate a Hull-White tree based on `Alpha` and `Sigma`, use these calibration routines:

- Black model:

```
CapPriceBLK = CapPrice;
```

- HW analytical model:

```
CapPriceHWAnalytical = sum(OptimCaplets);
```

- HW tree model to price the cap derived from the calibration process:

  1 Create `VolSpec` from the calibration parameters `Alpha` and `Sigma`:

```
VolDates    = EndDates;
VolCurve    = Sigma*ones(14,1);
AlphaDates  = EndDates;
AlphaCurve  = Alpha*ones(14,1);
```

```
HWVolSpec = hwvolspec(ValuationDate, VolDates, VolCurve,AlphaDates, AlphaCurve);
```

2 Create the `TimeSpec`:

```
HWTimeSpec  = hwtimespec(ValuationDate, EndDates, Compounding);
```

3 Build the HW tree using the `HW2000` method:

```
HWTree = hwtree(HWVolSpec, RateSpec, HWTimeSpec, 'Method', 'HW2000');
```

4 Price the cap:

```
Price = capbyhw(HWTree, Strike, Settle, Maturity, Reset, Basis, Principal);

disp('  ');
disp(['  CapPrice Black76 ..................:  ', num2str(CapPriceBLK,'%15.5f')]);
disp(['  CapPrice HW analytical..........:  ', num2str(CapPriceHWAnalytical,'%15.5f')]);
disp(['  CapPrice HW from capbyhw ..:  ', num2str(Price,'%15.5f')]);
disp('  ');

CapPrice Black76 ...........: 34.14220
CapPrice HW analytical.....: 34.18008
CapPrice HW from capbyhw ..: 34.14192
```

Price a portfolio of instruments using the calibrated HW tree. After
building a Hull-White tree, based on parameters calibrated from market data,
use `HWTree` to price a portfolio of these instruments:

- Two bonds

```
CouponRate = [0.07; 0.09];
Settle= ' Jan-21-2008';
Maturity = {'Mar-21-2010';'Mar-21-2011'};
Period = 1;
Face = 1000;
Basis = 0;
```

- Bond with an embedded American call option

```
CouponRateOEB = 0.08;
SettleOEB = ' Jan-21-2008';
MaturityOEB = 'Mar-21-2011';
```

```
OptSpec = 'call';
StrikeOEB = 950;
ExerciseDatesOEB = 'Mar-21-2011';
AmericanOpt= 1;
Period =1;
Face = 1000;
Basis =0;
```

To price this portfolio of instruments using the calibrated `HWTree`:

1 Use `instadd` to create the portfolio `InstSet`:

```
InstSet = instadd('Bond', CouponRate, Settle,  Maturity, Period, Basis, [], [], [], [], [], Face);
InstSet = instadd(InstSet,'OptEmBond', CouponRateOEB, SettleOEB, MaturityOEB, OptSpec,...
StrikeOEB,   ExerciseDatesOEB, 'AmericanOpt', AmericanOpt, 'Period', Period,...
'Face',Face,  'Basis', Basis);
```

2 Add the cap instrument used in the calibration:

```
SettleCap = ' Jan-21-2008';
MaturityCap = 'Mar-21-2011';
StrikeCap = 0.0690;
Reset = 4;
Principal = 1000;

InstSet = instadd(InstSet,'Cap', StrikeCap, SettleCap, MaturityCap, Reset, Basis, Principal);
```

3 Assign names to the portfolio instruments:

```
Names = {'7% Bond'; '8% Bond'; 'BondEmbCall'; '6.9% Cap'};
InstSet = instsetfield(InstSet, 'Index',1:4, 'FieldName', {'Name'}, 'Data', Names );
```

4 Examine the set of instruments contained in `InstSet`:

```
instdisp(InstSet)
```

IdxType	CoupRate	Settle	Mature	Period	Basis	EOMRule	IssueDate	1stCoupDate	LastCoupDate	StartDate	Face	Nam
1 Bond	0.07	21-Jan-2008	21-Mar-2010	1	0	NaN	NaN	NaN	NaN	NaN	1000	7% Bond
2 Bond	0.09	21-Jan-2008	21-Mar-2011	1	0	NaN	NaN	NaN	NaN	NaN	1000	8% Bond

```
IdxType CoupRate Settle Mature OptSpec Stke ExDate Per Basis EOMRule IssDate 1stCoupDate LstCoupDate Strt
3 OptEmBond 0.08 21-Jan-2008 21-Mar-2011 call 950  21-Jan-2008  21-Mar-2011  1  0  1  NaN  NaN NaN  NaN


Index Type Strike Settle     Maturity   CapReset Basis Principal Name
4 Cap  0.069  21-Jan-2008   21-Mar-2011   4      0     1000    6.9% Cap
```

5 Use `hwprice` to price the portfolio using the calibrated `HWTree`:

```
format bank
PricePortfolio = hwprice(HWTree, InstSet)


PricePortfolio =
         980.45
        1023.05
         945.73
          34.14
```

Interest-Rate Derivatives Using Closed-Form Solutions

Pricing Caps and Floors Using the Black Option Model

Caps and floors are contracts that allow the holder to be protected if interest rates rise or decrease. The Black model uses a forward price as an underlier in place of a spot price. The assumption is that the forward price at maturity of the option is log-normally distributed.

Closed-form solutions for pricing caps and floors using the Black model support the following tasks:

Task	Function
Price the interest rate caps using the Black option pricing model.	capbyblk
Price the interest rate floors using the Black option pricing model.	floorbyblk

Price Swaptions With Interest-Rate Models Using Simulation

Introduction

This example shows how to price European swaptions using interest-rate models in Financial Instruments Toolbox. Specifically, a Hull-White one factor model, a Linear Gaussian two-factor model, and a LIBOR Market Model are calibrated to market data and then used to generate interest-rate paths using Monte Carlo simulation.

The following sections set up the data that is then used with examples for "Simulate Interest-Rate Paths Using the Hull-White One-Factor Model" on page 2-97, "Simulate Interest-Rate Paths Using the Linear Gaussian Two-Factor Model" on page 2-100, and "Simulate Interest-Rate Paths Using the LIBOR Market Model" on page 2-104:

Construct Zero Curve

This example shows how to use `ZeroRates` for a zero curve that is hard-coded. You can also create a zero curve by bootstrapping the zero curve from market data (for example, deposits, futures/forwards, and swaps)

The hard-coded data for the zero curve is defined as:

```
Settle = datenum('21-Jul-2008');

% Zero Curve
CurveDates = daysadd(Settle,360*([1 3 5 7 10 20]),1);
ZeroRates = [1.9 2.6 3.1 3.5 4 4.3]'/100;

plot(CurveDates,ZeroRates)
datetick
title(['Zero Curve for ' datestr(Settle)]);
```
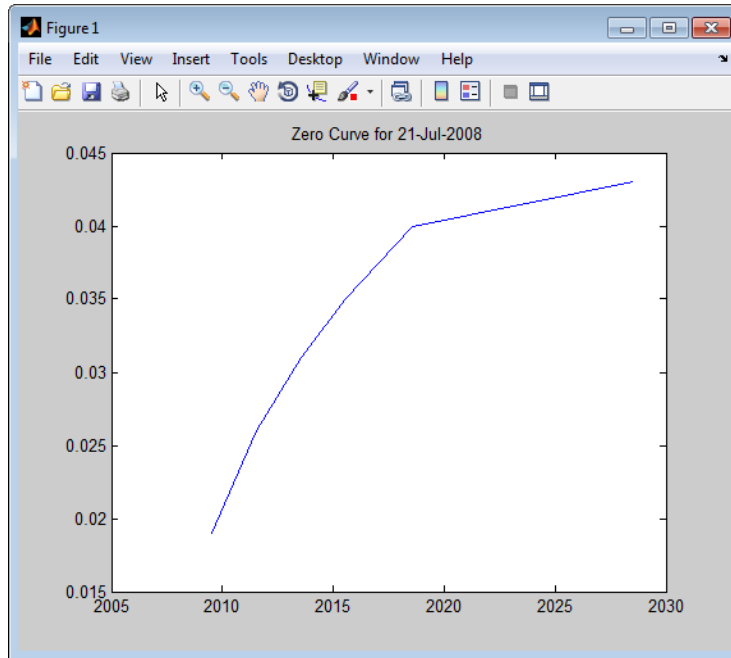
Construct an `IRCurve` object.

```
irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates)

irdc =

 IRDataCurve

    Type: Zero
     Settle: 733610 (21-Jul-2008)
   Compounding: 2
   Basis: 0 (actual/actual)
  InterpMethod: linear
   Dates: [6x1 double]
    Data: [6x1 double]
```

Construct the `RateSpec`.

```
RateSpec = intenvset('Rates',ZeroRates,'EndDates',CurveDates,'StartDate',Settle)
```

```
RateSpec =

            FinObj: 'RateSpec'
       Compounding: 2
              Disc: [6x1 double]
             Rates: [6x1 double]
          EndTimes: [6x1 double]
        StartTimes: [6x1 double]
          EndDates: [6x1 double]
        StartDates: 733610
     ValuationDate: 733610
             Basis: 0
       EndMonthRule: 1
```

Define Swaption Parameters

While Monte Carlo simulation is typically used to value more sophisticated derivatives (for example, Bermudan swaptions), in this example, the price of a European swaption is computed with an exercise date of 5 years and an underlying swap of 5 years.

```
InstrumentExerciseDate = datenum('21-Jul-2013');
InstrumentMaturity = datenum('21-Jul-2018');
InstrumentStrike = .045;
```

Compute the Black Model and the Swaption Volatility Matrix

Black's model is often used to price and quote European exercise interest-rate options, that is, caps, floors and swaptions. In the case of swaptions, Black's model is used to imply a volatility given the current observed market price. The following matrix shows the Black implied volatility for a range of swaption exercise dates (columns) and underlying swap maturities (rows).

```
SwaptionBlackVol = [22 21 19 17 15 13 12
    21 19 17 16 15 13 11
    20 18 16 15 14 12 11
    19 17 15 14 13 12 10
    18 16 14 13 12 11 10
    15 14 13 12 12 11 10
    13 13 12 11 11 10 9]/100;
```

```
ExerciseDates = [1:5 7 10];
Tenors = [1:5 7 10];

EurExDatesFull = repmat(daysadd(Settle,ExerciseDates*360,1)',...
    length(Tenors),1);
EurMatFull = reshape(daysadd(EurExDatesFull,...
    repmat(360*Tenors,1,length(ExerciseDates)),1),size(EurExDatesFull));
```

Select Calibration Instruments

Selecting the instruments to calibrate the model to is one of the tasks in calibration. For Bermudan swaptions, it is typical to calibrate to European swaptions that are co-terminal with the Bermudan swaption to be priced. In this case, all swaptions having an underlying tenor that matures before the maturity of the swaption to be priced (21-Jul-2018) are used in the calibration.

```
% Find the swaptions that expire on or before the maturity date of the
% sample swaption
relidx = find(EurMatFull <= InstrumentMaturity);
```

Compute Swaption Prices Using Black's Model

This example shows how to compute Swaption prices using Black's Model. The swaption prices are then used to compare the model's predicted values that are obtained from the calibration process.

To compute the swaption prices using Black's model:

```
SwaptionBlackPrices = zeros(size(SwaptionBlackVol));
SwaptionStrike = zeros(size(SwaptionBlackVol));

for iSwaption=1:length(ExerciseDates)
    for iTenor=1:length(Tenors)
        [~,SwaptionStrike(iTenor,iSwaption)] = swapbyzero(RateSpec,[NaN 0], Settle, EurMatFull(iTenor,iSwapt
            'StartDate',EurExDatesFull(iTenor,iSwaption),'LegReset',[1 1]);
        SwaptionBlackPrices(iTenor,iSwaption) = swaptionbyblk(RateSpec, 'call', SwaptionStrike(iTenor,iSwapt
            EurExDatesFull(iTenor,iSwaption), EurMatFull(iTenor,iSwaption), SwaptionBlackVol(iTenor,iSwaptio
    end
end
```

Define Simulation Parameters

This example shows how to use the simTermStructs method with HullWhite1F, LinearGaussian2F, and LiborMarketModel objects.

To demonstrate using the simTermStructs method with HullWhite1F, LinearGaussian2F, and LiborMarketModel objects, use the following simulation parameters:

```
nPeriods = 5;
DeltaTime = 1;
nTrials = 1000;

Tenor = (1:10)';

SimDates = daysadd(Settle,360*DeltaTime*(0:nPeriods),1)
SimTimes = diff(yearfrac(SimDates(1),SimDates))

% For 1 year periods and an evenly spaced tenor, the exercise row will be
% the sixth row and the swaption maturity will be the 5th column
exRow = 6;
endCol = 5;

SimDates =

     733610
     733975
     734340
     734705
     735071
     735436


SimTimes =

    1.0000
    1.0000
    1.0000
    1.0027
    1.0000
```

Simulate Interest-Rate Paths Using the Hull-White One-Factor Model

This example shows how to simulate interest-rate paths using the Hull-White one-factor model. Before beginning this example that uses a `HullWhite1F` model, make sure you have set up the data as described in:

- "Construct Zero Curve" on page 2-92

- "Define Swaption Parameters" on page 2-94

- "Compute the Black Model and the Swaption Volatility Matrix" on page 2-94

- "Select Calibration Instruments" on page 2-95

- "Compute Swaption Prices Using Black's Model" on page 2-95

- "Define Simulation Parameters" on page 2-96

The Hull-White one-factor model describes the evolution of the short rate and is specified using the zero curve, *alpha*, and *sigma* parameters for the equation

$$dr = [\theta(t) - a(t)r]dt + \sigma(t)dW$$

where:

dr is the change in the short-term interest rate over a small interval, *dt*.

r is the short-term interest rate.

Θ(t) is a function of time determining the average direction in which *r* moves, chosen such that movements in *r* are consistent with today's zero coupon yield curve.

a is the mean reversion rate.

dt is a small change in time.

σ is the annual standard deviation of the short rate.

W is the Brownian motion.

The Hull-White model is calibrated using the function `swaptionbyhw`, which constructs a trinomial tree to price the swaptions. Calibration consists of minimizing the difference between the observed market prices (computed above using the Black's implied swaption volatility matrix, see "Compute the Black Model and the Swaption Volatility Matrix" on page 2-94) and the model's predicted prices.

In this example, the Optimization Toolbox function `lsqnonlin` is used to find the parameter set that minimizes the difference between the observed and predicted values. However, other approaches (for example, simulated annealing) may be appropriate. Starting parameters and constraints for α and σ are set in the variables `x0`, `lb` and `ub`; these could also be varied depending upon the particular calibration approach.

Calibrate the set of parameters that minimize the difference between the observed and predicted values using `swaptionbyhw` and `lsqnonlin`.

```
TimeSpec = hwtimespec(Settle,daysadd(Settle,360*(1:11),1), 2);
HW1Fobjfun = @(x) SwaptionBlackPrices(relidx) - ...
    swaptionbyhw(hwtree(hwvolspec(Settle,'11-Aug-2015',x(2),'11-Aug-2015',x(1)), RateSpec, TimeSpec), 'call'
    EurExDatesFull(relidx), 0, Settle, EurMatFull(relidx));
options = optimset('disp','iter','MaxFunEvals',1000,'TolFun',1e-5);

% Find the parameters that minimize the difference between the observed and
% predicted prices
x0 = [.1 .01];
lb = [0 0];
ub = [1 1];
HW1Fparams = lsqnonlin(HW1Fobjfun,x0,lb,ub,options);

HW_alpha = HW1Fparams(1)
HW_sigma = HW1Fparams(2)
```

| | | | Norm of | First-order | |
Iteration	Func-count	f(x)	step	optimality	CG-iterations
0	3	0.481427		8.91	
1	6	0.321311	0.00637941	0.188	0
2	9	0.307638	0.0102855	17.1	0
3	12	0.306546	0.0128066	0.0889	0
4	15	0.306518	0.00342934	0.0245	0

```
    5        18      0.306518    5.66218e-05     9.99e-05            0

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the selected value of the function tolerance.


HW_alpha =

    0.0909


HW_sigma =

    0.0092
```

Construct the `HullWhite1F` model using the `HullWhite1F` constructor.

```
HW1F = HullWhite1F(RateSpec,HW_alpha,HW_sigma)

HW1F =

  HullWhite1F with properties:

    ZeroCurve: [1x1 IRDataCurve]
        Alpha: @(t,V)inAlpha
        Sigma: @(t,V)inSigma
```

Use Monte Carlo simulation to generate the interest-rate paths with `HullWhite1F.simTermStructs`.

```
HW1FSimPaths = HW1F.simTermStructs(nPeriods,'NTRIALS',nTrials,...
    'DeltaTime',DeltaTime,'Tenor',Tenor,'antithetic',true);
trialIdx = 1;
figure
surf(Tenor,SimDates,HW1FSimPaths(:,:,trialIdx))
datetick y keepticks keeplimits
title(['Evolution of the Zero Curve for Trial:' num2str(trialIdx) ' of Hull
xlabel('Tenor (Years)')
```
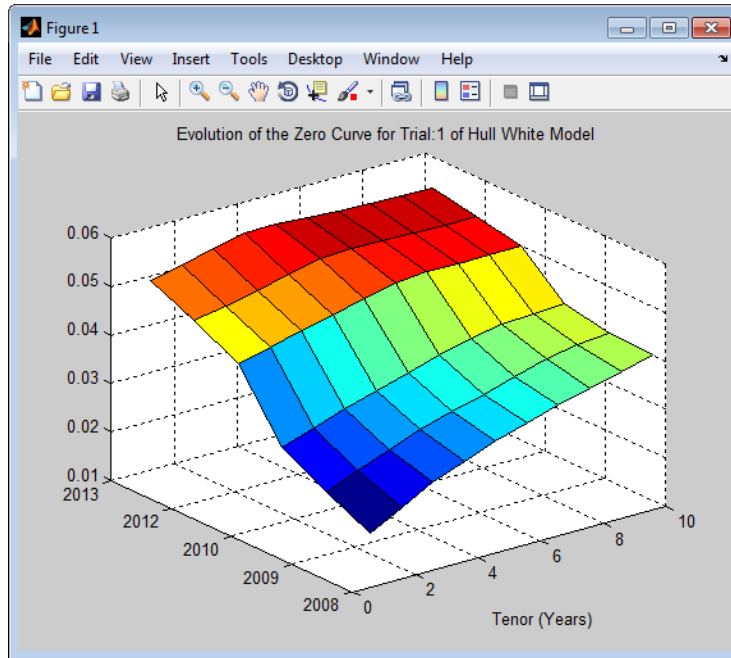
Price the European swaption.

```
DF = exp(bsxfun(@times,-HW1FSimPaths,repmat(Tenor',[nPeriods+1 1])));
SwapRate = (1 - DF(exRow,endCol,:))./sum(bsxfun(@times,1,DF(exRow,1:endCol,:)));
PayoffValue = 100*max(SwapRate-InstrumentStrike,0).*sum(bsxfun(@times,1,DF(exRow,1:endCol,:)));
RealizedDF = prod(exp(bsxfun(@times,-HW1FSimPaths(2:exRow,1,:),SimTimes(1:exRow-1))),1);
HW1F_SwaptionPrice = mean(RealizedDF.*PayoffValue);
```

```
 HW1F_SwaptionPrice =

    2.7649
```

Simulate Interest-Rate Paths Using the Linear Gaussian Two-Factor Model

This example shows how to simulate interest-rate paths using the Linear Gaussian two-factor model. Before beginning this example that uses a LinearGaussian2F model, make sure you have set up the data as described in:

The Linear Gaussian two-factor model (called the G2++ by Brigo and Mercurio, see "Interest-Rate Modeling Using Monte Carlo Simulation" on page C-12) is also a short rate model, but involves two factors. Specifically:

$$r(t) = x(t) + y(t) + \phi(t)$$

$$dx(t) = -a(t)x(t)dt + \sigma(t)dW_1(t), x(0) = 0$$

$$dy(t) = -b(t)y(t)dt + \eta(t)dW_2(t), y(0) = 0$$

where $dW_1(t)dW_2(t) = \rho dt$ is a two-dimensional Brownian motion with correlation ρ, and φ is a function chosen to match the initial zero curve.

The function swaptionbylg2f is used to compute analytic values of the swaption price for model parameters and consequently can be used to calibrate the model. Calibration consists of minimizing the difference between the observed market prices (computed above using the Black's implied swaption volatility matrix, see "Compute the Black Model and the Swaption Volatility Matrix" on page 2-94) and the model's predicted prices.

In this example, the approach is similar to "Simulate Interest-Rate Paths Using the Hull-White One-Factor Model" on page 2-97 and the Optimization Toolbox function lsqnonlin is used to minimize the difference between the observed swaption prices and the predicted swaption prices. However, other approaches (for example, simulated annealing) may also be appropriate. Starting parameters and constraints for a, b, η, ρ, and σ are set in the variables x0, lb, and ub; these could also be varied depending upon the particular calibration approach.

Calibrate the set of parameters that minimize the difference between the observed and predicted values using `swaptionbylg2f` and `lsqnonlin`.

```
G2PPobjfun = @(x) SwaptionBlackPrices(relidx) - swaptionbylg2f(irdc,x(1),x(2),x(3),x(4),x(5),SwaptionStrike(
    EurExDatesFull(relidx),EurMatFull(relidx),'Reset',1);
x0 = [.2 .1 .02 .01 -.5];
lb = [0 0 0 0 -1];
ub = [1 1 1 1 1];
LG2Fparams = lsqnonlin(G2PPobjfun,x0,lb,ub,options);
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
0	6	4.63874		27.7	
1	12	3.12695	0.116051	58.3	0
2	18	0.466886	0.110096	12.5	0
3	24	0.17667	0.0440836	3.2	0
4	30	0.17667	0.0688327	3.2	0
5	36	0.154169	0.0172082	0.246	0
6	42	0.150196	0.0344163	0.0356	0
7	48	0.143753	0.0688327	5.35	0
8	54	0.143753	0.0669431	5.35	0
9	60	0.140589	0.0167358	0.522	0
10	66	0.13942	0.0334716	0.42	0
11	72	0.13768	0.0669431	0.147	0
12	78	0.136894	0.133886	28.6	0
13	84	0.135024	0.0250289	3.56	0
14	90	0.13492	0.0139061	4.24	0
15	96	0.134879	0.00348578	0.232	0
16	102	0.134879	0.038352	0.232	0
17	108	0.134869	0.00836789	0.294	0

```
Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the selected value of the function tolerance.

LG2Fparams =

   0.1694    0.0355    0.0073    0.0106   -0.7291
```

Create the `G2PP` object and use Monte Carlo simulation to generate the interest-rate paths with `LinearGaussian2F.simTermStructs`.

```
LG2f_a = LG2Fparams(1);
LG2f_b = LG2Fparams(2);
LG2f_sigma = LG2Fparams(3);
LG2f_eta = LG2Fparams(4);
LG2f_rho = LG2Fparams(5);

G2PP = LinearGaussian2F(RateSpec,LG2f_a,LG2f_b,LG2f_sigma,LG2f_eta,LG2f_rho

G2PPSimPaths = G2PP.simTermStructs(nPeriods,'NTRIALS',nTrials,...
    'DeltaTime',DeltaTime,'Tenor',Tenor,'antithetic',true);

trialIdx = 1;
figure
surf(Tenor,SimDates,G2PPSimPaths(:,:,trialIdx))
datetick y keepticks keeplimits
title(['Evolution of the Zero Curve for Trial:' num2str(trialIdx) ' of G2++
xlabel('Tenor (Years)')
```
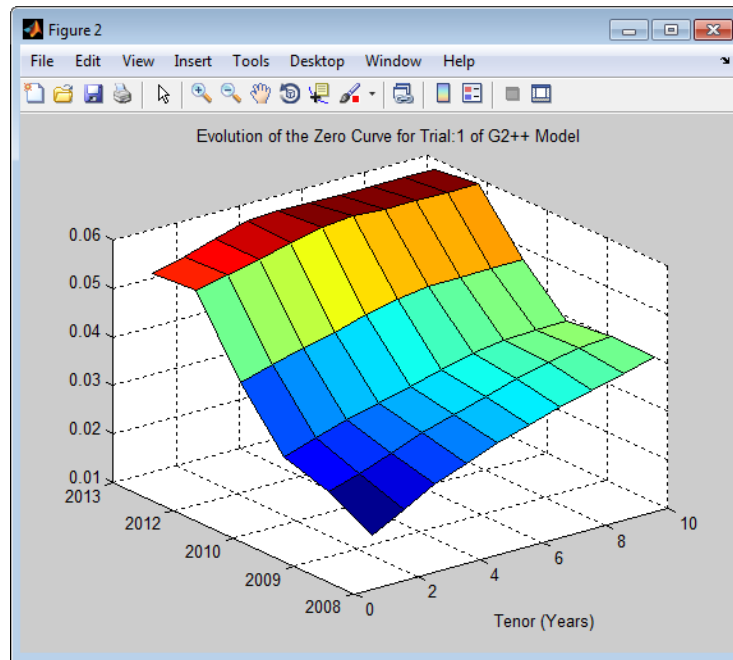
Price the European swaption.

```
DF = exp(bsxfun(@times,-G2PPSimPaths,repmat(Tenor',[nPeriods+1 1])));
SwapRate = (1 - DF(exRow,endCol,:))./sum(bsxfun(@times,1,DF(exRow,1:endCol,:)));
PayoffValue = 100*max(SwapRate-InstrumentStrike,0).*sum(bsxfun(@times,1,DF(exRow,1:endCol,:)));
RealizedDF = prod(exp(bsxfun(@times,-G2PPSimPaths(2:exRow,1,:),SimTimes(1:exRow-1))),1);
G2PP_SwaptionPrice = mean(RealizedDF.*PayoffValue)
```

```
G2PP_SwaptionPrice =

    2.6186
```

Simulate Interest-Rate Paths Using the LIBOR Market Model

This example shows how to simulate interest-rate paths using the LIBOR market model. Before beginning this example that uses a LiborMarketModel, make sure you have set up the data as described in:

The LIBOR Market Model (LMM) differs from short rate models in that it evolves a set of discrete forward rates. Specifically, the lognormal LMM specifies the following diffusion equation for each forward rate

$$\frac{dF_i(t)}{F_i} = -\mu_i dt + \sigma_i(t) dW_i$$

where:

W is an N-dimensional geometric Brownian motion with

$$dW_i(t)dW_j(t) = \rho_{ij}$$

The LMM relates the drifts of the forward rates based on no-arbitrage arguments. Specifically, under the Spot LIBOR measure, the drifts are expressed as

$$\mu_i(t) = -\sigma_i(t) \sum_{j=q(t)}^{i} \frac{\tau_j \rho_{i,j} \sigma_j(t) F_j(t)}{1 + \tau_j F_j(t)}$$

where:

τ_i is the time fraction associated with the i th forward rate

$q(t)$ is an index defined by the relation

$$T_{q(t)-1} < t < T_{q(t)}$$

and the Spot LIBOR numeraire is defined as

$$B(t) = P(t, T_{q(t)}) \prod_{n=0}^{q(t)-1} (1 + \tau_n F_n(T_n))$$

The choice with the LMM is how to model volatility and correlation and how to estimate the parameters of these models for volatility and correlation. In practice, you may use a combination of historical data (for example, observed correlation between forward rates) and current market data. For this example, only swaption data is used. Further, many different parameterizations of the volatility and correlation exist. For this example, two relatively straightforward parameterizations are used.

One of the most popular functional forms in the literature for volatility is:

$$\sigma_i(t) = \varphi_i(a(T_i - t) + b)e^{c(T_i - t)} + d$$

where φ adjusts the curve to match the volatility for the i th forward rate. For this example, all of the φ's will be taken to be 1. For the correlation, the following functional form is used:

$$\rho_{i,j} = e^{-\beta|i-j|}$$

Once the functional forms have been specified, these parameters need to be estimated using market data. One useful approximation, initially developed by Rebonato, is the following, which relates the Black volatility for a European swaption, given a set of volatility functions and a correlation matrix

$$(\upsilon_{\alpha,\beta}^{LFM})^2 = \sum_{i,j=\alpha+1}^{\beta} \frac{w_i(0)w_j(0)F_i(0)F_j(0)\rho_{i,j}}{S_{\alpha,\beta}(0)^2} \int_0^{T_\alpha} \sigma_i(t)\sigma_j(t)dt$$

where:

$$w_i(t) = \frac{\tau_i P(t, T_i)}{\sum_{k=\alpha+1}^{\beta} \tau_\kappa P(t, t_\kappa)}$$

This calculation is done using the function `blackvolbyrebonato` to compute analytic values of the swaption price for model parameters, and consequently, is then used to calibrate the model. Calibration consists of minimizing the difference between the observed implied swaption Black volatilities and the predicted Black volatilities.

In this example, the approach is similar to "Simulate Interest-Rate Paths Using the Hull-White One-Factor Model" on page 2-97 and "Simulate Interest-Rate Paths Using the Linear Gaussian Two-Factor Model" on page 2-100 where the Optimization Toolbox function `lsqnonlin` is used to minimize the difference between the observed swaption prices and the predicted swaption prices. However, other approaches (for example, simulated annealing) may also be appropriate. Starting parameters and constraints for a, b, d, and β are set in the variables `x0`, `lb`, and `ub`; these could also be varied depending upon the particular calibration approach.

Calibrate the set of parameters that minimize the difference between the observed and predicted values using `blackvolbyrebonato` and `lsqnonlin`.

```
nRates = 10;

CorrFunc = @(i,j,Beta) exp(-Beta*abs(i-j));

objfun = @(x) SwaptionBlackVol(relidx) - blackvolbyrebonato(RateSpec,...
    repmat({@(t) ones(size(t)).*(x(1)*t + x(2)).*exp(-x(3)*t) + x(4)},nRates-1,1),...
    CorrFunc(meshgrid(1:nRates-1)',meshgrid(1:nRates-1),x(5)),...
    EurExDatesFull(relidx),EurMatFull(relidx),'Period',1);

x0 = [.2 .05 1 .05 .2];
lb = [0 0 .5 0 .01];
ub = [1 1 2 .3 1];
LMMparams = lsqnonlin(objfun,x0,lb,ub,options)
```

Iteration	Func-count	f(x)	step	optimality	CG-iterations
0	6	0.212571		0.493	
1	12	0.0151459	0.229951	0.0352	0
2	18	0.0112531	0.193839	0.00108	0
3	24	0.00937762	0.310582	0.00688	0
4	30	0.00911658	0.0790873	7.03e-05	0
5	36	0.00911658	0.517804	7.03e-05	0

6	42	0.00907618	0.129451	0.0028	0
7	48	0.00904774	0.0428543	6.33e-05	0
8	54	0.00904497	0.258902	0.00771	0
9	60	0.00902206	0.0115259	4.5e-05	0
10	66	0.00901664	0.0647254	0.000899	0

```
Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the selected value of the function tolerance.

LMMparams =

    0.0797    0.2419    0.6073    0.0739    0.0100
```

Calculate `VolFunc` for the LMM object.

```
a = LMMparams(1);
b = LMMparams(2);
c = LMMparams(3);
d = LMMparams(4);

Beta = LMMparams(5);

VolFunc = repmat({@(t) ones(size(t)).*(a*t + b).*exp(-c*t) + d},nRates-1,1);
```

Plot the volatility function.

```
figure
fplot(VolFunc{1},[O 20])
title('Volatility Function')

CorrelationMatrix = CorrFunc(meshgrid(1:nRates-1)',meshgrid(1:nRates-1),Bet
```
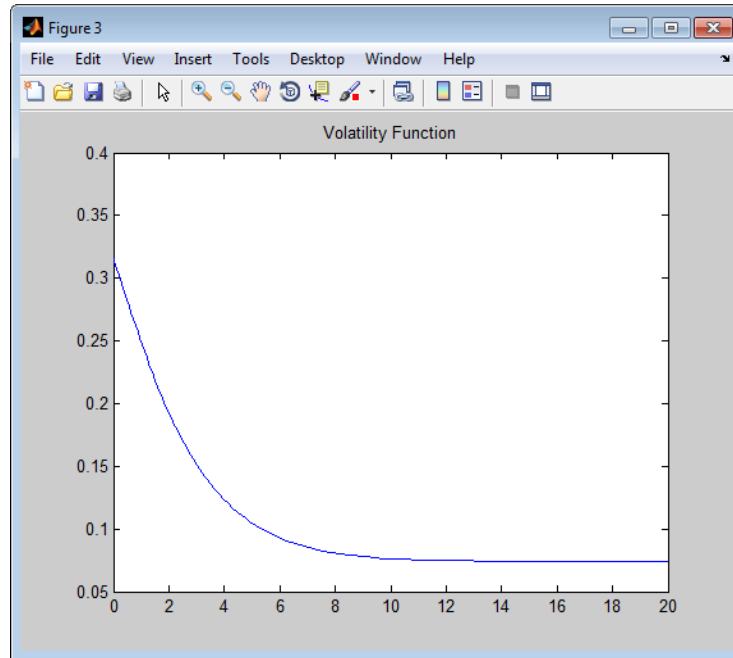
Inspect the correlation matrix.

```
disp('Correlation Matrix')
fprintf([repmat('%1.3f ',1,length(CorrelationMatrix)) ' \n'],CorrelationMatrix)
```

```
Correlation Matrix
1.000 0.990 0.980 0.970 0.961 0.951 0.942 0.932 0.923
0.990 1.000 0.990 0.980 0.970 0.961 0.951 0.942 0.932
0.980 0.990 1.000 0.990 0.980 0.970 0.961 0.951 0.942
0.970 0.980 0.990 1.000 0.990 0.980 0.970 0.961 0.951
0.961 0.970 0.980 0.990 1.000 0.990 0.980 0.970 0.961
0.951 0.961 0.970 0.980 0.990 1.000 0.990 0.980 0.970
0.942 0.951 0.961 0.970 0.980 0.990 1.000 0.990 0.980
0.932 0.942 0.951 0.961 0.970 0.980 0.990 1.000 0.990
0.923 0.932 0.942 0.951 0.961 0.970 0.980 0.990 1.000
```

Create the LMM object and use Monte Carlo simulation to generate the interest-rate paths with LiborMarketModel.simTermStructs.

```
LMM = LiborMarketModel(irdc,VolFunc,CorrelationMatrix,'Period',1);

[LMMZeroRates, ForwardRates] = LMM.simTermStructs(nPeriods,'nTrials',nTrials);

trialIdx = 1;
figure
tmpPlotData = LMMZeroRates(:,:,trialIdx);
tmpPlotData(tmpPlotData == 0) = NaN;
surf(Tenor,SimDates,tmpPlotData)
title(['Evolution of the Zero Curve for Trial:' num2str(trialIdx) ' of LIBOR Market Model'])
xlabel('Tenor (Years)')
```
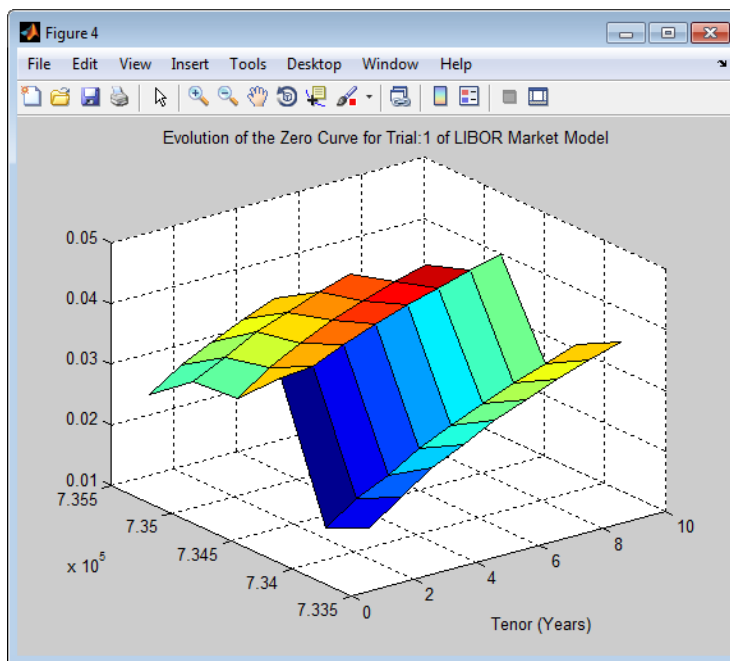

Price the European swaption.

```
DF = exp(bsxfun(@times,-LMMZeroRates,repmat(Tenor',[nPeriods+1 1])));
SwapRate = (1 - DF(exRow,endCol,:))./sum(bsxfun(@times,1,DF(exRow,1:endCol,:)));
PayoffValue = 100*max(SwapRate-InstrumentStrike,0).*sum(bsxfun(@times,1,DF(exRow,1:endCol,:)));
RealizedDF = prod(exp(bsxfun(@times,-LMMZeroRates(2:exRow,1,:),SimTimes(1:exRow-1))),1);
LMM_SwaptionPrice = mean(RealizedDF.*PayoffValue)
```

```
LMM_SwaptionPrice =

    2.8739
```

Compare Interest-Rate Modeling Results

This example shows how to compare the results for pricing a European swaptions with different interest-rate models.

Compare the results for pricing a European swaptions with interest-rate models using Monte Carlo simulation.

```
disp(' ')
fprintf('    # of Monte Carlo Trials: %8d\n'   , nTrials)
fprintf('    # of Time Periods/Trial: %8d\n\n' , nPeriods)
fprintf('HW1F European Swaption Price: %8.4f\n', HW1F_SwaptionPrice)
fprintf('LG2F Europesn Swaption Price: %8.4f\n', G2PP_SwaptionPrice)
fprintf(' LMM European Swaption Price: %8.4f\n', LMM_SwaptionPrice)


 # of Monte Carlo Trials:     1000
 # of Time Periods/Trial:        5

HW1F European Swaption Price:   2.7649
LG2F Europesn Swaption Price:   2.6186
 LMM European Swaption Price:   2.8739
```

References

Brigo, D. and F. Mercurio, *Interest Rate Models - Theory and Practice with Smile, Inflation and Credit* , Springer Finance, 2006.

Andersen, L. and V. Piterbarg, *Interest Rate Modeling*, Atlantic Financial Press. 2010.

Hull, J, *Options, Futures, and Other Derivatives*, Springer Finance, 2003.

Glasserman, P, *Monte Carlo Methods in Financial Engineering*, Prentice Hall, 2008.

Rebonato, R., K. McKay, and R. White, *The Sabr/Libor Market Model: Pricing, Calibration and Hedging for Complex Interest-Rate Derivatives*, John Wiley & Sons, 2010.

See Also   HullWhite1F **|** LinearGaussian2F **|** LiborMarketModel **|** HullWhite1F.simTermStructs **|** LinearGaussian2F.simTermStructs **|** LiborMarketModel.simTermStructs **|** capbylg2f | floorbylg2f | swaptionbylg2f | swaptionbyhw | blackvolbyrebonato | lsqnonlin

Graphical Representation of Trees

In this section...

Introduction

You can use the function `treeviewer` to display a graphical representation of a tree, allowing you to examine interactively the prices and rates on the nodes of the tree until maturity. To get started with this process, first load the data file `deriv.mat` included in this toolbox.

```
load deriv.mat
```

Note `treeviewer` price tree diagrams follow the convention that increasing prices appear on the upper branch of a tree and, consequently, decreasing prices appear on the lower branch. Conversely, for interest rate displays, *decreasing* interest rates appear on the upper branch (prices are rising) and *increasing* interest rates on the lower branch (prices are falling).

For information on the use of `treeviewer` to observe interest rate movement, see "Observing Interest Rates" on page 2-113. For information on using `treeviewer` to observe the movement of prices, see "Observing Instrument Prices" on page 2-117.

Observing Interest Rates

If you provide the name of an interest rate tree to the `treeviewer` function, it displays a graphical view of the path of interest rates. For example, here is the `treeviewer` representation of all the rates along both the up and down branches of `HJMTree`.

```
treeviewer(HJMTree)
```
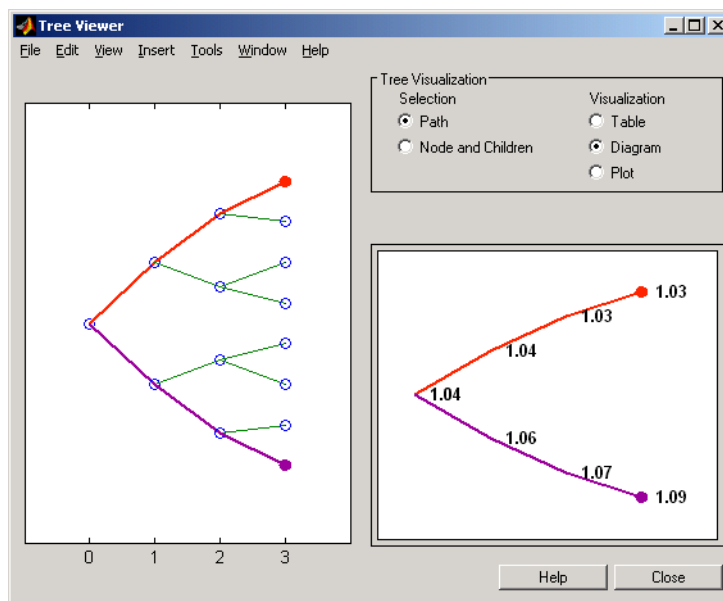
The example in "Isolating a Specific Node for a CRRTree" on page 3-19 used `bushpath` to find the path of forward rates along an HJM tree by taking the first branch up and then two branches down the rate tree.

```
FRates = bushpath(HJMTree.FwdTree, [1 2 2])

FRates =

    1.0356
    1.0364
    1.0526
    1.0674
```

With the `treeviewer` function you can display the identical information by clicking along the same sequence of nodes, as shown next.
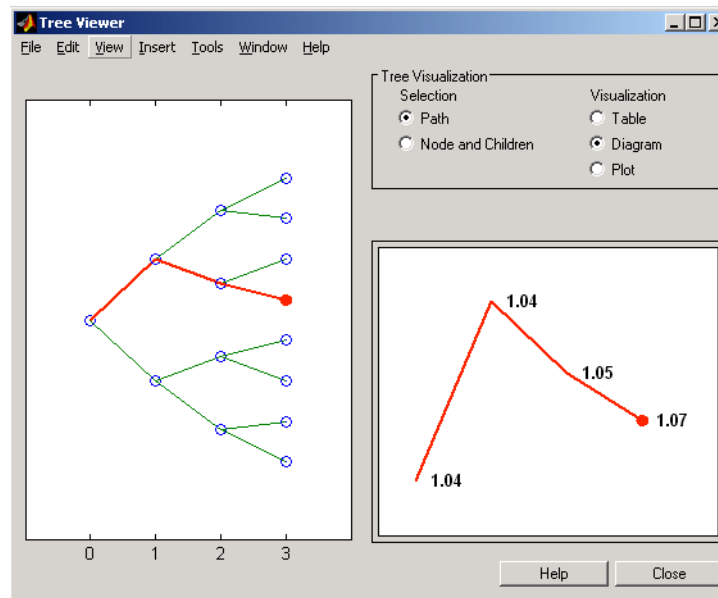
Next is a `treeviewer` representation of interest rates along several branches of `BDTTree`.

```
treeviewer(BDTTree)
```
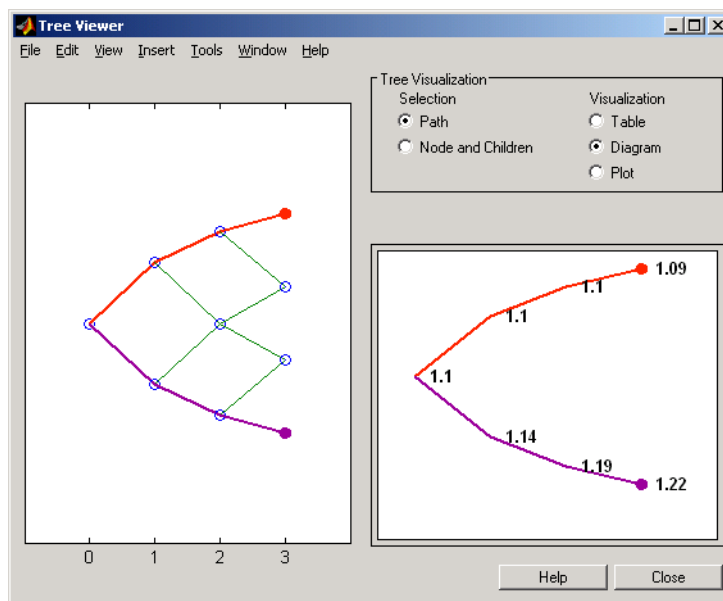
Note When using `treeviewer` with recombining trees, such as BDT, BK, and HW, you must click each node in succession from the beginning to the end. Because these trees can recombine, `treeviewer` is unable to complete the path automatically.

The example in "Isolating a Specific Node for a CRRTree" on page 3-19 used `treepath` to find the path of interest rates taking the first branch up and then two branches down the rate tree.

```
FRates = treepath(BDTTree.FwdTree, [1 2 2])

FRates =

    1.1000
    1.0979
    1.1377
    1.1606
```

You can display the identical information by clicking along the same sequence of nodes, as shown next.
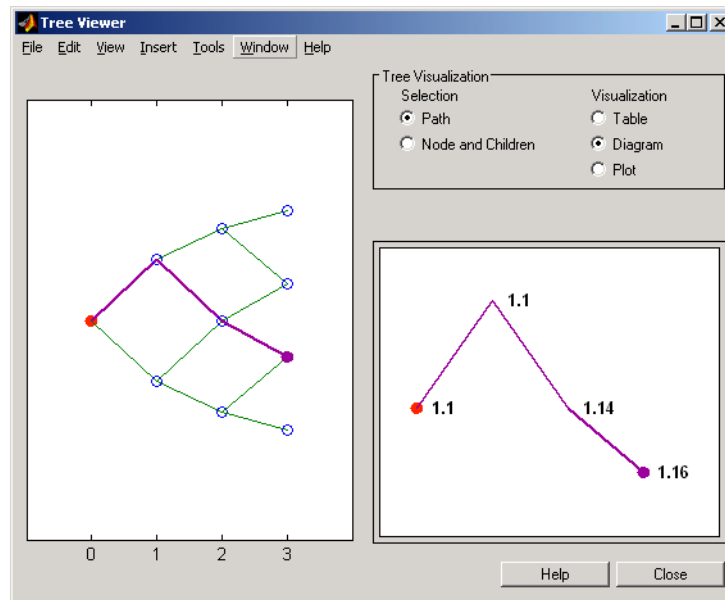
Observing Instrument Prices

To use treeviewer to display a tree of instrument prices, provide the name of an instrument set along with the name of a price tree in your call to treeviewer, for example:

```
load deriv.mat
[Price, PriceTree] = hjmprice(HJMTree, HJMInstSet);
treeviewer(PriceTree, HJMInstSet)
```

With `treeviewer` you select *each instrument individually* in the instrument portfolio for display.
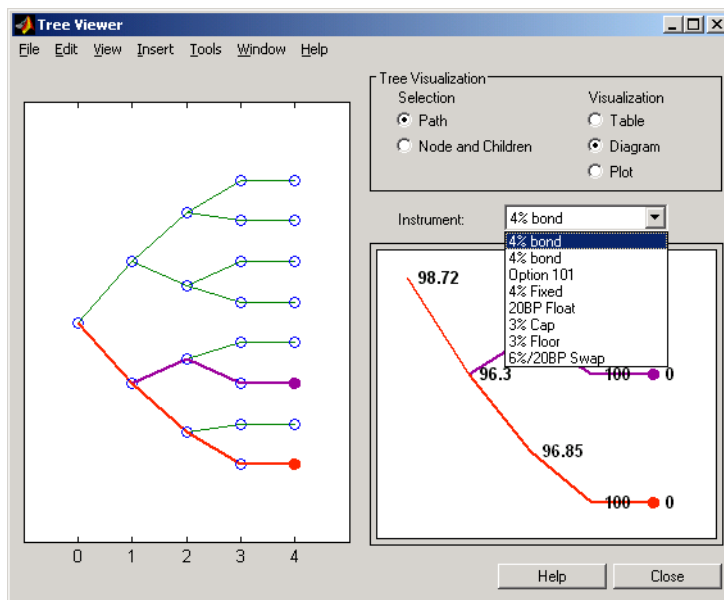
You can use an analogous process to view instrument prices based on the BDT interest rate tree included in `deriv.mat`.

```
load deriv.mat
[BDTPrice, BDTPriceTree] = bdtprice(BDTTree, BDTInstSet);
treeviewer(BDTPriceTree, BDTInstSet)
```
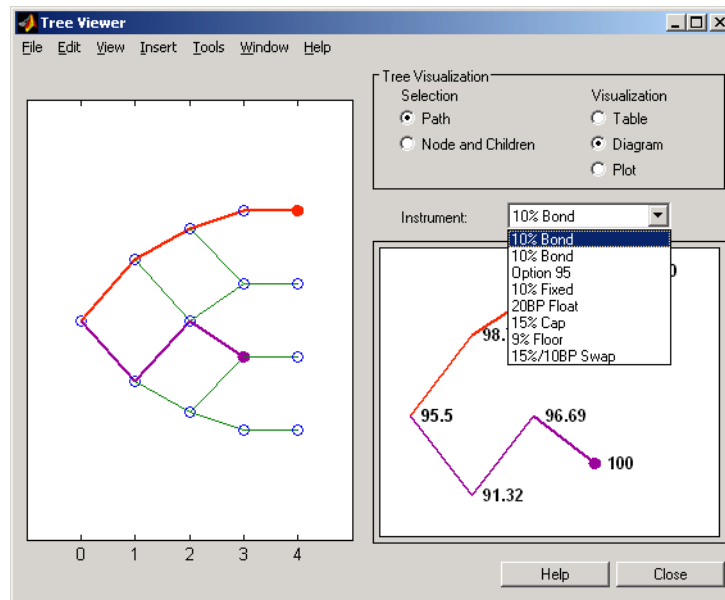
Valuation Date Prices

You can use treeviewer instrument-by-instrument to observe instrument prices through time. For the first 4% bond in the HJM instrument portfolio, treeviewer indicates a valuation date price of 98.72, the same value obtained by accessing the PriceTree structure directly.
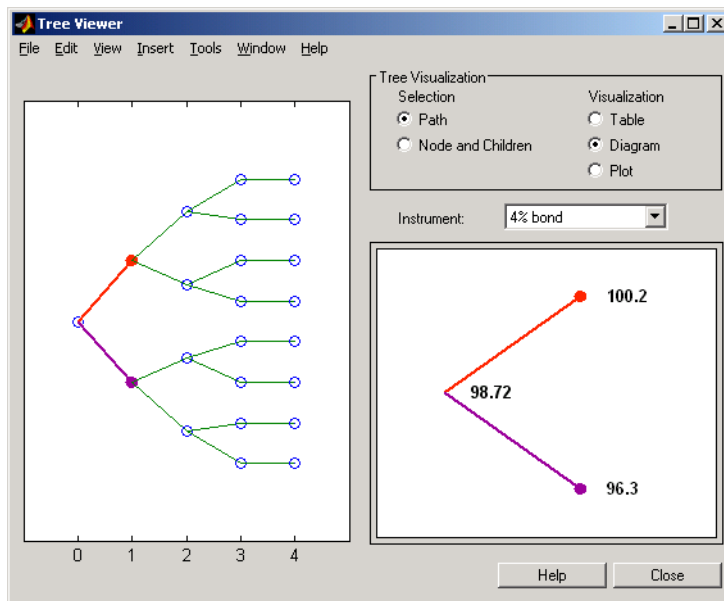
As a further example, look at the sixth instrument in the price vector, the 3% cap. At the valuation date, its value obtained directly from the structure is 6.2831. Use `treeviewer` on this instrument to confirm this price.
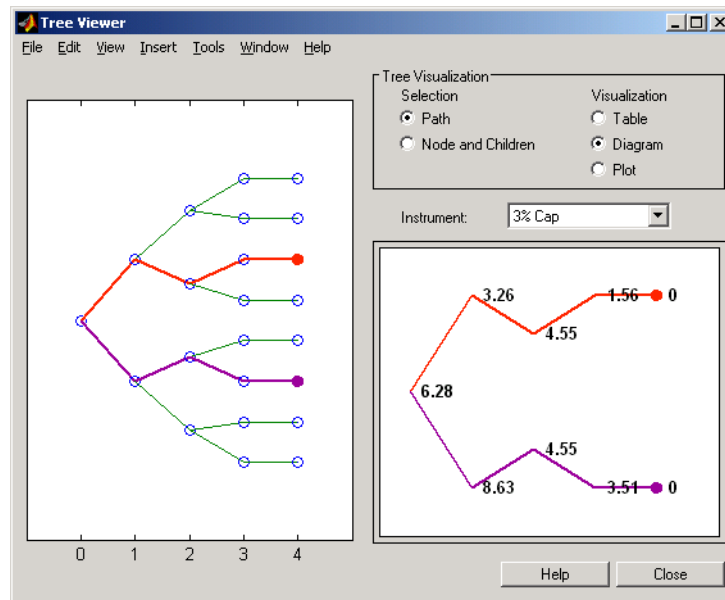
Additional Observation Times

The second node represents the first-rate observation time, `tObs = 1`. This node displays two states, one representing the branch going up and the other one representing the branch going down.

Examine the prices of the node corresponding to the up branch.

```
PriceTree.PBush{2}(:,:,1)

ans =

  100.1563
   99.7309
    0.1007
  100.1563
  100.3782
    3.2594
    0.1007
    3.5597
```

As before, you can use `treeviewer`, this time to examine the price for the 4% bond on the up branch. `treeviewer` displays a price of 100.2 for the first node of the up branch, as expected.
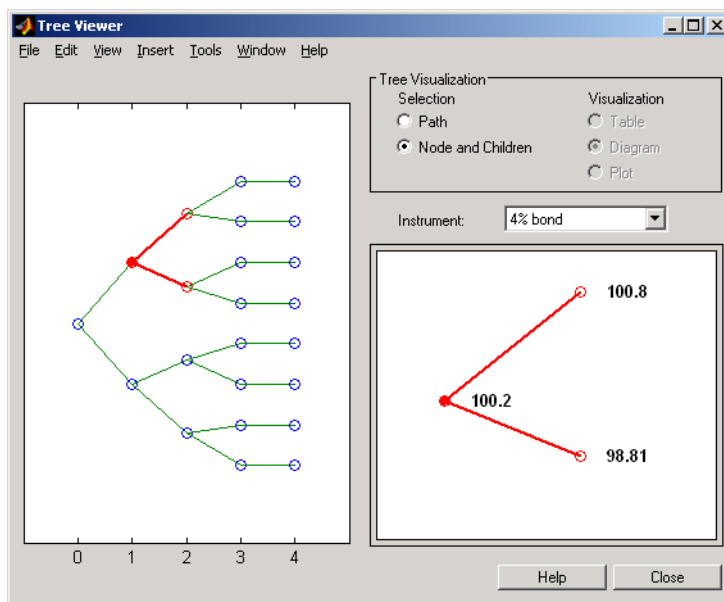
Now examine the corresponding down branch.

```
PriceTree.PBush{2}(:,:,2)

ans =

   96.3041
   94.1986
        0
   96.3041
  100.3671
    8.6342
        0
   -0.3923
```

Use `treeviewer` once again, now to observe the price of the 4% bond on the down branch. The displayed price of 96.3 conforms to the price obtained from direct access of the `PriceTree` structure. You may continue this process as far along the price tree as you want.

3

Equity Derivatives

Understanding Equity Trees

Introduction

Financial Instruments Toolbox software supports three types of recombining tree models to represent the evolution of stock prices:

- Cox-Ross-Rubinstein (CRR) model

- Equal probabilities (EQP) model

- Leisen-Reimer (LR) model

- Implied trinomial tree (ITT) model

For a discussion of recombining trees, see "Rate and Price Trees" on page 2-21.

The CRR, EQP, LR, and ITT models are examples of discrete time models. A discrete time model divides time into discrete bits; prices can only be computed at these specific times.

The CRR model is one of the most common methods used to model the evolution of stock processes. The strength of the CRR model lies in its simplicity. It is a good model when dealing with many tree levels. The CRR model yields the correct expected value for each node of the tree and provides a good approximation for the corresponding local volatility. The approximation becomes better as the number of time steps represented in the tree is increased.

The EQP model is another discrete time model. It has the advantage of building a tree with the exact volatility in each tree node, even with small

numbers of time steps. It also provides better results than CRR in some given trading environments, for example, when stock volatility is low and interest rates are high. However, this additional precision causes increased complexity, which is reflected in the number of calculations required to build a tree.

The LR model is another discrete time model. It has the advantage of producing estimates close to the Black-Scholes model using only a small number of steps, while also minimizing the oscillation.

The ITT model is a CRR-style implied trinomial tree which takes advantage of prices quoted from liquid options in the market with varying strikes and maturities to build a tree that more accurately represents the market. An ITT model is commonly used to price exotic options in such a way that they are consistent with the market prices of standard options.

Building Equity Binary Trees

The tree of stock prices is the fundamental unit representing the evolution of the price of a stock over a given period of time. The MATLAB functions `crrtree`, `eqptree`, and `lrtree` create CRR trees, EQP trees, and LR trees, respectively. These functions create an output tree structure along with information about the parameters used for creating the tree.

The functions `crrtree`, `eqptree`, and `lrtree` take three structures as input arguments:

- The stock parameter structure `StockSpec`
- The interest-rate term structure `RateSpec`
- The tree time layout structure `TimeSpec`

Calling Sequence for Equity Binary Trees

The calling syntax for `crrtree` is:

```
CRRTree = crrtree (StockSpec, RateSpec, TimeSpec)
```

Similarly, the calling syntax for `eqptree` is:

```
EQPTree = eqptree (StockSpec, RateSpec, TimeSpec)
```

And, the calling syntax for `lrtree` is:

```
LRTree = lrtree(StockSpec, RateSpec, TimeSpec, Strike)
```

All three functions require the structures `StockSpec`, `RateSpec`, and `TimeSpec` as input arguments:

- `StockSpec` is a structure that specifies parameters of the stock whose price evolution is represented by the tree. This structure, created using the function `stockspec`, contains information such as the stock's original price, its volatility, and its dividend payment information.

- `RateSpec` is the interest-rate specification of the initial rate curve. Create this structure with the function `intenvset`.

- `TimeSpec` is the tree time layout specification. Create these structures with the functions `crrtimespec`, `eqptimespec`, and `lrtimespec`. The structures contain information regarding the mapping of relevant dates into the tree structure, plus the number of time steps used for building the tree.

Specifying the Stock Structure for Equity Binary Trees

The structure `StockSpec` encapsulates the stock-specific information required for building the binary tree of an individual stock's price movement.

You generate `StockSpec` with the function `stockspec`. This function requires two input arguments and accepts up to three additional input arguments that depend on the existence and type of dividend payments.

The syntax for calling `stockspec` is:

```
StockSpec = stockspec(Sigma, AssetPrice, DividendType, ...
DividendAmounts, ExDividendDates)
```

where:

- `Sigma` is the decimal annual volatility of the underlying security.

- `AssetPrice` is the price of the stock at the valuation date.

- `DividendType` is a string specifying the type of dividend paid by the stock. Allowed values are `cash`, `constant`, or `continuous`.

- `DividendAmounts` has a value that depends on the specification of `DividendType`. For `DividendType` cash, `DividendAmounts` is a vector of cash dividends. For `DividendType` constant, it is a vector of constant annualized dividend yields. For `DividendType` continuous, it is a scalar representing a continuously annualized dividend yield.

- `ExDividendDates` also has a value that depends on the nature of `DividendType`. For `DividendType` cash or constant, `ExDividendDates` is vector of dividend dates. For `DividendType` continuous, `ExDividendDates` is ignored.

Stock Structure Example Using a Binary Tree

Consider a stock with a price of $100 and an annual volatility of 15%. Assume that the stock pays three cash $5.00 dividends on dates January 01, 2003; July 01, 2003; and January 01, 2004. You specify these parameters in MATLAB as:

```
Sigma = 0.15;
AssetPrice = 100;
DividendType = 'cash';
DividendAmounts = [5; 5; 5];
ExDividendDates = {'jan-01-2004', 'july-01-2005', 'jan-01-2006'};

StockSpec = stockspec(Sigma, AssetPrice, DividendType, ...
DividendAmounts, ExDividendDates)

StockSpec =

              FinObj: 'StockSpec'
               Sigma: 0.1500
          AssetPrice: 100
        DividendType: 'cash'
     DividendAmounts: [3x1 double]
     ExDividendDates: [3x1 double]
```

Specifying the Interest-Rate Term Structure for Equity Binary Trees

The `RateSpec` structure defines the interest rate environment used when building the stock price binary tree. "Modeling the Interest-Rate Term Structure" on page 2-33 explains how to create these structures using the

function `intenvset`, given the interest rates, the starting and ending dates for each rate, and the compounding value.

Specifying the Tree-Time Term Structure for Equity Binary Trees

The `TimeSpec` structure defines the tree layout of the binary tree:

- It maps the valuation and maturity dates to their corresponding times.

- It defines the time of the levels of the tree by dividing the time span between valuation and maturity into equally spaced intervals. By specifying the number of intervals, you define the granularity of the tree time structure.

The syntax for building a `TimeSpec` structure is:

```
TimeSpec = crrtimespec(ValuationDate, Maturity, NumPeriods)
TimeSpec = eqptimespec(ValuationDate, Maturity, NumPeriods)

TimeSpec = lrtimespec(ValuationDate, Maturity, NumPeriods)
```

where:

- `ValuationDate` is a scalar date marking the pricing date and first observation in the tree (location of the root node). You enter `ValuationDate` either as a serial date number (generated with `datenum`) or a date string.

- `Maturity` is a scalar date marking the maturity of the tree, entered as a serial date number or a date string.

- `NumPeriods` is a scalar defining the number of time steps in the tree; for example, `NumPeriods = 10` implies 10 time steps and 11 tree levels (0, 1, 2, ..., 9, 10).

TimeSpec Example Using a Binary Tree

Consider building a CRR tree, with a valuation date of January 1, 2003, a maturity date of January 1, 2008, and 20 time steps. You specify these parameters in MATLAB as:

```
ValuationDate = 'Jan-1-2003';
Maturity = 'Jan-1-2008';
NumPeriods = 20;
```

```
TimeSpec = crrtimespec(ValuationDate, Maturity, NumPeriods)

TimeSpec =

            FinObj: 'BinTimeSpec'
     ValuationDate: 731582
          Maturity: 733408
        NumPeriods: 20
             Basis: 0
       EndMonthRule: 1
              tObs: [1x21 double]
              dObs: [1x21 double]
```

Two vector fields in the `TimeSpec` structure are of particular interest: `dObs` and `tObs`. These two fields represent the observation times and corresponding dates of all tree levels, with `dObs(1)` and `tObs(1)`, respectively, representing the root node (`ValuationDate`), and `dObs(end)` and `tObs(end)` representing the last tree level (`Maturity`).

Note There is no relationship between the dates specified for the tree and the implied tree level times, and the maturities specified in the interest rate term structure. The rates in `RateSpec` are interpolated or extrapolated as required to meet the time distribution of the tree.

Examples of Binary Tree Creation

You can now use the `StockSpec` and `TimeSpec` structures described previously to build an equal probability tree (`EQPTree`), a CRR tree (`CRRTree`), or a LR tree (`LRTree`). First, you must define the interest rate term structure. For this example, assume that the interest rate is fixed at 10% annually between the valuation date of the tree (January 1, 2003) until its maturity.

```
ValuationDate = 'Jan-1-2003';
Maturity = 'Jan-1-2008';
Rate = 0.1;
RateSpec = intenvset('Rates', Rate, 'StartDates', ...
ValuationDate, 'EndDates', Maturity, 'Compounding', -1);
```

To build a `CRRTree`, enter:

```
CRRTree = crrtree(StockSpec, RateSpec, TimeSpec)

CRRTree =

       FinObj: 'BinStockTree'
       Method: 'CRR'
    StockSpec: [1x1 struct]
     TimeSpec: [1x1 struct]
     RateSpec: [1x1 struct]
         tObs: [1x21 double]
         dObs: [1x21 double]
        STree: {1x21 cell}
      UpProbs: [1x20 double]
```

To build an `EQPTree`, enter:

```
EQPTree = eqptree(StockSpec, RateSpec, TimeSpec)

EQPTree =

       FinObj: 'BinStockTree'
       Method: 'EQP'
    StockSpec: [1x1 struct]
     TimeSpec: [1x1 struct]
     RateSpec: [1x1 struct]
         tObs: [1x21 double]
         dObs: [1x21 double]
        STree: {1x21 cell}
      UpProbs: [1x20 double]
```

Building Implied Trinomial Trees

The tree of stock prices is the fundamental unit representing the evolution of the price of a stock over a given period of time. The MATLAB function `itttree` creates an output tree structure along with the information about the parameters used to create the tree.

The function `itttree` takes four structures as input arguments:

- The stock parameter structure `StockSpec`

- The interest-rate term structure `RateSpec`

- The tree time layout structure `TimeSpec`

- The stock option specification structure `StockOptSpec`

Calling Sequence for Implied Trinomial Trees

The calling syntax for `itttree` is:

```
ITTTree = itttree (StockSpec,RateSpec,TimeSpec,StockOptSpec)
```

- `StockSpec` is a structure that specifies parameters of the stock whose price evolution is represented by the tree. This structure, created using the function `stockspec`, contains information such as the stock's original price, its volatility, and its dividend payment information.

- `RateSpec` is the interest-rate specification of the initial rate curve. Create this structure with the function `intenvset`.

- `TimeSpec` is the tree time layout specification. Create these structures with the function `itttimespec`. This structure contains information regarding the mapping of relevant dates into the tree structure, plus the number of time steps used for building the tree.

- `StockOptSpec` is a structure containing parameters of European stock options instruments. Create this structure with the function `stockoptspec`.

Specifying the Stock Structure for Implied Trinomial Trees

The structure `StockSpec` encapsulates the stock-specific information required for building the trinomial tree of an individual stock's price movement.

You generate `StockSpec` with the function `stockspec`. This function requires two input arguments and accepts up to three additional input arguments that depend on the existence and type of dividend payments.

The syntax for calling `stockspec` is:

```
StockSpec = stockspec(Sigma, AssetPrice, DividendType, ...
DividendAmounts, ExDividendDates)
```

where:

- `Sigma` is the decimal annual volatility of the underlying security.

- `AssetPrice` is the price of the stock at the valuation date.

- `DividendType` is a string specifying the type of dividend paid by the stock. Allowed values are `cash`, `constant`, or `continuous`.

- `DividendAmounts` has a value that depends on the specification of `DividendType`. For `DividendType` `cash`, `DividendAmounts` is a vector of cash dividends. For `DividendType` `constant`, it is a vector of constant annualized dividend yields. For `DividendType` `continuous`, it is a scalar representing a continuously annualized dividend yield.

- `ExDividendDates` also has a value that depends on the nature of `DividendType`. For `DividendType` `cash` or `constant`, `ExDividendDates` is vector of dividend dates. For `DividendType` `continuous`, `ExDividendDates` is ignored.

Stock Structure Example Using an Implied Trinomial Tree

Consider a stock with a price of $100 and an annual volatility of 12%. Assume that the stock is expected to pay a dividend yield of 6%. You specify these parameters in MATLAB as:

```
So=100;
DividendYield = 0.06;
Sigma=.12;

StockSpec = stockspec(Sigma, So, 'continuous', DividendYield)

StockSpec =

            FinObj: 'StockSpec'
             Sigma: 0.1200
        AssetPrice: 100
      DividendType: 'continuous'
   DividendAmounts: 0.0600
   ExDividendDates: []
```

Specifying the Interest-Rate Term Structure for Implied Trinomial Trees

The structure `RateSpec` defines the interest rate environment used when building the stock price binary tree. "Modeling the Interest-Rate Term Structure" on page 2-33 explains how to create these structures using the function `intenvset`, given the interest rates, the starting and ending dates for each rate, and the compounding value.

Specifying the Tree-Time Term Structure for Implied Trinomial Trees

The `TimeSpec` structure defines the tree layout of the trinomial tree:

- It maps the valuation and maturity dates to their corresponding times.

- It defines the time of the levels of the tree by dividing the time span between valuation and maturity into equally spaced intervals. By specifying the number of intervals, you define the granularity of the tree time structure.

The syntax for building a `TimeSpec` structure is:

```
TimeSpec = itttimespec(ValuationDate, Maturity, NumPeriods)
```

where:

- `ValuationDate` is a scalar date marking the pricing date and first observation in the tree (location of the root node). You enter `ValuationDate` either as a serial date number (generated with `datenum`) or a date string.

- `Maturity` is a scalar date marking the maturity of the tree, entered as a serial date number or a date string.

- `NumPeriods` is a scalar defining the number of time steps in the tree; for example, `NumPeriods = 10` implies 10 time steps and 11 tree levels (0, 1, 2, ..., 9, 10).

TimeSpec Example Using an Implied Trinomial Tree

Consider building an ITT tree, with a valuation date of January 1, 2006, a maturity date of January 1, 2008, and four time steps. You specify these parameters in MATLAB as:

```
ValuationDate = '01-01-2006';
EndDate = '01-01-2008';
NumPeriods = 4;

TimeSpec = itttimespec(ValuationDate, EndDate, NumPeriods)

TimeSpec =

           FinObj: 'ITTTimeSpec'
    ValuationDate: 732678
         Maturity: 733408
       NumPeriods: 4
            Basis: 0
      EndMonthRule: 1
             tObs: [0 0.5000 1 1.5000 2]
             dObs: [732678 732860 733043 733225 733408]
```

Two vector fields in the `TimeSpec` structure are of particular interest: `dObs` and `tObs`. These two fields represent the observation times and corresponding dates of all tree levels, with `dObs(1)` and `tObs(1)`, respectively, representing the root node (`ValuationDate`), and `dObs(end)` and `tObs(end)` representing the last tree level (`Maturity`).

Specifying the Option Stock Structure for Implied Trinomial Trees

The `StockOptSpec` structure encapsulates the option-stock-specific information required for building the implied trinomial tree. You generate `StockOptSpec` with the function `stockoptspec`. This function requires five input arguments. An optional sixth argument `InterpMethod`, specifying the interpolation method, can be included. The syntax for calling `stockoptspec` is:

```
[StockOptSpec] = stockoptspec(OptPrice, Strike, Settle, Maturity, OptSpec)
```

where:

- `Optprice` is a NINST-by-1 vector of European option prices.

- `Strike` is a NINST-by-1 vector of strike prices.

- `Settle` is a scalar date marking the settlement date.

- `Maturity` is a NINST-by-1 vector of maturity dates.

- `OptSpec` is a NINST-by-1 cell array of strings `'call'` or `'put'`.

Option Stock Structure Example Using an Implied Trinomial Tree

Consider the following data quoted from liquid options in the market with varying strikes and maturity. You specify these parameters in MATLAB as:

```
Settle =    '01/01/06';

Maturity =    ['07/01/06';
    '07/01/06';
    '07/01/06';
    '07/01/06';
    '01/01/07';
    '01/01/07';
    '01/01/07';
    '01/01/07';
    '07/01/07';
    '07/01/07';
    '07/01/07';
    '07/01/07';
    '01/01/08';
    '01/01/08';
    '01/01/08';
    '01/01/08'];

Strike = [113;
    101;
    100;
     88;
    128;
    112;
    100;
     78;
    144;
    112;
    100;
     69;
```

```
                            162;
                            112;
                            100;
                             61];

        OptPrice =[                 0;
            4.807905472659144;
            1.306321897011867;
            0.048039195057173;
                            0;
            2.310953054191461;
            1.421950392866235;
            0.020414826276740;
                            0;
            5.091986935627730;
            1.346534812295291;
            0.005101325584140;
                            0;
            8.047628153217246;
            1.219653432150932;
            0.001041436654748];


        OptSpec = { 'call';
            'call';
            'put';
            'put';
            'call';
            'call';
            'put';
            'put';
            'call';
            'call';
            'put';
            'put';
            'call';
            'call';
            'put';
            'put'};
```

```
StockOptSpec = stockoptspec(OptPrice, Strike, Settle, Maturity, OptSpec)

StockOptSpec =

        FinObj: 'StockOptSpec'
      OptPrice: [16x1 double]
        Strike: [16x1 double]
        Settle: 732678
      Maturity: [16x1 double]
       OptSpec: {16x1 cell}
    InterpMethod: 'price'
```

Note The algorithm for building the ITT tree requires specifying option prices for all tree nodes. The maturities of those options correspond to those of the tree levels, and the strike to the prices on the tree nodes. The types of option are `Calls` for the nodes above the central nodes, and `Puts` for those below and including the central nodes.

Clearly, all these options will not be available in the market, hence making interpolation and extrapolation necessary to obtain the node option prices. The degree to which the tree reflects the market will unavoidably be tied to the results of these interpolations and extrapolations. Keeping in mind that extrapolation is less accurate than interpolation, and more so the further away the extrapolated points are from the data points, the function `itttree` issues a warning with a list of the options for which extrapolation was necessary.

In some cases, it may be desirable to view a list of ideal option prices to form an idea of the ranges needed. This can be achieved by calling the function `itttree` specifying only the first three input arguments. The second output argument is a structure array containing the list of ideal options needed.

Creating an Implied Trinomial Tree

You can now use the `StockSpec`, `TimeSpec`, and `StockOptSpec` structures described in "Stock Structure Example Using an Implied Trinomial Tree" on page 3-10, "TimeSpec Example Using an Implied Trinomial Tree" on page 3-11, and "Option Stock Structure Example Using an Implied Trinomial Tree" on page 3-13 to build an implied trinomial tree (ITT). First, you must define

the interest rate term structure. For this example, assume that the interest rate is fixed at 8% annually between the valuation date of the tree (January 1, 2006) until its maturity.

```
Rate = 0.08;
ValuationDate = '01-01-2006';
EndDate = '01-01-2008';

RateSpec = intenvset('StartDates', ValuationDate, 'EndDates', EndDate, ...
    'ValuationDate', ValuationDate, 'Rates', Rate, 'Compounding', -1);
```

To build an ITTTree, enter:

```
ITTTree = itttree(StockSpec, RateSpec, TimeSpec, StockOptSpec)

ITTTree =

        FinObj: 'ITStockTree'
     StockSpec: [1x1 struct]
  StockOptSpec: [1x1 struct]
      TimeSpec: [1x1 struct]
      RateSpec: [1x1 struct]
          tObs: [0 0.500000000000000 1 1.500000000000000 2]
          dObs: [732678 732860 733043 733225 733408]
         STree: {1x5 cell}
         Probs: {[3x1 double]  [3x3 double]  [3x5 double]  [3x7 double]}
```

Examining Equity Trees

Financial Instruments Toolbox software uses equity binary and implied trinomial trees to represent prices of equity options and of underlying stocks. At the highest level, these trees have structures wrapped around them. The structures encapsulate information required to interpret information in the tree.

To examine an equity binary or trinomial tree, load the data in the MAT-file deriv.mat into the MATLAB workspace.

```
load deriv.mat
```

Display the list of variables loaded from the MAT-file with the whos command.

```
Name            Size          Bytes  Class    Attributes

BDTInstSet      1x1           15956  struct
BDTTree         1x1            5138  struct
BKInstSet       1x1           15946  struct
BKTree          1x1            5904  struct
CRRInstSet      1x1           12434  struct
CRRTree         1x1            5058  struct
EQPInstSet      1x1           12434  struct
EQPTree         1x1            5058  struct
HJMInstSet      1x1           15948  struct
HJMTree         1x1            5838  struct
HWInstSet       1x1           15946  struct
HWTree          1x1            5904  struct
ITTInstSet      1x1           12438  struct
ITTTree         1x1            8862  struct
ZeroInstSet     1x1           10282  struct
ZeroRateSpec    1x1            1580  struct
```

Examining a CRRTree

You can examine in some detail the contents of the CRRTree structure contained in this file.

```
CRRTree


CRRTree =

     FinObj: 'BinStockTree'
     Method: 'CRR'
  StockSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 1 2 3 4]
       dObs: [731582 731947 732313 732678 733043]
      STree: {[100]  [110.5171 90.4837]  [122.1403 100 81.8731]  [1x4 double]  [1x5 double]}
    UpProbs: [0.7309 0.7309 0.7309 0.7309]
```

The Method field of the structure indicates that this is a CRR tree, not an EQP tree.

The fields `StockSpec`, `TimeSpec` and `RateSpec` hold the original structures passed into the function `crrtree`. They contain all the context information required to interpret the tree data.

The fields `tObs` and `dObs` are vectors containing the observation times and dates, the times and dates of the levels of the tree. In this particular case, `tObs` reveals that the tree has a maturity of 4 years (`tObs(end) = 4`) and that it has four time steps (the length of `tObs` is five).

The field `dObs` shows the specific dates for the tree levels, with a granularity of 1 day. This means that all values in `tObs` that correspond to a given day from 00:00 hours to 24:00 hours are mapped to the corresponding value in `dObs`. You can use the function `datestr` to convert these MATLAB serial dates into their string representations.

The field `UpProbs` is a vector representing the probabilities for up movements from any node in each level. This vector has 1 element per tree level. All nodes for a given level have the same probability of an up movement. In the specific case being examined, the probability of an up movement is 0.7309 for all levels, and the probability for a down movement is 0.2691 (1 - 0.7309).

Finally, the field `STree` contains the actual stock tree. It is represented in MATLAB as a cell array with each cell array element containing a vector of prices corresponding to a tree level. The prices are in descending order, that is, `CRRTree.STree{3}(1)` represents the topmost element of the third level of the tree, and `CRRTree.STree{3}(end)` represents the bottom element of the same level of the tree.

Examining an ITTTree

You can examine in some detail the contents of the `ITTTree` structure contained in this file.

```
ITTTree
```

```
ITTTree =

        FinObj: 'ITStockTree'
     StockSpec: [1x1 struct]
   StockOptSpec: [1x1 struct]
      TimeSpec: [1x1 struct]
```

```
RateSpec: [1x1 struct]
   tObs: [0 1 2 3 4]
   dObs: [732678 733043 733408 733773 734139]
  STree: {1x5 cell}
  Probs: {[3x1 double]  [3x3 double]  [3x5 double]  [3x7 double]}
```

The fields `StockSpec`, `StockOptSpec`, `TimeSpec`, and `RateSpec` hold the original structures passed into the function `itttree`. They contain all the context information required to interpret the tree data.

The fields `tObs` and `dObs` are vectors containing the observation times and dates, the times and dates of the levels of the tree. In this particular case, `tObs` reveals that the tree has a maturity of 4 years (`tObs(end) = 4`) and that it has four time steps (the length of `tObs` is five).

The field `dObs` shows the specific dates for the tree levels, with a granularity of 1 day. This means that all values in `tObs` that correspond to a given day from 00:00 hours to 24:00 hours are mapped to the corresponding value in `dObs`. You can use the function `datestr` to convert these MATLAB serial dates into their string representations.

The field `Probs` is a vector representing the probabilities for movements from any node in each level. This vector has three elements per tree node. In the specific case being examined, at `tObs= 1`, the probability for an up movement is 0.4675, and the probability for a down movement is 0.1934.

Finally, the field `STree` contains the actual stock tree. It is represented in MATLAB as a cell array with each cell array element containing a vector of prices corresponding to a tree level. The prices are in descending order, that is, `ITTTree.STree{4}(1)` represents the top element of the fourth level of the tree, and `ITTTree.STree{4}(end)` represents the bottom element of the same level of the tree.

Isolating a Specific Node for a CRRTree

The function `treepath` can isolate a specific set of nodes of a binary tree by specifying the path used to reach the final node. As an example, consider the nodes touched by starting from the root node, then following a down movement, then an up movement, and finally a down movement. You use a vector to specify the path, with 1 corresponding to an up movement and 2

corresponding to a down movement. An up-down-up path is then represented as [2 1 2]. To obtain the values of all nodes touched by this path, enter:

```
SVals = treepath(CRRTree.STree, [2 1 2])

SVals =

  100.0000
   90.4837
  100.0000
   90.4837
```

The first value in the vector SVals corresponds to the root node, and the last value corresponds to the final node reached by following the path indicated.

Isolating a Specific Node for an ITTTree

The function trintreepath can isolate a specific set of nodes of a trinomial tree by specifying the path used to reach the final node. As an example, consider the nodes touched by starting from the root node, then following an up movement, then a middle movement, and finally a down movement. You use a vector to specify the path, with 1 corresponding to an up movement, 2 corresponding to a middle movement, and 3 corresponding to a down movement. An up-down-middle-down path is then represented as [1 3 2 3]. To obtain the values of all nodes touched by this path, enter:

```
pathSVals = trintreepath(ITTTree, [1 3 2 3])

pathSVals =

   50.0000
   66.3448
   50.0000
   50.0000
   37.6819
```

The first value in the vector pathSVals corresponds to the root node, and the last value corresponds to the final node reached by following the path indicated.

Differences Between CRR and EQP Tree Structures

In essence, the structures representing CRR trees and EQP trees are similar. If you create a CRR or an EQP tree using identical input arguments, only a few of the tree structure fields differ:

- The Method field has a value of 'CRR' or 'EQP' indicating the method used to build the structure.

- The prices in the STree cell array have the same structure, but the prices within the cell array are different.

- For EQP, the structure field UpProb always holds a vector with all elements set to 0.5, while for CRR, these probabilities are calculated based on the input arguments passed when building the tree.

Supported Equity Derivatives

Asian Option

An *Asian* option is a path-dependent option with a payoff linked to the average value of the underlying asset during the life (or some part of the life) of the option. They are similar to lookback options in that there are two types of Asian options: fixed (average price option) and floating (average strike option). Fixed Asian options have a specified strike, while floating Asian options have a strike equal to the average value of the underlying asset over the life of the option.

There are four Asian option types, each with its own characteristic payoff formula:

- Fixed call (average price option): $\max(0, S_{av} - X)$

- Fixed put (average price option): $\max(0, X - S_{av})$

- Floating call (average strike option): $\max(0, S - S_{av})$

- Floating put (average strike option): $\max(0, S_{av} - S)$

where:

S_{av} is the average price of underlying asset.

S is the price of the underlying asset.

X is the strike price (applicable only to fixed Asian options).

S_{av} is defined using either a geometric or an arithmetic average.

The following functions support Asian options.

Function	Purpose
asianbycrr	Price Asian options from a CRR binomial tree.
asianbyeqp	Price Asian options from an EQP binomial tree.
asianbyitt	Price Asian options using an implied trinomial tree (ITT).
instasian	Construct an Asian option.
asianbyls	Price European or American Asian options using the Longstaff-Schwartz model.
asiansensbyls	Calculate prices and sensitivities of European or American Asian options using the Longstaff-Schwartz model.
asianbykv	Price European geometric Asian options using the Kemna Vorst model.
asiansensbykv	Calculate prices and sensitivities of European geometric Asian options using the Kemna Vorst model.
asianbylevy	Price European arithmetic Asian options using the Levy model.
asiansensbylevy	Calculate prices and sensitivities of European arithmetic Asian options using the Levy model.

Barrier Option

A *barrier* option is similar to a vanilla put or call option, but its life either begins or ends when the price of the underlying stock passes a predetermined barrier value. There are four types of barrier options.

Up Knock-In

This option becomes effective when the price of the underlying stock passes above a barrier that is above the initial stock price. Once the barrier has knocked in, it will not knock out even if the price of the underlying instrument moves below the barrier again.

Up Knock-Out

This option terminates when the price of the underlying stock passes above a barrier that is above the initial stock price. Once the barrier has knocked out, it will not knock in even if the price of the underlying instrument moves below the barrier again.

Down Knock-In

This option becomes effective when the price of the underlying stock passes below a barrier that is below the initial stock price. Once the barrier has knocked in, it will not knock out even if the price of the underlying instrument moves above the barrier again.

Down Knock-Out

This option terminates when the price of the underlying stock passes below a barrier that is below the initial stock price. Once the barrier has knocked out, it will not knock in even if the price of the underlying instrument moves above the barrier again.

Rebates

If a barrier option fails to exercise, the seller may pay a rebate to the buyer of the option. Knock-outs may pay a rebate when they are knocked out, and knock-ins may pay a rebate if they expire without ever knocking in.

The following functions support barrier options.

Function	Purpose
barrierbycrr	Price barrier options from a CRR binomial tree.
barrierbyeqp	Price barrier options from an EQP binomial tree.
barrierbyitt	Price barrier options using an implied trinomial tree (ITT).
instbarrier	Construct a barrier option.

Basket Option

A *basket* option is an option on a portfolio of several underlying equity assets. Payout for a basket option depends on the cumulative performance of the collection of the individual assets. A basket option tends to be cheaper than the corresponding portfolio of plain vanilla options for these reasons:

- If the basket components correlate negatively, movements in the value of one component neutralize opposite movements of another component. Unless all the components correlate perfectly, the basket option is cheaper than a series of individual options on each of the assets in the basket.

- A basket option minimizes transaction costs because an investor has to purchase only one option instead of several individual options.

The payoff for a basket option is as follows:

- For a call: $\max(\sum Wi * Si - K; 0)$

- For a put: $\max(\sum K - Wi * Si; 0)$

where:

Si is the price of asset i in the basket.

Wi is the quantity of asset i in the basket.

K is the strike price.

Financial Instruments Toolbox software supports Longstaff-Schwartz and Nengiu Ju models for pricing basket options. The Longstaff-Schwartz model supports both European, Bermuda, and American basket options. The Nengiu Ju model only supports European basket options. If you want to price either an American or Bermuda basket option, use the functions for the Longstaff-Schwartz model. To price a European basket option, use either the functions for the Longstaff-Schwartz model or the Nengiu Ju model.

Function	Purpose
basketbyls	Price basket options using the Longstaff-Schwartz model.
basketsensbyls	Calculate price and sensitivities for basket options using the Longstaff-Schwartz model.
basketbyju	Price European basket options using the Nengjiu Ju approximation model.
basketsensbyju	Calculate European basket options price and sensitivity using the Nengjiu Ju approximation model.
basketstockspec	Specify a basket stock structure.

Compound Option

A *compound* option is basically an option on an option; it gives the holder the right to buy or sell another option. With a compound option, a vanilla stock option serves as the underlying instrument. Compound options thus have two strike prices and two exercise dates.

There are four types of compound options:

- Call on a call
- Put on a put
- Call on a put
- Put on a call

> **Note** The payoff formulas for compound options are too complex for this discussion. If you are interested in the details, consult the paper by Mark Rubinstein entitled "Double Trouble," published in *Risk 5* (1991).

Consider the third type, a call on a put. It gives the holder the right to buy a put option. In this case, on the first exercise date, the holder of the compound option pay the first strike price and receives a put option. The put option gives the holder the right to sell the underlying asset for the second strike price on the second exercise date.

The following functions support compound options.

Function	Purpose
compoundbycrr	Price compound options from a CRR binomial tree.
compoundbyeqp	Price compound options from an EQP binomial tree.
compoundbyitt	Price compound options using an implied trinomial tree (ITT).
instcompound	Construct a compound option.

Lookback Option

A *lookback* option is a path-dependent option based on the maximum or minimum value the underlying asset achieves during the entire life of the option.

Financial Instruments Toolbox software supports two types of lookback options: fixed and floating. Fixed lookback options have a specified strike price, while floating lookback options have a strike price determined by the asset path. Consequently, there are a total of four lookback option types, each with its own characteristic payoff formula:

- Fixed call: $\max(0, S_{\max} - X)$

- Fixed put: $\max(0, X - S_{\min})$

- Floating call: $\max(0, S - S_{\min})$

- Floating put: $\max(0, S_{\max} - S)$

where:

$S_{\max}$ is the maximum price of underlying stock found along the particular path followed to the node.

$S_{\min}$ is the minimum price of underlying stock found along the particular path followed to the node.

S is the price of the underlying stock on the node.

X is the strike price (applicable only to fixed lookback options).

The following functions support lookback options.

Function	Purpose
lookbackbycrr	Price lookback options from a CRR binomial tree.
lookbackbyeqp	Price lookback options from an EQP binomial tree.
lookbackbyitt	Price lookback options using an implied trinomial tree (ITT).
instlookback	Construct a lookback option.

Digital Option

A *digital* option is an option whose payoff is characterized as having only two potential values: a fixed payout, when the option is in the money or a zero payout otherwise. This is the case irrespective of how far the asset price at maturity is above (call) or below (put) the strike.

Digital options are attractive to sellers because they guarantee a known maximum loss in the event that the option is exercised. This overcomes a fundamental problem with the vanilla options, where the potential loss is unlimited. Digital options are attractive to buyers because the option payoff is

a known constant amount, and this amount can be adjusted to provide the exact quantity of protection required.

Financial Instruments Toolbox supports four types of digital options:

- Cash-or-nothing option — Pays some fixed amount of cash if the option expires in the money.

- Asset-or-nothing option — Pays the value of the underlying security if the option expires in the money.

- Gap option — One strike decides if the option is in or out of money; another strike decides the size the size of the payoff.

- Supershare — Pays out a proportion of the assets underlying a portfolio if the asset lies between a lower and an upper bound at the expiry of the option.

The following functions calculate pricing and sensitivity for digital options.

Function	Purpose
cashbybls	Calculate the price of cash-or-nothing digital options using the Black-Scholes model.
assetbybls	Calculate the price of asset-or-nothing digital options using the Black-Scholes model.
gapbybls	Calculate the price of gap digital options using the Black-Scholes model.
supersharebybls	Calculate the price of supershare digital options using the Black-Scholes model.
cashsensbybls	Calculate the price and sensitivities of cash-or-nothing digital options using the Black-Scholes model.
assetsensbybls	Calculate the price and sensitivities of asset-or-nothing digital options using the Black-Scholes model.

Function	Purpose
gapsensbybls	Calculate the price and sensitivities of gap digital options using the Black-Scholes model.
supersharesensbybls	Calculate the price and sensitivities of supershare digital options using the Black-Scholes model.

Rainbow Option

A rainbow option payoff depends on the relative price performance of two or more assets. A *rainbow* option gives the holder the right to buy or sell the best or worst of two securities, or options that pay the best or worst of two assets.

Rainbow options are popular because of the lower premium cost of the structure relative to the purchase of two separate options. The lower cost reflects the fact that the payoff is generally lower than the payoff of the two separate options.

Financial Instruments Toolbox supports two types of rainbow options:

- Minimum of two assets — The option holder has the right to buy(sell) one of two risky assets, whichever one is worth less.

- Maximum of two assets — The option holder has the right to buy(sell) one of two risky assets, whichever one is worth more.

The following rainbow options speculate/hedge on two equity assets.

Function	Purpose
minassetbystulz	Calculate the European rainbow option price on minimum of two risky assets using the Stulz option pricing model.
minassetsensbystulz	Calculate the European rainbow option prices and sensitivities on minimum of two risky assets using the Stulz pricing model.

Function	Purpose
maxassetbystulz	Calculate the European rainbow option price on maximum of two risky assets using the Stulz option pricing model.
maxassetsensbystulz	Calculate the European rainbow option prices and sensitivities on maximum of two risky assets using the Stulz pricing model.

Vanilla Option

A *vanilla option* is a category of options that includes only the most standard components. A vanilla option has an expiration date and straightforward strike price. American-style options and European-style options are both categorized as vanilla options.

The payoff for a vanilla option is as follows:

- For a call: $\max(St - K, 0)$

- For a put: $\max(K - St, 0)$

where:

St is the price of the underlying asset at time t.

K is the strike price.

The following functions support specifying or pricing a vanilla option.

Function	Purpose
optstockbycrr	Calculate the price of a European, Bermuda, or American stock option using a CRR tree.
optstockbyeqp	Calculate the price of a European, Bermuda, or American stock option using an EQP tree.
optstockbyitt	Calculate the price of a European, Bermuda, or American stock option using an ITT tree.

Function	Purpose
optstockbylr	Calculate the price of a European, Bermuda, or American stock option using the Leisen-Reimer (LR) binomial tree model.
optstockbybls	Price options using the Black-Scholes option pricing model.
optstocksensbybls	Calculate option prices and sensitivities using the Black-Scholes option pricing model.
optstockbyrgw	Calculate American call option prices using the Roll-Geske-Whaley option pricing model.
optstocksensbyrgw	Calculate American call option prices and sensitivities using the Roll-Geske-Whaley option pricing model.
optstockbybjs	Price American options using the Bjerksund-Stensland 2002 option pricing model.
optstocksensbybjs	Calculate American option prices and sensitivities using the Bjerksund-Stensland 2002 option pricing model.
optstockbyls	Price vanilla options using the Longstaff-Schwartz model.
optstocksensbyls	Calculate vanilla option prices and sensitivities using the Longstaff-Schwartz model.
instoptstock	Specify a European or Bermuda option.

Bermuda Put and Call Schedule

A Bermuda option resembles a hybrid of American and European options. You exercise it on predetermined dates only, usually monthly. In Financial Instruments Toolbox software, you indicate the relevant information for a Bermuda option in two input matrices:

- Strike — Contains the strike price values for the option.

- ExerciseDates — Contains the schedule when you can exercise the option.

Spread Option

A *spread option* is an option written on the difference of two underlying assets. For example, a European call on the difference of two assets *X1* and *X2* would have the following pay off at maturity:

$$\max(X1 - X2 - K, 0)$$

where:

K is the strike price.

The following functions support spread options.

Function	Purpose
spreadbykirk	Price European spread options using the Kirk pricing model.
spreadsensbykirk	Calculate European spread option prices and sensitivities using the Kirk pricing model.
spreadbybjs	Price European spread options using the Bjerksund-Stensland pricing model.
spreadsensbybjs	Calculate European spread option prices and sensitivities using the Bjerksund-Stensland pricing model.
spreadbyfd	Price European or American spread options using the Alternate Direction Implicit (ADI) finite difference method.
spreadsensbyfd	Calculate price and sensitivities of European or American spread spread options using the Alternate Direction Implicit (ADI) finite difference method.
spreadbyls	Price European or American spread options using Monte Carlo simulations.
spreadsensbyls	Calculate price and sensitivities for European or American spread options using Monte Carlo simulations.

Supported Energy Derivatives

In this section...

Asian Option

An *Asian* option is a path-dependent option with a payoff linked to the average value of the underlying asset during the life (or some part of the life) of the option. They are similar to lookback options in that there are two types of Asian options: fixed (average price option) and floating (average strike option). Fixed Asian options have a specified strike, while floating Asian options have a strike equal to the average value of the underlying asset over the life of the option.

There are four Asian option types, each with its own characteristic payoff formula:

- Fixed call (average price option): $\max(0, S_{av} - X)$

- Fixed put (average price option): $\max(0, X - S_{av})$

- Floating call (average strike option): $\max(0, S - S_{av})$

- Floating put (average strike option): $\max(0, S_{av} - S)$

where:

S_{av} is the average price of underlying asset.

S is the price of the underlying asset.

X is the strike price (applicable only to fixed Asian options).

S_{av} is defined using either a geometric or an arithmetic average.

The following functions support Asian options.

Function	Purpose
asianbyls	Price European or American Asian options using the Longstaff-Schwartz model.
asiansensbyls	Calculate prices and sensitivities of European or American Asian options using the Longstaff-Schwartz model.
asianbykv	Price European geometric Asian options using the Kemna Vorst model.
asiansensbykv	Calculate prices and sensitivities of European geometric Asian options using the Kemna Vorst model.
asianbylevy	Price European arithmetic Asian options using the Levy model.
asiansensbylevy	Calculate prices and sensitivities of European arithmetic Asian options using the Levy model.

Vanilla Option

A *vanilla option* is a category of options that includes only the most standard components. A vanilla option has an expiration date and straightforward strike price. American-style options and European-style options are both categorized as vanilla options.

The payoff for a vanilla option is as follows:

- For a call: $\max(St - K, 0)$

- For a put: $\max(K - St, 0)$

where:

St is the price of the underlying asset at time t.

K is the strike price.

The following functions support specifying or pricing a vanilla option.

Function	Purpose
optstockbyls	Price European, Bermudan, or American vanilla options using the Longstaff-Schwartz model.
optstocksensbyls	Calculate European, Bermudan, or American vanilla option prices and sensitivities using the Longstaff-Schwartz model.

Spread Option

A *spread option* is an option written on the difference of two underlying assets. For example, a European call on the difference of two assets *X1* and *X2* would have the following pay off at maturity:

$$\max(X1 - X2 - K, 0)$$

where:

K is the strike price.

The following functions support spread options.

Function	Purpose
spreadbykirk	Price European spread options using the Kirk pricing model.
spreadsensbykirk	Calculate European spread option prices and sensitivities using the Kirk pricing model.
spreadbybjs	Price European spread options using the Bjerksund-Stensland pricing model.
spreadsensbybjs	Calculate European spread option prices and sensitivities using the Bjerksund-Stensland pricing model.

Function	Purpose
`spreadbyfd`	Price European or American spread options using the Alternate Direction Implicit (ADI) finite difference method.
`spreadsensbyfd`	Calculate price and sensitivities of European or American spread options using the Alternate Direction Implicit (ADI) finite difference method.
`spreadbyls`	Price European or American spread options using Monte Carlo simulations.
`spreadsensbyls`	Calculate price and sensitivities for European or American spread options using Monte Carlo simulations.

For more information on using spread options, see "Pricing European and American Spread Options" on page 3-38.

Pricing European and American Spread Options

This example shows how to price and calculate sensitivities for European and American spread options using various techniques. First, the price and sensitivities for an European spread option is calculated using closed form solutions. Then, price and sensitivities for an American spread option is calculated using finite difference and Monte Carlo simulations. Finally, further analysis is conducted on spread options with different range of inputs.

Spread options are options on the difference of two underlying asset prices. For example, a call option on the spread between two assets will have the following payoff at maturity:

$$max(X_1 - X_2 - K, 0)$$

where X_1 is the price of the first underlying asset, X_2 is the price of the second underlying asset, and K is the strike price. At maturity, if the spread $X_1 - X_2$ is greater than the strike price K, the option holder will exercise the option and gain the difference between the spread and the strike price. If the spread is less than 0, the option holder will not exercise the option, and the payoff is 0. Spread options are frequently traded in the energy market. Two examples are:

- *Crack spreads*: Options on the spread between refined petroleum products and crude oil. The spread represents the refinement margin made by the oil refinery by "cracking" the crude oil into a refined petroleum product.

- *Spark spreads*: Options on the spread between electricity and some type of fuel. The spread represents the margin of the power plant, which takes fuel to run its generator to produce electricity.

Overview of the Pricing Methods

There are several methods to price spread options, as discussed in [1]. This example uses the closed form, finite difference, and Monte Carlo simulations to price spread options. The advantages and disadvantages of each method is discussed below:

- Closed form solutions/approximations of partial differential equations (PDE) are advantageous because they are very fast, and extend well to

computing sensitivities (Greeks). However, closed form solutions are not always available, for example for American spread options.

- The finite difference method is a numerical procedure to solve PDEs by discretizing the price and time variables into a grid. A detailed analysis of this method can be found in [2]. It can handle cases where closed form solutions are not available. Also, finite difference extends well to calculating sensitivities because it outputs a grid of option prices for a range of underlying prices and times. However, it is slower than the closed form solutions.

- Monte Carlo simulation uses random sampling to simulate movements of the underlying asset prices. It handles cases where closed solutions do not exist. However, it usually takes a long time to run, especially if sensitivities need to be calculated.

Pricing an European Spread Option

The following example demonstrates the pricing of a crack spread option.

A refiner is concerned about its upcoming maintenance schedule and needs to protect against decreasing crude oil prices and increasing heating oil prices. During the maintenance the refiner needs to continue providing customers with heating oil to meet their demands. The refiner's strategy is to use spread options to manage its hedge.

On January 2013, the refiner buys a 1:1 crack spread option by purchasing heating oil futures and selling crude oil futures. CLF14 WTI crude oil futures is at \$100 per barrel and HOF14 heating oil futures contract is at \$2.6190 per gallon.

```
clear;

% Price, volatility, and dividend of heating oil
Price1gallon = 2.6190;      % $/gallon
Price1 = Price1gallon*42;    % $/barrel
Vol1 = 0.10;
Div1 = 0.03;

% Price, volatility, and dividend of WTI crude oil
```

```
Price2 = 100;      % $/barrel
Vol2 = 0.15;
Div2 = 0.02;

% Correlation of underlying prices
Corr = 0.3;

% Option type
OptSpec = 'call';

% Strike
Strike = 5;

% Settlement date
Settle = '01-Jan-2013';

% Maturity
Maturity = '01-Jan-2014';

% Risk free rate
RiskFreeRate = 0.05;
```

The pricing functions take an interest rate term structure and stock structure
as inputs. Also, we need to specify which outputs we are interested in.

```
% Define RateSpec
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
    'EndDates', Maturity, 'Rates', RiskFreeRate, 'Compounding', ...
    Compounding, 'Basis', Basis);

% Define StockSpec for the two assets
StockSpec1 = stockspec(Vol1, Price1, 'Continuous', Div1);
StockSpec2 = stockspec(Vol2, Price2, 'Continuous', Div2);

% Specify price and sensitivity outputs
OutSpec = {'Price', 'Delta', 'Gamma'};
```

The Financial Instruments Toolbox™ contains two types of closed form approximations for calculating price and sensitivities of European spread options: the Kirk's approximation (spreadbykirk, spreadsensbykirk) and the Bjerksund and Stensland model (spreadbybjs, spreadsensbybjs) [3].

The function spreadsensbykirk calculates prices and sensitivities for an European spread option using the Kirk's approximation.

```
% Kirk's approximation
[PriceKirk, DeltaKirk, GammaKirk] = ...
    spreadsensbykirk(RateSpec, StockSpec1, StockSpec2, Settle, ...
    Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec)


PriceKirk =

    8.3636


DeltaKirk =

    0.6108   -0.5590


GammaKirk =

    0.0225    0.0249
```

The function spreadsensbybjs calculates the prices and sensitivities for an European spread option using the Bjerksund and Stensland model. In [3], Bjerksund and Stensland explains that the Kirk's approximation tends to underprice the spread option when the strike is close to zero, and overprice when the strike is further away from zero. In comparison, the model by Bjerksund and Stensland has higher precision.

```
% Bjerksund and Stensland model
[PriceBJS, DeltaBJS, GammaBJS] = ...
    spreadsensbybjs(RateSpec, StockSpec1, StockSpec2, Settle, ...
    Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec)
```

3-41

```
PriceBJS =

    8.3662


DeltaBJS =

    0.6115   -0.5597


GammaBJS =

    0.0225    0.0248
```

A comparison of the calculated prices show that the two closed form models produce similar results for price and sensitivities. In addition to delta and gamma, the functions can also calculate theta, vega, lambda, and rho.

```
% Comparison of calculated prices
fprintf('Comparison of prices:\n\n');
fprintf('Kirk:              %f\n', PriceKirk);
fprintf('BJS:               %f\n\n', PriceBJS);

% Comparison of Delta
fprintf('Comparison of delta:\n\n');
fprintf('Kirk:              %f    %f\n', DeltaKirk(1), DeltaKirk(2));
fprintf('BJS:               %f    %f\n\n', DeltaBJS(1), DeltaBJS(2));

% Comparison of Gamma
fprintf('Comparison of gamma:\n\n');
fprintf('Kirk:              %f    %f\n', GammaKirk(1), GammaKirk(2));
fprintf('BJS:               %f    %f\n\n', GammaBJS(1), GammaBJS(2));

Comparison of prices:

Kirk:              8.363641
BJS:               8.366158
```

```
Comparison of delta:

Kirk:           0.610790        -0.558959
BJS:            0.611469        -0.559670


Comparison of gamma:

Kirk:           0.022533        0.024850
BJS:            0.022495        0.024819
```

Pricing an American Spread Option

Although the closed form approximations are fast and well suited for pricing
European spread options, they cannot price American spread options. Using
the finite difference method and the Monte Carlo method, an American spread
option can be priced. In this example, an American spread option is priced
with the same attributes as the above crack spread option.

The finite difference method numerically solves a PDE by discretizing the
underlying price and time variables into a grid. The Financial Instrument
Toolbox™ contains the functions spreadbyfd and spreadsensbyfd, which
calculate prices and sensitivities for European and American spread options
using the finite difference method. For the finite difference method, the
composition of the grid has a large impact on the quality of the output and the
execution time. Generally, a finely discretized grid will result in outputs that
are closer to the theoretical value, but it comes at the cost of longer execution
times. The composition of the grid is controlled using optional paramters
PriceGridSize, TimeGridSize, and AssetPriceMax.

```
% To indicate that we are pricing an American option, add an optional input
% of |AmericanOpt| with a value of 1 to the argument of the function.

% Finite difference method for American spread option
[PriceFD, DeltaFD, GammaFD, PriceGrid, AssetPrice1, ...
    AssetPrice2] = ...
    spreadsensbyfd(RateSpec, StockSpec1, StockSpec2, Settle, ...
    Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec, ...
    'PriceGridSize', [500 500], 'TimeGridSize', 100, ...
```

```
        'AssetPriceMax', [2000 2000], 'AmericanOpt', 1);

% Display price and sensitivities
PriceFD
DeltaFD
GammaFD


PriceFD =

    8.5504


DeltaFD =

    0.6297    -0.5776


GammaFD =

    0.0267     0.0365
```

The function spreadsensbyfd also returns a grid that contains the option prices for a range of underlying prices and times. The grid of option prices at time zero, which is the option prices at the settle date, can be plotted for a range of underlying prices.

```
% Plot option prices
figure;
mesh(AssetPrice1, AssetPrice2, PriceGrid(:, :, 1)');
title('American Spread Option Prices for Range of Underlying Prices');
xlabel('Price of underlying asset 1');
ylabel('Price of underlying asset 2');
zlabel('Price of spread option');
```

American Spread Option Prices for Range of Underlying Prices
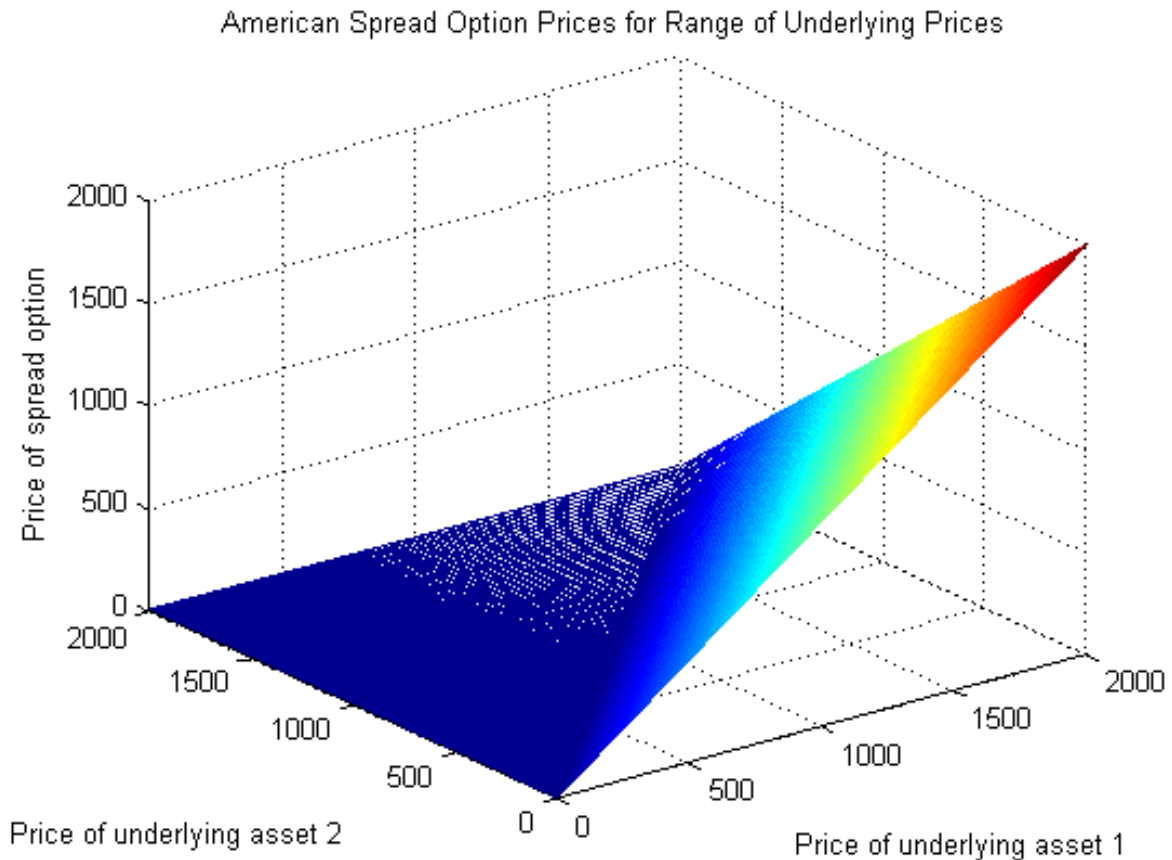
An American style option can be priced by Monte Carlo methods using the least square method of Longstaff and Schwartz [4]. The Financial Instruments Toolbox™ contains the functions spreadbyls and spreadsensbyls, which calculate prices and sensetivities of European and American options using simulations. The Monte Carlo simulation method in spreadsensbyls generates multiple paths of simulations according to a geometric Brownian motion (GBM) for the two underlying asset prices. Similar to the finite difference method where the granularity of the grid determined the quality of the output and the execution time, the quality of output and execution time of the Monte Carlo simluation depends on the number of paths (NumTrials)

and the number of time periods per path (`NumPeriods`). Also, note that the results obtained by Monte Carlo simulations are not deterministic. Each run will have different results depending on the simulation outcomes.

```
% To indicate that we are pricing an American option using the Longstaff
% and Schwartz method, add an optional input of |AmericanOpt| with a value
% of 1 to the argument of the function.

% Monte Carlo method for American spread option
[PriceMC, DeltaMC, GammaMC] = ...
    spreadsensbyls(RateSpec, StockSpec1, StockSpec2, Settle, ...
    Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec, ...
    'NumTrials', 1000, 'Antithetic', true, 'AmericanOpt', 1)


PriceMC =

    8.4999


DeltaMC =

    0.6325   -0.5931


GammaMC =

   -0.0873    0.0391
```

The results of the two models are compared. The prices and sensitivities calcluated by the Longstaff and Schwartz method will vary at each run, depending on the outcome of the simulations. It is important to note again that the quality of the results from the finite difference method and the Monte Carlo simulation depend on the optional input parameters. For example, increasing the number of paths (`NumTrials`) for the SPREADSENSBYLS function will result in more precise results at the cost of longer execution times.

```
% Comparison of calculated prices
fprintf('Comparison of prices:\n\n');
```

```
fprintf('Finite Difference: %f\n', PriceFD);
fprintf('Monte Carlo:       %f\n\n', PriceMC);

% Comparison of Delta
fprintf('Comparison of delta:\n\n');
fprintf('Finite Difference: %f     %f\n', DeltaFD(1), DeltaFD(2));
fprintf('Monte Carlo       : %f     %f\n\n', DeltaMC(1), DeltaMC(2));

% Comparison of Gamma
fprintf('Comparison of gamma:\n\n');
fprintf('Finite Difference: %f     %f\n', GammaFD(1), GammaFD(2));
fprintf('Monte Carlo       : %f     %f\n\n', GammaMC(1), GammaMC(2));

Comparison of prices:

Finite Difference: 8.550375
Monte Carlo:       8.499894

Comparison of delta:

Finite Difference: 0.629716     -0.577633
Monte Carlo       : 0.632549     -0.593106

Comparison of gamma:

Finite Difference: 0.026737     0.036472
Monte Carlo       : -0.087340     0.039120
```

Comparing Results for a Range of Strike Prices

As discussed earlier, the Kirk's approximation tends to overprice spread
options when the strike is further away from zero. To confirm this, a spread
option will be priced with the same attributes as before, but for a range
of strike prices.

```
% Specify outputs
OutSpec = {'Price', 'Delta'};

% Range of strike prices
```

```
Strike = [-25; -15; -5; 0; 5; 15; 25];
```

The results from the Kirk's approximation and the Bjerkland and Stensland model will be compared against the numerical approximation from the finite difference method. Since spreadsensbyfd can only price one option at a time, it is called in a loop for each strike value. The Monte Carlo simulation (spreadsensbyls) with a large number of trial paths could also be used as a benchmark, but the finite difference will be used for this example.

```
% Kirk's approximation
[PriceKirk, DeltaKirk] = ...
    spreadsensbykirk(RateSpec, StockSpec1, StockSpec2, Settle, ...
    Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec);

% Bjerkland and Stensland model
[PriceBJS, DeltaBJS] = ...
    spreadsensbybjs(RateSpec, StockSpec1, StockSpec2, Settle, ...
    Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec);

% Finite difference
PriceFD = zeros(numel(Strike), 1);
DeltaFD = zeros(numel(Strike), 2);
for i=1:numel(Strike)
    [PriceFD(i), DeltaFD(i,:)] = ...
    spreadsensbyfd(RateSpec, StockSpec1, StockSpec2, Settle, ...
    Maturity, OptSpec, Strike(i), Corr, 'OutSpec', OutSpec, ...
    'PriceGridSize', [500 500], 'TimeGridSize', 100, ...
    'AssetPriceMax', [2000 2000]);
end

% Comparison of calculated prices
fprintf('Prices for range of strikes:\n\n');
fprintf('Kirk    \tBJS      \tFD    \n');
for i=1:numel(Strike)
    fprintf('%f\t%f\t%f\n', PriceKirk(i), PriceBJS(i), PriceFD(i));
end

Prices for range of strikes:

Kirk     BJS        FD
```

```
32.707787 32.672353 32.676804
23.605307 23.577099 23.581775
15.236908 15.228510 15.233143
11.560332 11.560332 11.561086
8.363641 8.366158 8.370751
3.689909 3.678862 3.684599
1.243753 1.219079 1.225090
```

The difference in prices between the closed form and finite difference method is plotted below. It is clear that as the strike moves further away from 0, the difference between the Kirk's approximation and finite difference (red line) increases, while the difference between the Bjerkland and Stensland model and finite difference (blue line) stays at the same level. As stated in [3], the Kirk's approximation is overpricing the spread option when the strike is far away from 0.

```
% Plot of difference in price against the benchmark
figure;
plot(PriceKirk-PriceFD, 'Color', 'red');
hold on;
plot(PriceBJS-PriceFD, 'Color', 'blue');
hold off;
title('Difference in Price Against Finite Difference');
legend('Kirk', 'BJS', 'Location', 'EastOutside');
xlabel('Strike');
set(gca, 'XTickLabel', Strike);
ylabel('Difference in Price');
```

Difference in Price Against Finite Difference
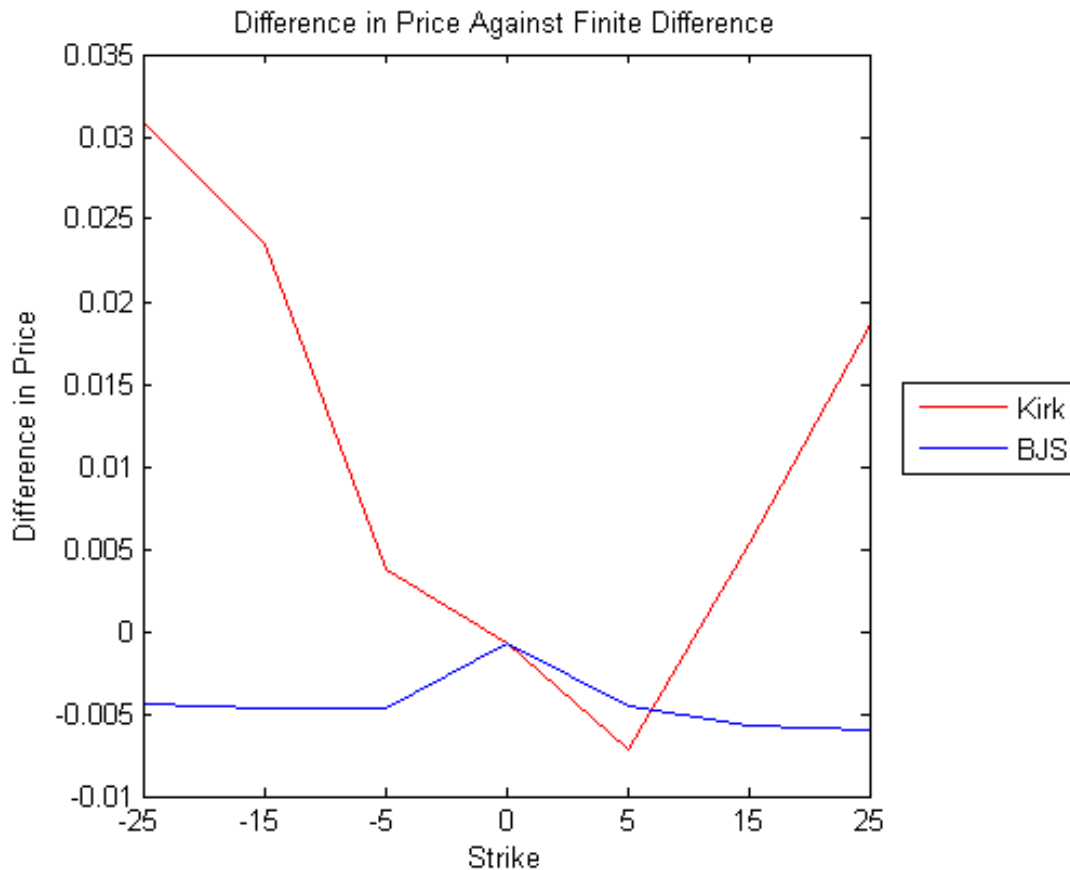
Next, the difference in delta between the closed form models and finite difference is plotted. The top plot shows the difference in delta for the first asset, and the bottom plot shows the difference in delta for the second asset. As seen from the small increments in the y-axis of order 10e-3, it can be seen that all three models (Kirk, BJS, finite difference) produce similar values for delta.

```
% Plot of difference in delta of first asset against the benchmark
figure;
subplot(2, 1, 1);
```

```
plot(DeltaKirk(:,1)-DeltaFD(:,1), 'Color', 'red');
hold on;
plot(DeltaBJS(:,1)-DeltaFD(:,1), 'Color', 'blue');
hold off;
title('Difference in Delta (Asset 1) Against FD');
legend('Kirk', 'BJS', 'Location', 'EastOutside');
xlabel('Strike');
set(gca, 'XTickLabel', Strike);
ylabel('Difference in Delta');

% Plot of difference in delta of second asset against the benchmark
subplot(2, 1, 2);
plot(DeltaKirk(:,2)-DeltaFD(:,2), 'Color', 'red');
hold on;
plot(DeltaBJS(:,2)-DeltaFD(:,2), 'Color', 'blue');
hold off;
title('Difference in Delta (Asset 2) Against FD');
legend('Kirk', 'BJS', 'Location', 'EastOutside');
xlabel('Strike');
set(gca, 'XTickLabel', Strike);
ylabel('Difference in Delta');
```
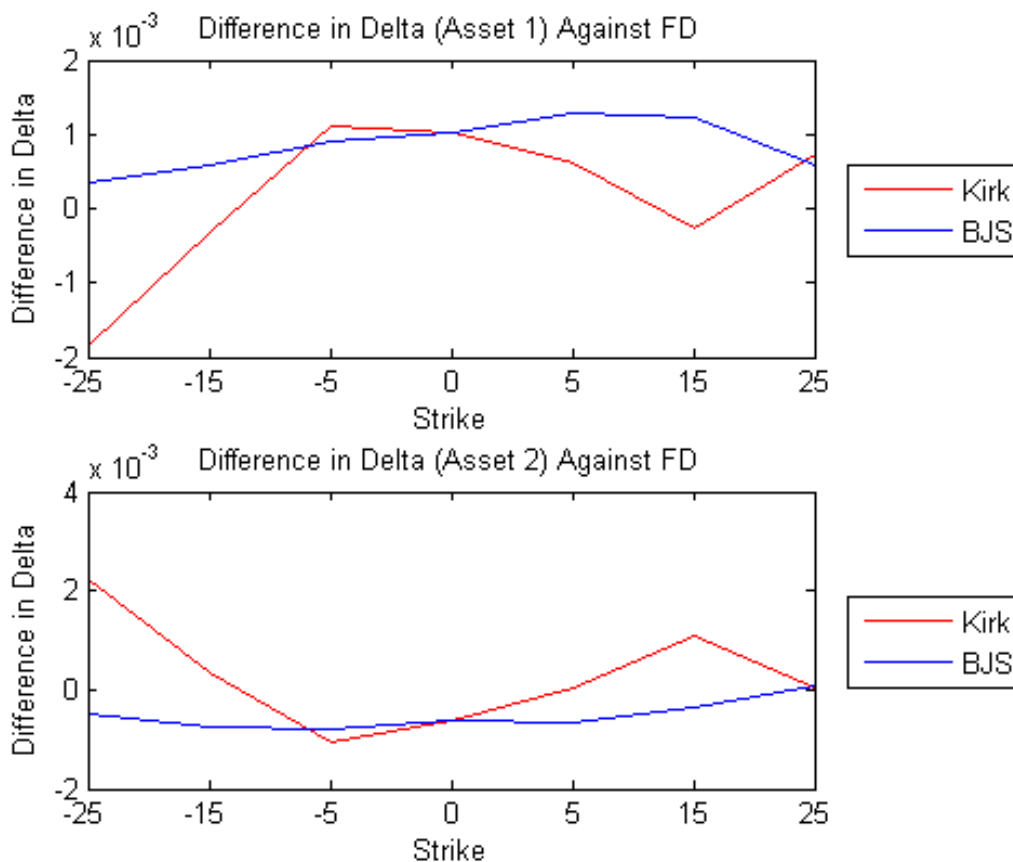
Analyzing Prices and Vega at Different Levels of Volatility

To further show the type of analysis that can be conducted using these models, the above spread option will now be priced at different levels of volatility for the first asset. The price and vega will be compared at three levels of volatility for the first asset: 0.1, 0.3, and 0.5. The Bjerkland and Stensland model will be used for this analysis.

```
% Strike
Strike = 5;
```

```
% Specify output
OutSpec = {'Price', 'Vega'};

% Different levels of volatility for asset 1
Vol1 = [0.1, 0.3, 0.5];

StockSpec1 = stockspec(Vol1, Price1, 'Continuous', Div1);

% Bjerkland and Stensland model
[PriceBJS, VegaBJS] = ...
    spreadsensbybjs(RateSpec, StockSpec1, StockSpec2, Settle, ...
    Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec);

% Display price
fprintf('\nPrices for different vol levels in asset 1:\n\n');
for i=1:numel(Vol1)
    fprintf('%f\n', PriceBJS(i));
end

% Display vega for first asset
fprintf('\nAsset 1 vega for different vol levels in asset 1:\n\n');
for i=1:numel(Vol1)
    fprintf('%f\n', VegaBJS(i,1));
end
fprintf('\n');

% Display vega for second asset
fprintf('\nAsset 2 vega for different vol levels in asset 1:\n\n');
for i=1:numel(Vol1)
    fprintf('%f\n', VegaBJS(i,2));
end
fprintf('\n');


Prices for different vol levels in asset 1:

8.366158
14.209112
21.795746
```

```
Asset 1 vega for different vol levels in asset 1:

15.534849
36.212192
38.794348


Asset 2 vega for different vol levels in asset 1:

29.437036
7.133657
-0.557852
```

The change in the price and vega with respect to the volatility of the first asset is plotted below. It can be observed that as the volatility of the first asset increases, the price of the spread option also increases. Also, the changes in vega indicate that the price of the spread option becomes more sensitive to the volatility of the first asset and less sensitive to the volatility of the second asset.

```
figure;

% Plot price for BJS model
subplot(2, 1, 1);
plot(PriceBJS, 'Color', 'red');
title('Price (BJS)');
legend('Price', 'Location', 'EastOutside');
xlabel('Vol of Asset 1');
set(gca, 'XTick', 1:3);
set(gca, 'XTickLabel', Vol1);
ylabel('Price');

% Plot of vega for BJS model
subplot(2, 1, 2);
plot(VegaBJS(:,1), 'Color', 'red');
hold on;
plot(VegaBJS(:,2), 'Color', 'blue');
hold off;
title('Vega (BJS)');
```

```
legend('Asset 1', 'Asset 2', 'Location', 'EastOutside');
xlabel('Vol of Asset 1');
set(gca, 'XTick', 1:3);
set(gca, 'XTickLabel', Vol1);
set(gca, 'YLim', [-1 40]);
ylabel('Vega');
```
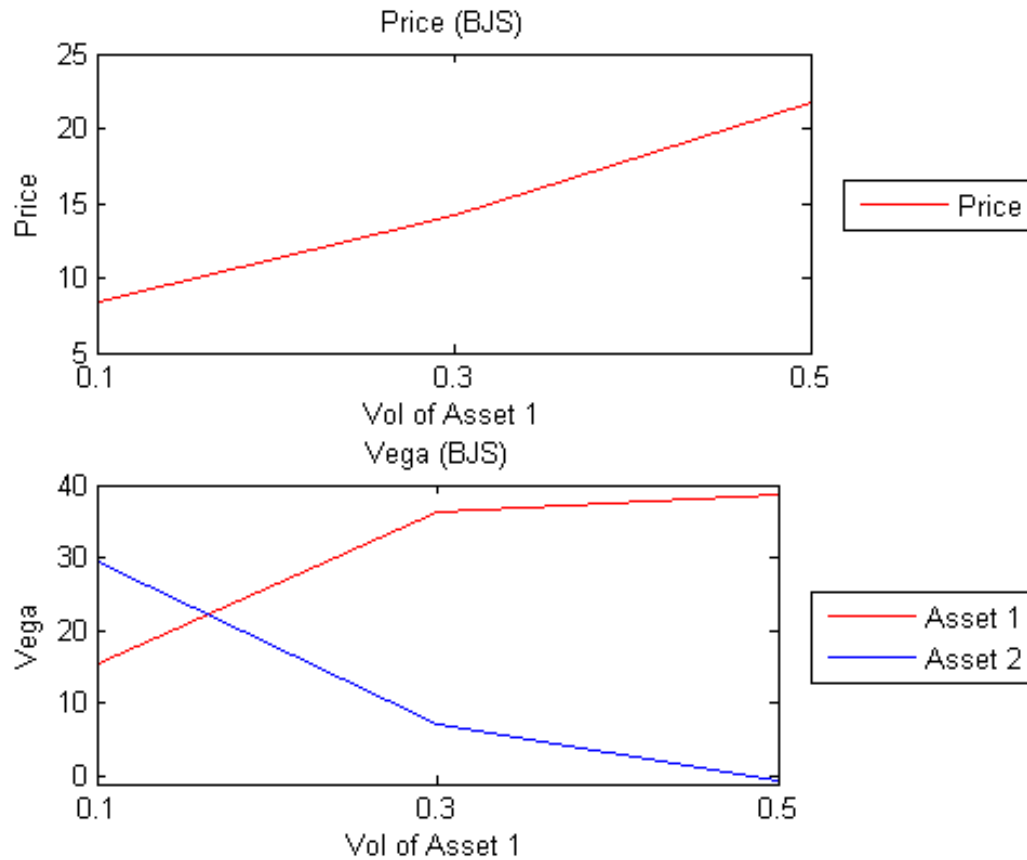

Summary

In this example, European and American spread options were priced and analyzed using various techniques. The Financial Instruments Toolbox™ provides functions for two types of closed form solutions (Kirk, BJS), the finite difference method, and the Monte Carlo simulation method. The closed form solutions are well suited for pricing and sensitivity calculation of European spread options because they are fast. However, they cannot price American spread options. The finite difference method and Monte Carlo method can price both European and American options. However, they are not as fast in pricing European spread options as compared to closed form solutions.

References

[1] Carmona, Rene, Durrleman, Valdo, Pricing and Hedging Spread Options, SIAM Review, Vol. 45, No. 4, pp. 627-685, Society for Industrial and Applied Mathematics, 2003.

[2] Wilmott, Paul, Dewynne, Jeff, Howison, Sam, Option Pricing, Oxford Financial Press, 1993.

[3] Bjerksund, Petter, Stensland, Gunnar, Closed form spread option valuation, Department of Finance, NHH, 2006.

[4] Longstaff, Francis A, Schwartz, Eduardo S, Valuing American Options by Simulation: A Simple Least-Squares Approach, Anderson Graduate School of Management, UC Los Angeles, 2001.

Hedging Strategies Using Spread Options

This example shows different hedging strategies to minimize exposure in the Energy market using Crack Spread Options.

Understanding Crack Spread Options

In the petroleum industry, refiners are concerned about the difference between their input costs (crude oil) and output prices (refined products - gasoline, heating oil, diesel fuel, etc). The differential between these two underlying commodities is reffered to as a Crack Spread. It represents the profit margin between crude oil and the refined products.

A Spread option is an option on the spread where the holder has the right, but not the obligation, to enter into a spot or forward spread contract. Crack Spread Options are often used to protect against declines in the crack spread or to monetise volatility or price expectations on the spread.

Example 1: Protecting Margins using a 1:1 Crack Spread Option

A marketer is interested in protecting his gasoline margin since current prices are strong. A crack spread option strategy will be used to mantain profits for the following season. In March the June WTI crude oil futures is at \$91.10 per barrel and RBOB gasoline futures contract is at \$2.72 per gallon. The marketer's strategy is a long crack call involving purchasing RBOB gasoline futures and selling crude oil futures.

```
OldFormat = get(0, 'format');
format bank

% Price and volatility of RBOB gasoline
Price1gallon = 2.72;          % $/gallon
Price1 = Price1gallon * 42;   % $/barrel
Vol1 = 0.39;

% Price and volatility of WTI crude oil
Price2 = 91.10;        % $/barrel
Vol2 = 0.34;

% Assume the following data
```

```
% Spread Option
Strike = 20;
OptSpec = 'call';
Settle =   '01-March-2013';
Maturity = '01-June-2013';
Corr = 0.45;     % Correlation of underlying commodities
```

Define RateSpec and StockSpec

```
% Define RateSpec
Rate = 0.035;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
    'EndDates', Maturity, 'Rates', Rate, 'Compounding', ...
    Compounding, 'Basis', Basis);

% Define StockSpec for the two assets
StockSpec1 = stockspec(Vol1, Price1);
StockSpec2 = stockspec(Vol2, Price2);
```

Price the Crack Spread Option

```
% Use the function 'spreadbybjs' in the Financial Instruments Toolbox(TM) t
% price the spread option using the Bjerksund and Stensland model.

Price = spreadbybjs(RateSpec, StockSpec1, StockSpec2, Settle, ...
                    Maturity, OptSpec, Strike, Corr)


Price =

        9.91
```

The 1:1 implied current crack spread between these two underlyings is $23.14 per barrel.

```
CrackSpread = Price1 - Price2    % $/barrel
```

```
CrackSpread =

        23.14
```

Suppose that by expiration day, June crude oil prices decrease to \$90.34 per barrel and gasoline prices rise to \$2.89 per gallon.

```
% The price changes cause the marketer's profit margin (the new implied cra
% spread) to increase from $23.14/barrel to $31.04/barrel:
NewCrackSpread = (2.89 * 42) - 90.34


NewCrackSpread =

        31.04
```

Since the marketer purchased a long crack call on the \$20 call, the option is now in the money by \$11.04

```
(NewCrackSpread - Strike)


ans =

        11.04
```

The marketer paid \$9.91 from the long crack call, this protects the margin by \$1.13.

```
(NewCrackSpread - Strike - Price)


ans =

        1.13
```

This strategy provides the marketer protection during spread increase scenarios.

Example 2: Creating a Floor with Crack Spread Options

A refiner is interested in covering its fixed and operating costs, but still profit from a favorable move in the market. In March the May WTI crude oil futures is at \$99.43 per barrel and RBOB gasoline futures contract is at \$3.04 per gallon. The refiner believes that the spread between those commodities of \$28.25 per barrel is favorable. Of this, \$11 corresponds to operating and fixed costs, and \$17.25 is the net refining margin. The refiner's strategy is to sell the crack spread by selling 10 RBOB gasoline futures and buying 10 crude oil futures.

```
% Price and volatility of RBOB gasoline
Price1gallon = 3.04;        % $/gallon
Price1 = Price1gallon * 42;   % $/barrel
Vol1 = 0.35;
Div1 = 0.0783;

% Price and volatility of WTI crude oil
Price2 = 99.43;         % $/barrel
Vol2 = 0.38;
Div2 = 0.0571;


% The refiner purchases 10 May RBOB gasoline crack spread puts with a strik
% price of $25.

% Assume the following data
% Spread Option
Strike = 25;
OptSpec = 'put';
Settle =   '01-March-2013';
Maturity = '01-May-2013';
Corr = 0.30;      % Correlation of underlying commodities
```

Define RateSpec and StockSpec

```
% Define RateSpec
Rate = 0.035;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
    'EndDates', Maturity, 'Rates', Rate, 'Compounding', ...
    Compounding, 'Basis', Basis);

% Define StockSpec for the two assets
StockSpec1 = stockspec(Vol1, Price1, 'Continuous', Div1);
StockSpec2 = stockspec(Vol2, Price2, 'Continuous', Div2);
```

Price the Crack Spread Option

```
% Use the function 'spreadbyfd' in the Financial Instruments Toolbox(TM)
% to price the American spread option using the finite difference method.

Price = spreadbyfd(RateSpec, StockSpec1, StockSpec2, Settle, ...
                   Maturity, OptSpec, Strike, Corr, 'AmericanOpt', 1)


Price =

        6.64
```

By expiration, if option is exercised, the refiner would have hedged the cost
of purchasing 10000 barrels of crude oil with the revenue of selling 10000
barrels of RBOB gasoline. The futures contract represent 1000 barrels of
crude oil and 42000 gallons of gasoline.

```
CostOfHedge = Price * 10000    % Option premium

% The hedge cost $66386 to implement and guarantees that neither a fall in
% RBOB gasoline prices or an increase in WTI crude oil prices will diminish
% the refining margin below $25.

ProfitMargin = 14 * 10000    %$
CrackingMargin = ProfitMargin - CostOfHedge
```

```
% This strategy allows a cracking margin of $73613.


CostOfHedge =

      66386.17


ProfitMargin =

     140000.00


CrackingMargin =

      73613.83
```

Another strategy for the refiner could be to buy the $22 puts at a price of $5.38.

```
StrikeNew = 22;

PriceNew = spreadbyfd(RateSpec, StockSpec1, StockSpec2, Settle, ...
                      Maturity, OptSpec, StrikeNew, Corr, 'AmericanOpt', 1

% This time the hedge would have cost $53823, but it also guarantees a $11
% per barrel or $56176 cracking margin.

NewCostOfHedge = PriceNew * 10000   % Option premium

NewProfitMargin = 11 * 10000
CrackingMargin = NewProfitMargin - NewCostOfHedge


PriceNew =

          5.38


NewCostOfHedge =
```

```
      53823.62


NewProfitMargin =

     110000.00


CrackingMargin =

      56176.38
```

Example 3: Using Collars to Reduce the Cost of Hedging

A refiner is concerned about its cost of hedging and decides to use a collar strategy. In April the crack spread is trading at \$4.23 per barrel. The refiner is not convinced to lock in this margin, but also wants to protect against price changes causing the refinery margin to decrease less than \$4 per barrel.

```
% Price and volatility of heating oil
Price1gallon = 2.52;         % $/gallon
Price1 = Price1gallon * 42;   % $/barrel
Vol1 = 0.38;
Div1 = 0.0762;

% Price and volatility of WTI crude oil
Price2 = 101.61;          % $/barrel
Vol2 = 0.34;
Div2 = 0.1169;


% To accomplish the collar strategy the refiner sells a call spread option
% with a strike of $4.50 and uses the premium income to offset the cost of
% purchasing a put spread option with a strike of $4. This allows the
% refiner to benefit if market prices move up and protects it if market
% prices move down.

% Assume the following data
```

```
Strike = [4.50;4];
OptSpec = {'call';'put'};
Settle =   '01-April-2013';
Maturity = '01-June-2013';
Corr = 0.35;        % Correlation of underlying commodities
```

Define RateSpec and StockSpec

```
% Define RateSpec
Rate = 0.035;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
    'EndDates', Maturity, 'Rates', Rate, 'Compounding', ...
    Compounding, 'Basis', Basis);

% Define StockSpec for the two assets
StockSpec1 = stockspec(Vol1, Price1, 'Continuous', Div1);
StockSpec2 = stockspec(Vol2, Price2, 'Continuous', Div2);
```

Price the Crack Spread Options

```
% Use the function 'spreadbybjs' in the Financial Instruments Toolbox(TM) t
% price the spread options using the Bjerksund and Stensland model.

Price = spreadbybjs(RateSpec, StockSpec1, StockSpec2, Settle, ...
                        Maturity, OptSpec, Strike, Corr)

% The collar strategy allows the refiner to  reduced the cost of the hedge
% to $0.63:
% CostOfHedge = Premium of Call - Premium of Put
CostOfHedge = Price(1) - Price(2)


Price =

          7.06
          6.43
```

```
CostOfHedge =

        0.63
```

The refiner is protected if the crack spread narrows to less than $4. If the crack spread widens to more than $4.50, the refiner will not benefit over this amount if he has hedged 100% of all its market exposure.

```
set(0, 'format', OldFormat);
```

Pricing Equity Derivatives Using Trees

Computing Instrument Prices

The portfolio pricing functions `crrprice`, `eqpprice`, and `ittprice` calculate the price of any set of supported instruments based on a binary equity price tree or an implied trinomial price tree. These functions are capable of pricing the following instrument types:

- Vanilla stock options
  - American and European puts and calls
- Exotic options
  - Asian
  - Barrier
  - Compound
  - Lookback
  - Stock options (Bermuda put and call schedules)

The syntax for calling the function `crrprice` is:

```
[Price, PriceTree] = crrprice(CRRTree, InstSet, Options)
```

The syntax for `eqpprice` is:

```
[Price, PriceTree] = eqpprice(EQPTree, InstSet, Options)
```

The syntax for `ittprice` is:

```
Price = ittprice(ITTTree, ITTInstSet, Options)
```

These functions require two input arguments: the equity price tree and the set of instruments, `InstSet`, and allow a third optional argument.

Required Arguments

`CRRTree` is a CRR equity price tree created using `crrtree`. `EQPTree` is an equal probability equity price tree created using `eqptree`. `ITTTree` is an ITT equity price tree created using `itttree`. See "Building Equity Binary Trees" on page 3-3 and "Building Implied Trinomial Trees" on page 3-8 to learn how to create these structures.

`InstSet` is a structure that represents the set of instruments to be priced independently using the model.

Optional Argument

You can enter a third optional argument, `Options`, used when pricing barrier options. For more specific information, see Appendix B, "Derivatives Pricing Options".

These pricing functions internally classify the instruments and call the appropriate individual instrument pricing function for each of the instrument types. The CRR pricing functions are `asianbycrr`, `barrierbycrr`, `compoundbycrr`, `lookbackbycrr`, and `optstockbycrr`. A similar set of functions exists for EQP and ITT pricing. You can also use these functions directly to calculate the price of sets of instruments of the same type. See the reference pages for these individual functions for further information.

Computing Prices Using CRR

Consider the following example, which uses the portfolio and stock price data in the MAT-file `deriv.mat` included in the toolbox. Load the data into the MATLAB workspace.

```
load deriv.mat
```

Use the MATLAB `whos` command to display a list of the variables loaded from the MAT-file.

```
Name              Size           Bytes  Class      Attributes

BDTInstSet        1x1            15956  struct
BDTTree           1x1             5138  struct
BKInstSet         1x1            15946  struct
BKTree            1x1             5904  struct
CRRInstSet        1x1            12434  struct
CRRTree           1x1             5058  struct
EQPInstSet        1x1            12434  struct
EQPTree           1x1             5058  struct
HJMInstSet        1x1            15948  struct
HJMTree           1x1             5838  struct
HWInstSet         1x1            15946  struct
HWTree            1x1             5904  struct
ITTInstSet        1x1            12438  struct
ITTTree           1x1             8862  struct
ZeroInstSet       1x1            10282  struct
ZeroRateSpec      1x1             1580  struct
```

`CRRTree` and `CRRInstSet` are the required input arguments to call the function `crrprice`.

Use `instdisp` to examine the set of instruments contained in the variable `CRRInstSet`.

`instdisp(CRRInstSet)`

```
Index Type     OptSpec Strike Settle      ExerciseDates AmericanOpt Name  Quantity
1     OptStock call    105    01-Jan-2003 01-Jan-2005   1           Call1 10
2     OptStock put     105    01-Jan-2003 01-Jan-2006   0           Put1  5

Index Type     OptSpec Strike Settle      ExerciseDates AmericanOpt BarrierSpec Barrier Rebate Name     Q
3     Barrier  call    105    01-Jan-2003 01-Jan-2006   1           ui          102     0      Barrier1 1

Index Type     UOptSpec UStrike USettle     UExerciseDates UAmericanOpt COptSpec CStrike CSettle      C
4     Compound call     130     01-Jan-2003 01-Jan-2006   1            put      5       01-Jan-2003  0
```

```
Index Type      OptSpec Strike Settle        ExerciseDates AmericanOpt Name      Quantity
5     Lookback call    115    01-Jan-2003   01-Jan-2006   0           Lookback1 7
6     Lookback call    115    01-Jan-2003   01-Jan-2007   0           Lookback2 9


Index Type  OptSpec Strike Settle        ExerciseDates AmericanOpt AvgType    AvgPrice AvgDate Name    Quan
7     Asian put     110    01-Jan-2003   01-Jan-2006   0           arithmetic NaN      NaN     Asian1 4
8     Asian put     110    01-Jan-2003   01-Jan-2007   0           arithmetic NaN      NaN     Asian2 6
```

Note Because of space considerations, the compound option above (`Index` 4) has been condensed to fit the page. The `instdisp` command displays all compound option fields on your computer screen.

The instrument set contains eight instruments:

- Two vanilla options (`Call1`, `Put1`)

- One barrier option (`Barrier1`)

- One compound option (`Compound1`)

- Two lookback options (`Lookback1`, `Lookback2`)

- Two Asian options (`Asian1`, `Asian2`)

Each instrument has a corresponding index that identifies the instrument prices in the price vector returned by `crrprice`.

Now use `crrprice` to calculate the price of each instrument in the instrument set.

```
Price = crrprice(CRRTree, CRRInstSet)

Price =

    8.2863
    2.5016
   12.1272
    3.3241
    7.6015
   11.7772
```

```
      4.1797
      3.4219
```

Computing Prices Using EQP

Load the data into the MATLAB workspace.

```
load deriv.mat
```

Use the MATLAB whos command to display a list of the variables loaded from the MAT-file.

```
 Name             Size           Bytes  Class      Attributes

  BDTInstSet       1x1            15956  struct
  BDTTree          1x1             5138  struct
  BKInstSet        1x1            15946  struct
  BKTree           1x1             5904  struct
  CRRInstSet       1x1            12434  struct
  CRRTree          1x1             5058  struct
  EQPInstSet       1x1            12434  struct
  EQPTree          1x1             5058  struct
  HJMInstSet       1x1            15948  struct
  HJMTree          1x1             5838  struct
  HWInstSet        1x1            15946  struct
  HWTree           1x1             5904  struct
  ITTInstSet       1x1            12438  struct
  ITTTree          1x1             8862  struct
  ZeroInstSet      1x1            10282  struct
  ZeroRateSpec     1x1             1580  struct
```

EQPTree and EQPInstSet are the input arguments required to call the function eqpprice.

Use the command instdisp to examine the set of instruments contained in the variable EQPInstSet.

```
instdisp(EQPInstSet)
```

```
Index Type     OptSpec Strike Settle       ExerciseDates AmericanOpt Name  Quantity
1     OptStock call    105    01-Jan-2003  01-Jan-2005   1           Call1 10
```

2	OptStock put	105	01-Jan-2003	01-Jan-2006	0		Put1	5

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	BarrierSpec	Barrier	Rebate	Name	Q
3	Barrier	call	105	01-Jan-2003	01-Jan-2006	1	ui	102	0	Barrier1	1

Index	Type	UOptSpec	UStrike	USettle	UExerciseDates	UAmericanOpt	COptSpec	CStrike	CSettle	C
4	Compound	call	130	01-Jan-2003	01-Jan-2006	1	put	5	01-Jan-2003	0

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	Name	Quantity
5	Lookback	call	115	01-Jan-2003	01-Jan-2006	0	Lookback1	7
6	Lookback	call	115	01-Jan-2003	01-Jan-2007	0	Lookback2	9

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	AvgType	AvgPrice	AvgDate	Name	Quan
7	Asian	put	110	01-Jan-2003	01-Jan-2006	0	arithmetic	NaN	NaN	Asian1	4
8	Asian	put	110	01-Jan-2003	01-Jan-2007	0	arithmetic	NaN	NaN	Asian2	6

```
>> instdisp(EQPInstSet)
```

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	Name	Quantity
1	OptStock	call	105	01-Jan-2003	01-Jan-2005	1	Call1	10
2	OptStock	put	105	01-Jan-2003	01-Jan-2006	0	Put1	5

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	BarrierSpec	Barrier	Rebate	Name	Q
3	Barrier	call	105	01-Jan-2003	01-Jan-2006	1	ui	102	0	Barrier1	1

Index	Type	UOptSpec	UStrike	USettle	UExerciseDates	UAmericanOpt	COptSpec	CStrike	CSettle	C
4	Compound	call	130	01-Jan-2003	01-Jan-2006	1	put	5	01-Jan-2003	0

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	Name	Quantity
5	Lookback	call	115	01-Jan-2003	01-Jan-2006	0	Lookback1	7
6	Lookback	call	115	01-Jan-2003	01-Jan-2007	0	Lookback2	9

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	AvgType	AvgPrice	AvgDate	Name	Quan
7	Asian	put	110	01-Jan-2003	01-Jan-2006	0	arithmetic	NaN	NaN	Asian1	4
8	Asian	put	110	01-Jan-2003	01-Jan-2007	0	arithmetic	NaN	NaN	Asian2	6

> **Note** Because of space considerations, the compound option above (`Index 4`) has been condensed to fit the page. The `instdisp` command displays all compound option fields on your computer screen.

The instrument set contains eight instruments:

- Two vanilla options (`Call1`, `Put1`)
- One barrier option (`Barrier1`)
- One compound option (`Compound1`)
- Two lookback options (`Lookback1`, `Lookback2`)
- Two Asian options (`Asian1`, `Asian2`)

Each instrument has a corresponding index that identifies the instrument prices in the price vector returned by `eqpprice`.

Now use `eqpprice` to calculate the price of each instrument in the instrument set.

```
Price = eqpprice(EQPTree, EQPInstSet)

Price =

    8.4791
    2.6375
   12.2632
    3.5091
    8.7941
   12.9577
    4.7444
    3.9178
```

Computing Prices Using ITT

Consider the following example, which uses the portfolio and stock price data in the MAT-file `deriv.mat` included in the toolbox. Load the data into the MATLAB workspace.

```
load deriv.mat
```

Use the MATLAB `whos` command to display a list of the variables loaded from the MAT-file.

```
Name              Size          Bytes  Class     Attributes

BDTInstSet        1x1           15956  struct
BDTTree           1x1            5138  struct
BKInstSet         1x1           15946  struct
BKTree            1x1            5904  struct
CRRInstSet        1x1           12434  struct
CRRTree           1x1            5058  struct
EQPInstSet        1x1           12434  struct
EQPTree           1x1            5058  struct
HJMInstSet        1x1           15948  struct
HJMTree           1x1            5838  struct
HWInstSet         1x1           15946  struct
HWTree            1x1            5904  struct
ITTInstSet        1x1           12438  struct
ITTTree           1x1            8812  struct
ZeroInstSet       1x1           10282  struct
ZeroRateSpec      1x1            1580  struct
```

`ITTTree` and `ITTInstSet` are the input arguments required to call the function `ittprice`. Use the command `instdisp` to examine the set of instruments contained in the variable `ITTInstSet`.

```
instdisp(ITTInstSet)
```

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	Name	Quantity
1	OptStock	call	105	01-Jan-2003	01-Jan-2005	1	Call1	10
2	OptStock	put	105	01-Jan-2003	01-Jan-2006	0	Put1	5

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	BarrierSpec	Barrier	Rebate	Name	Q
3	Barrier	call	105	01-Jan-2003	01-Jan-2006	1	ui	102	0	Barrier1	1

Index	Type	UOptSpec	UStrike	USettle	UExerciseDates	UAmericanOpt	COptSpec	CStrike	CSettle	C
4	Compound	call	130	01-Jan-2003	01-Jan-2006	1	put	5	01-Jan-2003	0

```
Index Type      OptSpec Strike Settle        ExerciseDates AmericanOpt Name       Quantity
5     Lookback call    115    01-Jan-2003   01-Jan-2006   0           Lookback1 7
6     Lookback call    115    01-Jan-2003   01-Jan-2007   0           Lookback2 9

Index Type  OptSpec Strike Settle        ExerciseDates AmericanOpt AvgType      AvgPrice AvgDate Name    Quan
7     Asian put     110    01-Jan-2003   01-Jan-2006   0           arithmetic NaN      NaN     Asian1 4
8     Asian put     110    01-Jan-2003   01-Jan-2007   0           arithmetic NaN      NaN     Asian2 6

>> instdisp(ITTInstSet)
Index Type      OptSpec Strike Settle        ExerciseDates AmericanOpt Name  Quantity
1     OptStock call    95     01-Jan-2006   31-Dec-2008   1           Call1 10
2     OptStock put     80     01-Jan-2006   01-Jan-2010   0           Put1  4

Index Type      OptSpec Strike Settle        ExerciseDates AmericanOpt BarrierSpec Barrier Rebate Name      Q
3     Barrier  call    85     01-Jan-2006   31-Dec-2008   1           ui          115     0      Barrier1 1

Index Type      UOptSpec UStrike USettle       UExerciseDates UAmericanOpt COptSpec CStrike CSettle      C
4     Compound call     99      01-Jan-2006   01-Jan-2010    1           put      5       01-Jan-2006  0

Index Type      OptSpec Strike Settle        ExerciseDates AmericanOpt Name      Quantity
5     Lookback call    85     01-Jan-2006   01-Jan-2008   0           Lookback1 7
6     Lookback call    85     01-Jan-2006   01-Jan-2010   0           Lookback2 9

Index Type  OptSpec Strike Settle        ExerciseDates AmericanOpt AvgType      AvgPrice AvgDate Name    Quan
7     Asian call    55     01-Jan-2006   01-Jan-2008   0           arithmetic NaN      NaN     Asian1 5
8     Asian call    55     01-Jan-2006   01-Jan-2010   0           arithmetic NaN      NaN     Asian2 7
```

The instrument set contains eight instruments:

- Two vanilla options (Call1, Put1)

- One barrier option (Barrier1)

- One compound option (Compound1)

- Two lookback options (Lookback1, Lookback2)

- Two Asian options (Asian1, Asian2)

Each instrument has a corresponding index that identifies the instrument prices in the price vector returned by ittprice.

Now use `ittprice` to calculate the price of each instrument in the instrument set.

```
Price = ittprice(ITTTree, ITTInstSet)

Price =

    1.650
   10.68
    2.407
    3.229
    0.542
    6.184
    3.205
    6.607
```

Examining Output from the Pricing Functions

The prices in the output vector `Price` correspond to the prices at observation time zero (`tObs = 0`), which is defined as the valuation date of the equity tree. The instrument indexing within `Price` is the same as the indexing within `InstSet`.

In the CRR example, the prices in the `Price` vector correspond to the instruments in this order.

```
InstNames = instget(CRRInstSet, 'FieldName','Name')

InstNames =

Call1
Put1
Barrier1
Compound1
Lookback1
Lookback2
Asian1
Asian2
```

Consequently, in the Price vector, the fourth element, 3.3241, represents the price of the fourth instrument (Compound1), and the sixth element, 11.7772, represents the price of the sixth instrument (Lookback2).

In the ITT example, the prices in the Price vector correspond to the instruments in this order.

```
InstNames = instget(ITTInstSet, 'FieldName','Name')

InstNames =

Call1
Put1
Barrier1
Compound1
Lookback1
Lookback2
Asian1
Asian2
```

Consequently, in the Price vector, the first element, 1.650, represents the price of the first instrument (Call1), and the eight element, 6.607, represents the price of the eighth instrument (Asian2).

Price Tree Output for CRR

If you call a pricing function with two output arguments, for example:

```
[Price, PriceTree] = crrprice(CRRTree, CRRInstSet)
```

you generate a price tree structure along with the price information.

This price tree structure PriceTree holds all pricing information.

```
PriceTree =
FinObj: 'BinPriceTree'
PTree: {[8x1 double] [8x2 double] [8x3 double] [8x4 double] [8x5 double]}
tObs: [0 1 2 3 4]
dObs: [731582 731947 732313 732678 733043]
```

The first field of this structure, `FinObj`, indicates that this structure represents a price tree. The second field, `PTree`, is the tree holding the prices of the instruments in each node of the tree. Finally, the third and fourth fields, `tObs` and `dObs`, represent the observation time and date of each level of `PTree`, with `tObs` using units in terms of compounding periods.

Using the command-line interface, you can directly examine `PriceTree.PTree`, the field within the `PriceTree` structure that contains the price tree with the price vectors at every state. The first node represents `tObs = 0`, corresponding to the valuation date.

```
PriceTree.PTree{1}

ans =
8.2863
2.5016
12.1272
3.3241
7.6015
11.7772
4.1797
3.4219
```

With this interface, you can observe the prices for all instruments in the portfolio at a specific time.

The function `eqpprice` also returns a price tree that you can examine in the same way.

Price Tree Output for ITT

If you call a pricing function with two output arguments, for example:

```
[Price, PriceTree] = ittprice(ITTTree, ITTInstSet)
```

you generate a price tree structure along with the price information.

This price tree structure `PriceTree` holds all pricing information.

```
PriceTree =
```

```
    FinObj: 'TrinPriceTree'
     PTree: {[8x1 double]  [8x3 double]  [8x5 double]  [8x7 double]  [8x9 double]}
      tObs: [0 1 2 3 4]
      dObs: [732678 733043 733408 733773 734139]
```

The first field of this structure, FinObj, indicates that this structure represents a trinomial price tree. The second field, PTree is the tree holding the prices of the instruments in each node of the tree. Finally, the third and fourth fields, tObs and dObs, represent the observation time and date of each level of PTree, with tObs using units in terms of compounding periods.

Using the command-line interface, you can directly examine PriceTree.PTree, the field within the PriceTree structure that contains the price tree with the price vectors at every state. The first node represents tObs = 0, corresponding to the valuation date.

```
PriceTree.PTree{1}

ans =

     1.6506
    10.6832
     2.4074
     3.2294
     0.5426
     6.1845
     3.2052
     6.6074
```

With this interface, you can observe the prices for all instruments in the portfolio at a specific time.

Prices for Lookback and Asian Options for Equity Trees

Lookback options and Asian options are path dependent, and, as such, there are no unique prices for any node except the root node. Consequently, the corresponding values for lookback and Asian options in the price tree are set to NaN, the only exception being the root node. This becomes apparent if you examine the prices in the second node (tobs = 1) of the CRR price tree:

```
PriceTree.PTree{2}
```

```
ans =

   11.9176         0
    0.9508    7.1914
   16.4600    2.6672
    2.5896    5.0000
       NaN       NaN
       NaN       NaN
       NaN       NaN
       NaN       NaN
```

Examining the prices in the second node (`tobs = 1`) of the ITT price tree displays:

```
PriceTree.PTree{2}

 ans =

    3.9022         0         0
    6.3736   13.3743   22.1915
    5.6914         0         0
    2.7663    3.8594    5.0000
       NaN       NaN       NaN
       NaN       NaN       NaN
       NaN       NaN       NaN
       NaN       NaN       NaN
```

Computing Instrument Sensitivities

Sensitivities can be reported either as dollar price changes or percentage price changes. The delta, gamma, and vega sensitivities that the toolbox computes are dollar sensitivities.

The functions `crrsens`, `eqpsens`, and `ittsens` compute the delta, gamma, and vega sensitivities of instruments using a stock tree. They also optionally return the calculated price for each instrument. The sensitivity functions require the same two input arguments used by the pricing functions (`CRRTree` and `CRRInstSet` for CRR, `EQPTree` and `EQPInstSet` for EQP, and `ITTTree` and `ITTInstSet` for ITT).

As with the instrument pricing functions, the optional input argument
Options is also allowed. You would include this argument if you want a
sensitivity function to generate a price for a barrier option as one of its
outputs and want to control the method that the toolbox uses to perform
the pricing operation. See Appendix B, "Derivatives Pricing Options" or the
derivset function for more information.

For path-dependent options (lookback and Asian), delta and gamma are
computed by finite differences in calls to crrprice, eqpprice, and ittprice.
For the other options (stock option, barrier, and compound), delta and gamma
are computed from the CRR, EQP, and ITT trees and the corresponding option
price tree. (See Chriss, Neil, *Black-Scholes and Beyond*, pp. 308-312.)

CRR Sensitivities Example

The calling syntax for the sensitivity function is:

```
[Delta, Gamma, Vega, Price] = crrsens(CRRTree, InstSet, Options)
```

Using the example data in deriv.mat, calculate the sensitivity of the
instruments.

```
load deriv.mat
[Delta, Gamma, Vega, Price] = crrsens(CRRTree, CRRInstSet);
```

You can conveniently examine the sensitivities and the prices by arranging
them into a single matrix.

```
format bank
All = [Delta, Gamma, Vega, Price]

All =

      0.59              0.04         53.45            8.29
     -0.31              0.03         67.00            2.50
      0.69              0.03         67.00           12.13
     -0.12             -0.01        -98.08            3.32
     -0.40         -45926.32         88.18            7.60
     -0.42        -112143.15        119.19           11.78
      0.60          45926.32         49.21            4.18
      0.82         112143.15         41.71            3.42
```

As with the prices, each row of the sensitivity vectors corresponds to the similarly indexed instrument in `CRRInstSet`. To view the per-dollar sensitivities, divide each dollar sensitivity by the corresponding instrument price.

```
All = [Delta ./ Price, Gamma ./ Price, Vega ./ Price, Price]

All =

      0.07             0.00             6.45             8.29
     -0.12             0.01            26.78             2.50
      0.06             0.00             5.53            12.13
     -0.04            -0.00           -29.51             3.32
     -0.05         -6041.77            11.60             7.60
     -0.04         -9522.02            10.12            11.78
      0.14         10987.98            11.77             4.18
      0.24         32771.92            12.19             3.42
```

ITT Sensitivities Example

The calling syntax for the sensitivity function is:

```
[Delta, Gamma, Vega, Price] = ittsens(ITTTree, ITTInstSet,
Options)
```

Using the example data in `deriv.mat`, calculate the sensitivity of the instruments.

```
load deriv.mat

warning('off', 'fininst:itttree:Extrapolation');
[Delta, Gamma, Vega, Price] = ittsens(ITTTree, ITTInstSet);
```

You can conveniently examine the sensitivities and the prices by arranging them into a single matrix.

```
format bank
All = [Delta, Gamma, Vega, Price]

All =
```

0.24	0.03	19.35	1.65
-0.43	0.02	49.69	10.68
0.35	0.04	12.29	2.41
-0.07	0.00	6.73	3.23
0.63	142945.66	38.90	0.54
0.60	22703.21	68.92	6.18
0.32	-142945.66	18.48	3.21
0.67	-22703.21	17.75	6.61

As with the prices, each row of the sensitivity vectors corresponds to the similarly indexed instrument in ITTInstSet.

Note In this example, the extrapolation warnings are turned off before calculating the sensitivities to avoid displaying many warnings on the Command Window as the sensitivities are calculated.

If the extrapolation warnings are turned on

```
warning('on', 'fininst:itttree:Extrapolation');
```

and ittsens is rerun, the extrapolation warnings scroll as the command executes:

```
[Delta, Gamma, Vega, Price] = ittsens(ITTTree, ITTInstSet)
```

```
Warning: The option set specified in StockOptSpec was too narrow for the generated tree.
This makes extrapolation necessary. The list of options outside of the
range of those specified in StockOptSpec are:

Option Type: 'call'   Maturity: 01-Jan-2007  Strike=66.3529
Option Type: 'put'    Maturity: 01-Jan-2007  Strike=50.0061
Option Type: 'put'    Maturity: 01-Jan-2008  Strike=50.0061
Option Type: 'put'    Maturity: 31-Dec-2008  Strike=50.0061
Option Type: 'call'   Maturity: 01-Jan-2010  Strike=155.0141
Option Type: 'put'    Maturity: 01-Jan-2010  Strike=50.006
> In itttree>InterpOptPrices at 675
  In itttree at 277
  In stocktreesens>stocktreedeltagamma_PD at 127
```

```
  In stocktreesens at 83
  In ittsens at 81

Warning: The option set specified in StockOptSpec was too narrow for the generated tree.
This made extrapolation necessary. Below is a list of the options that were outside of the
range of those specified in StockOptSpec.

Option Type: 'call'   Maturity: 01-Jan-2007  Strike=66.3367
Option Type: 'put'    Maturity: 01-Jan-2007  Strike=37.6773
Option Type: 'call'   Maturity: 01-Jan-2008  Strike=66.3367
Option Type: 'put'    Maturity: 01-Jan-2008  Strike=28.3951
Option Type: 'call'   Maturity: 31-Dec-2008  Strike=66.3367
Option Type: 'call'   Maturity: 01-Jan-2010  Strike=66.3367
Option Type: 'put'    Maturity: 01-Jan-2010  Strike=16.1276

> In itttree>InterpOptPrices at 675
  In itttree at 277
  In stocktreesens>stocktreedeltagamma_PD at 131
  In stocktreesens at 83
  In ittsens at 81

Warning: The option set specified in StockOptSpec was too narrow for the generated tree.
This made extrapolation necessary. Below is a list of the options that were outside of the
range of those specified in StockOptSpec.

Option Type: 'call'   Maturity: 01-Jan-2007  Strike=67.2897
Option Type: 'put'    Maturity: 01-Jan-2007  Strike=37.1528
Option Type: 'put'    Maturity: 01-Jan-2008  Strike=27.6066
Option Type: 'put'    Maturity: 31-Dec-2008  Strike=20.5132
Option Type: 'call'   Maturity: 01-Jan-2010  Strike=164.0157
Option Type: 'put'    Maturity: 01-Jan-2010  Strike=15.2424

> In itttree>InterpOptPrices at 675
  In itttree at 277
  In stocktreesens>stocktreevega at 191
  In stocktreesens at 92
  In ittsens at 81
```

These warnings are a consequence of having to extrapolate to find the option price of the tree nodes. In this example, the set of inputs options was too

narrow for the shift in the tree nodes introduced by the disturbance used to calculate the sensitivities. As a consequence extrapolation for some of the nodes was needed. Since the input data is quite close the extrapolated data, the error introduced by extrapolation is fairly low.

Graphical Representation of Equity Derivative Trees

You can use the function `treeviewer` to display a graphical representation of a tree, allowing you to examine interactively the prices and rates on the nodes of the tree until maturity. The graphical representations of CRR, EQP, and LR trees are equivalent to Black-Derman-Toy (BDT) trees, given that they are all binary recombining trees. The graphical representations of ITT trees are equivalent to Hull-White (HW) trees, given that they are all trinomial recombining trees. See "Graphical Representation of Trees" on page 2-113 for an overview on the use of `treeviewer` with CRR trees, EQP trees, LR trees, and ITT trees and their corresponding option price trees. Follow the instructions for BDT trees.

Equity Derivatives Using Closed-Form Solutions

Introduction

Financial Instruments Toolbox software supports four types of closed-form solutions and analytical approximations to calculate price and sensitivities (greeks) of vanilla options:

- Black-Scholes model
- Black model
- Roll-Geske-Whaley model
- Bjerksund-Stensland 2002 model

Black-Scholes Model

The Black-Scholes model is one of the most commonly used models to price European calls and puts. It serves as a basis for many closed-form solutions used for pricing options. The standard Black-Scholes model is based on the following assumptions:

- There are no dividends paid during the life of the option.
- The option can only be exercised at maturity.

- The markets operate under a Markov process in continuous time.

- No commissions are paid.

- The risk-free interest rate is known and constant.

- Returns on the underlying stocks are log-normally distributed.

Note The Black-Scholes model implemented in Financial Instruments Toolbox software allows dividends. The following three dividend methods are supported:

- Cash dividend

- Continuous dividend yield

- Constant dividend yield

However, not all Black-Scholes closed-form pricing functions support all three dividend methods. For more information on specifying the dividend methods, see stockspec.

Closed-form solutions based on a Black-Scholes model support the following tasks.

Task	Function
Price European options with different dividends using the Black-Scholes option pricing model.	optstockbybls
Calculate European option prices and sensitivities using the Black-Scholes option pricing model.	optstocksensbybls
Calculate implied volatility on European options using the Black-Scholes option pricing model.	impvbybls
Price European simple chooser options using Black-Scholes model.	chooserbybls

For an example using the Black-Scholes model, see "Pricing Using the Black-Scholes Model" on page 3-89.

Black Model

Use the Black model for pricing European options on physical commodities, forwards or futures. The Black model supported by Financial Instruments Toolbox software is a special case of the Black-Scholes model. The Black model uses a forward price as an underlier in place of a spot price. The assumption is that the forward price at maturity of the option is log-normally distributed.

Closed-form solutions for a Black model support the following tasks.

Task	Function
Price European options on futures using the Black option pricing model.	optstockbyblk
Calculate European option prices and sensitivities on futures using the Black option pricing model.	optstocksensbyblk
Calculate implied volatility for European options using the Black option pricing model.	impvbyblk

For an example using the Black model, see "Pricing Using the Black Model" on page 3-91.

Roll-Geske-Whaley Model

Use the Roll-Geske-Whaley approximation method to price American call options paying a single cash dividend. This model is based on the modification of the observed stock price for the present value of the dividend and also supports a compound option to account for the possibility of early exercise. The Roll-Geske-Whaley model has drawbacks due to an escrowed dividend price approach which may lead to arbitrage. For further explanation, see *Options, Futures, and Other Derivatives* by John Hull.

Closed-form solutions for a Roll-Geske-Whaley model support the following tasks.

Task	Function
Price American call options with a single cash dividend using the Roll-Geske-Whaley option pricing model.	optstockbyrgw
Calculate American call prices and sensitivities using the Roll-Geske-Whaley option pricing model.	optstocksensbyrgw
Calculate implied volatility for American call options using the Roll-Geske-Whaley option pricing model.	impvbyrgw

For an example using the Roll-Geske-Whaley model, see "Pricing Using the Roll-Geske-Whaley Model" on page 3-92.

Bjerksund-Stensland 2002 Model

Use the Bjerksund-Stensland 2002 model for pricing American puts and calls with continuous dividend yield. This model works by dividing the time to maturity of the option in two separate parts, each with its own flat exercise boundary (trigger price). The Bjerksund-Stensland 2002 method is a generalization of the Bjerksund and Stensland 1993 method and is considered to be computationally efficient . For further explanation, see *Closed Form Valuation of American Options* by Bjerksund and Stensland.

Closed-form solutions for a Bjerksund-Stensland 2002 model support the following tasks.

Task	Function
Price American options with continuous dividend yield using the	optstockbybjs

Task	Function
Bjerksund-Stensland 2002 option pricing model.	
Calculate American options prices and sensitivities using the Bjerksund-Stensland 2002 option pricing model.	optstocksensbybjs
Calculate implied volatility for American options using the Bjerksund-Stensland 2002 option pricing model.	impvbybjs

For an example using the Bjerksund-Stensland 2002 model, see "Pricing Using the Bjerksund-Stensland Model" on page 3-93.

Pricing Using the Black-Scholes Model

Consider a European stock option with an exercise price of $40 on January 1, 2008 that expires on July 1, 2008. Assume the underlying stock pays dividends of $0.50 on March 1 and June 1. The stock is trading at $40 and has a volatility of 30% per annum. The risk-free rate is 4% per annum. Using this data, calculate the price of a call and a put option on the stock using the Black-Scholes option pricing model:

```
Strike = 40;
AssetPrice = 40;
Sigma = .3;
Rates = 0.04;
Settle = 'Jan-01-08';
Maturity = 'Jul-01-08';

Div1 = 'March-01-2008';
Div2 = 'Jun-01-2008';
```

Create RateSpec and StockSpec:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, 'EndDates',...
Maturity, 'Rates', Rates, 'Compounding', -1);

StockSpec = stockspec(Sigma, AssetPrice, {'cash'}, 0.50,{Div1,Div2});
```

Define two options, one call and one put:

```
OptSpec = {'call'; 'put'};
```

Calculate the price of the European options:

```
Price = optstockbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)

Price =

    3.2063
    3.4027
```

The first element of the `Price` vector represents the price of the call ($3.21); the second is the price of the put ($3.40). Use the function `optstocksensbybls` to compute six sensitivities for the Black-Scholes model: `delta`, `gamma`, `vega`, `lambda`, `rho`, and `theta` and the `price` of the option.

The selection of output parameters and their order is determined by the optional input parameter `OutSpec`. This parameter is a cell array of strings, each one specifying a desired output parameter. The order in which these output parameters are returned by the function is the same as the order of the strings contained in `OutSpec`.

As an example, consider the same options as the previous example. To calculate their `Delta`, `Rho`, `Price`, and `Gamma`, build the cell array `OutSpec` as follows:

```
OutSpec = {'delta', 'rho', 'price', 'gamma'};

[Delta, Rho, Price, Gamma] =optstocksensbybls(RateSpec, StockSpec, Settle,...
Maturity, OptSpec, Strike, 'OutSpec', OutSpec)

Delta =

     0.5328
    -0.4672


Rho =
```

```
    8.7902
  -10.8138


Price =

    3.2063
    3.4027


Gamma =

    0.0480
    0.0480
```

Pricing Using the Black Model

Consider two European call options on a futures contract with exercise prices of $20 and $25 that expire on September 1, 2008. Assume that on May 1, 2008 the contract is trading at $20 and has a volatility of 35% per annum. The risk-free rate is 4% per annum. Using this data, calculate the price of the call futures options using the Black model:

```
Strike = [20; 25];
AssetPrice = 20;
Sigma = .35;
Rates = 0.04;
Settle = 'May-01-08';
Maturity = 'Sep-01-08';
```

Create RateSpec and StockSpec:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1);

StockSpec = stockspec(Sigma, AssetPrice);
```

Define the call option:

```
OptSpec = {'call'};
```

Calculate price and all sensitivities of the European futures options:

```
OutSpec = {'All'}

[Delta, Gamma, Vega, Lambda, Rho, Theta, Price] = optstocksensbyblk(RateSpec,...
StockSpec, Settle, Maturity, OptSpec, Strike, 'OutSpec', OutSpec);

Price =

    1.5903
    0.3037
```

The first element of the `Price` vector represents the price of the call with an exercise price of $20 ($1.59); the second is the price of the call with an exercise price of $25 ($2.89).

The function `impvbyblk` is used to compute the implied volatility using the Black option pricing model. Assuming that the previous European call futures are trading at $1.5903 and $0.3037, you can calculate their implied volatility:

```
Volatility = impvbyblk(RateSpec, StockSpec, Settle, Maturity,...
OptSpec,  Strike, Price);
```

As expected, you get volatilities of 35%. If the call futures were trading at $1.50 and $0.50 in the market, the implied volatility would be 33% and 42%:

```
Volatility = impvbyblk(RateSpec, StockSpec, Settle, Maturity,...
OptSpec,  Strike, [1.50;0.5])

Volatility =

    0.3301
    0.4148
```

Pricing Using the Roll-Geske-Whaley Model

Consider two American call options, with exercise prices of $110 and $100 on June 1, 2008, that expire on June 1, 2009. Assume the underlying stock pays dividends of $0.001 on December 1, 2008. The stock is trading at $80 and has a volatility of 20% per annum. The risk-free rate is 6% per annum. Using this

data, calculate the price of the American calls using the Roll-Geske-Whaley option pricing model:

```
AssetPrice = 80;
Settle = 'Jun-01-2008';
Maturity = 'Jun-01-2009';
Strike = [110; 100];

Rate = 0.06;
Sigma  = 0.2;

DivAmount = 0.001;
DivDate = 'Dec-01-2008';
```

Create `RateSpec` and `StockSpec`:

```
StockSpec = stockspec(Sigma, AssetPrice, {'cash'}, DivAmount, DivDate);

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rate, 'Compounding', -1);
```

Calculate the call prices:

```
Price  = optstockbyrgw(RateSpec, StockSpec, Settle, Maturity, Strike)

Price =

    0.8398
    2.0236
```

The first element of the `Price` vector represents the price of the call with an exercise price of \$110 (\$0.84); the second is the price of the call with an exercise price of \$100 (\$2.02).

Pricing Using the Bjerksund-Stensland Model

Consider four American stock options (two calls and two puts) with an exercise price of \$100 that expire on July 1, 2008. Assume the underlying stock pays a continuous dividend yield of 4% as of January 1, 2008. The stock has a volatility of 20% per annum and the risk-free rate is 8% per annum. Using this data, calculate the price of the American calls and puts assuming

the following current prices of the stock: $80, $90 (for the calls) and $100 and $110 (for the puts):

```
Settle = 'Jan-1-2008';
Maturity = 'Jul-1-2008';
Strike = 100;
AssetPrice = [80; 90; 100; 110];
DivYield = 0.04;

Rate = 0.08;
Sigma = 0.20;
```

Create `RateSpec` and `StockSpec`:

```
StockSpec = stockspec(Sigma, AssetPrice, {'continuous'}, DivYield);

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rate, 'Compounding', -1);
```

Define the option type:

```
OptSpec = {'call'; 'call'; 'put'; 'put'};
```

Compute the option prices:

```
Price = optstockbybjs(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)

Price =

    0.4144
    2.1804
    4.7253
    1.7164
```

The first two elements of the `Price` vector represent the price of the calls ($0.41 and $2.18), the last two elements represent the price of the put options ($4.72 and $1.72). Use the function `optstocksensbybjs` to compute six sensitivities for the Bjerksund-Stensland model: `delta`, `gamma`, `vega`, `lambda`, `rho`, and `theta` and the `price` of the option. The selection of output parameters and their order is determined by the optional input parameter `OutSpec`. This parameter is a cell array of strings, each one specifying a

desired output parameter. The order in which these output parameters are returned by the function is the same as the order of the strings contained in `OutSpec`. As an example, consider the same options as the previous example. To calculate their `delta`, `gamma`, and `price`, build the cell array `OutSpec` as follows:

```
OutSpec = {'delta', 'gamma', 'price'};
```

The outputs of `optstocksensbybjs` will be in the same order as in `OutSpec`.

```
[Delta ,Gamma, Price]= optstocksensbybjs(RateSpec, StockSpec, Settle,...
Maturity, OptSpec, Strike, 'OutSpec', OutSpec)

Delta =

    0.0843
    0.2912
    0.4803
    0.2261

Gamma =

    0.0136
    0.0267
    0.0304
    0.0217

Price =

    0.4144
    2.1804
    4.7253
    1.7164
```

For more information on the Bjerksund-Stensland model, see "Closed-Form Solutions Modeling" on page C-9.

Pricing European Call Options Using Different Equity Models

This example illustrates how the Financial Instruments Toolbox™ can be used to price European vanilla call options using different equity models.

The example compares call option prices using the Cox-Ross-Rubinstein model, the Leisen-Reimer model and the Black-Scholes closed formula.

Define the Call Instrument

Consider a European call option, with an exercise price of $30 on January 1, 2010. The option expires on Sep 1, 2010. Assume that the underlying stock provides no dividends. The stock is trading at $25 and has a volatility of 35% per annum. The annualized continuously compounded risk-free rate is 1.11% per annum.

```
% Option
Settle = 'Jan-01-2010';
Maturity = 'Sep-01-2010';
Strike = 30;
OptSpec ='call';

% Stock
AssetPrice = 25;
Sigma = .35;
```

Create the Interest Rate Term Structure

```
StartDates = '01 Jan 2010';
EndDates =   '01 Jan 2013';
Rates = 0.0111;
ValuationDate ='01 Jan 2010';
Compounding = -1;

RateSpec = intenvset('Compounding',Compounding,'StartDates', StartDates,...
                     'EndDates', EndDates, 'Rates', Rates,'ValuationDate',
```

Create the Stock Structure

Suppose we wish to create two scenarios. The first one assumes that AssetPrice is currently $25, the option is out of the money (OTM). The second scenario assumes that the option is at the money (ATM), and therefore AssetPriceATM = 30.

```
AssetPriceATM = 30;

StockSpec = stockspec(Sigma, AssetPrice);
StockSpecATM = stockspec(Sigma, AssetPriceATM);
```

Price the Options Using the Black-Scholes Closed Formula

Use the function 'optstockbybls' in the Financial Instruments Toolbox to compute the price of the European call options.

```
% Price the option with AssetPrice = 25
PriceBLS = optstockbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, St

% Price the option with AssetPrice = 30
PriceBLSATM = optstockbybls(RateSpec, StockSpecATM, Settle, Maturity, OptSp
```

Build the Cox-Ross-Rubinstein Tree

```
% Create the time specification of the tree
NumPeriods = 15;

CRRTimeSpec = crrtimespec(ValuationDate, Maturity, NumPeriods);

% Build the tree
CRRTree = crrtree(StockSpec, RateSpec, CRRTimeSpec);
CRRTreeATM = crrtree(StockSpecATM, RateSpec, CRRTimeSpec);
```

Build the Leisen-Reimer Tree

```
% Create the time specification of the tree
LRTimeSpec = lrtimespec(ValuationDate, Maturity, NumPeriods);

% Use the default method 'PP1' (Peizer-Pratt method 1 inversion)to build
% the tree
LRTree = lrtree(StockSpec, RateSpec, LRTimeSpec, Strike);
LRTreeATM = lrtree(StockSpecATM, RateSpec, LRTimeSpec, Strike);
```

Price the Options Using the Cox-Ross-Rubinstein (CRR) Model

```
PriceCRR = optstockbycrr(CRRTree, OptSpec, Strike, Settle, Maturity);
PriceCRRATM = optstockbycrr(CRRTreeATM, OptSpec, Strike, Settle, Maturity);
```

Price the Options Using the Leisen-Reimer (LR) Model

```
PriceLR = optstockbylr(LRTree, OptSpec, Strike, Settle, Maturity);
PriceLRATM = optstockbylr(LRTreeATM, OptSpec, Strike, Settle, Maturity);
```

Compare BLS, CRR and LR Results

```
sprintf('PriceBLS: \t%f\nPriceCRR: \t%f\nPriceLR:\t%f\n', PriceBLS, ...
    PriceCRR, PriceLR)

sprintf('\t== ATM ==\nPriceBLS ATM: \t%f\nPriceCRR ATM: \t%f\nPriceLR ATM:\
    PriceCRRATM,   PriceLRATM)


ans =

PriceBLS:  1.275075
PriceCRR:  1.294979
PriceLR: 1.275838



ans =

 == ATM ==
PriceBLS ATM:  3.497891
PriceCRR ATM:  3.553938
PriceLR ATM: 3.498571
```

Convergence of CRR and LR Models to a BLS Solution

The following tables compare call option prices using the CRR and LR models against the results obtained with the Black-Scholes formula.

While the CRR binomial model and the Black-Scholes model converge as the number of time steps gets large and the length of each step gets small, this convergence, except for at the money options, is anything but smooth or uniform.

The tables below show that the Leisen-Reimer model reduces the size of the error with even as few steps of 45.

```
    Strike = 30,    Asset Price = 30
-----------------------------------
  #Steps         LR          CRR
   15          3.4986       3.5539
   25          3.4981       3.5314
   45          3.4980       3.5165
   65          3.4979       3.5108
   85          3.4979       3.5077
  105          3.4979       3.5058
  201          3.4979       3.5020
  501          3.4979       3.4996
  999          3.4979       3.4987


    Strike = 30,    Asset Price = 25
-----------------------------------
  #Steps         LR          CRR
   15          1.2758       1.2950
   25          1.2754       1.2627
   45          1.2751       1.2851
   65          1.2751       1.2692
   85          1.2751       1.2812
  105          1.2751       1.2766
  201          1.2751       1.2723
  501          1.2751       1.2759
  999          1.2751       1.2756
```

Analyze the Effect of the Number of Periods on the Price of the Options

The following graphs show how convergence changes as the number of steps in the binomial calculation increases, as well as the impact on convergence to changes to the stock price. Observe that the Leisen-Reimer model removes the oscillation and produces estimates close to the Black-Scholes model using only a small number of steps.

```
NPoints = 300;

% Cox-Ross-Rubinstein
NumPeriodCRR  = 5 : 1 : NPoints;
NbStepCRR     = length(NumPeriodCRR);
PriceCRR = nan(NbStepCRR, 1);
PriceCRRATM = PriceCRR;

for i = 1 : NbStepCRR
    CRRTimeSpec = crrtimespec(ValuationDate, Maturity, NumPeriodCRR(i));
    CRRT = crrtree(StockSpec, RateSpec, CRRTimeSpec);
    PriceCRR(i) = optstockbycrr(CRRT, OptSpec, Strike,ValuationDate, Maturi

    CRRTATM = crrtree(StockSpecATM, RateSpec, CRRTimeSpec);
    PriceCRRATM(i) = optstockbycrr(CRRTATM, OptSpec, Strike,ValuationDate,
end;


% Now with Leisen-Reimer
NumPeriodLR  = 5 : 2 : NPoints;
NbStepLR     = length(NumPeriodLR);
PriceLR = nan(NbStepLR, 1);
PriceLRATM = PriceLR;

for i = 1 : NbStepLR
    LRTimeSpec = lrtimespec(ValuationDate, Maturity, NumPeriodLR(i));
    LRT = lrtree(StockSpec, RateSpec, LRTimeSpec, Strike);
    PriceLR(i) = optstockbylr(LRT, OptSpec, Strike,ValuationDate, Maturity)

    LRTATM = lrtree(StockSpecATM, RateSpec, LRTimeSpec, Strike);
    PriceLRATM(i) = optstockbylr(LRTATM, OptSpec, Strike,ValuationDate, Mat
end;
```

First scenario: Out of the Money call option

```
% For Cox-Ross-Rubinstein
plot(NumPeriodCRR, PriceCRR);hold on;
plot(NumPeriodCRR, PriceBLS*ones(NbStepCRR,1),'Color',[O 0.9 O], 'linewidth

% For Leisen-Reimer
plot(NumPeriodLR, PriceLR, 'Color',[0.9 O O], 'linewidth', 1.5);

% Concentrate in the area of interest by clipping on the Y axis at 5x the
% LR Price:
YLimDelta = 5*abs(PriceLR(1) - PriceBLS);
set(gca, 'ylim', [PriceBLS-YLimDelta PriceBLS+YLimDelta]);

% Annotate Plot
titleString = sprintf('\nConvergence of CRR and LR models to a BLS Solution
title(titleString)
ylabel('Option Price')
xlabel('Number of Steps')
legend('CRR', 'BLS', 'LR', 'Location', 'NorthEast')
```

Convergence of CRR and LR models to a BLS Solution (OTM)
Strike = 30, Asset Price = 25
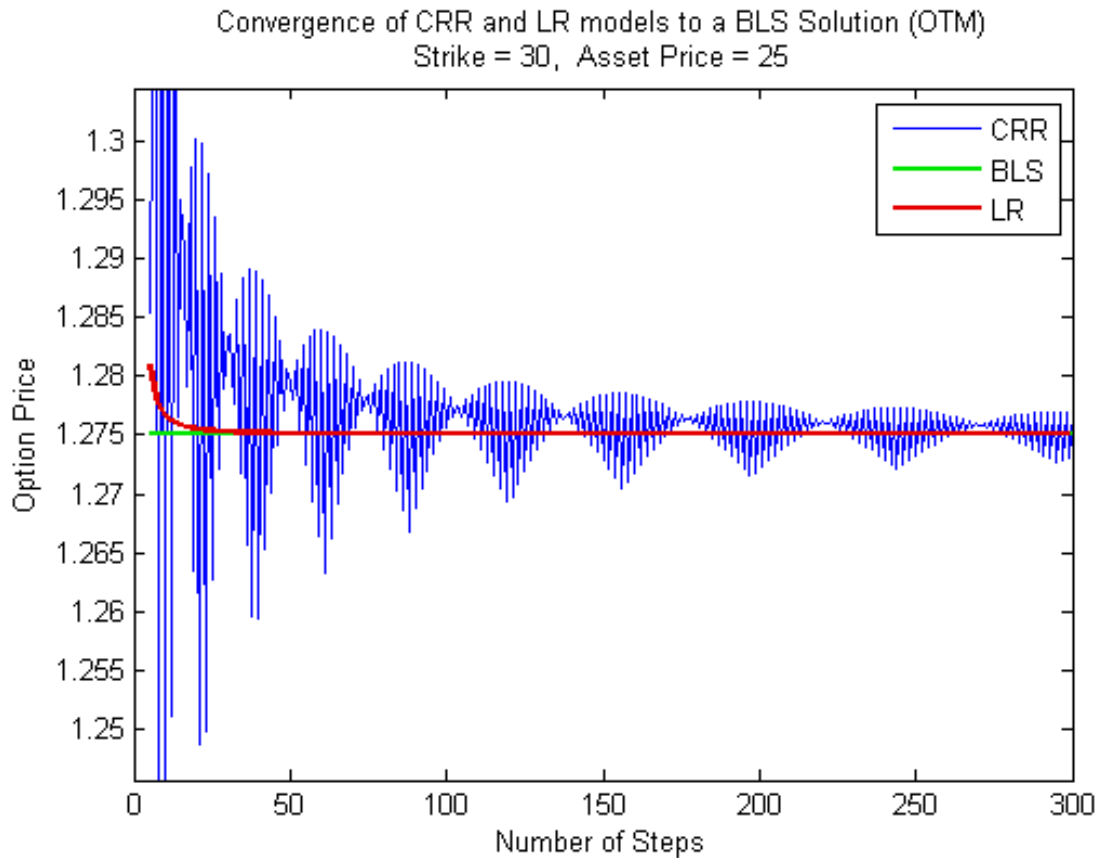
Second scenario: At the Money call option

```
% For Cox-Ross-Rubinstein
figure;
plot(NumPeriodCRR, PriceCRRATM);hold on;
plot(NumPeriodCRR, PriceBLSATM*ones(NbStepCRR,1),'Color',[0 0.9 0], 'linewi

% For Leisen-Reimer
plot(NumPeriodLR, PriceLRATM, 'Color',[0.9 0 0], 'linewidth', 1.5);
```

```
% Concentrate in the area of interest by clipping on the Y axis at 5x the
% LR Price:
YLimDelta = 5*abs(PriceLRATM(1) - PriceBLSATM);
set(gca, 'ylim', [PriceBLSATM-YLimDelta PriceBLSATM+YLimDelta]);

% Annotate Plot
titleString = sprintf('\nConvergence of CRR and LR models to a BLS Solution
title(titleString)
ylabel('Option Price')
xlabel('Number of Steps')
legend('CRR', 'BLS', 'LR', 'Location', 'NorthEast')
```

Convergence of CRR and LR models to a BLS Solution (ATM)
Strike = 30, Asset Price = 30

4
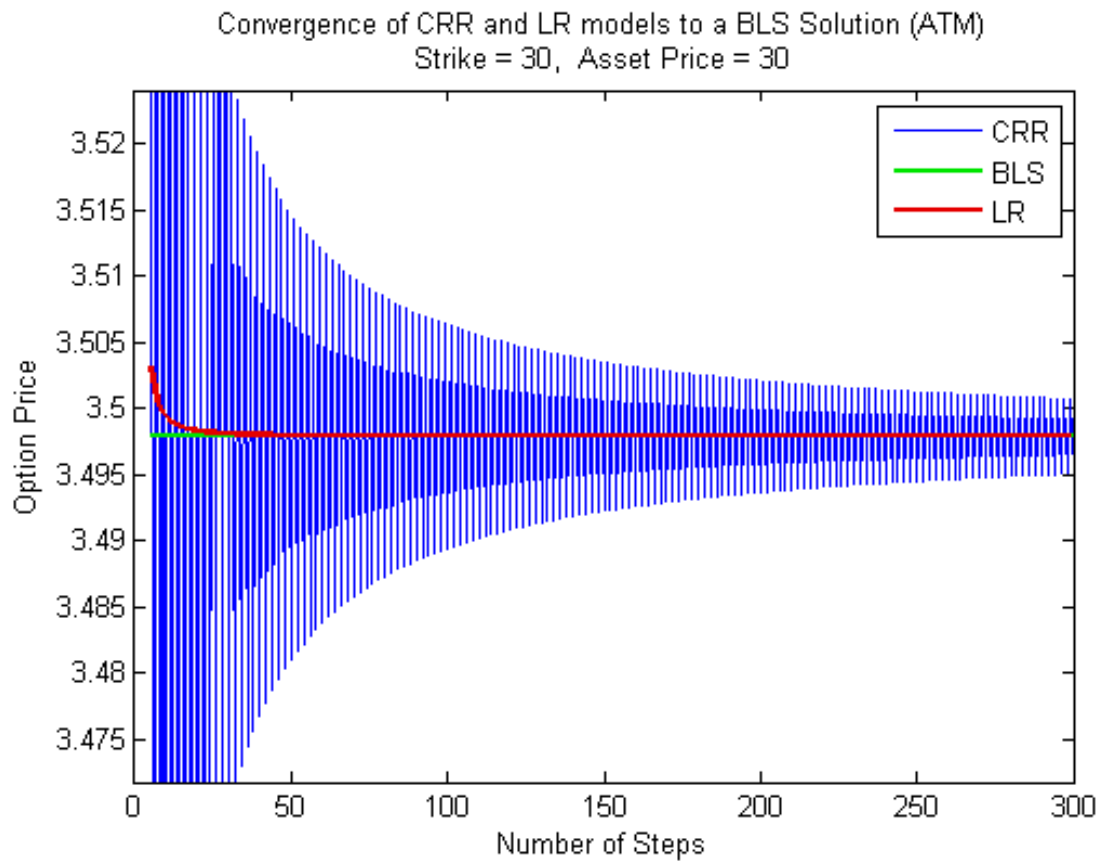
Hedging Portfolios

Hedging

Hedging is an important consideration in modern finance. Whether or not to hedge, how much portfolio insurance is adequate, and how often to rebalance a portfolio are important considerations for traders, portfolio managers, and financial institutions alike.

If there were no transaction costs, financial professionals would prefer to rebalance portfolios continually, thereby minimizing exposure to market movements. However, in practice, the transaction costs associated with frequent portfolio rebalancing may be expensive. Therefore, traders and portfolio managers must carefully assess the cost required to achieve a particular portfolio sensitivity (for example, maintaining delta, gamma, and vega neutrality). Thus, the hedging problem involves the fundamental tradeoff between portfolio insurance and the cost of such insurance coverage.

Hedging Functions

In this section...
"Introduction" on page 4-3
"Hedging with hedgeopt" on page 4-4
"Self-Financing Hedges with hedgeslf" on page 4-12

Introduction

Financial Instruments Toolbox software offers two functions for assessing the fundamental hedging tradeoff, hedgeopt and hedgeslf.

The first function, hedgeopt, addresses the most general hedging problem. It allocates an optimal hedge to satisfy either of two goals:

- Minimize the cost of hedging a portfolio given a set of target sensitivities.

- Minimize portfolio sensitivities for a given set of maximum target costs.

hedgeopt allows investors to modify portfolio allocations among instruments according to either of the goals. The problem is cast as a constrained linear least-squares problem. For additional information about hedgeopt, see "Hedging with hedgeopt" on page 4-4.

The second function, hedgeslf, attempts to allocate a self-financing hedge among a portfolio of instruments. In particular, hedgeslf attempts to maintain a constant portfolio value consistent with reduced portfolio sensitivities (that is, the rebalanced portfolio is hedged against market moves and is closest to being self-financing). If hedgeslf cannot find a self-financing hedge, it rebalances the portfolio to minimize overall portfolio sensitivities. For additional information on hedgeslf, see "Self-Financing Hedges with hedgeslf" on page 4-12.

The examples in this section consider the *delta*, *gamma*, and *vega* sensitivity measures. In this toolbox, when you work with *interest-rate derivatives*, delta is the price sensitivity measure of shifts in the forward yield curve, gamma is the delta sensitivity measure of shifts in the forward yield curve, and vega is the price sensitivity measure of shifts in the volatility process. See bdtsens

or `hjmsens` for details on the computation of sensitivities for interest-rate derivatives.

For *equity exotic options*, the underlying instrument is the stock price instead of the forward yield curve. Consequently, delta now represents the price sensitivity measure of shifts in the stock price, gamma is the delta sensitivity measure of shifts in the stock price, and vega is the price sensitivity measure of shifts in the volatility of the stock. See `crrsens`, `eqpsens`, or `ittsens` for details on the computation of sensitivities for equity derivatives.

For examples showing the computation of sensitivities for interest-rate based derivatives, see "Computing Instrument Sensitivities" on page 2-42. Likewise, for examples showing the computation of sensitivities for equity exotic options, see "Computing Instrument Sensitivities" on page 3-79.

Note The delta, gamma, and vega sensitivities that the toolbox calculates are dollar sensitivities.

Hedging with hedgeopt

Note The numerical results in this section are displayed in the MATLAB bank format. Although the calculations are performed in floating-point double precision, only two decimal places are displayed.

To illustrate the hedging facility, consider the portfolio `HJMInstSet` obtained from the example file `deriv.mat`. The portfolio consists of eight instruments: two bonds, one bond option, one fixed-rate note, one floating-rate note, one cap, one floor, and one swap.

Both hedging functions require some common inputs, including the current portfolio holdings (allocations), and a matrix of instrument sensitivities. To create these inputs, load the example portfolio into memory

```
load deriv.mat;
```

compute price and sensitivities

```
[Delta, Gamma, Vega, Price] = hjmsens(HJMTree, HJMInstSet);

Warning: Not all cash flows are aligned with the tree. Result will
be approximated.
```

and extract the current portfolio holdings.

```
Holdings = instget(HJMInstSet, 'FieldName', 'Quantity');
```

For convenience place the delta, gamma, and vega sensitivity measures into a matrix of sensitivities.

```
Sensitivities = [Delta Gamma Vega];
```

Each row of the `Sensitivities` matrix is associated with a different instrument in the portfolio, and each column with a different sensitivity measure.

To summarize the portfolio information

```
disp([Price  Holdings  Sensitivities])


  98.72       100.00      -272.65      1029.90         0.00
  97.53        50.00      -347.43      1622.69        -0.04
   0.05       -50.00        -8.08       643.40        34.07
  98.72        80.00      -272.65      1029.90         0.00
 100.55         8.00        -1.04         3.31            0
   6.28        30.00       294.97      6852.56        93.69
   0.05        40.00       -47.16      8459.99        93.69
   3.69        10.00      -282.05      1059.68         0.00
```

The first column above is the dollar unit price of each instrument, the second is the holdings of each instrument (the quantity held or the number of contracts), and the third, fourth, and fifth columns are the dollar delta, gamma, and vega sensitivities, respectively.

The current portfolio sensitivities are a weighted average of the instruments in the portfolio.

```
TargetSens  = Holdings' * Sensitivities
```

```
TargetSens =

    -61910.22      788946.21         4852.91
```

Maintaining Existing Allocations

To illustrate using hedgeopt, suppose that you want to maintain your existing portfolio. The first form of hedgeopt minimizes the cost of hedging a portfolio given a set of target sensitivities. If you want to maintain your existing portfolio composition and exposure, you should be able to do so without spending any money. To verify this, set the target sensitivities to the current sensitivities.

```
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, [], [], [], TargetSens)

Sens =

    -61910.22      788946.21         4852.91

Cost =

            0

Quantity' =

        100.00
         50.00
        -50.00
         80.00
          8.00
         30.00
         40.00
         10.00
```

Portfolio composition and sensitivities are unchanged, and the cost associated with doing nothing is zero. The cost is defined as the change in portfolio value. This number cannot be less than zero because the rebalancing cost is defined as a nonnegative number.

If `Value0` and `Value1` represent the portfolio value before and after rebalancing, respectively, the zero cost can also be verified by comparing the portfolio values.

```
Value0 = Holdings' * Price

Value0 =

    23674.62

Value1 = Quantity * Price

Value1 =

    23674.62
```

Partially Hedged Portfolio

Building on the example in "Maintaining Existing Allocations" on page 4-6, suppose you want to know the cost to achieve an overall portfolio dollar sensitivity of [-23000 -3300 3000], while allowing trading only in instruments 2, 3, and 6 (holding the positions of instruments 1, 4, 5, 7, and 8 fixed). To find the cost, first set the target portfolio dollar sensitivity.

```
TargetSens = [-23000 -3300 3000];
```

Then, specify the instruments to be fixed.

```
FixedInd = [1 4 5 7 8];
```

Finally, call `hedgeopt`

```
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, FixedInd, [], [], TargetSens);
```

and again examine the results.

```
Sens =

    -23000.00      -3300.00       3000.00

Cost =
```

```
          19174.02

Quantity' =

          100.00
         -141.03
          137.26
           80.00
            8.00
          -57.96
           40.00
           10.00
```

Recompute `Value1`, the portfolio value after rebalancing.

```
Value1 = Quantity * Price

Value1 =

      4500.60
```

As expected, the cost, $19174.02, is the difference between `Value0` and `Value1`, $23674.62 — $4500.60. Only the positions in instruments 2, 3, and 6 have been changed.

Fully Hedged Portfolio

The example in "Partially Hedged Portfolio" on page 4-7 illustrates a partial hedge, but perhaps the most interesting case involves the cost associated with a fully hedged portfolio (simultaneous delta, gamma, and vega neutrality). In this case, set the target sensitivity to a row vector of 0s and call `hedgeopt` again. The following example uses data from "Hedging with hedgeopt" on page 4-4.

```
TargetSens = [0 0 0];
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price, ...
Holdings, FixedInd, [], [], TargetSens);
```

Examining the outputs reveals that you have obtained a fully hedged portfolio

```
Sens =

        -0.00           -0.00           -0.00
```

but at an expense of over $20,000.

```
Cost =

     23055.90
```

The positions required to achieve a fully hedged portfolio

```
Quantity' =

        100.00
       -182.36
        -19.55
         80.00
          8.00
        -32.97
         40.00
         10.00
```

result in the new portfolio value

```
Value1 = Quantity * Price

Value1 =

     618.72
```

Minimizing Portfolio Sensitivities

The examples in "Fully Hedged Portfolio" on page 4-8 illustrate how to use
hedgeopt to determine the minimum cost of hedging a portfolio given a set
of target sensitivities. In these examples, portfolio target sensitivities are
treated as equality constraints during the optimization process. You tell
hedgeopt what sensitivities you want, and it tells you what it will cost to
get those sensitivities.

A related problem involves minimizing portfolio sensitivities for a given set of maximum target costs. For this goal, the target costs are treated as inequality constraints during the optimization process. You tell hedgeopt the most you are willing spend to insulate your portfolio, and it tells you the smallest portfolio sensitivities you can get for your money.

To illustrate this use of hedgeopt, compute the portfolio dollar sensitivities along the entire cost frontier. From the previous examples, you know that spending nothing replicates the existing portfolio, while spending $23,055.90 completely hedges the portfolio.

Assume, for example, you are willing to spend as much as $50,000, and want to see what portfolio sensitivities will result along the cost frontier. Assume that the same instruments are held fixed, and that the cost frontier is evaluated from $0 to $50,000 at increments of $1000.

```
MaxCost = [0:1000:50000];
```

Now, call hedgeopt.

```
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price, ...
Holdings, FixedInd, [], MaxCost);
```

With this data, you can plot the required hedging cost versus the funds available (the amount you are willing to spend)

```
plot(MaxCost/1000, Cost/1000, 'red'), grid
xlabel('Funds Available for Rebalancing ($1000''s)')
ylabel('Actual Rebalancing Cost ($1000''s)')
title ('Rebalancing Cost Profile')
```
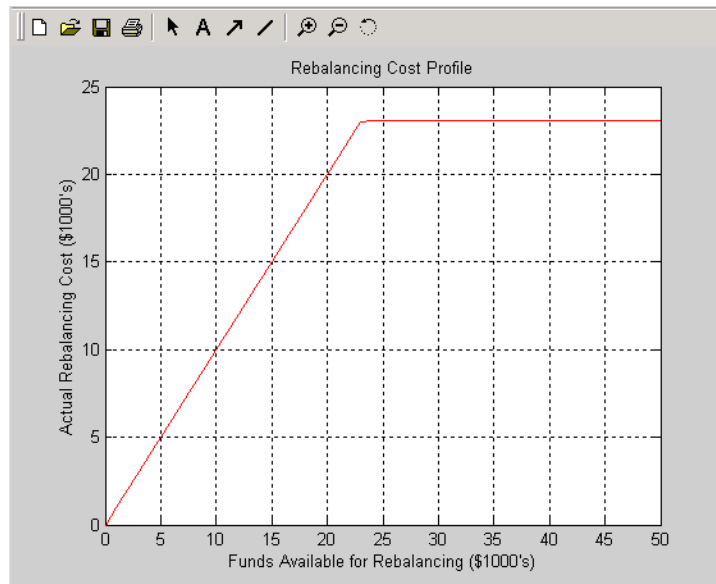
Rebalancing Cost Profile

and the portfolio dollar sensitivities versus the funds available.

```
figure
plot(MaxCost/1000, Sens(:,1), '-red')
hold('on')
plot(MaxCost/1000, Sens(:,2), '-.black')
plot(MaxCost/1000, Sens(:,3), '--blue')
grid
xlabel('Funds Available for Rebalancing ($1000''s)')
ylabel('Delta, Gamma, and Vega Portfolio Dollar Sensitivities')
title ('Portfolio Sensitivities Profile')
legend('Delta', 'Gamma', 'Vega', 0)
```
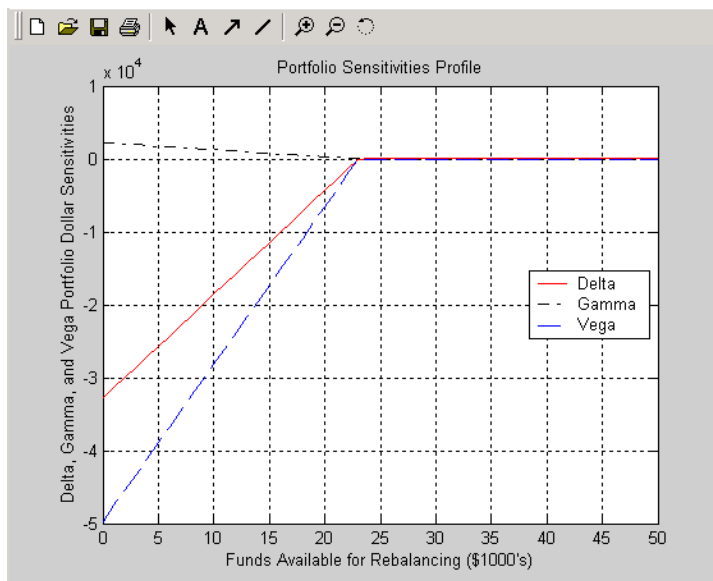
Funds Available for Rebalancing

Self-Financing Hedges with hedgeslf

The figures Rebalancing Cost Profile on page 4-11 and Funds Available for Rebalancing on page 4-12 indicate that there is no benefit because the funds available for hedging exceed $23,055.90, the point of maximum expense required to obtain simultaneous delta, gamma, and vega neutrality. You can also find this point of delta, gamma, and vega neutrality using hedgeslf.

```
[Sens, Value1, Quantity] = hedgeslf(Sensitivities, Price,...
Holdings, FixedInd);

Sens =

        -0.00
        -0.00
        -0.00

Value1 =

      618.72
```

```
Quantity =

         100.00
        -182.36
         -19.55
          80.00
           8.00
         -32.97
          40.00
          10.00
```

Similar to `hedgeopt`, `hedgeslf` returns the portfolio dollar sensitivities and instrument quantities (the rebalanced holdings). However, in contrast, the second output parameter of `hedgeslf` is the value of the rebalanced portfolio, from which you can calculate the rebalancing cost by subtraction.

```
Value0 - Value1

ans =

     23055.90
```

In this example, the portfolio is clearly not self-financing, so `hedgeslf` finds the best possible solution required to obtain zero sensitivities.

There is, in fact, a third calling syntax available for `hedgeopt` directly related to the results shown above for `hedgeslf`. Suppose, instead of directly specifying the funds available for rebalancing (the most money you are willing to spend), you want to simply specify the number of points along the cost frontier. This call to `hedgeopt` samples the cost frontier at 10 equally spaced points between the point of minimum cost (and potentially maximum exposure) and the point of minimum exposure (and maximum cost).

```
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, FixedInd, 10);

Sens =
    -32784.46       2231.83      -49694.33
    -29141.74       1983.85      -44172.74
    -25499.02       1735.87      -38651.14
```

```
        -21856.30        1487.89       -33129.55
        -18213.59        1239.91       -27607.96
        -14570.87         991.93       -22086.37
        -10928.15         743.94       -16564.78
         -7285.43         495.96       -11043.18
         -3642.72         247.98        -5521.59
             0.00          -0.00           0.00

Cost =
             0.00
          2561.77
          5123.53
          7685.30
         10247.07
         12808.83
         15370.60
         17932.37
         20494.14
         23055.90
```

Now plot this data.

```
figure
plot(Cost/1000, Sens(:,1), '-red')
hold('on')
plot(Cost/1000, Sens(:,2), '-.black')
plot(Cost/1000, Sens(:,3), '--blue')
grid
xlabel('Rebalancing Cost ($1000''s)')
ylabel('Delta, Gamma, and Vega Portfolio Dollar Sensitivities')
title ('Portfolio Sensitivities Profile')
legend('Delta', 'Gamma', 'Vega', 0)
```
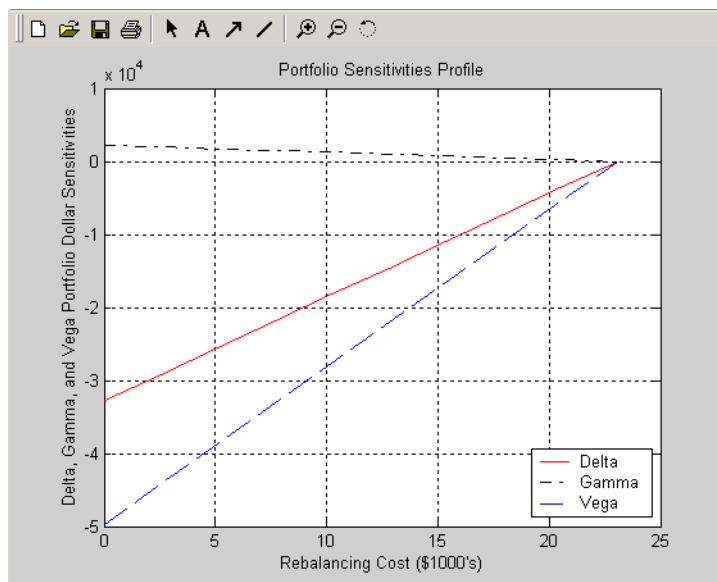
Rebalancing Cost

In this calling form, `hedgeopt` calls `hedgeslf` internally to determine the maximum cost needed to minimize the portfolio sensitivities ($23,055.90), and evenly samples the cost frontier between $0 and $23,055.90.

Note that both `hedgeopt` and `hedgeslf` cast the optimization problem as a constrained linear least squares problem. Depending on the instruments and constraints, neither function is guaranteed to converge to a solution. In some cases, the problem space may be unbounded, and additional instrument equality constraints, or user-specified constraints, may be necessary for convergence. See "Hedging with Constrained Portfolios" on page 4-38 for additional information.

Pricing and Hedging a Portfolio Using the Black-Karasinski Model

This example illustrates how MATLAB can be used to create a portfolio of interest rate derivatives securities, and price it using the Black-Karasinski interest rate model. The example also shows some hedging strategies to minimize exposure to market movements.

Create the Interest Rate Term Structure Based on Reported Data

The structure RateSpec is an interest rate term structure that defines the initial rate specification from which the tree rates are derived. Use the information of annualized zero coupon rates in the table below to populate the RateSpec structure.

```
  From              To            Rate
27 Feb 2007     27 Feb 2008       0.0493
27 Feb 2007     27 Feb 2009       0.0459
27 Feb 2007     27 Feb 2010       0.0450
27 Feb 2007     27 Feb 2012       0.0446
27 Feb 2007     27 Feb 2014       0.0445
27 Feb 2007     27 Feb 2017       0.0450
27 Feb 2007     27 Feb 2027       0.0473
```

This data could be retrieved from the Federal Reserve Statistical Release page by using the Datafeed Toolbox™. In this case, the Datafeed Toolbox™ will connect to FRED and pull back the rates of the following treasury notes.

```
  Terms     Symbol
 =======   ======
    1    =  DGS1
    2    =  DGS2
    3    =  DGS3
    5    =  DGS5
    7    =  DGS7
   10    =  DGS10
   20    =  DGS20
```

Create connection object:

```
  c = fred;
```

Create symbol fetch list:

```
FredNames  = { ...
  'DGS1'; ...      % 1  Year
  'DGS2'; ...      % 2  Year
  'DGS3'; ...      % 3  Year
  'DGS5'; ...      % 5  Year
  'DGS7'; ...      % 7  Year
  'DGS10'; ...     % 10 Year
  'DGS20'};        % 20 Year
```

Define Terms:

```
Terms = [ 1; ...      % 1  Year
          2; ...      % 2  Year
          3; ...      % 3  Year
          5; ...      % 5  Year
          7; ...      % 7  Year
         10; ...      % 10 Year
         20];         % 20 Year
```

Set StartDate to Feb 27, 2007:

```
  StartDate = datenum('Feb-27-2007');

  FredRet = fetch(c,FredNames,StartDate);
```

Set ValuationDate based on StartDate:

```
  ValuationDate = StartDate;

  EndDates = [];

  Rates =[];
```

Create EndDates:

```
  for idx = 1:length(FredRet)

   %Pull the rates associated with Feb 27, 2007. All the Fred Rates come
```

```
        %back as percents
        Rates = [Rates; ...
            FredRet(idx).Data(1,2) / 100];

        %Determine the EndDates by adding the Term to the year of the
        %StartDate
        EndDates = [EndDates; ...
            round(datenum(...
                year(StartDate)+ Terms(idx,1), ...
                month(StartDate),...
                day(StartDate)))];

    end
```

Use the function 'intenvset' to create the RateSpec with the following data:

```
Compounding = 1;
StartDate = '27-Feb-2007';
Rates = [0.0493; 0.0459; 0.0450; 0.0446; 0.0446; 0.0450; 0.0473];
EndDates = {'27-Feb-2008'; '27-Feb-2009';'27-Feb-2010'; '27-Feb-2012';...
            '27-Feb-2014' ; '27-Feb-2017'; '27-Feb-2027'};
ValuationDate = StartDate;

RateSpec = intenvset('Compounding',Compounding,'StartDates', StartDate,...
                     'EndDates', EndDates, 'Rates', Rates,'ValuationDate',


RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: [7x1 double]
             Rates: [7x1 double]
          EndTimes: [7x1 double]
        StartTimes: [7x1 double]
          EndDates: [7x1 double]
        StartDates: 733100
     ValuationDate: 733100
             Basis: 0
       EndMonthRule: 1
```

Specify the Volatility Model

Create the structure VolSpec that specifies the volatility process with the following data.

```
Volatility = [0.011892; 0.01563; 0.02021; 0.02125; 0.02165; 0.02065; 0.0180
Alpha = [0.0001];
VolSpec = bkvolspec(ValuationDate, EndDates, Volatility, EndDates(end), Alp


VolSpec =

            FinObj: 'BKVolSpec'
     ValuationDate: 733100
          VolDates: [7x1 double]
          VolCurve: [7x1 double]
        AlphaCurve: 1.0000e-04
        AlphaDates: 740405
    VolInterpMethod: 'linear'
```

Specify the Time Structure of the Tree

The structure TimeSpec specifies the time structure for an interest rate tree. This structure defines the mapping between the observation times at each level of the tree and the corresponding dates.

```
TimeSpec = bktimespec(ValuationDate, EndDates)


TimeSpec =

          FinObj: 'BKTimeSpec'
    ValuationDate: 733100
         Maturity: [7x1 double]
      Compounding: -1
            Basis: 0
      EndMonthRule: 1
```

Create the BK Tree

Use the previously computed values for RateSpec, VolSpec and TimeSpec
to create the BK tree.

```
BKTree = bktree(VolSpec, RateSpec, TimeSpec)


BKTree =

     FinObj: 'BKFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 1 2 3 5 7 10]
       dObs: [733100 733465 733831 734196 734926 735657 736753]
     CFlowT: {1x7 cell}
      Probs: {1x6 cell}
    Connect: {1x6 cell}
    FwdTree: {1x7 cell}
```

Observe the Interest Rate Tree

Visualize the interest rate evolution along the tree by looking at the output
structure BKTree. The function 'bktree' returns an inverse discount tree,
which you can convert into an interest rate tree with the 'cvtree' function.

```
BKTreeR = cvtree(BKTree)


BKTreeR =

     FinObj: 'BKRateTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 1 2 3 5 7 10]
       dObs: [733100 733465 733831 734196 734926 735657 736753]
```

```
     CFlowT: {1x7 cell}
      Probs: {1x6 cell}
    Connect: {1x6 cell}
   RateTree: {1x7 cell}
```

Look at the upper, middle and lower branch paths of the tree:

```
OldFormat = get(0, 'format');
format short

%Rate at root node:
RateRoot      = trintreepath(BKTreeR, 0)

%Rates along upper branch:
RatePathUp    = trintreepath(BKTreeR, [1 1 1 1 1 1])

%Rates along middle branch:
RatePathMiddle = trintreepath(BKTreeR, [2 2 2 2 2 2])

%Rates along lower branch:
RatePathDown = trintreepath(BKTreeR, [3 3 3 3 3 3])


RateRoot =

    0.0481


RatePathUp =

    0.0481
    0.0425
    0.0446
    0.0478
    0.0510
    0.0554
    0.0619
```

```
RatePathMiddle =

    0.0481
    0.0416
    0.0423
    0.0430
    0.0436
    0.0448
    0.0483


RatePathDown =

    0.0481
    0.0408
    0.0401
    0.0387
    0.0373
    0.0363
    0.0377
```

You can also display a graphical representation of the tree to examine
interactively the rates on the nodes of the tree until maturity. The function
treeviewer displays the structure of the rate tree in the left window. The tree
visualization in the right window is blank, but by selecting Table/Diagram
and clicking on the nodes you can examine the rates along the paths.

```
treeviewer(BKTreeR);
```
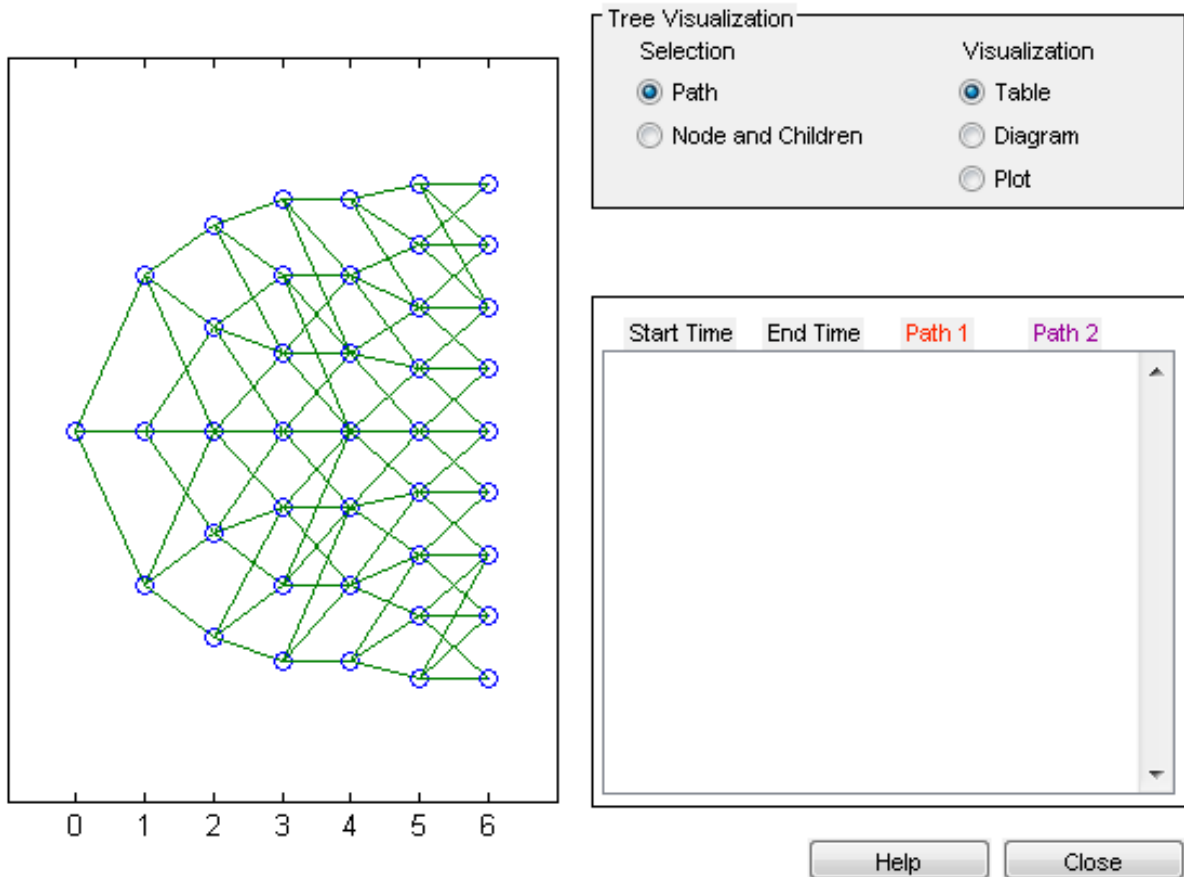
Create an Instrument Portfolio

Create a portfolio consisting of two bonds instruments and an option on the 5% Bond.

```
% Two Bonds
CouponRate = [0.04;0.05];
Settle = '27 Feb 2007';
Maturity = {'27 Feb 2009';'27 Feb 2010'};
Period = 1;
```

```
% American Option on the 5% Bond
OptSpec = {'call'};
Strike = 98;
ExerciseDates = '27 Feb 2010';
AmericanOpt = 1;

InstSet = instadd('Bond', CouponRate, Settle,  Maturity, Period);
InstSet = instadd(InstSet,'OptBond', 2, OptSpec, Strike, ExerciseDates, Ame

% Assign Names and Holdings
Holdings = [10; 15;3];
Names = {'4% Bond'; '5% Bond'; 'Option 98'};

InstSet = instsetfield(InstSet, 'Index',1:3, 'FieldName', {'Quantity'}, 'Da
InstSet = instsetfield(InstSet, 'Index',1:3, 'FieldName', {'Name'}, 'Data',
```

Examine the set of instruments contained in the variable InstSet.

```
instdisp(InstSet)

Index Type CouponRate Settle       Maturity      Period Basis EndMonthRu
1     Bond 0.04       27-Feb-2007  27-Feb-2009   1      0     1
2     Bond 0.05       27-Feb-2007  27-Feb-2010   1      0     1

Index Type     UnderInd OptSpec Strike ExerciseDates AmericanOpt Quantity N
3     OptBond 2         call    98     27-Feb-2010   1           3         O
```

Price the Portfolio Using the BK Model

Calculate the price of each instrument in the portfolio.

```
[Price, PTree] = bkprice(BKTree, InstSet)


Price =

   98.8843
  101.3476
    3.3476
```

```
PTree =

    FinObj: 'BKPriceTree'
     PTree: {1x8 cell}
    AITree: {1x8 cell}
      tObs: [0 1 2 3 5 7 10 20]
   Connect: {1x6 cell}
     Probs: {1x6 cell}
```

The prices in the output vector Price correspond to the prices at observation time zero (tObs = 0), which is defined as the Valuation Date of the interest-rate tree.

In the Price vector, the first element, 98.884, represents the price of the first instrument (4% Bond); the second element, 101.347, represents the price of the second instrument (5% Bond), and 3.347 represents the price of the American call option.

You can also display a graphical representation of the price tree to examine the prices on the nodes of the tree until maturity.

```
treeviewer(PTree,InstSet);
```
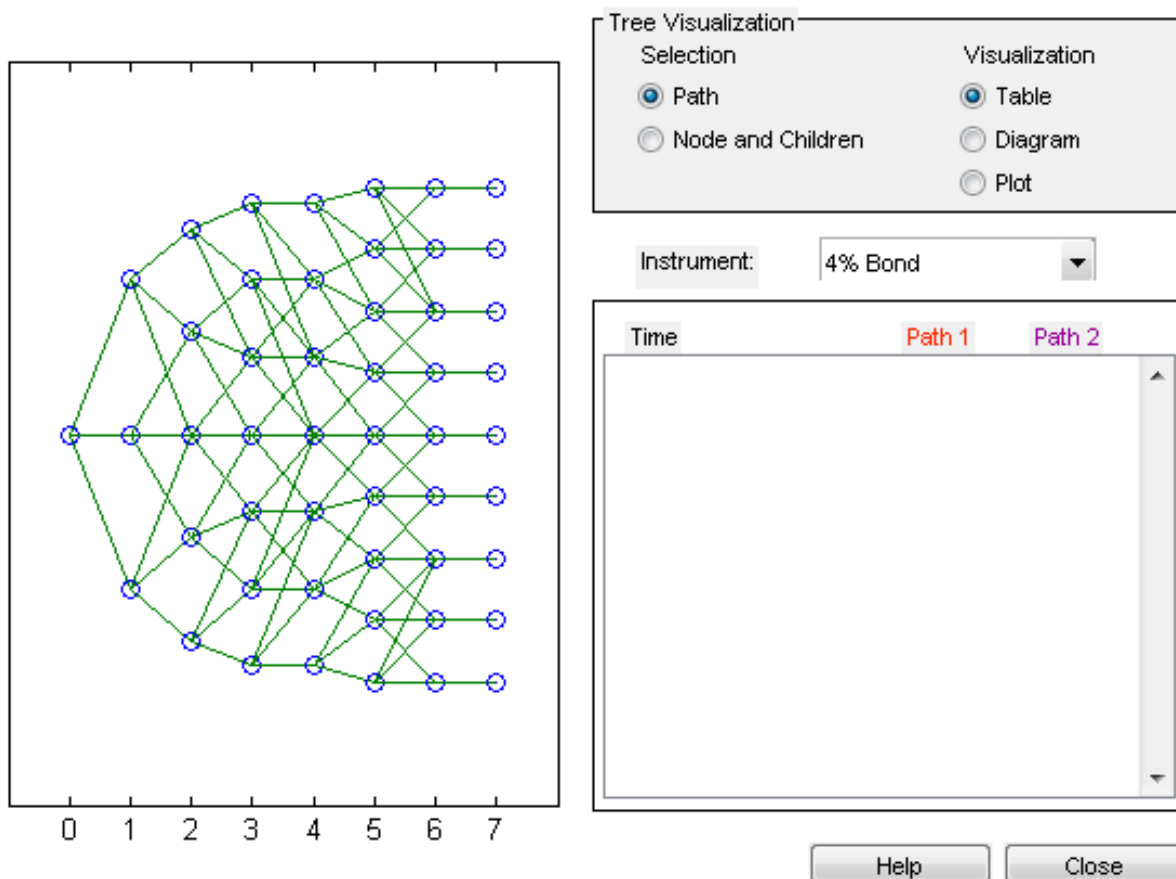
Add More Instruments to the Existing Portfolio

Add instruments to the existing portfolio: cap, floor, floating rate note, vanilla swap and a puttable and callable bond.

```
% Cap
StrikeC =0.035;
InstSet = instadd(InstSet,'Cap', StrikeC, Settle, '27 Feb 2010');

% Floor
StrikeF =0.05;
```

```
InstSet = instadd(InstSet,'Floor', StrikeF, Settle, '27 Feb 2009');

% Floating Rate Note
InstSet = instadd(InstSet,'Float', 30, Settle, '27 Feb 2009');

% Vanilla Swap
 LegRate =[0.04 5];
 InstSet = instadd(InstSet,'Swap', LegRate, Settle, '27 Feb 2010');

% Puttable and Callable Bonds
InstSet = instadd(InstSet,'OptEmBond', CouponRate, Settle, '27 Feb 2010', {
                  Strike, '27 Feb 2010','AmericanOpt', 1, 'Period', 1);

% Process Names and Holdings
Holdings = [15 ;5 ;8; 7; 9; 4];
Names = {'3.5% Cap';'5% Floor';'30BP Float';'4%/5BP Swap'; 'PuttBond'; 'Cal

InstSet = instsetfield(InstSet, 'Index',4:9, 'FieldName', {'Quantity'}, 'Da
InstSet = instsetfield(InstSet, 'Index',4:9, 'FieldName', {'Name'}, 'Data',
```

Examine the set of instruments contained in the variable InstSet.

```
instdisp(InstSet)
```

```
Index Type CouponRate Settle       Maturity      Period Basis EndMonthRu
1     Bond 0.04       27-Feb-2007  27-Feb-2009   1      0     1
2     Bond 0.05       27-Feb-2007  27-Feb-2010   1      0     1

Index Type    UnderInd OptSpec Strike ExerciseDates AmericanOpt Quantity N
3     OptBond 2        call    98     27-Feb-2010   1           3        0

Index Type Strike Settle       Maturity      CapReset Basis Principal Qu
4     Cap  0.035  27-Feb-2007  27-Feb-2010   1        0     100       15

Index Type  Strike Settle       Maturity      FloorReset Basis Principal
5     Floor 0.05   27-Feb-2007  27-Feb-2009   1          0     100

Index Type  Spread Settle       Maturity      FloatReset Basis Principal
6     Float 30     27-Feb-2007  27-Feb-2009   1          0     100
```

```
Index Type LegRate   Settle        Maturity      LegReset Basis Principal
7     Swap [0.04  5] 27-Feb-2007   27-Feb-2010   [NaN]    0     100

Index Type       CouponRate Settle       Maturity      OptSpec Strike Exe
8     OptEmBond  0.04       27-Feb-2007  27-Feb-2010   put     98     27-
9     OptEmBond  0.05       27-Feb-2007  27-Feb-2010   call    98     27-
```

Hedging

The idea behind hedging is to minimize exposure to market movements. As
the underlying changes, the proportions of the instruments forming the
portfolio may need to be adjusted to keep the sensitivities within the desired
range.

Calculate sensitivities using the BK model

```
[Delta, Gamma, Vega, Price] = bksens(BKTree, InstSet);
```

Get the current portfolio holdings.

```
Holdings = instget(InstSet, 'FieldName', 'Quantity');
```

Create a matrix of sensitivities

```
Sensitivities = [Delta Gamma Vega];
```

Each row of the Sensitivities matrix is associated with a different instrument
in the portfolio, and each column with a different sensitivity measure.

Display the information

```
format bank
disp([Price  Holdings  Sensitivities])

      98.88        10.00      -185.46        528.44         0.01
     101.35        15.00      -277.50       1044.93         0.03
       3.35         3.00      -223.50      11845.22         0.03
       2.77        15.00       250.00       2926.95        -0.03
       0.75         5.00      -132.97      11564.31         0.01
     100.56         8.00        -0.80          2.02         0.00
```

```
     -1.53          7.00         -272.07          1027.74          0.03
     98.60          9.00         -168.94         21705.60          0.03
     98.00          4.00          -54.00        -10800.29             0
```

The first column above is the dollar unit price of each instrument, the second is the number of contracts of each instrument, and the third, fourth, and fifth columns are the dollar delta, gamma, and vega sensitivities.

The current portfolio sensitivities are a weighted average of the instruments in the portfolio.

```
TargetSens  = Holdings' * Sensitivities


TargetSens =

     -7249.76     317579.35          0.86
```

Obtain a Neutral Sensitivity Portfolio

Suppose you want to obtain a delta, gamma and vega neutral portfolio. The function hedgeslf finds the reallocation in a portfolio of financial instruments closest to being self-financing (maintaining constant portfolio value).

```
[Sens, Value1, Quantity]= hedgeslf(Sensitivities, Price,Holdings)


Sens =

        -0.00
         0.00
        -0.00


Value1 =

      4637.56
```

```
Quantity =

        10.00
         5.59
        -3.21
         8.63
       -14.68
         9.74
        -4.31
        12.48
         8.80
```

The function hedgeslf returns the portfolio dollar sensitivities (Sens), the value of the rebalanced portfolio (Value1) and the new allocation for each instrument (Quantity). If Value0 and Value1 represent the portfolio value before and after rebalancing, you can verify the cost by comparing the portfolio values.

```
Value0 = Holdings' * Price


Value0 =

      4637.56
```

In this example, the portfolio is fully hedged (simultaneous delta, gamma, and vega neutrality) and self-financing (the values of the portfolio before and after balancing (Value0 and Value1) are the same.

Adding Constraints to Hedge a Portfolio

Suppose that you want to place upper and lower bounds on the individual instruments in the portfolio. Let's say that you want to bound the position of all instruments to within +/- 11 contracts.

Applying these constraints disallows the current positions in the fifth and eighth instruments. All other instruments are currently within the upper/lower bounds.

```
% Specify the lower and upper bounds
LowerBounds = [-11   -11   -11   -11   -11   -11   -11   -11   -11];
UpperBounds = [ 11    11    11    11    11    11    11    11    11];

% Use the function portcons to build the constraints
ConSet = portcons('AssetLims', LowerBounds, UpperBounds);

% Apply the constraints to the portfolio
[Sens, Value, Quantity1] = hedgeslf(Sensitivities, Price, Holdings, [], Con


Sens =

         -0.00
          0.00
          0.00


Value =

       4637.56


Quantity1 =

          2.56
         11.00
         -6.59
          3.60
         -4.67
         11.00
        -11.00
         11.00
         11.00
```

Observe that hedgeslf enforces the bounds on the fifth and eighth instruments, and the portfolio continues to be fully hedged and self-financing.

```
set(0, 'format', OldFormat);
```

Specifying Constraints with ConSet

Introduction

Both `hedgeopt` and `hedgeslf` accept an optional input argument, `ConSet`, that allows you to specify a set of linear inequality constraints for instruments in your portfolio. The examples in this section are brief. For additional information regarding portfolio constraint specifications, refer to "Analyzing Portfolios".

Setting Constraints

For the first example of setting constraints, return to the fully hedged portfolio example that used `hedgeopt` to determine the minimum cost of obtaining simultaneous delta, gamma, and vega neutrality (target sensitivities all 0). Recall that when `hedgeopt` computes the cost of rebalancing a portfolio, the input target sensitivities you specify are treated as equality constraints during the optimization process. The situation is reproduced next for convenience.

```
TargetSens = [0 0 0];
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, FixedInd, [], [], TargetSens);
```

The outputs provide a fully hedged portfolio

```
Sens =
        -0.00          -0.00          -0.00
```

at an expense of over $23,000.

```
Cost =
     23055.90
```

4-33

The positions required to achieve this fully hedged portfolio are

```
Quantity' =

        100.00
       -182.36
        -19.55
         80.00
          8.00
        -32.97
         40.00
         10.00
```

Suppose now that you want to place some upper and lower bounds on the individual instruments in your portfolio. You can specify these constraints, along with a variety of general linear inequality constraints, with Financial Toolbox function portcons.

As an example, assume that, in addition to holding instruments 1, 4, 5, 7, and 8 fixed as before, you want to bound the position of all instruments to within +/- 180 contracts (for each instrument, you cannot short or long more than 180 contracts). Applying these constraints disallows the current position in the second instrument (short 182.36). All other instruments are currently within the upper/lower bounds.

You can generate these constraints by first specifying the lower and upper bounds vectors and then calling portcons.

```
LowerBounds = [-180 -180 -180 -180 -180 -180 -180 -180];
UpperBounds = [ 180  180   180 180   180   180   180   180];
ConSet = portcons('AssetLims', LowerBounds, UpperBounds);
```

To impose these constraints, call hedgeopt with ConSet as the last input.

```
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, FixedInd, [], [], TargetSens, ConSet);
```

Examine the outputs and see that they are all set to NaN, indicating that the problem, given the constraints, is not solvable. Intuitively, the results mean

that you cannot obtain simultaneous delta, gamma, and vega neutrality with these constraints at any price.

To see how close you can get to portfolio neutrality with these constraints, call `hedgeslf`.

```
[Sens, Value1, Quantity] = hedgeslf(Sensitivities, Price,...
Holdings, FixedInd, ConSet);

Sens =

       -352.43
         21.99
       -498.77

Value1 =

        855.10

Quantity =

         100.00
        -180.00
         -37.22
          80.00
           8.00
         -31.86
          40.00
          10.00
```

`hedgeslf` enforces the lower bound for the second instrument, but the sensitivity is far from neutral. The cost to obtain this portfolio is

```
Value0 - Value1

ans =

      22819.52
```

Portfolio Rebalancing

As a final example of user-specified constraints, rebalance the portfolio using the second hedging goal of `hedgeopt`. Assume that you are willing to spend as much as $20,000 to rebalance your portfolio, and you want to know what minimum portfolio sensitivities you can get for your money. In this form, recall that the target cost ($20,000) is treated as an inequality constraint during the optimization process.

For reference, startup `hedgeopt` without any user-specified linear inequality constraints.

```
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, FixedInd, [], 20000);

Sens =

    -4345.36        295.81       -6586.64
Cost =

     20000.00

Quantity' =

        100.00
       -151.86
       -253.47
         80.00
          8.00
        -18.18
         40.00
         10.00
```

This result corresponds to the $20,000 point along the Portfolio Sensitivities Profile shown in the figure Rebalancing Cost on page 4-15.

Assume that, in addition to holding instruments 1, 4, 5, 7, and 8 fixed as before, you want to bound the position of all instruments to within +/- 150 contracts (for each instrument, you cannot short more than 150 contracts and you cannot long more than 150 contracts). These bounds disallow the current

position in the second and third instruments (-151.86 and -253.47). All other instruments are currently within the upper/lower bounds.

As before, you can generate these constraints by first specifying the lower and upper bounds vectors and then calling `portcons`.

```
LowerBounds = [-150 -150 -150 -150 -150 -150 -150 -150];
UpperBounds = [ 150  150  150  150  150  150  150  150];
ConSet = portcons('AssetLims', LowerBounds, UpperBounds);
```

To impose these constraints, again call `hedgeopt` with `ConSet` as the last input.

```
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings,FixedInd, [], 20000, [], ConSet);

Sens =

    -8818.47         434.43        -4010.79

Cost =

    19876.89

Quantity' =

       100.00
      -150.00
      -150.00
        80.00
         8.00
       -28.32
        40.00
        10.00
```

With these constraints, `hedgeopt` enforces the lower bound for the second and third instruments. The cost incurred is $19,876.89.

Hedging with Constrained Portfolios

Overview

Both hedging functions cast the optimization as a constrained linear least-squares problem. (See the function `lsqlin` in the Optimization Toolbox documentation for details.) In particular, `lsqlin` attempts to minimize the constrained linear least squares problem

$$\min_x \frac{1}{2}\|Cx - d\|_2^2 \quad \text{such that} \qquad A \cdot x \le b$$

$$Aeq \cdot x = beq$$
$$lb \le x \le ub$$

where C, A, and Aeq are matrices, and d, b, beq, lb, and ub are vectors. For Financial Instruments Toolbox software, x is a vector of asset holdings (contracts).

Depending on the constraint and the number of assets in the portfolio, a solution to a particular problem may or may not exist. Furthermore, if a solution is found, it may not be unique. For a unique solution to exist, the least squares problem must be sufficiently and appropriately constrained.

Example: Fully Hedged Portfolio

Recall that `hedgeopt` allows you to allocate an optimal hedge by one of two goals:

• Minimize the cost of hedging a portfolio given a set of target sensitivities.

• Minimize portfolio sensitivities for a given set of maximum target costs.

As an example, reproduce the results for the fully hedged portfolio example.

```
TargetSens = [0 0 0];
FixedInd   = [1 4 5 7 8];
[Sens,Cost,Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, FixedInd, [], [], TargetSens);

Sens =

        -0.00          -0.00          -0.00

Cost =

     23055.90

Quantity' =

         98.72
       -182.36
        -19.55
         80.00
          8.00
        -32.97
         40.00
         10.00
```

This example finds a unique solution at a cost of just over $23,000. The matrix C (formed internally by hedgeopt and passed to lsqlin) is the asset Price vector expressed as a row vector.

```
C = Price' = [98.72 97.53 0.05 98.72 100.55 6.28 0.05 3.69]
```

The vector d is the current portfolio value Value0 = 23674.62. The example maintains, as closely as possible, a constant portfolio value subject to the specified constraints.

Additional Constraints

In the absence of any additional constraints, the least squares objective involves a single equation with eight unknowns. This is an under-determined system of equations. Because such systems generally have an infinite number of solutions, you need to specify additional constraints to achieve a solution with practical significance.

The additional constraints can come from two sources:

- User-specified equality constraints
- Target sensitivity equality constraints imposed by `hedgeopt`

The example in "Fully Hedged Portfolio" on page 4-8 specifies five equality constraints associated with holding assets 1, 4, 5, 7, and 8 fixed. This reduces the number of unknowns from eight to three, which is still an under-determined system. However, when combined with the first goal of `hedgeopt`, the equality constraints associated with the target sensitivities in `TargetSens` produce an additional system of three equations with three unknowns. This additional system guarantees that the weighted average of the delta, gamma, and vega of assets 2, 3, and 6, together with the remaining assets held fixed, satisfy the overall portfolio target sensitivity needs in `TargetSens`.

Combining the least-squares objective equation with the three portfolio sensitivity equations provides an overall system of four equations with three unknown asset holdings. This is no longer an under-determined system, and the solution is as shown.

If the assets held fixed are reduced, for example, `FixedInd = [1 4 5 7]`, `hedgeopt` returns a no cost, fully hedged portfolio (`Sens = [0 0 0]` and `Cost = 0`).

If you further reduce `FixedInd` (for example, `[1 4 5]`, `[1 4]`, or even `[ ]`), `hedgeopt` always returns a no cost, fully hedged portfolio. In these cases, insufficient constraints result in an under-determined system. Although `hedgeopt` identifies no cost, fully hedged portfolios, there is nothing unique about them. These portfolios have little practical significance.

Constraints must be *sufficient* and *appropriately defined*. Additional constraints having no effect on the optimization are called *dependent constraints*. As a simple example, assume that parameter Z is constrained such that $Z \leq 1$. Furthermore, assume you somehow add another constraint that effectively restricts $Z \leq 0$. The constraint $Z \leq 1$ now has no effect on the optimization.

Example: Minimize Portfolio Sensitivities

To illustrate using hedgeopt to minimize portfolio sensitivities for a given maximum target cost, specify a target cost of $20,000 and determine the new portfolio sensitivities, holdings, and cost of the rebalanced portfolio.

```
MaxCost = 20000;
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, [1 4 5 7 8], [], MaxCost);

Sens =

     -4345.36        295.81       -6586.64

Cost =

      20000.00

Quantity' =

        100.00
       -151.86
       -253.47
         80.00
          8.00
        -18.18
         40.00
         10.00
```

This example corresponds to the $20,000 point along the cost axis in the figures Rebalancing Cost Profile on page 4-11, Funds Available for Rebalancing on page 4-12, and Rebalancing Cost on page 4-15.

When minimizing sensitivities, the maximum target cost is treated as an inequality constraint; in this case, `MaxCost` is the most you are willing to spend to hedge a portfolio. The least-squares objective matrix `C` is the matrix transpose of the input asset sensitivities

```
C = Sensitivities'
```

a `3`-by-`8` matrix in this example, and `d` is a `3`-by-`1` column vector of zeros, `[0 0 0]'`.

Without any additional constraints, the least-squares objective results in an under-determined system of three equations with eight unknowns. By holding assets 1, 4, 5, 7, and 8 fixed, you reduce the number of unknowns from eight to three. Now, with a system of three equations with three unknowns, `hedgeopt` finds the solution shown.

Example: Under-Determined System

Reducing the number of assets held fixed creates an under-determined system with meaningless solutions. For example, see what happens with only four assets constrained.

```
FixedInd = [1 4 5 7];
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, FixedInd, [], MaxCost);

Sens =

        -0.00            -0.00            -0.00

Cost =

     20000.00

Quantity' =

       100.00
      -149.31
       -14.91
        80.00
```

```
         8.00
       -34.64
        40.00
       -32.60
```

You have spent $20,000 (all the funds available for rebalancing) to achieve a fully hedged portfolio.

With an increase in available funds to $50,000, you still spend all available funds to get another fully hedged portfolio.

```
MaxCost  = 50000;
[Sens, Cost, Quantity] = hedgeopt(Sensitivities, Price,...
Holdings, FixedInd, [],MaxCost);

Sens =

        -0.00          0.00          0.00
Cost =

     50000.00

Quantity' =

       100.00
      -473.78
       -60.51
        80.00
         8.00
       -18.20
        40.00
       385.60
```

All solutions to an under-determined system are meaningless. You buy and sell various assets to obtain zero sensitivities, spending all available funds every time. If you reduce the number of fixed assets any further, this problem is insufficiently constrained, and you find no solution (the outputs are all NaN).

Note also that no solution exists whenever constraints are *inconsistent*. Inconsistent constraints create an infeasible solution space; the outputs are all NaN.

Example: Portfolio Constraints with hedgeslf

The other hedging function, hedgeslf, attempts to minimize portfolio sensitivities such that the rebalanced portfolio maintains a constant value (the rebalanced portfolio is hedged against market moves and is closest to being self-financing). If a self-financing hedge is not found, hedgeslf tries to rebalance a portfolio to minimize sensitivities.

From a least-squares systems approach, hedgeslf first attempts to minimize cost in the same way that hedgeopt does. If it cannot solve this problem (a no cost, self-financing hedge is not possible), hedgeslf proceeds to minimize sensitivities like hedgeopt. Thus, the discussion of constraints for hedgeopt is directly applicable to hedgeslf as well.

To illustrate this hedging facility using equity exotic options, consider the portfolio CRRInstSet obtained from the example MAT-file deriv.mat. The portfolio consists of eight option instruments: two stock options, one barrier, one compound, two lookback, and two Asian.

The hedging functions require inputs that include the current portfolio holdings (allocations) and a matrix of instrument sensitivities. To create these inputs, start by loading the example portfolio into memory

```
load deriv.mat;
```

Next, compute the prices and sensitivities of the instruments in this portfolio.

```
[Delta, Gamma, Vega, Price] = crrsens(CRRTree, CRRInstSet);
```

Extract the current portfolio holdings (the quantity held or the number of contracts).

```
Holdings = instget(CRRInstSet, 'FieldName', 'Quantity');
```

For convenience place the delta, gamma, and vega sensitivity measures into a matrix of sensitivities.

```
Sensitivities = [Delta Gamma Vega];
```

Each row of the `Sensitivities` matrix is associated with a different instrument in the portfolio and each column with a different sensitivity measure.

```
disp([Price  Holdings  Sensitivities])
```

```
        8.29           10.00            0.59            0.04          53.45
        2.50            5.00           -0.31            0.03          67.00
       12.13            1.00            0.69            0.03          67.00
        3.32            3.00           -0.12           -0.01         -98.08
        7.60            7.00           -0.40       -45926.32          88.18
       11.78            9.00           -0.42      -112143.15         119.19
        4.18            4.00            0.60        45926.32          49.21
        3.42            6.00            0.82       112143.15          41.71
```

The first column contains the dollar unit price of each instrument, the second contains the holdings of each instrument, and the third, fourth, and fifth columns contain the delta, gamma, and vega dollar sensitivities, respectively.

Suppose that you want to obtain a delta, gamma and vega neutral portfolio using `hedgeslf`.

```
[Sens, Value1, Quantity]= hedgeslf(Sensitivities, Price, ...
Holdings)

Sens =

        0.00
       -0.00
        0.00

Value1 =

      313.93

Quantity =

       10.00
        7.64
```

```
                    -1.56
                    26.13
                     9.94
                     3.73
                    -0.75
                     8.11
```

`hedgeslf` returns the portfolio dollar sensitivities (`Sens`), the value of the rebalanced portfolio (`Value1`) and the new allocation for each instrument (`Quantity`).

If `Value0` and `Value1` represent the portfolio value before and after rebalancing, respectively, you can verify the cost by comparing the portfolio values.

```
Value0= Holdings' * Price

Value0 =

        313.93
```

In this example, the portfolio is fully hedged (simultaneous delta, gamma, and vega neutrality) and self-financing (the values of the portfolio before and after balancing (`Value0` and `Value1`) are the same.

Suppose now that you want to place some upper and lower bounds on the individual instruments in your portfolio. By using Financial Toolbox function `portcons`, you can specify these constraints, along with a variety of general linear inequality constraints.

As an example, assume that, in addition to holding instrument 1 fixed as before, you want to bound the position of all instruments to within +/- 20 contracts (for each instrument, you cannot short or long more than 20 contracts). Applying these constraints disallows the current position in the fourth instrument (long 26.13). All other instruments are currently within the upper/lower bounds.

You can generate these constraints by first specifying the lower and upper bounds vectors and then calling `portcons`.

```
LowerBounds = [-20  -20  -20  -20  -20  -20  -20  -20];
UpperBounds = [20  20  20  20  20  20  20  20];
ConSet = portcons('AssetLims', LowerBounds, UpperBounds);
```

To impose these constraints, call hedgeslf with ConSet as the last input.

```
[Sens, Cost, Quantity1] = hedgeslf(Sensitivities, Price, ...
Holdings, 1, ConSet)

Sens =

        -0.00
         0.00
         0.00

Cost =

       313.93

Quantity1 =

        10.00
         5.28
        10.98
        20.00
        20.00
        -6.99
       -20.00
         9.39
```

Observe that hedgeslf enforces the upper bound on the fourth instrument, and the portfolio continues to be fully hedged and self-financing.

5

Mortgage-Backed Securities

What Are Mortgage-Backed Securities?

Mortgage-backed securities (MBSs) are a type of investment that represents ownership in a group of mortgages. Principal and interest from the individual mortgages are used to pay principal and interest on the MBS.

Ownership in a group of mortgages is typically represented by a *pass-through certificate* (PC). Most pass-through certificates are issued by the Government National Mortgage Agency, a branch of the United States government, or by one of two private corporations: Fannie Mae or Freddie Mac. With these certificates, homeowners' payments pass from the originating bank through the issuing agency to holders of the certificates. These agencies also frequently guarantee that the certificate holder receives timely payment of principal and interest from the PCs.

Fixed-Rate Mortgage Pool

Introduction

Financial Instruments Toolbox software supports calculations involved with generic fixed-rate mortgage pools and balloon mortgages. Pass-through certificates typically have embedded call options in the form of prepayment. Prepayment is an excess payment applied to the principal of a PC. These accelerated payments reduce the effective life of a PC.

The toolbox comes with a standard Bond Market Association (PSA) prepayment model and can generate multiples of standard prepayment speeds. The Public Securities Association provides a set of uniform practices for calculating the characteristics of mortgage-backed securities when there is an assumed prepayment function.

Alternatively, aside from the standard PSA implementation in this toolbox, you can supply your own projected prepayment vectors. At this time, however, custom prepayment functionality that incorporates pool-specific information and interest rate forecasts are not available in this toolbox. If you plan to use custom prepayment vectors in your calculations, you presumably already own such a suite in MATLAB.

Inputs to Functions

Because of the generic, all-purpose nature of the toolbox pass-through functions, you can fine-tune them to conform to a particular mortgage. Most functions require at least this set of inputs:

- Gross coupon rate
- Settlement date
- Issue (effective) date
- Maturity date

Typical optional inputs include standard prepayment speed (or customized vector), net coupon rate (if different from gross coupon rate), and payment delay in number of days.

All calculations are based on expected payment dates and actual cash flow to the investor. For example, when GrossRate and CouponRate differ as inputs to mbsdurp, the function returns a modified duration based on CouponRate. (A notable exception is mbspassthrough, which returns interest quantities based on the GrossRate.)

Generating Prepayment Vectors

You can generate PSA multiple prepayment vectors quickly. To generate prepayment vectors of 100 and 200 PSA, type

```
PSASpeed = [100, 200];
[CPR, SMM] = psaspeed2rate(PSASpeed);
```

This function computes two prepayment values: conditional prepayment rate (CPR) and single monthly mortality (SMM) rate. CPR is the percentage of outstanding principal prepaid in 1 year. SMM is the percentage of outstanding principal prepaid in 1 month. In other words, CPR is an annual version of SMM.

Since the entire 360-by-2 array is too long to show in this document, observe the SMM (100 and 200 PSA) plots, spaced one month apart, instead.
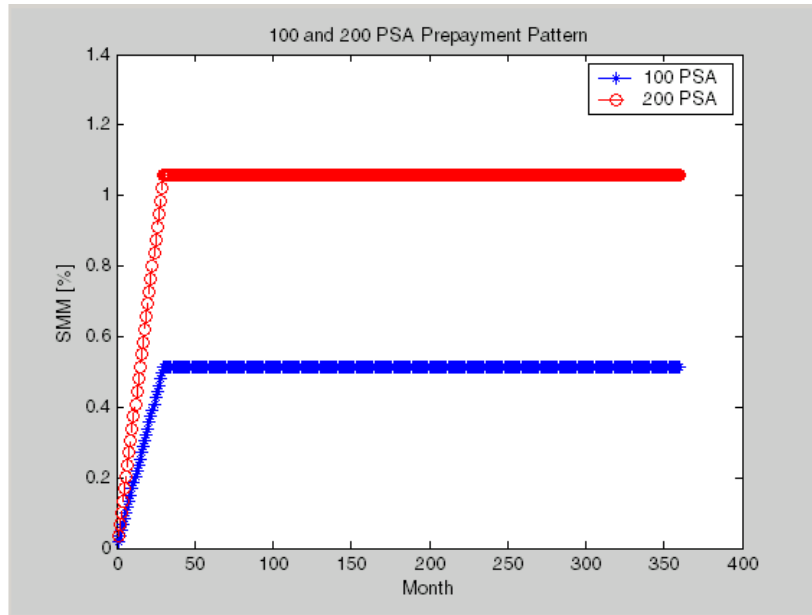
Prepayment assumptions form the basis upon which far more comprehensive MBS calculations are based. As an illustration observe the following example, which shows the use of the function mbscfamounts to generate cash flows and timings based on a set of standard prepayments.

Consider three mortgage pools that were sold on the issue date (which starts unamortized). The first two pools "balloon out" in 60 months, and the third is regularly amortized to the end. The prepayment speeds are assumed to be 100, 200, and 200 PSA, respectively.

```
Settle      = [datenum('1-Feb-2000');
               datenum('1-Feb-2000');
               datenum('1-Feb-2000')];

Maturity    = [datenum('1-Feb-2030')];

IssueDate   = datenum('1-Feb-2000');
GrossRate   = 0.08125;
CouponRate  = 0.075;
Delay = 14;
```

```
PSASpeed = [100, 200];
[CPR, SMM] = psaspeed2rate(PSASpeed);

PrepayMatrix = ones(360,3);
PrepayMatrix(1:60,1:2) = SMM(1:60,1:2);
PrepayMatrix(:,3) = SMM(:,2);

[CFlowAmounts, CFlowDates, TFactors, Factors] = ...
mbscfamounts(Settle, Maturity, IssueDate, GrossRate, ...
CouponRate, Delay, [], PrepayMatrix);
```

The fourth output argument, Factors, indicates the fraction of the balance still outstanding at the beginning of each month. A snapshot of this argument in the MATLAB Variables editor illustrates the 60-month life of the first two of the mortgages with balloon payments and the continuation of the third mortgage until the end (360 months).

You can readily see that mbscfamounts is the building block of most fixed rate and balloon pool cash flows.

Mortgage Prepayments

Prepayment is beneficial to the pass-through owner when a mortgage pool has been purchased at discount. The next example compares mortgage yields (compounded monthly) versus the purchase clean price with constant prepayment speed. The example illustrates that when you have purchased a pool at a discount, prepayment generates a higher yield with decreasing purchase price.

```
Price = [85; 90; 95];
Settle = datenum('15-Apr-2002');
Maturity = datenum('1 Jan 2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;
```

Compute the mortgage and bond-equivalent yields.

```
[MYield, BEMBSYield] = mbsyield(Price, Settle, Maturity, ...
IssueDate, GrossRate, CouponRate, Delay, Speed)

MYield =

    0.1018
    0.0918
    0.0828


BEMBSYield =

    0.1040
    0.0936
    0.0842
```

If for this same pool of mortgages, there was no prepayment (Speed = 0), the yields would decline to

```
MYield =

    0.0926
    0.0861
    0.0802


BEMBSYield =

    0.0944
    0.0877
    0.0815
```

Likewise, if the rate of prepayment doubled (`Speed = 200`), the yields would increase to

```
MYield =

    0.1124
    0.0984
    0.0858

BEMBSYield =

    0.1151
    0.1004
    0.0873
```

For the same prepayment vector, deeper discount pools earn higher yields. For more information, see `mbsprice` and `mbsyield`.

Risk Measurement

Financial Instruments Toolbox software provides the most basic risk measures of a pool portfolio:

- Modified duration
- Convexity
- Average life of pool

Consider the following example, which calculates the Macaulay and modified durations given the price of a mortgage pool.

```
Price = [95; 100; 105];
Settle = datenum('15-Apr-2002');
Maturity = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;
```

```
[YearDuration, ModDuration] = mbsdurp(Price, Settle, ...
Maturity, IssueDate, GrossRate, CouponRate, Delay, Speed)

YearDuration =

    6.1341
    6.3882
    6.6339

ModDuration =

    5.8863
    6.1552
    6.4159
```

Using Financial Instruments Toolbox functions, you can obtain modified duration and convexity from either price or yield, as long as you specify a prepayment vector or an assumed prepayment speed. The toolbox risk-measurement functions (mbsdurp, mbsdury, mbsconvp, mbsconvy, and mbswal) adhere to the guidelines listed in the *PSA Uniform Practices* manual.

Mortgage Pool Valuation

For accurate valuation of a mortgage pool, you must generate interest rate paths and use them with mortgage pool characteristics to properly value the pool. A widely used methodology is the option-adjusted spread (OAS). OAS measures the yield spread that is not directly attributable to the characteristics of a fixed-income investment.

Calculating OAS

Prepayment alters the cash flows of an otherwise regularly amortizing mortgage pool. A comprehensive option-adjusted spread calculation typically begins with the generation of a set of paths of spot rates to predict prepayment. A path is collection of i spot-rate paths, with corresponding j cash flows on each of those paths.

The effect of the OAS on pool pricing is shown mathematically in the following equation, where K is the option-adjusted spread.

$$PoolPrice = \frac{1}{NumberofPaths} \times \sum_{i}^{NumberofPaths} \sum_{j}^{CF_{ij}} \frac{CF_{ij}}{(1 + zerorates_{ij} + K)^{T_{ij}}}$$

Calculating Effective Duration

Alternatively, if you are more interested in the sensitivity of a mortgage pool to interest rate changes, use effective duration, which is a more appropriate measure. Effective duration is defined mathematically with the following equation.

$$Effective\ Duration = \frac{P(y + \Delta y) - P(y - \Delta y)}{2P(y)\Delta y}$$

Calculating Market Price

The toolbox has all the components required to calculate OAS and effective duration if you supply prepayment vectors or assumptions. For OAS, given a prepayment vector, you can generate a set of cash flows with `mbscfamounts`. Discounting these cash flows with the reference curve and then adding OAS produces the market price. See "Computing Option-Adjusted Spread" on page 5-10 for a discussion on the computation of option-adjusted spread.

Effective duration is a more difficult issue. While modified duration changes the discounting process (by changing the yield used to discount cash flows), effective duration must account for the change in cash flow because of the change in yield. A possible solution is to recompute prices using `mbsprice` for a small change in yield, in both the upwards and downwards directions. In this case, you must recompute the prepayment input. Internally, this alters the cash flows of the mortgage pool. Assuming that the OAS stays constant in all yield environments, you can apply a set of discounting factors to the cash flows in up and down yield environments to find the effective duration.

Computing Option-Adjusted Spread

The option-adjusted spread (OAS) is an amount of extra interest added above (or below if negative) the reference zero curve. To compute the OAS, you must provide the zero curve as an extra input. You can specify the zero curve in any intervals and with any compounding method. (To minimize any error due to interpolation, keep the intervals as regular and frequent as possible.)

You must supply a prepayment vector or specify a speed corresponding to a standard PSA prepayment vector.

One way to compute the appropriate zero curve for an agency is to look at its bond yields and bootstrap them from the shortest maturity onwards. You can do this with Financial Toolbox functions `zbtprice` and `zbtyield`.

The following example shows how to calculate an appropriate zero curve followed by computation of the pool's OAS. This example calculates the OAS of a 30-year fixed rate mortgage with about a 28-year weighted average maturity left, given an assumption of 0, 50, and 100 PSA prepayment speeds.

Create curve for `zerorates`.

```
Bonds = [datenum('11/21/2002')   0        100   0   2   1;
         datenum('02/20/2003')   0        100   0   2   1;
         datenum('07/31/2004')   0.03     100   2   3   1;
         datenum('08/15/2007')   0.035    100   2   3   1;
         datenum('08/15/2012')   0.04875  100   2   3   1;
         datenum('02/15/2031')   0.05375  100   2   3   1];

Yields = [0.0162;
          0.0163;
          0.0211;
          0.0328;
          0.0420;
          0.0501];
```

Since the above is Treasury data and not selected agency data, a term structure of spread is assumed. In this example, the spread declines proportionally from a maximum of 250 basis points at the shortest maturity.

```
Yields = Yields + 0.025 * (1./[1:6]');
```

Get parameters from `Bonds` matrix.

```
Settle = datenum('20-Aug-2002');
Maturity = Bonds(:,1);
CouponRate = Bonds(:,2);
Face = Bonds(:,3);
Period = Bonds(:,4);
```

```
Basis = Bonds(:,5);
EndMonthRule = Bonds(:,6);

[Prices, AccruedInterest] = bndprice(Yields, CouponRate, ...
Settle, Maturity, Period, Basis, EndMonthRule, [], [], [], [], ...
Face);
```

Use zbtprice to solve for zero rates.

```
[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);
ZeroCompounding = 2*ones(size(ZeroRatesP));
ZeroMatrix = [CurveDatesP, ZeroRatesP, ZeroCompounding];
```

Use output from zbtprice to calculate the OAS.

```
Price = 95;
Settle = datenum('20-Aug-2002');
Maturity = datenum('2-Jan-2030');
IssueDate = datenum('2-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Interpolation = 1;
PrepaySpeed = [0; 50; 100];

OAS = mbsprice2oas(ZeroMatrix, Price, Settle, Maturity, ...
IssueDate, GrossRate, CouponRate, Delay, Interpolation, ...
PrepaySpeed)

OAS =

    26.0502
    28.6348
    31.2222
```

This example shows that one cash flow set is being discounted and solved for its OAS, as contrasted with the NumberOfPaths set of cash flows as shown in "Mortgage Pool Valuation" on page 5-9. Averaging the sets of cash flows resulting from all simulations into one average cash flow vector and solving

for the OAS, discounts the averaged cash flows to have a present value of today's (average) price.

While this example uses the mortgage pool price (`mbsprice2oas`) to determine the OAS, you can also use yield to resolve it (`mbsyield2oas`). Also, there are reverse OAS functions that return prices and yields given OAS (`mbsoas2price` and `mbsoas2yield`).

The example also restates earlier examples that show discount securities benefit from higher level of prepayment, keeping everything else unchanged. The relation is reversed for premium securities.

Prepayments with Fewer Than 360 Months Remaining

When fewer than 360 months remain in the pool, the applicable PSA prepayment vector is "seasoned" by the pool's age. (Elements in the 360-element prepayment vector that represent past payments are skipped. For example, on a 30-year mortgage that is 10 months old, only the final 350 prepayments are applied.)

Assume, for example, that you have two 30-year loans, one new and another 10 months old. Both have the same PSA speed of 100 and prepay using the vectors plotted below.
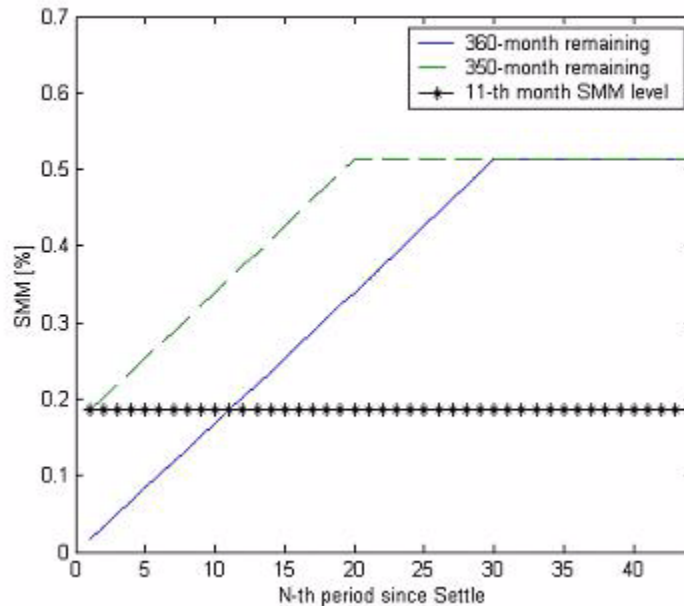
Still within the scope of relative valuation, you could also solve for the percentage of the standard PSA prepayment vector given the pool's arbitrary, user-supplied prepayment vector, such that the PSA speed gives the same Macaulay duration as the user-supplied prepayment vector.

If you supply a custom prepayment vector, you must account for the number of months remaining.

```
Price = 101;
Settle = datenum('1-Jan-2001');
Maturity = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
PrepayMatrix = 0.005*ones(348,1);
CouponRate   = 0.075;
Delay = 14;

ImpliedSpeed = mbsprice2speed(Price, Settle, Maturity, ...
IssueDate, GrossRate, PrepayMatrix, CouponRate, Delay)
```

```
ImpliedSpeed =

  104.2543
```

Examine the prepayment input. The remaining 29 years require 348 monthly elements in the prepayment vector. Suppose then, keeping everything the same, you change `Settle` to February 14, 2003.

```
Settle = datenum('14-Feb-2003');
```

You can use `cpncount` to count all incoming coupons received after `Settle` by invoking

```
NumCouponsRemaining = cpncount(Settle, Maturity, 12, 1, [], ...
IssueDate)

NumCouponsRemaining =
323
```

The input `12` defines the monthly payment frequency, `1` defines the 30/360 basis, and `IssueDate` defines aging and determination-of-holder date. Thus, you must supply a 323-element vector to account for a prepayment corresponding to each monthly payment.

Pools with Different Numbers of Coupons Remaining
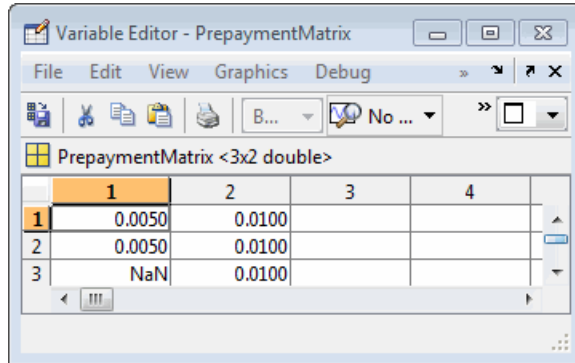
Suppose one pool has two remaining coupons, and the other has three. MATLAB software expects the prepayment matrix to be in the following format:

```
V11        V21
V12        V22
NaN        V23
```

V_{ij} denotes the single monthly mortality (SMM) rate for pool `i` during the `j`th coupon period since `Settle`.

The use of `NaN` to pad the prepayment matrix is necessary because MATLAB cannot concatenate vectors of different lengths into a matrix. Also, it can serve as an error check against any unintended operation (any MATLAB operation that would return `NaN`).

For example, assume that the 2 month pool has a constant SMM of 0.5% and the 3 month has a constant SMM of 1% in every period. The prepayment matrix you would create is depicted below.

Create this input in whatever manner is best for you.

Summary of Prepayment Data Vector Representation

- When you specify a PSA prepayment speed, MATLAB "seasons" the pool according to its age.

- When you specify your own prepayment matrix, identify the maximum number of coupons remaining using cpncount. Then supply the matrix elements up to the point when cash flow ceases to exist.

- When different length pools must exist in the same matrix, pad the shorter one(s) with NaN. Each column of the prepayment matrix corresponds to a specific pool.

Prepayment Modeling with a Two Factor Hull White Model and a LIBOR Market Model

This example shows how to model prepayment in MATLAB using functionality from the Financial Instruments Toolbox™. Specifically, a variation of the Richard and Roll prepayment model is implemented using a two factor Hull-White interest-rate model and a LIBOR Market Model to simulate future interest-rate paths. A mortgage-backed security is priced with both the custom and default prepayment models.

Introduction

Prepayment modeling is crucial to the analysis of mortgage-backed securities (MBS). Prepayments by individual mortgage holders affect both the amount and timing of cash flows -- and for collateralized mortgage obligations (e.g.: interest-only securities), prepayment can greatly affect the value of the securities.

PSA Model

The most basic prepayment model is the Public Securities Association (PSA) model, which assumes a ramp-up phase and then a constant conditional prepayment rate (CPR). The PSA model can be generated in MATLAB using the Financial Instruments Toolbox function PSASPEED2RATE

```
G2PP_CPR = psaspeed2rate([100 200]);
figure
plot(G2PP_CPR)
title('100 and 200 PSA Prepayment Speeds')
xlabel('Months')
ylabel('CPR')
ylim([0 .14])
legend({'100 PSA','200 PSA'}, 'Location', 'Best')
```

100 and 200 PSA Prepayment Speeds
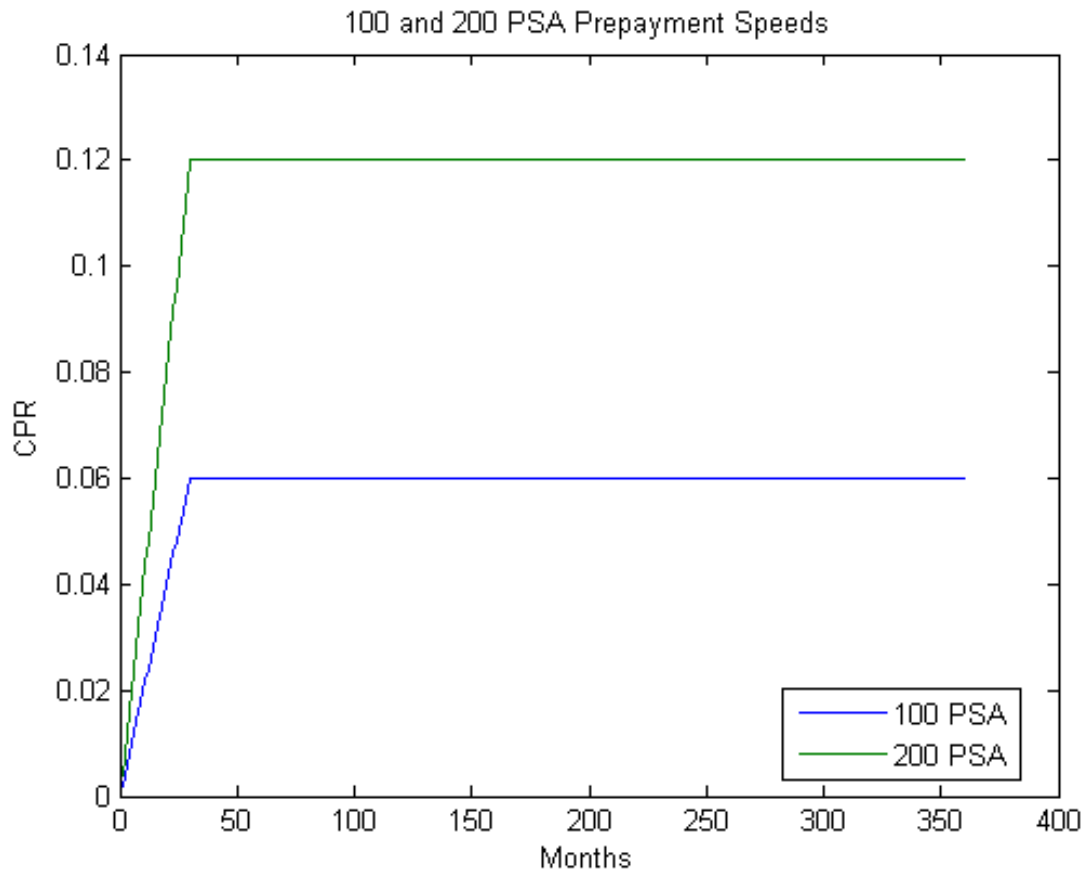
Mortgage-Backed Security

The MBS analyzed in this example matures in 2020 and has the properties outlined in this section. Cash flows are generated for PSA prepayment speeds simply by entering the PSA speed as an input argument.

```
% Parameters for MBS passthrough to be priced
Settle = datenum('15-Dec-2007');
Maturity = datenum('15-Dec-2020');
IssueDate = datenum('15-Dec-2000');
GrossRate = .0475;
```

```
CouponRate = .045;
Delay = 14;
Period = 12;
Basis = 4;

% Generate cash flows and dates for baseline case using 100 PSA
[CFlowAmounts, CFlowDates] = mbscfamounts(Settle,Maturity, IssueDate,...
    GrossRate, CouponRate, Delay,100);
CFlowTimes = yearfrac(Settle,CFlowDates);
NumCouponsRemaining = cpncount(Settle, Maturity, Period,Basis, 1, IssueDate
```

Richard and Roll Model

While prepayment modeling often involves quite complex and sophisticated modeling, often at the loan level, this example will use a slightly modified approach based on the model proposed by Richard and Roll in [6].

The Richard and Roll prepayment model involves the following factors:

- Refinancing incentive
- Seasonality (month of the year)
- Seasoning or age of the mortgage
- Burnout

Richard and Roll propose a multiplicative model of the following:

$$CPR = RefiIncentive * SeasoningMultiplier * SeasonalityMultiplier * BurnoutMultiplier$$

For the custom model in this example, the Burnout Multiplier, which describes the tendency of prepayment to slow when a significant number of homeowners have already refinanced, is ignored and the first three terms are used.

The refinancing incentive is a function of the ratio of the coupon-rate of the mortgage to the available mortgage rate at that particular point in time. For example, the Office of Thrift Supervision (OTS) proposes the following model:

$$Refi = .2406 - .1389 * arctan(5.952 * (1.089 - \frac{CouponRate}{MortgageRate}))$$

The refinancing incentive requires a simulation of future interest rates. This will be discussed later in this example.

```
C_M = .1:.1:2;
G2PP_Refi = .2406 - .1389 * atan(5.952*(1.089 - C_M));
figure
plot(C_M,G2PP_Refi)
xlabel('Coupon/Mortgage Rate')
ylabel('CPR')
title('Refinancing Incentive')
```
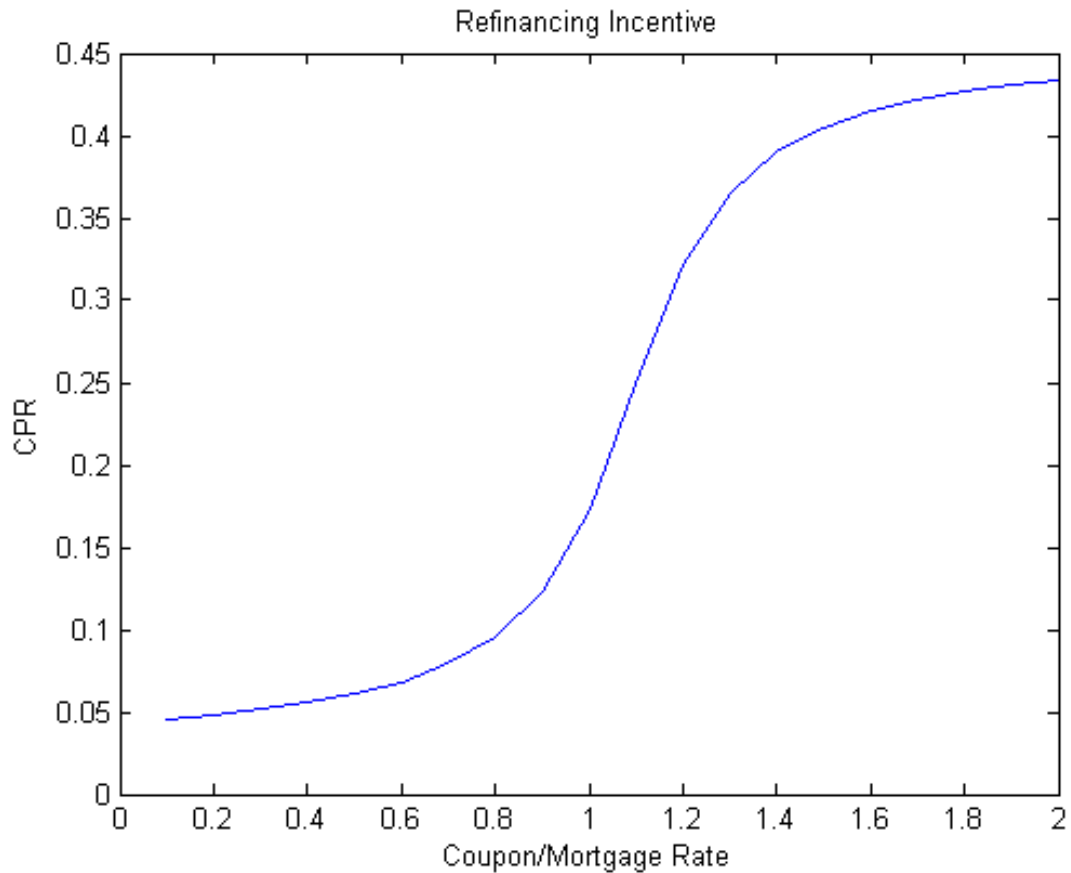
Seasoning captures the tendency of prepayment to ramp up at the beginning of a mortgage before leveling off. The OTS models the seasoning multiplier as follows:

```
Seasoning = ones(360,1);
Seasoning(1:29) = (1:29)/30;
figure
plot(Seasoning)
xlim([1 360])
title('Seasoning Multiplier')
xlabel('Months')
```
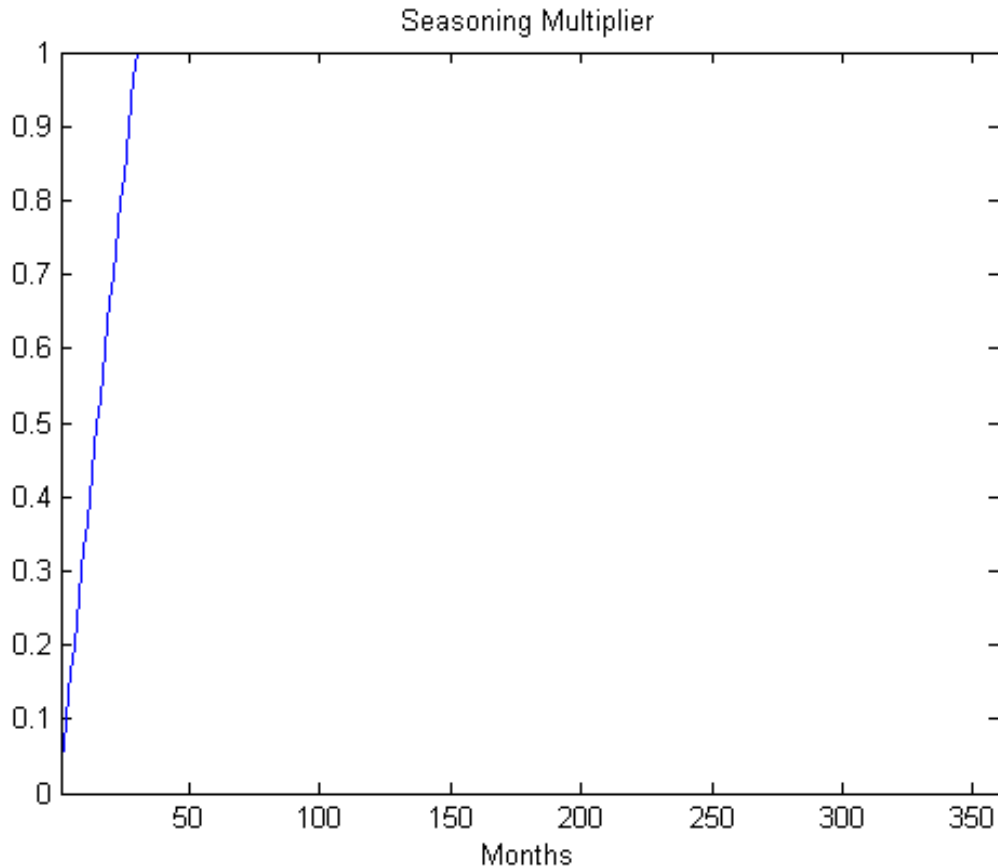
The seasonality multiplier simply models the seasonal behavior of prepayments -- this data is based on Figure 3 of [6], which applies to the behavior of Ginnie Mae 30-year, single-family MBSs.

```
Seasonality = [.94 .76 .73 .96 .98 .92 .99 1.1 1.18 1.21 1.23 .97];
figure
plot(Seasonality)
xlim([1 12])
set(gca,'xtick',1:12)
set(gca,'xtickLabel',{'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug',...
    'Sep','Oct','Nov','Dec'})
title('Seasonality Multiplier')
```

Seasonality Multiplier
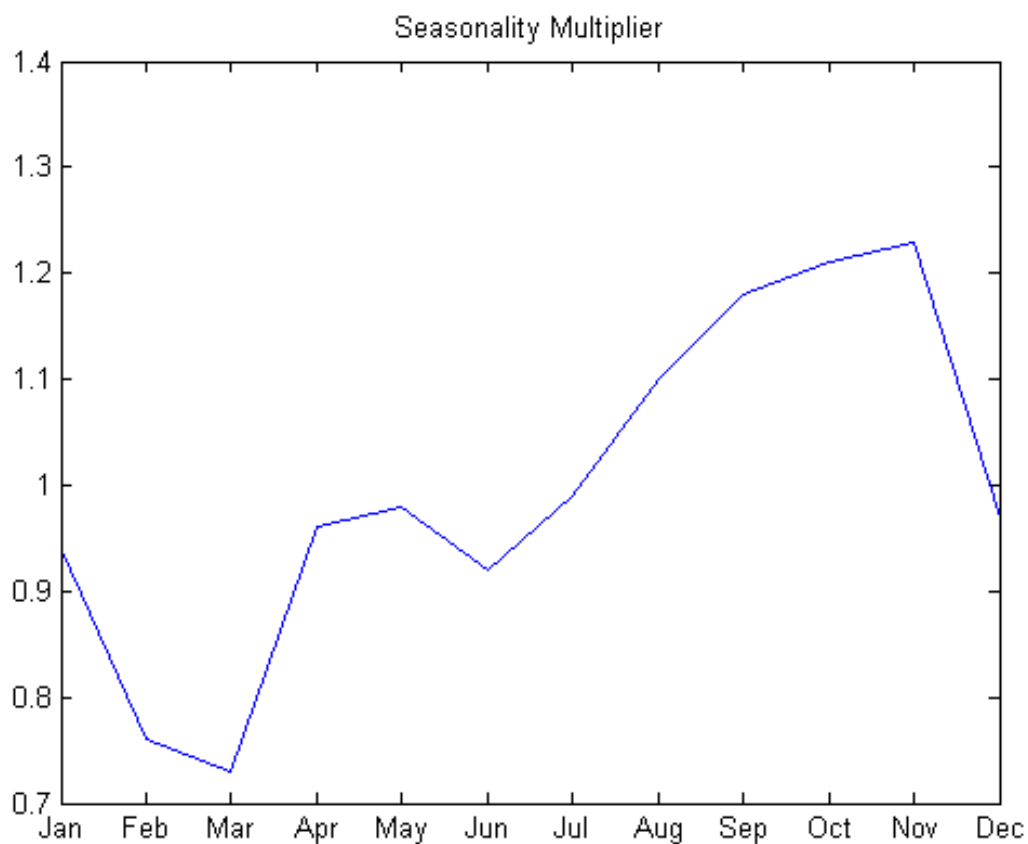
G2++ Interest-Rate Model

Since the refinancing incentive requires a simulation of future interest rates, an interest-rate model must be used. One choice is a two-factor additive Gaussian model, referred to as G2++ by Brigo and Mercurio [2].

The G2++ Interest Rate Model is:

$$r(t) = x(t) + y(t) + \varphi(t)$$

$$dx(t) = -ax(t)dt + \sigma dW_1(t)$$

$$dy(t) = -by(t)dt + \eta dW_2(t)$$

where $dW_1(t)dW_2(t)$ is a two-dimensional Brownian motion with correlation ρ

$$dW_1(t)dW_2(t) = \rho$$

$$\varphi(t) = f^M(0,T) + \frac{\sigma^2}{2a^2}(1 - e^{-aT})^2 + \frac{\eta^2}{2b^2}(1 - e^{-bT})^2 + \rho\frac{\sigma\eta}{ab}(1 - e^{-aT})(1 - e^{-bT})$$

and $r(t)$ is the short rate, a and b are mean reversion constants and σ and η are volatility constants, and $f^M(0,T)$ is the market forward rate, or the forward rate observed on the Settle date.

LIBOR Market Model

The LIBOR Market Model (LMM) differs from short-rate models in that it evolves a set of discrete forward rates. Specifically, the lognormal LMM specifies the following diffusion equation for each forward rate:

$$\frac{dF_i(t)}{F_i} = -\mu_i dt + \sigma_i(t)dW_i$$

where

dW is an N dimensional geometric Brownian motion with:

$$dW_i(t)dW_j(t) = \rho_{ij}$$

The LMM relates the drifts of the forward rates based on no-arbitrage arguments. Specifically, under the Spot LIBOR measure, the drifts are expressed as the following:

$$\mu_i(t) = -\sigma_i(t) \sum_{j=q(t)}^{i} \frac{\tau_j \rho_{i,j} \sigma_j(t) F_j(t)}{1 + \tau_j F_j(t)}$$

where

τ_i is the time fraction associated with the ith forward rate

$q(t)$ is an index function defined by the relation $T_{q(t)-1} < t < T_{q(t)}$

and the Spot LIBOR numeraire is defined as the following:

$$B(t) = P(t, T_{q(t)}) \prod_{n=0}^{q(t)-1} (1 + \tau_n F_n(T_n))$$

Given the above, the choice with the LMM is how to model volatility and correlation.

The volatility of the rates can be modeled with a stochastic volatility, but for this example a deterministic volatility is used, and so a functional form needs to be specified. One of the most popular functional forms in the literature is the following:

$$\sigma_i(t) = \phi_i(a(T_i - t) + b)e^{c(T_i - t)} + d$$

where ϕ adjusts the curve to match the volatility for the ith forward rate.

Similarly, the correlation between the forward rates needs to be specified. This can be estimated from historical data or fitted to option prices. For this example, the following functional form will be used:

$$\rho_{i,j} = e^{-\beta|i-j|}$$

Once the volatility and correlation are specified, the parameters need to be calibrated -- this can be done with historical or market data, typically swaptions or caps/floors. For this example, we simply use reasonable estimates for the correlation and volatility parameters.

```
% The volatility function to be used -- and one choice for the parameters
LMMVolFunc = @(a,t) (a(1)*t + a(2)).*exp(-a(3)*t) + a(4);
LMMVolParams = [.13 .04 .7 .08];

% Volatility specification
fplot(@(t) LMMVolFunc(LMMVolParams,t),[0 10])
title(['Volatility Function with parameters ' mat2str(LMMVolParams)])
```

```
ylabel('Volatility (%)')
xlabel('Tenor (years)')
```

Volatility Function with parameters [0.13 0.04 0.7 0.08]
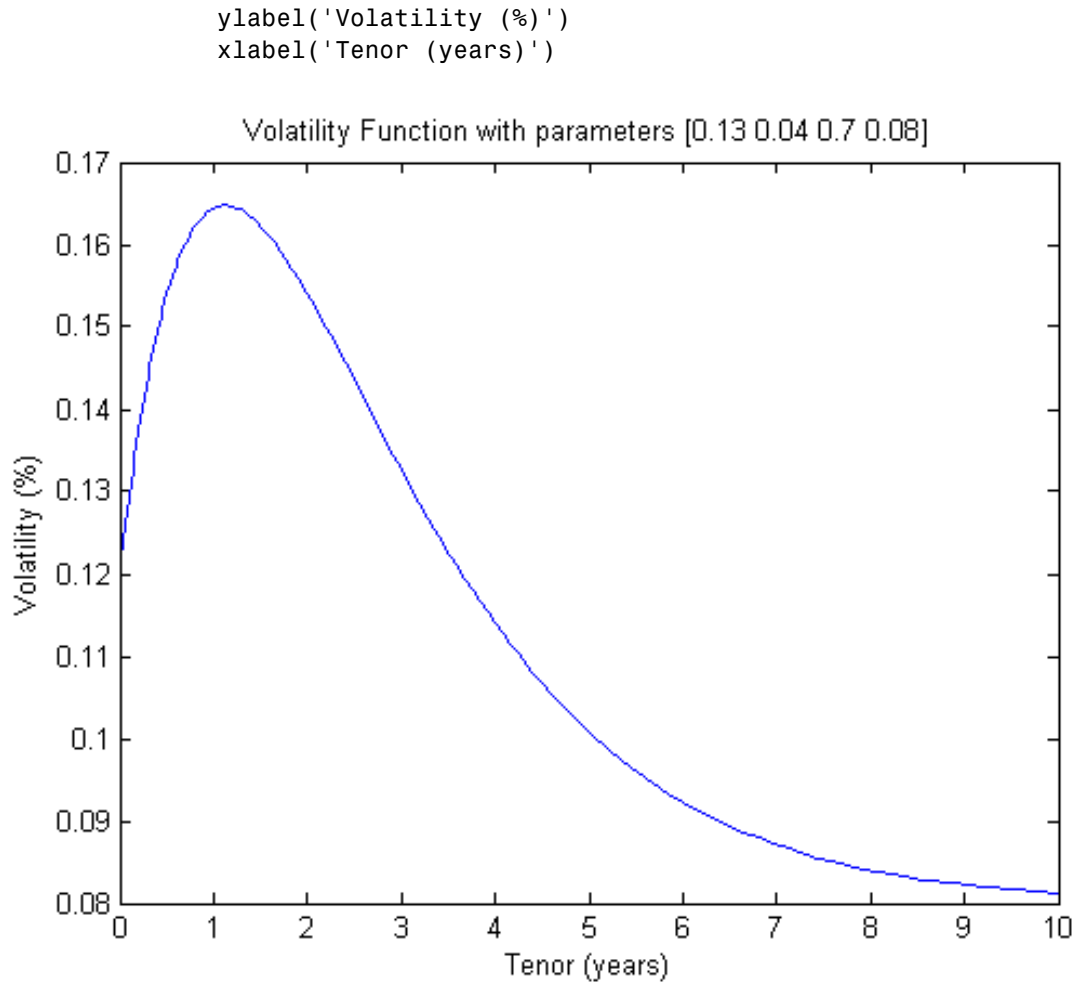
Calibration to Market Data

The parameters in the G2++ model can be calibrated to market data. Typically, the parameters are calibrated to observed interest-rate cap, floor and/or swaption data. For now, market cap data is used for calibration.

This data is hardcoded but could be imported into MATLAB with the
Database Toolbox™ or Datafeed Toolbox™.

```
% Zero Curve -- this data is hardcoded for now but could be bootstrapped
% using the bootstrap method of IRDataCurve
ZeroTimes = [3/12 6/12 1 5 7 10 20 30]';
ZeroRates = [0.033 0.034 0.035 0.040 0.042 0.044 0.048 0.0475]';
ZeroDates = daysadd(Settle,360*ZeroTimes,1);
DiscountRates = zero2disc(ZeroRates,ZeroDates,Settle);
irdc = IRDataCurve('Zero',Settle,ZeroDates,ZeroRates);

figure
plot(ZeroDates,ZeroRates)
datetick
title(['US Zero Curve for ' datestr(Settle)])

% Cap Data
Reset = 2;
Notional = 100;
CapMaturity = daysadd(Settle,360*[1:5 7 10 15 20 25 30],1);
CapVolatility = [.28 .30 .32 .31 .30 .27 .23 .2 .18 .17 .165]';

% ATM strikes could be computed with SWAPBYZERO
Strike = [0.0353 0.0366 0.0378 0.0390 0.0402 0.0421 0.0439 ...
    0.0456 0.0471 0.0471 0.0471]';

% This could be computed with CAPBYBLK
BlackCapPrices = [0.1532 0.6416 1.3366 2.0290 2.7366 4.2960 6.5992 ...
    9.6787 12.2580 14.0969 15.7873]';

figure
scatter(CapMaturity,CapVolatility)
datetick
title(['ATM Volatility for Caps on ' datestr(Settle)])

%
% To calibrate the model parameters, a parameter set will be found that
% minimizes the sum of the squared differences between the G2++ predicted
% Cap values and the observed Black Cap values. The Optimization Toolbox(TM
% function LSQNONLIN is used in this example, although other approaches
```

```
% (e.g. Global Optimization) may also be applicable. The function CAPBYLG2F
% is used to compute analytic values for the caps given parameter values.
%
% Upper and lower bounds for the model parameters are set to be
% relatively constrained. As Brigo and Mercurio discuss, the correlation
% parameter, $$ rho $$, can often be close to -1 when fitting a G2++ model
% to interest-rate cap prices. Therefore, $$ rho $$ is constrained
% to be between -.7 and .7 to ensure that the parameters represent a truly
% two-factor model. The remaining mean reversion and volatility
% parameters are constrained to be between 0 and .5. Calibration remains a
% complex task, and while the plot below indicates that the best fit
% parameters seem to do a reasonably good job of reproducing the Cap
% prices, it should be noted that the procedure outlined here simply
% represents one approach.

% Call to LSQNONLIN to calibrate parameters
objfun = @(x) BlackCapPrices - capbylg2f(irdc,x(1),x(2),x(3),x(4),x(5),Stri
x0 = [.5 .05 .1 .01 -.1];
lb = [0 0 0 0 -.7];
ub = [.5 .5 .5 .5 .7];

G2PP_Params = lsqnonlin(objfun,x0,lb,ub);

a = G2PP_Params(1);
b = G2PP_Params(2);
sigma = G2PP_Params(3);
eta = G2PP_Params(4);
rho = G2PP_Params(5);

% Compare the results
figure
scatter(CapMaturity,BlackCapPrices)
hold on
scatter(CapMaturity,capbylg2f(irdc,a,b,sigma,eta,rho,Strike,CapMaturity),'r
datetick
title('Market and Model Implied Prices')
ylabel('Price ($)')


Local minimum possible.
```

```
lsqnonlin stopped because the final change in the sum of squares relative t
its initial value is less than the default value of the function tolerance.
```


US Zero Curve for 15-Dec-2007

ATM Volatility for Caps on 15-Dec-2007

Market and Model Implied Prices
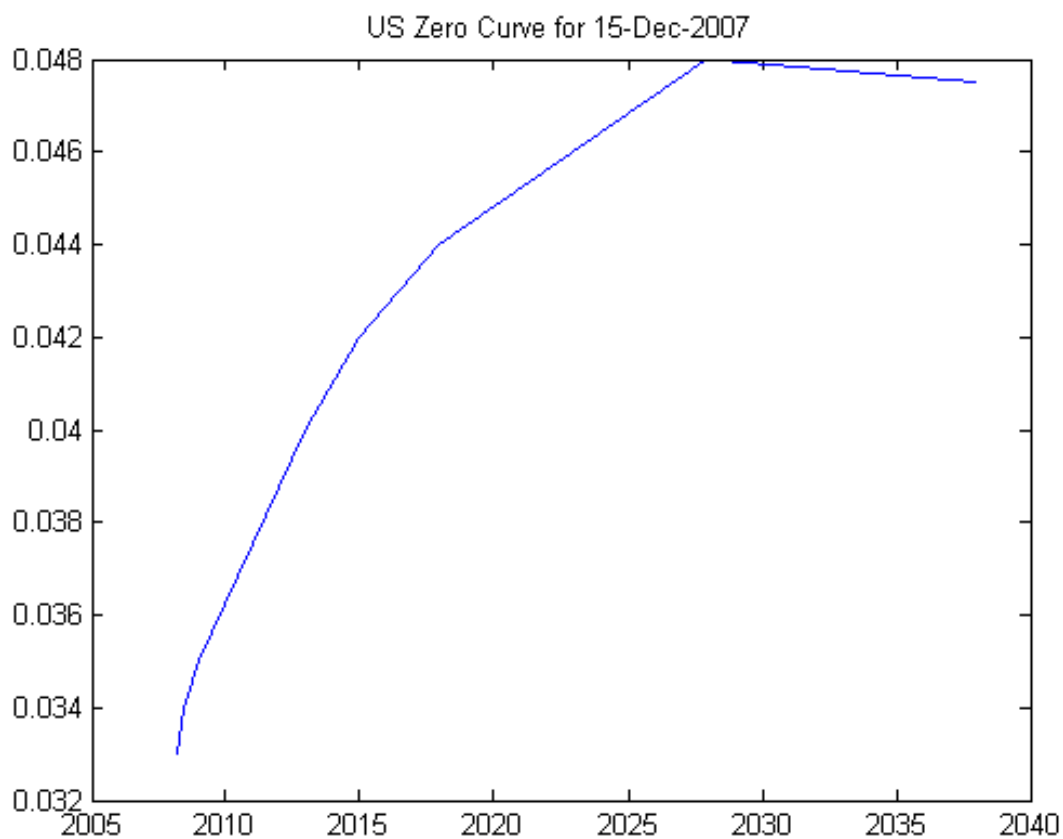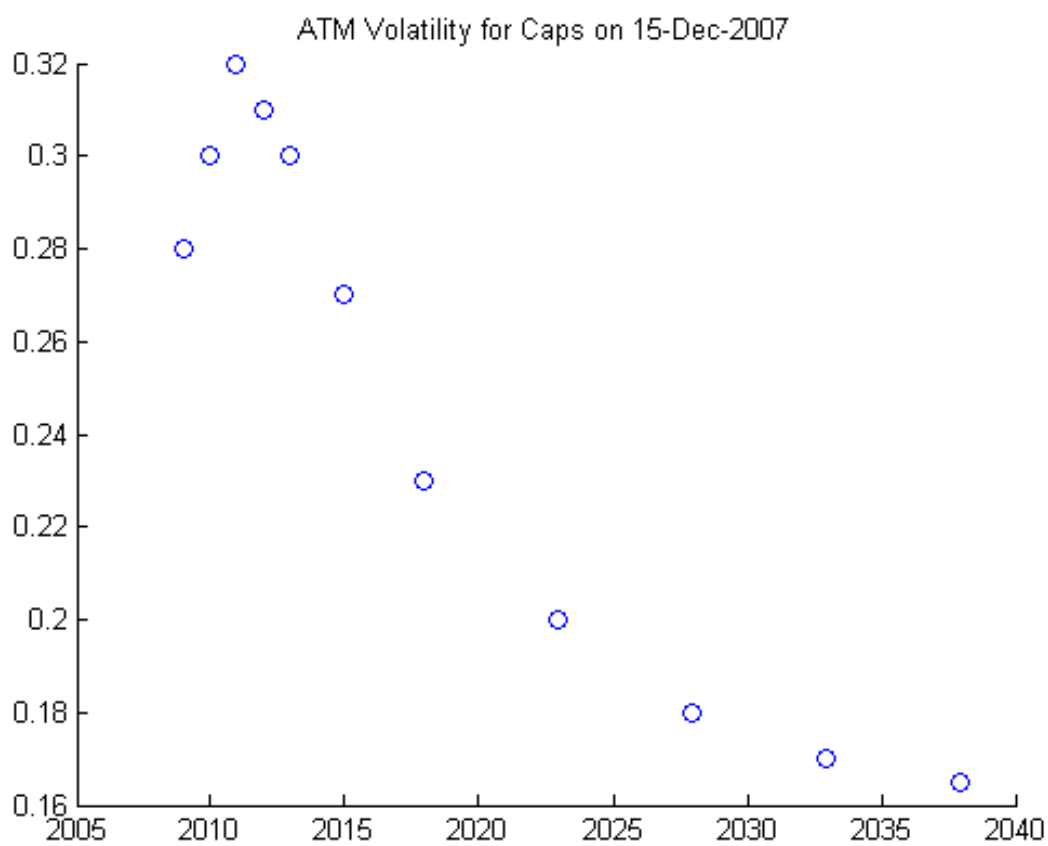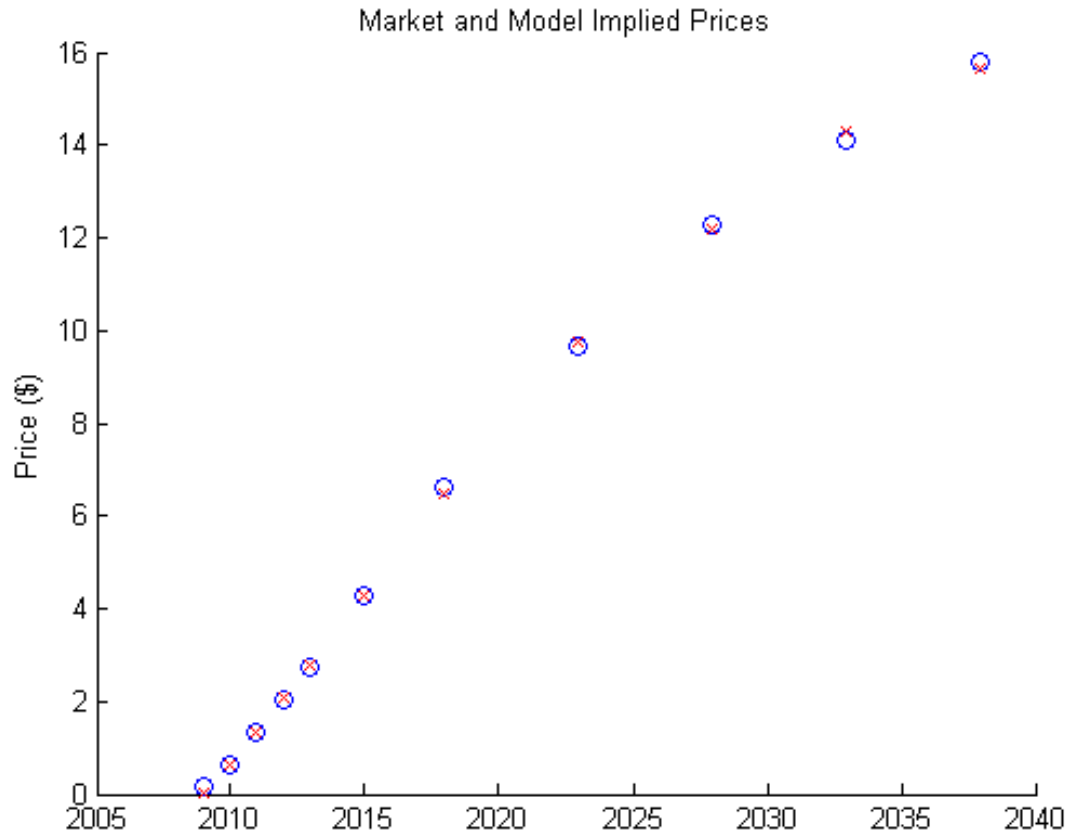
G2++ Model Implementation

The LinearGaussian2F model can be used to specify the G2++ model and simulate future paths interest rates.

```
% G2++ model from Brigo and Mercurio with time homogeneous volatility
% parameters
G2PP = LinearGaussian2F(irdc,a,b,sigma,eta,rho);
```

LIBOR Market Model Implementation

After the volatility and correlation have been calibrated, Monte Carlo simulation is used to evolve the rates forward in time. The LiborMarketModel object is used to simulate the forward rates.

While factor reduction is often used with the LMM to reduce computational complexity, there is no factor reduction in this example.

6M LIBOR rates are chosen to be evolved in this simulation. Since a monthly prepayment vector must be computed, interpolation is used to generate the intermediate rates. Simple linear interpolation is used.

```
numForwardRates = 46;

% Instead of being fit, VolPhi is simply hard-coded  --
% representative of a declining volatility over time.
VolPhi = linspace(1.2,.8,numForwardRates-1)';

Beta = .08;
CorrFunc = @(i,j,Beta) exp(-Beta*abs(i-j));
CorrMat = CorrFunc(meshgrid(1:numForwardRates-1)',meshgrid(1:numForwardRate

VolFunc = cell(length(VolPhi),1);
for jdx=1:length(VolPhi)
    VolFunc(jdx) = {@(t) VolPhi(jdx)*ones(size(t)).*(LMMVolParams(1)*t + ..
        LMMVolParams(2)).*exp(-LMMVolParams(3)*t) + LMMVolParams(4)};
end

LMM = LiborMarketModel(irdc,VolFunc,CorrMat);
```

G2++ Monte Carlo Simulation

The various interest-rate paths can be simulated by calling the simTermStructs method.

One limitation to two-factor Gaussian models like this one is that it does permit negative interest rates. This is a concern, particularly in low interest-rate environments. To handle this possibility, any interest-rate paths with negative rates are simply rejected.

```
nPeriods = NumCouponsRemaining;
```

```
nTrials = 100;
DeltaTime = 1/12;

% Generate factors and short rates
Tenor = [1/12 1 2 3 4 5 7 10 15 20 30];
G2PP_SimZeroRates = G2PP.simTermStructs(nPeriods,'NTRIALS',nTrials,...
    'Tenor',Tenor,'DeltaTime',DeltaTime);

SimDates = daysadd(Settle,360*DeltaTime*(0:nPeriods),1);

% Tenors that will be recovered for each simulation date. The stepsize is
% included here to facilitate computing a discount factor for each
% simulation path.

% Remove any paths that go negative
NegIdx = squeeze(any(any(G2PP_SimZeroRates < 0,1),2));
G2PP_SimZeroRates(:,:,NegIdx) = [];
nTrials = size(G2PP_SimZeroRates,3);

% Plot evolution of one sample path
trialIdx = 1;
figure
surf(Tenor,SimDates,G2PP_SimZeroRates(:,:,trialIdx))
datetick y keepticks keeplimits
title(['Evolution of the Zero Curve for Trial:' num2str(trialIdx) ' of G2++
xlabel('Tenor (Years)')
```

Evolution of the Zero Curve for Trial:1 of G2++ Model
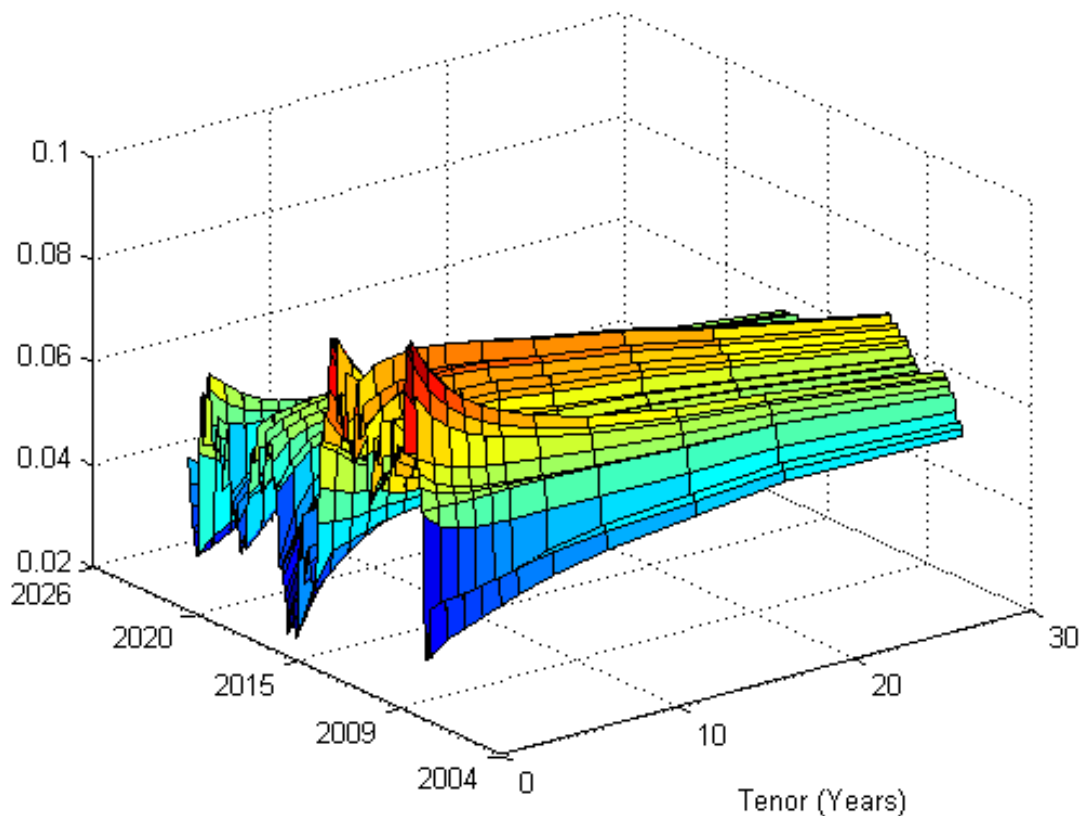
LIBOR Market Model Simulation

The various interest-rate paths can be simulated by calling the simTermStructs method of the LiborMarketModel object.

```
LMMPeriod = 2; % Semi-annual rates
LMMNumPeriods = NumCouponsRemaining/12*LMMPeriod; % Number of semi-annual p
LMMDeltaTime = 1/LMMPeriod;
LMMNTRIALS = 100;

% Simulate
```

```
[LMMZeroRates, LMMForwardRates] = LMM.simTermStructs(LMMNumPeriods,'nTrials
ForwardTimes = 1/2:1/2:numForwardRates/2;
LMMSimTimes = 0:1/LMMPeriod:LMMNumPeriods/LMMPeriod;

% Plot evolution of one sample path
trialIdx = 1;
figure
tmpPlotData = LMMZeroRates(:,:,trialIdx);
tmpPlotData(tmpPlotData == 0) = NaN;
surf(ForwardTimes,LMMSimTimes,tmpPlotData)
title(['Evolution of the Zero Curve for Trial:' num2str(trialIdx) ' of LIBO
xlabel('Tenor (Years)')
```

Evolution of the Zero Curve for Trial:1 of LIBOR Market Model
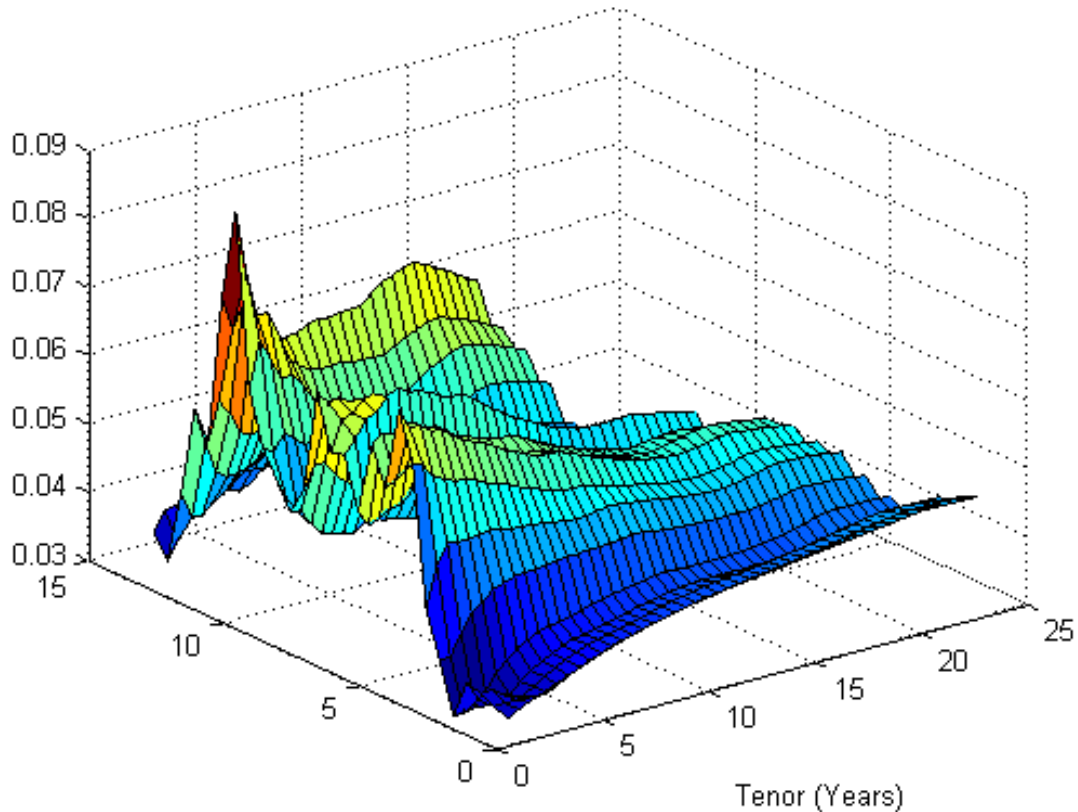
Compute Mortgage Rates from Simulation

Once the interest-rate paths have been simulated, the mortgage rate needs to be computed -- one approach, discussed by [7], is to compute the mortgage rate from a combination of the 2-year and 10-year rates.

For this example, the following is used:

$$MortgageRate = .024 + .2 * TwoYearRate + .6 * TenYearRate$$

```
% Compute mortgage rates from interest rate paths
```

```
TwoYearRates = squeeze(G2PP_SimZeroRates(:,Tenor == 2,:));
TenYearRates = squeeze(G2PP_SimZeroRates(:,Tenor == 7,:));
G2PP_MortgageRates = .024 + .2*TwoYearRates + .6*TenYearRates;

LMMMortgageRates = squeeze(.024 + .2*LMMZeroRates(:,4,:) + .6*LMMZeroRates(
LMMDiscountFactors = squeeze(cumprod(1./(1 + LMMZeroRates(:,1,:)*.5)));

% Interpolate to get monthly mortgage rates
MonthlySimTimes = 0:1/12:LMMNumPeriods/LMMPeriod;
LMMMonthlyMortgageRates = zeros(nPeriods+1,LMMNTRIALS);
LMMMonthlyDF = zeros(nPeriods+1,LMMNTRIALS);
for trialidx=1:LMMNTRIALS
   LMMMonthlyMortgageRates(:,trialidx) = interp1(LMMSimTimes,LMMMortgageRat
   LMMMonthlyDF(:,trialidx) = interp1(LMMSimTimes,LMMDiscountFactors(:,tria
end
```

Computing CPR and Generating and Valuing Cash Flows

Once the Mortgage Rates have been simulated, the CPR can be computed
from the multiplicative model for each interest-rate path.

```
% Compute Seasoning and Refinancing Multipliers
Seasoning = ones(nPeriods+1,1);
Seasoning(1:30) = 1/30*(1:30);
G2PP_Refi = .2406 - .1389 * atan(5.952*(1.089 - CouponRate./G2PP_MortgageRa
LMM_Refi = .2406 - .1389 * atan(5.952*(1.089 - CouponRate./LMMMonthlyMortga

% CPR is simply computed by evaluating the multiplicative model
G2PP_CPR = bsxfun(@times,G2PP_Refi,Seasoning.*(Seasonality(month(CFlowDates
LMM_CPR = bsxfun(@times,LMM_Refi,Seasoning.*(Seasonality(month(CFlowDates))

% Compute single monthly mortality (SMM) from CPR
G2PP_SMM = 1 - (1 - G2PP_CPR).^(1/12);
LMM_SMM = 1 - (1 - LMM_CPR).^(1/12);

% Plot CPR's against 100 PSA
CPR_PSA100 = psaspeed2rate(100);
figure
PSA_handle = plot(CPR_PSA100(1:nPeriods),'rx');
hold on
```

```
G2PP_handle = plot(G2PP_CPR,'b');
LMM_handle = plot(LMM_CPR,'g');
title('Prepayment Speeds')
legend([PSA_handle(1) G2PP_handle(1) LMM_handle(1)],{'100 PSA','G2PP CPR','
```
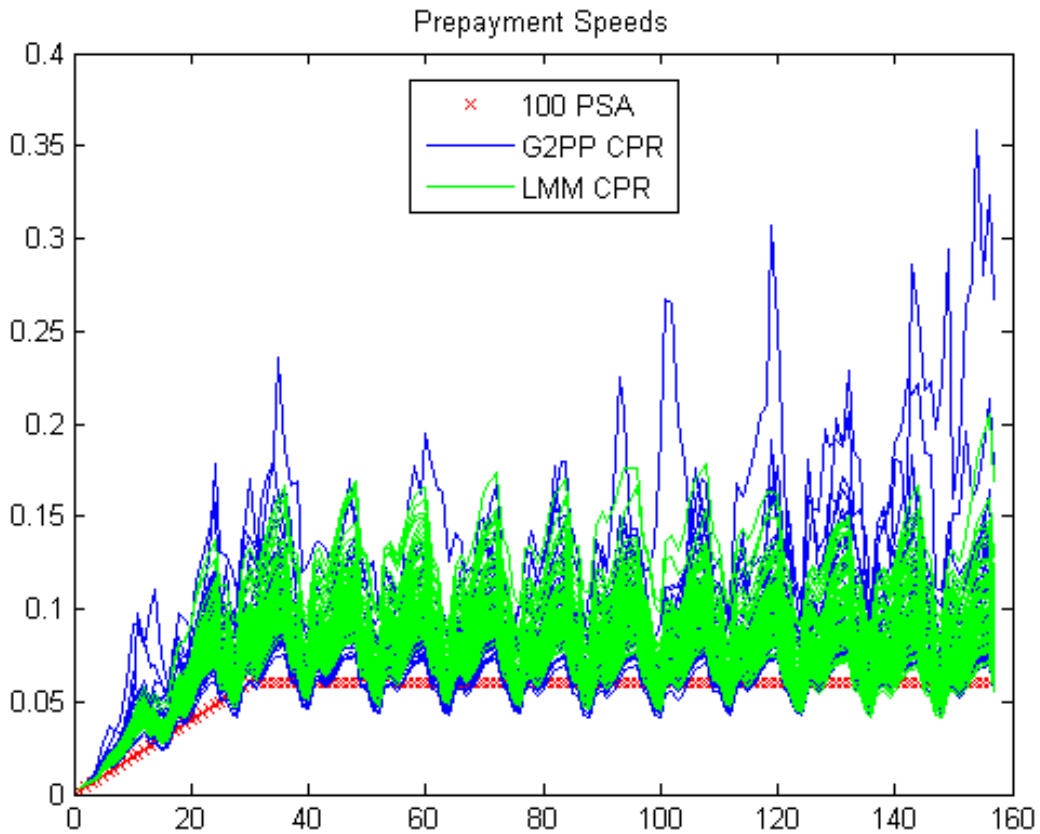

Generate Cash Flows and Compute Present Value

With a vector of single monthly mortalities (SMM) computed for each interest-rate path, cash flows for the MBS can be computed and discounted.

```
% Compute the baseline zero rate at each cash flow time
```

```
CFlowZero = interp1(ZeroTimes,ZeroRates,CFlowTimes,'linear','extrap');

% Compute DF for each cash flow time
CFlowDF_Zero = zero2disc(CFlowZero,CFlowDates,Settle);

% Compute the price of the MBS using the zero curve
Price_Zero = CFlowAmounts*CFlowDF_Zero';

% Generate the cash flows for each IR Path
G2PP_CFlowAmounts = mbscfamounts(Settle, ...
    repmat(Maturity,1,nTrials), IssueDate, GrossRate, CouponRate, Delay, []

% Compute the DF for each IR path
G2PP_CFlowDFSim = cumprod(exp(squeeze(-G2PP_SimZeroRates(:,1,:).*DeltaTime)

% Present value the cash flows for each MBS
G2PP_Price_Ind = sum(G2PP_CFlowAmounts.*G2PP_CFlowDFSim',2);
G2PP_Price = mean(G2PP_Price_Ind);

% Repeat for LMM
LMM_CFlowAmounts = mbscfamounts(Settle, ...
    repmat(Maturity,1,LMMNTRIALS), IssueDate, GrossRate, CouponRate, Delay,

% Present value the cash flows for each MBS
LMM_Price_Ind = sum(LMM_CFlowAmounts.*LMMMonthlyDF',2);
LMM_Price = mean(LMM_Price_Ind);
```

The results from the different approaches can be compared. The number of trials for the G2++ model will typically be less than 100 due to the filtering out of any paths that produce negative interest rates.

Additionally, while the number of trials for the G2++ model in this example is set to be 100, it is often the case that a larger number of simulations need to be run to produce an accurate valuation.

```
fprintf('                         # of Monte Carlo Trials: %8d\n'     , nTrials)
fprintf('                       # of Time Periods/Trial: %8d\n\n'  , nPeriods
fprintf('                          MBS Price with PSA 100: %8.4f\n'  , Price_Ze
fprintf(' MBS Price with Custom G2PP Prepayment Model: %8.4f\n\n', G2PP_Pri
fprintf(' MBS Price with Custom LMM Prepayment Model: %8.4f\n\n', LMM_Price
```

```
                    # of Monte Carlo Trials:      73
                    # of Time Periods/Trial:     156

                     MBS Price with PSA 100:  1.0187
 MBS Price with Custom G2PP Prepayment Model:  0.9863

 MBS Price with Custom LMM Prepayment Model:   1.0006
```

Conclusion

This example shows how to calibrate and simulate a G2++ interest-rate model and how to use the generated interest-rate paths in a prepayment model loosely based on the Richard and Roll model. This example also provides a useful starting point to using the G2++ and LMM interest-rate models in other financial applications.

Bibliography

This example is based on the following books, papers and journal articles:

1 Andersen, L. and V. Piterbarg (2010). Interest Rate Modeling, Atlantic Financial Press.

2 Brigo, D. and F. Mercurio (2001). Interest Rate Models - Theory and Practice with Smile, Inflation and Credit (2nd ed. 2006 ed.). Springer Verlag. ISBN 978-3-540-22149-4.

3 Hayre, L, ed., Salomon Smith Barney Guide to Mortgage-Backed and Asset-Backed Securities. New York: John Wiley & Sons, 2001b

4 Karpishpan, Y., O. Turel, and A. Hasha, Introducing the Citi LMM Term Structure Model for Mortgages, The Journal of Fixed Income, Volume 20 (2010) 44-58.

5 Rebonato, R., K. McKay, and R. White (2010). The Sabr/Libor Market Model: Pricing, Calibration and Hedging for Complex Interest-Rate Derivatives. John Wiley & Sons.

6 Richard, S. F., and R. Roll, 1989, "Prepayments on Fixed Rate Mortgage-Backed Securities" ,Journal of Portfolio Management.

7 Office of Thrift Supervision, "Net Portfolio Value Model Manual", March 2000.

8 Stein, H. J., Belikoff, A. L., Levin, K. and Tian, X., Analysis of Mortgage Backed Securities: Before and after the Credit Crisis (January 5, 2007). Credit Risk Frontiers: Subprime Crisis, Pricing and Hedging, CVA, MBS, Ratings, and Liquidity; Bielecki, Tomasz,; Damiano Brigo and Frederic Patras, eds., February 2011. Available at SSRN: http://ssrn.com/abstract=955358

Pricing Mortgage Backed Securities Using the Black-Derman-Toy Model

This example illustrates how the Financial Toolbox™ and Financial Instruments Toolbox™ can be used to price a level mortgage backed security using the BDT model.

Load the BDT Tree Stored in the Data File

```
load mbsexample.mat
```

Observe the Interest Rate Tree

Visualize the interest rate evolution along the tree by looking at the output structure BDTTree. BDTTree returns an inverse discount tree, which you can convert into an interest rate tree with the cvtree function.

```
BDTTreeR = cvtree(BDTTree);
```

Look at the upper branch and lower branch paths of the tree:

```
OldFormat = get(0, 'format');
format short

%Rate at root node:
RateRoot      = treepath(BDTTreeR.RateTree, [0])

%Rates along upper branch:
RatePathUp    = treepath(BDTTreeR.RateTree, [1 1 1 1 1])

%Rates along lower branch:
RatePathDown = treepath(BDTTreeR.RateTree, [2 2 2 2 2])


RateRoot =

    0.0399


RatePathUp =
```

```
        0.0399
        0.0397
        0.0391
        0.0383
        0.0373
        0.0360


RatePathDown =

        0.0399
        0.0470
        0.0550
        0.0638
        0.0734
        0.0841
```

Compute the Price Tree for the Non-Prepayable Mortgage

Let's say that we have a 3 year $10000 level prepayable loan, with a mortgage
interest rate of 4.64% semi-annually compounded.

```
MortgageAmount = 10000;
CouponRate = 0.0464;
Period = 2;
Settle='01-Jan-2007';
Maturity='01-Jan-2010';
Compounding = BDTTree.TimeSpec.Compounding;

format bank

% Use the function 'amortize.m' in the Financial Toolbox to calculate the m
% payment of the loan (MP), the interest and principal components, and the
% outstanding principal balance.

NumPeriods = date2time(Settle,Maturity, Compounding)';

[Principal, InterestPayment, OutstandingBalance, MP] = amortize(CouponRate/
```

5-43

```
% Display Principal, Interest and Outstanding balances
PrincipalAmount = Principal'
InterestPaymentAmount = InterestPayment'
OutstandingBalanceAmount =OutstandingBalance'

CFlowAmounts = MP*ones(1,NumPeriods);
% The CFlowDates are the same as the tree level dates
CFlowDates= {'01-Jul-2007' ,'01-Jan-2008' ,'01-Jul-2008' , '01-Jan-2009' ,

% Calculate the price of the non-prepayable mortgage
[PriceNonPrepayableMortgage, PriceTreeNonPrepayableMortgage] = cfbybdt(BDTT
for iLevel = 2:length(PriceTreeNonPrepayableMortgage.PTree)
    PriceTreeNonPrepayableMortgage.PTree{iLevel}(:,:)= PriceTreeNonPrepayab
end

% Look at the price of the mortgage today (tObs = 0)
PriceNonPrepayableMortgage

% The value of the non-prepayable mortgage is $10017.47. This value exceeds
% the $10000 amount borrowed since the homeowner received not only $10000,
% also a prepayment option.

% Look at the value of the mortgage on the last date, right after the last
% mortgage payment, is zero:
PriceTreeNonPrepayableMortgage.PTree{end};

% Visualize the price tree for the non-prepayable mortgage.
treeviewer(PriceTreeNonPrepayableMortgage)


PrincipalAmount =

       1572.59
       1609.07
       1646.40
       1684.60
       1723.68
       1763.67
```

```
InterestPaymentAmount =

       232.00
       195.52
       158.19
       119.99
        80.91
        40.92


OutstandingBalanceAmount =

      8427.41
      6818.34
      5171.94
      3487.35
      1763.67
         0.00


PriceNonPrepayableMortgage =

     10017.47
```
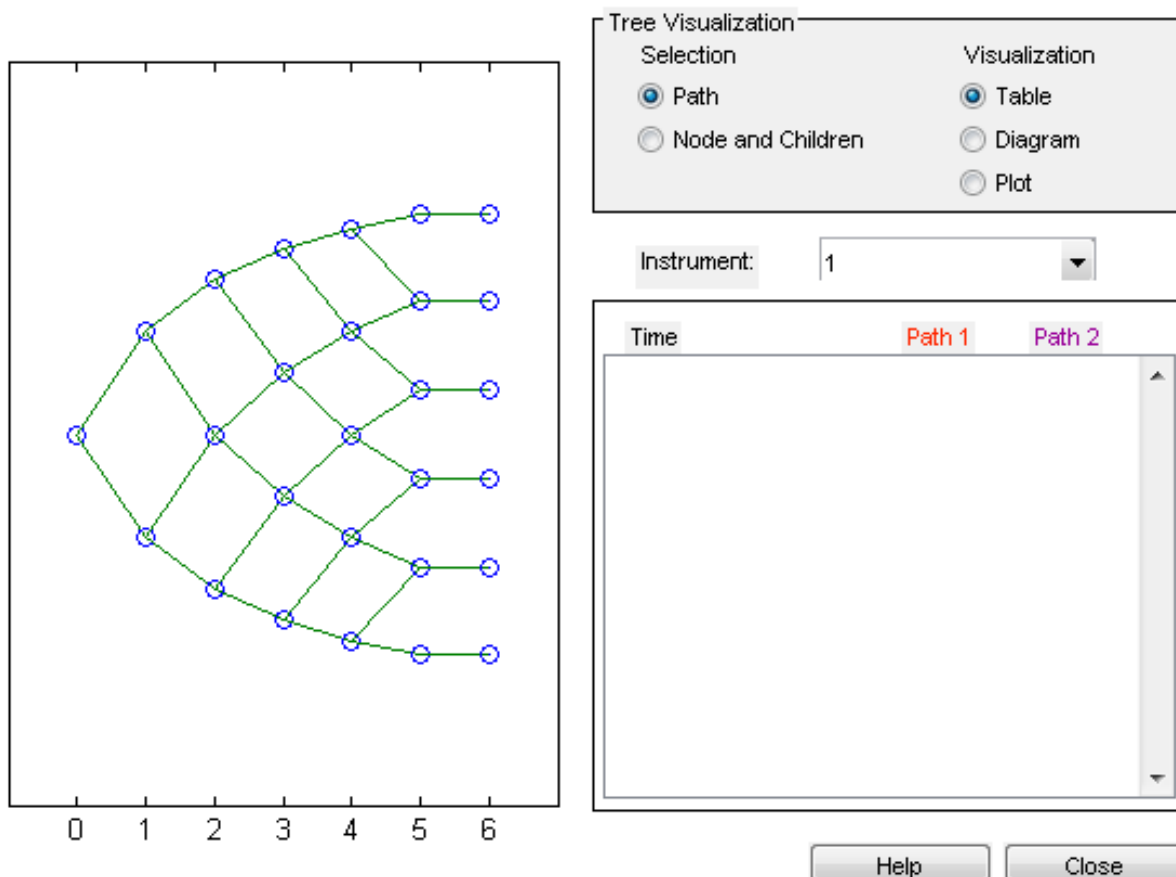
Compute the Price Tree of the Prepayment Option

```
% The Prepayment option is like a call option on a bond.
%
% The exercise price or strike will be equal to the outstanding principal a
% which has been calculated using the function 'amortize.m'.

OptSpec = 'call';
Strike = [MortgageAmount OutstandingBalance];
ExerciseDates =[Settle CFlowDates];
AmericanOpt = O;
```

```
Maturity = CFlowDates(end);

% Compute the price of the prepayment option:
[PricePrepaymentOption, PriceTreePrepaymentOption] = prepaymentbybdt(BDTTre
                        O, Settle, Maturity,[], [], [], ...
                        [], [], [],  [], O, [], CFlowAmounts);


% Look at the price of the prepayment option today (tObs = O)
PricePrepaymentOption

% The value of the prepayment option is $17.47 as expected.

% Visualize the price tree for the prepayment option
treeviewer(PriceTreePrepaymentOption)


PricePrepaymentOption =

        17.47
```
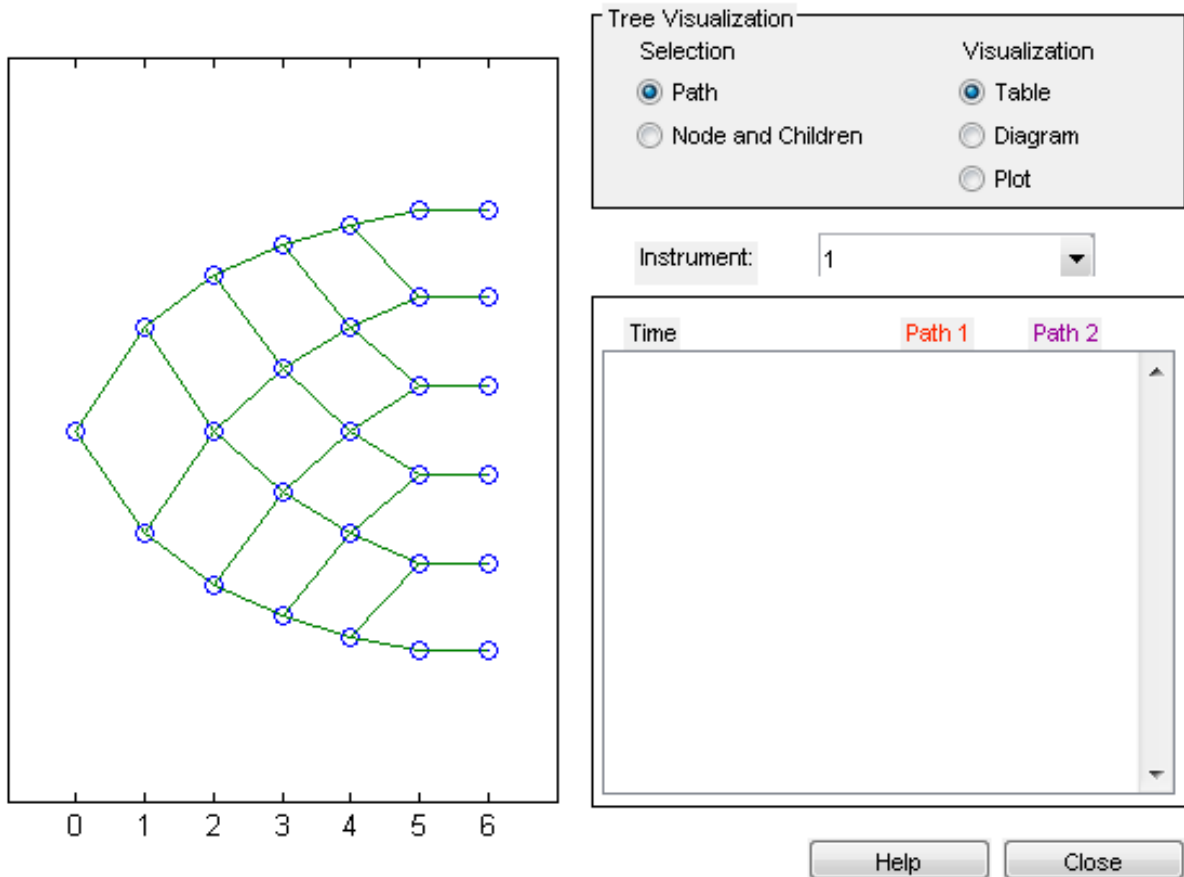
Calculate the Price Tree of the Prepayable Mortgage.

```
% Compute the price of the prepayable mortgage.

PricePrepayableMortgage = PriceNonPrepayableMortgage - PricePrepaymentOptio

PriceTreePrepayableMortgage = PriceTreeNonPrepayableMortgage;

for iLevel = 1:length(PriceTreeNonPrepayableMortgage.PTree)
    PriceTreePrepayableMortgage.PTree{iLevel}(:,:)= PriceTreeNonPrepayableM
        PriceTreePrepaymentOption.PTree{iLevel}(:,:);
```

```
end

% Look at the price of the prepayable mortgage today (tObs = 0)
PricePrepayableMortgage

% The value of the prepayable mortgage is $10000 as expected.

% Visualize the price and price tree for the prepayable mortgage
treeviewer(PriceTreePrepayableMortgage)
set(0, 'format', OldFormat);


PricePrepayableMortgage =

      10000.00
```

Using Collateralized Mortgage Obligations (CMOs)

In this section...
"What Are CMOs?" on page 5-51
"Prepayment Risk" on page 5-51
"CMO Workflow" on page 5-62

What Are CMOs?

Financial Instruments Toolbox supports collateralized mortgage obligations (CMOs) to provide investors with a greater range of risk and return characteristics than mortgage-backed securities (MBS). In contrast to an MBS, which simply redirects principal and interest cash flows to investors on a pro rata basis, a CMO structures cash flows to different tranches, or slices, to create securities that are better tailored to specific investors.

For example, banks might be primarily concerned with *extension risk*, or the risk that their investment lengthens in time due to increasing interest rates, given that they typically have short-term deposits as liabilities. Insurance companies and pension funds might be concerned primarily with *contraction risk*, or the risk that their investment will pay off too soon, with liabilities that have much longer lives. A CMO structure addresses the interest-rate risk of extension or contraction with a blend of short-term and long-term CMO securities, called tranches.

Prepayment Risk

Prepayment risk is the risk that the term of the security varies according to differing rates of repayment of principal by borrowers (repayments from refinancings, sales, curtailments, or foreclosures). In a CMO, you can structure the principal (and associated coupon) stream from the underlying mortgage pool collateral to allocate prepayment risk. If principal is prepaid faster than expected (for example, if mortgage rates fall and borrowers refinance), then the overall term of the mortgage pool collateral shortens.

You cannot remove prepayment risk, but you can reallocate it among CMO tranches so that some tranches have some protection against this risk, and other tranches will absorb more of this risk. To facilitate this allocation of

prepayment risk, CMOs are structured such that prepayments are allocated among tranches using a fixed set of rules. The most common schemes for prepayment tranching are:

- Sequential tranching, with or without, Z-bond tranching
- Schedule bond tranching
    - Planned amortization class (PAC) bonds
    - Target amortization class (TAC) bonds

Financial Instruments Toolbox supports these schemes for prepayment tranching for CMOs and tools for pricing and scheduling cash flows between the tranches, as well as analyzing the price and yield for CMOs. Financial Instruments Toolbox functionality for CMOs does not model credit risk. Therefore, this functionality is most appropriate for CMOs where credit risk is not an issue (for example, agency CMOs where the underlying mortgage pool collateral is insured for default by the agency Government-Sponsored Enterprises (GSEs), such as Fannie Mae and Freddie Mac).

Sequential Tranches Without a Z-Bond

All available principal and interest payments go to the first sequential tranche, until its balance decrements to zero, then to the second, and so on. For example, consider the following example where all principal and interest from the underlying mortgage pool is repaid on tranche A first, then tranche B, then tranche C. Note that interest is paid on each tranche as long as the principal for the tranche has not been retired.
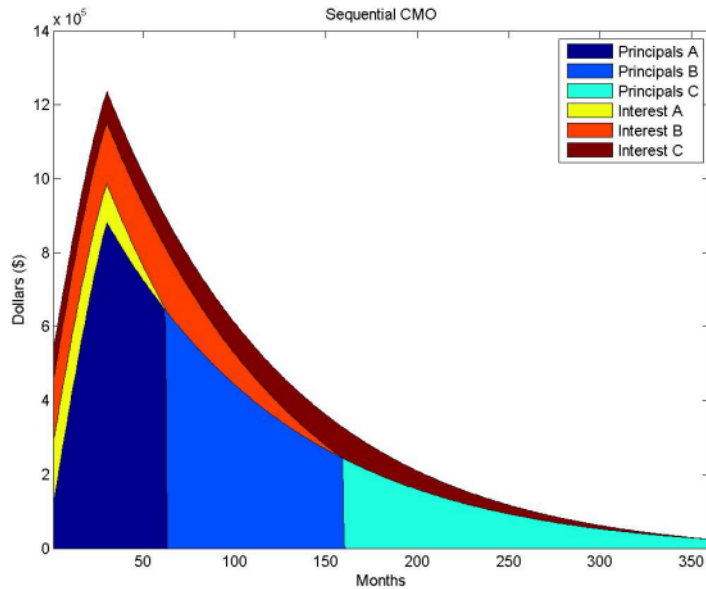
Sequential Tranches With a Z-Bond

The Z-bond, also called an accrual bond, is a type of interest and principal pay rule. The Z-bond tranche supports other sequential pay tranches by not receiving an interest payment. The interest payment that would have accrued to the Z-bond tranche pays off the principal of other bonds, and the principal of the Z-bond tranche increases. The Z-bond tranche starts receiving interest and principal payments only after the other tranches in the CMO have been fully paid. The Z-bond tranche is used in a sequential-pay structure to accelerate the principal repayments of the sequential-pay bonds.

A Z-bond differs from other CMO instruments because it is not tranching principal but interest. The Z-bond receives no cash flows until all other securities have been paid off. In the interim, the interest that is owed to the Z-bond is accrued to its principal. The following chart demonstrates the difference between a Z-bond and a normal sequential pay tranche. Note that the C tranche pays off sooner with the Z-bond, because the interest cash flows to the Z-bond are being used to pay down the principal of the C tranche.
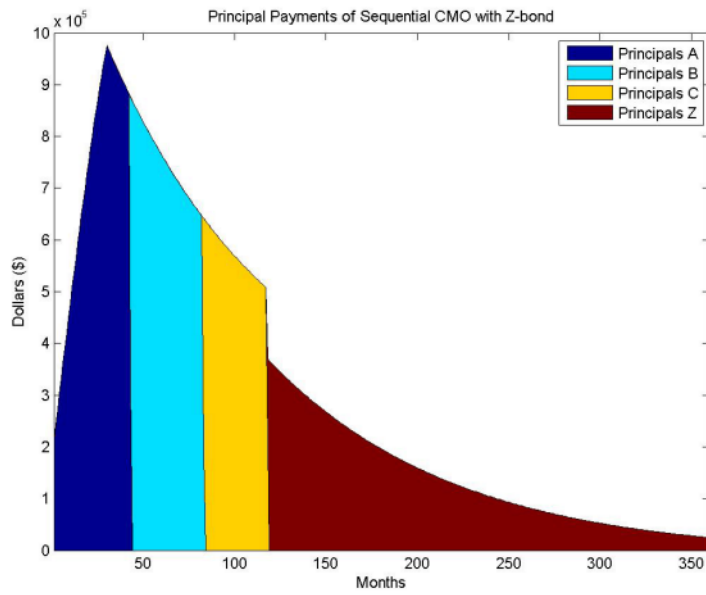
For comparison, the following graphic is the same sequential CMO with no Z-bond.
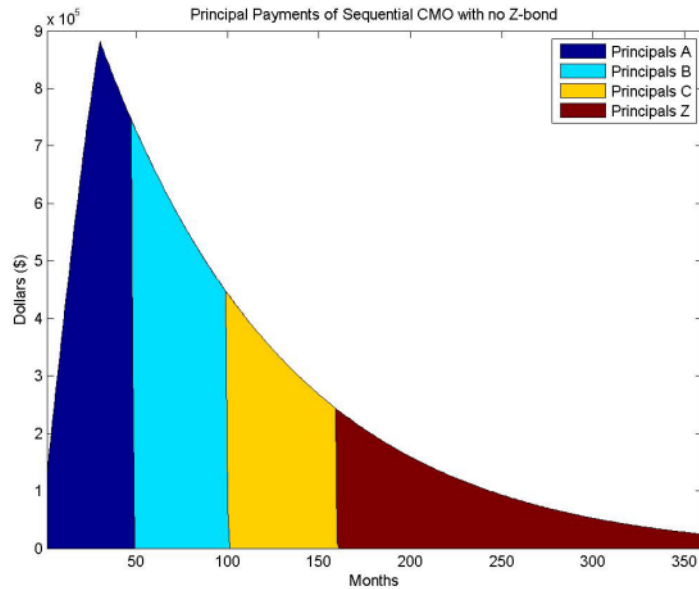
PAC Tranches

Planned amortization class (PAC) bonds help reduce the effects of prepayment risk. They are designed to produce more stable cash flows by redirecting prepayments from the underlying mortgage collateral to other classes (tranches) called companion or support classes. PAC bonds have a principal payment rate over a predetermined period of time. The PAC bond payment schedule is determined by two different prepayment rates, which together form a band (also called a collar). Early in the life of the CMO, the prepayment at the lower PSA yields a lower prepayment. Later in its life, the principal in the higher PSA declines enough that it yields a lower prepayment. The PAC tranche receives whichever rate is lower, so it will change prepayment at one PSA for the first part of its life, then switch to the other rate. The ability to stay on this schedule is maintained by a support bond, which absorbs excess prepayments, and receives less prepayments to prevent extension of average life.

However, the PAC is only protected from extension to the amount that prepayments are made on the underlying MBSs. If there is a sustained

period of fast prepayments, then that might completely eliminate a PAC bond's outstanding support class. When the principal of the associated PAC bond is exhausted, the CMO is called a "busted PAC", or "busted collar". Alternatively, in times of slow prepayments, amortization of the support bonds is delayed if there is not enough principal for the currently paying PAC bond. This extends the average life of the class.

A PAC bond protects against both extension and contraction risk by:

- Specifying a schedule of principal payments for the PAC bond

- Including support tranches that are allocated prepayments inside a specified prepayment band

PAC bonds typically specify a band expressed using the PSA model. A PAC bond with a range of 100 to 250% has this principal schedule.
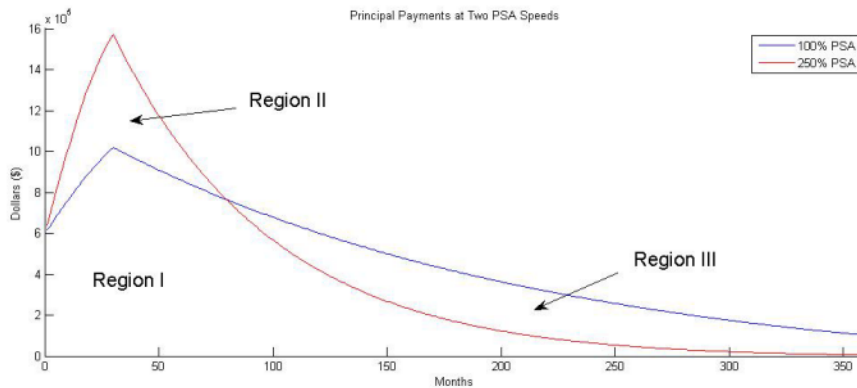
The principal repayment schedule is the minimum principal payment as Region 1 shows. Note that this is the principal payment schedule as long as the actual prepayment stays within the prepayment band of 100 to 250% PSA.

For example, for different prepayment speeds of 125%, 175%, and 225% PSA, the actual principal payments are shown in the following graphs. Note that at higher prepayment speeds, the support tranche is allocated principal earlier while the principal timing for the other tranches remains constant.
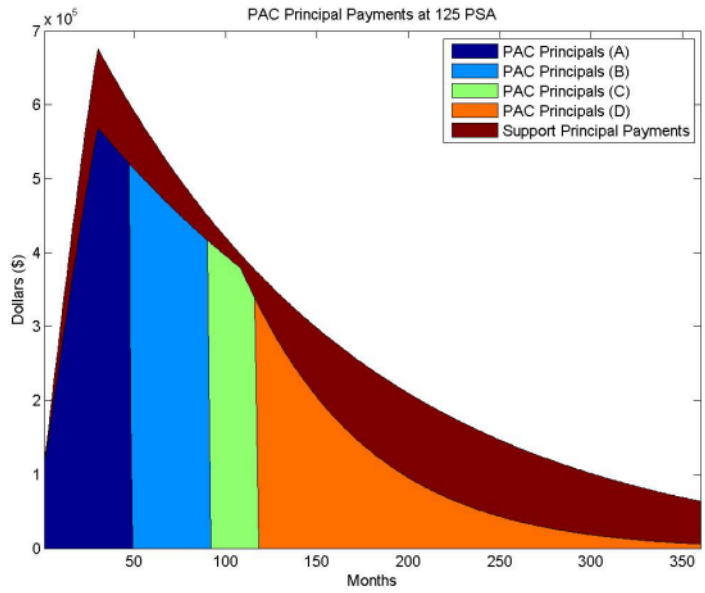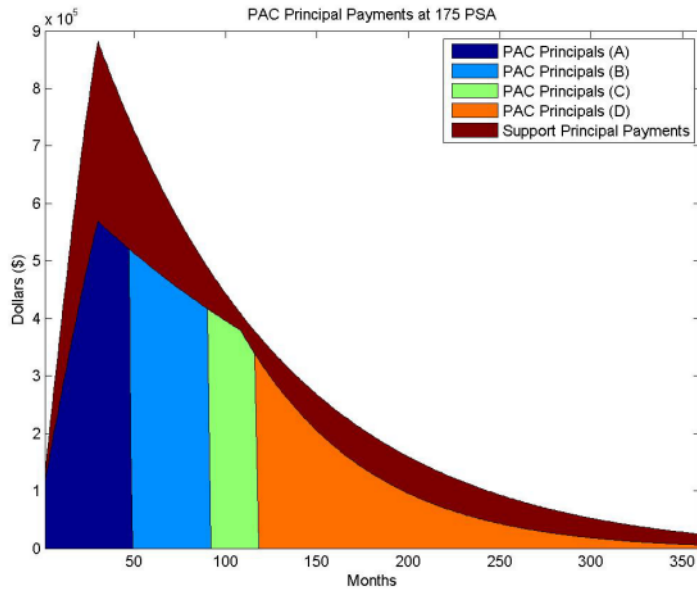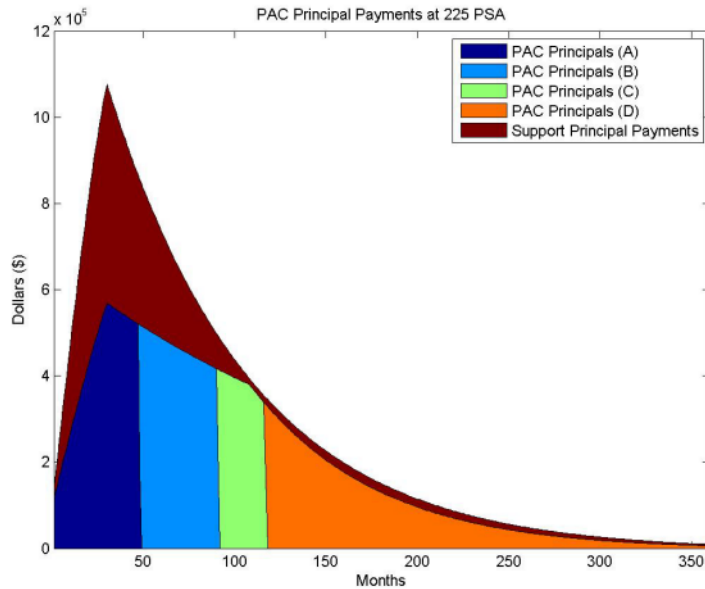
TAC Tranches

Target amortization class (TAC) bonds are similar to PAC bonds, but they do not provide protection against extension of average life. Create the schedule of principal payments by using just a single PSA. TAC bonds pay a "targeted" principal payment schedule at a single, constant prepayment speed. As long as the underlying mortgage collateral does not prepay at a rate slower than this speed, the TAC bond payment schedule is met. TAC bonds can protect against increasing prepayments and early retirement of the TAC bond investment. If the principal cash flow from the mortgage collateral exceeds the TAC schedule, the excess is allocated to TAC companion (support) classes. Alternatively, if prepayments fall below the speed necessary to maintain the TAC schedule, the weighted average life of the TAC is extended. The TAC bond does not protect against low prepayment rates.

For example, here is a TAC structure rated for 125%, 175%, and 450% PSA.
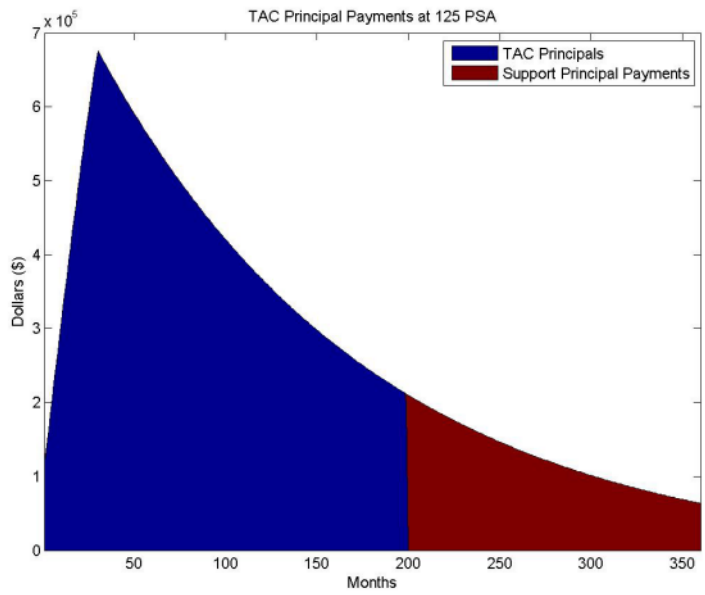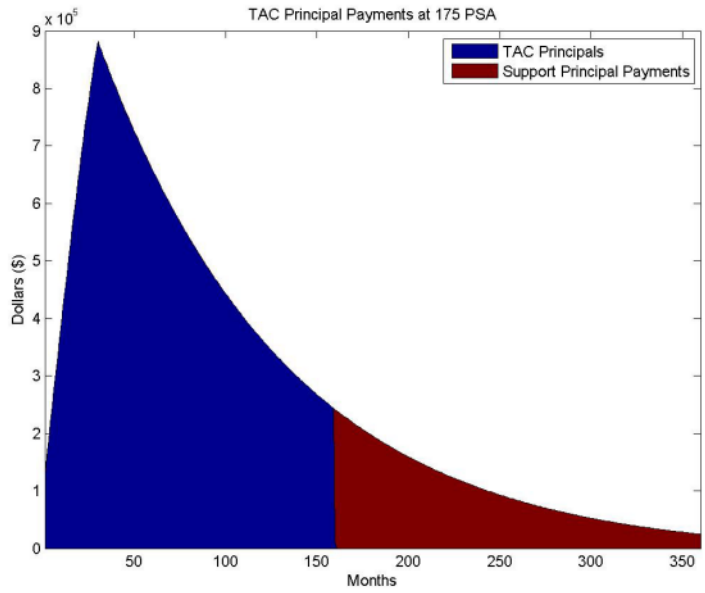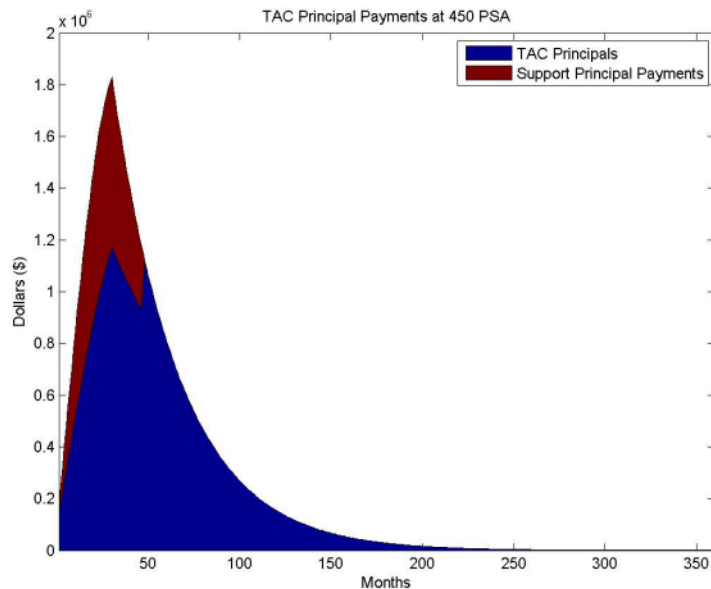
Note that for prepayments below 175% PSA, the TAC bond extends like a normal sequential pay CMO. TAC bonds are appealing because they offer higher yields than comparable PAC bonds. The unaddressed risk from low prepyament rates generally does not concern investors as much as risk from high prepayment rates.

CMO Workflow

In general, the CMO workflow is:

1 Calculate underlying mortgage cash flows.

2 Define CMO tranches

3 If using a PAC or TAC CMO, calculate the principal schedule.

4 Calculate cash flows for each tranche.

5 Analyze the CMO by computing price, yield, spread of CMO cash flows.

Calculate Underlying Mortgage Cash Flows

Underlying mortgage pool pass-through cash flows are calculated by the existing function `mbspassthrough`. The CMO cash flow functions require the principal payments (including prepayments) calculated from existing functions `mbspassthrough` or `mbscfamounts`.

```
principal = 10000000;
coupon = 0.06;
terms = 360;
psa = 150;

[principal_balance, monthly_payments, sched_principal_payments,...
interest_payments, prepayments] = mbspassthrough(principal,...
coupon, terms, terms, psa, []);

principal_payments = sched_principal_payments.' + prepayments.';
```

After determining principal payments for the underlying mortgage collateral, you can generate cash flows for a sequential CMO, with or without a Z-bond, by using `cmoseqcf`. For a PAC or TAC CMO, the cash flows are generated using `cmoschedcf`

Define CMO Tranches

Define CMO tranche; for example, define a CMO with 2 tranches:

```
TranchePrincipals = [500000; 500000];
TrancheCoupons = [0.06; 0.06];
```

If Using a PAC or TAC CMO, Calculate Principal Schedule

Calculate the PAC/TAC principal balance schedule based on a band of PSA speeds. For scheduled CMOs (PAC/TAC), the CMO cash flow functions additionally take in the principal balance schedule calculated by the CMO schedule function `cmosched`.

```
speed = [100 300];
[balanceSchedule, initialBalance] = cmosched(principal, coupon,...
terms, terms, speed, TranchePrincipals(1));
```

Calculate Cash Flows for Each Tranche

You can reuse the output from the cash flow generation functions to further divide the cash flows into tranches. For example, the output from `cmoschedcf` for a PAC tranche can be divided into sequential tranches by passing the principal cash flows of the PAC tranche into the `cmoschedcf` function. The output of the CMO cash flow functions are the principal and interest cash flows, as well as the principal balance.

```
[principal_balances, principal_cashflows, interest_cashflows] = cmoschedcf(principal_payments,...
TranchePrincipals, TrancheCoupons, balanceSchedule);
```

Analyze CMO by Computing Price, Yield, and Spread of CMO Cash Flows

The outputs from the CMO functions (`cmoseqcf` and `cmoschedcf`) are cash flows. The functions used to analyze a CMO are based on these cash flows. To that end, you can use `cfbyzero`, `cfspread`, `cfyield`, and `cfprice` to compute prices, yield, and spreads for the CMO cash flows. In addition, using the following, you can calculate a weighted average life (WAL) for each tranche in the CMO:

$$WAL = \sum_{i=1}^{n} \frac{P_i}{P} t_i$$

where:

P is the total principal.

P_i is the principal repayment of the coupon i.

$\frac{P_i}{P}$ is the fraction of the principal repaid in coupon i.

t_i is the time in years from the start to coupon i.

See Also cmoseqcf | cmosched | cmoschedcf | mbscfamounts | mbspassthrough

Related Examples
- "Create PAC and Sequential CMO" on page 5-66
- "Fixed-Rate Mortgage Pool" on page 5-3

Concepts
- "What Are Mortgage-Backed Securities?" on page 5-2

Create PAC and Sequential CMO

This example shows how to use an underlying mortgage-backed security (MBS) pool for a 30-year fixed-rate mortgage of 6% to define a PAC bond, and then define a sequential CMO from the PAC bond. Analyze the CMO by comparing the CMO spread to a zero-rate curve for a 30-year Treasury bond and then calculate the weighted-average life (WAL) for the PAC bond.

Step 1. Define the underlying mortgage pool.

```
principal = 100000000;
grossrate = 0.06;
coupon = 0.05;
originalTerm = 360;
termRemaining = 360;
speed = 100;
delay = 14;

Settle     = datenum('1-Jan-2011');
IssueDate  = datenum('1-Jan-2011');
Maturity   = addtodate(IssueDate, 360, 'month');
```

Step 2. Calculate underlying pool cash flow.

```
[CFlowAmounts, CFlowDates, ~, ~, ~, UnitPrincipal, UnitInterest, ...
UnitPrepayment] = mbscfamounts(Settle, Maturity, IssueDate, grossrate, ...
coupon, delay, speed, []);
```

Step 3. Calculate prepayments.

```
principalPayments = UnitPrincipal * principal;
netInterest = UnitInterest * principal;
prepayments = UnitPrepayment * principal;
dates = CFlowDates' + delay;
```

Step 4. Generate a plot for underlying MBS payments.

```
area([principalPayments'+prepayments', netInterest'])
title('Underlying MBS Payments');
legend('Principal Payments (incl. Prepayments)', 'Interest Payments')
```
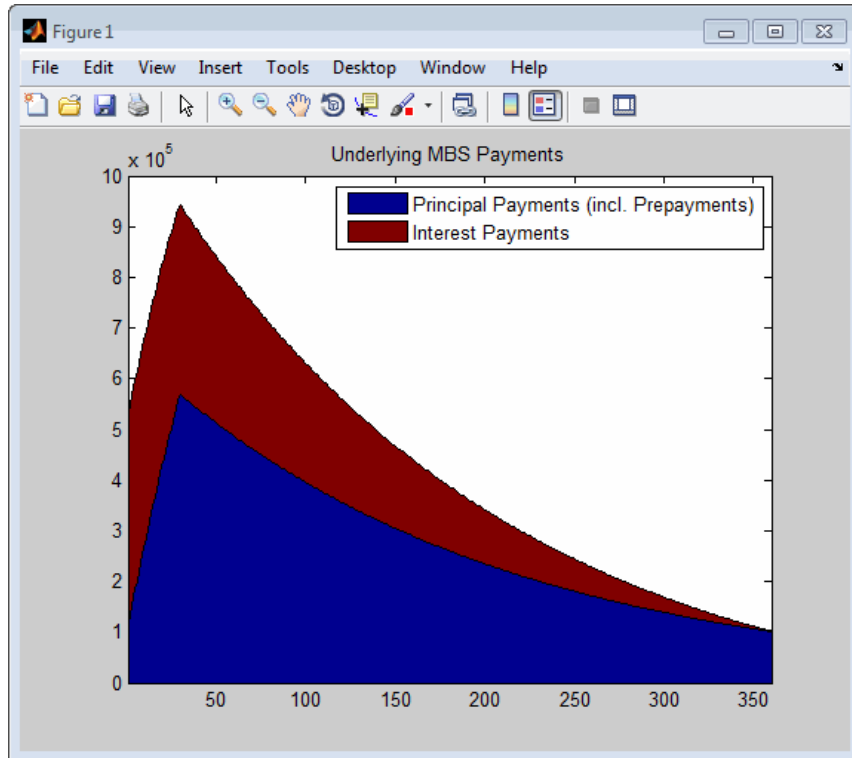
Step 5. Calculate the PAC schedule.

```
pacSpeed = [80 300];
[balanceSchedule, pacInitBalance] = ...
cmosched(principal, grossrate, originalTerm, termRemaining, ...
pacSpeed, []);
```

Step 6. Generate a plot for the PAC principal balance schedule.

```
figure;
area([pacInitBalance'; balanceSchedule'])
title('PAC Principal Balance Schedule');
legend('Principal Balance Schedule');
```
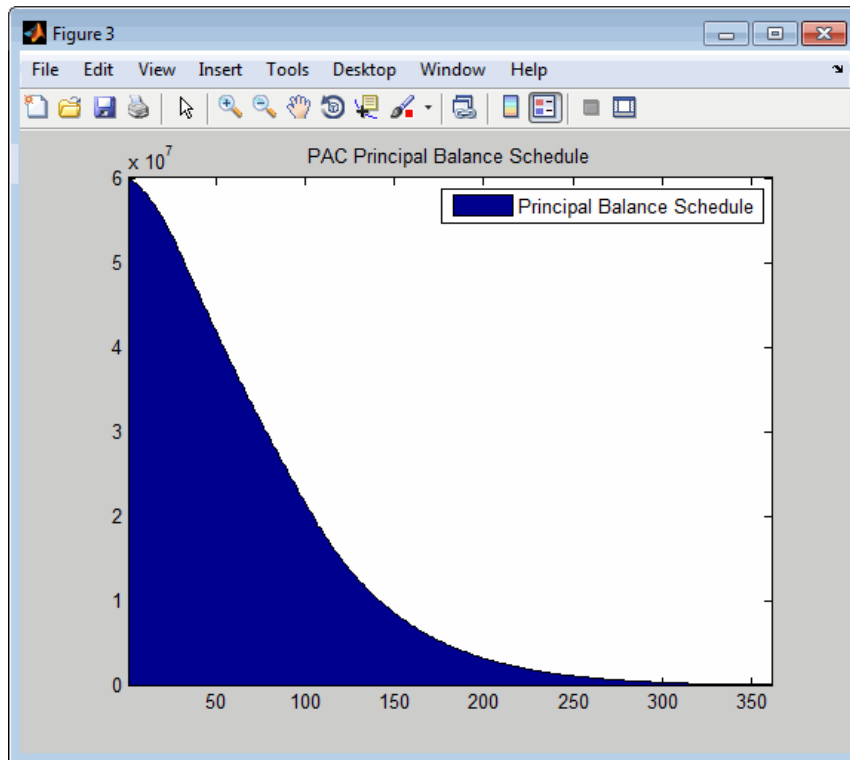
Step 7. Calculate PAC cash flow.

```
pacTranchePrincipals = [pacInitBalance; principal-pacInitBalance];
pacTrancheCoupons = [0.05; 0.05];
[pacBalances, pacPrincipals, pacInterests] = ...
cmoschedcf(principalPayments+prepayments, ...
pacTranchePrincipals, pacTrancheCoupons, balanceSchedule);
```

Step 8. Generate a plot for the PAC CMO tranches.

Generate a plot for the PAC CMO tranches:

```
figure;
area([pacPrincipals' pacInterests']);
title('PAC CMO (PAC and Support Tranches)');
legend('PAC Principal Payments', 'Support Principal Payments', ...
```

```
'PAC Interest Payments', 'Support Interest Payments');
```
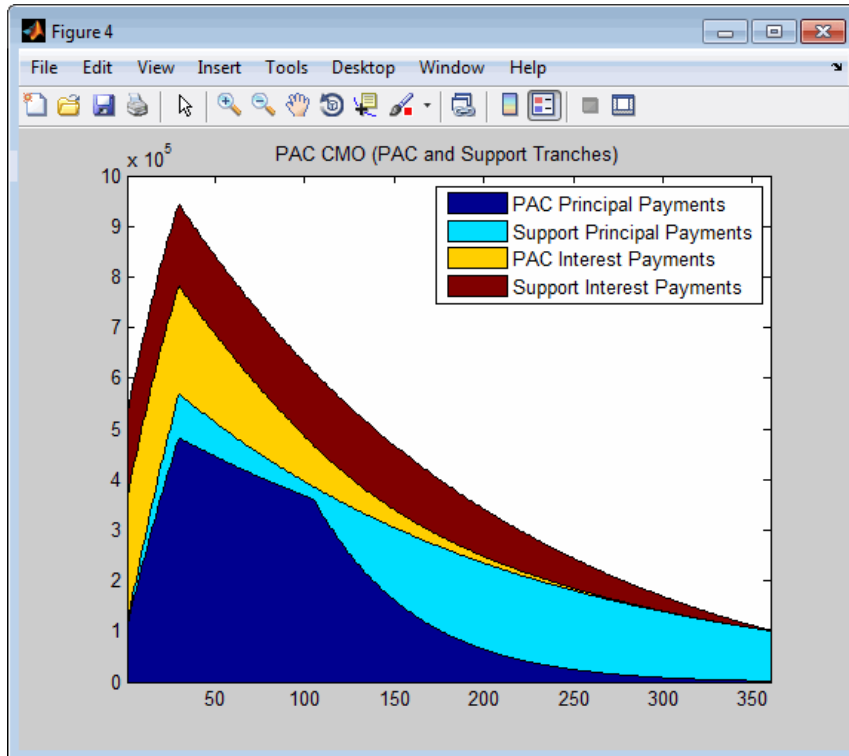

Step 9. Create sequential CMO from the PAC bond.

```
% CMO tranches, A, B, C, and D
seqTranchePrincipals = ...
[20000000; 20000000; 10000000; pacInitBalance-50000000];
seqTrancheCoupons = [0.05; 0.05; 0.05; 0.05];

seqTrancheCoupons =

    0.0500
    0.0500
    0.0500
    0.0500
```

Step 10. Calculate cash flows for each tranche.

```
[seqBalances, seqPrincipals, seqInterests] = ...
cmoseqcf(pacPrincipals(1, :), seqTranchePrincipals, ...
seqTrancheCoupons, false);
```

Step 11. Generate a plot for the sequential PAC CMO.

Generate a plot for the sequential PAC CMO:

```
figure
area([seqPrincipals' pacPrincipals(2, :)' pacInterests']);
title('Sequential PAC CMO (Sequential PAC and Support Tranches)');
legend('Sequential PAC Principals (A)', 'Sequential PAC Principals (B)', ...
'Sequential PAC Principals (C)', 'Sequential PAC Principals (D)', ...
'Support Principal Payments', 'PAC Interest Payments', ...
'Support Interest Payments');
```
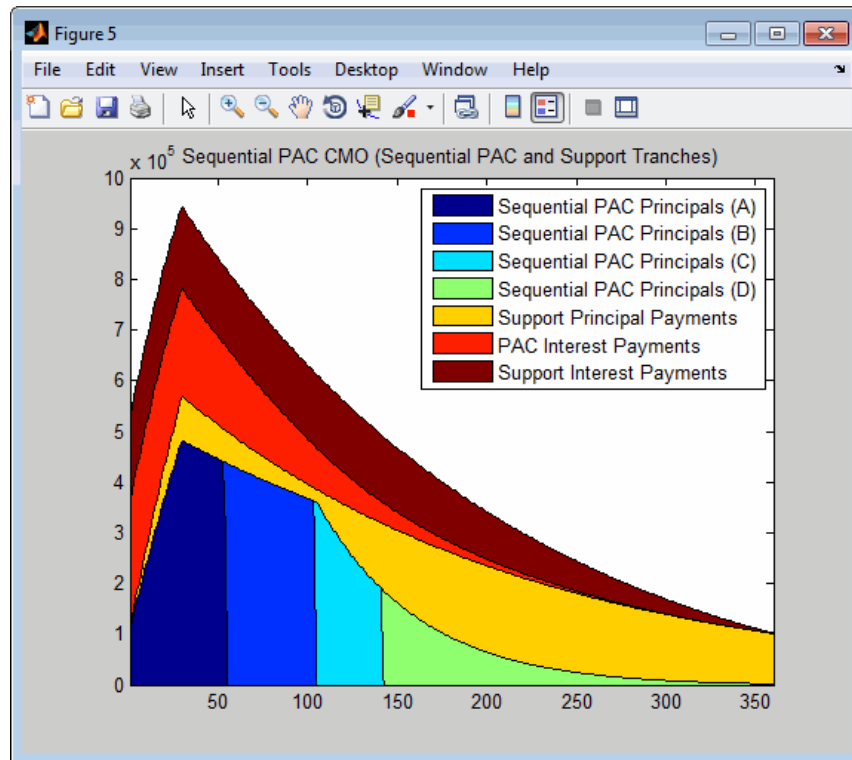
Step 12. Create the discount curve.

```
CurveSettle = datenum('1-Jan-2011');
ZeroRates = [0.01 0.03 0.10 0.19 0.45 0.81 1.76 2.50 3.18 4.09 4.38]'/100;
CurveTimes = [1/12 3/12 6/12 1 2 3 5 7 10 20 30]';
CurveDates = daysadd(CurveSettle, 360 * CurveTimes, 1);
zeroCurve = intenvset('Rates', ZeroRates, 'StartDates', CurveSettle, ...
'EndDates', CurveDates);


zeroCurve =

          FinObj: 'RateSpec'
     Compounding: 2
            Disc: [11x1 double]
           Rates: [11x1 double]
```

```
        EndTimes: [11x1 double]
      StartTimes: [11x1 double]
        EndDates: [11x1 double]
      StartDates: 734504
   ValuationDate: 734504
           Basis: 0
     EndMonthRule: 1
```

Step 13. Price the CMO cash flows.

The cash flow for the sequential PAC principal A tranche is calculated using
the cash flow functions cfbyzero, cfyield, cfprice, and cfspread.

```
cflows = seqPrincipals(1, :)+seqInterests(1, :);
cfdates = dates(2:end)';
price1 = cfbyzero(zeroCurve, cflows, cfdates, Settle, 4)
yield = cfyield(cflows, cfdates, price1, Settle, 'Basis', 4)
price2 = cfprice(cflows, cfdates, yield, Settle, 'Basis', 4)
spread = cfspread(zeroCurve, price2, cflows, cfdates, Settle, 'Basis', 4)
WAL = sum(cflows .* yearfrac(Settle, cfdates, 4)) / sum(cflows)


price1 =

   2.2109e+07


yield =

    0.0090


price2 =

   2.2109e+07


spread =

  -3.4093e-13
```

```
WAL =

    2.5408
```

The weighted average life (WAL) for the sequential PAC principal A tranche is 2.54 years.

See Also   cmoseqcf | cmosched | cmoschedcf | mbscfamounts | cfbyzero | cfyield | cfprice | cfspread | cfbyzero

Related Examples
- "Fixed-Rate Mortgage Pool" on page 5-3

Concepts   • "Using Collateralized Mortgage Obligations (CMOs)" on page 5-51

6

Debt Instruments

Agency Option-Adjusted Spreads

Often bonds are issued with embedded options, which then makes standard price/yield or spread measures irrelevant. For example, a municipality concerned about the chance that interest rates may fall in the future might issue bonds with a provision that allows the bond to be repaid before the bond's maturity. This is a call option on the bond and must be incorporated into the valuation of the bond. Option-adjusted spread (OAS), which adjusts a bond spread for the value of the option, is the standard measure for valuing bonds with embedded options. Financial Instruments Toolbox software supports computing option-adjusted spreads for bonds with single embedded options using the agency model.

The Securities Industry and Financial Markets Association (SIFMA) has a simplified approach to compute OAS for agency issues (Government Sponsored Entities like Fannie Mae and Freddie Mac) termed "Agency OAS". In this approach, the bond has only one call date (European call) and uses Black's model (a variation on Black Scholes, `http://en.wikipedia.org/wiki/Black_model`) to value the bond option. The price of the bond is computed as follows:

$$Price_{Callable} = Price_{NonCallable} - Price_{Option}$$

where

$Price_{Callable}$ is the price of the callable bond.

$Price_{NonCallable}$ is the price of the noncallable bond, i.e., price of the bond using `bndspread`.

$Price_{Option}$ is the price of the option, i.e., price of the option using Black's model.

The Agency OAS is the spread, when used in the previous formula, yields the market price. Financial Instruments Toolbox software supports these functions:

(Continued)

Agency OAS Functions	Purpose
agencyoas	Compute the OAS of the callable bond using the Agency OAS model.
agencyprice	Price the callable bond OAS using Agency using the OAS model.

Computing the Agency OAS for Bonds

To compute the Agency OAS using agencyoas, you must provide the zero curve as the input ZeroData. You can specify the zero curve in any intervals and with any compounding method. You can do this using Financial Toolbox™ functions zbtprice and zbtyield. Or, you can use IRDataCurve to construct an IRDataCurve object, and then use the getZeroRates to convert to dates and data for use in the ZeroData input.

After creating the ZeroData input for agencyoas, you can then:

1 Assign parameters for CouponRate, Settle, Maturity, Vol, CallDate, and Price.

2 Compute the option-adjusted spread using agencyoas to derive the OAS output.

   If you have the Agency OAS for the callable bond, you can use the OAS value as an input to agencyprice to determine the price for a callable bond.

In the following example, the Agency OAS is computed using agencyoas for a range of bond prices and the spread of an identically priced noncallable bond is calculated using bndspread.

```
%% Data
% Bond data -- note that there is only 1 call date
Settle = datenum('20-Jan-2010');
Maturity = datenum('30-Dec-2013');
Coupon = .022;
Vol = .5117;
```

```
CallDate = datenum('30-Dec-2010');
Period = 2;
Basis = 1;
Face = 100;

% Zero Curve data
ZeroTime = [.25 .5 1 2 3 4 5 7 10 20 30]';
ZeroDates = daysadd(Settle,360*ZeroTime,1);
ZeroRates = [.0008 .0017 .0045 .0102 .0169 .0224 .0274 .0347 .0414 .0530 .0740]';
ZeroData = [ZeroDates ZeroRates];
CurveCompounding = 2;
CurveBasis = 1;

Price = 94:104;
OAS = agencyoas(ZeroData, Price', Coupon, Settle,Maturity, Vol, CallDate,'Basis',Basis)
Spread = bndspread(ZeroData, Price', Coupon, Settle, Maturity)
plot(OAS,Price)
hold on
plot(Spread,Price,'r')
xlabel('Spread (bp)')
ylabel('Price')
title('AOAS and Spread for an Agency and Equivalent Noncallable Bond')
legend({'Callable Issue','Noncallable Issue'})

OAS =

  163.4942
  133.7306
  103.8735
   73.7505
   43.1094
   11.5608
  -21.5412
  -57.3869
  -98.5675
 -152.5226
 -239.6462


Spread =
```

```
   168.1412
   139.7047
   111.6123
    83.8561
    56.4286
    29.3227
     2.5314
   -23.9523
   -50.1348
   -76.0226
  -101.6218
```

The following plot demonstrates as the price increases, the value of the embedded option in the Agency issue increases, and the value of the issue itself does not increase as much as it would for a noncallable bond, illustrating the negative convexity of this issue:
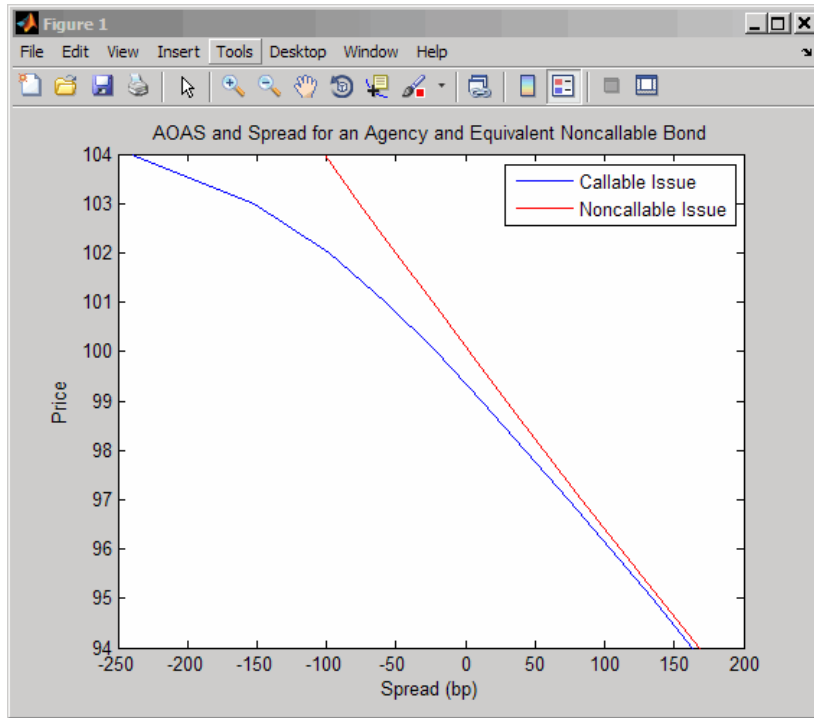
Using Zero-Coupon Bonds

In this section...

Introduction

A zero-coupon bond is a corporate, Treasury, or municipal debt instrument that pays no periodic interest. Typically, the bond is redeemed at maturity for its full face value. It will be a security issued at a discount from its face value, or it may be a coupon bond stripped of its coupons and repackaged as a zero-coupon bond.

Financial Instruments Toolbox software provides functions for valuing zero-coupon debt instruments. These functions supplement existing coupon bond functions such as bndprice and bndyield that are available in Financial Toolbox software.

Measuring Zero-Coupon Bond Function Quality

Zero-coupon function quality is measured by how consistent the results are with coupon-bearing bonds. Because the zero coupon's yield is bond-equivalent, comparisons with coupon-bearing bonds are possible.

In the textbook case, where time (*t*) is measured continuously and the rate (*r*) is continuously compounded, the value of a zero bond is the principal multiplied by e^{-rt}. In reality, the rate quoted is continuous and the basis can be variable, requiring a more consistent approach to meet the stricter demands of accurate pricing.

The following two examples

- "Pricing Treasury Notes" on page 6-8
- "Pricing Corporate Bonds" on page 6-10

show how the zero functions are consistent with supported coupon bond functions.

Pricing Treasury Notes

A Treasury note can be considered to be a package of zeros. The toolbox functions that price zeros require a coupon bond equivalent yield. That yield can originate from any type of coupon paying bond, with any periodic payment, or any accrual basis. The next example shows the use of the toolbox to price a Treasury note and compares the calculated price with the actual price quotation for that day.

```
Settle = datenum('02-03-2003');
MaturityCpn = datenum('05-15-2009');
Period = 2;
Basis = 0;

% Quoted yield.
QYield = 0.03342;

% Quoted price.
QPriceACT = 112.127;

CouponRate = 0.055;
```

Extract the cash flow and compute price from the sum of zeros discounted.

```
[CFlows, CDates] = cfamounts(CouponRate, Settle, MaturityCpn, ...
Period, Basis);
MaturityofZeros = CDates;
```

Compute the price of the coupon bond identically as a collection of zeros by multiplying the discount factors to the corresponding cash flows.

```
PriceofZeros = CFlows * zeroprice(QYield, Settle, ...
MaturityofZeros, Period, Basis)/100;
```

The following table shows the intermediate calculations.

Cash Flows	Discount Factors	Discounted Cash Flows
-1.2155	1.0000	-1.2155
2.7500	0.9908	2.7246
2.7500	0.9745	2.6799
2.7500	0.9585	2.6359
2.7500	0.9427	2.5925
2.7500	0.9272	2.5499
2.7500	0.9120	2.5080
2.7500	0.8970	2.4668
2.7500	0.8823	2.4263
2.7500	0.8678	2.3864
2.7500	0.8535	2.3472
2.7500	0.8395	2.3086
2.7500	0.8257	2.2706
102.7500	0.8121	83.4451
	Total	112.1263

Compare the quoted price and the calculated price based on zeros.

```
[QPriceACT PriceofZeros]

ans =

112.1270    112.1263
```

This example shows that `zeroprice` can satisfactorily price a Treasury note, a semiannual actual/actual basis bond, as if it were a composed of a series of zero-coupon bonds.

Pricing Corporate Bonds

You can similarly price a corporate bond, for which there is no corresponding zero-coupon bond, as opposed to a Treasury note, for which corresponding zeros exist. You can create a synthetic zero-coupon bond and arrive at the quoted coupon-bond price when you later sum the zeros.

```
Settle = datenum('02-05-2003');
MaturityCpn = datenum('01-14-2009');
Period = 2;
Basis = 1;
% Quoted yield.
QYield = 0.05974;
% Quoted price.
QPrice30 = 99.382;
CouponRate = 0.05850;
```

Extract cash flow and compute price from the sum of zeros.

```
[CFlows, CDates] = cfamounts(CouponRate, Settle, MaturityCpn, ...
Period, Basis);

Maturity = CDates;
```

Compute the price of the coupon bond identically as a collection of zeros by multiplying the discount factors to the corresponding cash flows.

```
Price30 = CFlows * zeroprice(QYield, Settle, Maturity, Period, ...
Basis)/100;
```

Compare quoted price and calculated price based on zeros.

```
[QPrice30 Price30]

ans =

99.3820    99.3828
```

As a test of fidelity, intentionally giving the wrong basis, say actual/actual (`Basis = 0`) instead of 30/360, gives a price of 99.3972. Such a systematic error, if recurring in a more complex pricing routine, quickly adds up to large inaccuracies.

In summary, the zero functions in MATLAB software facilitate extraction of present value from virtually any fixed-coupon instrument, up to any period in time.

Stepped-Coupon Bonds

In this section...
"Introduction" on page 6-12
"Cash Flows from Stepped-Coupon Bonds" on page 6-12
"Price and Yield of Stepped-Coupon Bonds" on page 6-14

Introduction

A stepped-coupon bond has a fixed schedule of changing coupon amounts. Like fixed coupon bonds, stepped-coupon bonds could have different periodic payments and accrual bases.

The functions stepcpnprice and stepcpnyield compute prices and yields of such bonds. An accompanying function stepcpncfamounts produces the cash flow schedules pertaining to these bonds.

Cash Flows from Stepped-Coupon Bonds

Consider a bond that has a schedule of two coupons. Suppose the bond starts out with a 2% coupon that steps up to 4% in 2 years and onward to maturity. Assume that the issue and settlement dates are both March 15, 2003. The bond has a 5 year maturity. Use stepcpncfamounts to generate the cash flow schedule and times.

```
Settle      = datenum('15-Mar-2003');
Maturity    = datenum('15-Mar-2008');
ConvDates   = [datenum('15-Mar-2005')];
CouponRates = [0.02, 0.04];

[CFlows, CDates, CTimes] = stepcpncfamounts(Settle, Maturity, ...
ConvDates, CouponRates)
```

Notably, ConvDates has 1 less element than CouponRates because MATLAB software assumes that the first element of CouponRates indicates the coupon schedule between Settle (March 15, 2003) and the first element of ConvDates (March 15, 2005), shown diagrammatically below.

	Pay 2% from March 15, 2003		Pay 4% from March 15, 2003
Effective 2% on March 15, 2003		Effective 4% on March 15, 2005	

Coupon Dates	Semiannual Coupon Payment
15-Mar-03	0
15-Sep-03	1
15-Mar-04	1
15-Sep-04	1
15-Mar-05	1
15-Sep-05	2
15-Mar-06	2
15-Sep-06	2
15-Mar-07	2
15-Sep-07	2
15-Mar-08	102

The payment on March 15, 2005 is still a 2% coupon. Payment of the 4% coupon starts with the next payment, September 15, 2005. March 15, 2005 is the end of first coupon schedule, not to be confused with the beginning of the second.

In summary, MATLAB takes user input as the end dates of coupon schedules and computes the next coupon dates automatically.

The payment due on settlement (zero in this case) represents the accrued interest due on that day. It is negative if such amount is nonzero. Comparison with `cfamounts` in Financial Toolbox shows that the two functions operate identically.

Price and Yield of Stepped-Coupon Bonds

The toolbox provides two basic analytical functions to compute price and yield for stepped-coupon bonds. Using the above bond as an example, you can compute the price when the yield is known.

You can estimate the yield to maturity as a number-of-year weighted average of coupon rates. For this bond, the estimated yield is:

$$\frac{(2 \times 2) + (4 \times 3)}{5}$$

.

or 3.33%. While definitely not exact (due to nonlinear relation of price and yield), this estimate suggests close to par valuation and serves as a quick first check on the function.

```
Yield = 0.0333;

[Price, AccruedInterest] = stepcpnprice(Yield, Settle, ...
Maturity, ConvDates, CouponRates)
```

The price returned is 99.2237 (per $100 notional), and the accrued interest is zero, consistent with our earlier assertions.

To validate that there is consistency among the stepped-coupon functions, you can use the above price and see if indeed it implies a 3.33% yield by using stepcpnyield.

```
YTM = stepcpnyield(Price, Settle, Maturity, ConvDates, ...
CouponRates)

YTM =

    0.0333
```

Term Structure Calculations

In this section...

Introduction

So far, a more formal definition of "yield" and its application has not been developed. In many situations when cash flow is available, discounting factors to the cash flows may not be immediately apparent. In other cases, what is relevant is often a *spread*, the difference between curves (also known as the term structure of spread).

All these calculations require one main ingredient, the Treasury spot, par-yield, or forward curve. Typically, the generation of these curves starts with a series of on-the-run and selected off-the-run issues as inputs.

MATLAB software uses these bonds to find spot rates one at a time, from the shortest maturity onwards, using bootstrap techniques. All cash flows are used to construct the spot curve, and rates between maturities (for these coupons) are interpolated linearly.

Computing Spot and Forward Curves

For an illustration of how this works, observe the use of `zbtyield` (or equivalently `zbtprice`) on a portfolio of six Treasury bills and bonds.

Bills	Maturity Date	Current Yield
3 month	4/17/03	1.15
6 month	7/17/03	1.18

Notes/Bonds	Coupon	Maturity Date	Current Yield
2 year	1.750	12/31/04	1.68
5 year	3.000	11/15/07	2.97
10 year	4.000	11/15/12	4.01
30 year	5.375	2/15/31	4.92

You can specify prices or yields to the bonds above to infer the spot curve. The function zbtyield accepts yields (bond-equivalent yield, to be exact).

To proceed, first assemble the above table into a variable called Bonds. The first column contains maturities, the second contains coupons, and the third contains notionals or face values of the bonds. (Note that bills have zero coupons.)

```
Bonds = [datenum('04/17/2003')    0        100;
         datenum('07/17/2003')    0        100;
         datenum('12/31/2004')    0.0175   100;
         datenum('11/15/2007')    0.03     100;
         datenum('11/15/2012')    0.04     100;
         datenum('02/15/2031')    0.05375  100];
```

Then specify the corresponding yields.

```
Yields  = [0.0115;
           0.0118;
           0.0168;
           0.0297;
           0.0401;
           0.0492];
```

You are now ready to compute the spot curve for each of these six maturities. The spot curve is based upon a settlement date of January 17, 2003.

```
Settle = datenum('17-Jan-2003');
[ZeroRates, CurveDates] = zbtyield(Bonds, Yields, Settle)
```

This gets you the Treasury spot curve for the day.

You can compute the forward curve from this spot curve with `zero2fwd`.

```
[ForwardRates, CurveDates] = zero2fwd(ZeroRates, CurveDates, ...
Settle)
```

Here the notion of forward rates refers to rates between the maturity dates shown above, not to a certain period (forward 3 month rates, for example).
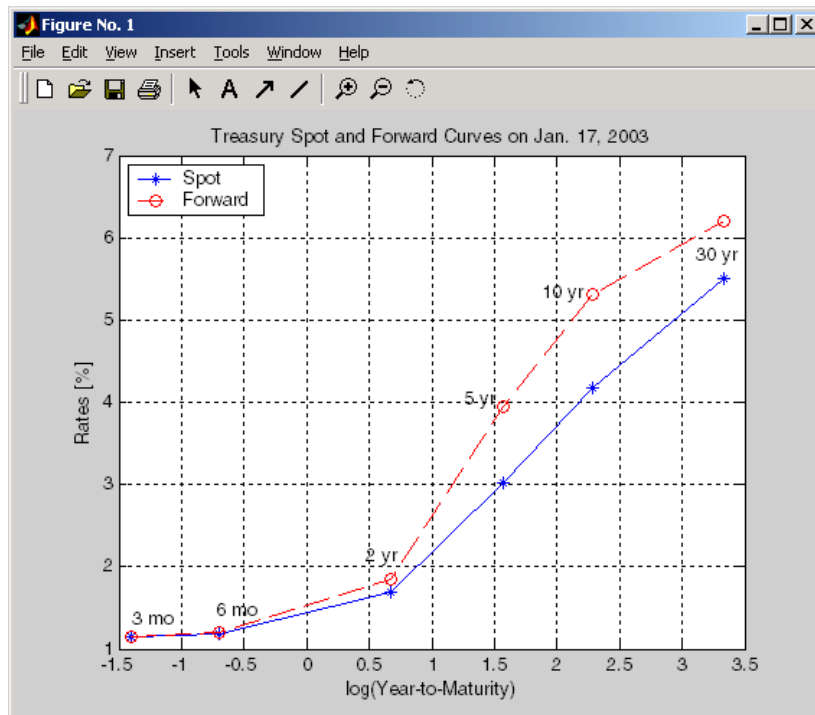
Computing Spreads

Calculating the spread between specific, fixed forward periods (such as the Treasury-Eurodollar spread) requires an extra step. Interpolate the zero rates (or zero prices, instead) for the corresponding maturities on the interval dates. Then use the interpolated zero rates to deduce the forward rates, and thus the spread of Eurodollar forward curve segments versus the relevant forward segments from Treasury bills.

Additionally, the variety of curve functions (including `zero2fwd`) helps to standardize such calculations. For instance, by making both rates quoted with quarterly compounding and on an actual/360 basis, the resulting spread structure is fully comparable. This avoids the small inconsistency that occurs when directly comparing the bond-equivalent yield of a Treasury bill to the quarterly forward rates implied by Eurodollar futures.

Noise in Curve Computations

When introducing more bonds in constructing curves, noise may become a factor and may need some "smoothing" (with splines, for example); this helps obtain a smoother forward curve.

The following spot and forward curves are constructed from 67 Treasury bonds. The fitted and bootstrapped spot curve (bottom right figure) displays comparable stability. The forward curve (upper-left figure) contains significant noise and shows an improbable forward rate structure. The noise is not necessarily bad; it could uncover trading opportunities for a relative-value approach. Yet, a more balanced approach is desired when the bootstrapped forward curve oscillates this much and contains a negative rate as large as -10% (not shown in the plot because it is outside the limits).
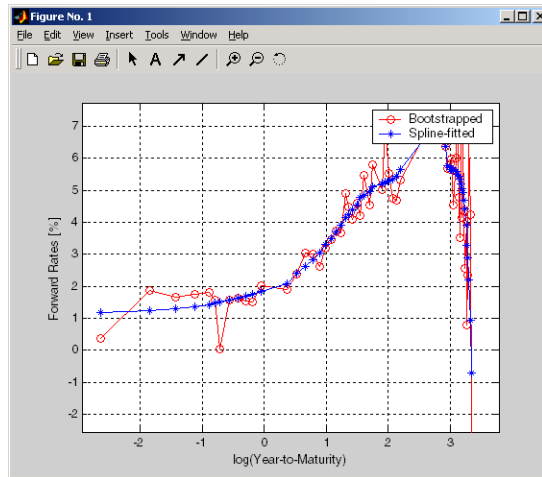
**Implied Forward Curves.
The jagged curve comes
from direct bootstrapping.
The smooth curve shows
the effect of smoothing
with splines.**
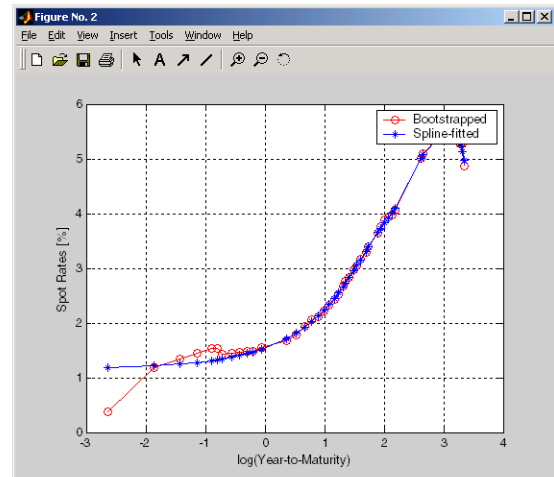
**Implied Spot Rate Curves.
These curves correspond to
the forward curve above.**

This example uses `termfit`, an example function from Financial Toolbox
software that also requires the use of Curve Fitting Toolbox™ software.

7

Derivative Securities

Interest Rate Swaps

Swap Pricing Assumptions

Financial Instruments Toolbox software contains the function `liborfloat2fixed`, which computes a fixed-rate par yield that equates the floating-rate side of a swap to the fixed-rate side. The solver sets the present value of the fixed side to the present value of the floating side without having to line up and compare fixed and floating periods.

Assumptions on Floating-Rate Input

- Rates are quarterly, for example, that of Eurodollar futures.

- Effective date is the first third Wednesday after the settlement date.

- All delivery dates are spaced 3 months apart.

- All periods start on the third Wednesday of delivery months.

- All periods end on the same dates of delivery months, 3 months after the start dates.

- Accrual basis of floating rates is actual/360.

- Applicable forward rates are estimated by interpolation in months when forward-rate data is not available.

Assumptions on Fixed-Rate Output

- Design allows you to create a bond of any coupon, basis, or frequency, based upon the floating-rate input.

- The start date is a valuation date, that is, a date when an agreement to enter into a contract by the settlement date is made.

- Settlement can be on or after the start date. If it is after, a forward fixed-rate contract results.

- Effective date is assumed to be the first third Wednesday after settlement, the same date as that of the floating rate.

- The end date of the bond is a designated number of years away, on the same day and month as the effective date.

- Coupon payments occur on anniversary dates. The frequency is determined by the period of the bond.

- Fixed rates are not interpolated. A fixed-rate bond of the same present value as that of the floating-rate payments is created.

Swap Pricing Example

This example shows the use of the functions in computing the fixed rate applicable to a series of 2-, 5-, and 10-year swaps based on Eurodollar market data. According to the Chicago Mercantile Exchange (`http://www.cmegroup.com`), Eurodollar data on Friday, October 11, 2002, was as shown in the following table.

Note This example illustrates swap calculations in MATLAB software. Timing of the data set used was not rigorously examined and was assumed to be the proxy for the swap rate reported on October 11, 2002.

Eurodollar Data on Friday, October 11, 2002

Month	Year	Settle
10	2002	98.21
11	2002	98.26
12	2002	98.3
1	2003	98.3
2	2003	98.31
3	2003	98.275
6	2003	98.12

Eurodollar Data on Friday, October 11, 2002 (Continued)

Month	Year	Settle
9	2003	97.87
12	2003	97.575
3	2004	97.26
6	2004	96.98
9	2004	96.745
12	2004	96.515
3	2005	96.33
6	2005	96.135
9	2005	95.955
12	2005	95.78
3	2006	95.63
6	2006	95.465
9	2006	95.315
12	2006	95.16
3	2007	95.025
6	2007	94.88
9	2007	94.74
12	2007	94.595
3	2008	94.48
6	2008	94.375
9	2008	94.28
12	2008	94.185
3	2009	94.1
6	2009	94.005
9	2009	93.925
12	2009	93.865

Eurodollar Data on Friday, October 11, 2002 (Continued)

Month	Year	Settle
3	2010	93.82
6	2010	93.755
9	2010	93.7
12	2010	93.645
3	2011	93.61
6	2011	93.56
9	2011	93.515
12	2011	93.47
3	2012	93.445
6	2012	93.41
9	2012	93.39

Using this data, you can compute 1-, 2-, 3-, 4-, 5-, 7-, and 10-year swap rates with the toolbox function `liborfloat2fixed`. The function requires you to input only Eurodollar data, the settlement date, and tenor of the swap. MATLAB software then performs the required computations.

To illustrate how this function works, first load the data contained in the supplied Excel® worksheet `EDdata.xls`.

```
[EDRawData, textdata] = xlsread('EDdata.xls');
```

Extract the month from the first column and the year from the second column. The rate used as proxy is the arithmetic average of rates on opening and closing.

```
Month = EDRawData(:,1);
Year  = EDRawData(:,2);
IMMData = (EDRawData(:,4)+EDRawData(:,6))/2;
EDFutData = [Month, Year, IMMData];
```

Next, input the current date.

```
Settle = datenum('11-Oct-2002');
```

To compute for the 2 year swap rate, set the tenor to 2.

```
Tenor = 2;
```

Finally, compute the swap rate with `liborfloat2fixed`.

```
[FixedSpec, ForwardDates, ForwardRates] = ...
liborfloat2fixed(EDFutData, Settle, Tenor)
```

MATLAB returns a par-swap rate of 2.23% using the default setting (quarterly compounding and 30/360 accrual), and forward dates and rates data (quarterly compounded).

```
FixedSpec =

      Coupon: 0.0223
      Settle: '16-Oct-2002'
    Maturity: '16-Oct-2004'
      Period: 4
       Basis: 1

ForwardDates =

      731505
      731596
      731687
      731778
      731869
      731967
      732058
      732149

ForwardRates =

    0.0178
    0.0168
    0.0171
    0.0189
```

```
     0.0216
     0.0250
     0.0280
     0.0306
```

In the `FixedSpec` output, note that the swap rate actually goes forward from the third Wednesday of October 2002 (October 16, 2002), 5 days after the original `Settle` input (October 11, 2002). This, however, is still the best proxy for the swap rate on `Settle`, as the assumption merely starts the swap's effective period and does not affect its valuation method or its length.

The correction suggested by Hull and White improves the result by turning on convexity adjustment as part of the input to `liborfloat2fixed`. (See Hull, J., *Options, Futures, and Other Derivatives*, 4th Edition, Prentice-Hall, 2000.) For a long swap, for example, 5 years or more, this correction could prove to be large.

The adjustment requires additional parameters:

- `StartDate`, which you make the same as `Settle` (the default) by providing an empty matrix `[]` as input.

- `ConvexAdj` to tell `liborfloat2fixed` to perform the adjustment.

- `RateParam`, which provides the parameters `a` and `S` as input to the Hull-White short rate process.

- Optional parameters `InArrears` and `Sigma`, for which you can use empty matrices `[]` to accept the MATLAB defaults.

- `FixedCompound`, with which you can facilitate comparison with values cited in Table H15 of *Federal Reserve Statistical Release* by turning the default quarterly compounding into semiannual compounding, with the (default) basis of 30/360.

```
StartDate = [];
Interpolation = [];
ConvexAdj = 1;
RateParam = [0.03; 0.017];
FixedCompound = 2;
[FixedSpec, ForwardDaates, ForwardRates] = ...
liborfloat2fixed(EDFutData, Settle, Tenor, StartDate, ...
```

```
Interpolation, ConvexAdj, RateParam, [], [], FixedCompound)
```

This returns 2.21% as the 2-year swap rate, quite close to the reported swap rate for that date.

Analogously, the following table summarizes the solutions for 1-, 3-, 5-, 7-, and 10-year swap rates (convexity-adjusted and unadjusted).

Calculated and Market Average Data of Swap Rates on Friday, October 11, 2002

Swap Length (Years)	Unadjusted	Adjusted	Table H15	Adjusted Error (Basis Points)
1	1.80%	1.79%	1.80%	-1
2	2.24%	2.21%	2.22%	-1
3	2.70%	2.66%	2.66%	0
4	3.12%	3.03%	3.04%	-1
5	3.50%	3.37%	3.36%	+1
7	4.16%	3.92%	3.89%	+3
10	4.87%	4.42%	4.39%	+3

Portfolio Hedging

You can use these results further, such as for hedging a portfolio. The `liborduration` function provides a duration-hedging capability. You can isolate assets (or liabilities) from interest-rate risk exposure with a swap arrangement.

Suppose you own a bond with these characteristics:

- $100 million face value
- 7% coupon paid semiannually
- 5% yield to maturity
- Settlement on October 11, 2002

- Maturity on January 15, 2010

- Interest accruing on an actual/365 basis

Use of the bnddury function from Financial Toolbox software shows a modified duration of 5.6806 years.

To immunize this asset, you can enter into a pay-fixed swap, specifically a swap in the amount of notional principal (*Ns*) such that *Ns*SwapDuration* + \$100M*5.6806 = 0 (or *Ns* = -100*5.6806/*SwapDuration*).

Suppose again, you choose to use a 5-, 7-, or 10-year swap (3.37%, 3.92%, and 4.42% from the previous table) as your hedging tool.

```
SwapFixRate = [0.0337; 0.0392; 0.0442];
Tenor = [5; 7; 10];
Settle = '11-Oct-2002';
PayFixDuration = liborduration(SwapFixRate, Tenor, Settle)
```

This gives a duration of -3.6835, -4.7307, and -6.0661 years for 5-, 7-, and 10-year swaps. The corresponding notional amount is computed by

```
Ns = -100*5.6806./PayFixDuration

Ns =

  154.2163
  120.0786
   93.6443
```

The notional amount entered in pay-fixed side of the swap instantaneously immunizes the portfolio.

Convertible Bond Valuation

A convertible bond (CB) is a debt instrument that can be converted into a predetermined amount of a company's equity at certain times before the bond's maturity.

Financial Instruments Toolbox software uses a binomial and trinomial tree approach (cbprice) to value convertible bonds. The value of a convertible bond is determined by the uncertainty of the related stock. Once the stock evolution is modeled, backward discounting is computed.

The last column of such trees provides the data to decide which is more profitable: the debt notional (plus interest, if any) or the equivalent number of shares per the notional.

Where debt prevails, the toolbox discounts backward with the risk-free rate plus the spread reflecting the credit risk of the issuer. Where stock prevails, the toolbox discounts with the risk-free rate. The intrinsic value of a convertible bond is the sum of the (probability-adjusted) debt and stock portions from the last node. This is compared to current stock price, to see if voluntary or forced conversion may take place. Otherwise, its value is the intrinsic value. From here, the same discounting process is repeated after adjusting debt portion to be equal to 0 if any conversion takes place. Dividends and coupons are handled discretely, at the date they occur.

The approach is equivalent to solving a one-dimensional partial differential equation such as one described by Tsiveriotis and Fernandes. (See Tsiveriotis, K. and C. Fernandes (1998), "Valuing Convertible Bonds with Credit Risk," *The Journal of Fixed Income*, 8 (3), 95-102.) Using the same example of bond specifications that they use (4% annual coupon, payable twice a year, callable after 2 years at 110, and redeemable at par in 5 years), the toolbox gives results like theirs.
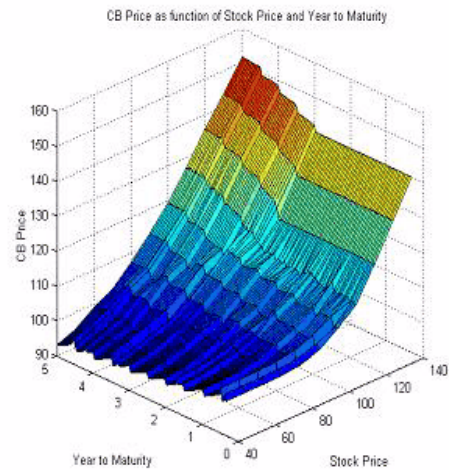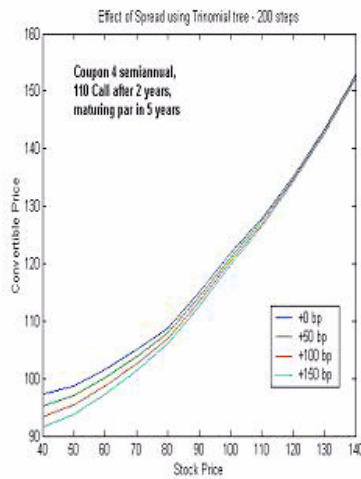
The figure on the left shows the bond "floor" of the convertible (a 5% yield, given a 4% coupon at about 97% par) when share prices are low.

The change of curvature located at the end of the second year is due to the activation of the embedded (soft) call feature (visible on the surface plot in the right figure).

Finally, there is the flat section when time is nearing expiration and share prices are high, indicating a delta of unity, a characteristic of in-the-money equity options embedded in a bond.

Bond Futures

In this section...

Supported Bond Futures

Bond futures are futures contracts where the commodity for delivery is a government bond. There are established global markets for government bond futures. Bond futures provide a liquid alternative for managing interest-rate risk.

In the U.S. market, the Chicago Mercantile Exchange (CME) offers futures on Treasury bonds and notes with maturities of 2, 5, 10, and 30 years. Typically, the following bond future contracts from the CME have maturities of 3, 6, 9, and 12 months:

- 30-year U.S. Treasury bond
- 10-year U.S. Treasury bond
- 5-year U.S. Treasury bond
- 2-year U.S. Treasury bond

The short position in a Treasury bond or note future contract must deliver to the long position in one of many possible existing Treasury bonds. For example, in a 30-year Treasury bond future, the short position must deliver a Treasury bond with at least 15 years to maturity. Because these bonds have different values, the bond future contract is standardized by computing a conversion factor. The conversion factor normalizes the price of a bond to a theoretical bond with a coupon of 6%. The price of a bond future contract is represented as:

$$InvoicePrice = FutPrice \times CF + AI$$

where:

FutPrice is the price of the bond future.

CF is the conversion factor for a bond to deliver in a futures contract.

AI is the accrued interest.

You can reference these conversion factors at U.S. Treasury Bond Futures Contract. The short position in a futures contract has the option of which bond to deliver and, in the U.S. bond market, when in the delivery month to deliver the bond. The short position typically chooses to deliver the bond known as the Cheapest to Deliver (CTD). The CTD bond most often delivers on the last delivery day of the month.

Financial Instruments Toolbox software supports the following bond futures:

- U.S. Treasury bonds and notes
- German Bobl, Bund, Buxl, and Schatz
- UK gilts
- Japanese government bonds (JGBs)

The functions supporting all bond futures are:

Function	Purpose
convfactor	Calculates bond conversion factors for U.S. Treasury bonds, German Bobl, Bund, Buxl, and Schatz, U.K. gilts, and JGBs.
bndfutprice	Prices bond future given repo rates.
bndfutimprepo	Calculates implied repo rates for a bond future given price.

The functions supporting U.S. Treasury bond futures are:

Function	Purpose
tfutbyprice	Calculates future prices of Treasury bonds given the spot price.
tfutbyyield	Calculates future prices of Treasury bonds given current yield.
tfutimprepo	Calculates implied repo rates for the Treasury bond future given price.
tfutpricebyrepo	Calculates Treasury bond futures price given the implied repo rates.
tfutyieldbyrepo	Calculates Treasury bond futures yield given the implied repo rates.

Analysis of Bond Futures

The following example demonstrates analyzing German Euro-Bund futures traded on Eurex. However, convfactor, bndfutprice, and bndfutimprepo apply to bond futures in the U.S., U.K., Germany, and Japan. The workflow for this analysis is:

1 Calculate bond conversion factors.

2 Calculate implied repo rates to find the CTD bond.

3 Price the bond future using the term implied repo rate.

Calculating Bond Conversion Factors

Use conversion factors to normalize the price of a particular bond for delivery in a futures contract. When using conversion factors, the assumption is that a bond for delivery has a 6% coupon. Use convfactor to calculate conversion factors for all bond futures from the U.S., Germany, Japan, and U.K.

For example, conversion factors for Euro-Bund futures on Eurex are listed at www.eurexchange.com. The delivery date for Euro-Bund futures is the 10th day of the month, as opposed to bond futures in the U.S., where the short position has the option of choosing when to deliver the bond.

For the 4% bond, compute the conversion factor with:

```
CF1 = convfactor('10-Sep-2009','04-Jul-2018', .04,.06,3)

CF1 =

    0.8659
```

This syntax for `convfactor` works fine for bonds with standard coupon periods. However, some deliverable bonds have long or short first coupon periods. Compute the conversion factors for such bonds using the optional input parameters `StartDate` and `FirstCouponDate`. Specify all optional input arguments for `convfactor` as parameter/value pairs:

```
CF2 = convfactor('10-Sep-2009','04-Jan-2019', .0375,'Convention',3,'startdate',...
datenum('14-Nov-2008'))

CF2 =

    0.8426
```

Calculating Implied Repo Rates to Find the CTD Bond

To determine the availability of the cheapest bond for deliverable bonds against a futures contract, compute the implied repo rate for each bond. The bond with the highest repo rate is the cheapest because it has the lowest initial value, thus yielding a higher return, provided you deliver it with the stated futures price. Use `bndfutimprepo` to calculate repo rates:

```
% Bond Properties
CouponRate = [.0425;.0375;.035];
Maturity = [datenum('04-Jul-2018');datenum('04-Jan-2019');datenum('04-Jul-2019')];
CF = [0.882668;0.842556;0.818193];
Price = [105.00;100.89;98.69];

% Futures Properties
FutSettle = '09-Jun-2009';
FutPrice = 118.54;
Delivery = '10-Sep-2009';

% Note that the default for BNDFUTIMPREPO is for the bonds to be
% semi-annual with a day count basis of 0.  Since these are German
% bonds, we need to have a Basis of 8 and a Period of 1
```

```
ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle, Delivery, CF, ...
CouponRate, Maturity,'Basis',8,'Period',1)


ImpRepo =

    0.0261
   -0.0022
   -0.0315
```

Pricing Bond Futures Using the Term Implied Repo Rate

Use bndfutprice to perform price calculations for all bond futures from the
U.S., Germany, Japan, and U.K. To price the bond, given a term repo rate:

```
% Assume a term repo rate of .0091;
RepoRate = .0091;
[FutPrice,AccrInt] = bndfutprice(RepoRate, Price(1), FutSettle,...
Delivery, CF(1), CouponRate(1), Maturity(1),...
'Basis',8,'Period',1)

FutPrice =

  118.0126


AccrInt =

    0.7918
```

Managing Present Value with Bond Futures

The Present Value of a Basis Point (PVBP) is used to manage interest-rate
risk. PVBP is a measure that quantifies the change in price of a bond given
a one-basis point shift in interest rates. The PVBP of a bond is computed
with the following:

$$PVBP_{Bond} = \frac{Duration \times MarketValue}{100}$$

The PVBP of a bond futures contract can be computed with the following:

$$PVBP_{Futures} = \frac{PVBP_{CTDBond}}{CTDConversionFactor}$$

Use `bnddurp` and `bnddury` from Financial Toolbox software to compute the modified durations of CTD bonds. For more information, see "Managing Interest Rate Risk with Bond Futures" on page 7-18 and "Fitting the Diebold Li Model" on page 7-27.

Managing Interest Rate Risk with Bond Futures

This example shows how to hedge the interest rate risk of a portfolio using bond futures.

Modifying the Duration of a Portfolio with Bond Futures

In managing a bond portfolio, you can use a benchmark portfolio to evaluate performance. Sometimes a manager is constrained to keep the portfolio's duration within a particular band of the duration of the benchmark. One way to modify the duration of the portfolio is to buy and sell bonds, however, there may be reasons why a portfolio manager wishes to maintain the existing composition of the portfolio (e.g., the current holdings reflect fundamental research/views about future returns). Therefore, another option for modifying the duration is to buy and sell bond futures.

Bond futures are futures contracts where the commodity to be delivered is a government bond that meets the standard outlined in the futures contract (e.g., the bond has a specified remaining time to maturity).

Since often many bonds are available, and each bond may have a different coupon, you can use a conversion factor to normalize the payment by the long to the short.

There exist well developed markets for government bond futures. Specifically, the Chicago Board of Trade offers futures on the following:

- 2 Year Note
- 3 Year Note
- 5 Year Note
- 10 Year Note
- 30 Year Bond

http://www.cmegroup.com/trading/interest-rates/

Eurex offers futures on the following:

- Euro-Schatz Futures 1.75 to 2.25

- Euro-Bobl Futures 4.5 to 5.5

- Euro-Bund Futures 8.5 to 10.5

- Euro-Buxl Futures 24.0 to 35

http://www.eurexchange.com

Bond futures can be used to modify the duration of a portfolio. Since bond futures derive their value from the underlying instrument, the duration of a bond futures contract is related to the duration of the underlying bond.

There are two challenges in computing this duration:

- Since there are many available bonds for delivery, the short in the contract has a choice in which bond to deliver.

- Some contracts allow the short flexibility in choosing the delivery date.

Typically, the bond used for analysis is the bond that is cheapest for the short to deliver (CTD).

One approach is to compute duration measures using the CTD's duration and the conversion factor.

For example, the Present Value of a Basis Point (PVBP) can be computed from the following:

$$PVBP_{Futures} = \frac{PVBP_{CTD}}{ConversionFactor_{CTD}}$$

$$PVBP_{CTD} = \frac{Duration_{CTD} * Price_{CTD}}{100}$$

Note that these definitions of duration for the futures contract are approximate, and do not account for the value of the delivery options for the short.

If the goal is to modify the duration of a portfolio, use the following:

$$NumContracts = \frac{(Dur_{Target} - Dur_{Initial}) * Value_{Portfolio}}{Dur_{CTD} * Price_{CTD} * ContractSize} * ConvFactor_{CTD}$$

Note that the contract size is typically for 100,000 face value of a bond -- so
the contract size is typically 1000, as the bond face value is 100.

The following example assumes an initial duration, portfolio value, and target
duration for a portfolio with exposure to the Euro interest rate. The June
Euro-Bund Futures contract is used to modify the duration of the portfolio.

Note that typically futures contracts are offered for March, June, September
and December.

```
% Assume the following for the portfolio and target
PortfolioDuration = 6.4;
PortfolioValue = 100000000;
BenchmarkDuration = 4.8;

% Deliverable Bunds -- note that these conversion factors may also be
% computed with the MATLAB(R) function CONVFACTOR
BondPrice = [106.46;108.67;104.30];
BondMaturity = datenum({'04-Jan-2018','04-Jul-2018','04-Jan-2019'});
BondCoupon = [.04;.0425;.0375];
ConversionFactor = [.868688;.880218;.839275];

% Futures data -- found from http://www.eurexchange.com
FuturesPrice = 122.17;
FuturesSettle = '23-Apr-2009';
FuturesDelivery = '10-Jun-2009';

% To find the CTD bond we can compute the implied repo rate
ImpliedRepo = bndfutimprepo(BondPrice,FuturesPrice,FuturesSettle,...
    FuturesDelivery,ConversionFactor,BondCoupon,BondMaturity);

% Note that the bond with the highest implied repo rate is the CTD
[CTDImpRepo,CTDIndex] = max(ImpliedRepo);

% Compute the CTD's Duration -- note the period and basis for German Bunds
Duration = bnddurp(BondPrice,BondCoupon,FuturesSettle,BondMaturity,1,8);
```

```
ContractSize = 1000;

% Use the formula above to compute the number of contracts to sell
NumContracts = (BenchmarkDuration - PortfolioDuration)*PortfolioValue./...
    (BondPrice(CTDIndex)*ContractSize*Duration(CTDIndex))*ConversionFactor(

disp(['To achieve the target duration, ' num2str(abs(round(NumContracts)))
    ' Euro-Bund Futures must be sold.'])
```

To achieve the target duration, 180 Euro-Bund Futures must be sold.

Modifying the Key Rate Durations of a Portfolio with Bond Futures

One of the shortcomings of using duration as a risk measure is that it assumes parallel shifts in the yield curve. While many studies have shown that this explains roughly 85% of the movement in the yield curve, changes in the slope or shape of the yield curve are not captured by duration -- and therefore, hedging strategies are not successful at addressing these dynamics.

One approach is to use key rate duration -- this is particularly relevant when using bond futures with multiple maturities, like Treasury futures.

The following example uses 2, 5, 10 and 30 year Treasury Bond futures to hedge the key rate duration of a portfolio.

Computing key rate durations requires a zero curve. This example uses the zero curve published by the Treasury and found at the following location:

http://www.ustreas.gov/offices/domestic-finance/debt-management/interest-rate/yield.shtm

Note that this zero curve could also be derived using the Interest Rate Curve functionality found in IRDataCurve and IRFunctionCurve.

```
% Assume the following for the portfolio and target, where the duration
% vectors are key rate durations at 2, 5, 10, and 30 years.
PortfolioDuration = [.5 1 2 6];
PortfolioValue = 100000000;
BenchmarkDuration = [.4 .8 1.6 5];

% The following are the CTD Bonds for the 30, 10, 5 and 2 year futures
```

```
% contracts -- these were determined using the procedure outlined in the
% previous section.
CTDCoupon = [4.75 3.125 5.125 7.5]'/100;
CTDMaturity = datenum({'3/31/2011','08/31/2013','05/15/2016','11/15/2024'})
CTDConversion = [0.9794 0.8953 0.9519 1.1484]';
CTDPrice = [107.34 105.91 117.00 144.18]';

ZeroRates = [0.07 0.10 0.31 0.50 0.99 1.38 1.96 2.56 3.03 3.99 3.89]'/100;
ZeroDates = daysadd(FuturesSettle,[30 360 360*2 360*3 360*5 ...
    360*7 360*10 360*15 360*20 360*25 360*30],1);

% Compute the key rate durations for each of the CTD bonds.
CTDKRD = bndkrdur([ZeroDates ZeroRates], CTDCoupon,FuturesSettle,...
    CTDMaturity,'KeyRates',[2 5 10 30]);

% Note that the contract size for the 2 Year Note Future is $200,000
ContractSize = [2000;1000;1000;1000];

NumContracts = (bsxfun(@times,CTDPrice.*ContractSize./CTDConversion,CTDKRD)
    (BenchmarkDuration - PortfolioDuration)'*PortfolioValue;

sprintf(['To achieve the target duration, \n' ...
    num2str(-round(NumContracts(1))) ' 2 Year Treasury Note Futures must be
    num2str(-round(NumContracts(2))) ' 5 Year Treasury Note Futures must be
    num2str(-round(NumContracts(3))) ' 10 Year Treasury Note Futures must b
    num2str(-round(NumContracts(4))) ' Treasury Bond Futures must be sold,


ans =

To achieve the target duration,
24 2 Year Treasury Note Futures must be sold,
47 5 Year Treasury Note Futures must be sold,
68 10 Year Treasury Note Futures must be sold,
120 Treasury Bond Futures must be sold,
```

Improving the Performance of a Hedge with Regression

An additional component to consider in hedging interest rate risk with bond futures, again related to movements in the yield curve, is that typically the yield curve moves more at the short end than at the long end.

Therefore, if a position is hedged with a future where the CTD bond has a maturity that is different than the portfolio this could lead to a situation where the hedge under- or over-compensates for the actual interest rate risk of the portfolio.

One approach is to perform a regression on historical yields at different maturities to determine a Yield Beta, which is a value that represents how much more the yield changes for different maturities.

This example shows how to use this approach with UK Long Gilt futures and historical data on Gilt Yields.

Market data on Gilt futures is found at the following:

http://www.euronext.com

Historical data on gilts is found at the following;

http://www.dmo.gov.uk

Note that while this approach does offer the possibility of improving the performance of a hedge, any analysis using historical data depends on historical relationships remaining consistent.

Also note that an additional enhancement takes into consideration the correlation between different maturities. While this approach is outside the scope of this example, you can use this to implement a minimum variance hedge.

```
% Assume the following for the portfolio and target
PortfolioDuration = 6.4;
PortfolioValue = 100000000;
BenchmarkDuration = 4.8;

% This is the CTD Bond for the Long Gilt Futures contract
CTDBondPrice = 113.40;
```

```
CTDBondMaturity = datenum('7-Mar-2018');
CTDBondCoupon = .05;
CTDConversionFactor = 0.9325024;

% Market data for the Long Gilt Futures contract
FuturesPrice = 120.80;
FuturesSettle = '23-Apr-2009';
FuturesDelivery = '10-Jun-2009';

CTDDuration = bnddurp(CTDBondPrice,CTDBondCoupon,FuturesSettle,CTDBondMatur

ContractSize = 1000;

NumContracts = (BenchmarkDuration - PortfolioDuration)*PortfolioValue./...
    (CTDBondPrice*ContractSize*CTDDuration)*CTDConversionFactor;

disp(['To achieve the target duration with a conventional hedge ' ...
    num2str(-round(NumContracts)) ...
    ' Long Gilt Futures must be sold.'])
```

To achieve the target duration with a conventional hedge 182 Long Gilt Futu

To improve the accuracy of this hedge, historical data is used to determine
a relationship between the standard deviation of the yields. Specifically,
standard deviation of yields is plotted and regressed vs bond duration. This
relationship is then used to compute a Yield Beta for the hedge.

```
% Load data from XLS spreadsheet
load ukbonddata_20072008

Duration = bnddury(Yield(1,:)',Coupon,Dates(1,:),Maturity);

scatter(Duration,100*std(Yield))
title('Standard Deviation of Yields for UK Gilts 2007-2008')
ylabel('Standard Deviation of Yields (%)')
xlabel('Duration')
annotation(gcf,'textbox',[0.4067 0.685 0.4801 0.0989],...
    'String',{'Note that the Standard Deviation',...
    'of Yields is greater at shorter maturities.'},...
    'FitBoxToText','off',...
```

```
        'EdgeColor','none');

stats = regstats(std(Yield),Duration);
YieldBeta = (stats.beta'*[1 PortfolioDuration]')./(stats.beta'*[1 CTDDurati
```


Standard Deviation of Yields for UK Gilts 2007-2008

Note that the Standard Deviation of Yields is greater at shorter maturities.
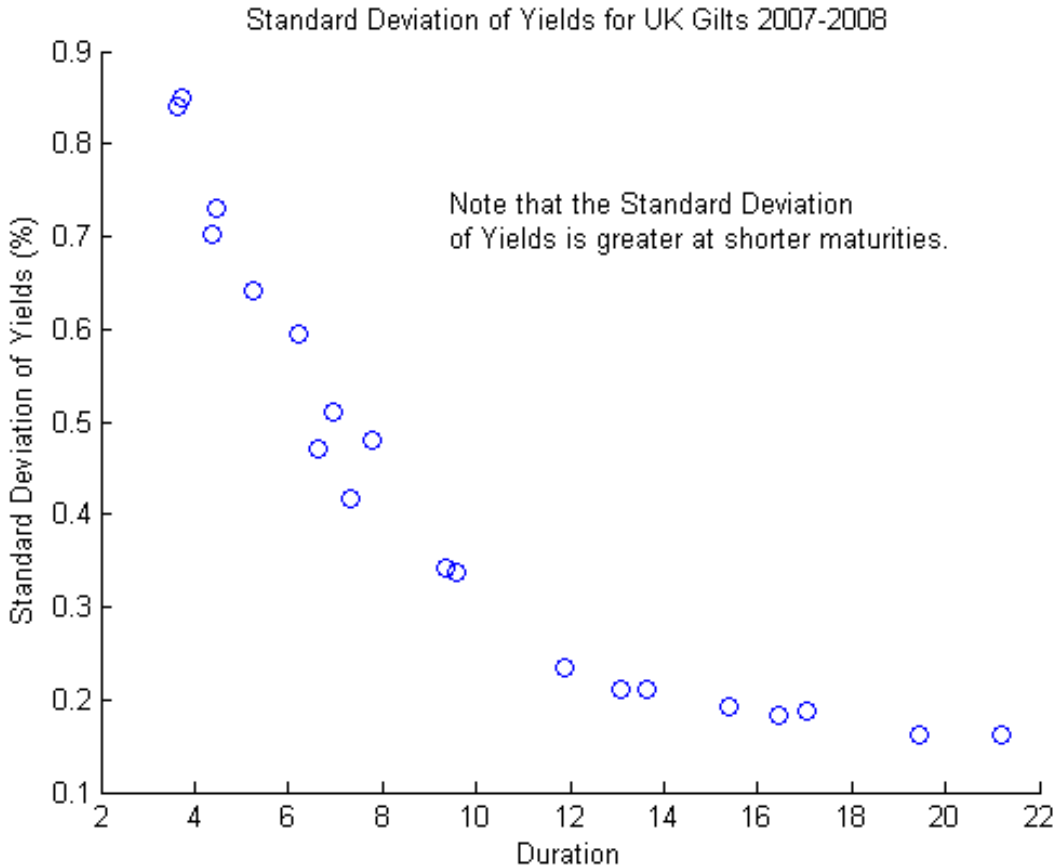
Now the Yield Beta is used to compute a new value for the number of contracts to be sold. Note that since the duration of the portfolio was less than the duration of the CTD Gilt, the number of futures to sell is actually greater than in the first case.

```
NumContracts = (BenchmarkDuration - PortfolioDuration)*PortfolioValue./...
```

```
        (CTDBondPrice*ContractSize*CTDDuration)*CTDConversionFactor*YieldBeta;

disp(['To achieve the target duration using a Yield Beta-modified hedge, '
    num2str(abs(round(NumContracts))) ...
    ' Long Gilt Futures must be sold.'])
```

```
To achieve the target duration using a Yield Beta-modified hedge, 193 Long
```

Bibliography

This example is based on the following books and papers:

[1] Burghardt, G., T. Belton, M. Lane and J. Papa. The Treasury Bond Basis. New York, NY: McGraw-Hill, 2005.

[2] Krgin, D. Handbook of Global Fixed Income Calculations. New York, NY: John Wiley & Sons, 2002.

[3] CFA Program Curriculum, Level III, Volume 4, Reading 31. CFA Institute, 2009.

Fitting the Diebold Li Model

This example shows how to construct a Diebold Li model of the US yield curve for each month from 1990 to 2010 and how to forecast future yield curves by fitting an autoregressive model to the time series of each parameter.

The paper can be found here:

http://www.ssc.upenn.edu/~fdiebold/papers/paper49/Diebold-Li.pdf

Load the Data

The data used are monthly Treasury yields from 1990 through 2010 for tenors of 1 Mo, 3 Mo, 6 Mo, 1 Yr, 2 Yr, 3 Yr, 5 Yr, 7 Yr, 10 Yr, 20 Yr, 30 Yr.

Daily data can be found here:

http://www.treasury.gov/resource-center/data-chart-center/interest-rates/Pages/TextView.

Data is stored in a MATLAB data file as a MATLAB dataset object.

```
load Data_USYieldCurve

% Extract data for the last day of each month
MonthYearMat = repmat((1990:2010)',1,12)';
EOMDates = lbusdate(MonthYearMat(:),repmat((1:12)',21,1));
MonthlyIndex = find(ismember(get(Dataset,'ObsNames'),datestr(EOMDates)));
Estimationdataset = Dataset(MonthlyIndex,:);
EstimationData = double(Estimationdataset);
```

Diebold Li Model

Diebold and Li start with the Nelson Siegel model

$$y = \beta_0 + (\beta_1 + \beta_2)\frac{\tau}{m}(1 - e^{\frac{-m}{\tau}}) - \beta_2 e^{\frac{-m}{\tau}}$$

and rewrite it to be the following:

$$y_t(\tau) = \beta_{1t} + \beta_{2t}\left(\frac{1 - e^{-\lambda_t\tau}}{\lambda_t\tau}\right) + \beta_{3t}\left(\frac{1 - e^{-\lambda_t\tau}}{\lambda_t\tau} - e^{-\lambda_t\tau}\right)$$

The above model allows the factors to be interpreted in the following way: Beta1 corresponds to the long term/level of the yield curve, Beta2 corresponds to the short term/slope, and Beta3 corresponds to the medium term/curvature.

Diebold and Li advocate setting the time factor, lambda, to maximize the loading on the medium term factor, Beta3, at 30 months. This also transforms the problem from a nonlinear fitting one to a simple linear regression.

```
% Explicitly set the time factor lambda
lambda_t = .0609;

% Construct a matrix of the factor loadings
% Tenors associated with data
TimeToMat = [3 6 9 12 24 36 60 84 120 240 360]';
X = [ones(size(TimeToMat)) (1 - exp(-lambda_t*TimeToMat))./(lambda_t*TimeTo
     ((1 - exp(-lambda_t*TimeToMat))./(lambda_t*TimeToMat) - exp(-lambda_t*T

% Plot the factor loadings
plot(TimeToMat,X)
title('Factor Loadings for Diebold Li Model with time factor of .0609')
xlabel('Maturity (months)')
ylim([0 1.1])
legend({'Beta1','Beta2','Beta3'},'location','east')
```

Factor Loadings for Diebold Li Model with time factor of .0609
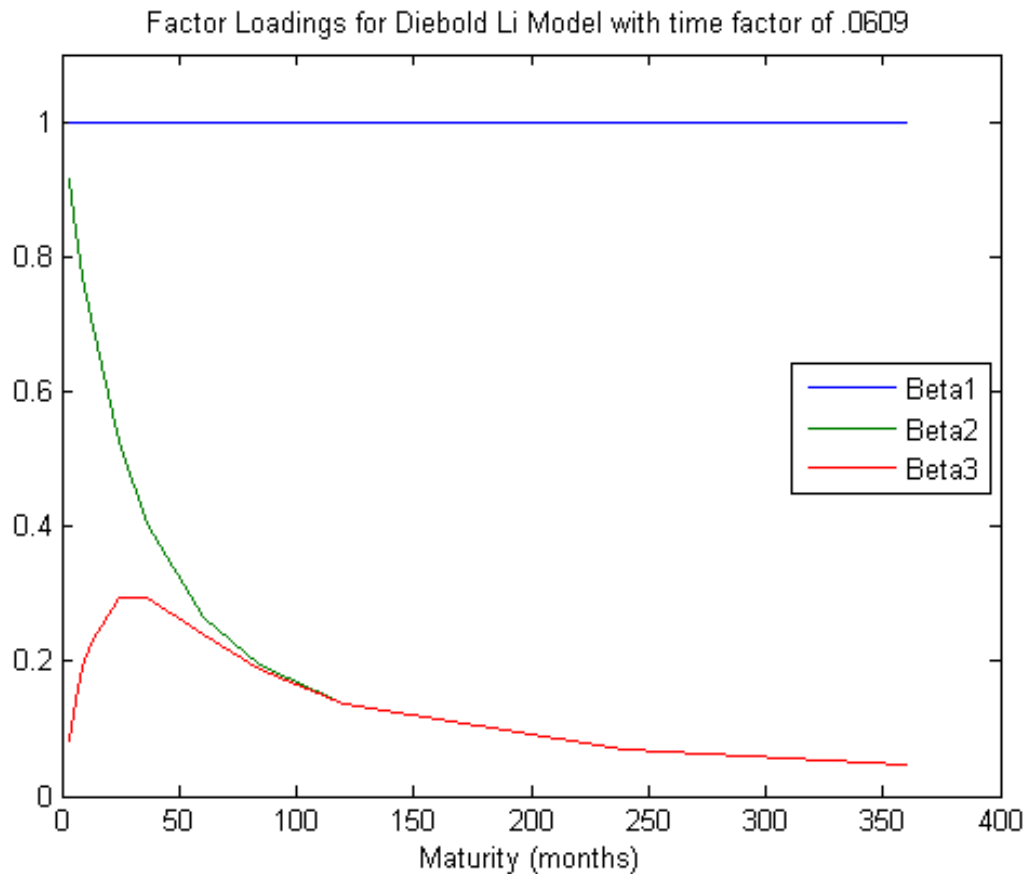
Fit the Model

A DieboldLi object has been developed to facilitate fitting the model from
yield data. The DieboldLi object inherits from the IRCurve object, so
the getZeroRates, getDiscountFactors, getParYields, getForwardRates
and toRateSpec methods are all implemented. Additionally, the method
fitYieldsFromBetas has been implemented to estimate the Beta parameters
given a lambda parameter for observed market yields.

The DieboldLi object will be used to fit a Diebold Li model for each month
from 1990 through 2010.

```
% Preallocate the Betas
Beta = zeros(size(EstimationData,1),3);

% Loop through and fit each end of month yield curve
for jdx=1:size(EstimationData,1)
    tmpCurveModel = DieboldLi.fitBetasFromYields(EOMDates(jdx),lambda_t*12,
    Beta(jdx,:) = [tmpCurveModel.Beta1 tmpCurveModel.Beta2 tmpCurveModel.Be
end
```

The Diebold Li fits on selected dates are included here

```
PlotSettles = datenum({'30-May-1997','31-Aug-1998','29-Jun-2001','31-Oct-20
figure
for jdx=1:length(PlotSettles)
    subplot(2,2,jdx)
    tmpIdx = find(strcmpi(get(Estimationdataset,'ObsNames'),datestr(PlotSet
    tmpCurveModel = DieboldLi.fitBetasFromYields(PlotSettles(jdx),lambda_t*
        daysadd(PlotSettles(jdx),30*TimeToMat),EstimationData(tmpIdx,:)');
    scatter(daysadd(PlotSettles(jdx),30*TimeToMat),EstimationData(tmpIdx,:)
    hold on
    PlottingDates = (PlotSettles(jdx)+30:30:PlotSettles(jdx)+30*360)';
    plot(PlottingDates,tmpCurveModel.getParYields(PlottingDates),'r-')
    title(['Yield Curve on ' datestr(PlotSettles(jdx))])
    datetick
end
```
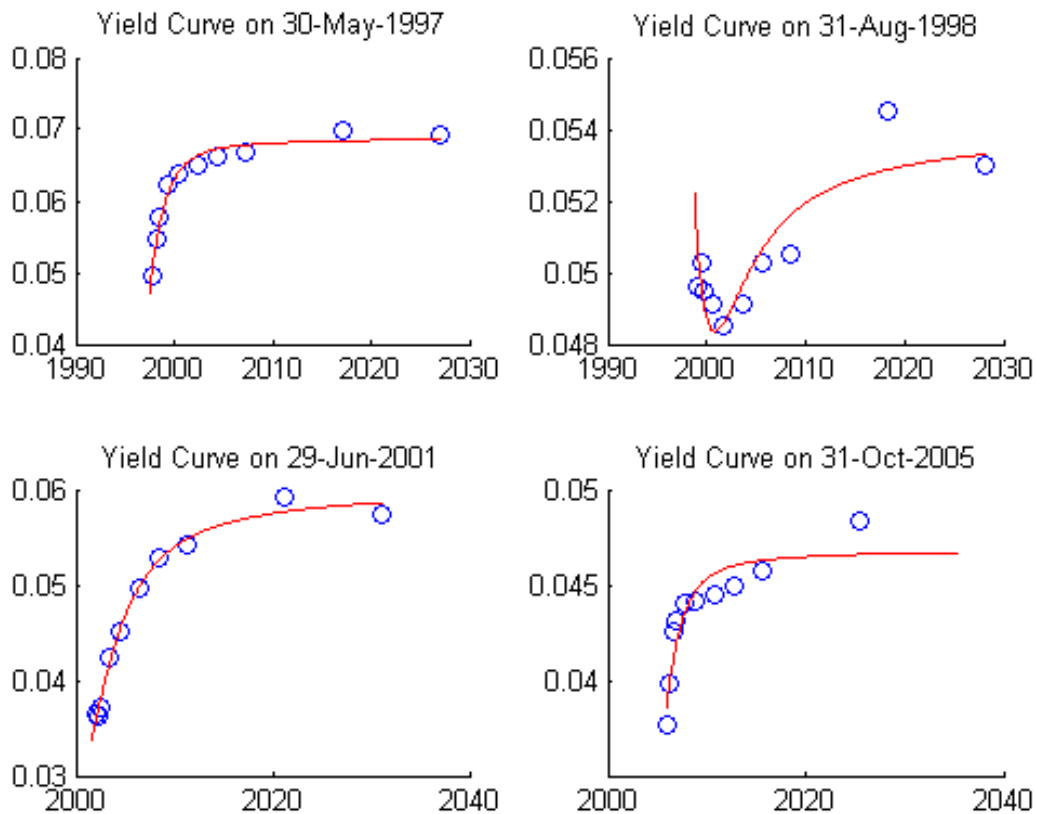
Forecasting

The Diebold Li model can be used to forecast future yield curves. Diebold and Li propose fitting an AR(1) model to the time series of each Beta parameter. This fitted model can then be used to forecast future values of each parameter, and by extension, future yield curves.

For this example the MATLAB function REGRESS is used to estimate the parameters for an AR(1) model for each Beta.

The confidence intervals for the regression fit are also used to generate two additional yield curve forecasts that serve as additional possible scenarios for the yield curve.

The MonthsLag variable can be adjusted to make different period ahead forecasts. For example, changing the value from 1 to 6 would change the forecast from a 1 month ahead to 6 month ahead forecast.

```
MonthsLag = 1;

[tmpBeta,bint] = regress(Beta(MonthsLag+1:end,1),[ones(size(Beta(MonthsLag+
ForecastBeta(1,1) = [1 Beta(end,1)]*tmpBeta;
ForecastBeta_Down(1,1) = [1 Beta(end,1)]*bint(:,1);
ForecastBeta_Up(1,1) = [1 Beta(end,1)]*bint(:,2);
[tmpBeta,bint]  = regress(Beta(MonthsLag+1:end,2),[ones(size(Beta(MonthsLag
ForecastBeta(1,2) = [1 Beta(end,2)]*tmpBeta;
ForecastBeta_Down(1,2) = [1 Beta(end,2)]*bint(:,1);
ForecastBeta_Up(1,2) = [1 Beta(end,2)]*bint(:,2);
[tmpBeta,bint]  = regress(Beta(MonthsLag+1:end,3),[ones(size(Beta(MonthsLag
ForecastBeta(1,3) = [1 Beta(end,3)]*tmpBeta;
ForecastBeta_Down(1,3) = [1 Beta(end,3)]*bint(:,1);
ForecastBeta_Up(1,3) = [1 Beta(end,3)]*bint(:,2);

% Forecasted yield curve
figure
Settle = daysadd(EOMDates(end),30*MonthsLag);
DieboldLi_Forecast = DieboldLi('ParYield',Settle,[ForecastBeta lambda_t*12]
DieboldLi_Forecast_Up = DieboldLi('ParYield',Settle,[ForecastBeta_Up lambda
DieboldLi_Forecast_Down = DieboldLi('ParYield',Settle,[ForecastBeta_Down la
PlottingDates = (Settle+30:30:Settle+30*360)';
plot(PlottingDates,DieboldLi_Forecast.getParYields(PlottingDates),'b-')
hold on
plot(PlottingDates,DieboldLi_Forecast_Up.getParYields(PlottingDates),'r-')
plot(PlottingDates,DieboldLi_Forecast_Down.getParYields(PlottingDates),'r-'
title(['Diebold Li Forecasted Yield Curves on ' datestr(EOMDates(end)) ' fo
legend({'Forecasted Curve','Additional Scenarios'},'location','southeast')
datetick
```

Diebold Li Forecasted Yield Curves on 31-Dec-2010 for 30-Jan-2011
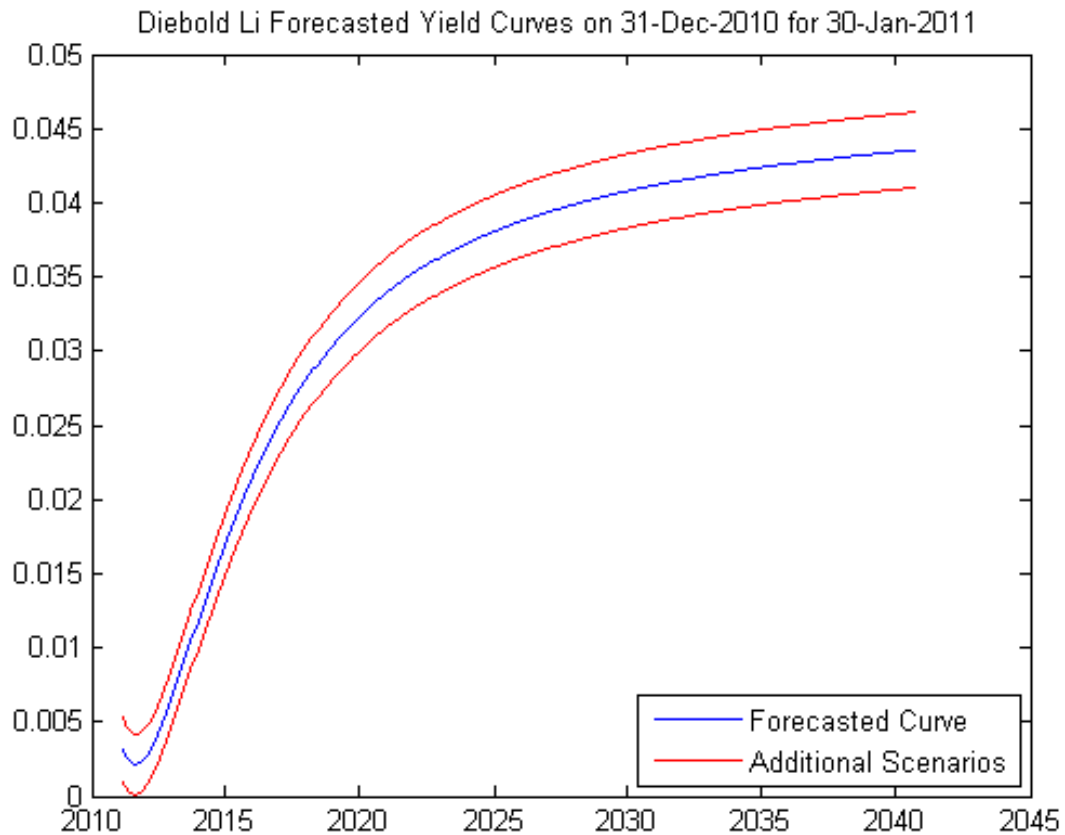
Bibliography

This example is based on the following paper:

[1] Francis X. Diebold, Canlin Li, Forecasting the term structure of government bond yields, Journal of Econometrics, Volume 130, Issue 2, February 2006, Pages 337-364

8

Credit Derivatives

Credit Default Swap (CDS)

A credit default swap (CDS) is a contract that protects against losses resulting from credit defaults. The transaction involves two parties, the protection buyer and the protection seller, and also a reference entity, usually a bond. The protection buyer pays a stream of premiums to the protection seller, who in exchange offers to compensate the buyer for the loss in the bond's value if a credit event occurs. The stream of premiums is called the premium leg, and the compensation when a credit event occurs is called the protection leg. Credit events usually include situations in which the bond issuer goes bankrupt, misses coupon payments, or enters a restructuring process. Financial Instruments Toolbox software supports:

CDS Functions

Function	Purpose
cdsbootstrap	Compute default probability parameters from CDS market quotes.
cdsspread	Compute breakeven spreads for the CDS contracts.
cdsprice	Compute the price for the CDS contracts.

Bootstrapping a Default Probability Curve

In a typical workflow, pricing a new CDS contract involves first estimating a default probability term structure using cdsbootstrap. This requires market quotes of existing CDS contracts, or quotes of CDS indices (e.g., iTraxx). The estimated default probability curve is then used as input to cdsspread or cdsprice. If the default probability information is already known, cdsbootstrap can be bypassed and cdsspread or cdsprice can be called directly.

The market information in this example is provided in the form of running spreads of CDS contracts maturing on the CDS standard payment dates closest to 1, 2, 3, 5, and 7 years from the valuation date.

```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10','20-Sep-11','20-Sep-12','20-Sep-14',...
```

```
'20-Sep-16'});
MarketSpreads = [140 175 210 265 310]';
MarketData = [MarketDates MarketSpreads];
ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
'17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle);
```

The bootstrapped default probability curve is plotted against time, in years, from the valuation date.

```
ProbTimes = yearfrac(Settle,ProbData(:,1));
figure
plot([O; ProbTimes],[O; ProbData(:,2)])
grid on
axis([O ProbTimes(end,1) O ProbData(end,2)])
xlabel('Time (years)')
ylabel('Cumulative Default Probability')
title('Bootstrapped Default Probability Curve')
```
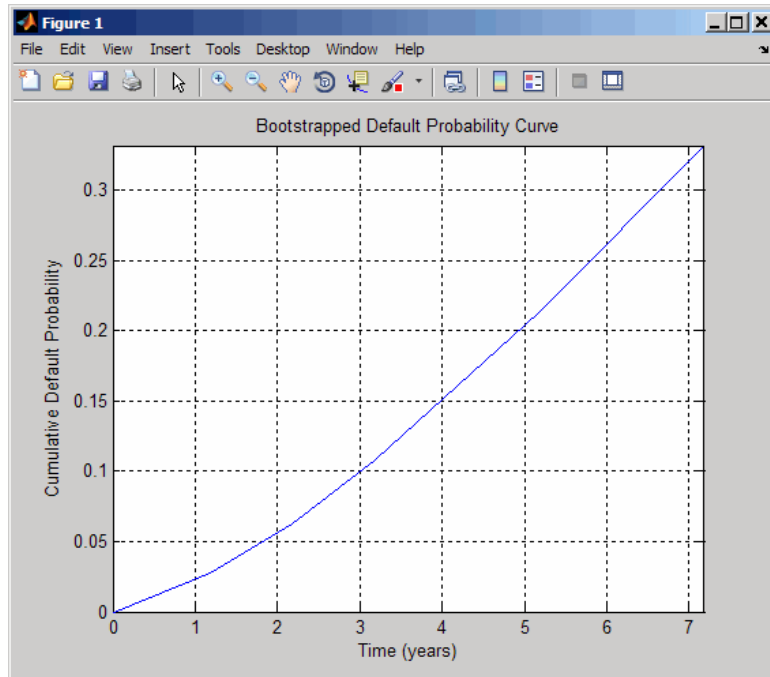
The associated hazard rates are returned as an optional output. The convention is that the first hazard rate applies from the settlement date to the first market date, the second hazard rate from the first to the second market date, etc., and the last hazard rate applies from the second-to-last market date onwards. The following plot displays the bootstrapped hazard rates, plotted against time, in years, from the valuation date:

```
HazTimes = yearfrac(Settle,HazData(:,1));
figure
stairs([0; HazTimes(1:end-1,1); HazTimes(end,1)+1],...
[HazData(:,2);HazData(end,2)])
grid on
axis([0 HazTimes(end,1)+1 0.9*HazData(1,2) 1.1*HazData(end,2)])
xlabel('Time (years)')
ylabel('Hazard Rate')
title('Bootstrapped Hazard Rates')
```
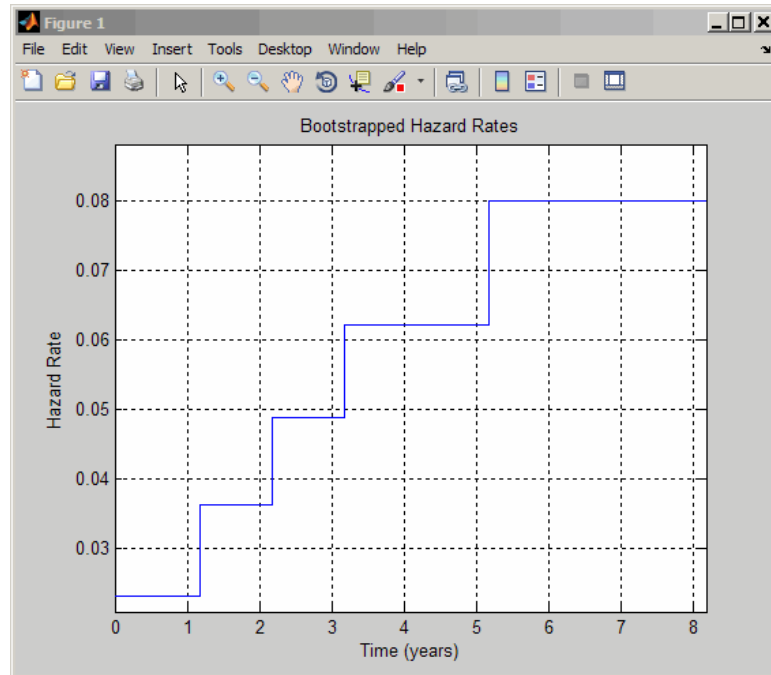
Finding Breakeven Spread for New CDS Contract

The breakeven, or running, spread is the premium a protection buyer must pay, with no upfront payments involved, to receive protection for credit events associated to a given reference entity. Spreads are expressed in basis points (bp). There is a notional amount associated to the CDS contract to determine the monetary amounts of the premium payments.

New quotes for CDS contracts can be obtained with cdsspread. After obtaining a default probability curve using cdsbootstrap, you get quotes that are consistent with current market conditions.

In this example, instead of standard CDS payment dates, define new maturity dates. Using the period from 3 and 5 years (CDS standard dates), maturities are defined within this range spaced at quarterly intervals (measuring time from the valuation date):

```
Settle = '17-Jul-2009';
```

```
MarketDates = datenum({'20-Sep-10','20-Sep-11','20-Sep-12','20-Sep-14',...
'20-Sep-16'});
MarketSpreads = [140 175 210 265 310]';
MarketData = [MarketDates MarketSpreads];
ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
'17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle);

Maturity1 = datestr(daysadd('17-Jul-09',360*(3.25:0.25:5),1));
Spread1 = cdsspread(ZeroData,ProbData,Settle,Maturity1);

figure
scatter(yearfrac(Settle,Maturity1),Spread1,'*')
hold on
scatter(yearfrac(Settle,MarketData(3:4,1)),MarketData(3:4,2))
hold off
grid on
xlabel('Time (years)')
ylabel('Spread (bp)')
title('CDS Spreads')
legend('New Quotes','Market','location','SouthEast')
```

This plot displays the resulting spreads:
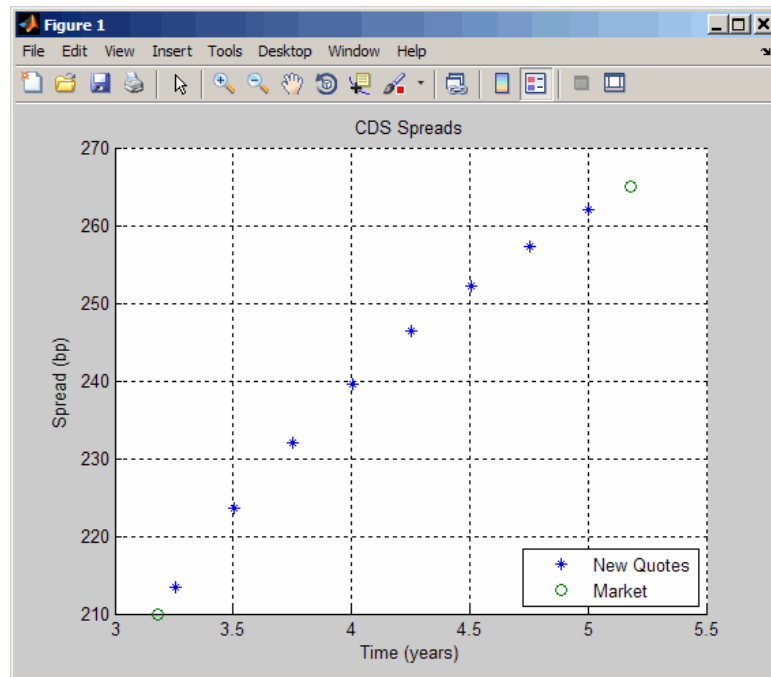
To evaluate the effect of the recovery rate parameter, instead of 40% (default value), use a recovery rate of 35%:

```
Spread1Rec35 = cdsspread(ZeroData,ProbData,Settle,Maturity1,...
'RecoveryRate',0.35);
```

```
figure
plot(yearfrac(Settle,Maturity1),Spread1,...
yearfrac(Settle,Maturity1),Spread1Rec35,'--')
grid on
xlabel('Time (years)')
ylabel('Spread (bp)')
title('CDS Spreads with Different Recovery Rates')
legend('40%','35%','location','SouthEast')
```

The resulting plot shows that smaller recovery rates produce higher premia, as expected, since in the event of default, the protection payments will be higher:

CDS Spreads with Different Recovery Rates
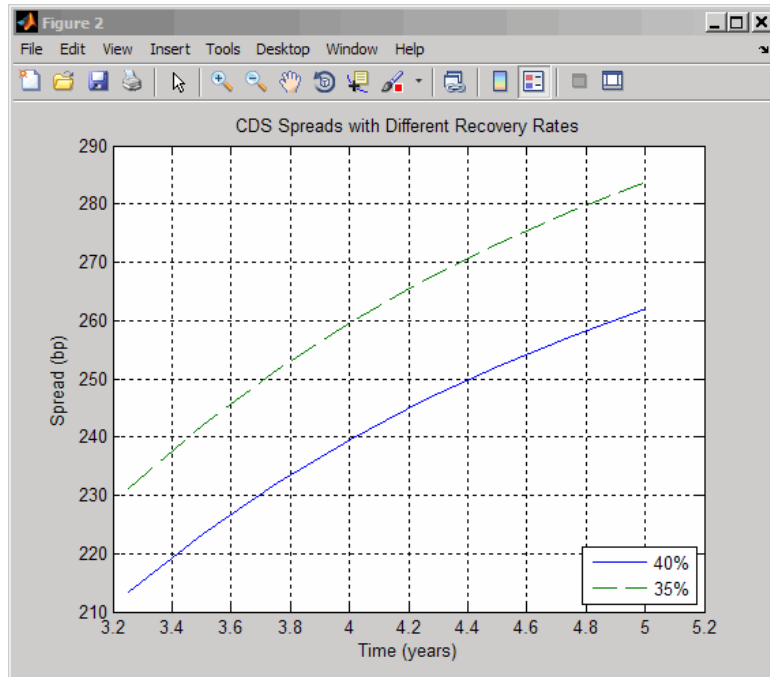
Valuing an Existing CDS Contract

The current value, or mark-to-market, of an existing CDS contract is the amount of money the contract holder would receive (if positive) or pay (if negative) to unwind this contract. The upfront of the contract is the current value expressed as a fraction of the notional amount of the contract, and it is commonly used to quote market values.

The value of existing CDS contracts is obtained with cdsprice. By default, cdsprice treats contracts as long positions. Whether a contract position is long or short is determined from a protection standpoint, that is, long means protection was bought, and short means protection was sold. In the following example, an existing CDS contract pays a premium that is lower than current market conditions. The price is positive, as expected, since it would be more costly to buy the same type of protection today.

```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10','20-Sep-11','20-Sep-12','20-Sep-14',...
```

```
'20-Sep-16'});
MarketSpreads = [140 175 210 265 310]';
MarketData = [MarketDates MarketSpreads];

ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
'17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle);

Maturity2 = '20-Sep-2012';
Spread2 = 196;

[Price,AccPrem,PaymentDates,PaymentTimes,PaymentCF] = cdsprice(ZeroData,...
ProbData,Settle,Maturity2,Spread2);

fprintf('Dirty Price: %8.2f\n',Price);
fprintf('Accrued Premium: %8.2f\n',AccPrem);
fprintf('Clean Price: %8.2f\n',Price-AccPrem);
fprintf('\nPayment Schedule:\n\n');
fprintf('Date \t\t Time Frac \t Amount\n');
for k = 1:length(PaymentDates)
   fprintf('%s \t %5.4f \t %8.2f\n',datestr(PaymentDates(k)),...
      PaymentTimes(k),PaymentCF(k));
end
```

This resulting payment schedule is:

```
Dirty Price: 41628.50
Accrued Premium: 15244.44
Clean Price: 26384.05

Payment Schedule:

Date     Time Frac    Amount
20-Sep-2009   0.1806   35388.89
20-Dec-2009   0.2528   49544.44
20-Mar-2010   0.2500   49000.00
20-Jun-2010   0.2556   50088.89
```

```
20-Sep-2010    0.2556    50088.89
20-Dec-2010    0.2528    49544.44
20-Mar-2011    0.2500    49000.00
20-Jun-2011    0.2556    50088.89
20-Sep-2011    0.2556    50088.89
20-Dec-2011    0.2528    49544.44
20-Mar-2012    0.2528    49544.44
20-Jun-2012    0.2556    50088.89
20-Sep-2012    0.2556    50088.89
```

Additionally, you can use `cdsprice` to value a portfolio of CDS contracts. In the following example, a simple hedged position with two vanilla CDS contracts, one long, one short, with slightly different spreads is priced in a single call and the value of the portfolio is the sum of the returned prices:

```
[Price2,AccPrem2] = cdsprice(ZeroData,ProbData,Settle,...
repmat(datenum(Maturity2),2,1),[Spread2;Spread2+3],...
'Notional',[1e7; -1e7]);

fprintf('Contract \t Dirty Price \t Acc Premium \t  Clean Price\n');
fprintf('    Long \t $ %9.2f \t $ %9.2f \t $ %9.2f \t\n',...
   Price2(1), AccPrem2(1), Price2(1) - AccPrem2(1));
fprintf('    Short \t $ %8.2f \t $ %8.2f \t $ %8.2f \t\n',...
   Price2(2), AccPrem2(2), Price2(2) - AccPrem2(2));
fprintf('Mark-to-market of hedged position: $ %8.2f\n',sum(Price2));
```

This resulting value of the portfolio is:

```
Contract    Dirty Price    Acc Premium     Clean Price
    Long    $  41628.50    $  15244.44    $  26384.05
    Short   $ -32708.11    $ -15477.78    $ -17230.33
Mark-to-market of hedged position: $  8920.39
```

Converting from Running to Upfront

A CDS market quote is given in terms of a standard spread (usually 100 bp or 500 bp) and an upfront payment, or in terms of an equivalent running or breakeven spread, with no upfront payment. The functions `cdsbootstrap`, `cdsspread`, and `cdsprice` perform upfront to running or running to upfront conversions.

For example, to convert the market quotes to upfront quotes for a standard spread of 100 bp:

```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10','20-Sep-11','20-Sep-12','20-Sep-14',...
'20-Sep-16'});
MarketSpreads = [140 175 210 265 310]';
MarketData = [MarketDates MarketSpreads];

ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
'17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle);

Maturity3 = MarketData(:,1);
Spread3Run = MarketData(:,2);
Spread3Std = 100*ones(size(Maturity3));
Price3 = cdsprice(ZeroData,ProbData,Settle,Maturity3,Spread3Std);
Upfront3 = Price3/10000000; % Standard notional of 10MM
display(Upfront3);
```

This resulting value is:

```
Upfront3 =

    0.0047
    0.0158
    0.0327
    0.0737
    0.1182
```

The conversion can be reversed to convert upfront quotes to market quotes:

```
ProbData3Upf = cdsbootstrap(ZeroData,[Maturity3 Upfront3 Spread3Std],Settle);
Spread3RunFromUpf = cdsspread(ZeroData,ProbData3Upf,Settle,Maturity3);
display([Spread3Run Spread3RunFromUpf]);
```

Comparing the results of this conversion to the original market spread demonstrates the reversal:

```
ans =

  140.0000   140.0000
  175.0000   175.0000
  210.0000   210.0000
  265.0000   265.0000
  310.0000   310.0000
```

Under the flat-hazard rate (FHR) quoting convention, a single market quote is used to calibrate a probability curve. This convention yields a single point in the probability curve, and a single hazard rate value. For example, assume a 4-year (standard dates) CDS contract with a current FHR-based running spread of 550 bp needs conversion to a CDS contract with a standard spread of 500 bp:

```
Maturity4 = datenum('20-Sep-13');
Spread4Run = 550;
ProbData4Run = cdsbootstrap(ZeroData,[Maturity4 Spread4Run],Settle);
Spread4Std = 500;
Price4 = cdsprice(ZeroData,ProbData4Run,Settle,Maturity4,Spread4Std);
Upfront4 = Price4/10000000;
fprintf('A running spread of %5.2f is equivalent to\n',Spread4Run);
fprintf('   a standard spread of %5.2f with an upfront of %8.7f\n',...
   Spread4Std,Upfront4);

A running spread of 550.00 is equivalent to
   a standard spread of 500.00 with an upfront of 0.0167576
```

To reverse the conversion:

```
ProbData4Upf = cdsbootstrap(ZeroData,[Maturity4 Upfront4 Spread4Std],Settle);
Spread4RunFromUpf = cdsspread(ZeroData,ProbData4Upf,Settle,Maturity4);
fprintf('A standard spread of %5.2f with an upfront of %8.7f\n',...
   Spread4Std,Upfront4);
fprintf('   is equivalent to a running spread of %5.2f\n',Spread4RunFromUpf);

A standard spread of 500.00 with an upfront of 0.0167576
    is equivalent to a running spread of 550.00
```

As discussed in Beumee et. al., 2009 (see "Credit Derivatives" on page C-15), the FHR approach is a quoting convention only, and leads to quotes

inconsistent with market data. For example, calculating the upfront for the 3-year (standard dates) CDS contract with a standard spread of 100 bp using the FHR approach and comparing the results to the upfront amounts previously calculated, demonstrates that the FHR-based approach yields a different upfront amount:

```
Maturity5 = MarketData(3,1);
Spread5Run = MarketData(3,2);
ProbData5Run = cdsbootstrap(ZeroData,[Maturity5 Spread5Run],Settle);
Spread5Std = 100;
Price5 = cdsprice(ZeroData,ProbData5Run,Settle,Maturity5,Spread5Std);
Upfront5 = Price5/10000000;
fprintf('Relative error of FHR-based upfront amount: %3.1f%%\n',...
    ((Upfront5-Upfront3(3))/Upfront3(3))*100);

Relative error of FHR-based upfront amount: -0.8%
```

Bootstrapping from Inverted Market Curves

The following two examples demonstrate the behavior of bootstrapping with inverted CDS market curves, that is, market quotes with higher spreads for short-term CDS contracts. The first example is handled normally by cdsbootstrap:

```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10','20-Sep-11','20-Sep-12','20-Sep-14',...
'20-Sep-16'});

ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
'17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

MarketSpreadsInv1 = [750 650 550 500 450]';
MarketDataInv1 = [MarketDates MarketSpreadsInv1];
[ProbDataInv1,HazDataInv1] = cdsbootstrap(ZeroData,MarketDataInv1,Settle)

ProbDataInv1 =

    1.0e+05 *
```

```
        7.3440    0.0000
        7.3477    0.0000
        7.3513    0.0000
        7.3586    0.0000
        7.3659    0.0000


HazDataInv1 =

    1.0e+05 *

        7.3440    0.0000
        7.3477    0.0000
        7.3513    0.0000
        7.3586    0.0000
        7.3659    0.0000
```

In the second example, cdsbootstrap generates a warning:

```
MarketSpreadsInv2 = [800 550 400 250 100]';
MarketDataInv2 = [MarketDates MarketSpreadsInv2];


[ProbDataInv2,HazDataInv2] = cdsbootstrap(ZeroData,MarketDataInv2,Settle);

Warning: Found non-monotone default probabilities (negative hazard rates)
```

A non-monotone bootstrapped probability curve implies negative default probabilities and negative hazard rates for certain time intervals. Extreme market conditions can lead to these types of situations. In these cases, you must assess the reliability and usefulness of the bootstrapped results.

The following plot illustrates these bootstrapped probability curves. The curves are concave, meaning that the marginal default probability decreases with time. This result is consistent with the market information that indicates a higher default risk in the short term. The second bootstrapped curve is non-monotone, as indicated by the warning.

```
ProbTimes = yearfrac(Settle, MarketDates);
figure
plot([0; ProbTimes],[0; ProbDataInv1(:,2)])
hold on
```

```
plot([O; ProbTimes],[O; ProbDataInv2(:,2)],'--')
hold off
grid on
axis([O ProbTimes(end,1) O ProbDataInv1(end,2)])
xlabel('Time (years)')
ylabel('Cumulative Default Probability')
title('Probability Curves for Inverted Spread Curves')
legend('1st instance','2nd instance','location','SouthEast')
```

The resulting plot
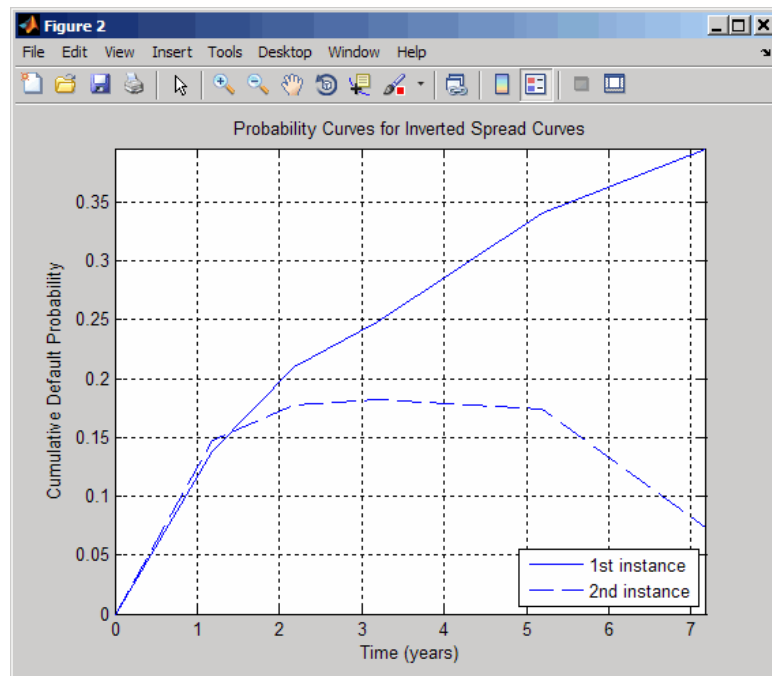
Also in line with the previous plot, the hazard rates for these bootstrapped curves are decreasing because the short-term risk is higher. Some bootstrapped parameters in the second curve are negative, as indicated by the warning.

```
HazTimes = yearfrac(Settle, MarketDates);
figure
```

```
stairs([O; HazTimes(1:end-1,1); HazTimes(end,1)+1],...
   [HazDataInv1(:,2);HazDataInv1(end,2)])
hold on
stairs([O; HazTimes(1:end-1,1); HazTimes(end,1)+1],...
   [HazDataInv2(:,2);HazDataInv2(end,2)],'--')
hold off
grid on
xlabel('Time (years)')
ylabel('Hazard Rate')
title('Hazard Rates for Inverted Spread Curves')
legend('1st instance','2nd instance','location','NorthEast')
```

The resulting plot shows the hazard rates for both bootstrapped curves:
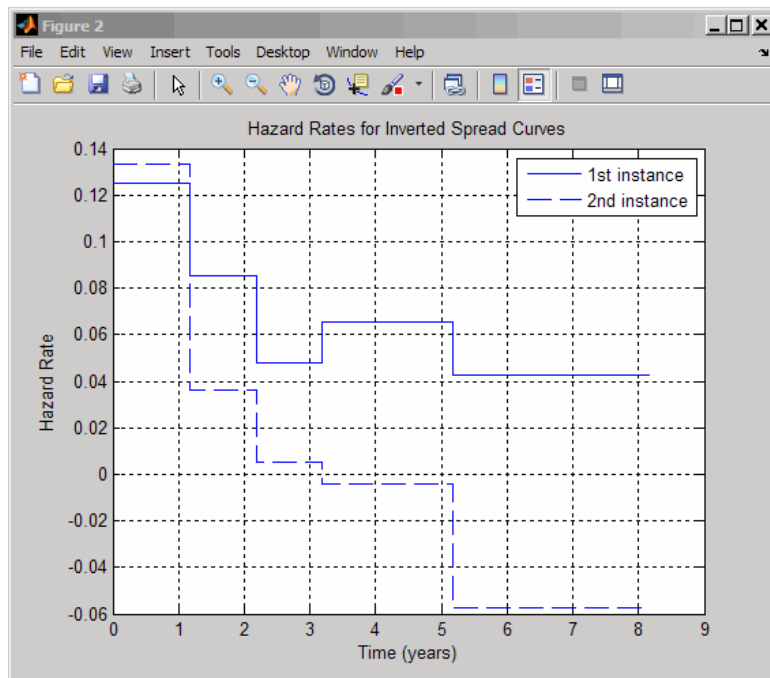
For further discussion on inverted curves, and their relationship to arbitrage, see O'Kane and Turnbull, 2003 ("Credit Derivatives" on page C-15).

Counterparty Credit Risk and CVA

This example shows how to compute the unilateral credit value (valuation) adjustment (CVA) for a bank holding a portfolio of vanilla interest rate swaps with several counterparties. CVA is the expected loss on an over-the-counter instrument or portfolio of instruments due to counterparty default. The CVA for a particular counterparty is defined as the sum over all points in time of the discounted expected exposure at each moment multiplied by the probability that the counterparty defaults at that moment, all multiplied by 1 minus the recovery rate. The CVA formula is:

$$CVA = (1 - R) \int_0^T discEE(t) dPD(t)$$

Where R is the recovery, `discEE` the discounted expected exposure at time t, and `PD` the default probability distribution.

The expected exposure is computed by first simulating many future scenarios of risk factors for the given instrument or portfolio. Risk factors can be interest rates, as in this example, but will differ based on the portfolio and can include FX rates, equity or commodity prices, or anything that will affect the market value of the instruments. Once a sufficient set of scenarios has been simulated, the contract or portfolio can be priced on a series of future dates for each scenario. The result is a matrix, or "cube", of instrument values.

These prices are converted into exposures after taking into account collateral agreements that the bank might have in place as well as netting agreements, as in this example, where the values of several instruments may offset each other, lowering their total exposure.

The instrument values for each scenario are discounted to compute the discounted exposures. The discounted expected exposures can then be computed by a simple average of the discounted exposures at each simulation date.

Finally, counterparty default probabilities are typically derived from credit default swap (CDS) market quotes and the CVA for the counterparty can be computed according to the above formula. We assume that a counterparty default is independent of its exposure (no wrong-way risk).

For this example we will work with a portfolio of vanilla interest rate swaps with the goal of computing the CVA for a particular counterparty.

Read Swap Portfolio

The portfolio of swaps is close to zero value at time t=0. Each swap is associated with a counterparty and may or may not be included in a netting agreement.

```
% Read swaps from spreadsheet
swapFile = 'cva-swap-portfolio.xls';
swapData = dataset('XLSFile',swapFile,'Sheet','Swap Portfolio');

swaps = struct(...
    'Counterparty',[],...
    'NettingID',[],...
    'Principal',[],...
    'Maturity',[],...
    'LegRate',[],...
    'LegType',[],...
    'LatestFloatingRate',[],...
    'FloatingResetDates',[]);

swaps.Counterparty = swapData.CounterpartyID;
swaps.NettingID    = swapData.NettingID;
swaps.Principal    = swapData.Principal;
swaps.Maturity     = swapData.Maturity;
swaps.LegType      = [swapData.LegType ~swapData.LegType];
swaps.LegRate      = [swapData.LegRateReceiving swapData.LegRatePaying];
swaps.LatestFloatingRate = swapData.LatestFloatingRate;
swaps.Period       = swapData.Period;
swaps.LegReset     = ones(size(swaps.LegType));

numSwaps = numel(swaps.Counterparty);
numCounterparties = max(swaps.Counterparty);
```

Create RateSpec from the Interest Rate Curve

```
Settle   = datenum('14-Dec-2007');
```

```
Tenor  = [3 6 12 5*12 7*12 10*12 20*12 30*12]';
ZeroRates = [0.033 0.034 0.035 0.040 0.042 0.044 0.048 0.0475]';

ZeroDates = datemnth(Settle,Tenor);
Compounding = 2;
Basis = 0;
RateSpec = intenvset('StartDates', Settle,'EndDates', ZeroDates,...
    'Rates', ZeroRates,'Compounding',Compounding,'Basis',Basis);

% Create an IRCurve object.  We will use this for computing instantaneous
% forward rates during the calculation of the Hull-White short rate path.
RateCurveObj = IRDataCurve('Zero',Settle,ZeroDates,ZeroRates,...
    'Compounding', Compounding,'Basis', Basis);

figure;
plot(ZeroDates, ZeroRates, 'o-');
xlabel('Date');
datetick('keeplimits');
ylabel('Zero rate'); grid on;
title('Yield Curve at Settle Date');
```

Yield Curve at Settle Date
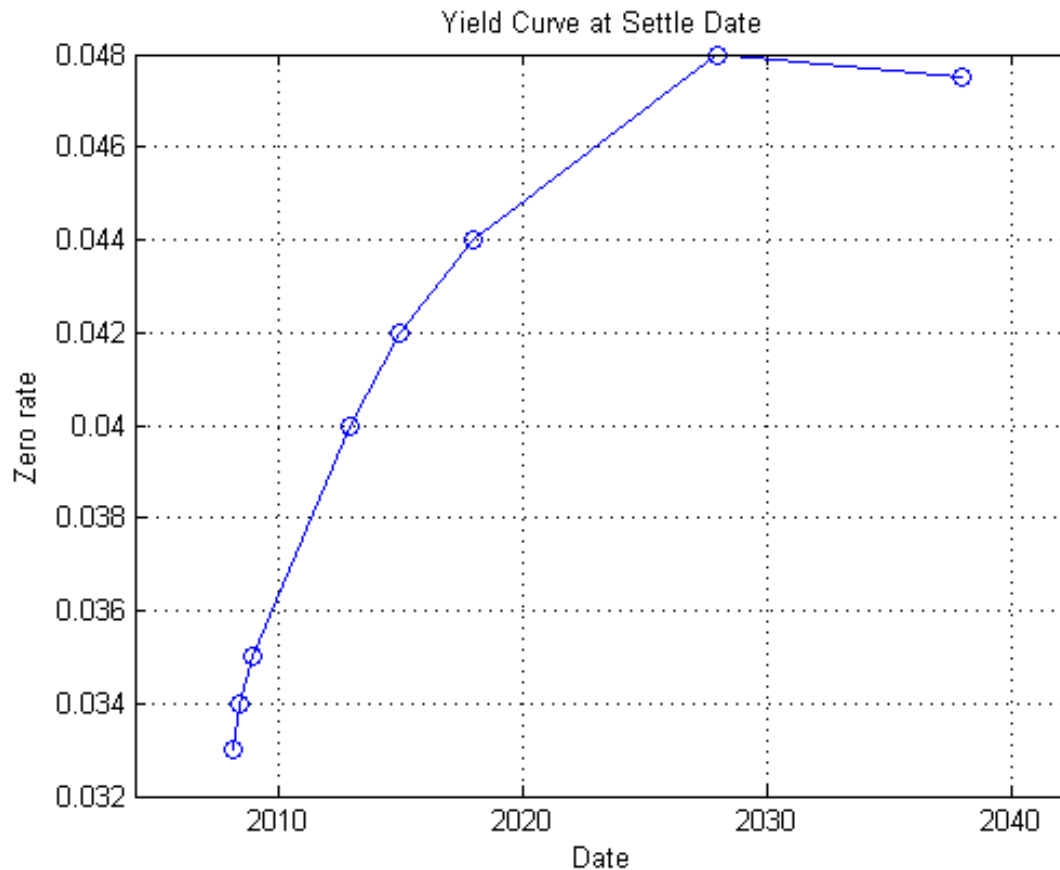
Set Changable Simulation Parameters

We can vary here the number of simulated interest rate scenarios we generate. We set our simulation dates to be more frequent at first, then turning less frequent further in the future.

```
% Number of Monte Carlo simulations
numScenarios = 1000;

% Compute monthly simulation dates, then quarterly dates later.
simulationDates = datemnth(Settle,0:12);
```

```
simulationDates = [simulationDates datemnth(simulationDates(end),3:3:74)]';
numDates = numel(simulationDates);
```

Compute Floating Reset Dates

For each simulation date, we compute previous floating reset date for each swap.

```
floatDates = cfdates(Settle-360,swaps.Maturity,swaps.Period);
swaps.FloatingResetDates = zeros(numSwaps,numDates);
for i = numDates:-1:1
    thisDate = simulationDates(i);
    floatDates(floatDates > thisDate) = 0;
    swaps.FloatingResetDates(:,i) = max(floatDates,[],2);
end
```

Setup Hull-White Single Factor Model

The risk factor we will simulate to value our instruments is the zero curve. For this example we will model the interest rate term structure using the one-factor Hull-White model. This is a model of the short rate and is defined as:

$$dr = [\theta(t) - ar]dt + \sigma dz$$

where

- dr : Change in the short rate after a small change in time, dt
- a : Mean reversion rate
- σ : Volatility of the short rate
- dz : A Weiner process (a standard normal process)
- $\theta(t)$: Drift function defined as:

$$\theta(t) = F_t(0,t) + aF(0,t) + \frac{\sigma^2}{2a}(1 - e^{-2at})$$

$F(0,t)$: Instantaneous forward rate at time t

$F_t(0,t)$: Partial derivative of F with respect to time

Once we have simulated a path of the short rate we generate a full yield curve at each simulation date using the formula:

$$R(t,T) = -\frac{1}{(T-t)} \ln A(t,T) + \frac{1}{(T-t)} B(t,T) r(t)$$

$$\ln A(t,T) = \ln \frac{P(0,T)}{P(0,t)} + B(t,T) F(0,t) - \frac{1}{4a^3} \sigma^2 (e^{-aT} - e^{-at})^2 (e^{2at} - 1)$$

$$B(t,T) = \frac{1 - e^{-a(T-t)}}{a}$$

$R(t,T)$: Zero rate at time t for a period of $T - t$

$P(t,T)$: Price of a zero coupon bond at time t that pays one dollar at time T

Each scenario contains the full term structure moving forward through time, modeled at each of our selected simulation dates.

Refer to "Calibrating the Hull-White Model Using Market Data" example in the Financial Instruments Toolbox™ Users' Guide for more details on Hull-White one-factor model calibration.

```
Alpha = 0.2;
Sigma = 0.015;

r0 = RateCurveObj.getZeroRates(Settle+1,'Compounding',-1);
t0 = Settle;

% Construct SDE object
FwdRates = RateCurveObj.getForwardRates(t0+1:max(swaps.Maturity),'Compoundi
hullwhite1 = hwv(Alpha,@(t,x) hw1LevelFun(t0,t,FwdRates,Alpha,Sigma),...
    Sigma,'StartState',r0) %#ok<NOPTS>

% Store all model calibration information
calibration.RateCurveObj = RateCurveObj;
calibration.Tenor = Tenor;
```

```
calibration.ShortRateModel = hullwhite1;
calibration.Alpha = Alpha;
calibration.Sigma = Sigma;


hullwhite1 =

   Class HWV: Hull-White/Vasicek
   ----------------------------------------
     Dimensions: State = 1, Brownian = 1
   ----------------------------------------
      StartTime: 0
     StartState: 0.0317681
    Correlation: 1
          Drift: drift rate function F(t,X(t))
       Diffusion: diffusion rate function G(t,X(t))
      Simulation: simulation method/function simByEuler
          Sigma: 0.015
          Level: function @(t,x)hw1LevelFun(t0,t,FwdRates,Alpha,Sigma)
          Speed: 0.2
```

Simulate Scenarios

For each scenario, we simulate the future interest rate curve at each valuation
date using the Hull-White one-factor interest rate model.

```
% Use reproducible random number generator (vary the seed to produce
% different random scenarios).
prevRNG = rng(0);

% Compute interest rate scenarios are their respective discount factors
[scenarios, dfactors] = hgenerateScenario(calibration,simulationDates,numSc

% Restore random number generator state
rng(prevRNG);
```

Inspect a Scenario

Create a surface plot of the yield curve evolution for a particular scenario.

```
i = 20;
figure;
surf(Tenor, simulationDates, scenarios(:,:,i))
axis tight
datetick('y','mmmyy');
xlabel('Tenor (Months)');
ylabel('Observation Date');
zlabel('Rates');
set(gca,'View',[-49 32]);
title(sprintf('Scenario %d Yield Curve Evolution\n',i));
```

Scenario 20 Yield Curve Evolution
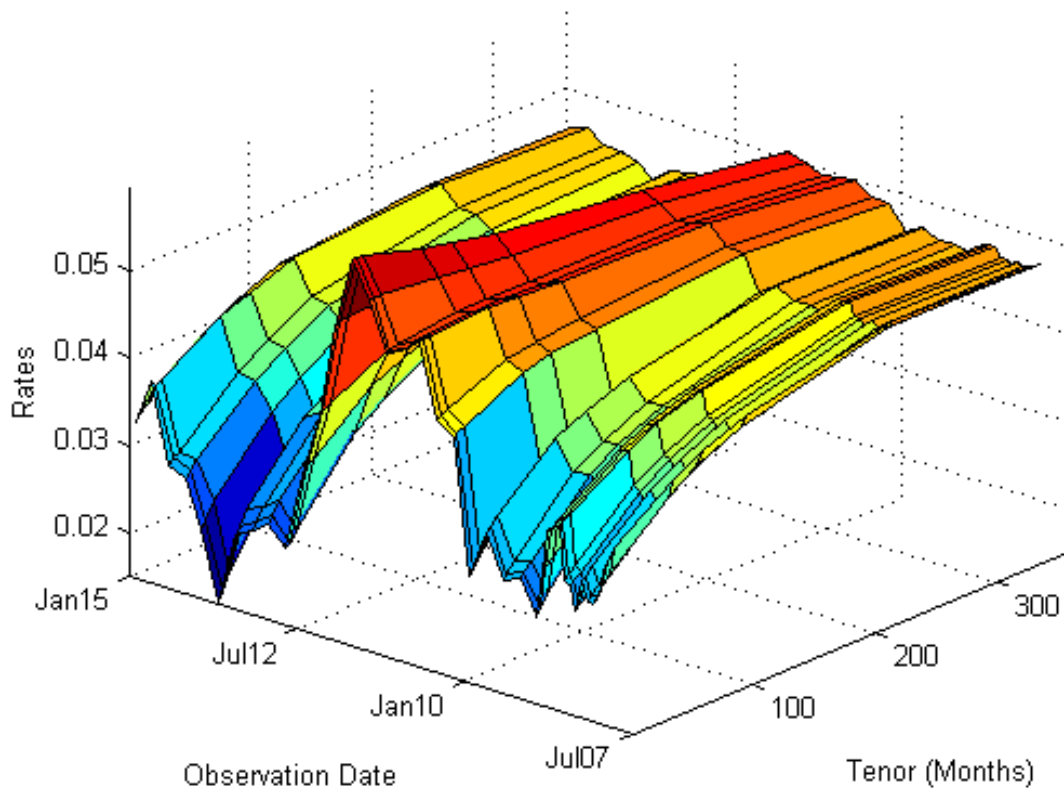
Compute Mark to Market Swap Prices

For each scenario the swap portfolio is priced at each future simulation date. Prices are computed using a price approximation function, `hswapapprox`. It is common in CVA applications to use simplified approximation functions when pricing instruments due to the performance requirements of these Monte Carlo simulations.

Since the simulation dates do not correspond to the swaps cash flow dates (where the floating rates are reset) we estimate the latest floating rate with the 1-year rate (all swaps have period 1 year) interpolated between the nearest simulated rate curves.

The swap prices are then aggregated into a "cube" of values which contains all future instrument values at each simulation date for each scenario. The resulting cube of instrument prices is a 3 dimensional matrix where each row represents a simulation date, each column an instrument, and each "page" a different simulated scenario.

```
% Compute all mark-to-market values for this scenario.  We use an
% approximation function here to improve performance.
values = hcomputeMTMValues(swaps,simulationDates,scenarios,Tenor);
```

Inspect Scenario Prices

Create a plot of the evolution of all swap prices for a particular scenario.

```
i = 32;
figure;
plot(simulationDates, values(:,:,i));
datetick;
ylabel('Mark-To-Market Price');
title(sprintf('Swap prices along scenario %d', i));
```

Swap prices along scenario 32
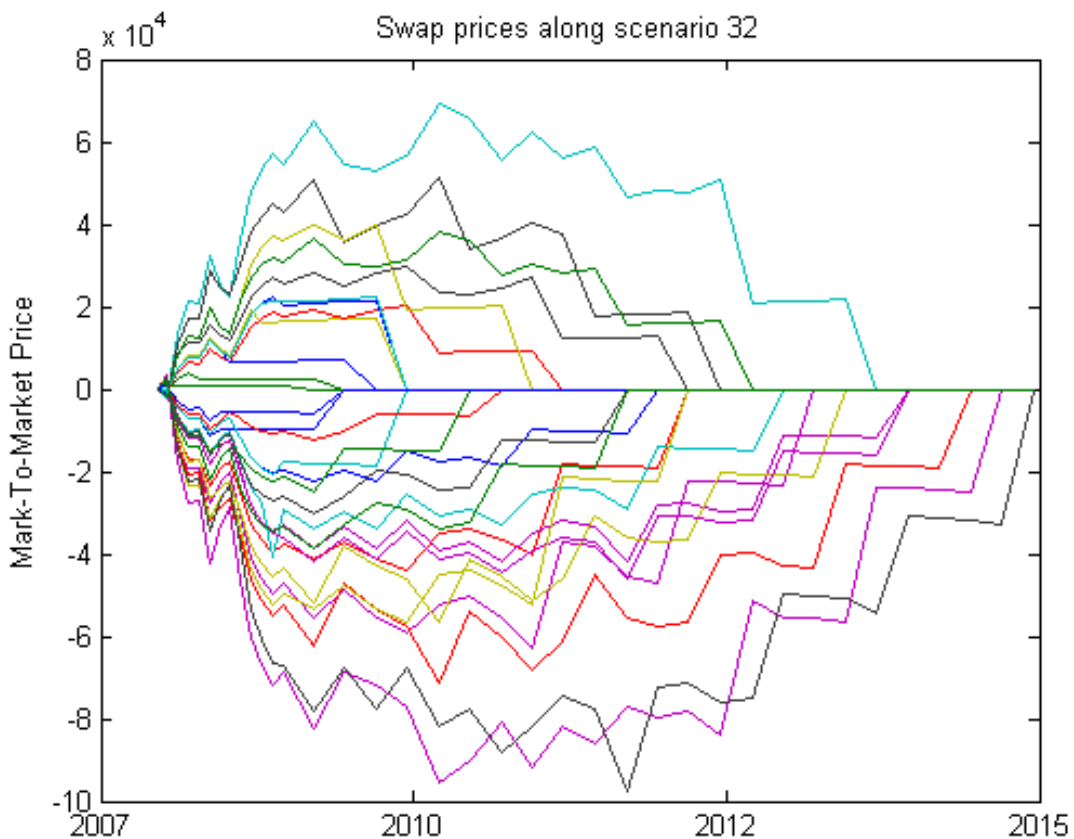
Visualize Simulated Portfolio Values

We plot the total portfolio value for each scenario of our simulation. As each scenario moves forward in time the values of the instruments will move up or down depending on how the modeled interest rate term structure changes. As the swaps get closer to maturity, their values will begin to approach zero since the aggregate value of all remaining cash flows will decrease after each cash flow date.

```
% View portfolio value over time
figure;
```

```
totalPortValues = squeeze(sum(values, 2));
plot(simulationDates,totalPortValues);
title('Total MTM Portfolio Value for All Scenarios');
datetick('x','mmmyy')
ylabel('Portfolio Value ($)')
xlabel('Simulation Dates')
```
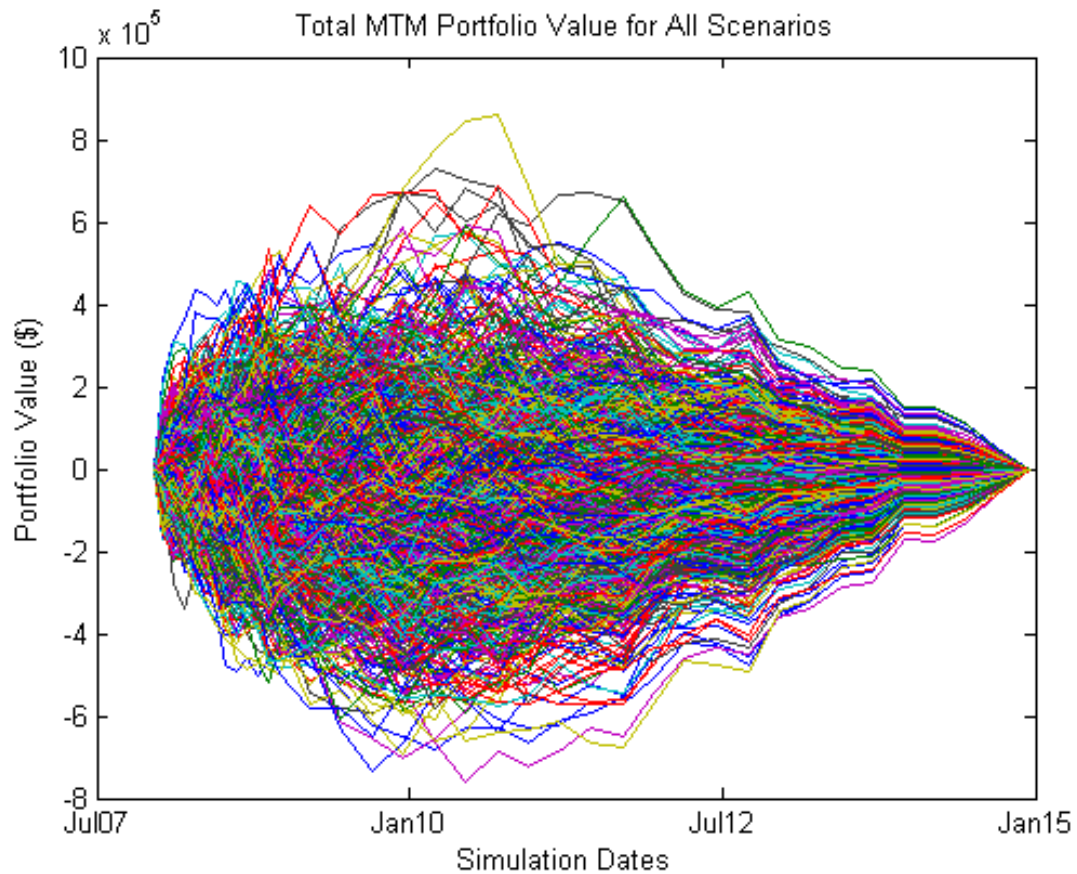

Compute Exposure by Counterparty

The exposure of a particular contract (i) at time t is the maximum of the contract value (Vi) and 0:

$$E_i(t) = \max\{V_i(t), 0\}$$

And the exposure for a particular counterparty is simply a sum of the individual contract exposures:

$$E_{cp}(t) = \sum E_i(t) = \sum \max\{V_i(t), 0\}$$

In the presence of netting agreements, however, contracts are aggregated together and can offset each other. Therefore the total exposure of all instruments in a netting agreement is:

$$E_{na}(t) = \max\{\sum V_i(t), 0\}$$

We compute these exposures for each counterparty at each simulation date.

Additive exposure is computed at the netting set level. Exposure of an unnetted instrument is equal to the market value of the instrument if the instrument has positive value, otherwise it is zero.

Instruments included in a netting agreement have exposure equal to their value (positive or negative) when the netting agreement has a positive aggregate value, otherwise their exposure is zero. A contract in a netting agreement can have a negative additive exposure, meaning its negative market value reduces the overall netting set exposure. See references for more details.

```
instrument_exposures = zeros(size(values));
unnettedIdx = swaps.NettingID == 0;
instrument_exposures(:,unnettedIdx,:) = max(values(:,unnettedIdx,:),0);

% We compute exposures per netting agreement, but in this case each
% counterparty has only a single netting agreement.
for i = 1:numCounterparties

    nettedIdx = swaps.NettingID == i;
    numInst = sum(nettedIdx);
```

```
    % Exposures for instruments under netting agreements
    nettingSetValues = values(:,nettedIdx,:);
    nettedExposure = max(sum(nettingSetValues,2),0);
    positiveIdx = repmat(nettedExposure > 0,[1 numInst]);

    % Individual instrument contributions to netting set exposure
    instrument_exposures(:,nettedIdx,:) = nettingSetValues .* positiveIdx;

end

% Sum the instrument exposures for each counterparty
exposures = zeros(numDates,numCounterparties,numScenarios);
for i = 1:numCounterparties

    cpSwapIdx = swaps.Counterparty == i;
    exposures(:,i,:) = sum(instrument_exposures(:,cpSwapIdx,:),2);

end
```

We plot the total portfolio exposure for each scenario in our simulation. Similar to the plot of instrument values, the exposures for each scenario will approach zero as the swaps mature.

```
% View portfolio exposure over time
figure;
totalPortExposure = squeeze(sum(exposures,2));
plot(simulationDates,totalPortExposure);
title('Portfolio Exposure for All Scenarios');
datetick('x','mmmyy')
ylabel('Exposure ($)')
xlabel('Simulation Dates')
```
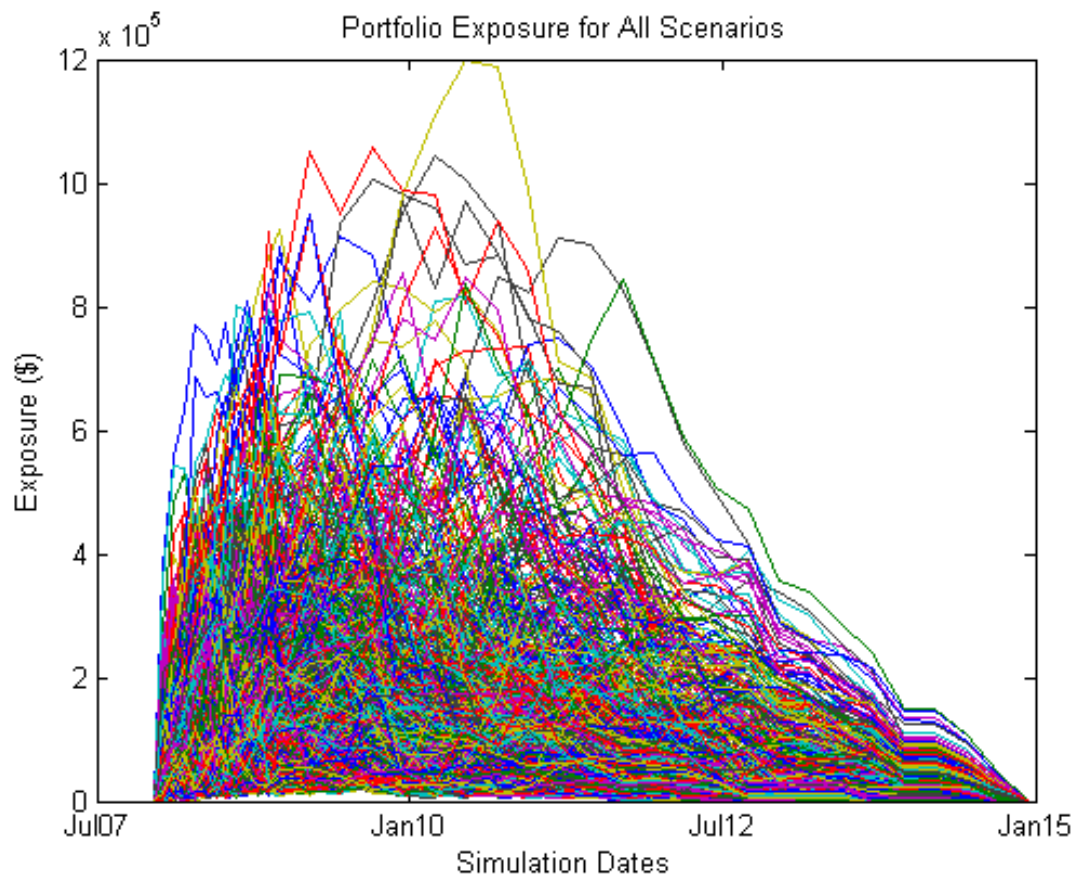
Exposure Profiles

Several exposure profiles are useful when analyzing the potential future exposure of a bank to a counterparty. Here we compute several (non-discounted) exposure profiles per counterparty as well as for the entire portfolio.

- PE : Peak Exposure : A high percentile (95%) of the distribution of exposures at any particular future date. Also called Potential Future Exposure (PFE).

- `MPE` : Maximum Peak Exposure : The maximum peak exposure across all dates

- `EE` : Expected Exposure : The mean (average) of the distribution of exposures at each date

- `EPE` : Expected Positive Exposure : Weighted average over time of the expected exposure

- `EffEE` : Effective Expected Exposure : The maximum expected exposure at any time, t, or previous time

- `EffEPE` : Effective Expected Positive Exposure : The weighted average of the effective expected exposure

For further definitions, see for example Basel II document in references.

```
% Compute entire portfolio exposure
expPort = squeeze(sum(exposures,2));

% Peak Exposure (same as Potential Future Exposure)
PEcp   = prctile(exposures,95,3);
PEport = prctile(expPort,95,2);

% Maximum Peak Exposure
MPEcp   = max(PEcp);
MPEport = max(PEport);

% Expected Exposure
EEcp   = mean(exposures,3);
EEport = mean(expPort,2);

% Expected Positive Exposure: Weighted average over time of EE
% * In continuous time, this is the average expected exposures over time,
%   an integral of EE(t) over the time interval, divided by the length of
%   the interval
% * Compute using a "trapezoidal" approach here
simTimeInterval = yearfrac(Settle, simulationDates, 1);
simTotalTime = simTimeInterval(end)-simTimeInterval(1);
EPEcp   = 0.5*(EEcp(1:end-1,:)+EEcp(2:end,:))' * diff(simTimeInterval)/simT
EPEport = 0.5*(EEport(1:end-1)+EEport(2:end))' * diff(simTimeInterval)/simT
```

```
% Effective Expected Exposure: Max EE up to time simTimeInterval
EffEEcp = zeros(size(EEcp));
for i = 1:size(EEcp,2)

    % Compute cumulative maximum
    m = EEcp(1,i);
    for j = 1:numel(simulationDates)
        if EEcp(j,i) > m
            m = EEcp(j,i);
        end
        EffEEcp(j,i) = m;
    end

end

% Compute cumulative maximum for portfolio
EffEEport = zeros(size(EEport));
m = EEport(1);
for i = 1:numel(simulationDates)
    if EEport(i) > m
        m = EEport(i);
    end
    EffEEport(i) = m;
end

% Effective Expected Positive Exposure: Weighted average over time of EffEE
EffEPEcp   = 0.5*(EffEEcp(1:end-1,:)+EffEEcp(2:end,:))' * diff(simTimeInter
EffEPEport = 0.5*(EffEEport(1:end-1)+EffEEport(2:end))' * diff(simTimeInter
```

We visualize the exposure profiles, first for the entire portfolio, then for a particular counterparty.

```
% Visualize portfolio exposure profiles
figure;
plot(simulationDates,PEport,...
    simulationDates,MPEport*ones(size(PEport)),...
    simulationDates,EEport,...
    simulationDates,EPEport*ones(size(PEport)),...
    simulationDates,EffEEport,...
    simulationDates,EffEPEport*ones(size(PEport)))
```

```
legend({'Peak Exposure (PE95%)','Max Peak Exposure (MPE95%)',...
    'Exp Exposure (EE)','Time-Avg EE (EPE)','Max past EE (EffEE)','Time-Avg

datetick('x','mmmyy')
title('Portfolio Exposure Profiles');
ylabel('Exposure ($)')
xlabel('Simulation Dates')
```
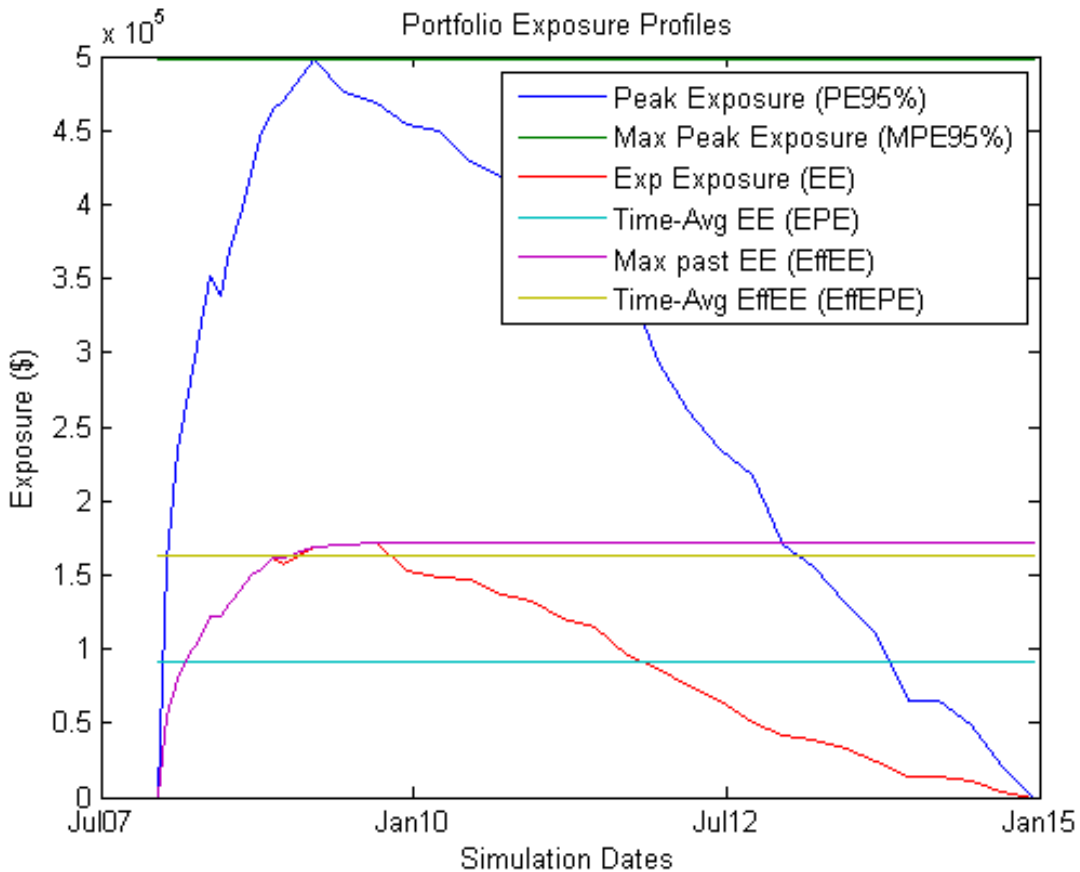

Visualize exposure profiles for a particular counterparty

```
cpIdx = 5;
```

```
figure;
plot(simulationDates,PEcp(:,cpIdx),...
    simulationDates,MPEcp(cpIdx)*ones(size(PEcp(:,cpIdx))),...
    simulationDates,EEcp(:,cpIdx),...
    simulationDates,EPEcp(cpIdx)*ones(size(PEcp(:,cpIdx))),...
    simulationDates,EffEEcp(:,cpIdx),...
    simulationDates,EffEPEcp(cpIdx)*ones(size(PEcp(:,cpIdx))))
legend({'Peak Exposure (PE95%)','Max Peak Exposure (MPE95%)',...
    'Exp Exposure (EE)','Time-Avg EE (EPE)','Max past EE (EffEE)','Time-Avg

datetick('x','mmmyy','keeplimits')
title(sprintf('Counterparty %d Exposure Profiles',cpIdx));
ylabel('Exposure ($)')
xlabel('Simulation Dates')
```

Counterparty 5 Exposure Profiles
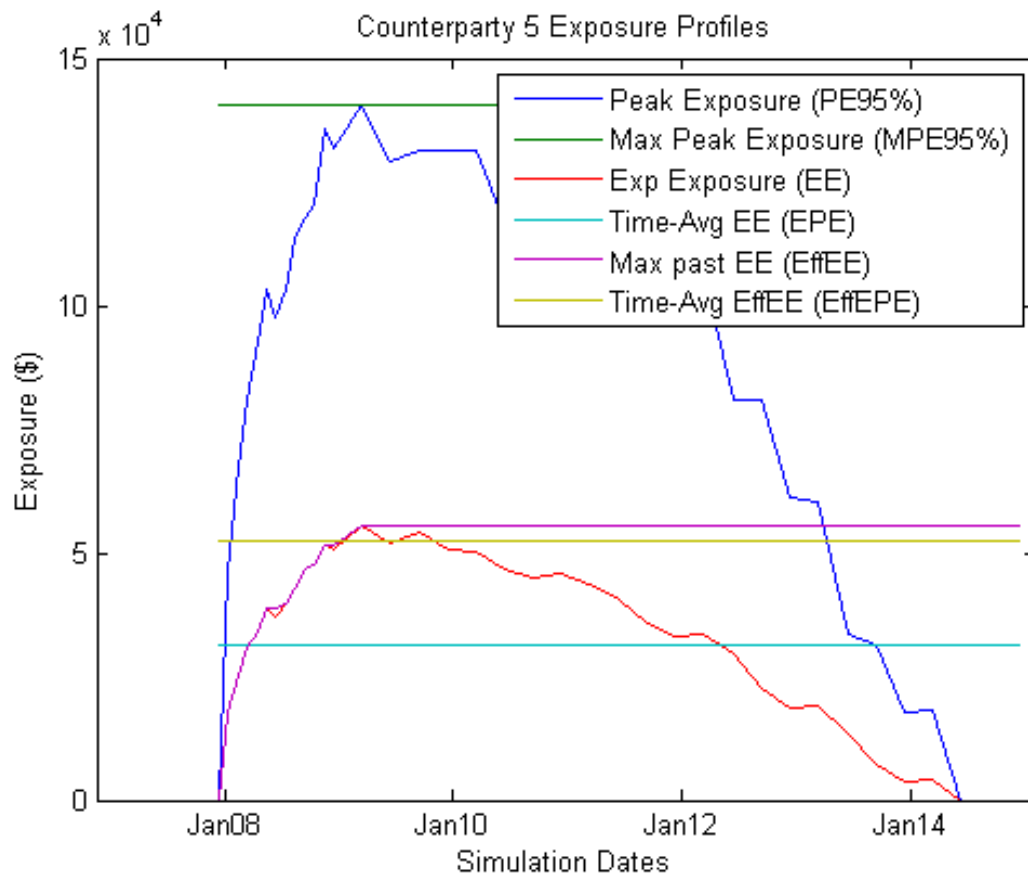
Discounted Exposures

We compute the discounted expected exposures using the discount factors
from each simulated interest rate scenario. The discount factor for a given
valuation date in a given scenario is the product of the incremental discount
factors from one simulation date to the next, along the interest rate path
of that scenario.

```
% Get discounted exposures per counterparty, for each scenario
discExp = zeros(size(exposures));
for i = 1:numScenarios
```

```
    discExp(:,:,i) = bsxfun(@times,dfactors(:,i),exposures(:,:,i));
end

% Discounted expected exposure
discEE = mean(discExp,3);
```

We plot the discounted expected exposures for the aggregate portfolio as well as for each counterparty.

```
% Portfolio discounted EE
figure;
plot(simulationDates,sum(discEE,2))
datetick('x','mmmyy','keeplimits')
title('Discounted Expected Exposure for Portfolio');
ylabel('Discounted Exposure ($)')
xlabel('Simulation Dates')

% Counterparty discounted EE
figure;
plot(simulationDates,discEE)
datetick('x','mmmyy','keeplimits')
title('Discounted Expected Exposure for Each Counterparty');
ylabel('Discounted Exposure ($)')
xlabel('Simulation Dates')
```

Discounted Expected Exposure for Portfolio

Discounted Expected Exposure for Each Counterparty
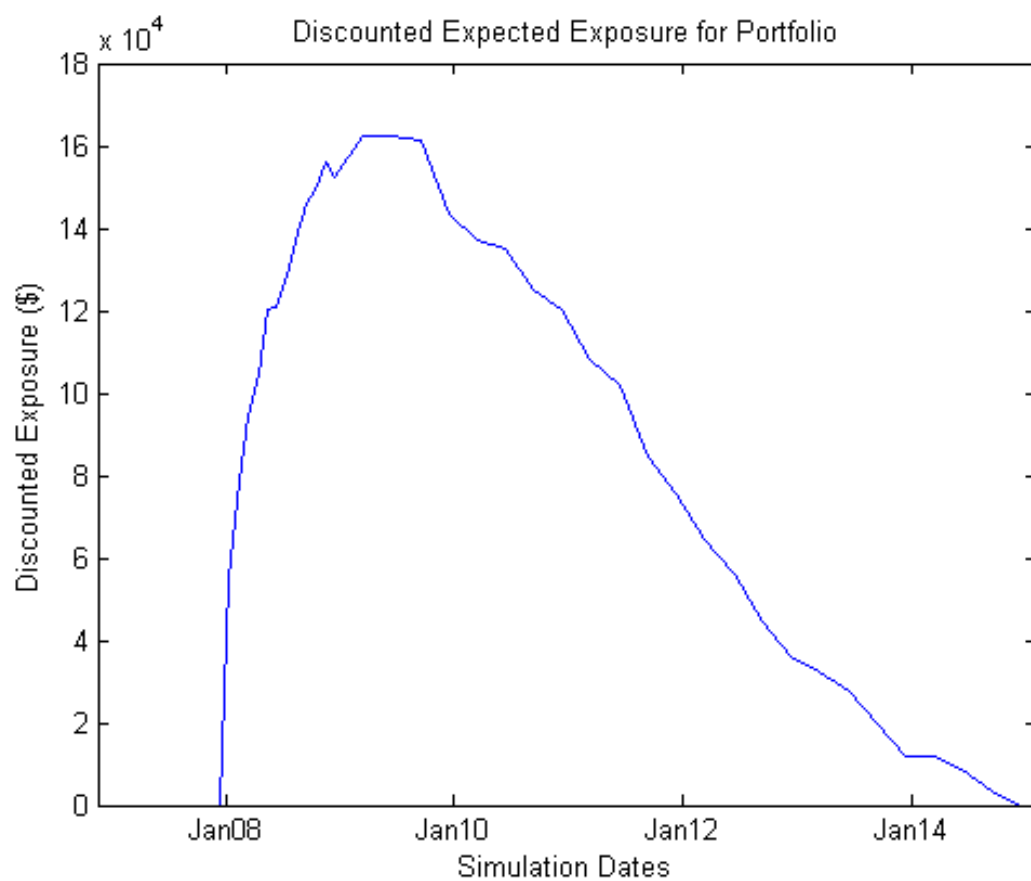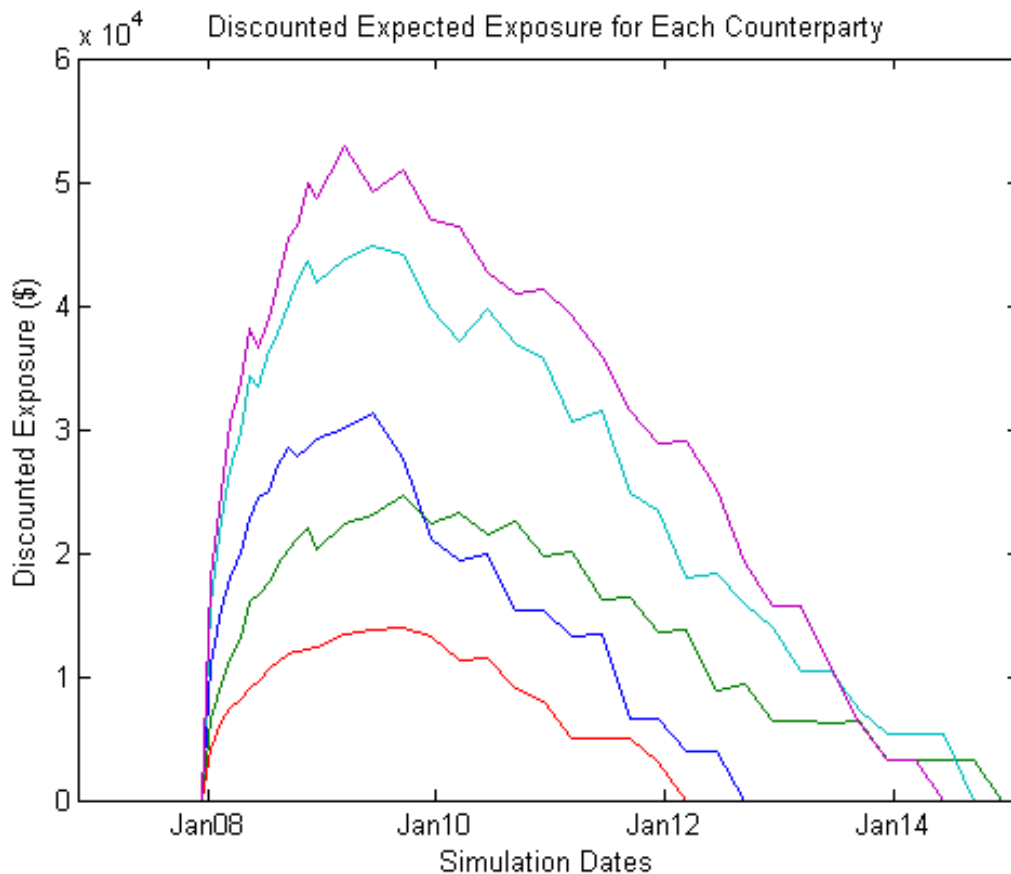
Calibrating Probability of Default Curve for Each Counterparty

The default probability for a given counterparty is implied by the current market spreads of the counterparty's CDS. We use the function cdsbootstrap to generate the cumulative probability of default at each simulation date.

```
% Import CDS market information for each counterparty
CDS = dataset('XLSfile',swapFile,'Sheet','CDS Spreads') %#ok<NOPTS>
CDSDates = datenum(CDS.Date);
CDSSpreads = double(CDS(:,2:end));
```

```
ZeroData = [RateSpec.EndDates RateSpec.Rates];

% Calibrate default probabilities for each counterparty
DefProb = zeros(length(simulationDates), size(CDSSpreads,2));
for i = 1:size(DefProb,2)
    probData = cdsbootstrap(ZeroData, [CDSDates CDSSpreads(:,i)],...
        Settle, 'probDates', simulationDates);
    DefProb(:,i) = probData(:,2);
end

% We plot of the cumulative probability of default for each counterparty.
figure;
plot(simulationDates,DefProb)
title('Default Probability Curve for Each Counterparty');
xlabel('Date');
grid on;
ylabel('Cumulative Probability')
datetick('x','mmmyy')
ylabel('Probability of Default')
xlabel('Simulation Dates')


CDS =

    Date              cp1     cp2     cp3     cp4     cp5
    '3/20/2008'       140      85     115     170     140
    '3/20/2009'       185     120     150     205     175
    '3/20/2010'       215     170     195     245     210
    '3/20/2011'       275     215     240     285     265
    '3/20/2012'       340     255     290     320     310
```

Default Probability Curve for Each Counterparty
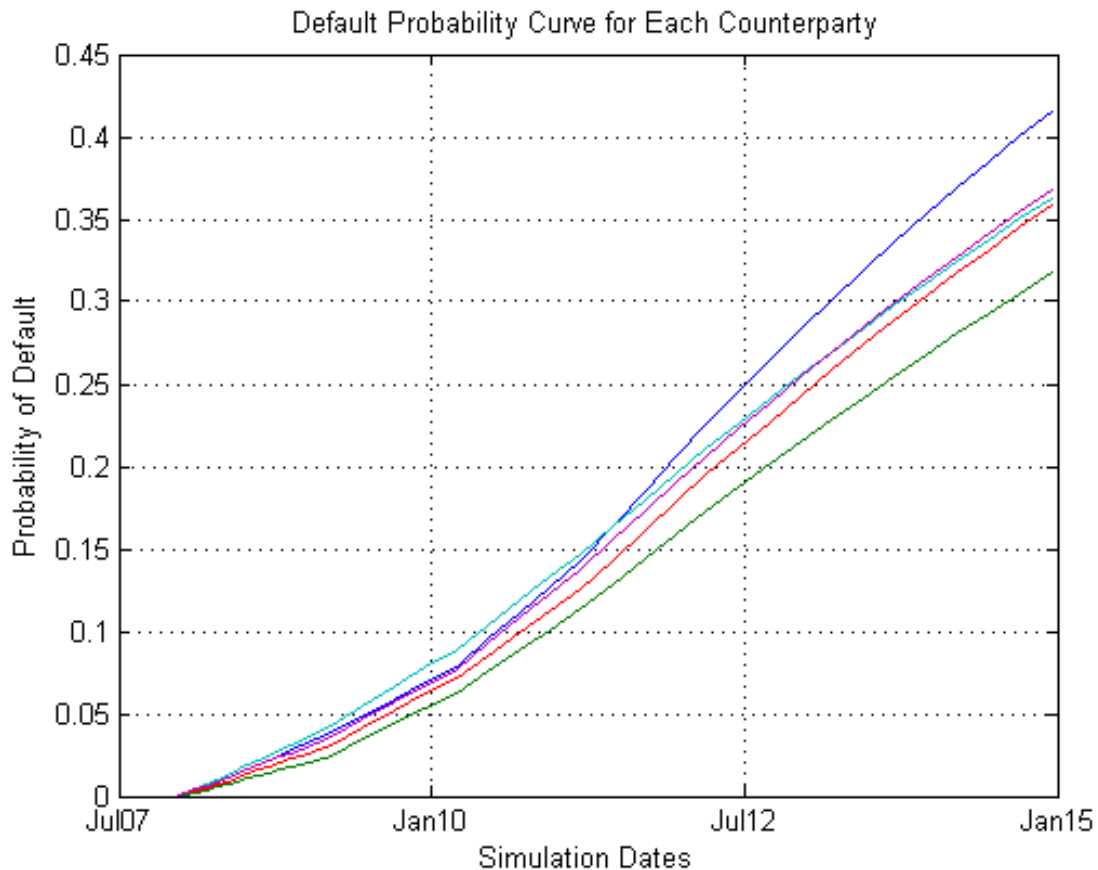
CVA Computation

The Credit Value (Valuation) Adjustment (CVA) formula is:

$$CVA = (1 - R) \int_0^T discEE(t)dPD(t)$$

Where R is the recovery, discEE the discounted expected exposure at time t, and PD the default probability distribution. This assumes the exposure is independent of default (no wrong-way risk), and it also assumes the exposures were obtained using risk-neutral probabilities.

Here we approximate the integral with a finite sum over the valuation dates as:

$$CVA(approx) = (1 - R) \sum_{i=2}^{n} discEE(t_i)(PD(t_i) - PD(t_{i-1}))$$

where t_1 is todays date, t_2,...,t_n the future valuation dates.

We assume CDS info corresponds to counterparty with index cpIdx. The computed CVA is the present market value of our credit exposure to counterparty cpIdx. For this example we set the recovery rate at 40%.

```
Recovery = 0.4;
CVA = (1-Recovery) * sum(discEE(2:end,:) .* diff(DefProb));
for i = 1:numel(CVA)
    fprintf('CVA for counterparty %d = $%.2f\n',i,CVA(i));
end

figure;
bar(CVA);
title('CVA for each counterparty');
xlabel('Counterparty');
ylabel('CVA $');
grid on;

CVA for counterparty 1 = $2225.23
CVA for counterparty 2 = $2431.85
CVA for counterparty 3 = $929.50
CVA for counterparty 4 = $5064.17
CVA for counterparty 5 = $5803.35
```

CVA for each counterparty
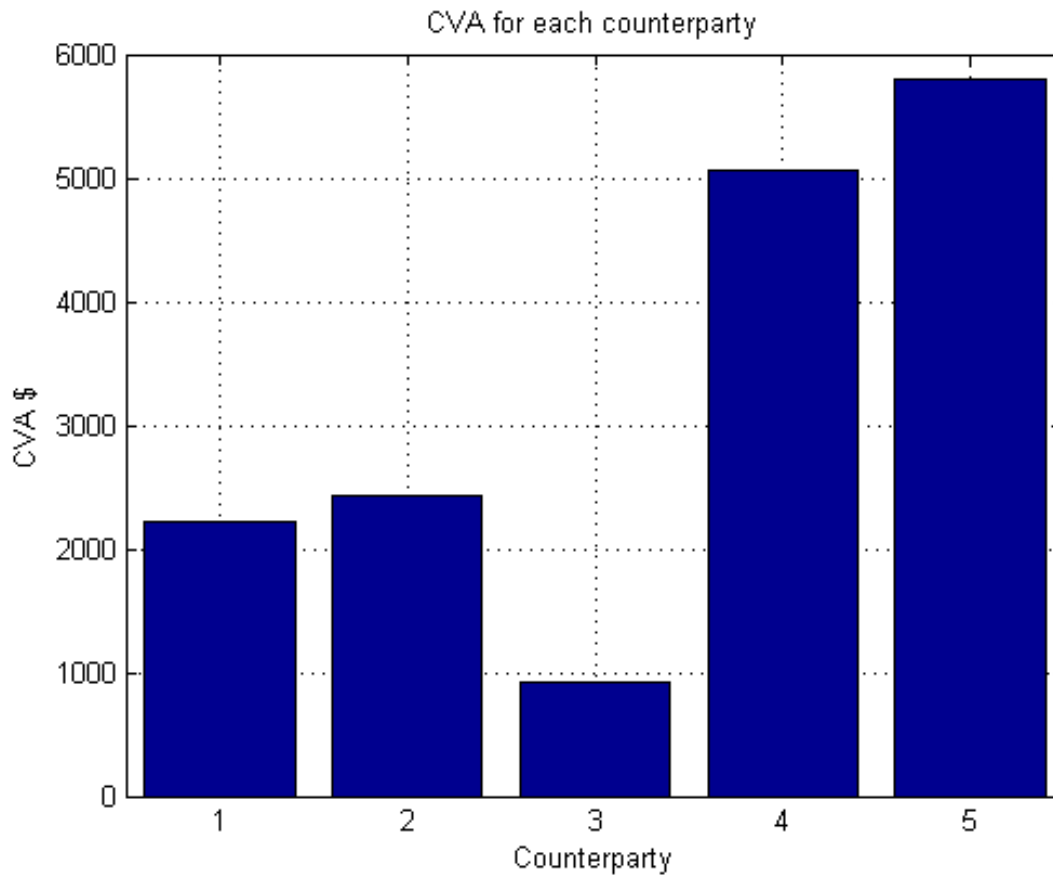
References

1 Pykhtin, Michael, and Steven Zhu, *A Guide to Modelling Counterparty Credit Risk*, GARP, July/August 2007, issue 37, pp. 16-22.

2 Pykhtin, Michael, and Dan Rosen, *Pricing Counterparty Risk at the Trade Level and CVA*, 2010.

3 Basel II: http://www.bis.org/publ/bcbs128.pdf page 256

First-to-Default Swaps

This example shows how to price first-to-default (FTD) swaps under the homogeneous loss assumption.

A first-to-default swap is an instrument that pays a predetermined amount when (and if) the first of a basket of credit instruments defaults. The credit instruments in the basket are usually bonds. If we assume that the loss amount following a credit event is the same for all credits in the basket, we are under the *homogeneous loss* assumption. This assumption makes models simpler, because any default in the basket triggers the same payment amount. This example is an implementation of the pricing methodology for these instruments, as described in O'Kane [2]. There are two steps in the methodology: a) Compute the survival probability for the basket numerically; b) Use this survival curve and standard single-name credit-default swap (CDS) functionality to find FTD spreads and to price existing FTD swaps.

Fit Probability Curves to Market Data

Given CDS market quotes for each issuer in the basket, use cdsbootstrap to calibrate individual default probability curves for each issuer.

```
% Interest rate curve
ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
'17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

% CDS spreads
%   Each row in MarketSpreads corresponds to a different issuer; each
%   column to a different maturity date (corresponding to MarketDates)
MarketDates = datenum({'20-Sep-10','20-Sep-11','20-Sep-12','20-Sep-14',...
'20-Sep-16'});
MarketSpreads = [
   160 195 230 285 330;
   130 165 205 260 305;
   150 180 210 260 300;
   165 200 225 275 295];
% Number of issuers equals number of rows in MarketSpreads
nIssuers = size(MarketSpreads,1);
```

```
% Settlement date
Settle = datenum('17-Jul-2009');
```

In practice, the time axis is discretized and the FTD survival curve is only evaluated at grid points. We use one point every three months. To request that `cdsbootstrap` returns default probability values over the specific grid points we want, use the optional argument 'ProbDates'. We add the original standard CDS market dates to the grid, otherwise the default probability information on those dates would be interpolated using the two closest dates on the grid, and the prices on market dates would be inconsistent with the original market data.

```
ProbDates = union(MarketDates,daysadd(Settle,360*(0.25:0.25:8),1));
nProbDates = length(ProbDates);
DefProb = zeros(nIssuers,nProbDates);

for ii=1:nIssuers
    MarketData = [MarketDates MarketSpreads(ii,:)'];
    ProbData = cdsbootstrap(ZeroData,MarketData,Settle,...
        'ProbDates',ProbDates);
    DefProb(ii,:) = ProbData(:,2)';
end
```

These are the calibrated default probability curves for each credit in the basket.

```
figure
plot(ProbDates',DefProb)
datetick
title('Individual Default Probability Curves')
ylabel('Cumulative Probability')
xlabel('Date')
```

Individual Default Probability Curves
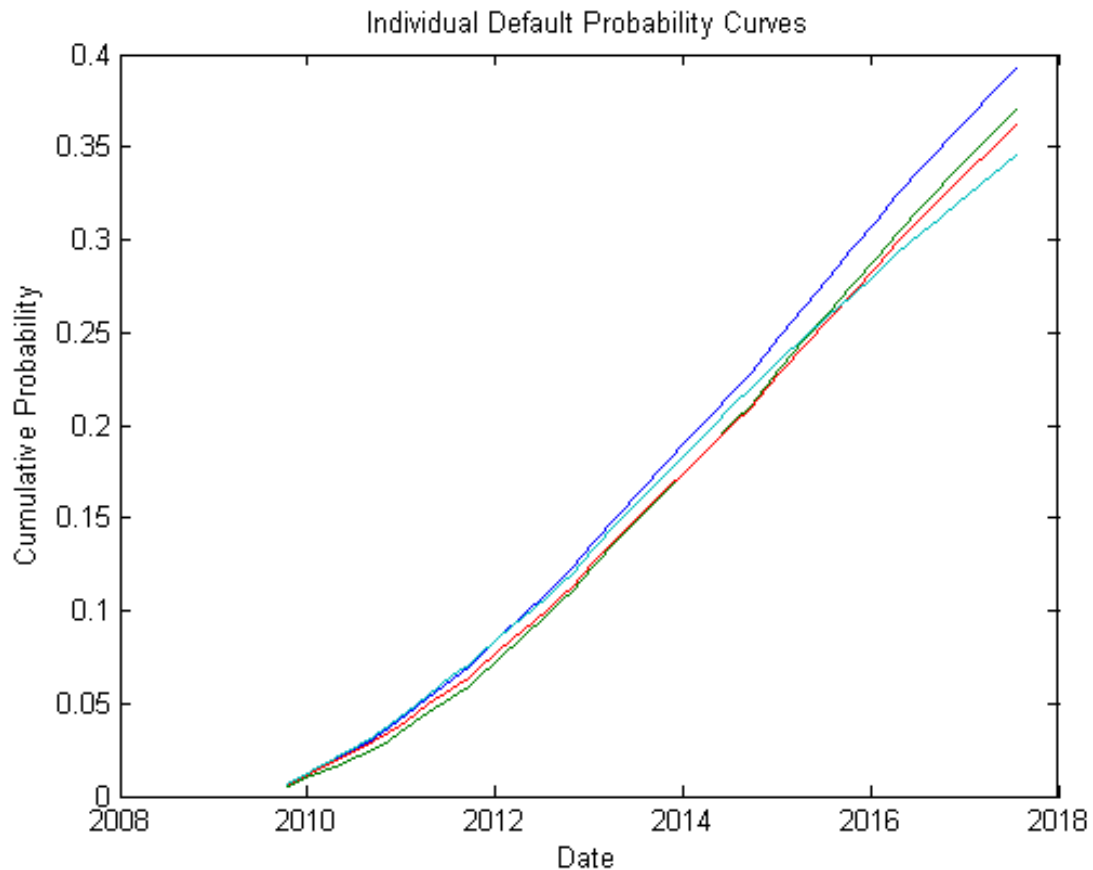
Determine Latent Variable Thresholds

Latent variables are used in different credit risk contexts, with different interpretations. In some contexts, a latent variable is a proxy for a *change in the value of assets*, and the domain of this variable is binned, with each bin corresponding to a credit rating. The bins limits, or thresholds, are determined from credit migration matrices. In our context, the latent variable is associated to a *time to default*, and the thresholds determine bins in a discretized time grid where defaults may occur.

Formally, if the time to default of a particular issuer is denoted by τ, and we know its default probability function $P(t)$, a latent variable A and corresponding thresholds $C(t)$ satisfy

$$Pr(\tau \leq t) = P(t) = Pr(A \leq C(t))$$

or

$$Pr(s \leq \tau \leq t) = P(t) - P(s) = Pr(C(s) \leq A \leq C(t))$$

These relationships make latent variable approaches convenient for both simulations and analytical derivations. Both $P(t)$ and $C(t)$ are functions of time.

The choice of a distribution for the variable A determines the thresholds $C(t)$. In the standard latent variable model, the variable A is chosen to follow a standard normal distribution, from which

$$C(t) = \Phi^{-1}(P(t))$$

where Φ is the cumulative standard normal distribution.

Use the previous formula to determine the *default-time thresholds*, or simply *default thresholds*, corresponding to the default probabilities previously obtained for the credits in the basket.

```
DefThresh = norminv(DefProb);
```

Derive Survival Curve for the Basket

Following O'Kane [2], we use a one-factor latent variable model to derive expressions for the survival probability function of the basket.

Given parameters β_i for each issuer i, and given independent standard normal variables Z and ϵ_i, the one-factor latent variable model assumes that the latent variable A_i associated to issuer i satisfies

$$A_i = \beta_i * Z + \sqrt{1 - \beta_i^2} * \epsilon_i$$

This induces a correlation between issuers i and j of $\beta_i \beta_j$. All latent variables A_i share the common factor Z as a source of uncertainty, but each latent variable also has an idiosyncratic source of uncertainty ϵ_i. The larger the coefficient β_i, the more the latent variable resembles the common factor Z.

Using the latent variable model, we derive an analytic formula for the survival probability of the basket.

The probability that issuer i survives past time t_j, in other words, that its default time τ_i is greater than t_j is

$$Pr(\tau_i > t_j) = 1 - Pr(A_i \leq C_i(t_j))$$

where $C_i(t_j)$ is the default threshold computed above for issuer i, for the j-th date in the discretization grid.

Conditional on the value of the one-factor Z, the probability that all issuers survive past time t_j is

$$Pr(\text{No defaults by time } t_j | Z) = Pr(\tau_i > t_j \text{ for all } i | Z) = \prod_i [1 - Pr(A_i \leq C_i(t_j) | Z)]$$

where the product is justified because all the ϵ_i's are independent. Therefore, conditional on Z, the A_i's are independent.

The unconditional probability of no defaults by time t_j is the integral over all values of Z of the previous conditional probability

$$Pr(\text{No defaults by time } t_j) = \int_Z \prod_i [1 - Pr(A_i \leq C_i(t_j) | Z)] \phi(Z) dZ$$

with $\phi(Z)$ the standard normal density.

By evaluating this one-dimensional integral for each point t_j in the grid, we get a discretization of the survival curve for the whole basket, which is the FTD survival curve.

The latent variable model can also be used to simulate default times, which is the back engine of many pricing methodologies for credit instruments.

Loeffler and Posch [1], for example, estimate the survival probability of a basket via simulation. In each simulated scenario a time to default is determined for each issuer. With some bookkeeping, the probability of having the first default on each bucket of the grid can be estimated from the simulation. The simulation approach is also discussed in O'Kane [2]. Simulation is very flexible and applicable to many credit instruments. However, analytic approaches are preferred, when available, because they are much faster and more accurate than simulation.

To compute the FTD survival probabilities in our example, we set all betas to the square root of a target correlation. Then we loop over all dates in the time grid to compute the one dimensional integral that gives the survival probability of the basket.

Regarding implementation, the conditional survival probability as a function of a scalar Z would be

```
condProb=@(Z)prod(normcdf((-DefThresh(:,jj)+beta*Z)./sqrt(1-beta.^2)));
```

However, the integration function we use requires that the function handle of the integrand accepts vectors. Although a loop around the scalar version of the conditional probability would work, it is far more efficient to vectorize the conditional probability using bsxfun.

```
beta = sqrt(0.25)*ones(nIssuers,1);

FTDSurvProb = zeros(size(ProbDates));
for jj=1:nProbDates
   % vectorized conditional probability as a function of Z
   vecCondProb = @(Z)prod(normcdf(bsxfun(@rdivide,...
      -repmat(DefThresh(:,jj),1,length(Z))+bsxfun(@times,beta,Z),...
      sqrt(1-beta.^2))));
   % truncate domain of normal distribution to [-5,5] interval
   FTDSurvProb(jj) = integral(@(Z)vecCondProb(Z).*normpdf(Z),-5,5);
end
FTDDefProb = 1-FTDSurvProb;
```

Compare the FTD probability to the default probabilities of the individual issuers.

```
figure
```

```
plot(ProbDates',DefProb)
datetick
hold on
plot(ProbDates,FTDDefProb,'LineWidth',3)
datetick
hold off
title('FTD and Individual Default Probability Curves')
ylabel('Cumulative Probability')
xlabel('Date')
```
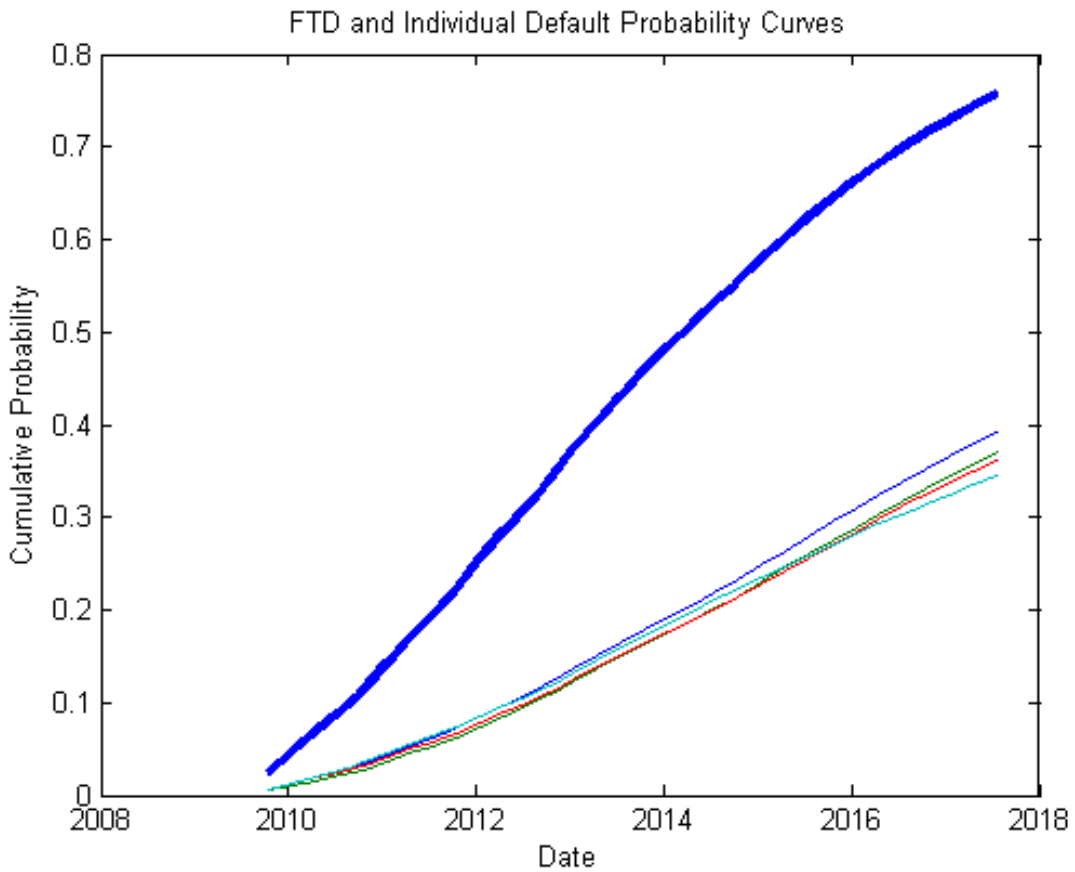
Find FTD Spreads and Price Existing FTD Swaps

Under the assumption that all instruments in the basket have the same recovery rate, or homogeneous loss assumption (see O'Kane in References), we get the spread for the FTD swap using the `cdsspread` function, but passing the FTD probability data just computed.

```
Maturity = MarketDates;
ProbDataFTD = [ProbDates, FTDDefProb];
FTDSpread = cdsspread(ZeroData,ProbDataFTD,Settle,Maturity);
```

Compare the FTD spreads with the individual spreads.

```
figure
plot(MarketDates,MarketSpreads')
datetick
hold on
plot(MarketDates,FTDSpread,'LineWidth',3)
hold off
title('FTD and Individual CDS Spreads')
ylabel('FTD Spread (bp)')
xlabel('Maturity Date')
```

FTD and Individual CDS Spreads
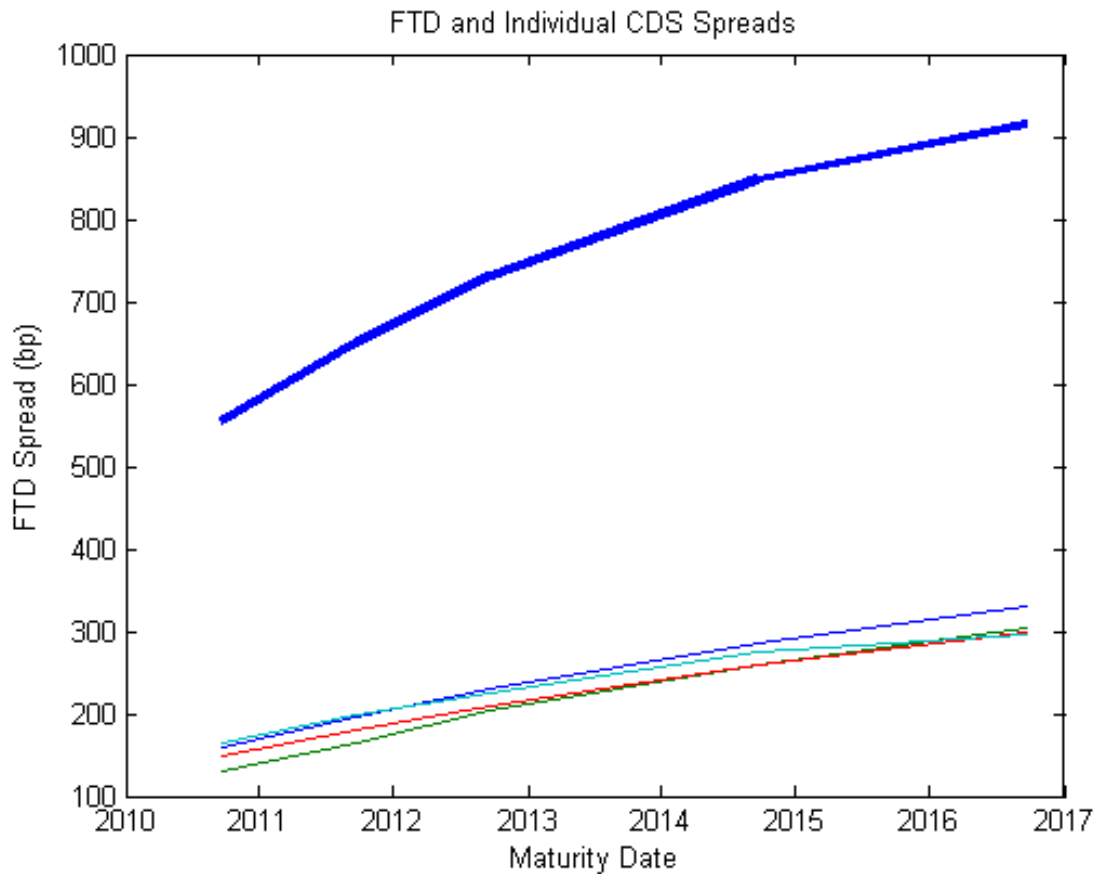
An existing FTD swap can be priced with `cdsprice`, using the same FTD probability.

```
Maturity0 = MarketDates(1); % Assume maturity on nearest market date
Spread0 = 540; % Spread of existing FTD contract
% Assume default values of recovery and notional
FTDPrice = cdsprice(ZeroData,ProbDataFTD,Settle,Maturity0,Spread0);
fprintf('Price of existing FTD contract: %g\n',FTDPrice)

Price of existing FTD contract: 17644.7
```

Analyze Sensitivity to Correlation

To illustrate the sensitivity of the FTD spreads to model parameters, we calculate the market spreads for a range of correlation values.

```
corr = [0 0.01 0.10 0.25 0.5 0.75 0.90 0.99 1];
FTDSpreadByCorr = zeros(length(Maturity),length(corr));
FTDSpreadByCorr(:,1) = sum(MarketSpreads)';
FTDSpreadByCorr(:,end) = max(MarketSpreads)';

for ii = 2:length(corr)-1
   beta = sqrt(corr(ii))*ones(nIssuers,1);
   FTDSurvProb = zeros(length(ProbDates));
   for jj=1:nProbDates
      % vectorized conditional probability as a function of Z
      condProb = @(Z)prod(normcdf(bsxfun(@rdivide,...
         -repmat(DefThresh(:,jj),1,length(Z))+bsxfun(@times,beta,Z),...
         sqrt(1-beta.^2))));
      % truncate domain of normal distribution to [-5,5] interval
      FTDSurvProb(jj) = integral(@(Z)condProb(Z).*normpdf(Z),-5,5);
   end
   FTDDefProb = 1-FTDSurvProb;
   ProbDataFTD = [ProbDates, FTDDefProb];
   FTDSpreadByCorr(:,ii) = cdsspread(ZeroData,ProbDataFTD,Settle,Maturity);
end
```

The FTD spreads lie in a band between the sum and the maximum of individual spreads. As the correlation increases to one, the FTD spreads decrease towards the maximum of the individual spreads in the basket (all credits default together). As the correlation decreases to zero, the FTD spreads approach the sum of the individual spreads (independent credits).

```
figure
legends = cell(1,length(corr));
plot(MarketDates,FTDSpreadByCorr(:,1),'k:')
legends{1} = 'Sum of Spreads';
datetick
hold on
for ii=2:length(corr)-1
```

```
    plot(MarketDates,FTDSpreadByCorr(:,ii),'LineWidth',3*corr(ii))
    legends{ii} = ['Corr ' num2str(corr(ii)*100) '%'];
end
plot(MarketDates,FTDSpreadByCorr(:,end),'k-.')
legends{end} = 'Max of Spreads';

hold off
title('FTD Spreads for Different Correlations')
ylabel('FTD Spread (bp)')
xlabel('Maturity Date')
legend(legends,'Location','NW')
```
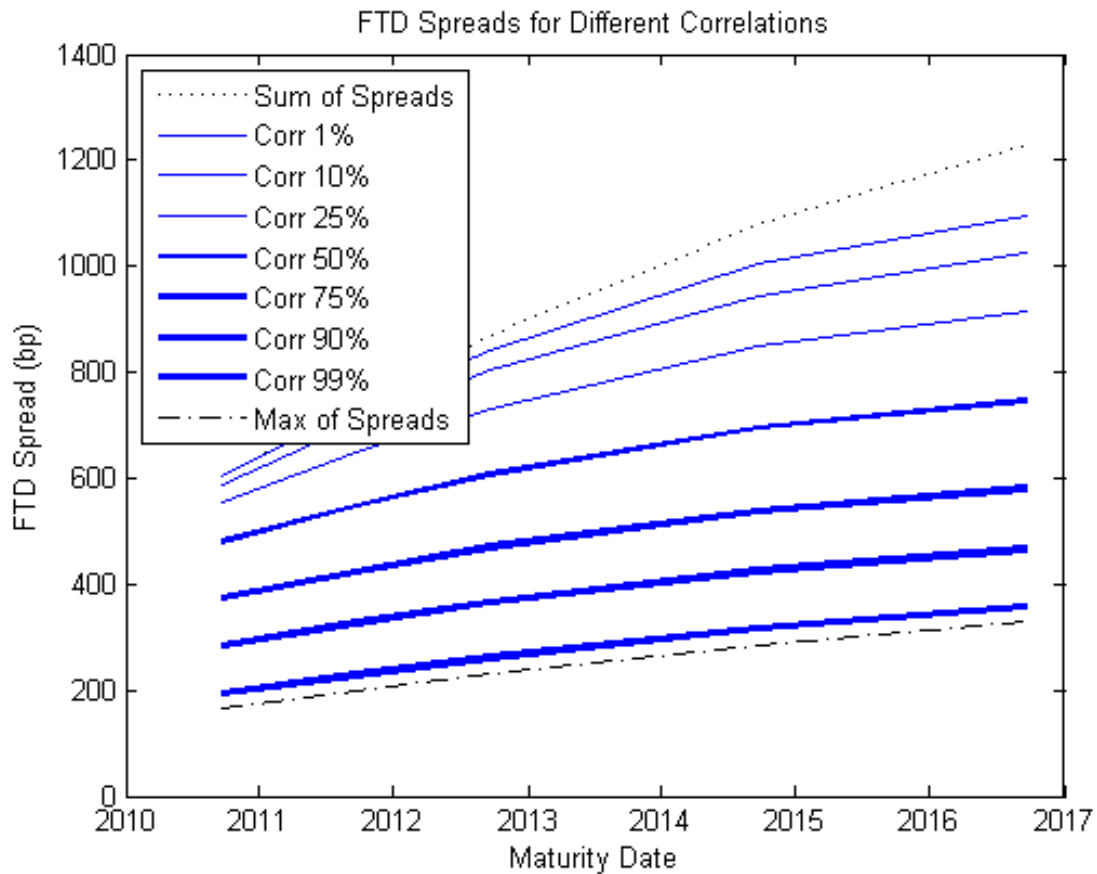
For short maturities and small correlations, the basket is effectively independent (the FTD spread is very close to the sum of individual spreads). The correlation effect becomes more significant for longer maturities.

Here is an alternative visualization of the dependency of FTD spreads on correlation.

```
figure
surf(corr,MarketDates,FTDSpreadByCorr)
datetick('y')
```

```
set(gca,'YDir','reverse')
view(-40,10)
title('FTD Spreads for Different Correlations and Maturities')
xlabel('Correlation')
ylabel('Maturity Date')
zlabel('FTD Spread (bp)')
```


FTD Spreads for Different Correlations and Maturities
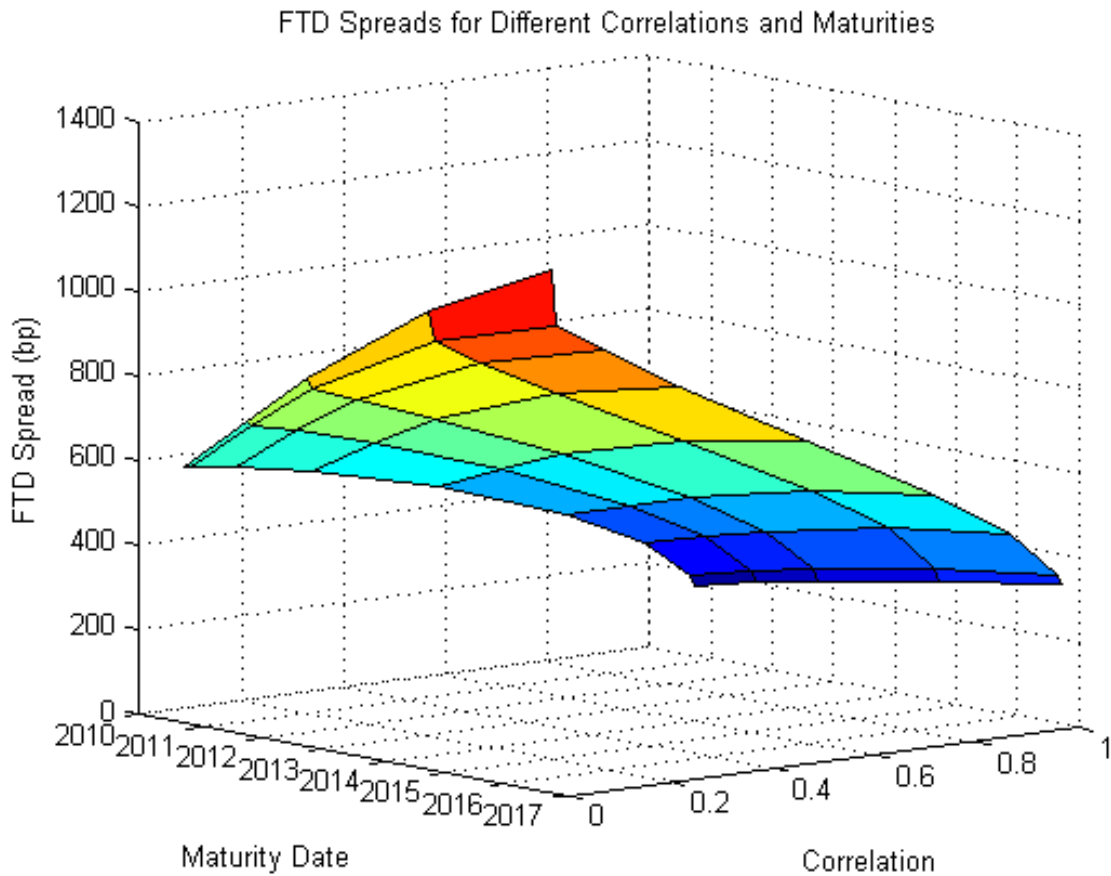
References

[1] Loeffler, Gunter and Peter Posch, *Credit risk modeling using Excel and VBA*, Wiley Finance, 2007.

[2] O'Kane, Dominic, *Modelling single-name and multi-name Credit Derivatives*, Wiley Finance, 2008.

Credit Default Swap Option

In this section...

A credit default swap (CDS) option, or credit default swaption, is a contract that provides the holder with the right, but not the obligation, to enter into a credit default swap in the future. CDS options can either be payer swaptions or receiver swaptions. If a payer swaption, the option holder has the right to enter into a CDS where they pay premiums; and, if a receiver swaption, the option holder receives premiums. Financial Instruments Toolbox software provides cdsoptprice for pricing payer and receiver credit default swaptions. Also, with some additional steps, cdsoptprice can be used for pricing multi-name CDS index options.

Pricing a Single-Name CDS Option

This example shows how to price a single-name CDS option using cdsoptprice. The function cdsoptprice is based on the Black's model as described in O'Kane (2008). The optional knockout argument for cdsoptprice supports two variations of the mechanics of a CDS option. CDS options can be knockout or non-knockout options.

- A knockout option cancels with no payments if there is a credit event before the option expiry date.

- A non-knockout option does not cancel if there is a credit event before the option expiry date. In this case, the option holder of a non-knockout payer swaption can take delivery of the underlying long protection CDS on the option expiry date and exercise the protection, delivering a defaulted obligation in return for par. This portion of protection from option initiation to option expiry is known as the front-end protection (FEP).

  While this distinction does not affect the receiver swaption, the price of a non-knockout payer swaption is obtained by adding the value of the FEP to the knockout payer swaption price.

Define the CDS instrument.

```
Settle = datenum('12-Jun-2012');
OptionMaturity = datenum('20-Sep-2012');
CDSMaturity = datenum('20-Sep-2017');
OptionStrike = 200;
SpreadVolatility = .4;
```

Define the zero rate.

```
Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [.5 .75 1.5 1.7 1.9 2.2]'/100;
Zero_Dates = daysadd(Settle,360*Zero_Time,1);
ZeroData = [Zero_Dates Zero_Rate]

ZeroData =

   1.0e+05 *

    7.3522    0.0000
    7.3540    0.0000
    7.3576    0.0000
    7.3613    0.0000
    7.3649    0.0000
    7.3686    0.0000
```

Define the market data.

```
Market_Time = [1 2 3 5 7 10]';
Market_Rate = [100 120 145 220 245 270]';
Market_Dates = daysadd(Settle,360*Market_Time,1);
MarketData = [Market_Dates Market_Rate];

ProbData = cdsbootstrap(ZeroData, MarketData, Settle)

ProbData =

   1.0e+05 *

    7.3540    0.0000
    7.3576    0.0000
```

```
7.3613     0.0000
7.3686     0.0000
7.3759     0.0000
7.3868     0.0000
```

Define the CDS option.

```
[Payer,Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, ...
    CDSMaturity, OptionStrike, SpreadVolatility, 'Knockout', true);
fprintf('   Payer: %.0f   Receiver: %.0f  (Knockout)\n',Payer,Receiver);
[Payer,Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, ...
    CDSMaturity, OptionStrike, SpreadVolatility, 'Knockout', false);
fprintf('   Payer: %.0f   Receiver: %.0f  (Non-Knockout)\n',Payer,Receiver);
```

```
  Payer: 196   Receiver: 23  (Knockout)
  Payer: 224   Receiver: 23  (Non-Knockout)
```

Pricing a CDS Index Option

This example shows how to price CDS index options by using `cdsoptprice` with the forward spread adjustment. Unlike a single-name CDS, a CDS portfolio index contains multiple credits. When one or more of the credits default, the corresponding contingent payments are made to the protection buyer but the contract still continues with reduced coupon payments. Considering the fact that the CDS index option does not cancel when some of the underlying credits default before expiry, one might attempt to price CDS index options using the Black's model for non-knockout single-name CDS option. However, Black's model in this form is not appropriate for pricing CDS index options because it does not capture the exercise decision correctly when the strike spread (K) is very high, nor does it ensure put-call parity when (K) is not equal to the contractual spread (O'Kane, 2008).

However, with the appropriate modifications, Black's model for single-name CDS options used in `cdsoptprice` can provide a good approximation for CDS index options. While there are some variations in the way the Black's model is modified for CDS index options, they usually involve adjusting the forward spread F, the strike spread K, or both. Here we describe the approach of adjusting the forward spread only. In the Black's model for single-name CDS options, the forward spread F is defined as:

$$F = S(t, t_E, T) = \frac{S(t,T)RPV01(t,T) - S(t,t_E)RPV01(t,t_E)}{RPV01(t,t_E,T)}$$

where

S is the spread.

RPV01 is the risky present value of a basis point (see `cdsrpv01`).

t is the valuation date.

t_E is the option expiry date.

T is the CDS maturity date.

To capture the exercise decision correctly for CDS index options, we use the knockout form of the Black's model and adjust the forward spread to incorporate the *FEP* as follows:

$$F_{Adj} = F + \frac{FEP}{RPV01(t, t_E, T)}$$

where the *FEP* can be expressed as

$$FEP = S(t, t_E)RPV01(t, t_E)$$

In `cdsoptprice`, forward spread adjustment can be made with the `AdjustedForwardSpread` parameter. When computing the adjusted forward spread, we can compute the spreads using `cdsspread` and the RPV01s using `cdsrpv01`.

Set up the data for the CDS index, its option, and zero curve. The underlying is a 5 year CDS index maturing on 20-Jun-2017 and the option expires on 20-Jun-2012. Note that a flat index spread is assumed when bootstrapping the default probability curve.

```
% CDS index and option data
Recovery = .4;
Basis = 2;
```

```
Period = 4;
CDSMaturity = datenum('20-Jun-2017');
ContractSpread = 100;
IndexSpread = 140;
BusDayConvention = 'follow';
Settle = datenum('13-Apr-2012');
OptionMaturity = datenum('20-Jun-2012');
OptionStrike = 140;
SpreadVolatility = .69;

% Zero curve data
MM_Time = [1 2 3 6]';
MM_Rate = [0.004111 0.00563 0.00757 0.01053]';
MM_Dates = daysadd(Settle,30*MM_Time,1);
Swap_Time = [1 2 3 4 5 6 7 8 9 10 12 15 20 30]';
Swap_Rate = [0.01387 0.01035 0.01145 0.01318 0.01508 0.01700 0.01868 ...
    0.02012 0.02132 0.02237 0.02408 0.02564 0.02612 0.02524]';
Swap_Dates = daysadd(Settle,360*Swap_Time,1);
ZeroBasis = 1;
ZeroCompounding = 1;

% Bootstrap the default probability curve assuming a flat index spread.
ZeroData = [MM_Dates MM_Rate;Swap_Dates Swap_Rate];
MarketDates = (Settle+1:30:CDSMaturity)';
MarketData = [MarketDates repmat(IndexSpread,length(MarketDates),1)];
ProbData = cdsbootstrap(ZeroData, MarketData, Settle);
```

Compute the spot and forward RPV01s, which will be used later in the computation of the FEP and the adjusted forward spread. For this purpose, we can use cdsrpv01.

```
% RPV01(t,T)
RPV01_CDSMaturity = cdsrpv01(ZeroData,ProbData,Settle,CDSMaturity,...
    'CleanRPV01',false)

% RPV01(t,t_E,T)
RPV01_OptionExpiryForward = cdsrpv01(ZeroData,ProbData,Settle,CDSMaturity,...
    'StartDate',OptionMaturity,'CleanRPV01',false)

% RPV01(t,t_E) = RPV01(t,T) - RPV01(t,t_E,T)
```

```
RPVO1_OptionExpiry = RPVO1_CDSMaturity - RPVO1_OptionExpiryForward
```

```
RPVO1_CDSMaturity =

    4.8523
```

```
RPVO1_OptionExpiryForward =

    4.5975
```

```
RPVO1_OptionExpiry =

    0.2549
```

Compute the spot spreads using cdsspread.

```
% S(t,t_E)
Spread_OptionExpiry = cdsspread(ZeroData,ProbData,Settle,OptionMaturity,...
    'Period',Period,'Basis',Basis,'BusDayConvention',BusDayConvention,...
    'PayAccruedPremium',true,'ZeroCompounding',...
    ZeroCompounding,'ZeroBasis',ZeroBasis,'recoveryrate',Recovery)

% S(t,T)
Spread_CDSMaturity = cdsspread(ZeroData,ProbData,Settle,CDSMaturity,...
    'Period',Period,'Basis',Basis,'BusDayConvention',BusDayConvention,...
    'PayAccruedPremium',true,'ZeroCompounding',...
    ZeroCompounding,'ZeroBasis',ZeroBasis,'recoveryrate',Recovery)
```

```
Spread_OptionExpiry =

  140.0005
```

```
Spread_CDSMaturity =

  140.0000
```

The spot spreads and RPV01s are then used to compute the forward spread.

```
% F = S(t,t_E,T)
ForwardSpread = (Spread_CDSMaturity.*RPV01_CDSMaturity - ...
    Spread_OptionExpiry.*RPV01_OptionExpiry)./RPV01_OptionExpiryForward
```

```
ForwardSpread =

   139.9999
```

Compute the front-end protection (FEP).

```
FEP = Spread_OptionExpiry * (cdsrpv01(ZeroData,ProbData,Settle,CDSMaturity)...
    - cdsrpv01(ZeroData,ProbData,Settle,CDSMaturity,'StartDate',OptionMaturity))
```

```
FEP =

   26.3461
```

Compute the adjusted forward spread.

```
AdjustedForwardSpread = ForwardSpread + FEP./RPV01_OptionExpiryForward
```

```
AdjustedForwardSpread =

   145.7305
```

Compute the option prices using cdsoptprice with the adjusted forward spread. Note again that the Knockout parameter should be set to be true because the FEP was already incorporated into the adjusted forward spread.

```
[Payer,Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, ...
    CDSMaturity, OptionStrike, SpreadVolatility,'Knockout',true,...
    'AdjustedForwardSpread', AdjustedForwardSpread,'PayAccruedPremium',true);
fprintf('    Payer: %.0f   Receiver: %.0f  \n',Payer,Receiver);
```

```
Payer: 92    Receiver: 66
```

References

O'Kane, D., *Modelling Single-name and Multi-name Credit Derivatives*, Wiley, 2008.

See Also   cdsoptprice | cdsspread | cdsrpv01

9

Interest-Rate Curve Objects

Interest-Rate Curve Objects

In this section...
"Class Structure" on page 9-2
"Workflow Using Interest-Rate Curve Objects" on page 9-3

Class Structure

Financial Instruments Toolbox class structure supports interest-rate curve objects. The class structure supports five classes.

Class Name	Description
"@IRCurve" on page A-4	Base abstract class for interest-rate curves. IRCurve is an abstract class; you cannot create instances of it directly. You can create IRFunctionCurve and IRDataCurve objects that are derived from this class.
"@IRDataCurve" on page A-7	Creates a representation of an interest-rate curve with dates and data. IRDataCurve is constructed directly by specifying dates and corresponding interest rates or discount factors, or you can bootstrap an IRDataCurve object from market data.
"@IRFunctionCurve" on page A-12	Creates a representation of an interest-rate curve with a function. IRFunctionCurve is constructed directly by specifying a function handle, or you can fit a function to market data using methods of the IRFunctionCurve object.
"@IRBootstrapOptions" on page A-2	The IRBootstrapOptions object lets you specify options relating to the bootstrapping of an IRDataCurve object.
"@IRFitOptions" on page A-10	The IRFitOptions object lets you specify options relating to the fitting process for an IRFunctionCurve object.

Workflow Using Interest-Rate Curve Objects

The supported workflow model for using interest-rate curve objects is:

1 Create an interest-rate curve based on an `IRDataCurve` object or an `IRFunctionCurve` object.

- To create an `IRDataCurve` object:
    - Use vectors of dates and data with interpolation methods.
    - Use bootstrapping based on market instruments.
- To create an `IRFunctionCurve` object:
    - Specify a function handle.
    - Fit a function using the Nelson-Siegel model, Svensson model, or smoothing spline model.
    - Fit a custom function.

2 Use methods of the `IRDataCurve` or `IRFunctionCurve` objects to extract forward, zero, discount factor, or par yield curves for the interest-rate curve object.

3 Convert an interest-rate curve from an `IRDataCurve` or `IRFunctionCurve` object to a `RateSpec` structure. This `RateSpec` structure is identical to the `RateSpec` produced by the Financial Instruments Toolbox function `intenvset`. Using the `RateSpec` for an interest-rate curve object, you can then use Financial Instruments Toolbox functions to model an interest-rate structure and price.

Creating Interest-Rate Curve Objects

Depending on your data and purpose for analysis, you can create an interest-rate curve object by using an `IRDataCurve` or `IRFunctionCurve` object.

To create an `IRDataCurve` object, you can:

- Use the `IRDataCurve` constructor.
- Use the `IRDataCurve` method `bootstrap`.

Using an `IRDataCurve` object, you can use the following methods to determine:

- Forward rate curve — `getForwardRates`
- Zero rate curve — `getZeroRates`
- Discount rate curve — `getDiscountFactors`
- Par yield curve — `getParYields`

Alternatively, to create an `IRFunctionCurve` object, you can:

- Use the `IRFunctionCurve` constructor and directly specify a function handle.
- Use `IRFunctionCurve` methods:
  - `fitNelsonSiegel` fits a "Fitting IRFunctionCurve Object Using Nelson-Siegel Method" on page 9-18 to market data for bonds.
  - `fitSvensson` fits a "Fitting IRFunctionCurve Object Using Svensson Method" on page 9-20 to market data for bonds.
  - `fitSmoothingSpline` fits a "Fitting IRFunctionCurve Object Using Smoothing Spline Method" on page 9-22 function to market data for bonds.
  - `fitFunction` custom fits an interest-rate curve object to market data for bonds.

Using an `IRFunctionCurve` object, you can use the following method to determine:

- Forward rate curve — `getForwardRates`

- Zero rate curve — `getZeroRates`

- Discount rate curve — `getDiscountFactors`

- Par yield curve — `getParYields`

In addition, you can convert an `IRDataCurve` or `IRFunctionCurve` to a `RateSpec` structure. For more information, see "Converting an IRDataCurve or IRFunctionCurve Object" on page 9-38.

Creating an IRDataCurve Object

In this section...
"IRDataCurve Constructor with Dates and Data" on page 9-6
"IRDataCurve Bootstrapping Based on Market Instruments" on page 9-7

IRDataCurve Constructor with Dates and Data

Use the IRDataCurve constructor with vectors of dates and data to create an interest-rate curve object. When constructing the IRDataCurve object, you can also use optional inputs to define how the interest-rate curve is constructed from the dates and data.

Example

In this example, you create the vectors for Dates and Data for an interest-rate curve:

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
```

Use the IRDataCurve constructor to build interest-rate objects based on the constant and pchip interpolation methods:

```
irdc_const = IRDataCurve('Forward',today,Dates,Data,'InterpMethod','constant');
irdc_pchip = IRDataCurve('Forward',today,Dates,Data,'InterpMethod','pchip');
```

Plot the forward and zero rate curves for the two IRDataCurve objects based on constant and pchip interpolation methods:

```
PlottingDates = daysadd(today,180:10:360*30,1);
plot(PlottingDates,irdc_const.getForwardRates(PlottingDates),'b')
hold on
plot(PlottingDates,irdc_pchip.getForwardRates(PlottingDates),'r')
plot(PlottingDates,irdc_const.getZeroRates(PlottingDates),'g')
plot(PlottingDates,irdc_pchip.getZeroRates(PlottingDates),'yellow')
legend({'Constant Forward Rates','PCHIP Forward Rates','Constant Zero Rates',...
'PCHIP Zero Rates'},'location','SouthEast')
title('Interpolation methods for IRDataCurve objects')
```

```
datetick
```

The plot demonstrates the relationship of the forward and zero rate curves.
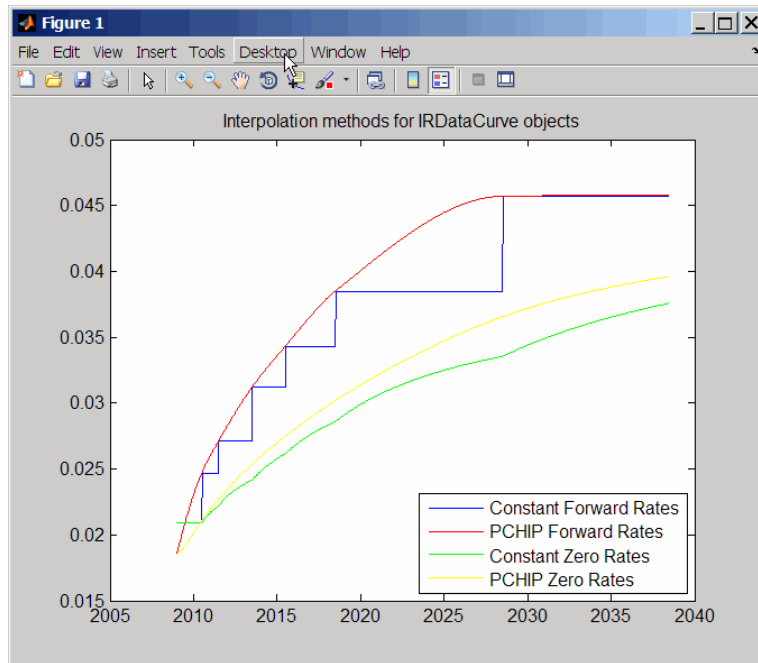
IRDataCurve Bootstrapping Based on Market Instruments

Use the bootstrapping method, based on market instruments, to create an interest-rate curve object. When bootstrapping, you also have the option to define a range of interpolation methods (`linear`, `spline`, `constant`, and `pchip`).

Example 1

In this example, you bootstrap a swap curve from deposits, Eurodollar Futures and swaps. The input market data for this example is hard-coded and specified as two cell arrays of data; one cell array indicates the type of instrument and the other contains the `Settle`, `Maturity` values and a market

quote for the instrument. For deposits and swaps, the quote is a rate; for the EuroDollar futures, the quote is a price. Although bonds are not used in this example, a bond would also be quoted with a price.

```
InstrumentTypes = {'Deposit';'Deposit';'Deposit';'Deposit';'Deposit'; ...
    'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Swap';'Swap';'Swap';'Swap';'Swap';'Swap';'Swap'};

Instruments = [datenum('08/10/2007'),datenum('08/17/2007'),.0532063; ...
    datenum('08/10/2007'),datenum('08/24/2007'),.0532000; ...
    datenum('08/10/2007'),datenum('09/17/2007'),.0532000; ...
    datenum('08/10/2007'),datenum('10/17/2007'),.0534000; ...
    datenum('08/10/2007'),datenum('11/17/2007'),.0535866; ...
    datenum('08/08/2007'),datenum('19-Dec-2007'),9485; ...
    datenum('08/08/2007'),datenum('19-Mar-2008'),9502; ...
    datenum('08/08/2007'),datenum('18-Jun-2008'),9509.5; ...
    datenum('08/08/2007'),datenum('17-Sep-2008'),9509; ...
    datenum('08/08/2007'),datenum('17-Dec-2008'),9505.5; ...
    datenum('08/08/2007'),datenum('18-Mar-2009'),9501; ...
    datenum('08/08/2007'),datenum('17-Jun-2009'),9494.5; ...
    datenum('08/08/2007'),datenum('16-Sep-2009'),9489; ...
    datenum('08/08/2007'),datenum('16-Dec-2009'),9481.5; ...
    datenum('08/08/2007'),datenum('17-Mar-2010'),9478; ...
    datenum('08/08/2007'),datenum('16-Jun-2010'),9474; ...
    datenum('08/08/2007'),datenum('15-Sep-2010'),9469.5; ...
    datenum('08/08/2007'),datenum('15-Dec-2010'),9464.5; ...
    datenum('08/08/2007'),datenum('16-Mar-2011'),9462.5; ...
    datenum('08/08/2007'),datenum('15-Jun-2011'),9456.5; ...
    datenum('08/08/2007'),datenum('21-Sep-2011'),9454; ...
    datenum('08/08/2007'),datenum('21-Dec-2011'),9449.5; ...
    datenum('08/08/2007'),datenum('08/08/2014'),.0530; ...
    datenum('08/08/2007'),datenum('08/08/2017'),.0545; ...
    datenum('08/08/2007'),datenum('08/08/2019'),.0551; ...
    datenum('08/08/2007'),datenum('08/08/2022'),.0559; ...
    datenum('08/08/2007'),datenum('08/08/2027'),.0565; ...
```

```
    datenum('08/08/2007'),datenum('08/08/2032'),.0566; ...
    datenum('08/08/2007'),datenum('08/08/2037'),.0566];
```

The `bootstrap` method is called as a static method of the "@IRDataCurve"
on page A-7 class. Inputs to this method include the curve `Type` (`zero`
or `forward`), `Settle` date, `InstrumentTypes`, and `Instrument` data.
The `bootstrap` method also supports optional arguments, including an
interpolation method, compounding, basis, and an options structure for
bootstrapping. For example, you are passing in an "@IRBootstrapOptions"
on page A-2 object which includes information for the `ConvexityAdjustment`
to forward rates.

```
IRsigma = .01;
CurveSettle = datenum('08/10/2007');
bootModel = IRDataCurve.bootstrap('Forward', CurveSettle, ...
InstrumentTypes, Instruments,'InterpMethod','pchip',...
'Compounding',-1,'IRBootstrapOptions',...
IRBootstrapOptions('ConvexityAdjustment',@(t) .5*IRsigma^2.*t.^2))

bootModel =

IRDataCurve

    Type: Forward
     Settle: 733264 (10-Aug-2007)
   Compounding: -1
   Basis: O (actual/actual)
  InterpMethod: pchip
   Dates: [29x1 double]
    Data: [29x1 double]
```

The `bootstrap` method uses an Optimization Toolbox function to solve for
any bootstrapped rates.

Plot the forward and zero curves:

```
PlottingDates = (CurveSettle+20:30:CurveSettle+365*25)';
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);
BootstrappedForwardRates = bootModel.getForwardRates(PlottingDates);
BootstrappedZeroRates = bootModel.getZeroRates(PlottingDates);
```

```
figure
hold on
plot(TimeToMaturity,BootstrappedForwardRates,'r')
plot(TimeToMaturity,BootstrappedZeroRates,'g')
title('Bootstrapped Curve')
xlabel('Time')
legend({'Forward','Zero'})
```

The plot demonstrates the forward and zero rate curves for the market data.
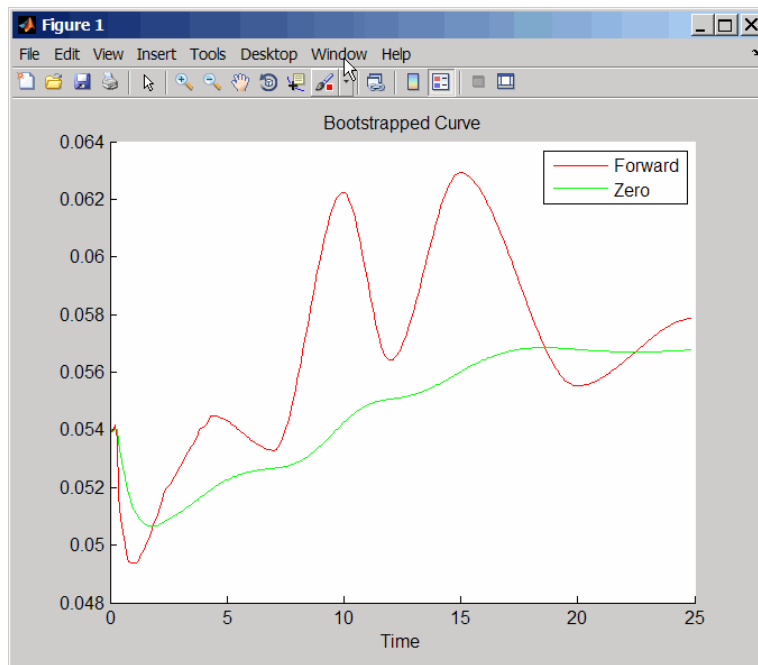
Example 2

In this example, you bootstrap a swap curve from deposits, Eurodollar futures and swaps. The input market data for this example is hard-coded and specified as two cell arrays of data; one cell array indicates the type of instrument and the other cell array contains the `Settle`, `Maturity` values and a market quote for the instrument. This example of bootstrapping also demonstrates the use of an `InstrumentBasis` for each `Instrument` type:

```
InstrumentTypes = {'Deposit';'Deposit';...
'Futures';'Futures';'Futures';'Futures';'Futures';'Futures';...
'Swap';'Swap';'Swap';'Swap';};

Instruments = [datenum('08/10/2007'),datenum('09/17/2007'),.0532000; ...
datenum('08/10/2007'),datenum('11/17/2007'),.0535866; ...
datenum('08/08/2007'),datenum('19-Dec-2007'),9485; ...
datenum('08/08/2007'),datenum('19-Mar-2008'),9502; ...
datenum('08/08/2007'),datenum('18-Jun-2008'),9509.5; ...
datenum('08/08/2007'),datenum('17-Sep-2008'),9509; ...
datenum('08/08/2007'),datenum('17-Dec-2008'),9505.5; ...
datenum('08/08/2007'),datenum('18-Mar-2009'),9501; ...
datenum('08/08/2007'),datenum('08/08/2014'),.0530; ...
datenum('08/08/2007'),datenum('08/08/2019'),.0551; ...
datenum('08/08/2007'),datenum('08/08/2027'),.0565; ...
datenum('08/08/2007'),datenum('08/08/2037'),.0566];

CurveSettle = datenum('08/10/2007');
```

The `bootstrap` method is called as a static method of the "@IRDataCurve"
on page A-7 class. Inputs to this method include the curve `Type` (`zero`
or `forward`), `Settle` date, `InstrumentTypes`, and `Instrument` data.
The `bootstrap` method also supports optional arguments, including an
interpolation method, compounding, basis, and an options structure for
bootstrapping. In this example, you are passing an additional `Basis` value for
each instrument type:

```
bootModel=IRDataCurve.bootstrap('Forward',CurveSettle,InstrumentTypes, ...
Instruments,'InterpMethod','pchip','InstrumentBasis',[repmat(2,8,1);repmat(0,4,1)])

bootModel =

 IRDataCurve

    Type: Forward
     Settle: 733264 (10-Aug-2007)
   Compounding: 2
   Basis: O (actual/actual)
  InterpMethod: pchip
   Dates: [12x1 double]
```

```
Data: [12x1 double]
```

The `bootstrap` method uses an Optimization Toolbox function to solve for any bootstrapped rates.

Plot the par yields curve using the `getParYields` method:

```
PlottingDates = (datenum('08/11/2007'):30:CurveSettle+365*25)';
plot(PlottingDates,bootModel.getParYields(PlottingDates),'r')
datetick
```

The plot demonstrates the par yields curve for the market data.
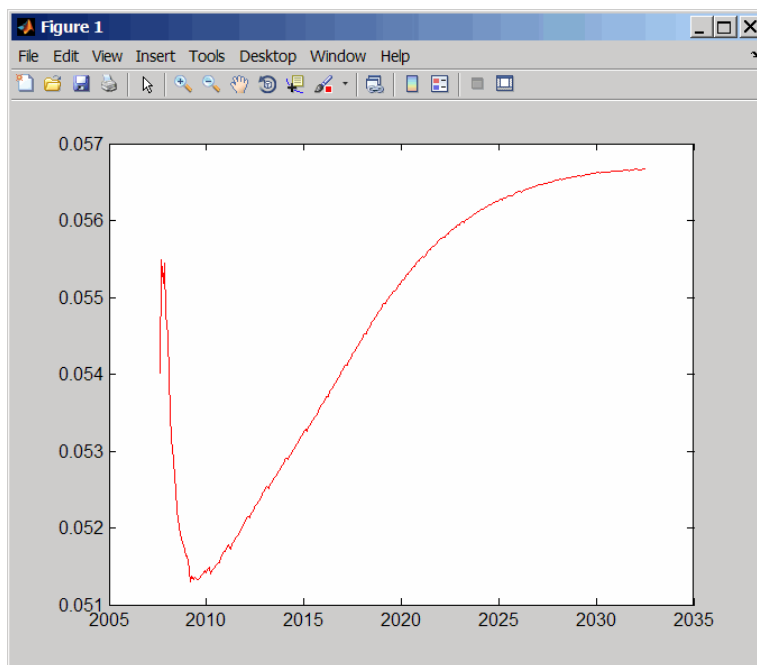
Bootstrapping a Swap Curve

This example shows how to bootstrap an interest rate curve, often referred to as a swap curve, using the new IRDataCurve object. The static bootstrap method takes as inputs a cell array of market instruments (which can be deposits, interest rate futures, swaps, and bonds) and bootstraps an interest rate curve of either the forward or the zero curve. It is also possible to specify multiple interpolation methods, including piecewise constant, linear, and Piecewise Cubic Hermite Interpolating Polynomial (PCHIP)

Obtain Data

A curve is bootstrapped from market data. In this example, we will bootstrap a swap curve from deposits, Eurodollar Futures and swaps.

For this example, we have hard-coded the input market data, which is simply specified as 2 cell arrays of data, one which indicates the type of instrument and a second cell array containing the Settle, Maturity, and Market Quote for the instrument. For deposits and swaps, the quote is a rate and for the EuroDollar Futures, the quote is a price. Although bonds are not used in this example, a bond would be quoted with a price.

```
InstrumentTypes = {'Deposit';'Deposit';'Deposit';'Deposit';'Deposit'; ...
    'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Swap';'Swap';'Swap';'Swap';'Swap';'Swap';'Swap'};

Instruments = [datenum('08/10/2007'),datenum('08/17/2007'),.0532063; ...
    datenum('08/10/2007'),datenum('08/24/2007'),.0532000; ...
    datenum('08/10/2007'),datenum('09/17/2007'),.0532000; ...
    datenum('08/10/2007'),datenum('10/17/2007'),.0534000; ...
    datenum('08/10/2007'),datenum('11/17/2007'),.0535866; ...
    datenum('08/08/2007'),datenum('19-Dec-2007'),9485; ...
    datenum('08/08/2007'),datenum('19-Mar-2008'),9502; ...
    datenum('08/08/2007'),datenum('18-Jun-2008'),9509.5; ...
    datenum('08/08/2007'),datenum('17-Sep-2008'),9509; ...
```

```
datenum('08/08/2007'),datenum('17-Dec-2008'),9505.5; ...
datenum('08/08/2007'),datenum('18-Mar-2009'),9501; ...
datenum('08/08/2007'),datenum('17-Jun-2009'),9494.5; ...
datenum('08/08/2007'),datenum('16-Sep-2009'),9489; ...
datenum('08/08/2007'),datenum('16-Dec-2009'),9481.5; ...
datenum('08/08/2007'),datenum('17-Mar-2010'),9478; ...
datenum('08/08/2007'),datenum('16-Jun-2010'),9474; ...
datenum('08/08/2007'),datenum('15-Sep-2010'),9469.5; ...
datenum('08/08/2007'),datenum('15-Dec-2010'),9464.5; ...
datenum('08/08/2007'),datenum('16-Mar-2011'),9462.5; ...
datenum('08/08/2007'),datenum('15-Jun-2011'),9456.5; ...
datenum('08/08/2007'),datenum('21-Sep-2011'),9454; ...
datenum('08/08/2007'),datenum('21-Dec-2011'),9449.5; ...
datenum('08/08/2007'),datenum('08/08/2014'),.0530; ...
datenum('08/08/2007'),datenum('08/08/2017'),.0545; ...
datenum('08/08/2007'),datenum('08/08/2019'),.0551; ...
datenum('08/08/2007'),datenum('08/08/2022'),.0559; ...
datenum('08/08/2007'),datenum('08/08/2027'),.0565; ...
datenum('08/08/2007'),datenum('08/08/2032'),.0566; ...
datenum('08/08/2007'),datenum('08/08/2037'),.0566];
```

Construct the Curve via Bootstrapping

The bootstrap method is called as a static method of the IRDataCurve class. Inputs to this method include the curve type (Zero or Forward), settle date, instrument types, instrument data and optional arguments including an interpolation method, compounding and an options structure for bootstrapping. Note that in this example, we are passing in an IRBootstrapOptions object which includes information for the convexity adjustment to forward rates.

```
IRsigma = .01;
CurveSettle = datenum('08/10/2007');
bootModel = IRDataCurve.bootstrap('Forward', CurveSettle, ...
    InstrumentTypes, Instruments,'InterpMethod','pchip',...
    'Compounding',-1,'IRBootstrapOptions',...
    IRBootstrapOptions('ConvexityAdjustment',@(t) .5*IRsigma^2.*t.^2));
```

Plot

We can now plot both the forward and zero curves

```
PlottingDates = (CurveSettle+20:30:CurveSettle+365*25)';
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);
BootstrappedForwardRates = bootModel.getForwardRates(PlottingDates);
BootstrappedZeroRates = bootModel.getZeroRates(PlottingDates);

figure
hold on
plot(TimeToMaturity,BootstrappedForwardRates,'r')
plot(TimeToMaturity,BootstrappedZeroRates,'g')
title('Bootstrapped Curve')
xlabel('Time')
legend({'Forward','Zero'})
```
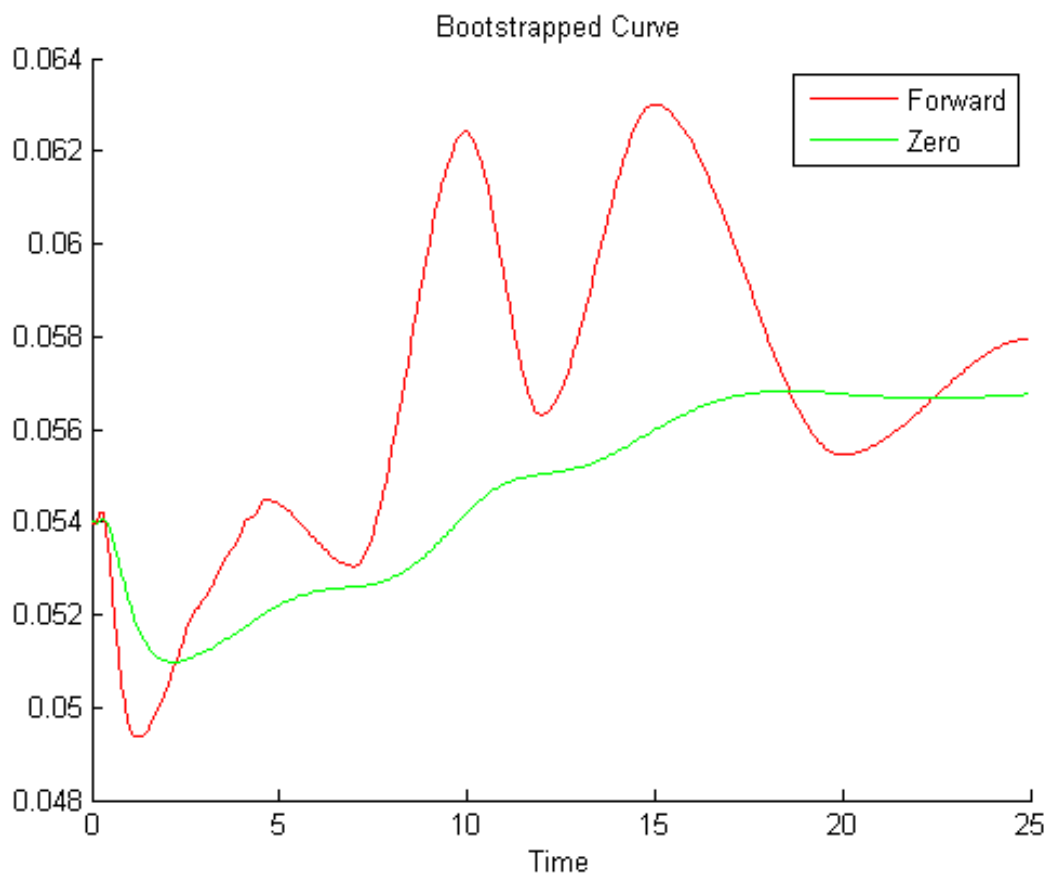
Bibliography

This example draws from the following papers and journal articles:

[1] Hagan, P., West, G. (2006), "Interpolation Methods for Curve Construction", Applied Mathematical Finance, Vol 13, No. 2

[2] Ron, Uri(2000), "A Practical Guide to Swap Curve Construction", Working Papers 00-17, Bank of Canada.

Creating an IRFunctionCurve Object

Fitting IRFunctionCurve Object Using a Function Handle

You can use the constructor `IRFunctionCurve` with a MATLAB function handle to define an interest-rate curve. For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.

Example

This example uses a `FunctionHandle` argument with a value `@(t) t.^2` to construct an interest-rate curve:

```
rr = IRFunctionCurve('Zero',today,@(t) t.^2)

rr =

  Properties:
    FunctionHandle: @(t)t.^2
              Type: 'Zero'
            Settle: 733600
       Compounding: 2
             Basis: 0
```

Fitting IRFunctionCurve Object Using Nelson-Siegel Method

Use the method, `fitNelsonSiegel`, for the Nelson-Siegel model that fits the empirical form of the yield curve with a prespecified functional form of the spot rates which is a function of the time to maturity of the bonds.

The Nelson-Siegel model represents a dynamic three-factor model: level, slope, and curvature. However, the Nelson-Siegel factors are unobserved, or latent, which allows for measurement error, and the associated loadings have economic restrictions (forward rates are always positive, and the discount factor approaches zero as maturity increases). For more information, see "Zero-coupon yield curves: technical documentation," *BIS Papers*, Bank for International Settlements, Number 25, October, 2005.

Example

This example uses `IRFunctionCurve` to model the default-free term structure of interest rates in the United Kingdom.

Load the data:

```
load ukdata20080430
```

Convert repo rates to be equivalent zero coupon bonds:

```
RepoCouponRate = repmat(0,size(RepoRates));
RepoPrice = bndprice(RepoRates, RepoCouponRate, RepoSettle, RepoMaturity);
```

Aggregate the data:

```
Settle = [RepoSettle;BondSettle];
Maturity = [RepoMaturity;BondMaturity];
CleanPrice = [RepoPrice;BondCleanPrice];
CouponRate = [RepoCouponRate;BondCouponRate];
Instruments = [Settle Maturity CleanPrice CouponRate];
InstrumentPeriod = [repmat(0,6,1);repmat(2,31,1)];
CurveSettle = datenum('30-Apr-2008');
```

The `IRFunctionCurve` object provides the capability to fit a Nelson-Siegel curve to observed market data with the `fitNelsonSiegel` method. The fitting

is done by calling the Optimization Toolbox function `lsqnonlin`. This method has required inputs of `Type`, `Settle`, and a matrix of instrument data.

```
NSModel = IRFunctionCurve.fitNelsonSiegel('Zero',CurveSettle,...
Instruments,'Compounding',-1,'InstrumentPeriod',InstrumentPeriod);
```

Plot the Nelson-Siegel interest-rate curve for forward rates:

```
PlottingDates = CurveSettle+20:30:CurveSettle+365*25;
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);
NSForwardRates = NSModel.getForwardRates(PlottingDates);
plot(TimeToMaturity,NSForwardRates)
title('Nelson Siegel model of UK instantaneous nominal forward curve')
```
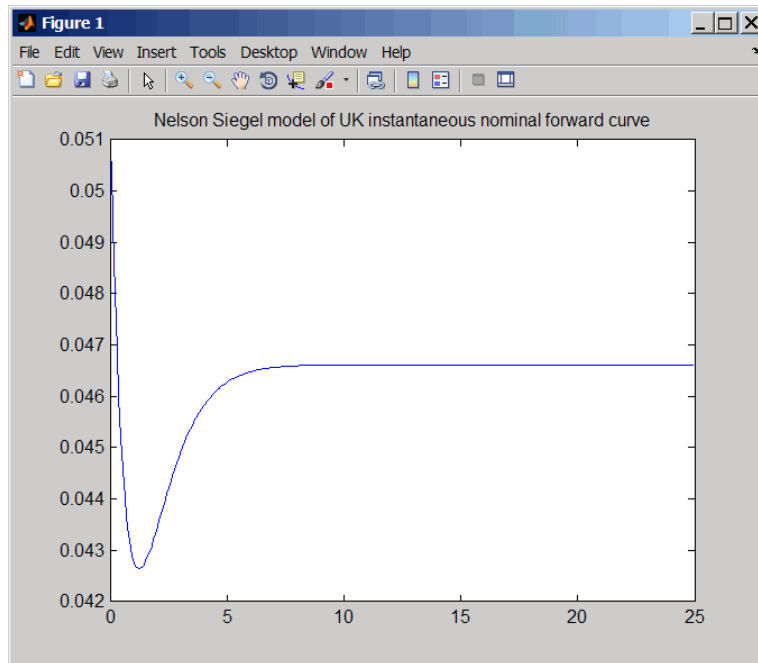
Fitting IRFunctionCurve Object Using Svensson Method

Use the method, `fitSvensson`, for the Svensson model to improve the flexibility of the curves and the fit for a Nelson-Siegel model. In 1994, Svensson extended Nelson and Siegel's function by adding a further term that allows for a second "hump." The extra precision is achieved at the cost of adding two more parameters, $\beta 3$ and $\tau 2$, which have to be estimated.

Example

In this example of using the `fitSvensson` method, an `IRFitOptions` structure, previously defined using the `IRFitOptions` constructor, is used. Thus, you must specify `FitType`, `InitialGuess`, `UpperBound`, `LowerBound`, and the `OptOptions` optimization parameters for `lsqnonlin`.

Load the data:

```
load ukdata20080430
```

Convert repo rates to be equivalent zero coupon bonds:

```
RepoCouponRate = repmat(0,size(RepoRates));
RepoPrice = bndprice(RepoRates, RepoCouponRate, RepoSettle, RepoMaturity);
```

Aggregate the data:

```
Settle = [RepoSettle;BondSettle];
Maturity = [RepoMaturity;BondMaturity];
CleanPrice = [RepoPrice;BondCleanPrice];
CouponRate = [RepoCouponRate;BondCouponRate];
Instruments = [Settle Maturity CleanPrice CouponRate];
InstrumentPeriod = [repmat(0,6,1);repmat(2,31,1)];
CurveSettle = datenum('30-Apr-2008');
```

Define `OptOptions` for the `IRFitOptions` constructor:

```
OptOptions = optimoptions('lsqnonlin','MaxFunEvals',1000);
fIRFitOptions = IRFitOptions([5.82 -2.55 -.87 0.45 3.9 0.44],...
'FitType','durationweightedprice','OptOptions',OptOptions,...
'LowerBound',[0 -Inf -Inf -Inf 0 0],'UpperBound',[Inf Inf Inf Inf Inf Inf]);
```

Fit the interest-rate curve using a Svensson model:

```
SvenssonModel = IRFunctionCurve.fitSvensson('Zero',CurveSettle,...
Instruments,'IRFitOptions',fIRFitOptions, 'Compounding',-1,...
'InstrumentPeriod',InstrumentPeriod);
```

```
Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the default value of the function tolerance.
```

The status message, output from `lsqnonlin`, indicates that the optimization to find parameters for the Svensson equation terminated successfully.

Plot the Svensson interest-rate curve for forward rates:

```
PlottingDates = CurveSettle+20:30:CurveSettle+365*25;
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);
SvenssonForwardRates = SvenssonModel.getForwardRates(PlottingDates);
plot(TimeToMaturity,SvenssonForwardRates)
title('Svensson model of UK instantaneous nominal forward curve')
```
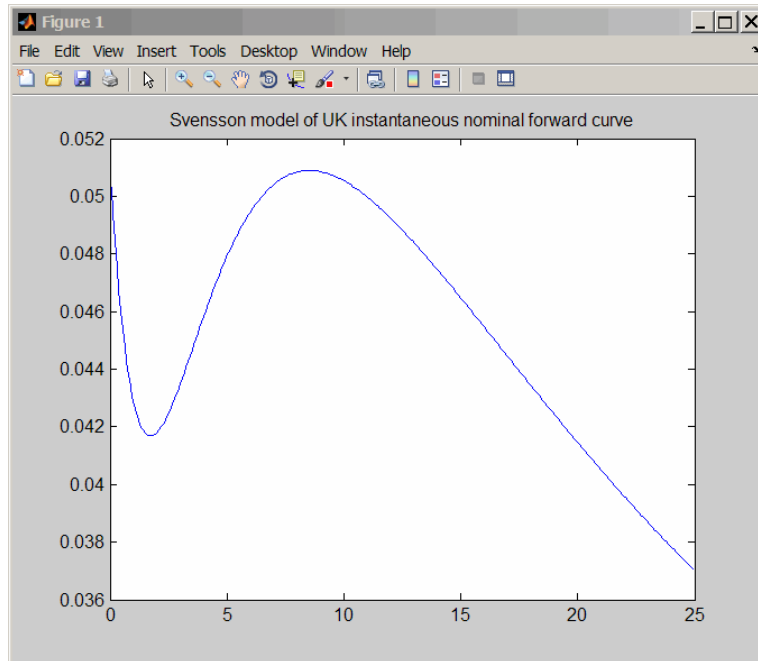
Fitting IRFunctionCurve Object Using Smoothing Spline Method

Use the method, `fitSmoothingSpline`, to model the term structure with a spline, specifically, the term structure represents the forward curve with a cubic spline.

Note You must have a license for Curve Fitting Toolbox software to use the `fitSmoothingSpline` method.

Example

The `IRFunctionCurve` object is used to fit a smoothing spline representation of the forward curve with a penalty function. Required inputs are `Type`, `Settle`, the matrix of `Instruments`, and `Lambdafun`, a function handle containing the penalty function

Load the data:

```
load ukdata20080430
```

Convert repo rates to be equivalent zero coupon bonds:

```
RepoCouponRate = repmat(0,size(RepoRates));
RepoPrice = bndprice(RepoRates, RepoCouponRate, RepoSettle, RepoMaturity);
```

Aggregate the data:

```
Settle = [RepoSettle;BondSettle];
Maturity = [RepoMaturity;BondMaturity];
CleanPrice = [RepoPrice;BondCleanPrice];
CouponRate = [RepoCouponRate;BondCouponRate];
Instruments = [Settle Maturity CleanPrice CouponRate];
InstrumentPeriod = [repmat(0,6,1);repmat(2,31,1)];
CurveSettle = datenum('30-Apr-2008');
```

Choose parameters for `Lambdafun`:

```
L = 9.2;
S = -1;
mu = 1;
```

Define the `Lambdafun` penalty function:

```
lambdafun = @(t) exp(L - (L-S)*exp(-t/mu));
t = 0:.1:25;
y = lambdafun(t);
figure
semilogy(t,y);
title('Penalty Function for VRP Approach')
ylabel('Penalty')
xlabel('Time')
```
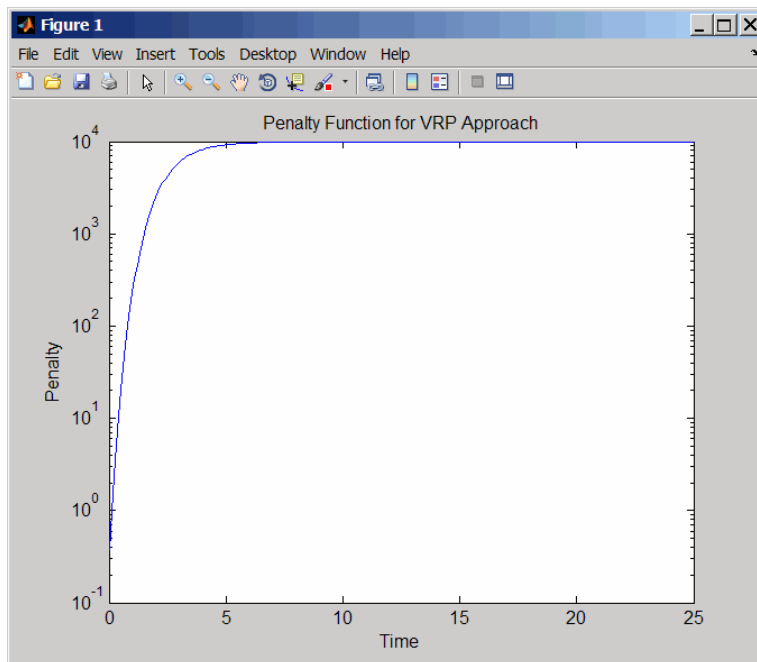
Use the `fitSmoothinSpline` method to fit the interest-rate curve and model the `Lambdafun` penalty function:

```
VRPModel = IRFunctionCurve.fitSmoothingSpline('Forward',CurveSettle,...
Instruments,lambdafun,'Compounding',-1, 'InstrumentPeriod',InstrumentPeriod);
```

Plot the smoothing spline interest-rate curve for forward rates:

```
PlottingDates = CurveSettle+20:30:CurveSettle+365*25;
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);
VRPForwardRates = VRPModel.getForwardRates(PlottingDates);
plot(TimeToMaturity,VRPForwardRates)
title('Smoothing Spline model of UK instantaneous nominal forward curve')
```
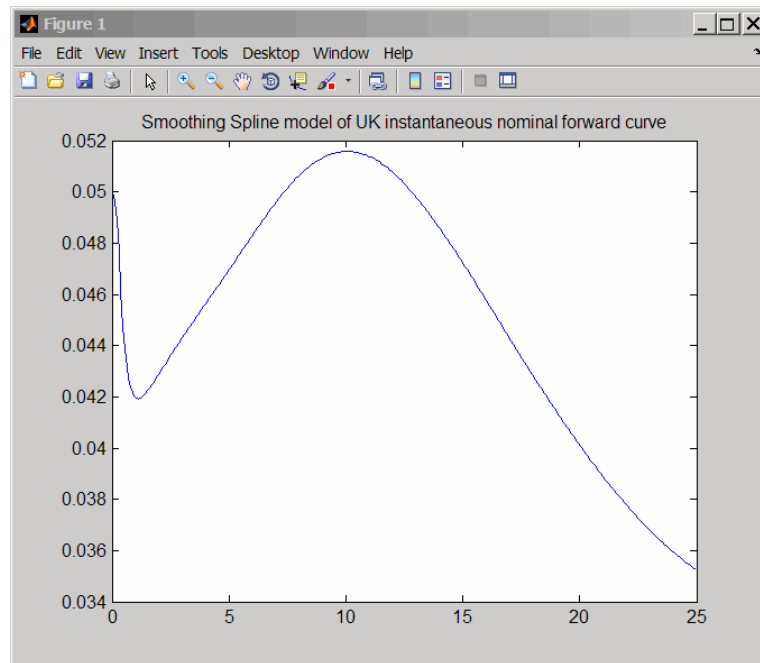
Using fitFunction to Create Custom Fitting Function

When using an `IRFunctionCurve` object, you can create a custom fitting function with the `fitFunction` method. To use `fitFunction`, you must define a `FunctionHandle`. In addition, you must also use the constructor `IRFitOptions` to define `IRFitOptionsObj` to support an `InitialGuess` for the parameters of the curve function.

Example

The following example demonstrates the use of `fitFunction` with a `FunctionHandle` and an `IRFitOptionsObj`:

```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];
```

```
CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
CurveSettle = datenum('30-Apr-2008');
```

Define the `FunctionHandle`:

```
functionHandle = @(t,theta) polyval(theta,t);
```

Define the `OptOptions` for `IRFitOptions`:

```
OptOptions = optimoptions('lsqnonlin','display','iter');
```

Define `fitFunction`:

```
CustomModel = IRFunctionCurve.fitFunction('Zero', CurveSettle, ...
functionHandle,Instruments, IRFitOptions([.05 .05 .05],'FitType','price',...
'OptOptions',OptOptions));
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
0	4	38036.7		4.92e+04	
1	8	38036.7	10	4.92e+04	0
2	12	38036.7	2.5	4.92e+04	0
3	16	38036.7	0.625	4.92e+04	0
4	20	38036.7	0.15625	4.92e+04	0
5	24	30741.5	0.0390625	1.72e+05	0
6	28	30741.5	0.078125	1.72e+05	0
7	32	30741.5	0.0195312	1.72e+05	0
8	36	28713.6	0.00488281	2.33e+05	0
9	40	20323.3	0.00976562	9.47e+05	0
10	44	20323.3	0.0195312	9.47e+05	0
11	48	20323.3	0.00488281	9.47e+05	0
12	52	20323.3	0.0012207	9.47e+05	0
13	56	19698.8	0.000305176	1.08e+06	0
14	60	17493	0.000610352	7e+06	0
15	64	17493	0.0012207	7e+06	0
16	68	17493	0.000305176	7e+06	0
17	72	15455.1	7.62939e-05	2.25e+07	0
18	76	15455.1	0.000177499	2.25e+07	0
19	80	13317.1	3.8147e-05	3.18e+07	0

20	84	12865.3	7.62939e-05	7.83e+07	0
21	88	11779.8	7.62939e-05	7.58e+06	0
22	92	11747.6	0.000152588	1.45e+05	0
23	96	11720.9	0.000305176	2.33e+05	0
24	100	11667.2	0.000610352	1.48e+05	0
25	104	11558.6	0.0012207	3.55e+05	0
26	108	11335.5	0.00244141	1.57e+05	0
27	112	10863.8	0.00488281	6.36e+05	0
28	116	9797.14	0.00976562	2.53e+05	0
29	120	6882.83	0.0195312	9.18e+05	0
30	124	6882.83	0.0373993	9.18e+05	0
31	128	3218.45	0.00934981	1.96e+06	0
32	132	612.703	0.0186996	3.01e+06	0
33	136	13.0998	0.0253882	3.05e+06	0
34	140	0.0762922	0.00154002	5.05e+04	0
35	144	0.0731652	3.61102e-06	29.9	0
36	148	0.0731652	6.32335e-08	0.063	0

```
Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the default value of the function tolerance.
```

Plot the custom function that is defined using `fitFunction`:

```
Yields = bndyield(CleanPrice,CouponRate,Settle(1),Maturity);
scatter(Maturity,Yields);
PlottingPoints = min(Maturity):30:max(Maturity);
hold on;
plot(PlottingPoints,CustomModel.getParYields(PlottingPoints),'r');
datetick
legend('Market Yields','Fitted Yield Curve')
title('Custom Function fit to Market Data')
```
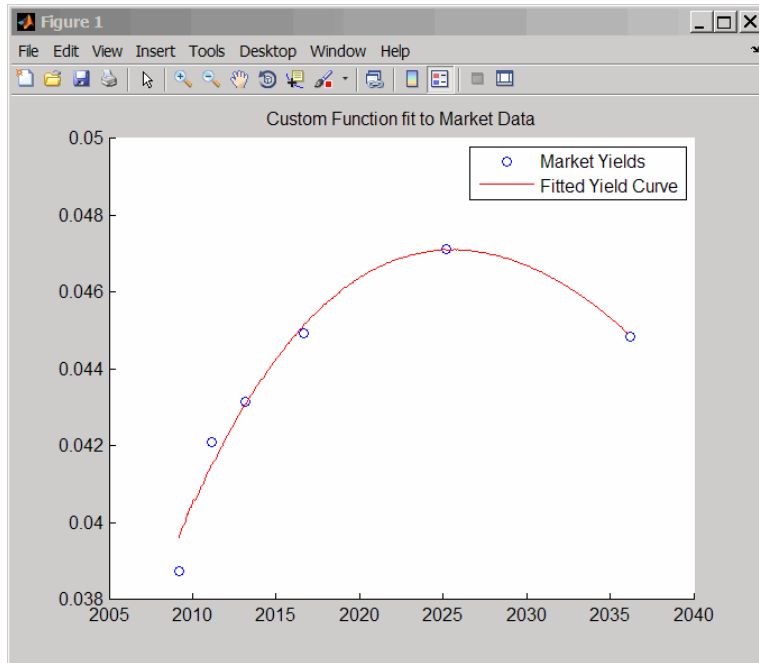
Fitting Interest Rate Curve Functions

This example shows how to use objects to model the term structure of interest rates (also referred to as the yield curve). This can be contrasted with modeling the term structure with vectors of dates and data and interpolating between the points (which can currently be done with the function PRBYZERO).

The term structure can refer to at least three different curves: the discount curve, zero curve, or forward curve.

The new object IRFunctionCurve allows the user to model an interest rate curve as a function.

This example will explore using IRFunctionCurve to model the default-free term structure of interest rates in the United Kingdom. Three different forms for the term structure will be implemented, which will be discussed in more detail later:

- Nelson-Siegel
- Svensson
- Smoothing Cubic Spline with a so-called Variable Roughness Penalty (VRP)

Choosing the Data

The first question in modeling the yield curve is what data should be used. To model a default-free yield curve, default-free, option-free market instruments must be used. The most significant component of the data set will be UK Government Bonds (known as Gilts). Historical data was retrieved from the following site:

http://www.dmo.gov.uk

Repo data will be used to construct the short end of the curve. Repo data was retrieved from the following site:

http://www.bba.org.uk

Note also that the data must be specified as a matrix where the columns are Settle, Maturity, CleanPrice and CouponRate -- and that instruments must be bonds or synthetically converted to bonds.

Market data for a close date of April 30, 2008, has been downloaded and saved to the following data file, which can be loaded into MATLAB with the following command

```
% Load the data
load ukdata20080430

% Convert repo rates to be equivalent zero coupon bonds
RepoCouponRate = repmat(0,size(RepoRates));
RepoPrice = bndprice(RepoRates, RepoCouponRate, RepoSettle, RepoMaturity);

% Aggregate the data
Settle = [RepoSettle;BondSettle];
Maturity = [RepoMaturity;BondMaturity];
CleanPrice = [RepoPrice;BondCleanPrice];
CouponRate = [RepoCouponRate;BondCouponRate];
Instruments = [Settle Maturity CleanPrice CouponRate];
InstrumentPeriod = [repmat(0,6,1);repmat(2,31,1)];

CurveSettle = datenum('30-Apr-2008');
```

Fit Nelson-Siegel Model to Market Data

The Nelson-Siegel model proposes that the instantaneous forward curve can be modeled with the following:

$$f = \beta_0 + \beta_1 e^{\frac{-m}{\tau}} + \beta_2 e^{\frac{-m}{\tau}} \frac{m}{\tau}$$

This can be integrated to derive an equation for the zero curve (see [6] for more information on the equations and the derivation):

$$s = \beta_0 + (\beta_1 + \beta_2)\frac{\tau}{m}(1 - e^{\frac{-m}{\tau}}) - \beta_2 e^{\frac{-m}{\tau}}$$

See [1] for more information.

The IRFunctionCurve object provides the capability to fit a Nelson Siegel curve to observed market data with the FITNELSONSIEGEL method. The fitting is done by calling the Optimization Toolbox™ function LSQNONLIN.

This method has required inputs: Curve Type, Curve Settle, and a matrix of instrument data.

Optional input arguments, specified in parameter value pairs, are:

• IRFitOptions structure: Provides capability to choose which quantity to be minimized (price, yield, or duration weighted price) and other optimization parameters (e.g.: upper and lower bounds for parameters)

• Curve Compounding and Basis (Day Count Convention)

• Additional instrument parameters, Period, Basis, FirstCouponDate, etc.

```
NSModel = IRFunctionCurve.fitNelsonSiegel('Zero',CurveSettle,...
    Instruments,'InstrumentPeriod',InstrumentPeriod);
```

Fit Svensson Model

A very similar model to the Nelson-Siegel is the Svensson model, which adds two additional parameters to account for greater flexibility in the term structure. This model proposes that the forward rate can be modeled with the following form:

$$f = \beta_0 + \beta_1 e^{\frac{-m}{\tau_1}} + \beta_2 e^{\frac{-m}{\tau_1}}\frac{m}{\tau_1} + \beta_3 e^{\frac{-m}{\tau_2}}\frac{m}{\tau_2}$$

As above, this can be integrated to derive an equation for the zero curve:

$$s = \beta_0 + \beta_1(1 - e^{\frac{-m}{\tau_1}})(-\frac{\tau_1}{m}) + \beta_2((1 - e^{\frac{-m}{\tau_1}})\frac{\tau_1}{m} - e^{\frac{m}{\tau_1}}) + \beta_3((1 - e^{\frac{-m}{\tau_2}})\frac{\tau_2}{m} - e^{\frac{m}{\tau_2}})$$

See [2] for more information.

Fitting the parameters to this model proceeds in a similar fashion to the Nelson-Siegel model.

```
SvenssonModel = IRFunctionCurve.fitSvensson('Zero',CurveSettle,...
    Instruments,'InstrumentPeriod',InstrumentPeriod);
```

Fit Smoothing Spline

The term structure can also be modeled with a spline -- specifically, one way
to model the term structure is by representing the forward curve with a cubic
spline. To ensure that the spline is sufficiently smooth, a penalty is imposed
relating to the curvature (second derivative) of the spline:

$$\sum_{i=1}^{N}[\frac{P_i - \hat{P}_i(f)}{D_i}]^2 + \int_0^M \lambda_t(m)[f''(m)]^2 dm$$

where the first term is the difference between the observed price P and the
predicted price, P_hat, (weighted by the bond's duration, D) summed over all
bonds in our data set and the second term is the penalty term (where lambda
is a penalty function and f is the spline)

See [3], [4], [5] below.

There have been different proposals for the specification of the penalty
function lambda. One approach, advocated by [4], and currently used by the
UK Debt Management Office, is a penalty function of the following form:

$$log(\lambda(m)) = L - (L - S)e^{\frac{-m}{\mu}}$$

The parameters L, S, and mu are typically estimated from historical data.

The IRFunctionCurve object can be used to fit a smoothing spline
representation of the forward curve with a penalty function.

Required inputs, like for methods above, are a Curve Type, Curve Settle,
Instrument matrix, and new for this method, a function handle containing
the penalty function

Optional parameters are similar to FITNELSONSIEGEL and FITSVENSSON

```
% Parameters chosen to be roughly similar to [4] below.
L = 9.2;
```

```
S = -1;
mu = 1;

lambdafun = @(t) exp(L - (L-S)*exp(-t/mu)); % Construct penalty function
t = 0:.1:25; % Construct data to plot penalty function
y = lambdafun(t);
figure
semilogy(t,y);
title('Penalty Function for VRP Approach')
ylabel('Penalty')
xlabel('Time')

VRPModel = IRFunctionCurve.fitSmoothingSpline('Forward',CurveSettle,...
    Instruments,lambdafun,'Compounding',-1,...
    'InstrumentPeriod',InstrumentPeriod);
```

Penalty Function for VRP Approach
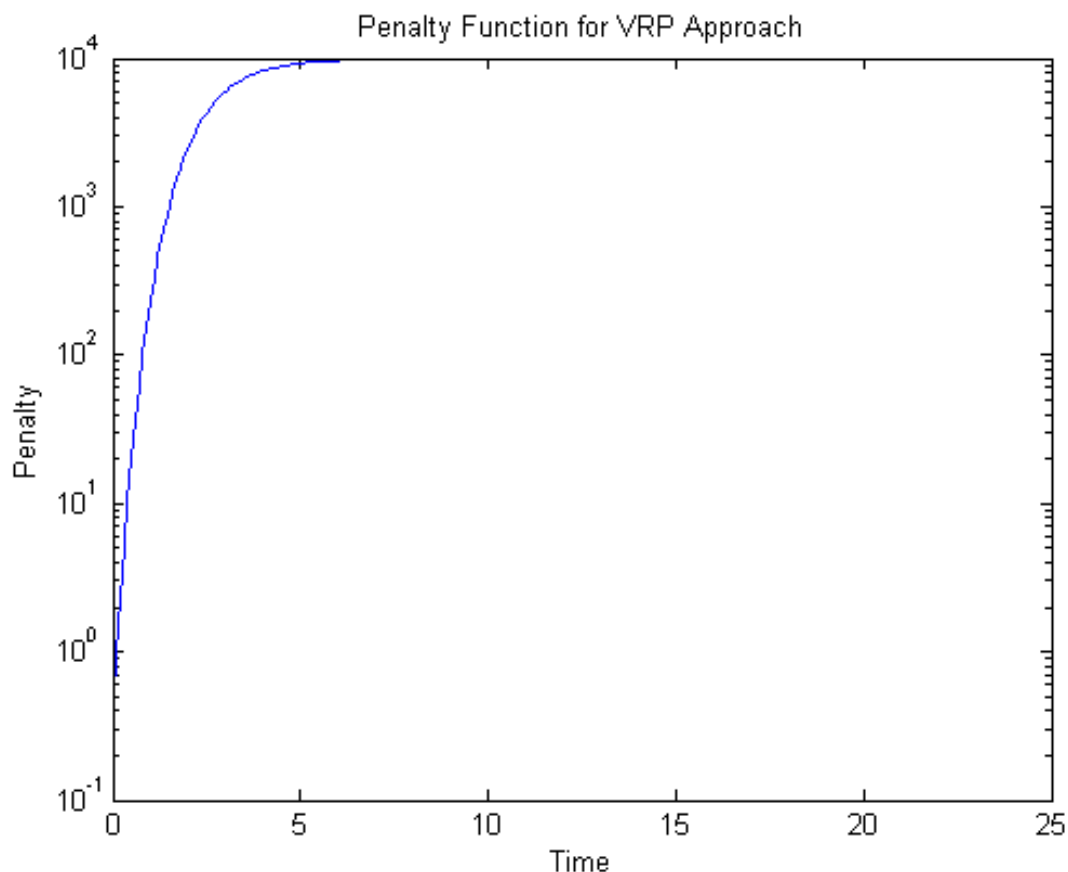
Use Fitted Curves and Plot Results

Once a curve has been constructed, methods can be called to extract the Forward and Zero Rates and the Discount Factors. This curve can also be converted into a RateSpec structure, to be used with functions in the Financial Instruments Toolbox™

```
PlottingDates = CurveSettle+20:30:CurveSettle+365*25;
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);

NSForwardRates = NSModel.getForwardRates(PlottingDates);
```

```
SvenssonForwardRates = SvenssonModel.getForwardRates(PlottingDates);
VRPForwardRates = VRPModel.getForwardRates(PlottingDates);

figure
hold on
plot(TimeToMaturity,NSForwardRates,'r')
plot(TimeToMaturity,SvenssonForwardRates,'g')
plot(TimeToMaturity,VRPForwardRates,'b')
title('UK instantaneous nominal forward curve')
xlabel('Years ahead')
legend({'Nelson Siegel','Svensson','VRP'})
```

UK instantaneous nominal forward curve
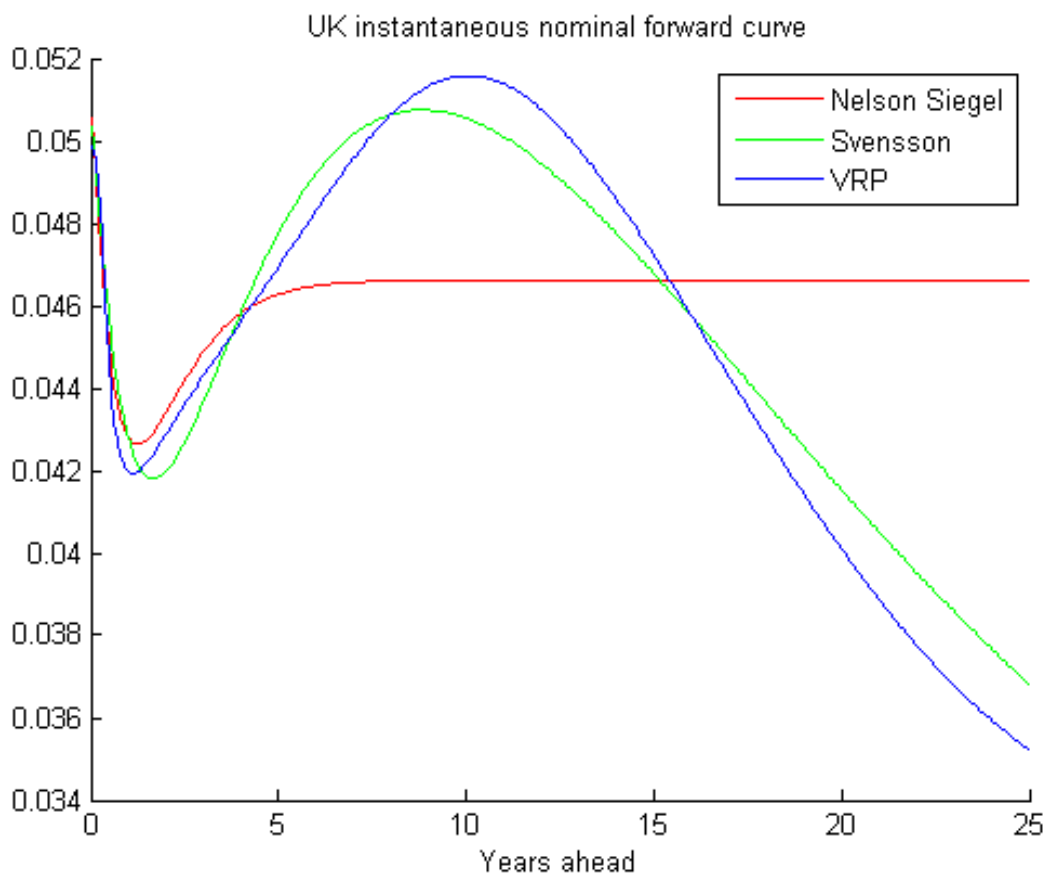
Compare with this Link

This link provides a live look at the derived yield curve published by the UK

http://www.bankofengland.co.uk

Bibliography

This example is based on the following papers and journal articles:

[1] Nelson, C.R., Siegel, A.F., (1987), "Parsimonious modelling of yield curves", Journal of Business, 60, pp 473-89

[2] Svensson, L.E.O. (1994), "Estimating and interpreting forward interest rates: Sweden 1992-4", International Monetary Fund, IMF Working Paper, 1994/114

[3] Fisher, M., Nychka, D., Zervos, D. (1995), "Fitting the term structure of interest rates with smoothing splines", Board of Governors of the Federal Reserve System, Federal Reserve Board Working Paper 95-1

[4] Anderson, N., Sleath, J. (1999), "New estimates of the UK real and nominal yield curves", Bank of England Quarterly Bulletin, November, pp 384-92

[5] Waggoner, D. (1997), "Spline Methods for Extracting Interest Rate Curves from Coupon Bond Prices", Federal Reserve Board Working Paper 97-10

[6] "Zero-coupon yield curves: technical documentation", BIS Papers No. 25 October 2005

[7] Bolder, D.J., Gusba,S (2002), "Exponentials, Polynomials, and Fourier Series: More Yield Curve Modelling at the Bank of Canada," Working Papers 02-29, Bank of Canada

[8] Bolder, D.J., Streliski, D (1999), "Yield Curve Modelling at the Bank of Canada," Technical Reports 84, Bank of Canada

Converting an IRDataCurve or IRFunctionCurve Object

In this section...

Introduction

The `IRDataCurve` and `IRFunctionCurve` objects for interest-rate curves support conversion to:

- A `RateSpec` structure. The `RateSpec` generated from an `IRDataCurve` or `IRFunctionCurve` object, using the `toRateSpec` method, is identical to the `RateSpec` structure created with `intenvset` using Financial Instruments Toolbox software.

- A vector of dates and data from an `IRDataCurve` object acceptable to `prbyzero`, `bkcall`, `bkput`, `tfutbyprice`, and `tfutbyyield` or any function that requires a term structure of interest rates.

Using the toRateSpec Method

To convert an `IRDataCurve` or `IRFunctionCurve` object to a `RateSpec` structure, you must first create an interest-rate curve object. Then, use the `toRateSpec` method for an `IRDataCurve` object or the`toRateSpec` method for an `IRFunctionCurve` object.

Example

Create a data vector from the following data:
http://www.ustreas.gov/offices/domestic-finance/debt-management/
interest-rate/yield.shtml:

```
Data = [1.85 1.84 1.91 2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[30 90 180 360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],2);
scatter(Dates,Data)
datetick
```
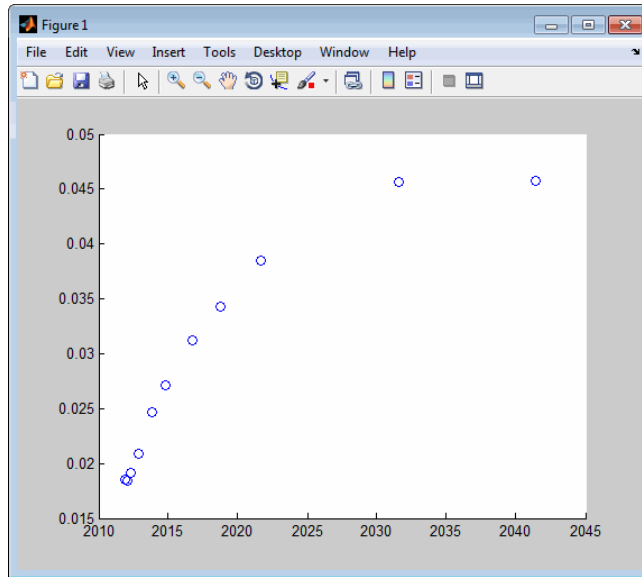
Create an IRDataCurve interest-rate curve object:

```
rr = IRDataCurve('Zero',today,Dates,Data);
```

Convert to a RateSpec:

```
rr.toRateSpec(today+30:30:today+365)

ans =
           FinObj: 'RateSpec'
      Compounding: 2
             Disc: [12x1 double]
            Rates: [12x1 double]
         EndTimes: [12x1 double]
       StartTimes: [12x1 double]
         EndDates: [12x1 double]
       StartDates: 733569
    ValuationDate: 733569
            Basis: 0
      EndMonthRule: 1
```

Using Vector of Dates and Data Methods

You can use the getZeroRates method for an IRDataCurve object with a Dates property to create a vector of dates and data acceptable for prbyzero in Financial Toolbox software and bkcall, bkput, tfutbyprice, and tfutbyyield in Financial Instruments Toolbox software.

Example

This is an example of using the IRDataCurve method getZeroRates with prbyzero:

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data,'InterpMethod','pchip');
Maturity = daysadd(today,8*360,1);
CouponRate = .055;
ZeroDates = daysadd(today,180:180:8*360,1);
ZeroRates = irdc.getZeroRates(ZeroDates);
BondPrice = prbyzero([Maturity CouponRate], today, ZeroRates, ZeroDates)

BondPrice =
  113.9221
```

10

Numerix Workflows

Working with Simple Numerix Trades

This example shows how to price a callable reverse floater using Numerix®
CAIL.

Initialize Numerix environment.

```
import com.numerix.integration.*;
import com.numerix.integration.implementation.*;

n = numerix('i:\NumeriX_java_10_3_0\data')

n =

              Path: 'i:\NumeriX_java_10_3_0\data'
    RepositoryPath: 'i:\NumeriX_java_10_3_0\data\Repository'
        Repository: [1x1 com.numerix.integration.implementation.FileSystemRepository]
           Context: [1x1 com.numerix.integration.implementation.LocalCalculationContext]
       LookupsPath: 'i:\NumeriX_java_10_3_0\data\Data\LookupRules'
       MarketsPath: 'i:\NumeriX_java_10_3_0\data\Data\Markets'
       FixingsPath: 'i:\NumeriX_java_10_3_0\data\Data\Fixings'
        TradesPath: 'i:\NumeriX_java_10_3_0\data\Data\Trades'
        Parameters: [1x1 com.numerix.integration.implementation.CalculationParameters]
```

Create a market.

```
quotes = java.util.HashMap;
quotes.put('IR.USD-LIBOR-3M.SWAP-1Y.MID', 0.0066056);
quotes.put('IR.USD-LIBOR-3M.SWAP-10Y.MID', 0.022465005);
quotes.put('IR.USD-LIBOR-3M.SWAP-20Y.MID', 0.027544995);
market = Market('EOD_14-NOV-2011', DateExtensions.date('14-Nov-2011'), quotes.entrySet);
```

Define a trade instance for a callable reverse floater based on instrument
template located in the `Repository`.

```
tradeDescriptor = 'TRADE.IR.CALLABLEREVERSEFLOATER';
tradeParameters = java.util.HashMap;
tradeParameters.put('Trade ID','1001');
tradeParameters.put('Quote Type', 'MID');
tradeParameters.put('Currency', 'USD');
tradeParameters.put('Notional', 1000000.0);
```

```
tradeParameters.put('Effective Date', DateExtensions.date('1-Dec-2011'));
tradeParameters.put('Termination Date', DateExtensions.date('1-Dec-2021'));
tradeParameters.put('IR Index', 'LIBOR');
tradeParameters.put('IR Index Tenor', '3M');
tradeParameters.put('Structured Freq', '3M');
tradeParameters.put('Structured Side', 'Receive');
tradeParameters.put('Structured Coupon Floor', 0.0);
tradeParameters.put('Structured Coupon UpBd', 0.08);
tradeParameters.put('StructuredCoupon Multiplier', 1.4);
tradeParameters.put('Structured Coupon Cap', 0.05);
tradeParameters.put('Structured Basis', 'ACT/360');
tradeParameters.put('Funding Freq', '3M');
tradeParameters.put('Funding Side', 'Pay');
tradeParameters.put('Funding Spread', 0.003);
tradeParameters.put('Funding Basis', 'ACT/360');
tradeParameters.put('Call Start Date', DateExtensions.date('1-Dec-2013'));
tradeParameters.put('Call End Date', DateExtensions.date('1-Dec-2020'));
tradeParameters.put('Option Side', 'Short');
tradeParameters.put('Option Type', 'Right to Terminate');
tradeParameters.put('Call Frequency', '3M');
tradeParameters.put('Model', 'IR.USD-LIBOR-3M.MID.DET');
tradeParameters.put('Method', 'BackwardAnalytic');
```

Create the trade instance.

```
trade = RepositoryExtensions.createTradeInstance(n.Repository, tradeDescriptor, tradeParameters)
```

Price the trade.

```
results = CalculationContextExtensions.calculate(n.Context, trade, market, Request.getAll);
```

Parse the results for MATLAB and display.

```
r = n.parseResults(results)
disp([r.Name r.Category r.Currency r.Data])
```

```
r =

    Category: {13x1 cell}
    Currency: {13x1 cell}
        Name: {13x1 cell}
```

```
       Data: {13x1 cell}

'Reporting Currency'        'Price'       ''        'USD'
'Structured Cashflow Log'   'Cashflow'    ''        {41x20 cell}
'Structured Leg PV Accrued' 'Price'       'USD'     [            0]
'PV'                        'Price'       'USD'     [  6.4133e+04]
'Structured Leg PV Clean'   'Price'       'USD'     [  4.2637e+05]
'Option PV'                 'Price'       'USD'     [-1.3220e+05]
'Funding Cashflow Log'      'Cashflow'    ''        {41x20 cell}
'Structured Leg PV'         'Price'       'USD'     [  4.2637e+05]
'Funding Leg PV'            'Price'       'USD'     [-2.3004e+05]
'Funding Leg PV Accrued'    'Price'       'USD'     [            0]
'Funding Leg PV Clean'      'Price'       'USD'     [-2.3004e+05]
'Yield Risk Report'         ''            ''        { 4x30 cell}
'Messages'                  ''            ''        { 4x1  cell}
```

See Also   numerix **|** parseResults **|**

Related Examples
- "Working with Advanced Numerix Trades" on page 10-5
- "Use Numerix to Price Cash Deposits" on page 10-12
- "Use Numerix for Interest-Rate Risk Assessment" on page 10-15

External Web Sites
- http://www.numerix.com/cail

Working with Advanced Numerix Trades

This example shows how to price multiple trades from MATLAB using Numerix CAIL.

Initialize Numerix environment.

```
import com.numerix.integration.*;
import com.numerix.integration.implementation.*;

n = numerix('i:\NumeriX_java_10_3_0\data')


n =


            Path: 'i:\NumeriX_java_10_3_0\data'
  RepositoryPath: 'i:\NumeriX_java_10_3_0\data\Repository'
      Repository: [1x1 com.numerix.integration.implementation.FileSystemRepository]
         Context: [1x1 com.numerix.integration.implementation.LocalCalculationContext]
     LookupsPath: 'i:\NumeriX_java_10_3_0\data\Data\LookupRules'
     MarketsPath: 'i:\NumeriX_java_10_3_0\data\Data\Markets'
     FixingsPath: 'i:\NumeriX_java_10_3_0\data\Data\Fixings'
      TradesPath: 'i:\NumeriX_java_10_3_0\data\Data\Trades'
      Parameters: [1x1 com.numerix.integration.implementation.CalculationParameters]
```

Specify the hybrid model for multiple trades.

```
hySpec = HybridModelSpecification;
hySpec.addHW1F('IR-USD', 'USD', 'LIBOR', '3M', 'MeanReversion(0.5),DiagonalSwaption(ATM, 10Y)');
hySpec.addHW1F('IR-EUR', 'EUR', 'EURIBOR', '6M', 'MeanReversion(0.5),DiagonalSwaption(ATM, 10Y)');
hySpec.addFXBlack('FX-USDEUR', 'USD', 'EUR', 'LIBOR', '3M', 'EURIBOR', '6M', 'StrikeFXEuropean(ATM, 10Y)');
%5 Specify the factor correlations.
hyCorrelations = HybridModelCorrelationMatrix(hySpec);
hyCorrelations.add('IR-USD', 'IR-EUR', 0.5);
hyCorrelations.add('IR-USD', 'FX-USDEUR', 0.25);
hyCorrelations.add('IR-EUR', 'FX-USDEUR', 0.25);

% Specify the model parameters.
hybridModelParameters = java.util.HashMap;
hybridModelParameters.put('Quote Type', 'MID');
hybridModelParameters.put('Payout Currency', 'USD');
```

```
hybridModelParameters.put('Specification', hySpec);
hybridModelParameters.put('Correlations', hyCorrelations);
```

Specify exposure calculation parameters.

```
observationDates = CustomObservationSchedule;
observationDates.add(DateExtensions.date(2011, 12, 1));
for y = 2012:2013
  for m = 1:12
    observationDates.add(DateExtensions.date(y, m, 1));
  end
end

exposureParameters = java.util.HashMap;
exposureParameters.put('Model ID', 'HYBRID');
exposureParameters.put('Observation Dates', observationDates);
```

Define the first trade instance.

```
tradeParameters1 = java.util.HashMap;
tradeParameters1.put('Trade ID', 'RVFL1001');
tradeParameters1.put('Quote Type', 'MID');
tradeParameters1.put('Currency', 'USD');
tradeParameters1.put('Notional', 1000000.0);
tradeParameters1.put('Effective Date', DateExtensions.date('1-Dec-2011'));
tradeParameters1.put('Termination Date', DateExtensions.date('1-Dec-2021'));
tradeParameters1.put('IR Index', 'LIBOR');
tradeParameters1.put('IR Index Tenor', '3M');
tradeParameters1.put('Structured Freq', '3M');
tradeParameters1.put('Structured Side', 'Receive');
tradeParameters1.put('Structured Coupon Floor', 0.0);
tradeParameters1.put('Structured Coupon UpBd', 0.08);
tradeParameters1.put('StructuredCoupon Multiplier', 1.4);
tradeParameters1.put('Structured Coupon Cap', 0.05);
tradeParameters1.put('Structured Basis', 'ACT/360');
tradeParameters1.put('Funding Freq', '3M');
tradeParameters1.put('Funding Side', 'Pay');
tradeParameters1.put('Funding Spread', 0.003);
tradeParameters1.put('Funding Basis', 'ACT/360');
tradeParameters1.put('Call Start Date', DateExtensions.date('1-Dec-2013'));
```

```
tradeParameters1.put('Call End Date', DateExtensions.date('1-Dec-2020'));
tradeParameters1.put('Option Side', 'Short');
tradeParameters1.put('Option Type', 'Right to Terminate');
tradeParameters1.put('Call Frequency', '3M');
tradeParameters1.put('Model', 'HYBRID');
tradeParameters1.put('Method', 'BackwardMC');
tradeInstance1 = RepositoryExtensions.createTradeInstance(n.Repository, 'TRADE.IR.CALLABLEREVERSEFLOATER', t
```

Define the second trade instance.

```
tradeParameters2 = java.util.HashMap;
tradeParameters2.put('Trade ID', 'CASHDEP1001');
tradeParameters2.put('Quote Type', 'MID');
tradeParameters2.put('Currency', 'USD');
tradeParameters2.put('Coupon Rate', 0.05);
tradeParameters2.put('Yield', 0.044);
tradeParameters2.put('Notional', 100.0);
tradeParameters2.put('Effective Date', DateExtensions.date('1-Apr-2012'));
tradeParameters2.put('Maturity', DateExtensions.date('1-Apr-2013'));
tradeParameters2.put('IR Index', 'LIBOR');
tradeParameters2.put('IR Index Tenor', '3M');
tradeParameters2.put('Model', 'HYBRID');
tradeParameters2.put('Method', 'BACKWARDMC');
tradeInstance2 = RepositoryExtensions.createTradeInstance(n.Repository, 'IR.CASHDEPOSIT', tradeParameters2);
```

Create the third trade instance.

```
tradeParameters3 = java.util.HashMap;
tradeParameters3.put('Trade ID', 'FXFWD1001');
tradeParameters3.put('Quote Type', 'MID');
tradeParameters3.put('Base Currency', 'USD');
tradeParameters3.put('Term Currency', 'EUR');
tradeParameters3.put('Delivery Date', DateExtensions.date('1-Jun-2012'));
tradeParameters3.put('Contract FX Forward Rate', 80.5);
tradeParameters3.put('Base Notional', 10000000.0);
tradeParameters3.put('Base IR Index', 'LIBOR');
tradeParameters3.put('Term IR Index', 'EURIBOR');
tradeParameters3.put('Base IR Index Tenor', '3m');
tradeParameters3.put('Term IR Index Tenor', '6m');
tradeParameters3.put('Calendar', 'NewYork Target');
```

10-7

```
tradeParameters3.put('Spot Lag', '2bd');
tradeParameters3.put('Model', 'HYBRID');
tradeParameters3.put('Method', 'BACKWARDMC');
tradeInstance3 = RepositoryExtensions.createTradeInstance(n.Repository, 'FX.FXFORWARD', tradeParameters3);
```

Set `tradeInstances` for all three trade instances.

```
tradeInstances = java.util.ArrayList;
tradeInstances.add(tradeInstance1);
tradeInstances.add(tradeInstance2);
tradeInstances.add(tradeInstance3);
n.Parameters.setInstances(tradeInstances);
```

Add a custom lookup so these trade instances reference the hybrid model.

```
n.Parameters.getLookups.add(0,ExactLookupRule('HYBRID','MODEL.HYBRID',hybridModelParameters.entrySet));
```

Add another custom lookup so that exposure report has parameters defined.

```
n.Parameters.getLookups.add(1,ExactLookupRule('RISK.REPORT.EXPOSURE','REPORT.EXPOSURE',exposureParameters.en
```

Perform the calculation.

```
results = n.Context.calculate(n.Parameters, Request.getExposure);
```

Parse the results for MATLAB and display.

```
r = n.parseResults(results)

disp([r.Trade(2) r.Market(2)])
disp([r.Results{2}.Name r.Results{2}.Category r.Results{2}.Currency r.Results{2}.Data])
disp([r.Results{2}.Name{1}])
disp([r.Results{2}.Data{1}])

r =

     Trade: {3x1 cell}
    Market: {3x1 cell}
   Results: {3x1 cell}

 'CASHDEP1001'    'EOD'
```

```
    'Exposure'                      ''     ''    {21x501 cell}
    'Exposure.Discount Factors'     ''     ''    {21x501 cell}
    'Messages'                      ''     ''    {12x1  cell}


Exposure
  Columns 1 through 3


    'DATE'                          'VALUE 1'           'VALUE 2'
    'Tue May 01 13:00:00 EDT 2012'  [104.198166609924]  [103.386222783828]
    'Fri Jun 01 13:00:00 EDT 2012'  [ 104.09953599675]  [102.117465067435]
    'Sun Jul 01 13:00:00 EDT 2012'  [105.524567506006]  [100.055731577867]
    'Wed Aug 01 13:00:00 EDT 2012'  [105.787455961524]  [100.318762976796]
    'Sat Sep 01 13:00:00 EDT 2012'  [104.417483614373]  [100.764337265155]
    'Mon Oct 01 13:00:00 EDT 2012'  [104.692275556824]  [100.980213613911]
    'Thu Nov 01 13:00:00 EDT 2012'  [104.443818312902]  [101.478508725115]
    'Sat Dec 01 12:00:00 EST 2012'  [104.736646932343]  [101.679769557039]
    'Tue Jan 01 12:00:00 EST 2013'  [104.577562970494]  [102.423339265735]
    'Fri Feb 01 12:00:00 EST 2013'  [ 104.28994278039]  [103.117326879887]
    'Fri Mar 01 12:00:00 EST 2013'  [ 104.70469459715]  [104.232180198939]
    'Mon Apr 01 13:00:00 EDT 2013'  [ 105.07334321718]  [ 105.05089338769]
    'Wed May 01 13:00:00 EDT 2013'  [              0]   [              0]
    'Sat Jun 01 13:00:00 EDT 2013'  [              0]   [              0]
    'Mon Jul 01 13:00:00 EDT 2013'  [              0]   [              0]
    'Thu Aug 01 13:00:00 EDT 2013'  [              0]   [              0]
    'Sun Sep 01 13:00:00 EDT 2013'  [              0]   [              0]
    'Tue Oct 01 13:00:00 EDT 2013'  [              0]   [              0]
    'Fri Nov 01 13:00:00 EDT 2013'  [              0]   [              0]
    'Sun Dec 01 12:00:00 EST 2013'  [              0]   [              0]


.


.


.


  Columns 499 through 501


    'VALUE 498'         'VALUE 499'          'VALUE 500'
    [ 105.36273206453]  [104.335982034187]   [104.141595030057]
    [105.904822463264]  [104.238089023172]   [104.276676080686]
```

[103.893060436208]	[103.613968079212]	[106.188617261199]
[103.183889382889]	[105.499763150412]	[105.440275818983]
[103.310404527817]	[105.233622768447]	[105.267337892552]
[103.274239052394]	[104.716952177783]	[104.33099834332]
[103.583983117053]	[104.710250522521]	[105.501004542869]
[103.379982561438]	[105.146939039653]	[104.681616459661]
[103.821169954095]	[105.567274949306]	[104.835971977691]
[104.016530403399]	[105.254054161819]	[104.842156238753]
[104.481475787501]	[105.197179985119]	[104.962752610848]
[105.061984636083]	[105.077227736476]	[105.077766765965]
[0]	[0]	[0]
[0]	[0]	[0]
[0]	[0]	[0]
[0]	[0]	[0]
[0]	[0]	[0]
[0]	[0]	[0]
[0]	[0]	[0]
[0]	[0]	[0]

Plot the results for the second trade instance, CASHDEP1001, with the corresponding Exposure Discount Factors.

```
figure('Tag','NumerixAdvancedRiskExample');

for ii=1:3
    % Get dates
    dates = cell2mat(r.Results{ii}.Data{expIndex}(2:end,1));
    dates = dates(:,4:end);
    dates = floor(datenum(dates));
    % Get exposures
    mtm = cell2mat(r.Results{ii}.Data{expIndex}(2:end,2:end))';
    exposures = max(0,mtm); % Exposure at contract level, no netting
    EE = mean(exposures); % Expected Exposure
    PFE = prctile(exposures,95); % Potential Future Exposure
    subplot(3,1,ii)
    plot(dates,EE,dates,PFE)
    title(r.Trade{ii})
    datetick
end
```
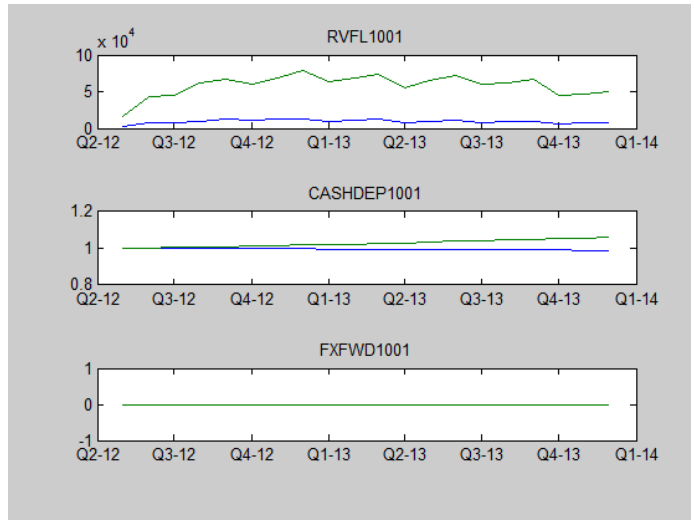
See Also   numerix **|** parseResults **|**

Related Examples
- "Working with Simple Numerix Trades" on page 10-2
- "Use Numerix to Price Cash Deposits" on page 10-12
- "Use Numerix for Interest-Rate Risk Assessment" on page 10-15

External Web Sites
- http://www.numerix.com/cail

Use Numerix to Price Cash Deposits

This example shows how to use the Numerix CAIL API to price a cash deposit from MATLAB. The trade parameters are read from the Cashdeposit1.csv in the Numerix Data Trades directory.

Initialize Numerix environment.

```
import com.numerix.integration.*;
import com.numerix.integration.implementation.*;

n = numerix('i:\NumeriX_java_10_3_0\data')

n =

              Path: 'i:\NumeriX_java_10_3_0\data'
    RepositoryPath: 'i:\NumeriX_java_10_3_0\data\Repository'
        Repository: [1x1 com.numerix.integration.implementation.FileSystemRepository]
           Context: [1x1 com.numerix.integration.implementation.LocalCalculationContext]
       LookupsPath: 'i:\NumeriX_java_10_3_0\data\Data\LookupRules'
       MarketsPath: 'i:\NumeriX_java_10_3_0\data\Data\Markets'
      FixingsPath: 'i:\NumeriX_java_10_3_0\data\Data\Fixings'
       TradesPath: 'i:\NumeriX_java_10_3_0\data\Data\Trades'
       Parameters: [1x1 com.numerix.integration.implementation.CalculationParameters]
```

Create a market.

```
market = Market('EOD_16-APR-2012', DateExtensions.date('16-APR-2012'), []);
```

Read the Cashdeposit1.csv file from the Numerix Trades directory.

```
[~,~,tradeInfo] = xlsread([n.TradesPath '\Cashdeposit1.csv'])

tradeInfo =

    'Template'          'String'      'TRADE.IR.CASHDEPOSIT'
    'Trade ID'          'ID'          'CASHDEP1001'
    'Quote Type'        'String'      'MID'
    'Effective Date'    'Date'        '4/1/2012'
    'Maturity'          'Date'        '4/1/2013'
    'Notional'          'Double'      [             100]
```

```
'Currency'          'Currency'     'USD'
'Coupon Rate'       'Double'       [                0.0500]
'Yield'             'Double'       [                0.0440]
'IR Index'          'String'       'Libor'
'IR Index Tenor'    'Tenor'        '3m'
```

Define a trade instance from the imported `CASHDEP1001` instrument.

```
tradeDescriptor = tradeInfo{1,3};
tradeParameters = java.util.HashMap;
numTradeInfoFields = size(tradeInfo,1);
for i = 2:numTradeInfoFields
  switch tradeInfo{i,2}
    case {'DATE','Date'}
      tradeParameters.put(tradeInfo{i,1},DateExtensions.date(datestr(tradeInfo{i,3},'dd-mmm-yyyy')));
    otherwise
      tradeParameters.put(tradeInfo{i,1},tradeInfo{i,3});
  end
end
```

Create the trade instance.

```
trade = RepositoryExtensions.createTradeInstance(n.Repository, tradeDescriptor, tradeParameters);
```

Price the trade.

```
results = CalculationContextExtensions.calculate(n.Context, trade, market, Request.getAll);
```

Parse the results for MATLAB and display.

```
r = n.parseResults(results)
disp([r.Name r.Category r.Currency r.Data])
```

```
r =

   Category: {9x1 cell}
   Currency: {9x1 cell}
       Name: {9x1 cell}
       Data: {9x1 cell}


   'Modified Duration'    'Price'    ''      [   0.9349]
```

```
'Accrued Interest'      'Price'    'USD'    [   0.2083]
'Reporting Currency'    'Price'    ''       'USD'
'PV'                    'Price'    'USD'    [ 100.7607]
'Instrument'            'Price'    ''       [1x85 char]
'Clean Price'           'Price'    'USD'    [ 100.5524]
'Convexity'             'Price'    ''       [   1.7481]
'YTM'                   'Price'    ''               []
'Messages'              ''         ''               []
```

See Also   numerix **|** parseResults **|**

Related Examples
- "Working with Simple Numerix Trades" on page 10-2
- "Working with Advanced Numerix Trades" on page 10-5
- "Use Numerix for Interest-Rate Risk Assessment" on page 10-15

External Web Sites
- http://www.numerix.com/cail

Use Numerix for Interest-Rate Risk Assessment

This example shows how to use the Numerix CAIL API for interest-rate curve stripping for risk assessment.

Initialize Numerix environment.

```
import com.numerix.integration.*;
import com.numerix.integration.implementation.*;

n = numerix('i:\NumeriX_java_10_3_0\data')

n =

             Path: 'i:\NumeriX_java_10_3_0\data'
   RepositoryPath: 'i:\NumeriX_java_10_3_0\data\Repository'
       Repository: [1x1 com.numerix.integration.implementation.FileSystemRepository]
          Context: [1x1 com.numerix.integration.implementation.LocalCalculationContext]
      LookupsPath: 'i:\NumeriX_java_10_3_0\data\Data\LookupRules'
      MarketsPath: 'i:\NumeriX_java_10_3_0\data\Data\Markets'
     FixingsPath: 'i:\NumeriX_java_10_3_0\data\Data\Fixings'
      TradesPath: 'i:\NumeriX_java_10_3_0\data\Data\Trades'
      Parameters: [1x1 com.numerix.integration.implementation.CalculationParameters]
```

Specify the current market associated with the Numerix CAIL environment.

```
markets = get(n.Parameters,'Markets');
currentMarket = markets.get(0);
outInstance = RefObject(currentMarket);
```

Define the interest-rate curve key `IR.USD-LIBOR-3M.MID`.

```
n.Context.tryResolveId('IR.USD-LIBOR-3M.MID',outInstance);
currentInstance = outInstance.argvalue;
```

Set the instance and market.

```
n.Parameters.setMarkets(java.util.Arrays.asList(currentMarket));
n.Parameters.setInstances(java.util.Arrays.asList(currentInstance));
```

Calculate the interest-rate curve stripping.

```
results = n.Context.calculate(n.Parameters,Request.getAll);
```

The calculation returns the results from stripping the interest-rate curve for
IR.USD-LIBOR-3M.MID. Parse the results for MATLAB and display.

```
% IR.USD-LIBOR-3M.MID.
r = n.parseResults(results)

disp([r.Instance r.Market])
disp([r.Results{1}.Name r.Results{1}.Category r.Results{1}.Currency r.Results{1}.Data])
disp([r.Results{1}.Name{1}])
disp([r.Results{1}.Data{1}])

r =

    Instance: {'IR.USD-LIBOR-3M.MID'}
      Market: {'EOD'}
     Results: {[1x1 struct]}

 'IR.USD-LIBOR-3M.MID'     'EOD'

'Curve Info'     ''     ''    {30x3 cell}
'Messages'       ''     ''    { 7x1 cell}

Curve Info

    'KEY'                              'DATE'                       'DISCOUNTFACTOR'
      'CASH RATE 16-APR-2012 17-APR-2012'    'Tue Apr 17 13:00:00 EDT 2012'    [      1.0000]
      'CASH RATE 16-APR-2012 18-APR-2012'    'Wed Apr 18 13:00:00 EDT 2012'    [      1.0000]
      'CASH RATE 16-APR-2012 23-APR-2012'    'Mon Apr 23 13:00:00 EDT 2012'    [      1.0000]
      'CASH RATE 16-APR-2012 30-APR-2012'    'Mon Apr 30 13:00:00 EDT 2012'    [      0.9999]
      'CASH RATE 16-APR-2012 16-MAY-2012'    'Wed May 16 13:00:00 EDT 2012'    [      0.9998]
      'CASH RATE 16-APR-2012 18-JUN-2012'    'Mon Jun 18 13:00:00 EDT 2012'    [      0.9994]
      'CASH RATE 16-APR-2012 16-JUL-2012'    'Mon Jul 16 13:00:00 EDT 2012'    [      0.9988]
      'CASH RATE 18-MAY-2012 12-AUG-2012'    'Sun Aug 12 13:00:00 EDT 2012'    [      0.9987]
      'CASH RATE 20-JUN-2012 20-SEP-2012'    'Thu Sep 20 13:00:00 EDT 2012'    [      0.9981]
      'CASH RATE 18-JUL-2012 18-OCT-2012'    'Thu Oct 18 13:00:00 EDT 2012'    [      0.9975]
      'CASH RATE 15-AUG-2012 15-NOV-2012'    'Thu Nov 15 12:00:00 EST 2012'    [      0.9973]
      'CASH RATE 19-SEP-2012 19-DEC-2012'    'Wed Dec 19 12:00:00 EST 2012'    [      0.9968]
      'CASH RATE 17-OCT-2012 17-JAN-2013'    'Thu Jan 17 12:00:00 EST 2013'    [      0.9962]
```

```
'CASH RATE 19-DEC-2012 19-MAR-2013'    'Tue Mar 19 13:00:00 EDT 2013'    [    0.9955]
'SWAP RATE 18-APR-2012 19-APR-2016'    'Tue Apr 19 13:00:00 EDT 2016'    [    0.9645]
'SWAP RATE 18-APR-2012 18-APR-2017'    'Tue Apr 18 13:00:00 EDT 2017'    [    0.9445]
'SWAP RATE 18-APR-2012 18-APR-2018'    'Wed Apr 18 13:00:00 EDT 2018'    [    0.9199]
'SWAP RATE 18-APR-2012 18-APR-2019'    'Thu Apr 18 13:00:00 EDT 2019'    [    0.8925]
'SWAP RATE 18-APR-2012 21-APR-2020'    'Tue Apr 21 13:00:00 EDT 2020'    [    0.8639]
'SWAP RATE 18-APR-2012 19-APR-2021'    'Mon Apr 19 13:00:00 EDT 2021'    [    0.8356]
'SWAP RATE 18-APR-2012 19-APR-2022'    'Tue Apr 19 13:00:00 EDT 2022'    [    0.8069]
'SWAP RATE 18-APR-2012 18-APR-2023'    'Tue Apr 18 13:00:00 EDT 2023'    [    0.7784]
'SWAP RATE 18-APR-2012 18-APR-2024'    'Thu Apr 18 13:00:00 EDT 2024'    [    0.7506]
'SWAP RATE 18-APR-2012 19-APR-2027'    'Mon Apr 19 13:00:00 EDT 2027'    [    0.6733]
'SWAP RATE 18-APR-2012 20-APR-2032'    'Tue Apr 20 13:00:00 EDT 2032'    [    0.5682]
'SWAP RATE 18-APR-2012 20-APR-2037'    'Mon Apr 20 13:00:00 EDT 2037'    [    0.4828]
'SWAP RATE 18-APR-2012 21-APR-2042'    'Mon Apr 21 13:00:00 EDT 2042'    [    0.4112]
'SWAP RATE 18-APR-2012 18-APR-2052'    'Thu Apr 18 13:00:00 EDT 2052'    [    0.3087]
'SWAP RATE 18-APR-2012 18-APR-2062'    'Tue Apr 18 13:00:00 EDT 2062'    [    0.2414]
```

See Also   numerix **|** parseResults **|**

Related Examples
- "Working with Simple Numerix Trades" on page 10-2
- "Working with Advanced Numerix Trades" on page 10-5
- "Use Numerix to Price Cash Deposits" on page 10-12

External Web Sites
- http://www.numerix.com/cail

A

Class Reference

@IRBootstrapOptions

Create specific options for bootstrapping an interest-rate curve object

In this section...
"Hierarchy" on page A-2
"Constructor" on page A-2
"Public Read-Only Properties" on page A-2
"Methods" on page A-3

Hierarchy

Superclasses: None

Subclasses: None

Constructor

IRBootstrapOptions

Public Read-Only Properties

Name	Description
ConvexityAdjustment	Controls the convexity adjustment to interest rate futures. This can be specified as a function handle that takes time to maturity as an input and returns a value which is ConvexityAdjustment. Alternatively, you can define ConvexityAdjustment as an N-by-1 vector of values, where N is the number of interest rate futures. In either case, the ConvexityAdjustment is subtracted from the futures rate.
	For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.

Methods

There are no methods.

@IRCurve

Base abstract class for interest-rate curve objects

In this section...
"Hierarchy" on page A-4
"Description" on page A-4
"Constructor" on page A-4
"Public Read-Only Properties" on page A-4
"Methods" on page A-6

Hierarchy

Superclasses: None

Subclasses: @IRDataCurve, @IRFunctionCurve

Description

IRCurve is an abstract class; you cannot create instances of it directly. You can create IRDataCurve and IRFunctionCurve objects that are derived from this class.

Constructor

@IRCurve is an abstract class. To construct an IRCurve object, use one of the subclass constructors, IRDataCurve or IRFunctionCurve.

Public Read-Only Properties

Name	Description
Type	Type of interest-rate curve: zero, forward, or discount.
Settle	Scalar or column vector of settlement dates.

Name	Description
Compounding	Scalar that sets the compounding frequency per year for the IRCurve object: • -1 = Continuous compounding • 1 = Annual compounding • 2 = Semiannual compounding (default) • 3 = Compounding three times per year • 4 = Quarterly compounding • 6 = Bimonthly compounding • 12 = Monthly compounding
Basis	Day-count basis of the interest-rate curve. A vector of integers. • 0 = actual/actual (default) • 1 = 30/360 (SIA) • 2 = actual/360 • 3 = actual/365 • 4 = 30/360 (BMA) • 5 = 30/360 (ISDA) • 6 = 30/360 (European) • 7 = actual/365 (Japanese) • 8 = actual/actual (ICMA) • 9 = actual/360 (ICMA) • 10 = actual/365 (ICMA) • 11 = 30/360E (ICMA) • 12 = actual/actual (ISDA) • 13 = BUS/252

Methods

Classes that inherit from the `IRCurve` abstract class must implement the following methods.

Method	Description
getForwardRates	Returns forward rates for input dates.
getZeroRates	Returns zero rates for input dates.
getDiscountFactors	Returns discount factors for input dates.
getParYields	Returns par yields for input dates.
toRateSpec	Converts to be a `RateSpec` object. This is identical to the `RateSpec` structure produced by the Financial Instruments Toolbox function `intenvset`.

@IRDataCurve

Represent interest-rate curve object based on vector of dates and data

In this section...
"Hierarchy" on page A-7
"Description" on page A-7
"Constructor" on page A-7
"Public Read-Only Properties" on page A-8
"Methods" on page A-9

Hierarchy

Superclasses: `@IRCurve`

Subclasses: None

Description

`IRDataCurve` is a representation of an interest-rate curve object with dates and data. You can construct this object directly by specifying dates and corresponding interest rates or discount factors; alternatively, you can bootstrap the object from market data. After an interest-rate curve object is constructed, you can:

- Calculate forward and zero rates and determine par yields.
- Extract the discount factors.
- Convert to a `RateSpec` structure that is identical to the `RateSpec` structure produced by the Financial Instruments Toolbox function `intenvset`.

Constructor

`IRDataCurve`

Public Read-Only Properties

Name	Description
Type	Type of interest-rate curve: zero, forward, or discount.
Settle	Scalar or column vector of settlement dates.
Compounding	Scalar that sets the compounding frequency per year for the IRCurve object: • -1 = Continuous compounding • 1 = Annual compounding • 2 = Semiannual compounding (default) • 3 = Compounding three times per year • 4 = Quarterly compounding • 6 = Bimonthly compounding • 12 = Monthly compounding
Basis	Day-count basis of the financial curve. A vector of integers. • 0 = actual/actual (default) • 1 = 30/360 (SIA) • 2 = actual/360 • 3 = actual/365 • 4 = 30/360 (BMA) • 5 = 30/360 (ISDA) • 6 = 30/360 (European) • 7 = actual/365 (Japanese) • 8 = actual/actual (ICMA) • 9 = actual/360 (ICMA) • 10 = actual/365 (ICMA) • 11 = 30/360E (ICMA)

Name	Description
	• 12 = actual/actual (ISDA) • 13 = BUS/252
Dates	Dates corresponding to rate data.
Data	Interest-rate data or discount factors for the curve object.
InterpMethod	Values are: • `'linear'` — Linear interpolation (default). • `'constant'` — Piecewise constant interpolation. • `'pchip'` — Piecewise cubic Hermite interpolation. • `'spline'` — Cubic spline interpolation.

Methods

The following table contains links to methods with supporting reference pages, including examples.

Method	Description
getForwardRates	Returns forward rates for input dates.
getZeroRates	Returns zero rates for input dates.
getDiscountFactors	Returns discount factors for input dates.
getParYields	Returns par yields for input dates.
toRateSpec	Converts to be a `RateSpec` object. This structure is identical to the `RateSpec`
bootstrap	Bootstraps an interest rate curve from market data.

@IRFitOptions

Object to specify fitting options for an `IRFunctionCurve` interest-rate curve object

In this section...
"Hierarchy" on page A-10
"Description" on page A-10
"Constructor" on page A-10
"Public Read-Only Properties" on page A-11
"Methods" on page A-11

Hierarchy

Superclasses: None

Subclasses: None

Description

The `IRFitOptions` object allows you to specify options relating to the fitting process for an `IRFunctionCurve` object. Input arguments are specified in parameter/value pairs. The `IRFitOptions` structure provides the capability to choose which quantity to be minimized and other optimization parameters.

Constructor

`IRFitOptions`

Public Read-Only Properties

Name	Description
FitType	Price, Yield, or DurationWeightedPrice determines which is minimized in the curve fitting process. DurationWeightedPrice is the default.
InitialGuess	Initial guess for the parameters of the curve function.
UpperBound	Upper bound for the parameters of the curve function.
LowerBound	Lower bound for the parameters of the curve function.
OptOptions	Optimization structure based on the output from the Optimization Toolbox function optimset or optimoptions. This optimization structure is evaluated by lsqnonlin.
A	Inequality constraint for parameters, ignored if OptimFunction is set to lsqnonlin.
b	Inequality constraint for parameters, ignored if OptimFunction is set to lsqnonlin.
OptimFunction	Optimization function used to fit function, either lsqnonlin or fmincon.

Methods

There are no methods.

@IRFunctionCurve

Represent an interest-rate curve object using a function

In this section...

Hierarchy

Superclasses: `@IRCurve`

Subclasses: None

Description

`IRFunctionCurve` is a representation of an interest-rate curve object. You can construct this object directly by specifying a function handle or a function can be fit to market data using methods of the object. After an interest-rate curve object is constructed; you can:

- Calculate forward and zero rates and determine par yields.

- Extract the discount factors.

- Convert to a `RateSpec` structure; this is identical to the `RateSpec` structure produced by the Financial Instruments Toolbox function `intenvset`.

Constructor

`IRFunctionCurve`

Public Read-Only Properties

Name	Description
Type	Type of interest-rate curve: zero, forward, or discount.
Settle	Scalar or column vector of settlement dates.
Compounding	Scalar that sets the compounding frequency per year for the IRCurve object: • -1 = Continuous compounding • 1 = Annual compounding • 2 = Semiannual compounding (default) • 3 = Compounding three times per year • 4 = Quarterly compounding • 6 = Bimonthly compounding • 12 = Monthly compounding
Basis	Day-count basis of the interest-rate curve. A vector of integers. • 0 = actual/actual (default) • 1 = 30/360 (SIA) • 2 = actual/360 • 3 = actual/365 • 4 = 30/360 (BMA) • 5 = 30/360 (ISDA) • 6 = 30/360 (European) • 7 = actual/365 (Japanese) • 8 = actual/actual (ICMA) • 9 = actual/360 (ICMA) • 10 = actual/365 (ICMA)

Name	Description
	• 11 = 30/360E (ICMA) • 12 = actual/actual (ISDA) • 13 = BUS/252
FunctionHandle	Function handle that defines the interest-rate curve. For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.

Methods

The following table contains links to methods with supporting reference pages, including examples.

Method	Description
getForwardRates	Returns forward rates for input dates.
getZeroRates	Returns zero rates for input dates.
getDiscountFactors	Returns discount factors for input dates.
getParYields	Returns par yields for input dates.
toRateSpec	Converts to be a RateSpec object. This is identical to the RateSpec structure
fitSvensson	Fits a Svensson function to market data.
fitNelsonSiegel	Fits a Nelson-Siegel function to market data.
fitSmoothingSpline	Fits a smoothing spline function to market data.
fitFunction	Fits a custom function to market data.

Functions — Alphabetical List

asianbycrr

Purpose	Price Asian option from Cox-Ross-Rubinstein binomial tree
Syntax	`Price = asianbycrr(CRRTree, OptSpec, Strike, Settle,` `ExerciseDates, AmericanOpt, AvgType, AvgPrice, AvgDate)`

Arguments

CRRTree	Stock tree structure created by `crrtree`.
OptSpec	NINST-by-1 list of string values `'Call'` or `'Put'`.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.
Settle	NINST-by-1 vector of Settle dates. The settle date for every Asian option is set to the valuation date of the stock tree. The Asian argument Settle is ignored.
ExerciseDates	For a European option (AmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (AmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.

AmericanOpt	(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.
AvgType	(Optional) String = 'arithmetic' for arithmetic average (default) or 'geometric' for geometric average.
AvgPrice	(Optional) Scalar representing the average price of the underlying asset at Settle. This argument is used when AvgDate < Settle. Default is the current stock price.
AvgDate	(Optional) Scalar representing the date on which the averaging period begins. Default = Settle.

Description   Price = asianbycrr(CRRTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, AvgType, AvgPrice, AvgDate) calculates the value of fixed- and floating-strike Asian options. To compute the value of a floating-strike Asian option, specify Strike as NaN. Fixed-strike Asian options are also known as average price options. Floating-strike Asian options are also known as average strike options.

Price is a NINST-by-1 vector of expected prices at time 0.

Asian options are priced using Hull-White (1993). Consequently, for these options only the root node contains a unique price.

Examples   Price a floating-strike Asian option using a CRR binomial tree.

Load the file deriv.mat, which provides CRRTree. The CRRTree structure contains the stock specification and time information needed to price the option.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'put';
Strike = NaN;
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2004';
```

Use asianbycrr to compute the price of the option.

```
Price = asianbycrr(CRRTree, OptSpec, Strike, Settle, ...
ExerciseDates)

Price =

 1.2177
```

References   Hull, J., and A. White, "Efficient Procedures for Valuing European and American Path-Dependent Options," *Journal of Derivatives*, Volume 1, pp. 21-31.

See Also   crrtree | instasian

Purpose   Price Asian option from Equal Probabilities binomial tree

Syntax   Price = asianbyeqp(EQPTree, OptSpec, Strike, Settle,
                ExerciseDates, AmericanOpt, AvgType, AvgPrice, AvgDate)

Arguments

EQPTree	Stock tree structure created by eqptree.
OptSpec	NINST-by-1 list of string values 'Call' or 'Put'.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.
Settle	NINST-by-1 vector of Settle dates. The settle date for every Asian option is set to the valuation date of the stock tree. The Asian argument Settle is ignored.
ExerciseDates	For a European option (AmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (AmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.

asianbyeqp

AmericanOpt	(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.
AvgType	(Optional) String = 'arithmetic' for arithmetic average (default) or 'geometric' for geometric average.
AvgPrice	(Optional) Scalar representing the average price of the underlying asset at Settle. This argument is used when AvgDate < Settle. Default is the current stock price.
AvgDate	(Optional) Scalar representing the date on which the averaging period begins.

Description   Price = asianbyeqp(EQPTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, AvgType, AvgPrice, AvgDate) calculates the value of fixed- and floating-strike Asian options. To compute the value of a floating-strike Asian option, specify Strike as NaN. Fixed-strike Asian options are also known as average price options. Floating-strike Asian options are also known as average strike options.

Price is a NINST-by-1 vector of expected prices at time 0.

Examples   Price a floating-strike Asian option using an EQP equity tree.

Load the file deriv.mat, which provides EQPTree. The EQPTree structure contains the stock specification and time information needed to price the option.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'put';
Strike = NaN;
```

```
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2004';
```

Use asianbyeqp to compute the price of the option.

```
Price = asianbyeqp(EQPTree, OptSpec, Strike, Settle, ...
ExerciseDates)

Price =

 1.2724
```

References   Hull, J., and A. White, "Efficient Procedures for Valuing European and American Path-Dependent Options," *Journal of Derivatives*, Volume 1, pp. 21-31.

See Also   eqptree | instasian

asianbyitt

Purpose	Price Asian options using implied trinomial tree (ITT)
Syntax	`Price = asianbyitt(ITTTree, OptSpec, Strike, Settle,` `ExerciseDates, AmericanOpt, AvgType, AvgPrice, AvgDate)`

Arguments

ITTTree	Stock tree structure created by `itttree`.
OptSpec	NINST-by-1 list of string values `'call'` or `'put'`.
Strike	NINST-by-1 vector of strike price values. Each row represents the schedule for one option.
Settle	NINST-by-1 vector of `Settle` dates. The settle date for every Asian option is set to the valuation date of the stock tree. The Asian argument `Settle` is ignored.
ExerciseDates	For a European option (`AmericanOpt = 0`):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date which is the option expiry date.
	For an American option (`AmericanOpt = 1`):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-`NaN` date is listed, or if `ExerciseDates` is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.

AmericanOpt	(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.
AvgType	(Optional) String = 'arithmetic' for arithmetic average (default) or 'geometric' for geometric average.
AvgPrice	(Optional) Scalar representing the average price of the underlying asset at Settle. This argument is used when AvgDate < Settle. Default is the current stock price.
AvgDate	(Optional) Scalar representing the date on which the averaging period begins.

Description   Price = asianbyitt(ITTTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, AvgType, AvgPrice, AvgDate) calculates the value of fixed- and floating-strike Asian options. To compute the value of a floating-strike Asian option, specify Strike as NaN. Fixed-strike Asian options are also known as average price options. Floating-strike Asian options are also known as average strike options.

Price is a NINST-by-1 vector of expected prices at time 0.

Note The Settle date for every Asian option is set to the ValuationDate of the stock tree. The Asian argument, Settle, is ignored.

Examples   Price a floating-strike Asian option using an ITT equity tree.

Load the file deriv.mat which provides the ITTTree. The ITTTree structure contains the stock specification and time information needed to price the option.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'put';
Strike = NaN;
Settle = '01-Jan-2006';
ExerciseDates = '01-Jan-2007';
```

Use `asianbyitt` to compute the price of the option.

```
Price = asianbyitt(ITTTree, OptSpec, Strike, Settle, ExerciseDates)

Price =

 1.0778
```

References   Hull, J., and A. White, "Efficient Procedures for Valuing European and American Path-Dependent Options," *Journal of Derivatives*, Volume 1, 1993, pp. 21-31.

See Also   instasian | itttree

Purpose   Price European or American Asian option using Longstaff-Schwartz model

Syntax   Price = asianbyls(RateSpec,StockSpec,OptSpec,Strike,Settle,
ExerciseDates)
Price = asianbyls(___ ,Name,Value)

[Price,Paths,Times,Z] = asianbyls(RateSpec,StockSpec,OptSpec,Strike,Se
ExerciseDates)
[Price,Paths,Times,Z] = asianbyls(___ ,Name,Value)

Description   Price = asianbyls(RateSpec,StockSpec,OptSpec,Strike,Settle,
ExerciseDates) returns fixed- and floating-strike Asian option prices
using the Longstaff-Schwartz model. asianbyls computes prices of
European and American Asian options. For American options, the
Longstaff-Schwartz least squares method is used to calculate the early
exercise premium. To compute the value of a floating-strike Asian
option, Strike should be specified as NaN. Fixed-strike Asian options
are also known as average price options and floating-strike Asian
options are also known as average strike options.

Price = asianbyls(___ ,Name,Value) returns fixed- and
floating-strike Asian option prices using optional name-value pair
arguments and the Longstaff-Schwartz model. asianbyls computes
prices of European and American Asian options. For American options,
the Longstaff-Schwartz least squares method is used to calculate the
early exercise premium. To compute the value of a floating-strike Asian
option, Strike should be specified as NaN. Fixed-strike Asian options
are also known as average price options and floating-strike Asian
options are also known as average strike options.

[Price,Paths,Times,Z] =
asianbyls(RateSpec,StockSpec,OptSpec,Strike,Settle,
ExerciseDates) returns fixed- and floating-strike Asian option Price,
Paths, Times, and Z values using the Longstaff-Schwartz model.
asianbyls computes prices of European and American Asian options.

For American options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium. To compute the value of a floating-strike Asian option, `Strike` should be specified as `NaN`. Fixed-strike Asian options are also known as average price options and floating-strike Asian options are also known as average strike options.

`[Price,Paths,Times,Z] = asianbyls( ___ ,Name,Value)` returns fixed and floating strike Asian option `Price`, `Paths`, `Times`, and `Z` values using optional name-value pair arguments and the Longstaff-Schwartz model. `asianbyls` computes prices of European and American Asian options. For American options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium. To compute the value of a floating-strike Asian option, `Strike` should be specified as `NaN`. Fixed-strike Asian options are also known as average price options and floating-strike Asian options are also known as average strike options.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the `RateSpec` obtained from `intenvset`. For information on the interest-rate specification, see `intenvset`.

Data Types
struct

StockSpec - Stock specification for underlying asset
structure

Stock specification for underlying asset, specified using `StockSpec` obtained from `stockspec`. For information on the stock specification, see `stockspec`.

`stockspec` can handle other types of underlying assets. For example, stocks, stock indices, and commodities. If dividends are not specified in `StockSpec`, dividends are assumed to be `0`.

Data Types
struct

OptSpec - Definition of option
string with a value of `'call'` or `'put'`

Definition of option, specified as `'call'` or `'put'` using a string.

Data Types
char

Strike - Option strike price value
nonnegative scalar integer

Option strike price value, specified with a nonnegative scalar integer. To compute the value of a floating-strike Asian option, Strike should be specified as NaN. Floating-strike Asian options are also known as average strike options.

Data Types
double

Settle - Settlement or trade date
date string | nonnegative scalar integer

Settlement or trade date for the Asian option, specified as a nonnegative scalar integer or date string.

Data Types
char

ExerciseDates - Option exercise dates
date string | nonnegative scalar integer

Option exercise dates, specified as a nonnegative scalar integer or date string:

- For a European option, use a 1-by-1 vector of dates. For a European option, there is only one ExerciseDates on the option expiry date.

- For an American option, use a 1-by-2 vector of exercise date boundaries. The option can be exercised on any date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is a 1-by-1 vector of serial date numbers or cell array of strings, the option can be exercised between Settle and the single listed ExerciseDates.

Data Types
char

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price = asianbyls(RateSpec,StockSpec,OptSpec,Strike,Settle,ExerciseDates,'NumTria

'AmericanOpt' - Option type
0 European (default) | scalar with value [0,1]

Option type, specified as NINST-by-1 positive integer scalar flags with values:

- 0 — European

- 1 — American

Data Types
single | double

'AvgType' - Average types
arithmetic (default) | string with values of arithmetic or geometric

Average types, specified as arithmetic for arithmetic average, or geometric for geometric average.

Data Types
double

'AvgPrice' - Average price of underlying asset at `Settle`
scalar

Average price of underlying asset at `Settle`, specified as a scalar.

Note Use this argument when `AvgDate` < `Settle`.

Data Types
`double`

'AvgDate' - Date averaging period begins
scalar

Date averaging period begins, specified as a scalar.

Data Types
`double`

'NumTrials' - Simulation trials
1000 (default) | scalar

Simulation trials, specified as a scalar number of independent sample paths.

Data Types
`double`

'NumPeriods' - Simulation periods per trial
100 (default) | scalar

Simulation periods per trial, specified as a scalar number. `NumPeriods` is considered only when pricing European Asian options. For American Asian options `NumPeriod` is equal to the number of exercise days during the life of the option.

Data Types
`double`

'Z' - Dependent random variates
scalar | nonnegative integer

Dependent random variates used to generate the Brownian motion vector (i.e., Wiener processes) that drive the simulation, specified as a NumPeriods-by-2-by-NumTrials 3-D time series array.

Data Types
single | double

'Antithetic' - Indicator for antithetic sampling
false (default) | logical flag with value of true or false

Logical flag to indicate antithetic sampling, specified with a value of true or false.

Data Types
logical

Output Arguments

Price - Expected price of Asian option
scalar

Expected price of the Asian option, returned as a 1-by-1 scalar.

Paths - Simulated paths of correlated state variables
vector

Simulated paths of correlated state variables, returned as a (NumPeriods + 1)-by-1-by-NumTrials 3-D time series array. Each row of Paths is the transpose of the state vector $X(t)$ at time t for a given trial.

Times - Observation times associated with simulated paths
vector

Observation times associated with the simulated paths, returned as a (NumPeriods + 1)-by-1 column vector of observation times associated with the simulated paths. Each element of Times is associated with the corresponding row of Paths.

Z - Dependent random variates

vector

Dependent random variates, returned, if Z is specified as an optional input argument, the same value is returned. Otherwise, Z contains the random variates generated internally.

Examples   **Compute the Price of an Asian Option Using the Longstaff-Schwartz Model**

Define the RateSpec.

```
Rates = 0.05;
StartDate = 'Jan-1-2013';
EndDate = 'Jan-1-2014';
RateSpec = intenvset('ValuationDate', StartDate, 'StartDates', StartDate, ...
'EndDates', EndDate,'Compounding', -1, 'Rates', Rates)


RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9512
             Rates: 0.0500
          EndTimes: 1
        StartTimes: 0
          EndDates: 735600
        StartDates: 735235
     ValuationDate: 735235
             Basis: 0
       EndMonthRule: 1
```

Define the StockSpec for the asset.

```
AssetPrice = 100;
Sigma = 0.2;
StockSpec = stockspec(Sigma, AssetPrice)

StockSpec =
```

```
                FinObj: 'StockSpec'
                 Sigma: 0.2000
            AssetPrice: 100
          DividendType: []
       DividendAmounts: 0
       ExDividendDates: []
```

Define the Asian 'call' option.

```
Settle = 'Jan-1-2013';
ExerciseDates = 'Jan-1-2014';
Strike = 110;
OptSpec = 'call';
```

Compute the price for the European arithmetic average price for the Asian option using the Longstaff-Schwartz model.

```
NumTrials = 10000;
NumPeriods = 100;
AvgType = 'arithmetic';
Antithetic= true;
Price= asianbyls(RateSpec, StockSpec, OptSpec, Strike, Settle, ExerciseDates, ...
'NumTrials', NumTrials, 'NumPeriods', NumPeriods,'Antithetic', Antithetic, 'AvgType', AvgType)

Price =

    1.9347
```

See Also   asiansensbyls | asianbycrr | intenvset | stockspec | asianbykv | asianbylevy

Concepts   • "Asian Option" on page 3-34

Purpose   Calculate European or American Asian option prices and sensitivities using Longstaff-Schwartz model

Syntax   PriceSens = asiansensbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,
ExerciseDates)
PriceSens = asiansensbyls(___ ,Name,Value)

[PriceSens,Path,Times,Z] = asiansensbyls(RateSpec,StockSpec,OptSpec,St
ExerciseDates)
[PriceSens,Path,Times,Z] = asiansensbyls(___ ,Name,Value)

Description   PriceSens =
asiansensbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,
ExerciseDates) returns Asian option prices and sensitivities for fixed-
and floating-strike Asian options using the Longstaff-Schwartz model.
asiansensbyls supports European and American Asian options. For
American options, the Longstaff-Schwartz least squares method is used
to calculate the early exercise premium. To compute the value of a
floating-strike Asian option, Strike should be specified as NaN.
Fixed-strike Asian options are also known as average price options and
floating-strike Asian options are also known as average strike options.

PriceSens = asiansensbyls(___ ,Name,Value) returns Asian option
prices and sensitivities for fixed- and floating-strike Asian options using
optional name-value pair arguments and the Longstaff-Schwartz model.
asiansensbyls supports European and American Asian options. For
American options, the Longstaff-Schwartz least squares method is
used to calculate the early exercise premium. To compute the value
of a floating-strike Asian option, Strike should be specified as NaN.
Fixed-strike Asian options are also known as average price options and
floating-strike Asian options are also known as average strike options.

[PriceSens,Path,Times,Z] =
asiansensbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,
ExerciseDates) returns Asian option prices and sensitivities
(PriceSens, Path, Times, and Z) for fixed- and floating-strike Asian

asiansensbyls

options using the Longstaff-Schwartz model. `asiansensbyls` supports European and American Asian options. For American options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium. To compute the value of a floating-strike Asian option, `Strike` should be specified as `NaN`. Fixed-strike Asian options are also known as average price options and floating-strike Asian options are also known as average strike options.

`[PriceSens,Path,Times,Z] = asiansensbyls( ___ ,Name,Value)` returns Asian option prices and sensitivities (`PriceSens`, `Path`, `Times`, and `Z`) for fixed- and floating-strike Asian options using optional name-value pair arguments and the Longstaff-Schwartz model. `asiansensbyls` supports European and American Asian options. For American options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium. To compute the value of a floating-strike Asian option, `Strike` should be specified as `NaN`. Fixed-strike Asian options are also known as average price options and floating-strike Asian options are also known as average strike options.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the `RateSpec` obtained from `intenvset`. For information on the interest-rate specification, see `intenvset`.

Data Types
struct

StockSpec - Stock specification for underlying asset
structure

Stock specification for underlying asset, specified using `StockSpec` obtained from `stockspec`. For information on the stock specification, see `stockspec`.

stockspec can handle other types of underlying assets. For example, stocks, stock indices, and commodities. If dividends are not specified in StockSpec, dividends are assumed to be 0.

Data Types
struct

OptSpec - Definition of option
string with value 'call' or 'put'

Definition of option, specified as 'call' or 'put' using a string.

Data Types
char

Strike - Option strike price value
nonnegative scalar integer

Option strike price value, specified with a nonnegative scalar integer. To compute the value of a floating-strike Asian option, Strike should be specified as NaN. Floating-strike Asian options are also known as average strike options.

Data Types
double

Settle - Settlement date or trade date
nonnegative scalar integer | date string

Settlement date or trade date for the Asian option, specified as a nonnegative scalar integer or date string.

Data Types
char

ExerciseDates - Option exercise dates
nonnegative integer | data string

Option exercise dates, specified as a nonnegative scalar integer or date string:

- For a European option, use a 1-by-1 vector of dates. For a European option, there is only one ExerciseDates on the option expiry date.

- For an American option, use a 1-by-2 vector of exercise date boundaries. The option can be exercised on any date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is a 1-by-1 vector of serial date numbers or cell array of strings, the option can be exercised between Settle and the single listed ExerciseDates.

Data Types
char

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: PriceSens = asiansensbyls(RateSpec,StockSpec,OptSpec,Strike,Settle,ExerciseDates,'Num NumPeriods,'Antithetic',Antithetic,'AvgType',AvgType,'OutSpec',{'All'})

'AmericanOpt' - Option type
0 European (default) | scalar with values [0,1]

Option type, specified as NINST-by-1 positive integer scalar flags with values:

- 0 — European

- 1 — American

Data Types
single | double

'AvgType' - Average types
arithmetic (default) | string with values of arithmetic or geometric

Average types, specified as `arithmetic` for arithmetic average, or `geometric` for geometric average.

Data Types
`double`

'AvgPrice' - Average price of underlying asset at `Settle`
scalar

Average price of underlying asset at `Settle`, specified as a scalar.

Note Use this argument when `AvgDate` < `Settle`.

Data Types
`double`

'AvgDate' - Date averaging period begins
scalar

Date averaging period begins, specified as a scalar.

Data Types
`double`

'NumTrials' - Simulation trials
1000 (default) | scalar

Simulation trials, specified as a scalar number of independent sample paths.

Data Types
`double`

'NumPeriods' - Simulation periods per trial
100 (default) | scalar

Simulation periods per trial, specified as a scalar number. `NumPeriods` is considered only when pricing European Asian options. For American

Asian options `NumPeriod` is equal to the number of exercise days during the life of the option.

Data Types
`double`

'Z' - Dependent random variates
scalar | nonnegative integer

Dependent random variates used to generate the Brownian motion vector (i.e., Wiener processes) that drive the simulation, specified as a `NumPeriods`-by-2-by-`NumTrials` 3-D time series array.

Data Types
`single | double`

'Antithetic' - Indicates antithetic sampling
false (default) | logical flag with value of `true` or `false`

Indicates antithetic sampling, specified with a value of `true` or `false`.

Data Types
`logical`

'OutSpec' - Define outputs
{`'Price'`} (default) | string with values `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'` | cell array of strings with values `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'`

Define outputs specifying NOUT- by-1 or 1-by-NOUT cell array of strings with possible values of `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'` and `'All'`.

`OutSpec = {'All'}` specifies that the output should be `Delta`, `Gamma`, `Vega`, `Lambda`, `Rho`, `Theta`, and `Price`, in that order. This is the same as specifying `OutSpec` to include each sensitivity:

Example: `OutSpec =`
`{'delta','gamma','vega','lambda','rho','theta','price'}`

Data Types
char | cell

Output Arguments

PriceSens - Expected price and sensitivities of Asian option
vector

Expected price and sensitivities of the Asian option, returned as an NINST-by-1 vector.

Path - Simulated paths of correlated state variables
vector

Simulated paths of correlated state variables, returned as a (NumPeriods + 1)-by-2-by-NumTrials 3-D time series array. Each row of Paths is the transpose of the state vector $X(t)$ at time t for a given trial.

Times - Observation times associated with simulated paths
vector

Observation times associated with simulated paths, returned as a (NumPeriods + 1)-by-1 column vector of observation times associated with the simulated paths. Each element of Times is associated with the corresponding row of Paths.

Z - Dependent random variates
vector

Dependent random variates, returned, if Z is specified as an optional input argument, the same value is returned. Otherwise, Z contains the random variates generated internally.

Examples

Compute the Price and Sensitivities of an Asian Option Using the Longstaff-Schwartz Model

Define the RateSpec.

```
Rates = 0.05;
StartDate = 'Jan-1-2013';
EndDate = 'Jan-1-2014';
```

```
RateSpec = intenvset('ValuationDate', StartDate, 'StartDates', StartDate, ...
'EndDates', EndDate,'Compounding', -1, 'Rates', Rates)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9512
             Rates: 0.0500
          EndTimes: 1
        StartTimes: 0
          EndDates: 735600
        StartDates: 735235
    ValuationDate: 735235
             Basis: 0
       EndMonthRule: 1
```

Define the StockSpec for the asset.

```
AssetPrice = 100;
Sigma = 0.2;
StockSpec = stockspec(Sigma, AssetPrice)

StockSpec =

              FinObj: 'StockSpec'
               Sigma: 0.2000
          AssetPrice: 100
         DividendType: []
     DividendAmounts: 0
     ExDividendDates: []
```

Define the Asian 'call' option.

```
Settle = 'Jan-1-2013';
ExerciseDates = 'Jan-1-2014';
Strike = 110;
OptSpec = 'call';
```

Compute the price for the European arithmetic average price and sensitivities for the Asian option using the Longstaff-Schwartz model.

```
NumTrials = 10000;
NumPeriods = 100;
AvgType = 'arithmetic';
Antithetic= true;
OutSpec = {'Price', 'Delta', 'Gamma'};
PriceSens = asiansensbyls(RateSpec, StockSpec, OptSpec, Strike, Settle, ExerciseDates, ...
'NumTrials', NumTrials, 'NumPeriods', NumPeriods,'Antithetic', Antithetic, 'AvgType', ...
AvgType,'OutSpec',OutSpec)

PriceSens =

    1.9536
```

See Also asianbyls | asianbycrr | stockspec | intenvset | asianbykv | asianbylevy

Concepts • "Asian Option" on page 3-34

asianbykv

Purpose	Prices European geometric Asian options using Kemna-Vorst model
Syntax	`Price = asianbykv(RateSpec,StockSpec,OptSpec,Strike,Settle,ExerciseDates)`
Description	`Price = asianbykv(RateSpec,StockSpec,OptSpec,Strike,Settle,ExerciseDates)` returns prices of European geometric Asian options using the Kemna-Vorst model.

Input Arguments

RateSpec - Interest-rate term structure
structure

The annualized continuously compounded interest-rate term structure specified by the `RateSpec` obtained from `intenvset`. For information on the interest-rate specification, see `intenvset`.

Data Types
struct

StockSpec - Stock specification for underlying asset
structure

Stock specification for underlying asset, specified using `StockSpec` obtained from `stockspec`. For information on the stock specification, see `stockspec`.

`stockspec` can handle other types of underlying assets. For example, stocks, stock indices and commodities. If dividends are not specified in `StockSpec`, dividends are assumed to be `0`.

Data Types
struct

OptSpec - Definition of option
string with value `'call'` or `'put'` | cell array of strings

Definition of option, specified as `'call'` or `'put'` using a NINST-by-1 cell array of strings.

Data Types
`cell | char`

Strike - Option strike price values

nonnegative integer | vector of nonnegative integers

Option strike price values, specified with nonnegative integers using a `NINST`-by-1 vector.

Data Types
`single | double`

Settle - Settlement dates or trade dates

nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates or trade dates for the Asian option, specified as a date string or as nonnegative integers using a `NINST`-by-1 vector or cell array of dates.

Data Types
`char | cell`

ExerciseDates - European option exercise dates

nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

European option exercise dates, specified as nonnegative integers or date strings using a `NINST`-by-1 vector or cell array of exercise dates. For an European option, there is only one `ExerciseDates` on the option expiry date.

Data Types
`char | cell`

Output Arguments

Price - Expected prices of an Asian option

vector

Expected prices of the Asian option, returned as an `NINST`-by-1 vector.

Examples

Compute the Price of an Asian Option Using the Kemna-Vorst Model

Define the `RateSpec`.

```
StartDates = 'Jan-1-2013';
EndDates = 'Jan-1-2014';
Rates = 0.035;
Basis = 1;
RateSpec = intenvset('ValuationDate', StartDates, 'StartDates', StartDates, ...
'EndDates', EndDates,'Rates', Rates,  'Compounding', -1, 'Basis', Basis)


RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9656
             Rates: 0.0350
          EndTimes: 1
        StartTimes: 0
          EndDates: 735600
        StartDates: 735235
    ValuationDate: 735235
             Basis: 1
       EndMonthRule: 1
```

Define the `StockSpec` for the asset.

```
AssetPrice = 100;
Sigma = 0.15;
DivType = 'continuous';
DivAmounts = 0.03;
StockSpec = stockspec(Sigma, AssetPrice, DivType, DivAmounts)


StockSpec =

            FinObj: 'StockSpec'
             Sigma: 0.1500
```

```
      AssetPrice: 100
     DividendType: {'continuous'}
  DividendAmounts: 0.0300
  ExDividendDates: []
```

Define the Asian `'call'` and `'put'` options.

```
Strike = 102;
OptSpec = {'put'; 'call'};
Settle = 'Jan-1-2013';
Maturity = 'Apr-1-2013';
```

Compute the European geometric Average Price for the Asian option using the Kemna-Vorst model.

```
Price = asiansensbykv(RateSpec, StockSpec, OptSpec, Strike, Settle, Maturity)

Price =

    2.8881
    0.9210
```

See Also   asiansensbykv | asianbycrr | intenvset | stockspec | asianbyls | asianbylevy

Concepts   • "Asian Option" on page 3-34

asiansensbykv

Purpose	Calculate prices and sensitivities of European geometric Asian options using Kemna-Vorst model

Syntax

```
PriceSens = asiansensbykv(RateSpec,StockSpec,OptSpec,StrikeSettle,
ExerciseDates)
PriceSens = asiansensbykv( ___ ,Name,Value)
```

Description

PriceSens = asiansensbykv(RateSpec,StockSpec,OptSpec,StrikeSettle, ExerciseDates) returns prices and sensitivities of European geometric Asian options using Kemna-Vorst model.

PriceSens = asiansensbykv(___ ,Name,Value) returns prices and sensitivities of European geometric Asian options using optional Name-Value pairs for the Kemna-Vorst model.

Input Arguments

RateSpec - Interest-rate term structure
structure

The annualized continuously compounded interest-rate term structure specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec - Stock specification for underlying asset
structure

Stock specification for underlying asset, specified using StockSpec obtained from stockspec. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, stocks, stock indices and commodities. If dividends are not specified in StockSpec, dividends are assumed to be 0.

Data Types
struct

OptSpec - Definition of option
string with values `'call'` or `'put'` | cell array of strings

Definition of option, specified as `'call'` or `'put'` using a `NINST`-by-1 cell array of strings.

Data Types
cell | char

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values, specified with nonnegative integers using a `NINST`- by-1 vector.

Data Types
single | double

Settle - Settlement dates or trade dates
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates or trade dates for the Asian option, specified as nonnegative integers or date strings using a `NINST`-by-1 vector or cell array of dates.

Data Types
char | cell

ExerciseDates - Option exercise dates
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

European option exercise dates, specified as nonnegative integers or date strings using a `NINST`-by-1 vector or cell array of exercise dates. For an European option, there is only one `ExerciseDates` on the option expiry date.

Data Types
char | cell

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: PriceSens = asiansensbykv(RateSpec,StockSpec,OptSpec,Strike,Settle,ExerciseDates,'Out

'OutSpec' - Define outputs

{'Price'} (default) | string with values: 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta' and 'All'. | cell array of strings with values: 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta' and 'All'.

Define outputs specifying NOUT- by-1 or 1-by-NOUT cell array of strings with possible values of 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta' and 'All'.

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as:

Example: OutSpec = {'delta','gamma','vega','lambda','rho','theta','price'}

Data Types
char | cell

Output Arguments

PriceSens - Expected prices and sensitivities of the Asian option
vector

Expected prices and sensitivities of the Asian option, returned as an 1-by-1 vector. If the OutSpec is not specified only price is returned.

Examples   **Compute the Price and Sensitivities of an Asian Option Using the Kemna-Vorst Model**

Define the `RateSpec`.

```
StartDates = 'Jan-1-2013';
EndDates = 'Jan-1-2014';
Rates = 0.035;
Basis = 1;
RateSpec = intenvset('ValuationDate', StartDates, 'StartDates', StartDates, ...
'EndDates', EndDates,'Rates', Rates,  'Compounding', -1, 'Basis', Basis)


RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9656
             Rates: 0.0350
          EndTimes: 1
        StartTimes: 0
          EndDates: 735600
        StartDates: 735235
     ValuationDate: 735235
             Basis: 1
       EndMonthRule: 1
```

Define the `StockSpec` for the asset.

```
AssetPrice = 100;
Sigma = 0.15;
DivType = 'continuous';
DivAmounts = 0.03;
StockSpec = stockspec(Sigma, AssetPrice, DivType, DivAmounts)


StockSpec =

            FinObj: 'StockSpec'
             Sigma: 0.1500
```

```
        AssetPrice: 100
      DividendType: {'continuous'}
   DividendAmounts: 0.0300
   ExDividendDates: []
```

Define the Asian `'call'` and `'put'` options.

```
Strike = 102;
OptSpec = {'put'; 'call'};
Settle = 'Jan-1-2013';
ExerciseDates = 'Jan-1-2014';
```

Compute the European geometric Average Price and sensitivities for the Asian option using the Kemna-Vorst model.

```
OutSpec = {'Price', 'Delta', 'Gamma'};
PriceSens = asiansensbykv(RateSpec, StockSpec, OptSpec, Strike,...
Settle, ExerciseDates,'OutSpec', OutSpec)

PriceSens =

    4.3871
    2.5163
```

See Also asianbykv | asianbycrr | stockspec | intenvset | asianbyls | asianbylevy

Concepts • "Asian Option" on page 3-34

Purpose   Price of European arithmetic Asian options using Levy model

Syntax   Price = asianbylevy(RateSpec,StockSpec,OptSpec,Strike,Settle,
                   ExerciseDates)

Description   Price =
                   asianbylevy(RateSpec,StockSpec,OptSpec,Strike,Settle,
                   ExerciseDates) returns European arithmetic average pricing for
                   Asian options using the Levy model.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec - Stock specification for underlying asset
structure

Stock specification for underlying asset, specified using StockSpec obtained from stockspec. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, stocks, stock indices, and commodities. If dividends are not specified in StockSpec, dividends are assumed to be 0.

Data Types
struct

OptSpec - Definition of option
string with values 'call' or 'put' | cell array of strings

Definition of option, specified as 'call' or 'put' using a NINST-by-1 cell array of strings.

Data Types
char | cell

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values, specified with nonnegative integers as a NINST-by-1 vector.

Data Types
single | double

Settle - Settlement dates or trade dates
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates or trade dates for the Asian option, specified as nonnegative integers or date strings using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

ExerciseDates - Option exercise dates
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Option exercise dates, specified as nonnegative integers or date strings using a NINST-by-1 vector or cell array of exercise dates. For a European option, there is only one ExerciseDates on the option expiry date.

Data Types
char | cell

Output Arguments

Price - Expected prices of Asian option
vector

Expected prices of the Asian option, returned as a NINST-by-1 vector.

Examples

Compute the Price of an Asian Option Using the Levy Model

Define the `RateSpec`.

```
Rates = 0.07;
StartDates = 'Jan-1-2013';
EndDates = 'Jan-1-2014';
RateSpec = intenvset('ValuationDate', StartDates, 'StartDates', StartDates, 'EndDates', ...
EndDates, 'Rates', Rates, 'Compounding', -1)


RateSpec =

              FinObj: 'RateSpec'
         Compounding: -1
                Disc: 0.9324
               Rates: 0.0700
            EndTimes: 1
          StartTimes: 0
            EndDates: 735600
          StartDates: 735235
      ValuationDate: 735235
               Basis: 0
         EndMonthRule: 1
```

Define the `StockSpec` for the asset.

```
AssetPrice = 6.8;
Sigma = 0.14;
DivType = 'continuous';
DivAmounts = 0.09;
StockSpec = stockspec(Sigma, AssetPrice, DivType, DivAmounts)


StockSpec =

              FinObj: 'StockSpec'
               Sigma: 0.1400
          AssetPrice: 6.8000
         DividendType: {'continuous'}
```

```
            DividendAmounts: 0.0900
            ExDividendDates: []
```

Define two options for `'call'` and `'put'`.

```
Settle = 'Jan-1-2013';
Maturity = 'July-1-2013';
Strike = 6.9;
OptSpec = {'call'; 'put'};
```

Compute the European arithmetic average price for the Asian option using the Levy model.

```
Price= asianbylevy(RateSpec, StockSpec, OptSpec, Strike, Settle, Maturity)

Price =

    0.0944
    0.2237
```

See Also   asiansensbylevy | asianbycrr | intenvset | stockspec | asianbyls | asianbykv

Concepts   • "Asian Option" on page 3-34

Purpose	Calculate prices and sensitivities of European arithmetic Asian options using Levy model
Syntax	PriceSens = asiansensbylevy(RateSpec,StockSpec,OptSpec,StrikeSettle, ExerciseDates) PriceSens = asiansensbylevy(___ ,Name,Value)
Description	PriceSens = asiansensbylevy(RateSpec,StockSpec,OptSpec,StrikeSettle, ExerciseDates) returns European average pricing and sensitivities for arithmetic Asian options using the Levy model. PriceSens = asiansensbylevy(___ ,Name,Value) returns European average pricing and sensitivities for arithmetic Asian options using the Levy model with optional name-value pair arguments.
Input Arguments	**RateSpec - Interest-rate term structure** structure Interest-rate term structure (annualized and continuously compounded), specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset. **Data Types** struct **StockSpec - Stock specification for underlying asset** structure Stock specification for underlying asset, specified using StockSpec obtained from stockspec. For information on the stock specification, see stockspec. stockspec can handle other types of underlying assets. For example, stocks, stock indices, and commodities. If dividends are not specified in StockSpec, dividends are assumed to be 0.

Data Types
struct

OptSpec - Definition of option
string with values `'call'` or `'put'` | cell array of strings

Definition of option, specified as `'call'` or `'put'` using a NINST-by-1 cell array of strings.

Data Types
char | cell

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values, specified with nonnegative integers using a NINST-by-1 vector.

Data Types
single | double

Settle - Settlement dates or trade dates
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates or trade dates for the Asian option, specified as nonnegative integers or date strings using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

ExerciseDates - Option exercise dates
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Option exercise dates, specified as nonnegative integers or date strings using a NINST-by-1 vector or cell array of exercise dates. For a European option, there is only one ExerciseDates on the option expiry date.

Data Types
char | cell

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments.
Name is the argument name and Value is the corresponding
value. Name must appear inside single quotes (' '). You can
specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

Example: PriceSens =
asiansensbylevy(RateSpec,StockSpec,OptSpec,Strike,Settle,ExerciseDates

'OutSpec' - Define outputs
{'Price'} (default) | string with values 'Price', 'Delta', 'Gamma',
'Vega', 'Lambda', 'Rho', 'Theta', and 'All'. | cell array of strings
with values 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho',
'Theta', and 'All'

Define outputs specifying NOUT- by-1 or 1-by-NOUT cell array of strings
with possible values of 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda',
'Rho', 'Theta' and 'All'.

OutSpec = {'All'} specifies that the output should be Delta, Gamma,
Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as
specifying OutSpec to include each sensitivity:

Example: OutSpec =
{'delta','gamma','vega','lambda','rho','theta','price'}

Data Types
char | cell

Output Arguments

PriceSens - Expected prices and sensitivities of Asian option
vector

Expected prices and sensitivities of the Asian option, returned as an
1-by-1 vector. If the OutSpec is not specified only the price is returned.

asiansensbylevy

Examples

Compute the Price and Sensitivities of an Asian Option Using the Levy Model

Define the `RateSpec`.

```
Rates = 0.07;
StartDates = 'Jan-1-2013';
EndDates = 'Jan-1-2014';
RateSpec = intenvset('ValuationDate', StartDates, 'StartDates', StartDates, 'EndDates', ...
EndDates, 'Rates', Rates, 'Compounding', -1)

RateSpec =

           FinObj: 'RateSpec'
      Compounding: -1
             Disc: 0.9324
            Rates: 0.0700
         EndTimes: 1
       StartTimes: 0
         EndDates: 735600
       StartDates: 735235
    ValuationDate: 735235
            Basis: 0
      EndMonthRule: 1
```

Define the `StockSpec` for the asset.

```
AssetPrice = 6.8;
Sigma = 0.14;
DivType = 'continuous';
DivAmounts = 0.09;
StockSpec = stockspec(Sigma, AssetPrice, DivType, DivAmounts)

StockSpec =

           FinObj: 'StockSpec'
            Sigma: 0.1400
        AssetPrice: 6.8000
```

```
        DividendType: {'continuous'}
     DividendAmounts: 0.0900
      ExDividendDates: []
```

Define two options for a 'call' and 'put'.

```
Settle = 'Jan-1-2013';
ExerciseDates = 'Jan-1-2014';
Strike = 6.9;
OptSpec = {'call'; 'put'};
```

Compute the European arithmetic average price and sensitivities for the Asian option using the Levy model.

```
OutSpec = {'Price', 'Delta', 'Gamma'};
PriceSens = asiansensbylevy(RateSpec, StockSpec, OptSpec, Strike,...
Settle, ExerciseDates,'OutSpec', OutSpec)

PriceSens =

    4.2895
    2.6001
```

See Also   asianbykv | asianbycrr | stockspec | intenvset | asianbyls
               | asianbykv

Concepts   • "Asian Option" on page 3-34

assetbybls

Purpose   Determine price of asset-or-nothing digital options using Black-Scholes model

Syntax   `Price = assetbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)`

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see `intenvset`.
StockSpec	Stock specification. See `stockspec`.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings `'call'` or `'put'`.
Strike	NINST-by-1 vector of payoff strike price values.

Description   `Price = assetbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)` computes asset-or-nothing option prices using the Black-Scholes option pricing model.

`Price` is a NINST-by-1 vector of expected option prices.

Examples   Consider two asset-or-nothing put options on a nondividend paying stock with a strike of 95 and 93 and expiring on January 30, 2009. On November 3, 2008 the stock is trading at 97.50. Using this data, calculate the price of the asset-or-nothing put options if the risk-free rate is 4.5% and the volatility is 22%.

Create the `RateSpec`:

```
Settle = 'Nov-3-2008';
Maturity = 'Jan-30-2009';
Rates = 0.045;
```

```
Compounding = -1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding);
```

Define the StockSpec:

```
AssetPrice = 97.50;
Sigma = .22;
StockSpec = stockspec(Sigma, AssetPrice);
```

Define the put options:

```
OptSpec = {'put'};
Strike = [95;93];
```

Calculate the price:

```
Paon = assetbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)

Paon =

   33.7666
   26.9662
```

See Also   assetsensbybls | cashbybls | gapbybls | supersharebybls

assetsensbybls

Purpose   Determine price and sensitivities of asset-or-nothing digital options using Black-Scholes model

Syntax
```
PriceSens = assetsensbybls(RateSpec, StockSpec, Settle,
Maturity, OptSpec, Strike)
PriceSens = assetsensbybls(RateSpec, StockSpec, Settle,
Maturity, OptSpec, Strike, OutSpec)
```

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see `intenvset`.
StockSpec	Stock specification. See `stockspec`.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings `'call'` or `'put'`.
Strike	NINST-by-1 vector of strike price values.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are:

                         • NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, or `'All'`.

                            For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output

should be `Price`, `Lambda`, and `Rho`, in that order.

To invoke from a function: `[Price, Lambda, Rho] = assetsensbybls(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})`

`OutSpec = {'All'}` specifies that the output should be `Delta`, `Gamma`, `Vega`, `Lambda`, `Rho`, `Theta`, and `Price`, in that order. This is the same as specifying `OutSpec` as `OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};`.

- Default is `OutSpec = {'Price'}`.

Description   `PriceSens = assetsensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)` computes asset-or-nothing option prices using the Black-Scholes option pricing model.

`PriceSens = assetsensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, OutSpec)` includes the parameter/value pairs defined for `OutSpec`, and computes asset-or-nothing option prices and sensitivities using the Black-Scholes option pricing model.

`PriceSens` is a `NINST`-by-1 vector of expected option prices and sensitivities.

Examples   Consider two asset-or-nothing put options on a nondividend paying stock with a strike of 95 and 93 and expiring on January 30, 2009. On November 3, 2008 the stock is trading at 97.50. Using this data, calculate the price and sensitivity of the asset-or-nothing put options if the risk-free rate is 4.5% and the volatility is 22%.

Create the `RateSpec`:

```
Settle = 'Nov-3-2008';
Maturity = 'Jan-30-2009';
```

```
Rates = 0.045;
Compounding = -1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding);
```

Define the StockSpec:

```
AssetPrice = 97.50;
Sigma = .22;
StockSpec = stockspec(Sigma, AssetPrice);
```

Define the put options:

```
OptSpec = {'put'};
Strike = [95;93];
```

Calculate the delta, price, and gamma:

```
OutSpec = { 'delta';'price';'gamma'};
[Delta, Price, Gamma] = assetsensbybls(RateSpec, StockSpec, Settle,...
Maturity, OptSpec, Strike, 'OutSpec', OutSpec)

Delta =

   -3.0833
   -2.8337


Price =

   33.7666
   26.9662


Gamma =

    0.0941
    0.1439
```

See Also   assetbybls

barrierbycrr

Purpose   Price barrier option from Cox-Ross-Rubinstein binomial tree

Syntax   [Price, PriceTree] = barrierbycrr(CRRTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, BarrierSpec, Barrier, Rebate, Options)

Arguments

CRRTree	Stock tree structure created by crrtree.
OptSpec	NINST-by-1 list of string values 'Call' or 'Put'.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.
Settle	NINST-by-1 vector of Settle dates. The settle date for every barrier option is set to the valuation date of the stock tree. The barrier argument Settle is ignored.
ExerciseDates	For a European option (AmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (AmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.

AmericanOpt	If AmericanOpt = O, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.
BarrierSpec	List of string values:
	'UI': Up Knock In
	'UO': Up Knock Out
	'DI': Down Knock In
	'DO': Down Knock Out
Barrier	Vector of barrier values.
Rebate	(Optional) NINST-by-1 matrix of rebate values. Default = 0. For Knock-in options, the rebate is paid at expiry. For Knock-out options, the rebate is paid when the barrier is reached.
Options	(Optional) Derivatives pricing options structure created with derivset.

See instbarrier for a description of barrier contract arguments.

Description [Price, PriceTree] = barrierbycrr(CRRTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, BarrierSpec, Barrier, Rebate, Options) computes the price of barrier options using a CRR binomial tree.

Price is a NINST-by-1 vector of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples Price a barrier option using a CRR binomial tree.

Load the file deriv.mat, which provides CRRTree. The CRRTree structure contains the stock specification and time information needed to price the option.

barrierbycrr

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'Call';
Strike = 105;
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2006';
AmericanOpt = 1;
BarrierSpec = 'UI';
Barrier = 102;

Price = barrierbycrr(CRRTree, OptSpec, Strike, Settle, ...
ExerciseDates, AmericanOpt, BarrierSpec, Barrier)

Price =

 12.1272
```

References   Derman, E., I. Kani, D. Ergener and I. Bardhan, "Enhanced Numerical Methods for Options with Barriers," *Financial Analysts Journal*, (Nov. - Dec. 1995), pp. 65-74.

See Also   crrtree | instbarrier

Purpose	Price barrier option from Equal Probabilities binomial tree

Syntax
```
[Price, PriceTree] = barrierbyeqp(EQPTree, OptSpec, Strike,
ExerciseDates, AmericanOpt, BarrierSpec, Barrier, Rebate,
Options)
```

Arguments

EQPTree	Stock tree structure created by eqptree.
OptSpec	NINST-by-1 list of string values 'Call' or 'Put'.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.
Settle	NINST-by-1 vector of Settle dates. The settle date for every barrier option is set to the valuation date of the stock tree. The barrier argument Settle is ignored.
ExerciseDates	For a European option (AmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (AmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.

barrierbyeqp

AmericanOpt	If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.	
BarrierSpec	List of string values:	
	'UI': Up Knock In	
	'UO': Up Knock Out	
	'DI': Down Knock In	
	'DO': Down Knock Out	
Barrier	Vector of barrier values.	
Rebate	(Optional) NINST-by-1 matrix of rebate values. Default = 0. For Knock-in options, the rebate is paid at expiry. For Knock-out options, the rebate is paid when the barrier is reached.	
Options	(Optional) Derivatives pricing options structure created with derivset.	

See instbarrier for a description of barrier contract arguments.

Description   [Price, PriceTree] = barrierbyeqp(EQPTree, OptSpec, Strike, ExerciseDates, AmericanOpt, BarrierSpec, Barrier, Rebate, Options) computes the price of barrier options using an equal probabilities binomial tree.

Price is a NINST-by-1 vector of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples   Price a barrier option using an EQP equity tree.

Load the file deriv.mat, which provides EQPTree. The EQPTree structure contains the stock specification and time information needed to price the option.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'Call';
Strike = 105;
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2006';
AmericanOpt = 1;
BarrierSpec = 'UI';
Barrier = 102;

Price = barrierbyeqp(EQPTree, OptSpec, Strike, Settle, ...
ExerciseDates, AmericanOpt, BarrierSpec, Barrier)

Price =

 12.2632
```

References   Derman, E., I. Kani, D. Ergener and I. Bardhan, "Enhanced Numerical Methods for Options with Barriers," *Financial Analysts Journal*, (Nov. - Dec. 1995), pp. 65-74.

See Also   eqptree | instbarrier

barrierbyitt

Purpose	Price barrier options using implied trinomial tree (ITT)
Syntax	`[Price, PriceTree] = barrierbyitt(ITTTree, OptSpec, Strike, ExerciseDates, AmericanOpt, BarrierSpec, Barrier, Rebate, Options)`

Arguments

`ITTTree`	Stock tree structure created by `itttree`.
`OptSpec`	NINST-by-1 list of string values `'Call'` or `'Put'`.
`Strike`	European and American option, NINST-by-1 vector of strike price values. Each row is the schedule for one option.
`Settle`	NINST-by-1 vector of `Settle` dates. The settle date for every barrier option is set to the valuation date of the stock tree. The barrier argument `Settle` is ignored.
`ExerciseDates`	For a European option (`AmericanOpt = 0`):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (`AmericanOpt = 1`):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-`NaN` date is listed, or if `ExerciseDates` is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.

AmericanOpt	If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.
BarrierSpec	List of string values:
	'UI': Up Knock In
	'UO': Up Knock Out
	'DI': Down Knock In
	'DO': Down Knock Out
Barrier	Vector of barrier values.
Rebate	(Optional) NINST-by-1 matrix of rebate values. Default = 0. For Knock In options, the rebate is paid at expiry. For Knock Out options, the rebate is paid when the barrier is reached.
Options	(Optional) Derivatives pricing options structure created with derivset.

See instbarrier for a description of barrier contract arguments.

Description   [Price, PriceTree] = barrierbyitt(ITTTree, OptSpec, Strike, ExerciseDates, AmericanOpt, BarrierSpec, Barrier, Rebate, Options) computes the price of barrier options using an implied trinomial tree.

Price is a NINST-by-1 vector of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Note The Settle date for every barrier option is set to the ValuationDate of the stock tree. The barrier argument, Settle, is ignored.

barrierbyitt

Examples   Price a barrier option using an ITT tree.

Load the file `deriv.mat` which provides the `ITTTree`. The `ITTTree` structure contains the stock specification and time information needed to price the option.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'Call';
Strike =  85;
Settle = '01-Jan-2006';
ExerciseDates = '31-Dec-2008';
AmericanOpt = 1;
BarrierSpec = 'UI';
Barrier =  115;

Price = barrierbyitt(ITTTree,OptSpec,Strike,Settle,ExerciseDates,AmericanOpt,...
BarrierSpec,Barrier)

Price =

2.407
```

References   Derman, E., I. Kani, D. Ergener, and I. Bardhan, "Enhanced Numerical Methods for Options with Barriers," *Financial Analysts Journal*, Nov.-Dec.,1995.

See Also   `instbarrier` | `itttree`

basketbyju

Purpose   Price European basket options using Nengjiu Ju approximation model

Syntax   ```
Price = basketbyju(RateSpec, BasketStockSpec, OptSpec, Strike,
Settle, Maturity)
```

**Description** `Price = basketbyju(RateSpec, BasketStockSpec, OptSpec,
Strike,
Settle, Maturity)` prices European basket options using the Nengjiu
Ju approximation model.

**Input Arguments**

**RateSpec**

Annualized, continuously compounded rate term structure. For more
information on the interest rate specification, see `intenvset`.

**BasketStockSpec**

`BasketStock` specification. For information on the basket of stocks
specification, see `basketstockspec`.

**OptSpec**

String or 2-by-1 cell array of the strings `'call'` or `'put'`.

**Strike**

Scalar for the option strike price.

**Settle**

Scalar of the settlement or trade date specified as a string or serial
date number.

**Maturity**

Maturity date specified as a string or serial date number.

**Output Arguments**

**Price**

Price of the basket option.

**Examples**  Find a European call basket option of two stocks. Assume that the stocks are currently trading at $10 and $11.50 with annual volatilities of 20% and 25%, respectively. The basket contains one unit of the first stock and one unit of the second stock. The correlation between the assets is 30%. On January 1, 2009, an investor wants to buy a 1-year call option with a strike price of $21.50. The current annualized, continuously compounded interest rate is 5%. Use this data to compute the price of the call basket option with the Ju approximation model.

```
Settle = 'Jan-1-2009';
Maturity  = 'Jan-1-2010';

% Define RateSpec
Rate = 0.05;
Compounding = -1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', ...
Settle, 'EndDates', Maturity, 'Rates', Rate, 'Compounding', Compounding);

% Define the Correlation matrix. Correlation matrices are symmetric, and
% have ones along the main diagonal.
Corr = [1 0.30; 0.30 1];

% Define BasketStockSpec
AssetPrice =  [10;11.50];
Volatility = [0.2;0.25];
Quantity = [1;1];
BasketStockSpec = basketstockspec(Volatility, AssetPrice, Quantity, Corr);

%Compute the price of the call basket option
OptSpec = {'call'};
Strike = 21.5;
PriceCorr30 = basketbyju(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, Maturity)
```

This returns:

```
PriceCorr30 =

    2.12214
```

Compute the price of the basket instrument for these two stocks with a correlation of 60%. Then compare this cost to the total cost of buying two individual call options:

```
Corr = [1 0.60; 0.60 1];

% Define the new BasketStockSpec
BasketStockSpec = basketstockspec(Volatility, AssetPrice, Quantity, Corr);

%Compute the price of the call basket option with Correlation = -0.60
PriceCorr60 = basketbyju(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, Maturity)
```

This returns:

```
PriceCorr60 =

    2.27566
```

The following table summarizes the sensitivity of the option to correlation changes. In general, the premium of the basket option decreases with lower correlation and increases with higher correlation.

Correlation	-0.60	-0.30	0	0.30	0.60
Premium	1.52830	1.76006	1.9527	2.1221	2.2756

Compute the cost of two vanilla 1-year call options using the Black-Scholes (BLS) model on the individual assets:

```
StockSpec = stockspec(Volatility, AssetPrice);
StrikeVanilla= [10;11.5];

PriceVanillaOption = optstockbybls(RateSpec, StockSpec, Settle, Maturity,...
OptSpec, StrikeVanilla)
```

This returns:

```
PriceVanillaOption =

    1.0451
    1.4186
```

Find the total cost of buying two individual call options:

```
sum(PriceVanillaOption)
```

This returns:

```
ans=2.4637
```

The total cost of purchasing two individual call options is $2.4637, compared to the maximum cost of the basket option of $2.27 with a correlation of 60%.

References   Nengjiu Ju, "Pricing Asian and Basket Options Via Taylor Expansion", *Journal of Computational Finance*, Vol. 5, 2002.

See Also   basketstockspec | basketsensbyju

How To   • "Basket Option" on page 3-25

Purpose   Price basket options using Longstaff-Schwartz model

Syntax
```
Price = basketbyls(RateSpec, BasketStockSpec, OptSpec, Strike,
Settle, ExerciseDates)
Price = basketbyls(RateSpec, BasketStockSpec, OptSpec, Strike,
Settle, ExerciseDates, 'ParameterName', ParameterValue ...)
```

Description   `Price = basketbyls(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, ExerciseDates)` prices basket options using the Longstaff-Schwartz model.

`Price = basketbyls(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, ExerciseDates, 'ParameterName', ParameterValue ...)` accepts optional inputs as one or more comma-separated parameter/value pairs. `'ParameterName'` is the name of the parameter inside single quotes. `'ParameterValue` is the value corresponding to `'ParameterName'`. Specify parameter-value pairs in any order. Names are case-insensitive and partial string matches are allowable, if no ambiguities exist.

Input Arguments

RateSpec

Annualized, continuously compounded rate term structure. For more information on the interest rate specification, see `intenvset`.

BasketStockSpec

BasketStock specification. For information on the basket of stocks specification, see `basketstockspec`.

OptSpec

String or 2-by-1 cell array of the strings `'call'` or `'put'`.

Strike

The option strike price:

- For a European or Bermuda option, `Strike` is a scalar (European) or 1-by-`NSTRIKES` (Bermuda) vector of strike price.

- For an American option, `Strike` is a scalar vector of the strike price.

Settle

Scalar of the settlement or trade date specified as a string or serial date number.

ExerciseDates

The exercise date for the option:

- For a European or Bermuda option, `ExerciseDates` is a 1-by-1 (European) or 1-by-`NSTRIKES` (Bermuda) vector of exercise dates. For a European option, there is only one `ExerciseDate` on the option expiry date.

- For an American option, `ExerciseDates` is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between, or including, the pair of dates on that row. If there is only one non-`NaN` date, or if `ExerciseDates` is 1-by-1, the option exercises between the `Settle` date and the single listed `ExerciseDate`.

Parameter–Value Pairs

AmericanOpt

Parameter values are a scalar flag.

- 0 — European/Bermuda

- 1 — American

Default: 0

NumPeriods

Parameter value is a scalar number of simulation periods per trial. `NumPeriods` is considered only when pricing European basket options.

For American and Bermuda basket options, `NumPeriod` equals the number of exercise days during the life of the option.

> **Default:** 100

NumTrials

Parameter value is a scalar number of independent sample paths (simulation trials).

> **Default:** 1000

Output Arguments

Price

Price of the basket option.

Examples

Find an American call basket option of three stocks. The stocks are currently trading at \$35, \$40 and \$45 with annual volatilities of 12%, 15% and 18%, respectively. The basket contains 33.33% of each stock. Assume the correlation between all pair of assets is 50%. On May 1, 2009, an investor wants to buy a three-year call option with a strike price of \$42. The current annualized continuously compounded interest rate is 5%. Use this data to compute the price of the call basket option using the Longstaff-Schwartz model.

```
Settle = 'May-1-2009';
Maturity  = 'May-1-2012';

% Define RateSpec
Rate = 0.05;
Compounding = -1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates',...
Settle, 'EndDates', Maturity, 'Rates', Rate, 'Compounding', Compounding);

% Define the Correlation matrix. Correlation matrices are symmetric,
% and have ones along the main diagonal.
Corr = [1 0.50 0.50; 0.50 1 0.50;0.50 0.50 1];
```

```
% Define BasketStockSpec
AssetPrice =  [35;40;45];
Volatility = [0.12;0.15;0.18];
Quantity = [0.333;0.333;0.333];
BasketStockSpec = basketstockspec(Volatility, AssetPrice, Quantity, Corr);

% Compute the price of the call basket option
OptSpec = {'call'};
Strike = 42;
AmericanOpt = 1; % American option
Price = basketbyls(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, Maturity,...
'AmericanOpt',AmericanOpt)
```

This returns:

```
Price =

   5.60499
```

Increase the number of simulation trials to 2000 to give the following results:

```
NumTrial = 2000;
Price = basketbyls(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, Maturity,...
'AmericanOpt',AmericanOpt,'NumTrials',NumTrial)

Price =

   5.6665
```

References Longstaff, F.A., and E.S. Schwartz, "Valuing American Options by Simulation: A Simple Least-Squares Approach", *The Review of Financial Studies*, Vol. 14, No. 1, Spring 2001, pp. 113–147.

See Also basketstockspec | basketsensbyls

How To • "Basket Option" on page 3-25

basketsensbyju

Purpose   Determine European basket options price and sensitivities using Nengjiu Ju approximation model

Syntax
```
PriceSens = basketsensbyju(RateSpec, BasketStockSpec, OptSpec, Strike,
Settle, Maturity)
PriceSens = basketsensbyju(RateSpec, BasketStockSpec, OptSpec, Strike,
Settle, Maturity, 'ParameterName', ParameterValue ...)
```

Description   PriceSens = basketsensbyju(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, Maturity) calculates prices and sensitivities for basket options using the Nengjiu Ju approximation model.

PriceSens = basketsensbyju(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, Maturity, *'ParameterName'*, *ParameterValue* ...) accepts optional inputs as one or more comma-separated parameter/value pairs. *'ParameterName'* is the name of the parameter inside single quotes. *'ParameterValue* is the value corresponding to *'ParameterName'*. Specify parameter-value pairs in any order. Names are case-insensitive and partial string matches are allowable, if no ambiguities exist.

Input Arguments

RateSpec

Annualized, continuously compounded rate term structure. For more information on the interest rate specification, see intenvset.

BasketStockSpec

BasketStock specification. For information on the basket of stocks specification, see basketstockspec.

OptSpec

String or 2-by-1 cell array of the strings 'call' or 'put'.

Strike

11-69

Scalar of the option strike price.

Settle

Scalar of the settlement or trade date specified as a string or serial date number.

Maturity

Maturity date, specified as a string or serial date number.

Parameter–Value Pairs

OutSpec

Parameter value is an NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'`. For example, OutSpec = {'Price', 'Lambda', 'Rho'} specifies that the output is Price, Lambda, and Rho, in that order.

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

> **Default:** OutSpec = {'Price'}

UndIdx

Scalar of the indice of the underlying instrument to compute the sensitivity.

> **Default:** UndIdx = []

Output Arguments

PriceSens

Expected prices or sensitivities values for the basket option.

Examples   Find a European call basket option of five stocks. Assume that the basket contains:

- 5% of the first stock trading at $110

- 15% of the second stock trading at $75

- 20% of the third stock trading at $40

- 25% of the fourth stock trading at $125

- 35% of the fifth stock trading at $92

These stocks have annual volatilities of 20% and the correlation between the assets is zero. On May 1, 2009, an investor wants to buy a 1-year call option with a strike price of $90. The current annualized, continuously compounded interest is 5%. Use this data to compute price and delta of the call basket option with the Ju approximation model.

```
Settle = 'May-1-2009';
Maturity  = 'May-1-2010';

% Define RateSpec
Rate = 0.05;
Compounding = -1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', ...
Settle, 'EndDates', Maturity, 'Rates', Rate, 'Compounding', Compounding);

% Define the Correlation matrix. Correlation matrices are symmetric, and
% have ones along the main diagonal.
NumInst  = 5;
InstIdx = ones(NumInst,1);
Corr = diag(ones(5,1), 0);

% Define BasketStockSpec
AssetPrice =  [110; 75; 40; 125; 92];
Volatility = 0.2;
Quantity = [0.05; 0.15; 0.2; 0.25; 0.35];
BasketStockSpec = basketstockspec(Volatility, AssetPrice, Quantity, Corr);
```

```
% Compute the price of the call basket option. Calculate also the delta
% of the first stock.
OptSpec = {'call'};
Strike = 90;
OutSpec = {'Price','Delta'};
UndIdx = 1; % First element in the basket
[Price, Delta] = basketsensbyju(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, ...
Maturity, 'OutSpec', OutSpec,'UndIdx', UndIdx)
```

This returns:

```
Price =

   5.16098

Delta =

   0.02972
```

Compute `Delta` with respect to the second asset:

```
UndIdx = 2; % Second element in the basket
OutSpec = {'Delta'};
Delta = basketsensbyju(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, Maturity, ...
'OutSpec',OutSpec,'UndIdx',UndIdx)
```

```
Delta =

  0.09063
```

References   Nengjiu Ju, "Pricing Asian and Basket Options Via Taylor Expansion", *Journal of Computational Finance*, Vol. 5, 2002.

See Also   basketstockspec | basketbyju

How To   • "Basket Option" on page 3-25

Purpose   Determine price and sensitivities for basket options using Longstaff-Schwartz model

Syntax
```
PriceSens = basketsensbyls(RateSpec, BasketStockSpec, OptSpec, Strike,
Settle, ExerciseDates)
PriceSens = basketsensbyls(RateSpec, BasketStockSpec, OptSpec, Strike,
Settle, ExerciseDates, 'ParameterName', ParameterValue ...)
```

Description   PriceSens = basketsensbyls(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, ExerciseDates) prices basket options using the Longstaff-Schwartz model.

PriceSens = basketsensbyls(RateSpec, BasketStockSpec, OptSpec, Strike, Settle, ExerciseDates, 'ParameterName', ParameterValue ...) accepts optional inputs as one or more comma-separated parameter/value pairs. 'ParameterName' is the name of the parameter inside single quotes. 'ParameterValue is the value corresponding to 'ParameterName'. Specify parameter-value pairs in any order. Names are case-insensitive and partial string matches are allowable, if no ambiguities exist.

Input Arguments

RateSpec

Annualized, continuously compounded rate term structure. For more information on the interest rate specification, see intenvset.

BasketStockSpec

BasketStock specification. For information on the basket of stocks specification, see basketstockspec.

OptSpec

String or 2-by-1 cell array of the strings 'call' or 'put'.

Strike

The option strike price:

- For a European or Bermuda option, `Strike` is a scalar (European) or 1-by-`NSTRIKES` (Bermuda) vector of strike price.

- For an American option, `Strike` is a scalar vector of strike price.

Settle

Scalar of settlement or trade date.

ExerciseDates

The exercise date for the option:

- For a European or Bermuda option, `ExerciseDates` is a 1-by-1 (European) or 1-by-`NSTRIKES` (Bermuda) vector of exercise dates. For a European option, there is only one `ExerciseDate` on the option expiry date.

- For an American option, `ExerciseDates` is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-`NaN` date, or if `ExerciseDates` is 1-by-1, the option exercises between the `Settle` date and the single listed `ExerciseDate`.

Parameter–Value Pairs

AmericanOpt

Parameter values are a scalar flag.

- 0 — European/Bermuda

- 1 — American

   Default: 0

NumPeriods

Parameter value is a scalar number of simulation periods. `NumPeriods` is considered only when pricing European basket options. For American and Bermuda basket options, `NumPeriod` equals the number of exercise days during the life of the option.

Default: 100

NumTrials

Parameter value is a scalar number of independent sample paths (simulation trials).

Default: 1000

OutSpec

Parameter value is an `NOUT`-by-`1` or `1`-by-`NOUT` cell array of strings indicating the nature and order of the outputs for the function. Possible values are `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'`. For example, `OutSpec = {'Price', 'Lambda', 'Rho'}` specifies that the output is `Price`, `Lambda`, and `Rho`, in that order.

`OutSpec = {'All'}` specifies that the output should be `Delta`, `Gamma`, `Vega`, `Lambda`, `Rho`, `Theta`, and `Price`, in that order. This is the same as specifying `OutSpec` as `OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};`.

Default: `OutSpec = {'Price'}`

UndIdx

Scalar of the indice of the underlying instrument to compute the sensitivity.

Default: `UndIdx = []`

basketsensbyls

Output Arguments

PriceSens

Expected prices or sensitivities values.

Examples

Find a European put basket option of two stocks. The basket contains 50% of each stock. The stocks are currently trading at $90 and $75, with annual volatilities of 15%. Assume that the correlation between the assets is zero. On May 1, 2009, an investor wants to buy a one-year put option with a strike price of $80. The current annualized, continuously compounded interest is 5%. Use this data to compute price and delta of the put basket option with the Longstaff-Schwartz approximation model.

```
Settle = 'May-1-2009';
Maturity  = 'May-1-2010';

% Define RateSpec
Rate = 0.05;
Compounding = -1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates',...
Settle, 'EndDates', Maturity, 'Rates', Rate, 'Compounding', Compounding);

% Define the Correlation matrix. Correlation matrices are symmetric,
% and have ones along the main diagonal.
NumInst  = 2;
InstIdx = ones(NumInst,1);
Corr = diag(ones(NumInst,1), 0);

% Define BasketStockSpec
AssetPrice =  [90; 75];
Volatility = 0.15;
Quantity = [0.50; 0.50];
BasketStockSpec = basketstockspec(Volatility, AssetPrice, Quantity, Corr);

% Compute the price of the put basket option. Calculate also the delta
% of the first stock.
OptSpec = {'put'};
```

```
Strike = 80;
OutSpec = {'Price','Delta'};
UndIdx = 1; % First element in the basket

[PriceSens, Delta] = basketsensbyls(RateSpec, BasketStockSpec, OptSpec,...
Strike, Settle, Maturity,'OutSpec', OutSpec,'UndIdx', UndIdx)
```

This returns:

```
PriceSens =

    1.08519

Delta =

  -0.10311
```

Compute the `Price` and `Delta` of the basket with a correlation of -20%:

```
NewCorr = [1 -0.20; -0.20 1];

% Define the new BasketStockSpec.
BasketStockSpec = basketstockspec(Volatility, AssetPrice, Quantity, NewCorr);

% Compute the price and delta of the put basket option.
[PriceSens, Delta] = basketsensbyls(RateSpec, BasketStockSpec, OptSpec,...
Strike, Settle, Maturity,'OutSpec', OutSpec,'UndIdx', UndIdx)

PriceSens =

   0.83903

Delta =

  -0.08847
```

basketsensbyls

References   Longstaff, F.A., and E.S. Schwartz, "Valuing American Options by Simulation: A Simple Least-Squares Approach", *The Review of Financial Studies*, Vol. 14, No. 1, Spring 2001, pp. 113–147.

See Also   basketstockspec | basketbyls

How To   • "Basket Option" on page 3-25

Purpose	Specify basket stock structure using Longstaff-Schwartz model

Syntax
BasketStockSpec = basketstockspec(Sigma, AssetPrice, Quantity, Correlation)
BasketStockSpec = basketstockspec(Sigma, AssetPrice, Quantity, Correlation, '*ParameterName*',*ParameterValue* ...)

Description
BasketStockSpec = basketstockspec(Sigma, AssetPrice, Quantity, Correlation) creates a basket stock structure.

BasketStockSpec = basketstockspec(Sigma, AssetPrice, Quantity, Correlation, '*ParameterName*',*ParameterValue* ...) accepts optional inputs as one or more comma-separated parameter/value pairs. '*ParameterName*' is the name of the parameter inside single quotes. '*ParameterValue* is the value corresponding to '*ParameterName*'. Specify parameter-value pairs in any order. Names are case-insensitive and partial string matches are allowable, if no ambiguities exist.

Input Arguments

Sigma

NINST-by-1 vector of decimal annual price volatility of the underlying security.

AssetPrice

NINST-by-1 vector of underlying asset price values at time 0.

Quantity

NINST-by-1 vector of quantities of the instruments contained in the basket.

Correlation

NINST-by-NINST matrix of correlation values.

basketstockspec

Parameter–Value Pairs

DividendAmounts

NINST-by-1 cell array specifying the dividend amounts for basket instruments. Each element of the cell array is a 1-by-NDIV row vector of cash dividends or a scalar representing a continuous annualized dividend yield for the corresponding instrument.

DividendType

NINST-by-1 cell array of strings specifying each stock's dividend type. Dividend type must be either `cash` for actual dollar dividends or `continuous` for continuous dividend yield. .

ExDividendDates

NINST-by-1 cell array specifying the ex-dividend dates for the basket instruments. Each row is a 1-by-NDIV matrix of ex-dividend dates for `cash` type. For rows that correspond to basket instruments with `continuous` dividend type, the cell is empty. If none of the basket instruments pay `continuous` dividends, do not specify `ExDividendDates`.

Output Arguments

BasketStockSpec

Structure encapsulating the properties of a basket stock structure.

Examples

Find a basket option of three stocks. The stocks are currently trading at $56, $92 and $125 with annual volatilities of 20%, 12% and 15%, respectively. The basket option contains 25% of the first stock, 40% of the second stock, and 35% of the third. The first stock provides a continuous dividend of 1%, while the other two provide no dividends. The correlation between the first and second asset is 30%, between the second and third asset 11%, and between the first and third asset 16%. Use this data to create the `BasketStockSpec` structure:

```
AssetPrice = [56;92;125];
Sigma = [0.20;0.12;0.15];
```

```
% Create the Correlation matrix. Correlation matrices are symmetric and
% have ones along the main diagonal.
NumInst  = 3;
Corr = zeros(NumInst,1);
Corr(1,2) = .30;
Corr(2,3) = .11;
Corr(1,3) = .16;
Corr = triu(Corr,1) + tril(Corr',-1) + diag(ones(NumInst,1), 0);

% Define dividends
DivType = cell(NumInst,1);
DivType{1}='continuous';
DivAmounts = cell(NumInst,1);
DivAmounts{1} = 0.01;

Quantity = [0.25; 0.40; 0.35];

BasketStockSpec = basketstockspec(Sigma, AssetPrice, Quantity, Corr, ...
'DividendType', DivType, 'DividendAmounts', DivAmounts)
```

This returns:

```
BasketStockSpec =

                FinObj: 'BasketStockSpec'
                 Sigma: [3x1 double]
            AssetPrice: [3x1 double]
              Quantity: [3x1 double]
           Correlation: [3x3 double]
          DividendType: {3x1 cell}
       DividendAmounts: {3x1 cell}
       ExDividendDates: {3x1 cell}
```

Examine the BasketStockSpec structure:

```
>>BasketStockSpec.Correlation
```

```
ans =

    1.0000    0.3000    0.1600
    0.3000    1.0000    0.1100
    0.1600    0.1100    1.0000
```

Find a basket option of two stocks. The stocks are currently trading at $60 and $55 with volatilities of 30% per annum. The basket option contains 50% of each stock. The first stock provides a cash dividend of $0.25 on May 1, 2009 and September 1, 2009. The second stock provides a continuous dividend of 3%. The correlation between the assets is 40%. Use this data to create the structure `BasketStockSpec`:

```
AssetPrice = [60;55];
Sigma = [0.30;0.30];

% Create the Correlation matrix. Correlation matrices are symmetric and
% have ones along the main diagonal.
Correlation = [1 0.40;0.40 1];

% Define dividends
NumInst  = 2;
DivType = cell(NumInst,1);
DivType{1}='cash';
DivType{2}='continuous';

DivAmounts = cell(NumInst,1);
DivAmounts{1} = [0.25 0.25];
DivAmounts{2} = 0.03;

ExDates = cell(NumInst,1);
ExDates{1} = {'May-1-2009' 'Sept-1-2009'};

Quantity = [0.5; 0.50];

BasketStockSpec = basketstockspec(Sigma, AssetPrice, Quantity, Correlation, ...
```

```
'DividendType', DivType, 'DividendAmounts', DivAmounts, 'ExDividendDates',ExDates)
```

This returns:

```
BasketStockSpec =

            FinObj: 'BasketStockSpec'
             Sigma: [2x1 double]
        AssetPrice: [2x1 double]
          Quantity: [2x1 double]
       Correlation: [2x2 double]
      DividendType: {2x1 cell}
   DividendAmounts: {2x1 cell}
   ExDividendDates: {2x1 cell}
```

Examine the BasketStockSpec structure:

```
>>BasketStockSpec.DividendType

ans =

    'cash'
    'continuous'
```

See Also   basketbyls | basketbyju | basketsensbyju | basketsensbyls |
               stockspec | intenvset

How To   • "Basket Option" on page 3-25

bdtprice

Purpose	Instrument prices from Black-Derman-Toy interest-rate tree
Syntax	[Price, PriceTree] = bdtprice(BDTTree, InstSet, Options)

Arguments

BDTTree	Interest-rate tree structure created by bdttree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type. Each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description [Price, PriceTree] = bdtprice(BDTTree, InstSet, Options) computes arbitrage-free prices for instruments using an interest-rate tree created with bdttree. All instruments contained in a financial instrument variable, InstSet, are priced.

Price is a number of instruments (NINST)-by-1 vector of prices for each instrument at time 0. The prices are computed by backward dynamic programming on the interest-rate tree. If an instrument cannot be priced, NaN is returned.

PriceTree is a MATLAB structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PTree contains the clean prices.

PriceTree.AITree contains the accrued interest.

PriceTree.tObs contains the observation times.

bdtprice handles instrument types: 'Bond', 'CashFlow', 'OptBond', 'OptEmBond', 'OptFloat', 'OptEmFloat', 'Fixed', 'Float', 'Cap',

'Floor', 'RangeFloat', 'Swap'. See instadd to construct defined types.

Related single-type pricing functions are:

- bondbybdt: Price a bond from a BDT tree.

- capbybdt: Price a cap from a BDT tree.

- cfbybdt: Price an arbitrary set of cash flows from a BDT tree.

- fixedbybdt: Price a fixed-rate note from a BDT tree.

- floatbybdt: Price a floating-rate note from a BDT tree.

- floorbybdt: Price a floor from a BDT tree.

- optbndbybdt: Price a bond option from a BDT tree.

- optfloatbybdt: Price a floating-rate note with an option from a BDT tree.

- optemfloatbybdt: Price a floating-rate note with an embedded option from a BDT tree.

- optembndbybdt: Price a bond with embedded option by a BDT tree.

- rangefloatbybdt: Price range floating note using a BDT tree.

- swapbybdt: Price a swap from a BDT tree.

- swaptionbybdt: Price a swaption from a BDT tree.

Examples   Load the BDT tree and instruments from the data file deriv.mat. Price the cap and bond instruments contained in the instrument set.

```
load deriv.mat;
BDTSubSet = instselect(BDTInstSet,'Type', {'Bond', 'Cap'});


instdisp(BDTSubSet)


%Table of instrument portfolio partially displayed:
Index Type   CouponRate Settle      Maturity    Period ...  Name ...
1     Bond   0.1        01-Jan-2000 01-Jan-2003 1      ...  10% bond
```

```
2     Bond   0.1        01-Jan-2000 01-Jan-2004 2      ... 10% bond

Index Type Strike Settle      Maturity    CapReset ... Name ...
3     Cap  0.15   01-Jan-2000 01-Jan-2004 1        ... 15% Cap
```

```
[Price, PriceTree] = bdtprice(BDTTree, BDTSubSet);
```

```
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

   95.5030
   93.9079
    1.4375
```

You can use `treeviewer` to see the prices of these three instruments along the price tree.

First 10% Bond (Maturity 2003)

Second 10% Bond (Maturity 2004)

15% Cap

Price the following multi-stepped coupon bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;
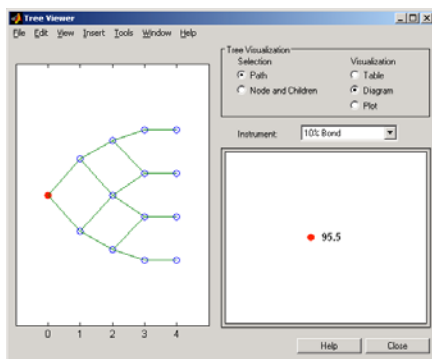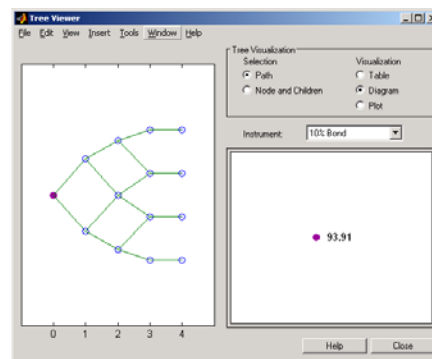
% Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create a portfolio of stepped coupon bonds with different maturities
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};

% Display the instrument portfolio
ISet = instbond(CouponRate, Settle, Maturity, 1);
instdisp(ISet)

%Table of instrument portfolio partially displayed:
Index Type CouponRate Settle  Maturity Period Basis EndMonthRule IssueDate FirstCouponDate ... Face
1 Bond [Cell]   01-Jan-2010 01-Jan-2011 1  0   1           NaN       NaN            ... 100
2 Bond [Cell]   01-Jan-2010 01-Jan-2012 1  0   1           NaN       NaN            ... 100
3 Bond [Cell]   01-Jan-2010 01-Jan-2013 1  0   1           NaN       NaN            ... 100
4 Bond [Cell]   01-Jan-2010 01-Jan-2014 1  0   1           NaN       NaN            ... 100

% Build the tree
% Assume the volatility to be 10%
Sigma = 0.1;
BDTTimeSpec = bdttimespec(ValuationDate, EndDates, Compounding);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
BDTT = bdttree(BDTVolSpec, RS, BDTTimeSpec);

% Compute the price of the  stepped coupon bonds
PBDT = bdtprice(BDTT, ISet)
```

```
PBDT =

   100.6763
   100.7368
   100.9266
   101.0115
```

Price a portfolio of stepped callable bonds and stepped vanilla bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

%Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create an instrument portfolio of 3 stepped callable bonds and three
% stepped vanilla bonds
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};
OptSpec='call';
Strike=100;
ExerciseDates='01-Jan-2011'; %Callable in one year

% Bonds with embedded option
ISet = instoptembnd(CouponRate, Settle, Maturity, OptSpec, Strike,...
ExerciseDates, 'Period', 1);
```

```
% Vanilla bonds
ISet = instbond(ISet, CouponRate, Settle, Maturity, 1);

% Display the instrument portfolio
instdisp(ISet)

%Table of instrument portfolio partially displayed:
```

Index	Type	CouponRate	Settle	Maturity	OptSpec	Strike	ExerciseDates	...	AmericanOpt
1	OptEmBond [Cell]	01-Jan-2010	01-Jan-2012	call	100	01-Jan-2011	...	0
2	OptEmBond [Cell]	01-Jan-2010	01-Jan-2013	call	100	01-Jan-2011	...	0
3	OptEmBond [Cell]	01-Jan-2010	01-Jan-2014	call	100	01-Jan-2011	...	0

Index	Type	CouponRate	Settle	Maturity	Period	Basis	EndMonthRule	...	Face
4	Bond [Cell]	01-Jan-2010	01-Jan-2012	1	0	1	...	100
5	Bond [Cell]	01-Jan-2010	01-Jan-2013	1	0	1	...	100
6	Bond [Cell]	01-Jan-2010	01-Jan-2014	1	0	1	...	100

```
% Build the tree
% Assume the volatility to be 10%
Sigma = 0.1;
BDTTimeSpec = bdttimespec(ValuationDate, EndDates, Compounding);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
BDTT = bdttree(BDTVolSpec, RS, BDTTimeSpec);

%The first three rows corresponds to the price of the stepped callable bonds and the
%last three rows corresponds to the price of the stepped vanilla bonds.
PBDT = bdtprice(BDTT, ISet)

PBDT =

   100.4799
   100.3228
   100.0840
   100.7368
   100.9266
   101.0115
```

Compute the price of a portfolio using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

%  Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create an instrument portfolio with two range notes and a floating rate
% note with the following data:
Spread = 200;
Settle = 'Jan-1-2011';
Maturity = 'Jan-1-2014';

% First Range Note:
RateSched(1).Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
RateSched(1).Rates  = [0.045 0.055; 0.0525  0.0675; 0.06 0.08];

% Second Range Note:
RateSched(2).Dates = {'Jan-1-2012'; 'Jan-1-2013' ; 'Jan-1-2014'};
RateSched(2).Rates  = [0.048 0.059; 0.055  0.068 ; 0.07 0.09];

 Create InstSet
InstSet = instadd('RangeFloat', Spread, Settle, Maturity, RateSched);

% Add a floating-rate note
InstSet = instadd(InstSet, 'Float', Spread, Settle, Maturity);

% Display the portfolio instrument
instdisp(InstSet)
```

Index	Type	Spread	Settle	Maturity	RateSched	FloatReset	Basis	Principal	EndMonthRule
1	RangeFloat	200	01-Jan-2011	01-Jan-2014	[Struct]	1	0	100	1
2	RangeFloat	200	01-Jan-2011	01-Jan-2014	[Struct]	1	0	100	1

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	EndMonthRule
3	Float	200	01-Jan-2011	01-Jan-2014	1	0	100	1

```
% The data to build the tree is as follows:
% Assume the volatility to be 10%.
Sigma = 0.1;
BDTTS = bdttimespec(ValuationDate, EndDates, Compounding);
BDTVS = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
BDTT = bdttree(BDTVS, RS, BDTTS);

% Price the portfolio
Price = bdtprice(BDTT, InstSet)

Price =

  100.2841
   98.0757
  105.5147
```

See Also bdtsens | bdttree | instadd | intenvprice | intenvsens

bdtsens

Purpose   Instrument prices and sensitivities from Black-Derman-Toy interest-rate tree

Syntax   [Delta, Gamma, Vega, Price] = bdtsens(BDTTree, InstSet, Options)

Arguments

BDTTree	Interest-rate tree structure created by bdttree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type. Each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Delta, Gamma, Vega, Price] = bdtsens(BDTTree, InstSet, Options) computes instrument sensitivities and prices for instruments using an interest-rate tree created with the bdttree function. NINST instruments from a financial instrument variable, InstSet, are priced. bdtsens handles instrument types: 'Bond', 'CashFlow', 'OptBond', 'OptEmBond', 'OptEmBond', 'OptFloat', 'OptEmFloat', 'Fixed', 'Float', 'Cap', 'Floor', 'RangeFloat', 'Swap'. See instadd for information on instrument types.

Delta is an NINST-by-1 vector of deltas, representing the rate of change of instrument prices with respect to changes in the interest rate. Delta is computed by finite differences in calls to bdttree. See bdttree for information on the observed yield curve.

Gamma is an NINST-by-1 vector of gammas, representing the rate of change of instrument deltas with respect to the changes in the interest rate. Gamma is computed by finite differences in calls to bdttree.

Vega is an NINST-by-1 vector of vegas, representing the rate of change of instrument prices with respect to the changes in the volatility

$\sigma(t,T)$. `Vega` is computed by finite differences in calls to `bdttree`. See `bdtvolspec` for information on the volatility process.

Note All sensitivities are returned as dollar sensitivities. To find the per-dollar sensitivities, divide by the respective instrument price.

`Price` is an `NINST`-by-1 vector of prices of each instrument. The prices are computed by backward dynamic programming on the interest-rate tree. If an instrument cannot be priced, `NaN` is returned.

`Delta` and `Gamma` are calculated based on yield shifts of 100 basis points. `Vega` is calculated based on a 1% shift in the volatility process.

Examples Load the tree and instruments from a data file. Compute `Delta` and `Gamma` for the cap and bond instruments contained in the instrument set.

```
load deriv.mat;
BDTSubSet = instselect(BDTInstSet,'Type', {'Bond', 'Cap'});

instdisp(BDTSubSet)

Index Type CouponRate Settle        Maturity     Period Name ...
1     Bond 0.1        01-Jan-2000   01-Jan-2003  1      10% Bond
2     Bond 0.1        01-Jan-2000   01-Jan-2004  2      10% Bond

Index Type Strike Settle      Maturity     CapReset... Name ...
3     Cap  0.15   01-Jan-2000 01-Jan-2004  1           15% Cap


[Delta, Gamma] = bdtsens(BDTTree, BDTSubSet)

Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Delta =
```

```
       -232.6681
       -281.0517
         63.8102

     Gamma =

       1.0e+03 *

         0.8037
         1.1819
         0.8535
```

See Also   bdtprice | bdttree | bdtvolspec | instadd

bdttimespec

Purpose   Specify time structure for Black-Derman-Toy interest-rate tree

Syntax   TimeSpec = bdttimespec(ValuationDate, Maturity, Compounding)

Arguments

ValuationDate	Scalar date marking the pricing date and first observation in the tree. Specify as serial date number or date string.
Maturity	Number of levels (depth) of the tree. A number of levels (NLEVELS)-by-1 vector of dates marking the cash flow dates of the tree. Cash flows with these maturities fall on tree nodes. Maturity should be in increasing order.
Compounding	(Optional) Scalar value representing the rate at which the input zero rates were compounded when annualized. Default = 1. This argument determines the formula for the discount factors:

Compounding = 1, 2, 3, 4, 6, 12

Disc = (1 + Z/F)^(-T), where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.

Compounding = 365

Disc = (1 + Z/F)^(-T), where F is the number of days in the basis year and T is a number of days elapsed computed by basis.

Compounding = -1

Disc = exp(-T*Z), where T is time in years.

bdttimespec

Description   TimeSpec = bdttimespec(ValuationDate, Maturity, Compounding) sets the number of levels and node times for a BDT tree and determines the mapping between dates and time for rate quoting.

TimeSpec is a structure specifying the time layout for bdttree. The state observation dates are [ValuationDate; Maturity(1:end-1)]. Because a forward rate is stored at the last observation, the tree can value cash flows out to Maturity.

Examples   Specify a five-period tree with annual nodes. Use annual compounding to report rates.

```
Compounding = 1;
ValuationDate = '01-01-2000';
Maturity = ['01-01-2001'; '01-01-2002'; '01-01-2003';
'01-01-2004'; '01-01-2005'];

TimeSpec = bdttimespec(ValuationDate, Maturity, Compounding)

TimeSpec =

            FinObj: 'BDTTimeSpec'
     ValuationDate: 730486
          Maturity: [5x1 double]
       Compounding: 1
             Basis: 0
       EndMonthRule: 1
```

See Also   bdttree | bdtvolspec

bdttree

Purpose	Construct Black-Derman-Toy interest-rate tree
Syntax	BDTTree = bdttree(VolSpec, RateSpec, TimeSpec)

Arguments

VolSpec	Volatility process specification. See bdtvolspec for information on the volatility process.
RateSpec	Interest-rate specification for the initial rate curve. See intenvset for information on declaring an interest-rate variable.
TimeSpec	Tree time layout specification. Defines the observation dates of the BDT tree and the Compounding rule for date to time mapping and price-yield formulas. See bdttimespec for information on the tree structure.

Description BDTTree = bdttree(VolSpec, RateSpec, TimeSpec) creates a structure containing time and interest-rate information on a recombining tree.

Examples Using the data provided, create a BDT volatility specification (VolSpec), rate specification (RateSpec), and tree time layout specification (TimeSpec). Then use these specifications to create a BDT tree with bdttree.

```
Compounding = 1;
ValuationDate = '01-01-2000';
StartDate = ValuationDate;
EndDates = ['01-01-2001'; '01-01-2002'; '01-01-2003';
'01-01-2004'; '01-01-2005'];
Rates = [.1; .11; .12; .125; .13];
Volatility = [.2; .19; .18; .17; .16];

RateSpec = intenvset('Compounding', Compounding,...
```

```
        'ValuationDate', ValuationDate,...
        'StartDates', StartDate,...
        'EndDates', EndDates,...
        'Rates', Rates);

BDTTimeSpec = bdttimespec(ValuationDate, EndDates, Compounding);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Volatility);
BDTTree = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec)

BDTTree =

     FinObj: 'BDTFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 1 2 3 4]
       dObs: [730486 730852 731217 731582 731947]
       TFwd: {[5x1 double]  [4x1 double]  [3x1 double]  [2x1 double]  [4]}
     CFlowT: {[5x1 double]  [4x1 double]  [3x1 double]  [2x1 double]  [5]}
    FwdTree: {[1.1000]  [1.0979 1.1432]  [1.0976 1.1377 1.1942]  [1.0872 1.1183 1.1606 1.2179]  [1x
```

Use treeviewer to observe the tree you have created.

```
treeviewer(BDTTree)
```

See Also   bdtprice | bdttimespec | bdtvolspec | intenvset

Purpose	Specify Black-Derman-Toy interest-rate volatility process
Syntax	Volspec = bdtvolspec(ValuationDate, VolDates, VolCurve, InterpMethod)

Arguments

ValuationDate	Scalar value representing the observation date of the investment horizon.
VolDates	Number of points (NPOINTS)-by-1 vector of yield volatility end dates.
VolCurve	NPOINTS-by-1 vector of yield volatility values in decimal form.
InterpMethod	(Optional) Interpolation method. Default is 'linear'. See interp1 for more information.

Description   Volspec = bdtvolspec(ValuationDate, VolDates, VolCurve, InterpMethod) creates a structure specifying the volatility for bdttree.

Examples   Using the data provided, create a BDT volatility specification (VolSpec).

```
ValuationDate = '01-01-2000';
EndDates = ['01-01-2001'; '01-01-2002'; '01-01-2003';
'01-01-2004'; '01-01-2005'];
Volatility = [.2; .19; .18; .17; .16];
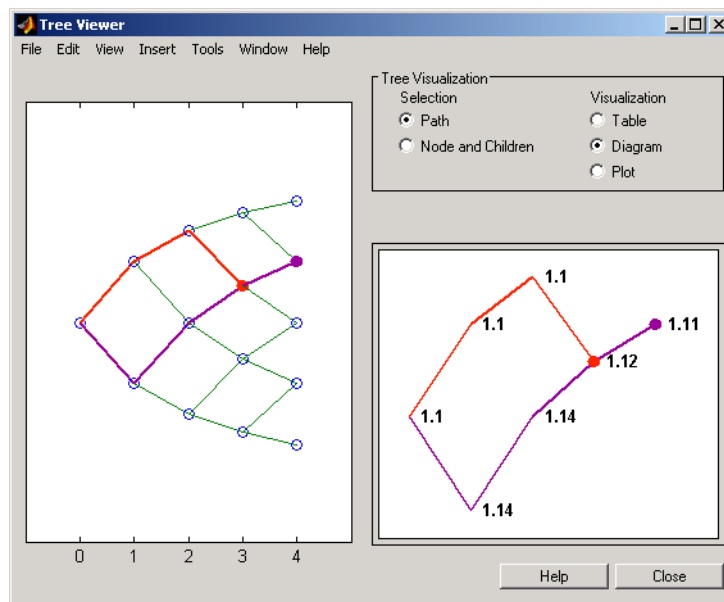
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Volatility)


BDTVolSpec =
              FinObj: 'BDTVolSpec'
       ValuationDate: 730486
             VolDates: [5x1 double]
             VolCurve: [5x1 double]
```

bdtvolspec

```
VolInterpMethod: 'linear'
```

See Also   bdttree | interp1

Purpose   Instrument prices from Black-Karasinski interest-rate tree

Syntax   [Price, PriceTree] = bkprice(BKTree, InstSet, Options)

Arguments

BKTree	Interest-rate tree structure created by bktree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type. Each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description [Price, PriceTree] = bkprice(BKTree, InstSet, Options)
computes arbitrage-free prices for instruments using an interest-rate
tree created with bktree. All instruments contained in a financial
instrument variable, InstSet, are priced.

Price is a number of instruments (NINST)-by-1 vector of prices for
each instrument. The prices are computed by backward dynamic
programming on the interest-rate tree. If an instrument cannot be
priced, NaN is returned.

PriceTree is a MATLAB structure of trees containing vectors of
instrument prices and accrued interest, and a vector of observation
times for each node.

PriceTree.PTree contains the clean prices.

PriceTree.AITree contains the accrued interest.

PriceTree.tObs contains the observation times.

bkprice handles instrument types: 'Bond', 'CashFlow', 'OptBond',
'OptEmBond', 'OptEmBond', 'OptFloat', 'OptEmFloat', 'Fixed',

'Float', 'Cap', 'Floor', 'RangeFloat', 'Swap'. See instadd to construct defined types.

Related single-type pricing functions are:

- bondbybk: Price a bond from a Black-Karasinski tree.
- capbybk: Price a cap from a Black-Karasinski tree.
- cfbybk: Price an arbitrary set of cash flows from a Black-Karasinski tree.
- fixedbybk: Price a fixed-rate note from a Black-Karasinski tree.
- floatbybk: Price a floating-rate note from a Black-Karasinski tree.
- floorbybk: Price a floor from a Black-Karasinski tree.
- optbndbybk: Price a bond option from a Black-Karasinski tree.
- optembndbybk: Price a bond with embedded option by a Black-Karasinski tree.
- optfloatbybdt: Price a floating-rate note with an option from a Black-Karasinski tree.
- optemfloatbybdt: Price a floating-rate note with an embedded option from a Black-Karasinski tree.
- rangefloatbybk: Price range floating note from a Black-Karasinski tree.
- swapbybk: Price a swap from a Black-Karasinski tree.
- swaptionbybk: Price a swaption from a Black-Karasinski tree.

Examples Load the BK tree and instruments from the data file deriv.mat. Price the cap and bond instruments contained in the instrument set.

```
load deriv.mat;
BKSubSet = instselect(BKInstSet,'Type', {'Bond', 'Cap'});


instdisp(BKSubSet)
```

```
%Table of instrument portfolio partially displayed:
Index Type  CouponRate Settle      Maturity    Period ... Name ...
1     Bond  0.03       01-Jan-2004 01-Jan-2007 1      ... 3% bond
2     Bond  0.03       01-Jan-2004 01-Jan-2008 2      ... 3% bond

Index Type Strike Settle      Maturity    CapReset ... Name ...
3     Cap  0.04   01-Jan-2004 01-Jan-2008 1        ... 4% Cap
```

```
[Price, PriceTree] = bkprice(BKTree, BKSubSet);

Price =

   98.1096
   95.6734
    2.2706
```

You can use `treeviewer` to see the prices of these three instruments along the price tree.

```
treeviewer(PriceTree, BKSubSet)
```

First 3% Bond (Maturity 2007)

Second 3% Bond (Maturity 2008)

4% Cap

Price the following multi-stepped coupon bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;
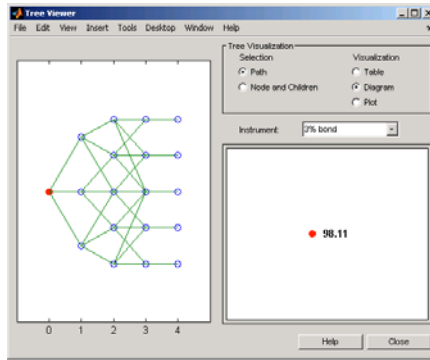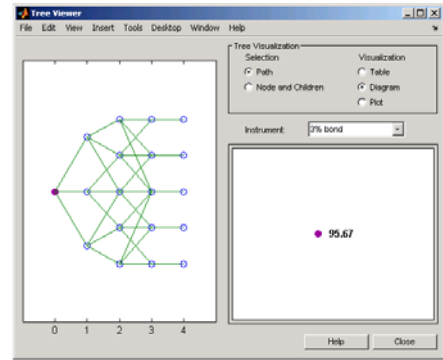
% Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create a portfolio of stepped coupon bonds with different maturities
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};

ISet = instbond(CouponRate, Settle, Maturity, 1);
instdisp(ISet)

%Table of instrument portfolio partially displayed:
Index Type CouponRate Settle  Maturity  Period Basis EndMonthRule ... Face
1 Bond [Cell]  01-Jan-2010    01-Jan-2011  1    0     1              ... 100
2 Bond [Cell]  01-Jan-2010    01-Jan-2012  1    0     1              ... 100
3 Bond [Cell]  01-Jan-2010    01-Jan-2013  1    0     1              ... 100
4 Bond [Cell]  01-Jan-2010    01-Jan-2014  1    0     1              ... 100

% Build the tree with the following data
VolDates = ['1-Jan-2011'; '1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014'];
VolCurve = 0.01;
AlphaDates = '01-01-2014';
AlphaCurve = 0.1;

BKVolSpec = bkvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RS.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RS, BKTimeSpec);
```

```
% Compute the price of the  stepped coupon bonds
PBK = bkprice(BKT, ISet)
```

```
 PBK =

   100.6763
   100.7368
   100.9266
   101.0115
```

Price a portfolio of stepped callable bonds and stepped vanilla bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

%Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create an instrument portfolio of 3 stepped callable bonds and three
% stepped vanilla bonds
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};
OptSpec='call';
Strike=100;
ExerciseDates='01-Jan-2011'; %Callable in one year

% Bonds with embedded option
```

```
ISet = instoptembnd(CouponRate, Settle, Maturity, OptSpec, Strike,...
ExerciseDates, 'Period', 1);

% Vanilla bonds
ISet = instbond(ISet, CouponRate, Settle, Maturity, 1);

% Display the instrument portfolio
instdisp(ISet)

%Table of instrument portfolio partially displayed:
Index Type CouponRate      Settle      Maturity    OptSpec Strike ExerciseDates ... AmericanOpt
1     OptEmBond [Cell]   01-Jan-2010  01-Jan-2012   call    100   01-Jan-2011    ... 0
2     OptEmBond [Cell]   01-Jan-2010  01-Jan-2013   call    100   01-Jan-2011    ... 0
3     OptEmBond [Cell]   01-Jan-2010  01-Jan-2014   call    100   01-Jan-2011    ... 0


Index Type CouponRate Settle        Maturity      Period Basis EndMonthRule ... Face
4     Bond [Cell]    01-Jan-2010  01-Jan-2012    1      0     1            ... 100
5     Bond [Cell]    01-Jan-2010  01-Jan-2013    1      0     1            ... 100
6     Bond [Cell]    01-Jan-2010  01-Jan-2014    1      0     1            ... 100


% Build the tree with the following data
VolDates = ['1-Jan-2011'; '1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014'];
VolCurve = 0.01;
AlphaDates = '01-01-2014';
AlphaCurve = 0.1;


BKVolSpec = bkvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RS.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RS, BKTimeSpec);

% The first three rows corresponds to the price of the stepped callable bonds
% and the last three rows corresponds to the price of the stepped vanilla bonds.

PBK = bkprice(BKT, ISet)
```

```
PBK =

   100.6735
   100.6763
   100.6763
   100.7368
   100.9266
   101.0115
```

Compute the price of a portfolio using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

%  Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create an instrument portfolio with two range notes and a floating rate
% note with the following data:
Spread = 200;
Settle = 'Jan-1-2011';
Maturity = 'Jan-1-2014';

% First Range Note:
RateSched(1).Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
RateSched(1).Rates  = [0.045 0.055; 0.0525  0.0675; 0.06 0.08];

% Second Range Note:
RateSched(2).Dates = {'Jan-1-2012'; 'Jan-1-2013' ; 'Jan-1-2014'};
```

```
RateSched(2).Rates  = [0.048 0.059; 0.055  0.068 ; 0.07 0.09];



% Create InstSet
InstSet = instadd('RangeFloat', Spread, Settle, Maturity, RateSched);

% Add a floating-rate note
InstSet = instadd(InstSet, 'Float', Spread, Settle, Maturity);

% Display the portfolio instrument
instdisp(InstSet)
```

Index	Type	Spread	Settle	Maturity	RateSched	FloatReset	Basis	Principal	EndMonthRule
1	RangeFloat	200	01-Jan-2011	01-Jan-2014	[Struct]	1	0	100	1
2	RangeFloat	200	01-Jan-2011	01-Jan-2014	[Struct]	1	0	100	1

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	EndMonthRule
3	Float	200	01-Jan-2011	01-Jan-2014	1	0	100	1

```
% The data to build the tree is as follows:
VolDates = ['1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014';'1-Jan-2015'];
VolCurve = 0.01;
AlphaDates = '01-01-2015';
AlphaCurve = 0.1;

BKVS = bkvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTS = bktimespec(RS.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVS, RS, BKTS);

% Price the portfolio
Price = bkprice(BKT, InstSet)

Price =

  105.5147
```

bkprice

```
101.4740
105.5147
```

See Also   bksens | bktree | instadd | intenvprice | intenvsens

Purpose   Instrument prices and sensitivities from Black-Karasinski interest-rate tree

Syntax   [Delta, Gamma, Vega, Price] = bksens(BKTree, InstSet, Options)

Arguments

BKTree	Interest-rate tree structure created by bktree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type. Each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Delta, Gamma, Vega, Price] = bksens(BKTree, InstSet, Options) computes instrument sensitivities and prices for instruments using an interest-rate tree created with the bktree function. NINST instruments from a financial instrument variable, InstSet, are priced. bksens handles instrument types: 'Bond', 'CashFlow', 'OptBond', 'OptEmBond', 'OptEmBond', 'OptFloat', 'OptEmFloat', 'Fixed', 'Float', 'Cap', 'Floor', 'RangeFloat', 'Swap'. See instadd for information on instrument types.

Delta is an NINST-by-1 vector of deltas, representing the rate of change of instrument prices with respect to changes in the interest rate. Delta is computed by finite differences in calls to bktree. See bktree for information on the observed yield curve.

Gamma is an NINST-by-1 vector of gammas, representing the rate of change of instrument deltas with respect to the changes in the interest rate. Gamma is computed by finite differences in calls to bktree.

Vega is an NINST-by-1 vector of vegas, representing the rate of change of instrument prices with respect to the changes in the volatility $\sigma(t,T)$. Vega is computed by finite differences in calls to bktree. See bkvolspec for information on the volatility process.

Note All sensitivities are returned as dollar sensitivities. To find the per-dollar sensitivities, divide by the respective instrument price.

Price is an NINST-by-1 vector of prices of each instrument. The prices are computed by backward dynamic programming on the interest-rate tree. If an instrument cannot be priced, NaN is returned.

Delta and Gamma are calculated based on yield shifts of 100 basis points. Vega is calculated based on a 1% shift in the volatility process.

Examples   Load the tree and instruments from a data file. Compute Delta and Gamma for the cap and bond instruments contained in the instrument set.

```
load deriv.mat;
BKSubSet = instselect(BKInstSet,'Type', {'Bond', 'Cap'});

instdisp(BKSubSet)

%Table of instrument portfolio partially displayed:
Index Type CouponRate Settle        Maturity     Period ... Name...
1     Bond 0.03       01-Jan-2004   01-Jan-2007  1      ... 3% Bond
2     Bond 0.03       01-Jan-2004   01-Jan-2008  1      ... 3% Bond

Index Type Strike Settle       Maturity     CapReset ... Name ...
3     Cap  0.04   01-Jan-2004  01-Jan-2008  1        ... 4% Cap

[Delta, Gamma] = bksens(BKTree, BKSubSet)

Delta =
```

```
   -285.7151
   -365.7048
    189.5319

Gamma =

  1.0e+003 *

     0.8456
     1.4345
     6.9999
```

See Also   bkprice | bktree | bkvolspec | instadd

bktimespec

Purpose	Specify time structure for Black-Karasinski tree
Syntax	TimeSpec = bktimespec(ValuationDate, Maturity, Compounding)

Arguments

ValuationDate	Scalar date marking the pricing date and first observation in the tree. Specify as a serial date number or date string.
Maturity	Number of levels (depth) of the tree. A number of levels (NLEVELS)-by-1 vector of dates marking the cash flow dates of the tree. Cash flows with these maturities fall on tree nodes. Maturity should be in increasing order.
Compounding	(Optional) Scalar value representing the rate at which the input zero rates were compounded when annualized. Default = -1 (continuous compounding). This argument determines the formula for the discount factors:
	Compounding = 1, 2, 3, 4, 6, 12 = F
	Disc = (1 + Z/F)^(-T), where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.
	Compounding = 365
	Disc = (1 + Z/F)^(-T), where F is the number of days in the basis year and T is a number of days elapsed computed by basis.
	Compounding = -1
	Disc = exp(-T*Z), where T is time in years.

Description   TimeSpec = bktimespec(ValuationDate, Maturity, Compounding)
sets the number of levels and node times for an BK tree and determines
the mapping between dates and time for rate quoting.

TimeSpec is a structure specifying the time layout for bktree. The
state observation dates are [Settle; Maturity(1:end-1)]. Because a
forward rate is stored at the last observation, the tree can value cash
flows out to Maturity.

Examples   Specify a four-period tree with annual nodes. Use annual compounding
to report rates.

```
ValuationDate = 'Jan-1-2004';
Maturity = ['12-31-2004'; '12-31-2005'; '12-31-2006';
'12-31-2007'];
Compounding = 1;
TimeSpec = bktimespec(ValuationDate, Maturity, Compounding)


TimeSpec =

            FinObj: 'BKTimeSpec'
     ValuationDate: 731947
          Maturity: [4x1 double]
       Compounding: 1
             Basis: 0
       EndMonthRule: 1
```

See Also   bktree | bkvolspec | hwtree

bktree

Purpose	Construct Black-Karasinski interest-rate tree
Syntax	BKTree = bktree(VolSpec, RateSpec, TimeSpec) BKTree = bktree(VolSpec, RateSpec, TimeSpec, Name,Value)

Description

BKTree = bktree(VolSpec, RateSpec, TimeSpec) creates a structure containing time and interest-rate information on a recombining tree.

BKTree = bktree(VolSpec, RateSpec, TimeSpec, Name,Value) creates a structure containing time and interest-rate information on a recombining tree with additional options specified by one or more Name,Value pair arguments.

Input Arguments

VolSpec

Volatility process specification. See bkvolspec for information on the volatility process.

RateSpec

Interest-rate specification for the initial rate curve. See intenvset for information on declaring an interest-rate variable.

TimeSpec

Tree time layout specification. Defines the observation dates of the BK tree and the compounding rule for date to time mapping and price-yield formulas. See bktimespec for information on the tree structure.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Method'

String specifying the Hull-White method upon which the tree-node connectivity algorithm is based. Possible values are HW1996 and HW2000.

Note bktree supports two tree-node connectivity algorithms. HW1996 is based on the original paper published in the *Journal of Derivatives*, and HW2000 is the general version of the algorithm, as specified in the paper published in August 2000.

Default: HW1996

Output Arguments

BKTree

Structure containing time and interest rate information of a trinomial recombining tree.

Examples

Using the data provided, create a BK volatility specification (VolSpec), rate specification (RateSpec), and tree time layout specification (TimeSpec). Then use these specifications to create a BK tree using bktree.

```
Compounding = -1;
ValuationDate = '01-01-2004';
StartDate = ValuationDate;
VolDates = ['12-31-2004'; '12-31-2005'; '12-31-2006';
'12-31-2007'];
VolCurve = 0.01;
AlphaDates = '01-01-2008';
AlphaCurve = 0.1;
Rates = [0.0275; 0.0312; 0.0363; 0.0415];

BKVolSpec = bkvolspec(ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);

RateSpec = intenvset('Compounding', Compounding,...
     'ValuationDate', ValuationDate,...
```

```
        'StartDates', ValuationDate,...
        'EndDates', VolDates,...
        'Rates', Rates);

BKTimeSpec = bktimespec(ValuationDate, VolDates, Compounding);

BKTree = bktree(BKVolSpec, RateSpec, BKTimeSpec)

BKTree =

     FinObj: 'BKFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 0.9973 1.9973 2.9973]
       dObs: [731947 732312 732677 733042]
     CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [3.9973]}
      Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
    Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
     FwdTree: {1x4 cell}
```

Use treeviewer to observe the tree you have created.

```
treeviewer(BKTree)
```

Using the data provided, create a Hull-White volatility specification (VolSpec), rate specification (RateSpec), and tree time layout specification (TimeSpec). Then use these specifications to create a Hull-White tree using hwtree.

```
Compounding = -1;
ValuationDate = '01-01-2004';
StartDate = ValuationDate;
VolDates = ['12-31-2004'; '12-31-2005'; '12-31-2006';
'12-31-2007'];
VolCurve = 0.01;
AlphaDates = '01-01-2008';
AlphaCurve = 0.1;
Rates = [0.0275; 0.0312; 0.0363; 0.0415];

HWVolSpec = hwvolspec(ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
```

```
RateSpec = intenvset('Compounding', Compounding,...
     'ValuationDate', ValuationDate,...
     'StartDates', ValuationDate,...
     'EndDates', VolDates,...
     'Rates', Rates);

HWTimeSpec = hwtimespec(ValuationDate, VolDates, Compounding);
HWTree = hwtree(HWVolSpec, RateSpec, HWTimeSpec)

HWTree =

      FinObj: 'HWFwdTree'
     VolSpec: [1x1 struct]
    TimeSpec: [1x1 struct]
    RateSpec: [1x1 struct]
        tObs: [0 0.9973 1.9973 2.9973]
        dObs: [731947 732312 732677 733042]
       CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [3.9973]}
        Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
      Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
      FwdTree: {1x4 cell}
```

Use treeviewer to observe the tree you have created.

```
treeviewer(HWTree)
```

References   Hull, J., and A. White, "Using Hull-White Interest Rate Trees", *Journal of Derivatives*, 1996.

Hull, J., and A. White, "The General Hull-White Model and Super Calibration", August 2000.

See Also | bkprice | bktimespec | bkvolspec | intenvset

Tutorials   • "Calibrating Hull-White Model Using Market Data" on page 2-79

bkvolspec

Purpose   Specify Black-Karasinski interest-rate volatility process

Syntax   Volspec = bkvolspec(ValuationDate, VolDates, VolCurve,
AlphaDates, AlphaCurve, InterpMethod)

Arguments

ValuationDate	Scalar value representing the observation date of the investment horizon.
VolDates	Number of points (NPOINTS)-by-1 vector of yield volatility end dates.
VolCurve	NPOINTS-by-1 vector of annualized yield volatility values in decimal form. Volatility is the standard deviation of proportional changes in the rate.
AlphaDates	NPOINTS-by-1 vector of mean reversion end dates.
AlphaCurve	NPOINTS-by-1 vector of positive mean reversion values in decimal form.
InterpMethod	(Optional) Interpolation method. Default is 'linear'. See interp1 for more information.

Description   Volspec = bkvolspec(ValuationDate, VolDates, VolCurve,
AlphaDates, AlphaCurve, InterpMethod) creates a structure
specifying the volatility for bktree.

Examples   Using the data provided, create a Black-Karasinski volatility
specification (VolSpec).

```
ValuationDate = '01-01-2004';
StartDate = ValuationDate;
VolDates = ['12-31-2004'; '12-31-2005'; '12-31-2006';
'12-31-2007'];
```

```
VolCurve = 0.01;
AlphaDates = '01-01-2008';
AlphaCurve = 0.1;
BKVolSpec = bkvolspec(ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve)

BKVolSpec =

              FinObj: 'BKVolSpec'
       ValuationDate: 731947
            VolDates: [4x1 double]
            VolCurve: [4x1 double]
          AlphaCurve: 0.1000
          AlphaDates: 733408
       VolInterpMethod: 'linear'
```

See Also   bktree | interp1

bondbybdt

Purpose   Price bond from Black-Derman-Toy interest-rate tree

Syntax   [Price, PriceTree] = bondbybdt(BDTTree, CouponRate, Settle,
                Maturity)
                [Price, PriceTree] = bondbybdt(BDTTree, CouponRate, Settle,
                Maturity, Period, Basis, EndMonthRule, IssueDate,
                FirstCouponDate, LastCouponDate, StartDate, Face,
                Options)
                [Price, PriceTree] = bondbybdt(BDTTree, CouponRate, Settle,
                Maturity,Name,Value)

**Input
Arguments**

BDTTree	Interest-rate tree structure created by bdttree.
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector or NINST-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.

The Settle date for every bond is set to the ValuationDate of the BDT
tree. The bond argument Settle is ignored.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Period

Coupons per year of the bond. A vector of integers. Values are 1, 2, 3, 4, 6, and 12.

> **Default:** 2

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Default: 0 (actual/actual)

EndMonthRule

End-of-month rule. A NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Default: 1

IssueDate

Date when a bond was issued.

FirstCouponDateDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure. If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

LastCouponDateDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify StartDate, the effective start date is the Settle date.

Face

Face or par value. Face is a NINST-by-1 vector or NINST-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid.

> **Default:** 100

Options

Derivatives pricing options structure created with derivset.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1 of logicals.

> **Default:** false

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with possible choices of business day convention:

- actual

bondbybdt

- `follow`

- `modifiedfollow`

- `previous`

- `modifiedprevious`

Default: `actual`

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

Default: If no dates are specified, `holidays.m` is used.

Description   `[Price, PriceTree] = bondbybdt(BDTTree, CouponRate, Settle, Maturity)` computes the price of a bond from a BDT interest-rate tree.

`[Price, PriceTree] = bondbybdt(BDTTree, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face, Options)` computes the price of a bond from a BDT interest-rate tree using optional input arguments.

`[Price, PriceTree] = bondbybdt(BDTTree, CouponRate, Settle, Maturity,Name,Value)` computes the price of a bond from a BDT interest-rate tree with additional options specified by one or more `Name,Value` pair arguments.

`Price` is a number of instruments (`NINST`)-by-1 matrix of expected prices at time 0.

`PriceTree` is a MATLAB structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node. Within `PriceTree`:

- `PriceTree.PTree` contains the clean prices.

- `PriceTree.AITree` contains the accrued interest.

- `PriceTree.tObs` contains the observation times.

`bondbybdt` computes prices of vanilla bonds, stepped coupon bonds, and amortizing bonds with no market purchase option and no call provisions.

Definitions

Vanilla Bond

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment.

Stepped Coupon Bond

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond.

Bond with an Amortization Schedule

An amortized bond is treated as an asset, with the discount amount being amortized to interest expense over the life of the bond.

Examples

Price a Bond Using a BDT Tree

Price a 10% bond using a BDT interest-rate tree.

Load `deriv.mat`, which provides `BDTTree`. The `BDTTree` structure contains the time and interest-rate information needed to price the bond.

```
load deriv.mat;
```

Define the bond using the required arguments. Other arguments use defaults.

```
CouponRate = 0.10;
Settle = '01-Jan-2000';
```

```
Maturity = '01-Jan-2003';
Period = 1;
```

Use bondbybdt to compute the price of the bond.

```
Price = bondbybdt(BDTTree, CouponRate, Settle, Maturity, Period)

Price =

  95.5030
```

Price a Stepped Coupon Bond

Price single stepped coupon bonds using market data.

Define the interest-rate term structure.

```
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;
```

Create the RateSpec.

```
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates, 'EndDates',...
EndDates, 'Rates', Rates, 'Compounding', Compounding)

RS =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: [4x1 double]
           Rates: [4x1 double]
        EndTimes: [4x1 double]
      StartTimes: [4x1 double]
        EndDates: [4x1 double]
      StartDates: 734139
```

```
      ValuationDate: 734139
              Basis: 0
        EndMonthRule: 1
```

Create the stepped bond instrument.

```
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
Period = 1;
```

Build the BDT tree and assume the volatility to be 10% using the following market data:

```
Sigma = 0.1;
BDTTimeSpec = bdttimespec(ValuationDate, EndDates);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
BDTT = bdttree(BDTVolSpec, RS, BDTTimeSpec)
```

Compute the price of the stepped coupon bonds.

```
PBDT= bondbybdt(BDTT, CouponRate, Settle,Maturity , Period)

PBDT =

  100.7246
  100.0945
  101.5900
  102.0820
```

Price Two Bonds with Amortization Schedules

Price two bonds with amortization schedules using the Face input argument to define the schedule.

Define the interest-rate term structure.

```
Rates = 0.035;
ValuationDate = '1-Nov-2011';
```

```
StartDates = ValuationDate;
EndDates = '1-Nov-2017';
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: 0.8135
             Rates: 0.0350
          EndTimes: 6
        StartTimes: 0
          EndDates: 737000
        StartDates: 734808
     ValuationDate: 734808
             Basis: 0
       EndMonthRule: 1
```

Create the bond instrument. The bonds have a coupon rate of 4% and 3.85%, a period of one year, and mature on 1-Nov-2017.

```
CouponRate = [0.04; 0.0385];
Settle ='1-Nov-2011';
Maturity = '1-Nov-2017';
Period = 1;
```

Define the amortizing schedule.

```
Face = {{'1-Nov-2015' 100;'1-Nov-2016' 85;'1-Nov-2017' 70};
{'1-Nov-2015' 100;'1-Nov-2016' 90;'1-Nov-2017' 80}};
```

Build the BDT tree and assume the volatility to be 10%.

```
MatDates = {'1-Nov-2012'; '1-Nov-2013';'1-Nov-2014';'1-Nov-2015';'1-Nov-2016';'1-Nov-2017'};
```

```
BDTTimeSpec = bdttimespec(ValuationDate, MatDates);
Volatility = 0.1;
BDTVolSpec = bdtvolspec(ValuationDate, MatDates, Volatility*ones(1,length(MatDates))');
BDTT = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);
```

Compute the price of the amortizing bonds.

```
Price = bondbybdt(BDTT, CouponRate, Settle, Maturity, 'Period',Period,...
'Face', Face)

Price =

  102.4791
  101.7786
```

See Also   bdttree | bdtprice | cfamounts | instbond

blackvolbyrebonato

Purpose	Compute Black volatility for LIBOR Market Model using Rebonato formula
Syntax	`outVol = blackvolbyrebonato(ZeroCurve,VolFunc,CorrMat, ExerciseDate,Maturity)` `outVol = blackvolbyrebonato( ___ ,Name,Value)`
Description	`outVol = blackvolbyrebonato(ZeroCurve,VolFunc,CorrMat, ExerciseDate,Maturity)` computes the Black volatility for a swaption using a LIBOR Market Model. `outVol = blackvolbyrebonato( ___ ,Name,Value)` computes the Black volatility for a swaption using a LIBOR Market Model with optional name-value pair arguments.
Input Arguments	**ZeroCurve - Zero-curve for `LiborMarketModel` model** structure Zero-curve for the `LiborMarketModel`, specified using `IRDataCurve` or `RateSpec`. **Data Types** `struct` **VolFunc - Function handle for volatility** cell array of function handles Function handle for volatility, specified by a `NumRates`-by-1 cell array of function handles. Each function handle must take time as an input and return a scalar volatility **Data Types** `cell	function_handle`    **CorrMat - Correlation matrix**   vector    Correlation matrix, specified by `NumRates`-by-`NumRates`.

Data Types
single | double

ExerciseDate - Swaption exercise date
nonnegative integer | vector of nonnegative integers | string of dates

Swaption exercise dates, specified by a NumSwaptions-by-1 vector of serial date numbers or date strings.

Data Types
single | double | cell

Maturity - Swaption maturity date
nonnegative integer | vector of nonnegative integers | string of dates

Swaption maturity dates, specified using a NumSwaptions-by-1 vector of dates or date strings.

Data Types
single | double | cell

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Vol =
blackvolbyrebonato(irdc,VolFunc,CorrMat,ExerciseDate,Maturity,'Period

'Period' - Compounding frequency of curve and reset of swaptions
2 (default) | positive integer from the set [1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Compounding frequency of curve and reset of swaptions, specified as a positive integers for the values 1,2,4,6,12 in a NumSwaptions-by-1 vector.

11-137

Data Types
single | double

Output Arguments

outVol - Black volatility for specified swaption
scalar | vector

Black volatility, returned as a vector for the specified swaptions.

Algorithms

The Rebonato approximation formula relates the Black volatility for a European swaption, given a set of volatility functions and a correlation matrix

$$(\upsilon_{\alpha,\beta}^{LFM})^2 = \sum_{i,j=\alpha+1}^{\beta} \frac{w_i(0)w_j(0)F_i(0)F_j(0)\rho_{i,j}}{S_{\alpha,\beta}(0)^2} \int_0^{T_\alpha} \sigma_i(t)\sigma_j(t)dt$$

where:

$$w_i(t) = \frac{\tau_i P(t,T_i)}{\sum_{k=\alpha+1}^{\beta} \tau_\kappa P(t,t_\kappa)}$$

Examples

Price Swaption for LIBOR Market Model Using the Rebonato Formula

Define the input maturity and tenor for a LIBOR Market Model (LMM) specified by the cell array of volatility function handles, and a correlation matrix for the LMM.

```
Settle = datenum('11-Aug-2004');

% Zero Curve
CurveTimes = (1:10)';
CurveDates = daysadd(Settle,360*CurveTimes,1);

ZeroRates = [0.03 0.033 0.036 0.038 0.04 0.042 0.043 0.044 0.045 0.046]';
```

```
% Construct an IRCurve
irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

LMMVolFunc = @(a,t) (a(1)*t + a(2)).*exp(-a(3)*t) + a(4);
LMMVolParams = [.3 -.02 .7 .14];

numRates = length(ZeroRates);
VolFunc(1:numRates-1) = {@(t) LMMVolFunc(LMMVolParams,t)};

Beta = .08;
CorrFunc = @(i,j,Beta) exp(-Beta*abs(i-j));
CorrMat = CorrFunc(meshgrid(1:numRates-1)',meshgrid(1:numRates-1),Beta);

ExerciseDate = datenum('11-Aug-2009');
Maturity = daysadd(ExerciseDate,360*[3;4],1);

Vol = blackvolbyrebonato(irdc,VolFunc,CorrMat,ExerciseDate,Maturity,'Period',1)

Vol =

    0.2167
    0.2031
```

References

[1] Brigo, D. and F. Mercurio, *Interest Rate Models - Theory and Practice*, Springer Finance, 2006.

See Also   LiborMarketModel **|**

Related Examples   • "Price Swaptions With Interest-Rate Models Using Simulation" on page 2-91

bondbybk

Purpose   Price bond from Black-Karasinski interest-rate tree

Syntax

```
[Price, PriceTree] = bondbybk(BKTree, CouponRate,
Settle, Maturity)
[Price, PriceTree] = bondbybk(BKTree, CouponRate,
Settle, Maturity, Period, Basis, EndMonthRule, IssueDate,
FirstCouponDate, LastCouponDate, StartDate, Face,
Options)
[Price, PriceTree] = bondbybk(BKTree, CouponRate,
Settle, Maturity, Name,Value)
```

Input Arguments

BKTree	Forward rate tree structure created by `bktree`.
CouponRate	Decimal annual rate. `CouponRate` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than `Maturity`. The `Settle` date for every bond is set to the `ValuationDate` of the BK Tree. The bond argument, `Settle`, is ignored.
Maturity	Maturity date. A vector of serial date numbers or date strings.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Period

Coupons per year of the bond. A vector of integers. Values are 1, 2, 3, 4, 6, and 12.

> **Default:** 2

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Default: 0 (actual/actual)

EndMonthRule

End-of-month rule. A NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Default: 1

IssueDate

Date when a bond was issued.

FirstCouponDateDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure. If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

LastCouponDateDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

Face

Face or par value. `Face` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid.

> **Default:** 100

Options

Derivatives pricing options structure created with `derivset`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** false

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

> **Default:** actual

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, holidays.m is used.

Description　[Price, PriceTree] = bondbybk(BKTree, CouponRate, Settle, Maturity) computes the price of a bond from a Black-Karasinski interest-rate tree.

[Price, PriceTree] = bondbybk(BKTree, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face, Options) computes the price of a bond from a Black-Karasinski interest-rate tree using optional input arguments.

[Price, PriceTree] = bondbybk(BKTree, CouponRate, Settle, Maturity, Name,Value) computes the price of a bond from a Black-Karasinski interest-rate tree with additional options specified by one or more Name,Value pair arguments.

Price is a number of instruments (NINST)-by-1 matrix of expected prices at time 0.

PriceTree is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node. Within PriceTree:

- PriceTree.PTree contains the clean prices.

- `PriceTree.AITree` contains the accrued interest.

- `PriceTree.tObs` contains the observation times.

`bondbybk` computes prices of vanilla bonds, stepped coupon bonds, and amortizing bonds with no market purchase option and no call provisions.

Definitions

Vanilla Bond

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment.

Stepped Coupon Bond

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond.

Bond with an Amortization Schedule

An amortized bond is treated as an asset, with the discount amount being amortized to interest expense over the life of the bond.

Examples

Price a Bond Using a BK Tree

Price a 4% bond using a Black-Karasinski interest-rate tree.

Load `deriv.mat`, which provides `BKTree`. The `BKTree` structure contains the time and interest-rate information needed to price the bond.

```
load deriv.mat;
```

Define the bond using the required arguments. Other arguments use defaults.

```
CouponRate = 0.04;
```

```
Settle = '01-Jan-2004';
Maturity = '31-Dec-2008';
```

Use bondbybk to compute the price of the bond.

```
Price = bondbybk(BKTree, CouponRate, Settle, Maturity)
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

  98.0300
```

Price a Stepped Coupon Bond

Price single stepped coupon bonds using market data.

Define the interest-rate term structure.

```
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012';...
'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;
```

Create the RateSpec.

```
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RS =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: [4x1 double]
           Rates: [4x1 double]
        EndTimes: [4x1 double]
      StartTimes: [4x1 double]
```

```
        EndDates: [4x1 double]
      StartDates: 734139
  ValuationDate: 734139
          Basis: 0
    EndMonthRule: 1
```

Create the stepped bond instrument.

```
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
Period = 1;
```

Build the BK tree using the following market data:

```
VolDates = ['1-Jan-2011'; '1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014'];
VolCurve = 0.01;
AlphaDates = '01-01-2014';
AlphaCurve = 0.1;

BKVolSpec = bkvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RS.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RS, BKTimeSpec);
```

Compute the price of the stepped coupon bonds.

```
PBK= bondbybk(BKT, CouponRate, Settle,Maturity , Period)

PBK =

  100.7246
  100.0945
  101.5900
  102.0820
```

Price a Bond with an Amortization Schedule

Price a bond with an amortization schedule using the `Face` input argument to define the schedule.

Define the interest-rate term structure.

```
Rates = 0.065;
ValuationDate = '1-Jan-2011';
StartDates = ValuationDate;
EndDates=  '1-Jan-2017';
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

           FinObj: 'RateSpec'
      Compounding: 1
             Disc: 0.6853
            Rates: 0.0650
         EndTimes: 6
       StartTimes: 0
         EndDates: 736696
       StartDates: 734504
    ValuationDate: 734504
            Basis: 0
      EndMonthRule: 1
```

Create the bond instrument. The bond has a coupon rate of 7%, a period of one year, and matures on 1-Jan-2017.

```
CouponRate = 0.07;
Settle ='1-Jan-2011';
Maturity = '1-Jan-2017';
Period = 1;
```

```
Face = {{'1-Jan-2015' 100;'1-Jan-2016' 90;'1-Jan-2017' 80}};
```

Build the BK tree with the following market data:

```
VolDates = ['1-Jan-2012'; '1-Jan-2013';...
'1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017'];
VolCurve = 0.01;
AlphaDates = '01-01-2017';
AlphaCurve = 0.1;

BKVolSpec = bkvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RateSpec.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RateSpec, BKTimeSpec);
```

Compute the price of the amortizing bond.

```
Price = bondbybk(BKT, CouponRate, Settle, Maturity, 'Period', Period,.
'Face', Face)

Price =

  102.3155
```

Compare the results with price of a vanilla bond.

```
PriceVanilla = bondbybk(BKT, CouponRate, Settle, Maturity, Period)

PriceVanilla =

  102.4205
```

See Also   bkprice | bktree | cfamounts | hwprice | hwtree | instbond

bondbyhjm

Purpose   Price bond from Heath-Jarrow-Morton interest-rate tree

Syntax   [Price, PriceTree] = bondbyhjm(HJMTree, CouponRate, Settle,
                 Maturity)
                 [Price, PriceTree] = bondbyhjm(HJMTree, CouponRate, Settle,
                 Maturity, Period, Basis, EndMonthRule, IssueDate,
                 FirstCouponDate, LastCouponDate, StartDate, Face,
                 Options)
                 [Price, PriceTree] = bondbyhjm(HJMTree, CouponRate,
                 Settle, Maturity, Name,Value)

**Input
Arguments**

HJMTree	Forward rate tree structure created by hjmtree.
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector or NINST-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.

The Settle date for every bond is set to the ValuationDate of the HJM
tree. The bond argument Settle is ignored.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Period

Coupons per year of the bond. A vector of integers. Values are 1, 2, 3, 4, 6, and 12.

> **Default:** 2

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Default: 0 (actual/actual)

EndMonthRule

End-of-month rule. A NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

  Default: 1

IssueDate

Date when a bond was issued.

FirstCouponDateDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure. If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

LastCouponDateDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

Face

Face or par value. `Face` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a `NumDates`-by-`2` cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid.

> **Default:** 100

Options

Derivatives pricing options structure created with `derivset`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** false

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- actual

- `follow`

- `modifiedfollow`

- `previous`

- `modifiedprevious`

   Default: `actual`

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

   Default: If no dates are specified, `holidays.m` is used.

Description   `[Price, PriceTree] = bondbyhjm(HJMTree, CouponRate, Settle, Maturity)` computes the price of a bond from an HJM forward-rate tree.

`[Price, PriceTree] = bondbyhjm(HJMTree, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face, Options)` computes the price of a bond from an HJM forward-rate tree with optional input arguments.

`[Price, PriceTree] = bondbyhjm(HJMTree, CouponRate, Settle, Maturity, Name,Value)` computes the price of a bond from an HJM forward-rate tree with additional options specified by one or more `Name,Value` pair arguments.

`Price` is a number of instruments (`NINST`)-by-1 matrix of expected prices at time 0.

`PriceTree` is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node. Within `PriceTree`:

- `PriceTree.PBush` contains the clean prices.

- `PriceTree.AIBush` contains the accrued interest.

- `PriceTree.tObs` contains the observation times.

`bondbyhjm` computes prices of vanilla bonds, stepped coupon bonds, and amortizing bonds with no market purchase option and no call provisions.

Definitions

Vanilla Bond

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment.

Stepped Coupon Bond

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond.

Bond with an Amortization Schedule

An amortized bond is treated as an asset, with the discount amount being amortized to interest expense over the life of the bond.

Examples

Price a Bond Using an HJM Tree

Price a 4% bond using an HJM interest-rate tree.

Load `deriv.mat`, which provides `HJMTree`. The `HJMTree` structure contains the time and interest-rate information needed to price the bond.

```
load deriv.mat;
```

Define the bond using the required arguments. Other arguments use defaults.

```
CouponRate = 0.04;
Settle = '01-Jan-2000';
```

```
Maturity = '01-Jan-2004';
```

Use bondbyhjm to compute the price of the bond.

```
Price = bondbyhjm(HJMTree, CouponRate, Settle, Maturity)
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

  97.5280
```

Price a Stepped Coupon Bond

Price single stepped coupon bonds using market data.

Define the interest-rate term structure.

```
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;
```

Create the RateSpec.

```
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

RS =

         FinObj: 'RateSpec'
    Compounding: 1
           Disc: [4x1 double]
          Rates: [4x1 double]
       EndTimes: [4x1 double]
     StartTimes: [4x1 double]
       EndDates: [4x1 double]
     StartDates: 734139
```

```
        ValuationDate: 734139
                Basis: 0
          EndMonthRule: 1
```

Create the stepped bond instrument.

```
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
Period = 1;
```

Build the HJM tree using the following market data:

```
Volatility = [.2; .19; .18; .17];
CurveTerm = [ 1;  2;   3;    4];
HJMTimeSpec = hjmtimespec(ValuationDate, EndDates);
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec,RS,HJMTimeSpec);
```

Compute the price of the stepped coupon bonds.

```
PHJM= bondbyhjm(HJMT, CouponRate, Settle,Maturity , Period)

PHJM =

  100.7246
  100.0945
  101.5900
  102.0820
```

Price a Bond with an Amortization Schedule

Price a bond with an amortization schedule using the Face input argument to define the schedule.

Define the interest-rate term structure.

```
Rates = 0.065;
ValuationDate = '1-Jan-2011';
StartDates = ValuationDate;
```

```
EndDates=  '1-Jan-2017';
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: 0.6853
           Rates: 0.0650
        EndTimes: 6
      StartTimes: 0
        EndDates: 736696
      StartDates: 734504
   ValuationDate: 734504
           Basis: 0
     EndMonthRule: 1
```

Create the bond instrument. The bond has a coupon rate of 7%, a period of one year, and matures on 1-Jan-2017.

```
CouponRate = 0.07;
Settle ='1-Jan-2011';
Maturity = '1-Jan-2017';
Period = 1;
Face = {{'1-Jan-2015' 100;'1-Jan-2016' 90;'1-Jan-2017' 80}};
```

Build the HJM tree using the following market data:

```
Volatility = [.2; .19; .18; .17];
CurveTerm = [ 1;  2;   3;    4];
MaTree = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015';...
'Jan-1-2016'; 'Jan-1-2017'};
HJMTimeSpec = hjmtimespec(ValuationDate, MaTree);
```

```
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec,RateSpec,HJMTimeSpec);
```

Compute the price of the amortizing bond.

```
Price = bondbyhjm(HJMT, CouponRate, Settle, Maturity, 'Period',...
Period, 'Face' , Face)

Price =

  102.3155
```

Compare the results with price of a vanilla bond.

```
PriceVanilla = bondbyhjm(HJMT, CouponRate, Settle, Maturity, Period)

PriceVanilla =

  102.4205
```

See Also   hjmtree | cfamounts | hjmprice | instbond

bondbyhw

Purpose   Price bond from Hull-White interest-rate tree

Syntax
```
[Price, PriceTree] = bondbyhw(HWTree, CouponRate,
Settle, Maturity)
[Price, PriceTree] = bondbyhw(HWTree, CouponRate,
Settle, Maturity, Period, Basis, EndMonthRule, IssueDate,
FirstCouponDate, LastCouponDate, StartDate, Face,
Options)
[Price, PriceTree] = bondbyhw(HWTree, CouponRate,
Settle, Maturity, Name,Value)
```

Input Arguments

HWTree	Forward-rate tree structure created by hwtree.
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector or NINST-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.

The Settle date for every bond is set to the ValuationDate of the HW tree. The bond argument Settle is ignored.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Period

Coupons per year of the bond. A vector of integers. Values are 1, 2, 3, 4, 6, and 12.

> **Default:** 2

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Default: `0` (actual/actual)

EndMonthRule

End-of-month rule. `NINST`-by-`1` vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Default: `1`

IssueDate

Date when a bond was issued.

FirstCouponDateDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

LastCouponDateDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

Face

Face or par value. `Face` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid.

> **Default:** 100

Options

Derivatives pricing options structure created with `derivset`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** false

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- actual

- `follow`

- `modifiedfollow`

- `previous`

- `modifiedprevious`

Default: `actual`

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

Default: If no dates are specified, `holidays.m` is used.

Description   `[Price, PriceTree] = bondbyhw(HWTree, CouponRate, Settle, Maturity)` computes the price of a bond from a Hull-White interest-rate tree.

`[Price, PriceTree] = bondbyhw(HWTree, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face, Options)` computes the price of a bond from a Hull-White interest-rate tree with optional input arguments.

`[Price, PriceTree] = bondbyhw(HWTree, CouponRate, Settle, Maturity, Name,Value)` computes the price of a bond from a Hull-White interest-rate tree with additional options specified by one or more `Name,Value` pair arguments.

`Price` is a number of instruments (`NINST`)-by-1 matrix of expected prices at time 0.

`PriceTree` is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node. Within `PriceTree`:

- `PriceTree.PTree` contains the clean prices.

- `PriceTree.AITree` contains the accrued interest.

- `PriceTree.tObs` contains the observation times.

bondbyhw computes prices of vanilla bonds, stepped coupon bonds, and amortizing bonds with no market purchase option and no call provisions.

Definitions

Vanilla Bond

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment.

Stepped Coupon Bond

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond.

Bond with an Amortization Schedule

An amortized bond is treated as an asset, with the discount amount being amortized to interest expense over the life of the bond.

Examples

Price a Bond Using the HW Tree

Price a 4% bond using a Hull-White interest-rate tree.

Load `deriv.mat`, which provides `HWTree`. The `HWTree` structure contains the time and interest-rate information needed to price the bond.

```
load deriv.mat;
```

Define the bond using the required arguments. Other arguments use defaults.

```
CouponRate = 0.04;
```

```
Settle = '01-Jan-2004';
Maturity = '31-Dec-2008';
```

Use bondbyhw to compute the price of the bond.

```
Price = bondbyhw(HWTree, CouponRate, Settle, Maturity)
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

  98.0483
```

Price a Stepped Coupon Bond

Price single stepped coupon bonds using market data.

Define the interest-rate term structure.

```
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;
```

Create the RateSpec.

```
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RS =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: [4x1 double]
             Rates: [4x1 double]
          EndTimes: [4x1 double]
        StartTimes: [4x1 double]
          EndDates: [4x1 double]
```

```
       StartDates: 734139
    ValuationDate: 734139
            Basis: 0
      EndMonthRule: 1
```

Create the stepped bond instrument.

```
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
Period = 1;
```

Build the HW tree using the following market data:

```
VolDates = ['1-Jan-2011'; '1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014'];
VolCurve = 0.01;
AlphaDates = '01-01-2014';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RS.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RS, HWTimeSpec);
```

Compute the price of the stepped coupon bonds.

```
PHW= bondbyhw(HWT, CouponRate, Settle,Maturity , Period)

PHW =

  100.7246
  100.0945
  101.5900
  102.0820
```

Price Two Bonds with Amortization Schedules

Price two bonds with amortization schedules using the Face input argument to define the schedules.

Define the interest rate term structure.

```
Rates = 0.035;
ValuationDate = '1-Nov-2011';
StartDates = ValuationDate;
EndDates = '1-Nov-2017';
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: 0.8135
             Rates: 0.0350
          EndTimes: 6
        StartTimes: 0
          EndDates: 737000
        StartDates: 734808
     ValuationDate: 734808
             Basis: 0
       EndMonthRule: 1
```

Create the bond instrument. The bonds have a coupon rate of 4% and 3.85%, a period of one year, and mature on 1-Nov-2017.

```
CouponRate = [0.04; 0.0385];
Settle ='1-Nov-2011';
Maturity = '1-Nov-2017';
Period = 1;
```

Define the amortizing schedule.

```
Face = {{'1-Nov-2015' 100;'1-Nov-2016' 85;'1-Nov-2017' 70};
{'1-Nov-2015' 100;'1-Nov-2016' 90;'1-Nov-2017' 80}};
```

Build the HW tree and assume the volatility to be 10%.

```
VolDates = ['1-Nov-2012'; '1-Nov-2013';'1-Nov-2014';'1-Nov-2015';'1-Nov-2016';'1-Nov-2017'];
VolCurve = 0.1;
AlphaDates = '01-01-2017';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec);
```

Compute the price of the amortizing bonds.

```
Price = bondbyhw(HWT, CouponRate, Settle, Maturity, 'Period',Period,...
'Face', Face)

Price =

  102.4791
  101.7786
```

See Also   bkprice | bktree | cfamounts | hwprice | hwtree | instbond

Purpose   Price bond from set of zero curves

Syntax
```
[Price, PriceNoAI, CFlowAmounts, CFlowDates] =
bondbyzero(RateSpec, CouponRate, Settle, Maturity)
[Price, PriceNoAI, CFlowAmounts, CFlowDates] =
bondbyzero(RateSpec, CouponRate, Settle, Maturity,
Period, Basis, EndMonthRule, IssueDate,
FirstCouponDate, LastCouponDate, StartDate, Face)
[Price, PriceNoAI, CFlowAmounts, CFlowDates] =
bondbyzero(RateSpec, CouponRate, Settle,
Maturity, Name, Value)
```

Description `[Price, PriceNoAI, CFlowAmounts, CFlowDates] =`
`bondbyzero(RateSpec, CouponRate, Settle, Maturity)` returns a
`NINST`-by-`NUMCURVES` matrix of clean bond prices. Each column arises
from one of the zero curves.

`[Price, PriceNoAI, CFlowAmounts, CFlowDates] =`
`bondbyzero(RateSpec, CouponRate, Settle, Maturity,`
`Period, Basis, EndMonthRule, IssueDate,`
`FirstCouponDate, LastCouponDate, StartDate, Face)` returns a
`NINST`-by-`NUMCURVES` matrix of clean bond prices. Each column arises
from one of the zero curves.

`[Price, PriceNoAI, CFlowAmounts, CFlowDates] =`
`bondbyzero(RateSpec, CouponRate, Settle,`
`Maturity, Name, Value)` returns a `NINST`-by-`NUMCURVES` matrix of
clean bond prices (each column arises from one of the zero curves)
with additional options specified by one or more `Name, Value` pair
arguments.

`bondbyzero` computes prices of vanilla bonds, stepped coupon bonds,
and amortizing bonds with no market purchase option and no call
provisions.

**Input
Arguments**

RateSpec

Structure containing the properties of an interest-rate structure. See `intenvset` for information on creating `RateSpec`.

CouponRate

Decimal annual rate. `CouponRate` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.

Settle

Settlement date. `Settle` must be either a scalar or `NINST`-by-1 vector of serial date numbers or date strings of the same value which represent the settlement date for each bond. `Settle` must be earlier than `Maturity`.

Maturity

Maturity date. `NINST`-by-1 vector of dates representing the maturity date for each swap.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Period

Coupons per year of the bond. A vector of integers. Values are 1, 2, 3, 4, 6, and 12.

> **Default:** 2

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

EndMonthRule

End-of-month rule. A `NINST`-by-`1` vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.
- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

IssueDate

Date when a bond was issued.

FirstCouponDateDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

LastCouponDateDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

Face

Face or par value. `Face` is a NINST-by-1 vector or NINST-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid.

Default: 100

Options

Derivatives pricing options structure created with `derivset`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** `false`

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- `actual`
- `follow`
- `modifiedfollow`
- `previous`
- `modifiedprevious`

> **Default:** `actual`

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

Default: If no dates are specified, `holidays.m` is used.

Output Arguments

Price

NINST-by-NUMCURVES matrix of clean bond prices. Each column arises from one of the zero curves.

PriceNoAI

NINST-by-NUMCURVES matrix of dirty bond price (clean + accrued interest). Each column arises from one of the zero curves.

CFlowAmounts

NINST-by-NUMCFS matrix of cash flows for each bond

CFlowDates

NUMCFS-by-1 matrix of payment dates for each bond

Definitions

Vanilla Bond

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment.

Stepped Coupon Bond

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond.

Bond with an Amortization Schedule

An amortized bond is treated as an asset, with the discount amount being amortized to interest expense over the life of the bond.

Examples

Price a Vanilla Bond

Price a 4% bond using a zero curve.

Load `deriv.mat`, which provides `ZeroRateSpec`, the interest-rate term structure, needed to price the bond.

```
load deriv.mat;
CouponRate = 0.04;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2004';
Price = bondbyzero(ZeroRateSpec, CouponRate, Settle, Maturity)

Price =

   97.5334
```

Price a Stepped Coupon Bond

Price single stepped coupon bonds using market data.

Define data for the interest-rate term structure.

```
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;
```

Create the `RateSpec`.

```
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RS =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: [4x1 double]
```

```
            Rates: [4x1 double]
         EndTimes: [4x1 double]
       StartTimes: [4x1 double]
         EndDates: [4x1 double]
       StartDates: 734139
    ValuationDate: 734139
            Basis: 0
      EndMonthRule: 1
```

Create the stepped bond instrument.

```
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
Period = 1;
```

Compute the price of the stepped coupon bonds.

```
PZero= bondbyzero(RS, CouponRate, Settle, Maturity ,Period)

PZero =

  100.7246
  100.0945
  101.5900
  102.0820
```

Price a Bond with an Amortizing Schedule

Price a bond with an amortizing schedule using the Face input argument to define the schedule.

Define data for the interest-rate term structure.

```
Rates = 0.065;
ValuationDate = '1-Jan-2011';
StartDates = ValuationDate;
EndDates=  '1-Jan-2017';
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

           FinObj: 'RateSpec'
      Compounding: 1
             Disc: 0.6853
            Rates: 0.0650
         EndTimes: 6
       StartTimes: 0
         EndDates: 736696
       StartDates: 734504
    ValuationDate: 734504
            Basis: 0
      EndMonthRule: 1
```

Create and price the amortizing bond instrument. The bond has a coupon rate of 7%, a period of one year, and matures on 1-Jan-2017.

```
CouponRate = 0.07;
Settle ='1-Jan-2011';
Maturity = '1-Jan-2017';
Period = 1;
Face = {{'1-Jan-2015' 100;'1-Jan-2016' 90;'1-Jan-2017' 80}};
Price = bondbyzero(RateSpec, CouponRate, Settle, Maturity, 'Period',...
Period, 'Face', Face)

Price =

  102.3155
```

Compare the results with price of a vanilla bond.

```
PriceVanilla = bondbyzero(RateSpec, CouponRate, Settle, Maturity,Period)
```

```
PriceVanilla =

  102.4205
```

See Also | swapbyzero | cfamounts | cfbyzero | fixedbyzero | floatbyzero

bushpath

Purpose	Extract entries from node of bushy tree
Syntax	Values = bushpath(Tree, BranchList)

Arguments

Tree	Bushy tree.
BranchList	Number of paths (NUMPATHS) by path length (PATHLENGTH) matrix containing the sequence of branchings.

Description   Values = bushpath(Tree, BranchList) extracts entries of a node of a bushy tree. The node path is described by the sequence of branchings taken, starting at the root. The top branch is number 1, the second-to-top is 2, and so on. Set the branch sequence to zero to obtain the entries at the root node.

Values is a number of values (NUMVALS)-by-NUMPATHS matrix containing the retrieved entries of a bushy tree.

Examples   Create an HJM tree by loading the example file.

load deriv.mat;

Then

FwdRates = bushpath(HJMTree.FwdTree, [1 2 1])

returns the rates at the tree nodes located by taking the up branch, then the down branch, and finally the up branch again.

FwdRates =

    1.0356
    1.0364
    1.0526

```
1.0463
```

You can visualize this with the `treeviewer` function.

```
treeviewer(HJMTree)
```


See Also   bushshape | mkbush

bushshape

Purpose	Retrieve shape of bushy tree
Syntax	[NumLevels, NumChild, NumPos, NumStates, Trim] = bushshape(Tree)

Arguments

Tree	Bushy tree.

Description   [NumLevels, NumChild, NumPos, NumStates, Trim] = bushshape(Tree) returns information on a bushy tree's shape.

NumLevels is the number of time levels of the tree.

NumChild is a 1-by-number of levels (NUMLEVELS) vector with the number of branches (children) of the nodes in each level.

NumPos is a 1-by-NUMLEVELS vector containing the length of the state vectors in each level.

NumStates is a 1-by-NUMLEVELS vector containing the number of state vectors in each level.

Trim is 1 if NumPos decreases by 1 when moving from one time level to the next. Otherwise, it is 0.

Examples   Create an HJM tree by loading the example file.

load deriv.mat;

With treeviewer you can see the general shape of the HJM interest-rate tree.

With this tree

```
[NumLevels, NumChild, NumPos, NumStates, Trim] =...
bushshape(HJMTree.FwdTree)
```

returns

```
NumLevels =
     4
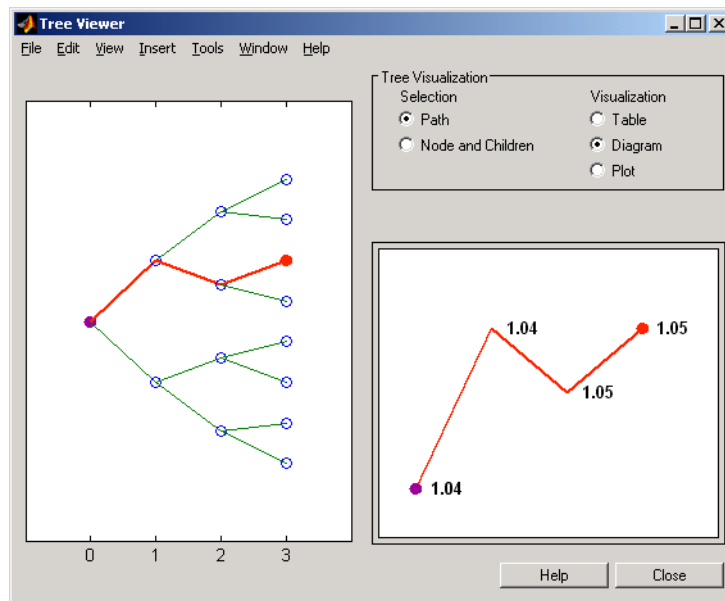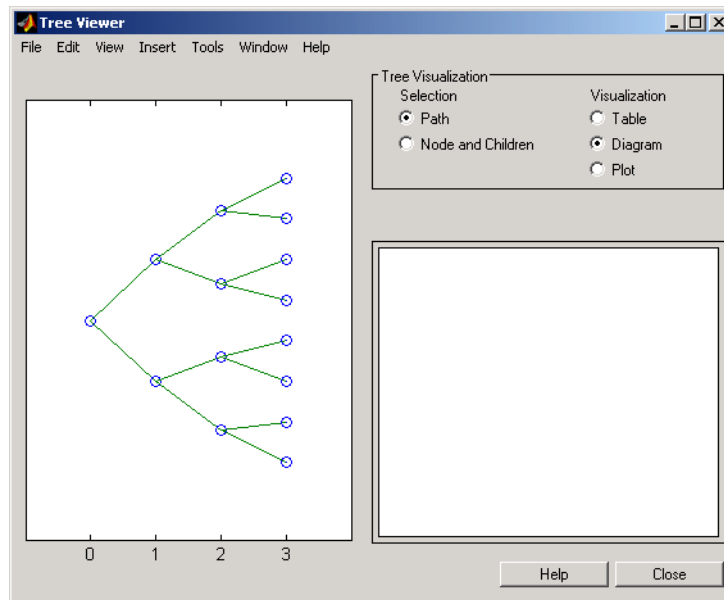
NumChild =
     2     2     2     0

NumPos =
     4     3     2     1

NumStates =
     1     2     4     8
```

```
Trim =
     1
```

You can recreate this tree using the mkbush function.

```
Tree = mkbush(NumLevels, NumChild(1), NumPos(1), Trim);
Tree = mkbush(NumLevels, NumChild, NumPos);
```

See Also   bushpath | mkbush

Purpose	Price cap instrument from Black-Derman-Toy interest-rate tree
Syntax	[Price, PriceTree] = capbybdt(BDTTree, Strike, Settle, Maturity, Reset, Basis, Principal, Options)

Arguments

BDTTree	Interest-rate tree structure created by bdttree.
Strike	Number of instruments (NINST)-by-1 vector of rates at which the cap is exercised.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the cap.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the cap.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

capbybdt

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal	(Optional) The notional principal amount. Default = 100.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Price, PriceTree] = capbybdt(BDTTree, Strike, Settle, Maturity, Reset, Basis, Principal, Options) computes the price of a cap instrument from a BDT interest-rate tree.

Price is the expected price of the cap at time 0.

PriceTree is the tree structure with values of the cap at each node.

The Settle date for every cap is set to the ValuationDate of the BDT tree. The cap argument Settle is ignored.

Examples   **Example 1**. Price a 3% cap instrument using a BDT interest-rate tree.

Load the file deriv.mat, which provides BDTTree. The BDTTree structure contains the time and interest-rate information needed to price the cap instrument.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
Strike = 0.03;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2004';
```

Use capbybdt to compute the price of the cap instrument.

```
Price = capbybdt(BDTTree, Strike, Settle, Maturity)

Price =

  28.4001
```

Example 2. This example shows the pricing of a 10% cap instrument using a newly created BDT tree.

First set the required arguments for the three needed specifications.

```
Compounding = 1;
ValuationDate = '01-01-2000';
StartDate = ValuationDate;
EndDates = ['01-01-2001'; '01-01-2002'; '01-01-2003';
'01-01-2004'; '01-01-2005'];
Rates = [.1; .11; .12; .125; .13];
Volatility = [.2; .19; .18; .17; .16];
```

Next create the specifications.

```
RateSpec = intenvset('Compounding', Compounding,...
'ValuationDate', ValuationDate,...
'StartDates', StartDate,...
'EndDates', EndDates,...
'Rates', Rates);
BDTTimeSpec = bdttimespec(ValuationDate, EndDates, Compounding);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Volatility);
```

Now create the BDT tree from the specifications.

```
BDTTree = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);
```

Set the cap arguments. Remaining arguments will use defaults.

```
CapStrike = 0.10;
Settlement = ValuationDate;
Maturity = '01-01-2002';
CapReset = 1;
```

Use `capbybdt` to find the price of the cap instrument.

```
Price= capbybdt(BDTTree, CapStrike, Settlement, Maturity,...
CapReset)

Price =

    1.7169
```

See Also   `bdttree` | `cfbybdt` | `floorbybdt` | `swapbybdt`

capbybk

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

| Principal | (Optional) The notional principal amount. Default = 100. |
| Options | (Optional) Derivatives pricing options structure created with derivset. |

Description   [Price, PriceTree] = capbybk(BKTree, Strike, Settle, Maturity,
Reset, Basis, Principal, Options) computes the price of a cap
instrument from a Black-Karasinski interest-rate tree.

Price is the expected price of the cap at time 0.

PriceTree is the tree structure with values of the cap at each node.

The Settle date for every cap is set to the ValuationDate of the BK
tree. The cap argument Settle is ignored.

Examples   Price a 3% cap instrument using a Black-Karasinski interest-rate tree.

Load the file deriv.mat, which provides BKTree. The BKTree structure
contains the time and interest-rate information needed to price the
cap instrument.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
Strike = 0.03;
Settle = '01-Jan-2005';
Maturity = '01-Jan-2009';
```

Use capbybk to compute the price of the cap instrument.

```
Price = capbybk(BKTree, Strike, Settle, Maturity)

Price =

  6.8337
```

See Also   cfbybk | floorbybk | bktree | swapbybk

capbyblk

Purpose	Price caps using Black option pricing model
Syntax	[CapPrice, Caplets] = capbyblk(RateSpec, Strike, Settle, Maturity, Volatility) [CapPrice, Caplets] = capbyblk(RateSpec, Strike, Settle, Maturity, Volatility, 'Name1', Value1...)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For more information, see intenvset.
Strike	NINST-by-1 vector of rates at which the cap is exercised, as a decimal number.
Settle	Scalar representing the settle date of the cap.
Maturity	Scalar representing the maturity date of the cap.
Volatility	NINST-by-1 vector of volatilities.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default is 1.
Principal	(Optional) NINST-by-1 vector representing the notional principal amount. Default is 100.
Basis	NINST-by-1 vector representing the basis used when annualizing the input forward rate.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

ValuationDate (Optional) Scalar representing the observation date of the investment horizons. The default is the Settle date.

Note All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

Description   [CapPrice, Caplets] = capbyblk(RateSpec, Strike, Settle, Maturity, Volatility)

[CapPrice, Caplets] = capbyblk(RateSpec, Strike, Settle, Maturity, Volatility, 'Name1', Value1...)

Use capbyblk to price caps using the Black option pricing model.

The outputs are:

- CapPrice — NINST-by-1 expected prices of the cap.

- Caplets — NINST-by-NCF array of caplets, padded with NaNs.

capbyblk

Examples   Consider an investor who gets into a contract that caps the interest rate
on a $100,000 loan at 8% quarterly compounded for 3 months, starting
on January 1, 2009. Assuming that on January 1, 2008 the zero rate
is 6.9394% continuously compounded and the volatility is 20%, use
this data to compute the cap price.

Calculate the `RateSpec`:

```
ValuationDate = 'Jan-01-2008';
EndDates ='April-01-2010';
Rates = 0.069394;
Compounding = -1;
Basis = 1;

RateSpec = intenvset('ValuationDate', ValuationDate, ...
'StartDates', ValuationDate,'EndDates', EndDates, ...
'Rates', Rates,'Compounding', Compounding,'Basis', Basis);
```

Compute the price of the cap:

```
Settle = 'Jan-01-2009'; % cap starts in a year
Maturity = 'April-01-2009';
Volatility = 0.20;
CapRate = 0.08;
CapReset = 4;
Principal=100000;

CapPrice = capbyblk(RateSpec,  CapRate, Settle, Maturity, Volatility,...
'Reset',CapReset,'ValuationDate',ValuationDate,'Principal', Principal,...
'Basis', Basis)

CapPrice =

   51.6125
```

See Also   `floorbyblk`

capbyhjm

Purpose	Price cap instrument from Heath-Jarrow-Morton interest-rate tree

Syntax

```
[Price, PriceTree] = capbyhjm(HJMTree, Strike,
Settle, Maturity, Reset, Basis, Principal, Options)
```

Arguments

HJMTree	Forward-rate tree structure created by hjmtree.
Strike	Number of instruments (NINST)-by-1 vector of rates at which the cap is exercised.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the cap.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the cap.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal   (Optional) The notional principal amount. Default = 100.

Options   (Optional) Derivatives pricing options structure created with derivset.

Description   [Price, PriceTree] = capbyhjm(HJMTree, Strike, Settle, Maturity, Reset, Basis, Principal, Options) computes the price of a cap instrument from an HJM tree.

Price is the expected price of the cap at time 0.

PriceTree is the tree structure with values of the cap at each node.

The Settle date for every cap is set to the ValuationDate of the HJM tree. The cap argument Settle is ignored.

Examples   Price a 3% cap instrument using an HJM forward-rate tree.

Load the file deriv.mat, which provides HJMTree. The HJMTree structure contains the time and forward-rate information needed to price the cap instrument.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
Strike = 0.03;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2004';
```

Use capbyhjm to compute the price of the cap instrument.

```
Price = capbyhjm(HJMTree, Strike, Settle, Maturity)

Price =

  6.2831
```

See Also   cfbyhjm | floorbyhjm | hjmtree | swapbyhjm

capbyhw

Purpose   Price cap instrument from Hull-White interest-rate tree

Syntax   [Price, PriceTree] = capbyhw(HWTree, Strike, Settle, Maturity, Reset, Basis, Principal, Options)

Arguments

HWTree	Interest-rate tree structure created by hwtree.
Strike	Number of instruments (NINST)-by-1 vector of rates at which the cap is exercised.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the cap.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the cap.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

   For more information, see **basis** on page Glossary-1.

| Principal | (Optional) The notional principal amount. Default = 100. |
| Options | (Optional) Derivatives pricing options structure created with derivset. |

Description [Price, PriceTree] = capbyhw(HWTree, Strike, Settle, Maturity, Reset, Basis, Principal, Options) computes the price of a cap instrument from a Hull-White interest-rate tree.

Price is the expected price of the cap at time 0.

PriceTree is the tree structure with values of the cap at each node.

The Settle date for every cap is set to the ValuationDate of the HW tree. The cap argument Settle is ignored.

Examples Price a 3% cap instrument using a Hull-White interest-rate tree.

Load the file deriv.mat, which provides HWTree. The HWTree structure contains the time and interest-rate information needed to price the cap instrument.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
Strike = 0.03;
Settle = '01-Jan-2005';
Maturity = '01-Jan-2009';
```

capbyhw

Use capbyhw to compute the price of the cap instrument.

```
Price = capbyhw(HWTree, Strike, Settle, Maturity)

Price =

  7.0707
```

See Also   cfbyhw | floorbyhw | hwtree | swapbyhw

Purpose	Price cap using Linear Gaussian two-factor model
Syntax	CapPrice = capbylg2f(ZeroCurve,a,b,sigma,eta,rho,Strike, Maturity) CapPrice = capbylg2f(___ , Name,Value)
Description	CapPrice = capbylg2f(ZeroCurve,a,b,sigma,eta,rho,Strike, Maturity) returns cap price for a two-factor additive Gaussian interest-rate model. CapPrice = capbylg2f(___ , Name,Value) returns cap price for a two-factor additive Gaussian interest-rate model using optional name-value pairs.
Input Arguments	**ZeroCurve - Zero-curve for Linear Gaussian two-factor model** structure Zero-curve for the Linear Gaussian two-factor model, specified using IRDataCurve or RateSpec. **Data Types** struct **a - Mean reversion for first factor for Linear Gaussian two-factor model** scalar Mean reversion for first factor for the Linear Gaussian two-factor model, specified as a scalar. **Data Types** single | double **b - Mean reversion for second factor for Linear Gaussian two-factor model** scalar

Mean reversion for second factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
single | double

sigma - Volatility for first factor for Linear Gaussian two-factor model
scalar

Volatility for first factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
single | double

eta - Volatility for second factor for Linear Gaussian two-factor model
scalar

Volatility for second factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
single | double

rho - Scalar correlation of the factors
scalar

Scalar correlation of the factors, specified as a scalar.

Data Types
single | double

Strike - Cap strike price
nonnegative integer | vector of nonnegative integers

Cap strike price, specified as a nonnegative integer using a NumCaps-by-1 vector.

Data Types
single | double

Maturity - Cap maturity date
nonnegative integer | vector of nonnegative integers | string of dates

Cap maturity date, specified using a `NumCaps`-by-1 vector of serial date numbers or date strings.

Data Types
single | double | cell

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: Price = capbylg2f(irdc,a,b,sigma,eta,rho,Strike,CapMaturity,'Reset',1,'Notiona

'Reset' - Frequency of cap payments per year
2 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of cap payments per year, specified as positive integers for the values 1,2,4,6,12] in a `NumCaps`-by-1 vector.

Data Types
single | double

'Notional' - Notional value of cap
100 (default) | nonnegative integer | vector of nonnegative integers

Notional value of cap, specified as a nonnegative integer using a `NumCaps`-by-1 vector of notional amounts.

Data Types
single | double

capbylg2f

Output Arguments	**CapPrice - Cap price** scalar | vector Expected prices of cap, returned as a scalar or an NumCaps-by-1 vector.
Algorithms	The following defines the two-factor additive Gaussian interest rate model, given the ZeroCurve, a, b, sigma, eta, and rho parameters:

$$r(t) = x(t) + y(t) + \phi(t)$$

$$dx(t) = -a(x)(t)dt + \sigma(dW_1(t), x(0) = 0$$

$$dy(t) = -b(y)(t)dt + \eta(dW_2(t), y(0) = 0$$

where $dW_1(t)dW_2(t) = \rho dt$ is a two-dimensional Brownian motion with correlation ρ and φ is a function chosen to match the initial zero curve.

Examples | **Price Cap Using a Linear Gaussian Two-Factor Model**

Define the ZeroCurve, a, b, sigma, eta, and rho parameters to price the cap.

```
Settle = datenum('15-Dec-2007');

ZeroTimes = [3/12 6/12 1 5 7 10 20 30]';
ZeroRates = [0.033 0.034 0.035 0.040 0.042 0.044 0.048 0.0475]';
CurveDates = daysadd(Settle,360*ZeroTimes);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

a = .07;
b = .5;
sigma = .01;
eta = .006;
rho = -.7;

CapMaturity = daysadd(Settle,360*[1:5 7 10 15 20 25 30],1);
```

```
Strike = [0.035 0.037 0.038 0.039 0.040 0.042 0.044 0.046 0.047 0.047

Price = capbylg2f(irdc,a,b,sigma,eta,rho,Strike,CapMaturity)

Price =

    0.0682
    0.3565
    0.8263
    1.3611
    1.9262
    3.1116
    4.8228
    7.5504
    9.8862
   11.7870
   12.8822
```

References

[1] Brigo, D. and F. Mercurio, *Interest Rate Models - Theory and Practice*, Springer Finance, 2006.

See Also floorbylg2f | swaptionbylg2fLinearGaussian2F |

Related Examples
• "Price Swaptions With Interest-Rate Models Using Simulation" on page 2-91

cashbybls

Purpose	Determine price of cash-or-nothing digital options using Black-Scholes model
Syntax	Price = cashbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, Payoff)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
Payoff	NINST-by-1 vector of payoff values or the amount to be paid at expiration.

Description Price = cashbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, Payoff) computes cash-or-nothing option prices using the Black-Scholes option pricing model.

Price is a NINST-by-1 vector of expected option prices.

Examples Consider a European call and put cash-or-nothing options on a futures contract with and exercise strike price of $90, a fixed payoff of $10 that expires on October 1, 2008. Assume that on January 1, 2008, the contract trades at $110, and has a volatility of 25% per annum and the risk-free rate is 4.5% per annum. Using this data, calculate the price of the call and put cash-or-nothing options on the futures contract.

Create the RateSpec:

```
Settle = 'Jan-1-2008';
Maturity = 'Oct-1-2008';
Rates = 0.045;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', Basis);
```

Define the StockSpec:

```
AssetPrice = 110;
Sigma = .25;
DivType = 'Continuous';
DivAmount = Rates;
StockSpec = stockspec(Sigma, AssetPrice, DivType, DivAmount);
```

Define the call and put options:

```
OptSpec = {'call'; 'put'};
Strike = 90;
Payoff = 10;
```

Calculate the price:

```
Pcon = cashbybls(RateSpec, StockSpec, Settle,...
Maturity, OptSpec, Strike, Payoff)

Pcon =

    7.6716
    1.9965
```

See Also   assetbybls | cashsensbybls | gapbybls | supersharebybls

cashsensbybls

Purpose Determine price and sensitivities of cash-or-nothing digital options using Black-Scholes model

Syntax
```
PriceSens = cashsensbybls(RateSpec, StockSpec, Settle,
Maturity, OptSpec, Strike, Payoff)
PriceSens = cashsensbybls(RateSpec, StockSpec, Settle,
Maturity, OptSpec, Strike, Payoff, OutSpec)
```

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see `intenvset`.
StockSpec	Stock specification. See `stockspec`.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings `'call'` or `'put'`.
Strike	NINST-by-1 vector of strike price values.
Payoff	NINST-by-1 vector of payoff values or the amount to be paid at expiration.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are: • NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, or `'All'`.

For example, `OutSpec = {'Price'; 'Lambda'; 'Rho'}` specifies that the output should be `Price`, `Lambda`, and `Rho`, in that order.

To invoke from a function: `[Price, Lambda, Rho] = cashsensbybls(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})`

`OutSpec = {'All'}` specifies that the output should be `Delta`, `Gamma`, `Vega`, `Lambda`, `Rho`, `Theta`, and `Price`, in that order. This is the same as specifying `OutSpec` as `OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};`.

- Default is `OutSpec = {'Price'}`.

Description `PriceSens = cashsensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, Payoff)` computes cash-or-nothing option prices using the Black-Scholes option pricing model.

`PriceSens = cashsensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, Payoff, OutSpec)` includes an `OutSpec` argument defined as parameter/value pairs, and computes cash-or-nothing option prices and sensitivities using the Black-Scholes option pricing model.

`PriceSens` is a `NINST`-by-`1` vector of expected option prices and sensitivities.

Examples Consider a European call and put cash-or-nothing options on a futures contract with an exercise price of $90, and a fixed payoff of $10 that expires on January 1, 2009. Assume that on October 1, 2008 the contract trades at $110, and has a volatility of 25% per annum and the risk-free rate is 4.5% per annum. Using this data, calculate the

price and sensitivity of the call and put cash-or-nothing options on the futures contract.

Create the `RateSpec`:

```
Settle = 'Jan-1-2008';
Maturity = 'Oct-1-2008';
Rates = 0.045;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', Basis);
```

Define the `StockSpec`:

```
AssetPrice = 110;
Sigma = .25;
DivType = 'Continuous';
DivAmount = Rates;
StockSpec = stockspec(Sigma, AssetPrice, DivType, DivAmount);
```

Define the call and put options:

```
OptSpec = {'call'; 'put'};
Strike = 90;
Payoff = 10;
```

Compute the gamma, theta, and price:

```
OutSpec = { 'gamma';'theta';'price'};
[Gamma, Theta, Price] = cashsensbybls(RateSpec, StockSpec,...
Settle, Maturity, OptSpec, Strike, Payoff, 'OutSpec', OutSpec)

Gamma =

   -0.0050
    0.0050
```

```
Theta =

   -2.2489
    1.8139


Price =

    7.6716
    1.9965
```

See Also cashbybls

cfbybdt

Purpose	Price cash flows from Black-Derman-Toy interest-rate tree

Syntax

```
[Price, PriceTree] = cfbybdt(BDTTree,
CFlowAmounts, CFlowDates, Settle, Basis, Options)
```

Arguments

BDTTree	Forward-rate tree structure created by `bdttree`.
CFlowAmounts	Number of instruments (NINST) by maximum number of cash flows (MOSTCFS) matrix of cash flow amounts. Each row is a list of cash flow values for one instrument. If an instrument has fewer than MOSTCFS cash flows, the end of the row is padded with NaNs.
CFlowDates	NINST-by-MOSTCFS matrix of cash flow dates. Each entry contains the serial date number of the corresponding cash flow in CFlowAmounts.
Settle	Settlement date. A vector of serial date numbers or date strings. The Settle date for every cash flow is set to the ValuationDate of the BDT tree. The cash flow argument, Settle, is ignored.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Options   (Optional) Derivatives pricing options structure created with `derivset`.

Description   `[Price, PriceTree] = cfbybdt(BDTTree, CFlowAmounts, CFlowDates, Settle, Basis, Options)` prices cash flows from a BDT interest-rate tree.

`Price` is an `NINST`-by-`1` vector of expected prices at time 0.

`PriceTree` is a tree structure with a vector of instrument prices at each node.

Examples   Price a portfolio containing two cash flow instruments paying interest annually over the four-year period from January 1, 2000 to January 1, 2004.

Load the file `deriv.mat`, which provides `BDTTree`. The `BDTTree` structure contains the time and interest-rate information needed to price the instruments.

```
load deriv.mat;
```

The valuation date (settle date) specified in `BDTTree` is January 1, 2000 (date number `730486`).

```
BDTTree.RateSpec.ValuationDate

ans =

      730486
```

Provide values for the other required arguments.

```
CFlowAmounts =[5 NaN 5.5 105; 5 0 6 105];
CFlowDates = [730852, NaN, 731582, 731947;
               730852, 731217, 731582, 731947];
```

Use this information to compute the prices of the two cash flow instruments.

```
[Price, PriceTree] = cfbybdt(BDTTree, CFlowAmounts, ...
CFlowDates, BDTTree.RateSpec.ValuationDate)

Price =

   74.0112
   74.3671

PriceTree =

    FinObj: 'BDTPriceTree'
      tObs: [0 1.00 2.00 3.00 4.00]
     PTree: {1x5 cell}
```

You can visualize the prices of the two cash flow instruments with the treeviewer function.

```
treeviewer(PriceTree)
```

See Also   bdttree | bdtprice | cfamounts | instcf

cfbybk

Purpose	Price cash flows from Black-Karasinski interest-rate tree
Syntax	[Price, PriceTree] = cfbybk(BKTree, CFlowAmounts, CFlowDates, Settle, Basis, Options)

Arguments

BKTree	Forward-rate tree structure created by bktree.
CFlowAmounts	Number of instruments (NINST) by maximum number of cash flows (MOSTCFS) matrix of cash flow amounts. Each row is a list of cash flow values for one instrument. If an instrument has fewer than MOSTCFS cash flows, the end of the row is padded with NaNs.
CFlowDates	NINST-by-MOSTCFS matrix of cash flow dates. Each entry contains the serial date number of the corresponding cash flow in CFlowAmounts.
Settle	Settlement date. A vector of serial date numbers or date strings. The Settle date for every cash flow is set to the ValuationDate of the BK tree. The cash flow argument, Settle, is ignored.
Basis	(Optional) Day-count basis of the instrument. A vector of integers. • 0 = actual/actual (default) • 1 = 30/360 (SIA) • 2 = actual/360 • 3 = actual/365 • 4 = 30/360 (BMA) • 5 = 30/360 (ISDA) • 6 = 30/360 (European) • 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Options   (Optional) Derivatives pricing options structure created with `derivset`.

Description   `[Price, PriceTree] = cfbybk(BKTree, CFlowAmounts, CFlowDates, Settle, Basis, Options)` prices cash flows from a Black-Karasinski interest-rate tree.

`Price` is an `NINST`-by-1 vector of expected prices at time 0.

`PriceTree` is a tree structure with a vector of instrument prices at each node.

Examples   Price a portfolio containing two cash flow instruments paying interest annually over the four-year period from January 1, 2005 to January 1, 2009.

Load the file `deriv.mat`, which provides `BKTree`. The `BKTree` structure contains the time and interest-rate information needed to price the instruments.

```
load deriv.mat;
```

The valuation date (settle date) specified in `BKTree` is January 1, 2004 (date number `731947`).

```
BKTree.RateSpec.ValuationDate

ans =

     731947
```

Provide values for the other required arguments.

```
CFlowAmounts =[5 NaN 5.5 105; 5 0 6 105];
CFlowDates = [732678, NaN, 733408,733774;
              732678, 733034, 733408, 734774];
```

Use this information to compute the prices of the two cash flow instruments.

```
[Price, PriceTree] = cfbybk(BKTree, CFlowAmounts, CFlowDates,...
BKTree.RateSpec.ValuationDate)

Price =

   93.3600
   81.6218

PriceTree =

    FinObj: 'BKPriceTree'
      tObs: [0 1 2 3 4]
     PTree: {[2x1 double]  [2x3 double]  [2x5 double]  [2x5
double]             [2x5 double]}
   Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
     Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
```

You can visualize the prices of the two cash flow instruments with the treeviewer function.

```
treeviewer(PriceTree)
```

See Also   bktree | bkprice | cfamounts | instcf

cfbyhjm

Purpose	Price cash flows from Heath-Jarrow-Morton interest-rate tree
Syntax	[Price, PriceTree] = cfbyhjm(HJMTree, CFlowAmounts, CFlowDates, Settle, Basis, Options)

Arguments

HJMTree	Forward-rate tree structure created by hjmtree.
CFlowAmounts	Number of instruments (NINST) by maximum number of cash flows (MOSTCFS) matrix of cash flow amounts. Each row is a list of cash flow values for one instrument. If an instrument has fewer than MOSTCFS cash flows, the end of the row is padded with NaNs.
CFlowDates	NINST-by-MOSTCFS matrix of cash flow dates. Each entry contains the serial date number of the corresponding cash flow in CFlowAmounts.
Settle	Settlement date. A vector of serial date numbers or date strings. The Settle date for every cash flow is set to the ValuationDate of the HJM tree. The cash flow argument, Settle, is ignored.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

...

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Options   (Optional) Derivatives pricing options structure created with derivset.

Description   [Price, PriceTree] = cfbyhjm(HJMTree, CFlowAmounts, CFlowDates, Settle, Basis, Options) prices cash flows from an HJM interest-rate tree.

Price is an NINST-by-1 vector of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples   Price a portfolio containing two cash flow instruments paying interest annually over the four-year period from January 1, 2000 to January 1, 2004.

Load the file deriv.mat, which provides HJMTree. The HJMTree structure contains the time and forward-rate information needed to price the instruments.

load deriv.mat;

The valuation date (settle date) specified in HJMTree is January 1, 2000 (date number 730486).

```
HWTree.RateSpec.ValuationDate

ans =

     730486
```

Provide values for the other required arguments.

```
CFlowAmounts =[5 NaN 5.5 105; 5 0 6 105];
CFlowDates = [730852, NaN, 731582, 731947;
              730852, 731217, 731582, 731947];
```

Use this information to compute the prices of the two cash flow instruments.

```
[Price, PriceTree] = cfbyhjm(HJMTree, CFlowAmounts,...
CFlowDates, HJMTree.RateSpec.ValuationDate)

Price =

   96.7805
   97.2188

PriceTree =

    FinObj: 'HJMPriceTree'
      tObs: [0 1.00 2.00 3.00 4.00]
     PBush: {1x5 cell}
```

You can visualize the prices of the two cash flow instruments with the `treeviewer` function.

```
treeviewer(PriceTree)
```

See Also   cfamounts | hjmprice | hjmtree | instcf

cfbyhw

Purpose	Price cash flows from Hull-White interest-rate tree
Syntax	[Price, PriceTree] = cfbyhw(HWTree, CFlowAmounts, CFlowDates, Settle, Basis, Options)

Arguments

HWTree	Forward-rate tree structure created by hwtree.
CFlowAmounts	Number of instruments (NINST) by maximum number of cash flows (MOSTCFS) matrix of cash flow amounts. Each row is a list of cash flow values for one instrument. If an instrument has fewer than MOSTCFS cash flows, the end of the row is padded with NaNs.
CFlowDates	NINST-by-MOSTCFS matrix of cash flow dates. Each entry contains the serial date number of the corresponding cash flow in CFlowAmounts.
Settle	Settlement date. A vector of serial date numbers or date strings. The Settle date for every cash flow is set to the ValuationDate of the HW tree. The cash flow argument, Settle, is ignored.
Basis	(Optional) Day-count basis of the instrument. A vector of integers. • 0 = actual/actual (default) • 1 = 30/360 (SIA) • 2 = actual/360 • 3 = actual/365 • 4 = 30/360 (BMA) • 5 = 30/360 (ISDA) • 6 = 30/360 (European) • 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page
  Glossary-1.

Options   (Optional) Derivatives pricing options structure
                   created with `derivset`.

Description   `[Price, PriceTree] = cfbyhw(HWTree, CFlowAmounts,`
                   `CFlowDates,`
                   `Settle, Basis, Options)` prices cash flows from a Hull-White
                   interest-rate tree.

                   `Price` is an `NINST`-by-1 vector of expected prices at time 0.

                   `PriceTree` is a tree structure with a vector of instrument prices at
                   each node.

Examples   Price a portfolio containing two cash flow instruments paying interest
                   annually over the four-year period from January 1, 2005 to January
                   1, 2009.

                   Load the file `deriv.mat`, which provides `HWTree`. The `HWTree` structure
                   contains the time and interest-rate information needed to price the
                   instruments.

                   `load deriv.mat;`

                   The valuation date (settle date) specified in `HWTree` is January 1, 2004
                   (date number `731947`).

```
HWTree.RateSpec.ValuationDate

ans =

    731947
```

Provide values for the other required arguments.

```
CFlowAmounts =[5 NaN 5.5 105; 5 0 6 105];
CFlowDates = [732678, NaN, 733408, 733774;
              732678, 733034, 733408, 734774];
```

Use this information to compute the prices of the two cash flow instruments.

```
[Price, PriceTree] = cfbyhw(HWTree, CFlowAmounts, CFlowDates,...
HWTree.RateSpec.ValuationDate)

Price =

   93.3789
   81.7651

PriceTree =

    FinObj: 'HWPriceTree'
      tObs: [0 1 2 3 4]
     PTree: {[2x1 double]  [2x3 double]  [2x5 double]  [2x5
double]               [2x5 double]}
   Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
     Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
```

You can visualize the prices of the two cash flow instruments with the treeviewer function.

```
treeviewer(PriceTree)
```

See Also   cfamounts | hwtree | hwprice | instcf

cfbyzero

Purpose	Price cash flows from set of zero curves
Syntax	`Price = cfbyzero(RateSpec, CFlowAmounts, CFlowDates, Settle, Basis)`

Arguments

RateSpec	Structure containing the properties of an interest-rate structure. See `intenvset` for information on creating `RateSpec`.
CFlowAmounts	Number of instruments (`NINST`) by maximum number of cash flows (`MOSTCFS`) matrix with entries listing cash flow amounts corresponding to each date in `CFlowDates`. Each row is a list of cash flow values for one instrument. If an instrument has fewer than `MOSTCFS` cash flows, the end of the row is padded with `NaN`s.
CFlowDates	`NINST`-by-`MOSTCFS` matrix of cash flow dates. Each entry contains the serial date of the corresponding cash flow in `CFlowAmounts`.
Settle	Settlement date on which the cash flows are priced. `Settle` must be either a scalar or `NINST`-by-1 vector of serial date numbers or date strings of the same value which represent the settlement date for each cash flow. `Settle` must be earlier than `Maturity`.
Basis	(Optional) Day-count basis of the instrument. A vector of integers. • 0 = actual/actual (default) • 1 = 30/360 (SIA) • 2 = actual/360 • 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

    For more information, see **basis** on page Glossary-1.

Description   `Price = cfbyzero(RateSpec, CFlowAmounts, CFlowDates, Settle,`
`Basis)` computes `Price`, an `NINST`-by-`NUMCURVES` matrix of cash flows prices. Each column arises from one of the zero curves.

Examples   Price a portfolio containing two cash flow instruments paying interest annually over the four-year period from January 1, 2000 to January 1, 2004.

Load the file `deriv.mat`, which provides `ZeroRateSpec`. The `ZeroRateSpec` structure contains the interest-rate information needed to price the instruments.

```
load deriv.mat
CFlowAmounts =[5 NaN 5.5 105;5 0 6 105];
CFlowDates = [730852, NaN, 731582,731947;
              730852, 731217, 731582, 731947];
Settle = 730486;
```

```
Price = cfbyzero(ZeroRateSpec, CFlowAmounts, CFlowDates, Settle)

Price =

   96.7804
   97.2187
```

See Also   bondbyzero | fixedbyzero | floatbyzero | swapbyzero

Purpose	Price European simple chooser options using Black-Scholes model

Syntax   Price = chooserbybls(RateSpec, StockSpec, Settle, Maturity, Strike)

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
Strike	NINST-by-1 vector of strike price values.
ChooseDate	NINST-by-1 vector of chooser dates.

Description   Price = chooserbybls(RateSpec, StockSpec, Settle, Maturity, Strike) computes the price for European simple chooser options using the Black-Scholes model.

Price is a NINST-by-1 vector of expected prices.

Note Only dividends of type continuous can be considered for choosers.

Examples   Consider a European chooser option with an exercise price of $60 on June 1, 2007. The option expires on December 2, 2007. Assume the underlying stock provides a continuous dividend yield of 5% per annum, is trading at $50, and has a volatility of 20% per annum. The annualized continuously compounded risk-free rate is 10% per annum. Assume that the choice must be made on August 31, 2007. Using this data:

```
AssetPrice = 50;
Strike = 60;
Settlement = 'Jun-1-2007';
Maturity = 'Dec-2-2007';
ChooseDate = 'Aug-31-2007';
RiskFreeRate = 0.1;
Sigma = 0.20;
Yield = 0.05
```

Define the `RateSpec` and `StockSpec`:

```
RateSpec = intenvset('Compounding', -1, 'Rates', RiskFreeRate, 'StartDates',...
Settlement, 'EndDates', Maturity);
StockSpec = stockspec(Sigma, AssetPrice,'continuous',Yield);
```

Price the chooser option:

```
Price  = chooserbybls(RateSpec, StockSpec, Settlement, Maturity,...
Strike, ChooseDate)

Price =

8.9308
```

References   Rubinstein, Mark, "Options for the Undecided," *Risk 4*, 1991.

See Also   blsprice | intenvset

Purpose   Create financial structure or return financial structure class name

Syntax
```
Obj = classfin(ClassName)
Obj = classfin(Struct, ClassName)
ClassName = classfin(Obj)
```

Arguments

ClassName	String containing the name of a financial structure class.
Struct	MATLAB structure to be converted into a financial structure.
Obj	Name of a financial structure.

Description   `Obj = classfin(ClassName)` and `Obj = classfin(Struct, ClassName)` create a financial structure of class `ClassName`.

`ClassName = classfin(Obj)` returns a string containing a financial structure's class name.

Examples   **Example 1**. Create an `HJMTimeSpec` financial structure and complete its fields. (Typically, the function `hjmtimespec` is used to create `HJMTimeSpec` structures).

```
TimeSpec = classfin('HJMTimeSpec');
TimeSpec.ValuationDate = datenum('Dec-10-1999');
TimeSpec.Maturity = datenum('Dec-10-2002');
TimeSpec.Compounding = 2;
TimeSpec.Basis = 0;
TimeSpec.EndMonthRule = 1;
TimeSpec =

          FinObj: 'HJMTimeSpec'
   ValuationDate: 730464
        Maturity: 731560
```

```
      Compounding: 2
            Basis: 0
      EndMonthRule: 1
```

Example 2. Convert an existing MATLAB structure into a financial structure.

```
TSpec.ValuationDate = datenum('Dec-10-1999');
TSpec.Maturity = datenum('Dec-10-2002');
TSpec.Compounding = 2;
TSpec.Basis = 0;
TSpec.EndMonthRule = 0;
TimeSpec = classfin(TSpec, 'HJMTimeSpec')

TimeSpec =

   ValuationDate: 730464
        Maturity: 731560
     Compounding: 2
           Basis: 0
     EndMonthRule: 0
           FinObj: 'HJMTimeSpec'
```

Example 3. Obtain a financial structure's class name.

```
load deriv.mat
ClassName = classfin(HJMTree)
ClassName =

HJMFwdTree
```

See Also   isafin

Purpose	Price compound option from Cox-Ross-Rubinstein binomial tree

Syntax

```
[Price, PriceTree] = compoundbycrr(CRRTree, UOptSpec, UStrike,
USettle, UExerciseDates, UAmericanOpt, COptSpec,
CStrike, CSettle,  CExerciseDates, CAmericanOpt)
```

Arguments

CRRTree	Stock tree structure created by crrtree.
UOptSpec	String = 'Call' or 'Put'.
UStrike	1-by-1 vector of strike price values.
USettle	1-by-1 vector of Settle dates.
UExerciseDates	For a European option (UAmericanOpt = 0):
	1-by-1 vector of exercise dates. For a European option, there is only one exercise date, the option expiry date.
	For an American option (UAmericanOpt = 1):
	1-by-2 vector of exercise date boundaries. The option can be exercised on any tree date. If only one non-NaN date is listed, or if ExerciseDates is 1-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.
UAmericanOpt	If UAmericanOpt = 0, NaN, or is unspecified, the option is a European option. If UAmericanOpt = 1, the option is an American option.
COptSpec	NINST-by-1 list of string values 'Call' or 'Put' of the compound option.
CStrike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.

compoundbycrr

CSettle	1-by-1 vector containing the settlement or trade date.
CExerciseDates	For a European option (CAmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (CAmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.
CAmericanOpt	(Optional) If CAmericanOpt = 0, NaN, or is unspecified, the option is a European option. If CAmericanOpt = 1, the option is an American option.

Description   [Price, PriceTree] = compoundbycrr(CRRTree, UOptSpec, UStrike, USettle, UExerciseDates, UAmericanOpt, COptSpec, CStrike, CSettle, CExerciseDates, CAmericanOpt) calculates the value of a compound option.

Price is a NINST-by-1 vector of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples   Price a compound option using a CRR binomial tree.

Load the file `deriv.mat`, which provides `CRRTree`. The `CRRTree` structure contains the stock specification and time information needed to price the option.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
UOptSpec = 'Call';
UStrike = 130;
USettle = '01-Jan-2003';
UExerciseDates = '01-Jan-2006';
UAmericanOpt = 1;
COptSpec = 'Put';
CStrike = 5;
CSettle = '01-Jan-2003';
CExerciseDates = '01-Jan-2005';

Price = compoundbycrr(CRRTree, UOptSpec, UStrike, USettle, ...
UExerciseDates, UAmericanOpt, COptSpec, CStrike, CSettle, ...
CExerciseDates)

Price =

 2.8482
```

References   Rubinstein, Mark, "Double Trouble," *Risk 5*, 1991, p. 73.

See Also   crrtree | instcompound

compoundbyeqp

Purpose	Price compound option from Equal Probabilities binomial tree

Syntax

```
[Price, PriceTree] = compoundbyeqp(EQPTree, UOptSpec, UStrike,
USettle, UExerciseDates, UAmericanOpt, COptSpec,
CStrike, CSettle,  CExerciseDates, CAmericanOpt)
```

Arguments

EQPTree	Stock tree structure created by eqptree.
UOptSpec	String = 'Call' or 'Put'.
UStrike	1-by-1 vector of strike price values.
USettle	1-by-1 vector of Settle dates.
UExerciseDates	For a European option (UAmericanOpt = 0):
	1-by-1 vector of exercise dates. For a European option, there is only one exercise date, the option expiry date.
	For an American option (UAmericanOpt = 1):
	1-by-2 vector of exercise date boundaries. The option can be exercised on any tree date. If only one non-NaN date is listed, or if ExerciseDates is 1-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.
UAmericanOpt	If UAmericanOpt = 0, NaN, or is unspecified, the option is a European option. If UAmericanOpt = 1, the option is an American option.
COptSpec	NINST-by-1 list of string values 'Call' or 'Put' of the compound option.
CStrike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.

CSettle	1-by-1 vector containing the settlement or trade date.
CExerciseDates	For a European option (CAmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (CAmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.
CAmericanOpt	If CAmericanOpt = 0, NaN, or is unspecified, the option is a European option. If CAmericanOpt = 1, the option is an American option.

Description [Price, PriceTree] = compoundbyeqp(EQPTree, UOptSpec, UStrike, USettle, UExerciseDates, UAmericanOpt, COptSpec, CStrike, CSettle, CExerciseDates, CAmericanOpt) calculates the value of a compound option.

Price is a NINST-by-1 vector of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

compoundbyeqp

Examples   Price a compound option using an EQP equity tree.

Load the file `deriv.mat,` which provides `EQPTree`. The `EQPTree` structure contains the stock specification and time information needed to price the option.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
UOptSpec = 'Call';
UStrike = 130;
USettle = '01-Jan-2003';
UExerciseDates = '01-Jan-2006';
UAmericanOpt = 1;
COptSpec = 'Put';
CStrike = 5;
CSettle = '01-Jan-2003';
CExerciseDates = '01-Jan-2005';

Price = compoundbyeqp(EQPTree, UOptSpec, UStrike, USettle, ...
UExerciseDates, UAmericanOpt, COptSpec, CStrike, CSettle, ...
CExerciseDates)

Price =

 3.3931
```

References   Rubinstein, Mark, "Double Trouble," *Risk 5*, 1991, p. 73

See Also   eqptree | instcompound

Purpose	Price compound options using implied trinomial tree (ITT)

Syntax
```
[Price, PriceTree] = compoundbyitt(ITTTree, UOptSpec, UStrike,
USettle, UExerciseDates, UAmericanOpt, COptSpec,
CStrike, CSettle,  CExerciseDates, CAmericanOpt)
```

Arguments

ITTTree	Stock tree structure created by itttree.
UOptSpec	String = 'call' or 'put'.
UStrike	1-by-1 vector of strike price values.
USettle	1-by-1 vector of Settle dates.
UExerciseDates	For a European option (UAmericanOpt = 0):
	1-by-1 vector of exercise dates. For a European option, there is only one exercise date, the option expiry date.
	For an American option (UAmericanOpt = 1):
	1-by-2 vector of exercise date boundaries. The option can be exercised on any tree date. If only one non-NaN date is listed, or if ExerciseDates is 1-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.
UAmericanOpt	If UAmericanOpt = 0, NaN, or is unspecified, the option is a European option. If UAmericanOpt = 1, the option is an American option.
COptSpec	NINST-by-1 list of string values 'Call' or 'Put' of the compound option.
CStrike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.

compoundbyitt

CSettle	1-by-1 vector containing the settlement or trade date.	
CExerciseDates	For a European option (CAmericanOpt = 0):	
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.	
	For an American option (CAmericanOpt = 1):	
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.	
CAmericanOpt	(Optional) If CAmericanOpt = 0, NaN, or is unspecified, the option is a European option. If CAmericanOpt = 1, the option is an American option.	

Description [Price, PriceTree] = compoundbyitt(ITTTree, UOptSpec, UStrike, USettle, UExerciseDates, UAmericanOpt, COptSpec, CStrike, CSettle, CExerciseDates, CAmericanOpt) calculates the value of a compound option by an ITT trinomial tree.

Price is a NINST-by-1 vector of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Note The Settle date is set to the ValuationDate of the stock tree.

Examples   Price a compound option using an ITT tree.

Load the file deriv.mat which provides the ITTTree. The ITTTree
structure contains the stock specification and time information needed
to price the option.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
UOptSpec = 'Call';
UStrike = 99;
USettle = '01-Jan-2006';
UExerciseDates = '01-Jan-2010';
UAmericanOpt = 1;
COptSpec = 'Put';
CStrike = 5;
CSettle = '01-Jan-2006';
CExerciseDates = '01-Jan-2010';

Price = compoundbyitt(ITTTree, UOptSpec, UStrike, USettle, ...
UExerciseDates, UAmericanOpt, COptSpec, CStrike, CSettle, ...
CExerciseDates)

Price =

 2.727
```

References   Rubinstein, Mark, "Double Trouble," *Risk 5*, 1991.

See Also   instcompound | itttree

crrprice

Purpose	Instrument prices from Cox-Ross-Rubinstein tree
Syntax	[Price, PriceTree] = crrprice(CRRTree, InstSet, Options)

Arguments

CRRTree	Stock price tree structure created by crrtree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type; each type can have different data fields. The stored data field is a row vector or string for each instrument. For more information about how to create the InstSet structure, see instadd.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Price, PriceTree] = crrprice(CRRTree, InstSet, Options) computes stock option prices using a CRR binomial tree created with crrtree.

Price is a number of instruments (NINST)-by-1 vector of prices for each instrument. The prices are computed by backward dynamic programming on the stock tree. If an instrument cannot be priced, NaN is returned.

PriceTree is a MATLAB structure of trees containing vectors of instrument prices and a vector of observation times for each node.

PriceTree.PTree contains the prices.

PriceTree.tObs contains the observation times.

PriceTree.dObs contains the observation dates.

crrprice handles instrument types: 'Asian', 'Barrier', 'Compound', 'Lookback', 'OptStock'. See instadd to construct defined types.

Related single-type pricing functions are:

- `asianbycrr`: Price an Asian option from a CRR tree.

- `barrierbycrr`: Price a barrier option from a CRR tree.

- `compoundbycrr`: Price a compound option from a CRR tree.

- `lookbackbycrr`: Price a lookback option from a CRR tree.

- `optstockbycrr`: Price an American, Bermuda, or European option from a CRR tree.

Examples   Load the CRR tree and instruments from the data file `deriv.mat`. Price the barrier and lookback options contained in the instrument set.

```
load deriv.mat;
CRRSubSet = instselect(CRRInstSet,'Type', ...
{'Barrier', 'Lookback'});

instdisp(CRRSubSet)

%Table of instrument portfolio partially displayed:
Index Type OptSpec Strike Settle   ExerciseDates AmericanOpt BarrierSpec ...
1     Barrier call 105   01-Jan-2003 01-Jan-2006 1           ui ...


Index Type      OptSpec Strike Settle      ExerciseDates AmericanOpt Name    Quantity
2     Lookback call     115    01-Jan-2003 01-Jan-2006   0           Lookback1 7
3     Lookback call     115    01-Jan-2003 01-Jan-2007   0           Lookback2 9

[Price, PriceTree] = crrprice(CRRTree, CRRSubSet)

Price =

   12.1272
    7.6015
   11.7772

PriceTree =
```

```
FinObj: 'BinPriceTree'
 PTree: {1x5 cell}
  tObs: [0 1 2 3 4]
  dObs: [731582 731947 732313 732678 733043]
```

You can use treeviewer to see the prices of these three instruments along the price tree.

```
treeviewer(PriceTree, CRRSubSet)
```


Barrier1

Lookback1

Lookback2

See Also   crrsens | crrtree | instadd

Purpose	Instrument prices and sensitivities from Cox-Ross-Rubinstein tree
Syntax	[Delta, Gamma, Vega, Price] = crrsens(CRRTree, InstSet, Options)

Arguments

CRRTree	Interest-rate tree structure created by crrtree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Delta, Gamma, Vega, Price] = crrsens(CRRTree, InstSet, Options) computes dollar sensitivities and prices for instruments using a binomial tree created with crrtree. NINST instruments from a financial instrument variable, InstSet, are priced. crrsens handles instrument types: 'Asian', 'Barrier', 'Compound', 'Lookback', 'OptStock'. See instadd for information on instrument types.

Delta is an NINST-by-1 vector of deltas, representing the rate of change of instrument prices with respect to changes in the stock price. Delta is computed by finite differences in calls to crrtree. See crrtree for information on the stock tree.

Gamma is an NINST-by-1 vector of gammas, representing the rate of change of instrument deltas with respect to the changes in the stock price. Gamma is computed by finite differences in calls to crrtree.

Vega is an NINST-by-1 vector of vegas, representing the rate of change of instrument prices with respect to the changes in the volatility of the stock. Vega is computed by finite differences in calls to crrtree.

Note All sensitivities are returned as dollar sensitivities. To find the
per-dollar sensitivities, divide by the respective instrument price.

Examples   Load the CRR tree and instruments from the data file `deriv.mat`.
Compute the `Delta` and `Gamma` sensitivities of the barrier and lookback
options contained in the instrument set.

```
load deriv.mat;
CRRSubSet = instselect(CRRInstSet,'Type', ...
{'Barrier', 'Lookback'});

instdisp(CRRSubSet)

%Table of instrument portfolio partially displayed:
Index Type OptSpec Strike Settle  ExerciseDates AmericanOpt BarrierSpec ...
1     Barrier call 105   01-Jan-2003 01-Jan-2006 1            ui ...


Index Type   OptSpec Strike Settle    ExerciseDates AmericanOpt Name Quantity
2     Lookback call    115   01-Jan-2003 01-Jan-2006 0          Lookback1 7
3     Lookback call    115   01-Jan-2003 01-Jan-2007 0          Lookback2 9

[Delta, Gamma] = crrsens(CRRTree, CRRSubSet)

Delta =

 0.6885
 0.6049
 0.8187

Gamma =

 0.0310
-0.0000
 0.0000
```

crrsens

See Also   crrprice | crrtree

Purpose	Specify time structure for Cox-Ross-Rubinstein tree

Syntax   TimeSpec = crrtimespec(ValuationDate, Maturity, NumPeriods)

Arguments

ValuationDate	Scalar date indicating the pricing date and first observation in the tree. A serial date number or date string.
Maturity	Scalar date indicating depth of the tree.
NumPeriods	Scalar determining number of time steps in the tree.

Description   TimeSpec = crrtimespec(ValuationDate, Maturity, NumPeriods) sets the number of levels and node times for a CRR binomial tree.

TimeSpec is a structure specifying the time layout for a CRR binomial tree.

Examples   Specify a four-period CRR tree with time steps of 1 year.

```
ValuationDate = '1-July-2002';
Maturity = '1-July-2006';
TimeSpec = crrtimespec(ValuationDate, Maturity, 4)

 TimeSpec =

        FinObj: 'BinTimeSpec'
   ValuationDate: 731398
        Maturity: 732859
      NumPeriods: 4
           Basis: 0
     EndMonthRule: 1
tObs: [0 1 2 3 4]
dObs: [1x5 double]
```

crrtimespec

Purpose   Construct Cox-Ross-Rubinstein stock tree

Syntax   CRRTree = crrtree(StockSpec, RateSpec, TimeSpec)

Arguments

StockSpec	Stock specification. See `stockspec` for information on creating a stock specification.
RateSpec	Interest-rate specification for the initial risk free rate curve. See `intenvset` for information on declaring an interest-rate variable.
TimeSpec	Tree time layout specification. Defines the observation dates of the CRR binomial tree. See `crrtimespec` for information on the tree structure.

Note The standard CRR tree assumes a constant interest rate, but `RateSpec` allows you to specify an interest-rate curve with varying rates. If you specify variable interest rates, the resulting tree will not be a standard CRR tree.

Description   CRRTree = crrtree(StockSpec, RateSpec, TimeSpec) creates a structure specifying the time layout for a CRR binomial tree.

Examples   Using the data provided, create a stock specification (StockSpec), rate specification (RateSpec), and tree time layout specification (TimeSpec). Then use these specifications to create a CRR tree with `crrtree`.

```
Sigma = 0.20;
AssetPrice = 50;
DividendType = 'cash';
DividendAmounts = [0.50; 0.50; 0.50; 0.50];
ExDividendDates = {'03-Jan-2003'; '01-Apr-2003'; '05-July-2003';
```

```
'01-Oct-2003'}

StockSpec = stockspec(Sigma, AssetPrice, DividendType, ...
DividendAmounts, ExDividendDates)

StockSpec =

              FinObj: 'StockSpec'
               Sigma: 0.2000
          AssetPrice: 50
         DividendType: 'cash'
      DividendAmounts: [4x1 double]
      ExDividendDates: [4x1 double]

RateSpec = intenvset('Rates', 0.05, 'StartDates',...
'01-Jan-2003', 'EndDates', '31-Dec-2003')

RateSpec =

           FinObj: 'RateSpec'
       Compounding: 2
             Disc: 0.9519
            Rates: 0.0500
         EndTimes: 1.9945
       StartTimes: 0
          EndDates: 731946
        StartDates: 731582
     ValuationDate: 731582
            Basis: 0
      EndMonthRule: 1

ValuationDate = '1-Jan-2003';
Maturity = '31-Dec-2003';
TimeSpec = crrtimespec(ValuationDate, Maturity, 4)

  TimeSpec =
```

```
         FinObj: 'BinTimeSpec'
  ValuationDate: 731582
       Maturity: 731946
      NumPeriods: 4
          Basis: 0
    EndMonthRule: 1
            tObs: [0 0.2493 0.4986 0.7479 0.9972]
            dObs: [731582 731673 731764 731855 731946]


CRRTree = crrtree(StockSpec, RateSpec, TimeSpec)


CRRTree =

       FinObj: 'BinStockTree'
       Method: 'CRR'
    StockSpec: [1x1 struct]
     TimeSpec: [1x1 struct]
     RateSpec: [1x1 struct]
         tObs: [0 0.2493 0.4986 0.7479 0.9972]
         dObs: [731582 731672 731763 731856 731946]
        STree: {1x5 cell}
       UpProbs: [0.5370 0.5370 0.5370 0.5370]
```

Use treeviewer to observe the tree you have created.

See Also   crrtimespec | intenvset | stockspec

Purpose   Convert inverse-discount tree to interest-rate tree

Syntax   RateTree = cvtree(Tree)

Arguments

    Tree   Heath-Jarrow-Morton, Black-Derman-Toy, Hull-White,
                 or Black-Karasinski tree structure using inverse-discount
                 notation for forward rates.

Description   RateTree = cvtree(Tree) converts a tree structure using
inverse-discount notation to a tree structure using rate notation for
forward rates.

Examples   Convert a Hull-White tree using inverse-discount notation to a
Hull-White tree displaying interest-rate notation.

```
load deriv.mat;
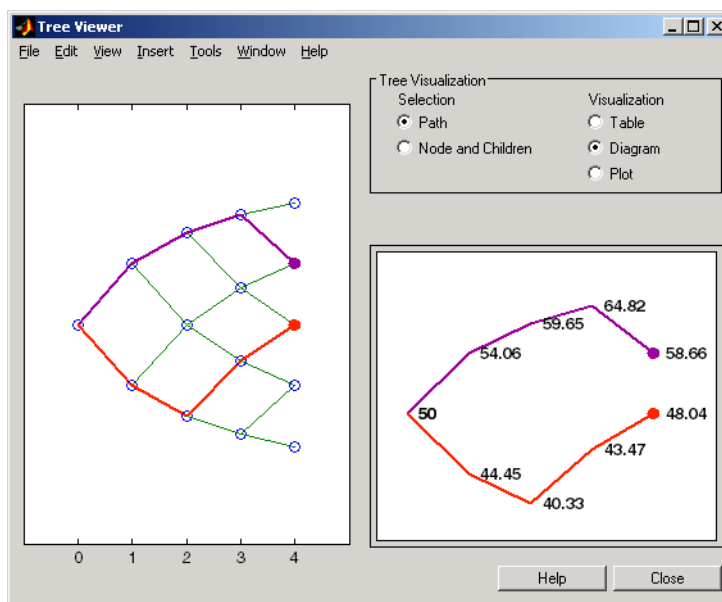
HWTree

HWTree =

     FinObj: 'HWFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 1 2 3]
       dObs: [731947 732313 732678 733043]
      CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
       Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
     Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
     FwdTree: {1x4 cell}

HWTree.FwdTree{1}
```

```
ans =
    1.0279

HWTree.FwdTree{2}

ans =
    1.0528    1.0356    1.0186
```

Use `treeviewer` to display the path of interest rates expressed in inverse-discount notation.

```
treeviewer(HWTree)
```


Use `cvtree` to convert the inverse-discount notation to interest-rate notation.

```
RTree = cvtree(HWTree)
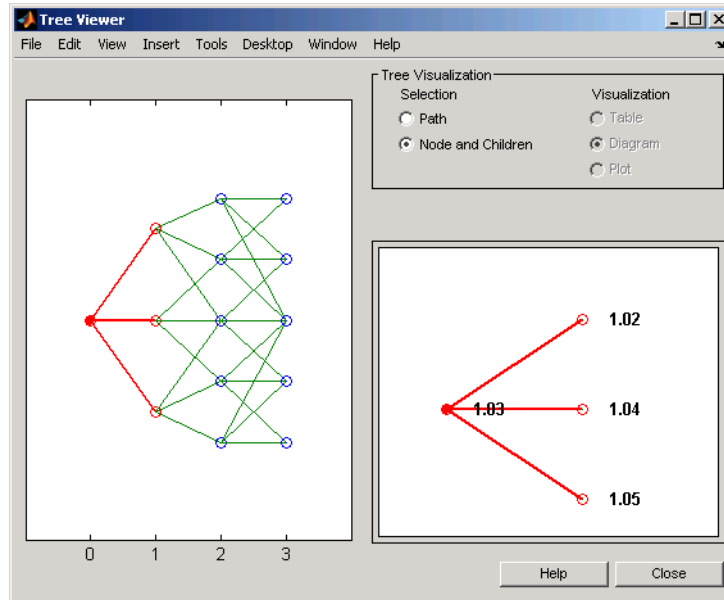
RTree =
```

```
       FinObj: 'HWRateTree'
      VolSpec: [1x1 struct]
     TimeSpec: [1x1 struct]
     RateSpec: [1x1 struct]
         tObs: [0 1 2 3]
         dObs: [731947 732313 732678 733043]
       CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
        Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
      Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
      RateTree: {1x4 cell}
```

```
RTree.RateTree{1}
```

```
ans =
    0.0275
```

```
RTree.RateTree{2}
```

```
ans =
    0.0514    0.0349    0.0185
```

Now use `treeviewer` to display the converted tree, showing the path
of interest rates expressed as forward rates.

See Also   disc2rate | rate2disc

Purpose	Time and frequency from dates
Syntax	[Times, F] = date2time(Settle, Dates, Compounding, Basis, EndMonthRule)

Arguments

Settle	Settlement date. A vector of serial date numbers or date strings.
Dates	Vector of dates corresponding to the compounding value.
Compounding	(Optional) Scalar value representing the rate at which the input zero rates were compounded when annualized. This argument determines the formula for the discount factors:
	Compounding = 1, 2, 3, 4, 6, 12 (Default = 2.)
	Disc = (1 + Z/F)^(-T), where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.
	Compounding = 365
	Disc = (1 + Z/F)^(-T), where F is the number of days in the basis year and T is a number of days elapsed computed by basis.
	Compounding = -1
	Disc = exp(-T*Z), where T is time in years.

date2time

Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

Description   [Times, F] = date2time(Settle, Dates, Compounding, Basis, EndMonthRule) computes time factors appropriate to compounded rate quotes beyond the settlement date.

Times is a vector of time factors.

F is a scalar of related compounding frequencies.

> **Note** To obtain accurate results from this function, the Basis and Dates arguments must be consistent. If the Dates argument contains months that have 31 days, Basis must be one of the values that allow months to contain more than 30 days; for example, Basis = 0, 3, or 7.

date2time is the inverse of time2date.

See Also   cftimes | disc2rate | rate2disc | time2date

datedisp

Purpose   Display date entries

Syntax   datedisp(NumMat, DateForm)
              CharMat = datedisp(NumMat, DateForm)

Arguments

NumMat	Numeric matrix to display.
DateForm	(Optional) Date format. See datestr for available and default format flags.

Description   datedisp(NumMat, DateForm) displays the matrix with the serial dates formatted as date strings, using a matrix with mixed numeric entries and serial date number entries. Integers between datenum('01-Jan-1900') and datenum('01-Jan-2200') are assumed to be serial date numbers, while all other values are treated as numeric entries.

CharMat is a character array representing NumMat. If no output variable is assigned, the function prints the array to the display (CharMat = datedisp(NumMat, DateForm)).

Examples
```
NumMat = [ 730730, 0.03, 1200, 730100;
           730731, 0.05, 1000, NaN]

NumMat =

   1.0e+05 *

    7.3073    0.0000    0.0120    7.3010
    7.3073    0.0000    0.0100       NaN

datedisp(NumMat)

   01-Sep-2000   0.03   1200   11-Dec-1998
```

```
02-Sep-2000   0.05   1000      NaN
```

Tips This function is identical to the datedisp function in Financial Toolbox software.

See Also datenum | datestr

derivget

Purpose	Get derivatives pricing options
Syntax	Value = derivget(Options, '*Parameter*')

Arguments

Options	Existing options specification structure, probably created from previous call to derivset.
Parameter	Must be 'Diagnostics', 'Warnings', 'ConstRate', or 'BarrierMethod'. It is sufficient to type only the leading characters that uniquely identify the parameter. Case is ignored for parameter names.

Description Value = derivget(Options, '*Parameter*') extracts the value of the named parameter from the derivative options structure Options. Parameter values can be 'off' or 'on', except for 'BarrierMethod', which can be 'unenhanced' or 'interp'. Specifying 'unenhanced' uses no correction calculation. Specifying 'interp' uses an enhanced valuation interpolating between nodes on barrier boundaries.

Examples **Example 1**. Create an Options structure with the value of Diagnostics set to 'on'.

```
Options = derivset('Diagnostics','on')
```

Use derivget to extract the value of Diagnostics from the Options structure.

```
Value = derivget(Options, 'Diagnostics')

Value =

on
```

Example 2. Use derivget to extract the value of ConstRate.

```
Value   = derivget(Options, 'ConstRate')
```

```
Value =
```

```
on
```

Because the value of `'ConstRate'` was not previously set with
`derivset`, the answer represents the default setting for `'ConstRate'`.

Example 3. Find the value of `'BarrierMethod'` in this structure.

```
derivget(Options ,'BarrierMethod')
```

```
ans =
```

```
unenhanced
```

See Also   barrierbycrr | barrierbyeqp | derivset

derivset

Purpose	Set or modify derivatives pricing options

Syntax
```
Options = derivset(Options, 'Parameter1', Value1,
... 'Parameter4', Value4)
Options = derivset(OldOptions, NewOptions)
Options = derivset
derivset
```

Arguments

Options	(Optional) Existing options specification structure, probably created from a previous call to derivset.
Parameter*n*	The parameter must be 'Diagnostics', 'Warnings', 'ConstRate', or 'BarrierMethod'. Parameters can be entered in any order.
Value*n*	(BDT, BK, HJM, or HW pricing only) The parameter values for the following three options can be 'on' or 'off':

      • 'Diagnostics' 'on' generates diagnostic information. The default is 'Diagnostics' 'off'.

      • 'Warnings' 'on' (default) displays a warning message when executing a pricing function.

      • 'ConstRate' 'on' (default) assumes a constant rate between tree nodes.

      For pricing barrier options, the 'BarrierMethod' pricing option can be 'unenhanced' (default) or 'interp'. Specifying 'unenhanced' uses no correction calculation. Specifying 'interp' uses an enhanced valuation interpolating between nodes on barrier boundaries.

OldOptions	Existing options specification structure.
NewOptions	New options specification structure.

Description   Options = derivset(Options, 'Parameter1', Value1,
... 'Parameter4', Value4) creates a derivatives pricing options
structure Options in which the named parameters have the specified
values. Any unspecified value is set to the default value for that
parameter when Options is passed to the pricing function. It is
sufficient to type only the leading characters that uniquely identify the
parameter name. Case is also ignored for parameter names.

If the optional input argument Options is specified, derivset modifies
an existing pricing options structure by changing the named parameters
to the specified values.

Note For parameter *values*, correct case and the complete string are
required; if an invalid string is provided, the default is used.

Options = derivset(OldOptions, NewOptions) combines an existing
options structure OldOptions with a new options structure NewOptions.
Any parameters in NewOptions with nonempty values overwrite the
corresponding old parameters in OldOptions.

Options = derivset creates an options structure Options whose fields
are set to the default values.

derivset with no input or output arguments displays all parameter
names and information about their possible values.

Examples   Options = derivset('Diagnostics','on')

enables the display of additional diagnostic information that appears
when executing pricing functions.

derivset

```
Options = derivset(Options, 'ConstRate', 'off')
```

changes the ConstRate parameter in the existing Options structure so that the assumption of constant rates between tree nodes no longer applies.

With no input or output arguments derivset displays all parameter names and information about their possible values.

```
derivset

    Diagnostics: [ on   | {off} ]
       Warnings: [ {on} | off   ]
      ConstRate: [ {on} | off   ]
   BarrierMethod: [ {unenhanced} | interp   ]
```

See Also   barrierbycrr | barrierbyeqp | derivget

disc2rate

Purpose   Interest rates from cash flow discounting factors

Syntax   **Usage 1**: Interval points are input as times in periodic units.

```
Rates = disc2rate(Compounding, Disc, EndTimes)
```

```
Rates = disc2rate(Compounding, Disc, EndTimes, StartTimes)
```

Usage 2: `ValuationDate` is passed and interval points are input as dates.
```
[Rates, EndTimes, StartTimes] = disc2rate(Compounding, Disc,
EndDates, StartDates, ValuationDate)
[Rates, EndTimes, StartTimes] = disc2rate(Compounding, Disc,
EndDates, StartDates, ValuationDate, Basis, EndMonthRule)
```

Arguments

Compounding	Scalar value representing the rate at which the input zero rates were compounded when annualized. This argument determines the formula for the discount factors:
	Compounding = 1, 2, 3, 4, 6, 12
	Disc = $(1 + Z/F)^{(-T)}$, where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.
	Compounding = 365
	Disc = $(1 + Z/F)^{(-T)}$, where F is the number of days in the basis year and T is a number of days elapsed computed by basis.
	Compounding = -1
	Disc = $\exp(-T*Z)$, where T is time in years.
Disc	Number of points (NPOINTS) by number of curves (NCURVES) matrix of discounts. Disc are

unit bond prices over investment intervals from `StartTimes`, when the cash flow is valued, to `EndTimes`, when the cash flow is received.

EndTimes NPOINTS-by-1 vector or scalar of times in periodic units ending the interval to discount over.

Note When `ValuationDate` is not passed, the `EndTimes` and `StartTimes` arguments are interpreted as times.

StartTimes (Optional) NPOINTS-by-1 vector or scalar of times in periodic units starting the interval to discount over. Default = 0.

EndDates NPOINTS-by-1 vector or scalar of serial maturity dates ending the interval to discount over.

Note When `ValuationDate` is passed, `EndDates` and `StartDates` arguments are interpreted as dates. The date `ValuationDate` is used as the zero point for computing the times.

StartDates (Optional) NPOINTS-by-1 vector or scalar of serial dates starting the interval to discount over. Default = `ValuationDate`. `StartDates` must be earlier than `EndDates`.

ValuationDate Scalar value in serial date number form representing the observation date of the investment horizons entered in `StartDates` and `EndDates`. Required in **Usage 2**. Omitted or passed as an empty matrix to invoke **Usage 1**.

Basis	(Optional) Day-count basis of the instrument when using **Usage 2**. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) End-of-month rule when using **Usage 2**. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

disc2rate

Description   **Usage 1**: `Rates = disc2rate(Compounding, Disc, EndTimes)` or
`Rates = disc2rate(Compounding, Disc, EndTimes, StartTimes)`
where interval points are input as times in periodic units.

Usage 2: `[Rates, EndTimes, StartTimes] =
disc2rate(Compounding, Disc, EndDates, StartDates,
ValuationDate)` or `[Rates, EndTimes, StartTimes] =
disc2rate(Compounding, Disc, EndDates, StartDates,
ValuationDate, Basis, EndMonthRule)` where `ValuationDate` is
passed and interval points are input as dates.

`disc2rate` computes the yields over a series of `NPOINTS` time intervals
given the cash flow discounts over those intervals. `NCURVES` different
rate curves can be translated at once if they have the same time
structure. The time intervals can represent a zero or a forward curve.

`Rates` is an `NPOINTS`-by-`NCURVES` column vector of yields in decimal form
over the `NPOINTS` time intervals.

Specify the investment intervals with either input times (**Usage 1**)
or input dates (**Usage 2**). Entering `ValuationDate` invokes the date
interpretation; omitting `ValuationDate` invokes the default time
interpretations.

For **Usage 1**:

- `StartTimes` is an `NPOINTS`-by-1 column vector of times starting the
  interval to discount over, measured in periodic units.

- `EndTimes` is an `NPOINTS`-by-1 column vector of times ending the
  interval to discount over, measured in periodic units.

For **Usage 2**:

- `StartDates` is an `NPOINTS`-by-1 column vector of serial dates starting
  the interval to discount over, measured in days.

- `EndDates` is an `NPOINTS`-by-1 column vector of serial date ending the
  interval to discount over, measured in days.

If `Compounding = 365` (daily), `StartTimes` and `EndTimes` are measured
in days for **Usage 2**. Otherwise, for **Usage 1**, the arguments contain

values, T, computed from SIA semiannual time factors, Tsemi, by the formula T = Tsemi/2 * F, where F is the compounding frequency.

See Also   rate2disc | ratetimes

eqpprice

Purpose	Instrument prices from Equal Probabilities binomial tree
Syntax	[Price, PriceTree] = eqpprice(EQPTree, InstSet, Options)

Arguments

EQPTree	Stock price tree structure created by eqptree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description [Price, PriceTree] = eqpprice(EQPTree, InstSet, Options) computes stock option prices using an EQP binomial tree created with eqptree.

Price is a number of instruments (NINST)-by-1 vector of prices for each instrument. The prices are computed by backward dynamic programming on the stock tree. If an instrument cannot be priced, NaN is returned.

PriceTree is a MATLAB structure of trees containing vectors of instrument prices and a vector of observation times for each node.

PriceTree.PTree contains the prices.

PriceTree.tObs contains the observation times.

PriceTree.dObs contains the observation dates.

eqpprice handles instrument types: 'Asian', 'Barrier', 'Compound', 'Lookback', 'OptStock'. See instadd to construct defined types.

Related single-type pricing functions are:

• asianbyeqp: Price an Asian option from an EQP tree.

- barrierbyeqp: Price a barrier option from an EQP tree.

- compoundbyeqp: Price a compound option from an EQP tree.

- lookbackbyeqp: Price a lookback option from an EQP tree.

- optstockbyeqp: Price an American, Bermuda, or European option from an EQP tree.

Examples   Load the EQP tree and instruments from the data file deriv.mat. Price the put options contained in the instrument set.

```
load deriv.mat;
EQPSubSet = instselect(EQPInstSet, 'FieldName', 'OptSpec', ...
'Data', 'put')

instdisp(EQPSubSet)

%Table of instrument portfolio partially displayed:
Index Type     OptSpec Strike Settle      ExerciseDates AmericanOpt Name...
1     OptStock put     105    01-Jan-2003 01-Jan-2006   0           Put 105...


Index Type  OptSpec Strike Settle      ExerciseDates AmericanOpt AvgType...
2     Asian put     110    01-Jan-2003 01-Jan-2006   0           arithmetic...
3     Asian put     110    01-Jan-2003 01-Jan-2007   0           arithmetic...


[Price, PriceTree] = eqpprice(EQPTree, EQPSubSet)

Price =

    2.6375
    4.7444
    3.9178

PriceTree =

    FinObj: 'BinPriceTree'
```

```
PTree: {1x5 cell}
 tObs: [0 1 2 3 4]
 dObs: [731582 731947 732313 732678 733043]
```

You can use treeviewer to see the prices of these three instruments along the price tree.

treeviewer(PriceTree, EQPSubSet)

Put1

Asian11

Asian2

See Also   eqpsens | eqptimespec | eqptree

Purpose	Instrument prices and sensitivities from Equal Probabilities binomial tree
Syntax	[Delta, Gamma, Vega, Price] = eqpsens(EQPTree, InstSet, Options)

Arguments

EQPTree	Interest-rate tree structure created by `eqptree`.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with `derivset`.

Description [Delta, Gamma, Vega, Price] = eqpsens(EQPTree, InstSet, Options) computes dollar sensitivities and prices for instruments using a binomial tree created with `eqptree`. NINST instruments from a financial instrument variable, InstSet, are priced. eqpsens handles instrument types: 'Asian', 'Barrier', 'Compound', 'Lookback', and 'OptStock'. See instadd for information on instrument types.

Delta is an NINST-by-1 vector of deltas, representing the rate of change of instrument prices with respect to changes in the stock price. Delta is computed by finite differences in calls to eqptree. See eqptree for information on the stock tree.

Gamma is an NINST-by-1 vector of gammas, representing the rate of change of instrument deltas with respect to the changes in the stock price. Gamma is computed by finite differences in calls to eqptree.

Vega is an NINST-by-1 vector of vegas, representing the rate of change of instrument prices with respect to the changes in the volatility of the stock. Vega is computed by finite differences in calls to eqptree.

> **Note** All sensitivities are returned as dollar sensitivities. To find the
> per-dollar sensitivities, divide by the respective instrument price.

Examples   Load the EQP tree and instruments from the data file `deriv.mat`.
Compute the `Delta` and `Gamma` sensitivities of the put options contained
in the instrument set.

```
load deriv.mat;

EQPSubSet = instselect(EQPInstSet, 'FieldName', 'OptSpec', ...
'Data', 'put')

instdisp(EQPSubSet)

%Table of instrument portfolio partially displayed:
Index Type     OptSpec Strike Settle       ExerciseDates AmericanOpt Name...
1     OptStock put     105    01-Jan-2003 01-Jan-2006    0           Put 105...

Index Type  OptSpec Strike Settle      ExerciseDates AmericanOpt AvgType...
2     Asian put     110    01-Jan-2003 01-Jan-2006   0           arithmetic...
3     Asian put     110    01-Jan-2003 01-Jan-2007   0           arithmetic...

[Delta, Gamma] = eqpsens(EQPTree, EQPSubSet)

Delta =

  -0.2336
  -0.5443
  -0.4516

Gamma =

  0.0218
  0.0000
```

```
0.0000
```

See Also   eqpprice | eqptree

eqptimespec

Purpose	Specify time structure for Equal Probabilities binomial tree
Syntax	TimeSpec = eqptimespec(ValuationDate, Maturity, NumPeriods)

Arguments

ValuationDate	Scalar date indicating the pricing date and first observation in the tree. A serial date number or date string.
Maturity	Scalar date indicating depth of the tree.
NumPeriods	Scalar determining number of time steps in the tree.

Description TimeSpec = eqptimespec(ValuationDate, Maturity, NumPeriods) sets the number of levels and node times for an equal probabilities tree.

TimeSpec is a structure specifying the time layout for an equal probabilities tree.

Examples Specify a four-period tree with time steps of 1 year.

```
ValuationDate = '1-July-2002';
Maturity = '1-July-2006';
TimeSpec = eqptimespec(ValuationDate, Maturity, 4)

TimeSpec =

            FinObj: 'BinTimeSpec'
     ValuationDate: 731398
          Maturity: 732859
        NumPeriods: 4
             Basis: 0
       EndMonthRule: 1
tObs: [0 1 2 3 4]
dObs: [1x5 double]
```

See Also　　　eqptree | stockspec

eqptree

Purpose	Construct Equal Probabilities stock tree

Syntax   EQPTree = eqptree(StockSpec, RateSpec, TimeSpec)

Arguments

StockSpec	Stock specification. See stockspec for information on creating a stock specification.
RateSpec	Interest-rate specification for the initial risk free rate curve. See intenvset for information on declaring an interest-rate variable.
TimeSpec	Tree time layout specification. Defines the observation dates of the equal probabilities binomial tree. See eqptimespec for information on the tree structure.

Note The standard equal probabilities tree assumes a constant interest rate, but RateSpec allows you to specify an interest-rate curve with varying rates. If you specify variable interest rates, the resulting tree will not be a standard equal probabilities tree.

Description   EQPTree = eqptree(StockSpec, RateSpec, TimeSpec) constructs an equal probabilities stock tree.

EQPTree is a MATLAB structure specifying the time layout for an equal probabilities stock tree.

Examples   Using the data provided, create a stock specification (StockSpec), rate specification (RateSpec), and tree time layout specification (TimeSpec). Then use these specifications to create an EQP tree with eqptree.

```
Sigma = 0.20;
AssetPrice = 50;
```

```
DividendType = 'cash';
DividendAmounts = [0.50; 0.50; 0.50; 0.50];
ExDividendDates = {'03-Jan-2003'; '01-Apr-2003'; '05-July-2003';
'01-Oct-2003'}

StockSpec = stockspec(Sigma, AssetPrice, DividendType, ...
DividendAmounts, ExDividendDates)

StockSpec =

                FinObj: 'StockSpec'
                 Sigma: 0.2000
            AssetPrice: 50
           DividendType: 'cash'
        DividendAmounts: [4x1 double]
        ExDividendDates: [4x1 double]

RateSpec = intenvset('Rates', 0.05, 'StartDates',...
'01-Jan-2003', 'EndDates', '31-Dec-2003')

RateSpec =

             FinObj: 'RateSpec'
        Compounding: 2
               Disc: 0.9519
              Rates: 0.0500
           EndTimes: 1.9945
         StartTimes: 0
           EndDates: 731946
         StartDates: 731582
      ValuationDate: 731582
              Basis: 0
        EndMonthRule: 1

ValuationDate = '1-Jan-2003';
Maturity = '31-Dec-2003';
TimeSpec = eqptimespec(ValuationDate, Maturity, 4)
```

```
TimeSpec =

             FinObj: 'BinTimeSpec'
      ValuationDate: 731582
           Maturity: 731946
          NumPeriods: 4
              Basis: 0
        EndMonthRule: 1
               tObs: [0 0.2493 0.4986 0.7479 0.9972]
               dObs: [731582 731673 731764 731855 731946]


EQPTree = eqptree(StockSpec, RateSpec, TimeSpec)

EQPTree =

        FinObj: 'BinStockTree'
        Method: 'EQP'
     StockSpec: [1x1 struct]
      TimeSpec: [1x1 struct]
      RateSpec: [1x1 struct]
          tObs: [0 0.2493 0.4986 0.7479 0.9972]
          dObs: [731582 731672 731763 731856 731946]
         STree: {1x5 cell}
        UpProbs: [0.5000 0.5000 0.5000 0.5000]
```

Use treeviewer to observe the tree you have created.

See Also   eqptimespec | intenvset | stockspec

fixedbybdt

Purpose	Price fixed-rate note from Black-Derman-Toy interest-rate tree
Syntax	[Price, PriceTree] = fixedbybdt(BDTTree, CouponRate, Settle, Maturity) [Price, PriceTree] = fixedbybdt(BDTTree, CouponRate, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule) [Price, PriceTree] = fixedbybdt(BDTTree, CouponRate, Settle, Maturity, Name,Value)

Input Arguments

BDTTree	Interest-rate tree structure created by bdttree.
CouponRate	Decimal annual rate.
Settle	Settlement dates. Number of instruments (NINST)-by-1 vector of dates representing the settlement dates of the fixed-rate note.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the fixed-rate note.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

NINST-by-1 vector representing the frequency of payments per year.

> **Default:** 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual

- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** `0` (actual/actual)

Principal

The notional principal amount.

> **Default:** `100`

Options

Derivatives pricing options structure created with `derivset`.

EndMonthRule

End-of-month rule. A `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Default: `1`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1 of logicals.

Default: `False`

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with possible choices of business day convention:

- `actual`
- `follow`
- `modifiedfollow`
- `previous`
- `modifiedprevious`

Default: `actual`

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB date numbers.

Default: If no dates are specified, holidays.m is used.

Description   [Price, PriceTree] = fixedbybdt(BDTTree, CouponRate, Settle, Maturity) computes the price of a fixed-rate note from a BDT interest-rate tree.

[Price, PriceTree] = fixedbybdt(BDTTree, CouponRate, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule) computes the price of a fixed-rate note from a BDT interest-rate tree using additional input arguments.

[Price, PriceTree] = fixedbybdt(BDTTree, CouponRate, Settle, Maturity, Name,Value) computes the price of a price of a fixed-rate note from a BDT interest-rate tree with additional options specified by one or more Name,Value pair arguments.

Price is an NINST-by-1 vector of expected prices of the fixed-rate note at time 0.

PriceTree is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PTree contains the clean prices.

PriceTree.AITree contains the accrued interest.

PriceTree.tObs contains the observation times.

The Settle date for every fixed-rate note is set to the ValuationDate of the BDT tree. The fixed-rate note argument Settle is ignored.

fixedbybdt

Examples   Price a 10% fixed-rate note using a BDT interest-rate tree.

Load the file `deriv.mat`, which provides `BDTTree`. The `BDTTree` structure contains the time and interest-rate information needed to price the note.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
CouponRate = 0.10;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2004';
Reset = 1;
```

Use `fixedbybdt` to compute the price of the note.

```
Price = fixedbybdt(BDTTree, CouponRate, Settle, Maturity, Reset)

Price =

  92.9974
```

See Also   bdttree | bondbybdt | capbybdt | cfbybdt | floatbybdt |
floorbybdt | swapbybdt

Purpose   Price fixed-rate note from Black-Karasinski interest-rate tree

Syntax   [Price, PriceTree] = fixedbybk(BKTree, CouponRate, Settle, Maturity)
[Price, PriceTree] = fixedbybk(BKTree, CouponRate, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule)
[Price, PriceTree] = fixedbybk(BKTree, CouponRate, Settle, Maturity, Name,Value)

Input Arguments

BKTree	Interest-rate tree structure created by bktree.
CouponRate	Decimal annual rate.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the fixed-rate note.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the fixed-rate note.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

NINST-by-1 vector representing the frequency of payments per year.

> **Default:** 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual

- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

The notional principal amount.

> **Default:** 100

Options

Derivatives pricing options structure created with `derivset`.

EndMonthRule

End-of-month rule. A `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

  Default: 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1 of logicals.

  Default: False

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with possible choices of business day convention:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

  Default: actual

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB date numbers.

Default: If no dates are specified, holidays.m is used.

Description   [Price, PriceTree] = fixedbybk(BKTree, CouponRate, Settle, Maturity) computes the price of a fixed-rate note from a Black-Karasinski tree.

[Price, PriceTree] = fixedbybk(BKTree, CouponRate, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule) computes the price of a fixed-rate note from a Black-Karasinski tree using optional input arguments.

[Price, PriceTree] = fixedbybk(BKTree, CouponRate, Settle, Maturity, Name,Value) computes the price of a price of a fixed-rate note from a Black-Karasinski interest-rate tree with additional options specified by one or more Name,Value pair arguments.

Price is an NINST-by-1 vector of expected prices of the fixed-rate note at time 0.

PriceTree is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PTree contains the clean prices.

PriceTree.AITree contains the accrued interest.

PriceTree.tObs contains the observation times.

The Settle date for every fixed-rate note is set to the ValuationDate of the BK tree. The fixed-rate note argument Settle is ignored.

Examples   Price a 5% fixed-rate note using a Black-Karasinski interest-rate tree.

Load the file deriv.mat, which provides BKTree. The BKTree structure contains the time and interest-rate information needed to price the note.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
CouponRate = 0.05;
Settle = '01-Jan-2005';
Maturity = '01-Jan-2006';
```

Use fixedbybk to compute the price of the note.

```
Price = fixedbybk(BKTree, CouponRate, Settle, Maturity)

Price =

  103.5126
```

See Also   bktree | bondbybk | capbybk | cfbybk | floatbybk | floorbybk | swapbybk

fixedbyhjm

Purpose   Price fixed-rate note from Heath-Jarrow-Morton interest-rate tree

Syntax   [Price, PriceTree] = fixedbyhjm(HJMTree, CouponRate, Settle,
                 Maturity)
                 [Price, PriceTree] = fixedbyhjm(HJMTree, CouponRate, Settle,
                 Maturity, Reset, Basis, Principal, Options, EndMonthRule)
                 [Price, PriceTree] = fixedbyhjm(HJMTree, CouponRate, Settle,
                 Maturity, Name,Value)

Input Arguments

HJMTree	Forward-rate tree structure created by hjmtree.
CouponRate	Decimal annual rate.
Settle	Settlement dates. Number of instruments (NINST)-by-1 vector of dates representing the settlement dates of the fixed-rate note.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the fixed-rate note.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

NINST-by-1 vector representing the frequency of payments per year.

> **Default:** 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

The notional principal amount.

> **Default:** 100

Options

Derivatives pricing options structure created with `derivset`.

EndMonthRule

End-of-month rule. A `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1 of logicals.

> **Default:** False

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with possible choices of business day convention:

- `actual`
- `follow`
- `modifiedfollow`
- `previous`
- `modifiedprevious`

> **Default:** actual

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, holidays.m is used.

Description   [Price, PriceTree] = fixedbyhjm(HJMTree, CouponRate, Settle, Maturity) computes the price of a fixed-rate note from a HJM forward-rate tree.

[Price, PriceTree] = fixedbyhjm(HJMTree, CouponRate, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule) computes the price of a fixed-rate note from a HJM forward-rate tree using optional input asrguments.

[Price, PriceTree] = fixedbyhjm(HJMTree, CouponRate, Settle, Maturity, Name,Value) computes the price of a price of a fixed-rate note from a HJM forward-rate tree with additional options specified by one or more Name,Value pair arguments.

Price is an NINST-by-1 vector of expected prices of the fixed-rate note at time 0.

PriceTree is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PBush contains the clean prices.

PriceTree.AIBush contains the accrued interest.

PriceTree.tObs contains the observation times.

The Settle date for every fixed-rate note is set to the ValuationDate of the HJM tree. The fixed-rate note argument Settle is ignored.

fixedbyhjm

Examples   Price a 4% fixed-rate note using an HJM forward-rate tree.

Load the file `deriv.mat`, which provides `HJMTree`. The `HJMTree` structure contains the time and forward-rate information needed to price the note.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
CouponRate = 0.04;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2003';
```

Use `fixedbyhjm` to compute the price of the note.

```
Price = fixedbyhjm(HJMTree, CouponRate, Settle, Maturity)

Price =

  98.7159
```

See Also   bondbyhjm | capbyhjm | cfbyhjm | floatbyhjm | floorbyhjm | hjmtree | swapbyhjm

Purpose   Price fixed-rate note from Hull-White interest-rate tree

Syntax   ```
[Price, PriceTree] = fixedbyhw(HWTree, CouponRate, Settle,
Maturity)
[Price, PriceTree] = fixedbyhw(HWTree, CouponRate, Settle,
Maturity, Reset, Basis, Principal, Options, EndMonthRule)
[Price, PriceTree] = fixedbyhw(HWTree, CouponRate, Settle,
Maturity, Name,Value)
```

Arguments

HWTree	Interest-rate tree structure created by hwtree.
CouponRate	Decimal annual rate.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the fixed-rate note.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the fixed-rate note.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

NINST-by-1 vector representing the frequency of payments per year.

> **Default:** 1

Basis

Day-count basis of the instrument. A vector of integers.

• 0 = actual/actual

- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

The notional principal amount.

> **Default:** 100

Options

Derivatives pricing options structure created with `derivset`.

EndMonthRule

End-of-month rule. A `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

   Default: 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

   Default: False

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- `actual`

- `follow`

- `modifiedfollow`

- `previous`

- `modifiedprevious`

   Default: actual

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, `holidays.m` is used.

Description   `[Price, PriceTree] = fixedbyhw(HWTree, CouponRate, Settle, Maturity)` computes the price of a fixed-rate note from a Hull-White tree.

`[Price, PriceTree] = fixedbyhw(HWTree, CouponRate, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule)` computes the price of a fixed-rate note from a Hull-White tree using optional input arguments.

`[Price, PriceTree] = fixedbyhw(HWTree, CouponRate, Settle, Maturity, Name,Value)` computes the price of a price of a fixed-rate note from a Hull-White tree with additional options specified by one or more `Name,Value` pair arguments.

`Price` is an `NINST`-by-1 vector of expected prices of the fixed-rate note at time 0.

`PriceTree` is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

`PriceTree.PTree` contains the clean prices.

`PriceTree.AITree` contains the accrued interest.

`PriceTree.tObs` contains the observation times.

The `Settle` date for every fixed-rate note is set to the `ValuationDate` of the HW tree. The fixed-rate note argument `Settle` is ignored.

Examples   Price a 5% fixed-rate note using a Hull-White interest-rate tree.

Load the file `deriv.mat`, which provides `HWTree`. The `HWTree` structure contains the time and interest-rate information needed to price the note.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
CouponRate = 0.05;
Settle = '01-Jan-2005';
Maturity = '01-Jan-2006';
```

Use `fixedbyhw` to compute the price of the note.

```
Price = fixedbyhw(HWTree, CouponRate, Settle, Maturity)

Price =

  103.5126
```

See Also   bondbyhw | capbyhw | cfbyhw | floatbyhw | floorbyhw | hwtree | swapbyhw

fixedbyzero

Purpose	Price fixed-rate note from set of zero curves

Syntax

```
[Price, PriceNoAI, CFlowAmounts, CFlowDates] =
fixedbyzero(RateSpec, CouponRate, Settle, Maturity)
[Price, PriceNoAI, CFlowAmounts, CFlowDates] =
fixedbyzero(RateSpec, CouponRate, Settle, Maturity,
Reset, Basis,  Principal, EndMonthRule)
[Price PriceNoAI, CFlowAmounts, CFlowDates] =
fixedbyzero(RateSpec, CouponRate, Settle, Maturity,
Name, Value)
```

Description

[Price, PriceNoAI, CFlowAmounts, CFlowDates] =
fixedbyzero(RateSpec, CouponRate, Settle, Maturity) computes
the price of a fixed-rate note from a set of zero curves. All inputs are
either scalars or NINST-by-1 vectors unless otherwise specified. Any
date may be a serial date number or date string. An optional argument
may be passed as an empty matrix [].

[Price, PriceNoAI, CFlowAmounts, CFlowDates] =
fixedbyzero(RateSpec, CouponRate, Settle, Maturity,
Reset, Basis, Principal, EndMonthRule) computes the price of a
fixed-rate note from a set of zero curves using optional input arguments.
All inputs are either scalars or NINST-by-1vectors unless otherwise
specified. Any date may be a serial date number or date string. An
optional argument may be passed as an empty matrix [].

[Price PriceNoAI, CFlowAmounts, CFlowDates] =
fixedbyzero(RateSpec, CouponRate, Settle, Maturity,
Name, Value) computes the price of a fixed-rate note from a set of zero
curves with additional options specified by one or more Name, Value
pair arguments.

Input Arguments

RateSpec

Structure containing the properties of an interest-rate structure. See
intenvset for information on creating RateSpec.

CouponRate

Decimal annual rate.

Settle

Settlement date. `Settle` must be either a scalar or `NINST`-by-1 vector of serial date numbers or date strings of the same value which represent the settlement date for each bond. `Settle` must be earlier than `Maturity`.

Maturity

Maturity date.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

`NINST`-by-1 vector representing the frequency of payments per year.

Default: 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

The notional principal amount.

> **Default:** 100

Options

Derivatives pricing options structure created with `derivset`.

EndMonthRule

End-of-month rule. A `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can

specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1
of logicals.

> **Default:** False

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with
possible choices of business day convention:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

> **Default:** actual

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB
date numbers.

> **Default:** If no dates are specified, holidays.m is used.

Output Arguments

Price

A number of instruments (NINST) by number of curves (NUMCURVES)
matrix of fixed-rate note prices. Each column arises from one of the
zero curves.

fixedbyzero

PriceNoAI

A NINST-by-NUMCURVES matrix of dirty bond price (clean + accrued interest). Each column arises from one of the zero curves.

CFlowAmounts

A NINST-by-NUMCFS matrix of cash flows for each bond.

CFlowDates

A NINST-by-NUMCFS matrix of payment dates for each bond.

Examples   Price a 4% fixed-rate note using a set of zero curves. Load the file deriv.mat, which provides ZeroRateSpec, the interest-rate term structure needed to price the note.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
CouponRate = 0.04;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2003';
```

Use fixedbyzero to compute the price of the note:

```
Price = fixedbyzero(ZeroRateSpec, CouponRate, Settle, Maturity)

Price =

  98.7159
```

See Also | bondbyzero | cfbyzero | floatbyzero | swapbyzero

Purpose   Price floating-rate note from Black-Derman-Toy interest-rate tree

Syntax
```
[Price, PriceTree] = floatbybdt(BDTTree, Spread,
Settle, Maturity)
[Price, PriceTree] = floatbybdt(BDTTree, Spread,
Settle, Maturity, Reset, Basis, Principal, Options,
EndMonthRule)
[Price, PriceTree] = floatbybdt(BDTTree, Spread, Settle,
Maturity,Name,Value)
```

Input Arguments

BDTTree	Interest-rate tree structure created by bdttree.
Spread	Number of instruments (NINST)-by-1 vector of number of basis points over the reference rate.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the floating-rate note.

> **Note** The Settle date for every floating-rate note is set to the ValuationDate of the BDT Tree. The floating-rate note argument Settle is ignored.

Maturity	NINST-by-1 vector of dates representing the maturity dates of the floating-rate note.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

NINST-by-1 vector representing the frequency of payments per year.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Default: 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

NINST-by-1 vector of notional principal amounts or NINST-by-1 cell array. For the latter case, each element of the cell array is a NumDates-by-2 matrix where the first column is dates and the second column is associated principal amount. The date indicates the last day that the principal value is valid.

> **Default:** 100

Options

Derivatives pricing options structure created with derivset.

EndMonthRule

End-of-month rule. NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** `false`

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- `actual`
- `follow`
- `modifiedfollow`
- `previous`
- `modifiedprevious`

> **Default:** `actual`

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, `holidays.m` is used.

CapRate

`NINST`-by-1 decimal annual rate or `NINST`-by-1 cell array, where each element is a `NumDates`-by-2 cell array, and the cell array first column is dates, and the second column is associated cap rates. The date indicates the last day that the cap rate is valid.

FloorRate

NINST-by-1 decimal annual rate or NINST-by-1 cell array, where each element is a NumDates-by-2 cell array, and the cell array first column is dates, and the second column is associated floor rates. The date indicates the last day that the floor rate is valid.

Description   [Price, PriceTree] = floatbybdt(BDTTree, Spread, Settle, Maturity) computes the price of a floating-rate note from a BDT tree.

[Price, PriceTree] = floatbybdt(BDTTree, Spread, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule) computes the price of a floating-rate note with optional inputs from a BDT tree.

[Price, PriceTree] = floatbybdt(BDTTree, Spread, Settle, Maturity,Name,Value) computes the price of a floating-rate note from a BDT tree with additional options specified by one or more Name,Value pair arguments..

Price is an NINST-by-1 vector of expected prices of the floating-rate note at time 0.

PriceTree is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PTree contains the clean prices.

PriceTree.AITree contains the accrued interest.

PriceTree.tObs contains the observation times.

The Settle date for every floating-rate note is set to the ValuationDate of the BDT tree. The floating-rate note argument Settle is ignored.

Examples   **Price a Floating-Rate Note Using a BDT Tree**

Price a 20-basis point floating-rate note using a BDT interest-rate tree.

Load the file deriv.mat, which provides BDTTree. The BDTTree structure contains the time and interest-rate information needed to price the note.

```
load deriv.mat;
```

Define the floating-rate note using the required arguments. Other arguments use defaults.

```
Spread = 20;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2003';
```

Use floatbybdt to compute the price of the note.

```
Price = floatbybk(BKTree, Spread, Settle, Maturity)

Price =

  100.4865
```

Price an Amortizing Floating-Rate Note

Price an amortizing floating-rate note using the Principal input argument to define the amortization schedule.

Create the RateSpec.

```
Rates = [0.03583; 0.042147; 0.047345; 0.052707; 0.054302];
ValuationDate = '15-Nov-2011';
StartDates = ValuationDate;
EndDates = {'15-Nov-2012';'15-Nov-2013';'15-Nov-2014' ;'15-Nov-2015';'15-Nov-2016'};
Compounding = 1;
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: [5x1 double]
           Rates: [5x1 double]
        EndTimes: [5x1 double]
```

```
      StartTimes: [5x1 double]
        EndDates: [5x1 double]
      StartDates: 734822
   ValuationDate: 734822
           Basis: 0
     EndMonthRule: 1
```

Create the floating-rate instrument using the following data:

```
Settle ='15-Nov-2011';
Maturity = '15-Nov-2015';
Spread = 15;
```

Define the floating-rate note amortizing schedule.

```
Principal ={{'15-Nov-2012' 100;'15-Nov-2013' 70;'15-Nov-2014' 40;'15-Nov-2015' 10}};
```

Build the BDT tree and assume volatility is 10%.

```
MatDates = {'15-Nov-2012'; '15-Nov-2013';'15-Nov-2014';'15-Nov-2015';'15-Nov-2016';'15-Nov-2017'};
BDTTimeSpec = bdttimespec(ValuationDate, MatDates);
Volatility = 0.10;
BDTVolSpec = bdtvolspec(ValuationDate, MatDates, Volatility*ones(1,length(MatDates))');
BDTT = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);
```

Compute the price of the amortizing floating-rate note.

```
Price = floatbybdt(BDTT, Spread, Settle, Maturity, 'Principal', Principal)

Price =

  100.3059
```

Price a Collar with a Floating-Rate Note

Price a collar with a floating-rate note using the CapRate and FloorRate input argument to define the collar pricing.

Create the RateSpec.

```
Rates = [0.0287; 0.03024; 0.03345; 0.03861; 0.04033];
ValuationDate = '1-April-2012';
StartDates = ValuationDate;
EndDates = {'1-April-2013';'1-April-2014';'1-April-2015' ;...
'1-April-2016';'1-April-2017'};
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);
```

Build the BDT tree and assume volatility is 5%.

```
MatDates = {'1-April-2013'; '1-April-2014';'1-April-2015';'1-April-2016';'1-April-2017';'1-April-2018'};
BDTTimeSpec = bdttimespec(ValuationDate, MatDates);
Volatility = 0.05;
BDTVolSpec = bdtvolspec(ValuationDate, MatDates, Volatility*ones(1,length(MatDates))');
BDTT = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);
```

Create the floating rate note instrument.

```
Settle ='1-April-2012';
Maturity = '1-April-2016';
Spread = 10;
Principal = 100;
```

Compute the price of a collared floating-rate note.

```
CapStrike = {{'1-April-2013' 0.03; '1-April-2015' 0.055}};
FloorStrike = {{'1-April-2013' 0.025; '1-April-2015' 0.04}};

Price = floatbybdt(BDTT, Spread, Settle, Maturity, 'CapRate',...
CapStrike, 'FloorRate', FloorStrike)

Price =

  101.2414
```

See Also   bdttree | bondbybdt | capbybdt | cfbybdt | fixedbybdt | floorbybdt | swapbybdt

floatbybk

Purpose	Price floating-rate note from Black-Karasinski interest-rate tree

Syntax

```
[Price, PriceTree] = floatbybk(BKTree, Spread,
Settle, Maturity)
[Price, PriceTree] = floatbybk(BKTree, Spread,
Settle, Maturity, Reset, Basis, Principal, Options,
EndMonthRule)
[Price, PriceTree] = floatbybk(BKTree, Spread, Settle,
Maturity,Name,Value)
```

Input Arguments

BKTree	Interest-rate tree structure created by bktree.
Spread	Number of instruments (NINST)-by-1 vector of number of basis points over the reference rate.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the floating-rate note.

> **Note** The Settle date for every floating-rate note is set to the ValuationDate of the BK tree. The floating-rate note argument Settle is ignored.

Maturity	NINST-by-1 vector of dates representing the maturity dates of the floating-rate note.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

NINST-by-1 vector representing the frequency of payments per year.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Default: 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

NINST-by-1 vector of notional principal amounts or NINST-by-1 cell array. For the latter case, each element of the cell array is a NumDates-by-2 matrix where the first column is dates and the second column is associated principal amount. The date indicates the last day that the principal value is valid.

> **Default:** 100

Options

Derivatives pricing options structure created with derivset.

EndMonthRule

End-of-month rule. NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** `false`

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- `actual`

- `follow`

- `modifiedfollow`

- `previous`

- `modifiedprevious`

> **Default:** `actual`

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, `holidays.m` is used.

CapRate

`NINST`-by-1 decimal annual rate or `NINST`-by-1 cell array, where each element is a `NumDates`-by-2 cell array, and the cell array first column is dates, and the second column is associated cap rates. The date indicates the last day that the cap rate is valid.

FloorRate

NINST-by-1 decimal annual rate or NINST-by-1 cell array, where each element is a NumDates-by-2 cell array, and the cell array first column is dates, and the second column is associated floor rates. The date indicates the last day that the floor rate is valid.

Description   [Price, PriceTree] = floatbybk(BKTree, Spread, Settle, Maturity) computes the price of a floating-rate note from a Black-Karasinski tree.

[Price, PriceTree] = floatbybk(BKTree, Spread, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule) computes the price of a floating-rate note with optional inputs from a Black-Karasinski tree.

[Price, PriceTree] = floatbybk(BKTree, Spread, Settle, Maturity,Name,Value) computes the price of a floating-rate note from a Black-Karasinski tree with additional options specified by one or more Name,Value pair arguments.

Price is an NINST-by-1 vector of expected prices of the floating-rate note at time 0.

PriceTree is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PTree contains the clean prices.

PriceTree.AITree contains the accrued interest.

PriceTree.tObs contains the observation times.

The Settle date for every floating-rate note is set to the ValuationDate of the BK tree. The floating-rate note argument Settle is ignored.

Examples   **Price a Floating-Rate Note Using a Black-Karasinski Tree**

Price a 20–basis point floating-rate note using a Black-Karasinski interest-rate tree.

Load the file deriv.mat, which provides BKTree. The BKTree structure contains the time and interest-rate information needed to price the note.

```
load deriv.mat;
```

Define the floating-rate note using the required arguments. Other arguments use defaults.

```
Spread = 20;
Settle = '01-Jan-2005';
Maturity = '01-Jan-2006';
```

Use `floatbybk` to compute the price of the note.

```
Price = floatbybk(BKTree, Spread, Settle, Maturity)

Price =

  100.3825
```

Price an Amortizing Floating-Rate Note

Price an amortizing floating-rate note using the `Principal` input argument to define the amortization schedule.

Create the `RateSpec`.

```
Rates = [0.03583; 0.042147; 0.047345; 0.052707; 0.054302];
ValuationDate = '15-Nov-2011';
StartDates = ValuationDate;
EndDates = {'15-Nov-2012';'15-Nov-2013';'15-Nov-2014' ;'15-Nov-2015';'15-Nov-2016'};
Compounding = 1;
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: [5x1 double]
           Rates: [5x1 double]
```

```
        EndTimes: [5x1 double]
      StartTimes: [5x1 double]
        EndDates: [5x1 double]
      StartDates: 734822
   ValuationDate: 734822
           Basis: 0
     EndMonthRule: 1
```

Create the floating-rate instrument using the following data:

```
Settle ='15-Nov-2011';
Maturity = '15-Nov-2015';
Spread = 15;
```

Define the floating-rate note amortizing schedule.

```
Principal ={{'15-Nov-2012' 100;'15-Nov-2013' 70;'15-Nov-2014' 40;'15-Nov-2015' 10}};
```

Build the BK tree and assume the volatility is 10%.

```
VolDates = ['15-Nov-2012'; '15-Nov-2013';'15-Nov-2014';'15-Nov-2015';'15-Nov-2016';'15-Nov-2017'];
VolCurve = 0.1;
AlphaDates = '15-Nov-2017';
AlphaCurve = 0.1;

BKVolSpec = bkvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RateSpec.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RateSpec, BKTimeSpec);
```

Compute the price of the amortizing floating-rate note.

```
Price = floatbybk(BKT, Spread, Settle, Maturity, 'Principal', Principal)

Price =

  100.3059
```

Price a Collar with a Floating-Rate Note

Price a collar with a floating-rate note using the `CapRate` and `FloorRate` input argument to define the collar pricing.

Price a portfolio of collared floating-rate notes using the following data:

```
Rates = [0.0287; 0.03024; 0.03345; 0.03861; 0.04033];
ValuationDate = '1-April-2012';
StartDates = ValuationDate;
EndDates = {'1-April-2013';'1-April-2014';'1-April-2015' ;...
'1-April-2016';'1-April-2017'};
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);
```

Build the BK tree and assume the volatility to be 5%.

```
VolDates = ['1-April-2013';'1-April-2014';'1-April-2015';'1-April-2016';...
'1-April-2017';'1-April-2018'];
VolCurve = 0.05;
AlphaDates = '15-Nov-2018';
AlphaCurve = 0.1;

BKVolSpec = bkvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RateSpec.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RateSpec, BKTimeSpec);
```

Create the floating-rate note instrument.

```
Settle ='1-April-2012';
Maturity = '1-April-2016';
Spread = [15;10];
Principal = 100;
```

Compute the price of the two vanilla floaters.

```
Price = floatbybk(BKT, Spread, Settle, Maturity)

Price =

  100.5519
  100.3680
```

Compute the price of the collared floating-rate notes.

```
CapStrike = {{'1-April-2013' 0.045; '1-April-2014' 0.05;...
'1-April-2015' 0.06}; 0.06};

FloorStrike = {{'1-April-2013' 0.035; '1-April-2014' 0.04;...
'1-April-2015' 0.05}; 0.03};
PriceCollared = floatbybk(BKT, Spread, Settle, Maturity,...
'CapRate', CapStrike,'FloorRate', FloorStrike)

PriceCollared =

  102.8537
  100.4910
```

See Also   bktree | bondbybk | capbybk | cfbybk | fixedbybk | floorbybk |
swapbybk

Purpose	Price floating-rate note from Heath-Jarrow-Morton interest-rate tree

Syntax
```
[Price, PriceTree] = floatbybk(HJMTree, Spread,
Settle, Maturity)
[Price, PriceTree] = floatbybk(HJMTree, Spread,
Settle, Maturity, Reset, Basis, Principal, Options,
EndMonthRule)
[Price, PriceTree] = floatbybk(HJMTree, Spread, Settle,
Maturity,Name,Value)
```

Input Arguments

HJMTree	Forward-rate tree structure created by hjmtree.
Spread	Number of instruments (NINST)-by-1 vector of number of basis points over the reference rate.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the floating-rate note.

> **Note** The Settle date for every floating-rate note is set to the ValuationDate of the HJM tree. The floating-rate note argument Settle is ignored.

Maturity	NINST-by-1 vector of dates representing the maturity dates of the floating-rate note.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

NINST-by-1 vector representing the frequency of payments per year.

> **Default:** 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

NINST-by-1 vector of notional principal amounts or NINST-by-1 cell array. For the latter case, each element of the cell array is a NumDates-by-2 matrix where the first column is dates and the second

column is associated principal amount. The date indicates the last day that the principal value is valid.

> **Default:** 100

Options

Derivatives pricing options structure created with `derivset`.

EndMonthRule

End-of-month rule. `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** false

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

Default: actual

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB date numbers.

Default: If no dates are specified, holidays.m is used.

CapRate

NINST-by-1 decimal annual rate or NINST-by-1 cell array, where each element is a NumDates-by-2 cell array, and the cell array first column is dates, and the second column is associated cap rates. The date indicates the last day that the cap rate is valid.

FloorRate

NINST-by-1 decimal annual rate or NINST-by-1 cell array, where each element is a NumDates-by-2 cell array, and the cell array first column is dates, and the second column is associated floor rates. The date indicates the last day that the floor rate is valid.

Description   [Price, PriceTree] = floatbybk(HJMTree, Spread, Settle, Maturity) computes the price of a floating-rate note from an HJM tree.

[Price, PriceTree] = floatbybk(HJMTree, Spread, Settle, Maturity, Reset, Basis, Principal, Options,

EndMonthRule) computes the price of a floating-rate note with optional inputs from an HJM tree.

`[Price, PriceTree] = floatbybk(HJMTree, Spread, Settle, Maturity,Name,Value)` computes the price of a floating-rate note from an HJM tree with additional options specified by one or more `Name,Value` pair arguments.

`Price` is an NINST-by-1 vector of expected prices of the floating-rate note at time 0.

`PriceTree` is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

`PriceTree.PBush` contains the clean prices.

`PriceTree.AIBush` contains the accrued interest.

`PriceTree.tObs` contains the observation times.

The `Settle` date for every floating-rate note is set to the `ValuationDate` of the HJM tree. The floating-rate note argument `Settle` is ignored.

Examples   **Price a Floating-Rate Note Using an HJM Tree**

Price a 20-basis point floating-rate note using an HJM forward-rate tree.

Load the file `deriv.mat`, which provides HJMTree. The HJMTree structure contains the time and interest-rate information needed to price the note.

```
load deriv.mat;
```

Define the floating-rate note using the required arguments. Other arguments use defaults.

```
Spread = 20;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2003';
```

Use `floatbyhjm` to compute the price of the note.

```
Price = floatbyhw(HJMTree, Spread, Settle, Maturity)

Price =

  100.5529
```

Price an Amortizing Floating-Rate Note

Price an amortizing floating-rate note using the `Principal` input argument to define the amortization schedule.

Create the `RateSpec`.

```
Rates = [0.03583; 0.042147; 0.047345; 0.052707; 0.054302];
ValuationDate = '15-Nov-2011';
StartDates = ValuationDate;
EndDates = {'15-Nov-2012';'15-Nov-2013';'15-Nov-2014' ;'15-Nov-2015';'15-Nov-2016'};
Compounding = 1;
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: [5x1 double]
           Rates: [5x1 double]
        EndTimes: [5x1 double]
      StartTimes: [5x1 double]
        EndDates: [5x1 double]
      StartDates: 734822
   ValuationDate: 734822
           Basis: 0
     EndMonthRule: 1
```

Create the floating-rate instrument using the following data:

```
Settle ='15-Nov-2011';
Maturity = '15-Nov-2015';
```

```
Spread = 15;
```

Define the floating-rate note amortizing schedule.

```
Principal ={{'15-Nov-2012' 100;'15-Nov-2013' 70;'15-Nov-2014' 40;'15-Nov-2015' 10}};
```

Build the HJM tree using the following data:

```
MatDates = {'15-Nov-2012'; '15-Nov-2013';'15-Nov-2014';'15-Nov-2015';'15-Nov-2016';'15-Nov-2017'};
HJMTimeSpec = hjmtimespec(RateSpec.ValuationDate, MatDates);
Volatility = [.10; .08; .06; .04];
CurveTerm = [ 1; 2; 3; 4];
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec,RateSpec,HJMTimeSpec);
```

Compute the price of the amortizing floating-rate note.

```
Price = floatbyhjm(HJMT, Spread, Settle, Maturity, 'Principal', Principal)
```

```
Price =

  100.3059
```

Price a Collar with a Floating-Rate Note

Price a collar with a floating-rate note using the `CapRate` and `FloorRate` input argument to define the collar pricing.

Price a portfolio of collared floating-rate notes using the following data:

```
Rates = [0.0287; 0.03024; 0.03345; 0.03861; 0.04033];
ValuationDate = '1-April-2012';
StartDates = ValuationDate;
EndDates = {'1-April-2013';'1-April-2014';'1-April-2015' ;...
'1-April-2016';'1-April-2017'};
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
```

```
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);
```

Build the HJM tree with the following data:

```
MatDates = {'1-April-2013'; '1-April-2014';'1-April-2015';...
'1-April-2016';'1-April-2017';'1-April-2018'};
HJMTimeSpec = hjmtimespec(RateSpec.ValuationDate, MatDates);
Volatility = [.10; .08; .06; .04];
CurveTerm = [ 1; 2; 3; 4];
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec,RateSpec,HJMTimeSpec);
```

Create the floating-rate note instrument.

```
Settle ='1-April-2012';
Maturity = '1-April-2016';
Spread = 10;
Principal = 100;
```

Compute the price of two capped collared floating-rate notes.

```
CapStrike = [0.04;0.055];
PriceCapped = floatbyhjm(HJMT, Spread, Settle, Maturity,...
'CapRate', CapStrike)

PriceCapped =

  98.9986
 100.2051
```

Compute the price of two collared floating-rate notes.

```
FloorStrike = [0.035;0.040];
PriceCollared = floatbyhjm(HJMT, Spread, Settle, Maturity,...
'CapRate', CapStrike, 'FloorRate', FloorStrike)

PriceCollared =

  99.9246
```

```
102.2321
```

See Also   bondbyhjm | capbyhjm | cfbyhjm | fixedbyhjm | floorbyhjm |
hjmtree | swapbyhjm

Purpose	Price floating-rate note from Hull-White interest-rate tree

Syntax

```
[Price, PriceTree] = floatbybk(HWTree, Spread,
Settle, Maturity)
[Price, PriceTree] = floatbybk(HWTree, Spread,
Settle, Maturity, Reset, Basis, Principal, Options,
EndMonthRule)
[Price, PriceTree] = floatbybk(HWTree, Spread, Settle,
Maturity,Name,Value)
```

Input Arguments

HWTree	Interest-rate tree structure created by hwtree.
Spread	Number of instruments (NINST)-by-1 vector of number of basis points over the reference rate.
Settle	Settlement dates. NINST-by-1 vector of dates representing the settlement dates of the floating-rate note.

Note The Settle date for every floating-rate note is set to the ValuationDate of the HW tree. The floating-rate note argument Settle is ignored.

Maturity	NINST-by-1 vector of dates representing the maturity dates of the floating-rate note.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

Reset

NINST-by-1 vector representing the frequency of payments per year.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Default: 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

NINST-by-1 vector of notional principal amounts or NINST-by-1 cell array. For the latter case, each element of the cell array is a NumDates-by-2 matrix where the first column is dates and the second column is associated principal amount. The date indicates the last day that the principal value is valid.

> **Default:** 100

Options

Derivatives pricing options structure created with derivset.

EndMonthRule

End-of-month rule. NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1 of logicals.

> **Default:** false

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with possible choices of business day convention:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

> **Default:** actual

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, holidays.m is used.

CapRate

NINST-by-1 decimal annual rate or NINST-by-1 cell array, where each element is a NumDates-by-2 cell array, and the cell array first column is dates, and the second column is associated cap rates. The date indicates the last day that the cap rate is valid.

FloorRate

floatbyhw

NINST-by-1 decimal annual rate or NINST-by-1 cell array, where each element is a NumDates-by-2 cell array, and the cell array first column is dates, and the second column is associated floor rates. The date indicates the last day that the floor rate is valid.

Description   [Price, PriceTree] = floatbybk(HWTree, Spread, Settle, Maturity) computes the price of a floating-rate note from a Hull-White tree.

[Price, PriceTree] = floatbybk(HWTree, Spread, Settle, Maturity, Reset, Basis, Principal, Options, EndMonthRule) computes the price of a floating-rate note with optional inputs from a Hull-White tree.

[Price, PriceTree] = floatbybk(HWTree, Spread, Settle, Maturity,Name,Value) computes the price of a floating-rate note from a Hull-White tree with additional options specified by one or more Name,Value pair arguments.

Price is an NINST-by-1 vector of expected prices of the floating-rate note at time 0.

PriceTree is a structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PTree contains the clean prices.

PriceTree.AITree contains the accrued interest.

PriceTree.tObs contains the observation times.

The Settle date for every floating-rate note is set to the ValuationDate of the HW tree. The floating-rate note argument Settle is ignored.

Examples   **Price a Floating-Rate Note Using a Hull-White Tree**

Price a 20-basis point floating-rate note using a Hull-White interest-rate tree.

Load the file deriv.mat, which provides HWTree. The HWTree structure contains the time and interest-rate information needed to price the note.

```
load deriv.mat;
```

Define the floating-rate note using the required arguments. Other arguments use defaults.

```
Spread = 20;
Settle = '01-Jan-2005';
Maturity = '01-Jan-2006';
```

Use `floatbyhw` to compute the price of the note.

```
Price = floatbyhw(HWTree, Spread, Settle, Maturity)

Price =

  100.3825
```

Price an Amortizing Floating-Rate Note

Price an amortizing floating-rate note using the `Principal` input argument to define the amortization schedule.

Create the `RateSpec`.

```
Rates = [0.03583; 0.042147; 0.047345; 0.052707; 0.054302];
ValuationDate = '15-Nov-2011';
StartDates = ValuationDate;
EndDates = {'15-Nov-2012';'15-Nov-2013';'15-Nov-2014' ;'15-Nov-2015';'15-Nov-2016'};
Compounding = 1;
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

           FinObj: 'RateSpec'
      Compounding: 1
             Disc: [5x1 double]
            Rates: [5x1 double]
         EndTimes: [5x1 double]
```

```
       StartTimes: [5x1 double]
         EndDates: [5x1 double]
       StartDates: 734822
    ValuationDate: 734822
            Basis: 0
      EndMonthRule: 1
```

Create the floating-rate instrument using the following data:

```
Settle ='15-Nov-2011';
Maturity = '15-Nov-2015';
Spread = 15;
```

Define the floating-rate note amortizing schedule.

```
Principal ={{'15-Nov-2012' 100;'15-Nov-2013' 70;'15-Nov-2014' 40;'15-Nov-2015' 10}};
```

Build the HW tree and assume the volatility is 10%.

```
VolDates = ['15-Nov-2012'; '15-Nov-2013';'15-Nov-2014';'15-Nov-2015';'15-Nov-2016';'15-Nov-2017'];
VolCurve = 0.1;
AlphaDates = '15-Nov-2017';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec);
```

Compute the price of the amortizing floating-rate note.

```
Price = floatbyhw(HWT, Spread, Settle, Maturity, 'Principal', Principal)

Price =

  100.3059
```

Price a Collar with a Floating-Rate Note

Price a collar with a floating-rate note using the `CapRate` and `FloorRate` input argument to define the collar pricing.

Price two collared floating-rate notes using the following data:

```
Rates = [0.0287; 0.03024; 0.03345; 0.03861; 0.04033];
ValuationDate = '1-April-2012';
StartDates = ValuationDate;
EndDates = {'1-April-2013';'1-April-2014';'1-April-2015' ;...
'1-April-2016';'1-April-2017'};
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);
```

Build the HW tree and assume the volatility to be 5%.

```
VolDates = ['1-April-2013';'1-April-2014';'1-April-2015';...
'1-April-2016';'1-April-2017';'1-April-2018'];
VolCurve = 0.05;
AlphaDates = '15-Nov-2018';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec);
```

Create the floating-rate note instrument.

```
Settle ='1-April-2012';
Maturity = '1-April-2016';
Spread = 10;
Principal = 100;
```

Compute the price of a vanilla floater.

```
Price = floatbyhw(HWT, Spread, Settle, Maturity)

Price =

 100.3680
```

Compute the price of the collared floating-rate notes.

```
CapStrike = {{'1-April-2014' 0.045; '1-April-2015' 0.05;...
 '1-April-2016' 0.06}; 0.06};

FloorStrike = {{'1-April-2014' 0.035; '1-April-2015' 0.04;...
 '1-April-2016' 0.05}; 0.03};
PriceCollared = floatbyhw(HWT, Spread, Settle, Maturity,....
'CapRate', CapStrike,'FloorRate', FloorStrike)

PriceCollared =

 102.0317
 100.9189
```

See Also
bondbyhw | capbyhw | cfbyhw | fixedbyhw | floorbyhw | hwtree | swapbyhw

Purpose	Price floating-rate note from set of zero curves

Syntax

```
[Price,PriceNoAI,OutputCashFlows,CFlowDates] =
floatbyzero(RateSpec,Spread,Settle,Maturity)
[Price,PriceNoAI,OutputCashFlows,CFlowDates] =
floatbyzero(RateSpec,Spread,Settle,Maturity,
Reset,Basis,Principal,EndMonthRule)
[Price,PriceNoAI,OutputCashFlows,CFlowDates] =
floatbyzero(RateSpec,Spread,Settle,Maturity,Name,Value)
```

Description

[Price,PriceNoAI,OutputCashFlows,CFlowDates] = floatbyzero(RateSpec,Spread,Settle,Maturity) computes the price of a floating-rate note from a set of zero curves.

[Price,PriceNoAI,OutputCashFlows,CFlowDates] = floatbyzero(RateSpec,Spread,Settle,Maturity, Reset,Basis,Principal,EndMonthRule) computes the price of a floating-rate note from a set of zero curves using optional input arguments.

[Price,PriceNoAI,OutputCashFlows,CFlowDates] = floatbyzero(RateSpec,Spread,Settle,Maturity,Name,Value) computes the price of a floating-rate note from a set of zero curves with additional options specified by one or more Name,Value pair arguments.

Input Arguments

RateSpec

Structure containing the properties of an interest-rate structure. See intenvset for information on creating RateSpec.

Spread

Number of basis points over the reference rate.

Settle

Settlement date. Settle must be either a scalar or NINST-by-1 vector of serial date numbers or date strings of the same value which represent

the settlement date for each bond. `Settle` must be earlier than
`Maturity`.

Maturity

Maturity date.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as
name-value pair arguments. You cannot mix ordered syntax with
name-value pair arguments.

Reset

`NINST`-by-1 vector representing the frequency of payments per year.

> **Default:** 1

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** `0` (actual/actual)

Principal

`NINST`-by-1 vector of notional principal amounts or `NINST`-by-1 cell array. For the latter case, each element of the cell array is a `NumDates`-by-`2` matrix where the first column is dates and the second column is associated principal amount. The date indicates the last day that the principal value is valid.

> **Default:** `100`

EndMonthRule

End-of-month rule. `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** `1`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1 of logicals.

> **Default:** false

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with possible choices of business day convention:

- actual
- follow
- modifiedfollow
- previous
- modifiedprevious

> **Default:** actual

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, holidays.m is used.

LatestFloatingRate

Rate for the next floating payment set at the last reset date. NINST-by-1 of scalars.

> **Default:** If no LatestFloatingRate is specified, the floating rate at the previous reset date is computed from the RateSpec.

ForwardRateSpec

The `RateSpec` to be used in generating floating cash flows. Note that spot rates should be used when specifying `ForwardRateSpec`.

> **Default:** If no `ForwardRateSpec` is specified, the `RateSpec` is used to generate floating cash flows.

Output Arguments

Price

Number of instruments (`NINST`) by number of curves (`NUMCURVES`) matrix of floating-rate note prices. Each column arises from one of the zero curves.

PriceNoAI

`NINST`-by-`NUMCURVES` matrix of dirty bond price (clean + accrued interest). Each column arises from one of the zero curves.

OutputCashFlows

`NINST`-by-`NUMCFS` matrix of cash flows for each bond.

Note If there is more than one curve specified in the `RateSpec` input, then the first `NCURVES` rows correspond to the first bond, the second `NCURVES` rows correspond to the second bond, and so on.

CFlowDates

`NINST`-by-`NUMCFS` matrix of payment dates for each bond.

Examples

Price a Floating-Rate Note Using a Set of Zero Curves

Price a 20-basis point floating-rate note using a set of zero curves.

Load `deriv.mat`, which provides `ZeroRateSpec`, the interest-rate term structure, needed to price the bond.

```
load deriv.mat;
```

Define the floating-rate note using the required arguments. Other arguments use defaults.

```
Spread = 20;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2003';
```

Use floatbyzero to compute the price of the note.

```
Price = floatbyzero(ZeroRateSpec, Spread, Settle, Maturity)

Price =

   100.5529
```

Price an Amortizing Floating-Rate Note

Price an amortizing floating-rate note using the Principal input argument to define the amortization schedule.

Create the RateSpec.

```
Rates = [0.03583; 0.042147; 0.047345; 0.052707; 0.054302];
ValuationDate = '15-Nov-2011';
StartDates = ValuationDate;
EndDates = {'15-Nov-2012';'15-Nov-2013';'15-Nov-2014' ;'15-Nov-2015';'15-Nov-2016'};
Compounding = 1;
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: [5x1 double]
           Rates: [5x1 double]
        EndTimes: [5x1 double]
      StartTimes: [5x1 double]
```

```
       EndDates: [5x1 double]
     StartDates: 734822
  ValuationDate: 734822
          Basis: 0
    EndMonthRule: 1
```

Create the floating-rate instrument using the following data:

```
Settle ='15-Nov-2011';
Maturity = '15-Nov-2015';
Spread = 15;
```

Define the floating-rate note amortizing schedule.

```
Principal ={{'15-Nov-2012' 100;'15-Nov-2013' 70;'15-Nov-2014' 40;'15-Nov-2015' 10}};
```

Compute the price of the amortizing floating-rate note.

```
Price  = floatbyzero(RateSpec, Spread, Settle, Maturity, 'Principal', Principal)
```

```
Price =

  100.3059
```

Specify the Rate at the Instrument's Starting Date When it Cannot be Obtained From the `RateSpec`

If `Settle` is not on a reset date of a floating-rate note, `floatbyzero` attempts to obtain the latest floating rate before `Settle` from `RateSpec` or the `LatestFloatingRate` parameter. When the reset date for this rate is out of the range of `RateSpec` (and `LatestFloatingRate` is not specified), `floatbyzero` fails to obtain the rate for that date and generates an error. This example shows how to use the `LatestFloatingRate` input parameter to avoid the error.

Create the error condition when a floating-rate instrument's `StartDate` cannot be determined from the `RateSpec`.

```
load deriv.mat;
```

```
Spread = 20;
Settle = '01-Jan-2000';
Maturity = '01-Dec-2003';

Price = floatbyzero(ZeroRateSpec, Spread, Settle, Maturity)


Error using floatbyzero (line 256)
The rate at the instrument starting date cannot be obtained from RateSpec.
 Its reset date (01-Dec-1999) is out of the range of dates contained in RateSpec.
 This rate is required to calculate cash flows at the instrument starting date.
 Consider specifying this rate with the 'LatestFloatingRate' input parameter.
```

Here, the reset date for the rate at `Settle` was 01-Dec-1999, which was earlier than the valuation date of `ZeroRateSpec` (01-Jan-2000). This error can be avoided by specifying the rate at the instrument's starting date using the `LatestFloatingRate` name-value pair argument.

Define `LatestFloatingRate` and calculate the floating-rate price.

```
Price = floatbyzero(ZeroRateSpec, Spread, Settle, Maturity, 'LatestFloatingRate', 0.03)

Price =

  100.0285
```

See Also | bondbyzero | cfbyzero | fixedbyzero | swapbyzero

Purpose	Price floor instrument from Black-Derman-Toy interest-rate tree
Syntax	[Price, PriceTree] = floorbybdt(BDTTree, Strike, Settle, Maturity, Reset, Basis, Principal, Options)

Arguments

BDTTree	Interest-rate tree structure created by bdttree.
Strike	Number of instruments (NINST)-by-1 vector of rates at which the floor is exercised.
Settle	Settlement date. NINST-by-1 vector of dates representing the settlement dates of the floor. The Settle date for every floor is set to the ValuationDate of the BDT tree. The floor argument Settle is ignored.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the floor.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal	(Optional) The notional principal amount. Default = `100`.
Options	(Optional) Derivatives pricing options structure created with `derivset`.

Description [Price, PriceTree] = floorbybdt(BDTTree, Strike, Settlement, Maturity, Reset, Basis, Principal, Options) computes the price of a floor instrument from a BDT interest-rate tree.

Price is an NINST-by-1 vector of the expected prices of the floor at time 0.

PriceTree is the tree structure with values of the floor at each node.

Examples **Example 1**. Price a 10% floor instrument using a BDT interest-rate tree.

Load the file `deriv.mat`, which provides BDTTree. BDTTree contains the time and interest-rate information needed to price the floor instrument.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
Strike = 0.10;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2004';
```

Use `floorbybdt` to compute the price of the floor instrument.

```
Price = floorbybdt(BDTTree, Strike, Settle, Maturity)

Price =

   0.2428
```

Example 2. Here is a second example, showing the pricing of a 10% floor instrument using a newly created BDT tree.

First set the required arguments for the three needed specifications.

```
Compounding = 1;
ValuationDate = '01-01-2000';
StartDate = ValuationDate;
EndDates = ['01-01-2001'; '01-01-2002'; '01-01-2003';
'01-01-2004'; '01-01-2005'];
Rates = [.1; .11; .12; .125; .13];
Volatility = [.2; .19; .18; .17; .16];
```

Next create the specifications.

```
RateSpec = intenvset('Compounding', Compounding,...
'ValuationDate', ValuationDate,...
'StartDates', StartDate,...
'EndDates', EndDates,...
'Rates', Rates);
BDTTimeSpec = bdttimespec(ValuationDate, EndDates, Compounding);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Volatility);
```

Now create the BDT tree from the specifications.

```
BDTTree = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);
```

Set the floor arguments. Remaining arguments will use defaults.

```
FloorStrike = 0.10;
Settlement = ValuationDate;
```

```
Maturity = '01-01-2002';
FloorReset = 1;
```

Finally, use `floorbybdt` to find the price of the floor instrument.

```
Price= floorbybdt(BDTTree, FloorStrike, Settlement, Maturity,...
FloorReset)

Price =

   0.0863
```

See Also   bdttree | capbybdt | cfbybdt | swapbybdt

Purpose   Price floor instrument from Black-Karasinski interest-rate tree

Syntax   [Price, PriceTree] = floorbybk(BKTree, Strike, Settle, Maturity, Reset, Basis, Principal, Options)

Arguments

BKTree	Interest-rate tree structure created by bktree.
Strike	Number of instruments (NINST)-by-1 vector of rates at which the floor is exercised.
Settle	Settlement date. NINST-by-1 vector of dates representing the settlement dates of the floor. The Settle date for every floor is set to the ValuationDate of the BK tree. The floor argument Settle is ignored.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the floor.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal   (Optional) The notional principal amount. Default = 100.

Options   (Optional) Derivatives pricing options structure created with `derivset`.

Description   `[Price, PriceTree] = floorbybk(BKTree, Strike, Settlement, Maturity, Reset, Basis, Principal, Options)` computes the price of a floor instrument from a Black-Karasinski tree.

`Price` is an `NINST`-by-1 vector of the expected prices of the floor at time 0.

`PriceTree` is the tree structure with values of the floor at each node.

Examples   Price a 3% floor instrument using a Black-Karasinski interest-rate tree.

Load the file `deriv.mat`, which provides `BKTree`. The `BKTree` structure contains the time and interest rate information needed to price the floor instrument.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
Strike = 0.03;
Settle = '01-Jan-2005';
Maturity = '01-Jan-2009';
```

Use `floorbyhw` to compute the price of the floor instrument.

```
Price = floorbybk(BKTree, Strike, Settle, Maturity)

Price =

  0.2061
```

See Also   bktree | capbybk | cfbybk | swapbybk

floorbyblk

Purpose Price floors using Black option pricing model

Syntax
```
[FloorPrice, Floorlets] = floorbyblk(RateSpec, Strike, Settle,
Maturity, Volatility)
[FloorPrice, Floorlets] = floorbyblk(RateSpec, Strike, Settle,
Maturity, Volatility, 'Name1', Value1...)
```

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For more information, see intenvset.
Strike	NINST-by-1 vector of rates at which the floor is exercised, as a decimal number.
Settle	Scalar representing the settle date of the floor.
Maturity	Scalar representing the maturity date of the floor.
Volatility	NINST-by-1 vector of volatilities.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default is 1.
Principal	(Optional) NINST-by-1 vector representing the notional principal amount. Default is 100.
Basis	NINST-by-1 vector representing the basis used when annualizing the input forward rate.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

ValuationDate (Optional) Scalar representing the observation date of the investment horizons. The default is the Settle date.

Note All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

Description   [FloorPrice, Floorlets] = floorbyblk(RateSpec, Strike, Settle, Maturity, Volatility)

[FloorPrice, Floorlets] = floorbyblk(RateSpec, Strike, Settle, Maturity, Volatility, 'Name1', Value1...)

Usefloorbyblk to price floors using the Black option pricing model.

The outputs are:

- `FloorPrice` — NINST-by-1 expected prices of the floor.

- `Floorlets` — NINST-by-NCF array of floorlets, padded with `NaN`s.

Examples

Consider an investor who gets into a contract that floors the interest rate on a $100,000 loan at 6% quarterly compounded for 3 months, starting on January 1, 2009'. Assuming that on January 1, 2008 the zero rate is 6.9394% continuously compounded and the volatility is 20%, use this data to compute the floor price.

Calculate the `RateSpec`:

```
ValuationDate = 'Jan-01-2008';
EndDates ='April-01-2010';
Rates = 0.069394;
Compounding = -1;
Basis = 1;

RateSpec = intenvset('ValuationDate', ValuationDate, ...
'StartDates', ValuationDate,'EndDates', EndDates, ...
'Rates', Rates,'Compounding', Compounding,'Basis', Basis);
```

Compute the price of the cap:

```
Settle = 'Jan-01-2009'; % floor starts in a year
Maturity = 'April-01-2009';
Volatility = 0.20;
FloorRate = 0.06;
FloorReset = 4;
Principal=100000;

FloorPrice = floorbyblk(RateSpec, FloorRate, Settle, Maturity, Volatility,...
'Reset',FloorReset,'ValuationDate',ValuationDate,'Principal', Principal,...
'Basis', Basis)

FloorPrice =

   37.4864
```

See Also   `capbyblk`

floorbyhjm

Purpose	Price floor instrument from Heath-Jarrow-Morton interest-rate tree

Syntax

```
[Price, PriceTree] = floorbyhjm(HJMTree, Strike, Settle, Maturity,
Reset, Basis, Principal, Options)
```

Arguments

HJMTree	Forward-rate tree structure created by hjmtree.
Strike	Number of instruments (NINST)-by-1 vector of rates at which the floor is exercised.
Settle	Settlement date. NINST-by-1 vector of dates representing the settlement dates of the floor. The Settle date for every floor is set to the ValuationDate of the HJM tree. The floor argument Settle is ignored.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the floor.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal	(Optional) The notional principal amount. Default = 100.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Price, PriceTree] = floorbyhjm(HJMTree, Strike, Settlement, Maturity, Reset, Basis, Principal, Options) computes the price of a floor instrument from an HJM tree.

Price is an NINST-by-1 vector of the expected prices of the floor at time 0.

PriceTree is the tree structure with values of the floor at each node.

Examples   Price a 3% floor instrument using an HJM forward-rate tree.

Load the file deriv.mat, which provides HJMTree. The HJMTree structure contains the time and forward-rate information needed to price the floor instrument.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
Strike = 0.03;
Settle = '01-Jan-2000';
Maturity = '01-Jan-2004';
```

floorbyhjm

Use `floorbyhjm` to compute the price of the floor instrument.

```
Price = floorbyhjm(HJMTree, Strike, Settle, Maturity)

Price =

  0.0486
```

See Also   capbyhjm | cfbyhjm | hjmtree | swapbyhjm

Purpose	Price floor instrument from Hull-White interest-rate tree
Syntax	[Price, PriceTree] = floorbyhw(HWTree, Strike, Settle, Maturity, Reset, Basis, Principal, Options)

Arguments

HWTree	Interest-rate tree structure created by hwtree.
Strike	Number of instruments (NINST)-by-1 vector of rates at which the floor is exercised.
Settle	Settlement date. NINST-by-1 vector of dates representing the settlement dates of the floor. The Settle date for every floor is set to the ValuationDate of the HW tree. The floor argument Settle is ignored.
Maturity	NINST-by-1 vector of dates representing the maturity dates of the floor.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal	(Optional) The notional principal amount. Default = 100.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description [Price, PriceTree] = floorbyhw(HWTree, Strike, Settlement, Maturity, Reset, Basis, Principal, Options) computes the price of a floor instrument from an HW tree.

Price is an NINST-by-1 vector of the expected prices of the floor at time 0.

PriceTree is the tree structure with values of the floor at each node.

Examples Price a 3% floor instrument using a Hull-White interest-rate tree.

Load the file deriv.mat, which provides HWTree. The HWTree structure contains the time and interest rate information needed to price the floor instrument.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
Strike = 0.03;
Settle = '01-Jan-2005';
Maturity = '01-Jan-2009';
```

Use floorbyhw to compute the price of the floor instrument.

```
Price = floorbyhw(HWTree, Strike, Settle, Maturity)

Price =

  0.4616
```

See Also   capbyhw | cfbyhw | hwtree | swapbyhw

floorbylg2f

Purpose	Price floor using Linear Gaussian two-factor model

Syntax

```
FloorPrice = floorbylg2f(ZeroCurve,a,b,sigma,eta,rho,Strike,
Maturity)
CapPrice = floorbylg2f( ___ ,Name,Value)
```

Description   `FloorPrice = floorbylg2f(ZeroCurve,a,b,sigma,eta,rho,Strike, Maturity)` returns the floor price for a two-factor additive Gaussian interest-rate model.

`CapPrice = floorbylg2f( ___ ,Name,Value)` returns the floor price for a two-factor additive Gaussian interest-rate model using optional name-value pairs.

Input Arguments

ZeroCurve - Zero curve for Linear Gaussian two-factor model
structure

Zero curve for the Linear Gaussian two-factor model, specified using `IRDataCurve` or `RateSpec`.

Data Types
struct

a - Mean reversion for first factor for Linear Gaussian two-factor model
scalar

Mean reversion for the first factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
single | double

b - Mean reversion for second factor for Linear Gaussian two-factor model
scalar

Mean reversion for the second factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
single | double

sigma - Volatility for first factor for Linear Gaussian two-factor model
scalar

Volatility for the first factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
single | double

eta - Volatility for second factor for Linear Gaussian two-factor model
scalar

Volatility for the second factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
single | double

rho - Scalar correlation of factors
scalar

Scalar correlation of the factors, specified as a scalar.

Data Types
single | double

Strike - Floor strike price
nonnegative integer | vector of nonnegative integers

Floor strike price specified, as a nonnegative integer using a NumFloors-by-1 vector of floor strike prices.

Data Types
single | double

Maturity - Floor maturity date
nonnegative integer | vector of nonnegative integers | string of dates

Floor maturity date, specified using a NumFloors-by-1 vector of serial date numbers or date strings.

Data Types
single | double | cell

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price = floorbylg2f(irdc,a,b,sigma,eta,rho,Strike,FloorMaturity,'Reset',1,'Notion

'Reset' - Frequency of floor payments per year
2 (default) | positive integer from the set [1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of floor payments per year, specified as positive integers for the values [1,2,4,6,12] in a NumFloors-by-1 vector.

Data Types
single | double

'Notional' - Notional value of floor
100 (default) | nonnegative integer | vector of nonnegative integers

Notional value of the floor, specified as a nonnegative integer using a NumFloors-by-1 vector of notional amounts.

Data Types
single | double

Output Arguments

FloorPrice - Floor price
scalar | vector

Floor price, returned as a scalar or a NumFloors-by-1 vector.

Algorithms

The following defines the two-factor additive Gaussian interest-rate model, given the ZeroCurve, a, b, sigma, eta, and rho parameters:

$$r(t) = x(t) + y(t) + \phi(t)$$

$$dx(t) = -a(x)(t)dt + \sigma(dW_1(t), x(0) = 0$$

$$dy(t) = -b(y)(t)dt + \eta(dW_2(t), y(0) = 0$$

where $dW_1(t)dW_2(t) = \rho dt$ is a two-dimensional Brownian motion with correlation ρ and φ is a function chosen to match the initial zero curve.

Examples

Price Floor Using a Linear Gaussian Two-Factor Model

Define the ZeroCurve, a, b, sigma, eta, and rho parameters to compute the floor price.

```
Settle = datenum('15-Dec-2007');

ZeroTimes = [3/12 6/12 1 5 7 10 20 30]';
ZeroRates = [0.033 0.034 0.035 0.040 0.042 0.044 0.048 0.0475]';
CurveDates = daysadd(Settle,360*ZeroTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

a = .07;
b = .5;
sigma = .01;
eta = .006;
rho = -.7;

FloorMaturity = daysadd(Settle,360*[1:5 7 10 15 20 25 30],1);
```

```
Strike = [0.035 0.037 0.038 0.039 0.040 0.042 0.044 0.046 0.047 0.047 0.0

Price = floorbylg2f(irdc,a,b,sigma,eta,rho,Strike,FloorMaturity)

Price =

   -0.0650
   -0.3530
   -0.8188
   -1.3493
   -1.9103
   -3.0893
   -4.8082
   -7.4953
   -9.8782
  -11.7803
  -12.9396
```

References

[1] Brigo, D. and F. Mercurio, *Interest Rate Models - Theory and Practice*, Springer Finance, 2006.

See Also   capbylg2f | swaptionbylg2fLinearGaussian2F |

Related Examples   • "Price Swaptions With Interest-Rate Models Using Simulation" on page 2-91

Purpose   Determine price of gap digital options using Black-Scholes model

Syntax   Price = gapbybls(RateSpec, StockSpec, Settle, Maturity,
                 OptSpec, Strike, StrikeThreshold)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of payoff strike price values.
StrikeThreshold	NINST-by-1 vector of strike values that determine if the option pays off.

Description Price = gapbybls(RateSpec, StockSpec, Settle, Maturity,
                 OptSpec, Strike, StrikeThreshold) computes gap option prices
                 using the Black-Scholes option pricing model.

                 Price is a NINST-by-1 vector of expected option prices.

Examples   Consider a gap call and put options on a nondividend paying stock with
                 a strike of 57 and expiring on January 1, 2008. On July 1, 2008 the stock
                 is trading at 50. Using this data, compute the price of the option if the
                 risk-free rate is 9%, the strike threshold is 50, and the volatility is 20%.

                 Create the RateSpec:

```
Settle = 'Jan-1-2008';
Maturity = 'Jul-1-2008';
```

```
Compounding = -1;
Rates = 0.09;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', 1);
```

Define the `StockSpec`:

```
AssetPrice = 50;
Sigma = .2;
StockSpec = stockspec(Sigma, AssetPrice);
```

Define the call and put options:

```
OptSpec = {'call'; 'put'};
Strike = 57;
StrikeThreshold = 50;
```

Calculate the price:

```
Pgap = gapbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec,...
Strike, StrikeThreshold)

Pgap =

   -0.0053
    4.4866
```

See Also   assetbybls | cashbybls | gapsensbybls | supersharebybls

Purpose   Determine price and sensitivities of gap digital options using Black-Scholes model

Syntax   PriceSens = gapsensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, StrikeThreshold)
PriceSens = gapsensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, StrikeThreshold, OutSpec)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
StrikeThreshold	NINST-by-1 vector of strike values that determine if the option pays off.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs may in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are:

- NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', or 'All'.

For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output should be Price, Lambda, and Rho, in that order.

To invoke from a function: [Price, Lambda, Rho] = gapsensbybls(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

- Default is OutSpec = {'Price'}.

Description   PriceSens = gapsensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, StrikeThreshold) computes gap option prices using the Black-Scholes option pricing model.

PriceSens = gapsensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, StrikeThreshold, OutSpec) includes an OutSpec argument defined as parameter/value pairs, and computes gap option prices and sensitivities using the Black-Scholes option pricing model.

PriceSens is a NINST-by-1 vector of expected option prices and sensitivities.

Examples   Consider a gap call and put options on a nondividend paying stock with a strike of 57 and expiring on January 1, 2008. On July 1, 2008 the stock is trading at 50. Using this data, compute the price and sensitivity of the option if the risk-free rate is 9%, the strike threshold is 50, and the volatility is 20%.

Create the RateSpec:

```
Settle = 'Jan-1-2008';
Maturity = 'Jul-1-2008';
Compounding = -1;
Rates = 0.09;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', 1);
```

Define the StockSpec:

```
AssetPrice = 50;
Sigma = .2;
StockSpec = stockspec(Sigma, AssetPrice);
```

Define the call and put options:

```
OptSpec = {'call'; 'put'};
Strike = 57;
StrikeThreshold = 50;
```

Calculate the price:

```
Pgap = gapbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec,...
Strike, StrikeThreshold)

Pgap =

   -0.0053
    4.4866
```

Compute the gamma and delta:

```
OutSpec = {'gamma'; 'delta'};
[Gamma ,Delta] = gapsensbybls(RateSpec, StockSpec, Settle, Maturity,...
OptSpec, Strike, StrikeThreshold, 'OutSpec', OutSpec)

Gamma =

    0.0724
```

gapsensbybls

```
          0.0724

     Delta =

          0.2852
         -0.7148
```

See Also   gapbybls

Purpose	Allocate optimal hedge for target costs or sensitivities

Syntax   [PortSens, PortCost, PortHolds] = hedgeopt(Sensitivities, Price, CurrentHolds, FixedInd, NumCosts, TargetCost, TargetSens, ConSet)

Arguments

Sensitivities	Number of instruments (NINST) by number of sensitivities (NSENS) matrix of dollar sensitivities of each instrument. Each row represents a different instrument. Each column represents a different sensitivity.
Price	NINST-by-1 vector of portfolio instrument unit prices.
CurrentHolds	NINST-by-1 vector of contracts allocated to each instrument.
FixedInd	(Optional) Number of fixed instruments (NFIXED)-by-1 vector of indices of instruments to hold fixed. For example, to hold the first and third instruments of a 10 instrument portfolio unchanged, set FixedInd = [1 3]. Default = [], no instruments held fixed.
NumCosts	(Optional) Number of points generated along the cost frontier when a vector of target costs (TargetCost) is not specified. The default is 10 equally spaced points between the point of minimum cost and the point of minimum exposure. When specifying TargetCost, enter NumCosts as an empty matrix [].

TargetCost	(Optional) Vector of target cost values along the cost frontier. If TargetCost is empty, or not entered, hedgeopt evaluates NumCosts equally spaced target costs between the minimum cost and minimum exposure. When specified, the elements of TargetCost should be positive numbers that represent the maximum amount of money the owner is willing to spend to rebalance the portfolio.
TargetSens	(Optional) 1-by-NSENS vector containing the target sensitivity values of the portfolio. When specifying TargetSens, enter NumCosts and TargetCost as empty matrices [].
ConSet	(Optional) Number of constraints (NCONS) by number of instruments (NINST) matrix of additional conditions on the portfolio reallocations. An eligible NINST-by-1 vector of contract holdings, PortWts, satisfies all the inequalities A*PortWts <= b, where A = ConSet(:,1:end-1) and b = ConSet(:,end).

Notes

The user-specified constraints included in `ConSet` may be created with the functions `pcalims` or `portcons`. However, the `portcons` default `PortHolds` positivity constraints are typically inappropriate for hedging problems since short-selling is usually required.

`NPOINTS`, the number of rows in `PortSens` and `PortHolds` and the length of `PortCost`, is inferred from the inputs. When the target sensitivities, `TargetSens`, is entered, `NPOINTS = 1`; otherwise `NPOINTS = NumCosts`, or is equal to the length of the `TargetCost` vector.

Not all problems are solvable (for example, the solution space may be infeasible or unbounded, or the solution may fail to converge). When a valid solution is not found, the corresponding rows of `PortSens` and `PortHolds` and the elements of `PortCost` are padded with `NaNs` as placeholders.

Description   `[PortSens, PortCost, PortHolds] = hedgeopt(Sensitivities, Price, CurrentHolds, FixedInd, NumCosts, TargetCost, TargetSens, ConSet)` allocates an optimal hedge by one of two criteria:

- Minimize portfolio sensitivities (exposure) for a given set of target costs.

- Minimize the cost of hedging a portfolio given a set of target sensitivities.

Hedging involves the fundamental tradeoff between portfolio insurance and the cost of insurance coverage. This function lets investors modify portfolio allocations among instruments to achieve either of the criteria. The chosen criterion is inferred from the input argument list. The problem is cast as a constrained linear least-squares problem.

`PortSens` is a number of points (`NPOINTS`)-by-`NSENS` matrix of portfolio sensitivities. When a perfect hedge exists, `PortSens` is zeros. Otherwise, the best hedge possible is chosen.

hedgeopt

PortCost is a 1-by-NPOINTS vector of total portfolio costs.

PortHolds is an NPOINTS-by-NINST matrix of contracts allocated to each instrument. These are the reallocated portfolios.

See Also   hedgeslf | pcalims | portcons | portopt | lsqlin

Purpose   Self-financing hedge

Syntax   [PortSens, PortValue, PortHolds] = hedgeslf(Sensitivities,
                   Price, CurrentHolds, FixedInd, ConSet)

Arguments

Sensitivities	Number of instruments (NINST) by number of sensitivities (NSENS) matrix of dollar sensitivities of each instrument. Each row represents a different instrument. Each column represents a different sensitivity.
Price	NINST-by-1 vector of instrument unit prices.
CurrentHolds	NINST-by-1 vector of contracts allocated in each instrument.
FixedInd	(Optional) Empty or number of fixed instruments (NFIXED)-by-1 vector of indices of instruments to hold fixed. The default is FixedInd = 1; the holdings in the first instrument are held fixed. If NFIXED instruments will not be changed, enter all their locations in the portfolio in a vector. If no instruments are to be held fixed, enter FixedInd = [].
ConSet	(Optional) Number of constraints (NCONS)-by-NINST matrix of additional conditions on the portfolio reallocations. An eligible NINST-by-1 vector of contract holdings, PortHolds, satisfies all the inequalities A*PortHolds <= b, where A = ConSet(:,1:end-1) and b = ConSet(:,end).

hedgeslf

Description   [PortSens, PortValue, PortHolds] = hedgeslf(Sensitivities, Price, CurrentHolds, FixedInd, ConSet) allocates a self-financing hedge among a collection of instruments. hedgeslf finds the reallocation in a portfolio of financial instruments that hedges the portfolio against market moves and that is closest to being self-financing (maintaining constant portfolio value). By default the first instrument entered is hedged with the other instruments.

PortSens is a 1-by-NSENS vector of portfolio dollar sensitivities. When a perfect hedge exists, PortSens is zeros. Otherwise, the best possible hedge is chosen.

PortValue is the total portfolio value (scalar). When a perfectly self-financing hedge exists, PortValue is equal to dot(Price, CurrentWts) of the initial portfolio.

PortHolds is an NINST-by-1 vector of contracts allocated to each instrument. This is the reallocated portfolio.

Notes

- The constraints PortHolds(FixedInd) = CurrentHolds(FixedInd) are appended to any constraints passed in ConSet. Pass FixedInd = [] to specify all constraints through ConSet.

- The default constraints generated by portcons are inappropriate, since they require the sum of all holdings to be positive and equal to one.

- hedgeself first tries to find the allocations of the portfolio that make it closest to being self-financing, while reducing the sensitivities to 0. If no solution is found, it finds the allocations that minimize the sensitivities. If the resulting portfolio is self-financing, PortValue is equal to the value of the original portfolio.

Examples   **Example 1**. Perfect sensitivity cannot be reached.

```
Sens = [0.44  0.32; 1.0 0.0];
Price = [1.2; 1.0];
WO = [1; 1];
[PortSens, PortValue, PortHolds]= hedgeslf(Sens, Price, WO)

PortSens =

    0.0000
    0.3200

PortValue =

    0.7600

PortHolds =

    1.0000
   -0.4400
```

Example 2. Constraints are in conflict.

```
Sens = [0.44  0.32; 1.0 0.0];
Price = [1.2; 1.0];
WO = [1; 1];
ConSet = pcalims([2 2])

% O.K. if nothing fixed.

[PortSens, PortValue, PortHolds]= hedgeslf(Sens, Price, WO,...
[], ConSet)

PortSens =

    2.8800
    0.6400
```

```
PortValue =

    4.4000

PortHolds =

     2
     2

% WO(1) is not greater than 2.

[PortSens, PortValue, PortHolds] = hedgeslf(Sens, Price, WO,...
1, ConSet)

??? Error using ==> hedgeslf
Overly restrictive allocation constraints implied by ConSet and
by fixing the weight of instruments(s): 1
```

Example 3. Constraints are impossible to meet.

```
Sens = [0.44  0.32; 1.0 0.0];
Price = [1.2; 1.0];
WO = [1; 1];
ConSet = pcalims([2 2],[1 1]);

[PortSens, PortValue, PortHolds] = hedgeslf(Sens, Price, WO,...
[],ConSet)

??? Error using ==> hedgeslf
Overly restrictive allocation constraints specified in ConSet
```

See Also   hedgeopt | lsqlin | portcons

hjmprice

Purpose	Instrument prices from Heath-Jarrow-Morton interest-rate tree
Syntax	Price = hjmprice(HJMTree, InstSet, Options)

Arguments

HJMTree	Heath-Jarrow-Morton tree sampling a forward-rate process. See hjmtree for information on creating HJMTree.
InstSet	Variable containing a collection of instruments. Instruments are categorized by type. Each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description Price = hjmprice(HJMTree, InstSet, Options) computes arbitrage-free prices for instruments using an interest-rate tree created with hjmtree. A subset of NINST instruments from a financial instrument variable, InstSet, are priced.

Price is a NINST-by-1 vector of prices for each instrument. The prices are computed by backward dynamic programming on the interest-rate tree. If an instrument cannot be priced, NaN is returned.

PriceTree is a MATLAB structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PBush contains the clean prices.

PriceTree.AIBush contains the accrued interest.

PriceTree.tObs contains the observation times.

hjmprice handles instrument types: 'Bond', 'CashFlow', 'OptBond', 'OptEmBond', 'OptEmBond', 'OptFloat', 'OptEmFloat', 'Fixed',

hjmprice

'Float', 'Cap', 'Floor', 'RangeFloat', 'Swap'. See instadd to construct defined types.

Related single-type pricing functions are:

- bondbyhjm: Price a bond from an HJM tree.
- capbyhjm: Price a cap from an HJM tree.
- cfbyhjm: Price an arbitrary set of cash flows from an HJM tree.
- fixedbyhjm: Price a fixed-rate note from an HJM tree.
- floatbyhjm: Price a floating-rate note from an HJM tree.
- floorbyhjm: Price a floor from an HJM tree.
- optbndbyhjm: Price a bond option from an HJM tree.
- optembndbyhjm: Price a bond with embedded option by an HJM tree.
- optfloatbybdt: Price a floating-rate note with an option from an HJM tree.
- optemfloatbybdt: Price a floating-rate note with an embedded option from an HJM tree.
- rangefloatbyhjm: Price range floating note using an HJM tree.
- swapbyhjm: Price a swap from an HJM tree.
- swaptionbyhjm: Price a swaption from an HJM tree.

Examples   Load the HJM tree and instruments from the data file deriv.mat. Price the cap and bond instruments contained in the instrument set.

```
load deriv.mat;
HJMSubSet = instselect(HJMInstSet,'Type', {'Bond', 'Cap'});

instdisp(HJMSubSet)
```

```
%Table of instrument portfolio partially displayed:
Index Type CouponRate Settle      Maturity    Period Basis ... Name     Quantity
1     Bond 0.04       01-Jan-2000 01-Jan-2003 1         NaN  ... 4% bond 100
```

```
2      Bond   0.04        01-Jan-2000 01-Jan-2004 2        NaN ... 4% bond   50

Index Type Strike Settle      Maturity    CapReset Basis ... Name     Quantity
3      Cap   0.03   01-Jan-2000 01-Jan-2004 1        NaN   ...   3% Cap   30
```

```
[Price, PriceTree] = hjmprice(HJMTree, HJMSubSet)
```

```
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

   98.7159
   97.5280
    6.2831

PriceTree =

    FinObj: 'HJMPriceTree'
     PBush: {[3x1 double]  [3x1x2 double]  [3x2x2 double]  [3x4x2 double]  [3x8 double]}
    AIBush: {[3x1 double]  [3x1x2 double]  [3x2x2 double]  [3x4x2 double]  [3x8 double]}
      tObs: [0 1 2 3 4]
```

You can use `treeviewer` to see the prices of these three instruments along the price tree.

```
treeviewer(PriceTree, HJMSubSet)
```
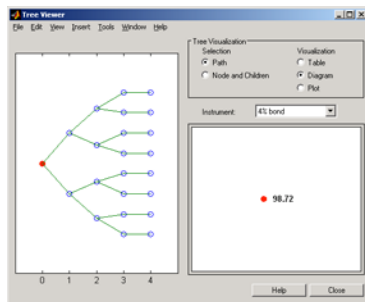
First 4% Bond (Maturity 2003)
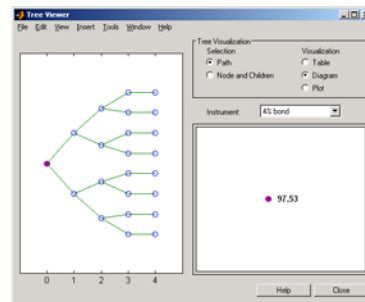
Second 4% Bond (Maturity 2004)

3% Cap

Price the following multi-stepped coupon bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

% Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);
```

```
% Create a portfolio of stepped coupon bonds with different maturities
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};

ISet = instbond(CouponRate, Settle, Maturity, 1);
instdisp(ISet)

%Table of instrument portfolio partially displayed:
Index Type CouponRate Settle         Maturity    Period Basis EndMonthRule ... Face
1    Bond [Cell]    01-Jan-2010   01-Jan-2011   1      0     1            ... 100
2    Bond [Cell]    01-Jan-2010   01-Jan-2012   1      0     1            ... 100
3    Bond [Cell]    01-Jan-2010   01-Jan-2013   1      0     1            ... 100
4    Bond [Cell]    01-Jan-2010   01-Jan-2014   1      0     1            ... 100
```

```
% Build the tree with the following data
Volatility = [.2; .19; .18; .17];
CurveTerm = [ 1;  2;   3;   4];
HJMTimeSpec = hjmtimespec(ValuationDate, EndDates);
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec,RS,HJMTimeSpec);

% Compute the price of the  stepped coupon bonds
PHJM = hjmprice(HJMT, ISet)

PHJM =

  100.6763
  100.7368
  100.9266
  101.0115
```

Price a portfolio of stepped callable bonds and stepped vanilla bonds using the following data:

```
% Price a portfolio of stepped callable bonds and stepped vanilla bonds
% using the following data.

% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

%Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create an instrument portfolio of 3 stepped callable bonds and three
% stepped vanilla bonds
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};
OptSpec='call';
Strike=100;
ExerciseDates='01-Jan-2011'; %Callable in one year

% Bonds with embedded option
ISet = instoptembnd(CouponRate, Settle, Maturity, OptSpec, Strike,...
ExerciseDates, 'Period', 1);

% Vanilla bonds
ISet = instbond(ISet, CouponRate, Settle, Maturity, 1);

% Display the instrument portfolio
instdisp(ISet)

%Table of instrument portfolio partially displayed:
Index Type     CouponRate Settle       Maturity   OptSpec Strike ExerciseDates ... AmericanOpt
1   OptEmBond [Cell]      01-Jan-2010  01-Jan-2012  call    100    01-Jan-2011 ... 0
```

| 2 | OptEmBond [Cell] | 01-Jan-2010 | 01-Jan-2013 | call | 100 | 01-Jan-2011 ... 0 |
| 3 | OptEmBond [Cell] | 01-Jan-2010 | 01-Jan-2014 | call | 100 | 01-Jan-2011 ... 0 |

Index	Type	CouponRate Settle	Maturity	Period	Basis	EndMonthRule ... Face
4	Bond [Cell]	01-Jan-2010	01-Jan-2012	1	0	1 ... 100
5	Bond [Cell]	01-Jan-2010	01-Jan-2013	1	0	1 ... 100
6	Bond [Cell]	01-Jan-2010	01-Jan-2014	1	0	1 ... 100

```
% Build the tree with the following data
Volatility = [.2; .19; .18; .17];
CurveTerm = [ 1;  2;   3;    4];
HJMTimeSpec = hjmtimespec(ValuationDate, EndDates);
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec,RS,HJMTimeSpec);

%The first three rows corresponds to the price of the stepped callable bonds
% and the last three rows corresponds to the price of the stepped vanilla bonds.
PHJM = hjmprice(HJMT, ISet)

PHJM =

  100.3682
  100.1557
   99.9232
  100.7368
  100.9266
  101.0115
```

Compute the price of a portfolio using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

%  Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create an instrument portfolio with two range notes and a floating rate
% note with the following data:
Spread = 200;
Settle = 'Jan-1-2011';
Maturity = 'Jan-1-2014';

% First Range Note:
RateSched(1).Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
RateSched(1).Rates  = [0.045 0.055; 0.0525  0.0675; 0.06 0.08];

% Second Range Note:
RateSched(2).Dates = {'Jan-1-2012'; 'Jan-1-2013' ; 'Jan-1-2014'};
RateSched(2).Rates  = [0.048 0.059; 0.055  0.068 ; 0.07 0.09];

% Create InstSet
InstSet = instadd('RangeFloat', Spread, Settle, Maturity, RateSched);

% Add a floating-rate note
InstSet = instadd(InstSet, 'Float', Spread, Settle, Maturity);

% Display the portfolio instrument
instdisp(InstSet)

Index Type      Spread Settle      Maturity   RateSched FloatReset Basis Principal EndMonthRule
1     RangeFloat 200   01-Jan-2011 01-Jan-2014 [Struct] 1          0     100       1
```

| 2 | RangeFloat | 200 | 01-Jan-2011 | 01-Jan-2014 | [Struct] | 1 | 0 | 100 | 1 |

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	EndMonthRule
3	Float	200	01-Jan-2011	01-Jan-2014	1	0	100	1

```
% The data to build the tree is as follows:
Volatility = [.2; .19; .18; .17];
CurveTerm = [ 1;  2;   3;   4];
MaTree = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
HJMTS = hjmtimespec(ValuationDate, MaTree);
HJMVS = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVS, RS, HJMTS);

% Price the portfolio
Price = hjmprice(HJMT, InstSet)

Price =

   91.1555
   90.6656
  105.5147
```

See Also   hjmsens | hjmtree | hjmvolspec | instadd | intenvprice |
                 intenvsens

hjmsens

Purpose	Instrument prices and sensitivities from Heath-Jarrow-Morton interest-rate tree

Syntax

[Delta, Gamma, Vega, Price] = hjmsens(HJMTree, InstSet, Options)

Arguments

HJMTree	Heath-Jarrow-Morton tree sampling a forward-rate process. See hjmtree for information on creating HJMTree.
InstSet	Variable containing a collection of instruments. Instruments are categorized by type. Each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Delta, Gamma, Vega, Price] = hjmsens(HJMTree, InstSet, Options) computes instrument sensitivities and prices for instruments using an interest-rate tree created with hjmtree. NINST instruments from a financial instrument variable, InstSet, are priced. hjmsens handles instrument types: 'Bond', 'CashFlow', 'OptBond', 'OptEmBond', 'OptEmBond', 'OptFloat', 'OptEmFloat', 'Fixed', 'Float', 'Cap', 'Floor', 'RangeFloat', 'Swap'. See instadd for information on instrument types.

Delta is an NINST-by-1 vector of deltas, representing the rate of change of instrument prices with respect to changes in the interest rate. Delta is computed by finite differences in calls to hjmtree. See hjmtree for information on the observed yield curve.

Gamma is an NINST-by-1 vector of gammas, representing the rate of change of instrument deltas with respect to the changes in the interest rate. Gamma is computed by finite differences in calls to hjmtree.

Vega is an NINST-by-1 vector of vegas, representing the rate of change of instrument prices with respect to the changes in the volatility

$\sigma(t,T)$. Vega is computed by finite differences in calls to hjmtree. See hjmvolspec for information on the volatility process.

Note All sensitivities are returned as dollar sensitivities. To find the per-dollar sensitivities, divide by the respective instrument price.

Price is an NINST-by-1 vector of prices of each instrument. The prices are computed by backward dynamic programming on the interest-rate tree. If an instrument cannot be priced, NaN is returned.

Delta and Gamma are calculated based on yield shifts of 100 basis points. Vega is calculated based on a 1% shift in the volatility process.

Examples   Load the tree and instruments from a data file. Compute Delta and Gamma for the cap and bond instruments contained in the instrument set.

```
load deriv.mat;
HJMSubSet = instselect(HJMInstSet,'Type', {'Bond', 'Cap'});
instdisp(HJMSubSet)

%Table of instrument portfolio partially displayed:
Index Type   CouponRate Settle      Maturity    Period ...   Name
1     Bond   0.04       01-Jan-2000 01-Jan-2003 1      ...    4% bond
2     Bond   0.04       01-Jan-2000 01-Jan-2004 2      ...    4% bond

Index Type Strike Settle      Maturity    CapReset ...   Name ...
3     Cap  0.03   01-Jan-2000 01-Jan-2004 1        ...   3% Cap ...

[Delta, Gamma] = hjmsens(HJMTree, HJMSubSet)

Warning: Not all cash flows are aligned with the tree. Result will
be approximated.
```

```
Delta =

 -272.6462
 -347.4315
  294.9700

Gamma =

  1.0e+003 *

    1.0299
    1.6227
    6.8526
```

See Also   hjmprice | hjmtree | hjmvolspec | instadd

Purpose   Specify time structure for Heath-Jarrow-Morton interest-rate tree

Syntax   TimeSpec = hjmtimespec(ValuationDate, Maturity, Compounding)

Arguments

ValuationDate	Scalar date marking the pricing date and first observation in the tree. Specify as serial date number or date string.
Maturity	Number of levels (depth) of the tree. A number of levels (NLEVELS)-by-1 vector of dates marking the cash flow dates of the tree. Cash flows with these maturities fall on tree nodes. Maturity should be in increasing order.
Compounding	(Optional) Scalar value representing the rate at which the input zero rates were compounded when annualized. Default = 1. This argument determines the formula for the discount factors:

Compounding = 1, 2, 3, 4, 6, 12

Disc = $(1 + Z/F)^{\wedge}(-T)$, where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.

Compounding = 365

Disc = $(1 + Z/F)^{\wedge}(-T)$, where F is the number of days in the basis year and T is a number of days elapsed computed by basis.

Compounding = -1

Disc = $\exp(-T*Z)$, where T is time in years.

hjmtimespec

Description   TimeSpec = hjmtimespec(ValuationDate, Maturity, Compounding) sets the number of levels and node times for an HJM tree and determines the mapping between dates and time for rate quoting.

TimeSpec is a structure specifying the time layout for hjmtree. The state observation dates are [Settle; Maturity(1:end-1)]. Because a forward rate is stored at the last observation, the tree can value cash flows out to Maturity.

Examples   Specify an eight-period tree with semiannual nodes (every six months). Use exponential compounding to report rates.

```
Compounding = -1;
ValuationDate = '15-Jan-1999';
Maturity = datemnth(ValuationDate, 6*(1:8)');
TimeSpec = hjmtimespec(ValuationDate, Maturity, Compounding)


TimeSpec =

            FinObj: 'HJMTimeSpec'
     ValuationDate: 730135
          Maturity: [8x1 double]
       Compounding: -1
             Basis: 0
       EndMonthRule: 1
```

See Also   hjmtree | hjmvolspec

Purpose   Construct Heath-Jarrow-Morton interest-rate tree

Syntax   HJMTree = hjmtree(VolSpec, RateSpec, TimeSpec)

Arguments

VolSpec	Volatility process specification. Sets the number of factors and the rules for computing the volatility $\sigma(t, T)$ for each factor. See hjmvolspec for information on the volatility process.
RateSpec	Interest-rate specification for the initial rate curve. See intenvset for information on declaring an interest-rate variable.
TimeSpec	Tree time layout specification. Defines the observation dates of the HJM tree and the compounding rule for date to time mapping and price-yield formulas. See hjmtimespec for information on the tree structure.

Description   HJMTree = hjmtree(VolSpec, RateSpec, TimeSpec) creates a structure containing time and forward-rate information on a bushy tree.

Examples   Using the data provided, create an HJM volatility specification (VolSpec), rate specification (RateSpec), and tree time layout specification (TimeSpec). Then use these specifications to create an HJM tree using hjmtree.

```
Compounding = 1;
ValuationDate = '01-01-2000';
StartDate = ['01-01-2000'; '01-01-2001'; '01-01-2002'; '01-01-2003'; '01-01-2004'];
EndDates = ['01-01-2001'; '01-01-2002'; '01-01-2003'; '01-01-2004'; '01-01-2005'];
Rates = [.1; .11; .12; .125; .13];
Volatility = [.2; .19; .18; .17; .16];
CurveTerm = [1; 2; 3; 4; 5];
```

```
HJMVolSpec = hjmvolspec('Stationary', Volatility , CurveTerm);

RateSpec = intenvset('Compounding', Compounding,...
          'ValuationDate', ValuationDate,...
          'StartDates', StartDate,...
          'EndDates', EndDates,...
          'Rates', Rates);

HJMTimeSpec = hjmtimespec(ValuationDate, EndDates, Compounding);
HJMTree = hjmtree(HJMVolSpec, RateSpec, HJMTimeSpec)

HJMTree =

     FinObj: 'HJMFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 1 2 3 4]
       dObs: [730486 730852 731217 731582 731947]
        TFwd: {[5x1 double]  [4x1 double]  [3x1 double]  [2x1 double]  [4]}
      CFlowT: {[5x1 double]  [4x1 double]  [3x1 double]  [2x1 double]  [5]}
     FwdTree: {[5x1 double]  [4x1x2 double]  [3x2x2 double]  [2x4x2 double]  [1x8x2 double]}
```

Use `treeviewer` to observe the tree you have created.

```
treeviewer(HJMTree)
```
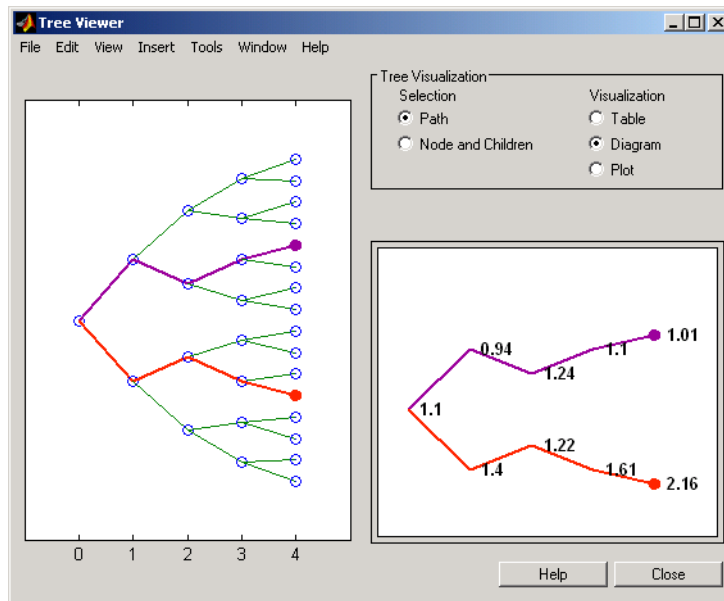
See Also   hjmprice | hjmtimespec | hjmvolspec | intenvset

hjmvolspec

Purpose	Specify Heath-Jarrow-Morton interest-rate volatility process
Syntax	Volspec = hjmvolspec(varargin)

Arguments

The arguments to `hjmvolspec` vary according to the type and number of volatility factors specified when calling the function. Factors are specified by pairs of names and parameter sets. Factor names can be `'Constant'`, `'Stationary'`, `'Exponential'`, `'Vasicek'`, or `'Proportional'`. The parameter set is specific for each of these factor types:

- Constant volatility (Ho-Lee):
  VolSpec = hjmvolspec('Constant', Sigma_0)

- Stationary volatility:
  VolSpec = hjmvolspec('Stationary', CurveVol, CurveTerm)

- Exponential volatility:
  VolSpec = hjmvolspec('Exponential', Sigma_0, Lambda)

- Vasicek, Hull-White:
  VolSpec = hjmvolspec('Vasicek', Sigma_0, CurveDecay, CurveTerm)

- Nearly proportional stationary:
  VolSpec = hjmvolspec('Proportional', CurveProp, CurveTerm, MaxSpot)

You can specify more than one factor by concatenating names and parameter sets.

The following table defines the various arguments to `hjmvolspec`.

Argument	Description
Sigma_0	Scalar base volatility over a unit time.
Lambda	Scalar decay factor.
CurveVol	Number of curves (NCURVES)-by1 vector of Vol values at sample points.

Argument	Description
CurveDecay	NCURVES-by-1 vector of Decay values at sample points.
CurveProp	NCURVES-by-1 vector of Prop values at sample points.
CurveTerm	NCURVES-by-1 vector of Term sample points.

Note See the volatility specifications formulas below for a description of Vol, Decay, Prop, and Term.

Description
Volspec = hjmvolspec(varargin) computes VolSpec, a structure that specifies the volatility model for hjmtree.

hjmvolspec specifies an HJM forward-rate volatility process. Each factor is specified with one of the functional forms.

Volatility Specification	Formula
Constant	$\sigma(t,T) = \text{Sigma\_0}$
Stationary	$\sigma(t,T) = \text{Vol}(\text{T-t}) = \text{Vol}(\text{Term})$
Exponential	$\sigma(t,T) = \text{Sigma\_0*exp(-Lambda*(T-t))}$
Vasicek, Hull-White	$\sigma(t,T) = \text{Sigma\_0*exp(-Decay(T-t))}$
Proportional	$\sigma(t,T) = \text{Prop(T-t)*max(SpotRate(t), MaxSpot)}$

hjmvolspec

The volatility process is $\sigma(t, T)$, where t is the observation time and T is the starting time of a forward rate. In a stationary process, the volatility term is T–t. Multiple factors can be specified sequentially.

The time values T, t, and Term are in coupon interval units specified by the Compounding input of hjmtimespec. For instance if Compounding = 2, Term = 1 is a semiannual period (six months).

Examples   **Example 1**. Volatility is single-factor proportional.

```
CurveProp = [0.11765; 0.08825; 0.06865];
CurveTerm = [1; 2; 3];
VolSpec = hjmvolspec('Proportional', CurveProp, CurveTerm, 1e6)


 VolSpec =
           FinObj: 'HJMVolSpec'
      FactorModels: {'Proportional'}
        FactorArgs: {{1x3 cell}}
        SigmaShift: 0
        NumFactors: 1
         NumBranch: 2
           PBranch: [0.5000 0.5000]
        Fact2Branch: [-1 1]
```

Example 2. Volatility is two-factor exponential and constant.

```
VolSpec = hjmvolspec('Exponential', 0.1, 1, 'Constant', 0.2)


VolSpec =
           FinObj: 'HJMVolSpec'
      FactorModels: {'Exponential'  'Constant'}
        FactorArgs: {{1x2 cell}  {1x1 cell}}
        SigmaShift: 0
        NumFactors: 2
         NumBranch: 3
           PBranch: [0.2500 0.2500 0.5000]
        Fact2Branch: [2x3 double]
```

See Also   hjmtimespec | hjmtree

HullWhite1F

Superclasses

Purpose   Create Hull-White one-factor model

Description   The Hull-White one-factor model is specified using the zero curve, alpha, and sigma parameters for the equation

$$dr = [\theta(t) - a(t)r]dt + \sigma(t)dW$$

where:

dr is the change in the short-term interest rate over a small interval.

r is the short-term interest rate.

Θ(t) is a function of time determining the average direction in which *r* moves, chosen such that movements in *r* are consistent with today's zero coupon yield curve.

a is the mean reversion rate.

dt is a small change in time.

σ is the annual standard deviation of the short rate.

W is the Brownian motion.

Construction   OBJ = HullWhite1F(ZeroCurve,alpha,sigma) constructs an object for a Hull-White one-factor model.

For example:

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

alpha = .1;
sigma = .01;
```

```
HW1F = HullWhite1F(irdc,alpha,sigma);
```

Properties   The following properties are from the HullWhite1F class.

ZeroCurve

ZeroCurve is specified using the output from IRDataCurve or RateSpec. This is the zero curve used to evolve the path of future interest rates.

Attributes:

SetAccess	public
GetAccess	public

Alpha

Mean reversion specified either as a scalar or function handle which takes time as an input and returns a scalar mean reversion value.

Attributes:

SetAccess	public
GetAccess	public

Sigma

Volatility specified either as a scalar or function handle which takes time as an input and returns a scalar mean volatility.

Attributes:

SetAccess	public
GetAccess	public

HullWhite1F

Methods

simTermStructs	Simulate term structures for Hull-White one-factor model

Definitions   **Hull-White One-Factor Model**

The Hull-White model is a single-factor, no-arbitrage yield curve model in which the short-term rate of interest is the random factor or state variable. No-arbitrage means that the model parameters are consistent with the bond prices implied in the zero coupon yield curve.

Copy Semantics   Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB documentation.

Examples   **Construct a Hull-White One-Factor Model**

Construct a Hull-White one-factor model.

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

alpha = .1;
sigma = .01;

HW1F = HullWhite1F(irdc,alpha,sigma)

HW1F =

  HullWhite1F with properties:

    ZeroCurve: [1x1 IRDataCurve]
        Alpha: @(t,V)inAlpha
        Sigma: @(t,V)inSigma
```

Use the `simTermStructs` method with the `HullWhite1F` model to simulate term structures.

```
SimPaths = HW1F.simTermStructs(10,'nTrials',100)

SimPaths(:,:,1) =

    0.0100    0.0180    0.0240    0.0290    0.0330    0.0340    0.0350
    0.0200    0.0276    0.0330    0.0375    0.0408    0.0409    0.0409
    0.0199    0.0275    0.0329    0.0373    0.0403    0.0403    0.0403
    0.0234    0.0308    0.0359    0.0401    0.0428    0.0424    0.0421
    0.0226    0.0299    0.0350    0.0392    0.0415    0.0412    0.0409
    0.0277    0.0346    0.0393    0.0434    0.0452    0.0444    0.0436
    0.0249    0.0316    0.0365    0.0406    0.0422    0.0415    0.0410
    0.0311    0.0372    0.0417    0.0455    0.0465    0.0455    0.0444
    0.0326    0.0383    0.0427    0.0464    0.0470    0.0458    0.0446
    0.0276    0.0333    0.0378    0.0416    0.0421    0.0413    0.0406
    0.0306    0.0359    0.0401    0.0437    0.0438    0.0428    0.0418

.
.
.

SimPaths(:,:,100) =

    0.0100    0.0180    0.0240    0.0290    0.0330    0.0340    0.0350
    0.0204    0.0280    0.0334    0.0378    0.0411    0.0412    0.0412
    0.0234    0.0308    0.0360    0.0403    0.0432    0.0429    0.0426
    0.0209    0.0284    0.0336    0.0380    0.0407    0.0405    0.0404
    0.0204    0.0278    0.0330    0.0374    0.0398    0.0396    0.0395
    0.0200    0.0272    0.0323    0.0367    0.0388    0.0385    0.0385
    0.0259    0.0325    0.0373    0.0414    0.0430    0.0423    0.0417
    0.0241    0.0306    0.0354    0.0395    0.0407    0.0402    0.0398
    0.0217    0.0280    0.0328    0.0370    0.0380    0.0376    0.0374
    0.0235    0.0294    0.0341    0.0381    0.0387    0.0382    0.0379
    0.0213    0.0270    0.0317    0.0356    0.0361    0.0358    0.0356
```

HullWhite1F

References   Hull, J. *Options, Futures, and Other Derivatives*, Prentice-Hall, 2011.

See Also   LiborMarketModel **|** LinearGaussian2F **|** hwcalbycap |
                      hwcalbyfloorsimTermStructs **|**

**Related   • "Price Swaptions With Interest-Rate Models Using Simulation" on
Examples**   page 2-91
                     • Class Attributes
                     • Property Attributes

Purpose	Simulate term structures for Hull-White one-factor model

Syntax

```
[ZeroRates,ForwardRates] = simTermStructs(nPeriods)
[ZeroRates,ForwardRates] = simTermStructs(nPeriods,
Name,Value)
```

Description [ZeroRates,ForwardRates] = simTermStructs(nPeriods) simulates future zero curve paths using a specified HullWhite1F object.

[ZeroRates,ForwardRates] = simTermStructs(nPeriods, Name,Value) simulates future zero curve paths using a specified HullWhite1F object with additional options specified by one or more Name,Value pair arguments.

Input Arguments

nPeriods

Number of simulation periods.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'deltaTime'

Scalar time step between periods.

Default: 1

'nTrials'

Scalar number of trials.

Default: 1

'antithetic'

Boolean scalar flag indicating whether antithetic sampling is used to generate the Gaussian random variates that drive the zero-drift, unit-variance rate Brownian vector $dW(t)$. For details, see `simBySolution`.

Default: false

'Z'

Direct specification of the dependent random noise process used to generate the zero-drift, unit-variance rate Brownian vector $dW(t)$ that drives the simulation. For details, see `simBySolution` for the HWV model.

Default: Uses default for `simBySolution`. If you do not specify a value for Z, `simBySolution` generates Gaussian variates.

'Tenor'

Numeric vector of maturities to compute at each time step.

Default: tenor of the `HullWhite1F` object's zero curve

Output Arguments

ZeroRates

nPeriods+1-by-nTenors-by-nTrials matrix of simulated zero-rate term structures.

ForwardRates

nPeriods+1-by-nTenors-by-nTrials matrix of simulated forward-rate term structures.

Attributes

Access	public
Static	false
Hidden	false

To learn about attributes of methods, see Method Attributes in the MATLAB Object-Oriented Programming documentation.

Examples   **Simulate Term Structures for the HullWhite1F Model**

Create a HW1F object.

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

alpha = .1;
sigma = .01;

HW1F = HullWhite1F(irdc,alpha,sigma)

HW1F =

  HullWhite1F with properties:

    ZeroCurve: [1x1 IRDataCurve]
        Alpha: @(t,V)inAlpha
        Sigma: @(t,V)inSigma
```

Simulate the term structures for the specified HW1F object.

```
SimPaths = HW1F.simTermStructs(10,'nTrials',100)

SimPaths(:,:,1) =

    0.0100    0.0180    0.0240    0.0290    0.0330    0.0340    0.0350    0.0340
    0.0179    0.0256    0.0311    0.0356    0.0390    0.0393    0.0395    0.0369
    0.0159    0.0237    0.0292    0.0338    0.0370    0.0373    0.0377    0.0356
    0.0176    0.0253    0.0307    0.0352    0.0380    0.0381    0.0382    0.0359
    0.0182    0.0257    0.0310    0.0354    0.0379    0.0379    0.0380    0.0357
```

```
       0.0209     0.0281     0.0332     0.0375     0.0396     0.0392     0.0391     0.0364
       0.0251     0.0318     0.0366     0.0407     0.0423     0.0416     0.0411     0.0377
       0.0235     0.0300     0.0349     0.0390     0.0403     0.0397     0.0394     0.0364
       0.0219     0.0282     0.0330     0.0371     0.0382     0.0378     0.0376     0.0351
       0.0242     0.0301     0.0347     0.0387     0.0393     0.0387     0.0383     0.0356
       0.0250     0.0305     0.0350     0.0388     0.0392     0.0385     0.0381     0.0354

.
.
.

SimPaths(:,:,100) =

       0.0100     0.0180     0.0240     0.0290     0.0330     0.0340     0.0350     0.0340
       0.0229     0.0303     0.0356     0.0400     0.0432     0.0431     0.0428     0.0392
       0.0208     0.0284     0.0337     0.0381     0.0411     0.0411     0.0410     0.0379
       0.0217     0.0292     0.0344     0.0387     0.0414     0.0412     0.0410     0.0378
       0.0222     0.0295     0.0346     0.0389     0.0413     0.0409     0.0407     0.0375
       0.0215     0.0287     0.0337     0.0380     0.0401     0.0397     0.0395     0.0367
       0.0217     0.0285     0.0335     0.0378     0.0395     0.0391     0.0389     0.0362
       0.0214     0.0280     0.0329     0.0371     0.0385     0.0381     0.0379     0.0355
       0.0216     0.0278     0.0327     0.0368     0.0379     0.0375     0.0373     0.0350
       0.0203     0.0263     0.0311     0.0352     0.0360     0.0358     0.0357     0.0338
       0.0181     0.0239     0.0287     0.0328     0.0334     0.0333     0.0335     0.0322
```

See Also   HullWhite1F |

Related Examples   • "Price Swaptions With Interest-Rate Models Using Simulation" on page 2-91

Purpose   Calibrate Hull-White tree using caps

Syntax   [Alpha, Sigma, OptimOut] = hwcalbycap(RateSpec, MarketStrike,
                MarketMaturity, MarketVolatility, Strike, Settle, Maturity)
                [Alpha, Sigma, OptimOut] = hwcalbycap(RateSpec, MarketStrike,
                MarketMaturity, MarketVolatility, Strike, Settle, Maturity,
                'Name1', Value1...)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For more information, see intenvset.
MarketStrike	NSTRIKES-by-1 vector of market cap strikes, as a decimal number.
MarketMaturity	NMATS-by-1 vector of market cap maturity dates.
MarketVolatility	NSTRIKES-by-NMATS matrix of market flat volatilities, where NSTRIKES is the number of caplet strikes from MarketStrike and NMATS is the caplet maturity dates from MarketMaturity.
Strike	Scalar representing the rate at which the cap is exercised, as a decimal number.
Settle	Scalar representing the settle date of the cap.
Maturity	Scalar representing the maturity date of the cap.
Reset	(Optional) Scalar representing the frequency of payments per year. Default is 1.

Principal	(Optional) Scalar representing the notional principal amount. Default is `100`.
Basis	(Optional) NINST-by-1 vector representing the basis used when annualizing the input forward rate.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

ValuationDate	(Optional) Scalar representing the observation date of the investment horizons. The default is the `Settle` date.

LB	(Optional) 2-by-1 vector of the lower bounds, defined as [LBSigma; LBAlpha], used in the search algorithm function. Default is LB =[0;0]. For more information, see lsqnonlin.
UB	(Optional) 2-by-1 vector of the upper bounds, defined as [UBSigma; LBAlpha], used in the search algorithm function. Default is UB =[ ](unbound). For more information, see lsqnonlin.
X0	(Optional) 2-by-1 vector of the initial values, defined as [Sigma0; Alpha0], used in the search algorithm function. Default is X0 = [0.5;0.5]. For more information, see lsqnonlin.
OptimOptions	(Optional) Structure with optimization parameters. For more information, see optimoptions.

Note All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

Description   [Alpha, Sigma, OptimOut] = hwcalbycap(RateSpec, MarketStrike,
MarketMaturity, MarketVolatility, Strike, Settle, Maturity)

[Alpha, Sigma, OptimOut] = hwcalbycap(RateSpec, MarketStrike,

MarketMaturity, MarketVolatility, Strike, Settle, Maturity,
'Name1', Value1...)

Use hwcalbycap to estimate the Alpha (mean reversion) and Sigma
(volatility) using cap market data and the Hull-White model.

The outputs are:

- Alpha — Scalar representing the mean reversion value obtained from
  calibrating the cap using market information.

- Sigma — Scalar representing the volatility value obtained from
  calibrating the cap using market information.

- OptimOut — Structure with optimization results.

Examples   For an example, see "Calibrating Hull-White Model Using Market
Data" on page 2-79.

See Also   capbyblk | hwcalbyfloor | hwtree | lsqnonlin | HullWhite1F

Purpose   Calibrate Hull-White tree using floors

Syntax   [Alpha, Sigma, OptimOut] = hwcalbyfloor(RateSpec,
                MarketStrike, MarketMaturity, MarketVolatility, Strike,
                Settle, Maturity)
                [Alpha, Sigma, OptimOut] = hwcalbyfloor(RateSpec,
                MarketStrike, MarketMaturity, MarketVolatility, Strike,
                Settle, Maturity, 'Name1', Value1...)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For more information, see intenvset.
MarketStrike	NSTRIKES-by-1 vector of market floor strikes as a decimal number.
MarketMaturity	NMATS-by-1 vector of market floor maturity dates.
MarketVolatility	NSTRIKES-by-NMATS matrix of market flat volatilities.
Strike	Scalar representing the rate at which the floor is exercised, as a decimal number.
Settle	Scalar representing the settle date of the floor.
Maturity	Scalar representing the maturity date of the floor.
Reset	(Optional) Scalar representing the frequency of payments per year. Default is 1.

Principal	(Optional) Scalar representing the notional principal amount. Default is `100`.
Basis	(Optional) `NINST`-by-1 vector representing the basis used when annualizing the input forward rate.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

ValuationDate	(Optional) Scalar representing the observation date of the investment horizons. The default is the `Settle` date.

LB	(Optional) 2-by-1 vector of the lower bounds, defined as [LBSigma; LBAlpha], used in the search algorithm function. Default is LB =[0;0]. For more information, see lsqnonlin.
UB	(Optional) 2-by-1 vector of the upper bounds, defined as [UBSigma; LBAlpha], used in the search algorithm function. Default is UB =[ ](unbound). For more information, see lsqnonlin.
X0	(Optional) 2-by-1 vector of the initial values, defined as [Sigma0; Alpha0], used in the search algorithm function. Default is X0 = [0.5;0.5]. For more information, see lsqnonlin.
OptimOptions	(Optional) Structure with optimization parameters. For more information, see optimoptions.

Note All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

Description   [Alpha, Sigma, OptimOut] = hwcalbyfloor(RateSpec, MarketStrike, MarketMaturity, MarketVolatility, Strike, Settle, Maturity)

hwcalbyfloor

```
[Alpha, Sigma, OptimOut] = hwcalbyfloor(RateSpec,
MarketStrike, MarketMaturity, MarketVolatility, Strike,
Settle, Maturity, 'Name1', Value1...)
```

Use hwcalbyfloor to estimate the Alpha (mean reversion) and Sigma (volatility) using floor market data and the Hull-White model.

The outputs are:

- Alpha — Scalar representing the mean reversion value obtained from calibrating the floor using market information.

- Sigma — Scalar representing the volatility value obtained from calibrating the floor using market information.

- OptimOut — Structure with optimization results.

Examples   For an example, see "Calibrating Hull-White Model Using Market Data" on page 2-79.

See Also   floorbyblk | hwcalbycap | hwtree | lsqnonlin | HullWhite1F

hwprice

Purpose	Instrument prices from Hull-White interest-rate tree

Syntax   [Price, PriceTree] = hwprice(HWTree, InstSet, Options)

Arguments

HWTree	Interest-rate tree structure created by hwtree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Price, PriceTree] = hwprice(HWTree, InstSet, Options) computes arbitrage-free prices for instruments using an interest-rate tree created with hwtree. All instruments contained in a financial instrument variable, InstSet, are priced.

Price is a number of instruments (NINST)-by-1 vector of prices for each instrument. The prices are computed by backward dynamic programming on the interest-rate tree. If an instrument cannot be priced, NaN is returned.

PriceTree is a MATLAB structure of trees containing vectors of instrument prices and accrued interest, and a vector of observation times for each node.

PriceTree.PTree contains the clean prices.

PriceTree.AITree contains the accrued interest.

PriceTree.tObs contains the observation times.

hwprice handles instrument types: 'Bond', 'CashFlow', 'OptBond', 'OptEmBond', 'OptEmBond', 'OptFloat', 'OptEmFloat', 'Fixed',

hwprice

'Float', 'Cap', 'Floor', 'RangeFloat', 'Swap'. See instadd to construct defined types.

Related single-type pricing functions are:

- bondbyhw: Price a bond from a Hull-White tree.
- capbyhw: Price a cap from a Hull-White tree.
- cfbyhw: Price an arbitrary set of cash flows from a Hull-White tree.
- fixedbyhw: Price a fixed-rate note from a Hull-White tree.
- floatbyhw: Price a floating-rate note from a Hull-White tree.
- floorbyhw: Price a floor from a Hull-White tree.
- optbndbyhw: Price a bond option from a Hull-White tree.
- optembndbyhw: Price a bond with embedded option by a Hull-White tree.
- optfloatbybdt: Price a floating-rate note with an option from a Hull-White tree.
- optemfloatbybdt: Price a floating-rate note with an embedded option from a Hull-White tree.
- rangefloatbyhw: Price range floating note using a Hull-White tree.
- swapbyhw: Price a swap from a Hull-White tree.
- swaptionbyhw: Price a swaption from a Hull-White tree.

Examples   Load the HW tree and instruments from the data file deriv.mat. Price the cap and bond instruments contained in the instrument set.

```
load deriv.mat;
HWSubSet = instselect(HWInstSet,'Type', {'Bond', 'Cap'});

instdisp(HWSubSet)

%Table of instrument portfolio partially displayed:
Index Type   CouponRate Settle      Maturity     Period ...   Name ...
```

```
1      Bond   0.04        01-Jan-2004 01-Jan-2007 1        ...  4% bond
2      Bond   0.04        01-Jan-2004 01-Jan-2008 1        ...  4% bond

Index Type Strike Settle      Maturity     CapReset ...  Name ...
3      Cap  0.06   01-Jan-2004 01-Jan-2008 1        ...  6% Cap

[Price, PriceTree] = hwprice(HWTree, HWSubSet);

Price =

  100.9188
   99.3296
    0.5837
```

You can use treeviewer to see the prices of these three instruments along the price tree.

```
treeviewer(PriceTree, HWSubSet)
```
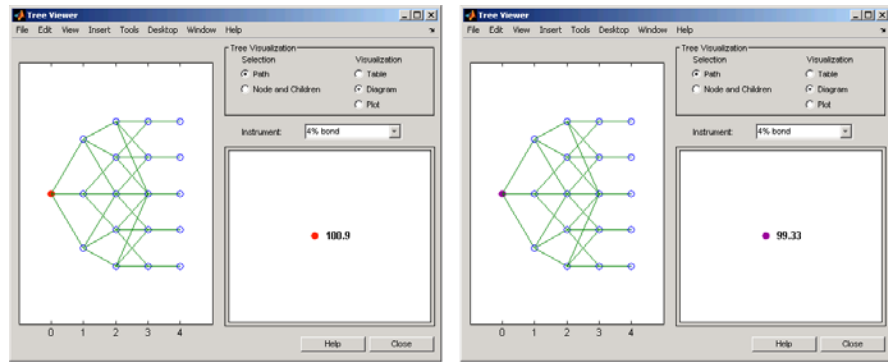
First 4% Bond (Maturity 2007)

Second 4% Bond (Maturity 2008)

6% Cap

Price the following multi-stepped coupon bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

% Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create a portfolio of stepped coupon bonds with different maturities
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};

ISet = instbond(CouponRate, Settle, Maturity, 1);
instdisp(ISet)

%Table of instrument portfolio partially displayed:
Index Type CouponRate Settle          Maturity    Period Basis EndMonthRule ... Face
1     Bond [Cell]     01-Jan-2010     01-Jan-2011  1      0     1            ... 100
2     Bond [Cell]     01-Jan-2010     01-Jan-2012  1      0     1            ... 100
3     Bond [Cell]     01-Jan-2010     01-Jan-2013  1      0     1            ... 100
4     Bond [Cell]     01-Jan-2010     01-Jan-2014  1      0     1            ... 100

% Build the tree with the following data
VolDates = ['1-Jan-2011'; '1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014'];
VolCurve = 0.01;
AlphaDates = '01-01-2014';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RS.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RS, HWTimeSpec);
```

```
% Compute the price of the  stepped coupon bonds
PHW = hwprice(HWT, ISet)

PHW =

  100.6763
  100.7368
  100.9266
  101.0115
```

Price a portfolio of stepped callable bonds and stepped vanilla bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

%Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create an instrument portfolio of 3 stepped callable bonds and three
% stepped vanilla bonds
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};
OptSpec='call';
Strike=100;
ExerciseDates='01-Jan-2011'; %Callable in one year

% Bonds with embedded option
```

```
ISet = instoptembnd(CouponRate, Settle, Maturity, OptSpec, Strike,...
ExerciseDates, 'Period', 1);

% Vanilla bonds
ISet = instbond(ISet, CouponRate, Settle, Maturity, 1);

% Display the instrument portfolio
instdisp(ISet)

%Table of instrument portfolio partially displayed:
Index Type    CouponRate Settle         Maturity       OptSpec Strike ExerciseDates ... AmericanOpt
1   OptEmBond [Cell]    01-Jan-2010   01-Jan-2012   call    100   01-Jan-2011   ... 0
2   OptEmBond [Cell]    01-Jan-2010   01-Jan-2013   call    100   01-Jan-2011   ... 0
3   OptEmBond [Cell]    01-Jan-2010   01-Jan-2014   call    100   01-Jan-2011   ... 0

Index Type CouponRate Settle         Maturity       Period Basis EndMonthRule ... Face
4    Bond [Cell]    01-Jan-2010   01-Jan-2012   1      0     1            ... 100
5    Bond [Cell]    01-Jan-2010   01-Jan-2013   1      0     1            ... 100
6    Bond [Cell]    01-Jan-2010   01-Jan-2014   1      0     1            ... 100
```

```
% Build the tree with the following data
VolDates = ['1-Jan-2011'; '1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014'];
VolCurve = 0.01;
AlphaDates = '01-01-2014';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RS.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RS, HWTimeSpec);

%The first three rows corresponds to the price of the stepped callable bonds
% and the last three rows corresponds to the price of the stepped vanilla bonds.

PHW = hwprice(HWT, ISet)
```

```
PHW =

    100.4222
    100.2275
    100.0502
    100.7368
    100.9266
    101.0115
```

Compute the price of a portfolio using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

%  Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Create an instrument portfolio with two range notes and a floating rate
% note with the following data:
Spread = 200;
Settle = 'Jan-1-2011';
Maturity = 'Jan-1-2014';

% First Range Note:
RateSched(1).Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
RateSched(1).Rates  = [0.045 0.055; 0.0525  0.0675; 0.06 0.08];

% Second Range Note:
RateSched(2).Dates = {'Jan-1-2012'; 'Jan-1-2013' ; 'Jan-1-2014'};
```

```
RateSched(2).Rates  = [0.048 0.059; 0.055  0.068 ; 0.07 0.09];


% Create InstSet
InstSet = instadd('RangeFloat', Spread, Settle, Maturity, RateSched);

% Add a floating-rate note
InstSet = instadd(InstSet, 'Float', Spread, Settle, Maturity);

% Display the portfolio instrument
instdisp(InstSet)
```

Index	Type	Spread	Settle	Maturity	RateSched	FloatReset	Basis	Principal	EndMonthRule
1	RangeFloat	200	01-Jan-2011	01-Jan-2014	[Struct]	1	0	100	1
2	RangeFloat	200	01-Jan-2011	01-Jan-2014	[Struct]	1	0	100	1

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	EndMonthRule
3	Float	200	01-Jan-2011	01-Jan-2014	1	0	100	1

```
% The data to build the tree is as follows:
VolDates = ['1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014';'1-Jan-2015'];
VolCurve = 0.01;
AlphaDates = '01-01-2015';
AlphaCurve = 0.1;

HWVS = hwvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTS = hwtimespec(RS.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVS, RS, HWTS);

% Price the portfolio
Price = hwprice(HWT, InstSet)

Price =

   99.4075
```

hwprice

```
    98.1003
   105.5147
```

See Also   hwsens | hwtree | instadd | intenvprice | intenvsens

Purpose   Instrument prices and sensitivities from Hull-White interest-rate tree

Syntax   [Delta, Gamma, Vega, Price] = hwsens(HWTree, InstSet, Options)

Arguments

HWTree	Interest-rate tree structure created by hwtree.
InstSet	Variable containing a collection of NINST instruments. Instruments are categorized by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Delta, Gamma, Vega, Price] = hwsens(HWTree, InstSet, Options) computes instrument sensitivities and prices for instruments using an interest-rate tree created with the hwtree function. NINST instruments from a financial instrument variable, InstSet, are priced. hwsens handles instrument types: 'Bond', 'CashFlow', 'OptBond', 'OptEmBond', 'OptEmBond', 'OptFloat', 'OptEmFloat', 'Fixed', 'Float', 'Cap', 'Floor', 'RangeFloat', 'Swap'. See instadd for information on instrument types.

Delta is an NINST-by-1 vector of deltas, representing the rate of change of instrument prices with respect to changes in the interest rate. Delta is computed by finite differences in calls to hwtree. See hwtree for information on the observed yield curve.

Gamma is an NINST-by-1 vector of gammas, representing the rate of change of instrument deltas with respect to the changes in the interest rate. Gamma is computed by finite differences in calls to hwtree.

Vega is an NINST-by-1 vector of vegas, representing the rate of change of instrument prices with respect to the changes in the volatility $\sigma(t,T)$.

Vega is computed by finite differences in calls to hwtree. See hwvolspec for information on the volatility process.

Note All sensitivities are returned as dollar sensitivities. To find the per-dollar sensitivities, divide by the respective instrument price.

Price is an NINST-by-1 vector of prices of each instrument. The prices are computed by backward dynamic programming on the interest-rate tree. If an instrument cannot be priced, NaN is returned.

Delta and Gamma are calculated based on yield shifts of 100 basis points. Vega is calculated based on a 1% shift in the volatility process.

Examples   Load the tree and instruments from a data file. Compute Delta and Gamma for the cap and bond instruments contained in the instrument set.

```
load deriv.mat;
HWSubSet = instselect(HWInstSet,'Type', {'Bond', 'Cap'});

instdisp(HWSubSet)

%Table of instrument portfolio partially displayed:
Index Type CouponRate Settle        Maturity    Period ...  Name ...
1     Bond 0.04       01-Jan-2004  01-Jan-2007  1      ... 4% Bond
2     Bond 0.04       01-Jan-2004  01-Jan-2008  1      ... 4% Bond

Index Type Strike Settle       Maturity    CapReset ... Name ...
3     Cap  0.06   01-Jan-2004 01-Jan-2008 1        ... 6% Cap

[Delta, Gamma] = hwsens(HWTree, HWSubSet)

Delta =

       -291.26
       -374.64
         59.55
```

```
Gamma =

     858.41
    1460.88
    4843.65
```

See Also   hwprice | hwtree | hwvolspec | instadd

hwtimespec

Purpose	Specify time structure for Hull-White interest-rate tree
Syntax	TimeSpec = hwtimespec(ValuationDate, Maturity, Compounding)

Arguments

ValuationDate	Scalar date marking the pricing date and first observation in the tree. Specify as a serial date number or date string
Maturity	Number of levels (depth) of the tree. A number of levels (NLEVELS)-by-1 vector of dates marking the cash flow dates of the tree. Cash flows with these maturities fall on tree nodes. Maturity should be in increasing order.
Compounding	(Optional) Scalar value representing the rate at which the input zero rates were compounded when annualized. Default = -1 (continuous compounding). This argument determines the formula for the discount factors: Compounding = 1, 2, 3, 4, 6, 12 = F Disc = (1 + Z/F)^(-T), where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year. Compounding = 365 Disc = (1 + Z/F)^(-T), where F is the number of days in the basis year and T is a number of days elapsed computed by basis. Compounding = -1 Disc = exp(-T*Z), where T is time in years.

Description   TimeSpec = hwtimespec(ValuationDate, Maturity, Compounding)
sets the number of levels and node times for a Hull-White tree and
determines the mapping between dates and time for rate quoting.

TimeSpec is a structure specifying the time layout for hwtree. The
state observation dates are [Settle; Maturity(1:end-1)]. Because a
forward rate is stored at the last observation, the tree can value cash
flows out to Maturity.

Examples   Specify a four-period tree with annual nodes. Use annual compounding
to report rates.

```
ValuationDate = 'Jan-1-2004';
Maturity = ['12-31-2004'; '12-31-2005'; '12-31-2006';
'12-31-2007'];
Compounding = 1;
TimeSpec = hwtimespec(ValuationDate, Maturity, Compounding)


TimeSpec =

            FinObj: 'HWTimeSpec'
     ValuationDate: 731947
          Maturity: [4x1 double]
       Compounding: 1
             Basis: 0
       EndMonthRule: 1
```

See Also   bktree | hwtree | hwvolspec

hwtree

Purpose	Construct Hull-White interest-rate tree
Syntax	HWTree = hwtree(VolSpec, RateSpec, TimeSpec) HWTree = hwtree(VolSpec, RateSpec, TimeSpec, Name,Value)
Description	HWTree = hwtree(VolSpec, RateSpec, TimeSpec) creates a structure containing time and interest-rate information on a recombining tree. HWTree = hwtree(VolSpec, RateSpec, TimeSpec, Name,Value) creates a structure containing time and interest-rate information on a recombining tree with additional options specified by one or more Name,Value pair arguments.

Input Arguments

VolSpec

Volatility process specification. See hwvolspec for information on the volatility process.

RateSpec

Interest-rate specification for the initial rate curve. See intenvset for information on declaring an interest-rate variable.

TimeSpec

Tree time layout specification. Defines the observation dates of the HW tree and the compounding rule for date to time mapping and price-yield formulas. See hwtimespec for information on the tree structure.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Method'

String specifies the Hull-White method upon which the tree-node connectivity algorithm is based. Possible values are HW1996 and HW2000.

Note hwtree supports two tree-node connectivity algorithms. HW1996 is based on the original paper published in the *Journal of Derivatives*, and HW2000 is the general version of the algorithm, as specified in the paper published in August 2000.

Default: HW1996

Output Arguments

HWTree

Structure containing time and interest rate information of a trinomial recombining tree.

Examples

Using the data provided, create a Hull-White volatility specification (VolSpec), rate specification (RateSpec), and tree time layout specification (TimeSpec). Then use these specifications to create a Hull-White tree using hwtree.

```
Compounding = -1;
ValuationDate = '01-01-2004';
StartDate = ValuationDate;
VolDates = ['12-31-2004'; '12-31-2005'; '12-31-2006';
'12-31-2007'];
VolCurve = 0.01;
AlphaDates = '01-01-2008';
AlphaCurve = 0.1;
Rates = [0.0275; 0.0312; 0.0363; 0.0415];

HWVolSpec = hwvolspec(ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);

RateSpec = intenvset('Compounding', Compounding,...
     'ValuationDate', ValuationDate,...
```

```
            'StartDates', ValuationDate,...
            'EndDates', VolDates,...
            'Rates', Rates);

HWTimeSpec = hwtimespec(ValuationDate, VolDates, Compounding);
HWTree = hwtree(HWVolSpec, RateSpec, HWTimeSpec)

HWTree =

     FinObj: 'HWFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 0.9973 1.9973 2.9973]
       dObs: [731947 732312 732677 733042]
     CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [3.9973]}
      Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
    Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
     FwdTree: {1x4 cell}
```

Use treeviewer to observe the tree you have created.

```
treeviewer(HWTree)
```

Using the data provided, create a Hull-White volatility specification (VolSpec), rate specification (RateSpec), and tree time layout specification (TimeSpec). Then use these specifications to create a Hull-White tree using hwtree.

```
Compounding = -1;
ValuationDate = '01-01-2004';
StartDate = ValuationDate;
VolDates = ['12-31-2004'; '12-31-2005'; '12-31-2006';
'12-31-2007'];
VolCurve = 0.01;
AlphaDates = '01-01-2008';
AlphaCurve = 0.1;
Rates = [0.0275; 0.0312; 0.0363; 0.0415];

HWVolSpec = hwvolspec(ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
```

```
RateSpec = intenvset('Compounding', Compounding,...
     'ValuationDate', ValuationDate,...
     'StartDates', ValuationDate,...
     'EndDates', VolDates,...
     'Rates', Rates);

HWTimeSpec = hwtimespec(ValuationDate, VolDates, Compounding);
HWTree = hwtree(HWVolSpec, RateSpec, HWTimeSpec)

HWTree =

     FinObj: 'HWFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 0.9973 1.9973 2.9973]
       dObs: [731947 732312 732677 733042]
     CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [3.9973]}
      Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
    Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
    FwdTree: {1x4 cell}
```

Use treeviewer to observe the tree you have created.

```
treeviewer(HWTree)
```

References   Hull, J., and A. White, "Using Hull-White Interest Rate Trees", *Journal of Derivatives*, 1996.

Hull, J., and A. White, "The General Hull-White Model and Super Calibration", August 2000.

See Also | hwcalbycap | hwcalbyfloor | hwprice | hwtimespec | hwvolspec | intenvset

Tutorials   • "Calibrating Hull-White Model Using Market Data" on page 2-79

hwvolspec

Purpose	Specify Hull-White interest-rate volatility process

Syntax   Volspec = hwvolspec(ValuationDate, VolDates, VolCurve, AlphaDates, AlphaCurve, InterpMethod)

Arguments

ValuationDate	Scalar value representing the observation date of the investment horizon.
VolDates	Number of points (NPOINTS)-by-1 vector of yield volatility end dates.
VolCurve	NPOINTS-by-1 vector or scalar of yield volatility values in decimal form.
AlphaDates	MPOINTS-by-1 vector of mean reversion end dates.
AlphaCurve	MPOINTS-by-1 vector of positive mean reversion values or scalar in decimal form.
InterpMethod	(Optional) Interpolation method. Default is 'linear'. See interp1 for more information.

Note The number of points in VolCurve and AlphaCurve do not have to be the same.

Description   Volspec = hwvolspec(ValuationDate, VolDates, VolCurve, AlphaDates, AlphaCurve, InterpMethod) creates a structure specifying the volatility for hwtree.

The volatility process is such that the variance of r(t + dt) - r(t) is defined as follows: V = (Volatility.^2 .* (1 - exp(-2*Alpha .* dt))) ./ (2 * Alpha). For more information on using Hull-White interest rate trees, see "Hull-White (HW) and Black-Karasinski (BK) Modeling" on page C-4.

Examples   Using the data provided, create a Hull-White volatility specification
                   (VolSpec).

```
ValuationDate = '01-01-2004';
StartDate = ValuationDate;
VolDates = ['12-31-2004'; '12-31-2005'; '12-31-2006';
'12-31-2007'];
VolCurve = 0.01;
AlphaDates = '01-01-2008';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve)

HWVolSpec =

              FinObj: 'HWVolSpec'
       ValuationDate: 731947
            VolDates: [4x1 double]
            VolCurve: [4x1 double]
          AlphaCurve: 0.1000
          AlphaDates: 733408
       VolInterpMethod: 'linear'
```

See Also   bktree | hwcalbycap | hwcalbyfloor | interp1

impvbybjs

Purpose Determine implied volatility using Bjerksund-Stensland 2002 option pricing model

Syntax
```
Volatility = impvbybjs(RateSpec, StockSpec, Settle,
Maturity, OptSpec, Strike, OptPrice, 'Name1', Value1...)
```

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
OptPrice	NINST-by-1 vector of American option prices from which the implied volatility of the underlying asset are derived.

Note All optional inputs are specified as matching parameter name/parameter value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/parameter value pairs in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

Limit	(Optional) 1-by-2 positive vector representing the lower and upper bound of the implied volatility search interval. Default is [0.1 10], or 10% to 1000% per annum.
Tolerance	(Optional) Positive scalar implied volatility termination tolerance. Default is 1e-6.

Description   Volatility = impvbybjs(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, OptPrice, 'Name1', Value1...) computes implied volatility using the Bjerksund-Stensland 2002 option pricing model.

Volatility is a NINST-by-1 vector of expected implied volatility values. If no solution is found, a NaN is returned.

Note impvbybjs computes implied volatility of American options with continuous dividend yield using the Bjerksund-Stensland option pricing model.

Examples   Consider three American call options with exercise prices of $100 that expire on July 1, 2008. The underlying stock is trading at $100 on January 1, 2008 and pays a continuous dividend yield of 10%. The annualized continuously compounded risk-free rate is 10% per annum and the option prices are $4.063, $6.77 and $9.46. Using this data, calculate the implied volatility of the stock using the Bjerksund-Stensland 2002 option pricing model:

```
AssetPrice = 100;
Settle = 'Jan-1-2008';
Maturity = 'Jul-1-2008';
Strike = 100;
DivAmount = 0.1;
Rate = 0.1;
```

impvybjs

Define `RateSpec` and `StockSpec`:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rate, 'Compounding', -1, 'Basis', 1);

StockSpec = stockspec(NaN, AssetPrice, {'continuous'}, DivAmount);
```

Calculate the implied volatility of the call options:

```
OptSpec = {'call'};
OptionPrice = [4.063;6.77;9.46];

ImpVol =  impvybjs(RateSpec, StockSpec, Settle, Maturity, OptSpec,...
Strike, OptionPrice)

ImpvVol =

    0.1500
    0.2501
    0.3500
```

The implied volatility is 15% for the first call, and 25% and 35% for
the second and third call options.

References   Bjerksund, P. and G. Stensland, *Closed-Form Approximation of
American Options*, Scandinavian Journal of Management, 1993, Vol. 9,
Suppl., pp. S88-S99.

Bjerksund, P. and G. Stensland, *Closed Form
Valuation of American Options*, Discussion paper 2002
(http://brage.bibsys.no/nhh/bitstream/URN:NBN:no-bibsys_brage_22301/1/bjerksund%2

See Also   optstockbybjs | optstocksensbybjs

Purpose Determine implied volatility using Black option pricing model

Syntax
```
Volatility = impvbyblk(RateSpec, StockSpec, Settle,
Maturity, Strike, OptPrice, 'Name1', Value1...)
```

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
OptPrice	NINST-by-1 vector of European option prices from which the implied volatility of the underlying asset are derived.

Note All optional inputs are specified as matching parameter name/parameter value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/parameter value pairs in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

Limit	(Optional) Positive scalar representing the upper bound of the implied volatility search interval. Default is 10, or 1000% per annum.
Tolerance	(Optional) Positive scalar implied volatility termination tolerance. Default is 1e-6.

impvbyblk

Description   `Volatility = impvbyblk(RateSpec, StockSpec, Settle, Maturity, Strike, OptPrice, 'Name1', Value1...)` computes implied volatility using the Black option pricing model.

`Volatility` is a `NINST-by-1` vector of expected implied volatility values. If no solution is found, a `NaN` is returned.

Examples   Consider a European call and put options on a futures contract with exercise prices of $30 for the put and $40 for the call that expire on September 1, 2008. Assume that on May 1, 2008 the contract is trading at $35. The annualized continuously compounded risk-free rate is 5% per annum. What are the implied volatilities of the stock, if on that date, the call price is $1.14 and the put price is $0.82

```
AssetPrice = 35;
Strike = [30; 40];
Rates = 0.05;
Settle = 'May-01-08';
Maturity = 'Sep-01-08';
```

Create `RateSpec` and `StockSpec`:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1);

StockSpec = stockspec(NaN, AssetPrice);
```

Define the options:

```
OptSpec = {'put';'call'};
```

Calculate the implied volatility of the options:

```
Price = [1.14;0.82];
Volatility = impvbyblk(RateSpec, StockSpec, Settle, Maturity, OptSpec,...
Strike, Price)
```

```
Volatility =

    0.4052
    0.3021
```

The implied volatility would be 41% and 30%.

See Also   optstockbyblk | optstocksensbyblk

impvbybls

Purpose	Determine implied volatility using Black-Scholes option pricing model
Syntax	Volatility = impvbybls(RateSpec, StockSpec, Settle, Maturity, Strike, OptPrice, 'Name1', Value1...)

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
OptPrice	NINST-by-1 vector of European option prices from which the implied volatility of the underlying asset are derived.

Note All optional inputs are specified as matching parameter name/parameter value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/parameter value pairs in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

Limit	(Optional) Positive scalar representing the upper bound of the implied volatility search interval. Default is 10, or 1000% per annum.
Tolerance	(Optional) Positive scalar implied volatility termination tolerance. Default is 1e-6.

Description   Volatility = impvbybls(RateSpec, StockSpec, Settle, Maturity, Strike, OptPrice, 'Name1', Value1...) computes implied volatility using the Black-Scholes option pricing model.

Volatility is a NINST-by-1 vector of expected implied volatility values. If no solution is found, a NaN is returned.

Examples   Consider a European call and put options with an exercise price of $40 that expires on June 1, 2008. The underlying stock is trading at $45 on January 1, 2008 and the risk-free rate is 5% per annum. The option price is $7.10 for the call and $2.85 for the put. Using this data, calculate the implied volatility of the European call and put using the Black-Scholes option pricing model:

```
AssetPrice = 45;
Settlement = 'Jan-01-2008';
Maturity = 'June-01-2008';
Strike = 40;
Rates = 0.05;
OptionPrice = [7.10; 2.85];
OptSpec = {'call';'put'};
```

Define RateSpec and StockSpec :

```
RateSpec = intenvset('ValuationDate', Settlement, 'StartDates', Settlement,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1, 'Basis', 1);

StockSpec = stockspec(NaN, AssetPrice);
```

Calculate the implied volatility of the options:

```
ImpvVol =  impvbybls(RateSpec, StockSpec, Settlement, Maturity, OptSpec,...
Strike, OptionPrice)

ImpvVol =
```

```
                    0.3175
                    0.4878
```

The implied volatility is 31.75% for the call and 48.78% for the put.

See Also   optstockbybls | optstocksensbybls

Purpose	Determine implied volatility using Roll-Geske-Whaley option pricing model for American call option
Syntax	`Volatility = impvbyrgw(RateSpec, StockSpec, Settle, Maturity, Strike, OptPrice, 'Name1', Value1...)`

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see `intenvset`.
StockSpec	Stock specification. See `stockspec`.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
Strike	NINST-by-1 vector of strike price values.
OptPrice	NINST-by-1 vector of American call option prices from which the implied volatility of the underlying asset are derived.

Note All optional inputs are specified as matching parameter name/parameter value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/parameter value pairs in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

Limit	(Optional) Positive scalar representing the upper bound of the implied volatility search interval. Default is `10`, or 1000% per annum.
Tolerance	(Optional) Positive scalar implied volatility termination tolerance. Default is `1e-6`.

impvbyrgw

Description
Volatility = impvbyrgw(RateSpec, StockSpec, Settle, Maturity, Strike, OptPrice, 'Name1', Value1...) computes implied volatility using the Roll-Geske-Whaley option pricing model.

Volatility is a NINST-by-1 vector of expected implied volatility values. If no solution is found, a NaN is returned.

Note impvbyrgw computes implied volatility of American calls with a single cash dividend using the Roll-Geske-Whaley option pricing model.

Examples
Assume that on July 1, 2008 a stock is trading at $13 and pays a single cash dividend of $0.25 on November 1, 2008. The American call option with a strike price of $15 expires on July 1, 2009 and is trading at $1.346. The annualized continuously compounded risk-free rate is 5% per annum. Calculate the implied volatility of the stock using the Roll-Geske-Whaley option pricing model:

```
AssetPrice = 13;
Strike = 15;
Rates = 0.05;
Settle = 'July-01-08';
Maturity = 'July-01-09';
```

Define StockSpec and RateSpec:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
 'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1);

StockSpec = stockspec(NaN, AssetPrice, {'cash'}, 0.25, {'Nov 1,2008'});
```

Calculate the implied volatility of the option:

```
Price = [1.346];
Volatility = impvbyrgw(RateSpec, StockSpec, Settle, Maturity, Strike, Price)

Volatility =
```

```
0.3539
```

See Also   optstockbyrgw | optstocksensbyrgw

instadd

Purpose   Add types to instrument collection

Syntax   InstSet = instadd(InstSetOld, TypeString, Data1, Data2, ...)
                InstSet = instadd('CashFlow', CFlowAmounts, CFlowDates, Settle,
                Basis)
                InstSet = instadd('CashFlow', CFlowAmounts, CFlowDates, Settle,
                Basis)
                InstSet = instadd('Barrier', OptSpec, Strike, Settle,
                ExerciseDates, AmericanOpt, BarrierType, Barrier, Rebate)
                InstSet = instadd('Bond', CouponRate, Settle, Maturity, Period,
                Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
                StartDate, Face)
                InstSet = instadd('OptEmBond', CouponRate, Settle, Maturity,
                OptSpec, Strike, ExerciseDates, 'AmericanOpt', AmericanOpt, 'Period',
                Period, 'Basis', Basis, 'EndMonthRule', EndMonthRule,'Face',Face,
                'IssueDate', IssueDate, 'FirstCouponDate', FirstCouponDate,
                'LastCouponDate', LastCouponDate, 'StartDate', StartDate)
                InstSet = instadd('OptBond', BondIndex, OptSpec, Strike,
                ExerciseDates, AmericanOpt)
                InstSet = instadd('Cap', Strike, Settle, Maturity, Reset, Basis,
                Principal)
                InstSet = instadd('Compound', UOptSpec, UStrike, USettle,
                UExerciseDates, UAmericanOpt,COptSpec, CStrike, CSettle,
                CExerciseDates, CAmericanOpt)
                InstSet = instadd('Fixed', CouponRate, Settle, Maturity, Reset,
                Basis, Principal, EndMonthRule)
                InstSet = instadd('Float', Spread, Settle, Maturity, Reset, Basis,
                Principal, EndMonthRule)
                InstSet = instadd('Floor', Strike, Settle, Maturity, Reset, Basis,
                Principal)
                InstSet = instadd('Lookback', OptSpec, Strike, Settle,
                ExerciseDates, AmericanOpt)
                InstSet = instadd('OptFloat', OptSpec, Strike, Settle,
                ExerciseDates, AmericanOpt)
                InstSet = instadd('OptEmFloat', OptSpec, Strike, Settle,
                ExerciseDates, AmericanOpt)
                InstSet = instadd('RangeFloat', Spread, Settle, Maturity,

```
RateSched, Reset, Basis, Principal, EndMonthRule)
InstSet = instadd('OptStock', OptSpec, Strike, Settle, Maturity,
AmericanOpt)
InstSet = instadd('Swap', LegRate, Settle, Maturity, LegReset,
Basis, Principal, LegType, EndMonthRule, StartDate)
InstSet = instadd('Swaption', OptSpec, Strike, ExerciseDates,
Spread, Settle, Maturity, AmericanOpt, SwapReset,
Basis, Principal)
```

Description   `InstSet = instadd(InstSetOld, TypeString, Data1, Data2,
...)` adds an instruments to an existing collection.

`InstSet = instadd('CashFlow', CFlowAmounts, CFlowDates,
Settle,
Basis)` adds an arbitrary cash flow instrument. (See also `instcf`.)

`InstSet = instadd('CashFlow', CFlowAmounts, CFlowDates,
Settle,
Basis)` adds an asian instrument. (See also `instasian`.)

`InstSet = instadd('Barrier', OptSpec, Strike, Settle,
ExerciseDates, AmericanOpt, BarrierType, Barrier, Rebate)`
adds a barrier instrument. (See also `instbarrier`.)

`InstSet = instadd('Bond', CouponRate, Settle, Maturity,
Period,
Basis, EndMonthRule, IssueDate, FirstCouponDate,
LastCouponDate,
StartDate, Face)` adds a bond instrument. (See also `instbond`.)

`InstSet = instadd('OptEmBond', CouponRate, Settle,
Maturity,
OptSpec, Strike, ExerciseDates, 'AmericanOpt', AmericanOpt,
'Period',
Period, 'Basis', Basis, 'EndMonthRule',
EndMonthRule,'Face',Face,
'IssueDate', IssueDate, 'FirstCouponDate', FirstCouponDate,
'LastCouponDate', LastCouponDate, 'StartDate', StartDate)`

adds a bond with embedded option instrument. (See also `instoptembnd`.)

`InstSet = instadd('OptBond', BondIndex, OptSpec, Strike, ExerciseDates, AmericanOpt)` adds a bond option instrument. (See also `instoptbnd`.)

`InstSet = instadd('Cap', Strike, Settle, Maturity, Reset, Basis, Principal)` adds a cap instrument. (See also `instcap`.)

`InstSet = instadd('Compound', UOptSpec, UStrike, USettle, UExerciseDates, UAmericanOpt,COptSpec, CStrike, CSettle, CExerciseDates, CAmericanOpt)` adds a compound instrument. (See also `instcompound`.)

`InstSet = instadd('Fixed', CouponRate, Settle, Maturity, Reset, Basis, Principal, EndMonthRule)` adds a fixed-rate note instrument. (See also `instfixed`.)

`InstSet = instadd('Float', Spread, Settle, Maturity, Reset, Basis, Principal, EndMonthRule)` adds a floating-rate note instrument. (See also `instfloat`.)

`InstSet = instadd('Floor', Strike, Settle, Maturity, Reset, Basis, Principal)` adds a floor instrument. (See also `instfloor`.)

`InstSet = instadd('Lookback', OptSpec, Strike, Settle, ExerciseDates, AmericanOpt)` adds a lookback instrument. (See also `instlookback`.)

`InstSet = instadd('OptFloat', OptSpec, Strike, Settle, ExerciseDates, AmericanOpt)` adds a floating-rate option instrument. (See also `instoptfloat`.)

`InstSet = instadd('OptEmFloat', OptSpec, Strike, Settle, ExerciseDates, AmericanOpt)` adds a floating-rate embedded option instrument. (See also `instoptemfloat`.)

InstSet = instadd('RangeFloat', Spread, Settle, Maturity, RateSched, Reset, Basis, Principal, EndMonthRule) adds a range floating note instrument. (See also instrangefloat.)

InstSet = instadd('OptStock', OptSpec, Strike, Settle, Maturity, AmericanOpt) adds a stock option instrument. (See also instoptstock.)

InstSet = instadd('Swap', LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType, EndMonthRule, StartDate) adds a swap instrument. (See also instswap.)

InstSet = instadd('Swaption', OptSpec, Strike, ExerciseDates, Spread, Settle, Maturity, AmericanOpt, SwapReset, Basis, Principal) adds a swaption instrument. (See also instswaption.)

instadd stores instruments of types 'Asian', 'Barrier', 'Bond', 'Cap', 'CashFlow', 'Compound', 'Fixed', 'Float', 'Floor', 'Lookback', 'OptBond', 'OptStock', 'Swap', or 'Swaption'. Financial Instruments Toolbox provides pricing and sensitivity routines for these instruments.

Input Arguments

InstSetOld

Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument. For more information on instrument data parameters, see the reference entries for individual instrument types. For example, see instcap for additional information on the cap instrument.

Output Arguments

InstSet

InstSet is an instrument set variable containing the new input data.

Examples

Create a Portfolio with Two Cap Instruments and a 4% Bond

Define the bond:

instadd

```
Strike = [0.06; 0.07];
CouponRate = 0.04;
Settle = '06-Feb-2000';
Maturity = '15-Jan-2003';
```

Create a portfolio with two cap instruments and a 4% bond and then display the portfolio:

```
InstSet = instadd('Cap', Strike, Settle, Maturity);
InstSet = instadd(InstSet, 'Bond', CouponRate, Settle, Maturity);
instdisp(InstSet)
```

Index	Type	Strike	Settle	Maturity	CapReset	Basis	Principal
1	Cap	0.06	06-Feb-2000	15-Jan-2003	1	0	100
2	Cap	0.07	06-Feb-2000	15-Jan-2003	1	0	100

Index	Type	CouponRate	Settle	Maturity	Period	Basis	EndMonthRule	IssueDate	FirstCouponDate	LastCouponDat
3	Bond	0.04	06-Feb-2000	15-Jan-2003	2	0	1	NaN	NaN	NaN

See Also   instasian | instbarrier | instbond | instcap | instcf | instcompound | instfixed | instfloat | instfloor | instlookback | instoptbnd | instoptembnd | instoptstock | instswap | instswaption | instaddfield | instdisp

Related Examples
- "Portfolio Creation" on page 1-7
- "Creating Instruments or Properties" on page 1-19

Concepts
- "Instrument Constructors" on page 1-18

Purpose   Add new instruments to instrument collection

Syntax   InstSet = instaddfield('FieldName', FieldList, 'Data',
                DataList, 'Type', TypeString)
                InstSet = instaddfield('FieldName', FieldList, 'FieldClass',
                ClassList, 'Data', DataList, 'Type', TypeString)
                InstSetNew = instaddfield(InstSet, 'FieldName', FieldList,
                'Data', DataList,'Type',TypeString)

Arguments

FieldList	String or number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field. FieldList cannot be named with the reserved name Type or Index.
DataList	Number of instruments (NINST)-by-M array or NFIELDS-by-1 cell array of data contents for each field. Each row in a data array corresponds to a separate instrument. Single rows are copied to apply to all instruments to be worked on. The number of columns is arbitrary, and data is padded along columns.
ClassList	(Optional) String or NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how DataList is parsed. Valid strings are 'dble', 'date', and 'char'. The 'FieldClass', ClassList pair is always optional. ClassList is inferred from existing field names or from the data if not entered.

instaddfield

`TypeString`	String specifying the type of instrument added. Instruments of different types can have different `Fieldname` collections.
`InstSet`	Variable containing a collection of instruments. Instruments are classified by type; each type can have different `Data` fields. The stored `Data` field is a row vector or string for each instrument.

Description   Use `instaddfield` to create your own types of instruments or to append new instruments to an existing collection. Argument value pairs can be entered in any order.

```
InstSet = instaddfield('FieldName', FieldList, 'Data',
DataList, 'Type', TypeString)
```

```
InstSet = instaddfield('FieldName', FieldList,
'FieldClass',
ClassList, 'Data', DataList, 'Type', TypeString) creates an
```
instrument variable.

```
InstSetNew = instaddfield(InstSet, 'FieldName', FieldList,
'Data', DataList,'Type',TypeString) adds instruments to an
```
existing instrument set, `InstSet`. The output `InstSetNew` is a new instrument set containing the input data.

Examples   Build a portfolio around July options.

```
Strike  Call    Put
 95     12.2    2.9
100      9.2    4.9
105      6.8    7.4

Strike = (95:5:105)'
CallP = [12.2; 9.2; 6.8]
```

Enter three call options with data fields `Strike`, `Price`, and `Opt`.

```
InstSet = instaddfield('Type','Option','FieldName',...
{'Strike','Price','Opt'}, 'Data',{ Strike, CallP, 'Call'});
 instdisp(InstSet)

Index Type    Strike Price Opt
1     Option  95    12.2  Call
2     Option 100     9.2  Call
3     Option 105     6.8  Call
```

Add a futures contract and set the input parsing class.

```
InstSet = instaddfield(InstSet,'Type','Futures',...
'FieldName',{'Delivery','F'},'FieldClass',{'date','dble'},...
'Data' ,{'01-Jul-99',104.4 });
instdisp(InstSet)

Index Type    Strike Price Opt
1     Option  95    12.2  Call
2     Option 100     9.2  Call
3     Option 105     6.8  Call

Index Type    Delivery       F
4     Futures 01-Jul-1999   104.4
```

Add a put option.

```
FN = instfields(InstSet,'Type','Option')
InstSet = instaddfield(InstSet,'Type','Option',...
'FieldName',FN, 'Data',{105, 7.4, 'Put'});
instdisp(InstSet)

Index Type    Strike Price Opt
1     Option  95    12.2  Call
2     Option 100     9.2  Call
3     Option 105     6.8  Call

Index Type    Delivery       F
4     Futures 01-Jul-1999   104.4
```

```
Index Type    Strike Price Opt
5     Option 105     7.4  Put
```

Make a placeholder for another put.

```
InstSet = instaddfield(InstSet,'Type','Option',...
'FieldName','Opt','Data','Put')
instdisp(InstSet)

Index Type    Strike Price Opt
1     Option  95   12.2  Call
2     Option 100    9.2  Call
3     Option 105    6.8  Call

Index Type    Delivery       F
4     Futures 01-Jul-1999   104.4

Index Type    Strike Price Opt
5     Option 105     7.4  Put
6     Option NaN     NaN  Put
```

Add a cash instrument.

```
InstSet = instaddfield(InstSet, 'Type', 'TBill',...
'FieldName','Price','Data',99)
instdisp(InstSet)

Index Type    Strike Price Opt
1     Option  95   12.2  Call
2     Option 100    9.2  Call
3     Option 105    6.8  Call

Index Type    Delivery       F
4     Futures 01-Jul-1999   104.4

Index Type    Strike Price Opt
5     Option 105     7.4  Put
```

```
6      Option NaN      NaN  Put

Index Type  Price
7     TBill 99
```

See Also   instdisp | instget | instgetcell | instsetfield | instadd

instasian

Purpose	Construct Asian option
Syntax	InstSet = instasian(InstSet, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, AvgType, AvgPrice, AvgDate)
	[FieldList, ClassList, TypeString] = instasian

Arguments

InstSet	Instrument variable. This argument is specified only when adding asian instruments to an existing instrument set. See instget for more information on the InstSet variable.
OptSpec	NINST-by-1 list of string values 'Call' or 'Put'.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.
Settle	NINST-by-1 vector of Settle dates.
ExerciseDates	For a European option (AmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (AmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.

AmericanOpt	(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.
AvgType	(Optional) String = 'arithmetic' for arithmetic average (default) or 'geometric' for geometric average.
AvgPrice	(Optional) Scalar representing the average price of the underlying asset at Settle. This argument is used when AvgDate < Settle. Default is the current stock price.
AvgDate	(Optional) Scalar representing the date on which the averaging period begins. Default = Settle.

Data arguments are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Description   InstSet = instasian(InstSet, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, AvgType, AvgPrice, AvgDate) specifies an Asian option.

[FieldList, ClassList, TypeString] = instasian displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are 'dble', 'date', and 'char'.

TypeString is a string specifying the type of instrument added. For an Asian option instrument, TypeString = 'Asian'.

instasian

Examples

Create an Asian Option Instrument

Load the example instrument set, `deriv.mat`, and set the required values for an asian option instrument.

```
load deriv.mat
```

Create a subportfolio with barrier and lookback options.

```
CRRSubSet = instselect(CRRInstSet,'Type',{'Barrier', 'Lookback'});
```

Define the asian instrument.

```
OptSpec = 'put';
Strike = NaN;
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2004';
```

Add a floating strike asian option to the instrument set.

```
InstSet = instasian(CRRSubSet, OptSpec, Strike, Settle, ExerciseDates);
instdisp(InstSet)
```

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	BarrierSpec	Barrier	Rebate	Name	Quantity
1	Barrier	call	105	01-Jan-2003	01-Jan-2006	1	ui	102	0	Barrier1	1

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	Name	Quantity
2	Lookback	call	115	01-Jan-2003	01-Jan-2006	0	Lookback1	7
3	Lookback	call	115	01-Jan-2003	01-Jan-2007	0	Lookback2	9

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	AvgType	AvgPrice	AvgDate	Name	Quantity
4	Asian	put	NaN	01-Jan-2003	01-Jan-2004	0	arithmetic	NaN	NaN		NaN

See Also   instadd | instdisp | instget

Purpose   Construct barrier option

Syntax   InstSet = instbarrier(InstSet, OptSpec, Strike, Settle,
                   ExerciseDates, AmericanOpt, BarrierSpec, Barrier, Rebate)
                   [FieldList, ClassList, TypeString] = instbarrier

Arguments

InstSet	Instrument variable. This argument is specified only when adding barrier instruments to an existing instrument set. See instget for more information on the InstSet variable.
OptSpec	NINST-by-1 list of string values 'Call' or 'Put'.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.
Settle	NINST-by-1 vector of Settle dates.
ExerciseDates	For a European option (AmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (AmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.

instbarrier

AmericanOpt	If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.
BarrierSpec	List of string values:
	'UI': Up Knock In
	'UO': Up Knock Out
	'DI': Down Knock In
	'DO': Down Knock Out
Barrier	Vector of barrier values.
Rebate	(Optional) Vector of rebate values.

Data arguments are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Description InstSet = instbarrier(InstSet, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt, BarrierSpec, Barrier, Rebate) specifies a barrier option.

[FieldList, ClassList, TypeString] = instbarrier displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are 'dble', 'date', and 'char'.

TypeString is a string specifying the type of instrument added. For a barrier option instrument, TypeString = 'Barrier'.

Examples

Create Two Barrier Option Instruments

Create an instrument set of two barrier options with the following data:

```
OptSpec = {'put';'call'};
Strike = 112;
Settle = '01-Jan-2012';
ExerciseDates = '01-Jan-2015';
BarrierSpec = {'do';'ui'};
Barrier = [101;102];
AmericanOpt = 0;
```

Create the instrument set (InstSet) for the two barrier options.

```
InstSet = instbarrier(OptSpec, Strike, Settle, ExerciseDates,AmericanOpt, BarrierSpec, Barrier);
```

Display the instrument set.

```
instdisp(InstSet)
```

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	BarrierSpec	Barrier	Rebate
1	Barrier	put	112	01-Jan-2012	01-Jan-2015	0	do	101	0
2	Barrier	call	112	01-Jan-2012	01-Jan-2015	0	ui	102	0

See Also

instadd | instdisp | instget

instbond

Purpose	Construct bond instrument
Syntax	InstSet = instbond(InstSet, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face) [FieldList, ClassList, TypeString] = instbond

Arguments

InstSet	Instrument variable. This argument is specified only when adding bond instruments to an existing instrument set. See instget for more information on the InstSet variable.
CouponRate	Decimal annual rate indicating the annual percentage rate used to determine the coupons payable on a bond. CouponRate is a NINST-by-1 vector or NINST-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 1, 2, 3, 4, 6, and 12. Default = 2.

11-482

Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)
	• 10 = actual/365 (ICMA)
	• 11 = 30/360E (ICMA)
	• 12 = actual/actual (ISDA)
	• 13 = BUS/252
	For more information, see **basis** on page Glossary-1.
EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when a bond was issued.

FirstCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.
StartDate	(Optional) Date when a bond actually starts (i.e. the date when a bond's cash flows can be considered). To make an instrument forward starting, specify this date as a future date. If StartDate is not explicitly specified, the effective start date is the Settle date.
Face	(Optional) Face or par value. Face is a NINST-by-1 vector or NINST-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a NumDates-by-2 cell array, where the first

column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid. Default = 100.

Data arguments are number of instruments (NINST)-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Description   InstSet = instbond(InstSet, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face) creates a new instrument set containing bond instruments or adds bond instruments to a existing instrument set.

[FieldList, ClassList, TypeString] = instbond displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are 'dble', 'date', and 'char'.

TypeString is a string specifying the type of instrument added. For a bond instrument, TypeString = 'Bond'.

See Also   hjmprice | instaddfield | instdisp | instget | intenvprice

instcap

Purpose	Construct cap instrument

Syntax

```
InstSet = instcap(InstSet, Strike, Settle, Maturity, Reset,
Basis, Principal)
[FieldList, ClassList, TypeString] = instcap
```

Arguments

InstSet	Instrument variable. This argument is specified only when adding cap instruments to an existing instrument set. See instget for more information on the InstSet variable.
Strike	Rate at which the cap is exercised, as a decimal number.
Settle	Settlement date. Serial date number representing the settlement date of the cap.
Maturity	Serial date number representing the maturity date of the cap.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal   (Optional) The notional principal amount. Default = `100`.

Description   `InstSet = instcap(InstSet, Strike, Settle, Maturity, Reset, Basis, Principal)` creates a new instrument set containing cap instruments or adds cap instruments to an existing instrument set.

`[FieldList, ClassList, TypeString] = instcap` displays the classes.

`FieldList` is a number of fields (`NFIELDS`)-by-1 cell array of strings listing the name of each data field for this instrument type.

`ClassList` is an `NFIELDS`-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are `'dble'`, `'date'`, and `'char'`.

`TypeString` is a string specifying the type of instrument added. For a cap instrument, `TypeString = 'Cap'`.

See Also   hjmprice | instaddfield | instbond | instdisp | instfloor | instswap | intenvprice

Purpose	Construct cash flow instrument

Syntax

```
InstSet = instcf(InstSet, CFlowAmounts, CFlowDates, Settle,
Basis)
[FieldList, ClassList, TypeString] = instcf
```

Arguments

InstSet	Instrument variable. This argument is specified only when adding cash flow instruments to an existing instrument set. See instget for more information on the InstSet variable.
CFlowAmounts	Number of instruments (NINST) by maximum number of cash flows (MOSTCFS) matrix of cash flow amounts. Each row is a list of cash flow values for one instrument. If an instrument has fewer than MOSTCFS cash flows, the end of the row is padded with NaNs.
CFlowDates	NINST-by-MOSTCFS matrix of cash flow dates. Each entry contains the date of the corresponding cash flow in CFlowAmounts.
Settle	Settlement date on which the cash flows are priced.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Only one data argument is required to create an instrument. Other arguments can be omitted or passed as empty matrices []. Dates can be input as serial date numbers or date strings.

Description   InstSet = instcf(InstSet, CFlowAmounts, CFlowDates, Settle, Basis) creates a new instrument set from data arrays or adds instruments of type CashFlow to an instrument set.

[FieldList, ClassList, TypeString] = instcf lists field metadata for an instrument of type CashFlow.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are 'dble', 'date', and 'char'.

TypeString specifies the type of instrument added; for example, TypeString = 'CashFlow'.

See Also   instadd | instdisp | instget | intenvprice

instcompound

Purpose	Construct compound option
Syntax	InstSet = instcompound(InstSet, UOptSpec, UStrike, USettle, UExerciseDates, UAmericanOpt, COptSpec, CStrike, CSettle, CExerciseDates, CAmericanOpt) [FieldList, ClassList, TypeString] = instcompound

Arguments

InstSet	Instrument variable. This argument is specified only when adding compound instruments to an existing instrument set. See instget for more information on the InstSet variable.
UOptSpec	String = 'Call' or 'Put'.
UStrike	1-by-1 vector of strike price values.
USettle	1-by-1 vector of Settle dates.
UExerciseDates	For a European option (UAmericanOpt = 0): 1-by-1 vector of exercise dates. For a European option, there is only one exercise date, the option expiry date. For an American option (UAmericanOpt = 1): 1-by-2 vector of exercise date boundaries. The option can be exercised on any tree date. If only one non-NaN date is listed, or if ExerciseDates is 1-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.
UAmericanOpt	If UAmericanOpt = 0, NaN, or is unspecified, the option is a European option. If UAmericanOpt = 1, the option is an American option.
COptSpec	NINST-by-1 list of string values 'Call' or 'Put' of the compound option.

CStrike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.
CSettle	1-by-1 vector containing the settlement or trade date.
CExerciseDates	For a European option (CAmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (CAmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.
CAmericanOpt	If CAmericanOpt = 0, NaN, or is unspecified, the option is a European option. If CAmericanOpt = 1, the option is an American option.

Description   InstSet = instcompound(InstSet, UOptSpec, UStrike, USettle, UExerciseDates, UAmericanOpt, COptSpec, CStrike, CSettle, CExerciseDates, CAmericanOpt) specifies a compound option.

[FieldList, ClassList, TypeString] = instcompound displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are `'dble'`, `'date'`, and `'char'`.

TypeString is a string specifying the type of instrument added. For a compound option instrument, TypeString = `'Compound'`.

Examples

Create a Compound Option Instrument

Define a compound option instrument with the following data:

```
UOptSpec = 'Call';
UStrike = 130;
USettle = '01-Jan-2012';
UExerciseDates = '01-Jan-2015';
UAmericanOpt = 0;
COptSpec = 'Put';
CStrike = 5;
CSettle = '01-Jan-2012';
CExerciseDates = '01-Jan-2014';
CAmericanOpt = 0;

InstSet = instcompound(UOptSpec, UStrike, USettle,UExerciseDates, ...
UAmericanOpt, COptSpec, CStrike, CSettle,CExerciseDates, CAmericanOpt)

InstSet = instcompound(UOptSpec, UStrike, USettle,UExerciseDates, ...
UAmericanOpt, COptSpec, CStrike, CSettle,CExerciseDates)

InstSet =

        FinObj: 'Instruments'
    IndexTable: [1x1 struct]
          Type: {'Compound'}
     FieldName: {{10x1 cell}}
    FieldClass: {{10x1 cell}}
     FieldData: {{10x1 cell}}
```

```
InstSet =

        FinObj: 'Instruments'
    IndexTable: [1x1 struct]
          Type: {'Compound'}
     FieldName: {{10x1 cell}}
    FieldClass: {{10x1 cell}}
     FieldData: {{10x1 cell}}
```

Display the instrument set.

```
instdisp(InstSet)
```

Index	Type	UOptSpec	UStrike	USettle	UExerciseDates	UAmericanOpt	COptSpec	CStrike	CSettle	CE
1	Compound	Call	130	01-Jan-2012	01-Jan-2015	0	Put	5	01-Jan-2012	

See Also   instadd | instdisp | instget

instdelete

Purpose	Complement of instrument set by matching conditions
Syntax	ISubSet = instdelete(InstSet, 'FieldName', FieldList, 'Data', DataList, 'Index', IndexSet, 'Type', TypeList)

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
FieldList	String or number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field to match with data values.
DataList	Number of values (NVALUES)-by-M array or NFIELDS-by-1 cell array of acceptable data values for each field. Each row lists a data row value to search for in the corresponding FieldList. The number of columns is arbitrary and matching will ignore trailing NaNs or spaces.
IndexSet	(Optional) Number of instruments (NINST)-by-1 vector restricting positions of instruments to check for matches. The default is all indices available in the instrument variable.
TypeList	(Optional) String or number of types (NTYPES)-by-1 cell array of strings restricting instruments to match one of TypeList types. The default is all types in the instrument variable.

Note Argument value pairs can be entered in any order. The `InstSet` variable must be the first argument. `'FieldName'` and `'Data'` arguments must appear together or not at all.

Description   The output argument `ISubSet` contains instruments *not* matching the input criteria. Instruments are deleted from `ISubSet` if all the `Field`, `Index`, and `Type` conditions are met. An instrument meets an individual `Field` condition if the stored `FieldName` data matches any of the rows listed in the `DataList` for that `FieldName`. See `instfind` for more examples on matching criteria.

Examples   Retrieve the instrument set variable `ExampleInst` from the data file `InstSetExamples.mat`. The variable contains three types of instruments: `Option`, `Futures`, and `TBill`.

```
load InstSetExamples;
instdisp(ExampleInst)

Index Type   Strike Price Opt  Contracts
1     Option 95     12.2 Call     0
2     Option 100     9.2 Call     0
3     Option 105     6.8 Call  1000

Index Type    Delivery       F     Contracts
4     Futures 01-Jul-1999   104.4 -1000

Index Type   Strike Price Opt  Contracts
5     Option 105     7.4 Put  -1000
6     Option 95      2.9 Put      0

Index Type  Price Maturity      Contracts
7     TBill 99     01-Jul-1999   6
```

Create a new variable, `ISet`, with all options deleted.

instdelete

```
ISet = instdelete(ExampleInst, 'Type','Option');
instdisp(ISet)

Index Type    Delivery        F      Contracts
1     Futures 01-Jul-1999   104.4  -1000

Index Type  Price Maturity       Contracts
2     TBill 99    01-Jul-1999    6
```

See Also   instaddfield | instfind | instget | instselect

Purpose   Display instruments

Syntax   `CharTable = instdisp(InstSet)`

Arguments

`InstSet`	Variable containing a collection of instruments. See `instaddfield` for examples on constructing the variable.

Description   `CharTable = instdisp(InstSet)` creates a character array displaying the contents of an instrument collection, `InstSet`. If instdisp is called without output arguments, the table is displayed in the Command Window.

> **Note** When using `instdisp`, a value of `NaN` in one of the columns for an instrument indicates that the default value for that parameter will be used in the instrument's pricing function.

`CharTable` is a character array with a table of instruments in `InstSet`. For each instrument row, the `Index` and `Type` are printed along with the field contents. Field headers are printed at the tops of the columns.

Examples   Retrieve the instrument set `ExampleInst` from the data file `InstSetExamples.mat`. `ExampleInst` contains three types of instruments: `Option`, `Futures`, and `TBill`.

```
load InstSetExamples;
instdisp(ExampleInst)

Index Type    Strike Price Opt   Contracts
1     Option  95     12.2  Call     0
2     Option  100    9.2   Call     0
```

```
3      Option 105    6.8  Call  1000

Index Type    Delivery       F     Contracts
4      Futures 01-Jul-1999   104.4 -1000

Index Type   Strike Price Opt  Contracts
5      Option 105      7.4 Put  -1000
6      Option  95      2.9 Put     0

Index Type Price Maturity      Contracts
7      TBill 99    01-Jul-1999  6
```

Create a swap instrument and use instdisp to display the instrument. Notice that value of NaN in two columns for this instrument indicates that the default values for LegReset and LegType parameters will be used in the swap instrument's pricing function.

```
LegRate1 = [0.065, 0];
Settle1 = datenum('jan-1-2007');
Maturity1 = datenum('jan-1-2012');

ISet = instswap(LegRate1, Settle1, Maturity1);
instdisp(ISet)
```

```
Index Type LegRate    Settle       Maturity    LegReset Basis Principal LegType EndMonthRule
1     Swap [0.065 0] 01-Jan-2007  01-Jan-2012 [NaN]    0     100       [NaN]   1
```

See Also   datestr | num2str | instaddfield | instget

Purpose   List field names

Syntax   FieldList = instfields(InstSet, 'Type', TypeList)

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
TypeList	(Optional) String or number of types (NTYPES)-by-1 cell array of strings listing the instrument types to query.

Description   FieldList = instfields(InstSet, 'Type', TypeList) retrieves the list of fields stored in an instrument variable.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field corresponding to the listed types.

Examples   Retrieve the instrument set ExampleInst from the data file InstSetExamples.mat. ExampleInst contains three types of instruments: Option, Futures, and TBill.

```
load InstSetExamples;
instdisp(ExampleInst)

Index Type    Strike Price Opt  Contracts
1     Option  95     12.2  Call     0
2     Option 100      9.2  Call     0
3     Option 105      6.8  Call  1000

Index Type    Delivery     F      Contracts
4     Futures 01-Jul-1999  104.4  -1000
```

```
Index Type    Strike Price Opt  Contracts
5     Option 105      7.4 Put  -1000
6     Option  95      2.9 Put      0

Index Type Price Maturity        Contracts
7     TBill 99    01-Jul-1999  6
```

Get the fields listed for type 'Option'.

```
[FieldList, ClassList] = instfields(ExampleInst, 'Type',...
'Option')

FieldList =

    'Strike'
    'Price'
    'Opt'
    'Contracts'

ClassList =

    'dble'
    'dble'
    'char'
    'dble'
```

Get the fields listed for types 'Option' and 'TBill'.

```
FieldList = instfields(ExampleInst, 'Type', {'Option', 'TBill'})

FieldList =

    'Strike'
    'Opt'
    'Price'
    'Maturity'
    'Contracts'
```

Get all the fields listed in any type in the variable.

```
FieldList = instfields(ExampleInst)

FieldList =

    'Delivery'
    'F'
    'Strike'
    'Opt'
    'Price'
    'Maturity'
    'Contracts'
```

See Also   instdisp | instlength | insttypes

instfind

Purpose	Search instruments for matching conditions
Syntax	`IndexMatch = instfind(InstSet, 'FieldName', FieldList, 'Data',` `DataList,'Index', IndexSet, 'Type', TypeList)`

Arguments

`InstSet`	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
`FieldList`	String or number of fields (`NFIELDS`)-by-1 cell array of strings listing the name of each data field to match with data values.
`DataList`	Number of values (`NVALUES`)-by-M array or `NFIELDS`-by-1 cell array of acceptable data values for each field. Each row lists a data row value to search for in the corresponding `FieldList`. The number of columns is arbitrary, and matching will ignore trailing `NaN`s or spaces.
`IndexSet`	(Optional) Number of instruments (`NINST`)-by-1 vector restricting positions of instruments to check for matches. The default is all indices available in the instrument variable.
`TypeList`	(Optional) String or number of types (`NTYPES`)-by-1 cell array of strings restricting instruments to match one of `TypeList` types. The default is all types in the instrument variable.

Argument value pairs can be entered in any order. The `InstSet` variable must be the first argument. `'FieldName'` and `'Data'` arguments must appear together or not at all.

Description   IndexMatch = instfind(InstSet, 'FieldName', FieldList,
'Data',
DataList,'Index', IndexSet, 'Type', TypeList) returns indices
of instruments matching Type, Field, or Index values.

IndexMatch is an NINST-by-1 vector of positions of instruments
matching the input criteria. Instruments are returned in IndexMatch if
all the Field, Index, and Type conditions are met. An instrument meets
an individual Field condition if the stored FieldName data matches any
of the rows listed in the DataList for that FieldName.

Examples   Retrieve the instrument set ExampleInst from the data file
InstSetExamples.mat. ExampleInst contains three types of
instruments: Option, Futures, and TBill.

```
load InstSetExamples;
instdisp(ExampleInst)

Index Type    Strike Price Opt  Contracts
1     Option 95     12.2 Call     0
2     Option 100     9.2 Call     0
3     Option 105     6.8 Call  1000

Index Type    Delivery       F      Contracts
4     Futures 01-Jul-1999   104.4 -1000

Index Type    Strike    Price Opt Contracts
5     Option 105      7.4  Put  -1000
6     Option 95       2.9  Put     0

Index Type  Price Maturity       Contracts
7     TBill 99    01-Jul-1999     6
```

Make a vector, Opt95, containing the indexes within ExampleInst of
the options struck at 95.

```
Opt95 = instfind(ExampleInst, 'FieldName','Strike','Data','95')
```

```
Opt95 =

     1
     6
```

Locate the futures and Treasury bill instruments within `ExampleInst`.

```
Types = instfind(ExampleInst,'Type',{'Futures';'TBill'})

Types =

     4
     7
```

See Also   instaddfield | instget | instgetcell | instselect

instfixed

Purpose	Construct fixed-rate instrument

Syntax

```
InstSet = instfixed(InstSet, CouponRate, Settle, Maturity,
Reset, Basis, Principal, EndMonthRule)
[FieldList, ClassList, TypeString] = instfixed
```

Arguments

InstSet	Instrument variable. This argument is specified only when adding fixed-rate note instruments to an existing instrument set. See instget for more information on the InstSet variable.
CouponRate	Decimal annual rate.
Settle	Settlement date. Date string or serial date number representing the settlement date of the fixed-rate note.
Maturity	Date string or serial date number representing the maturity date of the fixed-rate note.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)

instfixed

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal   (Optional) The notional principal amount. Default = `100`.

EndMonthRule   (Optional) NINST-by-1 vector representing the End-of-month rule. Default = `1`.

Data arguments are number of instruments (NINST)-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with `NaN`. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices `[]`.

Description   InstSet = instfixed(InstSet, CouponRate, Settle, Maturity, Reset, Basis, Principal, EndMonthRule) creates a new instrument set containing fixed-rate instruments or adds fixed-rate instruments to an existing instrument set.

[FieldList, ClassList, TypeString] = instfixed displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are `'dble'`, `'date'`, and `'char'`.

TypeString is a string specifying the type of instrument added. For a fixed-rate instrument, TypeString = `'Fixed'`.

See Also hjmprice | instaddfield | instbond | instcap | instdisp |
instswap | intenvprice

instfloat

Purpose	Construct floating-rate instrument

Syntax
```
InstSet = instfloat(InstSet, Spread, Settle, Maturity, Reset,
Basis, Principal, EndMonthRule)
InstSet = instfloat(InstSet, Spread, Settle, Maturity, Reset,
Basis, Principal, EndMonthRule, CapRate, FloorRate)
[FieldList, ClassList, TypeString] = instfloat
```

Arguments

InstSet	Instrument variable. This argument is specified only when adding floating-rate note instruments to an existing instrument set. See instget for more information on the InstSet variable.
Spread	Number of basis points over the reference rate.
Settle	Settlement date. Date string or serial date number representing the settlement date of the floating-rate note.
Maturity	Date string or serial date number representing the maturity date of the floating-rate note.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal	(Optional) The notional principal amount. Default = `100`.
EndMonthRule	(Optional) `NINST`-by-`1` vector representing the End-of-month rule. Default = `1`.
CapRate	(Optional) `NINST`-by-`1` decimal annual rate or `NINST`-by-`1` cell array, where each element is a `NumDates`-by-`2` cell array, and the cell array first column is dates, and the second column is associated cap rates. The date indicates the last day that the cap rate is valid.
FloorRate	(Optional) `NINST`-by-`1` decimal annual rate or `NINST`-by-`1` cell array, where each element is a `NumDates`-by-`2` cell array, and the cell array first column is dates, and the second column is associated floor rates. The date indicates the last day that the floor rate is valid.

Data arguments are number of instruments (`NINST`)-by-`1` vectors, scalar, or empty. Fill unspecified entries in vectors with `NaN`. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices `[ ]`.

instfloat

Description   InstSet = instfloat(InstSet, Spread, Settle, Maturity, Reset,
Basis, Principal, EndMonthRule) creates a new instrument set
containing floating-rate instruments or adds floating-rate instruments
to an existing instrument set.

InstSet = instfloat(InstSet, Spread, Settle, Maturity, Reset,
Basis, Principal, EndMonthRule, CapRate, FloorRate) creates
a new instrument set containing capped floating-rate instruments or
adds capped floating-rate instruments to an existing instrument set.

[FieldList, ClassList, TypeString] = instfloat displays the
classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings
listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class
of each field. The class determines how arguments are parsed. Valid
strings are 'dble', 'date', and 'char'.

TypeString is a string specifying the type of instrument added. For a
floating-rate instrument, TypeString = 'Float'.

See Also   hjmprice | instaddfield | instbond | instcap | instdisp |
instswap | intenvprice

Purpose   Construct floor instrument

Syntax   InstSet = instfloor(InstSet, Strike, Settle, Maturity, Reset,
               Basis, Principal)
               [FieldList, ClassList, TypeString] = instfloor

Arguments

InstSet	Instrument variable. This argument is specified only when adding floor instruments to an existing instrument set. See instget for more information on the InstSet variable.
Strike	Rate at which the floor is exercised, as a decimal number.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Reset	(Optional) NINST-by-1 vector representing the frequency of payments per year. Default = 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal   (Optional) The notional principal amount. Default = 100.

Description   InstSet = instfloor(InstSet, Strike, Settle, Maturity, Reset,
Basis, Principal) creates a new instrument set containing floor instruments or adds floor instruments to an existing instrument set.

[FieldList, ClassList, TypeString] = instfloor displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are 'dble', 'date', and 'char'.

TypeString is a string specifying the type of instrument added. For a floor instrument, TypeString = 'Floor'.

See Also   hjmprice | instaddfield | instbond | instcap | instdisp | instswap | intenvprice

Purpose   Data from instrument variable

Syntax   [Data_1, Data_2,...,Data_n] = instget(InstSet, 'FieldName',
                FieldList, 'Index', IndexSet, 'Type', TypeList)

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
FieldList	(Optional) String or number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field to match with data values. FieldList entries can also be either 'Type' or 'Index'; these return type strings and index numbers respectively. The default is all fields available for the returned set of instruments.
IndexSet	(Optional) Number of instruments (NINST)-by-1 vector of positions of instruments to work on. If TypeList is also entered, instruments referenced must be one of TypeList types and contained in IndexSet. The default is all indices available in the instrument variable.
TypeList	(Optional) String or number of types (NTYPES)-by-1 cell array of strings restricting instruments to match one of TypeList types. The default is all types in the instrument variable.

Argument value pairs can be entered in any order. The InstSet variable must be the first argument.

instget

Description   [Data_1, Data_2,...,Data_n] = instget(InstSet, 'FieldName', FieldList, 'Index', IndexSet, 'Type', TypeList) retrieves data arrays from an instrument variable.

Data_1 is an NINST-by-M array of data contents for the first field in FieldList. Each row corresponds to a separate instrument in IndexSet. Unavailable data is returned as NaN or as spaces.

Data_n is an NINST-by-M array of data contents for the last field in FieldList.

Examples   Retrieve the instrument set ExampleInst from the data file. InstSetExamples.mat. ExampleInst contains three types of instruments: Option, Futures, and TBill.

```
load InstSetExamples;
instdisp(ExampleInst)

Index Type    Strike Price Opt  Contracts
1     Option  95     12.2  Call    0
2     Option 100      9.2  Call    0
3     Option 105      6.8  Call  1000

Index Type    Delivery        F     Contracts
4     Futures 01-Jul-1999  104.4 -1000

Index Type    Strike Price Opt  Contracts
5     Option 105      7.4 Put  -1000
6     Option  95      2.9 Put     0

Index Type Price Maturity        Contracts
7     TBill 99    01-Jul-1999    6
```

Extract the price from all instruments.

```
P = instget(ExampleInst,'FieldName','Price')

P =
```

```
    12.2000
     9.2000
     6.8000
         NaN
     7.4000
     2.9000
    99.0000
```

Get all the prices and the number of contracts held.

```
[P,C] = instget(ExampleInst, 'FieldName', {'Price', 'Contracts'})

P =

    12.2000
     9.2000
     6.8000
         Nan
     7.4000
     2.9000
    99.0000

C =

         0
         0
      1000
     -1000
     -1000
         0
         6
```

Compute a value V. Create a new variable ISet that appends V to
ExampleInst.

```
V = P.*C
ISet = instsetfield(ExampleInst, 'FieldName', 'Value', 'Data',...
```

```
V);
instdisp(ISet)

Index Type    Strike Price Opt  Contracts Value
1      Option  95     12.2 Call    0          0
2      Option 100      9.2 Call    0          0
3      Option 105      6.8 Call  1000       6800

Index Type    Delivery        F    Contracts Value
4      Futures 01-Jul-1999   104.4 -1000     NaN

Index Type    Strike Price Opt  Contracts Value
5      Option 105      7.4 Put  -1000      -7400
6      Option  95      2.9 Put     0          0

Index Type Price Maturity       Contracts Value
7      TBill 99    01-Jul-1999   6           594
```

Look at only the instruments that have nonzero `Contracts`.

```
Ind = find(C ~= 0)

Ind =

     3
     4
     5
     7
```

Get the `Type` and `Opt` parameters from those instruments. (Only options have a stored `'Opt'` field.)

```
[T,O] = instget(ExampleInst, 'Index', Ind, 'FieldName',...
{'Type', 'Opt'})

T =

Option
```

```
Futures
Option
TBill

O =

Call

Put
```

Create a string report of holdings `Type`, `Opt`, and `Value`.

```
rstring = [T, O, num2str(V(Ind))]

rstring =

Option Call    6800
Futures         NaN
Option Put    -7400
TBill           594
```

See Also   instaddfield | instdisp | instgetcell

instgetcell

Purpose	Data and context from instrument variable

Syntax

```
[DataList, FieldList, ClassList] =
instgetcell(InstSet,  'FieldName', FieldList, 'Index',
IndexSet, 'Type', TypeList)
```

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
FieldList	(Optional) String or number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field to match with data values. FieldList should not be either Type or Index; these field names are reserved. The default is all fields available for the returned set of instruments.
IndexSet	(Optional) Number of instruments (NINST)-by-1 vector of positions of instruments to work on. If TypeList is also entered, instruments referenced must be one of TypeList types and contained in IndexSet. The default is all indices available in the instrument variable.
TypeList	(Optional) String or number of types (NTYPES)-by-1 cell array of strings restricting instruments to match one of TypeList types. The default is all types in the instrument variable.

Argument value pairs can be entered in any order. The InstSet variable must be the first argument.

Description

```
[DataList, FieldList, ClassList] =
instgetcell(InstSet, 'FieldName', FieldList, 'Index',
```

IndexSet, 'Type', TypeList) retrieves data and context from an instrument variable.

DataList is an NFIELDS-by-1 cell array of data contents for each field. Each cell is an NINST-by-M array, where each row corresponds to a separate instrument in IndexSet. Any data which is not available is returned as NaN or as spaces.

FieldList is an NFIELDS-by-1 cell array of strings listing the name of each field in DataList.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are 'dble', 'date', and 'char'.

IndexSet is an NINST-by-1 vector of positions of instruments returned in DataList.

TypeSet is an NINST-by-1 cell array of strings listing the type of each instrument row returned in DataList.

Examples   Retrieve the instrument set ExampleInst from the data file InstSetExamples.mat. ExampleInst contains three types of instruments: Option, Futures, and TBill.

```
load InstSetExamples;
instdisp(ExampleInst)

Index Type    Strike Price Opt  Contracts
1     Option  95    12.2  Call     0
2     Option 100     9.2  Call     0
3     Option 105     6.8  Call  1000

Index Type    Delivery        F     Contracts
4     Futures 01-Jul-1999   104.4  -1000

Index Type    Strike Price Opt  Contracts
5     Option 105     7.4  Put  -1000
6     Option  95     2.9  Put      0
```

```
Index Type Price Maturity      Contracts
7     TBill 99    01-Jul-1999  6
```

Get the prices and contracts from all instruments.

```
FieldList = {'Price'; 'Contracts'}
DataList = instgetcell(ExampleInst, 'FieldName', FieldList )
P = DataList{1}
C = DataList{2}

P =

   12.2000
    9.2000
    6.8000
        NaN
    7.4000
    2.9000
   99.0000

C =

       0
       0
    1000
   -1000
   -1000
       0
       6
```

Get all the option data: Strike, Price, Opt, Contracts.

```
[DataList, FieldList, ClassList] = instgetcell(ExampleInst,...
'Type','Option')

DataList =

    [5x1 double]
```

```
    [5x1 double]
    [5x4 char  ]
    [5x1 double]

FieldList =

    'Strike'
    'Price'
    'Opt'
    'Contracts'

ClassList =

    'dble'
    'dble'
    'char'
    'dble'
```

Look at the data as a comma-separated list. Type `help lists` for more information on cell array lists.

```
DataList{:}

ans =

     95
    100
    105
    105
     95

ans =

    12.2100
     9.2000
     6.8000
     7.3900
```

```
      2.9000

ans =

   Call
   Call
   Call
   Put
   Put

ans =

      0
      0
    100
   -100
      0
```

See Also   instaddfield | instdisp | instget

Purpose   Count instruments

Syntax   NInst = instlength(InstSet)

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.

Description   NInst = instlength(InstSet) computes NInst, the number of instruments contained in the variable, InstSet.

See Also   instdisp | instfields | insttypes

instlookback

Purpose	Construct lookback option
Syntax	InstSet = instlookback(InstSet, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt) [FieldList, ClassList, TypeString] = instlookback

Arguments

InstSet	Instrument variable. This argument is specified only when adding lookback instruments to an existing instrument set. See instget for more information on the InstSet variable.
OptSpec	NINST-by-1 list of string values 'Call' or 'Put'.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option.
Settle	NINST-by-1 vector of Settle dates.
ExerciseDates	For a European option (AmericanOpt = 0): NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date. For an American option (AmericanOpt = 1): NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the valuation date of the stock tree and the single listed exercise date.
AmericanOpt	(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If

AmericanOpt = 1, the option is an American option.

Data arguments are number of instruments (NINST)-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Description   InstSet = instlookback(InstSet, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt) specifies a lookback option.

[FieldList, ClassList, TypeString] = instlookback displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are 'dble', 'date', and 'char'.

TypeString is a string specifying the type of instrument added. For a lookback option instrument, TypeString = 'Lookback'.

Examples   **Create a Lookback Option Instrument**

Define a floating strike lookback instrument with the following data:

```
OptSpec = 'call';
Strike = NaN;
Settle = '01-Jan-2012';
ExerciseDates = '01-Jan-2015';
```

Create the instrument set.

```
InstSet = instlookback(OptSpec, Strike, Settle, ExerciseDates);
```

Display the lookback instrument.

instlookback

```
instdisp(InstSet)

Index Type     OptSpec Strike Settle      ExerciseDates AmericanOpt
1     Lookback call    NaN    01-Jan-2012 01-Jan-2015   0
```

See Also instadd | instdisp | instget

Purpose   Construct bond option

Syntax   InstSet = instoptbnd(InstSet, BondIndex, OptSpec, Strike,
                ExerciseDates)
                InstSet = instoptbnd(InstSet, BondIndex, OptSpec, Strike,
                ExerciseDates, AmericanOpt)
                [FieldList, ClassList, TypeString] = instoptbnd

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
BondIndex	Number of instruments (NINST)-by-1 vector of indices pointing to underlying instruments of Type 'Bond' which are also stored in InstSet. See instbond for information on specifying the bond data.
OptSpec	NINST-by-1 list of string values 'Call' or 'Put'.

Note The interpretation of the Strike and ExerciseDates arguments depends upon the setting of the AmericanOpt argument. If AmericanOpt = 0, NaN, or is unspecified, the option is a European or Bermuda option. If AmericanOpt = 1, the option is an American option.

Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	For an American option:
	NINST-by-1 vector of strike price values for each option.
ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.

Data arguments are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Description   `InstSet = instoptbnd(InstSet, BondIndex, OptSpec, Strike, ExerciseDates)` specifies a European or Bermuda option.

`InstSet = instoptbnd(InstSet, BondIndex, OptSpec, Strike, ExerciseDates, AmericanOpt)` specifies an American option if `AmericanOpt` is set to `1`. If `AmericanOpt` is not set to `1`, the function specifies a European or Bermuda option.

`[FieldList, ClassList, TypeString] = instoptbnd` displays the classes.

`FieldList` is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

`ClassList` is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are `'dble'`, `'date'`, and `'char'`.

`TypeString` is a string specifying the type of instrument added. For a bond option instrument, `TypeString = 'OptBond'`.

See Also   hjmprice | instadd | instdisp | instget

instoptembnd

Purpose	Construct bond with embedded option

Syntax

```
InstSet = instoptembnd (CouponRate, Settle, Maturity,
OptSpec, Strike, ExerciseDates, 'AmericanOpt',
AmericanOpt, 'Period', Period, 'Basis', Basis,
'EndMonthRule', EndMonthRule,'Face', Face, 'IssueDate',
IssueDate, 'FirstCouponDate', FirstCouponDate,
'LastCouponDate', LastCouponDate,'StartDate',StartDate)
InstSet = instoptembnd(InstSetOld, CouponRate,...)
[FieldList, ClassList, TypeString] = instoptembnd
```

Arguments

CouponRate	Decimal annual rate indicating the annual percentage rate used to determine the coupons payable on a bond. CouponRate is a NINST-by-1 vector or NINST-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	NINST-by-1 vector of settlement dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 vector of string values 'Call' or 'Put'.

For a European or Bermuda option

Strike	NINST-by-NSTRIKES matrix of strike price values. Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaN's.

ExerciseDates	NINST-by-NSTRIKES matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one ExerciseDate on the option expiry date.
AmericanOpt	(Optional) NINST-by-1 vector of flags. AmericanOpt is 0 for each European or Bermuda option. The default is 0 if AmericanOpt is NaN or not entered.

For an American option

Strike	NINST-by-1 vector of strike price values for each option.
ExerciseDates	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed ExerciseDate.
AmericanOpt	NINST-by-1 vector of flags. AmericanOpt is 1 for each American option. The AmericanOpt argument is required to invoke American exercise rules.
Period	(Optional) NINST-by-1 matrix for coupons per year. The default value is 2.

Basis	(Optional) Day-count basis of the instrument. Basis is a vector of integers with the following possible values:

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) NINST-by-1 matrix for the end-of-month rule. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. When the value is 0, the end-of-month rule is ignored, meaning that a bond's coupon payment date is always the same numerical day of the month. When the value is 1, the end-of-month rule is set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

IssueDate	(Optional) NINST-by-1 matrix for the bond issue date.
FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure. If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.
StartDate	(Optional) NINST-by-1 matrix for date when a bond actually starts (i.e. the date from which a bond's cash flows can be considered). To make an instrument forward starting, specify this date as a future date. If StartDate is not explicitly specified, the effective start date is the Settle date.
Face	(Optional) Face value. Face is a NINST-by-1 vector or NINST-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated face value. The

date indicates the last day that the face value
is valid. Default is 100.

Note Data arguments are NINST-by-1 vectors, scalar, or empty. Fill
unspecified entries in vectors with NaN. Only one data argument is
required to create the instrument. The others may be omitted or passed
as empty matrices [].

Description   InstSet = instoptembnd (CouponRate, Settle, Maturity,
OptSpec, Strike, ExerciseDates, 'AmericanOpt',
AmericanOpt, 'Period', Period, 'Basis', Basis,
'EndMonthRule', EndMonthRule,'Face', Face, 'IssueDate',
IssueDate, 'FirstCouponDate', FirstCouponDate,
'LastCouponDate', LastCouponDate,'StartDate',StartDate)
creates InstSet, a variable containing a collection of instruments.

Note instopembnd uses optional parameter name/value pairs such
that, 'Name1', Value1, 'Name2', Value2, and so on, are a variable
length list of name/value pairs.

Instruments are broken down by type and each type can have different
data fields. Each stored data field has a row vector or string for each
instrument. See instget for more information on the InstSet variable.

InstSet = instoptembnd(InstSetOld, CouponRate,...) adds
'OptEmBond' instruments to an instrument variable.

[FieldList, ClassList, TypeString] = instoptembnd lists field
metadata for the 'OptEmBond' instrument.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings
listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are `'dble'`, `'date'`, and `'char'`.

TypeString is a string specifying the type of instrument added. For a bond option instrument, TypeString = `'OptEmBond'`.

Examples   To create a bond with embedded options with the following data:

```
Settle = 'jan-1-2007';
Maturity  = 'jan-1-2010';
CouponRate = 0.07;
OptSpec = 'call';
Strike= 100;
ExerciseDates= {'jan-1-2008' '01-Jan-2010'};
AmericanOpt=1;
Period = 1;

InstSet = instoptembnd(CouponRate, ...
Settle, Maturity, OptSpec, Strike,  ExerciseDates,'AmericanOpt', AmericanOpt, ...
'Period', Period);
```

To display the instrument:

```
 instdisp(InstSet)
```

See Also   instadd | instdisp | instget

instoptfloat

Purpose	Create option instrument on floating-rate note or add instrument to current portfolio

Syntax	InstSet = instoptfloat(FloatIndex,OptSpec,Strike,
	   ExerciseDates)
	InstSet = instoptfloat(FloatIndex,OptSpec,Strike,
	   ExerciseDates,AmericanOpt)
	
	InstSet = instoptfloat(InstSetOld, ___)
	
	[FieldList,ClassList,TypeString] = instoptfloat

Description	InstSet = instoptfloat(FloatIndex,OptSpec,Strike, ExerciseDates) to specify an European option for a floating-rate note. InstSet = instoptfloat(FloatIndex,OptSpec,Strike, ExerciseDates,AmericanOpt) to specify an American or Bermuda option for a floating-rate note. InstSet = instoptfloat(InstSetOld, ___) to add instruments to an existing portfolio. [FieldList,ClassList,TypeString] = instoptfloat lists the field metadata for the 'OptFloat' instrument.
Input Arguments	**FloatIndex - Indices pointing to underlying instruments** vector of nonnegative integers Indices pointing to underlying instruments of Type 'Float' specified by a NINST-by-1 vector. The instruments of Type 'Float' are also stored in the InstSet variable. For more information, see instfloat. **Data Types** double **OptSpec - Definition of option**

string | cell array of strings

Definition of option as `'call'` or `'put'` specified as a `NINST`-by-1 cell array of strings for `'call'` or `'put'`.

Data Types
`char` | `cell`

Strike - Option strike price values for European, Bermuda, or American option

nonnegative integer | vector of nonnegative integers

Option strike price values for option (European, Bermuda or American) specified as nonnegative integers using as `NINST`-by-`NSTRIKES` vector of strike price values.

- For a European or Bermuda option — `NINST`-by-`NSTRIKES` matrix of strike price values. Each row is the schedule for one option. If an option has fewer than `NSTRIKES` exercise opportunities, the end of the row is padded with `NaNs`.

- For an American Option — `NINST`-by-1 vector of strike price values for each option.

Data Types
`single` | `double`

ExerciseDates - Exercise date for option (European, Bermuda, or American)

nonnegative integer | vector of nonnegative integers

Exercise date for option (European, Bermuda, or American) specified as nonnegative integers using a `NINST`-by-`NSTRIKES` or `NINST`-by-2 vector of for the option exercise dates, depending on the option type.

- For a European or Bermuda option — `NINST`-by-`NSTRIKES` matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one `ExerciseDate` on the option expiry date

- For an American option — NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle date and the single listed ExerciseDate.

Data Types
char | cell

AmericanOpt - Option type
0 if AmericanOpt is NaN or not entered (default) | scalar | vector of positive integers[0,1]

Option type specified as NINST-by-1 positive integer scalar flags with values.

- For an European or Bermuda option — AmericanOpt is 0 for each European or Bermuda option.

- For an American option — AmericanOpt is 1 for each American option. The AmericanOpt argument is required to invoke American exercise rules.

Data Types
single | double

Output Arguments

InstSet - Variable containing a collection of instruments
scalar | vector

Variable containing a collection of instruments returned as a scalar or vector with the instruments broken down by type and each type can have different data fields. Each stored data field has a row vector or string for each instrument. For more information on the InstSet variable, see instget.

FieldList - Data field for instrument type
string | cell array

Data field for instrument type returned as a NFIELDS-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList - Data class of each field
string with value: 'dble', 'date', 'char' | cell array with values: 'dble', 'date', 'char'

Data class of each field returned as aNFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments will be parsed.

TypeString - Type of instrument added
string with value 'OptFloat'

Type of instrument added returned as a string. The string for a floating-rate option instrument is TypeString = 'OptFloat'.

Examples   **Create an Instrument Portfolio with a Call Option for a Floating-Rate Note**

Define the floating-rate note:

```
Settle = 'Nov-1-2012';
Maturity  = 'Nov-1-2015';
Spread = 50;
Reset = 1;
```

Create InstSet:

```
InstSet = instfloat(Spread, Settle, Maturity, Reset)

InstSet =

        FinObj: 'Instruments'
    IndexTable: [1x1 struct]
          Type: {'Float'}
     FieldName: {{9x1 cell}}
    FieldClass: {{9x1 cell}}
     FieldData: {{9x1 cell}}
```

instoptfloat

Display the instrument:

```
instdisp(InstSet)
```

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	EndMonthRule	CapRate	FloorRat
1	Float	50	01-Nov-2012	01-Nov-2015	1	0	100	1	Inf	-Inf

Add a European call option to the instrument portfolio:

```
OptSpec = 'call';
Strike = 100;
ExerciseDates = 'Nov-1-2015';
```

Create `InstSet`:

```
InstSet = instoptfloat(InstSet, 1, OptSpec, Strike, ExerciseDates)

InstSet =

        FinObj: 'Instruments'
     IndexTable: [1x1 struct]
           Type: {2x1 cell}
      FieldName: {2x1 cell}
     FieldClass: {2x1 cell}
      FieldData: {2x1 cell}
```

Display the instrument:

```
instdisp(InstSet)
```

Index	Type	Spread	Settle	Maturity	FloatReset	Basis	Principal	EndMonthRule	CapRate	FloorRat
1	Float	50	01-Nov-2012	01-Nov-2015	1	0	100	1	Inf	-Inf

Index	Type	UnderInd	OptSpec	Strike	ExerciseDates	AmericanOpt
2	OptFloat	1	call	100	01-Nov-2015	0

See Also   instadd | instoptemfloat

Purpose	Create embedded option instrument on floating-rate note or add instrument to current portfolio

Syntax

InstSet = instopemtfloat(Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates)
InstSet = instopemtfloat(___ ,Name,Value)

InstSet = instoptfloat(InstSetOld,Spread,Settle,Maturity,
OptSpec,Strike,ExerciseDates)

InstSet = instoptfloat(___ ,Name,Value)

[FieldList,ClassList,TypeString] = instoptfloat

Description

InstSet = instopemtfloat(Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates) creates an embedded option instrument for a floating-rate note.

InstSet = instopemtfloat(___ ,Name,Value) creates an embedded option instrument for a floating-rate note using optional name-value pair arguments.

InstSet = instoptfloat(InstSetOld,Spread,Settle,Maturity,
OptSpec,Strike,ExerciseDates) adds embedded option instruments for a floating-rate note instruments to an existing portfolio.

InstSet = instoptfloat(___ ,Name,Value) adds embedded option instruments for a floating-rate note to an existing portfolio using optional name-value pair arguments.

[FieldList,ClassList,TypeString] = instoptfloat lists field metadata for the 'OptEmFloat' instrument.

instoptemfloat

Input Arguments

Spread - Number of basis points over the reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate specified as a vector nonnegative integers for the number of instruments (NINST)-by-1).

Data Types
single | double

Settle - Settlement dates of floating-rate note
ValuationDate of HW Tree (default) | nonnegative integer | vector of nonnegative integers

Settlement dates of floating-rate note specified as nonnegative integers using a NINST-by-1 vector of dates.

Data Types
char

Maturity - Floating-rate note maturity date
nonnegative integer | vector of nonnegative integers

Floating-rate note maturity date specified as nonnegative integers using a NINST-by-1 vector of dates.

Data Types
char

OptSpec - Definition of option
string | cell array of strings

Definition of option as 'call' or 'put' specified as a NINST-by-1 cell array of strings for 'call' or 'put'.

Data Types
char | cell

Strike - Embedded option strike price values
nonnegative integer | vector of nonnegative integers

Embedded option strike price values for option specified as nonnegative integers using as `NINST-by-NSTRIKES` or `NINST-by-1` vector of strike price values, depending on the type of option.

- For a European or Bermuda Option — `NINST-by-NSTRIKES` matrix of strike price values where each row is the schedule for one option. If an option has fewer than `NSTRIKES` exercise opportunities, the end of the row is padded with `NaNs`.

- For an American Option — `NINST-by-1` vector of strike price values for each option.

Data Types
single | double

ExerciseDates - Exercise date for embedded option
nonnegative integer | vector of nonnegative integers

Exercise date for embedded option specified as nonnegative integers using a `NINST-by-NSTRIKES` or `NINST-by-2` vector of the option exercise dates, depending on the type of option.

- For a European or Bermuda Option — `NINST-by-NSTRIKES` of exercise dates where each row is the schedule for one option. For a European option, there is only one `ExerciseDate` on the option expiry date.

- For an American Option — `NINST-by-2` vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-`NaN` date is listed, or if `ExerciseDates` is `NINST-by-1`, the option can be exercised between the underlying bond `Settle` date and the single listed `ExerciseDate`.

Data Types
char | cell

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can

instoptemfloat

specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

Example: InstSet =
instoptemfloat(Spread,Settle,Maturity,OptSpec,Strike,ExerciseDates,'Reset

'AmericanOpt' - Embedded option type
0 if AmericanOpt is NaN or not entered (default) | scalar | vector of
positive integers[0,1]

Embedded option type specified as NINST-by-1 positive integer scalar
flags with values.

- For a European or Bermuda option — AmericanOpt is 0 for each
  European or Bermuda option. The default is 0 if AmericanOpt is
  NaN or not entered.

- For an American option — AmericanOpt is 1 for each American
  option. The AmericanOpt argument is required to invoke American
  exercise rules.

Data Types
single | double

'Reset' - Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6, 12] | vector of
positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the
values 1,2,4,6,12] in a NINST-by-1 vector.

Data Types
single | double

'Basis' - Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

Data Types
single | double

'Principal' - Principal values

100 (default) | nonnegative integer | vector of nonnegative integers | cell array of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1 vector of notional principal amounts.

Data Types
single | double

'Options' - Structure containing derivatives pricing options

structure

Structure containing derivatives pricing options specified using `derivset`.

Data Types
`struct`

'EndMonthRule' - End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as a nonnegative integer [0, 1] using a `NINST-by-1` vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
`single | double`

'CapRate' - Indicates the last day that the cap rate is valid
nonnegative decimal | cell array of nonnegative decimals

`NINST-by-1` decimal annual rate or `NINST-by-1` cell array where each element is a `NumDates-by-2` cell array where the first column is dates and the second column is associated cap rates.

Data Types
`single | double`

'FloorRate' - Indicates the last day that the floor rate is valid
nonnegative decimal | cell array of nonnegative decimals

`NINST-by-1` decimal annual rate or `NINST-by-1` cell array where each element is a `NumDates-by-2` cell array where the first column is dates and the second column is associated floor rates.

Data Types
`single | double`

Output Arguments

InstSet - Variable containing a collection of instruments
scalar | vector

Variable containing a collection of instruments returned as a scalar or vector with the instruments broken down by type and each type can have different data fields. Each stored data field has a row vector or string for each instrument. For more information on the InstSet variable, see instget.

FieldList - Name of each data field
string | cell array of strings

NFIELDS-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList - Determines how arguments are parsed
string with value: 'dble', 'date', or 'char' | cell array of strings with values: 'dble', 'date', or 'char'

NFIELDS-by-1 cell array of strings listing the data class of each field.

TypeString - Type of instrument added
string with value 'OptEmFloat'

String specifying the type of instrument added where TypeString = 'OptEmFloat'.

Examples

Create an Instrument Portfolio with a Embedded Option Floating-Rate Note

Define the embedded call option:

```
Settle = 'Nov-1-2012';
Maturity  = 'Nov-1-2015';
Spread = 25;
OptSpec = 'call';
Strike= 100;
ExerciseDates = 'Nov-1-2015';
Reset = 1;
```

Create `InstSet`:

```
InstSet = instoptemfloat(Spread, Settle, Maturity, OptSpec,...
Strike,  ExerciseDates,'Reset', Reset)

InstSet =

      FinObj: 'Instruments'
   IndexTable: [1x1 struct]
        Type: {'OptEmFloat'}
   FieldName: {{13x1 cell}}
  FieldClass: {{13x1 cell}}
   FieldData: {{13x1 cell}}
```

Display the instrument:

```
instdisp(InstSet)
```

```
Index Type Spread Settle     Maturity    OptSpec Strike ExerciseDates  FloatReset Basis Principal EndMont
1 OptEmFloat 25  01-Nov-2012 01-Nov-2015   call    100   01-Nov-2015    1          0     100       1
```

See Also   instadd | instoptfloat

Concepts   • **basis** on page Glossary-1

Purpose	Construct stock option

Syntax
```
InstSet = instoptstock(InstSet, OptSpec, Strike,
Settle, ExerciseDates)
InstSet = instoptstock(InstSet, OptSpec, Strike,
Settle, ExerciseDates, AmericanOpt)
[FieldList, ClassList, TypeString] = instoptstock
```

Arguments

InstSet	Instrument variable. This argument is specified only when adding stock option instruments to an existing instrument set. See instget for more information on the InstSet variable.
OptSpec	NINST-by-1 list of string values 'Call' or 'Put'.

Note The interpretation of the Strike and ExerciseDates arguments depends upon the setting of the AmericanOpt argument. If AmericanOpt = 0, NaN, or is unspecified, the option is a European or Bermuda option. If AmericanOpt = 1, the option is an American option.

Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	American option: NINST-by-1 vector of strike price values for each option.

instoptstock

Settle	NINST-by-1 vector of settlement dates.
ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.

Data arguments are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Description   InstSet = instoptstock(InstSet, OptSpec, Strike, Settle, ExerciseDates) specifies a European or Bermuda option.

InstSet = instoptstock(InstSet, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt) specifies an American option if AmericanOpt is set to 1. If AmericanOpt is not set to 1, the function specifies a European or Bermuda option.

[FieldList, ClassList, TypeString] = instoptstock displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are `'dble'`, `'date'`, and `'char'`.

TypeString is a string specifying the type of instrument added. For a stock option instrument, TypeString = `'OptStock'`.

Examples   **Create a Stock Option Instrument**

Create an instrument set of two stock options with the following data:

```
OptSpec = {'put';'call'};
Strike = [95;98];
Settle = '01-May-2012';
ExerciseDates = {'01-May-2014';'01-May-2015'};
AmericanOpt = [0;1];
```

Create the stock option instruments.

```
InstSet = instoptstock(OptSpec, Strike,Settle, ExerciseDates, American

InstSet =

        FinObj: 'Instruments'
     IndexTable: [1x1 struct]
          Type: {'OptStock'}
     FieldName: {{5x1 cell}}
    FieldClass: {{5x1 cell}}
     FieldData: {{5x1 cell}}
```

Display the instrument set.

```
instdisp(InstSet)
```

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt
1	OptStock	put	95	01-May-2012	01-May-2014	0
2	OptStock	call	98	01-May-2012	01-May-2015	1

See Also   instadd | instdisp | instget

instrangefloat

Purpose   Construct range note instrument

Syntax   ISet = instrangefloat(Spread,Settle,Maturity,RateSched,
                   Reset,Basis,Principal,EndMonthRule)

Description   ISet = instrangefloat(Spread,Settle,Maturity,RateSched,
                   Reset,Basis,Principal,EndMonthRule) creates a new range
                   instrument from data arrays.

Input   **Spread**
Arguments
                   Number of basis points over the reference rate.

                   Settle

                   NINST-by-1 vector of dates representing the settle date of the
                   floating-rate note.

                   Maturity

                   NINST-by-1 vector of dates representing the maturity date of the
                   floating-rate note.

                   RateSched

                   NINST-by-1 vector of structures representing the range of rates within
                   which cash flows are nonzero. Each element of the structure array
                   contains two fields:

                   • RateSched.Dates — NDates-by-1 cell array of dates corresponding to
                     the range schedule.

                   • RateSched.Rates —NDates-by-2 array with the first column
                     containing the lower bound of the range and the second
                     column containing the upper bound of the range. Cash flow
                     for date RateSched.Dates(n) is nonzero for rates in the range
                     RateSched.Rates(n,1) < Rate < RateSched.Rate (n,2).

                   Reset

(Optional) `NINST`-by-1 vector representing the frequency of payments per year.

> **Default:** 1

Basis

(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

(Optional) `NINST`-by-1 vector of the notional principal amount.

> **Default:** 100

EndMonthRule

(Optional) NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

Default: 1 (in effect)

Note Data arguments are number of instruments NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. You can omit or pass the others as empty matrices [ ]. However, you cannot price the instrument when using the range note pricing function if you are missing any of the required input arguments.

Output Arguments

ISet

Variable containing a collection of instruments. Instruments are divided by type and each type can have different data fields. Each stored data field has a row vector or string for each instrument. Values are:

- FieldList — NFIELDS-by-1 cell array of strings listing the name of each data field for this instrument type.

- ClassList — NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are'dble', 'date', and 'char'.

- TypeString — String specifying the type of instrument added. TypeString = 'RangeFloat'.

For more information, on ISet see instget.

Definitions

Range Note Instrument

A range note is a structured (market-linked) security whose coupon rate is equal to the reference rate as long as the reference rate is within a certain range. If the reference rate is outside of the range, the coupon rate is 0 for that period. This type of instrument entitles the holder

to cash flows that depend on the level of some reference interest rate and are floored to be positive. The note holder gets directs exposure to the reference rate. In return for the drawback that no interest will be paid for the time the range is left, they offer higher coupon rates than comparable standard products, like vanilla floating notes.

Examples   Create a range note instrument:

```
% Create an instrument portfolio with a range note:
Spread = 100;
Settle = 'Jan-1-2011';
Maturity = 'Jan-1-2014';

RateSched.Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
RateSched.Rates  = [0.045 0.055; 0.0525  0.0675; 0.06 0.08];

% Create InstSet
InstSet = instrangefloat(Spread, Settle, Maturity, RateSched);

% Display the portfolio instrument
instdisp(InstSet)
```

Index	Type	Spread	Settle	Maturity	RateSched	FloatReset	Basis	Principal	EndMonthRule
1	RangeFloat	100	01-Jan-2011	01-Jan-2014	[Struct]	1	0	100	1

```
% Add another range note
% Second Range Note:
Spread2 = 200;
Settle2 = 'Jan-1-2011';
Maturity2 = 'Jan-1-2013';
RateSched2.Dates = {'Jan-1-2012'; 'Jan-1-2013'};
RateSched2.Rates  = [0.048 0.059; 0.055  0.068];

InstSet = instrangefloat(InstSet, Spread2, Settle2, Maturity2, RateSch

% Display the portfolio instrument
instdisp(InstSet)
```

instrangefloat

```
    Index Type      Spread Settle     Maturity    RateSched FloatReset Basis Principal EndMonthRule

    1  RangeFloat 100  01-Jan-2011  01-Jan-2014  [Struct]  1        0     100       1

    2  RangeFloat 200  01-Jan-2011  01-Jan-2013  [Struct]  1        0     100       1
```

References   Jarrow, Robert, *Modelling Fixed Income Securities and Interest Rate Options*, Stanford Economics and Finance, 2nd edition, 2002.

See Also | instbond | instcap | instswap | instaddfield | instdisp | intenvprice | rangefloatbybk | rangefloatbybdt | rangefloatbyhw | rangefloatbyhjm

Purpose   Create instrument subset by matching conditions

Syntax   InstSubSet = instselect(InstSet, 'FieldName', FieldList,
                'Data', DataList, 'Index', IndexSet,'Type', TypeList)

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.
FieldList	String or number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field to match with data values.
DataList	Number of values (NVALUES)-by-M array or NFIELDS-by-1 cell array of acceptable data values for each field. Each row lists a data row value to search for in the corresponding FieldList. The number of columns is arbitrary and matching will ignore trailing NaNs or spaces.
IndexSet	(Optional) Number of instruments (NINST)-by-1 vector restricting positions of instruments to check for matches. The default is all indices available in the instrument variable.
TypeList	(Optional) String or number of types (NTYPES)-by-1 cell array of strings restricting instruments to match one of TypeList types. The default is all types in the instrument variable.

Argument value pairs can be entered in any order. The InstSet variable must be the first argument. 'FieldName' and 'Data' arguments must appear together or not at all. 'Index' and 'Type' arguments are each optional.

11-557

instselect

Description InstSubSet = instselect(InstSet, 'FieldName', FieldList, 'Data', DataList, 'Index', IndexSet,'Type', TypeList) creates an instrument subset (InstSubSet) from an existing set of instruments (InstSet).

InstSubSet is a variable containing instruments matching the input criteria. Instruments are returned in InstSubSet if all the Field, Index, and Type conditions are met. An instrument meets an individual Field condition if the stored FieldName data matches any of the rows listed in the DataList for that FieldName. See instfind for examples on matching criteria.

Examples Retrieve the instrument set ExampleInst from the data file InstSetExamples.mat. The variable contains three types of instruments: Option, Futures, and TBill.

```
load InstSetExamples
instdisp(ExampleInst)
```

```
Index Type   Strike Price Opt  Contracts
1     Option 95     12.2  Call     0
2     Option 100     9.2  Call     0
3     Option 105     6.8  Call  1000

Index Type    Delivery       F    Contracts
4     Futures 01-Jul-1999   104.4 -1000

Index Type   Strike  Price Opt Contracts
5     Option 105     7.4  Put  -1000
6     Option  95     2.9  Put      0

Index Type  Price Maturity       Contracts
7     TBill 99    01-Jul-1999     6
```

Make a new portfolio containing only options struck at 95.

```
Opt95 = instselect(ExampleInst, 'FieldName', 'Strike',...
'Data', '95')
```

```
instdisp(Opt95)

Opt95 =

Index Type    Strike Price Opt  Contracts
1     Option 95      12.2 Call    0
2     Option 95       2.9 Put     0
```

Make a new portfolio containing only futures and Treasury bills.

```
FutTBill = instselect(ExampleInst,'Type',{'Futures';'TBill'})

instdisp(FutTBill) =

Index Type    Delivery       F      Contracts
1     Futures 01-Jul-1999   104.4 -1000

Index Type  Price Maturity       Contracts
2     TBill 99    01-Jul-1999     6
```

See Also   instaddfield | instdelete | instfind | instget | instgetcell

instsetfield

Purpose	Add or reset data for existing instruments
Syntax	`InstSet = instsetfield(InstSet, 'FieldName', FieldList, 'Data', DataList)` `InstSet = instsetfield(InstSet, 'FieldName', FieldList, 'Data', DataList, 'Index', IndexSet, 'Type', TypeList)`

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument. `InstSet` must be the first argument in the list.
FieldList	String or number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field. `FieldList` cannot be named with the reserved names `Type` or `Index`.
DataList	Number of instruments (NINST)-by-M array or NFIELDS-by-1 cell array of data contents for each field. Each row in a data array corresponds to a separate instrument. Single rows are copied to apply to all instruments to be worked on. The number of columns is arbitrary, and data is padded along columns.
IndexSet	NINST-by-1 vector of positions of instruments to work on. If `TypeList` is also entered, instruments referenced must be one of `TypeList` types and contained in `IndexSet`.
TypeList	String or number of types (NTYPES)-by-1 cell array of strings restricting instruments worked on to match one of `TypeList` types.

Argument value pairs can be entered in any order.

Description   instsetfield sets data for existing instruments in a collection variable.

InstSet = instsetfield(InstSet, 'FieldName', FieldList, 'Data', DataList) resets or adds fields to every instrument.

InstSet = instsetfield(InstSet, 'FieldName', FieldList, 'Data', DataList, 'Index', IndexSet, 'Type', TypeList) resets or adds fields to a subset of instruments.

The output InstSet is a new instrument set variable containing the input data.

Examples   Retrieve the instrument set ExampleInstSF from the data file InstSetExamples.mat. ExampleInstSF contains three types of instruments: Option, Futures, and TBill.

```
load InstSetExamples;
ISet = ExampleInstSF;
instdisp(ISet)

Index Type    Strike Price Opt
1     Option  95     12.2  Call
2     Option  100     9.2  Call
3     Option  105     6.8  Call

Index Type     Delivery       F
4     Futures  01-Jul-1999   104.4

Index Type    Strike Price Opt
5     Option  105     7.4  Put
6     Option  NaN     NaN  Put

Index Type  Price
7     TBill 99
```

Enter data for the option in Index 6: Price 2.9 for a Strike of 95.

```
ISet = instsetfield(ISet, 'Index',6,...
'FieldName',{'Strike','Price'}, 'Data',{ 95 , 2.9 });
```

instsetfield

```
instdisp(ISet)

Index Type    Strike Price Opt
1     Option  95    12.2  Call
2     Option  100    9.2  Call
3     Option  105    6.8  Call
Index Type    Delivery       F
4     Futures 01-Jul-1999   104.4
Index Type    Strike Price Opt
5     Option  105    7.4  Put
6     Option  95     2.9  Put

Index Type  Price
7     TBill 99
```

Create a new field `Maturity` for the cash instrument.

```
MDate = datenum('7/1/99');
ISet = instsetfield(ISet, 'Type', 'TBill', 'FieldName',...
'Maturity','FieldClass', 'date', 'Data', MDate);
instdisp(ISet)

Index Type  Price  Maturity
7     TBill 99     01-Jul-1999
```

Create a new field `Contracts` for all instruments.

```
ISet = instsetfield(ISet, 'FieldName', 'Contracts', 'Data', 0);
instdisp(ISet)

Index Type    Strike Price Opt  Contracts
1     Option  95    12.2  Call 0
2     Option  100    9.2  Call 0
3     Option  105    6.8  Call 0

Index Type    Delivery       F     Contracts
4     Futures 01-Jul-1999   104.4 0
```

```
Index Type    Strike Price Opt  Contracts
5     Option 105     7.4 Put  0
6     Option  95     2.9 Put  0

Index Type  Price  Maturity     Contracts
7     TBill 99     01-Jul-1999  0
```

Set the Contracts fields for some instruments.

```
ISet = instsetfield(ISet,'Index',[3; 5; 4; 7],...
'FieldName','Contracts',  'Data', [1000; -1000; -1000; 6]);


instdisp(ISet)

Index Type    Strike Price Opt  Contracts
1     Option  95    12.2 Call    0
2     Option 100     9.2 Call    0
3     Option 105     6.8 Call  1000

Index Type    Delivery       F     Contracts
4     Futures 01-Jul-1999    104.4 -1000

Index Type    Strike Price Opt  Contracts
5     Option 105     7.4 Put  -1000
6     Option  95     2.9 Put      0

Index Type  Price  Maturity     Contracts
7     TBill 99     01-Jul-1999  6
```

See Also   instaddfield | instdisp | instget | instgetcell

instswap

Purpose	Construct swap instrument
Syntax	InstSet = instswap(InstSet, LegRate, Settle, Maturity) InstSet = instswap(InstSet, LegRate, Settle, Maturity, InstSet, LegReset, Basis, Principal, LegType, EndMonthRule) InstSet = instswap(InstSet, LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType, EndMonthRule, StartDate) [FieldList, ClassList, TypeString] = instswap

Arguments

InstSet	Instrument variable. This argument is specified only when adding a swap to an existing instrument set. See instget for more information on the InstSet variable.
LegRate	Number of instruments (NINST)-by-2 matrix, with each row defined as: [CouponRate Spread] or [Spread CouponRate] CouponRate is the decimal annual rate. Spread is the number of basis points over the reference rate. The first column represents the receiving leg, while the second column represents the paying leg.
Settle	Settlement date. NINST-by-1 vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. NINST-by-1 vector of dates representing the maturity date for each swap.
LegReset	(Optional) NINST-by-2 matrix representing the reset frequency per year for each swap. Default = [1 1].

Basis   (Optional) Day-count basis of the instrument. A
               vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Principal   (Optional) NINST-by-1 vector or NINST-by-1 cell
               array of the notional principal amounts or principal
               value schedules. For the latter case, each element
               of the cell array is a NumDates-by-2 matrix where
               the first column is dates and the second column is
               its associated notional principal value. The date
               indicates the last day that the principal value is
               valid.. Default = 100.

11-565

instswap

LegType	(Optional) NINST-by-2 matrix. Each row represents an instrument. Each column indicates if the corresponding leg is fixed (1) or floating (0). This matrix defines the interpretation of the values entered in LegRate. Default is [1,0] for each instrument.
EndMonthRule	(Optional) NINST-by-1 vector representing the End-of-month rule. Default = 1.
StartDate	(Optional) NINST-by-1 vector of dates when the swaps actually start. Default is Settle.

Data arguments are number of instruments (NINST)-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument; the others may be omitted or passed as empty matrices [].

Description   InstSet = instswap(InstSet, LegRate, Settle, Maturity) creates a new instrument set containing swap instruments or adds swap instruments to an existing instrument set.

InstSet = instswap(InstSet, LegRate, Settle, Maturity, InstSet, LegReset, Basis, Principal, LegType, EndMonthRule) uses optional input arguments to create a new instrument set containing swap instruments or adds swap instruments to an existing instrument set.

InstSet = instswap(InstSet, LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType, EndMonthRule, StartDate) to create a new forward swap instrument or to add a forward swap instrument to an existing portfolio.

[FieldList, ClassList, TypeString] = instswap displays the classes.

FieldList is a number of fields (NFIELDS)-by-1 cell array of strings listing the name of each data field for this instrument type.

ClassList is an NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments are parsed. Valid strings are 'dble', 'date', and 'char'.

TypeString is a string specifying the type of instrument added. For a swap instrument, TypeString = 'Swap'.

Definitions   **Amortizing Swap**

In an amortizing swap, the notional principal decreases periodically because it is tied to an underlying financial instrument with a declining (amortizing) principal balance, such as a mortgage.

Forward Swap

Agreement to enter into an interest-rate swap arrangement on a fixed date in future.

Examples   **Create a Vanilla Swap Instrument**

Create a vanilla swap using market data.

Use the following market data to create a swap instrument.

```
LegRate = [0.065, 0]
Settle = 'jan-1-2007';
Maturity = 'jan-1-2012';
LegReset = [1, 1];
Basis = 0
Principal = 100
LegType = [1, 0]

InstSet = instswap(LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType)

LegRate =

    0.0650         0


Basis =
```

```
         0


Principal =

   100


LegType =

    1     0


InstSet =

        FinObj: 'Instruments'
     IndexTable: [1x1 struct]
          Type: {'Swap'}
     FieldName: {{9x1 cell}}
    FieldClass: {{9x1 cell}}
     FieldData: {{9x1 cell}}
```

View the swap instrument using instdisp.

```
instdisp(InstSet)
```

```
Index Type LegRate    Settle       Maturity  LegReset Basis Principal LegType EndMonthRule StartDate
1     Swap [0.065  0] 01-Jan-2007 01-Jan-2012 [1  1]    0     100       [1  0]  1            NaN
```

See Also   hjmprice | instaddfield | instbond | instcap | instdisp |
instfloor | intenvprice

Purpose   Construct swaption instrument

Syntax   InstSet = instswaption(OptSpec, Strike, ExerciseDates,
             Spread, Settle, Maturity)
             InstSet = instswaption(OptSpec, Strike, ExerciseDates,
             Spread, Settle, Maturity, AmericanOpt,
             SwapReset, Basis, Principal)
             InstSet = instswaption(InstSetOld, OptSpec, Strike,
             ExerciseDates, Spread, ...)
             [FieldList, ClassList, TypeString] = instswaption;

Arguments   Fill unspecified entries in vectors with the value NaN. Only one data
             argument is required to create the instruments; the others may be
             omitted or passed as empty matrices [ ]. Type [FieldList, ClassList]
             = instswaption to see the classes. Dates can be input as serial date
             numbers or date strings.

OptSpec | NINST-by-1 cell array of strings 'call' or 'put'. A 'call' swaption entitles the buyer to pay the fixed rate. A 'put' swaption entitles the buyer to receive the fixed rate.

Strike | NINST-by-1 vector of strike swap rate values.

For a European option:

ExerciseDates | NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one ExerciseDate on the option expiry date.

AmericanOpt | NINST-by-1 vector of flags. AmericanOpt is 0 for each European option. The default is 0 if AmericanOpt is NaN or not entered.

For an American option:

ExerciseDates	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying swap Settle and the single listed ExerciseDate.
AmericanOpt	NINST-by-1 vector of flags. AmericanOpt is 1 for each American option. The AmericanOpt argument is required to invoke American exercise rules.

For an American or a European option:

Spread	NINST-by-1 vector representing the number of basis points over the reference rate.
Settle	NINST-by-1 vector of dates representing the settle date for each swap.
Maturity	NINST-by-1 vector of dates representing the maturity date for each swap.
SwapReset	(Optional) NINST-by-1 vector representing the reset frequency per year for the underlying swap. Default is 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page
  Glossary-1.

Principal   (Optional) NINST-by-1 vector of the notional
principal amounts. Default is 100.

Description   To specify a European option: InstSet = instswaption(OptSpec,
Strike, ExerciseDates,
Spread, Settle, Maturity)

To specify an American option: InstSet = instswaption(OptSpec,
Strike, ExerciseDates,
Spread, Settle, Maturity, AmericanOpt,
SwapReset, Basis, Principal)

To add swaption instruments to an instrument variable: InstSet =
instswaption(InstSetOld, OptSpec, Strike,
ExerciseDates, Spread, ...)

To list field metadata for the swaption instrument: [FieldList,
ClassList, TypeString] = instswaption;

Outputs:

instswaption

InstSet	Variable containing a collection of instruments. Instruments are broken down by type and each type can have different data fields. Each stored data field has a row vector or string for each instrument. For more information on the ISet variable, see `instget`.
FieldList	NFIELDS-by-1 cell array of strings listing the name of each data field for this instrument type.
ClassList	NFIELDS-by-1 cell array of strings listing the data class of each field. The class determines how arguments will be parsed. Valid strings are `'dble'`, `'date'`, and `'char'`.
TypeString	String specifying the type of instrument added. TypeString = `'Swaption'`.

Examples   Create two European swaption instruments with the following data:

```
OptSpec = {'Call'; 'Put'}
Strike = .05;
ExerciseDates = 'jan-1-2011';
Spread=0;
Settle = 'jan-1-2007';
Maturity = 'jan-1-2012';
AmericanOpt = 0;

InstSet = instswaption(OptSpec, Strike, ExerciseDates, Spread, Settle, Maturity, ...
 AmericanOpt);
```

View the two European swaption instruments by using `instdisp`:

```
instdisp(InstSet)
```

Indx	Type	OptSpec	Stke	ExerDates	Spread	Settle	Maturity	AmerOpt	SwpReset	Basis	Prinpal
1	Swaption	Call	0.05	01-Jan-2011	0	01-Jan-2007	01-Jan-2012	0	1	0	100
2	Swaption	Put	0.05	01-Jan-2011	0	01-Jan-2007	01-Jan-2012	0	1	0	100

See Also   instadd | instget | instdisp

insttypes

Purpose	List types

Syntax   TypeList = insttypes(InstSet)

Arguments

InstSet	Variable containing a collection of instruments. Instruments are classified by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.

Description   TypeList = insttypes(InstSet) retrieves a list of types stored in an instrument variable.

TypeList is a number of types (NTYPES)-by-1 cell array of strings listing the Type of instruments contained in the variable.

Examples   Retrieve the instrument set variable ExampleInst from the data file InstSetExamples.mat. ExampleInst contains three types of instruments: Option, Futures, and TBill.

```
load InstSetExamples;
instdisp(ExampleInst)

Index Type    Strike Price Opt  Contracts
1     Option  95    12.2   Call     0
2     Option  100    9.2   Call     0
3     Option  105    6.8   Call  1000

Index Type     Delivery        F      Contracts
4     Futures  01-Jul-1999    104.4  -1000

Index Type    Strike Price Opt  Contracts
5     Option  105    7.4   Put   -1000
6     Option  95     2.9   Put       0
```

```
Index Type Price Maturity      Contracts
7     TBill 99   01-Jul-1999   6
```

List all of the types included in ExampleInst.

```
TypeList = insttypes(ExampleInst)

TypeList =
        'Futures'
        'Option'
        'TBill'
```

See Also   instdisp | instfields | instlength

intenvget

Purpose	Properties of interest-rate structure
Syntax	ParameterValue = intenvget(RateSpec,'ParameterName')

Arguments

RateSpec	A structure containing the properties of an interest-rate structure. See intenvset for information on creating RateSpec.
ParameterName	String indicating the parameter name to be accessed. The value of the named parameter is extracted from the structure RateSpec. It is sufficient to type only the leading characters that uniquely identify the parameter. Case is ignored for parameter names.

Description ParameterValue = intenvget(RateSpec,'ParameterName') obtains the value of the named parameter ParameterName extracted from RateSpec.

Examples Use intenvset to set the interest-rate structure.

```
RateSpec = intenvset('Rates', 0.05, 'StartDates',...
'20-Jan-2000', 'EndDates', '20-Jan-2001')
```

Now use intenvget to extract the values from RateSpec.

```
[R, RateSpec] = intenvget(RateSpec, 'Rates')

R =

    0.0500
RateSpec =

  FinObj: 'RateSpec'
```

```
            Compounding: 2
            Disc: 0.9518
            Rates: 0.0500
            EndTimes: 2
            StartTimes: 0
            EndDates: 730871
            StartDates: 730505
            ValuationDate: 730505
            Basis: 0
            EndMonthRule: 1
```

See Also   intenvset

intenvprice

Purpose	Price instruments from set of zero curves
Syntax	`Price = intenvprice(RateSpec, InstSet)`

Arguments

RateSpec	A structure containing the properties of an interest-rate structure. See `intenvset` for information on creating `RateSpec`.
InstSet	Variable containing a collection of instruments. Instruments are categorized by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.

Description

`Price = intenvprice(RateSpec, InstSet)` computes arbitrage-free prices for instruments against a set of zero coupon bond rate curves.

`Price` is a number of instruments (NINST) by number of curves (NUMCURVES) matrix of prices of each instrument. If an instrument cannot be priced, a `NaN` is returned in that entry.

`intenvprice` handles the following instrument types: `'Bond'`, `'CashFlow'`, `'Fixed'`, `'Float'`, `'Swap'`. See `instadd` for information about constructing defined types.

See single-type pricing functions to retrieve pricing information.

bondbyzero	Price bonds from a set of zero curves.
cfbyzero	Price arbitrary cash flow instrument from a set of zero curves.
fixedbyzero	Fixed-rate note prices from a set of zero curves.
floatbyzero	Floating-rate note prices from a set of zero curves.
swapbyzero	Swap prices from a set of zero curves.

Examples   **Load Zero Curves and Instruments from Data File**

Load the zero curves and instruments.

```
load deriv.mat
instdisp(ZeroInstSet)
```

```
Index Type CouponRate Settle        Maturity     Period Basis EndMonthRule IssueDate FirstCouponD
1     Bond 0.04       01-Jan-2000   01-Jan-2003  1      NaN   NaN          NaN       NaN
2     Bond 0.04       01-Jan-2000   01-Jan-2004  2      NaN   NaN          NaN       NaN

Index Type  CouponRate Settle       Maturity     FixedReset Basis Principal Name     Quantity
3     Fixed 0.04       01-Jan-2000  01-Jan-2003  1          NaN   NaN       4% Fixed 80

Index Type  Spread Settle       Maturity     FloatReset Basis Principal Name       Quantity
4     Float 20     01-Jan-2000  01-Jan-2003  1          NaN   NaN       20BP Float 8

Index Type LegRate     Settle       Maturity     LegReset Basis Principal LegType Name         Qu
5     Swap [0.06  20] 01-Jan-2000  01-Jan-2003  [1  1]   NaN   NaN       [NaN]   6%/20BP Swap 10
```

Price the instruments.

```
Price = intenvprice(ZeroRateSpec, ZeroInstSet)

Price =

  98.7159
  97.5334
  98.7159
 100.5529
   3.6923
```

Price a Multi-Stepped Coupon Bond

Price the following multi-stepped coupon bonds using market data.

The data for the interest rate term structure is as follows:

```
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;
```

Create `RateSpec`.

```
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);
```

Create a portfolio of stepped coupon bonds with different maturities.

```
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2011';'01-Jan-2012';'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2011' .042;'01-Jan-2012' .05; '01-Jan-2013' .06; '01-Jan-2014' .07}};
ISet = instbond(CouponRate, Settle, Maturity, 1);
instdisp(ISet)

Index Type CouponRate Settle      Maturity    Period Basis EndMonthRule ... Face
1  Bond [Cell]   01-Jan-2010   01-Jan-2011   1      0     1            ... 100
2  Bond [Cell]   01-Jan-2010   01-Jan-2012   1      0     1            ... 100
3  Bond [Cell]   01-Jan-2010   01-Jan-2013   1      0     1            ... 100
4  Bond [Cell]   01-Jan-2010   01-Jan-2014   1      0     1            ... 100
```

Table of instrument portfolio is partially displayed.

Compute the price of the stepped coupon bonds.

```
PZero = intenvprice(RS, ISet)

PZero =

  100.6763
  100.7368
  100.9266
  101.0115
```

See Also   hjmprice | hjmsens | instadd | intenvsens | intenvset

intenvsens

Purpose	Instrument price and sensitivities from set of zero curves
Syntax	[Delta, Gamma, Price] = intenvsens(RateSpec, InstSet)

Arguments

RateSpec	A structure containing the properties of an interest-rate structure. See intenvset for information on creating RateSpec.
InstSet	Variable containing a collection of instruments. Instruments are categorized by type; each type can have different data fields. The stored data field is a row vector or string for each instrument.

Description [Delta, Gamma, Price] = intenvsens(RateSpec, InstSet) computes dollar prices and price sensitivities for instruments that use a zero coupon bond rate structure.

Delta is a number of instruments (NINST) by number of curves (NUMCURVES) matrix of deltas, representing the rate of change of instrument prices with respect to shifts in the observed forward yield curve. Delta is computed by finite differences.

Gamma is an NINST-by-NUMCURVES matrix of gammas, representing the rate of change of instrument deltas with respect to shifts in the observed forward yield curve. Gamma is computed by finite differences.

Note Both sensitivities are returned as dollar sensitivities. To find the per-dollar sensitivities, divide by the respective instrument price.

Price is an NINST-by-NUMCURVES matrix of prices of each instrument. If an instrument cannot be priced, a NaN is returned.

intenvsens handles the following instrument types: `'Bond'`,
`'CashFlow'`, `'Fixed'`, `'Float'`, `'Swap'`. See instadd for information
about constructing defined types.

Examples   Load the tree and instruments from a data file.

```
load deriv.mat
instdisp(ZeroInstSet)
```

```
Index Type CouponRate Settle       Maturity    Period Basis EndMonthRule IssueDate FirstCouponD
1     Bond 0.04       01-Jan-2000  01-Jan-2003 1      NaN   NaN          NaN       NaN
2     Bond 0.04       01-Jan-2000  01-Jan-2004 2      NaN   NaN          NaN       NaN

Index Type  CouponRate Settle       Maturity     FixedReset Basis Principal Name     Quantity
3     Fixed 0.04       01-Jan-2000  01-Jan-2003  1          NaN   NaN       4% Fixed 80

Index Type  Spread Settle       Maturity     FloatReset Basis Principal Name       Quantity
4     Float 20     01-Jan-2000  01-Jan-2003  1          NaN   NaN       20BP Float 8

Index Type LegRate      Settle       Maturity     LegReset Basis Principal LegType Name       Qu
5     Swap [0.06 20] 01-Jan-2000  01-Jan-2003  [1  1]   NaN   NaN       [NaN]   6%/20BP Swap 10
```

```
[Delta, Gamma] = intenvsens(ZeroRateSpec, ZeroInstSet)

Delta =

 -272.6403
 -347.4386
 -272.6403
   -1.0445
 -282.0405

Gamma =

  1.0e+003 *
```

```
                        1.0298
                        1.6227
                        1.0298
                        0.0033
                        1.0596
```

See Also   hjmprice | hjmsens | instadd | intenvprice | intenvset

Purpose	Set properties of interest-rate structure
Syntax	[RateSpec, RateSpecOld] = intenvset(RateSpec, 'Argument1', Value1, 'Argument2', Value2, ...) [RateSpec, RateSpecOld] = intenvset intenvset

Arguments

RateSpec   (Optional) An existing interest-rate specification structure to be changed, probably created from a previous call to intenvset.

Arguments may be chosen from the following table and specified in any order.

Compounding   Scalar value representing the rate at which the input zero rates or forward rates were compounded when annualized. Default = 2. This argument determines the formula for the discount factors:

Compounding = 1, 2, 3, 4, 6, 12

Disc = (1 + Z/F)^(-T), where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.

Compounding = 365

Disc = (1 + Z/F)^(-T), where F is the number of days in the basis year and T is a number of days elapsed computed by basis.

Compounding = -1

	`Disc = exp(-T*Z)`, where `T` is time in years.
Disc	Number of points (NPOINTS) by number of curves (NCURVES) matrix of unit bond prices over investment intervals from StartDates, when the cash flow is valued, to EndDates, when the cash flow is received.
Rates	Number of points (NPOINTS) by number of curves (NCURVES) matrix of rates in decimal form. For example, 5% is 0.05 in Rates. Rates are the yields over investment intervals from StartDates, when the cash flow is valued, to EndDates, when the cash flow is received.
EndDates	NPOINTS-by-1 vector or scalar of serial maturity dates ending the interval to discount over.
StartDates	NPOINTS-by-1 vector or scalar of serial dates starting the interval to discount over. Default = ValuationDate. StartDates must be earlier than EndDates.
ValuationDate	(Optional) Scalar value in serial date number form representing the observation date of the investment horizons entered in StartDates and EndDates. Default = min(StartDates).
Basis	(Optional) Day-count basis of the instrument. A scalar of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

EndMonthRule   (Optional) End-of-month rule. A scalar. This rule applies only when EndDates is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

It is sufficient to type only the leading characters that uniquely identify the parameter. Case is ignored for argument names.

When creating a new RateSpec, the set of arguments passed to intenvset must include StartDates, EndDates, and either Rates or Disc.

Call intenvset with no input or output arguments to display a list of argument names and possible values.

Description   [RateSpec, RateSpecOld] = intenvset(RateSpec, 'Argument1', Value1, 'Argument2', Value2, ...) creates an interest term

intenvset

structure (`RateSpec`) in which the input argument list is specified as argument name /argument value pairs. The argument name portion of the pair must be recognized as a valid field of the output structure `RateSpec`; the argument value portion of the pair is then assigned to its paired field.

If the optional argument `RateSpec` is specified, `intenvset` modifies an existing interest term structure `RateSpec` by changing the named argument to the specified values and recalculating the arguments dependent on the new values.

`[RateSpec, RateSpecOld] = intenvset` creates an interest term structure `RateSpec` with all fields set to `[]`.

`intenvset` with no input or output arguments displays a list of argument names and possible values.

`RateSpecOld` is a structure containing the properties of an interest-rate structure before the changes introduced by the call to `intenvset`.

Examples

Create a RateSpec for a Zero Curve

Use `intenvset` to create a `RateSpec` for a zero curve.

Use `intenvset` to create a `RateSpec` for a zero curve.

```
RateSpec = intenvset('Rates', 0.05, 'StartDates',...
'20-Jan-2000', 'EndDates', '20-Jan-2001')

RateSpec =

           FinObj: 'RateSpec'
      Compounding: 2
             Disc: 0.9518
            Rates: 0.0500
         EndTimes: 2
       StartTimes: 0
         EndDates: 730871
       StartDates: 730505
    ValuationDate: 730505
```

```
              Basis: 0
        EndMonthRule: 1
```

Now change the Compounding argument to 1 (annual).

```
RateSpec = intenvset(RateSpec, 'Compounding', 1)

RateSpec =

              FinObj: 'RateSpec'
         Compounding: 1
                Disc: 0.9518
               Rates: 0.0506
            EndTimes: 1
          StartTimes: 0
            EndDates: 730871
          StartDates: 730505
       ValuationDate: 730505
               Basis: 0
         EndMonthRule: 1
```

Calling intenvset with no input or output arguments displays a list of argument names and possible values.

```
intenvset
```

```
 Compounding: [ 1 | {2} | 3 | 4 | 6 | 12 | 365 | -1 ]
                Disc: [ scalar | vector (NPOINTS x 1) ]
               Rates: [ scalar | vector (NPOINTS x 1) ]
            EndDates: [ scalar | vector (NPOINTS x 1) ]
          StartDates: [ scalar | vector (NPOINTS x 1) ]
       ValuationDate: [ scalar ]
               Basis: [ {0} | 1 | 2 | 3 ]
        EndMonthRule: [ 0 | {1} ]
```

Create a RateSpec for a Forward Curve

Use intenvset to create a RateSpec for a forward curve.

Use `intenvset` to create a `RateSpec` for a forward curve.

```
RateSpec = intenvset('Rates', 0.05, 'StartDates',...
'20-Jan-2001', 'EndDates', '20-Jan-2002', 'ValuationDate','20-Jan-2000')
```

```
RateSpec =

          FinObj: 'RateSpec'
     Compounding: 2
            Disc: 0.9518
           Rates: 0.0500
        EndTimes: 4
      StartTimes: 2
        EndDates: 731236
      StartDates: 730871
   ValuationDate: 730505
           Basis: 0
     EndMonthRule: 1
```

Now change the `Compounding` argument to 1 (annual).

```
RateSpec = intenvset(RateSpec, 'Compounding', 1)
```

```
RateSpec =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: 0.9518
           Rates: 0.0506
        EndTimes: 2
      StartTimes: 1
        EndDates: 731236
      StartDates: 730871
   ValuationDate: 730505
           Basis: 0
     EndMonthRule: 1
```

Create RateSpec Using Two Curves

Create a `RateSpec` for two interest-rate curves.

Define data for the interest-rate term structure and use `intenvset` to create a `RateSpec`.

```
StartDates = '01-Oct-2011';
EndDates = ['01-Oct-2012'; '01-Oct-2013';'01-Oct-2014';'01-Oct-2015'];
Rates = [[0.0356;0.041185;0.04489;0.047741],[0.0325;0.0423;0.0437;0.0465]];
RateSpec = intenvset('Rates', Rates, 'StartDates',StartDates,...
'EndDates', EndDates, 'Compounding', 1)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: [4x2 double]
             Rates: [4x2 double]
          EndTimes: [4x1 double]
        StartTimes: [4x1 double]
          EndDates: [4x1 double]
        StartDates: 734777
     ValuationDate: 734777
             Basis: 0
       EndMonthRule: 1
```

To look at the `Rates` for the two interest-rate curves:

```
RateSpec.Rates

ans =

    0.0356    0.0325
    0.0412    0.0423
    0.0449    0.0437
    0.0477    0.0465
```

See Also   intenvget | intenvprice | intenvset

isafin

Purpose	True if input argument is financial structure type or financial object class
Syntax	IsFinObj = isafin(Obj, ClassName)

Arguments

Obj	Name of a financial structure.
ClassName	String containing the name of a financial structure class.

Description   IsFinObj = isafin(Obj, ClassName) returns True (1) if input argument is a financial structure type or financial object class, otherwise False (0) is returned.

Examples
```
load deriv.mat
IsFinObj = isafin(HJMTree, 'HJMFwdTree')

IsFinObj =

     1
```

See Also   classfin

Purpose   Price instruments using implied trinomial tree (ITT)

Syntax   Price = ittprice(ITTTree, InstSet)
                 Price = ittprice(ITTTree, InstSet, Options)
                 [Price, PriceTree] = ittprice(ITTTree, InstSet, Options)

Arguments

ITTTree	Implied trinomial stock tree. See itttree for information on creating the variable ITTTree.
InstSet	Variable containing a collection of NINST instruments. Instruments are broken down by type and each type can have different data fields.
Options	(Optional) Structure created using derivset containing derivative pricing options.

Description   Price = ittprice(ITTTree, InstSet)

Price = ittprice(ITTTree, InstSet, Options)

[Price, PriceTree] = ittprice(ITTTree, InstSet, Options)

The outputs for ittprice are:

- Price is a NINST-by-1 vector of prices of each instrument at time 0. The prices are computed by backward dynamic programming on the stock tree. If an instrument cannot be priced, a NaN is returned in that entry.

- PriceTree is a structure containing trees of vectors of instrument prices and a vector of observation times for each node.

  - PriceTree.PTree contains the prices.

  - PriceTree.tObs contains the observation times.

  - PriceTree.dObs contains the observation dates.

ittprice

ittprice computes prices for instruments using an implied trinomial tree created with `itttree`.

Note `ittprice` handles the following instrument types: optstock, barrier, Asian, lookback, and compound. Use `instadd` to construct the defined types.

When using an implied trinomial tree, pricing of path-dependent options is done using Hull-White. Consequently, for these options there are no unique prices on the tree nodes with the exception of the root node. The corresponding nodes of the tree are populated with NaNs for these particular options. For information on single-type pricing functions to retrieve state-by-state pricing tree information, see the following:

- `barrierbyitt` for pricing barrier options using an ITT tree
- `optstockbyitt` for pricing American, European or Bermuda options using an ITT tree
- `asianbyitt` for pricing Asian options using an ITT tree
- `lookbackbyitt` for pricing lookback options using an ITT tree
- `compoundbyitt` for price compound options using an ITT tree

Examples
Load the ITT tree and instruments from the data file `deriv.mat`.

```
load deriv.mat
```

Price the barrier and Asian options contained in the instrument set.

```
ITTSubSet = instselect(ITTInstSet,'Type', {'Barrier', 'Asian'});
```

```
instdisp(ITTSubSet)
```

```
Index Type OptSpec Strike Settle ExerDates AmerOpt BarrSpec Barr Rebate Name Quantity
1  Barrier call   85  01-Jan-2006 31-Dec-2008    1       ui     115    0  Barrier1 1
```

```
IndxType OptSpec Strike Settle ExerDates AmerOpt AvgType AvgPrice AvgDate Name Quantity
2  Asian call   55  01-Jan-2006  01-Jan-2008  0   arithmetic NaN   NaN  Asian1 5
3  Asian call   55  01-Jan-2006  01-Jan-2010  0   arithmetic NaN   NaN  Asian2 7
```

```
[Price, PriceTree] = ittprice(ITTTree, ITTSubSet)

Price =
    2.4074
    3.2052
    6.6074
PriceTree =
    FinObj: 'TrinPriceTree'
     PTree: {[3x1 double]  [3x3 double]  [3x5 double]  [3x7 double]  |
      tObs: [0 1 2 3 4]
      dObs: [732678 733043 733408 733773 734139]
```

See Also ittsens | itttree

ittsens

Purpose   Instrument sensitivities and prices using implied trinomial tree (ITT)

Syntax
```
[Delta, Gamma, Vega] = ittsens(ITTTree, InstSet)
[Delta, Gamma, Vega, Price] = ittsens(ITTTree, InstSet)
[Delta, Gamma, Vega, Price] = ittsens(ITTTree, InstSet,
Options)
```

Arguments

ITTTree	Implied trinomial stock tree. See itttree for information on creating the variable ITTTree.
InstSet	Variable containing a collection of NINST instruments. Instruments are broken down by type and each type can have different data fields.
Options	(Optional) Structure created using derivset containing derivative pricing options.

Description   [Delta, Gamma, Vega] = ittsens(ITTTree, InstSet)

[Delta, Gamma, Vega, Price] = ittsens(ITTTree, InstSet)

[Delta, Gamma, Vega, Price] = ittsens(ITTTree, InstSet,
Options)

The outputs for ittsens are:

- Delta is a NINST-by-1 vector of deltas, representing the rate of change of instruments prices with respect to changes in the stock price.

- Gamma is a NINST-by-1 vector of gammas, representing the rate of change of instruments deltas with respect to changes in the stock price.

- Vega is a NINST-by-1 vector of vegas, representing the rate of change of instruments prices with respect to changes in the volatility of the stock. Vega is computed by finite differences in calls to itttree.

- `Price` is a `NINST`-by-1 vector of prices of each instrument. The prices are computed by backward dynamic programming on the stock tree. If an instrument cannot be priced, a NaN is returned.

`ittsens` computes dollar sensitivities and prices for instruments using an ITT tree created with `itttree`.

Note `ittsens` handles the following instrument types: optstock, barrier, Asian, lookback, and compound. Use `instadd` to construct the defined types.

For path-dependent options (lookbacks and Asians), `Delta` and `Gamma` are computed by finite differences in calls to `ittprice`. For the rest of the options (optstock, barrier, and compound), `Delta` and `Gamma` are computed from the ITT tree and the corresponding option price tree.

All sensitivities are returned as dollar sensitivities. To find the per-dollar sensitivities, they must be divided by their respective instrument price.

Examples   Load the ITT tree and instruments from the data file `deriv.mat`. Compute the `Delta` and `Gamma` sensitivities of vanilla options and barrier option contained in the instrument set.

```
load deriv.mat
ITTSubSet = instselect(ITTInstSet,'Type', {'OptStock', 'Barrier'});

instdisp(ITTSubSet)
```

Index	Type	OptSpec	Strike	Settle	ExerciseDates	AmericanOpt	Name	Quantity
1	OptStock	call	95	01-Jan-2006	31-Dec-2008	1	Call1	10
2	OptStock	put	80	01-Jan-2006	01-Jan-2010	0	Put1	4

Index	Type	OptSpec	Strike	Settle	ExercDates	AmerOpt	BarrSpec	Barr	Rebate	Name	Quantity

```
3  Barrier call  85  01-Jan-2006 31-Dec-2008  1  ui  115  0  Barrier1 1
```

```
[Delta, Gamma] = ittsens(ITTTree, ITTSubSet)
```

```
Warning: The option set specified in StockOptSpec was too narrow for the generated tree.
This made extrapolation necessary. Below is a list of the options that were outside of
the range of those specified in StockOptSpec.
```

```
Option Type: 'call'  Maturity: 01-Jan-2007  Strike=67.2897
Option Type: 'put'   Maturity: 01-Jan-2007  Strike=37.1528
Option Type: 'put'   Maturity: 01-Jan-2008  Strike=27.6066
Option Type: 'put'   Maturity: 31-Dec-2008  Strike=20.5132
Option Type: 'call'  Maturity: 01-Jan-2010  Strike=164.0157
Option Type: 'put'   Maturity: 01-Jan-2010  Strike=15.2424
```

```
> In itttree>InterpOptPrices at 675
  In itttree at 277
  In stocktreesens>stocktreevega at 191
  In stocktreesens at 92
  In ittsens at 81
```

```
Delta =

   0.2387
  -0.4283
   0.3482


Gamma =

  0.0260
  0.0188
  0.0380
```

References   Chriss, Neil. and I. Kawaller, *Black-Scholes and Beyond: Options Pricing Models*, McGraw-Hill, 1996, pp. 308-312.

See Also   ittprice | itttree

Purpose	Specify time structure using implied trinomial tree (ITT)
Syntax	TimeSpec = itttimespec(ValuationDate, Maturity, NumPeriods)

Arguments

ValuationDate	Scalar date marking the pricing date and first observation in the tree. Specify ValuationDate as a serial date number or date string.
Maturity	Scalar date marking the depth of the tree.
NumPeriods	Scalar that determines how many time steps are in the tree.

Description

TimeSpec = itttimespec(ValuationDate, Maturity, NumPeriods)
creates the structure specifying the time layout for an ITT tree.

Examples

Specify a four-period tree with time steps of 1 year.

```
ValuationDate = '1-July-2006';
Maturity = '1-July-2010';
TimeSpec = itttimespec(ValuationDate, Maturity, 4)


TimeSpec =

             FinObj: 'ITTTimeSpec'
      ValuationDate: 732859
           Maturity: 734320
         NumPeriods: 4
              Basis: 0
        EndMonthRule: 1
               tObs: [0 1 2 3 4]
               dObs: [732859 733224 733589 733954 734320]
```

See Also

itttree | stockspec

ittree

Purpose	Build implied trinomial stock tree
Syntax	itttree(StockSpec, RateSpec, TimeSpec, StockOptSpec)

Arguments

StockSpec	Stock specification. For more information, see stockspec.
RateSpec	Interest rate specification of the initial risk-free rate curve. For more information on declaring an interest rate variable, see intenvset.
TimeSpec	Tree time layout specification. Defines the observation dates of the implied trinomial tree. For more information on the tree structure, see itttimespec.
StockOptSpec	Option stock specification. For more information, see stockoptspec.

Description itttree(StockSpec, RateSpec, TimeSpec, StockOptSpec) creates the itttree structure specifying stock and time information for an implied trinomial tree.

Examples For this example, assume that the interest rate is fixed at 8% annually between the valuation date of the tree (January 1, 2006) until its maturity.

```
Rate = 0.08;
ValuationDate = '01-01-2006';
EndDate = '01-01-2008';

RateSpec = intenvset('StartDates', ValuationDate, 'EndDates', EndDate, ...
    'ValuationDate', ValuationDate, 'Rates', Rate, 'Compounding', -1);
```

To build an ITTTree, create `StockSpec`, `TimeSpec`, and `StockOptSpec` structures.

To create the`StockSpec` structure:

```
Sigma = 0.20;
AssetPrice = 50;
DividendType = 'cash';
DividendAmounts = [0.50; 0.50; 0.50; 0.50];
ExDividendDates = {'03-Jan-2007'; '01-Apr-2007'; '05-July-2007';'01-Oct-2007'}

StockSpec = stockspec(Sigma, AssetPrice, DividendType, ...
DividendAmounts, ExDividendDates)

StockSpec =

                FinObj: 'StockSpec'
                 Sigma: 0.2000
            AssetPrice: 50
          DividendType: 'cash'
       DividendAmounts: [4x1 double]
       ExDividendDates: [4x1 double]
```

The syntax for building a `TimeSpec` structure is `TimeSpec = itttimespec(ValuationDate, Maturity, NumPeriods)`.

Consider building an ITT tree, with a valuation date of January 1, 2006; a maturity date of January 1, 2008; and four time steps.

```
ValuationDate = '01-01-2006';
EndDate = '01-01-2008';
NumPeriods = 4;

TimeSpec = itttimespec(ValuationDate, EndDate, NumPeriods)

TimeSpec =

             FinObj: 'ITTTimeSpec'
```

```
      ValuationDate: 732678
           Maturity: 733408
         NumPeriods: 4
              Basis: O
        EndMonthRule: 1
               tObs: [0 0.5000 1 1.5000 2]
               dObs: [732678 732860 733043 733225 733408]
```

The syntax for building a StockOptSpec structure is [StockOptSpec]
= stockoptspec(OptPrice, Strike, Settle, Maturity, OptSpec).

```
Settle =   '01/01/06';

Maturity =   ['07/01/06';
    '07/01/06';
    '07/01/06';
    '07/01/06';
    '01/01/07';
    '01/01/07';
    '01/01/07';
    '01/01/07';
    '07/01/07';
    '07/01/07';
    '07/01/07';
    '07/01/07';
    '01/01/08';
    '01/01/08';
    '01/01/08';
    '01/01/08'];

Strike = [113;
    101;
    100;
     88;
    128;
    112;
    100;
```

```
              78;
             144;
             112;
             100;
              69;
             162;
             112;
             100;
              61];

OptPrice =[                   0;
           4.807905472659144;
           1.306321897011867;
           0.048039195057173;
                            0;
           2.310953054191461;
           1.421950392866235;
           0.020414826276740;
                            0;
           5.091986935627730;
           1.346534812295291;
           0.005101325584140;
                            0;
           8.047628153217246;
           1.219653432150932;
           0.001041436654748];


OptSpec = { 'call';
            'call';
            'put';
            'put';
            'call';
            'call';
            'put';
            'put';
            'call';
```

```
       'call';
       'put';
       'put';
       'call';
       'call';
       'put';
       'put'};

StockOptSpec = stockoptspec(OptPrice, Strike, Settle, Maturity, OptSpec)

StockOptSpec =

         FinObj: 'StockOptSpec'
       OptPrice: [16x1 double]
         Strike: [16x1 double]
         Settle: 732678
       Maturity: [16x1 double]
        OptSpec: {16x1 cell}
     InterpMethod: 'price'
```

Note In this example, the extrapolation warnings are turned on to display warnings on the Command Window. These warnings are a consequence of having to extrapolate to find the option price of the tree nodes. In this example, the set of inputs options was too narrow for the shift in the tree nodes introduced by the disturbance used to calculate the sensitivities. As a consequence extrapolation for some of the nodes was needed.

Use the following command to turn on extrapolation warnings:

```
warning('on', 'fininst:itttree:Extrapolation');
```

Use the StockSpec, RateSpec, TimeSpec, and StockOptSpec structure to create an ITTTree.

```
ITTTree = itttree(StockSpec, RateSpec, TimeSpec, StockOptSpec)
```

```
Warning: The option set specified in StockOptSpec was too narrow for the generated tree.
This made extrapolation necessary. Below is a list of the options that were outside of
the range of those specified in StockOptSpec.

Option Type: 'call'   Maturity: 02-Jul-2006  Strike=60.7466
Option Type: 'put'    Maturity: 02-Jul-2006  Strike=50.0731
Option Type: 'put'    Maturity: 02-Jul-2006  Strike=41.3344
Option Type: 'call'   Maturity: 01-Jan-2007  Strike=73.8592
Option Type: 'call'   Maturity: 01-Jan-2007  Strike=60.8227
Option Type: 'put'    Maturity: 01-Jan-2007  Strike=50.1492
Option Type: 'put'    Maturity: 01-Jan-2007  Strike=41.4105
Option Type: 'put'    Maturity: 01-Jan-2007  Strike=34.2559
Option Type: 'call'   Maturity: 02-Jul-2007  Strike=88.8310
Option Type: 'call'   Maturity: 02-Jul-2007  Strike=72.9081
Option Type: 'call'   Maturity: 02-Jul-2007  Strike=59.8715
Option Type: 'put'    Maturity: 02-Jul-2007  Strike=49.1980
Option Type: 'put'    Maturity: 02-Jul-2007  Strike=40.4594
Option Type: 'put'    Maturity: 02-Jul-2007  Strike=33.3047
Option Type: 'put'    Maturity: 02-Jul-2007  Strike=27.4470
Option Type: 'call'   Maturity: 01-Jan-2008  Strike=107.2895
Option Type: 'call'   Maturity: 01-Jan-2008  Strike=87.8412
Option Type: 'call'   Maturity: 01-Jan-2008  Strike=71.9183
Option Type: 'call'   Maturity: 01-Jan-2008  Strike=58.8817
Option Type: 'put'    Maturity: 01-Jan-2008  Strike=48.2083
Option Type: 'put'    Maturity: 01-Jan-2008  Strike=39.4696
Option Type: 'put'    Maturity: 01-Jan-2008  Strike=32.3150
Option Type: 'put'    Maturity: 01-Jan-2008  Strike=26.4573
Option Type: 'put'    Maturity: 01-Jan-2008  Strike=21.6614

> In itttree>InterpOptPrices at 675
  In itttree at 277

ITTTree =

        FinObj: 'ITStockTree'
      StockSpec: [1x1 struct]
   StockOptSpec: [1x1 struct]
```

```
            TimeSpec: [1x1 struct]
            RateSpec: [1x1 struct]
                tObs: [0 0.500000000000000 1 1.500000000000000 2]
                dObs: [732678 732860 733043 733225 733408]
               STree: {1x5 cell}
               Probs: {[3x1 double]  [3x3 double]  [3x5 double]  [3x7 double]}
```

See Also   intenvset | itttimespec | stockoptspec | stockspec

Superclasses

Purpose   Create LIBOR Market Model

Description   The LIBOR Market Model (LMM) differs from short rate models in that
it evolves a set of discrete forward rates. Specifically, the lognormal
LMM specifies the following diffusion equation for each forward rate

$$\frac{dF_i(t)}{F_i} = -\mu_i dt + \sigma_i(t)dW_i$$

where:

W is an N-dimensional geometric Brownian motion with

$$dW_i(t)dW_j(t) = \rho_{ij}$$

The LMM relates the drifts of the forward rates based on no-arbitrage
arguments. Specifically, under the Spot LIBOR measure, the drifts
are expressed as

$$\mu_i(t) = -\sigma_i(t) \sum_{j=q(t)}^{i} \frac{\tau_j \rho_{i,j} \sigma_j(t) F_j(t)}{1 + \tau_j F_j(t)}$$

where:

τ_i is the time fraction associated with the *i* th forward rate

q(t) is an index defined by the relation

$$T_{q(t)-1} < t < T_{q(t)}$$

and the Spot LIBOR numeraire is defined as

$$B(t) = P(t, T_{q(t)}) \prod_{n=0}^{q(t)-1} (1 + \tau_n F_n(T_n))$$

LiborMarketModel

Construction
OBJ = LiborMarketModel(ZeroCurve,VolFunc,Correlation)
constructs a LIBOR Market Model object.

For example:

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

LMMVolFunc = @(a,t) (a(1)*t + a(2)).*exp(-a(3)*t) + a(4);
LMMVolParams = [.3 -.02 .7 .14];

numRates = 20;
VolFunc(1:numRates) = {@(t) LMMVolFunc(LMMVolParams,t)};

Beta = .08;
CorrFunc = @(i,j,Beta) exp(-Beta*abs(i-j));
Correlation = CorrFunc(meshgrid(1:numRates)',meshgrid(1:numRates),Beta);

LMM = LiborMarketModel(irdc,VolFunc,Correlation,'Tenor',1:numRates);
```

Properties
The following properties are from the LiborMarketModel class.

ZeroCurve

ZeroCurve is specified using the output from IRDataCurve or
RateSpec. This is the zero curve used to evolve the path of future
interest rates.

Attributes:

SetAccess	public
GetAccess	public

VolFunc

NumRates-by-1 cell array of function handles. Each function handle must take time as an input and, return a scalar volatility.

Attributes:

SetAccess	public
GetAccess	public

Correlation

NumRates-by-NumRates correlation matrix.

Attributes:

SetAccess	public
GetAccess	public

NumFactors

Number of Brownian factors. The default is NaN, where the number of factors is equal to the number of rates.

Attributes:

SetAccess	public
GetAccess	public

Period

Period of the forward rates. The default is default is 2, meaning forward rates are spaced at 0, .5, 1, 1.5 and so on.

Note Correlation and VolFunc are sized with NumRates-by-1 since the first rate is locked in and essentially dead.

Attributes:

LiborMarketModel

SetAccess	public
GetAccess	public

Methods

simTermStructs	Simulate term structures for LIBOR Market Model

Definitions   **LIBOR Market Model**

The LIBOR Market Model, also called the BGM Model (Brace, Gatarek, Musiela Model) is a financial model of interest rates. The quantities that are modeled are a set of forward rates (also called forward LIBORs) which have the advantage of being directly observable in the market, and whose volatilities are naturally linked to traded contracts.

Copy Semantics   Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB documentation.

Examples   **Construct a LIBOR Market Model**

Construct a LMM object.

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

LMMVolFunc = @(a,t) (a(1)*t + a(2)).*exp(-a(3)*t) + a(4);
LMMVolParams = [.3 -.02 .7 .14];

numRates = 20;
VolFunc(1:numRates-1) = {@(t) LMMVolFunc(LMMVolParams,t)};
```

```
Beta = .08;
CorrFunc = @(i,j,Beta) exp(-Beta*abs(i-j));
Correlation = CorrFunc(meshgrid(1:numRates-1)',meshgrid(1:numRates-1),Beta);


LMM = LiborMarketModel(irdc,VolFunc,Correlation,'Period',1)


LMM =

  LiborMarketModel with properties:

       ZeroCurve: [1x1 IRDataCurve]
     VolFunctions: {1x19 cell}
      Correlation: [19x19 double]
       NumFactors: NaN
           Period: 1
```

Simulate the term structures for the specified LMM object.

```
[ZeroRates, ForwardRates] = LMM.simTermStructs(10,'nTrials',100);
```

References   Brigo, D. and F. Mercurio, *Interest Rate Models - Theory and Practice*, Springer Finance, 2006.

See Also   HullWhite1F **|** LinearGaussian2F **|** simTermStructs **|**

Related Examples
• "Price Swaptions With Interest-Rate Models Using Simulation" on page 2-91
• Class Attributes
• Property Attributes

LiborMarketModel.simTermStructs

Purpose	Simulate term structures for LIBOR Market Model
Syntax	[ZeroRates,ForwardRates] = simTermStructs(nPeriods) [ZeroRates,ForwardRates] = simTermStructs(nPeriods, Name,Value)
Description	[ZeroRates,ForwardRates] = simTermStructs(nPeriods) simulates future zero curve paths using a specified LiborMarketModel object. [ZeroRates,ForwardRates] = simTermStructs(nPeriods, Name,Value) simulates future zero curve paths using a specified LiborMarketModel object with additional options specified by one or more Name,Value pair arguments.

Input Arguments

nPeriods

Number of simulation periods.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'deltaTime'

Scalar time step between periods.

Default: 1

'nTrials'

Scalar number of trials.

Default: 1

'antithetic'

Boolean scalar flag indicating whether antithetic sampling is used to generate the Gaussian random variates that drive the zero-drift, unit-variance rate Brownian vector $dW(t)$. For details, see simBySolution.

Default: false

'Z'

Direct specification of the dependent random noise process used to generate the zero-drift, unit-variance rate Brownian vector $dW(t)$ that drives the simulation. For details, see simBySolution for the GBM model.

Default: Uses default for simBySolution. If you do not specify a value for Z, simBySolution generates Gaussian variates.

'Period'

Period of the forward rates.

> **Note** Correlation and VolFunc are sized with NumRates-by-1 since the first rate is locked in and essentially dead.

Numeric vector of maturities to compute at each time step.

Default: 2 meaning forward rates are spaced at 0, .5, 1, 1.5 and so on.

Output Arguments

ZeroRates

nPeriods+1-by-nTenors-by-nTrials matrix of simulated zero-rate term structures.

ForwardRates

nPeriods+1-by-nTenors-by-nTrials matrix of simulated forward-rate term structures.

Attributes

Access	public
Static	false
Hidden	false

To learn about attributes of methods, see Method Attributes in the MATLAB Object-Oriented Programming documentation.

Examples

Simulate Term Structures for a LIBOR Market Model

Create a LMM object.

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

LMMVolFunc = @(a,t) (a(1)*t + a(2)).*exp(-a(3)*t) + a(4);
LMMVolParams = [.3 -.02 .7 .14];

numRates = 20;
VolFunc(1:numRates-1) = {@(t) LMMVolFunc(LMMVolParams,t)};

Beta = .08;
CorrFunc = @(i,j,Beta) exp(-Beta*abs(i-j));
Correlation = CorrFunc(meshgrid(1:numRates-1)',meshgrid(1:numRates-1),Bet

LMM = LiborMarketModel(irdc,VolFunc,Correlation,'Period',1)

LMM =
```

```
LiborMarketModel with properties:

    ZeroCurve: [1x1 IRDataCurve]
  VolFunctions: {1x19 cell}
   Correlation: [19x19 double]
    NumFactors: NaN
        Period: 1
```

Simulate the term structures for the specified LMM object.

```
[ZeroRates, ForwardRates] = LMM.simTermStructs(20,'nTrials',100);
```

See Also   LiborMarketModel **|** blackvolbyrebonato

Related Examples   • "Price Swaptions With Interest-Rate Models Using Simulation" on page 2-91

LinearGaussian2F

Superclasses

Purpose   Create two-factor additive Gaussian interest-rate model

Description   The two-factor additive Gaussian interest rate-model is specified using the zero curve, a, b, sigma, eta, and rho parameters for these equations:

$$r(t) = x(t) + y(t) + \phi(t)$$

$$dx(t) = -a(t)x(t)dt + \sigma(t)dW_1(t), x(0) = 0$$

$$dy(t) = -b(t)y(t)dt + \eta(t)dW_2(t), y(0) = 0$$

where $dW_1(t)dW_2(t) = \rho dt$ is a two-dimensional Brownian motion with correlation ρ, and φ is a function chosen to match the initial zero curve.

Construction   OBJ = LinearGaussian2F(ZeroCurve,a,b,sigma,eta,rho) constructs an object for a two-factor additive Gaussian interest-rate model.

For example:

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

a = .07;
b = .5;
sigma = .01;
eta = .006;
rho = -.7;

G2PP = LinearGaussian2F(irdc,a,b,sigma,eta,rho);
```

Properties   The following properties are from the `LinearGaussian2F` class.

ZeroCurve

    `ZeroCurve` is specified using `IRDataCurve` or `RateSpec`. This is the zero curve used to evolve the path of future interest rates.

    Attributes:

SetAccess	public
GetAccess	public

a

    Mean reversion for the first factor, specified either as a scalar or function handle which takes time as input and returns a scalar mean reversion value.

    Attributes:

SetAccess	public
GetAccess	public

b

    Mean reversion for the second factor, specified either as a scalar or as a function handle which takes time as input and returns a scalar mean reversion value.

    Attributes:

SetAccess	public
GetAccess	public

sigma

    Volatility for the first factor, specified either as a scalar or function handle which takes time as input and returns a scalar mean volatility.

Attributes:

SetAccess	public
GetAccess	public

eta

Volatility for the second factor specified, either as a scalar or function handle which takes time as input and returns a scalar mean volatility.

Attributes:

SetAccess	public
GetAccess	public

rho

Scalar correlation of the factors.

Attributes:

SetAccess	public
GetAccess	public

Methods

simTermStructs	Simulate term structures for two-factor additive Gaussian interest-rate model

Definitions

Two-Factor Additive Gaussian Interest-Rate Model

Short-rate model based on two factors where the short rate is the sum of the two factors and a deterministic function, in this case $\varphi(t)$, which is chosen to match the initial term structure.

Copy Semantics

Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB documentation.

Examples

Construct a Two-Factor Additive Gaussian Interest-Rate Model

Construct a two-factor additive Gaussian interest-rate model.

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

a = .07;
b = .5;
sigma = .01;
eta = .006;
rho = -.7;

G2PP = LinearGaussian2F(irdc,a,b,sigma,eta,rho)

G2PP =

  LinearGaussian2F with properties:

    ZeroCurve: [1x1 IRDataCurve]
            a: @(t,V)ina
            b: @(t,V)inb
        sigma: @(t,V)insigma
          eta: @(t,V)ineta
          rho: -0.7000
```

Use the `simTermStructs` method to simulate term structures based on the LinearGaussian2F model.

```
 SimPaths = G2PP.simTermStructs(10,'nTrials',100)
```

```
SimPaths(:,:,1) =

    0.0100    0.0180    0.0240    0.0290    0.0330    0.0340    0.0350
    0.0326    0.0375    0.0417    0.0449    0.0446    0.0437    0.0427
    0.0375    0.0402    0.0423    0.0405    0.0387    0.0375    0.0358
    0.0369    0.0389    0.0358    0.0336    0.0329    0.0320    0.0309
    0.0310    0.0252    0.0223    0.0217    0.0216    0.0209    0.0210
    0.0177    0.0165    0.0172    0.0179    0.0181    0.0183    0.0192
    0.0052    0.0076    0.0092    0.0098    0.0102    0.0114    0.0132
    0.0341    0.0351    0.0350    0.0347    0.0346    0.0347    0.0345
    0.0334    0.0327    0.0321    0.0320    0.0320    0.0322    0.0319
    0.0343    0.0338    0.0339    0.0341    0.0343    0.0343    0.0336
    0.0319    0.0318    0.0317    0.0316    0.0315    0.0311    0.0300

.
.
.

SimPaths(:,:,100) =

    0.0100    0.0180    0.0240    0.0290    0.0330    0.0340    0.0350
    0.0264    0.0312    0.0354    0.0387    0.0385    0.0379    0.0373
    0.0469    0.0504    0.0529    0.0513    0.0497    0.0484    0.0462
    0.0525    0.0548    0.0518    0.0496    0.0486    0.0472    0.0451
    0.0499    0.0439    0.0407    0.0397    0.0392    0.0377    0.0364
    0.0561    0.0543    0.0543    0.0541    0.0534    0.0518    0.0500
    0.0510    0.0525    0.0531    0.0526    0.0519    0.0508    0.0493
    0.0443    0.0444    0.0436    0.0427    0.0421    0.0414    0.0404
    0.0205    0.0183    0.0167    0.0160    0.0158    0.0159    0.0162
    0.0246    0.0235    0.0233    0.0234    0.0235    0.0238    0.0236
    0.0238    0.0233    0.0230    0.0229    0.0228    0.0227    0.0220
```

References   Brigo, D. and F. Mercurio, *Interest Rate Models - Theory and Practice*, Springer Finance, 2006.

See Also   HullWhite1F **|** LiborMarketModel **|** simTermStructs **|** capbylg2f | floorbylg2f | swaptionbylg2f

Related Examples

- "Price Swaptions With Interest-Rate Models Using Simulation" on page 2-91
- Class Attributes
- Property Attributes

LinearGaussian2F.simTermStructs

Purpose	Simulate term structures for two-factor additive Gaussian interest-rate model

Syntax	`[ZeroRates,ForwardRates] = simTermStructs(nPeriods)`
	`[ZeroRates,ForwardRates] = simTermStructs(nPeriods,`
	`Name,Value)`

Description	`[ZeroRates,ForwardRates] = simTermStructs(nPeriods)` simulates future zero curve paths using a specified `LinearGaussian2F` object. `[ZeroRates,ForwardRates] = simTermStructs(nPeriods,` `Name,Value)` simulates future zero curve paths using a specified `LinearGaussian2F` object with additional options specified by one or more `Name,Value` pair arguments.
Input Arguments	**nPeriods** Number of simulation periods.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'deltaTime'

Scalar time step between periods.

Default: 1

'nTrials'

Scalar number of trials.

Default: 1

'antithetic'

Boolean scalar flag indicating whether antithetic sampling is used to generate the Gaussian random variates that drive the zero-drift, unit-variance rate Brownian vector $dW(t)$. For details, see simBySolution for the HWV model.

Default: false

'Z'

Direct specification of the dependent random noise process used to generate the zero-drift, unit-variance rate Brownian vector $dW(t)$ that drives the simulation. For details, see simBySolution for the HWV model.

Default: Uses default for simBySolution. If you do not specify a value for Z, simBySolution generates Gaussian variates.

'Tenor'

Numeric vector of maturities to compute at each time step.

Default: tenor of the LinearGaussian2F object's zero curve

Output Arguments

ZeroRates

nPeriods+1-by-nTenors-by-nTrials matrix of simulated zero-rate term structures.

ForwardRates

nPeriods+1-by-nTenors-by-nTrials matrix of simulated forward-rate term structures.

Attributes

Access	public
Static	false
Hidden	false

To learn about attributes of methods, see Method Attributes in the MATLAB Object-Oriented Programming documentation.

Examples

Simulate Term Structures for the LinearGaussian2F Model

Create a two-factor additive Gaussian interest-rate model.

```
Settle = datenum('15-Dec-2007');
CurveTimes = [1:5 7 10 20]';
ZeroRates = [.01 .018 .024 .029 .033 .034 .035 .034]';
CurveDates = daysadd(Settle,360*CurveTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

a = .07;
b = .5;
sigma = .01;
eta = .006;
rho = -.7;

G2PP = LinearGaussian2F(irdc,a,b,sigma,eta,rho)

G2PP =

  LinearGaussian2F with properties:

    ZeroCurve: [1x1 IRDataCurve]
            a: @(t,V)ina
            b: @(t,V)inb
        sigma: @(t,V)insigma
          eta: @(t,V)ineta
```

```
            rho: -0.7000
```

Use the `simTermStructs` method to simulate term structures based
on the `LinearGaussian2F` model.

```
 SimPaths = G2PP.simTermStructs(10,'nTrials',100)

SimPaths(:,:,1) =

SimPaths(:,:,1) =

    0.0100    0.0180    0.0240    0.0290    0.0330    0.0340    0.0350
    0.0214    0.0266    0.0310    0.0345    0.0345    0.0342    0.0339
    0.0379    0.0422    0.0453    0.0442    0.0429    0.0423    0.0408
    0.0445    0.0464    0.0433    0.0410    0.0401    0.0389    0.0372
    0.0491    0.0425    0.0390    0.0378    0.0371    0.0354    0.0341
    0.0301    0.0283    0.0285    0.0287    0.0284    0.0279    0.0279
    0.0231    0.0240    0.0244    0.0241    0.0237    0.0236    0.0239
    0.0345    0.0348    0.0341    0.0334    0.0330    0.0328    0.0324
    0.0435    0.0419    0.0405    0.0398    0.0394    0.0387    0.0376
    0.0419    0.0401    0.0393    0.0388    0.0384    0.0376    0.0362
    0.0431    0.0417    0.0405    0.0397    0.0389    0.0377    0.0355

.
.
.

SimPaths(:,:,100) =

    0.0100    0.0180    0.0240    0.0290    0.0330    0.0340    0.0350
    0.0192    0.0241    0.0284    0.0318    0.0317    0.0314    0.0313
    0.0179    0.0220    0.0253    0.0244    0.0235    0.0236    0.0235
    0.0342    0.0368    0.0341    0.0322    0.0317    0.0311    0.0303
    0.0338    0.0283    0.0256    0.0251    0.0250    0.0244    0.0243
    0.0220    0.0202    0.0205    0.0209    0.0208    0.0207    0.0212
    0.0174    0.0178    0.0179    0.0174    0.0169    0.0169    0.0175
    0.0085    0.0080    0.0070    0.0063    0.0062    0.0068    0.0081
```

```
-0.0047    -0.0058    -0.0063    -0.0061    -0.0054    -0.0038    -0.0016
-0.0100    -0.0113    -0.0114    -0.0109    -0.0102    -0.0086    -0.0064
-0.0129    -0.0125    -0.0119    -0.0110    -0.0101    -0.0084    -0.0064
```

See Also LinearGaussian2F |

Related Examples
- "Price Swaptions With Interest-Rate Models Using Simulation" on page 2-91

Purpose   Price lookback option from Cox-Ross-Rubinstein tree

Syntax   ```
PriceTree = lookbackbycrr(CRRTree, OptSpec, Strike,
Settle, ExerciseDates, AmericanOpt)
```

**Arguments**

| | |
|---|---|
| CRRTree | Stock tree structure created by `crrtree`. |
| OptSpec | Number of instruments (NINST)-by-1 cell array of strings `'call'` or `'put'`. |
| Strike | NINST-by-1 vector of strike price values. Each row is the schedule for one option. To calculate the value of a floating-strike lookback option, specify Strike as NaN. |
| Settle | NINST-by-1 vector of Settle dates. The settle date for every lookback is set to the valuation date of the stock tree. The lookback argument Settle is ignored. |
| ExerciseDates | For a European option (AmericanOpt = 0): |
| | NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date. |
| | For an American option (AmericanOpt = 1): |
| | NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised |

# lookbackbycrr

<table>
<tr><td></td><td>between the valuation date of the stock tree and the single listed exercise date.</td></tr>
<tr><td>AmericanOpt</td><td>(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.</td></tr>
</table>

**Description**  PriceTree = lookbackbycrr(CRRTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt) calculates the value of fixed- and floating-strike lookback options. Data arguments are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Price is a NINST-by-1 vector of expected option prices at time 0.

---

**Note** lookbackbycrr calculates values of fixed and floating strike lookback options. To compute the value of a floating strike lookback option, strike should be specified as NaN. Pricing of lookback options is done using Hull-White (1993). Consequently, for these options there are not unique prices on the tree nodes with the exception of the root node.

---

**Examples**  Price a lookback option using a CRR binomial tree.

Load the file deriv.mat, which provides CRRTree. The CRRTree structure contains the stock specification and time information needed to price the option.

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'Call';
Strike = 115;
```

```
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2006';

Price = lookbackbycrr(CRRTree, OptSpec, Strike, Settle, ...
ExerciseDates)

Price =

    7.6015
```

References Hull, J., and A. White, "Efficient Procedures for Valuing European and American Path-Dependent Options," *Journal of Derivatives*, Fall 1993, pp. 21-31.

See Also crrtree | instlookback

lookbackbyeqp

Purpose   Price lookback option from Equal Probabilities binomial tree

Syntax   [Price, PriceTree] = lookbackbyeqp(EQPTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt)

Arguments

EQPTree	Stock tree structure created by eqptree.
OptSpec	Number of instruments (NINST)-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option. To calculate the value of a floating-strike lookback option, specify Strike as NaN.
Settle	NINST-by-1 vector of Settle dates. The settle date for every lookback is set to the valuation date of the stock tree. The lookback argument Settle is ignored.
ExerciseDates	For a European option (AmericanOpt = 0):
	NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option (AmericanOpt = 1):
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised

	between the valuation date of the stock tree and the single listed exercise date.
AmericanOpt	(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.

Description   Price = lookbackbyeqp(EQPTree, OptSpec, Strike, ExerciseDates, AmericanOpt) calculates the value of fixed- and floating-strike lookback options. Data arguments are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Price is a NINST-by-1 vector of expected option prices at time 0.

Note lookbackbyeqp calculates values of fixed and floating strike lookback options. To compute the value of a floating strike lookback option, strike should be specified as NaN. Pricing of lookback options is done using Hull-White (1993). Consequently, for these options there are not unique prices on the tree nodes with the exception of the root node.

Examples   Price a lookback option using an EQP equity tree.

Load the file deriv.mat, which provides EQPTree. The EQPTree structure contains the stock specification and time information needed to price the option.

load deriv.mat

Set the required values. Other arguments will use defaults.

OptSpec = 'Call';
Strike = 115;

```
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2006';

Price = lookbackbyeqp(EQPTree, OptSpec, Strike, Settle, ...
ExerciseDates)

Price =

 8.7941
```

References   Hull, J., and A. White, "Efficient Procedures for Valuing European and American Path-Dependent Options," *Journal of Derivatives*, Fall 1993, pp. 21-31.

See Also   eqptree | instlookback

Purpose	Price lookback option using implied trinomial tree (ITT)
Syntax	[Price, PriceTree] = lookbackbyitt(ITTTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt)

Arguments

ITTTree	Stock tree structure created by `itttree`.
OptSpec	Number of instruments (NINST)-by-1 cell array of strings `'call'` or `'put'`.
Strike	NINST-by-1 vector of strike price values. Each row is the schedule for one option. To calculate the value of a floating-strike lookback option, specify Strike as NaN.
Settle	NINST-by-1 vector of Settle dates. The settle date for every lookback option is set to the ValuationDate of the stock tree. The lookback argument Settle is ignored.
ExerciseDates	For a European option (AmericanOpt = 0): NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date. For an American option (AmericanOpt = 1): NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any tree date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised

lookbackbyitt

	between the valuation date of the stock tree and the single listed exercise date.
AmericanOpt	(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European option. If AmericanOpt = 1, the option is an American option.

Description   Price = lookbackbyitt(ITTTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt) calculates the value of fixed- and floating-strike lookback options. Data arguments for lookbackbyitt are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument; the others may be omitted or passed as empty matrices [].

Price is a NINST-by-1 vector of expected option prices at time 0.

Note lookbackbyitt calculates values of fixed and floating strike lookback options. To compute the value of a floating strike lookback option, strike should be specified as NaN. Pricing of lookback options is done using Hull-White (1993). Consequently, for these options there are not unique prices on the tree nodes with the exception of the root node.

Examples   Price a lookback option using an ITT equity tree.

Load the file deriv.mat which provides the ITTTree. The ITTTree structure contains the stock specification and time information needed to price the option.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'Call';
Strike = 85;
```

```
Settle = '01-Jan-2006';
ExerciseDates = '01-Jan-2008';

Price = lookbackbyitt(ITTTree, OptSpec, Strike, Settle, ExerciseDates)

Price =

    0.5426
```

References   Hull, J., and A. White, "Efficient Procedures for Valuing European and American Path-Dependent Options," *Journal of Derivatives*, Fall 1993, pp. 21-31.

See Also   instlookback | itttree

lrtimespec

Purpose   Specify time structure for Leisen-Reimer binomial tree

Syntax   TimeSpec = lrtimespec(ValuationDate, Maturity, NumPeriods)

Description `TimeSpec = lrtimespec(ValuationDate, Maturity, NumPeriods)`
                 specifies a time structure for a Leisen-Reimer stock tree.

Input Arguments

ValuationDate

Scalar date marking the pricing date and first observation in the Leisen-Reimer stock tree. Specify `ValuationDate` as a serial date number or date string.

Maturity

Scalar date marking the depth of the Leisen-Reimer stock tree.

NumPeriods

Scalar value determining how many time steps are in the Leisen-Reimer stock tree.

Note Leisen-Reimer requires the number of steps to be an odd number.

Output Arguments

TimeSpec

Structure specifying the time layout for a Leisen-Reimer stock tree.

Examples   Specify a 5-period tree with time steps of 1 year:

```
ValuationDate = '1-July-2010';
Maturity = '1-July-2015';
TimeSpec = lrtimespec(ValuationDate, Maturity, 5)

TimeSpec =
```

```
         FinObj: 'BinTimeSpec'
 ValuationDate: 734320
      Maturity: 736146
     NumPeriods: 5
          Basis: 0
   EndMonthRule: 1
           tObs: [0 1 2 3 4 5]
           dObs: [734320 734685 735050 735415 735780 736146]
```

References   Leisen D.P., M. Reimer, "Binomial Models for Option Valuation –
Examining and Improving Convergence," *Applied Mathematical
Finance*, Number 3, 1996, pp. 319-346.

See Also | stockspec | lrtree

lrtree

Purpose	Build Leisen-Reimer stock tree
Syntax	LRTree = lrtree(StockSpec, RateSpec, TimeSpec, Strike) LRTree = lrtree(StockSpec, RateSpec, TimeSpec, Strike, Name,Value)
Description	LRTree = lrtree(StockSpec, RateSpec, TimeSpec, Strike) constructs a Leisen-Reimer stock tree. LRTree = lrtree(StockSpec, RateSpec, TimeSpec, Strike, Name,Value) constructs a Leisen-Reimer stock tree with additional options specified by one or more Name,Value pair arguments.

Input Arguments

StockSpec

Stock specification. For more information, see stockspec.

RateSpec

Interest rate specification of the initial risk-free rate curve. For more information, see intenvset.

TimeSpec

Tree time layout specification. For more information, see lrtimespec.

Strike

Scalar defining the option strike.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Method'

String value. Use PP1 for Peizer-Pratt method 1 inversion and PP2 for Peizer-Pratt method 2 inversion. For more information on PP1 and PP2 methods, see "Leisen-Reimer Tree (LR) Modeling" on page C-7.

Default: PP1

Output Arguments

LRTree

Structure specifying stock and time information for a Leisen-Reimer tree.

Examples

Consider a European put option with an exercise price of $30 that expires on June 1, 2010. The underlying stock is trading at $30 on January 1, 2010 and has a volatility of 30% per annum. The annualized continuously compounded risk-free rate is 5% per annum. Using this data, create a Leisen-Reimer tree with 101 steps using the PP1 method.

```
AssetPrice = 30;
Strike = 30;

ValuationDate = 'Jan-1-2010';
Maturity = 'June-1-2010';

% Define StockSpec
Sigma = 0.3;
StockSpec = stockspec(Sigma, AssetPrice);

% Define RateSpec
Rates = 0.05;
Settle = ValuationDate;
Basis = 1;
Compounding = -1;

RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', Basis);

% Build the Leisen-Reimer (LR) tree with 101 steps
```

```
LRTimeSpec = lrtimespec(ValuationDate, Maturity, 101);

% Use the PP1 method
LRMethod  = 'PP1';

LRTree = lrtree(StockSpec, RateSpec, LRTimeSpec, Strike, ...
'method', LRMethod)

LRTree =

      FinObj: 'BinStockTree'
      Method: 'LR'
   Submethod: 'PP1'
      Strike: 30
   StockSpec: [1x1 struct]
    TimeSpec: [1x1 struct]
    RateSpec: [1x1 struct]
        tObs: [1x102 double]
        dObs: [1x102 double]
       STree: {1x102 cell}
      UpProbs: [101x1 double]
```

References   Leisen D.P., M. Reimer, "Binomial Models for Option Valuation –
Examining and Improving Convergence," *Applied Mathematical
Finance*, Number 3, 1996, pp. 319-346.

See Also | stockspec | lrtimespec | intenvset | optstockbylr |
optstocksensbylr

Purpose   Determine European rainbow option price on maximum of two risky assets using Stulz option pricing model

Syntax   Price = maxassetbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec1	Stock specification for asset 1. See stockspec.
StockSpec2	Stock specification for asset 2. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
Corr	NINST-by-1 vector of correlation between the underlying asset prices.

Description Price = maxassetbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr) computes rainbow option prices using the Stulz option pricing model.

Price is a NINST-by-1 vector of expected option prices.

Examples   Consider a European rainbow option that gives the holder the right to buy either $100,000 worth of an equity index at a strike price of 1000 (asset 1) or $100,000 of a government bond (asset 2) with a strike price of 100% of face value, whichever is worth more at the end of 12 months. On January 15, 2008, the equity index is trading at 950, pays a dividend

of 2% annually and has a return volatility of 22%. Also on January 15, 2008, the government bond is trading at 98, pays a coupon yield of 6%, and has a return volatility of 15%. The risk-free rate is 5%. Using this data, if the correlation between the rates of return is -0.5, 0, and 0.5, calculate the price of the European rainbow option.

Since the asset prices in this example are in different units, it is necessary to work in either index points (asset 1) or in dollars (asset 2). The European rainbow option allows the holder to buy the following: 100 units of the equity index at $1000 each (for a total of $100,000) or 1000 units of the government bonds at $100 each (for a total of $100,000). To convert the bond price (asset 2) to index units (asset 1), you must make the following adjustments:

- Multiply the strike price and current price of the government bond by 10 (1000/100).

- Multiply the option price by 100, considering that there are 100 equity index units in the option.

Once these adjustments are introduced, the strike price is the same for both assets ($1000).

Create the `RateSpec`:

```
Settle = 'Jan-15-2008';
Maturity = 'Jan-15-2009';
Rates = 0.05;
Basis = 1;

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1, 'Basis', Basis);
```

Create the two `StockSpec` definitions:

```
AssetPrice1 = 950;   % Asset 1 => Equity index
AssetPrice2 = 980;   % Asset 2 => Government bond
Sigma1 = 0.22;
Sigma2 = 0.15;
Div1 = 0.02;
```

```
Div2 = 0.06;

StockSpec1 = stockspec(Sigma1, AssetPrice1, 'continuous', Div1);
StockSpec2 = stockspec(Sigma2, AssetPrice2, 'continuous', Div2);
```

Calculate the price of the options for different correlation levels:

```
Strike = 1000 ;
Corr = [-0.5; 0; 0.5];
OptSpec = 'call';

Price = maxassetbystulz(RateSpec, StockSpec1, StockSpec2,...
Settle, Maturity, OptSpec, Strike, Corr)

Price =

  111.6683
  103.7715
   92.4412
```

These are the prices of one unit. This means that the premium is 11166.83, 10377.15, and 9244.12 (for 100 units).

See Also   intenvset | maxassetsensbystulz | minassetbystulz | stockspec

maxassetsensbystulz

Purpose   Determine European rainbow option prices and sensitivities on maximum of two risky assets using Stulz pricing model

Syntax   PriceSens = maxassetsensbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr)
PriceSens = maxassetsensbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr, OutSpec)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec1	Stock specification for asset 1. See stockspec.
StockSpec2	Stock specification for asset 2. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
Corr	NINST-by-1 vector of correlation between the underlying asset prices.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are:
	• NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs

for the function. Possible values are `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, or `'All'`.

For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output should be Price, Lambda, and Rho, in that order.

To invoke from a function: [Price, Lambda, Rho] = maxassetsensbystulz(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

- Default is OutSpec = {'Price'}.

Description PriceSens = maxassetsensbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr) computes rainbow option prices using the Stulz option pricing model.

PriceSens = maxassetsensbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr, OutSpec) computes rainbow option prices and sensitivities using the Stulz option pricing model.

PriceSens is a NINST-by-1 or NINST-by-2 vector of expected prices and sensitivities values.

Examples Consider a European rainbow option that gives the holder the right to buy either $100,000 of an equity index at a strike price of 1000 (asset 1) or $100,000 of a government bond (asset 2) with a strike price of 100% of face value, whichever is worth more at the end of 12 months. On

January 15, 2008, the equity index is trading at 950, pays a dividend of 2% annually, and has a return volatility of 22%. Also on January 15, 2008, the government bond is trading at 98, pays a coupon yield of 6%, and has a return volatility of 15%. The risk-free rate is 5%. Using this data, calculate the price and sensitivity of the European rainbow option if the correlation between the rates of return is -0.5, 0, and 0.5.

Since the asset prices in this example are in different units, it is necessary to work in either index points (for asset 1) or in dollars (for asset 2). The European rainbow option allows the holder to buy the following: 100 units of the equity index at $1000 each (for a total of $100,000) or 1000 units of the government bonds at $100 each (for a total of $100,000). To convert the bond price (asset 2) to index units (asset 1), you must make the following adjustments:

- Multiply the strike price and current price of the government bond by 10 (1000/100).

- Multiply the option price by 100, considering that there are 100 equity index units in the option.

Once these adjustments are introduced, the strike price is the same for both assets ($1000).

Create the `RateSpec`:

```
Settle = 'Jan-15-2008';
Maturity = 'Jan-15-2009';
Rates = 0.05;
Basis = 1;

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1, 'Basis', Basis);
```

Create the two `StockSpec` definitions:

```
AssetPrice1 = 950;   % Asset 1 => Equity index
AssetPrice2 = 980;   % Asset 2 => Government bond
Sigma1 = 0.22;
Sigma2 = 0.15;
```

```
Div1 = 0.02;
Div2 = 0.06;

StockSpec1 = stockspec(Sigma1, AssetPrice1, 'continuous', Div1);
StockSpec2 = stockspec(Sigma2, AssetPrice2, 'continuous', Div2);
```

Calculate the price and delta for different correlation levels:

```
Strike = 1000 ;
Corr = [-0.5; 0; 0.5];
OutSpec = {'price'; 'delta'};
OptSpec = 'call';
[Price, Delta] = maxassetsensbystulz(RateSpec, StockSpec1, StockSpec2,...
Settle, Maturity, OptSpec, Strike, Corr,'OutSpec', OutSpec)

Price =

  111.6683
  103.7715
   92.4412

Delta =

    0.4594    0.3698
    0.4292    0.3166
    0.4053    0.2512
```

The output `Delta` has two columns: the first column represents the `Delta` with respect to the equity index (asset 1), and the second column represents the `Delta` with respect to the government bond (asset 2). The value 0.4595 represents `Delta` with respect to one unit of the equity index. Since there are 100 units of the equity index, the overall `Delta` would be 45.94 (100 * 0.4594) for a correlation level of -0.5. To calculate the `Delta` with respect to the government bond, remember that an adjusted price of 980 was used instead of 98. Therefore, for example, the `Delta` with respect to government bond, for a correlation of 0.5 would be 251.2 (0.2512 * 100 * 10).

maxassetsensbystulz

See Also   `intenvset | maxassetbystulz | stockspec`

Purpose   Determine European rainbow option prices on minimum of two risky assets using Stulz option pricing model

Syntax   Price = minassetbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec1	Stock specification for asset 1. See stockspec.
StockSpec2	Stock specification for asset 2. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
Corr	NINST-by-1 vector of correlation between the underlying asset prices.

Description   Price = minassetbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr) computes option prices using the Stulz option pricing model.

Price is a NINST-by-1 vector of expected option prices.

Examples   Consider a European rainbow put option that gives the holder the right to sell either stock A or stock B at a strike of 50.25, whichever has the lower value on the expiration date May 15, 2009. On November 15, 2008, stock A is trading at 49.75 with a continuous annual dividend yield of 4.5% and has a return volatility of 11%. Stock B is trading at 51 with a continuous dividend yield of 5% and has a return volatility

of 16%. The risk-free rate is 4.5%. Using this data, if the correlation between the rates of return is -0.5, 0, and 0.5, calculate the price of the minimum of two assets that are European rainbow put options.

Create the `RateSpec`:

```
Settle = 'Nov-15-2008';
Maturity = 'May-15-2009';
Rates = 0.045;
Basis = 1;

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1, 'Basis', Basis);
```

Create the two `StockSpec` definitions:

```
AssetPriceA = 49.75;
AssetPriceB = 51;
SigmaA = 0.11;
SigmaB = 0.16;
DivA = 0.045;
DivB = 0.05;

StockSpecA = stockspec(SigmaA, AssetPriceA, 'continuous', DivA);
StockSpecB = stockspec(SigmaB, AssetPriceB, 'continuous', DivB);
```

Compute the price of the options for different correlation levels:

```
Strike = 50.25;
Corr = [-0.5;0;0.5];
OptSpec = 'put';
Price = minassetbystulz(RateSpec, StockSpecA, StockSpecB, Settle,...
Maturity, OptSpec, Strike, Corr)

Price =

    3.4320
    3.1384
    2.7694
```

The values 3.43, 3.14, and 2.77 are the price of the European rainbow put options with a correlation level of -0.5, 0, and 0.5 respectively.

See Also   `intenvset` | `maxassetbystulz` | `minassetsensbystulz` | `stockspec`

minassetsensbystulz

Purpose   Determine European rainbow option prices and sensitivities on minimum of two risky assets using Stulz pricing model

Syntax

```
PriceSens = minassetsensbystulz(RateSpec, StockSpec1,
StockSpec2, Settle, Maturity, OptSpec, Strike, Corr)
PriceSens = minassetsensbystulz(RateSpec, StockSpec1,
StockSpec2, Settle, Maturity, OptSpec, Strike,
Corr, OutSpec)
```

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see `intenvset`.
StockSpec1	Stock specification for asset 1. See `stockspec`.
StockSpec2	Stock specification for asset 2. See `stockspec`.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings `'call'` or `'put'`.
Strike	NINST-by-1 vector of strike price values.
Corr	NINST-by-1 vector of correlation between the underlying asset prices.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are: • NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs

for the function. Possible values are `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, or `'All'`.

For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output should be Price, Lambda, and Rho, in that order.

To invoke from a function: [Price, Lambda, Rho] = minassetsensbystulz(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

- Default is OutSpec = {'Price'}.

Description　PriceSens = minassetsensbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr) computes rainbow option prices using the Stulz option pricing model.

PriceSens = minassetsensbystulz(RateSpec, StockSpec1, StockSpec2, Settle, Maturity, OptSpec, Strike, Corr, OutSpec) computes rainbow option prices and sensitivities using the Stulz option pricing model.

PriceSens is a NINST-by-1 or NINST-by-2 vector of expected prices and sensitivities.

Examples　Consider a European rainbow put option that gives the holder the right to sell either stock A or stock B at a strike of 50.25, whichever has the lower value on the expiration date May 15, 2009. On November 15, 2008, stock A is trading at 49.75 with a continuous annual dividend

yield of 4.5% and has a return volatility of 11%. Stock B is trading at 51 with a continuous dividend yield of 5% and has a return volatility of 16%. The risk-free rate is 4.5%. Using this data, if the correlation between the rates of return is -0.5, 0, and 0.5, calculate the price and sensitivity of the minimum of two assets that are European rainbow put options.

Create the `RateSpec`:

```
Settle = 'Nov-15-2008';
Maturity = 'May-15-2009';
Rates = 0.045;
Basis = 1;

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1, 'Basis', Basis);
```

Create the two `StockSpec` definitions:

```
AssetPriceA = 49.75;
AssetPriceB = 51;
SigmaA = 0.11;
SigmaB = 0.16;
DivA = 0.045;
DivB = 0.05;

StockSpecA = stockspec(SigmaA, AssetPriceA, 'continuous', DivA);
StockSpecB = stockspec(SigmaB, AssetPriceB, 'continuous', DivB);
```

Calculate price and delta for different correlation levels:

```
Strike = 50.25;
Corr = [-0.5;0;0.5];
OptSpec = 'put';
OutSpec = {'Price'; 'delta'};
[P, D] = minassetsensbystulz(RateSpec, StockSpecA, StockSpecB,...
Settle, Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec)

P =
```

```
         3.4320
         3.1384
         2.7694

    D =

        -0.4183   -0.3496
        -0.3746   -0.3189
        -0.3304   -0.2905
```

The output Delta has two columns: the first column represents the Delta with respect to the stock A (asset 1), and the second column represents the Delta with respect to the stock B (asset 2). The value 0.4183 represents Delta with respect to the stock A for a correlation level of -0.5. The Delta with respect to stock B, for a correlation of zero is -0.3189.

See Also   intenvset | minassetbystulz | stockspec

mkbush

Purpose	Create bushy tree
Syntax	[Tree, NumStates] = mkbush(NumLevels, NumChild, NumPos, Trim, NodeVal)

Arguments

NumLevels	Number of time levels of the tree.
NumChild	1-by- number of levels (NUMLEVELS) vector with number of branches (children) of the nodes in each level.
NumPos	1-by-NUMLEVELS vector containing the length of the state vectors in each time level.
Trim	(Optional) Scalar 0 or 1. If Trim = 1, NumPos decreases by 1 when moving from one time level to the next. Otherwise, if Trim = 0 (Default), NumPos does not decrease.
NodeVal	(Optional) Initial value at each node of the tree. Default = NaN.

Description [Tree, NumStates] = mkbush(NumLevels, NumChild, NumPos, Trim,
NodeVal) creates a bushy tree Tree with initial values NodeVal at each node. NumStates is a 1-by-NUMLEVELS vector containing the number of state vectors in each level.

Examples Create a tree with four time levels, two branches per node, and a vector of three elements in each node with each element initialized to NaN.

```
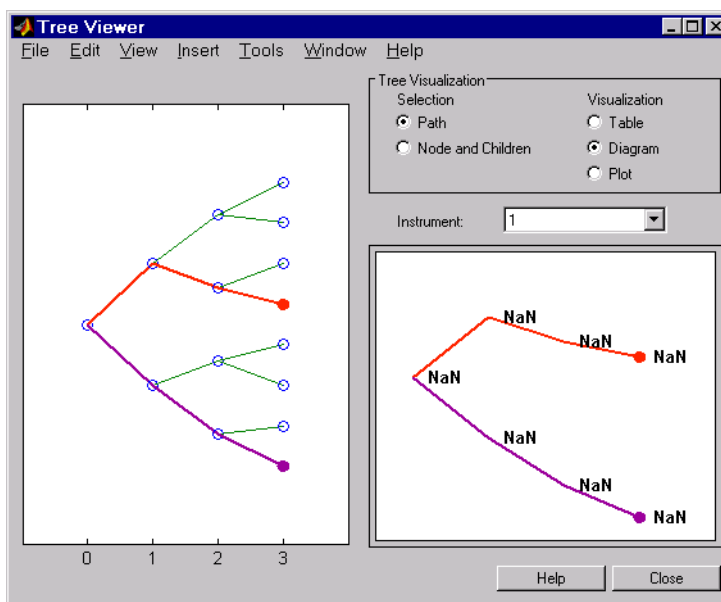Tree = mkbush(4, 2, 3);
treeviewer(Tree)
```

See Also   bushpath | bushshape

mktree

Purpose	Create recombining binomial tree

Syntax

```
Tree = mktree(NumLevels, NumPos, NodeVal, IsPriceTree)
```

Arguments

NumLevels	Number of time levels of the tree.
NumPos	1-by-NUMLEVELS vector containing the length of the state vectors in each time level.
NodeVal	(Optional) Initial value at each node of the tree. Default = NaN.
IsPriceTree	(Optional) Boolean determining if a final horizontal branch is added to the tree. Default = 0.

Description Tree = mktree(NumLevels, NumPos, NodeVal, IsPriceTree) creates a recombining tree Tree with initial values NodeVal at each node.

Examples Create a recombining tree of four time levels with a vector of two elements in each node and each element initialized to NaN.

```
Tree = mktree(4, 2);
```

See Also treepath | treeshape

mktrintree

Purpose   Create recombining trinomial tree

Syntax   TrinTree = mktrintree(NumLevels, NumPos, NumStates, NodeVal)

Arguments

NumLevels	Number of time levels of the tree.
NumPos	1-by-NUMLEVELS vector containing the length of the state vectors in each time level.
NumStates	1-by-NUMLEVELS vector containing the number of state vectors in each time level.
NodeVal	(Optional) Initial value at each node of the tree. Default = NaN.

Description   TrinTree = mktrintree(NumLevels, NumPos, NumStates, NodeVal) creates a recombining tree Tree with initial values NodeVal at each node.

Examples   Create a recombining trinomial tree of four time levels with a vector of two elements in each node and each element initialized to NaN.

TrinTree = mktrintree(4, [2 2 2 2], [1 3 5 7]);

See Also   trintreepath | trintreeshape

mmktbybdt

Purpose   Create money-market tree from Black-Derman-Toy interest-rate tree

Syntax   MMktTree = mmktbybdt(BDTTree)

Arguments

    BDTTree   Interest-rate tree structure created by bdttree.

Description MMktTree = mmktbybdt(BDTTree) creates a money-market tree from
an interest-rate tree structure created by bdttree.

Examples
```
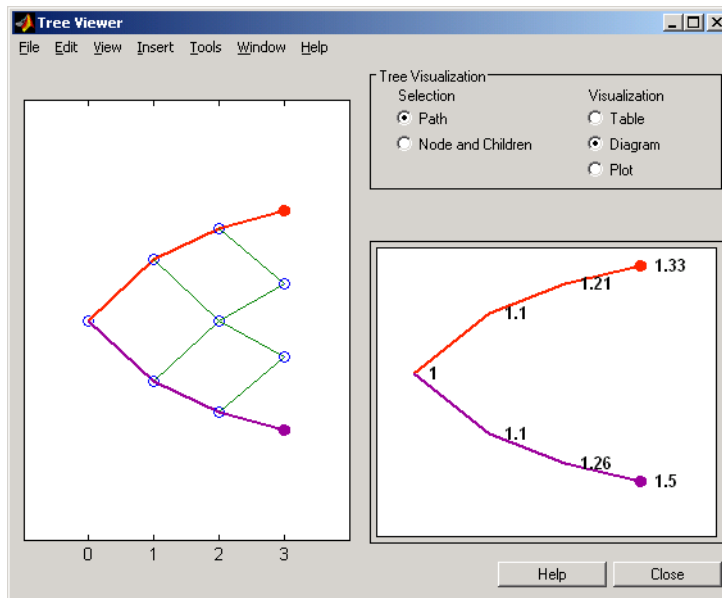load deriv.mat;
MMktTree = mmktbybdt(BDTTree);
treeviewer(MMktTree)
```

See Also   bdttree

mmktbyhjm

Purpose   Create money-market tree from Heath-Jarrow-Morton interest-rate tree

Syntax   MMktTree = mmktbyhjm(HJMTree)

Arguments

HJMTree	Forward-rate tree structure created by hjmtree.

Description   MMktTree = mmktbyhjm(HJMTree) creates a money-market tree from a forward-rate tree structure created by hjmtree.

Examples
```
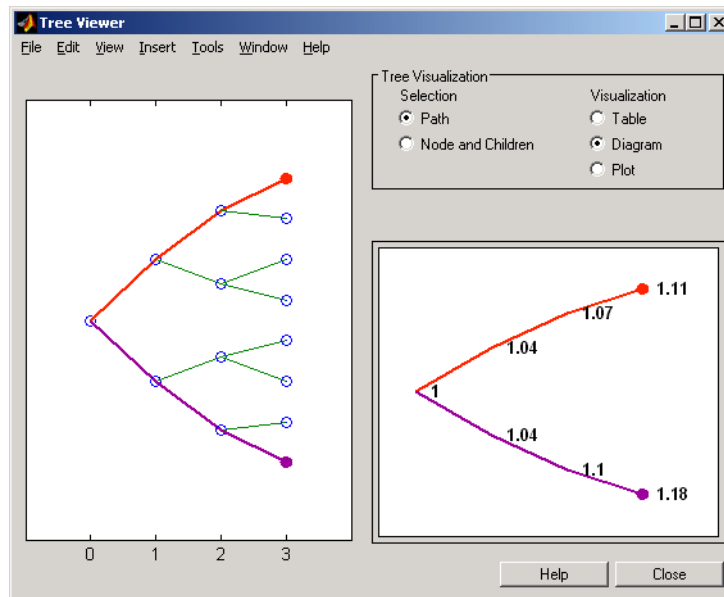load deriv.mat;
MMktTree = mmktbyhjm(HJMTree);
treeviewer(MMktTree)
```

See Also   hjmtree

numerix

Purpose   Create numerix object to set up Numerix CAIL environment

Description   The numerix object makes the Numerix engine directly accessible from
                     MATLAB. To use the capabilities of Numerix CAIL, you must have
                     CAIL client software installed on your local desktop.

Construction   N = numerix(DATADIRECTORYPATH) constructs the numerix object and
                     sets up the Numerix CrossAsset Integration Layer (CAIL) environment
                     given the path, DATADIRECTORYPATH.

Properties   The following properties are from the numerix class.

Path

Defines the path for DATADIRECTORYPATH. This path is the location
of the templates and is created by the client installation of
CAIL. At the highest level, a template defines the interface; it
encapsulates the instructions for performing the calculation, the
calculation's required and optional input parameters, and the
calculation's outputs.

Attributes:

SetAccess	public
GetAccess	public

RepositoryPath

RepositoryPath defines the path location for the repository
directory in a file system.

Attributes:

SetAccess	public
GetAccess	public

Repository

Repositories are collections of templates and are defined as a directory in a file system.

Attributes:

SetAccess	public
GetAccess	public

Context

The calculation context manages all of the CAIL information.`Context` contains the location of the template repository and is responsible for creating a CAIL application context in which to perform the calculations.

Attributes:

SetAccess	public
GetAccess	public

LookupsPath

Defines the path for the numeric instruments data types.

Attributes:

SetAccess	public
GetAccess	public

MarketsPath

Defines a the path for the logical schema for naming all of the market data. `MarketsPath` enables you to provide a data dictionary to map business market data to CAIL to reduce the task of inputting market data into CAIL objects directly.

Attributes:

| SetAccess | public |
| GetAccess | public |

FixingsPath

Defines the path for the schema for naming historical fixing data for rates and prices.

Attributes:

| SetAccess | public |
| GetAccess | public |

TradesPath

Defines the path to the trade instrument definitions.

Attributes:

| SetAccess | public |
| GetAccess | public |

Parameters

Defines the calculation parameters and market data, if available.

Attributes:

| SetAccess | public |
| GetAccess | public |

Methods

| parseResults | Converts Numerix CAIL data to MATLAB data types |

Definitions

CrossAsset Integration Layer (CAIL)

The CrossAsset Integration Layer (CAIL) is an application programming interface (API), which extends the data-driven approach of Numerix.

The calculation workflow of CAIL is:.

1 Select a trade template for a specified deal or trade from the repository. The trade template specifies a set of inputs, a set of outputs, and the dependencies on other information (model, market data, calendar, etc.).

2 Provide the input parameters to the trade template and call the calculation context. The calculation context follows the dependency path to collect the needed information and produces the output specified by the template.

numerix

Copy Semantics
Value. To learn how value classes affect copy operations, see Copying Objects in the MATLAB documentation.

Examples

Construct a Numerix Object

Initialize Numerix CAIL environment.

```
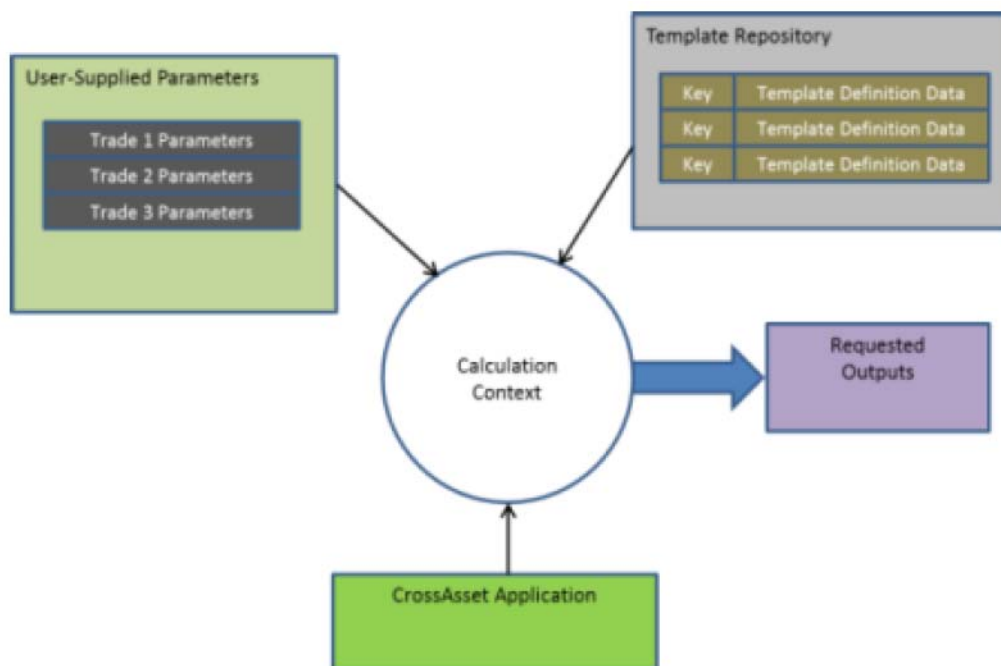import com.numerix.integration.*;
import com.numerix.integration.implementation.*;
```

Construct a numerix object.

```
n = numerix('i:\NumeriX_java_10_3_0\data')

n =

            Path: 'i:\NumeriX_java_10_3_0\data'
  RepositoryPath: 'i:\NumeriX_java_10_3_0\data\Repository'
      Repository: [1x1 com.numerix.integration.implementation.FileSystemRepository]
         Context: [1x1 com.numerix.integration.implementation.LocalCalculationContext]
     LookupsPath: 'i:\NumeriX_java_10_3_0\data\Data\LookupRules'
     MarketsPath: 'i:\NumeriX_java_10_3_0\data\Data\Markets'
     FixingsPath: 'i:\NumeriX_java_10_3_0\data\Data\Fixings'
      TradesPath: 'i:\NumeriX_java_10_3_0\data\Data\Trades'
      Parameters: [1x1 com.numerix.integration.implementation.CalculationParameters]
```

See Also   parseResults **|**

Related Examples
- "Working with Simple Numerix Trades" on page 10-2
- "Working with Advanced Numerix Trades" on page 10-5
- "Use Numerix to Price Cash Deposits" on page 10-12
- "Use Numerix for Interest-Rate Risk Assessment" on page 10-15
- Class Attributes
- Property Attributes

External Web Sites
- http://www.numerix.com/cail

Purpose	Converts Numerix CAIL data to MATLAB data types
Syntax	R = parseResults(N,Results)
Description	R = parseResults(N,Results) returns Numerix CAIL data in native MATLAB data types.

Input Arguments

N

Numerix object constructed using numerix.

Results

Result instances for each trade instance.

Output Arguments

R

Results from Numerix table output represented as MATLAB data types.

Attributes

Access	public
Static	false
Hidden	false

To learn about attributes of methods, see Method Attributes in the MATLAB Object-Oriented Programming documentation.

Examples

Return Results for Numerix CAIL API to Price a Callable Reverse Floater

Initialize Numerix environment.

```
import com.numerix.integration.*;
import com.numerix.integration.implementation.*;

n = numerix('i:\NumeriX_java_10_3_0\data')
```

```
n =

                Path: 'i:\NumeriX_java_10_3_O\data'
      RepositoryPath: 'i:\NumeriX_java_10_3_O\data\Repository'
          Repository: [1x1 com.numerix.integration.implementation.FileSystemRepository]
             Context: [1x1 com.numerix.integration.implementation.LocalCalculationContext]
         LookupsPath: 'i:\NumeriX_java_10_3_O\data\Data\LookupRules'
         MarketsPath: 'i:\NumeriX_java_10_3_O\data\Data\Markets'
         FixingsPath: 'i:\NumeriX_java_10_3_O\data\Data\Fixings'
          TradesPath: 'i:\NumeriX_java_10_3_O\data\Data\Trades'
          Parameters: [1x1 com.numerix.integration.implementation.CalculationParameters]
```

Create a market.

```
quotes = java.util.HashMap;
quotes.put('IR.USD-LIBOR-3M.SWAP-1Y.MID', 0.0066056);
quotes.put('IR.USD-LIBOR-3M.SWAP-10Y.MID', 0.022465005);
quotes.put('IR.USD-LIBOR-3M.SWAP-20Y.MID', 0.027544995);
market = Market('EOD_14-NOV-2011', DateExtensions.date('14-Nov-2011'), quotes.entrySet);
```

Define a trade instance based on instrument template found in the `Repository`.

```
tradeDescriptor = 'TRADE.IR.CALLABLEREVERSEFLOATER';
tradeParameters = java.util.HashMap;
tradeParameters.put('Trade ID','1001');
tradeParameters.put('Quote Type', 'MID');
tradeParameters.put('Currency', 'USD');
tradeParameters.put('Notional', 1000000.0);
tradeParameters.put('Effective Date', DateExtensions.date('1-Dec-2011'));
tradeParameters.put('Termination Date', DateExtensions.date('1-Dec-2021'));
tradeParameters.put('IR Index', 'LIBOR');
tradeParameters.put('IR Index Tenor', '3M');
tradeParameters.put('Structured Freq', '3M');
tradeParameters.put('Structured Side', 'Receive');
tradeParameters.put('Structured Coupon Floor', 0.0);
tradeParameters.put('Structured Coupon UpBd', 0.08);
tradeParameters.put('StructuredCoupon Multiplier', 1.4);
```

```
tradeParameters.put('Structured Coupon Cap', 0.05);
tradeParameters.put('Structured Basis', 'ACT/360');
tradeParameters.put('Funding Freq', '3M');
tradeParameters.put('Funding Side', 'Pay');
tradeParameters.put('Funding Spread', 0.003);
tradeParameters.put('Funding Basis', 'ACT/360');
tradeParameters.put('Call Start Date', DateExtensions.date('1-Dec-2013'));
tradeParameters.put('Call End Date', DateExtensions.date('1-Dec-2020'));
tradeParameters.put('Option Side', 'Short');
tradeParameters.put('Option Type', 'Right to Terminate');
tradeParameters.put('Call Frequency', '3M');
tradeParameters.put('Model', 'IR.USD-LIBOR-3M.MID.DET');
tradeParameters.put('Method', 'BackwardAnalytic');
```

Create a trade instance.

```
trade = RepositoryExtensions.createTradeInstance(n.Repository, tradeDescriptor, tradeParameters);
```

Price the trades.

```
results = CalculationContextExtensions.calculate(n.Context, trade, market, Request.getAll);
```

Parse the results for MATLAB and display.

```
r = n.parseResults(results)
disp([r.Name r.Category r.Currency r.Data])

r =

    Category: {13x1 cell}
    Currency: {13x1 cell}
        Name: {13x1 cell}
        Data: {13x1 cell}

    'Reporting Currency'        'Price'       ''       'USD'
    'Structured Cashflow Log'   'Cashflow'    ''        {41x20 cell}
    'Structured Leg PV Accrued' 'Price'       'USD'    [          0]
    'PV'                        'Price'       'USD'    [ 6.4133e+04]
```

```
'Structured Leg PV Clean'    'Price'      'USD'    [ 4.2637e+05]
'Option PV'                  'Price'      'USD'    [-1.3220e+05]
'Funding Cashflow Log'       'Cashflow'   ''          {41x20 cell}
'Structured Leg PV'          'Price'      'USD'    [ 4.2637e+05]
'Funding Leg PV'             'Price'      'USD'    [-2.3004e+05]
'Funding Leg PV Accrued'     'Price'      'USD'    [          0]
'Funding Leg PV Clean'       'Price'      'USD'    [-2.3004e+05]
'Yield Risk Report'          ''           ''          { 4x30 cell}
'Messages'                   ''           ''          { 4x1  cell}
```

See Also numerix **|**

Related Examples

- "Working with Simple Numerix Trades" on page 10-2
- "Working with Advanced Numerix Trades" on page 10-5
- "Use Numerix to Price Cash Deposits" on page 10-12
- "Use Numerix for Interest-Rate Risk Assessment" on page 10-15

External Web Sites

- http://www.numerix.com/cail

Purpose   Determine option adjusted spread using Black-Derman-Toy model

Syntax   [OAS, OAD, OAC] = oasbybdt(BDTTree, Price, CouponRate, Settle, Maturity, OptSpec, Strike, ExerciseDates)
[OAS, OAD, OAC] = oasbybdt(BDTTree, Price, CouponRate, Settle, Maturity, OptSpec, Strike, ExerciseDates, Name,Value)

Description   [OAS, OAD, OAC] = oasbybdt(BDTTree, Price, CouponRate, Settle, Maturity, OptSpec, Strike, ExerciseDates) calculates option adjusted spread (OAS), duration (OAD), and convexity (OAC) using a BDT model.

[OAS, OAD, OAC] = oasbybdt(BDTTree, Price, CouponRate, Settle, Maturity, OptSpec, Strike, ExerciseDates, Name,Value) calculates option adjusted spread (OAS), duration (OAD), and convexity (OAC) using a BDT model with additional options specified by one or more Name,Value pair arguments.

Input Arguments

BDTTree

Interest-rate tree structure created by bdttree.

Price

NINST-by-1 vector of market prices of bonds with embedded options.

CouponRate

NINST-by-1 vector for decimal annual rate.

Settle

NINST-by-1 vector for settlement date.

Maturity

NINST-by-1 vector for maturity date.

OptSpec

NINST-by-1 cell array of strings for `'call'` or `'put'`.

Strike

Matrix of strike price values for supported option types:

- European option: NINST-by-1 vector of strike price values.

- Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.

  Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.

- American option: NINST-by-1 vector of strike price values for each option.

ExerciseDates

Matrix of exercise callable or puttable dates for supported option types:

- NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.

- American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option is exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option is exercised between the underlying bond Settle and the single listed exercise date.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding

value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'AmericanOpt'

NINST-by-1 vector for option flags: 0 (European/Bermuda) or 1 (American).

> **Default:** 0 (European/Bermuda)

'Basis'

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

> **Default:** 1 (in effect)

'IssueDate'

NINST-by-1 vector of bond issue date.

> **Default:** If you do not specify an IssueDate, the cash flow payment dates are determined from other inputs.

'Face'

NINST-by-1 vector for face value.

> **Default:** 100

'FirstCouponDate'

NINST-by-1 vector. Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

> **Default:** If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

'LastCouponDate'

NINST-by-1 vector. Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

Default: If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

'Period'

`NINST`-by-1 vector for coupons per year.

> **Default:** 2 per year

'Options'

Structure created with `derivset` containing derivatives pricing options.

> **Default:** None

Output Arguments

OAS

`NINST`-by-1 vector for option adjusted spread.

OAD

`NINST`-by-1 vector for option adjusted duration.

OAC

`NINST`-by-1 vector for option adjusted convexity.

Definitions

Bond with Embedded Options

A bond with embedded option allows the issuer to buy back (callable) or redeem (puttable) the bond at a predetermined price at specified future dates. Financial Instruments Toolbox software supports American, European, and Bermuda callable and puttable bonds.

The pricing for a bond with embedded options is as follows:

- Callable bond — The holder bought a bond and sold a call option to the issuer. For example, if interest rates go down by the time of the call date, the issuer is able to refinance its debt at a cheaper level and can call the bond. The price of a callable bond is:

Price callable bond = Price Option free bond − Price call option

- Puttable bond — The holder bought a bond and a put option. For example, if interest rates rise, the future value of coupon payments becomes less valuable. Therefore, the investor can sell the bond back to the issuer and then lend proceeds elsewhere at a higher rate. The price of a puttable bond is:

Price puttable bond = Price Option free bond + Price put option

Examples   Compute OAS using the Black-Derman-Toy (BDT) model with:

```
ValuationDate = 'Oct-1-2010';
Rates = [0.035; 0.042; 0.047; 0.052];
StartDates = ValuationDate;
EndDates = datemnth(ValuationDate, 12:12:48)';
Compounding = 1;
% Define RateSpec
RateSpec = intenvset('ValuationDate', ValuationDate,...
'StartDates', StartDates, 'EndDates', EndDates, ...
'Rates', Rates, 'Compounding', Compounding);

% Specify VolsSpec and TimeSpec
Sigma = 0.20;
VS = bdtvolspec(ValuationDate, EndDates, Sigma*ones(size(EndDates)));
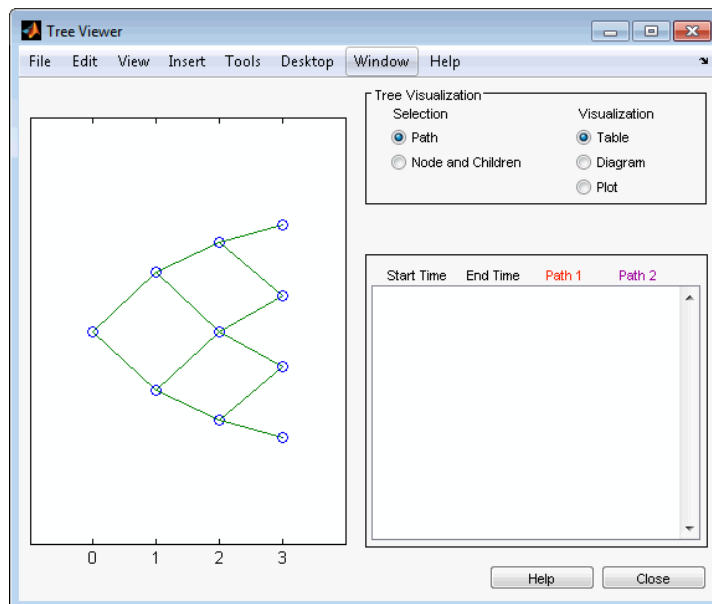TS = bdttimespec(ValuationDate, EndDates, Compounding);

% Build the BDT tree
BDTTree = bdttree(VS, RateSpec, TS);
BDTTreenew = cvtree(BDTTree);

% Instrument information
CouponRate = 0.065;
Settle = ValuationDate;
Maturity = '01-Oct-2014';
```

```
OptSpec = 'call';
Strike = 100;
ExerciseDates ='01-Oct-2011';
Period = 1;
Price = 101.58;

% Compute the OAS
OAS = oasbybdt(BDTTree, Price, CouponRate, Settle, Maturity,...
OptSpec, Strike, ExerciseDates, 'Period', Period)

OAS =

   36.5591
```

Use treeviewer to observe the tree you created:

```
treeviewer(BDTTree)
```

oasbybdt

References   Fabozzi, F., *Handbook of Fixed Income Securities*, McGraw-Hill, 7th edition, 2005.

Windas, T., *Introduction to Option-Adjusted Spread Analysis*, Bloomberg Press, 3rd edition, 2007.

See Also | bdttree | bdtprice | instoptembnd | optembndbybdt | oasbyhjm | oasbyhw | oasbybk

Purpose   Determine option adjusted spread using Black-Karasinski model

Syntax   [OAS, OAD, OAC] = oasbybk(BKTree, Price, CouponRate, Settle,
                Maturity, OptSpec, Strike, ExerciseDates)
                [OAS, OAD, OAC] = oasbybk(BKTree, Price, CouponRate, Settle,
                Maturity, OptSpec, Strike, ExerciseDates,
                Name,Value)

Description [OAS, OAD, OAC] = oasbybk(BKTree, Price, CouponRate,
                Settle,
                Maturity, OptSpec, Strike, ExerciseDates) calculates option
                adjusted spread (OAS), duration (OAD), and convexity (OAC) using a BK
                model.

                [OAS, OAD, OAC] = oasbybk(BKTree, Price, CouponRate,
                Settle,
                Maturity, OptSpec, Strike, ExerciseDates,
                Name,Value) calculates option adjusted spread (OAS), duration (OAD),
                and convexity (OAC) using a BK model with additional options specified
                by one or more Name,Value pair arguments.

Input   BKTree
Arguments
                Interest-rate tree structure created by bktree.

                Price

                NINST-by-1 vector of market prices of bonds with embedded options.

                CouponRate

                NINST-by-1 vector for decimal annual rate.

                Settle

                NINST-by-1 vector for settlement date.

                Maturity

NINST-by-1 vector for maturity date.

OptSpec

NINST-by-1 cell array of strings for `'call'` or `'put'`.

Strike

Matrix of strike price values for supported option types:

- European option: NINST-by-1 vector of strike price values.

- Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.

  Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.

- American option: NINST-by-1 vector of strike price values for each option.

ExerciseDates

Matrix of exercise callable or puttable dates for supported option types:

- NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.

- American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option is exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option is exercised between the underlying bond Settle and the single listed exercise date.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding

value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'AmericanOpt'

NINST-by-1 vector for option flags: 0 (European/Bermuda) or 1 (American).

   Default: 0 (European/Bermuda)

'Basis'

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

   Default: 0 (actual/actual)

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

> **Default:** 1 (in effect)

'IssueDate'

NINST-by-1 vector of bond issue date.

> **Default:** If you do not specify an IssueDate, the cash flow payment dates are determined from other inputs

'Face'

NINST-by-1 vector for face value.

> **Default:** 100

'FirstCouponDate'

NINST-by-1 vector. Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

> **Default:** If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

'LastCouponDate'

NINST-by-1 vector. Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

Default: If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

'Period'

`NINST`-by-1 vector for coupons per year.

Default: 2 per year

'Options'

Structure created with `derivset` containing derivatives pricing options.

Default: None

Output Arguments

OAS

`NINST`-by-1 vector for option adjusted spread.

OAD

`NINST`-by-1 vector for option adjusted duration.

OAC

`NINST`-by-1 vector for option adjusted convexity.

Definitions

Bond with Embedded Options

A bond with embedded option allows the issuer to buy back (callable) or redeem (puttable) the bond at a predetermined price at specified future dates. Financial Instruments Toolbox software supports American, European, and Bermuda callable and puttable bonds.

The pricing for a bond with embedded options is as follows:

• Callable bond — The holder bought a bond and sold a call option to the issuer. For example, if interest rates go down by the time of the call date, the issuer is able to refinance its debt at a cheaper level and can call the bond. The price of a callable bond is:

Price callable bond = Price Option free bond − Price call option

- Puttable bond — The holder bought a bond and a put option. For example, if interest rates rise, the future value of coupon payments becomes less valuable. Therefore, the investor can sell the bond back to the issuer and then lend proceeds elsewhere at a higher rate. The price of a puttable bond is:

Price puttable bond = Price Option free bond + Price put option

Examples   Compute OAS and OAD using the Black-Karasinski (BK) model with:

```
ValuationDate = 'Aug-2-2010';
Rates = [0.0355; 0.0382; 0.0427; 0.0489];
StartDates = ValuationDate;
EndDates = datemnth(ValuationDate, 12:12:48)';
Compounding = 1;

% Define RateSpec
RateSpec = intenvset('ValuationDate', ValuationDate,...
'StartDates', StartDates,'EndDates', EndDates, ...
'Rates', Rates,'Compounding', Compounding);

% Specify VolsSpec and TimeSpec
Sigma = 0.10;
Alpha = 0.01;
VS = bkvolspec(ValuationDate, EndDates, Sigma*ones(size(EndDates)),...
EndDates, Alpha*ones(size(EndDates)));
TS = bktimespec(ValuationDate, EndDates, Compounding);

% Build the BK tree
BKTree = bktree(VS, RateSpec, TS);

% Instrument information
CouponRate = 0.045;
```

```
Settle = ValuationDate;
Maturity = '02-Aug-2014';
OptSpec = 'put';
Strike = 100;
ExerciseDates ='02-Aug-2013';
Period = 1;
AmericanOpt = 1;
Price = 101;

% Compute OAS and OAD
[OAS, OAD] = oasbybk(BKTree, Price, CouponRate, Settle, Maturity,...
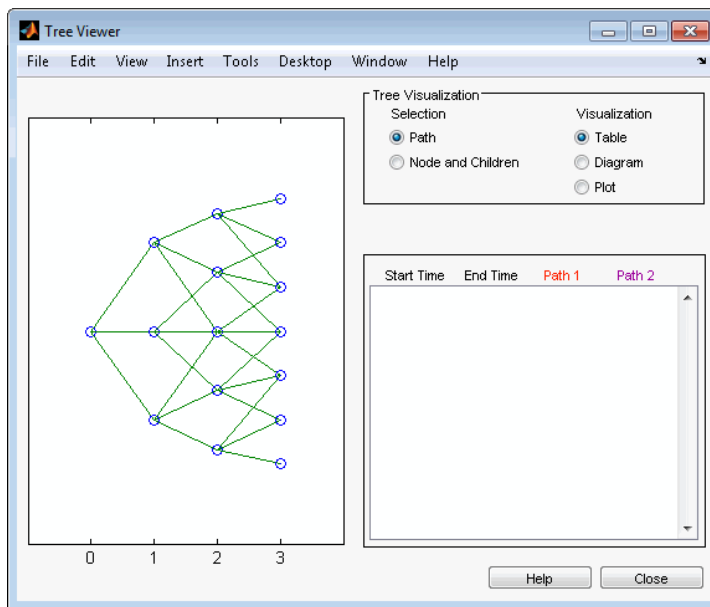OptSpec, Strike, ExerciseDates, 'Period', Period, 'AmericanOpt', AmericanOpt)

OAS =

   21.0839

OAD =

    1.7833
```

Use treeviewer to observe the tree you created:

```
treeviewer(BKTree)
```

References   Fabozzi, F., *Handbook of Fixed Income Securities*, McGraw-Hill, 7th edition, 2005.

Windas, T., *Introduction to Option-Adjusted Spread Analysis*, Bloomberg Press, 3rd edition, 2007.

See Also | bktree | bkprice | instoptembnd | optembndbybk | oasbyhjm | oasbyhw | oasbybdt

Purpose   Determine option adjusted spread using Heath-Jarrow-Morton model

Syntax   [OAS, OAD, OAC] = oasbyhjm(HJMTree, Price, CouponRate, Settle,
                Maturity, OptSpec, Strike, ExerciseDates)
                [OAS, OAD, OAC] = oasbyhjm(HJMTree, Price, CouponRate, Settle,
                Maturity, OptSpec, Strike, ExerciseDates,
                Name,Value)

Description [OAS, OAD, OAC] = oasbyhjm(HJMTree, Price, CouponRate,
                Settle,
                Maturity, OptSpec, Strike, ExerciseDates) calculates option
                adjusted spread (OAS), duration (OAD), and convexity (OAC) using an
                HJM model.

                [OAS, OAD, OAC] = oasbyhjm(HJMTree, Price, CouponRate,
                Settle,
                Maturity, OptSpec, Strike, ExerciseDates,
                Name,Value) calculates option adjusted spread (OAS), duration (OAD),
                and convexity (OAC) using an HJM model with additional options
                specified by one or more Name,Value pair arguments.

**Input
Arguments**   **HJMTree**

                Interest-rate tree structure created by hjmtree.

                Price

                NINST-by-1 vector of market prices of bonds with embedded options.

                CouponRate

                NINST-by-1 vector for decimal annual rate.

                Settle

                NINST-by-1 vector for settlement date.

                Maturity

NINST-by-1 vector for maturity date.

OptSpec

NINST-by-1 cell array of strings for `'call'` or `'put'`.

Strike

Matrix of strike price values for supported option types:

- European option: NINST-by-1 vector of strike price values.

- Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.

  Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.

- American option: NINST-by-1 vector of strike price values for each option.

ExerciseDates

Matrix of exercise callable or puttable dates for supported option types:

- NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.

- American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option is exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option is exercised between the underlying bond Settle and the single listed exercise date.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding

value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'AmericanOpt'

NINST-by-1 vector for option flags: 0 (European/Bermuda) or 1 (American).

> **Default:** 0 (European/Bermuda)

'Basis'

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

oasbyhjm

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

   Default: 1 (in effect)

'IssueDate'

NINST-by-1 vector of bond issue date.

   Default: If you do not specify an IssueDate, the cash flow payment dates are determined from other inputs.

'Face'

NINST-by-1 vector for face value.

   Default: 100

'FirstCouponDate'

NINST-by-1 vector. Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

   Default: If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

'LastCouponDate'

NINST-by-1 vector. Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

Default: If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

'Period'

`NINST`-by-1 vector for coupons per year.

Default: 2 per year

'Options'

Structure created with `derivset` containing derivatives pricing options.

Default: None

Output Arguments

OAS

`NINST`-by-1 vector for option adjusted spread.

OAD

`NINST`-by-1 vector for option adjusted duration.

OAC

`NINST`-by-1 vector for option adjusted convexity.

Definitions

Bond with Embedded Options

A bond with embedded option allows the issuer to buy back (callable) or redeem (puttable) the bond at a predetermined price at specified future dates. Financial Instruments Toolbox software supports American, European, and Bermuda callable and puttable bonds.

The pricing for a bond with embedded options is as follows:

• Callable bond — The holder bought a bond and sold a call option to the issuer. For example, if interest rates go down by the time of the call date, the issuer is able to refinance its debt at a cheaper level and can call the bond. The price of a callable bond is:

Price callable bond = Price Option free bond − Price call option

- Puttable bond — The holder bought a bond and a put option. For example, if interest rates rise, the future value of coupon payments becomes less valuable. Therefore, the investor can sell the bond back to the issuer and then lend proceeds elsewhere at a higher rate. The price of a puttable bond is:

Price puttable bond = Price Option free bond + Price put option

Examples   Compute OAS using the Heath-Jarrow-Morton (HJM) model with:

```
ValuationDate = 'Nov-1-2010';
Rates = [0.0356; 0.0427; 0.0478; 0.0529];
StartDates = ValuationDate;
EndDates = datemnth(ValuationDate, 12:12:48)';
Compounding = 1;

% Define RateSpec
RateSpec = intenvset('ValuationDate', ValuationDate,...
'StartDates', StartDates,'EndDates', EndDates, ...
'Rates', Rates,'Compounding', Compounding);

% Specify VolsSpec and TimeSpec
Sigma = 0.02;
VS = hjmvolspec('Constant', Sigma)
TS = hjmtimespec(ValuationDate, EndDates, Compounding);

% Build the HJM tree
HJMTree = hjmtree(VS, RateSpec, TS);
HJMTreenew = cvtree(HJMTree);

% Instrument information
CouponRate = 0.05;
Settle = ValuationDate;
```

```
Maturity = '01-Nov-2014';
OptSpec = 'call';
Strike = 100;
ExerciseDates ='01-Nov-2011';
Period = 1;
Price = 97.5;

% Compute the OAS
OAS = oasbyhjm(HJMTree, Price, CouponRate, Settle, Maturity, OptSpec, Strike,...
ExerciseDates, 'Period', Period)

VS =

          FinObj: 'HJMVolSpec'
    FactorModels: {'Constant'}
       FactorArgs: {{1x1 cell}}
       SigmaShift: 0
       NumFactors: 1
        NumBranch: 2
           PBranch: [0.5000 0.5000]
      Fact2Branch: [-1 1]

OAS =

    5.0601
```

Use `treeviewer` to observe the tree you created:

```
treeviewer(HJMTree)
```

oasbyhjm

References   Fabozzi, F., *Handbook of Fixed Income Securities*, McGraw-Hill, 7th edition, 2005.

Windas, T., *Introduction to Option-Adjusted Spread Analysis*, Bloomberg Press, 3rd edition, 2007.

See Also | hjmtree | hjmprice | instoptembnd | optembndbyhjm | oasbybdt | oasbyhw | oasbybk

Purpose   Determine option adjusted spread using Hull-White model

Syntax   [OAS, OAD, OAC] = oasbyhw(HWTree, Price, CouponRate, Settle,
                Maturity, OptSpec, Strike, ExerciseDates)
                [OAS, OAD, OAC] = oasbyhw(HWTree, Price, CouponRate, Settle,
                Maturity, OptSpec, Strike, ExerciseDates,
                Name,Value)

Description [OAS, OAD, OAC] = oasbyhw(HWTree, Price, CouponRate,
                Settle,
                Maturity, OptSpec, Strike, ExerciseDates) calculates option
                adjusted spread (OAS), duration (OAD), and convexity (OAC) using an
                HW model.

                [OAS, OAD, OAC] = oasbyhw(HWTree, Price, CouponRate,
                Settle,
                Maturity, OptSpec, Strike, ExerciseDates,
                Name,Value) calculates option adjusted spread (OAS), duration (OAD),
                and convexity (OAC) using an HW model with additional options
                specified by one or more Name,Value pair arguments.

Input Arguments

HWTree

Interest-rate tree structure created by hwtree.

Price

NINST-by-1 vector of market prices of bonds with embedded options.

CouponRate

NINST-by-1 vector for decimal annual rate.

Settle

NINST-by-1 vector for settlement date.

Maturity

NINST-by-1 vector for maturity date.

OptSpec

NINST-by-1 cell array of strings for `'call'` or `'put'`.

Strike

Matrix of strike price values for supported option types:

- European option: NINST-by-1 vector of strike price values.

- Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.

  Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.

- American option: NINST-by-1 vector of strike price values for each option.

ExerciseDates

Matrix of exercise callable or puttable dates for supported option types:

- NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.

- American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option is exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option is exercised between the underlying bond Settle and the single listed exercise date.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding

value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'AmericanOpt'

NINST-by-1 vector for option flags: 0 (European/Bermuda) or 1 (American).

     Default: 0 (European/Bermuda)

'Basis'

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

     Default: 0 (actual/actual)

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

>**Default:** 1 (in effect)

'Face'

NINST-by-1 vector for face value.

>**Default:** 100

'IssueDate'

NINST-by-1 vector of bond issue date.

>**Default:** If you do not specify an IssueDate, the cash flow payment dates are determined from other inputs.

'FirstCouponDate'

NINST-by-1 vector. Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

>**Default:** If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

'LastCouponDate'

NINST-by-1 vector. Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

Default: If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.

'Period'

NINST-by-1 vector for coupons per year.

Default: 2 per year

'Options'

Structure created with derivset containing derivatives pricing options.

Default: None

Output Arguments

OAS

NINST-by-1 vector for option adjusted spread.

OAD

NINST-by-1 vector for option adjusted duration.

OAC

NINST-by-1 vector for option adjusted convexity.

Definitions

Bond with Embedded Options

A bond with embedded option allows the issuer to buy back (callable) or redeem (puttable) the bond at a predetermined price at specified future dates. Financial Instruments Toolbox software supports American, European, and Bermuda callable and puttable bonds.

The pricing for a bond with embedded options is as follows:

- Callable bond — The holder bought a bond and sold a call option to the issuer. For example, if interest rates go down by the time of the call date, the issuer is able to refinance its debt at a cheaper level and can call the bond. The price of a callable bond is:

Price callable bond = Price Option free bond − Price call option

- Puttable bond — The holder bought a bond and a put option. For example, if interest rates rise, the future value of coupon payments becomes less valuable. Therefore, the investor can sell the bond back to the issuer and then lend proceeds elsewhere at a higher rate. The price of a puttable bond is:

Price puttable bond = Price Option free bond + Price put option

Examples

Compute OAS and OAD using the Hull-White (HW) model with:

```
ValuationDate = 'October-25-2010';
Rates = [0.0355; 0.0382; 0.0427; 0.0489];
StartDates = ValuationDate;
EndDates = datemnth(ValuationDate, 12:12:48)';
Compounding = 1;

% Define RateSpec
RateSpec = intenvset('ValuationDate', ValuationDate,...
'StartDates', StartDates, 'EndDates', EndDates, ...
'Rates', Rates,'Compounding', Compounding);

% Specify VolsSpec and TimeSpec
Sigma = 0.05;
Alpha = 0.01;
VS = hwvolspec(ValuationDate, EndDates, Sigma*ones(size(EndDates)),...
EndDates, Alpha*ones(size(EndDates)));
TS = hwtimespec(ValuationDate, EndDates, Compounding);

% Build the HW tree
HWTree = hwtree(VS, RateSpec, TS);

% Instrument information
CouponRate = 0.045;
```

```
Settle = ValuationDate;
Maturity = '25-October-2014';
OptSpec = 'call';
Strike = 100;
ExerciseDates = {'25-October-2010','25-October-2013'};
Period = 1;
AmericanOpt = 0;
Price = 97;

% Compute the OAS
[OAS, OAD] = oasbyhw(HWTree, Price, CouponRate, Settle, Maturity,...
OptSpec, Strike, ExerciseDates, 'Period', Period, 'AmericanOpt', AmericanOpt)

OAS =

  -12.4436

OAD =

    3.3045
```

At a 5% volatility, the OAS is -12.44 basis points. A negative OAS means that the callable bond is expensive (overvalued) on a relative value basis. OAS depends on the assumed interest rate volatility, so, if a 1% interest rate volatility is assumed (Sigma = 0.01), the OAS is 51 basis points (positive), and in this case the bond is attractive (underpriced).

Use treeviewer to observe the tree you created:

```
treeviewer(HWTree)
```

References   Fabozzi, F., *Handbook of Fixed Income Securities*, McGraw-Hill, 7th edition, 2005.

Windas, T., *Introduction to Option-Adjusted Spread Analysis*, Bloomberg Press, 3rd edition, 2007.

See Also | hwtree | hwprice | instoptembnd | optembndbyhw | oasbybdt | oasbyhjm | oasbybk

Purpose   Price bond option from Black-Derman-Toy interest-rate tree

Syntax   [Price, PriceTree] = optbndbybdt(BDTTree, OptSpec, Strike,
ExerciseDates, AmericanOpt, CouponRate, Settle, Maturity,
Period, Basis, EndMonthRule, IssueDate, FirstCouponDate,
LastCouponDate, StartDate, Face, Options)

Arguments

BDTTree	Forward-rate tree structure created by bdttree.
OptSpec	Number of instruments (NINST)-by-1 cell array of string values 'Call' or 'Put'.
Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	For an American option:
	NINST-by-1 vector of strike price values for each option.

optbndbybdt

ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.
AmericanOpt	NINST-by-1 vector of flags: 0 (European/Bermuda) or 1 (American).
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 1, 2, 3, 4, 6, and 12. Default = 2.

Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when a bond was issued.

FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure. If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.
StartDate	(Optional) Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify StartDate, the effective start date is the Settle date.
Face	(Optional) Face or par value. Face is a NINST-by-1 vector. Default = 100.
Options	(Optional) Derivatives pricing options structure created with derivset.

The Settle date for every bond is set to the ValuationDate of the BDT tree. The bond argument Settle is ignored.

Description   [Price, PriceTree] = optbndbybdt(BDTTree, OptSpec, Strike, ExerciseDates, AmericanOpt, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face, Options) computes the price of a bond option from a BDT interest-rate tree.

Price is an NINST-by-1 matrix of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples   **Example 1**. Using the BDT interest-rate tree in the deriv.mat file, price a European call option on a 10% bond with a strike of 95. The exercise date for the option is Jan. 01, 2002. The settle date for the bond is Jan. 01, 2000, and the maturity date is Jan. 01, 2003.

Load the file deriv.mat, which provides BDTTree. The BDTTree structure contains the time and forward-rate information needed to price the bond.

```
load deriv.mat;
```

Use optbondbybdt to compute the price of the option.

```
Price = optbndbybdt(BDTTree,'Call',95,'01-Jan-2002',...
0,0.10,'01-Jan-2000','01-Jan-2003',1)

Price =

    1.7657
```

Example 2. Now use optbndbybdt to compute the price of a put option on the same bond.

```
Price = optbndbybdt(BDTTree,'Put',95,'01-Jan-2002',...
0,0.10,'01-Jan-2000','01-Jan-2003',1)
```

```
Price =

    0.5740
```

See Also   bdtprice | bdttree | instoptbnd

Purpose   Price bond option from Black-Karasinski interest-rate tree

Syntax   [Price, PriceTree] = optbndbybk(BKTree, OptSpec, Strike,
                ExerciseDates, AmericanOpt, CouponRate, Settle, Maturity,
                Period, Basis, EndMonthRule, IssueDate, FirstCouponDate,
                LastCouponDate, StartDate, Face, Options)

Arguments

BKTree	Forward-rate tree structure created by bktree.
OptSpec	Number of instruments (NINST)-by-1 cell array of string values 'Call' or 'Put'.
Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST-by-number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	For an American option:
	NINST-by-1 vector of strike price values for each option.

ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.
AmericanOpt	NINST-by-1 vector of flags: 0 (European/Bermuda) or 1 (American).
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 1, 2, 3, 4, 6, and 12. Default = 2.

Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when a bond was issued.

optbndbybk

FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.
StartDate	(Optional) Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.
Face	(Optional) Face or par value. `Face` is a NINST-by-1 vector. Default = 100.
Options	(Optional) Derivatives pricing options structure created with `derivset`.

The Settle date for every bond is set to the ValuationDate of the BK tree. The bond argument Settle is ignored.

Description   [Price, PriceTree] = optbndbybk(BKTree, OptSpec, Strike, ExerciseDates, AmericanOpt, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face, Options) computes the price of a bond option from a Black-Karasinski interest rate tree.

Price is an NINST-by-1 matrix of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples   **Example 1**. Using the BK interest rate tree in the deriv.mat file, price a European call option on a 4% bond with a strike of 96. The exercise date for the option is Jan. 01, 2006. The settle date for the bond is Jan. 01, 2005, and the maturity date is Jan. 01, 2009.

Load the file deriv.mat, which provides BKTree. The BKTree structure contains the time and forward-rate information needed to price the bond.

```
load deriv.mat;
```

Use optbondbybk to compute the price of the option.

```
Price = optbndbybk(BKTree,'Call',96,'01-Jan-2006',...
0,0.04,'01-Jan-2005','01-Jan-2009')

Warning: OptBonds are valued at Tree ValuationDate rather than Settle
> In optbndbytrintree at 43
  In optbndbybk at 88
Warning: Not all cash flows are aligned with the tree. Result will be
approximated.
> In optbndbytrintree at 151
  In optbndbybk at 88

Price =
```

```
     0.1512
```

Example 2. Now use `optbndbybk` to compute the price of a put option
on the same bond.

```
Price = optbndbybk(BKTree,'Put',96,'01-Jan-2006',...
0,0.04,'01-Jan-2005','01-Jan-2009')

Warning: OptBonds are valued at Tree ValuationDate rather than Settle
> In optbndbytrintree at 43
  In optbndbybk at 88
Warning: Not all cash flows are aligned with the tree. Result will be
approximated.
> In optbndbytrintree at 151
  In optbndbybk at 88

Price =

    0.0272
```

See Also   bkprice | bktree | instoptbnd

Purpose   Price bond option from Heath-Jarrow-Morton interest-rate tree

Syntax   [Price, PriceTree] = optbndbyhjm(HJMTree, OptSpec, Strike,
                 ExerciseDates, AmericanOpt, CouponRate, Settle, Maturity,
                 Period, Basis, EndMonthRule, IssueDate, FirstCouponDate,
                 LastCouponDate, StartDate, Face, Options)

Arguments

HJMTree	Forward-rate tree structure created by hjmtree.
OptSpec	Number of instruments (NINST)-by-1 cell array of string values 'Call' or 'Put'.
Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	For an American option:
	NINST-by-1 vector of strike price values for each option.

ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.
AmericanOpt	NINST-by-1 vector of flags: 0 (European/Bermuda) or 1 (American).
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 1, 2, 3, 4, 6, and 12. Default = 2.

Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when a bond was issued.

FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure. If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.
StartDate	(Optional) Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify StartDate, the effective start date is the Settle date.
Face	(Optional) Face or par value. Face is a NINST-by-1 vector. Default = 100.
Options	(Optional) Derivatives pricing options structure created with derivset.

The Settle date for every bond is set to the ValuationDate of the HJM tree. The bond argument Settle is ignored.

Description   [Price, PriceTree] = optbndbyhjm(HJMTree, OptSpec, Strike,
                     ExerciseDates, AmericanOpt, CouponRate, Settle, Maturity,
                     Period, Basis, EndMonthRule, IssueDate, FirstCouponDate,
                     LastCouponDate, StartDate, Face, Options) computes the price of
                     a bond option from an HJM forward-rate tree.

                     Price is an NINST-by-1 matrix of expected prices at time 0.

                     PriceTree is a tree structure with a vector of instrument prices at
                     each node.

Examples   Using the HJM forward-rate tree in the deriv.mat file, price a
                     European call option on a 4% bond with a strike of 96. The exercise date
                     for the option is Jan. 01, 2003. The settle date for the bond is Jan. 01,
                     2000, and the maturity date is Jan. 01, 2004.

                     Load the file deriv.mat, which provides HJMTree. The HJMTree
                     structure contains the time and forward-rate information needed to
                     price the bond.

                     ```
                     load deriv.mat;
                     ```

                     Use optbondbyhjm to compute the price of the option.

                     ```
                     Price = optbndbyhjm(HJMTree,'Call',96,'01-Jan-2003',...
                     0,0.04,'01-Jan-2000','01-Jan-2004')
                     ```

                     ```
                     Warning: Not all cash flows are aligned with the tree. Result will
                     be approximated.

                     Price =

                         2.2410
                     ```

See Also   hjmprice | hjmtree | instoptbnd

optbndbyhw

Purpose	Price bond option from Hull-White interest-rate tree
Syntax	[Price, PriceTree] = optbndbyhw(HWTree, OptSpec, Strike, ExerciseDates, AmericanOpt, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face, Options)

Arguments

HWTree	Forward-rate tree structure created by hwtree.
OptSpec	Number of instruments (NINST)-by-1 cell array of string values 'Call' or 'Put'.
Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	For an American option:
	NINST-by-1 vector of strike price values for each option.

ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.
AmericanOpt	NINST-by-1 vector of flags: 0 (European/Bermuda) or 1 (American).
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 1, 2, 3, 4, 6, and 12. Default = 2.

Basis	(Optional) Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when a bond was issued.

FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure. If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.
StartDate	(Optional) Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify StartDate, the effective start date is the Settle date.
Face	(Optional) Face or par value. Face is a NINST-by-1 vector. Default = 100.
Options	(Optional) Derivatives pricing options structure created with derivset.

optbndbyhw

The Settle date for every bond is set to the ValuationDate of the HW tree. The bond argument Settle is ignored.

Description   [Price, PriceTree] = optbndbyhw(HWTree, OptSpec, Strike, ExerciseDates, AmericanOpt, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face, Options) computes the price of a bond option from a Hull-White interest rate tree.

Price is an NINST-by-1 matrix of expected prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples   **Example 1**. Using the HW interest rate tree in the deriv.mat file, price a European call option on a 4% bond with a strike of 96. The exercise date for the option is Jan. 01, 2006. The settle date for the bond is Jan. 01, 2005, and the maturity date is Jan. 01, 2009.

Load the file deriv.mat, which provides HWTree. The HWTree structure contains the time and forward-rate information needed to price the bond.

```
load deriv.mat;
```

Use optbondbyhw to compute the price of the option.

```
Price = optbndbyhw(HWTree,'Call',96,'01-Jan-2006',...
0,0.04,'01-Jan-2005','01-Jan-2009')

Warning: OptBonds are valued at Tree ValuationDate rather than Settle
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

    1.1556
```

Example 2. Now use optbondbyhw to compute the price of a put option on the same bond.

```
Price = optbndbyhw(HWTree,'Put',96,'01-Jan-2006',...
0,0.04,'01-Jan-2005','01-Jan-2009')

Warning: OptBonds are valued at Tree ValuationDate rather than Settle
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

Price =

    1.0150
```

See Also   hwprice | hwtree | instoptbnd

optembndbybdt

Purpose	Price bonds with embedded options by Black-Derman-Toy interest-rate tree
Syntax	[Price, PriceTree] = optembndbybdt(BDTTree, CouponRate, Settle, Maturity, OptSpec, Strike, ExerciseDates, 'Name1', Value1, 'Name2', Value2, ...)

Arguments

BDTTree	Interest-rate tree structure created by bdttree.
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector or NINST-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	NINST-by-1 matrix for the settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	NINST-by-1 matrix for the maturity date. A vector of serial date numbers or date strings.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.

Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	For an American option:
	NINST-by-1 vector of strike price values for each option.
ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.
'Name1', 'Value1' 'Name2', 'Value2'...	(Optional) The name/value pairs are a variable length list of parameters. All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may

be specified in any order; names are case insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are as follows:

- `AmericanOpt` is a `NINST-by-1` matrix for flags options: `0` (European/Bermuda) or `1` (American). Default is `0`.

- `Period` is a `NINST-by-1` matrix for coupons per year. The default value is `2`.

- `Basis` is a day-count basis of the instrument. `Basis` is a vector of integers with the following possible values:

  - 0 = actual/actual (default)

  - 1 = 30/360 (SIA)

  - 2 = actual/360

  - 3 = actual/365

  - 4 = 30/360 (BMA)

  - 5 = 30/360 (ISDA)

  - 6 = 30/360 (European)

  - 7 = actual/365 (Japanese)

  - 8 = actual/actual (ICMA)

  - 9 = actual/360 (ICMA)

  - 10 = actual/365 (ICMA)

  - 11 = 30/360E (ICMA)

  - 12 = actual/actual (ISDA)

  - 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

- `EndMonthRule` is a `NINST`-by-`1` matrix for the end-of-month rule. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. When the value is `0` the end-of-month rule is ignored; this means that a bond's coupon payment date is always the same numerical day of the month. Use `1` to set the rule on; this is the default value and means that a bond's coupon payment date is always the last actual day of the month.

- `IssueDate` is a `NINST`-by-`1` matrix for the bond issue date.

- `FirstCouponDate` is a `NINST`-by-`1` matrix for a date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

- `LastCouponDate` is a `NINST`-by-`1` matrix for a last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a `LastCouponDate`, the cash flow

payment dates are determined from other inputs.

- `StartDate` is a `NINST`-by-1 matrix for date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

- Face value. `Face` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid. Default is 100.

- `Options` is a derivatives pricing options structure created with `derivset`.

Note The `Settle` date for every bond with an embedded option is set to the `ValuationDate` of the BDT tree; the bond's argument for `Settle` date is ignored.

Description [Price, PriceTree] = optembndbybdt(BDTTree, CouponRate, Settle, Maturity, OptSpec, Strike, ExerciseDates, 'Name1', Value1, 'Name2', Value2, ...) prices bonds with embedded options using a BDT interest-rate tree.

Price is a number of instruments (NINST)-by-1 matrix of expected prices at time 0.

PriceTree is a MATLAB structure of trees containing vectors of instrument prices and observation times for each node. Within PriceTree

- PriceTree.PTree contains the clean prices.

- PriceTree.tObs contains the observation times.

optembndbybdt computes prices of vanilla bonds with embedded options, stepped coupon bonds with embedded options, and bonds with sinking fund option provisions.

Definitions

Vanilla Bond with Embedded Option

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment. A vanilla bond with an embedded option is where an option contract has an underlying asset of a vanilla bond.

Stepped Coupon Bond with Callable and Puttable Features

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond. Stepped coupon bonds can have options features (call and puts).

Sinking Fund Bond with Embedded Option

A sinking fund bond is a coupon bond with a sinking fund provision. This provision obligates the issuer to amortize portions of the principal prior to maturity, affecting bond prices since the time of the principal repayment changes. This means that investors receive the coupon and a portion of the principal paid back over time. These types of bonds reduce credit risk, since it lowers the probability of investors not receiving their principal payment at maturity.

The bond may have a sinking fund option provision allowing the issuer to retire the sinking fund obligation either by purchasing the bonds to

be redeemed from the market or by calling the bond via a sinking fund call, whichever is cheaper. If interest rates are high, then the issuer will buy back the requirement amount of bonds from the market since bonds will be cheap, but if interest rates are low (bond prices are high), then most likely the issuer will be buying the bonds at the call price. Unlike a call feature, however, if a bond has a sinking fund option provision, it is an obligation, not an option, for the issuer to buy back the increments of the issue as stated. Because of this, a sinking fund bond trades at a lower price than a non-sinking fund bond.

Examples   To price a callable bond using the BDT model, create a BDTTree with the following data:

```
ZeroRates = [ 0.035;0.04;0.045];
Compounding = 1;
StartDates = ['jan-1-2007';'jan-1-2008';'jan-1-2009'];
EndDates   = ['jan-1-2008';'jan-1-2009';'jan-1-2010'];
ValuationDate = 'jan-1-2007';
```

Create a `RateSpec`:

```
RateSpec = intenvset('Rates', ZeroRates, 'StartDates', ValuationDate, 'EndDates', ...
EndDates, 'Compounding', Compounding, 'ValuationDate', ValuationDate);
```

Specify a `VolSpec`:

```
Volatility = 0.10 * ones (3,1);
VolSpec = bdtvolspec(ValuationDate, EndDates, Volatility);
```

Specify a `TimeSpec`:

```
TimeSpec = bdttimespec(ValuationDate, EndDates, Compounding);
```

Build the BDTTree:

```
BDTTree = bdttree(VolSpec, RateSpec, TimeSpec);
```

To compute the price of an American callable bond that pays a 5.25% annual coupon, matures in Jan-1-2010, and is callable on Jan-1-2008 and 01-Jan-2010:

```
BondSettlement = 'jan-1-2007';
BondMaturity   = 'jan-1-2010';
CouponRate = 0.0525;
Period = 1;
OptSpec = 'call';
Strike = [100];
ExerciseDates = {'jan-1-2008' '01-Jan-2010'};
AmericanOpt = 1;

PriceCallBond = optembndbybdt(BDTTree, CouponRate, BondSettlement, BondMaturity,...
OptSpec, Strike, ExerciseDates, 'Period', 1,'AmericanOp', 1)

PriceCallBond =

101.4750
```

Price the following single stepped callable bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

% Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Instrument
Settle = '01-Jan-2010';
```

```
Maturity = {'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
OptSpec='call';
Strike=100;
ExerciseDates='01-Jan-2012';  %Callable in two years

% Build the tree
% Assume the volatility to be 10%
Sigma = 0.1;
BDTTimeSpec = bdttimespec(ValuationDate, EndDates, Compounding);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
BDTT = bdttree(BDTVolSpec, RS, BDTTimeSpec);

% The first row corresponds to the price of the callable bond with maturity
% of three years. The second row corresponds to the price of the callable bond
% with maturity of four years.
PBDT=  optembndbybdt(BDTT, CouponRate, Settle, Maturity ,OptSpec, Strike,...
ExerciseDates, 'Period', 1)


PBDT =

   100.0945
   100.0297
```

A corporation issues a three year bond with a sinking fund obligation requiring the company to sink 1/3 of face value after the first year and 1/3 after the second year. The company has the option to buy the bonds in the market or call them at $98. The following data describes the details needed for pricing the sinking fund bond:

```
% The data for the interest rate term structure is as follows:
Rates = [0.1;0.1;0.1;0.1];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
```

```
Compounding = 1;

% Create RateSpec
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding);

% Build the BDT tree
% Assume the volatility to be 10%
Sigma = 0.1;
BDTTimeSpec = bdttimespec(ValuationDate, EndDates);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
BDTT = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);

% Instrument
% The bond has a coupon rate of 9%, a period of one year and matures in
% 1-Jan-2014. Face decreases 1/3 after the first year and 1/3 after the
% second year.
CouponRate = 0.09;
Settle = 'Jan-1-2011';
Maturity =  'Jan-1-2014';
Period = 1;
Face = { ...
            {'Jan-1-2012'  100; ...
             'Jan-1-2013'   66.6666; ...
             'Jan-1-2014'   33.3333};
        };

% Option provision
OptSpec = 'call';
Strike = [98 98];
ExerciseDates ={'Jan-1-2012', 'Jan-1-2013'};

% Price of non-sinking fund bond.
PNSF = bondbybdt(BDTT, CouponRate, Settle, Maturity, Period)

PNSF =
```

optembndbybdt

```
        97.5131

% Price of the bond with the option sinking provision.
PriceSF = optembndbybdt(BDTT, CouponRate, Settle, Maturity,...
OptSpec, Strike, ExerciseDates,'Period', Period, 'Face', Face)

PriceSF =

        96.8364
```

See Also   bdtprice | bdttree | cfamounts | instoptembnd

optembndbybk

Purpose   Price bonds with embedded options by Black-Karasinski interest-rate tree

Syntax
```
[Price, PriceTree] = optembndbybk(BKTree, CouponRate,
Settle, Maturity, OptSpec, Strike, Strike, ExerciseDates,
'Name1', Value1, 'Name2', Value2, ...)
```

Arguments

BKTree	Interest-rate tree structure created by bktree.
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector or NINST-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	NINST-by-1 matrix for the settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	NINST-by-1 matrix for the maturity date. A vector of serial date numbers or date strings.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.

Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	For an American option:
	NINST-by-1 vector of strike price values for each option.
ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.
'Name1', Value1, 'Name2', Value2 ...	(Optional) The name/value pairs are a variable length list of parameters. All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may

be specified in any order; names are case insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are as follows:

- AmericanOpt is a NINST-by-1 matrix for flags options: 0 (European/Bermuda) or 1 (American). Default is 0.

- Period is a NINST-by-1 matrix for coupons per year. Default is 2.

- Basis is a day-count basis of the instrument. Basis is a vector of integers with the following supported values:
  - 0 = actual/actual (default)
  - 1 = 30/360 (SIA)
  - 2 = actual/360
  - 3 = actual/365
  - 4 = 30/360 (BMA)
  - 5 = 30/360 (ISDA)
  - 6 = 30/360 (European)
  - 7 = actual/365 (Japanese)
  - 8 = actual/actual (ICMA)
  - 9 = actual/360 (ICMA)
  - 10 = actual/365 (ICMA)
  - 11 = 30/360E (ICMA)
  - 12 = actual/actual (ISDA)
  - 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

- `EndMonthRule` is a `NINST`-by-`1` matrix for
  the end-of-month rule. This rule applies
  only when `Maturity` is an end-of-month
  date for a month having 30 or fewer days.
  When the value is `0`, the end-of-month
  rule is ignored, meaning that a bond's
  coupon payment date is always the same
  numerical day of the month. When the
  value is `1`, the end-of-month rule is set on
  (default), meaning that a bond's coupon
  payment date is always the last actual day
  of the month.

- `IssueDate` is a `NINST`-by-`1` matrix for the
  bond issue date.

- `FirstCouponDate` is a `NINST`-by-`1` matrix
  for a date when a bond makes its first
  coupon payment; used when bond has
  an irregular first coupon period. When
  `FirstCouponDate` and `LastCouponDate`
  are both specified, `FirstCouponDate` takes
  precedence in determining the coupon
  payment structure. If you do not specify a
  `FirstCouponDate`, the cash flow payment
  dates are determined from other inputs.

- `LastCouponDate` is a `NINST`-by-`1` matrix
  for a last coupon date of a bond before
  the maturity date; used when bond has
  an irregular last coupon period. In the
  absence of a specified `FirstCouponDate`, a
  specified `LastCouponDate` determines the
  coupon structure of the bond. The coupon
  structure of a bond is truncated at the
  `LastCouponDate`, regardless of where it
  falls, and is followed only by the bond's
  maturity cash flow date. If you do not

specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

- `StartDate` is a NINST-by-1 matrix for date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

- Face value. `Face` is a NINST-by-1 vector or NINST-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid. Default is 100.

- `Options` is a derivatives pricing options structure created with `derivset`.

Note The `Settle` date for every bond with embedded option is set to the `ValuationDate` of the BKTree; the bond's argument for `Settle` date is ignored.

Description [Price, PriceTree] = optembndbybk(BKTree, CouponRate, Settle, Maturity, OptSpec, Strike, Strike, ExerciseDates, 'Name1', Value1, 'Name2', Value2, ...) prices bonds with embedded options by a BK interest-rate tree.

Price is a number of instruments (NINST)-by-1 matrix of expected prices at time 0.

PriceTree is a structure of trees containing vectors of instrument prices and observation times for each node. Within PriceTree:

- PriceTree.PTree contains the clean prices.

- PriceTree.tObs contains the observation times.

optembndbybk computes prices of vanilla bonds with embedded options, stepped coupon bonds with embedded options, and bonds with sinking fund option provisions.

Definitions

Vanilla Bond with Embedded Option

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment. A vanilla bond with an embedded option is where an option contract has an underlying asset of a vanilla bond.

Stepped Coupon Bond with Callable and Puttable Features

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond. Stepped coupon bonds can have options features (call and puts).

Sinking Fund Bond with Embedded Option

A sinking fund bond is a coupon bond with a sinking fund provision. This provision obligates the issuer to amortize portions of the principal prior to maturity, affecting bond prices since the time of the principal repayment changes. This means that investors receive the coupon and a portion of the principal paid back over time. These types of bonds reduce credit risk, since it lowers the probability of investors not receiving their principal payment at maturity.

The bond may have a sinking fund option provision allowing the issuer to retire the sinking fund obligation either by purchasing the bonds to be redeemed from the market or by calling the bond via a sinking fund call, whichever is cheaper. If interest rates are high, then the issuer will buy back the requirement amount of bonds from the market since bonds will be cheap, but if interest rates are low (bond prices are high), then most likely the issuer will be buying the bonds at the call price. Unlike a call feature, however, if a bond has a sinking fund option provision, it is an obligation, not an option, for the issuer to buy back the increments of the issue as stated. Because of this, a sinking fund bond trades at a lower price than a non-sinking fund bond.

Examples   Create a `BKTree` with the following data:

```
ZeroRates = [ 0.035;0.04;0.045];
Compounding = 1;
StartDates = ['jan-1-2007';'jan-1-2008';'jan-1-2009'];
EndDates   = ['jan-1-2008';'jan-1-2009';'jan-1-2010'];
ValuationDate = 'jan-1-2007';
```

Create a `RateSpec`:

```
RateSpec = intenvset('Rates', ZeroRates, 'StartDates', ValuationDate, 'EndDates', ...
EndDates, 'Compounding', Compounding, 'ValuationDate', ValuationDate);
```

Specify a `TimeSpec`:

```
BKTimeSpec = bktimespec(ValuationDate, EndDates, Compounding);
```

Specify a `VolSpec`:

```
VolDates = ['jan-1-2008';'jan-1-2009';'jan-1-2010'];
VolCurve = 0.01;
AlphaDates = 'jan-1-2010';
AlphaCurve = 0.1;
BKVolSpec = bkvolspec(ValuationDate, VolDates, VolCurve, AlphaDates, AlphaCurve);
```

Build a `BKTree`:

```
BKTree = bktree(BKVolSpec, RateSpec, BKTimeSpec);
```

To compute the price of an American puttable bond that pays an annual coupon of 5.25% , matures on January 1, 2010, and is callable on January 1, 2008 and January 1, 2010:

```
BondSettlement = 'jan-1-2007';
BondMaturity   = 'jan-1-2010';
CouponRate = 0.0525;
Period = 1;
OptSpec = 'put';
Strike = [100];
ExerciseDates = {'jan-1-2008' '01-Jan-2010'};
AmericanOpt = 1;

PricePutBondBK = optembndbybk(BKTree, CouponRate, BondSettlement, BondMaturity,...
OptSpec, Strike, ExerciseDates,'Period', 1, 'AmericanOpt', 1)

PricePutBondBK =

102.3820
```

Price the following single stepped callable bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

% Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);
```

```
% Instrument
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
OptSpec='call';
Strike=100;
ExerciseDates='01-Jan-2012';  %Callable in two years

% Build the tree with the following data
VolDates = ['1-Jan-2011'; '1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014'];
VolCurve = 0.01;
AlphaDates = '01-01-2014';
AlphaCurve = 0.1;


BKVolSpec = bkvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RS.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RS, BKTimeSpec);

% The first row corresponds to the price of the callable bond with maturity
% of three years. The second row corresponds to the price of the callable bond
% with maturity of four years.
PBK= optembndbybk(BKT, CouponRate,  Settle, Maturity ,OptSpec, Strike,...
ExerciseDates, 'Period', 1)

PBK =

  100.0945
  100.0945
```

See Also   bkprice | cfamounts | bktree | instoptembnd

optembndbyhjm

| **Purpose** | Price bonds with embedded options by Heath-Jarrow-Morton interest-rate tree |

Syntax

```
[Price, PriceTree] = optembndbyhjm(HJMTree, CouponRate,
Settle, Maturity, OptSpec, Strike, ExerciseDates,
'Name1', Value1, 'Name2', Value2, ...)
```

Arguments

HJMTree	Interest-rate tree structure created by hjmtree.
CouponRate	Decimal annual rate. CouponRate is a NINST-by-1 vector or NINST-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a NumDates-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	NINST-by-1 matrix for the settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	NINST-by-1 matrix for the maturity date. A vector of serial date numbers or date strings.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.

Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	For an American option:
	NINST-by-1 vector of strike price values for each option.
ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.
'Name1', Value1, 'Name2', Value2 ...	(Optional) The name/value pairs are a variable length list of parameters. All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may

be specified in any order; names are case insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are as follows:

- `AmericanOpt` is a `NINST`-by-`1` matrix for flags options: `0` (European/Bermuda) or `1` (American). Default is `0`.

- `Period` is a `NINST`-by-`1` matrix for coupons per year. Default is `2`.

- `Basis` is a day-count basis of the instrument. `Basis` is a vector of integers with the following supported values:

  - 0 = actual/actual (default)

  - 1 = 30/360 (SIA)

  - 2 = actual/360

  - 3 = actual/365

  - 4 = 30/360 (BMA)

  - 5 = 30/360 (ISDA)

  - 6 = 30/360 (European)

  - 7 = actual/365 (Japanese)

  - 8 = actual/actual (ICMA)

  - 9 = actual/360 (ICMA)

  - 10 = actual/365 (ICMA)

  - 11 = 30/360E (ICMA)

  - 12 = actual/actual (ISDA)

  - 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

- `EndMonthRule` is a `NINST`-by-`1` matrix for the end-of-month rule. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. When the value is `0`, the end-of-month rule is ignored, meaning that a bond's coupon payment date is always the same numerical day of the month. When the value is `1`, the end-of-month rule is set on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

- `IssueDate` is a `NINST`-by-`1` matrix for the bond issue date.

- `FirstCouponDate` is a `NINST`-by-`1` matrix for a date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

- `LastCouponDate` is a `NINST`-by-`1` matrix for a last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not

optembndbyhjm

specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

- `StartDate` is a `NINST`-by-1 matrix for date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

- Face value. `Face` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid. Default is 100.

- `Options` is a derivatives pricing options structure created with `derivset`.

Note The `Settle` date for every bond with embedded option is set to the `ValuationDate` of the HJM tree; the bond's argument for `Settle` date is ignored.

Description   [Price, PriceTree] = optembndbyhjm(HJMTree, CouponRate, Settle, Maturity, OptSpec, Strike, ExerciseDates, 'Name1', Value1, 'Name2', Value2, ...) prices bonds with embedded options by an HJM interest-rate tree.

`Price` is a number of instruments (NINST)-by-1 matrix of expected prices at time 0.

`PriceTree` is a structure of trees containing vectors of instrument prices and observation times for each node. Within `PriceTree`:

- `PriceTree.PBush` contains the clean prices.

- `PriceTree.tObs` contains the observation times.

`optembndbyhjm` computes prices of vanilla bonds with embedded options, stepped coupon bonds with embedded options, and bonds with sinking fund option provisions.

Definitions

Vanilla Bond with Embedded Option

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment. A vanilla bond with an embedded option is where an option contract has an underlying asset of a vanilla bond.

Stepped Coupon Bond with Callable and Puttable Features

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond. Stepped coupon bonds can have options features (call and puts).

Sinking Fund Bond with Embedded Option

A sinking fund bond is a coupon bond with a sinking fund provision. This provision obligates the issuer to amortize portions of the principal prior to maturity, affecting bond prices since the time of the principal repayment changes. This means that investors receive the coupon and a portion of the principal paid back over time. These types of bonds reduce credit risk, since it lowers the probability of investors not receiving their principal payment at maturity.

The bond may have a sinking fund option provision allowing the issuer to retire the sinking fund obligation either by purchasing the bonds to be redeemed from the market or by calling the bond via a sinking fund call, whichever is cheaper. If interest rates are high, then the issuer will buy back the requirement amount of bonds from the market since bonds will be cheap, but if interest rates are low (bond prices are high), then most likely the issuer will be buying the bonds at the call price. Unlike a call feature, however, if a bond has a sinking fund option provision, it is an obligation, not an option, for the issuer to buy back the increments of the issue as stated. Because of this, a sinking fund bond trades at a lower price than a non-sinking fund bond.

Examples   Create an HJMTree with the following data:

```
Rates = [0.05;0.06;0.07];
Compounding = 1;
StartDates = ['jan-1-2007';'jan-1-2008';'jan-1-2009'];
EndDates   = ['jan-1-2008';'jan-1-2009';'jan-1-2010'];
ValuationDate = 'jan-1-2007';
```

Create a RateSpec:

```
RateSpec = intenvset('Rates', Rates, 'StartDates', ValuationDate, 'EndDates', ...
EndDates, 'Compounding', Compounding, 'ValuationDate', ValuationDate);
```

Specify a VolSpec:

```
VolSpec = hjmvolspec('Constant', 0.01);
```

Specify a TimeSpec:

```
TimeSpec = hjmtimespec(ValuationDate, EndDates, Compounding);
```

Build an HJMTree:

```
HJMTree = hjmtree(VolSpec, RateSpec, TimeSpec);
```

To compute the price of an American callable bond that pays a 6% annual coupon and matures and is callable on January 1, 2010:

```
BondSettlement = 'jan-1-2007';
BondMaturity   = 'jan-1-2010';
CouponRate = 0.06;
Period = 1;
OptSpec = 'call';
Strike = [98];
ExerciseDates = '01-Jan-2010';
AmericanOpt = 1;

[PriceCallBond,PT] = optembndbyhjm(HJMTree, CouponRate, BondSettlement, BondMaturity,...
OptSpec, Strike, ExerciseDates, 'Period', 1,'AmericanOp',1)


PriceCallBond =

95.9492

PT =

FinObj: 'HJMPriceTree'
     tObs: [0 1 2 3]
    PBush: {[95.9492]  [1x1x2 double]  [1x2x2 double]  [98 98 98 98]}
```

Price the following single stepped callable bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

% Create RateSpec
```

```
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Instrument
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
OptSpec='call';
Strike=100;
ExerciseDates='01-Jan-2012';  %Callable in two years

% Build the tree with the following data
Volatility = [.2; .19; .18; .17];
CurveTerm = [ 1;  2;   3;    4];
HJMTimeSpec = hjmtimespec(ValuationDate, EndDates);
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec, RS, HJMTimeSpec);

% The first row corresponds to the price of the callable bond with maturity
% of three years. The second row corresponds to the price of the callable
% bond with maturity of four years.
PHJM= optembndbyhjm(HJMT, CouponRate, Settle, Maturity ,OptSpec, Strike,...
ExerciseDates, 'Period', 1)

PHJM =

  100.0484
   99.8009
```

See Also   hjmprice | cfamounts | hjmtree | instoptembnd

Purpose   Price bonds with embedded options by Hull-White interest-rate tree

Syntax   [Price, PriceTree] = optembndbyhw(HWTree, CouponRate,
                Settle, Maturity, OptSpec, Strike, ExerciseDates,
                'Name1', Value1, 'Name2', Value2, ...)

Arguments

HWTree	Interest-rate tree structure created by `hwtree`.
CouponRate	Decimal annual rate. `CouponRate` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of decimal annual rates, or decimal annual rate schedules. For the latter case of a variable coupon schedule, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated rate. The date indicates the last day that the coupon rate is valid.
Settle	`NINST`-by-1 matrix for the settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than `Maturity`.
Maturity	`NINST`-by-1 matrix for the maturity date. A vector of serial date numbers or date strings.
OptSpec	`NINST`-by-1 cell array of strings `'call'` or `'put'`.

Strike	European option: `NINST`-by-`1` vector of strike price values.
	Bermuda option: `NINST` by number of strikes (`NSTRIKES`) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than `NSTRIKES` exercise opportunities, the end of the row is padded with `NaNs`.
	For an American option:
	`NINST`-by-`1` vector of strike price values for each option.
ExerciseDates	`NINST`-by-`1` (European option) or `NINST`-by-`NSTRIKES` (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	`NINST`-by-`2` vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-`NaN` date is listed, or if `ExerciseDates` is `NINST`-by-`1`, the option can be exercised between the underlying bond `Settle` and the single listed exercise date.
`'Name1'`, `Value1` `'Name2'`, `Value2` ...	(Optional) The name/value pairs are a variable length list of parameters. All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may

be specified in any order; names are case insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are as follows:

- AmericanOpt is a NINST-by-1 matrix for flags options: 0 (European/Bermuda) or 1 (American). Default is 0.

- Period is a NINST-by-1 matrix for coupons per year. Default is 2.

- Basis is a day-count basis of the instrument. Basis is a vector of integers with the following supported values:

  - 0 = actual/actual (default)
  - 1 = 30/360 (SIA)
  - 2 = actual/360
  - 3 = actual/365
  - 4 = 30/360 (BMA)
  - 5 = 30/360 (ISDA)
  - 6 = 30/360 (European)
  - 7 = actual/365 (Japanese)
  - 8 = actual/actual (ICMA)
  - 9 = actual/360 (ICMA)
  - 10 = actual/365 (ICMA)
  - 11 = 30/360E (ICMA)
  - 12 = actual/actual (ISDA)
  - 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

- `EndMonthRule` is a `NINST`-by-`1` matrix for the end-of-month rule. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. When the value is `0`, the end-of-month rule is ignored, meaning that a bond's coupon payment date is always the same numerical day of the month. When the value is `1`, the end-of-month rule is set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

- `IssueDate` is a `NINST`-by-`1` matrix for the bond issue date.

- `FirstCouponDate` is a `NINST`-by-`1` matrix for a date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

- `LastCouponDate` is a `NINST`-by-`1` matrix for a last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not

specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

- `StartDate` is a `NINST`-by-1 matrix for date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

- Face value. `Face` is a `NINST`-by-1 vector or `NINST`-by-1 cell array of face values, or face value schedules. For the latter case, each element of the cell array is a `NumDates`-by-2 cell array, where the first column is dates and the second column is its associated face value. The date indicates the last day that the face value is valid. Default is 100.

- `Options` is a derivatives pricing options structure created with `derivset`.

Note The `Settle` date for every bond with embedded option is set to the `ValuationDate` of the HW tree; the bond's argument for `Settle` date is ignored.

Description   `[Price, PriceTree] = optembndbyhw(HWTree, CouponRate, Settle, Maturity, OptSpec, Strike, ExerciseDates, 'Name1', Value1, 'Name2', Value2, ...)` prices bonds with embedded options by a HW interest-rate tree.

`Price` is a number of instruments (`NINST`)-by-1 matrix of expected prices at time 0.

`PriceTree` is a structure of trees containing vectors of instrument prices and observation times for each node. Within `PriceTree`:

- `PriceTree.PTree` contains the clean prices.

- `PriceTree.tObs` contains the observation times.

`optembndbyhw` computes prices of vanilla bonds with embedded options, stepped coupon bonds with embedded options, and bonds with sinking fund option provisions.

Definitions

Vanilla Bond with Embedded Option

A vanilla coupon bond is a security representing an obligation to repay a borrowed amount at a designated time and to make periodic interest payments until that time. The issuer of a bond makes the periodic interest payments until the bond matures. At maturity, the issuer pays to the holder of the bond the principal amount owed (face value) and the last interest payment. A vanilla bond with an embedded option is where an option contract has an underlying asset of a vanilla bond.

Stepped Coupon Bond with Callable and Puttable Features

A step-up and step-down bond is a debt security with a predetermined coupon structure over time. With these instruments, coupons increase (step up) or decrease (step down) at specific times during the life of the bond. Stepped coupon bonds can have options features (call and puts).

Sinking Fund Bond with Embedded Option

A sinking fund bond is a coupon bond with a sinking fund provision. This provision obligates the issuer to amortize portions of the principal prior to maturity, affecting bond prices since the time of the principal repayment changes. This means that investors receive the coupon and a portion of the principal paid back over time. These types of bonds reduce credit risk, since it lowers the probability of investors not receiving their principal payment at maturity.

The bond may have a sinking fund option provision allowing the issuer to retire the sinking fund obligation either by purchasing the bonds to be redeemed from the market or by calling the bond via a sinking fund call, whichever is cheaper. If interest rates are high, then the issuer will buy back the requirement amount of bonds from the market since bonds will be cheap, but if interest rates are low (bond prices are high), then most likely the issuer will be buying the bonds at the call price. Unlike a call feature, however, if a bond has a sinking fund option provision, it is an obligation, not an option, for the issuer to buy back the increments of the issue as stated. Because of this, a sinking fund bond trades at a lower price than a non-sinking fund bond.

Examples   Create a `HWTree` with the following data:

```
ZeroRates = [ 0.035;0.04;0.045];
Compounding = 1;
StartDates = ['jan-1-2007';'jan-1-2008';'jan-1-2009'];
EndDates   = ['jan-1-2008';'jan-1-2009';'jan-1-2010'];
ValuationDate = 'jan-1-2007';
```

Create a `RateSpec`:

```
RateSpec = intenvset('Rates', ZeroRates, 'StartDates', ValuationDate, 'EndDates', ...
EndDates, 'Compounding', Compounding, 'ValuationDate', ValuationDate);
```

Specify a `TimeSpec`:

```
HWTimeSpec = hwtimespec(ValuationDate, EndDates, Compounding);
```

Specify a `VolSpec`:

```
VolDates = ['jan-1-2008';'jan-1-2009';'jan-1-2010'];
VolCurve = 0.01;
AlphaDates = 'jan-1-2010';
AlphaCurve = 0.1;
HWVolSpec = hwvolspec(ValuationDate, VolDates, VolCurve, AlphaDates, AlphaCurve);
```

Build a `HWTree`:

optembndbyhw

```
HWTree = hwtree(HWVolSpec, RateSpec, HWTimeSpec);
```

Compute the price of an American puttable bond that pays an annual coupon of 5.25%, matures on January 1, 2010, and is puttable from January 1, 2008 to January 1, 2010:

```
BondSettlement = 'jan-1-2007';
BondMaturity   = 'jan-1-2010';
CouponRate = 0.0525;
Period = 1;
OptSpec = 'put';
Strike = [100];
ExerciseDates = {'jan-1-2008' '01-Jan-2010'};
AmericanOpt = 1;

PricePutBondHW = optembndbyhw(HWTree, CouponRate, BondSettlement, BondMaturity,...
OptSpec, Strike, ExerciseDates,'Period', 1, 'AmericanOpt', 1)

PricePutBondHW =

102.8801
```

Price the following single stepped callable bonds using the following data:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2010';
StartDates = ValuationDate;
EndDates = {'Jan-1-2011'; 'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'};
Compounding = 1;

% Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);
```

```
% Instrument
Settle = '01-Jan-2010';
Maturity = {'01-Jan-2013';'01-Jan-2014'};
CouponRate = {{'01-Jan-2012' .0425;'01-Jan-2014' .0750}};
OptSpec='call';
Strike=100;
ExerciseDates='01-Jan-2012';  %Callable in two years

% Build the tree with the following data
VolDates = ['1-Jan-2011'; '1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014'];
VolCurve = 0.01;
AlphaDates = '01-01-2014';
AlphaCurve = 0.1;


HWVolSpec = hwvolspec(RS.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RS.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RS, HWTimeSpec);

% The first row corresponds to the price of the callable bond with maturity
% of three years. The second row corresponds to the price of the callable
% bond with maturity of four years.

PHW= optembndbyhw(HWT, CouponRate,  Settle, Maturity,OptSpec, Strike,...
ExerciseDates, 'Period', 1)

PHW =

  100.0521
   99.8322
```

A corporation issues a two year bond with a sinking fund obligation requiring the company to sink 1/3 of face value after the first year. The company has the option to buy the bonds in the market or call them at

\$99. The following data describes the details needed for pricing the sinking fund bond:

```
% The data for the interest rate term structure is as follows:
Rates = [0.1;0.1;0.1;0.1];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

% Create RateSpec
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

% Build the HW tree
% The data to build the tree is as follows:
VolDates = ['1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014';'1-Jan-2015'];
VolCurve = 0.01;
AlphaDates = '01-01-2015';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec);


% Instrument
% The bond has a coupon rate of 9%, a period of one year and matures in
% 1-Jan-2013. Face decreases 1/3 after the first year.
CouponRate = 0.09;
Settle = 'Jan-1-2011';
Maturity =  'Jan-1-2013';
Period = 1;
Face = { ...
            {'Jan-1-2012'  100; ...
             'Jan-1-2013'   66.6666}; ...
```

```
                };

% Option provision
OptSpec = 'call';
Strike = 99;
ExerciseDates = 'Jan-1-2012';

% Price of non-sinking fund bond.
PNSF = bondbyhw(HWT, CouponRate, Settle, Maturity, Period)

PNSF =

   98.2645

% Price of the bond with the option sinking provision.
PriceSF = optembndbyhw(HWT, CouponRate, Settle, Maturity,...
OptSpec, Strike, ExerciseDates, 'Period', Period, 'Face', Face)

PriceSF =

   98.1594
```

See Also   hwprice | cfamounts | hwtree | instoptembnd

optemfloatbybdt

Purpose	Price embedded option on floating-rate note BDT interest-rate tree

Syntax

```
Price = optemfloatbybdt(BDTTree,Spread,Settle,Maturity,
OptSpec,Strike,ExerciseDates)
Price = optemfloatbybdt(BDTTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates,AmericanOpt,Reset,Basis,Principal,
Options,EndMonthRule)
[Price,PriceTree] = optemfloatbybdt( ___ )

Price = optemfloatbybdt(BDTTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates)
Price = optemfloatbybdt( ___ , Name,Value)
[Price,PriceTree] = optemfloatbybdt( ___ )
```

Description

`Price = optemfloatbybdt(BDTTree,Spread,Settle,Maturity,
OptSpec,Strike,ExerciseDates)` returns the embedded option price
of the instrument(s) at valuation date.

`Price =
optemfloatbybdt(BDTTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates,AmericanOpt,Reset,Basis,Principal,
Options,EndMonthRule)` returns the embedded option price of the
instrument(s) at valuation date using optional arguments.

`[Price,PriceTree] = optemfloatbybdt( ___ )` returns the option
price of the instrument(s) at valuation date and a structure of trees
containing vectors of option prices at each node.

`Price =
optemfloatbybdt(BDTTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates)` returns the embedded option price of
the instrument(s) at valuation date.

`Price = optemfloatbybdt( ___ , Name,Value)` returns the embedded option price of the instrument(s) at valuation date using optional name-value pair arguments.

`[Price,PriceTree] = optemfloatbybdt( ___ )` returns the embedded option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node using optional name-value pair arguments.

Input Arguments

BDTTree - Interest-rate tree structure
binomial tree structure

Interest-rate tree specified as a structure by using `bdttree`.

Data Types
struct

Spread - Number of basis points over the reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate specified as a vector nonnegative integers for the number of instruments (NINST)-by-1).

Data Types
single | double

Settle - Settlement dates of floating-rate note
`ValuationDate` of HW Tree (default) | nonnegative integer | vector of nonnegative integers

Settlement dates of floating-rate note specified as nonnegative integers using a NINST-by-1 vector of dates.

Note The `Settle` date for every floating-rate note is set to the `ValuationDate` of the HW Tree. The floating-rate note argument `Settle` is ignored.

Data Types
`char`

OptSpec - Definition of option
string | cell array of strings

Definition of option as `'call'` or `'put'` specified as a `NINST-by-1` cell array of strings for `'call'` or `'put'`.

Data Types
`cell` | `char`

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values specified nonnegative integers using as `NINST-by-NSTRIKES` vector of strike price values.

Data Types
`single` | `double`

ExerciseDates - Exercise date for option (European, Bermuda, or American)
nonnegative integer | vector of nonnegative integers

Exercise date for option (European, Bermuda, or American) specified as nonnegative integers using a `NINST-by-NSTRIKES` or `NINST-by-2` vector of for the option exercise dates.

- If a European or Bermuda option, the `ExerciseDates` is a 1-by-1 (European) or 1-by-`NSTRIKES` (Bermuda) vector of exercise dates. For a European option, there is only one `ExerciseDate` on the option expiry date.

- If an American option, then `ExerciseDates` is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-`NaN` date, or if `ExerciseDates` is 1-by-1, the option exercises between the `Settle` date and the single listed `ExerciseDate`.

Data Types
char | cell

AmericanOpt - (Optional) Option type
scalar | vector of positive integers[0,1]

Option type specified as NINST-by-1 positive integer scalar flags with values:

- 0 — European/Bermuda

- 1 — American

Data Types
single | double

Reset - (Optional) Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the values [1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
double

Basis - (Optional) Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] | vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using a NINST-by-1 vector. The Basis value represents the basis used when annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
double

Principal - (Optional) Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1 vector of notional principal amounts.

Data Types
double

Options - (Optional) Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using derivset.

Data Types
struct

EndMonthRule - (Optional) End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price = optemfloatbybdt(BDTTree,Spread,Settle,Maturity,OptSpec,Strike,Exercise

'AmericanOpt' - Option type
scalar | vector of positive integers[0,1]

Option type specified as NINST-by-1 positive integer scalar flags with values:

- 0 — European/Bermuda

- 1 — American

Data Types
single | double

'Reset' - Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the values [1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
char

'Basis' - Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] | vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using a NINST-by-1 vector. The Basis value represents the basis used when annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
char

'Principal' - Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a `NINST`-by-1 vector of notional principal amounts.

Data Types
char

'Options' - Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using `derivset`.

Data Types
struct

'EndMonthRule' - End-of-month rule flag

1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as a nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
char

Output Arguments

Price - Expected prices of the floating-rate note embedded option at time 0
scalar | vector

Expected prices of the floating-rate note embedded option at time 0 is returned as a scalar or an NINST-by-1 vector.

PriceTree - Structure of trees containing vectors of embedded option prices at each node
tree structure

Structure of trees containing vectors of instrument prices and accrued interest and a vector of observation times for each node returned as:

- PriceTree.PTree contains embedded option prices.

Examples

Price Callable Embedded Option for Floating-Rate Note

Define the interest-rate term structure.

```
Rates = [0.03;0.034;0.038;0.04];
ValuationDate = 'Jan-1-2012';
StartDates = ValuationDate;
EndDates = {'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'; 'Jan-1-2016'};
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

           FinObj: 'RateSpec'
      Compounding: 1
             Disc: [4x1 double]
            Rates: [4x1 double]
         EndTimes: [4x1 double]
       StartTimes: [4x1 double]
         EndDates: [4x1 double]
       StartDates: 734869
    ValuationDate: 734869
            Basis: 0
      EndMonthRule: 1
```

Build the BDT tree and assume a volatility of 10%.

```
Sigma = 0.1;
BDTTimeSpec = bdttimespec(ValuationDate, EndDates);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
BDTT = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);

BDTT =

      FinObj: 'BDTFwdTree'
     VolSpec: [1x1 struct]
    TimeSpec: [1x1 struct]
    RateSpec: [1x1 struct]
        tObs: [0 1 2 3]
        dObs: [734869 735235 735600 735965]
        TFwd: {[4x1 double]  [3x1 double]  [2x1 double]  [3]}
      CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
     FwdTree: {1x4 cell}
```

Define the floater instruments with the embedded call option.

```
Spread = 10;
Settle = 'Jan-1-2012';
Maturity =  {'Jan-1-2015';'Jan-1-2016'};
Period = 1;
OptSpec = {'call'};
Strike = 101;
ExerciseDates = 'Jan-1-2015';
```

Compute the price of the floaters with the embedded call.

```
Price= optemfloatbybdt(BDTT, Spread, Settle, Maturity, OptSpec, Strike,...
ExerciseDates)

Price =

  100.2800
  100.3655
```

See Also   optembndbybdt | optemfloatbyhjm | optemfloatbybk |
optemfloatbyhw | instoptemfloat

Concepts   • **basis** on page Glossary-1

| **Purpose** | Price embedded option on floating-rate note BK interest-rate tree |

Syntax

```
Price = optemfloatbybk(BKTree,Spread,Settle,Maturity,
OptSpec,Strike,ExerciseDates)
Price = optemfloatbybk(BKTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates,AmericanOpt,Reset,Basis,Principal,
Options,EndMonthRule)
[Price,PriceTree] = optemfloatbybk( ___ )

Price = optemfloatbybk(BKTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates)
Price = optemfloatbybk( ___ , Name,Value)
[Price,PriceTree] = optemfloatbybk( ___ )
```

Description

`Price = optemfloatbybk(BKTree,Spread,Settle,Maturity, OptSpec,Strike,ExerciseDates)` returns the embedded option price of the instrument(s) at valuation date.

```
Price =
optemfloatbybk(BKTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates,AmericanOpt,Reset,Basis,Principal,
Options,EndMonthRule)
```
returns the embedded option price of the instrument(s) at valuation date using optional arguments.

`[Price,PriceTree] = optemfloatbybk( ___ )` returns the option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node.

```
Price =
optemfloatbybk(BKTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates)
```
returns the embedded option price of the instrument(s) at valuation date.

optemfloatbybk

Price = optemfloatbybk(___ , Name,Value) returns the embedded option price of the instrument(s) at valuation date using optional name-value pair arguments.

[Price,PriceTree] = optemfloatbybk(___) returns the embedded option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node using optional name-value pair arguments.

Input Arguments

BKTree - Interest-rate tree structure
binomial tree structure

Interest-rate tree specified as a structure by using bktree.

Data Types
struct

Spread - Number of basis points over the reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate specified as a vector nonnegative integers for the number of instruments (NINST)-by-1).

Data Types
single | double

Settle - Settlement dates of floating-rate note
ValuationDate of HW Tree (default) | nonnegative integer | vector of nonnegative integers

Settlement dates of floating-rate note specified as nonnegative integers using a NINST-by-1 vector of dates.

Note The Settle date for every floating-rate note is set to the ValuationDate of the HW Tree. The floating-rate note argument Settle is ignored.

Data Types
char

OptSpec - Definition of option
string | cell array of strings

Definition of option as `'call'` or `'put'` specified as a NINST-by-1 cell array of strings for `'call'` or `'put'`.

Data Types
cell | char

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values specified nonnegative integers using as NINST-by-NSTRIKES vector of strike price values.

Data Types
single | double

ExerciseDates - Exercise date for option (European, Bermuda, or American)
nonnegative integer | vector of nonnegative integers

Exercise date for option (European, Bermuda, or American) specified as nonnegative integers using a NINST-by-NSTRIKES or NINST-by-2 vector of for the option exercise dates.

- If a European or Bermuda option, the ExerciseDates is a 1-by-1 (European) or 1-by-NSTRIKES (Bermuda) vector of exercise dates. For a European option, there is only one ExerciseDate on the option expiry date.

- If an American option, then ExerciseDates is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-NaN date, or if ExerciseDates is 1-by-1, the option exercises between the Settle date and the single listed ExerciseDate.

optemfloatbybk

Data Types
char | cell

AmericanOpt - (Optional) Option type
scalar | vector of positive integers[0,1]

Option type specified as NINST-by-1 positive integer scalar flags with values:

- 0 — European/Bermuda

- 1 — American

Data Types
single | double

Reset - (Optional) Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the values [1,2,3,4,6,12] in a NINST-by-1 vector. .

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
double

Basis - (Optional) Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] | vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using a NINST-by-1 vector. The Basis value represents the basis used when annualizing the input forward-rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

Data Types
double

Principal - (Optional) Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1 vector of notional principal amounts.

Data Types
double

Options - (Optional) Structure containing derivatives pricing options

structure

Structure containing derivatives pricing options specified using derivset.

Data Types
struct

EndMonthRule - (Optional) End-of-month rule flag

1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price = optemfloatbybk(BKTree,Spread,Settle,Maturity,OptSpec,Strike,ExerciseDates

'AmericanOpt' - Option type

scalar | vector of positive integers[0,1]

Option type specified as NINST-by-1 positive integer scalar flags with values:

- 0 — European/Bermuda

- 1 — American

Data Types
single | double

'Reset' - Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of
positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the
values [1,2,3,4,6,12] in a NINST-by-1 vector. .

Note Payments on floating-rate notes (FRNs) are determined by the
effective interest-rate between reset dates. If the reset period for a
FRN spans more than one tree level, calculating the payment becomes
impossible due to the recombining nature of the tree. That is, the tree
path connecting the two consecutive reset dates can not be uniquely
determined because there will be more than one possible path for
connecting the two payment dates.

Data Types
char

'Basis' - Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
char

'Principal' - Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a `NINST`-by-1 vector of notional principal amounts.

Data Types
char

'Options' - Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using `derivset`.

Data Types
struct

'EndMonthRule' - End-of-month rule flag

1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as a nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
char

Output Arguments

Price - Expected prices of the floating-rate note embedded option at time 0
scalar | vector

Expected prices of the floating-rate note embedded option at time 0 is returned as a scalar or an NINST-by-1 vector.

PriceTree - Structure of trees containing vectors of embedded option prices at each node
tree structure

Structure of trees containing vectors of instrument prices and accrued interest and a vector of observation times for each node returned as:

- PriceTree.PTree contains embedded option prices.

Examples

Price European Callable Embedded Option for Floating-Rate Note

Define the interest-rate term structure.

```
Rates = [0.03;0.034;0.038;0.04];
ValuationDate = 'Jan-1-2012';
StartDates = ValuationDate;
```

```
EndDates = {'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'; 'Jan-1-2016'};
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: [4x1 double]
             Rates: [4x1 double]
          EndTimes: [4x1 double]
        StartTimes: [4x1 double]
          EndDates: [4x1 double]
        StartDates: 734869
     ValuationDate: 734869
             Basis: 0
       EndMonthRule: 1
```

Build the BK tree.

```
VolDates = ['1-Jan-2013'; '1-Jan-2014'; '1-Jan-2015';'1-Jan-2016'];
VolCurve = 0.01;
AlphaDates = '01-01-2016';
AlphaCurve = 0.1;

BKVolSpec = bkvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
           AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RateSpec.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RateSpec, BKTimeSpec)

BKT =

      FinObj: 'BKFwdTree'
     VolSpec: [1x1 struct]
```

```
    TimeSpec: [1x1 struct]
    RateSpec: [1x1 struct]
        tObs: [0 1 2 3]
        dObs: [734869 735235 735600 735965]
      CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
       Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
     Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
     FwdTree: {[1.0300]  [1.0387 1.0380 1.0374]  [1x5 double]  [1x5 double]}
```

The floater instrument has a spread of 15, a period of one year, and matures and is callable on Jan-1-2015.

```
Spread = 15;
Settle = 'Jan-1-2012';
Maturity =  'Jan-1-2015';
Period = 1;
OptSpec = {'call'};
Strike =101;
ExerciseDates = 'Jan-1-2015';
```

Compute the price of the floater with the embedded call.

```
Price= optemfloatbybk(BKT, Spread, Settle, Maturity,...
OptSpec, Strike, ExerciseDates)

Price =

  100.4201
```

See Also   optembndbybk | optemfloatbyhjm | optemfloatbybdt | optemfloatbyhw | instoptemfloat

Concepts   • **basis** on page Glossary-1

optemfloatbyhjm

Purpose   Price embedded option on floating-rate note HJM interest-rate tree

Syntax
```
Price = optemfloatbyhjm(HJMTree,Spread,Settle,Maturity,
OptSpec,Strike,ExerciseDates)
Price = optemfloatbyhjm(HJMTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates,AmericanOpt,Reset,Basis,Principal,
Options,EndMonthRule)
[Price,PriceTree] = optemfloatbyhjm( ___ )

Price = optemfloatbyhjm(HJMTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates)
Price = optemfloatbyhjm( ___ ,Name,Value)
[Price,PriceTree] = optemfloatbyhjm( ___ )
```

Description   `Price = optemfloatbyhjm(HJMTree,Spread,Settle,Maturity,` `OptSpec,Strike,ExerciseDates)` returns the embedded option price of the instrument(s) at the valuation date.

```
Price =
optemfloatbyhjm(HJMTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates,AmericanOpt,Reset,Basis,Principal,
Options,EndMonthRule)
```
returns the embedded option price of the instrument(s) at the valuation date using optional arguments.

`[Price,PriceTree] = optemfloatbyhjm( ___ )` returns the option price of the instrument(s) at the valuation date and a structure of trees containing vectors of option prices at each node.

```
Price =
optemfloatbyhjm(HJMTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates)
```
returns the embedded option price of the instrument(s) at the valuation date.

`Price = optemfloatbyhjm( ___ ,Name,Value)` returns the embedded option price of the instrument(s) at the valuation date using optional name-value pair arguments.

`[Price,PriceTree] = optemfloatbyhjm( ___ )` returns the embedded option price of the instrument(s) at the valuation date and a structure of trees containing vectors of option prices at each node using optional name-value pair arguments.

Input Arguments

HJMTree - Interest-rate tree structure
binomial tree structure

Interest-rate tree, specified as a structure by using `hjmtree`.

Data Types
struct

Spread - Number of basis points over reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate, specified as a vector of nonnegative integers for the number of instruments (NINST)-by-1.

Data Types
single | double

Settle - Settlement dates of floating-rate note
`ValuationDate` of HW tree (default) | nonnegative integer | vector of nonnegative integers

Settlement dates of floating-rate note, specified as nonnegative integers using a NINST-by-1 vector of dates.

Note The `Settle` date for every floating-rate note is set to the `ValuationDate` of the HW tree. The floating-rate note argument `Settle` is ignored.

optemfloatbyhjm

Data Types
char

OptSpec - Option definition
string | cell array of strings

Option definition specified as a `'call'` or `'put'` using a NINST-by-1 cell array of strings.

Data Types
cell | char

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values, specified as nonnegative integers using a NINST-by-NSTRIKES vector of strike price values.

Data Types
single | double

ExerciseDates - Exercise date for European, Bermuda, or American option
nonnegative integer | vector of nonnegative integers

Exercise date for European, Bermuda, or American option, specified as nonnegative integers using a NINST-by-NSTRIKES or NINST-by-2 vector of option exercise dates.

- For a European or Bermuda option, ExerciseDates is a 1-by-1 (European) or 1-by-NSTRIKES (Bermuda) vector of exercise dates. For a European option, there is only one ExerciseDate on the option expiry date.

- For an American option, ExerciseDates is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-NaN date, or if ExerciseDates is 1-by-1, the option exercises between the Settle date and the single listed ExerciseDate.

Data Types
char | cell

AmericanOpt - (Optional) Option type
scalar | vector of positive integers[0,1]

Option type, specified as NINST-by-1 positive integer flags with values:

- 0 — European/Bermuda

- 1 — American

Data Types
single | double

Reset - (Optional) Frequency of payments per year
1 (default) | positive integer from the set [1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year, specified as positive integers for the values [1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates cannot be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
double

Basis - (Optional) Day-count basis of instrument
0 (actual/actual) (default) | positive integers of the set [1...13] | vector of positive integers of the set [1...13]

optemfloatbyhjm

Day-count basis of the instrument, specified as a positive integer using a `NINST`-by-1 vector. The `Basis` value represents the basis used when annualizing the input forward-rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

Data Types
`double`

Principal - (Optional) Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values, specified as a nonnegative integer using a `NINST`-by-1 vector of notional principal amounts.

Data Types
`double`

Options - (Optional) Structure containing derivatives pricing options

structure

Structure containing derivatives pricing options, specified using `derivset`.

Data Types
`struct`

EndMonthRule - (Optional) End-of-month rule flag

1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag, specified as nonnegative integer `0` or `1` using a `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
`double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: Price = optemfloatbyhjm(HJMTree,Spread,Settle,Maturity,OptSpec,Strike,Exercise

'AmericanOpt' - Option type

scalar | vector of positive integers[0,1]

Option type, specified as `NINST`-by-1 positive integer scalar flags with values:

optemfloatbyhjm

- `0` — European/Bermuda

- `1` — American

Data Types
single | double

'Reset' - Frequency of payments per year
1 (default) | positive integer from the set [1,2,3,4,6,12] | vector of
positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year, specified as positive integers for the
values [1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the
effective interest-rate between reset dates. If the reset period for a
FRN spans more than one tree level, calculating the payment becomes
impossible due to the recombining nature of the tree. That is, the tree
path connecting the two consecutive reset dates cannot be uniquely
determined because there will be more than one possible path for
connecting the two payment dates.

Data Types
char

'Basis' - Day-count basis of instrument
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument, specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

Data Types
char

'Principal' - Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values, specified as a nonnegative integer using a NINST-by-1 vector of notional principal amounts.

Data Types
char

'Options' - Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using derivset.

Data Types
struct

'EndMonthRule' - End-of-month rule flag

1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag, specified as a nonnegative integer [0,1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
char

Output Arguments

Price - Expected prices of floating-rate note embedded option at time 0
scalar | vector

Expected prices of the floating-rate note embedded option at time 0, returned as a scalar or a NINST-by-1 vector.

PriceTree - Structure of trees containing vectors of embedded option prices at each node
tree structure

Structure of trees containing vectors of instrument prices and accrued interest and a vector of observation times for each node, returned as:

- PriceTree.PTree contains embedded option prices.

Examples

Price European Callable Embedded Option for a Floating-Rate Note

Define the interest-rate term structure.

```
Rates = [0.03;0.035;0.040;0.045];
ValuationDate = 'Jan-1-2012';
StartDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
```

```
EndDates = {'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'; 'Jan-1-2016'};
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: [4x1 double]
             Rates: [4x1 double]
          EndTimes: [4x1 double]
        StartTimes: [4x1 double]
          EndDates: [4x1 double]
        StartDates: [4x1 double]
     ValuationDate: 734869
             Basis: 0
       EndMonthRule: 1
```

Build the HJM tree.

```
VolSpec = hjmvolspec('Constant', 0.01);
TimeSpec = hjmtimespec(RateSpec.ValuationDate, EndDates, Compounding);
HJMTree = hjmtree(VolSpec, RateSpec, TimeSpec)

HJMTree =

      FinObj: 'HJMFwdTree'
      VolSpec: [1x1 struct]
     TimeSpec: [1x1 struct]
     RateSpec: [1x1 struct]
         tObs: [0 1 2 3]
         dObs: [734869 735235 735600 735965]
         TFwd: {[4x1 double]  [3x1 double]  [2x1 double]  [3]}
        CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
```

```
                   FwdTree: {[4x1 double]  [3x1x2 double]  [2x2x2 double]  [1x4x2 double]}
```

The floater instrument has a spread of 15, a period of one year, and matures and is callable on Jan-1-2016.

```
Spread = 15;
Settle = 'Jan-1-2012';
Maturity =  'Jan-1-2016';
Period = 1;
OptSpec = {'call'};
Strike = 95;
ExerciseDates = 'Jan-1-2016';
```

Compute the price of the floater with the embedded call.

```
Price = optemfloatbyhjm(HJMTree, Spread, Settle, Maturity,...
OptSpec, Strike, ExerciseDates)

Price =

   96.2355
```

See Also   optembndbyhjm | optemfloatbyhw | optemfloatbybdt | optemfloatbybk | instoptemfloat

Concepts   • **basis** on page Glossary-1

optemfloatbyhw

Purpose   Price embedded option on floating-rate note HW interest-rate tree

Syntax   ```
Price = optemfloatbyhw(HWTree,Spread,Settle,Maturity,
OptSpec,Strike,ExerciseDates)
Price = optemfloatbyhw(HWTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates,AmericanOpt,Reset,Basis,Principal,
Options,EndMonthRule)
[Price,PriceTree] = optemfloatbyhw(___)

Price = optemfloatbyhw(HWTree,Spread,Settle,Maturity,OptSpec,
Strike,ExerciseDates)
Price = optemfloatbyhw(___ ,Name,Value)
[Price,PriceTree] = optemfloatbyhw(___)
```

Description   `Price = optemfloatbyhw(HWTree,Spread,Settle,Maturity,`
`OptSpec,Strike,ExerciseDates)` returns the embedded option price
of the instrument(s) at the valuation date.

`Price =`
`optemfloatbyhw(HWTree,Spread,Settle,Maturity,OptSpec,`
`Strike,ExerciseDates,AmericanOpt,Reset,Basis,Principal,`
`Options,EndMonthRule)` returns the embedded option price of the
instrument(s) at the valuation date using optional arguments.

`[Price,PriceTree] = optemfloatbyhw( ___ )` returns the option price
of the instrument(s) at the valuation date and a structure of trees
containing vectors of option prices at each node.

`Price =`
`optemfloatbyhw(HWTree,Spread,Settle,Maturity,OptSpec,`
`Strike,ExerciseDates)` returns the embedded option price of the
instrument(s) at the valuation date.

optemfloatbyhw

Price = optemfloatbyhw(___ ,Name,Value) returns the embedded
option price of the instrument(s) at the valuation date using optional
name-value pair arguments.

[Price,PriceTree] = optemfloatbyhw(___) returns the embedded
option price of the instrument(s) at the valuation date and a structure
of trees containing vectors of option prices at each node using optional
name-value pair arguments.

**Input
Arguments**

HWTree - Interest-rate tree structure
binomial tree structure

Interest-rate tree, specified as a structure by using hwtree.

Data Types
struct

Spread - Number of basis points over reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate, specified as a vector of
nonnegative integers for the number of instruments (NINST)-by-1).

Data Types
single | double

Settle - Settlement dates of floating-rate note
ValuationDate of HW tree (default) | nonnegative integer | vector of
nonnegative integers

Settlement dates of the floating-rate note, specified as nonnegative
integers using a NINST-by-1 vector of dates.

Note The Settle date for every floating-rate note is set to the
ValuationDate of the HW tree. The floating-rate note argument
Settle is ignored.

Data Types
char

OptSpec - Option definition
string | cell array of strings

Option definition, specified as `'call'` or `'put'` using a NINST-by-1 cell array of strings.

Data Types
cell | char

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values specified as nonnegative integers using a NINST-by-NSTRIKES vector of strike price values.

Data Types
single | double

ExerciseDates - Exercise date for option European, Bermuda, or American option
nonnegative integer | vector of nonnegative integers

Exercise date for European, Bermuda, or American option, specified as nonnegative integers using a NINST-by-NSTRIKES or NINST-by-2 vector of option exercise dates.

- For a European or Bermuda option, ExerciseDates is a 1-by-1 (European) or 1-by-NSTRIKES (Bermuda) vector of exercise dates. For a European option, there is only one ExerciseDate on the option expiry date.

- For an American option, ExerciseDates is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-NaN date, or if ExerciseDates is 1-by-1, the option exercises between the Settle date and the single listed ExerciseDate.

optemfloatbyhw

Data Types
char | cell

AmericanOpt - (Optional) Option type
scalar | vector of positive integers [0,1]

Option type, specified as NINST-by-1 positive integer scalar flags with values:

- 0 — European/Bermuda

- 1 — American

Data Types
single | double

Reset - (Optional) Frequency of payments per year
1 (default) | positive integer from the set [1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the values [1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates cannot be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
double

Basis - (Optional) Day-count basis of instrument
0 (actual/actual) (default) | positive integers of the set [1...13] | vector of positive integers of the set [1...13]

Day-count basis of the instrument, specified as a positive integer using a `NINST`-by-1 vector. The `Basis` value represents the basis used when annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
`double`

Principal - (Optional) Principal values

100 (default) | nonnegative integer | vector of nonnegative integers

Principal values, specified as a nonnegative integer using a `NINST`-by-1 vector of notional principal amounts.

Data Types
`double`

optemfloatbyhw

Options - (Optional) Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options, specified using `derivset`.

Data Types
struct

EndMonthRule - (Optional) End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag, specified as nonnegative integer [0,1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price = optemfloatbyhw(HWTree,Spread,Settle,Maturity,OptSpec,Strike,ExerciseDates

'AmericanOpt' - Option type
scalar | vector of positive integers[0,1]

Option type, specified as NINST-by-1 positive integer scalar flags with values:

- 0 — European/Bermuda

- 1 — American

Data Types
single | double

'Reset' - Frequency of payments per year
1 (default) | positive integer from the set [1,2,3,4,6,12] | vector of
positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year, specified as positive integers for the
values [1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the
effective interest-rate between reset dates. If the reset period for a
FRN spans more than one tree level, calculating the payment becomes
impossible due to the recombining nature of the tree. That is, the tree
path connecting the two consecutive reset dates cannot be uniquely
determined because there will be more than one possible path for
connecting the two payment dates.

Data Types
char

'Basis' - Day-count basis of instrument
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument, specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
char

'Principal' - Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified, as a nonnegative integer using a NINST-by-1
vector of notional principal amounts.

Data Types
char

'Options' - Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options, specified using
derivset.

Data Types
struct

'EndMonthRule' - End-of-month rule flag

1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag, specified as a nonnegative integer[0,1] using a `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
char

Output Arguments

Price - Expected prices of floating-rate note embedded option at time 0
scalar | vector

Expected prices of the floating-rate note embedded option at time 0, returned as a scalar or an `NINST`-by-1 vector.

PriceTree - Structure of trees containing vectors of embedded option prices at each node
tree structure

Structure of trees containing vectors of instrument prices and accrued interest and a vector of observation times for each node, returned as:

- `PriceTree.PTree` contains embedded option prices.

Examples

Price European Callable Embedded Option for Floating-Rate Note

Define the interest-rate term structure.

```
Rates = [0.03;0.034;0.038;0.04];
ValuationDate = 'Jan-1-2012';
StartDates = ValuationDate;
```

```
EndDates = {'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'; 'Jan-1-2016'};
Compounding = 1;
```

Create the `RateSpec`.

```
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates',...
StartDates, 'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: [4x1 double]
             Rates: [4x1 double]
          EndTimes: [4x1 double]
        StartTimes: [4x1 double]
          EndDates: [4x1 double]
        StartDates: 734869
     ValuationDate: 734869
             Basis: 0
       EndMonthRule: 1
```

Build the HW tree using the following:

```
VolDates = ['1-Jan-2013'; '1-Jan-2014'; '1-Jan-2015';'1-Jan-2016'];
VolCurve = 0.01;
AlphaDates = '01-01-2016';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
            AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec)

HWT =

     FinObj: 'HWFwdTree'
     VolSpec: [1x1 struct]
```

```
        TimeSpec: [1x1 struct]
        RateSpec: [1x1 struct]
           tObs: [0 1 2 3]
           dObs: [734869 735235 735600 735965]
          CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
           Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
         Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
         FwdTree: {[1.0300]  [1.0553 1.0381 1.0211]  [1.0813 1.0636 1.0462 1.0291 1.0123]  [1.0815 1.063
```

Define the floater instruments with the embedded call option.

```
Spread = 10;
Settle = 'Jan-1-2012';
Maturity =  {'Jan-1-2015';'Jan-1-2016'};
Period = 1;
OptSpec = {'call'};
Strike = 101;
ExerciseDates = 'Jan-1-2015';
```

Compute the price of the floaters with the embedded call.

```
Price= optemfloatbybdt(BDTT, Spread, Settle, Maturity, OptSpec, Strike,...
ExerciseDates)

Price =

  100.2800
  100.3655
```

See Also   optembndbyhw | optemfloatbyhjm | optemfloatbybdt |
optemfloatbybk | instoptemfloat

Concepts   • **basis** on page Glossary-1

optfloatbybdt

Purpose	Price options on floating-rate notes for BDT interest-rate tree

Syntax

```
Price = optfloatbybdt(BDTTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
Price = optfloatbybdt(BDTTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity,Reset,Basis,Principal,Options,EndMonthRule)
[Price,PriceTree] = optfloatbybdt( ___ )

Price = optfloatbybdt(BDTTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
Price = optfloatbybdt( ___ , Name,Value)
[Price,PriceTree] = optfloatbybdt( ___ )
```

Description

```
Price =
optfloatbybdt(BDTTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
```
returns the option price of the instrument(s) at valuation date.

```
Price =
optfloatbybdt(BDTTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity,Reset,Basis,Principal,Options,EndMonthRule)
```
returns the option price of the instrument(s) at valuation date using optional arguments.

`[Price,PriceTree] = optfloatbybdt( ___ )` returns the option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node.

```
Price =
optfloatbybdt(BDTTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
```
returns the option price of the instrument(s) at valuation date.

`Price = optfloatbybdt( ___ , Name,Value)` returns the option price of the instrument(s) at valuation date using optional Name-Value pairs.

[Price,PriceTree] = optfloatbybdt(___) returns the option price
of the instrument(s) at valuation date and a structure of trees containing
vectors of option prices at each node using optional name-value pair
arguments.

Input Arguments

BDTTree - Interest-rate tree structure
binomial tree structure

Interest-rate tree specified as a structure by using bdttree.

Data Types
struct

OptSpec - Definition of option
string | cell array of strings

Definition of option as 'call' or 'put' specified as a NINST-by-1 cell
array of strings for 'call' or 'put'.

Data Types
cell | char

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values specified nonnegative integers using as
NINST-by-NSTRIKES vector of strike price values.

Data Types
single | double

ExerciseDates - Exercise date for option (European, Bermuda, or American)
nonnegative integer | vector of nonnegative integers

Exercise date for option (European, Bermuda, or American) specified as
nonnegative integers using a NINST-by-NSTRIKES or NINST-by-2 vector
of for the option exercise dates.

• If a European or Bermuda option, the ExerciseDates is a 1-by-1
  (European) or 1-by-NSTRIKES (Bermuda) vector of exercise dates. For

a European option, there is only one `ExerciseDate` on the option expiry date.

- If an American option, then `ExerciseDates` is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-`NaN` date, or if `ExerciseDates` is 1-by-1, the option exercises between the `Settle` date and the single listed `ExerciseDate`.

Data Types
`char` | `cell`

AmericanOpt - Option type
scalar | vector of positive integers`[0,1]`

Option type specified as `NINST`-by-1 positive integer scalar flags with values:

- `0` — European/Bermuda

- `1` — American

Data Types
`single` | `double`

Spread - Number of basis points over the reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate specified as a vector nonnegative integers for the number of instruments (`NINST`)-by-1).

Data Types
`single` | `double`

Settle - Settlement dates of floating-rate note
`ValuationDate` of HW Tree (default) | nonnegative integer | vector of nonnegative integers

Settlement dates of floating-rate note specified as nonnegative integers using a `NINST`-by-1 vector of dates.

Note The Settle date for every floating-rate note is set to the ValuationDate of the HW Tree. The floating-rate note argument Settle is ignored.

Data Types
char

Maturity - Floating-rate note maturity date
nonnegative integer | vector of nonnegative integers

Floating-rate note maturity date specified as nonnegative integers using a NINST-by-1 vector of dates.

Data Types
char

Reset - (Optional) Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the values [1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
double

optfloatbybdt

Basis - (Optional) Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

Data Types
double

Principal - (Optional) Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1
vector of notional principal amounts.

Data Types
double

Options - (Optional) Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using derivset.

Data Types
struct

EndMonthRule - (Optional) End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price=
optfloatbybdt(BDTTree,OptSpec,Strike,ExerciseDates,AmericanOpt,Spread,

'Reset' - Frequency of payments per year

optfloatbybdt

1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of
positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the
values 1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the
effective interest-rate between reset dates. If the reset period for a
FRN spans more than one tree level, calculating the payment becomes
impossible due to the recombining nature of the tree. That is, the tree
path connecting the two consecutive reset dates can not be uniquely
determined because there will be more than one possible path for
connecting the two payment dates.

Data Types
char

'Basis' - Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
char

'Principal' - Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1 vector of notional principal amounts.

Data Types
char

'Options' - Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using derivset.

Data Types
struct

'EndMonthRule' - End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as a nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

optfloatbybdt

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
char

<table><tr><td>**Output Arguments**</td><td>**Price - Expected prices of the floating-rate note option at time 0**
scalar | vector

Expected prices of the floating-rate note option at time 0 is returned as a scalar or an NINST-by-1 vector.

PriceTree - Structure of trees containing vectors of option prices at each node
tree structure

Structure of trees containing vectors of instrument prices and accrued interest and a vector of observation times for each node returned as:

- PriceTree.PTree contains option prices.</td></tr></table>

Examples

Compute the Price of American Call and Put Options on a Floating-Rate Note

Define the interest-rate term structure.

```
Rates = [0.03;0.034;0.038;0.04];
ValuationDate = 'Jan-1-2012';
StartDates = ValuationDate;
EndDates = {'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'; 'Jan-1-2016'};
Compounding = 1;
```

Create the RateSpec.

```
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding)

RateSpec =
```

```
          FinObj: 'RateSpec'
     Compounding: 1
            Disc: [4x1 double]
           Rates: [4x1 double]
        EndTimes: [4x1 double]
      StartTimes: [4x1 double]
        EndDates: [4x1 double]
      StartDates: 734869
   ValuationDate: 734869
           Basis: 0
     EndMonthRule: 1
```

Build the BDT tree and assume a volatility of 10%.

```
Sigma = 0.1;
BDTTimeSpec = bdttimespec(ValuationDate, EndDates);
BDTVolSpec = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
BDTT = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec)

BDTT =

      FinObj: 'BDTFwdTree'
     VolSpec: [1x1 struct]
    TimeSpec: [1x1 struct]
    RateSpec: [1x1 struct]
        tObs: [0 1 2 3]
        dObs: [734869 735235 735600 735965]
        TFwd: {[4x1 double]  [3x1 double]  [2x1 double]  [3]}
       CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
     FwdTree: {1x4 cell}
```

The floater instrument has a spread of 10, a period of one year, and matures on Jan-1-2016.

```
Spread = 10;
Settle = 'Jan-1-2012';
Maturity =  'Jan-1-2016';
```

```
Period = 1;
```

Define the option for the floating-rate note.

```
OptSpec = {'call'; 'put'};
Strike = [100;101];
ExerciseDates = 'Jan-1-2015';
AmericanOpt = 1;
```

Compute the price of the call and put options.

```
Price= optfloatbybdt(BDTT,  OptSpec, Strike, ExerciseDates,AmericanOpt, Spread,...
Settle, Maturity)

Price =

    0.3655
    0.8087
```

See Also   bdttree | cfbybdt | capbybdt | swapbybdt | floorbybdt |
floatbybdt | bondbybdt | instoptfloat

Concepts   • **basis** on page Glossary-1

Purpose   Price options on floating-rate notes for BK interest-rate tree

Syntax
```
Price = optfloatbybk(BKTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
Price = optfloatbybk(BKTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity,Reset,Basis,Principal,Options,EndMonthRule)
[Price,PriceTree] = optfloatbybk( ___ )

Price = optfloatbybk(BKTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
Price = optfloatbybk( ___ , Name,Value)
[Price,PriceTree] = optfloatbybk( ___ )
```

Description   `Price = optfloatbybk(BKTree,OptSpec,Strike,ExerciseDates,AmericanOpt, Spread,Settle,Maturity)` returns the option price of the instrument(s) at valuation date.

`Price = optfloatbybk(BKTree,OptSpec,Strike,ExerciseDates,AmericanOpt, Spread,Settle,Maturity,Reset,Basis,Principal,Options,EndMonthRule)` returns the option price of the instrument(s) at valuation date using optional arguments.

`[Price,PriceTree] = optfloatbybk( ___ )` returns the option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node.

`Price = optfloatbybk(BKTree,OptSpec,Strike,ExerciseDates,AmericanOpt, Spread,Settle,Maturity)` returns the option price of the instrument(s) at valuation date.

`Price = optfloatbybk( ___ , Name,Value)` returns the option price of the instrument(s) at valuation date using optional Name-Value pairs.

optfloatbybk

[Price,PriceTree] = optfloatbybk(___) returns the option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node using optional name-value pair arguments.

Input Arguments

BKTree - Interest-rate tree structure
binomial tree structure

Interest-rate tree specified as a structure by using `bktree`.

Data Types
struct

OptSpec - Definition of option
string | cell array of strings

Definition of option as `'call'` or `'put'` specified as a NINST-by-1 cell array of strings for `'call'` or `'put'`.

Data Types
cell | char

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values specified nonnegative integers using as NINST-by-NSTRIKES vector of strike price values.

Data Types
single | double

ExerciseDates - Exercise date for option (European, Bermuda, or American)
nonnegative integer | vector of nonnegative integers

Exercise date for option (European, Bermuda, or American) specified as nonnegative integers using a NINST-by-NSTRIKES or NINST-by-2 vector of for the option exercise dates.

- If a European or Bermuda option, the ExerciseDates is a 1-by-1 (European) or 1-by-NSTRIKES (Bermuda) vector of exercise dates. For

a European option, there is only one `ExerciseDate` on the option expiry date.

- If an American option, then `ExerciseDates` is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-`NaN` date, or if `ExerciseDates` is 1-by-1, the option exercises between the `Settle` date and the single listed `ExerciseDate`.

Data Types
char | cell

AmericanOpt - Option type
scalar | vector of positive integers[0,1]

Option type specified as `NINST`-by-1 positive integer scalar flags with values:

- 0 — European/Bermuda
- 1 — American

Data Types
single | double

Spread - Number of basis points over the reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate specified as a vector nonnegative integers for the number of instruments (`NINST`)-by-1).

Data Types
single | double

Settle - Settlement dates of floating-rate note
`ValuationDate` of HW Tree (default) | nonnegative integer | vector of nonnegative integers

Settlement dates of floating-rate note specified as nonnegative integers using a `NINST`-by-1 vector of dates.

optfloatbybk

Note The `Settle` date for every floating-rate note is set to the
`ValuationDate` of the HW Tree. The floating-rate note argument
`Settle` is ignored.

Data Types
char

Maturity - Floating-rate note maturity date
nonnegative integer | vector of nonnegative integers

Floating-rate note maturity date specified as nonnegative integers
using a `NINST`-by-1 vector of dates.

Data Types
char

Reset - (Optional) Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of
positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the
values [1,2,3,4,6,12] in a `NINST`-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the
effective interest-rate between reset dates. If the reset period for a
FRN spans more than one tree level, calculating the payment becomes
impossible due to the recombining nature of the tree. That is, the tree
path connecting the two consecutive reset dates can not be uniquely
determined because there will be more than one possible path for
connecting the two payment dates.

Data Types
double

Basis - (Optional) Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
double

Principal - (Optional) Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1
vector of notional principal amounts.

Data Types
double

Options - (Optional) Structure containing derivatives pricing options

structure

Structure containing derivatives pricing options specified using derivset.

Data Types
struct

EndMonthRule - (Optional) End-of-month rule flag

1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price= optfloatbybk(BKTTree,OptSpec,Strike,ExerciseDates,AmericanOpt,Spread,Sett

'Reset' - Frequency of payments per year

1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the values 1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
char

'Basis' - Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] | vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using a NINST-by-1 vector. The Basis value represents the basis used when annualizing the input forward-rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)

optfloatbybk

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
char

'Principal' - Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1
vector of notional principal amounts.

Data Types
char

'Options' - Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using
derivset.

Data Types
struct

'EndMonthRule' - End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as a nonnegative integer [0, 1]
using a NINST-by-1 vector. This rule applies only when Maturity is an
end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always
  the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
char

Output Arguments

Price - Expected prices of the floating-rate note option at time 0
scalar | vector

Expected prices of the floating-rate note option at time 0 is returned as a scalar or an NINST-by-1 vector.

PriceTree - Structure of trees containing vectors of option prices at each node
tree structure

Structure of trees containing vectors of instrument prices and accrued interest and a vector of observation times for each node returned as:

- PriceTree.PTree contains option prices.

Examples

Compute the Price of American and European Call Options on a Floating-Rate Note

Define the interest-rate term structure.

```
Rates = [0.03;0.034;0.038;0.04];
ValuationDate = 'Jan-1-2012';
StartDates = ValuationDate;
EndDates = {'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'; 'Jan-1-2016'};
Compounding = 1;
```

Create the RateSpec.

```
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding)

RateSpec =
```

```
            FinObj: 'RateSpec'
       Compounding: 1
              Disc: [4x1 double]
             Rates: [4x1 double]
          EndTimes: [4x1 double]
        StartTimes: [4x1 double]
          EndDates: [4x1 double]
        StartDates: 734869
     ValuationDate: 734869
             Basis: 0
       EndMonthRule: 1
```

Build the BK tree.

```
VolDates = ['1-Jan-2013'; '1-Jan-2014'; '1-Jan-2015';'1-Jan-2016'];
VolCurve = 0.01;
AlphaDates = '01-01-2016';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
          AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec)


HWT =

       FinObj: 'HWFwdTree'
      VolSpec: [1x1 struct]
     TimeSpec: [1x1 struct]
     RateSpec: [1x1 struct]
         tObs: [0 1 2 3]
         dObs: [734869 735235 735600 735965]
       CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
        Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
      Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
      FwdTree: {[1.0300]  [1.0553 1.0381 1.0211]  [1x5 double]  [1x5 doubl
```

The floater instrument has a spread of 10, a period of one year, and matures on Jan-1-2016.

```
Spread = 10;
Settle = 'Jan-1-2012';
Maturity =  'Jan-1-2016';
Period = 1;
```

Define the option for the floating-rate note.

```
OptSpec = {'call'};
Strike = 95;
ExerciseDates = 'Jan-1-2016';
AmericanOpt = [0;1];
```

Compute the price of the call options.

```
Price= optfloatbyhw(HWT,  OptSpec, Strike, ExerciseDates,AmericanOpt,...
Spread, Settle, Maturity)

Price =

    4.2740
    5.3655
```

See Also   bktree | cfbybk | capbybk | swapbybk | floorbybk |
floatbybk | bondbybk | instoptfloat

Concepts   • **basis** on page Glossary-1

optfloatbyhjm

Purpose	Price options on floating-rate notes for HJM interest-rate tree

Syntax

```
Price = optfloatbyhjm(HJMTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
Price = optfloatbyhjm(HJMTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity,Reset,Basis,Principal,Options,EndMonthRule)
[Price,PriceTree] = optfloatbyhjm( ___ )

Price = optfloatbyhjm(HJMTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
Price = optfloatbyhjm( ___ , Name,Value)
[Price,PriceTree] = optfloatbyhjm( ___ )
```

Description

```
Price =
optfloatbyhjm(HJMTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
```
returns the option price of the instrument(s) at valuation date.

```
Price =
optfloatbyhjm(HJMTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity,Reset,Basis,Principal,Options,EndMonthRule)
```
returns the option price of the instrument(s) at valuation date using optional arguments.

```
[Price,PriceTree] = optfloatbyhjm( ___ )
```
returns the option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node.

```
Price =
optfloatbyhjm(HJMTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
```
returns the option price of the instrument(s) at valuation date.

```
Price = optfloatbyhjm( ___ , Name,Value)
```
returns the option price of the instrument(s) at valuation date using optional Name-Value pairs.

[Price,PriceTree] = optfloatbyhjm(___) returns the option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node using optional name-value pair arguments.

Input Arguments

HJMTree - Interest-rate tree structure
binomial tree structure

Interest-rate tree specified as a structure by using hjmtree.

Data Types
struct

OptSpec - Definition of option
string | cell array of strings

Definition of option as 'call' or 'put' specified as a NINST-by-1 cell array of strings for 'call' or 'put'.

Data Types
cell | char

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values specified nonnegative integers using as NINST-by-NSTRIKES vector of strike price values.

Data Types
single | double

ExerciseDates - Exercise date for option (European, Bermuda, or American)
nonnegative integer | vector of nonnegative integers

Exercise date for option (European, Bermuda, or American) specified as nonnegative integers using a NINST-by-NSTRIKES or NINST-by-2 vector of for the option exercise dates.

- If a European or Bermuda option, the ExerciseDates is a 1-by-1 (European) or 1-by-NSTRIKES (Bermuda) vector of exercise dates. For

a European option, there is only one `ExerciseDate` on the option expiry date.

- If an American option, then `ExerciseDates` is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-`NaN` date, or if `ExerciseDates` is 1-by-1, the option exercises between the `Settle` date and the single listed `ExerciseDate`.

Data Types
char | cell

AmericanOpt - Option type
scalar | vector of positive integers[0,1]

Option type specified as `NINST`-by-1 positive integer scalar flags with values:

- 0 — European/Bermuda

- 1 — American

Data Types
single | double

Spread - Number of basis points over the reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate specified as a vector nonnegative integers for the number of instruments (NINST)-by-1).

Data Types
single | double

Settle - Settlement dates of floating-rate note
`ValuationDate` of HW Tree (default) | nonnegative integer | vector of nonnegative integers

Settlement dates of floating-rate note specified as nonnegative integers using a `NINST`-by-1 vector of dates.

> **Note** The Settle date for every floating-rate note is set to the ValuationDate of the HW Tree. The floating-rate note argument Settle is ignored.

Data Types
char

Maturity - Floating-rate note maturity date
nonnegative integer | vector of nonnegative integers

Floating-rate note maturity date specified as nonnegative integers using a NINST-by-1 vector of dates.

Data Types
char

Reset - (Optional) Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the values [1,2,3,4,6,12] in a NINST-by-1 vector.

> **Note** Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
double

Basis - (Optional) Day-count basis of the instrument

0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
double

Principal - (Optional) Principal values

100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1
vector of notional principal amounts.

Data Types
double

Options - (Optional) Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using
derivset.

Data Types
struct

EndMonthRule - (Optional) End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as nonnegative integer [0, 1] using
a NINST-by-1 vector. This rule applies only when Maturity is an
end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always
  the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always
  the last actual day of the month.

Data Types
double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments.
Name is the argument name and Value is the corresponding
value. Name must appear inside single quotes (' '). You can
specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

Example: Price=
optfloatbyhjm(HJMTree,OptSpec,Strike,ExerciseDates,AmericanOpt,Spread,

'Reset' - Frequency of payments per year

1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the values [1,2,3,4,6,12] in a NINST-by-1 vector. .

Note Payments on floating-rate notes (FRNs) are determined by the effective interest-rate between reset dates. If the reset period for a FRN spans more than one tree level, calculating the payment becomes impossible due to the recombining nature of the tree. That is, the tree path connecting the two consecutive reset dates can not be uniquely determined because there will be more than one possible path for connecting the two payment dates.

Data Types
char

'Basis' - Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] | vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using a NINST-by-1 vector. The Basis value represents the basis used when annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
char

'Principal' - Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1
vector of notional principal amounts.

Data Types
char

'Options' - Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using
derivset.

Data Types
struct

'EndMonthRule' - End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as a nonnegative integer [0, 1]
using a NINST-by-1 vector. This rule applies only when Maturity is an
end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always
  the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
char

Output Arguments

Price - Expected prices of the floating-rate note option at time 0
scalar | vector

Expected prices of the floating-rate note option at time 0 is returned as a scalar or an NINST-by-1 vector.

PriceTree - Structure of trees containing vectors of option prices at each node
tree structure

Structure of trees containing vectors of instrument prices and accrued interest and a vector of observation times for each node returned as:

- PriceTree.PTree contains option prices.

Examples

Compute the Price of American and European Call Options on a Floating-Rate Note

Define the interest-rate term structure.

```
Rates = [0.03;0.035;0.040;0.045];
ValuationDate = 'Jan-1-2012';
StartDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
EndDates = {'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'; 'Jan-1-2016'};
Compounding = 1;
```

Create the RateSpec.

```
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding)

RateSpec =
```

```
         FinObj: 'RateSpec'
    Compounding: 1
           Disc: [4x1 double]
          Rates: [4x1 double]
       EndTimes: [4x1 double]
     StartTimes: [4x1 double]
       EndDates: [4x1 double]
     StartDates: [4x1 double]
  ValuationDate: 734869
          Basis: 0
    EndMonthRule: 1
```

Build the HJM tree.

```
VolSpec = hjmvolspec('Constant', 0.01);
TimeSpec = hjmtimespec(RateSpec.ValuationDate, EndDates, Compounding);
HJMTree = hjmtree(VolSpec, RateSpec, TimeSpec)


HJMTree =

     FinObj: 'HJMFwdTree'
    VolSpec: [1x1 struct]
   TimeSpec: [1x1 struct]
   RateSpec: [1x1 struct]
       tObs: [0 1 2 3]
       dObs: [734869 735235 735600 735965]
       TFwd: {[4x1 double]  [3x1 double]  [2x1 double]  [3]}
     CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
    FwdTree: {[4x1 double]  [3x1x2 double]  [2x2x2 double]  [1x4x2 double]}
```

The floater instrument has a spread of 10, a period of one year, and matures on Jan-1-2015.

```
Spread = 10;
Settle = 'Jan-1-2012';
Maturity =  'Jan-1-2015';
Period = 1;
```

optfloatbyhjm

Define the option for the floating-rate note.

```
OptSpec = {'call'};
Strike = 95;
ExerciseDates = 'Jan-1-2015';
AmericanOpt = [0;1];
```

Compute the price of the call options.

```
Price= optfloatbyhjm(HJMTree,  OptSpec, Strike, ExerciseDates,AmericanOpt,...
Spread, Settle, Maturity)

Price =

    4.5098
    5.2811
```

See Also   hjmtree | cfbyhjm | capbyhjm | swapbyhjm | floorbyhw | floatbyhjm | bondbyhjm | instoptfloat

Concepts   • **basis** on page Glossary-1

Purpose	Price options on floating-rate notes for HW interest-rate tree

Syntax

```
Price = optfloatbyhw(HWTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
Price= optfloatbyhw(HWTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity,Reset,Basis,Principal,Options,EndMonthRule)
[Price,PriceTree] = optfloatbyhw( ___ )

Price = optfloatbyhw(HWTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
Price = optfloatbyhw( ___ , Name,Value)
[Price,PriceTree] = optfloatbyhw( ___ )
```

Description

```
Price =
optfloatbyhw(HWTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
```
returns the option price of the instrument(s) at valuation date.

```
Price=
optfloatbyhw(HWTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity,Reset,Basis,Principal,Options,EndMonthRule)
```
returns the option price of the instrument(s) at valuation date using optional arguments.

```
[Price,PriceTree] = optfloatbyhw( ___ )
```
returns the option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node.

```
Price =
optfloatbyhw(HWTree,OptSpec,Strike,ExerciseDates,AmericanOpt,
Spread,Settle,Maturity)
```
returns the option price of the instrument(s) at valuation date.

```
Price = optfloatbyhw( ___ , Name,Value)
```
returns the option price of the instrument(s) at valuation date using optional Name-Value pairs.

optfloatbyhw

[Price,PriceTree] = optfloatbyhw(___) returns the option price of the instrument(s) at valuation date and a structure of trees containing vectors of option prices at each node using optional Name-Value pairs.

Input Arguments

HWTree - Interest-rate tree structure
binomial tree structure

Interest-rate tree specified as a structure by using `hwtree`.

Data Types
struct

OptSpec - Definition of option
string | cell array of strings

Definition of option as `'call'` or `'put'` specified as a NINST-by-1 cell array of strings for `'call'` or `'put'`.

Data Types
cell | char

Strike - Option strike price values
nonnegative integer | vector of nonnegative integers

Option strike price values specified nonnegative integers using as NINST-by-NSTRIKES vector of strike price values.

Data Types
single | double

ExerciseDates - Exercise date for option (European, Bermuda, or American)
nonnegative integer | vector of nonnegative integers

Exercise date for option (European, Bermuda, or American) specified as nonnegative integers using a NINST-by-NSTRIKES or NINST-by-2 vector of for the option exercise dates.

- If a European or Bermuda option, the ExerciseDates is a 1-by-1 (European) or 1-by-NSTRIKES (Bermuda) vector of exercise dates. For

a European option, there is only one `ExerciseDate` on the option expiry date.

- If an American option, then `ExerciseDates` is a 1-by-2 vector of exercise date boundaries. The option exercises on any date between or including the pair of dates on that row. If there is only one non-`NaN` date, or if `ExerciseDates` is 1-by-1, the option exercises between the `Settle` date and the single listed `ExerciseDate`.

Data Types
char | cell

AmericanOpt - Option type
scalar | vector of positive integers[0,1]

Option type specified as `NINST`-by-1 positive integer scalar flags with values:

- `0` — European/Bermuda
- `1` — American

Data Types
single | double

Spread - Number of basis points over the reference rate
nonnegative integer | vector of nonnegative integers

Number of basis points over the reference rate specified as a vector nonnegative integers for the number of instruments (`NINST`)-by-1).

Data Types
single | double

Settle - Settlement dates of floating-rate note
`ValuationDate` of HW Tree (default) | nonnegative integer | vector of nonnegative integers

Settlement dates of floating-rate note specified as nonnegative integers using a `NINST`-by-1 vector of dates.

> **Note** The `Settle` date for every floating-rate note is set to the
> `ValuationDate` of the HW Tree. The floating-rate note argument
> `Settle` is ignored.

Data Types
char

Maturity - Floating-rate note maturity date
nonnegative integer | vector of nonnegative integers

Floating-rate note maturity date specified as nonnegative integers
using a `NINST`-by-1 vector of dates.

Data Types
char

Reset - (Optional) Frequency of payments per year
1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of
positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the
values [1,2,3,4,6,12] in a `NINST`-by-1 vector.

> **Note** Payments on floating-rate notes (FRNs) are determined by the
> effective interest-rate between reset dates. If the reset period for a
> FRN spans more than one tree level, calculating the payment becomes
> impossible due to the recombining nature of the tree. That is, the tree
> path connecting the two consecutive reset dates can not be uniquely
> determined because there will be more than one possible path for
> connecting the two payment dates.

Data Types
double

Basis - (Optional) Day-count basis of the instrument

0 (actual/actual) (default) | positive integers of the set [1...13] | vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using a NINST-by-1 vector. The Basis value represents the basis used when annualizing the input forward-rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

Data Types
double

Principal - (Optional) Principal values

100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1 vector of notional principal amounts.

optfloatbyhw

Data Types
double

Options - (Optional) Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using derivset.

Data Types
struct

EndMonthRule - (Optional) End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price= optfloatbyhw(HWTree,OptSpec,Strike,ExerciseDates,AmericanOpt,Spread,Settl

'Reset' - Frequency of payments per year

1 (default) | positive integer from the set[1,2,3,4,6,12] | vector of
positive integers from the set [1,2,3,4,6,12]

Frequency of payments per year specified as positive integers for the
values [1,2,3,4,6,12] in a NINST-by-1 vector.

Note Payments on floating-rate notes (FRNs) are determined by the
effective interest-rate between reset dates. If the reset period for a
FRN spans more than one tree level, calculating the payment becomes
impossible due to the recombining nature of the tree. That is, the tree
path connecting the two consecutive reset dates can not be uniquely
determined because there will be more than one possible path for
connecting the two payment dates.

Data Types
char

'Basis' - Day-count basis of the instrument
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Day-count basis of the instrument specified as a positive integer using
a NINST-by-1 vector. The Basis value represents the basis used when
annualizing the input forward-rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
char

'Principal' - Principal values
100 (default) | nonnegative integer | vector of nonnegative integers

Principal values specified as a nonnegative integer using a NINST-by-1 vector of notional principal amounts.

Data Types
char

'Options' - Structure containing derivatives pricing options
structure

Structure containing derivatives pricing options specified using derivset.

Data Types
struct

'EndMonthRule' - End-of-month rule flag
1 (in effect) (default) | nonnegative integer [0,1]

End-of-month rule flag is specified as a nonnegative integer [0, 1] using a NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

• 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Data Types
char

Output Arguments

Price - Expected prices of the floating-rate note option at time 0
scalar | vector

Expected prices of the floating-rate note option at time 0 is returned as a scalar or an NINST-by-1 vector.

PriceTree - Structure of trees containing vectors of option prices at each node
tree structure

Structure of trees containing vectors of instrument prices and accrued interest and a vector of observation times for each node returned as:

• PriceTree.PTree contains option prices.

Examples

Compute the Price of American and European Call Options on a Floating-Rate Note

Define the interest-rate term structure.

```
Rates = [0.03;0.034;0.038;0.04];
ValuationDate = 'Jan-1-2012';
StartDates = ValuationDate;
EndDates = {'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'; 'Jan-1-2016'};
Compounding = 1;
```

Create the RateSpec.

```
RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding)

RateSpec =
```

```
            FinObj: 'RateSpec'
       Compounding: 1
              Disc: [4x1 double]
             Rates: [4x1 double]
          EndTimes: [4x1 double]
        StartTimes: [4x1 double]
          EndDates: [4x1 double]
        StartDates: 734869
     ValuationDate: 734869
             Basis: 0
       EndMonthRule: 1
```

Build the HW tree using the following:

```
VolDates = ['1-Jan-2013'; '1-Jan-2014'; '1-Jan-2015';'1-Jan-2016'];
VolCurve = 0.01;
AlphaDates = '01-01-2016';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
          AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec)

HWT =

      FinObj: 'HWFwdTree'
      VolSpec: [1x1 struct]
     TimeSpec: [1x1 struct]
     RateSpec: [1x1 struct]
         tObs: [0 1 2 3]
         dObs: [734869 735235 735600 735965]
        CFlowT: {[4x1 double]  [3x1 double]  [2x1 double]  [4]}
        Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
      Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
      FwdTree: {[1.0300]  [1.0553 1.0381 1.0211]  [1x5 double]  [1x5 double]}
```

The floater instrument has a spread of 10, a period of one year, and matures on Jan-1-2016.

```
Spread = 10;
Settle = 'Jan-1-2012';
Maturity =  'Jan-1-2016';
Period = 1;
```

Define the option for the floating-rate note.

```
OptSpec = {'call'};
Strike = 95;
ExerciseDates = 'Jan-1-2016';
AmericanOpt = [0;1];
```

Compute the price of the call options.

```
Price= optfloatbyhw(HWT,  OptSpec, Strike, ExerciseDates,AmericanOpt,...
Spread, Settle, Maturity)

Price =

    4.2740
    5.3655
```

See Also   hwtree | cfbyhw | capbyhw | swapbyhw | floorbyhw |
floatbyhw | bondbyhw | instoptfloat

Concepts   • **basis** on page Glossary-1

optstockbybjs

Purpose   Price American options using Bjerksund-Stensland 2002 option pricing model

Syntax   Price = optstockbybjs(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.

Description Price = optstockbybjs(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike) computes American option prices with continuous dividend yield using the Bjerksund-Stensland 2002 option pricing model.

Price is a NINST-by-1 vector of expected option prices.

Note optstockbybjs computes prices of American options with continuous dividend yield using the Bjerksund-Stensland option pricing model.

Examples   Consider two American stock options (a call and a put) with an exercise price of $100. The options expire on April 1, 2008. Assume the

underlying stock pays a continuous dividend yield of 4% as of January 1, 2008. The stock has a volatility of 20% per annum and the annualized continuously compounded risk-free rate is 8% per annum. Using this data, calculate the price of the American call and put, assuming the following current prices of the stock: $90 (for the call) and $120 (for the put):

```
Settle = 'Jan-1-2008';
Maturity = 'April-1-2008';
Strike = 100;
AssetPrice = [90;120];
DivYield = 0.04;
Rate = 0.08;
Sigma = 0.20;
```

Define `StockSpec` and `RateSpec`:

```
StockSpec = stockspec(Sigma, AssetPrice, {'continuous'}, DivYield);

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rate, 'Compounding', -1);
```

Define the option type:

```
OptSpec = {'call'; 'put'};
```

Compute the option prices using the Bjerksund-Stensland 2002 option pricing model:

```
Price = optstockbybjs(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)

Price =

    0.8420
    0.1108
```

The first element of the `Price` vector represents the price of the call ($0.84); the second element represents the price of the put option ($0.11).

optstockbybjs

References   Bjerksund, P. and G. Stensland, *Closed-Form Approximation of American Options*, Scandinavian Journal of Management, 1993, Vol. 9, Suppl., pp. S88-S99.

Bjerksund, P. and G. Stensland, *Closed Form Valuation of American Options*, Discussion paper 2002 (http://brage.bibsys.no/nhh/bitstream/URN:NBN:no-bibsys_brage_22301/1/bjerksund%2

See Also   impvbybjs | intenvset | optstocksensbybjs | stockspec

Purpose   Price options on futures using Black option pricing model

Syntax   Price = optstockbyblk(RateSpec, StockSpec, Settle, Maturity,
                  OptSpec, Strike)

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.

Description   Price = optstockbyblk(RateSpec, StockSpec, Settle,
                  Maturity,
                  OptSpec, Strike) computes option prices on futures using the Black
                  option pricing model.

                  Price is a NINST-by-1 vector of expected option prices.

Examples   Consider two European call options on a futures contract with exercise
                  prices of $20 and $25 that expire on September 1, 2008. Assume that
                  on May 1, 2008 the contract is trading at $20, and has a volatility of
                  35% per annum. The risk-free rate is 4% per annum. Using this data,
                  calculate the price of the call futures options using the Black model:

```
Strike = [20; 25];
AssetPrice = 20;
Sigma = .35;
Rates = 0.04;
```

optstockbyblk

```
Settle = 'May-01-08';
Maturity = 'Sep-01-08';
```

Create `RateSpec` and `StockSpec`:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
 'EndDates', Maturity, 'Rates', Rates, 'Compounding', -1);
```

```
StockSpec = stockspec(Sigma, AssetPrice);
```

Define the call options:

```
OptSpec = {'call'};
```

Calculate the price using the Black option pricing model:

```
Price = optstockbyblk(RateSpec, StockSpec, Settle, Maturity,...
OptSpec, Strike)

Price =
    1.5903
    0.3037
```

See Also   impvbyblk | intenvset | optstocksensbyblk | stockspec

Purpose	Price options using Black-Scholes option pricing model
Syntax	`Price = optstockbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)`

Arguments

`RateSpec`	The annualized continuously compounded rate term structure. For information on the interest rate specification, see `intenvset`.
`StockSpec`	Stock specification. See `stockspec`.
`Settle`	NINST-by-1 vector of settlement or trade dates.
`Maturity`	NINST-by-1 vector of maturity dates.
`OptSpec`	NINST-by-1 cell array of strings `'call'` or `'put'`.
`Strike`	NINST-by-1 vector of strike price values.

Description `Price = optstockbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)` computes option prices using the Black-Scholes option pricing model.

`Price` is a NINST-by-1 vector of expected option prices.

optstockbybls

Examples Consider two European options, a call and a put, with an exercise price of $29 on January 1, 2008. The options expire on May 1, 2008. Assume that the underlying stock for the call option provides a cash dividend of $0.50 on February 15, 2008. The underlying stock for the put option provides a continuous dividend yield of 4.5% per annum. The stocks are trading at $30 and have a volatility of 25% per annum. The annualized continuously compounded risk-free rate is 5% per annum. Using this data, compute the price of the options using the Black-Scholes model:

```
Strike = 29;
AssetPrice = 30;
Sigma = .25;
Rates = 0.05;
Settle = 'Jan-01-2008';
Maturity = 'May-01-2008';
```

Define `RateSpec`:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, 'EndDates',...
Maturity, 'Rates', Rates, 'Compounding', -1);
```

Define `StockSpec`:

```
DividendType = {'cash';'continuous'};
DividendAmounts = [0.50; 0.045];
ExDividendDates = {'Feb-15-2008';NaN};

StockSpec = stockspec(Sigma, AssetPrice, DividendType, DividendAmounts,...
ExDividendDates);
```

Price the call and the put options using the Black-Scholes model:

```
OptSpec = {'call'; 'put'};

Price = optstockbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike)

Price =

    2.2030
    1.2025
```

See Also   impvbybls | intenvset | optstocksensbybls | stockspec

optstockbycrr

Purpose	Price stock option from Cox-Ross-Rubinstein tree

Syntax
```
[Price, PriceTree] = optstockbycrr(CRRTree, OptSpec, Strike,
Settle, ExerciseDates, AmericanOpt)
```

Arguments

CRRTree	Stock tree structure created by crrtree.
OptSpec	Number of instruments (NINST)-by-1 cell array of strings 'call' or 'put'.

Note The interpretation of the Strike and ExerciseDates arguments depends upon the setting of the AmericanOpt argument. If AmericanOpt = 0, NaN, or is unspecified, the option is a European or Bermuda option. If AmericanOpt = 1, the option is an American option.

Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	American option: NINST-by-1 vector of strike price values for each option.

Settle	NINST-by-1 vector of settlement or trade dates.
ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.

Data arguments are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Description   [Price, PriceTree] = optstockbycrr(CRRTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt) computes the price of a European, Bermuda, or American stock option.

Price is a NINST-by-1 vector of expected option prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples   Price a stock option using a CRR binomial tree.

Load the file deriv.mat, which provides CRRTree. The CRRTree structure contains the stock specification and time information needed to price the option.

optstockbycrr

```
load deriv.mat;
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'Call';
Strike = 105;
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2005';

Price = optstockbycrr(CRRTree, OptSpec, Strike, Settle, ...
ExerciseDates)

Price =

 8.2863
```

See Also   crrtree | instoptstock

Purpose	Price stock option from Equal Probabilities binomial tree

Syntax   [Price, PriceTree] = optstockbyeqp(EQPTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt)

Arguments

EQPTree	Stock tree structure created by eqptree.
OptSpec	Number of instruments (NINST)-by-1 cell array of strings 'call' or 'put'.

Note The interpretation of the Strike and ExerciseDates arguments depends upon the setting of the AmericanOpt argument. If AmericanOpt = 0, NaN, or is unspecified, the option is a European or Bermuda option. If AmericanOpt = 1, the option is an American option.

Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	American option: NINST-by-1 vector of strike price values for each option.

optstockbyeqp

Settle	NINST-by-1 vector of settlement or trade dates.
ExerciseDates	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date. For an American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.

Data arguments are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument. The others may be omitted or passed as empty matrices [].

Description   [Price, PriceTree] = optstockbyeqp(EQPTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt) computes the price of a European/Bermuda or American stock option.

Price is a NINST-by-1 vector of expected option prices at time 0.

PriceTree is a tree structure with a vector of instrument prices at each node.

Examples   Price a stock option using an EQP equity tree.

Load the file deriv.mat, which provides EQPTree. The EQPTree structure contains the stock specification and time information needed to price the option.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'Call';
Strike = 105;
Settle = '01-Jan-2003';
ExerciseDates = '01-Jan-2005';

Price = optstockbyeqp(EQPTree, OptSpec, Strike, Settle, ...
ExerciseDates)

Price =

 8.4791
```

See Also   eqptree | instoptstock

optstockbyitt

Purpose   Price options on stocks using implied trinomial tree (ITT)

Syntax   [Price, PriceTree] = optstockbyitt(ITTTree, OptSpec, Strike,
                 Settle, ExerciseDates, AmericanOpt)

Arguments

ITTTree	Stock tree structure created by `itttree`.
OptSpec	Number of instruments (NINST)-by-1 cell array of strings `'call'` or `'put'`.

> **Note** The interpretation of the Strike and ExerciseDates arguments depends on the setting of the AmericanOpt argument. If AmericanOpt = 0, NaN, or is unspecified, the option is a European or Bermuda option. If AmericanOpt = 1, the option is an American option.

Strike	European option: NINST-by-1 vector of strike price values.
	Bermuda option: NINST by number of strikes (NSTRIKES) matrix of strike price values.
	Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.
	American option: NINST-by-1 vector of strike price values for each option.
Settle	NINST-by-1 vector of settlement or trade dates.

11-870

ExerciseDates	For a European or Bermuda option:
	NINST-by-1 (European option) or NINST-by-NSTRIKES (Bermuda option) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one exercise date, the option expiry date.
	For an American option:
	NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying bond Settle and the single listed exercise date.
AmericanOpt	(Optional) If AmericanOpt = 0, NaN, or is unspecified, the option is a European or Bermuda option. If AmericanOpt = 1, the option is an American option.

Note Data arguments for optstockbyitt are NINST-by-1 vectors, scalar, or empty. Fill unspecified entries in vectors with NaN. Only one data argument is required to create the instrument; the others may be omitted or passed as empty matrices [].

Description   [Price, PriceTree] = optstockbyitt(ITTTree, OptSpec, Strike, Settle, ExerciseDates, AmericanOpt) computes the price of a European/Bermuda or American stock option.

Price is a NINST-by-1 vector of expected option prices at time 0.

optstockbyitt

PriceTree is a tree structure with a vector of instrument prices at each node.

Note The Settle date for every option is set to the ValuationDate of the stock tree. The option argument, Settle, is ignored.

Examples Price a stock option using an ITT equity tree.

Load the file deriv.mat which provides the ITTTree. The ITTTree structure contains the stock specification and time information needed to price the option.

```
load deriv.mat
```

Set the required values. Other arguments will use defaults.

```
OptSpec = 'Put';
Strike = 80;
Settle = '01-Jan-2006';
ExerciseDates = ' 01-Jan-2010 ';
```

```
Price = optstockbyitt(ITTTree, OptSpec, Strike, Settle,ExerciseDates)

Price =

 10.68
```

References Chriss, Neil A., E. Derman, and I. Kani, "Implied trinomial trees of the volatility smile," *Journal of Derivatives*, 1996.

See Also instoptstock | itttree | stockoptspec

11-872

Purpose	Price options on stocks using Leisen-Reimer binomial tree model

Syntax
```
[Price, PriceTree] = optstockbylr(LRTree, OptSpec, Strike,
Settle, ExerciseDates)
[Price, PriceTree] = optstockbylr(LRTree, OptSpec, Strike,
Settle, ExerciseDates, Name,Value)
```

Description [Price, PriceTree] = optstockbylr(LRTree, OptSpec, Strike, Settle, ExerciseDates) computes option prices on stocks using the Leisen-Reimer binomial tree model.

[Price, PriceTree] = optstockbylr(LRTree, OptSpec, Strike, Settle, ExerciseDates, Name,Value) computes option prices on stocks using the Leisen-Reimer binomial tree model with additional options specified by one or more Name,Value pair arguments.

Input Arguments

LRTree

Stock tree structure created by lrtree.

OptSpec

NINST-by-1 cell array of strings 'call' or 'put'.

Strike

NINST-by-1 (European/American) or NINST-by-NSTRIKES (Bermuda) matrix of strike price values. Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.

Settle

NINST-by-1 matrix of settlement or trade dates.

Note The settle date for every option is set to the ValuationDate of the stock tree. The option argument, Settle, is ignored.

ExerciseDates

NINST-by-1 (European/American) or NINST-by-NSTRIKEDATES (Bermuda) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one ExerciseDate on the option expiry date. For the American type, the option can be exercised on any tree data between the ValuationDate and tree maturity. The last element of each row must be the same as the maturity of the tree.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'AmericanOpt'

NINST-by-1 flags with a value of 0 (European/Bermuda) or 1 (American).

> **Default:** 0

Output Arguments

Price

NINST-by-1 expected prices at time 0.

PriceTree

Tree structure with a vector of instrument prices at each node.

Examples

Consider European call and put options with an exercise price of $95 that expire on July 1, 2010. The underlying stock is trading at $100 on January 1, 2010, provides a continuous dividend yield of 3% per annum and has a volatility of 20% per annum. The annualized continuously compounded risk-free rate is 8% per annum. Using this data, compute the price of the options using the Leisen-Reimer model with a tree of 15 and 55 time steps.

```
AssetPrice  = 100;
```

```
Strike = 95;

ValuationDate = 'Jan-1-2010';
Maturity = 'July-1-2010';

% Define StockSpec
Sigma = 0.2;
DividendType = 'continuous';
DividendAmounts = 0.03;

StockSpec = stockspec(Sigma, AssetPrice, DividendType, DividendAmounts);


% Define RateSpec
Rates = 0.08;
Settle = ValuationDate;
Basis = 1;
Compounding = -1;

RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', Basis);

% Build the Leisen-Reimer (LR) tree with 15 and 55 time steps
LRTimeSpec15  = lrtimespec(ValuationDate, Maturity, 15);
LRTimeSpec55  = lrtimespec(ValuationDate, Maturity, 55);

% Use the PP2 method
LRMethod  = 'PP2';

LRTree15 = lrtree(StockSpec, RateSpec, LRTimeSpec15, Strike, 'method', LRMethod);
LRTree55 = lrtree(StockSpec, RateSpec, LRTimeSpec55, Strike, 'method', LRMethod);


% Price the call and the put options using the LR model:
OptSpec = {'call'; 'put'};

PriceLR15 = optstockbylr(LRTree15, OptSpec, Strike, Settle, Maturity);
```

```
PriceLR55 = optstockbylr(LRTree55, OptSpec, Strike, Settle, Maturity);

% Calculate price using the Black-Scholes model (BLS) to compare values with
% the LR model:
PriceBLS = optstockbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike);

% Compare values of BLS and LR
[PriceBLS PriceLR15 PriceLR55]

ans =

    9.7258    9.7252    9.7257
    2.4896    2.4890    2.4895
```

Use treeviewer to display the LRTree of 15 time steps:

```
treeviewer(LRTree15)
```
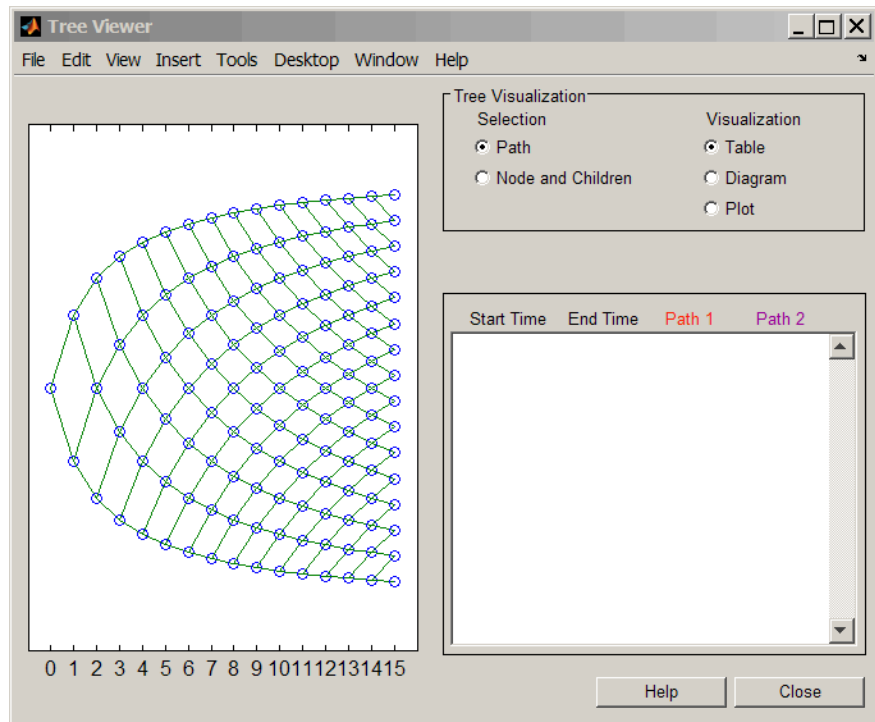
References   Leisen D.P., M. Reimer, "Binomial Models for Option Valuation – Examining and Improving Convergence," *Applied Mathematical Finance*, Number 3, 1996, pp. 319-346.

See Also | instoptstock | lrtree | optstocksensbylr

optstockbyls

Purpose	Price European, Bermudan, or American vanilla options using Longstaff-Schwartz model

Syntax	`Price = optstockbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,ExerciseDates)`
	`Price = optstockbyls( ___ ,Name,Value)`
	
	`[Price,Path,Times,Z] = optstockbyls(RateSpec,StockSpec,OptSpec,StrikeSett`
	`ExerciseDates)`
	`[Price,Path,Times,Z] = optstockbyls( ___ ,Name,Value)`

Description	`Price = optstockbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,ExerciseDates)` returns vanilla option prices using the Longstaff-Schwartz model. `optstockbyls` computes prices of European, Bermudan, and American vanilla options. For American and Bermudan options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium. `Price = optstockbyls( ___ ,Name,Value)` returns vanilla option prices using the Longstaff-Schwartz model using optional name-value pair arguments. `optstockbyls` computes prices of European, Bermudan, and American vanilla options. For American and Bermudan options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium. `[Price,Path,Times,Z] = optstockbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,ExerciseDates)` returns vanilla option prices using the Longstaff-Schwartz model. `optstockbyls` computes prices of European, Bermudan, and American vanilla options. For American and Bermudan options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium.

`[Price,Path,Times,Z] = optstockbyls( ___ ,Name,Value)` returns vanilla option prices using the Longstaff-Schwartz model using optional name-value pair arguments. `optstockbyls` computes prices of European, Bermudan, and American vanilla options. For American and Bermudan options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the `RateSpec` obtained from `intenvset`. For information on the interest-rate specification, see `intenvset`.

Data Types
struct

StockSpec - Stock specification for underlying asset
structure

Stock specification for underlying asset, specified using `StockSpec` obtained from `stockspec`. For information on the stock specification, see `stockspec`.

`stockspec` can handle other types of underlying assets. For example, stocks, stock indices, and commodities.

Data Types
struct

OptSpec - Definition of option
string with value `'call'` or `'put'`

Definition of option, specified as `'call'` or `'put'` using a string.

Data Types
char

Strike - Option strike price values
nonnegative scalar integer

Option strike price value, specified with nonnegative scalar integer:

- For a European option, use a scalar of strike price.

- For a Bermuda option, use a 1-by-NSTRIKES vector of strike prices.

- For an American option, use a scalar of strike price.

Data Types
single | double

Settle - Settlement date or trade date
date string | nonnegative scalar integer

Settlement date or trade date for the vanilla option, specified as a date string or nonnegative scalar integer.

Data Types
char

ExerciseDates - Option exercise dates
date string | nonnegative integer

Option exercise dates, specified as a date string or nonnegative scalar integer:

- For a European option, use a 1-by-1 vector of dates. For a European option, there is only one ExerciseDates on the option expiry date.

- For a Bermuda option, use a 1-by-NSTRIKES vector of dates.

- For an American option, use a 1-by-2 vector of exercise date boundaries. The option can be exercised on any date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is a 1-by-1 vector of serial date numbers or cell array of strings, the option can be exercised between Settle and the single listed ExerciseDates.

Data Types
char

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price = optstockbyls(RateSpec,StockSpec,OptSpec,Strike,Settle,ExerciseDates,'A

'AmericanOpt' - Option type
0 European or Bermuda (default) | scalar with values [0,1]

Option type, specified as positive integer scalar flags with values:

- 0 — European or Bermuda

- 1 — American

Data Types
single | double

'NumTrials' - Simulation trials
1000 (default) | scalar

Simulation trials, specified as a scalar number of independent sample paths.

Data Types
double

'NumPeriods' - Simulation periods per trial
100 (default) | scalar

Simulation periods per trial, specified as a scalar number. NumPeriods is considered only when pricing European vanilla options. For American and Bermuda vanilla options, NumPeriod is equal to the number of Exercise days during the life of the option.

Data Types
double

optstockbyls

'Z' - Dependent random variates
TBD (default) | scalar | nonnegative integer

Dependent random variates used to generate the Brownian motion vector (i.e., Wiener processes) that drive the simulation, specified as a NumPeriods-by-2-by-NumTrials 3-D time series array.

Data Types
single | double

'Antithetic' - Indicator for antithetic sampling
false (default) | logical flag with value of true or false

Indicator for antithetic sampling, specified with a value of true or false.

Data Types
logical

Output Arguments

Price - Expected price of vanilla option
scalar

Expected price of the vanilla option, returned as a 1-by-1 scalar.

Path - Simulated paths of correlated state variables
vector

Simulated paths of correlated state variables, returned as a (NumPeriods + 1)-by-2-by-NumTrials 3-D time series array. Each row of Paths is the transpose of the state vector $X(t)$ at time t for a given trial.

Times - Observation times associated with simulated paths
vector

Observation times associated with simulated paths, returned as a (NumPeriods + 1)-by-1 column vector of observation times associated with the simulated paths. Each element of Times is associated with the corresponding row of Paths.

Z - Dependent random variates

vector

Dependent random variates, if Z is specified as an optional input
argument, the same value is returned. Otherwise, Z contains the
random variates generated internally.

Examples   **Compute the Price of a Vanilla Option**

Define the RateSpec.

```
StartDates = 'Jan-1-2013';
EndDates = 'Jan-1-2015';
Rates = 0.05;
RateSpec = intenvset('ValuationDate', StartDates, 'StartDates', StartDates, ...
'EndDates', EndDates, 'Rates', Rates)


RateSpec =

            FinObj: 'RateSpec'
       Compounding: 2
              Disc: 0.9060
             Rates: 0.0500
          EndTimes: 4
        StartTimes: 0
          EndDates: 735965
        StartDates: 735235
     ValuationDate: 735235
             Basis: 0
       EndMonthRule: 1
```

Define the StockSpec for the asset.

```
AssetPrice = 100;
Sigma = 0.1;
StockSpec = stockspec(Sigma, AssetPrice)


StockSpec =
```

```
              FinObj: 'StockSpec'
               Sigma: 0.1000
          AssetPrice: 100
        DividendType: []
     DividendAmounts: 0
     ExDividendDates: []
```

Define the vanilla option.

```
OptSpec = 'put';
Settle = 'Jan-1-2013';
ExerciseDates = 'Jan-1-2015';
Strike = 105;
```

Compute the vanilla option price using the Longstaff-Schwartz model.

```
Antithetic = true;
Price = optstockbyls(RateSpec, StockSpec, OptSpec, Strike, Settle, ...
ExerciseDates, 'Antithetic', Antithetic)

Price =

    3.2292
```

See Also   optstocksensbyls

Concepts   • "Vanilla Option" on page 3-35

optstocksensbyls

Purpose	Calculate European, Bermudan, or American vanilla option prices and sensitivities using Longstaff-Schwartz model

Syntax

```
PriceSens = optstocksensbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,
ExerciseDates)
PriceSens = optstocksensbyls( ___ ,Name,Value)

[PriceSens,Path,Times,Z] = optstocksensbyls(RateSpec,StockSpec,OptSpec
ExerciseDates)
[PriceSens,Path,Times,Z] = optstocksensbyls( ___ ,Name,Value)
```

Description PriceSens =
optstocksensbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,
ExerciseDates) returns vanilla option prices and sensitivities using
the Longstaff-Schwartz model. optstockbyls computes prices of
European, Bermudan, and American vanilla options. For American and
Bermudan options, the Longstaff-Schwartz least squares method is
used to calculate the early exercise premium.

PriceSens = optstocksensbyls(___ ,Name,Value) returns vanilla
option prices using the Longstaff-Schwartz model and using optional
name-value pair arguments. optstockbyls computes prices and
sensitivities of European, Bermudan, and American vanilla options.
For American and Bermudan options, the Longstaff-Schwartz least
squares method is used to calculate the early exercise premium.

[PriceSens,Path,Times,Z] =
optstocksensbyls(RateSpec,StockSpec,OptSpec,StrikeSettle,
ExerciseDates) returns vanilla option prices and sensitivities using
the Longstaff-Schwartz model. optstockbyls computes prices of
European, Bermudan, and American vanilla options. For American and
Bermudan options, the Longstaff-Schwartz least squares method is
used to calculate the early exercise premium.

optstocksensbyls

[PriceSens,Path,Times,Z] = optstocksensbyls(___ ,Name,Value)
returns vanilla option prices and sensitivities using the
Longstaff-Schwartz model and using optional name-value pair
arguments. `optstockbyls` computes prices of European, Bermudan,
and American vanilla options. For American and Bermudan options,
the Longstaff-Schwartz least squares method is used to calculate the
early exercise premium.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously
compounded), specified by the `RateSpec` obtained from `intenvset`. For
information on the interest-rate specification, see `intenvset`.

Data Types
struct

StockSpec - Stock specification for underlying asset
structure

Stock specification for underlying asset, specified using `StockSpec`
obtained from `stockspec`. For information on the stock specification,
see `stockspec`.

`stockspec` can handle other types of underlying assets. For example,
stocks, stock indices, and commodities.

Data Types
struct

OptSpec - Definition of option
string values `'call'` or `'put'`

Definition of option, specified as `'call'` or `'put'` using a string.

Data Types
char

Strike - Option strike price value

nonnegative scalar integer

Option strike price value, specified with a nonnegative scalar integer:

- For a European option, use a scalar of strike price.

- For a Bermuda option, use a 1-by-NSTRIKES vector of strike price.

- For an American option, use a scalar of strike price.

Data Types
single | double

Settle - Settlement date or trade date
date string | nonnegative scalar integer

Settlement date or trade date for the vanilla option, specified as a date string or a nonnegative integer.

Data Types
char

ExerciseDates - Option exercise date
date string | nonnegative scalar integer

Option exercise date, specified as a date string or nonnegative scalar integer:

- For a European option, use a 1-by-1 vector of dates. For a European option, there is only one ExerciseDates on the option expiry date.

- For a Bermuda option, use a 1-by-NSTRIKES vector of dates.

- For an American option, use a 1-by-2 vector of exercise date boundaries. The option can be exercised on any date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is a 1-by-1 vector of serial date numbers or cell array of strings, the option can be exercised between Settle and the single listed ExerciseDates.

Data Types
char

optstocksensbyls

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments.
`Name` is the argument name and `Value` is the corresponding
value. `Name` must appear inside single quotes (' '). You can
specify several name and value pair arguments in any order as
`Name1,Value1,...,NameN,ValueN`.

Example: `Price = optstocksensbyls(RateSpec,StockSpec,`
`OptSpec,Strike,Settle,ExerciseDates,'AmericanOpt','1','NumTrials','2000',`

'AmericanOpt' - Option type
0 European or Bermuda (default) | scalar with values [0,1]

Option type, specified as a positive integer scalar flag with values:

- `0` — European or Bermuda

- `1` — American

Data Types
`single` | `double`

'NumTrials' - Simulation trials
1000 (default) | scalar

Simulation trials, specified as a scalar number of independent sample
paths.

Data Types
`double`

'NumPeriods' - Simulation periods per trial
100 (default) | scalar

Simulation periods per trial, specified as a scalar number. `NumPeriods`
is considered only when pricing European vanilla options. For American
and Bermuda vanilla options, `NumPeriod` is equal to the number of
`Exercise` days during the life of the option.

Data Types
`double`

'Z' - Dependent random variates
scalar | nonnegative integer

Dependent random variates used to generate the Brownian motion vector (i.e., Wiener processes) that drive the simulation, specified as a NumPeriods-by-2-by-NumTrials 3-D time series array.

Data Types
single | double

'Antithetic' - Indicator for antithetic sampling
false (default) | logical flag with value of true or false

Indicator for antithetic sampling, specified with a value of true or false.

Data Types
logical

'OutSpec' - Define outputs
{'Price'} (default) | string with values 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', and 'All' | cell array of strings with values: 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', and 'All'

Define outputs specifying NOUT- by-1 or 1-by-NOUT cell array of strings with possible values of 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta' and 'All'.

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec to include each sensitivity:

Example: OutSpec = {'delta','gamma','vega','lambda','rho','theta','price'}

Data Types
char | cell

optstocksensbyls

Output Arguments

PriceSens - Expected price and sensitivities of vanilla option
vector

Expected price and sensitivities of the vanilla option, returned as a `NINST`-by-1 vector.

Path - Simulated paths of correlated state variables
vector

Simulated paths of correlated state variables, returned as a (`NumPeriods` + 1)-by-2-by-`NumTrials` 3-D time series array. Each row of `Paths` is the transpose of the state vector $X(t)$ at time t for a given trial.

Times - Observation times associated with simulated paths
vector

Observation times associated with simulated paths, returned as a (`NumPeriods` + 1)-by-1 column vector of observation times associated with the simulated paths. Each element of `Times` is associated with the corresponding row of `Paths`.

Z - Dependent random variates
vector

Dependent random variates, if `Z` is specified as an optional input argument, the same value is returned. Otherwise, `Z` contains the random variates generated internally.

Examples

Compute the Price and Sensitivities of a Vanilla Option

Define the `RateSpec`.

```
StartDates = 'Jan-1-2013';
EndDates = 'Jan-1-2015';
Rates = 0.05;
RateSpec = intenvset('ValuationDate', StartDates, 'StartDates', StartDates, ...
'EndDates', EndDates, 'Rates', Rates)

RateSpec =
```

```
            FinObj: 'RateSpec'
       Compounding: 2
              Disc: 0.9060
             Rates: 0.0500
          EndTimes: 4
        StartTimes: 0
          EndDates: 735965
        StartDates: 735235
     ValuationDate: 735235
             Basis: 0
       EndMonthRule: 1
```

Define the StockSpec for the asset.

```
AssetPrice = 100;
Sigma = 0.1;
DivType = 'continuous';
DivAmounts = 0.04;
StockSpec = stockspec(Sigma, AssetPrice, DivType, DivAmounts)


StockSpec =

             FinObj: 'StockSpec'
              Sigma: 0.1000
         AssetPrice: 100
        DividendType: {'continuous'}
    DividendAmounts: 0.0400
    ExDividendDates: []
```

Define the vanilla option.

```
OptSpec = 'call';
Settle = 'jan-1-2013';
ExerciseDates = 'jan-1-2015';
Strike = 105;
```

optstocksensbyls

Compute the `Delta` sensitivity for the vanilla option using the Longstaff-Schwartz model.

```
Antithetic = true;
OutSpec = {'Delta'};
PriceSens = optstocksensbyls(RateSpec, StockSpec, OptSpec, Strike, ...
Settle, ExerciseDates,'Antithetic', Antithetic, 'OutSpec', OutSpec)

PriceSens =

    0.4178
```

See Also   optstockbyls

Concepts   • "Vanilla Option" on page 3-35

optstockbyrgw

Purpose	Determine American call option prices using Roll-Geske-Whaley option pricing model

Syntax

```
Price = optstockbyrgw(RateSpec, StockSpec, Settle,
Maturity, Strike)
```

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
Strike	NINST-by-1 vector of strike price values.

Description Price = optstockbyrgw(RateSpec, StockSpec, Settle, Maturity, Strike) computes the American call option prices using the Roll-Geske-Whaley option pricing model.

Price is a NINST-by-1 vector of expected call option prices.

Note optstockbyrgw computes prices of American calls with a single cash dividend using the Roll-Geske-Whaley option pricing model.

Examples Consider an American call option with an exercise price of $22 that expires on February 1, 2009. The underlying stock is trading at $20 on June 1, 2008 and has a volatility of 20% per annum. The annualized continuously compounded risk-free rate is 6.77% per annum. The stock pays a single dividend of $2 on September 1, 2008. Using this data,

compute price of the American call option using the Roll-Geske-Whaley option pricing model:

```
Settle = 'Jun-01-2008';
Maturity = 'Feb-01-2009';
AssetPrice = 20;
Strike = 22;
Sigma  = 0.2;
Rate = 0.0677;
DivAmount = 2;
DivDate = 'Sep-01-2008';
```

Define StockSpec and RateSpec:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, 'EndDates',...
Maturity, 'Rates', Rate, 'Compounding', -1, 'Basis', 0);

StockSpec = stockspec(Sigma, AssetPrice, {'cash'}, DivAmount, DivDate);
```

Compute the price of the American call :

```
Price  = optstockbyrgw(RateSpec, StockSpec, Settle, Maturity,Strike)

Price =

    0.3359
```

See Also   impvbyrgw | intenvset | optstocksensbyrgw | stockspec

Purpose   Determine American option prices and sensitivities using Bjerksund-Stensland 2002 option pricing model

Syntax   PriceSens = optstocksensbybjs(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, 'Name1', Value1...)

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may be specified in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are:

- NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are: 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', or 'All'.

  For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output should be Price, Lambda, and Rho, in that order.

optstocksensbybjs

To invoke from a function: [Price, Lambda, Rho] = optstocksensbybjs(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

- Default is OutSpec = {'Price'}.

Description   PriceSens = optstocksensbybjs(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, 'Name1', Value1...) computes American option prices and sensitivities using the Bjerksund-Stensland 2002 option pricing model.

optstocksensbybjs can be used to compute six sensitivities for the Bjerksund-Stensland 2002 model: delta, gamma, vega, lambda, rho, and theta. This function is also capable of returning the price of the option. The selection of output parameters and their order is determined by the optional input parameter OutSpec. This parameter is a cell array of strings, each one specifying a desired output parameter. The order in which these output parameters are returned by the function is the same as the order of the strings contained in OutSpec.

PriceSens is a NINST-by-1 vector of expected prices or sensitivities values.

Note optstocksensbybjs computes prices of American options with continuous dividend yield using the Bjerksund-Stensland option pricing model. All sensitivities are evaluated by computing a discrete approximation of the partial derivative. This means that the option is revalued with a fractional change for each relevant parameter, and the change in the option value divided by the increment, is the approximated sensitivity value.

Examples Consider four American put options with an exercise price of $100. The options expire on October 1, 2008. Assume the underlying stock pays a continuous dividend yield of 4% and has a volatility of 40% per annum. The annualized continuously compounded risk-free rate is 8% per annum. Using this data, calculate the delta, gamma, and price of the American put options, assuming the following current prices of the stock on July 1, 2008: $90, $100, $110 and $120:

```
Settle = 'July-1-2008';
Maturity = 'October-1-2008';
Strike = 100;
AssetPrice = [90;100;110;120];
Rate = 0.08;
Sigma = 0.40;
DivYield = 0.04;
```

Define StockSpec and RateSpec:

```
StockSpec = stockspec(Sigma, AssetPrice, {'continuous'}, DivYield);

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rate, 'Compounding', -1);
```

Define the option type:

```
OptSpec = {'put'};
```

Compute delta, gamma, and price of the put options using the
Bjerksund-Stensland 2002 option pricing model:

```
OutSpec = {'Delta', 'Gamma', 'Price'};

[Delta, Gamma, Price] = optstocksensbybjs(RateSpec, StockSpec, Settle, Maturity,...
OptSpec, Strike, 'OutSpec', OutSpec)

Delta =

   -0.6572
   -0.4434
   -0.2660
   -0.1442

Gamma =

    0.0217
    0.0202
    0.0150
    0.0095

Price =

   12.9467
    7.4571
    3.9539
    1.9495
```

References   Bjerksund, P. and G. Stensland, *Closed-Form Approximation of American Options*, Scandinavian Journal of Management, 1993, Vol. 9, Suppl., pp. S88-S99.

Bjerksund, P. and G. Stensland, *Closed Form Valuation of American Options*, Discussion paper 2002 (http://brage.bibsys.no/nhh/bitstream/URN:NBN:no-bibsys_brage_22301/1/bjerksund%2

See Also   impvbybjs | intenvset | optstockbybjs | stockspec

Purpose	Determine option prices and sensitivities on futures using Black pricing model

Syntax

```
PriceSens = optstocksensbyblk(RateSpec, StockSpec, Settle,
Maturity, OptSpec, Strike, 'Name1', Value1...)
```

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may be specified in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are:

- NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are: 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', or 'All'.

  For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output should be Price, Lambda, and Rho, in that order.

optstocksensbyblk

To invoke from a function: [Price, Lambda, Rho] = optstocksensbyblk(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

- Default is OutSpec = {'Price'}.

Description   PriceSens = optstocksensbyblk(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, 'Name1', Value1...) computes option prices and sensitivities on futures using the Black pricing model.

PriceSens is a NINST-by-1 vector of expected future prices or sensitivities values.

Examples   Consider a European put option on a futures contract with an exercise price of $60 that expires on June 30, 2008. On April 1, 2008 the underlying stock is trading at $58 and has a volatility of 9.5% per annum. The annualized continuously compounded risk-free rate is 5% per annum. Using this data, compute delta, gamma, and the price of the put option.

```
AssetPrice = 58;
Strike = 60;
Sigma = .095;
Rates = 0.05;
Settle = 'April-01-08';
Maturity = 'June-30-08';
```

Create RateSpec and StockSpec:

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, 'EndDates',...
Maturity, 'Rates', Rates, 'Compounding', -1, 'Basis', 1);


StockSpec = stockspec(Sigma, AssetPrice);
```

Define the options:

```
OptSpec = {'put'};
```

Compute `Delta`, `Gamma` and `Price` for the European put option:

```
OutSpec = {'Delta','Gamma','Price'};
[Delta, Gamma, Price] = optstocksensbyblk(RateSpec, StockSpec, Settle,...
Maturity, OptSpec, Strike,'OutSpec', OutSpec)

Delta =

   -0.7469


Gamma =

    0.1130


Price =

    2.3569
```

See Also   impvbyblk | intenvset | optstockbyblk | stockspec

optstocksensbybls

Purpose	Determine option prices and sensitivities using Black-Scholes option pricing model
Syntax	PriceSens = optstocksensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, 'Name1', Value1...)

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may be specified in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are:

- NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are: 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', or 'All'.

  For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output should be Price, Lambda, and Rho, in that order.

To invoke from a function: [Price, Lambda, Rho] = optstocksensbybls(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

- Default is OutSpec = {'Price'}.

Description   PriceSens = optstocksensbybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, 'Name1', Value1...) computes option prices and sensitivities using the Black-Scholes option pricing model.

PriceSens is a NINST-by-1 vector of expected prices or sensitivities values.

optstocksensbybls

Examples Consider a European call and put options with an exercise price of $30 that expires on June 1, 2008. The underlying stock is trading at $30 on January 1, 2008 and has a volatility of 30% per annum. The annualized continuously compounded risk-free rate is 5% per annum. Using this data, compute the `delta`, `gamma`, and `price` of the options using the Black-Scholes model.

```
AssetPrice = 30;
Strike = 30;
Sigma = .30;
Rates = 0.05;
Settle = 'January-01-2008';
Maturity = 'June -01-2008';
```

Define `RateSpec` and `StockSpec` :

```
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, 'EndDates',...
```

```
Maturity, 'Rates', Rates, 'Compounding',-1, 'Basis', 1);

StockSpec = stockspec(Sigma, AssetPrice);
```

Define the options:

```
OptSpec = {'call', 'put'};
```

Compute delta, gamma, and price for the European options:

```
OutSpec = {'Delta','Gamma','Price'};
[Delta, Gamma, Price] = optstocksensbybls(RateSpec, StockSpec, Settle,...
Maturity, OptSpec, Strike,'OutSpec', OutSpec)

Delta =

    0.5810
   -0.4190


Gamma =

    0.0673
    0.0673


Price =

    2.6126
    1.9941
```

See Also   impvbybls | intenvset | optstockbybls | stockspec

optstocksensbylr

Purpose	Determine option prices and sensitivities using Leisen-Reimer binomial tree model
Syntax	PriceSens = optstocksensbylr(LRTree, OptSpec, Strike, Settle, ExerciseDates) PriceSens = optstocksensbylr(LRTree, OptSpec, Strike, Settle, ExerciseDates, Name,Value)
Description	PriceSens = optstocksensbylr(LRTree, OptSpec, Strike, Settle, ExerciseDates) calculates option prices and sensitivities using a Leisen-Reimer binomial tree model. PriceSens = optstocksensbylr(LRTree, OptSpec, Strike, Settle, ExerciseDates, Name,Value) constructs a Leisen-Reimer stock tree with additional options specified by one or more Name,Value pair arguments.

Input Arguments

LRTree

Stock tree structure created by lrtree.

OptSpec

NINST-by-1 cell array of strings 'call' or 'put'.

Strike

NINST-by-1 (European/American) or NINST-by-NSTRIKES (Bermuda) matrix of strike price values. Each row is the schedule for one option. If an option has fewer than NSTRIKES exercise opportunities, the end of the row is padded with NaNs.

Settle

NINST-by-1 matrix of settlement or trade dates.

ExerciseDates

NINST-by-1(European/American) or NINST-by-NSTRIKEDATES (Bermuda) matrix of exercise dates. Each row is the schedule for one option. For a European option, there is only one ExerciseDate on the option expiry date. For the American type, the option can be exercised on any tree data between the ValuationDate and tree maturity. The last element of each row must be the same as the maturity of the tree.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'AmericanOpt'

NINST-by-1 flags with values of 0 (European/Bermuda) or 1 (American).

> **Default:** 0

'OutSpec'

NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are Price, Delta, Gamma, Vega, Lambda, Rho, and All.

> **Default:** Price

Output Arguments

PriceSens

NINST-by-1 expected prices or sensitivities values.

Examples

Consider European call and put options with an exercise price of $100 that expire on December 1, 2010. The underlying stock is trading at $100 on June 1, 2010 and has a volatility of 30% per annum. The annualized continuously compounded risk-free rate is 7% per annum. Using this data, compute the price, delta and gamma of the options

using the Leisen-Reimer model with a tree of 25 time steps and the PP2 method.

```
AssetPrice  = 100;
Strike = 100;

ValuationDate = 'June-1-2010';
Maturity = 'December-1-2010';

%Define StockSpec
Sigma = 0.3;

StockSpec = stockspec(Sigma, AssetPrice);

% Define RateSpec
Rates = 0.07;
Settle = ValuationDate;
Basis = 1;
Compounding = -1;

RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', Basis);

%Build the Leisen-Reimer (LR) tree with 25 time steps
LRTimeSpec  = lrtimespec(ValuationDate, Maturity, 25);

% Use the PP2 method
LRMethod  = 'PP2';

TreeLR = lrtree(StockSpec, RateSpec, LRTimeSpec, Strike, 'method', LRMethod);

%Compute prices and sensitivities using the LR model:
OptSpec = {'call'; 'put'};
OutSpec = {'Price', 'Delta', 'Gamma'};

[Price, Delta, Gamma] = optstocksensbylr(TreeLR, OptSpec, Strike, Settle, ...
Maturity, 'OutSpec', OutSpec)
```

```
Price =

   10.1332
    6.6937

Delta =

    0.6056
   -0.3944

Gamma =

    0.0185
    0.0185
```

References   Leisen D.P., M. Reimer, "Binomial Models for Option Valuation –
Examining and Improving Convergence," *Applied Mathematical
Finance*, Number 3, 1996, pp. 319-346.

See Also | optstockbylr | lrtree

optstocksensbyrgw

Purpose
Determine American call option prices and sensitivities using Roll-Geske-Whaley option pricing model

Syntax
```
PriceSens = optstocksensbyrgw(RateSpec, StockSpec, Settle,
Maturity, OptSpec, Strike, 'Name1', Value1...)
```

Arguments

RateSpec	The annualized continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
Strike	NINST-by-1 vector of strike price values.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may be specified in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are:
	• NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are: 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', or 'All'.
	For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output should be Price, Lambda, and Rho, in that order.

To invoke from a function: [Price, Lambda, Rho] = optstocksensbyrgw(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

- Default is OutSpec = {'Price'}.

Description
PriceSens = optstocksensbyrgw(RateSpec, StockSpec, Settle, Maturity, OptSpec, Strike, 'Name1', Value1...) computes American call option prices and sensitivities using the Roll-Geske-Whaley option pricing model.

PriceSens is a NINST-by-1 vector of expected prices or sensitivities values.

Note optstocksensbyrgw computes prices of American calls with a single cash dividend using the Roll-Geske-Whaley option pricing model. All sensitivities are evaluated by computing a discrete approximation of the partial derivative. This means that the option is revalued with a fractional change for each relevant parameter, and the change in the option value divided by the increment, is the approximated sensitivity value.

Examples
Consider an American stock option with an exercise price of $82 on January 1, 2008 that expires on May 1, 2008. Assume the underlying stock pays dividends of $4 on April 1, 2008. The stock is trading at $80 and has a volatility of 30% per annum. The risk-free rate is 6% per annum. Using this data, calculate the price and the value of delta

and `gamma` of the American call using the Roll-Geske-Whaley option pricing model:

```
AssetPrice = 80;
Settle = 'Jan-01-2008';
Maturity = 'May-01-2008';
Strike = 82;
Rate = 0.06;
Sigma  = 0.3;
DivAmount = 4;
DivDate = 'Apr-01-2008';
```

Define `StockSpec` and `RateSpec`:

```
StockSpec = stockspec(Sigma, AssetPrice, {'cash'}, DivAmount, DivDate);

RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rate, 'Compounding', -1, 'Basis', 1);
```

Define `OutSpec`:

```
OutSpec = {'Price', 'Delta', 'Gamma'};
```

Calculate the call `Price`, `Delta`, and `Gamma`:

```
[Price, Delta, Gamma]  = optstocksensbyrgw(RateSpec, StockSpec, Settle,...
Maturity, Strike,'OutSpec', OutSpec)

Price =

    4.3860


Delta =

    0.5022
```

```
Gamma =

   0.0336
```

See Also   impvbyrgw | intenvset | optstockbyrgw | stockspec

rangefloatbybdt

Purpose	Price range floating note using Black-Derman-Toy tree
Syntax	Price = rangefloatbybdt(BDTTree,Spread,Settle,Maturity, RateSched) [Price,PriceTree] = rangefloatbybdt(BDTTree,Spread,Settle, Maturity, RateSched, Name,Value)
Description	Price = rangefloatbybdt(BDTTree,Spread,Settle,Maturity, RateSched) calculates the price of the range note instrument at the valuation date using a BDT model.
	[Price,PriceTree] = rangefloatbybdt(BDTTree,Spread,Settle, Maturity, RateSched, Name,Value) calculates the price of the range note instrument at the valuation date and the price evolution for one or more range instruments using a BDT model with additional options specified by one or more Name,Value pair arguments.

Input Arguments

BDTTree

Interest-rate tree structure created by bdttree.

Spread

NINST-by-1 vector of the number of basis points over the reference rate.

Settle

NINST-by-1 vector of dates representing the settle date of the range floating note.

Note The Settle date for every range floating instrument is set to the ValuationDate of the BDT tree. The range floating note argument Settle is ignored.

Maturity

NINST-by-1 vector of dates representing the maturity date of the floating-rate note.

RateSched

NINST-by-1 vector of structures representing the range of rates within which cash flows are nonzero. Each element of the structure array contains two fields:

- `RateSched.Dates` — NDates-by-1 cell array of dates corresponding to the range schedule.

- `RateSched.Rates` — NDates-by-2 array with the first column containing the lower bound of the range and the second column containing the upper bound of the range. Cash flow for date `RateSched.Dates`(n) is nonzero for rates in the range `RateSched.Rates`(n,1) < Rate < `RateSched.Rate` (n,2).

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Basis'

NINST-by-1 vector representing the day-count basis used when annualizing the input forward rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

> **Default:** 1 (in effect)

'Options'

Structure created with derivset containing derivatives pricing options.

> **Default:** None

'Principal'

NINST-by-1 vector of the notional principal amount.

> **Default:** 100

'Reset'

NINST-by-1 vector representing the frequency of payments per year.

Default: 1

Output Arguments

Price

NINST-by-1 vector for expected prices at time 0.

PriceTree

Structure containing trees of vectors of instrument prices and accrued interest, and a vector of observation times for each node. Values are:

- PriceTree.PTree contains the clean prices.

- PriceTree.AITree contains the accrued interest.

- PriceTree.tObs contains the observation times.

Definitions

Range Note Instrument

A range note is a structured (market-linked) security whose coupon rate is equal to the reference rate as long as the reference rate is within a certain range. If the reference rate is outside of the range, the coupon rate is 0 for that period. This type of instrument entitles the holder to cash flows that depend on the level of some reference interest rate and are floored to be positive. The note holder gets directs exposure to the reference rate. In return for the drawback that no interest will be paid for the time the range is left, they offer higher coupon rates than comparable standard products, like vanilla floating notes.

Examples

Compute the price of a range note:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

%  Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
```

```
                    'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding);

                    % Instrument
                    % The range note matures in Jan-1-2014 and has the following RateSchedule:
                    Spread = 100;
                    Settle = 'Jan-1-2011';
                    Maturity = 'Jan-1-2014';
                    RateSched(1).Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
                    RateSched(1).Rates = [0.045 0.055 ; 0.0525  0.0675; 0.06 0.08];

                    % The data to build the tree is as follows:
                    % Assume the volatility to be 10%.
                    Sigma = 0.1;
                    BDTTS = bdttimespec(ValuationDate, EndDates, Compounding);
                    BDTVS = bdtvolspec(ValuationDate, EndDates, Sigma*ones(1, length(EndDates))');
                    BDTT = bdttree(BDTVS, RS, BDTTS);

                    %Price the instrument
                    Price = rangefloatbybdt(BDTT, Spread, Settle, Maturity, RateSched)

                    Price =

                       97.5267
```

References   Jarrow, Robert, *Modelling Fixed Income Securities and Interest Rate Options*, Stanford Economics and Finance, 2nd edition, 2002.

See Also | bdttree | cfbybdt | floatbybdt | swapbybdt | floorbybdt | fixedbybdt | bondbybdt | rangefloatbyhjm | rangefloatbybk | instrangefloat | rangefloatbyhw

Purpose	Price range floating note using Black-Karasinski tree

Syntax
```
Price = rangefloatbybk(BKTree,Spread,Settle,Maturity,
RateSched)
[Price,PriceTree] = rangefloatbybk(BKTree,Spread,Settle,
Maturity, RateSched, Name,Value)
```

Description   `Price = rangefloatbybk(BKTree,Spread,Settle,Maturity,`
`RateSched)` calculates the price of the range note instrument at the
valuation date using a BK model.

`[Price,PriceTree] = rangefloatbybk(BKTree,Spread,Settle,`
`Maturity, RateSched, Name,Value)` calculates the price of the range
note instrument at the valuation date and the price evolution for one
or more range instruments using a BK model with additional options
specified by one or more `Name,Value` pair arguments.

Input Arguments

BKTree

Interest-rate tree structure created by `bktree`.

Spread

`NINST`-by-1 vector of the number of basis points over the reference rate.

Settle

`NINST`-by-1 vector of dates representing the settle date of the range
floating note.

Note The `Settle` date for every range floating instrument is set to
the `ValuationDate` of the BK tree. The range floating note argument
`Settle` is ignored.

Maturity

NINST-by-1 vector of dates representing the maturity date of the floating-rate note.

RateSched

NINST-by-1 vector of structures representing the range of rates within which cash flows are nonzero. Each element of the structure array contains two fields:

- `RateSched.Dates` — NDates-by-1 cell array of dates corresponding to the range schedule.

- `RateSched.Rates` — NDates-by-2 array with the first column containing the lower bound of the range and the second column containing the upper bound of the range. Cash flow for date `RateSched.Dates`(n) is nonzero for rates in the range `RateSched.Rates`(n,1) < Rate < `RateSched.Rate` (n,2).

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Basis'

NINST-by-1 vector representing the day-count basis used when annualizing the input forward rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

> **Default:** 1 (in effect)

'Options'

Structure created with derivset containing derivatives pricing options.

> **Default:** None

'Principal'

NINST-by-1 vector of the notional principal amount.

> **Default:** 100

'Reset'

NINST-by-1 vector representing the frequency of payments per year.

Default: 1

Output Arguments

Price

NINST-by-1 vector for expected prices at time 0.

PriceTree

Structure containing trees of vectors of instrument prices and accrued interest, and a vector of observation times for each node. Values are:

- PriceTree.PTree contains the clean prices.

- PriceTree.AITree contains the accrued interest.

- PriceTree.tObs contains the observation times.

Definitions

Range Note Instrument

A range note is a structured (market-linked) security whose coupon rate is equal to the reference rate as long as the reference rate is within a certain range. If the reference rate is outside of the range, the coupon rate is 0 for that period. This type of instrument entitles the holder to cash flows that depend on the level of some reference interest rate and are floored to be positive. The note holder gets directs exposure to the reference rate. In return for the drawback that no interest will be paid for the time the range is left, they offer higher coupon rates than comparable standard products, like vanilla floating notes.

Examples

Compute the price of a range note:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

%  Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
```

```
                'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding);

                % Instrument
                % The range note matures in Jan-1-2014 and has the following RateSchedule:
                Spread = 100;
                Settle = 'Jan-1-2011';
                Maturity = 'Jan-1-2014';
                RateSched(1).Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
                RateSched(1).Rates = [0.045 0.055 ; 0.0525  0.0675; 0.06 0.08];

                % The data to build the tree is as follows:
                VolDates = ['1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014';'1-Jan-2015'];
                VolCurve = 0.01;
                AlphaDates = '01-01-2015';
                AlphaCurve = 0.1;

                BKVS = bkvolspec(RS.ValuationDate, VolDates, VolCurve,...
                AlphaDates, AlphaCurve);
                BKTS = bktimespec(RS.ValuationDate, VolDates, Compounding);
                BKT = bktree(BKVS, RS, BKTS);

                %Price the instrument
                Price = rangefloatbybk(BKT, Spread, Settle, Maturity, RateSched)

                Price =

                  102.7574
```

References   Jarrow, Robert, *Modelling Fixed Income Securities and Interest Rate Options*, Stanford Economics and Finance, 2nd edition, 2002.

See Also | bktree | cfbybk | capbybk | swapbybk | floorbybk | fixedbybk | bondbybk | rangefloatbyhjm | rangefloatbybdt | rangefloatbyhw | instrangefloat

rangefloatbyhjm

Purpose	Price range floating note using Heath-Jarrow-Morton tree
Syntax	Price = rangefloatbyhjm(HJMTree,Spread,Settle,Maturity, RateSched) [Price,PriceTree] = rangefloatbyhjm(HJMTree,Spread,Settle, Maturity, RateSched, Name,Value)
Description	Price = rangefloatbyhjm(HJMTree,Spread,Settle,Maturity, RateSched) calculates the price of the range note instrument at the valuation date using an HJM model.
	[Price,PriceTree] = rangefloatbyhjm(HJMTree,Spread,Settle, Maturity, RateSched, Name,Value) calculates the price of the range note instrument at the valuation date and the price evolution for one or more range instruments using an HJM model with additional options specified by one or more Name,Value pair arguments.

Input Arguments

HJMTree

Interest-rate tree structure created by hjmtree.

Spread

NINST-by-1 vector of the number of basis points over the reference rate.

Settle

NINST-by-1 vector of dates representing the settle date of the range floating note.

Note The Settle date for every range floating instrument is set to the ValuationDate of the HJM tree. The range floating note argument Settle is ignored.

Maturity

NINST-by-1 vector of dates representing the maturity date of the floating-rate note.

RateSched

NINST-by-1 vector of structures representing the range of rates within which cash flows are nonzero. Each element of the structure array contains two fields:

- `RateSched.Dates` — NDates-by-1 cell array of dates corresponding to the range schedule.

- `RateSched.Rates` — NDates-by-2 array with the first column containing the lower bound of the range and the second column containing the upper bound of the range. Cash flow for date `RateSched.Dates`(n) is nonzero for rates in the range `RateSched.Rates`(n,1) < Rate < `RateSched.Rate` (n,2).

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Basis'

NINST-by-1 vector representing the day-count basis used when annualizing the input forward rate tree.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

> **Default:** 1 (in effect)

'Options'

Structure created with derivset containing derivatives pricing options.

> **Default:** None

'Principal'

NINST-by-1 vector of the notional principal amount.

> **Default:** 100

'Reset'

NINST-by-1 vector representing the frequency of payments per year.

Default: 1

Output
Arguments

Price

NINST-by-1 vector for expected prices at time 0.

PriceTree

Structure containing trees of vectors of instrument prices and accrued interest, and a vector of observation times for each node. Values are:

- PriceTree.PTree contains the clean prices.

- PriceTree.AITree contains the accrued interest.

- PriceTree.tObs contains the observation times.

Definitions   **Range Note Instrument**

A range note is a structured (market-linked) security whose coupon rate is equal to the reference rate as long as the reference rate is within a certain range. If the reference rate is outside of the range, the coupon rate is 0 for that period. This type of instrument entitles the holder to cash flows that depend on the level of some reference interest rate and are floored to be positive. The note holder gets directs exposure to the reference rate. In return for the drawback that no interest will be paid for the time the range is left, they offer higher coupon rates than comparable standard products, like vanilla floating notes.

Examples   Compute the price of a range note:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

%  Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
```

```
                    'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding);

                    % Instrument
                    % The range note matures in Jan-1-2014 and has the following RateSchedule:
                    Spread = 100;
                    Settle = 'Jan-1-2011';
                    Maturity = 'Jan-1-2014';
                    RateSched(1).Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
                    RateSched(1).Rates = [0.045 0.055 ; 0.0525  0.0675; 0.06 0.08];

                    % The data to build the tree is as follows:
                    Volatility = [.2; .19; .18; .17];
                    CurveTerm = [ 1;  2;   3;   4];
                    MaTree = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
                    HJMTS = hjmtimespec(ValuationDate, MaTree);
                    HJMVS = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
                    HJMT = hjmtree(HJMVS, RS, HJMTS);

                    %Price the instrument
                    Price = rangefloatbyhjm(HJMT, Spread, Settle, Maturity, RateSched)

                    Price =

                       90.2348
```

References   Jarrow, Robert, *Modelling Fixed Income Securities and Interest Rate Options*, Stanford Economics and Finance, 2nd edition, 2002.

See Also | hjmtree | cfbyhjm | floatbyhjm | swapbyhjm | floorbyhjm | fixedbyhjm | bondbyhjm | rangefloatbybk | rangefloatbybdt | rangefloatbyhw | instrangefloat

Purpose	Price range floating note using Hull-White tree

Syntax
```
Price = rangefloatbyhw(HWTree,Spread,Settle,Maturity,
RateSched)
[Price,PriceTree] = rangefloatbyhw(HWTree,Spread,Settle,
Maturity, RateSched, Name,Value)
```

Description   `Price = rangefloatbyhw(HWTree,Spread,Settle,Maturity,`
`RateSched)` calculates the price of the range note instrument at the
valuation date using an HW model.

`[Price,PriceTree] = rangefloatbyhw(HWTree,Spread,Settle,`
`Maturity, RateSched, Name,Value)` calculates the price of the range
note instrument at the valuation date and the price evolution for one or
more range instruments using an HW model with additional options
specified by one or more `Name,Value` pair arguments.

Input Arguments

HWTree

Interest-rate tree structure created by `hwtree`.

Spread

`NINST`-by-1 vector of the number of basis points over the reference rate.

Settle

`NINST`-by-1 vector of dates representing the settle date of the range
floating note.

Note The `Settle` date for every range floating instrument is set to
the `ValuationDate` of the HW tree. The range floating note argument
`Settle` is ignored.

Maturity

NINST-by-1 vector of dates representing the maturity date of the floating-rate note.

RateSched

NINST-by-1 vector of structures representing the range of rates within which cash flows are nonzero. Each element of the structure array contains two fields:

- RateSched.Dates — NDates-by-1 cell array of dates corresponding to the range schedule.

- RateSched.Rates — NDates-by-2 array with the first column containing the lower bound of the range and the second column containing the upper bound of the range. Cash flow for date RateSched.Dates(n) is nonzero for rates in the range RateSched.Rates(n,1) < Rate < RateSched.Rate (n,2).

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Basis'

NINST-by-1 vector representing the day-count basis used when annualizing the input forward rate tree.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

>   **Default:** 0 (actual/actual)

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (in effect) and 0 (not in effect).

>   **Default:** 1 (in effect)

'Options'

Structure created with derivset containing derivatives pricing options.

>   **Default:** None

'Principal'

NINST-by-1 vector of the notional principal amount.

>   **Default:** 100

'Reset'

NINST-by-1 vector representing the frequency of payments per year.

Default: 1

Output Arguments

Price

NINST-by-1 vector for expected prices at time 0.

PriceTree

Structure containing trees of vectors of instrument prices and accrued interest, and a vector of observation times for each node. Values are:

- PriceTree.PTree contains the clean prices.

- PriceTree.AITree contains the accrued interest.

- PriceTree.tObs contains the observation times.

Definitions

Range Note Instrument

A range note is a structured (market-linked) security whose coupon rate is equal to the reference rate as long as the reference rate is within a certain range. If the reference rate is outside of the range, the coupon rate is 0 for that period. This type of instrument entitles the holder to cash flows that depend on the level of some reference interest rate and are floored to be positive. The note holder gets directs exposure to the reference rate. In return for the drawback that no interest will be paid for the time the range is left, they offer higher coupon rates than comparable standard products, like vanilla floating notes.

Examples

Compute the price of a range note:

```
% The data for the interest rate term structure is as follows:
Rates = [0.035; 0.042147; 0.047345; 0.052707];
ValuationDate = 'Jan-1-2011';
StartDates = ValuationDate;
EndDates = {'Jan-1-2012'; 'Jan-1-2013'; 'Jan-1-2014'; 'Jan-1-2015'};
Compounding = 1;

% Create RateSpec
RS = intenvset('ValuationDate', ValuationDate, 'StartDates', StartDates,...
```

```
                       'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding);

                       % Instrument
                       % The range note matures in Jan-1-2014 and has the following RateSchedule:
                       Spread = 100;
                       Settle = 'Jan-1-2011';
                       Maturity = 'Jan-1-2014';
                       RateSched(1).Dates = {'Jan-1-2012'; 'Jan-1-2013'  ; 'Jan-1-2014'};
                       RateSched(1).Rates = [0.045 0.055 ; 0.0525  0.0675; 0.06 0.08];

                       % The data to build the tree is as follows:
                       VolDates = ['1-Jan-2012'; '1-Jan-2013'; '1-Jan-2014';'1-Jan-2015'];
                       VolCurve = 0.01;
                       AlphaDates = '01-01-2015';
                       AlphaCurve = 0.1;

                       HWVS = hwvolspec(RS.ValuationDate, VolDates, VolCurve,...
                       AlphaDates, AlphaCurve);
                       HWTS = hwtimespec(RS.ValuationDate, VolDates, Compounding);
                       HWT = hwtree(HWVS, RS, HWTS);

                       %Price the instrument
                       Price = rangefloatbyhw(HWT, Spread, Settle, Maturity, RateSched)

                       Price =

                          96.6501
```

References   Jarrow, Robert, *Modelling Fixed Income Securities and Interest Rate Options*, Stanford Economics and Finance, 2nd edition, 2002.

See Also | hwtree | cfbyhw | capbyhw | swapbyhw | floorbyhw | fixedbyhw | bondbyhw | rangefloatbybk | rangefloatbybdt | rangefloatbyhjm | instrangefloat

rate2disc

Purpose   Discount factors from interest rates

Syntax   **Usage 1**: Interval points are input as times in periodic units.
```
Disc = rate2disc(Compounding, Rates, EndTimes)
Disc = rate2disc(Compounding, Rates, EndTimes, StartTimes)
```

Usage 2: ValuationDate is passed and interval points are input as dates.
```
[Disc, EndTimes, StartTimes] = rate2disc(Compounding, Rates,
EndDates, StartDates, ValuationDate)
[Disc, EndTimes, StartTimes] = rate2disc(Compounding, Rates,
EndDates, StartDates, ValuationDate, Basis, EndMonthRule)
```

Arguments

Compounding	Scalar value representing the rate at which the input zero rates were compounded when annualized. This argument determines the formula for the discount factors:
	Compounding = 1, 2, 3, 4, 6, 12
	Disc = $(1 + Z/F)^{(-T)}$, where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.
	Compounding = 365
	Disc = $(1 + Z/F)^{(-T)}$, where F is the number of days in the basis year and T is a number of days elapsed computed by basis.
	Compounding = -1
	Disc = $\exp(-T*Z)$, where T is time in years.
Rates	Number of points (NPOINTS) by number of curves (NCURVES) matrix of rates in decimal form. For example, 5% is 0.05 in Rates. Rates are the yields over investment intervals from

	StartTimes, when the cash flow is valued, to EndTimes, when the cash flow is received.
EndTimes	NPOINTS-by-1 vector or scalar of times in periodic units ending the interval to discount over.
StartTimes	(Optional) NPOINTS-by-1 vector or scalar of times in periodic units starting the interval to discount over. Default = 0.
EndDates	NPOINTS-by-1 vector or scalar of serial maturity dates ending the interval to discount over.

Note When ValuationDate is not passed, the third and fourth arguments (EndTimes and StartTimes) are interpreted as times.

Note : When ValuationDate is passed, the third and fourth arguments (EndDates and StartDates) are interpreted as dates. The date ValuationDate is used as the zero point for computing the times.

StartDates	(Optional) NPOINTS-by-1 vector or scalar of serial dates starting the interval to discount over. StartDates must be earlier than EndDates. Default = ValuationDate.
ValuationDate	Scalar value in serial date number form representing the observation date of the investment horizons entered in StartDates and EndDates. Required in **Usage 2**. Omitted or passed as an empty matrix to invoke **Usage 1**.

rate2disc

Basis	(Optional) Day-count basis of the instrument when using **Usage 2**. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) End-of-month rule when using **Usage 2**. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

Description   Usage 1: `Disc = rate2disc(Compounding, Rates, EndTimes)` or
`Disc = rate2disc(Compounding, Rates, EndTimes, StartTimes)`
where interval points are input as times in periodic units.

Usage 2: `[Disc, EndTimes, StartTimes] =`
`rate2disc(Compounding, Rates, EndDates, StartDates,`
`ValuationDate)` or `[Disc, EndTimes, StartTimes] =`
`rate2disc(Compounding, Rates, EndDates, StartDates,`
`ValuationDate, Basis, EndMonthRule)` where `ValuationDate` is
passed and interval points are input as dates.

`rate2disc` computes the discounts over a series of `NPOINTS` time
intervals given the annualized yield over those intervals. `NCURVES`
different rate curves can be translated at once if they have the same
time structure. The time intervals can represent a zero curve or a
forward curve.

The output `Disc` is an `NPOINTS`-by-`NCURVES` column vector of discount
factors in decimal form representing the value at time `StartTime` of a
unit cash flow received at time `EndTime`.

You can specify the investment intervals either with input times
(**Usage 1**) or with input dates (**Usage 2**). Entering `ValuationDate`
invokes the date interpretation; omitting `ValuationDate` invokes the
default time interpretations.

For **Usage 1**:

• `StartTimes` is an `NPOINTS`-by-1 column vector of times starting the
  interval to discount over, measured in periodic units.

• `EndTimes` is an `NPOINTS`-by-1 column vector of times ending the
  interval to discount over, measured in periodic units.

For **Usage 2**:

• `StartDates` is an `NPOINTS`-by-1 column vector of serial dates starting
  the interval to discount over, measured in days.

• `EndDates` is an `NPOINTS`-by-1 column vector of serial dates ending
  the interval to discount over, measured in days.

If `Compounding = 365` (daily), `StartDates` and `EndDates` are measured in days as in **Usage 2**. Otherwise, in **Usage 1**, the arguments contain values, `T`, computed from SIA semiannual time factors, `Tsemi`, by the formula `T = Tsemi/2*F`, where `F` is the compounding frequency.

Examples

Example 1 demonstrates **Usage 1**. Compute discounts from a zero curve at 6 months, 12 months, and 24 months. The times to the cash flows are 1, 2, and 4. You are computing the present value (at time 0) of the cash flows.

```
Compounding = 2;
Rates = [0.05; 0.06; 0.065];
EndTimes   = [1; 2; 4];
Disc = rate2disc(Compounding, Rates, EndTimes)

Disc =
    0.9756
    0.9426
    0.8799
```

Example 2 demonstrates **Usage 2**. Compute discounts from a zero curve at 6 months, 12 months, and 24 months. Use dates to specify the ending time horizon.

```
Compounding = 2;
Rates = [0.05; 0.06; 0.065];
EndDates = ['10/15/97'; '04/15/98'; '04/15/99'];
ValuationDate = '4/15/97';
Disc = rate2disc(Compounding, Rates, EndDates, [], ValuationDate)

Disc =
    0.9756
    0.9426
    0.8799
```

Example 3 demonstrates **Usage 1**. Compute discounts from the 1-year forward rates beginning now, in 6 months, and in 12 months. Use

monthly compounding. The times to the cash flows are 12, 18, 24, and the forward times are 0, 6, 12.

```
Compounding = 12;
Rates = [0.05; 0.04; 0.06];
EndTimes = [12; 18; 24];
StartTimes = [0; 6; 12];
Disc = rate2disc(Compounding, Rates, EndTimes, StartTimes)

Disc =
    0.9513
    0.9609
    0.9419
```

See Also disc2rate | ratetimes

ratetimes

Purpose   Change time intervals defining interest-rate environment

Syntax   [Rates, EndTimes, StartTimes] = ratetimes(Compounding,
RefRates, RefEndTimes, RefStartTimes, EndTimes, StartTimes)
[Rates, EndTimes, StartTimes] = ratetimes(Compounding,
RefRates, RefEndDates, RefStartDates, EndDates, StartDates,
ValuationDate)

Usage 1: ValuationDate not passed; third through sixth arguments
are interpreted as times.
[Rates, EndTimes, StartTimes] = ratetimes(Compounding,
RefRates, RefEndTimes, RefStartTimes, EndTimes, StartTimes)

Usage 2: ValuationDate passed and interval points input as dates.
[Rates, EndTimes, StartTimes] = ratetimes(Compounding,
RefRates, RefEndDates, RefStartDates, EndDates, StartDates,
ValuationDate)

Arguments

Compounding	Scalar value representing the rate at which the input zero rates were compounded when annualized. This argument determines the formula for the discount factors:
	Compounding = 1, 2, 3, 4, 6, 12
	Disc = $(1 + Z/F)^{(-T)}$, where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.
	Compounding = 365
	Disc = $(1 + Z/F)^{(-T)}$, where F is the number of days in the basis year and T is a number of days elapsed computed by basis.
	Compounding = -1

	Disc = exp(-T*Z), where T is time in years.
RefRates	NREFPTS-by-NCURVES matrix of reference rates in decimal form. RefRates are the yields over investment intervals from RefStartTimes, when the cash flow is valued, to RefEndTimes, when the cash flow is received.
RefEndTimes	NREFPTS-by-1 vector or scalar of times in periodic units ending the intervals corresponding to RefRates.
RefStartTimes	(Optional) NREFPTS-by-1 vector or scalar of times in periodic units starting the intervals corresponding to RefRates. Default = 0.
EndTimes	NPOINTS-by-1 vector or scalar of times in periodic units ending the interval to discount over.
StartTimes	(Optional) NPOINTS-by-1 vector or scalar of times in periodic units starting the interval to discount over. Default = 0.
RefEndDates	NREFPTS-by-1 vector or scalar of serial dates ending the intervals corresponding to RefRates.
RefStartDates	(Optional) NREFPTS-by-1 vector or scalar of serial dates starting the intervals corresponding to RefRates. Default = ValuationDate.
EndDates	NPOINTS-by-1 vector or scalar of serial maturity dates ending the interval to discount over.

ratetimes

StartDates	(Optional) NPOINTS-by-1 vector or scalar of serial dates starting the interval to discount over. StartDates must be earlier than EndDates.
	Default = ValuationDate.
ValuationDate	Scalar value in serial date number form representing the observation date of the investment horizons entered in StartDates and EndDates. Required in Usage 2. Omitted or passed as an empty matrix to invoke Usage 1.

Description   [Rates, EndTimes, StartTimes] = ratetimes(Compounding, RefRates, RefEndTimes, RefStartTimes, EndTimes, StartTimes) and [Rates, EndTimes, StartTimes] = ratetimes(Compounding, RefRates, RefEndDates, RefStartDates, EndDates, StartDates, ValuationDate) change time intervals defining an interest-rate environment.

ratetimes takes an interest-rate environment defined by yields over one collection of time intervals and computes the yields over another set of time intervals. The zero rate is assumed to be piece-wise linear in time.

Rates is an NPOINTS-by-NCURVES matrix of rates implied by the reference interest-rate structure and sampled at new intervals.

StartTimes is an NPOINTS-by-1 column vector of times starting the new intervals where rates are desired, measured in periodic units.

EndTimes is an NPOINTS-by-1 column vector of times ending the new intervals, measured in periodic units.

If Compounding = 365 (daily), StartTimes and EndTimes are measured in days. The arguments otherwise contain values, T, computed from SIA semiannual time factors, Tsemi, by the formula $T = Tsemi/2 * F$, where F is the compounding frequency.

You can specify the investment intervals either with input times (Usage 1) or with input dates (Usage 2). Entering the argument `ValuationDate` invokes the date interpretation; omitting `ValuationDate` invokes the default time interpretations.

Examples   **Example 1**. The reference environment is a collection of zero rates at 6, 12, and 24 months. Create a collection of 1-year forward rates beginning at 0, 6, and 12 months.

```
RefRates = [0.05; 0.06; 0.065];
RefEndTimes = [1; 2; 4];
StartTimes = [0; 1; 2];
EndTimes   = [2; 3; 4];
Rates = ratetimes(2, RefRates, RefEndTimes, 0, EndTimes,...
StartTimes)

Rates =
 0.0600
 0.0688
 0.0700
```

Example 2. Interpolate a zero yield curve to different dates. Zero curves start at the default date of `ValuationDate`.

```
RefRates = [0.04; 0.05; 0.052];
RefDates = [729756; 729907; 730121];
Dates    = [730241; 730486];
ValuationDate  = 729391;
Rates = ratetimes(2, RefRates, RefDates, [], Dates, [],...
ValuationDate)

Rates =
 0.0520
 0.0520
```

See Also   disc2rate | rate2disc

spreadbykirk

Purpose	Price European spread options using Kirk pricing model
Syntax	Price = spreadbykirk(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr)
Description	Price = spreadbykirk(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr) returns the price for a European spread option using the Kirk pricing model.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec1 - Stock specification for underlying asset 1
structure

Stock specification for underlying asset 1. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

StockSpec2 - Stock specification for underlying asset 2
structure

Stock specification for underlying asset 2. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

Settle - Settlement dates for spread option
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates for the spread option, specified as a date string or as nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

Maturity - Maturity date for spread option
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Maturity date for spread option, specified as a date string or as nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

OptSpec - Definition of option
string with values 'call' or 'put' | cell array of strings

Definition of option as 'call' or 'put', specified as a NINST-by-1 cell array of strings.

Data Types
cell | char

Strike - Option strike price values
integer | vector of integers

Option strike price values, specified as an integer using a `NINST-by-1` vector of strike price values.

If `Strike` is equal to `0`, the function computes the price of an exchange option.

Data Types
`single` | `double`

Corr - Correlation between underlying asset prices
integer | vector of integers

Correlation between underlying asset prices, specified as an integer using a `NINST-by-1` vector.

Data Types
`single` | `double`

Output Arguments

Price - Expected prices of spread option
vector

Expected prices of the spread option, returned as a `NINST-by-1` vector.

Examples

Compute the Price of a Spread Option Using the Kirk Model

Define the spread option dates.

```
Settle = '01-Jan-2012';
Maturity = '01-April-2012';
```

Define asset 1.

```
% Price and volatility of RBOB gasoline
  Price1gallon = 2.85;         % $/gallon
  Price1 = Price1gallon * 42;   % $/barrel
  Vol1 = 0.29;
```

Define asset 2.

```
% Price and volatility of WTI crude oil
```

```
Price2 = 93.20;          % $/barrel
Vol2 = 0.36;
```

Define the correlation between the underlying asset prices of asset 1 and asset 2.

```
Corr = 0.42;
```

Define the spread option.

```
OptSpec = 'call';
Strike = 20;
```

Define the `RateSpec`.

```
rates = 0.05;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', rates, ...
'Compounding', Compounding, 'Basis', Basis)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9876
             Rates: 0.0500
          EndTimes: 0.2500
        StartTimes: 0
          EndDates: 734960
        StartDates: 734869
    ValuationDate: 734869
             Basis: 1
       EndMonthRule: 1
```

Define the `StockSpec` for the two assets.

```
StockSpec1 = stockspec(Vol1, Price1)
```

spreadbykirk

```
StockSpec2 = stockspec(Vol2, Price2)

StockSpec1 =

              FinObj: 'StockSpec'
               Sigma: 0.2900
          AssetPrice: 119.7000
         DividendType: []
     DividendAmounts: 0
     ExDividendDates: []


StockSpec2 =

              FinObj: 'StockSpec'
               Sigma: 0.3600
          AssetPrice: 93.2000
         DividendType: []
     DividendAmounts: 0
     ExDividendDates: []
```

Compute the European spread option price based on the Kirk model.

```
Price = spreadbykirk(RateSpec, StockSpec1, StockSpec2, Settle, ...
Maturity, OptSpec, Strike, Corr)

Price =

   11.1904
```

References

Carmona, R., Durrleman, V., "Pricing and Hedging Spread Options,"
SIAM Review, Vol. 45, No. 4, pp. 627–685, Society for Industrial and
Applied Mathematics, 2003.

See Also   spreadsensbykirk | spreadbybjs | spreadbyfd | spreadbyls

Related Examples

- "Pricing European and American Spread Options" on page 3-38
- "Hedging Strategies Using Spread Options" on page 3-57

Concepts

- "Spread Option" on page 3-36

spreadbybjs

Purpose	Price European spread options using Bjerksund-Stensland pricing model
Syntax	Price = spreadbybjs(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr)
Description	Price = spreadbybjs(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr) returns the price for a European spread option using the Bjerksund-Stensland pricing model.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec1 - Stock specification for underlying asset 1
structure

Stock specification for underlying asset 1. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

StockSpec2 - Stock specification for underlying asset 2
structure

Stock specification for underlying asset 2. For information on the stock specification, see `stockspec`.

`stockspec` can handle other types of underlying assets. For example, for physical commodities the price is represented by `StockSpec.Asset`, the volatility is represented by `StockSpec.Sigma`, and the convenience yield is represented by `StockSpec.DividendAmounts`.

Data Types
`struct`

Settle - Settlement dates for spread option
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates for the spread option, specified as a date string or as nonnegative integers using a `NINST`-by-1 vector or cell array of dates.

Data Types
`char` | `cell`

Maturity - Maturity date for spread option
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Maturity date for spread option, specified as a date string or as nonnegative integers using a `NINST`-by-1 vector or cell array of dates.

Data Types
`char` | `cell`

OptSpec - Definition of option
string with values `'call'` or `'put'` | cell array of strings

Definition of option as `'call'` or `'put'`, specified as a `NINST`-by-1 cell array of strings.

Data Types
`char` | `cell`

Strike - Option strike price values

integer | vector of integers

Option strike price values, specified as an integer using a NINST-by-1 vector of strike price values.

If Strike is equal to zero the function computes the price of an exchange option.

Data Types
single | double

Corr - Correlation between underlying asset prices

integer | vector of integers

Correlation between underlying asset prices, specified as an integer using a NINST-by-1 vector.

Data Types
single | double

Output Arguments

Price - Expected prices of spread option

vector

Expected prices of the spread option, returned as a NINST-by-1 vector.

Examples

Compute the Price of a Spread Option Using the Bjerksund-Stensland Model

Define the spread option dates.

```
Settle = '01-Jan-2012';
Maturity = '01-April-2012';
```

Define asset 1.

```
% Price and volatility of RBOB gasoline
  Price1gallon = 2.85;        % $/gallon
  Price1 = Price1gallon * 42;  % $/barrel
  Vol1 = 0.29;
```

Define asset 2.

```
% Price and volatility of WTI crude oil
  Price2 = 93.20;        % $/barrel
  Vol2 = 0.36;
```

Define the correlation between the underlying asset prices of asset 1 and asset 2.

```
Corr = 0.42;
```

Define the spread option.

```
OptSpec = 'call';
Strike = 20;
```

Define the `RateSpec`.

```
rates = 0.05;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', rates, ...
'Compounding', Compounding, 'Basis', Basis)


RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9876
             Rates: 0.0500
          EndTimes: 0.2500
        StartTimes: 0
          EndDates: 734960
        StartDates: 734869
     ValuationDate: 734869
             Basis: 1
       EndMonthRule: 1
```

Define the `StockSpec` for the two assets.

```
StockSpec1 = stockspec(Vol1, Price1)
StockSpec2 = stockspec(Vol2, Price2)


StockSpec1 =

             FinObj: 'StockSpec'
              Sigma: 0.2900
         AssetPrice: 119.7000
       DividendType: []
    DividendAmounts: 0
    ExDividendDates: []


StockSpec2 =

             FinObj: 'StockSpec'
              Sigma: 0.3600
         AssetPrice: 93.2000
       DividendType: []
    DividendAmounts: 0
    ExDividendDates: []
```

Compute the European spread option price based on the Bjerksund-Stensland model.

```
Price = spreadbybjs(RateSpec, StockSpec1, StockSpec2, Settle, ...
Maturity, OptSpec, Strike, Corr)

Price =

   11.2000
```

References

Carmona, R., Durrleman, V., "Pricing and Hedging Spread Options," *SIAM Review*, Vol. 45, No. 4, pp. 627–685, Society for Industrial and Applied Mathematics, 2003.

Bjerksund, Petter, Stensland, Gunnar, "Closed form spread option valuation," Department of Finance, NHH, 2006.

See Also spreadsensbykirk | spreadbybjs | spreadbyfd | spreadbyls

Related Examples
- "Pricing European and American Spread Options" on page 3-38
- "Hedging Strategies Using Spread Options" on page 3-57

Concepts
- "Spread Option" on page 3-36

spreadbyfd

Purpose	Price European or American spread options using finite difference method

Syntax	`Price = spreadbyfd(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,`
	`OptSpec,Strike,Corr)`
	`Price = spreadbyfd( ___ ,Name,Value)`
	
	`[Price,PriceGrid,AssetPrice1,AssetPrice2,Times] = spreadbyfd(RateSpec,Std`
	`OptSpec,Strike,Corr)`
	`[Price,PriceGrid,AssetPrice1,AssetPrice2,`
	`   Times] = spreadbyfd( ___ ,Name,Value)`

Description	`Price =`
	`spreadbyfd(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,`
	`OptSpec,Strike,Corr)` returns the price of European or American call or put spread options using the Alternate Direction Implicit (ADI) finite difference method. The spread is between the asset defined in `StockSpec1` minus the asset defined in `StockSpec2`.
	
	`Price = spreadbyfd( ___ ,Name,Value)` returns the price of European or American call or put spread options using the Alternate Direction Implicit (ADI) finite difference method using optional name-value pair arguments. The spread is between the asset defined in `StockSpec1` minus the asset defined in `StockSpec2`.
	
	`[Price,PriceGrid,AssetPrice1,AssetPrice2,Times] =`
	`spreadbyfd(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,`
	`OptSpec,Strike,Corr)` returns the `Price`, `PriceGrid`, `AssetPrice1`, `AssetPrice2`, and `Times` for a European or American call or put spread options using the Alternate Direction Implicit (ADI) finite difference method. The spread is between the asset defined in `StockSpec1` minus the asset defined in `StockSpec2`.
	
	`[Price,PriceGrid,AssetPrice1,AssetPrice2, Times] =`
	`spreadbyfd( ___ ,Name,Value)` returns the `Price`, `PriceGrid`,

AssetPrice1, AssetPrice2, and Times for a European or American call or put spread options using the Alternate Direction Implicit (ADI) finite difference method using optional name-value pair arguments. The spread is between the asset defined in StockSpec1 minus the asset defined in StockSpec2.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec1 - Stock specification for underlying asset 1
structure

Stock specification for underlying asset 1. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

StockSpec2 - Stock specification for underlying asset 2
structure

Stock specification for underlying asset 2. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

Settle - Settlement dates for spread option

nonnegative integer | vector of nonnegative integers | date string |
cell array of date strings

Settlement dates for the spread option, specified as a date string or as
nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

Maturity - Maturity date for spread option

nonnegative integer | vector of nonnegative integers | date string |
cell array of date strings

Maturity date for spread option, specified as a date string or as
nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

OptSpec - Definition of option

string with values 'call' or 'put' | cell array of strings

Definition of option as 'call' or 'put', specified as a NINST-by-1 cell
array of strings.

Data Types
char | cell

Strike - Option strike price values

integer | vector of integers

Option strike price values, specified as an integer using a NINST-by-1
vector of strike price values.

Data Types
single | double

Corr - Correlation between underlying asset prices
integer | vector of integers

Correlation between underlying asset prices, specified as an integer using a NINST-by-1 vector.

Data Types
single | double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: [Price,PriceGrid,AssetPrice1,AssetPrice2,Times] = spreadbyfd(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,OptSpec,Str: 'AssetPriceMax','PriceGridSize','TimeGridSize','AmericanOpt',0)

'AssetPriceMax' - Maximum price for price grid boundary
[1000 1000] (default) | array

Maximum price for price grid boundary, specified by a 1-by-2 array. The first entry in the array corresponds to the first asset defined by StockSpec1 and the second entry corresponds to the second asset defined by StockSpec2.

For the finite difference method, the composition of the grid affects the quality of the output as well as the execution time. It is highly recommended to use the optional argumentsAssetPriceMax, PriceGridSize, and TimeGridSize to control the composition of the grid to ensure the quality of the output and a reasonable execution time.

Data Types
single | double

'PriceGridSize' - Size for finite difference grid
[300,300] (default) | array

Size for finite difference grid, specified by a 1-by-2 array. The first entry corresponds to the first asset defined by `StockSpec1` and the second entry corresponds to the second asset defined by `StockSpec2`.

For the finite difference method, the composition of the grid affects the quality of the output as well as the execution time. It is highly recommended to use the optional arguments`AssetPriceMax`, `PriceGridSize`, and `TimeGridSize` to control the composition of the grid to ensure the quality of the output and a reasonable execution time.

Data Types
`single` | `double`

'TimeGridSize' - Size of the time grid for finite difference grid
100 (default) | scalar | nonnegative integer

Size of the time grid for finite difference grid, specified as a nonnegative integer.

For the finite difference method, the composition of the grid affects the quality of the output as well as the execution time. It is highly recommended to use the optional arguments`AssetPriceMax`, `PriceGridSize`, and `TimeGridSize` to control the composition of the grid to ensure the quality of the output and a reasonable execution time.

Data Types
`single` | `double`

'AmericanOpt' - Option type
0 European (default) | scalar | vector of positive integers[0,1]

Option type, specified as `NINST`-by-1 positive integer scalar flags with values:

- `0` — European

- `1` — American

Data Types
`single` | `double`

Output Arguments

Price - Expected prices of spread option

vector

Expected prices of the spread option, returned as a NINST-by-1 vector.

PriceGrid - Grid containing prices calculated by finite difference method

array

Grid containing prices calculated by finite difference method, returned as a 3-D grid with a size of `PriceGridSize(1)` * `PriceGridSize(2)` * `TimeGridSize`. The price for $t = 0$ is contained in `PriceGrid(:, :, 1)`.

AssetPrice1 - Prices for first asset defined by `StockSpec1`

vector

Prices for first asset defined by `StockSpec1`, corresponding to the first dimension of `PriceGrid`, returned as a vector.

AssetPrice2 - Prices for second asset defined by `StockSpec2`

vector

Prices for second asset defined by `StockSpec2`, corresponding to the second dimension of `PriceGrid`, returned as a vector.

Times - Times corresponding to third dimension of `PriceGrid`

vector

Times corresponding to third dimension of `PriceGrid`, returned as a vector.

Examples

Compute the Price of a Spread Option Using the Alternate Direction Implicit (ADI) Finite Difference Method

Define the spread option dates.

```
Settle = '01-Jan-2012';
Maturity = '01-April-2012';
```

Define asset 1.

```
% Price and volatility of RBOB gasoline
  Price1gallon = 2.85;          % $/gallon
  Price1 = Price1gallon * 42;   % $/barrel
```

Define asset 2.

```
% Price and volatility of WTI crude oil
  Price2 = 93.20;        % $/barrel
  Vol2 = 0.36;
```

Define the correlation between the underlying asset prices of asset 1 and asset 2.

```
Corr = 0.42;
```

Define the spread option.

```
OptSpec = 'call';
Strike = 20;
```

Define the RateSpec.

```
rates = 0.05;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', rates, ...
'Compounding', Compounding, 'Basis', Basis)

RateSpec =

          FinObj: 'RateSpec'
     Compounding: -1
            Disc: 0.9876
           Rates: 0.0500
        EndTimes: 0.2500
      StartTimes: 0
        EndDates: 734960
      StartDates: 734869
```

```
       ValuationDate: 734869
               Basis: 1
         EndMonthRule: 1
```

Define the `StockSpec` for the two assets.

```
StockSpec1 = stockspec(Vol1, Price1)
StockSpec2 = stockspec(Vol2, Price2)

StockSpec1 =

               FinObj: 'StockSpec'
                Sigma: 0.2900
           AssetPrice: 119.7000
          DividendType: []
       DividendAmounts: 0
       ExDividendDates: []


StockSpec2 =

               FinObj: 'StockSpec'
                Sigma: 0.3600
           AssetPrice: 93.2000
          DividendType: []
       DividendAmounts: 0
       ExDividendDates: []
```

Compute the spread option price based on the Alternate Direction
Implicit (ADI) finite difference method.

```
[Price, PriceGrid, AssetPrice1, AssetPrice2, Times] = ...
 spreadbyfd(RateSpec, StockSpec1, StockSpec2, Settle, ...
 Maturity, OptSpec, Strike, Corr);
```

Display the price.

```
Price
```

```
Price =

    11.2147
```

Plot the finite difference grid.

```
mesh(AssetPrice1, AssetPrice2, PriceGrid(:, :, 1)');
    title('Spread Option Prices for Range of Underlying Prices');
    xlabel('Price of underlying asset 1');
    ylabel('Price of underlying asset 2');
    zlabel('Price of spread option');
```

References

Carmona, R., Durrleman, V., "Pricing and Hedging Spread Options," *SIAM Review*, Vol. 45, No. 4, pp. 627–685, Society for Industrial and Applied Mathematics, 2003.

Villeneuve, S., Zanette, A., "Parabolic ADI Methods for Pricing American Options on Two Stocks," *Mathematics of Operations Research*, Vol. 27, No. 1, pp. 121–149, INFORMS, 2002.

Ikonen, S., Toivanen, J., *Efficient Numerical Methods for Pricing American Options Under Stochastic Volatility*, Wiley InterScience, 2007.

See Also   spreadsensbyfd | spreadbybjs | spreadbykirk | spreadbyls

Related Examples
- "Pricing European and American Spread Options" on page 3-38
- "Hedging Strategies Using Spread Options" on page 3-57

Concepts
- "Spread Option" on page 3-36

spreadbyls

Purpose	Price European or American spread options using Monte Carlo simulations

Syntax	`Price = spreadbyls(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,`
	`OptSpec,Strike,Corr)`
	`Price = spreadbyls( ___ ,Name,Value)`
	
	`[Price,Paths,Times,Z] = spreadbyls(RateSpec,StockSpec1,StockSpec2,Settle,`
	`OptSpec,Strike,Corr)`
	`[Price,Paths,Times,Z] = spreadbyls( ___ ,Name,Value)`

Description	`Price = spreadbyls(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,OptSpec,Strike,Corr)` returns the price of a European or American call or put spread option using Monte Carlo simulations. `Price = spreadbyls( ___ ,Name,Value)` returns the price of a European or American call or put spread option using Monte Carlo simulations using optional name-value pair arguments. `[Price,Paths,Times,Z] = spreadbyls(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,OptSpec,Strike,Corr)` returns the `Price`, `Paths`, `Times`, and `Z` of a European or American call or put spread option using Monte Carlo simulations. `[Price,Paths,Times,Z] = spreadbyls( ___ ,Name,Value)` returns the `Price`, `Paths`, `Times`, and `Z` of a European or American call or put spread option using Monte Carlo simulations using optional name-value pair arguments.

Input Arguments

RateSpec - Interest-rate term structure

structure

Interest-rate term structure (annualized and continuously compounded), specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec1 - Stock specification for underlying asset 1

structure

Stock specification for underlying asset 1. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

StockSpec2 - Stock specification for underlying asset 2

structure

Stock specification for underlying asset 2. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

Settle - Settlement date for spread option

date string | nonnegative scalar integer

Settlement date for the spread option specified, as a date string or nonnegative scalar integer.

Data Types
char

Maturity - Maturity date for spread option
date string | nonnegative scalar integer

Maturity date for spread option, specified as a date string or a nonnegative scalar integer.

Data Types
char

OptSpec - Definition of option
string with values `'call'` or `'put'`

Definition of option as `'call'` or `'put'`, specified as a string.

Data Types
char

Strike - Option strike price value
nonnegative scalar integer

Option strike price value, specified, as a nonnegative scalar integer.

Data Types
single | double

Corr - Correlation between underlying asset prices
scalar integer

Correlation between underlying asset prices, specified as an scalar integer.

Data Types
single | double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments.
Name is the argument name and Value is the corresponding
value. Name must appear inside single quotes (' '). You can
specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

Example: Price =
spreadbyls(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,OptSpec,Stri

'AmericanOpt' - Option type
0 European (default) | scalar with value [0,1]

Option type, specified as an integer scalar flag with value:

- 0 — European

- 1 — American

For American options, the Longstaff-Schwartz least squares method is
used to calculate the early exercise premium.

Data Types
single | double

'NumTrials' - Scalar number of independent sample paths
1000 (default) | nonnegative scalar integer

Scalar number of independent sample paths (simulation trials),
specified as a nonnegative integer.

Data Types
single | double

'NumPeriods' - Scalar number of simulation periods per trial
100 (default) | nonnegative scalar integer

Scalar number of simulation periods per trial, specified as a nonnegative
integer. NumPeriods is considered only when pricing European basket

options. For American spread options, `NumPeriod` is equal to the number of exercise days during the life of the option.

Data Types
`single | double`

'Z' - Time series array of dependent random variates
vector

Time series array of dependent random variates specified as a `NumPeriods`-by-2-by-`NumTrials` 3-D array. The Z value generates the Brownian motion vector (i.e., Wiener processes) that drives the simulation.

Data Types
`single | double`

'Antithetic' - Indicator for antithetic sampling
false (default) | scalar logical flag with value of true or false

Indicator for antithetic sampling, specified with a value of `true` or `false`.

Data Types
`logical`

Output Arguments

Price - Expected price of spread option
scalar

Expected price of the spread option, returned as a 1-by-1 scalar.

Paths - Simulated paths of correlated state variables
vector

Simulated paths of correlated state variables, returned as a `NumPeriods + 1`-by-2-by-`NumTrials` 3-D time series array. Each row of `Paths` is the transpose of the state vector $X(t)$ at time t for a given trial.

Times - Observation times associated with simulated paths
vector

Observation times associated with simulated paths, returned as a `NumPeriods + 1`-by-1 column vector of observation times associated with the simulated paths. Each element of `Times` is associated with the corresponding row of `Paths`.

Z - Time series array of dependent random variates
vector

Time series array of dependent random variates, returned as a `NumPeriods`-by-2-by-`NumTrials` 3-D array when Z is specified as an input argument. If the Z input argument is not specified, then the Z output argument contains the random variates generated internally.

Examples **Compute the Price of a Spread Option Using Monte Carlo Simulation**

Define the spread option dates.

```
Settle = '01-Jan-2012';
Maturity = '01-April-2012';
```

Define asset 1.

```
% Price and volatility of RBOB gasoline
  Price1gallon = 2.85;        % $/gallon
  Price1 = Price1gallon * 42;   % $/barrel
  Vol1 = 0.29;
```

Define asset 2.

```
 % Price and volatility of WTI crude oil
  Price2 = 93.20;        % $/barrel
  Vol2 = 0.36;
```

Define the correlation between the underlying asset prices of asset 1 and asset 2.

```
Corr = 0.42;
```

Define the spread option.

```
OptSpec = 'call';
Strike = 20;
```

Define the `RateSpec`.

```
rates = 0.05;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', rates, ...
'Compounding', Compounding, 'Basis', Basis)


RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9876
             Rates: 0.0500
          EndTimes: 0.2500
        StartTimes: 0
          EndDates: 734960
        StartDates: 734869
     ValuationDate: 734869
             Basis: 1
       EndMonthRule: 1
```

Define the `StockSpec` for the two assets.

```
StockSpec1 = stockspec(Vol1, Price1)
StockSpec2 = stockspec(Vol2, Price2)


StockSpec1 =

            FinObj: 'StockSpec'
             Sigma: 0.2900
         AssetPrice: 119.7000
```

```
          DividendType: []
      DividendAmounts: 0
      ExDividendDates: []


StockSpec2 =

               FinObj: 'StockSpec'
                Sigma: 0.3600
           AssetPrice: 93.2000
         DividendType: []
      DividendAmounts: 0
      ExDividendDates: []
```

Compute the spread option price using Monte Carlo simulation based on the Longstaff-Schwartz model.

```
Price = spreadbyls(RateSpec, StockSpec1, StockSpec2, Settle, ...
Maturity, OptSpec, Strike, Corr)

Price =

   10.9162
```

References

Carmona, R., Durrleman, V., "Pricing and Hedging Spread Options," *SIAM Review*, Vol. 45, No. 4, pp. 627–685, Society for Industrial and Applied Mathematics, 2003.

See Also   spreadsensbyls | spreadbybjs | spreadbyfd | spreadbykirk

Related Examples
- "Pricing European and American Spread Options" on page 3-38
- "Hedging Strategies Using Spread Options" on page 3-57

Concepts
- "Spread Option" on page 3-36

spreadsensbykirk

Purpose	Calculate European spread option prices and sensitivities using Kirk pricing model

Syntax	PriceSens = spreadbykirk(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr) PriceSens = spreadsensbykirk(___ ,Name,Value)

Description   PriceSens =
spreadbykirk(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,
OptSpec,Strike,Corr) returns the European spread option prices and
sensitivities using the Kirk pricing model.

PriceSens = spreadsensbykirk(___ ,Name,Value) returns the
European spread option prices and sensitivities using the Kirk pricing
model with optional name-value pair arguments.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously
compounded), specified by the RateSpec obtained from intenvset. For
information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec1 - Stock specification for underlying asset 1
structure

Stock specification for underlying asset 1. For information on the stock
specification, see stockspec.

stockspec can handle other types of underlying assets. For example,
for physical commodities the price is represented by StockSpec.Asset,
the volatility is represented by StockSpec.Sigma, and the convenience
yield is represented by StockSpec.DividendAmounts.

Data Types
struct

StockSpec2 - Stock specification for underlying asset 2
structure

Stock specification for underlying asset 2. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

Settle - Settlement dates for spread option
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates for the spread option, specified as a date string or as nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

Maturity - Maturity date for spread option
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Maturity date for spread option, specified as a date string or as nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

OptSpec - Definition of option
string with values 'call' or 'put' | cell array of strings

spreadsensbykirk

Definition of option as `'call'` or `'put'`, specified as a NINST-by-1 cell array of strings.

Data Types
`char` | `cell`

Strike - Option strike price values
integer | vector of integers

Option strike price values, specified as an integer using as NINST-by-1 vector of strike price values.

If `Strike` is equal to zero, this function computes the price and sensitivities of an exchange option.

Data Types
`single` | `double`

Corr - Correlation between underlying asset prices
integer | vector of integers

Correlation between underlying asset prices, specified as an integer using as NINST-by-1 vector.

Data Types
`single` | `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `PriceSens = spreadsensbykirk(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,OptSpec,S`

'OutSpec' - Define outputs
`{'Price'}` (default) | string with values `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'` | cell array of strings

with values `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'`

Define outputs specifying NOUT- by-1 or 1-by-NOUT cell array of strings with possible values of `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'`.

`OutSpec = {'All'}` specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying `OutSpec` to include each sensitivity:

Example: `OutSpec = {'delta','gamma','vega','lambda','rho','theta','price'}`

Data Types
char | cell

Output Arguments

PriceSens - Expected price or sensitivities values of spread option
vector

Expected price or sensitivities values of the spread option, returned as a NINST-by-1 or NINST-by-2 vector.

Examples

Compute the Price and Sensitivities of a Spread Option Using the Kirk Model

Define the spread option dates.

```
Settle = '01-Jun-2012';
Maturity = '01-Sep-2012';
```

Define asset 1.

```
% Price and volatility of RBOB gasoline
Price1gallon = 2.85;        % $/gallon
Price1 = Price1gallon * 42;  % $/barrel
Vol1 = 0.29;
```

Define asset 2.

```
% Price and volatility of WTI crude oil
  Price2 = 93.20;        % $/barrel
  Vol2 = 0.36;
```

Define the correlation between the underlying asset prices of asset 1 and asset 2.

```
Corr = 0.42;
```

Define the spread option.

```
OptSpec = 'call';
Strike = 20;
```

Define the RateSpec.

```
rates = 0.05;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', rates, ...
'Compounding', Compounding, 'Basis', Basis)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9876
             Rates: 0.0500
          EndTimes: 0.2500
        StartTimes: 0
          EndDates: 735113
        StartDates: 735021
     ValuationDate: 735021
             Basis: 1
       EndMonthRule: 1
```

Define the StockSpec for the two assets.

```
StockSpec1 = stockspec(Vol1, Price1)
StockSpec2 = stockspec(Vol2, Price2)


StockSpec1 =

              FinObj: 'StockSpec'
               Sigma: 0.2900
          AssetPrice: 119.7000
        DividendType: []
     DividendAmounts: 0
     ExDividendDates: []


StockSpec2 =

              FinObj: 'StockSpec'
               Sigma: 0.3600
          AssetPrice: 93.2000
        DividendType: []
     DividendAmounts: 0
     ExDividendDates: []
```

Compute the spread option price and sensitivities based on the Kirk model.

```
OutSpec = {'Price', 'Delta', 'Gamma'};
[Price, Delta, Gamma] = spreadsensbykirk(RateSpec, StockSpec1, StockSpec2, Settle, ...
Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec)

Price =

   11.1904


Delta =

    0.6722   -0.6067
```

spreadsensbykirk

```
Gamma =

    0.0191    0.0217
```

References

Carmona, R., Durrleman, V., "Pricing and Hedging Spread Options," *SIAM Review*, Vol. 45, No. 4, pp. 627–685, Society for Industrial and Applied Mathematics, 2003.

See Also   spreadbykirk | spreadbybjs | spreadbyfd | spreadbyls

Related Examples
- "Pricing European and American Spread Options" on page 3-38
- "Hedging Strategies Using Spread Options" on page 3-57

Concepts
- "Spread Option" on page 3-36

Purpose	Calculate European spread option prices and sensitivities using Bjerksund-Stensland pricing model

Syntax

```
PriceSens = spreadbybjs(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,
OptSpec,Strike,Corr)
PriceSens = spreadsensbybjs( ___ ,Name,Value)
```

Description

PriceSens = spreadbybjs(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr) returns the European spread option prices and sensitivities using the Bjerksund-Stensland pricing model.

PriceSens = spreadsensbybjs(___ ,Name,Value) returns the European spread option prices and sensitivities using the Bjerksund-Stensland pricing model with optional name-value pair arguments.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec1 - Stock specification for underlying asset 1
structure

Stock specification for underlying asset 1. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

StockSpec2 - Stock specification for underlying asset 2
structure

Stock specification for underlying asset 2. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

Settle - Settlement dates for spread option
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates for the spread option, specified as a date string or as nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

Maturity - Maturity date for spread option
nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Maturity date for spread option, specified as a date string or as nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

OptSpec - Definition of option
string with values 'call' or 'put' | cell array of strings

Definition of option as `'call'` or `'put'`, specified as a `NINST-by-1` cell array of strings.

Data Types
`char` | `cell`

Strike - Option strike price values
integer | vector of integers

Option strike price values, specified as an integer using a `NINST-by-1` vector of strike price values.

If `Strike` is equal to zero the function computes the price and sensitivities of an exchange option.

Data Types
`single` | `double`

Corr - Correlation between underlying asset prices
integer | vector of integers

Correlation between underlying asset prices, specified as an integer using a `NINST-by-1` vector.

Data Types
`single` | `double`

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `PriceSens = spreadsensbykirk(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,OptSpe`

'OutSpec' - Define outputs
`{'Price'}` (default) | string with values `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'` | cell array of strings

with values `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'`

Define outputs specifying NOUT- by-1 or 1-by-NOUT cell array of strings with possible values of `'Price'`, `'Delta'`, `'Gamma'`, `'Vega'`, `'Lambda'`, `'Rho'`, `'Theta'`, and `'All'`.

`OutSpec = {'All'}` specifies that the output should be `Delta`, `Gamma`, `Vega`, `Lambda`, `Rho`, `Theta`, and `Price`, in that order. This is the same as specifying `OutSpec` to include each sensitivity:

Example: `OutSpec = {'delta','gamma','vega','lambda','rho','theta','price'}`

Data Types
char | cell

Output Arguments

PriceSens - Expected prices or sensitivities values of spread option
vector

Expected prices or sensitivities values of the spread option, returned as a NINST-by-1 or NINST-by-2 vector.

Examples

Compute the Price and Sensitivities of a Spread Option Using the Bjerksund-Stensland Model

Define the spread option dates.

```
Settle = '01-Jun-2012';
Maturity = '01-Sep-2012';
```

Define asset 1.

```
% Price and volatility of RBOB gasoline
  Price1gallon = 2.85;          % $/gallon
  Price1 = Price1gallon * 42;   % $/barrel
  Vol1 = 0.29;
```

Define asset 2.

```
% Price and volatility of WTI crude oil
  Price2 = 93.20;           % $/barrel
  Vol2 = 0.36;
```

Define the correlation between the underlying asset prices of asset 1 and asset 2.

```
Corr = 0.42;
```

Define the spread option.

```
OptSpec = 'call';
Strike = 20;
```

Define the `RateSpec`.

```
rates = 0.05;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', rates, ...
'Compounding', Compounding, 'Basis', Basis)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9876
             Rates: 0.0500
          EndTimes: 0.2500
        StartTimes: 0
          EndDates: 735113
        StartDates: 735021
     ValuationDate: 735021
             Basis: 1
       EndMonthRule: 1
```

Define the `StockSpec` for the two assets.

```
StockSpec1 = stockspec(Vol1, Price1)
StockSpec2 = stockspec(Vol2, Price2)


StockSpec1 =

              FinObj: 'StockSpec'
               Sigma: 0.2900
          AssetPrice: 119.7000
         DividendType: []
     DividendAmounts: 0
      ExDividendDates: []


StockSpec2 =

              FinObj: 'StockSpec'
               Sigma: 0.3600
          AssetPrice: 93.2000
         DividendType: []
     DividendAmounts: 0
      ExDividendDates: []
```

Compute the spread option price and sensitivities based on the Kirk model.

```
OutSpec = {'Price', 'Delta', 'Gamma'};
[Price, Delta, Gamma] = spreadsensbybjs(RateSpec, StockSpec1, StockSpec2, Settle, ...
Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec)

Price =

   11.2000


Delta =

    0.6737   -0.6082
```

```
Gamma =

    0.0190    0.0216
```

References

Carmona, R., Durrleman, V., "Pricing and Hedging Spread Options," *SIAM Review*, Vol. 45, No. 4, pp. 627–685, Society for Industrial and Applied Mathematics, 2003.

Bjerksund, Petter, Stensland, Gunnar, "Closed form spread option valuation," Department of Finance, NHH, 2006.

See Also   spreadbykirk | spreadbybjs | spreadbyfd | spreadbyls

Related Examples
- "Pricing European and American Spread Options" on page 3-38
- "Hedging Strategies Using Spread Options" on page 3-57

Concepts
- "Spread Option" on page 3-36

Purpose	Calculate price and sensitivities for European or American spread options using Monte Carlo simulations

Syntax

```
PriceSens = spreadsensbyls(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,
OptSpec,Strike,Corr)
PriceSens = spreadsensbyls( ___ ,Name,Value)

[PriceSens,Paths,Times,Z] = spreadsensbyls(RateSpec,StockSpec1,StockSpec2,
OptSpec,Strike,Corr)
[PriceSens,Paths,Times,Z] = spreadsensbyls( ___ ,Name,Value)
```

Description

`PriceSens = spreadsensbyls(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr)` returns the price of a European or American call or put spread option using Monte Carlo simulations.

`PriceSens = spreadsensbyls( ___ ,Name,Value)` returns the price of a European or American call or put spread option using Monte Carlo simulations with optional name-value pair arguments.

`[PriceSens,Paths,Times,Z] = spreadsensbyls(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr)` returns the `PriceSens`, `Paths`, `Times`, and `Z` of a European or American call or put spread option using Monte Carlo simulations.

`[PriceSens,Paths,Times,Z] = spreadsensbyls( ___ ,Name,Value)` returns the `PriceSens`, `Paths`, `Times`, and `Z` of a European or American call or put spread option using Monte Carlo simulations with optional name-value pair arguments.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the `RateSpec` obtained from `intenvset`. For information on the interest-rate specification, see `intenvset`.

Data Types
struct

StockSpec1 - Stock specification for underlying asset 1
structure

Stock specification for underlying asset 1. For information on the stock specification, see `stockspec`.

`stockspec` can handle other types of underlying assets. For example, for physical commodities the price is represented by `StockSpec.Asset`, the volatility is represented by `StockSpec.Sigma`, and the convenience yield is represented by `StockSpec.DividendAmounts`.

Data Types
struct

StockSpec2 - Stock specification for underlying asset 2
structure

Stock specification for underlying asset 2. For information on the stock specification, see `stockspec`.

`stockspec` can handle other types of underlying assets. For example, for physical commodities the price is represented by `StockSpec.Asset`, the volatility is represented by `StockSpec.Sigma`, and the convenience yield is represented by `StockSpec.DividendAmounts`.

Data Types
struct

Settle - Settlement date for spread option
date string | nonnegative scalar integer

Settlement date for the spread option, specified as a date string or as a nonnegative scalar integer.

Data Types
char

Maturity - Maturity date for spread option
date string | nonnegative scalar integer

Maturity date for spread option, specified as a date string or as a nonnegative scalar integer.

Data Types
char

OptSpec - Definition of option
string with values `'call'` or `'put'`

Definition of option as `'call'` or `'put'`, specified as a string.

Data Types
char

Strike - Option strike price value
nonnegative scalar integer

Option strike price value, specified as a scalar integer.

Data Types
single | double

Corr - Correlation between underlying asset prices
scalar integer

Correlation between underlying asset prices, specified as a scalar integer.

Data Types
single | double

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Example: `PriceSens = spreadbyls(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,OptSpec,Str`

'AmericanOpt' - Option type
0 European (default) | scalar with values [0,1]

Option type, specified as a scalar integer flag with values:

- `0` — European

- `1` — American

For American options, the Longstaff-Schwartz least squares method is used to calculate the early exercise premium.

Data Types
single | double

'NumTrials' - Number of independent sample paths
1000 (default) | nonnegative scalar integer

Number of independent sample paths (simulation trials) specified as a nonnegative scalar integer.

Data Types
single | double

'NumPeriods' - Number of simulation periods per trial
100 (default) | nonnegative scalar integer

Number of simulation periods per trial, specified as a nonnegative scalar integer. `NumPeriods` is considered only when pricing European basket options. For American spread options, `NumPeriods` is equal to the number of exercise days during the life of the option.

Data Types
single | double

'Z' - Time series array of dependent random variates
vector

Time series array of dependent random variates, specified as a NumPeriods-by-2-by-NumTrials 3-D array. The Z value generates the Brownian motion vector (i.e., Wiener processes) that drives the simulation.

Data Types
single | double

'Antithetic' - Indicator for antithetic sampling
false (default) | logical flag with value of true or false

Indicator for antithetic sampling, specified with value of true or false.

Data Types
logical

'OutSpec' - Define outputs
{'Price'} (default) | string with values 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', and 'All' | cell array of strings with values 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', and 'All'

Define outputs specifying NOUT- by-1 or 1-by-NOUT cell array of strings with possible values of 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', and 'All'.

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec to include each sensitivity:

Example: OutSpec = {'delta','gamma','vega','lambda','rho','theta','price'}

Data Types
char | cell

Output Arguments

PriceSens - Expected price and sensitivities of spread option
vector

Expected price and sensitivities of the spread option, returned as a
`NINST`-by-1 vector.

Paths - Simulated paths of correlated state variables
vector

Simulated paths of correlated state variables, returned as a `NumPeriods`
+ 1-by-2-by-`NumTrials` 3-D time series array. Each row of `Paths` is the
transpose of the state vector $X(t)$ at time t for a given trial.

Times - Observation times associated with simulated paths
vector

Observation times associated with simulated paths, returned as a
`NumPeriods` + 1-by-1 column vector of observation times associated
with the simulated paths. Each element of `Times` is associated with
the corresponding row of `Paths`.

Z - Time series array of dependent random variates
vector

Time series array of dependent random variates, returned as a
`NumPeriods`-by-2-by-`NumTrials` 3-D array when `Z` is specified as an
input argument. If the `Z` input argument is not specified, then the `Z`
output argument contains the random variates generated internally.

Examples

Compute the Price and Sensitivities of a Spread Option Using Monte Carlo Simulation

Define the spread option dates.

```
Settle = '01-Jun-2012';
Maturity = '01-Sep-2012';
```

Define asset 1.

```
% Price and volatility of RBOB gasoline
  Price1gallon = 2.85;         % $/gallon
  Price1 = Price1gallon * 42;  % $/barrel
  Vol1 = 0.29;
```

Define asset 2.

```
% Price and volatility of WTI crude oil
  Price2 = 93.20;        % $/barrel
  Vol2 = 0.36;
```

Define the correlation between the underlying asset prices of asset 1 and asset 2.

```
Corr = 0.42;
```

Define the spread option.

```
OptSpec = 'call';
Strike = 20;
```

Define the RateSpec.

```
rates = 0.05;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', rates, ...
'Compounding', Compounding, 'Basis', Basis)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9876
             Rates: 0.0500
          EndTimes: 0.2500
        StartTimes: 0
          EndDates: 735113
```

```
        StartDates: 735021
    ValuationDate: 735021
            Basis: 1
      EndMonthRule: 1
```

Define the `StockSpec` for the two assets.

```
StockSpec1 = stockspec(Vol1, Price1)
StockSpec2 = stockspec(Vol2, Price2)

StockSpec1 =

              FinObj: 'StockSpec'
               Sigma: 0.2900
          AssetPrice: 119.7000
         DividendType: []
     DividendAmounts: 0
     ExDividendDates: []


StockSpec2 =

              FinObj: 'StockSpec'
               Sigma: 0.3600
          AssetPrice: 93.2000
         DividendType: []
     DividendAmounts: 0
     ExDividendDates: []
```

Compute the spread option price and sensitivities using Monte Carlo simulation based on the Longstaff-Schwartz model.

```
OutSpec = {'Price', 'Delta', 'Gamma'};
[Price, Delta, Gamma] = spreadsensbyls(RateSpec, StockSpec1, StockSpec2, ...
Settle, Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec)

Price =
```

```
        11.6644


Delta =

    0.6663   -0.5968


Gamma =

    0.0173    0.0195
```

References

Carmona, R., Durrleman, V., "Pricing and Hedging Spread Options," *SIAM Review*, Vol. 45, No. 4, pp. 627–685, Society for Industrial and Applied Mathematics, 2003.

See Also   spreadbyls | spreadbybjs | spreadbyfd | spreadbykirk

Related Examples
- "Pricing European and American Spread Options" on page 3-38
- "Hedging Strategies Using Spread Options" on page 3-57

Concepts
- "Spread Option" on page 3-36

Purpose	Calculate price and sensitivities of European or American spread options using finite difference method
Syntax	PriceSens = spreadbyfd(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr) PriceSens = spreadbyfd(___ ,Name,Value) [PriceSens,PriceGrid,AssetPrice1,AssetPrice2,Times] = spreadbyfd(RateS OptSpec,Strike,Corr) [PriceSens,PriceGrid,AssetPrice1,AssetPrice2, Times] = spreadbyfd(___ ,Name,Value)
Description	PriceSens = spreadbyfd(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr) returns the price and sensitivities of European or American call or put spread options using the Alternate Direction Implicit (ADI) finite difference method. The spread is between the asset defined in StockSpec1 minus the asset defined in StockSpec2. PriceSens = spreadbyfd(___ ,Name,Value) returns the price and sensitivities of European or American call or put spread options using the Alternate Direction Implicit (ADI) finite difference method using optional name-value pair arguments. The spread is between the asset defined in StockSpec1 minus the asset defined in StockSpec2. [PriceSens,PriceGrid,AssetPrice1,AssetPrice2,Times] = spreadbyfd(RateSpec,StockSpec1,StockSpec2,Settle,Maturity, OptSpec,Strike,Corr) returns the PriceSens, PriceGrid, AssetPrice1, AssetPrice2, and Times for European or American call or put spread options using the Alternate Direction Implicit (ADI) finite difference method. The spread is between the asset defined in StockSpec1 minus the asset defined in StockSpec2. [PriceSens,PriceGrid,AssetPrice1,AssetPrice2, Times] = spreadbyfd(___ ,Name,Value) returns the PriceSens, PriceGrid,

AssetPrice1, AssetPrice2, and Times for European or American call or put spread options using the Alternate Direction Implicit (ADI) finite difference method using optional name-value pair arguments. The spread is between the asset defined in StockSpec1 minus the asset defined in StockSpec2.

Input Arguments

RateSpec - Interest-rate term structure
structure

Interest-rate term structure (annualized and continuously compounded), specified by the RateSpec obtained from intenvset. For information on the interest-rate specification, see intenvset.

Data Types
struct

StockSpec1 - Stock specification for underlying asset 1
structure

Stock specification for underlying asset 1. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

StockSpec2 - Stock specification for underlying asset 2
structure

Stock specification for underlying asset 2. For information on the stock specification, see stockspec.

stockspec can handle other types of underlying assets. For example, for physical commodities the price is represented by StockSpec.Asset, the volatility is represented by StockSpec.Sigma, and the convenience yield is represented by StockSpec.DividendAmounts.

Data Types
struct

Settle - Settlement dates for spread option

nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Settlement dates for the spread option, specified as date strings or as nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

Maturity - Maturity date for spread option

nonnegative integer | vector of nonnegative integers | date string | cell array of date strings

Maturity date for spread option, specified as date strings or as nonnegative integers using a NINST-by-1 vector or cell array of dates.

Data Types
char | cell

OptSpec - Definition of option

string with values 'call' or 'put' | cell array of strings

Definition of option as 'call' or 'put', specified as a NINST-by-1 cell array of strings.

Data Types
char | cell

Strike - Option strike price values

integer | vector of integers

Option strike price values, specified as an integer using a NINST-by-1 vector of strike price values.

Data Types
single | double

Corr - Correlation between underlying asset prices
integer | vector of integers

Correlation between underlying asset prices, specified as an integer using aNINST-by-1 vector.

Data Types
single | double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name, Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example:
[PriceSens,PriceGrid,AssetPrice1,AssetPrice2,Times] = spreadbyfd(RateSpec,StockSpec1,StockSpec2,Settle,Maturity,OptSpec,Strike,

'OutSpec' - Define outputs
{'Price'} (default) | string with values 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', and 'All' | cell array of strings with values 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', and 'All'

Define outputs specifying NOUT- by-1 or 1-by-NOUT cell array of strings with possible values of 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', and 'All'.

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec to include each sensitivity:

Example: OutSpec = {'delta','gamma','vega','lambda','rho','theta','price'}

Data Types
char | cell

'AssetPriceMax' - Maximum price for price grid boundary
[1000 1000] (default) | array

Maximum price for price grid boundary, specified by a 1-by-2 array. The first entry in the array corresponds to the first asset defined by StockSpec1 and the second entry corresponds to the second asset defined by StockSpec2.

For the finite difference method, the composition of the grid affects the quality of the output as well as the execution time. It is highly recommended to use the optional argumentsAssetPriceMax, PriceGridSize, and TimeGridSize to control the composition of the grid to ensure the quality of the output and a reasonable execution time.

Data Types
single | double

'PriceGridSize' - Size for finite difference grid
[300,300] (default) | array

Size for finite difference grid, specified by a 1-by-2 array. The first entry corresponds to the first asset defined by StockSpec1 and the second entry corresponds to the second asset defined by StockSpec2.

For the finite difference method, the composition of the grid affects the quality of the output as well as the execution time. It is highly recommended to use the optional argumentsAssetPriceMax, PriceGridSize, and TimeGridSize to control the composition of the grid to ensure the quality of the output and a reasonable execution time.

Data Types
single | double

'TimeGridSize' - Size of time grid for finite difference grid
100 (default) | scalar | nonnegative integer

Size of time grid for finite difference grid, specified as a nonnegative integer.

For the finite difference method, the composition of the grid affects the quality of the output as well as the execution time. It is

spreadsensbyfd

highly recommended to use the optional arguments `AssetPriceMax`, `PriceGridSize`, and `TimeGridSize` to control the composition of the grid to ensure the quality of the output and a reasonable execution time.

Data Types
`single | double`

'AmericanOpt' - Option type
0 European (default) | scalar | vector of positive integers`[0,1]`

Option type, specified as `NINST`-by-1 positive integer scalar flags with values:

- `0` — European
- `1` — American

Data Types
`single | double`

Output Arguments

PriceSens - Expected prices and sensitivities of spread option
vector

Expected prices and sensitivities of the spread option, returned as an `NINST`-by-1 vector.

PriceGrid - Grid containing prices calculated by finite difference method
array

Grid containing prices calculated by finite difference method, returned as a 3-D grid with a size of `PriceGridSize(1) * PriceGridSize(2) * TimeGridSize`. The price for t = 0 is contained in `PriceGrid(:, :, 1)`.

AssetPrice1 - Prices for first asset defined by `StockSpec1`
vector

Prices for first asset defined by `StockSpec1`, corresponding to the first dimension of `PriceGrid`, returned as a vector.

AssetPrice2 - Prices for second asset defined by `StockSpec2`

vector

Prices for second asset defined by `StockSpec2`, corresponding to the second dimension of `PriceGrid`, returned as a vector.

Times - Times corresponding to third dimension of `PriceGrid`
vector

Times corresponding to third dimension of `PriceGrid`, returned as a vector.

Examples   **Compute the Price of a Spread Option Using the Alternate Direction Implicit (ADI) Finite Difference Method**

Define the spread option dates.

```
Settle = '01-Jun-2012';
Maturity = '01-Sep-2012';
```

Define asset 1.

```
% Price and volatility of RBOB gasoline
  Price1gallon = 2.85;        % $/gallon
  Price1 = Price1gallon * 42;   % $/barrel
  Vol1 = 0.29;
```

Define asset 2.

```
 % Price and volatility of WTI crude oil
  Price2 = 93.20;        % $/barrel
  Vol2 = 0.36;
```

Define the correlation between the underlying asset prices of asset 1 and asset 2.

```
Corr = 0.42;
```

Define the spread option.

```
OptSpec = 'call';
```

```
Strike = 20;
```

Define the `RateSpec`.

```
rates = 0.05;
Compounding = -1;
Basis = 1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle, ...
'EndDates', Maturity, 'Rates', rates, ...
'Compounding', Compounding, 'Basis', Basis)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.9876
             Rates: 0.0500
           EndTimes: 0.2500
         StartTimes: 0
           EndDates: 735113
         StartDates: 735021
     ValuationDate: 735021
             Basis: 1
       EndMonthRule: 1
```

Define the `StockSpec` for the two assets.

```
StockSpec1 = stockspec(Vol1, Price1)
StockSpec2 = stockspec(Vol2, Price2)

StockSpec1 =

              FinObj: 'StockSpec'
               Sigma: 0.2900
          AssetPrice: 119.7000
         DividendType: []
     DividendAmounts: 0
     ExDividendDates: []
```

```
StockSpec2 =

              FinObj: 'StockSpec'
               Sigma: 0.3600
          AssetPrice: 93.2000
        DividendType: []
    DividendAmounts: 0
    ExDividendDates: []
```

Compute the spread option price and sensitivities based on the Alternate Direction Implicit (ADI) finite difference method.

```
OutSpec = {'Price', 'Delta', 'Gamma'};
[Price, Delta, Gamma, PriceGrid, AssetPrice1, AssetPrice2, Times] = ...
spreadsensbyfd(RateSpec, StockSpec1, StockSpec2, Settle, ...
Maturity, OptSpec, Strike, Corr, 'OutSpec', OutSpec);
```

Display the price and sensitivities.

```
Price
Delta
Gamma

Price =

   11.2147


Delta =

    0.6695   -0.6077


Gamma =

    0.0244    0.0405
```

spreadsensbyfd

Plot the finite difference grid.

```
mesh(AssetPrice1, AssetPrice2, PriceGrid(:, :, 1)');
    title('Spread Option Prices for Range of Underlying Prices');
    xlabel('Price of underlying asset 1');
    ylabel('Price of underlying asset 2');
    zlabel('Price of spread option');
```


References

Carmona, R., Durrleman, V., "Pricing and Hedging Spread Options,"
SIAM Review, Vol. 45, No. 4, pp. 627–685, Society for Industrial and
Applied Mathematics, 2003.

Villeneuve, S., Zanette, A., "Parabolic ADI Methods for Pricing American Options on Two Stocks," *Mathematics of Operations Research*, Vol. 27, No. 1, pp. 121–149, INFORMS, 2002.

Ikonen, S., Toivanen, J., *Efficient Numerical Methods for Pricing American Options Under Stochastic Volatility*, Wiley InterScience, 2007.

See Also   spreadsensbyfd | spreadbybjs | spreadbykirk | spreadbyls

Related Examples
- "Pricing European and American Spread Options" on page 3-38
- "Hedging Strategies Using Spread Options" on page 3-57

Concepts
- "Spread Option" on page 3-36

stockoptspec

Purpose	Specify European stock option structure

Syntax

```
[StockOptSpec] = stockoptspec(OptPrice, Strike, Settle,
Maturity, OptSpec, InterpMethod)
```

Arguments

OptPrice	NINST-by-1 vector of European option prices.
Strike	NINST-by-1 vector of strike prices.
Settle	Scalar date marking the settlement date.
Maturity	NINST-by-1 vector of maturity dates.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'.
InterpMethod	(Optional) Method of interpolation to use for option prices. InterpMethod is [{'price'}	'vol']. The default is 'price'. By specifying 'vol', implied volatilities will be used for interpolation purposes. The interpolated values will then be used to calculate the implicit interpolated prices.

Description

```
[StockOptSpec] = stockoptspec(OptPrice, Strike, Settle,
Maturity, OptSpec, InterpMethod)
```
creates a structure encapsulating the properties of a stock option structure.

Examples

Consider the following data quoted from liquid options in the market with varying strikes and maturity. You specify these parameters in MATLAB as follows:

```
Settle =   '01/01/06';

Maturity =   ['07/01/06';
    '07/01/06';
    '07/01/06';
    '07/01/06';
```

```
            '01/01/07';
            '01/01/07';
            '01/01/07';
            '01/01/07';
            '07/01/07';
            '07/01/07';
            '07/01/07';
            '07/01/07';
            '01/01/08';
            '01/01/08';
            '01/01/08';
            '01/01/08'];

Strike = [113;
    101;
    100;
     88;
    128;
    112;
    100;
     78;
    144;
    112;
    100;
     69;
    162;
    112;
    100;
     61];

OptPrice =[              0;
    4.807905472659144;
    1.306321897011867;
    0.048039195057173;
                    0;
    2.310953054191461;
    1.421950392866235;
```

```
                        0.020414826276740;
                                         0;
                        5.091986935627730;
                        1.346534812295291;
                        0.005101325584140;
                                         0;
                        8.047628153217246;
                        1.219653432150932;
                        0.001041436654748];


OptSpec = { 'call';
    'call';
    'put';
    'put';
    'call';
    'call';
    'put';
    'put';
    'call';
    'call';
    'put';
    'put';
    'call';
    'call';
    'put';
    'put'};

StockOptSpec = stockoptspec(OptPrice, Strike, Settle, Maturity, OptSpec)

StockOptSpec =

          FinObj: 'StockOptSpec'
        OptPrice: [16x1 double]
          Strike: [16x1 double]
          Settle: 732678
        Maturity: [16x1 double]
```

```
        OptSpec: {16x1 cell}
    InterpMethod: 'price'
```

See Also   ittprice | itttree | stockspec

stockspec

Purpose	Create stock structure
Syntax	StockSpec = stockspec(Sigma, AssetPrice, DividendType, DividendAmounts, ExDividendDates)

Arguments

Sigma	NINST-by-1 decimal annual price volatility of underlying security.
AssetPrice	NINST-by-1 vector of underlying asset price values at time 0.
DividendType	(Optional) NINST-by-1 cell array of strings specifying each stock's dividend type. Dividend type must be either cash for actual dollar dividends, constant for constant dividend yield, or continuous for continuous dividend yield. This function does not handle stock option dividends.

> **Note** Dividends are assumed to be paid in cash. Noncash dividends (stock) are not allowed. When combining two or more type of dividends, shorter rows should be padded with the value NaN.

DividendAmounts	(Optional) NINST-by-NDIV matrix of cash dividends or NINST-by-1 vector representing a constant or continuous annualized dividend yield.
ExDividendDates	(Optional) NINST-by-NDIV matrix of ex-dividend dates for cash type or NINST-by-1 vector of ex-dividend dates for constant dividend type. For continuous dividend type, this argument should be ignored.

Description   StockSpec = stockspec(Sigma, AssetPrice, DividendType, DividendAmounts, ExDividendDates) creates a MATLAB structure containing the properties of a stock.

Note StockSpec handles other types of underliers when pricing instruments other than equities.

Examples   **Example 1**. Consider a stock that provides four cash dividends of $0.50 on January 3, 2008, April 1, 2008, July 5, 2008 and October 1, 2008. The stock is trading at $50, and has a volatility of 20% per annum. Using this data, create the structure StockSpec:

```
AssetPrice = 50;
Sigma = 0.20;

DividendType = {'cash'};
DividendAmounts = [0.50, 0.50, 0.50, 0.50];
ExDividendDates = {'03-Jan-2008', '01-Apr-2008', '05-July-2008', '01-Oct-2008'};

StockSpec = stockspec(Sigma, AssetPrice, DividendType, DividendAmounts, ExDividendDates)

StockSpec =
```

```
            FinObj: 'StockSpec'
             Sigma: 0.2000
        AssetPrice: 50
       DividendType: {'cash'}
    DividendAmounts: [0.5000 0.5000 0.5000 0.5000]
    ExDividendDates: [733410 733499 733594 733682]
```

Examine the StockSpec structure:

```
datedisp(StockSpec.ExDividendDates)
03-Jan-2008    01-Apr-2008    05-Jul-2008    01-Oct-2008


StockSpec.DividendType

ans =

    'cash'
```

The StockSpec structure encapsulates the information of the stock and its four cash dividends.

Example 2. Consider two stocks that are trading at $40 and $35. The first one provides two cash dividends of $0.25 on March 1, 2008 and June 1, 2008. The second stock provides a continuous dividend yield of 3%. The stocks have a volatility of 30% per annum. Using this data, create the structure StockSpec:

```
AssetPrice = [40; 35];
Sigma = .30;

DividendType = {'cash'; 'continuous'};
DividendAmount = [0.25, 0.25 ; 0.03 NaN];

DividendDate1 = 'March-01-2008';
DividendDate2 = 'Jun-01-2008';

StockSpec = stockspec(Sigma, AssetPrice, DividendType, DividendAmount,...
{ DividendDate1, DividendDate2 ; NaN NaN})
```

```
StockSpec =

              FinObj: 'StockSpec'
               Sigma: [2x1 double]
          AssetPrice: [2x1 double]
        DividendType: {2x1 cell}
     DividendAmounts: [2x2 double]
     ExDividendDates: [2x2 double]
```

Examine the StockSpec structure:

```
datedisp(StockSpec.ExDividendDates)
01-Mar-2008    01-Jun-2008
  NaN            NaN


StockSpec.DividendType

ans =

    'cash'
    'continuous'
```

The StockSpec structure encapsulates the information of the two stocks and their dividends.

See Also   crrprice | crrtree | intenvset | optstockbybjs | optstockbyblk
| optstockbyls | optstockbybls | spreadbykirk | spreadbybjs |
spreadbyfd | spreadbyls | optstockbyrgw

supersharebybls

Purpose   Calculate price of supershare digital options using Black-Scholes model

Syntax   Price = supersharebybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, StrikeLow, StrikeHigh)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
StrikeLow	NINST-by-1 vector of low strike price values.
StrikeHigh	NINST-by-1 vector of high strike price values.

Description   Price = supersharebybls(RateSpec, StockSpec, Settle, Maturity, OptSpec, StrikeLow, StrikeHigh) computes supershare digital option prices using the Black-Scholes model.

Price is a NINST-by-1 vector of expected option prices.

Examples   Consider a supershare based on a portfolio of nondividend paying stocks with a lower strike of 350 and an upper strike of 450. The value of the portfolio on November 1, 2008 is 400. The risk-free rate is 4.5% and the volatility is 18%. Using this data, calculate the price of the supershare option on February 1, 2009.

Create the RateSpec:

```
Settle = 'Nov-1-2008';
```

```
Maturity = 'Feb-1-2009';
Rates = 0.045;
Basis = 1;
Compounding = -1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', Basis);
```

Define the StockSpec:

```
AssetPrice = 400;
Sigma = .18;
StockSpec = stockspec(Sigma, AssetPrice);
```

Define the high and low strike points:

```
StrikeLow = 350;
StrikeHigh = 450;
```

Calculate the price:

```
Pssh = supersharebybls(RateSpec, StockSpec, Settle, Maturity,...
StrikeLow, StrikeHigh)

Pssh =

    0.9411
```

See Also   assetbybls | cashbybls | gapbybls | supersharesensbybls

supersharesensbybls

Purpose	Calculate price and sensitivities of supershare digital options using Black-Scholes model
Syntax	PriceSens = supersharesensbybls(RateSpec, StockSpec, Settle, Maturity, StrikeLow, StrikeHigh) PriceSens = supersharesensbybls(RateSpec, StockSpec, Settle, Maturity, StrikeLow, StrikeHigh, OutSpec)

Arguments

RateSpec	The annualized, continuously compounded rate term structure. For information on the interest rate specification, see intenvset.
StockSpec	Stock specification. See stockspec.
Settle	NINST-by-1 vector of settlement or trade dates.
Maturity	NINST-by-1 vector of maturity dates.
StrikeLow	NINST-by-1 vector of low strike price values.
StrikeHigh	NINST-by-1 vector of high strike price values.
OutSpec	(Optional) All optional inputs are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. You can specify parameter name/value pairs in any order. Names are case-insensitive and partial string matches are allowed provided no ambiguities exist. Valid parameter names are: • NOUT-by-1 or 1-by-NOUT cell array of strings indicating the nature and order of the outputs for the function. Possible values are 'Price', 'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', or 'All'. For example, OutSpec = {'Price'; 'Lambda'; 'Rho'} specifies that the output

should be Price, Lambda, and Rho, in that order.

To invoke from a function: [Price, Lambda, Rho] = supersharesensbybls(..., 'OutSpec', {'Price', 'Lambda', 'Rho'})

OutSpec = {'All'} specifies that the output should be Delta, Gamma, Vega, Lambda, Rho, Theta, and Price, in that order. This is the same as specifying OutSpec as OutSpec = {'Delta', 'Gamma', 'Vega', 'Lambda', 'Rho', 'Theta', 'Price'};.

- Default is OutSpec = {'Price'}.

Description PriceSens = supersharesensbybls(RateSpec, StockSpec, Settle, Maturity, StrikeLow, StrikeHigh) computes supershare option prices using the Black-Scholes option pricing model.

PriceSens = supersharesensbybls(RateSpec, StockSpec, Settle, Maturity, StrikeLow, StrikeHigh, OutSpec) includes an OutSpec argument defined as parameter/value pairs, and computes supershare option prices and sensitivities using the Black-Scholes option pricing model.

PriceSens is a NINST-by-1 vector of expected option prices and sensitivities.

Examples Consider a supershare based on a portfolio of nondividend paying stocks with a lower strike of 350 and an upper strike of 450. The value of the portfolio on November 1, 2008 is 400. The risk-free rate is 4.5% and the volatility is 18%. Using this data, calculate the price and sensitivity of the supershare option on February 1, 2009.

Create the RateSpec:

supersharesensbybls

```
Settle = 'Nov-1-2008';
Maturity = 'Feb-1-2009';
Rates = 0.045;
Basis = 1;
Compounding = -1;
RateSpec = intenvset('ValuationDate', Settle, 'StartDates', Settle,...
'EndDates', Maturity, 'Rates', Rates, 'Compounding', Compounding, 'Basis', Basis);
```

Define the `StockSpec`:

```
AssetPrice = 400;
Sigma = .18;
StockSpec = stockspec(Sigma, AssetPrice);
```

Define the high and low strike points:

```
StrikeLow = 350;
StrikeHigh = 450;
```

Calculate the price:

```
Pssh = supersharebybls(RateSpec, StockSpec, Settle, Maturity,...
StrikeLow, StrikeHigh)

Pssh =

    0.9411
```

Compute the delta and theta of the supershare option:

```
OutSpec = { 'delta';'theta'};
[Delta, Theta]= supersharesensbybls(RateSpec, StockSpec, Settle,...
Maturity, StrikeLow, StrikeHigh, 'OutSpec', OutSpec)

Delta =

   -0.0010
```

```
Theta =

   -1.0102
```

See Also   supersharebybls

swapbybdt

Purpose   Price swap instrument from Black-Derman-Toy interest-rate tree

Syntax
```
[Price, PriceTree, CFTree, SwapRate] = swapbybdt(BDTTree,
LegRate, Settle, Maturity)
[Price, PriceTree, CFTree, SwapRate] = swapbybdt(BDTTree,
LegRate, Settle, Maturity, LegReset, Basis, Principal,
LegType, EndMonthRule)
[Price, PriceTree, CFTree, SwapRate] = swapbybdt(BDTTree,
LegRate, Settle, Maturity, Name,Value)
```

Input Arguments

BDTTree	Interest-rate tree structure created by bdttree.
LegRate	Number of instruments (NINST)-by-2 matrix, with each row defined as:
	[CouponRate Spread] or [Spread CouponRate]
	CouponRate is the decimal annual rate. Spread is the number of basis points over the reference rate. The first column represents the receiving leg, while the second column represents the paying leg.
Settle	Settlement date. NINST-by-1 vector of serial date numbers or date strings representing the settlement date for each swap. Settle must be earlier than Maturity.
Maturity	Maturity date. NINST-by-1 vector of dates representing the maturity date for each swap.

The Settle date for every swap is set to the ValuationDate of the BDT tree. The swap argument Settle is ignored.

This function also calculates the SwapRate (fixed rate) so that the value of the swap is initially 0. To do this, enter CouponRate as NaN.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

LegReset

NINST-by-2 matrix representing the reset frequency per year for each swap. NINST-by-1 vector representing the frequency of payments per year.

>**Default:** [1 1]

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

NINST-by-1 vector or NINST-by-1 cell array of the notional principal amounts or principal value schedules. For the latter case, each element of the cell array is a NumDates-by-2 call array where the first column is dates and the second column is its associated notional principal value. The date indicates the last day that the principal value is valid.

> **Default:** 100

LegType

NINST-by-2 matrix. Each row represents an instrument. Each column indicates if the corresponding leg is fixed (1) or floating (0). This matrix defines the interpretation of the values entered in LegRate.

> **Default:** [1 0] for each instrument

Options

Derivatives pricing options structure created with derivset.

EndMonthRule

End-of-month rule. NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments.
`Name` is the argument name and `Value` is the corresponding
value. `Name` must appear inside single quotes (`' '`). You can
specify several name and value pair arguments in any order as
`Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1
of logicals.

> **Default:** `false`

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with
possible choices of business day convention:

- `actual`

- `follow`

- `modifiedfollow`

- `previous`

- `modifiedprevious`

> **Default:** `actual`

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB
date numbers.

> **Default:** If no dates are specified, `holidays.m` is used.

StartDate

NINST-by-1 vector of dates when the swap actually starts. Use this argument to price forward swaps, i.e., swaps that start in a future date

> **Default:** Settle date

Description   [Price, PriceTree, CFTree, SwapRate] = swapbybdt(BDTTree, LegRate, Settle, Maturity) computes the price of a swap instrument from a BDT interest-rate tree.

[Price, PriceTree, CFTree, SwapRate] = swapbybdt(BDTTree, LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType, EndMonthRule) computes the price of a swap instrument from a BDT interest-rate tree using optional input arguments.

[Price, PriceTree, CFTree, SwapRate] = swapbybdt(BDTTree, LegRate, Settle, Maturity, Name,Value) computes the price of a swap instrument from a BDT interest-rate tree with additional options specified by one or more Name,Value pair arguments.

Price is number of instruments (NINST)-by-1 expected prices of the swap at time 0.

PriceTree is a tree structure with a vector of the swap values at each node.

CFTree is a tree structure with a vector of the swap cash flows at each node. This structure contains only NaNs because with binomial recombining trees, cash flows cannot be computed accurately at each node of a tree.

SwapRate is a NINST-by-1 vector of rates applicable to the fixed leg such that the swaps' values are zero at time 0. This rate is used in calculating the swaps' prices when the rate specified for the fixed leg in LegRate is NaN. SwapRate is padded with NaN for those instruments in which CouponRate is not set to NaN.

Definitions

Amortizing Swap

In an amortizing swap, the notional principal decreases periodically because it is tied to an underlying financial instrument with a declining (amortizing) principal balance, such as a mortgage.

Forward Swap

Agreement to enter into an interest-rate swap arrangement on a fixed date in future.

Examples

Price Interest-Rate Swap

Price an interest-rate swap with a fixed receiving leg and a floating paying leg. Payments are made once a year, and the notional principal amount is $100. The values for the remaining arguments are:

- Coupon rate for fixed leg: 0.15 (15%)

- Spread for floating leg: 10 basis points

- Swap settlement date: Jan. 01, 2000

- Swap maturity date: Jan. 01, 2003

Based on the information above, set the required arguments and build the `LegRate`, `LegType`, and `LegReset` matrices:

```
Settle = '01-Jan-2000';
Maturity = '01-Jan-2003';
Basis = 0;
Principal = 100;
LegRate = [0.15 10]; % [CouponRate Spread]
LegType = [1 0]; % [Fixed Float]
LegReset = [1 1]; % Payments once per year
```

Price the swap using the `BDTTree` included in the MAT-file `deriv.mat`. `BDTTree` contains the time and forward-rate information needed to price the instrument.

```
load deriv.mat;
```

swapbybdt

Use swapbybdt to compute the price of the swap.

```
Price  = swapbybdt(BDTTree, LegRate, Settle, Maturity,...
LegReset, Basis, Principal, LegType)

Price =

  7.4222
```

Using the previous data, calculate the swap rate, the coupon rate for the fixed leg, such that the swap price at time = 0 is zero.

```
LegRate = [NaN 20];

[Price, PriceTree, CFTree, SwapRate] = swapbybdt(BDTTree,...
LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType)

Price =

 -1.4211e-014

PriceTree =

    FinObj: 'BDTPriceTree'
      tObs: [0 1 2 3 4]
     PTree: {1x5 cell}
CFTree =

    FinObj: 'BDTCFTree'
      tObs: [0 1 2 3 4]
    CFTree: {[NaN] [NaN NaN] [NaN NaN NaN] [NaN NaN NaN NaN] ...}

SwapRate =

    0.1205
```

Price an Amortizing Swap

Price an amortizing swap using the `Principal` input argument to define the amortization schedule.

Create the `RateSpec`.

```
Rates = 0.035;
ValuationDate = '1-Jan-2011';
StartDates = ValuationDate;
EndDates = '1-Jan-2017';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: 0.8135
             Rates: 0.0350
          EndTimes: 6
        StartTimes: 0
          EndDates: 736696
        StartDates: 734504
     ValuationDate: 734504
             Basis: 0
       EndMonthRule: 1
```

Create the swap instrument using the following data:

```
Settle ='1-Jan-2011';
Maturity = '1-Jan-2017';
Period = 1;
LegRate = [0.04 10];
```

Define the swap amortizing schedule.

```
Principal ={{'1-Jan-2013' 100;'1-Jan-2014' 80;'1-Jan-2015' 60;'1-Jan-2016' 40; '1-Jan-2017' 20}};
```

Build the BDT tree and assume volatility is 10%.

```
MatDates = {'1-Jan-2012'; '1-Jan-2013';'1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017'};
BDTTimeSpec = bdttimespec(ValuationDate, MatDates);
Volatility = 0.10;
BDTVolSpec = bdtvolspec(ValuationDate, MatDates, Volatility*ones(1,length(MatDates))');
BDTT = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);
```

Compute the price of the amortizing swap.

```
Price = swapbybdt(BDTT, LegRate, Settle, Maturity, 'Principal' , Principal)

Price =

    1.4574
```

Price a Forward Swap

Price a forward swap using the StartDate input argument to define the future starting date of the swap.

Create the RateSpec.

```
Rates = 0.0325;
ValuationDate = '1-Jan-2012';
StartDates = ValuationDate;
EndDates = '1-Jan-2018';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: 0.8254
```

```
             Rates: 0.0325
          EndTimes: 6
        StartTimes: 0
          EndDates: 737061
        StartDates: 734869
     ValuationDate: 734869
             Basis: 0
       EndMonthRule: 1
```

Build the tree with a volatility of 10%.

```
MatDates = {'1-Jan-2013'; '1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017';'1-Jan-2018'};
BDTTimeSpec = bdttimespec(ValuationDate, MatDates);
Volatility = 0.10;
BDTVolSpec = bdtvolspec(ValuationDate, MatDates, Volatility*ones(1,length(MatDates))');
BDTT = bdttree(BDTVolSpec, RateSpec, BDTTimeSpec);
```

Compute the price of a forward swap that starts in two years (Jan 1, 2014) and matures in three years with a forward swap rate of 3.85%.

```
Settle ='1-Jan-2012';
Maturity = '1-Jan-2017';
StartDate = '1-Jan-2014';
LegRate = [0.0385 10];

Price = swapbybdt(BDTT, LegRate, Settle, Maturity, 'StartDate', StartDate)

Price =

    1.3203
```

Using the previous data, compute the forward swap rate, the coupon rate for the fixed leg, such that the forward swap price at time = 0 is zero.

```
LegRate = [NaN 10];
[Price, ~,~, SwapRate] = swapbybdt(BDTT, LegRate, Settle, Maturity, 'StartDate', StartDate)
```

```
Price =

   -7.1054e-14


SwapRate =

   0.0335
```

See Also   bdttree | capbybdt | cfbybdt | floorbybdt

Purpose   Price swap instrument from Black-Karasinski interest-rate tree

Syntax   [Price, PriceTree, SwapRate] = swapbybk(BKTree,
LegRate, Settle, Maturity)
[Price, PriceTree, SwapRate] = swapbybk(BKTree,
LegRate, Settle, Maturity, LegReset, Basis, Principal,
LegType, EndMonthRule)
[Price, PriceTree, SwapRate] = swapbybk(BKTree,
LegRate, Settle, Maturity, Name,Value)

**Input
Arguments**

BKTree	Interest-rate tree structure created by bktree.
LegRate	Number of instruments (NINST)-by-2 matrix, with each row defined as:
	[CouponRate Spread] or [Spread CouponRate]
	CouponRate is the decimal annual rate. Spread is the number of basis points over the reference rate. The first column represents the receiving leg, while the second column represents the paying leg.
Settle	Settlement date. NINST-by-1 vector of serial date numbers or date strings representing the settlement date for each swap. Settle must be earlier than Maturity.
Maturity	Maturity date. NINST-by-1 vector of dates representing the maturity date for each swap.

The Settle date for every swap is set to the ValuationDate of the BK tree. The swap argument Settle is ignored.

This function also calculates the SwapRate (fixed rate) so that the value of the swap is initially zero. To do this, enter CouponRate as NaN.

swapbybk

swapbybk computes prices of vanilla swaps, amortizing swaps and forward swaps.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

LegReset

NINST-by-2 matrix representing the reset frequency per year for each swap. NINST-by-1 vector representing the frequency of payments per year.

> **Default:** [1 1]

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

NINST-by-1 vector or NINST-by-1 cell array of the notional principal amounts or principal value schedules. For the latter case, each element of the cell array is a NumDates-by-2 call array where the first column is dates and the second column is its associated notional principal value. The date indicates the last day that the principal value is valid.

> **Default:** 100

LegType

NINST-by-2 matrix. Each row represents an instrument. Each column indicates if the corresponding leg is fixed (1) or floating (0). This matrix defines the interpretation of the values entered in LegRate.

> **Default:** [1 0] for each instrument

Options

Derivatives pricing options structure created with derivset.

EndMonthRule

End-of-month rule. A NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Default: 1

StartDate

NINST-by-1 vector of dates when the swap actually starts. Use this argument to price forward swaps, i.e., swaps that start in a future date

Default: Settle date

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1 of logicals.

Default: false

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with possible choices of business day convention:

- actual
- follow
- modifiedfollow
- previous
- modifiedprevious

Default: actual

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, `holidays.m` is used.

Description   `[Price, PriceTree, SwapRate] = swapbybk(BKTree, LegRate, Settle, Maturity)` computes the price of a swap instrument from a Black-Karasinski interest-rate tree.

`[Price, PriceTree, SwapRate] = swapbybk(BKTree, LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType, EndMonthRule)` computes the price of a swap instrument from a Black-Karasinski interest-rate tree using optional input arguments.

`[Price, PriceTree, SwapRate] = swapbybk(BKTree, LegRate, Settle, Maturity, Name,Value)` computes the price of a swap instrument from a Black-Karasinski interest-rate tree with additional options specified by one or more `Name,Value` pair arguments.

`Price` is the number of instruments (NINST)-by-1 expected prices of the swap at time 0.

`PriceTree` is the tree structure with a vector of the swap values at each node:

- `PriceTree.PTree` contains swap values.

- `PriceTree.tObs` contains the observation times.

- `PriceTree.Connect` contains the connectivity vectors. Each element in the cell array describes how nodes in that level connect to the next. For a given tree level, there are `NumNodes` elements in the vector, and they contain the index of the node at the next level that the middle branch connects to. Subtracting 1 from that value indicates where the up-branch connects to, and adding 1 indicates where the down branch connects to.

- `PriceTree.Probs` contains the probability arrays. Each element of the cell array contains the up, middle, and down transition probabilities for each node of the level.

`SwapRate` is a `NINST`-by-1 vector of rates applicable to the fixed leg such that the swaps' values are zero at time 0. This rate is used in calculating the swaps' prices when the rate specified for the fixed leg in `LegRate` is NaN. The `SwapRate` output is padded with `NaN` for those instruments in which `CouponRate` is not set to `NaN`.

Definitions   **Amortizing Swap**

In an amortizing swap, the notional principal decreases periodically because it is tied to an underlying financial instrument with a declining (amortizing) principal balance, such as a mortgage.

Forward Swap

Agreement to enter into an interest-rate swap arrangement on a fixed date in future.

Examples   **Price Interest-Rate Swap**

Price an interest-rate swap with a fixed receiving leg and a floating paying leg. Payments are made once a year, and the notional principal amount is $100. The values for the remaining arguments are:

- Coupon rate for fixed leg: 0.15 (15%)

- Spread for floating leg: 10 basis points

- Swap settlement date: Jan. 01, 2005

- Swap maturity date: Jan. 01, 2008

Based on the information above, set the required arguments and build the `LegRate`, `LegType`, and `LegReset` matrices:

```
Settle = '01-Jan-2005';
Maturity = '01-Jan-2008';
Basis = 0;
Principal = 100;
```

```
LegRate = [0.15 10]; % [CouponRate Spread]
LegType = [1 0]; % [Fixed Float]
LegReset = [1 1]; % Payments once per year
```

Price the swap using the BKTree included in the MAT-file deriv.mat. The BKTree structure contains the time and forward-rate information needed to price the instrument.

```
load deriv.mat;
```

Use swapbybk to compute the price of the swap.

```
Price  = swapbybk(BKTree, LegRate, Settle, Maturity, LegReset,...
Basis, Principal, LegType)

Price =

   39.1827
```

Using the previous data, calculate the swap rate, which is the coupon rate for the fixed leg, such that the swap price at time = 0 is zero.

```
LegRate = [NaN 20];

[Price, PriceTree, SwapRate] = swapbybk(BKTree, LegRate, ...
Settle, Maturity, LegReset, Basis, Principal, LegType)

Price =

     0

PriceTree =

    FinObj: 'BKPriceTree'
      PTree: {1x5 cell}
       tObs: [0 1 2 3 4]
    Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
     Probs: {[3x1 double]  [3x3 double]  [3x5 double]}
```

```
SwapRate =

    0.0438
```

Price an Amortizing Swap

Price an amortizing swap using the `Principal` input argument to define the amortization schedule.

Create the `RateSpec`.

```
Rates = 0.035;
ValuationDate = '1-Jan-2011';
StartDates = ValuationDate;
EndDates = '1-Jan-2017';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

           FinObj: 'RateSpec'
      Compounding: 1
             Disc: 0.8135
            Rates: 0.0350
         EndTimes: 6
       StartTimes: 0
         EndDates: 736696
       StartDates: 734504
    ValuationDate: 734504
            Basis: 0
      EndMonthRule: 1
```

Create the swap instrument using the following data:

```
Settle ='1-Jan-2011';
Maturity = '1-Jan-2017';
```

```
Period = 1;
LegRate = [0.04 10];
```

Define the swap amortizing schedule.

```
Principal ={{'1-Jan-2013' 100;'1-Jan-2014' 80;'1-Jan-2015' 60;'1-Jan-2016' 40; '1-Jan-2017' 20}};
```

Build the BK tree and assume volatility is 10%.

```
MatDates = {'1-Jan-2012'; '1-Jan-2013';'1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017'};
BKTimeSpec = bktimespec(ValuationDate, MatDates);
Volatility = 0.10;
AlphaDates = '01-01-2017';
AlphaCurve = 0.1;
BKVolSpec = bkvolspec(ValuationDate, MatDates, Volatility*ones(1,length(MatDates))',...
AlphaDates, AlphaCurve);
BKT = bktree(BKVolSpec, RateSpec, BKTimeSpec);
```

Compute the price of the amortizing swap.

```
Price = swapbybk(BKT, LegRate, Settle, Maturity, 'Principal' , Principal)

Price =

    1.4574
```

Price a Forward Swap

Price a forward swap using the `StartDate` input argument to define the future starting date of the swap.

Create the `RateSpec`.

```
Rates = 0.0374;
ValuationDate = '1-Jan-2012';
StartDates = ValuationDate;
EndDates = '1-Jan-2018';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
```

```
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)


RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: 0.8023
             Rates: 0.0374
          EndTimes: 6
        StartTimes: 0
          EndDates: 737061
        StartDates: 734869
     ValuationDate: 734869
             Basis: 0
       EndMonthRule: 1
```

Build a BK tree.

```
VolDates = {'1-Jan-2013'; '1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017';'1-Jan-2018'};
VolCurve = 0.1;
AlphaDates = '01-01-2018';
AlphaCurve = 0.1;


BKVolSpec = bkvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
BKTimeSpec = bktimespec(RateSpec.ValuationDate, VolDates, Compounding);
BKT = bktree(BKVolSpec, RateSpec, BKTimeSpec);
```

Compute the price of a forward swap that starts in a year (Jan 1, 2013) and matures in four years with a forward swap rate of 4.25%.

```
Settle ='1-Jan-2012';
Maturity = '1-Jan-2017';
StartDate = '1-Jan-2013';
LegRate = [0.0425 10];


Price = swapbybk(BKT, LegRate, Settle, Maturity, 'StartDate', StartDate)
```

```
Price =

    1.4434
```

Using the previous data, compute the forward swap rate, the coupon rate for the fixed leg, such that the forward swap price at time = 0 is zero.

```
LegRate = [NaN 10];
[Price, ~,SwapRate] = swapbybk(BKT, LegRate, Settle, Maturity, 'StartDate', StartDate)

Price =

    1.4211e-14


SwapRate =

    0.0384
```

See Also   bktree | bondbybk | capbybk | fixedbybk | floorbybk

swapbyhjm

Purpose	Price swap instrument from Heath-Jarrow-Morton interest-rate tree
Syntax	[Price, PriceTree, CFTree, SwapRate] = swapbyhjm(HJMTree, LegRate, Settle, Maturity)
	[Price, PriceTree, CFTree, SwapRate] = swapbyhjm(HJMTree, LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType, EndMonthRule)
	[Price, PriceTree, CFTree, SwapRate] = swapbyhjm(HJMTree, LegRate, Settle, Maturity, Name,Value)

Input Arguments

HJMTree	Forward-rate tree structure created by hjmtree.
LegRate	Number of instruments (NINST)-by-2 matrix, with each row defined as:
	[CouponRate Spread] or [Spread CouponRate]
	CouponRate is the decimal annual rate. Spread is the number of basis points over the reference rate. The first column represents the receiving leg, while the second column represents the paying leg.
Settle	Settlement date. NINST-by-1 vector of serial date numbers or date strings representing the settlement date for each swap. Settle must be earlier than Maturity.
Maturity	Maturity date. NINST-by-1 vector of dates representing the maturity date for each swap.

The Settle date for every swap is set to the ValuationDate of the HJM tree. The swap argument Settle is ignored.

This function also calculates the SwapRate (fixed rate) so that the value of the swap is initially zero. To do this, enter CouponRate as NaN.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

LegReset

NINST-by-2 matrix representing the reset frequency per year for each swap. NINST-by-1 vector representing the frequency of payments per year.

> **Default:** [1 1]

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

swapbyhjm

For more information, see **basis** on page Glossary-1.

>**Default:** 0 (actual/actual)

Principal

NINST-by-1 vector or NINST-by-1 cell array of the notional principal amounts or principal value schedules. For the latter case, each element of the cell array is a NumDates-by-2 call array where the first column is dates and the second column is its associated notional principal value. The date indicates the last day that the principal value is valid.

>**Default:** 100

LegType

NINST-by-2 matrix. Each row represents an instrument. Each column indicates if the corresponding leg is fixed (1) or floating (0). This matrix defines the interpretation of the values entered in LegRate.

>**Default:** [1 0] for each instrument

Options

Derivatives pricing options structure created with derivset.

EndMonthRule

End-of-month rule. A NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

>**Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** `false`

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- `actual`
- `follow`
- `modifiedfollow`
- `previous`
- `modifiedprevious`

> **Default:** `actual`

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

> **Default:** If no dates are specified, `holidays.m` is used.

StartDate

NINST-by-1 vector of dates when the swap actually starts. Use this argument to price forward swaps, i.e., swaps that start in a future date

Default: Settle date

Description [Price, PriceTree, CFTree, SwapRate] = swapbyhjm(HJMTree, LegRate, Settle, Maturity) computes the price of a swap instrument from an HJM interest-rate tree.

[Price, PriceTree, CFTree, SwapRate] = swapbyhjm(HJMTree, LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType, EndMonthRule) computes the price of a swap instrument from an HJM interest-rate tree with optional input arguments.

[Price, PriceTree, CFTree, SwapRate] = swapbyhjm(HJMTree, LegRate, Settle, Maturity, Name,Value) computes the price of a swap instrument from an HJM interest-rate tree with additional options specified by one or more Name,Value pair arguments.

Price is the number of instruments (NINST-by-1) expected prices of the swap at time 0.

PriceTree is the tree structure with a vector of the swap values at each node.

CFTree is the tree structure with a vector of the swap cash flows at each node.

SwapRate is a NINST-by-1 vector of rates applicable to the fixed leg such that the swaps' values are zero at time 0. This rate is used in calculating the swaps' prices when the rate specified for the fixed leg in LegRate is NaN. The SwapRate output is padded with NaN for those instruments in which CouponRate is not set to NaN.

Definitions **Amortizing Swap**

In an amortizing swap, the notional principal decreases periodically because it is tied to an underlying financial instrument with a declining (amortizing) principal balance, such as a mortgage.

Forward Swap

Agreement to enter into an interest-rate swap arrangement on a fixed date in future.

Examples   **Price an Interest-Rate Swap**

Price an interest-rate swap with a fixed receiving leg and a floating paying leg. Payments are made once a year, and the notional principal amount is $100. The values for the remaining arguments are:

• Coupon rate for fixed leg: 0.06 (6%)

• Spread for floating leg: 20 basis points

• Swap settlement date: Jan. 01, 2000

• Swap maturity date: Jan. 01, 2003

Based on the information above, set the required arguments and build the LegRate, LegType, and LegReset matrices:

```
Settle = '01-Jan-2000';
Maturity = '01-Jan-2003';
Basis = 0;
Principal = 100;
LegRate = [0.06 20]; % [CouponRate Spread]
LegType = [1 0]; % [Fixed Float]
LegReset = [1 1]; % Payments once per year
```

Price the swap using the HJMTree included in the MAT-file deriv.mat. The HJMTree structure contains the time and forward-rate information needed to price the instrument.

```
load deriv.mat;
```

Use swapbyhjm to compute the price of the swap.

```
[Price, PriceTree, CFTree] = swapbyhjm(HJMTree, LegRate,...
Settle, Maturity, LegReset, Basis, Principal, LegType)
```

```
Price =

    3.6923

PriceTree =

    FinObj: 'HJMPriceTree'
      tObs: [0 1 2 3 4]
     PBush: {1x5 cell}

CFTree =

    FinObj: 'HJMCFTree'
      tObs: [0 1 2 3 4]
    CFBush: {[0] [1x1x2 double] [1x2x2 double] ... [1x8 double]}
```

Use `treeviewer` to examine `CFTree` graphically and see the cash flows from the swap along both the up and the down branches. A positive cash flow indicates an inflow (income - payments > 0), while a negative cash flow indicates an outflow (income - payments < 0).

```
treeviewer(CFTree)
```
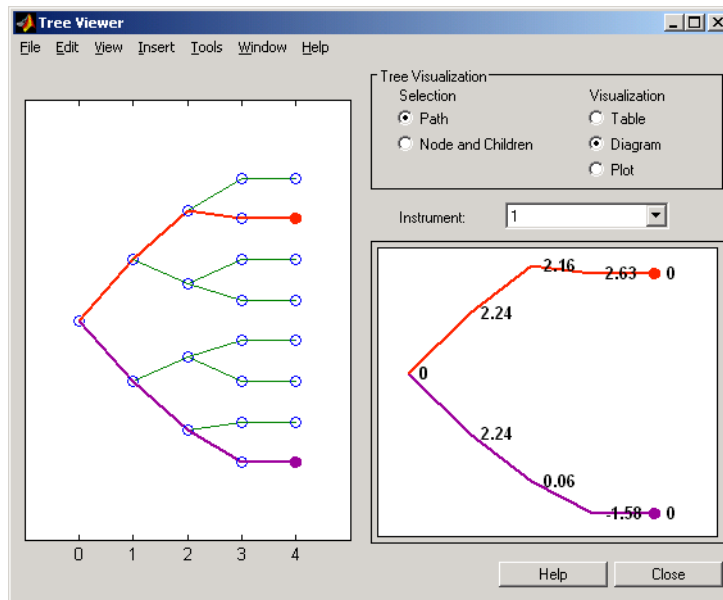
In this example, you have sold a swap (receive fixed rate and pay floating rate). At time t = 3, if interest rates go down, your cash flow is positive ($2.63), meaning that you will receive this amount. But if interest rates go up, your cash flow is negative (-$1.58), meaning that you owe this amount.

treeviewer price tree diagrams follow the convention that increasing prices appear on the upper branch of a tree and, consequently, decreasing prices appear on the lower branch. Conversely, for interest-rate displays, *decreasing* interest rates appear on the upper branch (prices are rising) and *increasing* interest rates on the lower branch (prices are falling).

Using the previous data, calculate the swap rate, which is the coupon rate for the fixed leg, such that the swap price at time = 0 is zero.

```
LegRate = [NaN 20];
```

```
[Price, PriceTree, CFTree, SwapRate] = swapbyhjm(HJMTree,...
LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType)

Price =

   0

PriceTree =

FinObj: 'HJMPriceTree'
  tObs: [0 1 2 3 4]
 PBush:{[0] [1x1x2 double] [1x2x2 double] ... [1x8 double]}

CFTree =

FinObj: 'HJMCFTree'
  tObs: [0 1 2 3 4]
CFBush:{[0] [1x1x2 double] [1x2x2 double] ... [1x8 double]}

SwapRate =

   0.0466
```

Price an Amortizing Swap

Price an amortizing swap using the `Principal` input argument to define the amortization schedule.

Create the `RateSpec`.

```
Rates = 0.035;
ValuationDate = '1-Jan-2011';
StartDates = ValuationDate;
EndDates = '1-Jan-2017';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
```

```
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

              FinObj: 'RateSpec'
         Compounding: 1
                Disc: 0.8135
               Rates: 0.0350
            EndTimes: 6
          StartTimes: 0
            EndDates: 736696
          StartDates: 734504
      ValuationDate: 734504
               Basis: 0
        EndMonthRule: 1
```

Create the swap instrument using the following data:

```
Settle ='1-Jan-2011';
Maturity = '1-Jan-2017';
Period = 1;
LegRate = [0.04 10];
```

Define the swap amortizing schedule.

```
Principal ={{'1-Jan-2013' 100;'1-Jan-2014' 80;'1-Jan-2015' 60;'1-Jan-2016' 40; '1-Jan-2017' 20}};
```

Build the HJM tree using the following data:

```
MatDates = {'1-Jan-2012'; '1-Jan-2013';'1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017'};
HJMTimeSpec = hjmtimespec(RateSpec.ValuationDate, MatDates);
Volatility = [.10; .08; .06; .04];
CurveTerm = [ 1; 2; 3; 4];
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec,RateSpec,HJMTimeSpec);
```

Compute the price of the amortizing swap.

```
Price = swapbyhjm(HJMT, LegRate, Settle, Maturity, 'Principal', Principal)
```

```
Price =

    1.4574
```

Price a Forward Swap

Price a forward swap using the `StartDate` input argument to define the future starting date of the swap.

Create the `RateSpec`.

```
Rates = 0.0374;
ValuationDate = '1-Jan-2012';
StartDates = ValuationDate;
EndDates = '1-Jan-2018';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

          FinObj: 'RateSpec'
     Compounding: 1
            Disc: 0.8023
           Rates: 0.0374
        EndTimes: 6
      StartTimes: 0
        EndDates: 737061
      StartDates: 734869
   ValuationDate: 734869
           Basis: 0
     EndMonthRule: 1
```

Build an HJM tree.

```
MatDates = {'1-Jan-2013'; '1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017';'1-Jan-2018'};
HJMTimeSpec = hjmtimespec(RateSpec.ValuationDate, MatDates);
Volatility = [.10; .08; .06; .04];
```

```
CurveTerm = [ 1; 2; 3; 4];
HJMVolSpec = hjmvolspec('Proportional', Volatility, CurveTerm, 1e6);
HJMT = hjmtree(HJMVolSpec,RateSpec,HJMTimeSpec);
```

Compute the price of a forward swap that starts in a year (Jan 1, 2013) and matures in four years with a forward swap rate of 4.25%.

```
Settle ='1-Jan-2012';
Maturity = '1-Jan-2017';
StartDate = '1-Jan-2013';
LegRate = [0.0425 10];

Price = swapbyhjm(HJMT, LegRate, Settle, Maturity, 'StartDate', StartD

Price =

    1.4434
```

Using the previous data, compute the forward swap rate, the coupon rate for the fixed leg, such that the forward swap price at time = 0 is zero.

```
LegRate = [NaN 10];
[Price, ~,~, SwapRate] = swapbyhjm(HJMT, LegRate, Settle, Maturity, 'StartDate', StartDate)

Price =

    0


SwapRate =

    0.0384
```

See Also   capbyhjm | cfbyhjm | floorbyhjm | hjmtree | treeviewer

swapbyhw

Purpose	Price swap instrument from Hull-White interest-rate tree

Syntax	`[Price, PriceTree, SwapRate] = swapbyhw(HWTree,`
	`LegRate, Settle, Maturity)`
	`[Price, PriceTree, SwapRate] = swapbyhw(HWTree,`
	`LegRate, Settle, Maturity, LegReset, Basis,`
	`Principal, LegType, Options, EndMonthRule)`
	`[Price, PriceTree, SwapRate] = swapbyhw(HWTree,`
	`LegRate, Settle, Maturity, Name,Value)`

**Input
Arguments**

HWTree	Forward-rate tree structure created by `hwtree`.
LegRate	Number of instruments (NINST)-by-2 matrix, with each row defined as: `[CouponRate Spread]` or `[Spread CouponRate]` `CouponRate` is the decimal annual rate. `Spread` is the number of basis points over the reference rate. The first column represents the receiving leg, while the second column represents the paying leg.
Settle	Settlement date. NINST-by-1 vector of serial date numbers or date strings representing the settlement date for each swap. `Settle` must be earlier than `Maturity`.
Maturity	Maturity date. NINST-by-1 vector of dates representing the maturity date for each swap.

The `Settle` date for every swap is set to the `ValuationDate` of the HW tree. The swap argument `Settle` is ignored.

This function also calculates the `SwapRate` (fixed rate) so that the value of the swap is initially zero. To do this, enter `CouponRate` as `NaN`.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

LegReset

NINST-by-2 matrix representing the reset frequency per year for each swap. NINST-by-1 vector representing the frequency of payments per year.

Default: [1 1]

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

Principal

NINST-by-1 vector or NINST-by-1 cell array of the notional principal amounts or principal value schedules. For the latter case, each element of the cell array is a NumDates-by-2 call array where the first column is dates and the second column is its associated notional principal value. The date indicates the last day that the principal value is valid.

> **Default:** 100

LegType

NINST-by-2 matrix. Each row represents an instrument. Each column indicates if the corresponding leg is fixed (1) or floating (0). This matrix defines the interpretation of the values entered in LegRate.

> **Default:** [1 0] for each instrument

Options

Derivatives pricing options structure created with derivset.

EndMonthRule

End-of-month rule. NINST-by-1 vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days.

- 0 = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- 1 = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

> **Default:** 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. NINST-by-1 of logicals.

Default: false

BusinessDayConvention

Require payment dates to be business dates. NINST-by-1 cell array with possible choices of business day convention:

- actual
- follow
- modifiedfollow
- previous
- modifiedprevious

Default: actual

Holidays

Holidays used for business day convention. NHOLIDAYS-by-1 of MATLAB date numbers.

Default: If no dates are specified, holidays.m is used.

StartDate

NINST-by-1 vector of dates when the swap actually starts. Use this argument to price forward swaps, i.e., swaps that start in a future date

> **Default:** Settle date

Description

[Price, PriceTree, SwapRate] = swapbyhw(HWTree, LegRate, Settle, Maturity) computes the price of a swap instrument from a Hull-White interest-rate tree.

[Price, PriceTree, SwapRate] = swapbyhw(HWTree, LegRate, Settle, Maturity, LegReset, Basis, Principal, LegType, Options, EndMonthRule) computes the price of a swap instrument from a Hull-White interest-rate tree with optional input arguments.

[Price, PriceTree, SwapRate] = swapbyhw(HWTree, LegRate, Settle, Maturity, Name,Value) computes the price of a swap instrument from a Hull-White interest-rate tree with additional options specified by one or more Name,Value pair arguments.

Price is number of instruments (NINST)-by-1 expected prices of the swap at time 0.

PriceTree is the tree structure with a vector of the swap values at each node.

SwapRate is a NINST-by-1 vector of rates applicable to the fixed leg such that the swaps' values are zero at time 0. This rate is used in calculating the swaps' prices when the rate specified for the fixed leg in LegRate is NaN. The SwapRate output is padded with NaNs for those instruments in which CouponRate is not set to NaN.

Definitions

Amortizing Swap

In an amortizing swap, the notional principal decreases periodically because it is tied to an underlying financial instrument with a declining (amortizing) principal balance, such as a mortgage.

Forward Swap

Agreement to enter into an interest-rate swap arrangement on a fixed date in future.

Examples

Price an Interest-Rate Swap

Price an interest-rate swap with a fixed receiving leg and a floating paying leg. Payments are made once a year, and the notional principal amount is $100. The values for the remaining arguments are:

- Coupon rate for fixed leg: 0.06 (6%)

- Spread for floating leg: 20 basis points

- Swap settlement date: Jan. 01, 2005

- Swap maturity date: Jan. 01, 2008

Based on the information above, set the required arguments and build the `LegRate`, `LegType`, and `LegReset` matrices:

```
Settle = '01-Jan-2005';
Maturity = '01-Jan-2008';
Basis = 0;
Principal = 100;
LegRate = [0.06 20]; % [CouponRate Spread]
LegType = [1 0]; % [Fixed Float]
LegReset = [1 1]; % Payments once per year
```

Price the swap using the `HWTree` included in the MAT-file `deriv.mat`. The `HWTree` structure contains the time and forward-rate information needed to price the instrument.

```
load deriv.mat;
```

Use swapbyhw to compute the price of the swap.

```
[Price, PriceTree, SwapRate] = swapbyhw(HWTree, LegRate, ...
Settle, Maturity, LegReset, Basis, Principal, LegType)
```

```
Price =

    5.9109

PriceTree =

     FinObj: 'HWPriceTree'
      PTree: {1x5 cell}
       tObs: [0 1 2 3 4]
    Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
      Probs: {[3x1 double]  [3x3 double]  [3x5 double]}

SwapRate =

   NaN
```

Using the previous data, calculate the swap rate, which is the coupon rate for the fixed leg, such that the swap price at time = 0 is zero.

```
LegRate = [NaN 20];

[Price, PriceTree, SwapRate] = swapbyhw(HWTree, LegRate, ...
Settle, Maturity, LegReset, Basis, Principal, LegType)

Price =

   1.4211e-014

PriceTree =

FinObj: 'HWPriceTree'
      PTree: {1x5 cell}
       tObs: [0 1 2 3 4]
    Connect: {[2]  [2 3 4]  [2 2 3 4 4]}
      Probs: {[3x1 double]  [3x3 double]  [3x5 double]}

SwapRate =
```

```
    0.0438
```

Price an Amortizing Swap

Price an amortizing swap using the `Principal` input argument to define the amortization schedule.

Create the `RateSpec`.

```
Rates = 0.035;
ValuationDate = '1-Jan-2011';
StartDates = ValuationDate;
EndDates = '1-Jan-2017';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: 0.8135
             Rates: 0.0350
          EndTimes: 6
        StartTimes: 0
          EndDates: 736696
        StartDates: 734504
     ValuationDate: 734504
             Basis: 0
       EndMonthRule: 1
```

Create the swap instrument using the following data:

```
Settle ='1-Jan-2011';
Maturity = '1-Jan-2017';
Period = 1;
```

```
LegRate = [0.04 10];
```

Define the swap amortizing schedule.

```
Principal ={{'1-Jan-2013' 100;'1-Jan-2014' 80;'1-Jan-2015' 60;'1-Jan-2016' 40; '1-Jan-2017' 20}};
```

Build the HW tree using the following data:

```
VolDates = ['1-Jan-2012'; '1-Jan-2013';'1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017'];
VolCurve = 0.1;
AlphaDates = '01-01-2017';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec);
```

Compute the price of the amortizing swap.

```
Price = swapbyhw(HWT, LegRate, Settle, Maturity, 'Principal', Principal)

Price =

    1.4574
```

Price a Forward Swap

Price a forward swap using the `StartDate` input argument to define the future starting date of the swap.

Create the `RateSpec`.

```
Rates = 0.0374;
ValuationDate = '1-Jan-2012';
StartDates = ValuationDate;
EndDates = '1-Jan-2018';
Compounding = 1;
```

```
RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

              FinObj: 'RateSpec'
         Compounding: 1
                Disc: 0.8023
               Rates: 0.0374
            EndTimes: 6
          StartTimes: 0
            EndDates: 737061
          StartDates: 734869
       ValuationDate: 734869
               Basis: 0
         EndMonthRule: 1
```

Build an HW tree.

```
VolDates = {'1-Jan-2013'; '1-Jan-2014';'1-Jan-2015';'1-Jan-2016';'1-Jan-2017';'1-Jan-2018'};
VolCurve = 0.1;
AlphaDates = '01-01-2018';
AlphaCurve = 0.1;

HWVolSpec = hwvolspec(RateSpec.ValuationDate, VolDates, VolCurve,...
AlphaDates, AlphaCurve);
HWTimeSpec = hwtimespec(RateSpec.ValuationDate, VolDates, Compounding);
HWT = hwtree(HWVolSpec, RateSpec, HWTimeSpec);
```

Compute the price of a forward swap that starts in a year (Jan 1, 2013)
and matures in four years with a forward swap rate of 4.25%.

```
Settle ='1-Jan-2012';
Maturity = '1-Jan-2017';
StartDate = '1-Jan-2013';
LegRate = [0.0425 10];

Price = swapbyhw(HWT, LegRate, Settle, Maturity, 'StartDate', StartDate)
```

```
Price =

    1.4434
```

Using the previous data, compute the forward swap rate, the coupon rate for the fixed leg, such that the forward swap price at time = 0 is zero.

```
LegRate = [NaN 10];
[Price, ~,SwapRate] = swapbyhw(HWT, LegRate, Settle, Maturity, 'StartDate', StartDate)

Price =

    1.4211e-14


SwapRate =

    0.0384
```

See Also   bondbyhw | capbyhw | cfbyhw | floorbyhw | fixedbyhw | hwtree

Purpose   Price swap instrument from set of zero curves

Syntax
```
[Price, SwapRate AI, RecCF, RecCFDates, PayCF, PayCFDates] =
swapbyzero(RateSpec, LegRate, Settle, Maturity)
[Price, SwapRate AI, RecCF, RecCFDates, PayCF, PayCFDates] =
swapbyzero(RateSpec, LegRate, Settle, Maturity,
LegReset, Basis,  Principal, LegType, EndMonthRule)
[Price, SwapRate, AI, RecCF, RecCFDates, PayCF, PayCFDates] =
swapbyzero(RateSpec, LegRate, Settle, Maturity,
Name, Value)
```

Description   `[Price, SwapRate AI, RecCF, RecCFDates, PayCF, PayCFDates]`
`=`
`swapbyzero(RateSpec, LegRate, Settle, Maturity)` prices a swap
instrument from a set of zero coupon bond rates. All inputs are either
scalars or `NINST`-by-1 vectors unless otherwise specified. Any date can
be a serial date number or date string. An optional argument can be
passed as an empty matrix `[]`.

`[Price, SwapRate AI, RecCF, RecCFDates, PayCF, PayCFDates]`
`=`
`swapbyzero(RateSpec, LegRate, Settle, Maturity,`
`LegReset, Basis, Principal, LegType, EndMonthRule)` prices a
swap instrument from a set of zero coupon bond rates with optional
input arguments. All inputs are either scalars or `NINST`-by-1 vectors
unless otherwise specified. Any date can be a serial date number or date
string. An optional argument can be passed as an empty matrix `[]`.

`[Price, SwapRate, AI, RecCF, RecCFDates, PayCF, PayCFDates]`
`=`
`swapbyzero(RateSpec, LegRate, Settle, Maturity,`
`Name, Value)` prices a swap instrument from a set of zero coupon bond
rates with additional options specified by one or more `Name, Value`
pair arguments.

swapbyzero

Input Arguments

RateSpec

Structure containing the properties of an interest-rate structure. See `intenvset` for information on creating `RateSpec`.

`RateSpec` can also be a 1-by-2 input variable of `RateSpecs`, with the second `RateSpec` structure containing the discount curve(s) for the paying leg. If only one `RateSpec` structure is specified, then this `RateSpec` is used to discount both legs.

LegRate

Number of instruments (`NINST`)-by-2 matrix, with each row defined as:

`[CouponRate Spread]` or `[Spread CouponRate]`

`CouponRate` is the decimal annual rate. `Spread` is the number of basis points over the reference rate. The first column represents the receiving leg, while the second column represents the paying leg.

Settle

Settlement date. `Settle` must be either a scalar or `NINST`-by-1 vector of serial date numbers or date strings of the same value which represent the settlement date for each swap. `Settle` must be earlier than `Maturity`.

Maturity

Maturity date. `NINST`-by-1 vector of dates representing the maturity date for each swap.

Ordered Input or Name-Value Pair Arguments

Enter the following optional inputs using an ordered syntax or as name-value pair arguments. You cannot mix ordered syntax with name-value pair arguments.

LegReset

NINST-by-2 matrix representing the reset frequency per year for each swap. NINST-by-1 vector representing the frequency of payments per year.

Default: [1 1]

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Default: 0 (actual/actual)

Principal

NINST-by-1 vector or NINST-by-1 cell array of the notional principal amounts or principal value schedules. For the latter case, each element

of the cell array is a `NumDates`-by-2 call array where the first column is dates and the second column is its associated notional principal value. The date indicates the last day that the principal value is valid.

Default: 100

LegType

`NINST`-by-2 matrix. Each row represents an instrument. Each column indicates if the corresponding leg is fixed (1) or floating (0). This matrix defines the interpretation of the values entered in `LegRate`.

Default: [1 0] for each instrument

Options

Derivatives pricing options structure created with `derivset`.

EndMonthRule

End-of-month rule. `NINST`-by-1 vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days.

- `0` = Ignore rule, meaning that a bond coupon payment date is always the same numerical day of the month.

- `1` = Set rule on, meaning that a bond coupon payment date is always the last actual day of the month.

Default: 1

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

AdjustCashFlowsBasis

Adjust the cash flows based on the actual period day count. `NINST`-by-1 of logicals.

> **Default:** `false`

BusinessDayConvention

Require payment dates to be business dates. `NINST`-by-1 cell array with possible choices of business day convention:

- `actual`

- `follow`

- `modifiedfollow`

- `previous`

- `modifiedprevious`

> **Default:** `actual`

ForwardRateSpec

Forward rate spec to use in generating cash flows. Note that spot rates should be used when specifying `ForwardRateSpec`.

> **Default:** If not specified, then the `RateSpec` is used both for discounting cash flows and generating floating cash flows.

Holidays

Holidays used for business day convention. `NHOLIDAYS`-by-1 of MATLAB date numbers.

> **Default:** If none specified, `holidays.m` is used.

LatestFloatingRate

swapbyzero

Rate for the next floating payment, set at the last reset date. `NINST`-by-1 of scalars.

> **Default:** If not specified, then the `RateSpec` must contain this information.

StartDate

`NINST`-by-1 vector of dates when the swap actually starts. Use this argument to price forward swaps, i.e., swaps that start in a future date

> **Default:** `Settle` date

Output Arguments

Price

Number of instruments (`NINST`) by number of curves (`NUMCURVES`) matrix of swap prices. Each column arises from one of the zero curves.

SwapRate

`NINST`-by-`NUMCURVES` matrix of rates applicable to the fixed leg such that the swap's values are zero at time 0. This rate is used in calculating the swaps' prices when the rate specified for the fixed leg in `LegRate` is `NaN`. The `SwapRate` output is padded with `NaN` for those instruments in which `CouponRate` is not set to `NaN`.

Output cash flows, cash flow dates, and accrued interest.

AI

`NINST`-by-`NUMCURVES` matrix of accrued interest.

RecCF

`NINST`-by-`NUMCURVES` matrix of cash flows for the receiving leg.

> **Note** If there is more than one curve specified in the `RateSpec` input,
> then the first NCURVES row corresponds to the first swap, the second
> NCURVES row correspond to the second swap, and so on.

RecCFDates

NINST-by-NUMCURVES matrix of payment dates for the receiving leg.

PayCF

NINST-by-NUMCURVES matrix of cash flows for the paying leg.

PayCFDates

NINST-by-NUMCURVES matrix of payment dates for the paying leg.

Definitions **Amortizing Swap**

In an amortizing swap, the notional principal decreases periodically
because it is tied to an underlying financial instrument with a declining
(amortizing) principal balance, such as a mortgage.

Forward Swap

Agreement to enter into an interest-rate swap arrangement on a fixed
date in future.

Examples **Price an Interest-Rate Swap**

Price an interest-rate swap with a fixed receiving leg and a floating
paying leg. Payments are made once a year, and the notional principal
amount is $100. The values for the remaining arguments are:

- Coupon rate for fixed leg: 0.06 (6%)

- Spread for floating leg: 20 basis points

- Swap settlement date: Jan. 01, 2000

- Swap maturity date: Jan. 01, 2003

Based on the information above, set the required arguments and build the `LegRate`, `LegType`, and `LegReset` matrices:

```
Settle = '01-Jan-2000';
Maturity = '01-Jan-2003';
Basis = 0;
Principal = 100;
LegRate = [0.06 20]; % [CouponRate Spread]
LegType = [1 0]; % [Fixed Float]
LegReset = [1 1]; % Payments once per year
```

Load the file `deriv.mat`, which provides `ZeroRateSpec`, the interest-rate term structure needed to price the bond.

```
load deriv.mat;
```

Use `swapbyzero` to compute the price of the swap.

```
Price = swapbyzero(ZeroRateSpec, LegRate, Settle, Maturity,...
LegReset, Basis, Principal, LegType)

Price =
   3.6923
```

Using the previous data, calculate the swap rate, which is the coupon rate for the fixed leg, such that the swap price at time = 0 is zero.

```
LegRate = [NaN 20];

[Price, SwapRate] = swapbyzero(ZeroRateSpec, LegRate, Settle,...
Maturity, LegReset, Basis, Principal, LegType)

Price =

 -1.4211e-014
```

```
SwapRate =
   0.0466
```

Use swapbyzero with name-value pair arguments for LegRate, LegType, LatestFloatingRate, AdjustCashFlowsBasis, and BusinessDayConvention to calculate output for Price, SwapRate, AI, RecCF, RecCFDates, PayCF, and PayCFDates:

```
Settle = datenum('08-Jun-2010');
RateSpec = intenvset('Rates', [.005 .0075 .01 .014 .02 .025 .03]',...
'StartDates',Settle, 'EndDates',{'08-Dec-2010','08-Jun-2011',...
'08-Jun-2012','08-Jun-2013','08-Jun-2015','08-Jun-2017','08-Jun-2020'}');
Maturity = datenum('15-Sep-2020');
LegRate = [.025 50];
LegType = [1 0]; % fixed/floating
LatestFloatingRate = .005;

[Price, SwapRate, AI, RecCF, RecCFDates, PayCF,PayCFDates] = ...
swapbyzero(RateSpec, LegRate, Settle, Maturity,'LegType',LegType,...
'LatestFloatingRate',LatestFloatingRate,'AdjustCashFlowsBasis',true,...
'BusinessDayConvention','modifiedfollow')

Price =

   -3.3937

SwapRate =

   NaN

AI =

    1.4575


RecCF =
```

```
Columns 1 through 10

-1.8219    1.2603    1.2603    1.2740    1.2671    1.2466    1.2534    1.2603    1.2603  1.2740

Columns 11 through 12

1.2671  101.2534


RecCFDates =

Columns 1 through 8

734297      734396      734761      735129      735493      735857      736222    736588

Columns 9 through 12

736953      737320      737684      738049


PayCF =

Columns 1 through 10

-0.3644    0.2521    0.7082    1.0116    1.4423    1.6380    1.9161    2.1038  2.2768  2.2766

Columns 11 through 12

2.4370  102.3432


PayCFDates =

Columns 1 through 8

734297      734396      734761      735129      735493      735857      736222      736588
```

```
Columns 9 through 12

736953      737320      737684      738049
```

Price Swaps By Specifying Multiple Term Structures Using RateSpec

Price three swaps using two interest-rate curves.

Define data for the interest-rate term structure.

```
StartDates = '01-May-2012';
EndDates = {'01-May-2013'; '01-May-2014';'01-May-2015';'01-May-2016'};
Rates = [[0.0356;0.041185;0.04489;0.047741],[0.0366;0.04218;0.04589;0.04974]];
```

Create the `RateSpec` using `intenvset`.

```
RateSpec = intenvset('Rates', Rates, 'StartDates',StartDates,...
'EndDates', EndDates, 'Compounding', 1)


RateSpec =

           FinObj: 'RateSpec'
      Compounding: 1
             Disc: [4x2 double]
            Rates: [4x2 double]
         EndTimes: [4x1 double]
       StartTimes: [4x1 double]
         EndDates: [4x1 double]
       StartDates: 734990
    ValuationDate: 734990
            Basis: 0
      EndMonthRule: 1
```

Look at the `Rates` for the two interest-rate curves.

```
RateSpec.Rates

ans =
```

```
        0.0356    0.0366
        0.0412    0.0422
        0.0449    0.0459
        0.0477    0.0497
```

Define the swap instruments.

```
Settle = '01-May-2012';
Maturity = '01-May-2015';
LegRate = [0.06 10];
Principal = [100;50;100];   % Three notional amounts
```

Price three swaps using two curves.

```
Price = swapbyzero(RateSpec, LegRate, Settle, Maturity, 'Principal', Principal)

Price =

    3.9688    3.6869
    1.9844    1.8434
    3.9688    3.6869
```

Price Swap By Specifying Multiple Term Structures Using a 1-by-2 RateSpec

Price a swap using two interest-rate curves.

Define data for the two interest-rate term structures.

```
StartDates = '01-May-2012';
EndDates = {'01-May-2013'; '01-May-2014';'01-May-2015';'01-May-2016'};
Rates1 = [0.0356;0.041185;0.04489;0.047741];
Rates2 = [0.0366;0.04218;0.04589;0.04974];
```

Create the RateSpec using intenvset.

```
RateSpecReceiving = intenvset('Rates', Rates1, 'StartDates',StartDates,...
'EndDates', EndDates, 'Compounding', 1);
```

```
RateSpecPaying= intenvset('Rates', Rates2, 'StartDates',StartDates,...
'EndDates', EndDates, 'Compounding', 1);
RateSpec=[RateSpecReceiving RateSpecPaying]


RateSpec =

1x2 struct array with fields:
    FinObj
    Compounding
    Disc
    Rates
    EndTimes
    StartTimes
    EndDates
    StartDates
    ValuationDate
    Basis
    EndMonthRule
```

Define the swap instruments.

```
Settle = '01-May-2012';
Maturity = '01-May-2015';
LegRate = [0.06 10];
Principal = [100;50;100];
```

Price three swaps using the two curves.

```
Price = swapbyzero(RateSpec, LegRate, Settle, Maturity, 'Principal', Principal)


Price =

    3.9693
    1.9846
    3.9693
```

swapbyzero

Price an Amortizing Swap

Price an amortizing swap using the `Principal` input argument to define the amortization schedule.

Create the `RateSpec`.

```
Rates = 0.035;
ValuationDate = '1-Jan-2011';
StartDates = ValuationDate;
EndDates = '1-Jan-2017';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding);

RateSpec =

           FinObj: 'RateSpec'
      Compounding: 1
             Disc: 0.8135
            Rates: 0.0350
         EndTimes: 6
       StartTimes: 0
         EndDates: 736696
       StartDates: 734504
    ValuationDate: 734504
            Basis: 0
      EndMonthRule: 1
```

Create the swap instrument using the following data:

```
Settle ='1-Jan-2011';
Maturity = '1-Jan-2017';
Period = 1;
LegRate = [0.04 10];
```

Define the swap amortizing schedule.

```
Principal ={{'1-Jan-2013' 100;'1-Jan-2014' 80;'1-Jan-2015' 60;'1-Jan-2016' 40; '1-Jan-2017' 20}};
```

Compute the price of the amortizing swap.

```
Price = swapbyzero(RateSpec, LegRate, Settle, Maturity, 'Principal' , Principal)

Price =

    1.4574
```

Price a Forward Swap

Price a forward swap using the StartDate input argument to define the future starting date of the swap.

Create the RateSpec.

```
Rates = 0.0325;
ValuationDate = '1-Jan-2012';
StartDates = ValuationDate;
EndDates = '1-Jan-2018';
Compounding = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', StartDates,...
'EndDates', EndDates,'Rates', Rates, 'Compounding', Compounding)

RateSpec =

            FinObj: 'RateSpec'
       Compounding: 1
              Disc: 0.8254
             Rates: 0.0325
          EndTimes: 6
        StartTimes: 0
          EndDates: 737061
        StartDates: 734869
     ValuationDate: 734869
             Basis: 0
       EndMonthRule: 1
```

swapbyzero

Compute the price of a forward swap that starts in a year (Jan 1, 2013) and matures in three years with a forward swap rate of 4.27%.

```
Settle ='1-Jan-2012';
StartDate = '1-Jan-2013';
Maturity = '1-Jan-2016';
LegRate = [0.0427 10];

Price = swapbyzero(RateSpec, LegRate, Settle, Maturity, 'StartDate' , StartDate)

Price =

    2.5083
```

Using the previous data, compute the forward swap rate, the coupon rate for the fixed leg, such that the forward swap price at time = 0 is zero.

```
LegRate = [NaN 10];
[Price, SwapRate] = swapbyzero(RateSpec, LegRate, Settle, Maturity,...
'StartDate' , StartDate)

Price =

     0


SwapRate =

    0.0335
```

Specify the Rate at the Instrument's Starting Date When it Cannot be Obtained From the `RateSpec`

If `Settle` is not on a reset date of a floating-rate note, `swapbyzero` attempts to obtain the latest floating rate before `Settle` from `RateSpec` or the `LatestFloatingRate` parameter. When the reset date for this rate is out of the range of `RateSpec` (and `LatestFloatingRate` is not specified), `swapbyzero` fails to obtain the rate for that

date and generates an error. This example shows how to use the
`LatestFloatingRate` input parameter to avoid the error.

Create the error condition when a swap instrument's `StartDate` cannot
be determined from the `RateSpec`.

```
Settle = '01-Jan-2000';
Maturity = '01-Dec-2003';
Basis = 0;
Principal = 100;
LegRate = [0.06 20]; % [CouponRate Spread]
LegType = [1 0]; % [Fixed Float]
LegReset = [1 1]; % Payments once per year

load deriv.mat;

Price = swapbyzero(ZeroRateSpec, LegRate, Settle, Maturity,...
'LegReset', LegReset, 'Basis', Basis, 'Principal', Principal, ...
'LegType', LegType)

Error using floatbyzero (line 256)
The rate at the instrument starting date cannot be obtained from RateSpec.
 Its reset date (01-Dec-1999) is out of the range of dates contained in RateSpec.
 This rate is required to calculate cash flows at the instrument starting date.
 Consider specifying this rate with the 'LatestFloatingRate' input parameter.

Error in swapbyzero (line 289)
[FloatFullPrice, FloatPrice,FloatCF,FloatCFDates] = floatbyzero(FloatRateSpec, Spreads, Settle,...
```

Here, the reset date for the rate at `Settle` was 01-Dec-1999, which was
earlier than the valuation date of `ZeroRateSpec` (01-Jan-2000). This
error can be avoided by specifying the rate at the swap instrument's
starting date using the `LatestFloatingRate` input parameter.

Define `LatestFloatingRate` and calculate the floating-rate price.

```
Price = swapbyzero(ZeroRateSpec, LegRate, Settle, Maturity,...
'LegReset', LegReset, 'Basis', Basis, 'Principal', Principal, ...
```

```
'LegType', LegType, 'LatestFloatingRate', 0.03)

Price =

    4.7594
```

See Also | bondbyzero | intenvset | cfbyzero | fixedbyzero | floatbyzero

Purpose   Price swaption from Black-Derman-Toy interest-rate tree

Syntax   [Price, PriceTree] = swaptionbybdt(BDTTree, OptSpec, Strike,
                ExerciseDates, Spread, Settle, Maturity,
                'Name1', Value1, 'Name2', Value2)

Arguments

BDTTree	Interest-rate tree structure created by bdttree.	
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'. A call swaption entitles the buyer to pay the fixed rate. A put swaption entitles the buyer to receive the fixed rate.	
Strike	NINST-by-1 vector for strike swap rate values.	
ExerciseDates	For a European option: NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one ExerciseDate on the option expiry date.	
	For an American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying swap Settle and the single listed ExerciseDate.	
Spread	NINST-by-1 vector representing the number of basis points over the reference rate.	
Settle	NINST-by-1 vector of dates representing the settle date for each swap.	

swaptionbybdt

Maturity	NINST-by-1 vector of dates representing the maturity date for each swap.

Note All optional inputs that follow are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may be specified in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

AmericanOpt	(Optional) NINST-by-1 flags options:
	• 0 for European options
	• 1 for American options
SwapReset	(Optional) NINST-by-1 vector representing the reset frequency per year for the underlying swap. Default is 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page
  Glossary-1.

Principal	(Optional) NINST-by-1 vector of the notional principal amounts. Default is 100.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description [Price, PriceTree] = swaptionbybdt(BDTTree, OptSpec,
Strike,
ExerciseDates, Spread, Settle, Maturity,
'Name1', Value1, 'Name2', Value2) computes the price of a
swaption from a BDT interest-rate tree. The swaption may be a call
swaption or a put swaption.

Note The Settle date for every swaption is set to the ValuationDate
of the BDT tree. The swap argument Settle is ignored.

A call swaption or payer swaption allows the option buyer to enter into
an interest rate swap in which the buyer of the option pays the fixed
rate and receives the floating rate.

A put swaption or receiver swaption allows the option buyer to enter
into an interest rate swap in which the buyer of the option receives the
fixed rate and pays the floating rate.

Price is a NINST-by-1 vector of expected swaption prices at time 0.

PriceTree is a MATLAB structure of trees containing vectors of swaption instrument prices and a vector of observation times for each node. Within PriceTree:

- PriceTree.PTree contains the clean prices.

- PriceTree.tObs contains the observation times.

Examples   Price a 5-year call swaption using a BDT interest-rate tree.

Assume that interest rate and volatility are fixed at 6% and 20% annually between the valuation date of the tree until its maturity. Build a tree with the following data:

```
Rates = 0.06 * ones (10,1);
StartDates = ['jan-1-2007';'jan-1-2008';'jan-1-2009';'jan-1-2010';'jan-1-2011';...
'jan-1-2012';'jan-1-2013';'jan-1-2014';'jan-1-2015';'jan-1-2016'];

EndDates =['jan-1-2008';'jan-1-2009';'jan-1-2010';'jan-1-2011';'jan-1-2012';...
'jan-1-2013';'jan-1-2014';'jan-1-2015';'jan-1-2016';'jan-1-2017'];
ValuationDate = 'jan-1-2007';
Compounding = 1;
```

Determine the RateSpec:

```
RateSpec = intenvset('Rates', Rates, 'StartDates', StartDates, 'EndDates', EndDates, ...
'Compounding', Compounding);
```

Use VolSpec to compute the interest rate volatility:

```
Volatility = 0.20 * ones (10,1);  VolSpec = bdtvolspec(ValuationDate,...
EndDates, Volatility);
```

Use TimeSpec to specify the structure of the time layout for an equal probabilities tree:

```
TimeSpec = bdttimespec(ValuationDate, EndDates, Compounding);
```

Build the BDT tree:

```
BDTTree = bdttree(VolSpec, RateSpec, TimeSpec);
```

Use the following swaption arguments:

```
SwapSettlement = 'jan-1-2007';
SwapMaturity   = 'jan-1-2015';
Spread = 0;
SwapReset = 1;
Principal = 100;
OptSpec = 'call';
Strike=.062;
ExerciseDates = 'jan-1-2012';
Basis=1;
```

Price the swaption

```
[Price, PriceTree] = swaptionbybdt(BDTTree, OptSpec, Strike, ExerciseDates, ...
Spread, SwapSettlement, SwapMaturity, 'SwapReset', SwapReset, ...
'Basis', Basis, 'Principal', Principal)
```

to return

```
Price =        2.0592

PriceTree =
FinObj: 'BDTPriceTree'
tObs: [0 1 2 3 4 5 6 7 8 9 10]
PTree: {1x11 cell}
```

See Also   bdttree | instswaption | swapbybdt

swaptionbybk

Purpose   Price swaption from Black-Karasinski interest-rate tree

Syntax   [Price, PriceTree] = swaptionbybk(BKTTree, OptSpec, Strike,
                 ExerciseDates, Spread, Settle, Maturity,
                 'Name1', Value1, 'Name2', Value2)

Arguments

BKTree	Interest-rate tree structure created by bktree.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'. A call swaption entitles the buyer to pay the fixed rate. A put swaption entitles the buyer to receive the fixed rate.
Strike	NINST-by-1 vector for strike swap rate values.
ExerciseDates	For a European option: NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one ExerciseDate on the option expiry date.
	For an American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying swap Settle and the single listed ExerciseDate.
Spread	NINST-by-1 vector representing the number of basis points over the reference rate.
Settle	NINST-by-1 vector of dates representing the settle date for each swap.

Maturity	NINST-by-1 vector of dates representing the maturity date for each swap.

Note All optional inputs that follow are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may be specified in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

AmericanOpt	(Optional) NINST-by-1 flags options:
	• 0 for European options
	• 1 for American options
SwapReset	(Optional) NINST-by-1 vector representing the reset frequency per year for the underlying swap. Default is 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal	(Optional) NINST-by-1 vector of the notional principal amounts. Default is 100.
Options	(Optional) Derivatives pricing options structure created with `derivset`.

Description [Price, PriceTree] = swaptionbybk(BKTTree, OptSpec, Strike, ExerciseDates, Spread, Settle, Maturity, 'Name1', Value1, 'Name2', Value2) computes the price of a swaption from a BK interest-rate tree.

> **Note** The Settle date for every swaption is set to the ValuationDate of the BK tree. The swap argument Settle is ignored.

The swaption may be a call swaption or a put swaption.

A call swaption or payer swaption allows the option buyer to enter into an interest rate swap in which the buyer of the option pays the fixed rate and receives the floating rate.

A put swaption or receiver swaption allows the option buyer to enter into an interest rate swap in which the buyer of the option receives the fixed rate and pays the floating rate.

Price is a NINST-by-1 vector of expected swaption prices at time 0.

PriceTree is a MATLAB structure of trees containing vectors of swaption instrument prices and a vector of observation times for each node. Within PriceTree:

- PriceTree.PTree contains the clean prices.

- PriceTree.tObs contains the observation times.

Examples

Price a 4-year call and put swaption using a BK interest-rate tree with the following data.

Specify the RateSpec, assuming the interest rate is fixed at 7% annually:

```
 Rates =0.07 * ones (10,1);
Compounding = 2;
StartDates = ['jan-1-2007';'jul-1-2007';'jan-1-2008';'jul-1-2008';'jan-1-2009'; ...
'jul-1-2009'; 'jan-1-2010'; 'jul-1-2010';'jan-1-2011';'jul-1-2011'];
EndDates =['jul-1-2007';'jan-1-2008';'jul-1-2008';'jan-1-2009';'jul-1-2009'; ...
'jan-1-2010'; 'jul-1-2010';'jan-1-2011';'jul-1-2011';'jan-1-2012'];
ValuationDate = 'jan-1-2007';
RateSpec = intenvset('Rates', Rates, 'StartDates', StartDates, 'EndDates', EndDates,...
'Compounding', Compounding);
```

Use BKVolSpec to compute the interest rate volatility:

```
Volatility = 0.10*ones(10,1);
AlphaCurve = 0.05*ones(10,1);
AlphaDates = EndDates;
BKVolSpec = bkvolspec(ValuationDate, EndDates, Volatility, AlphaDates, AlphaCurve);
```

Use BKTimeSpec to specify the structure of the time layout for the BK interest-rate tree.

```
BKTimeSpec = bktimespec(ValuationDate, EndDates, Compounding);
```

Build the BK tree:

```
BKTree = bktree(BKVolSpec, RateSpec, BKTimeSpec);
```

swaptionbybk

Use the following arguments for a 5-year swap and 4-year swaption:

```
SwapSettlement = 'jan-1-2007';
SwapMaturity   = 'jan-1-2012';
Spread = 0;
SwapReset = 2 ;
Principal = 100;
OptSpec = {'call' ;'put'};
Strike= [ 0.07 ; 0.0725];
ExerciseDates = 'jan-1-2011';
Basis=1;
```

Price the swaption

```
PriceSwaption = swaptionbybk(BKTree, OptSpec, Strike, ExerciseDates, ...
Spread, SwapSettlement, SwapMaturity, 'SwapReset', SwapReset, 'Basis', Basis, ...
'Principal', Principal)
```

to return

```
PriceSwaption =
0.3593
0.4756
```

See Also   bktree | instswaption | swapbybk

swaptionbyblk

Purpose   Price European swaption instrument using Black model

Syntax   Price = swaptionbyblk(RateSpec, OptSpec, Strike,
                  Settle, ExerciseDates, Maturity, Volatility)
                  Price = swaptionbyblk(RateSpec, OptSpec, Strike, Settle,
                  ExerciseDates, Maturity, Volatility, Name, Value)

Description   Price = swaptionbyblk(RateSpec, OptSpec, Strike,
                  Settle, ExerciseDates, Maturity, Volatility) prices swaptions
                  using the Black option pricing model.

                  Price = swaptionbyblk(RateSpec, OptSpec, Strike, Settle,
                  ExerciseDates, Maturity, Volatility, Name, Value) prices
                  swaptions using the Black option pricing model with additional options
                  specified by one or more Name,Value pair arguments.

                  Note When pricing swaptions using the Black model, the underlying is
                  not a regular swap, but a forward on a swap.

Input   RateSpec
Arguments
                  Structure containing the properties of an interest-rate structure. See
                  intenvset for information on creating RateSpec.

                  RateSpec can be a NINST-by-2 input variable of RateSpecs, with the
                  second input being the discount curve for the paying leg if different
                  than the receiving leg. If only one curve is specified, than it is used to
                  discount both legs.

                  OptSpec

                  NINST-by-1 cell array of strings 'call' or 'put'. A call swaption
                  entitles the buyer to pay the fixed rate. A put swaption entitles the
                  buyer to receive the fixed rate.

                  Strike

NINST-by-1 vector of strike swap rate values.

Settle

Settlement date. `NINST-by-1` vector of serial date numbers or date strings representing the settlement date for each swap. `Settle` must be earlier than `Maturity`.

ExerciseDate

For a European option: `NINST-by-1` vector of exercise dates. Each row is the schedule for one option. When using an European option, there is only one `ExerciseDate` on the option expiry date.

Maturity

Maturity date. `NINST-by-1` vector of dates representing the maturity date for each swap.

Volatility

Volatilities values. `NINST-by-1` vector of volatilities.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Basis

Day-count basis of the instrument. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365

- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Default: 0 (actual/actual)

Principal

NINST-by-1 vector of the notional principal amount.

Default: 100

SwapReset

NINST-by-1 vector representing the reset frequency per year for the underlying forward swap.

Default: 1

Output Arguments

Price

NINST-by-1 vector of prices for the swaptions at time 0.

Definitions

Forward Swap

A forward swap is a swap that starts at a future date.

Examples

Price a European Swaption Using the Black Model Where the Yield Curve is Flat at 6%

Price a European swaption that gives the holder the right to enter in five years into a three-year paying swap where a fixed-rate of 6.2% is paid and floating is received. Assume that the yield curve is flat at 6% per annum with continuous compounding, the volatility of the swap rate is 20%, the principal is $100, and payments are exchanged semiannually.

Create the `RateSpec`.

```
Rate = 0.06;
Compounding  = -1;
ValuationDate = 'Jan-1-2010';
EndDates =   'Jan-1-2020';
Basis = 1;

RateSpec = intenvset('ValuationDate', ValuationDate,'StartDates', ValuationDate, ...
'EndDates', EndDates, 'Rates', Rate, 'Compounding', Compounding, 'Basis', Basis);

RateSpec =

            FinObj: 'RateSpec'
       Compounding: -1
              Disc: 0.5488
             Rates: 0.0600
          EndTimes: 10
        StartTimes: 0
          EndDates: 737791
        StartDates: 734139
     ValuationDate: 734139
             Basis: 1
       EndMonthRule: 1
```

Price the swaption using the Black model.

```
Settle = 'Jan-1-2011';
ExerciseDates = 'Jan-1-2016';
```

```
Maturity = 'Jan-1-2019';
Reset = 2;
Principal = 100;
Strike = 0.062;
Volatility = 0.2;
OptSpec = 'call';

Price= swaptionbyblk(RateSpec, OptSpec, Strike, Settle, ExerciseDates, Maturity, ...
Volatility, 'Reset', Reset, 'Principal', Principal, 'Basis', Basis)

Price =

    2.0710
```

Price a European Swaption Using the Black Model Where the Yield Curve Is Incrementally Increasing

Price a European swaption that gives the holder the right to enter into a 5-year receiving swap in a year, where a fixed rate of 3% is received and floating is paid. Assume that the 1-year, 2-year, 3-year, 4-year and 5- year zero rates are 3%, 3.4%, 3.7%, 3.9% and 4% with continuous compounding. The swap rate volatility is 21%, the principal is $1000, and payments are exchanged semiannually.

Create the `RateSpec`.

```
ValuationDate = 'Jan-1-2010';
EndDates = {'Jan-1-2011';'Jan-1-2012';'Jan-1-2013';'Jan-1-2014';'Jan-1-2015'};
Rates = [0.03; 0.034 ; 0.037; 0.039; 0.04;];
Compounding = -1;
Basis = 1;

RateSpec = intenvset('ValuationDate', ValuationDate, 'StartDates', ValuationDate, ...
'EndDates', EndDates, 'Rates', Rates, 'Compounding', Compounding,'Basis', Basis)

RateSpec =

           FinObj: 'RateSpec'
      Compounding: -1
```

```
                     Disc: [5x1 double]
                    Rates: [5x1 double]
                 EndTimes: [5x1 double]
               StartTimes: [5x1 double]
                 EndDates: [5x1 double]
               StartDates: 734139
            ValuationDate: 734139
                    Basis: 1
              EndMonthRule: 1
```

Price the swaption using the Black model.

```
Settle = 'Jan-1-2011';
ExerciseDates = 'Jan-1-2012';
Maturity = 'Jan-1-2017';
Strike = 0.03;
Volatility = 0.21;
Principal =1000;
Reset = 2;
OptSpec = 'put';

Price = swaptionbyblk(RateSpec, OptSpec, Strike, Settle,  ExerciseDates, ...
Maturity, Volatility,'Basis', Basis, 'Reset', Reset,'Principal', Principal)

Price =

    0.5771
```

See Also | bondbyzero | cfbyzero | fixedbyzero | floatbyzero

swaptionbyhjm

Purpose	Price swaption from Heath-Jarrow-Morton interest-rate tree

Syntax

```
[Price, PriceTree] = swaptionbyhjm(HJMTree, OptSpec, Strike,
ExerciseDates, Spread, Settle, Maturity,
'Name1', Value1, 'Name2', Value2)
```

Arguments

HJMTree	Interest-rate tree structure created by hjmtree.
OptSpec	NINST-by-1 cell array of strings 'call' or 'put'. A call swaption entitles the buyer to pay the fixed rate. A put swaption entitles the buyer to receive the fixed rate.
Strike	NINST-by-1 vector of strike swap rate values.
ExerciseDates	For a European option: NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one ExerciseDate on the option expiry date.
	For an American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if ExerciseDates is NINST-by-1, the option can be exercised between the underlying swap Settle and the single listed ExerciseDate.
Spread	NINST-by-1 vector representing the number of basis points over the reference rate.
Settle	NINST-by-1 vector of dates representing the settle date for each swap.

Maturity	NINST-by-1 vector of dates representing the maturity date for each swap.

Note All optional inputs that follow are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may be specified in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

AmericanOpt	(Optional) NINST-by-1 flags options:
	• 0 for European options
	• 1 for American options
SwapReset	(Optional) NINST-by-1 vector representing the reset frequency per year for the underlying swap. Default is 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Principal	(Optional) NINST-by-1 vector of the notional principal amounts. Default is 100.
Options	(Optional) Derivatives pricing options structure created with derivset.

Description   [Price, PriceTree] = swaptionbyhjm(HJMTree, OptSpec, Strike, ExerciseDates, Spread, Settle, Maturity, 'Name1', Value1, 'Name2', Value2) computes the price of a swaption from a HJM interest-rate tree.

Note The Settle date for every swaption is set to the ValuationDate of the HJM tree. The swap argument Settle is ignored.

The swaption may be a call swaption or a put swaption.

A call swaption or payer swaption allows the option buyer to enter into an interest rate swap in which the buyer of the option pays the fixed rate and receives the floating rate.

A put swaption or receiver swaption allows the option buyer to enter into an interest rate swap in which the buyer of the option receives the fixed rate and pays the floating rate.

Price is a (NINST-by-1 vector of expected swaption prices at time 0.

`PriceTree` is a MATLAB structure of trees containing vectors of swaption instrument prices and a vector of observation times for each node. Within `PriceTree`:

- `PriceTree.PTree` contains the clean prices.

- `PriceTree.tObs` contains the observation times.

Examples   Price a 1-year call swaption using an HJM interest-rate tree.

Assume that interest rate is fixed at 5% annually between the valuation date of the tree until its maturity. Build a tree with the following data.

Specify the `RateSpec`:

```
Rates = [ 0.05;0.05;0.05;0.05];
StartDates = 'jan-1-2007';
EndDates =['jan-1-2008';'jan-1-2009';'jan-1-2010';'jan-1-2011'];
ValuationDate = StartDates;
Compounding = 1;
RateSpec = intenvset('Rates', Rates, 'StartDates', StartDates, 'EndDates',...
EndDates, 'Compounding', Compounding);
```

Use `VolSpec` to compute the interest rate volatility:

```
VolSpec=hjmvolspec('Constant',0.01);
```

Use `TimeSpec` to specify the structure of the time layout for the HJM interest-rate tree:

```
 TimeSpec = hjmtimespec(ValuationDate, EndDates, Compounding);
```

Build the HJM tree:

```
HJMTree = hjmtree(VolSpec, RateSpec, TimeSpec);
```

Use the following swaption arguments:

```
SwapSettlement = 'jan-1-2007';
SwapMaturity   = 'jan-1-2010';
Spread = [0];
```

```
SwapReset = 1;
Basis  = 1;
Principal = 100;
OptSpec = 'call';
Strike=0.05;
ExerciseDates = '01-Jan-2008';
```

Price the swaption

```
[Price, PriceTree] = swaptionbyhjm(HJMTree, OptSpec, Strike, ExerciseDates, ...
Spread, SwapSettlement, SwapMaturity,'SwapReset', SwapReset, ...
'Basis', Basis, 'Principal', Principal)
```

to return

```
Price =

   0.9296

PriceTree =

    FinObj: 'HJMPriceTree'
    tObs: [5x1 double]
    PBush: {[0.9296]  [1x1x2 double]  [1x2x2 double]  [1x4x2 double]  [0 0 0 0 0 0 0 0]}
```

See Also   hjmtree | instswaption | swapbyhjm

swaptionbyhw

Purpose	Price swaption from Hull-White interest-rate tree

Syntax

```
[Price, PriceTree] = swaptionbyhw(HWTree, OptSpec, Strike,
ExerciseDates, Spread, Settle, Maturity,
'Name1', Value1, 'Name2', Value2)
```

Arguments

HWTree	Interest-rate tree structure created by `hwtree`.
OptSpec	NINST-by-1 cell array of strings `'call'` or `'put'`. A call swaption entitles the buyer to pay the fixed rate. A put swaption entitles the buyer to receive the fixed rate.
Strike	NINST-by-1 vector for strike swap rate values.
ExerciseDates	For a European option: NINST-by-1 vector of exercise dates. Each row is the schedule for one option. For a European option, there is only one `ExerciseDate` on the option expiry date.
	For an American option: NINST-by-2 vector of exercise date boundaries. For each instrument, the option can be exercised on any coupon date between or including the pair of dates on that row. If only one non-NaN date is listed, or if `ExerciseDates` is NINST-by-1, the option can be exercised between the underlying swap `Settle` and the single listed `ExerciseDate`.
Spread	NINST-by-1 vector representing the number of basis points over the reference rate.
Settle	NINST-by-1 vector of dates representing the settle date for each swap.

Maturity	NINST-by-1 vector of dates representing the maturity date for each swap.

Note All optional inputs that follow are specified as matching parameter name/value pairs. The parameter name is specified as a character string, followed by the corresponding parameter value. Parameter name/value pairs may be specified in any order; names are case-insensitive and partial string matches are allowed provided no ambiguities exist.

AmericanOpt	(Optional) NINST-by-1 flags options:
	• 0 for European options
	• 1 for American options
SwapReset	(Optional) NINST-by-1 vector representing the reset frequency per year for the underlying swap. Default is 1.
Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)

swaptionbyhw

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Principal	(Optional) NINST-by-1 vector of the notional principal amounts. Default is 100.
Options	(Optional) Derivatives pricing options structure created with `derivset`.

Description   [Price, PriceTree] = swaptionbyhw(HWTree, OptSpec, Strike, ExerciseDates, Spread, Settle, Maturity, 'Name1', Value1, 'Name2', Value2) computes the price of a swaption from a HW interest-rate tree.

> **Note** The Settle date for every swaption is set to the ValuationDate of the HW tree. The swap argument Settle is ignored.

The swaption may be a call swaption or a put swaption.

A call swaption or payer swaption allows the option buyer to enter into an interest rate swap in which the buyer of the option pays the fixed rate and receives the floating rate.

A put swaption or receiver swaption allows the option buyer to enter into an interest rate swap in which the buyer of the option receives the fixed rate and pays the floating rate.

Price is a NINST-by-1 vector of expected swaption prices at time 0.

`PriceTree` is a MATLAB structure of trees containing vectors of swaption instrument prices and a vector of observation times for each node. Within `PriceTree`:

- `PriceTree.PTree` contains the clean prices.

- `PriceTree.tObs` contains the observation times.

Examples   Price a 3-year put swaption using an HW interest-rate tree with the following data.

Specify the `RateSpec`:

```
Rates =0.075 * ones (10,1);
Compounding = 2;
StartDates = ['jan-1-2007';'jul-1-2007';'jan-1-2008';'jul-1-2008';'jan-1-2009';...
'jul-1-2009';'jan-1-2010'; 'jul-1-2010';'jan-1-2011';'jul-1-2011'];
EndDates =['jul-1-2007';'jan-1-2008';'jul-1-2008';'jan-1-2009';'jul-1-2009';...
'jan-1-2010';'jul-1-2010';'jan-1-2011';'jul-1-2011';'jan-1-2012'];
ValuationDate = 'jan-1-2007';
RateSpec = intenvset('Rates', Rates, 'StartDates', StartDates, 'EndDates',...
EndDates, 'Compounding', Compounding);
```

Use `HWVolSpec` to compute the interest rate volatility:

```
Volatility = 0.05*ones(10,1);
AlphaCurve = 0.01*ones(10,1);
AlphaDates = EndDates;
HWVolSpec = hwvolspec(ValuationDate, EndDates, Volatility, AlphaDates, AlphaCurve);
```

Use `HWTimeSpec` to specify the structure of the time layout for an HW interest-rate tree:

```
HWTimeSpec = hwtimespec(ValuationDate, EndDates, Compounding);
```

Build the HW tree:

```
HWTree = hwtree(HWVolSpec, RateSpec, HWTimeSpec);
```

Use the following arguments for a 5-year swap and 3-year swaption:

swaptionbyhw

```
SwapSettlement = 'jan-1-2007';
SwapMaturity   = 'jan-1-2012';
Spread = 0;
SwapReset = 2 ;
Principal = 100;
OptSpec = 'put';
Strike= 0.04;
ExerciseDates = 'jan-1-2010';
Basis=1;
```

Price the swaption

```
PriceSwaption = swaptionbyhw(HWTree, OptSpec, Strike, ExerciseDates, ...
Spread, SwapSettlement, SwapMaturity,'SwapReset', SwapReset, ...
'Basis', Basis,'Principal', Principal)
```

to return

```
PriceSwaption =

    2.9081
```

See Also   hwtree | instswaption | swapbyhw

Purpose   Price European swaption using Linear Gaussian two-factor model

Syntax   Price = swaptionbylg2f(ZeroCurve,a,b,sigma,eta,rho,
                 Strike,ExerciseDate,Maturity)
                 Price = swaptionbylg2f(___ , Name,Value)

Description   Price = swaptionbylg2f(ZeroCurve,a,b,sigma,eta,rho,
                 Strike,ExerciseDate,Maturity) returns the European swaption
                 price for a two-factor additive Gaussian interest-rate model.

                 Price = swaptionbylg2f(___ , Name,Value) returns the Eurpoean
                 swaption price for two-factor additive Gaussian interest-rate model
                 using optional name-value pairs.

Input Arguments

ZeroCurve - Zero-curve for Linear Gaussian two-factor model
structure

Zero-curve for the Linear Gaussian two-factor model, specified using
IRDataCurve or RateSpec.

Data Types
struct

a - Mean reversion for first factor for Linear Gaussian two-factor model
scalar

Mean reversion for first factor for the Linear Gaussian two-factor
model, specified as a scalar.

Data Types
single | double

b - Mean reversion for second factor for Linear Gaussian two-factor model
scalar

Mean reversion for second factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
`single | double`

sigma - Volatility for first factor for Linear Gaussian two-factor model
scalar

Volatility for first factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
`single | double`

eta - Volatility for second factor for Linear Gaussian two-factor model
scalar

Volatility for second factor for the Linear Gaussian two-factor model, specified as a scalar.

Data Types
`single | double`

rho - Scalar correlation of the factors
scalar

Scalar correlation of the factors, specified as a scalar.

Data Types
`single | double`

Strike - Swaption strike price
nonnegative integer | vector of nonnegative integers

Swaption strike price, specified as a nonnegative integer using a `NumSwaptions`-by-1 vector.

Data Types
single | double

ExerciseDate - Swaption exercise dates
nonnegative integer | vector of nonnegative integers

Swaption exercise dates, specified as a NumSwaptions-by-1 vector of serial date numbers or date strings.

Data Types
single | double

Maturity - Swaption maturity date
nonnegative integer | vector of nonnegative integers | string of dates

Swaption maturity date, specified using a NumSwaptions-by-1 vector of serial date numbers or date strings.

Data Types
single | double | cell

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: Price = swaptionbylg2f(irdc,a,b,sigma,eta,rho,Strike,ExerciseDate,Maturity,'Re

'Reset' - Frequency of swaption payments per year
2 (default) | positive integer from the set[1,2,3,4,6,12] | vector of positive integers from the set [1,2,3,4,6,12]

Frequency of swaption payments per year, specified as positive integers for the values 1,2,4,6,12] in a NumSwaptions-by-1 vector.

Data Types
single | double

'Notional' - Notional value of swaption
100 (default) | nonnegative integer | vector of nonnegative integers

Notional value of swaption, specified as a nonnegative integer using a
`NumSwaptions`-by-1 vector of notional amounts.

Data Types
`single | double`

'OptSpec' - Option specification for the swaption
`'call'` (default) | string with value of `'call'` or `'put'` | cell array of
string with values of `'call'` or `'put'`

Option specification for the swaption, specified as a string or a
`NumSwaptions`-by-1 cell array of strings with a value of `'call'` or `'put'`.
A `'call'` swaption entitles the buyer to pay the fixed rate. A `'put'`
swaption entitles the buyer to receive the fixed rate.

Data Types
`char | cell`

Output Arguments

Price - Swaption price
scalar | vector

Swaption price, returned as a scalar or an `NumSwaptions`-by-1 vector.

Algorithms
The following defines the swaption price for a two-factor additive
Gaussian interest-rate model, given the `ZeroCurve`, a, b, sigma, eta,
and rho parameters:

$$r(t) = x(t) + y(t) + \phi(t)$$

$$dx(t) = -a(x)(t)dt + \sigma(dW_1(t), x(0) = 0$$

$$dy(t) = -b(y)(t)dt + \eta(dW_2(t), y(0) = 0$$

where $dW_1(t)dW_2(t) = \rho dt$ is a two-dimensional Brownian motion with
correlation ρ and φ is a function chosen to match the initial zero curve.

Examples   **Price European Swaption for a Linear Gaussian Two-Factor Model**

Define the `ZeroCurve`, `a`, `b`, `sigma`, `eta`, and `rho` parameters to compute the price of the swaption.

```
Settle = datenum('15-Dec-2007');

ZeroTimes = [3/12 6/12 1 5 7 10 20 30]';
ZeroRates = [0.033 0.034 0.035 0.040 0.042 0.044 0.048 0.0475]';
CurveDates = daysadd(Settle,360*ZeroTimes,1);

irdc = IRDataCurve('Zero',Settle,CurveDates,ZeroRates);

a = .07;
b = .5;
sigma = .01;
eta = .006;
rho = -.7;

Reset = 1;
ExerciseDate = daysadd(Settle,360*5,1);
Maturity = daysadd(ExerciseDate,360*[3;4],1);
Strike = .05;

Price = swaptionbylg2f(irdc,a,b,sigma,eta,rho,Strike,ExerciseDate,Maturity,'Reset',Reset)

Price =

    1.1083
    1.4562
```

References

[1] Brigo, D. and F. Mercurio, *Interest Rate Models - Theory and Practice*, Springer Finance, 2006.

swaptionbylg2f

See Also   capbylg2f | floorbylg2fLinearGaussian2F **|**

Related   • "Price Swaptions With Interest-Rate Models Using Simulation" on
Examples   page 2-91

time2date

Purpose   Dates from time and frequency

Syntax   Dates = time2date(Settle, Times, Compounding, Basis, EndMonthRule)

Arguments

Settle	Settlement date. A vector of serial date numbers or date strings.
Times	Vector of times corresponding to the compounding value. Times must be equal to or greater than 0.
Compounding	(Optional) Scalar value representing the rate at which the input zero rates were compounded when annualized. Default = 2. This argument determines the formula for the discount factors:
	Compounding = 1, 2, 3, 4, 6, 12 = F
	Disc = (1 + Z/F)^(-T), where F is the compounding frequency, Z is the zero rate, and T is the time in periodic units; for example, T = F is 1 year.
	Compounding = 365
	Disc = (1 + Z/F)^(-T), where F is the number of days in the basis year and T is a number of days elapsed computed by basis.
	Compounding = -1
	Disc = exp(-T*Z), where T is time in years.

Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)
	• 10 = actual/365 (ICMA)
	• 11 = 30/360E (ICMA)
	• 12 = actual/actual (ISDA)
	• 13 = BUS/252
	For more information, see **basis** on page Glossary-1.
EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

Description   Dates = time2date(Settle, Times, Compounding, Basis,
EndMonthRule) computes dates corresponding to compounded rate
quotes between Settle and time factors.

> **Note** To obtain accurate results from this function, the Basis and
> Dates arguments must be consistent. If the Dates argument contains
> months that have 31 days, Basis must be one of the values that allow
> months to contain more than 30 days; for example, Basis = 0, 3, or 7.

The time2date function is the inverse of date2time.

Examples   Show that date2time and time2date are the inverse of each other.
First compute the time factors using date2time.

```
Settle = '1-Sep-2002';
Dates = datenum(['31-Aug-2005'; '28-Feb-2006'; '15-Jun-2006';
                 '31-Dec-2006']);
Compounding = 2;
Basis = 0;
EndMonthRule = 1;
Times = date2time(Settle, Dates, Compounding, Basis,...
                  EndMonthRule)


Times =

    5.9945
    6.9945
    7.5738
    8.6576
```

Now use the calculated Times in time2date and compare the calculated
dates with the original set.

```
Dates_calc = time2date(Settle, Times, Compounding, Basis,...
                  EndMonthRule)
```

```
Dates_calc =

     732555
     732736
     732843
     733042


datestr(Dates_calc)

ans =

31-Aug-2005
28-Feb-2006
15-Jun-2006
31-Dec-2006
```

See Also   cftimes | date2time | cfamounts | disc2rate | rate2disc

| **Purpose** | Entries from node of recombining binomial tree |

Syntax   Values = treepath(Tree, BranchList)

Arguments

| Tree | Recombining binomial tree. |
| BranchList | Number of paths (NUMPATHS) by path length (PATHLENGTH) matrix containing the sequence of branchings. |

Description   Values = treepath(Tree, BranchList) extracts entries of a node of a recombining binomial tree. The node path is described by the sequence of branchings taken, starting at the root. The top branch is number one, the second-to-top is two, and so on. Set the branch sequence to zero to obtain the entries at the root node.

Values is a number of values (NUMVALS)-by-NUMPATHS matrix containing the retrieved entries of a recombining tree.

Examples   Create a BDT tree by loading the example file.

load deriv.mat;

Then

FwdRates = treepath(BDTTree.FwdTree, [1 2 1])

returns the rates at the tree nodes located by taking the up branch, then the down branch, and finally the up branch again.

FwdRates =

    1.1000
    1.0979
    1.1377

```
1.1183
```

You can visualize this with the `treeviewer` function.

```
treeviewer(BDTTree)
```
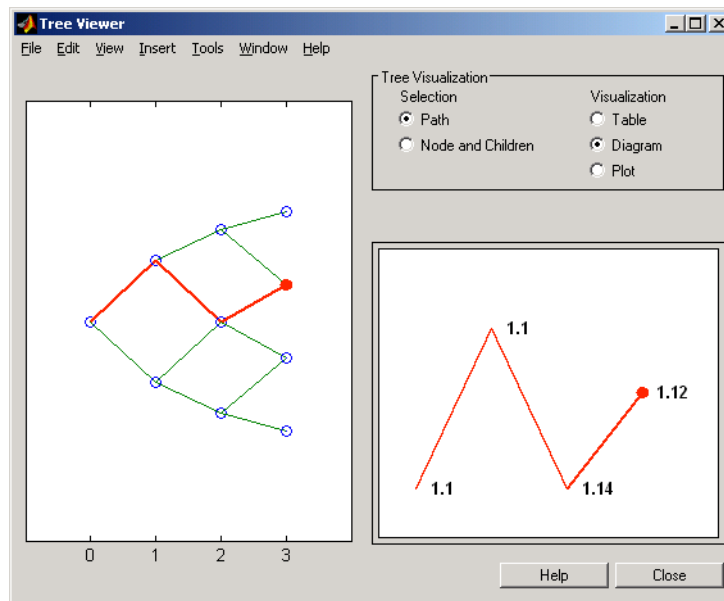

See Also   `mktree | treeshape`

Purpose   Shape of recombining binomial tree

Syntax   [NumLevels, NumPos, IsPriceTree] = treeshape(Tree)

Arguments

Tree	Recombining binomial tree.

Description   [NumLevels, NumPos, IsPriceTree] = treeshape(Tree) returns
                  information on a recombining binomial tree's shape.

                  NumLevels is the number of time levels of the tree.

                  NumPos is a 1-by-NUMLEVELS vector containing the length of the state
                  vectors in each level.

                  IsPriceTree is a Boolean determining if a final horizontal branch is
                  present in the tree.

Examples   Create a BDT tree by loading the example file.

                  load deriv.mat;

                  With treeviewer you can see the general shape of the BDT interest-rate
                  tree.

                  treeviewer(BDTTree)

treeshape
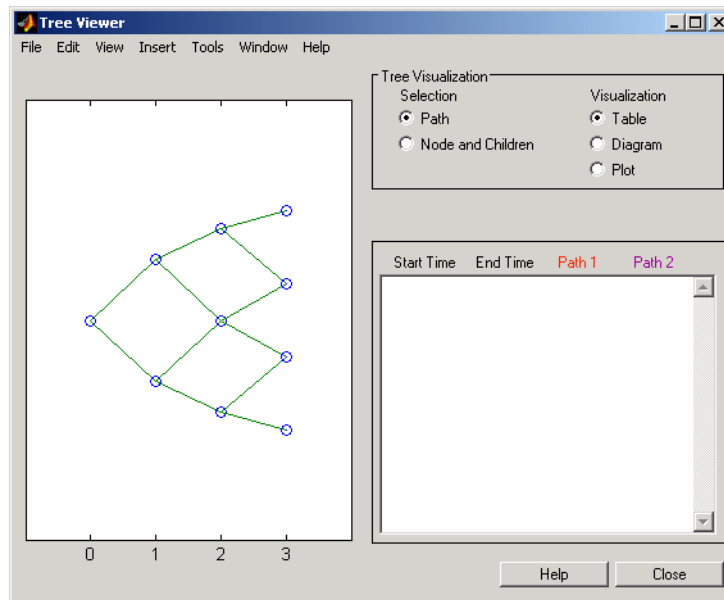
With this tree

```
[NumLevels, NumPos, IsPriceTree] = treeshape(BDTTree.FwdTree)
```

returns

```
NumLevels  =
     4

NumPos     =
     1     1     1     1

IsPriceTree =
     0
```

See Also   mktree | treepath

Purpose   Tree information

Syntax   treeviewer(Tree)
                   treeviewer(PriceTree, InstSet)
                   treeviewer(CFTree, InstSet)

Arguments

Tree	Tree can be any of the following types of trees.

Interest-rate trees:

- Black-Derman-Toy (BDTTree)

- Black-Karasinski (BKTree)

- Heath-Jarrow-Morton (HJMTree)

- Hull-White (HWTree)

  For information on creating interest-rate trees, see:

- bktree for information on creating BKTree.

- bdttree for information on creating BDTTree.

- hjmtree for information on creating HJMTree.

- hwtree for information on creating HWTree.

  Money market trees:

- Money market tree (MMktTree)

  For information on creating money-market trees, see:

- mmktbybdt for information on creating a money-market tree from a BDT interest-rate tree.

- mmktbyhjm for information on creating a money-market tree from an HJM interest-rate tree.

> **Note** Money market trees cannot be created from BK or HW interest-rate trees.

Stock price trees:

- Cox-Ross-Rubinstein (CRRTree)
- Implied Trinomial tree (ITTTree)
- Leisen-Reimer stock tree (LRTree)
- Equal probabilities (EQPTree)

  For information on creating stock price trees, see:

- crrtree for information on creating CRRTree.
- eqptree for information on creating EQPTree.
- itttree for information on creating ITTTree.
- lrtree for information on creating LRTree.

Cash flow trees:

- Black-Derman-Toy (BDTCFTree)
- Heath-Jarrow-Morton (HJMCFTree)

  Cash flow trees are created as outputs from the swap functions swapbyhjm and swapbybdt.

> **Note** For the function swapbybdt, which uses a recombining binomial tree, this structure contains only NaNs because cash flows cannot be accurately calculated at every tree node for floating-rate notes.

PriceTree	PriceTree is a Black-Derman-Toy (BDTPriceTree), Black-Karasinski (BKPriceTree), Heath-Jarrow-Morton (HJMPriceTree), Hull-White (HWPriceTree), Cox-Ross-Rubinstein (crrprice), Equal probabilities (eqpprice), or Implied Trinomial tree (ittprice) tree of instrument prices.
CFTree	CFTree is a tree of swap cash flows. You create cash flow trees when executing the Black-Derman-Toy and Heath-Jarrow-Morton swap functions. (Black-Derman-Toy cash flow trees contain only NaNs.)
InstSet	(Optional) Variable containing a collection of instruments whose prices or cash flows are contained in a tree. The collection can be created with the function instadd or as a cell array containing the names of the instruments. To display the names of the instruments, the field Name should exist in InstSet. If InstSet is not passed, treeviewer uses default instruments names (numbers) when displaying prices or cash flows.

Description   treeviewer(Tree) displays an interest rate, stock price, or money-market tree.

treeviewer(PriceTree, InstSet) displays a tree of instrument prices. If you provide the name of an instrument set (InstSet) and you have named the instruments using the field Name, the treeviewer display identifies the instrument being displayed with its name. (See Example 3 for a description.) If you do not provide the optional InstSet argument, the instruments are identified by their sequence number in the instrument set. (See Example 6 for a description.)

treeviewer(CFTree, InstSet) displays a cash flow tree that has been created with swapbybdt or swapbyhjm. If you provide the name of an instrument set (InstSet) containing cash flow names, the treeviewer

display identifies the instrument being displayed with its name. (See Example 3 for a description.) If the optional `InstSet` argument is not present, the instruments are identified by their sequence number in the instrument set. See Example 6 for a description.)

`treeviewer` price tree diagrams follow the convention that increasing prices appear on the upper branch of a tree and, consequently, decreasing prices appear on the lower branch. Conversely, for interest rate displays, *decreasing* interest rates appear on the upper branch (prices are rising) and *increasing* interest rates on the lower branch (prices are falling).

`treeviewer` provides an interactive display of prices or interest rates. The display is activated by clicking the nodes along the price or interest rate path shown in the left pane when the function is called. For HJM trees you select the endpoints of the path, and `treeviewer` displays all data from beginning to end. With recombining trees, such as BDT, BK and HW, you must click *each* node in succession from the beginning (`t = 1`) to the last node (`t = n`). Do not include the *root node*, the node at `t = 0`. If you do not click the nodes in the proper order, you are reminded with the message

```
Parent of selected node must be selected.
```

Examples

Example 1. Display an HJM Interest-Rate Tree.

```
load deriv.mat
treeviewer(HJMTree)
```

The `treeviewer` function displays the structure of an HJM tree in the left pane. The tree visualization in the right pane is blank.
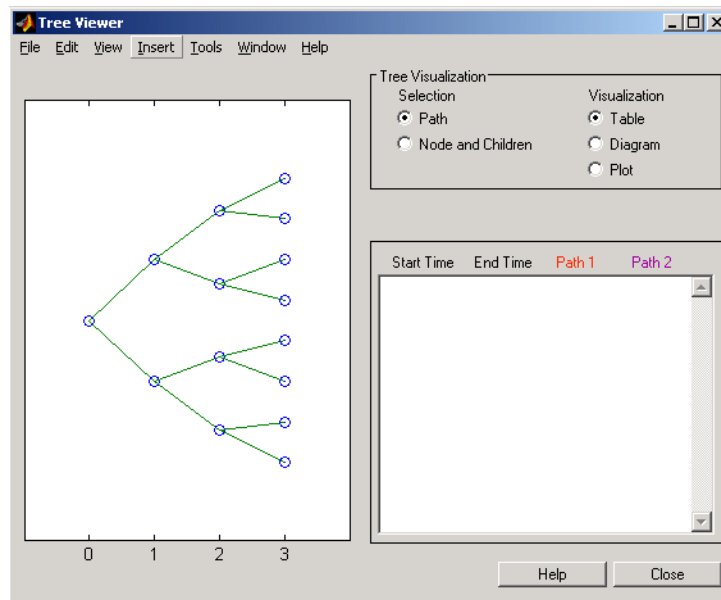
To visualize the actual interest-rate tree, go to the **Tree Visualization** pane and click on **Path** (the default) and **Diagram**. Now, select the first path by clicking on the last node (t = 3) of the upper branch.
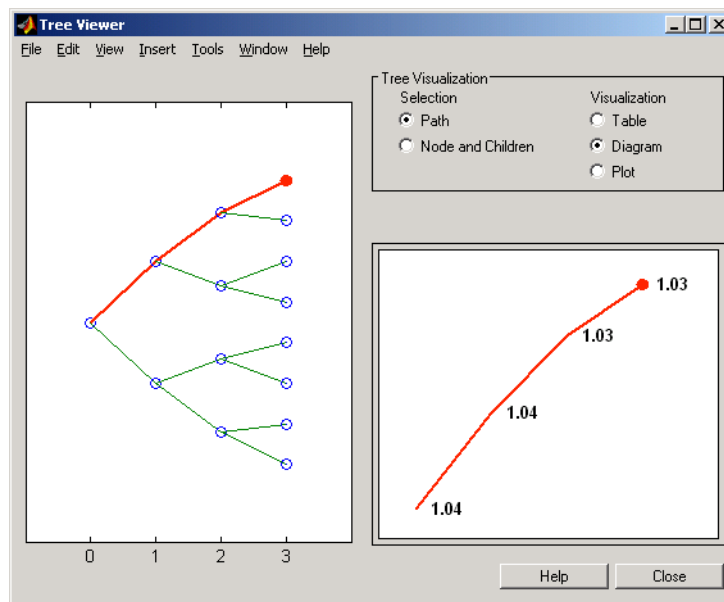
Note that the entire upper path is highlighted in red.

To complete the process, select a second path by clicking on the last node (t = 3) of another branch. The second path is highlighted in purple. The final display looks like this.
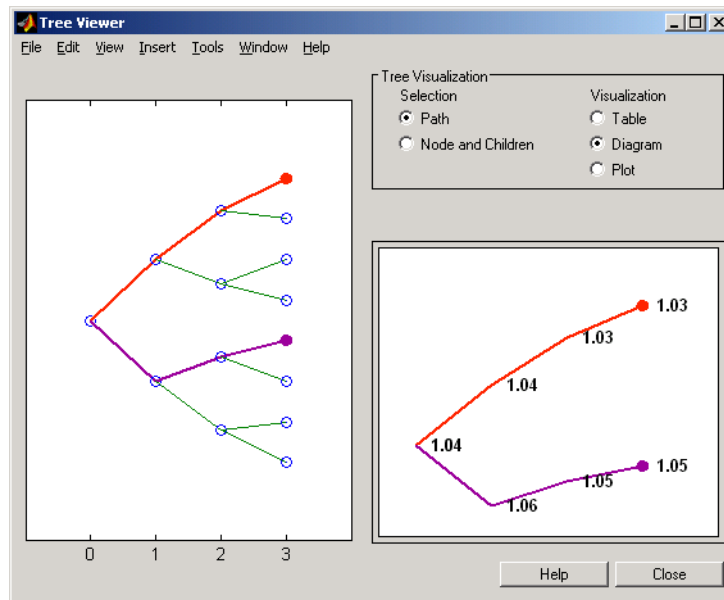
Alternative Forms of Display

The **Tree Visualization** pane allows you to select alternative ways to display tree data. For example, if you select **Path** and **Table** as your visualization choices, the final display above instead appears in tabular form.
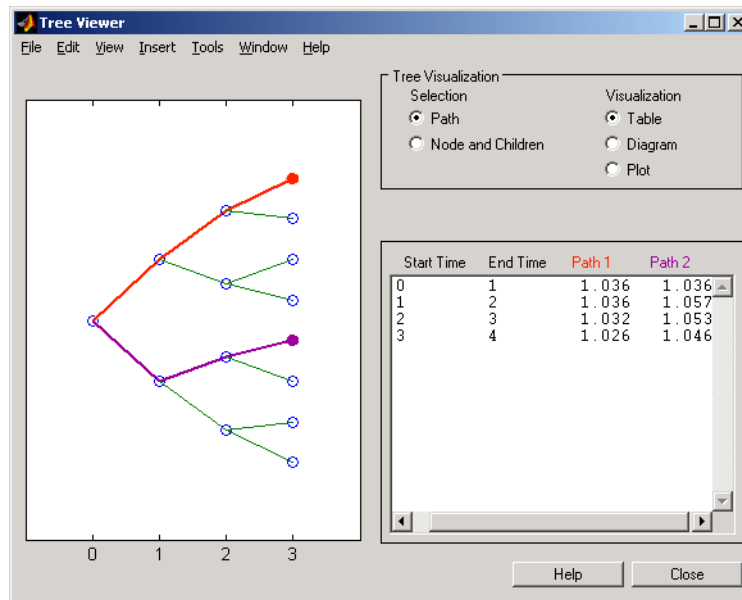
To see a plot of interest rates along the chosen branches, click **Path** and **Plot** in the **Tree Visualization** pane.
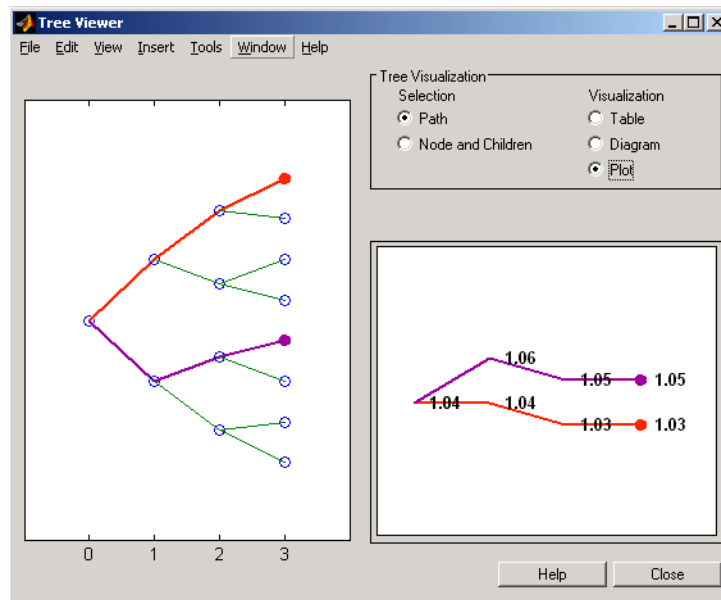
Note that with **Plot** selected, rising interest rates are shown on the upper branch and declining interest rates on the lower.

Finally, if you clicked **Node and Children** under **Tree Visualization**, you restrict the data displayed to just the selected parent node and its children.

treeviewer
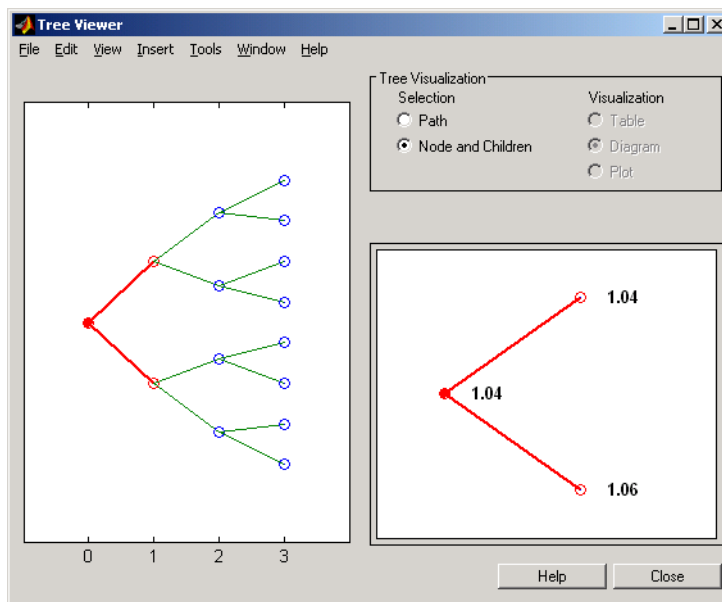
With **Node and Children** selected, the choices under **Visualization** are unavailable.

Example 2. Display a BDT Interest-Rate Tree.

```
load deriv.mat
treeviewer(BDTTree)
```

The treeviewer function displays the structure of a BDT tree in the left pane. The tree visualization in the right pane is blank.
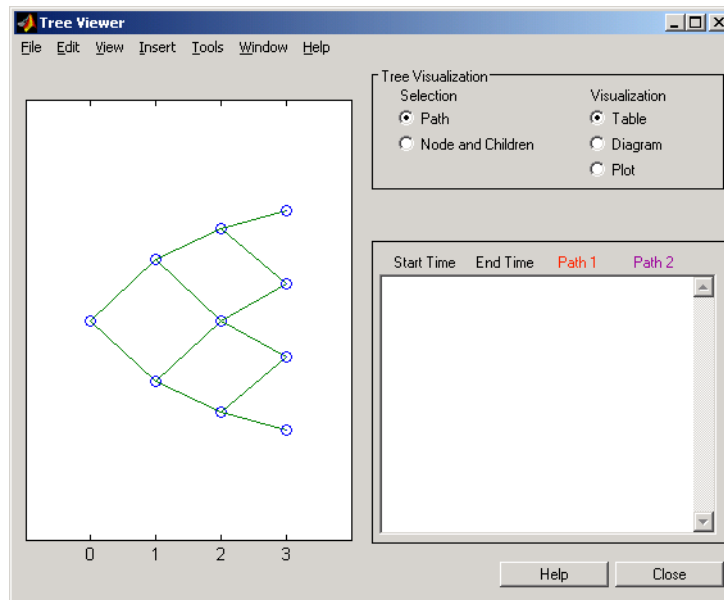
To visualize the actual interest-rate tree, go to the **Tree Visualization** pane and click **Path** (the default) and **Diagram**. Now, select the first path by clicking on the first node of the up branch (t = 1). Continue by clicking the down branch at the next node (t = 2). The two figures below show the treeviewer path diagrams for these selections.
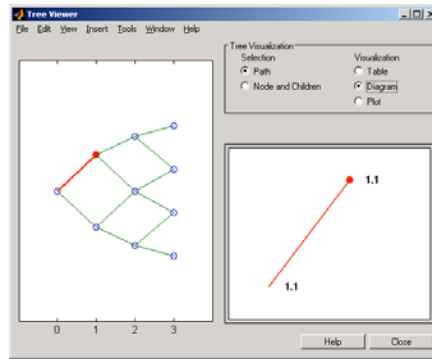
t = 1
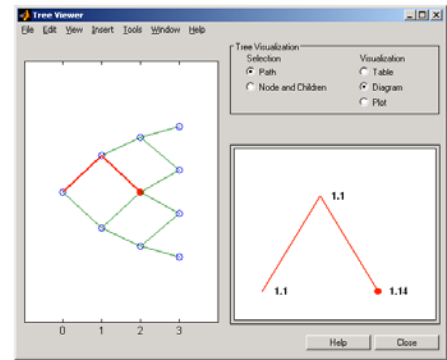
t = 2

Continue clicking all nodes in succession until you reach the end of the branch. Note that the entire path you have selected is highlighted in red.

Select a second path by clicking the first node of the lower branch (t = 1). Continue clicking lower nodes as you did on the first branch. Note that the second branch is highlighted in purple. The final display looks like this.
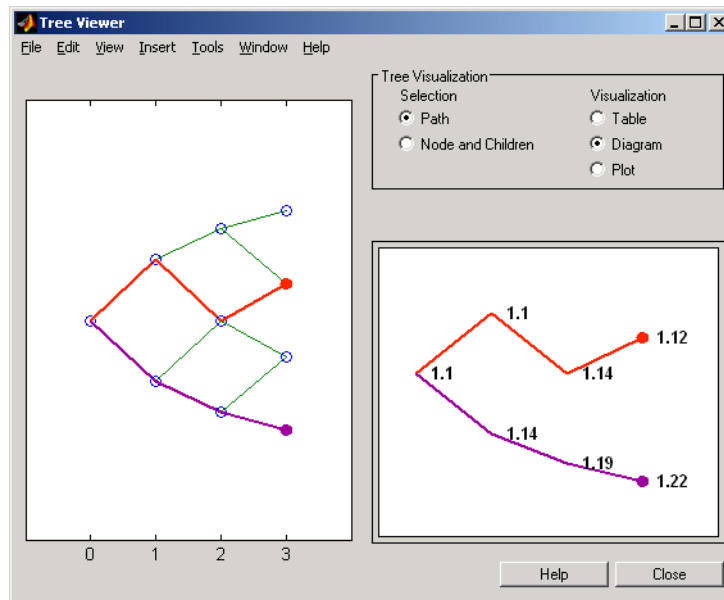
Example 3. Display an HJM Price Tree for Named Instruments.

```
load deriv.mat
[Price, PriceTree] = hjmprice(HJMTree, HJMInstSet);
treeviewer(PriceTree, HJMInstSet)
```

treeviewer
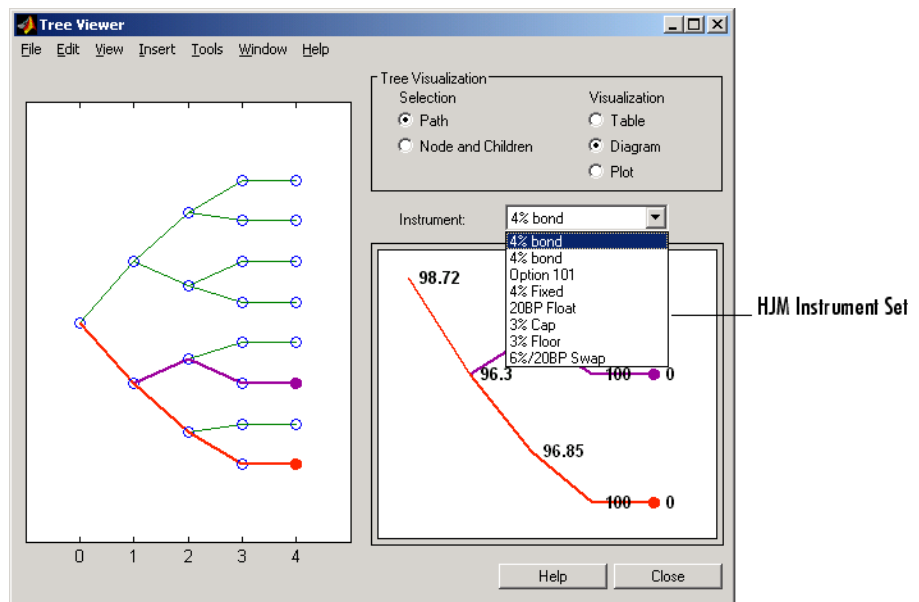
Example 4. Display a BDT Price Tree for Named Instruments.

```
load deriv.mat
[Price, PriceTree] = bdtprice(BDTTree, BDTInstSet);
treeviewer(PriceTree, BDTInstSet)
```
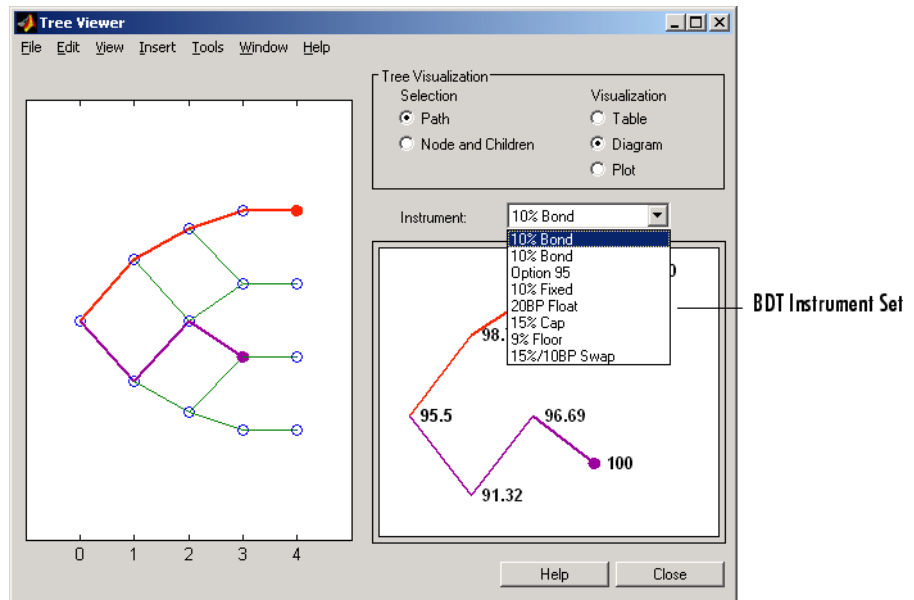
BDT Instrument Set

Example 5. Display an HJM Price Tree with Renamed Instruments.

```
load deriv.mat
[Price, PriceTree] = hjmprice(HJMTree, HJMInstSet);
Names = {'Bond1', 'Bond2', 'Option', 'Fixed','Float', 'Cap',...
'Floor', 'Swap'};
treeviewer(PriceTree, Names)
```
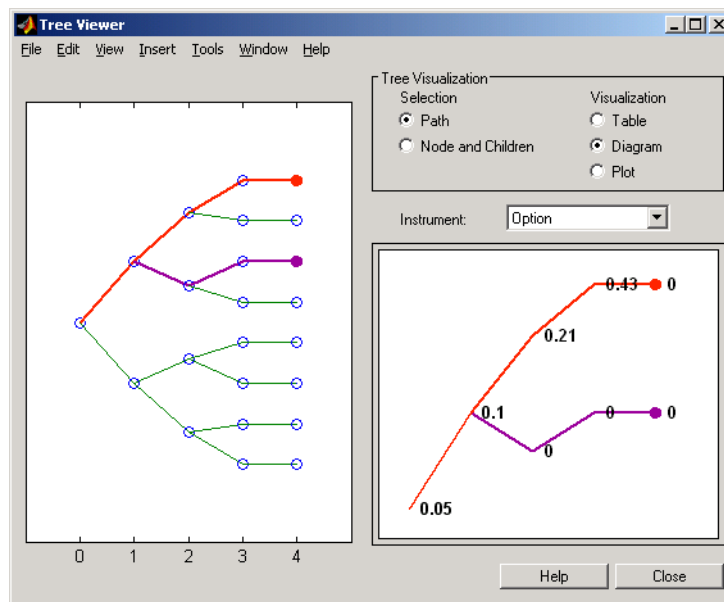
Example 6. Display an HJM Price Tree Using Default Instrument Names (Numbers).

```
load deriv.mat
[Price, PriceTree] = hjmprice(HJMTree, HJMInstSet);
treeviewer(PriceTree)
```
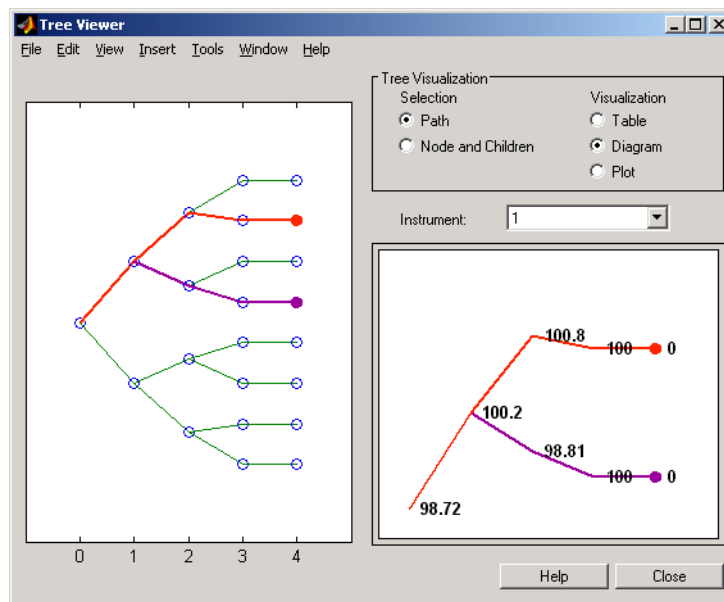
See Also bdttree | bktree | crrtree | eqptree | hjmtree | hwtree | instadd | itttree | lrtree | mmktbybdt | mmktbyhjm | swapbybdt | swapbyhjm

trintreepath

Purpose	Entries from node of recombining trinomial tree
Syntax	Values = trintreepath(TrinTree, BranchList)

Arguments

TrinTree	Recombining price or interest-rate trinomial tree.
BranchList	Number of paths (NUMPATHS) by path length (PATHLENGTH) matrix containing the sequence of branchings.

Description
Values = trintreepath(TrinTree, BranchList) extracts entries of a node of a recombining trinomial tree. The node path is described by the sequence of branchings taken, starting at the root. The top branch is number 1, the middle branch is 2, and the bottom branch is 3. Set the branch sequence to 0 to obtain the entries at the root node.

Values is a number of values (NUMVALS)-by-NUMPATHS matrix containing the retrieved entries of a recombining tree.

Examples
Create a Hull-White tree by loading the example file.

```
load deriv.mat;
```

Then, for example

```
FwdRates = trintreepath(HWTree, [1 2 3])
```

returns the rates at the tree nodes located by starting at 0, taking the up branch at the first node, the middle branch at the second node, and finally the bottom branch at the third node.

```
FwdRates =

    1.0279
```

```
1.0528
1.0652
1.0591
```

You can visualize this with the `treeviewer` function.

`treeviewer(HWTree)`
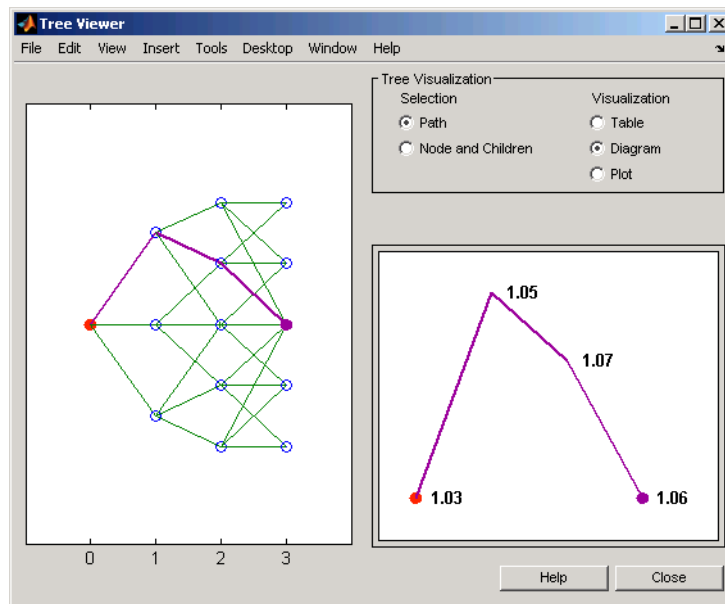
See Also   `mktrintree` | `trintreeshape`

trintreeshape

Purpose	Shape of recombining trinomial tree

Syntax

```
[NumLevels, NumPos, NumStates] = trintreeshape(TrinTree)
```

Arguments

TrinTree	Recombining price or interest-rate trinomial tree.

Description

[NumLevels, NumPos, NumStates] = trintreeshape(TrinTree) returns information on a recombining trinomial tree's shape.

NumLevels is the number of time levels of the tree.

NumPos is a 1-by-NUMLEVELS vector containing the length of the state vectors in each level.

NumStates is a 1-by-NUMLEVELS vector containing the number of state vectors in each level.

Examples

Create a Hull-White tree by loading the example file.

```
load deriv.mat;
```

With treeviewer you can see the general shape of the HW interest-rate tree.

```
treeviewer(HWTree)
```
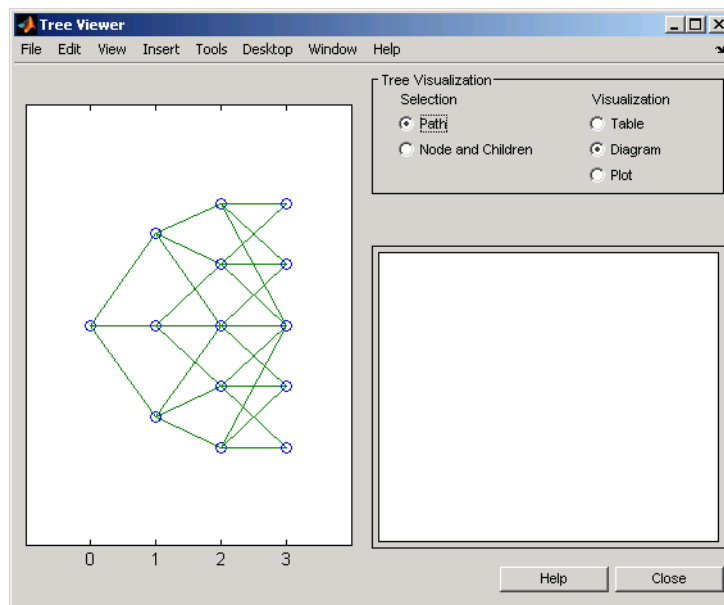
With this tree

```
[NumLevels, NumPos, NumStates] = trintreeshape(HWTree)
```

returns

```
NumLevels  =
     4

NumPos     =
    1     1     1     1

NumStates =
    1     3     5     5
```

See Also   mktrintree | trintreepath

Purpose	Determine option-adjusted spread of callable bond using Agency OAS model
Syntax	OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate) OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate, Name,Value)
Description	OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate) computes OAS of a callable bond given price using the Agency OAS model. OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate, Name,Value) computes OAS of a callable bond given price using the Agency OAS model with additional options specified by one or more Name,Value pair arguments.
Input Arguments	**ZeroData** Zero curve represented as a numRates-by-2 matrix where the first column is zero dates and the second column is the accompanying zero rates. **Price** numBonds-by-1 vector of prices. **CouponRate** numBonds-by-1 vector of coupon rates in decimal form. **Settle** Scalar MATLAB date number for the settlement date for all bonds and the zero data.

Note The Settle date must be an identical settlement date for all the bonds and the zero curve.

Maturity

numBonds-by-1 vector of maturity dates.

Vol

numBonds-by-1 vector of volatilities in decimal form. This is the volatility of interest rates corresponding to the time of the CallDate.

CallDate

numBonds-by-1 vector of call dates.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

'Basis'

N-by-1 vector of day-count basis:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

'CurveBasis'

Basis of the zero curve, where the choices are identical to Basis.

> **Default:** 0 (actual/actual)

'CurveCompounding'

Compounding frequency of the zero curve. Possible values include: −1, 0, 1, 2 , 3, 4, 6, 12.

> **Default:** 2 (Semi-annual)

'EndMonthRule'

End-of-month rule; 1, indicating in effect, and 0, indicating rule not in effect for the bond(s). When 1, the rule is in effect for the bond(s), this means that a security that pays coupon interest on the last day of the month will always make payment on the last day of the month.

> **Default:** 1 — Indicates in effect

'Face'

Face value of the bond.

>   **Default:** 100

'FirstCouponDate'

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

>   **Default:** If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

'InterpMethod'

Interpolation method used to obtain points from the zero curve. Values are:

- linear — linear interpolation
- cubic — piecewise cubic spline interpolation
- pchip — piecewise cubic Hermite interpolation

>   **Default:** linear

'IssueDate'

Bond issue date.

>   **Default:** If you do not specify an IssueDate, the cash flow payment dates are determined from other inputs.

'LastCouponDate'

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at

the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

> **Default:** If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

'Period'

Number of coupon payments per year. Possible values include: 0, 1, 2 , 3, 4, 6, 12.

> **Default:** 2

'StartDate'

Forward starting date of payments.

> **Default:** If you do not specify a `StartDate`, the effective start date is the `Settle` date.

Output Arguments

OAS

`numBonds`-by-1 matrix of option-adjusted spreads.

Definitions

Agency OAS Model

The BMA European Callable Securities Formula provides a standard methodology for computing price and option-adjusted spread for European Callable Securities (ECS).

Examples

Compute the agency `OAS` value:

```
Settle = datenum('20-Jan-2010');
ZeroRates = [.07 .164 .253 1.002 1.732 2.226 2.605 3.316 ...
3.474 4.188 4.902]'/100;
ZeroDates = daysadd(Settle,360*[.25 .5 1 2 3 4 5 7 10 20 30],1);
ZeroData = [ZeroDates ZeroRates];

Maturity = datenum('30-Dec-2013');
```

```
CouponRate = .022;
Price = 99.155;
Vol = .5117;
CallDate = datenum('30-Dec-2010');
OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate)

OAS =

8.6279
```

References   SIFMA, The BMA European Callable Securities Formula,
                 http://www.sifma.org.

See Also | agencyprice

Tutorials   • "Agency Option-Adjusted Spreads" on page 6-2

agencyprice

Purpose	Price callable bond using Agency OAS model
Syntax	`Price = agencyprice(ZeroData, OAS, CouponRate, Settle, Maturity, Vol, CallDate)` `Price = agencyprice(ZeroData, OAS, CouponRate, Settle, Maturity, Vol, CallDate, Name,Value)`

Description `Price = agencyprice(ZeroData, OAS, CouponRate, Settle, Maturity, Vol, CallDate)` computes the price for a callable bond, given OAS, using the Agency OAS model.

`Price = agencyprice(ZeroData, OAS, CouponRate, Settle, Maturity, Vol, CallDate, Name,Value)` computes the price for a callable bond, given OAS, using the Agency OAS model with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments

ZeroData

Zero curve represented as a `numRates`-by-2 matrix where the first column is zero dates and the second column is the accompanying zero rates.

OAS

`numBonds`-by-1 vector of option-adjusted spreads, expressed as a decimal (i.e., 50 basis points is entered as `.005`).

CouponRate

`numBonds`-by-1 vector of coupon rates in decimal form.

Settle

Scalar MATLAB date number for the settlement date for all the bonds and the zero data.

> **Note** The Settle date must be an identical settlement date for all
> bonds and the zero curve.

Maturity

numBonds-by-1 vector of maturity dates.

Vol

numBonds-by-1 vector of volatilities in decimal form. This is the
volatility of interest rates corresponding to the time of the CallDate.

CallDate

numBonds-by-1 vector of call dates.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments.
Name is the argument name and Value is the corresponding
value. Name must appear inside single quotes (' '). You can
specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

'Basis'

N-by-1 vector of day-count basis:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

'CurveBasis'

Basis of the zero curve, where the choices are identical to Basis.

> **Default:** 0 (actual/actual)

'CurveCompounding'

Compounding frequency of the curve. Possible values include: −1, 0, 1, 2 , 3, 4, 6, 12.

> **Default:** 2 (Semi-annual)

'EndMonthRule'

End-of-month rule; 1, indicating in effect, and 0, indicating rule not in effect for the bond(s). When 1, the rule is in effect for the bond(s). This means that a security that pays coupon interest on the last day of the month will always make payment on the last day of the month.

> **Default:** 1 — Indicates in effect

'Face'

Face value of the bond.

> **Default:** 100

'FirstCouponDate'

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

> **Default:** If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

'InterpMethod'

Interpolation method used to obtain points from the zero curve. Values are:

- linear — linear interpolation
- cubic — piecewise cubic spline interpolation
- pchip — piecewise cubic Hermite interpolation

> **Default:** linear

'IssueDate'

Bond issue date.

> **Default:** If you do not specify an IssueDate, the cash flow payment dates are determined from other inputs.

'LastCouponDate'

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at

the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

> **Default:** If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

'Period'

Number of coupon payments per year. Possible values include: 0, 1, 2 , 3, 4, 6, 12.

> **Default:** 2

'StartDate'

Forward starting date of payments.

> **Default:** If you do not specify a `StartDate`, the effective start date is the `Settle` date.

Output Arguments

Price

`numBonds`-by-1 matrix of the price.

Definitions

Agency OAS Model

The BMA European Callable Securities Formula provides a standard methodology for computing price and option-adjusted spread for European Callable Securities (ECS).

Examples

Compute the agency `Price`:

```
Settle = datenum('20-Jan-2010');
ZeroRates = [.07 .164 .253 1.002 1.732 2.226 2.605 3.316 ...
3.474 4.188 4.902]'/100;
ZeroDates = daysadd(Settle,360*[.25 .5 1 2 3 4 5 7 10 20 30],1);
ZeroData = [ZeroDates ZeroRates];

Maturity = datenum('30-Dec-2013');
```

```
CouponRate = .022;
OAS = 6.53/10000;
Vol = .5117;
CallDate = datenum('30-Dec-2010');
Price = agencyprice(ZeroData, OAS, CouponRate, Settle, Maturity, Vol, CallDate)

Price =

99.4226
```

References   SIFMA, The BMA European Callable Securities Formula,
                   http://www.sifma.org.

See Also | agencyoas

Tutorials   • "Agency Option-Adjusted Spreads" on page 6-2

bkcall

Purpose	Price European call option on bonds using Black model
Syntax	CallPrice = bkcall(Strike, ZeroData, Sigma, BondData, Settle, Expiry, Period, Basis, EndMonthRule, InterpMethod, StrikeConvention)

Arguments

Strike	Scalar or number of options (NOPT)-by-1 vector of strike prices.
ZeroData	Two-column (optionally three-column) matrix containing zero (spot) rate information used to discount future cash flows.
	• Column 1: Serial maturity date associated with the zero rate in the second column.
	• Column 2: Annualized zero rates, in decimal form, appropriate for discounting cash flows occurring on the date specified in the first column. All dates must occur after Settle (dates must correspond to future investment horizons) and must be in ascending order.
	• Column 3 (optional): Annual compounding frequency. Values are 1 (annual), 2 (semiannual, default), 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).
Sigma	Scalar or NOPT-by-1 vector of annualized price volatilities required by Black's model.

BondData	Row vector with three (optionally four) columns or `NOPT`-by-`3` (optionally `NOPT`-by-`4`) matrix specifying characteristics of underlying bonds in the form: `[CleanPrice CouponRate Maturity Face]` `CleanPrice` is the price excluding accrued interest. `CouponRate` is the decimal coupon rate. `Maturity` is the bond maturity date in serial date number format. `Face` is the face value of the bond. If unspecified, the face value is assumed to be 100.
Settle	Settlement date of the options. May be a serial date number or date string. `Settle` also represents the starting reference date for the input zero curve.
Expiry	Scalar or `NOPT`-by-`1` vector of option maturity dates. May be a serial date number or date string.
Period	(Optional) Number of coupons per year for the underlying bond. Default = 2 (semiannual). Supported values are 0, 1, 2, 3, 4, 6, and 12.

bkcall

Basis	(Optional) Day-count basis of the bond. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)
	• 10 = actual/365 (ICMA)
	• 11 = 30/360E (ICMA)
	• 12 = actual/actual (ISDA)
	• 13 = BUS/252
	For more information, see **basis** on page Glossary-1.
EndMonthRule	(Optional) End-of-month rule. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

InterpMethod	(Optional) Scalar integer zero curve interpolation method. For cash flows that do not fall on a date found in the `ZeroData` spot curve, indicates the method used to interpolate the appropriate zero discount rate. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See `interp1` for more information.
StrikeConvention	(Optional) Scalar or `NOPT`-by-1 vector of option contract strike price conventions.
	`StrikeConvention = 0` (default) defines the strike price as the cash (dirty) price paid for the underlying bond.
	`StrikeConvention = 1` defines the strike price as the quoted (clean) price paid for the underlying bond. When evaluating Black's model, the accrued interest of the bond at option expiration is added to the input strike price.

Description   CallPrice = bkcall(Strike, ZeroData, Sigma, BondData, Settle, Expiry, Period, Basis, EndMonthRule, InterpMethod, StrikeConvention) using Black's model, derives an NOPT-by-1 vector of prices of European call options on bonds.

If cash flows occur beyond the dates spanned by ZeroData, the input zero curve, the appropriate zero rate for discounting such cash flows is obtained by extrapolating the nearest rate on the curve (that is, if a cash flow occurs before the first or after the last date on the input zero curve, a flat curve is assumed).

In addition, you can use the Financial Instruments Toolbox method getZeroRates for an IRDataCurve object with a Dates property to create a vector of dates and data acceptable for bkcall. For more information, see "Converting an IRDataCurve or IRFunctionCurve Object" on page 9-38.

bkcall

Examples   This example is based on Example 22.1, page 512, of Hull. (See References below.)

Consider a European call option on a bond maturing in 9.75 years. The underlying bond has a clean price of \$935, a face value of \$1000, and pays 10% semiannual coupons. Since the bond matures in 9.75 years, a \$50 coupon will be paid in 3 months and again in 9 months. Also, assume that the annualized volatility of the forward bond price is 9%. Furthermore, suppose the option expires in 10 months and has a strike price of \$1000, and that the annualized continuously compounded risk-free discount rates for maturities of 3, 9, and 10 months are 9%, 9.5%, and 10%, respectively.

```
% Specify the option information.
Settle      = '15-Mar-2004';
Expiry      = '15-Jan-2005'; % 10 months from settlement
Strike      = 1000;
Sigma       = 0.09;
Convention  = [0 1]';

% Specify the interest-rate environment.
ZeroData    = [datenum('15-Jun-2004')  0.09   -1; % 3 months
               datenum('15-Dec-2004')  0.095  -1; % 9 months
               datenum(Expiry)         0.10   -1]; % 10 months

% Specify the bond information.
CleanPrice  = 935;
CouponRate  = 0.1;
Maturity    = '15-Dec-2013';  % 9.75 years from settlement
Face        = 1000;
BondData    = [CleanPrice CouponRate datenum(Maturity) Face];
Period      = 2;
Basis       = 1;

% Call Black's model.
CallPrices = bkcall(Strike, ZeroData, Sigma, BondData, Settle,...
Expiry, Period, Basis, [], [], Convention)
```

```
CallPrices =

   9.4873
   7.9686
```

When the strike price is the dirty price (Convention = 0), the call option value is $9.49. When the strike price is the clean price (Convention = 1), the call option value is $7.97.

References   [1] Hull, John C., *Options, Futures, and Other Derivatives*, Prentice Hall, 5th edition, 2003, pp. 287-288, 508-515.

See Also   bkput

bkcaplet

Purpose   Price interest-rate caplet using Black model

Syntax   CapPrices = bkcaplet(CapData, FwdRates, ZeroPrice, Settle, StartDate, EndDate, Sigma)

Arguments

CapData	Number of caps (NCAP)-by-2 matrix containing cap rates and bases: [CapRates Basis]. Values for bases may be:

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

FwdRates	Scalar or NCAP-by-1 vector containing forward rates in decimal. FwdRates accrue on the same basis as CapRates.

ZeroPrice	Scalar or NCAP-by-1 vector containing zero coupon prices with maturities corresponding to those of each cap in CapData, per $100 nominal value.
Settle	Scalar or NCAP-by-1 vector of identical elements containing settlement date of caplets.
StartDate	Scalar or NCAP-by-1 vector containing start dates of the caplets.
EndDate	Scalar or NCAP-by-1 vector containing maturity dates of caplets.
Sigma	Scalar or NCAP-by-1 vector containing volatility of forward rates in decimal, corresponding to each caplet.

Description   CapPrices = bkcaplet(CapData, FwdRates, ZeroPrice, Settle, StartDate, EndDate, Sigma) computes the prices of interest-rate caplets for every $100 face value of principal.

Examples   Given a notional amount of $1,000,000, compute the value of a caplet on October 15, 2002 that starts on October 15, 2003 and ends on January 15, 2004.

```
CapData = [0.08, 1];
FwdRates = 0.07;
ZeroPrice = 100*exp(-0.065*1.25);
Settle = datenum('15-Oct-2002');
BeginDates = datenum('15-Oct-2003');
EndDates = datenum('15-Jan-2004');
Sigma = 0.20;
```

Because the caplet is $100 notional, divide $1,000,000 by $100.

```
Notional  = 1000000/100;

CapPrice = Notional*bkcaplet(CapData, FwdRates, ZeroPrice, ...
Settle, BeginDates, EndDates, Sigma)
```

```
CapPrice =

  519.0046
```

See Also   bkfloorlet

Purpose   Price interest-rate floorlet using Black model

Syntax   FloorPrices = bkfloorlet(FloorData, FwdRates, ZeroPrice, Settle, StartDate, EndDate, Sigma)

Arguments

FloorData   Number of floors (NFLR)-by-2 matrix containing floor
              rates and bases: [FloorRate Basis].
              Values for bases may be:

              • 0 = actual/actual (default)

              • 1 = 30/360 (SIA)

              • 2 = actual/360

              • 3 = actual/365

              • 4 = 30/360 (BMA)

              • 5 = 30/360 (ISDA)

              • 6 = 30/360 (European)

              • 7 = actual/365 (Japanese)

              • 8 = actual/actual (ICMA)

              • 9 = actual/360 (ICMA)

              • 10 = actual/365 (ICMA)

              • 11 = 30/360E (ICMA)

              • 12 = actual/actual (ISDA)

              • 13 = BUS/252
              For more information, see **basis** on page Glossary-1.

FwdRates   Scalar or NFLR-by-1 vector containing forward rates
              in decimal. FwdRates accrue on the same basis as
              FloorRates.

bkfloorlet

ZeroPrice	Scalar or NFLR-by-1 vector containing zero coupon prices with maturities corresponding to those of each floor in FloorData, per $100 nominal value.
Settle	Scalar or NFLR-by-1 vector of identical elements containing settlement date of floorlets.
StartDate	Scalar or NFLR-by-1 vector containing start dates of the floorlets.
EndDate	Scalar or NFLR-by-1 vector containing maturity dates of floorlets.
Sigma	Scalar or NFLR-by-1 vector containing volatility of forward rates in decimal, corresponding to each floorlet.

Description FloorPrices = bkfloorlet(FloorData, FwdRates, ZeroPrice, Settle, StartDate, EndDate, Sigma) computes the prices of interest-rate floorlets for every $100 of notional value.

Examples Given a notional amount of $1,000,000, compute the value of a floorlet on October 15, 2002 that starts on October 15, 2003 and ends on January 15, 2004.

```
FloorData = [0.08, 1];
FwdRates = 0.07;
ZeroPrice = 100*exp(-0.065*1.25);
Settle = datenum('15-Oct-2002');
BeginDates = datenum('15-Oct-2003');
EndDates = datenum('15-Jan-2004');
Sigma = 0.20;

% Because floorlet is $100 notional, divide $1,000,000 by $100.
Notional = 1000000/100;

FloorPrice = Notional*bkfloorlet(FloorData, FwdRates, ...
ZeroPrice, Settle, BeginDates, EndDates, Sigma)

FloorPrice =
```

2823.91

See Also   bkcaplet

bkput

Purpose   Price European put option on bonds using Black model

Syntax   PutPrice = bkput(Strike, ZeroData, Sigma, BondData, Settle, Expiry, Period, Basis, EndMonthRule, InterpMethod, StrikeConvention)

Arguments

Strike	Scalar or number of options (NOPT)-by-1 vector of strike prices.
ZeroData	Two-column (optionally three-column) matrix containing zero (spot) rate information used to discount future cash flows.
	• Column 1: Serial maturity date associated with the zero rate in the second column.
	• Column 2: Annualized zero rates, in decimal form, appropriate for discounting cash flows occurring on the date specified in the first column. All dates must occur after Settle (dates must correspond to future investment horizons) and must be in ascending order.
	• Column 3 (optional): Annual compounding frequency. Values are 1 (annual), 2 (semiannual, default), 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).
Sigma	Scalar or NOPT-by-1 vector of annualized price volatilities required by Black's model.

BondData	Row vector with three (optionally four) columns or NOPT-by-3 (optionally NOPT-by-4) matrix specifying characteristics of underlying bonds in the form [CleanPrice CouponRate Maturity Face] where:
	• CleanPrice is the price excluding accrued interest.
	• CouponRate is the decimal coupon rate.
	• Maturity is the bond maturity date in serial date number format.
	• Face is the face value of the bond. If unspecified, the face value is assumed to be 100.
Settle	Settlement date of the options. May be a serial date number or date string. Settle also represents the starting reference date for the input zero curve.
Expiry	Scalar or NOPT-by-1 vector of option maturity dates. May be a serial date number or date string.
Period	(Optional) Number of coupons per year for the underlying bond. Default = 2 (semiannual). Supported values are 0, 1, 2, 3, 4, 6, and 12.

bkput

Basis	(Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule	(Optional) End-of-month rule. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

InterpMethod	(Optional) Scalar integer zero curve interpolation method. For cash flows that do not fall on a date found in the ZeroData spot curve, indicates the method used to interpolate the appropriate zero discount rate. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See interp1 for more information.
StrikeConvention	(Optional) Scalar or NOPT-by-1 vector of option contract strike price conventions.
	StrikeConvention = 0 (default) defines the strike price as the cash (dirty) price paid for the underlying bond.
	StrikeConvention = 1 defines the strike price as the quoted (clean) price paid for the underlying bond. The accrued interest of the bond at option expiration is added to the input strike price when evaluating Black's model.

Description   PutPrice = bkput(Strike, ZeroData, Sigma, BondData, Settle, Expiry, Period, Basis, EndMonthRule, InterpMethod, StrikeConvention) using Black's model, derives an NOPT-by-1 vector of prices of European put options on bonds.

If cash flows occur beyond the dates spanned by ZeroData, the input zero curve, the appropriate zero rate for discounting such cash flows is obtained by extrapolating the nearest rate on the curve (that is, if a cash flow occurs before the first or after the last date on the input zero curve, a flat curve is assumed).

In addition, you can use the Financial Instruments Toolbox method getZeroRates for an IRDataCurve object with a Dates property to create a vector of dates and data acceptable for bkput. For more information, see "Converting an IRDataCurve or IRFunctionCurve Object" on page 9-38.

bkput

Examples   This example is based on example 22.2, page 514, of Hull. (See References below.)

Consider a European put option on a bond maturing in 10 years. The underlying bond has a clean price of $122.82, a face value of $100, and pays 8% semiannual coupons. Also, assume that the annualized volatility of the forward bond yield is 20%. Furthermore, suppose the option expires in 2.25 years and has a strike price of $115, and that the annualized continuously compounded risk free zero (spot) curve is flat at 5%. For a hypothetical settlement date of March 15, 2004, the following code illustrates the use of Black's model to duplicate the put prices in Example 22.2 of the Hull reference. In particular, it illustrates how to convert a broker's yield volatility to a price volatility suitable for Black's model.

```
% Specify the option information.
Settle      = '15-Mar-2004';
Expiry      = '15-Jun-2006';  % 2.25 years from settlement
Strike      = 115;
YieldSigma  = 0.2;
Convention  = [0; 1];

% Specify the interest-rate environment. Since the
% zero curve is flat, interpolation into the curve always returns
% 0.05. Thus, the following curve is not unique to the solution.
ZeroData    = [datenum('15-Jun-2004') 0.05   -1;
               datenum('15-Dec-2004') 0.05   -1;
               datenum(Expiry)        0.05   -1];

% Specify the bond information.
CleanPrice  = 122.82;
CouponRate  = 0.08;
Maturity    = '15-Mar-2014'; % 10 years from settlement
Face        = 100;
BondData    = [CleanPrice CouponRate datenum(Maturity) Face];
Period      = 2; % semiannual coupons
Basis       = 1; % 30/360 day-count basis
```

```
% Convert a broker's yield volatility quote to a price volatility
% required by Black's model. To duplicate Example 22.2 in Hull,
% first compute the periodic (semiannual) yield to maturity from
% the clean bond price.
Yield  = bndyield(CleanPrice, CouponRate, Settle, Maturity,...
Period, Basis);

% Compute the duration of the bond at option expiration. Most
% fixed-income sensitivity analyses use the modified duration
% statistic to examine the impact of small changes in periodic
% yields on bond prices. However, Hull's example operates in
% continuous time (annualized instantaneous volatilities and
% continuously compounded zero yields for discounting coupons).
% To duplicate Hull's results, use the second output of BNDDURY,
% the Macaulay duration.
[Modified, Macaulay] = bnddury(Yield, CouponRate, Expiry,...
Maturity, Period, Basis);

% Convert the yield-to-maturity from a periodic to a
% continuous yield.
Yield  = Period .* log(1 + Yield./Period);

% Finally, convert the yield volatility to a price volatility via
% Hull's Equation 22.6 (page 514).
PriceSigma = Macaulay .* Yield .* YieldSigma;

% Finally, call Black's model.
PutPrices  = bkput(Strike, ZeroData, PriceSigma, BondData,...
Settle, Expiry, Period, Basis, [], [], Convention)

PutPrices =

    1.7838
    2.4071
```

When the strike price is the dirty price (`Convention = 0`), the call option value is \$1.78. When the strike price is the clean price (`Convention = 1`), the call option value is \$2.41.

References   [1] Hull, John C., *Options, Futures, and Other Derivatives*, Prentice Hall, 5th edition, 2003, pp. 287-288, 508-515.

See Also   bkcall

Purpose	Implied repo rates for bond future given price

Syntax
```
ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle,
Delivery, ConvFactor, CouponRate, Maturity)
ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle,
Delivery, ConvFactor, CouponRate, Maturity,
'ParameterName', 'ParameterValue ...)
```

Description
ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle,
Delivery, ConvFactor, CouponRate, Maturity) computes the
implied repo rate for a bond future given the price of a bond, the bond
properties, the price of the bond future, and the bond conversion factor.

ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle,
Delivery, ConvFactor, CouponRate, Maturity,
'ParameterName', 'ParameterValue ...) accepts optional inputs as
one or more comma-separated parameter/value pairs. 'ParameterName'
is the name of the parameter inside single quotes. ParameterValue is
the value corresponding to 'ParameterName'. Specify parameter-value
pairs in any order. Names are case-insensitive.

Input Arguments

Price

numBonds-by-1 vector of bond prices.

FutPrice

numBonds-by-1 vector of future prices

FutSettle

numBonds-by-1 vector of future settle dates.

Delivery

numBonds-by-1 vector of future delivery dates.

ConvFactor

numBonds-by-1 vector of bond conversion factors. For more information, see convfactor.

CouponRate

numBonds-by-1 vector of coupon rates in decimal form.

Maturity

numBonds-by-1 vector of coupon rates in decimal form.

Parameter–Value Pairs

Basis

Day-count basis. Possible values include

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0

EndMonthRule

End-of-month rule. Values are:

- 0 — Rule is not in effect for the bond.
- 1 — Rule is in effect for the bond. This means that a security that pays coupon interest on the last day of the month always makes payment on the last day of the month.

> **Default:** 1

Face

Face value of the bond. `Face` has no impact on key rate duration. This calling sequence is preserved for consistency.

> **Default:** 100

FirstCouponDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

> **Default:** If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

IssueDate

Issue date for a bond.

LastCouponDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

> **Default:** If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

Period

Number of coupons payments per year. Possible values include:

- 0
- 1
- 2
- 3
- 4
- 6
- 12

> **Default:** 2

ReinvestBasis

Day count basis for resinvestment rate.

> **Default:** Identical to `RepoBasis`.

ReinvestRate

Rate for reinvesting intermediate coupons from the bond.

> **Default:** Identical to `ImpRepo`.

RepoBasis

Day count basis for `ImpRepo`.

> **Default:** 2

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

Output Arguments

ImpRepo

Implied repo rate, or the repo rate that would produce the price input.

Definitions

`bndfutimprepo` computes the implied repo rate for a bond future given:

- Price of a bond
- Bond properties
- Price of the bond future
- Bond conversion factor

The default behavior is that the coupon reinvestment rate matches the repo rate. However, you can specify a separate reinvestment rate using optional inputs.

Examples

Compute the repro rate for a bond future:

```
bndfutimprepo(129,98,'9/21/2000','12/29/2000',1.3136,.0875,'8/15/2020')
```

This returns:

```
ans =
    0.0584
```

bndfutimprepo

References   Burghardt, G., T. Belton, M. Lane, and J. Papa, *The Treasury Bond Basis*, McGraw-Hill, 2005.

Krgin, Dragomir, *Handbook of Global Fixed Income Calculations*, John Wiley & Sons, 2002.

See Also   bndfutprice | convfactor

How To   • "Bond Futures" on page 7-12

Purpose   Price bond future given repo rates

Syntax   [FutPrice,AccrInt] = bndfutprice(RepoRate, Price, FutSettle,
                Delivery, ConvFactor, CouponRate, Maturity)
                FutPrice,AccrInt] = bndfutprice(RepoRate, FutPrice, FutSettle,
                Delivery, ConvFactor, CouponRate, Maturity,
                '*ParameterName*','*ParameterValue ...*)

Description [FutPrice,AccrInt] = bndfutprice(RepoRate, Price,
                FutSettle,
                Delivery, ConvFactor, CouponRate, Maturity) computes the price
                of a bond futures contract for one or more bonds given a repo rate, and
                bond properties, including the bond conversion factor.

                FutPrice,AccrInt] = bndfutprice(RepoRate, FutPrice,
                FutSettle,
                Delivery, ConvFactor, CouponRate, Maturity,
                '*ParameterName*','*ParameterValue ...*) accepts optional inputs as
                one or more comma-separated parameter/value pairs. '*ParameterName*'
                is the name of the parameter inside single quotes. *ParameterValue* is
                the value corresponding to '*ParameterName*'. Specify parameter-value
                pairs in any order. Names are case-insensitive.

**Input
Arguments**   **RepoRate**

                numBonds-by-1 vector of repo rates.

                Price

                numBonds-by-1 vector of bond prices

                FutSettle

                numBonds-by-1 vector of future settle dates.

                Delivery

numBonds-by-1 vector of future delivery dates.

ConvFactor

numBonds-by-1 vector of bond conversion factors. For more information, see convfactor.

CouponRate

numBonds-by-1 vector of coupon rates in decimal form.

Maturity

numBonds-by-1 vector of coupon rates in decimal form.

Parameter–Value Pairs

Basis

Day-count basis. Possible values include

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

    Default: 0

EndMonthRule

End-of-month rule. Values are:

- 0 — Rule is not in effect for the bond.

- 1 — Rule is in effect for the bond. This means that a security that pays coupon interest on the last day of the month always makes payment on the last day of the month.

    Default: 1

IssueDate

Issue date for a bond.

Face

Face value of the bond. Face has no impact on key rate duration. This calling sequence is preserved for consistency.

    Default: 100

FirstCouponDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

    Default: If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

LastCouponDate

bndfutprice

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

>**Default:** If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

Period

Number of coupons payments per year. Possible values include:

- 0
- 1
- 2
- 3
- 4
- 6
- 12

>**Default:** 2

ReinvestBasis

Day count basis for reinvestment rate.

>**Default:** Identical to `RepoBasis`.

ReinvestRate

Compounding convention for reinvestment rate.

>**Default:** Identical to `RepoRate`.

RepoBasis

Day count basis for `RepoRate`.

> **Default:** 2

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify `StartDate`, the effective start date is the `Settle` date.

Output Arguments

FutPrice

Quoted futures price, per $100 notional.

AccrInt

Accrued interest due at delivery date, per $100 notional.

Definitions

`bndfutprice` computes the price of a bond futures contract for one or more bonds, given a repo rate, and bond properties, including the bond conversion factor. The default behavior is that the coupon reinvestment rate matches the repo rate. However, you can specify a separate reinvestment rate using optional inputs.

Examples

Compute the price for a bond future:

```
bndfutprice(.064, 129, '9/21/2000','12/29/2000', 1.3136, .0875, '8/15/2020')
```

The returns:

```
ans =
   98.1516
```

References

Burghardt, G., T. Belton, M. Lane, and J. Papa, *The Treasury Bond Basis*, McGraw-Hill, 2005.

bndfutprice

Krgin, Dragomir, *Handbook of Global Fixed Income Calculations*, John Wiley & Sons, 2002.

See Also   bndfutimprepo | convfactor

How To   • "Bond Futures" on page 7-12

Purpose	Bootstrap interest-rate curve from market data

Class   @IRDataCurve

Syntax   Dcurve = IRDataCurve.bootstrap(Type, Settle, InstrumentTypes, Instruments)

Dcurve = IRDataCurve.bootstrap(Type, Settle, InstrumentTypes, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

Type	Type of interest-rate curve. Acceptable values are: discount, forward, or zero.
Settle	Scalar or column vector of settlement dates.
InstrumentTypes	N-by-1 cell array (where N is the number of instruments) indicating what kind of instrument is in the Instruments matrix. Acceptable values are deposit, futures, swap, and bond.
Instruments	N-by-3 data matrix for Instruments where the first column is Settle date, the second column is Maturity, and the third column is the market quote (dates must be MATLAB date numbers).

Note The market quote represents the following for each instrument:

- `deposit`: rate
- `futures`: price (e.g., 9628.54)
- `swap`: rate
- `bond`: clean price

Compounding   (Optional) Scalar that sets the compounding frequency per year for an `IRDataCurve` object:

- -1 = Continuous compounding
- 1 = Annual compounding
- 2 = Semiannual compounding (default)
- 3 = Compounding three times per year
- 4 = Quarterly compounding
- 6 = Bimonthly compounding
- 12 = Monthly compounding

Basis   (Optional) Day-count basis of the interest-rate curve. A scalar of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

bootstrap

InterpMethod	(Optional) Values are:
	• `'linear'` — Linear interpolation (default).
	• `'constant'` — Piecewise constant interpolation.
	• `'pchip'` — Piecewise cubic Hermite interpolation.
	• `'spline'` — Cubic spline interpolation.
IRBootstrapOptionsObj	(Optional) An `IRBootstrapOptions` object.

Instrument Parameters

For each bond `Instrument`, you can specify the following additional instrument parameters as parameter/value pairs. For example, `InstrumentBasis` distinguishes a bond instrument's `Basis` value from the curve's `Basis` value.

InstrumentCouponRate	(Optional) Decimal number indicating the annual percentage rate used to determine the coupons payable on an instrument.
InstrumentPeriod	(Optional) Coupons per year of the instrument. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.

InstrumentBasis   (Optional) Day-count basis of the instrument.
A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page
Glossary-1.

InstrumentEndMonthRule (Optional) End-of-month rule. A vector.
This rule applies only when Maturity is an
end-of-month date for a month having 30 or
fewer days. 0 = ignore rule, meaning that an
instrument's coupon payment date is always
the same numerical day of the month. 1 = set
rule on (default), meaning that an instrument's
coupon payment date is always the last actual
day of the month.

InstrumentIssueDate (Optional) Date when an instrument was issued.

InstrumentFirstCouponDate (Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

InstrumentLastCouponDate (Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

InstrumentFace (Optional) Face or par value. Default = 100.

Note When using Instrument parameter/value pairs, you can specify simple interest for an Instrument by specifying the InstrumentPeriod value as 0. If InstrumentBasis and InstrumentPeriod are not specified for an Instrument, the following default values are used:

- deposit instrument uses Basis as 2 (act/360) and Period is 0 (simple interest).

- futures instrument uses Basis as 2 (act/360) and Period is 4 (quarterly).

- swap instrument uses Basis as 2 (act/360) and Period is 2.

- bond instrument uses Basis as 0 (act/act) and Period is 2.

Description   Dcurve = IRDataCurve.bootstrap(Type, Settle, InstrumentTypes, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...) bootstraps an interest-rate curve from market data. The dates of the bootstrapped curve correspond to the maturity dates of the input instruments. You must enter the optional arguments for Basis, Compounding, Interpmethod, and IRBootstrapOptionsObj as parameter/value pairs.

Examples   **Use the `bootstrap` Method to Create an `IRDataCurve` Object**

In this bootstrapping example, InstrumentTypes, Instruments, and a Settle date are defined:

```
InstrumentTypes = {'Deposit';'Deposit';...
'Futures';'Futures';'Futures';'Futures';'Futures';'Futures';...
'Swap';'Swap';'Swap';'Swap';};

Instruments = [datenum('08/10/2007'),datenum('09/17/2007'),.0532000; ..
datenum('08/10/2007'),datenum('11/17/2007'),.0535866; ...
datenum('08/08/2007'),datenum('19-Dec-2007'),9485; ...
datenum('08/08/2007'),datenum('19-Mar-2008'),9502; ...
datenum('08/08/2007'),datenum('18-Jun-2008'),9509.5; ...
```

```
datenum('08/08/2007'),datenum('17-Sep-2008'),9509; ...
datenum('08/08/2007'),datenum('17-Dec-2008'),9505.5; ...
datenum('08/08/2007'),datenum('18-Mar-2009'),9501; ...
datenum('08/08/2007'),datenum('08/08/2014'),.0530; ...
datenum('08/08/2007'),datenum('08/08/2019'),.0551; ...
datenum('08/08/2007'),datenum('08/08/2027'),.0565; ...
datenum('08/08/2007'),datenum('08/08/2037'),.0566];

CurveSettle = datenum('08/10/2007');
```

Use the `bootstrap` method to create an `IRDataCurve` object.

```
bootModel = IRDataCurve.bootstrap('Forward', CurveSettle, ...
InstrumentTypes, Instruments,'InterpMethod','pchip')

bootModel =

 IRDataCurve

    Type: Forward
     Settle: 733264 (10-Aug-2007)
   Compounding: 2
   Basis: 0 (actual/actual)
  InterpMethod: pchip
   Dates: [12x1 double]
    Data: [12x1 double]
```

To create the plot for the bootstrapped market data:

```
PlottingDates = (datenum('08/11/2007'):30:CurveSettle+365*25)';
plot(PlottingDates,bootModel.getParYields(PlottingDates),'r')
set(gca,'ylim',[0 .06])
datetick
```
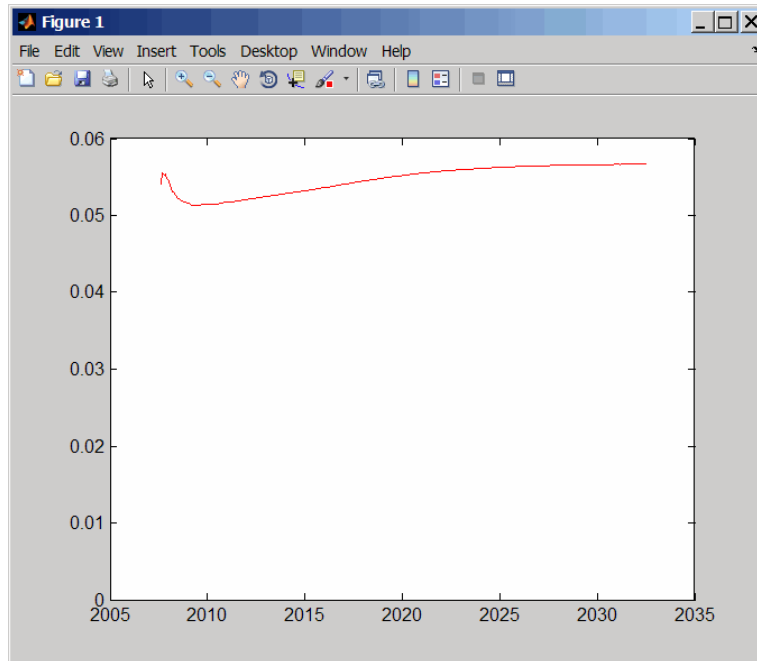
For an example of bootstrapping using instrument parameters support for prepending the word Instrument to the parameter field, see Related Examples.

Use the `bootstrap` Method to Create an `IRDataCurve` Object That Includes Bonds

In this bootstrapping example, `InstrumentTypes`, `Instruments`, and a `Settle` date are defined:

```
CurveSettle = datenum('8-Mar-2010');

InstrumentTypes = {'Deposit';'Deposit';'Deposit';'Deposit';...
    'Futures';'Futures';'Futures';'Futures';'Swap';'Swap';'Bond';'Bond

Instruments = [datenum('8-Mar-2010'),datenum('8-Apr-2010'),.003; ...
```

```
    datenum('8-Mar-2010'),datenum('8-Jun-2010'),.005; ...
    datenum('8-Mar-2010'),datenum('8-Sep-2010'),.007; ...
    datenum('8-Mar-2010'),datenum('8-Mar-2011'),.009; ...
    datenum('8-Mar-2010'),datenum('18-Jun-2011'),9840; ...
    datenum('8-Mar-2010'),datenum('17-Sep-2011'),9820; ...
    datenum('8-Mar-2010'),datenum('17-Dec-2011'),9810; ...
    datenum('8-Mar-2010'),datenum('18-Mar-2012'),9800; ...
    datenum('8-Mar-2010'),datenum('8-Mar-2015'),.025; ...
    datenum('8-Mar-2010'),datenum('8-Mar-2020'),.035; ...
    datenum('8-Mar-2010'),datenum('8-Mar-2030'),99; ...
    datenum('8-Mar-2010'),datenum('8-Mar-2040'),101];
```

When bonds are used,`InstrumentCouponRate` must be specified:

```
InstrumentCouponRate = [zeros(10,1);.045;.05];
```

Note, for parameters that are only applicable to bonds
(e.g.:`InstumentFirstCouponDate`, `InstrumentLastCouponDate`,
`InstumentIssueDate`,`InstumentFace`) the entries for non-bond
instruments (deposits and futures) are ignored.

Use the `bootstrap` method to create an `IRDataCurve` object.

```
bootModel = IRDataCurve.bootstrap('Forward', CurveSettle, ...
InstrumentTypes, Instruments,'InterpMethod','pchip',...
'InstrumentCouponRate',InstrumentCouponRate)

bootModel =

 IRDataCurve

    Type: Forward
     Settle: 734205 (08-Mar-2010)
   Compounding: 2
   Basis: 0 (actual/actual)
  InterpMethod: pchip
   Dates: [12x1 double]
    Data: [12x1 double]
```

To create the plot for the bootstrapped market data:

```
PlottingDates = daysadd(CurveSettle,30:30:30*360,1);
plot(PlottingDates,bootModel.getParYields(PlottingDates),'r')
set(gca,'ylim',[0 .06])
datetick
```
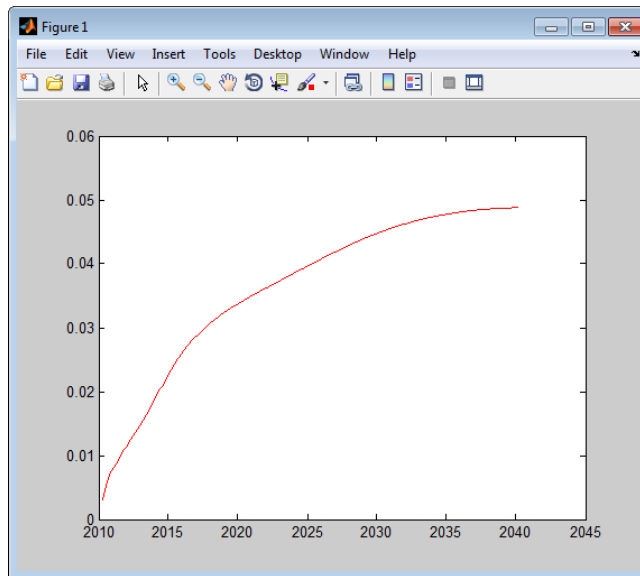

See Also "@IRDataCurve" on page A-7 **|** "@IRBootstrapOptions" on page A-2 **|**

Related Examples

• "IRDataCurve Bootstrapping Based on Market Instruments" on page 9-7

cbprice

Purpose	Price convertible bond

Syntax

```
[CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] =
cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ConvRatio,
NumSteps, IssueDate, Settle, Maturity, CouponRate)
[CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] =
cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ConvRatio,
NumSteps, IssueDate, Settle, Maturity, CouponRate,
Name,Value)
```

Description

```
[CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] =
cbprice(RiskFreeRate, StaticSpread, Sigma, Price,
ConvRatio,
NumSteps, IssueDate, Settle, Maturity, CouponRate) price a
```
convertible bond with a one-factor lattice method.

```
[CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] =
cbprice(RiskFreeRate, StaticSpread, Sigma, Price,
ConvRatio,
NumSteps, IssueDate, Settle, Maturity, CouponRate,
Name,Value) price a convertible bond with a one-factor lattice method
```
with additional options specified by one or more `Name,Value` pair
arguments.

Input Arguments

RiskFreeRate

Annual yield of the risk-free bond with the same maturity as the convertible, compounded continuously. Scalar value of risk-free rates is in decimal. (Recommended value is the yield of a risk-free bond with the same maturity as the convertible.)

StaticSpread

Scalar value of the constant spread to risk-free rate. Adding `StaticSpread` to the `RiskFreeRate` produces the issuer's yield, which reflects the credit risk.

Sigma

Scalar value of the annual volatility environment in decimal.

Price

Scalar value of the price of the asset at the settlement or valuation date.

ConvRatio

Scalar value of the number of assets convertible to one bond.

NumSteps

Scalar value of the number of steps within the binomial tree.

IssueDate

Scalar value of the issue date of the convertible bond.

Settle

Scalar value of the settlement date of the convertible bond.

Maturity

Scalar value of the maturity date of the convertible bond.

CouponRate

Scalar value of the coupon rate in decimal form or a `C`-by-`2` vector of dates and associated coupon rates.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (`' '`). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

'Basis'

Day-count basis of the bond. A vector of integers.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 0 (actual/actual)

'CallType'

Scalar value for the call type. Values are 0 for a call on cash price, or 1 for a call on clean price.

> **Default:** 0 (cash price)

'CallInfo'

Two-column matrix containing the call information. The first column is the call dates and the second column is the call prices for every $100 face of the bond. The call, in the amount of call prices, is activated *after* the corresponding call date.

Default: No call feature

'ConvInfo'

Two-column matrix containing convertible information. The first column is the convertible dates and the second column is whether the issue is convertible or not.

Default: Bond is always convertible

'DividendInfo'

Two-column matrix of dividend information. The first column is the ex-dividend date and the second column is the corresponding amount. Enter any amount known at any time; only the amounts that are within the lifespan of the option are used. If the DividendType is 2, DividendInfo is a 1-by-2 matrix where the first entry is the Settle date and the second entry is the continuous dividend yield.

Default: No dividend

'DividendType'

Scalar value for dividend type. Values are:

- 0 — Dollar dividend
- 1 — Dividend yield
- 2 — Continuous dividend yield

Default: 0 (Dollar dividend)

'EndMonthRule'

NINST-by-1 vector for end-of-month rule. Values are 1 (on, in effect) and 0 (off, not in effect).

Default: 1 (on, in effect)

'Period'

Scalar value for number of coupon payments. Values are:

- 1 — One coupon per year

- 2 — Semiannual

- 3 — Three times a year

- 4 — Quarterly

- 6 — Bimonthly compounding

- 12 — Monthly

> **Default:** 2 (Semiannual)

'PutInfo'

Two-column matrix containing put information. The first column is the put dates and the second column is the put prices for every $100 face of the bond. The put, in the amount of put prices, is activated *after* the corresponding put date.

> **Default:** No put feature

'PutType'

Scalar value for put type. Value are 0 for a put on cash price or 1 for a put on clean price.

> **Default:** 0 (put on cash price)

'TreeType'

Scalar value for tree type. Values are 0 for binomial lattice or 1 for trinomial lattice.

> **Default:** 0 (binomial lattice)

Output Arguments

CbMatrix

Matrix of CB prices in binomial format. Price of convertible is
CbMatrix(1,1).

UndMatrix

Matrix of stock prices in binomial format.

DebtMatrix

Matrix of CB debt component in binomial format.

EqtyMatrix

Matrix of CB equity component in binomial format.

Definitions **Convertible Bond**

A convertible bond (CB) is a debt instrument that you can convert into a
predetermined amount of the issuing company's equity at certain times
before the bond's maturity. In addition to standard bond features (for
example, maturity date, face value, coupon), a convertible bond often
has callable and puttable features.

Examples Perform a spread effect analysis of a 4% coupon convertible bond
callable at 110 at the end of the second year, maturing at par in 5 years,
with yield to maturity of 5%, and spread (of yield to maturity versus
5-year treasury) of 0, 50, 100, and 150 basis points. The underlying
stock pays no dividend.

```
RiskFreeRate = 0.05;
Sigma        = 0.3;
ConvRatio    = 1;
NumSteps     = 200;
IssueDate    = '2-Jan-2002';
Settle       = '2-Jan-2002';
Maturity     = '2-Jan-2007';
CouponRate   = 0.04;
```

```
Period      = 2;
Basis       = 1;
EndMonthRule = 1;
DividendType = 0;
DividendInfo = [];
CallInfo    = [datenum('2-Jan-2004') , 110];
CallType    = 1;
TreeType    = 1;
Spreads = 0:0.005:0.015;
Prices = 40:10:140;
stock = repmat(Prices',1,length(Spreads));

convprice = zeros(length(Prices),length(Spreads));

  for spreadidx = 1:length(Spreads)
      for priceidx = 1:length(Prices)
        [CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] = ...
            cbprice(RiskFreeRate, Spreads(spreadidx), Sigma, Prices(priceidx), ...
            ConvRatio, NumSteps, IssueDate, Settle, ...
            Maturity, CouponRate, Period, Basis, EndMonthRule, ...
            DividendType, DividendInfo, CallType, CallInfo, TreeType);

        convprice(priceidx,spreadidx) = CbMatrix(1,1);
      end
  end

plot(stock,convprice);
legend({'+0 bp'; '+50 bp'; '+100 bp'; '+150 bp'});
title ('Effect of Spread using Trinomial tree - 200 steps')
xlabel('Stock Price');
ylabel('Convertible Price');
text(50, 150, ['Coupon 4 semiannual,', sprintf('\n'), ...
'110 Call-on-clean after 2 years,' sprintf('\n'), ...
'maturing par in 5 years'],'fontweight','Bold')
```
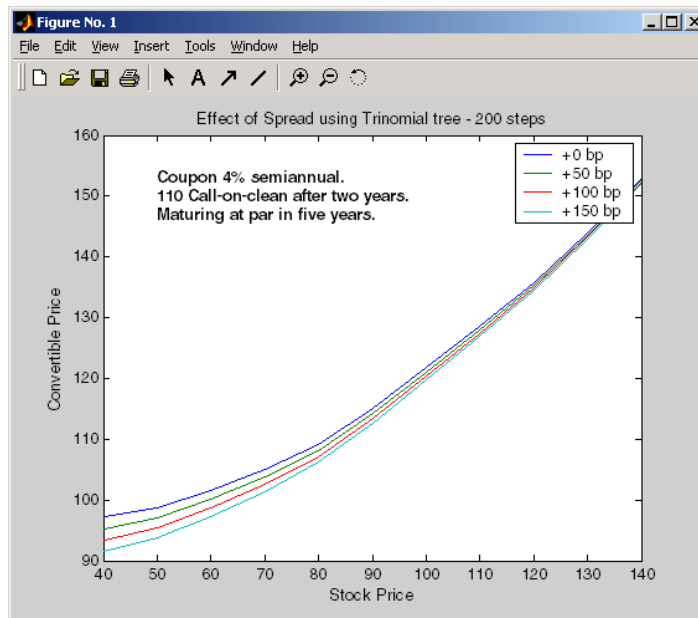
References   Andersen, L. and D. Buffum, "Calibration and implementation of convertible bonds models," Working paper, Banc of America Securities, 2003.

Ayache, E., P.A. Forsyth, and K.R. Vetzal, "Valuation of Convertible Bonds with Credit Risk," *Journal of Derivatives*, 11 (Fall 2003), 9-29.

Tsiveriotis, K. and C. Fernandes, "Valuing Convertible Bonds with Credit Risk," *Journal of Fixed Income* 8, 95-102, 1998

Zabolotnyuk, Yuriy, Jones, Robert A. and Veld, Chris H., "An Empirical Comparison of Convertible Bond Valuation Models," (October 15, 2009). Available at SSRN: `http://ssrn.com/abstract=994805`.

Tutorials   • "Convertible Bond Valuation" on page 7-10

cdsbootstrap

Purpose
Bootstrap default probability curve from credit default swap market quotes

Syntax
```
[ProbData, HazData] = cdsbootstrap(ZeroData, MarketData,
Settle)
[ProbData, HazData] = cdsbootstrap(ZeroData, MarketData,
Settle, Name,Value)
```

Description
`[ProbData, HazData] = cdsbootstrap(ZeroData, MarketData, Settle)` bootstraps the default probability curve using credit default swap (CDS) market quotes. The market quotes can be expressed as a list of maturity dates and corresponding CDS market spreads, or as a list of maturities and corresponding upfronts and standard spreads for standard CDS contracts. The estimation uses the standard model of the survival probability.

`[ProbData, HazData] = cdsbootstrap(ZeroData, MarketData, Settle, Name,Value)` bootstraps the default probability curve using CDS market quotes with additional options specified by one or more `Name,Value` pair arguments. The market quotes can be expressed as a list of maturity dates and corresponding CDS market spreads, or as a list of maturities and corresponding upfronts and standard spreads for standard CDS contracts. The estimation uses the standard model of the survival probability.

Input Arguments

ZeroData

M-by-2 vector of dates and zero rates or an `IRDataCurve` object of zero rates.

MarketData

N-by-2 matrix of dates and corresponding market spreads or N-by-3 matrix of dates, upfronts, and standard spreads of CDS contracts.

Settle

Settlement date is a serial date number or date string. This must be earlier than or equal to the dates in `MarketData`.

Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name,Value` arguments. `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

Note Any optional input of size N-by-1 is also acceptable as an array of size 1-by-N, or as a single value applicable to all contracts. Single values are internally expanded to an array of size N-by-1.

'Basis'

N-by-1 vector of day-count basis of the CDS:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

>   **Default:** 2 (actual/360)

'BusDayConvention'

String or N-by-1 cell array of strings of business day conventions.
Values are:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

>   **Default:** actual

'PayAccruedPremium'

N-by-1 vector of Boolean flags, True (default), if accrued premiums are
paid upon default, False otherwise.

>   **Default:** True

'Period'

N-by-1 vector of the number of premiums per year of the CDS. Allowed
values are 1, 2, 3, 4, 6, and 12.

>   **Default:** 4

'ProbDates'

P-by-1 vector of dates for ProbData.

Default: Column of dates in `MarketData`

'RecoveryRate'

N-by-1 vector of recovery rates, expressed as a decimal from `0` to `1`.

Default: `0.4`

'TimeStep'

Positive integer indicating the number of days to take as time step for the numerical integration.

Default: `10` (days)

'ZeroBasis'

Basis of the zero curve. Choices are identical to `Basis`.

Default: `0` (actual/actual)

'ZeroCompounding'

Compounding frequency of the zero curve. Allowed values are:

- `1` — Annual compounding
- `2` — Semiannual compounding
- `3` — Compounding three times per year
- `4` — Quarterly compounding
- `6` — Bimonthly compounding
- `12` — Monthly compounding
- `-1` — Continuous compounding

cdsbootstrap

Default: 2 (Semiannual compounding)

Output Arguments

ProbData

P-by-2 matrix with dates and corresponding cumulative default probability values. The dates match those in MarketData, unless the optional input parameter ProbDates is provided.

HazData

N-by-2 matrix with dates and corresponding hazard rate values for the standard survival probability model. The dates match those in MarketData.

Note A warning is displayed when non-monotone default probabilities (i.e., negative hazard rates) are found.

Examples

Use cdsbootstrap with market quotes for CDS contracts to generate ProbData and HazData values:

```
Settle = '17-Jul-2009';
Spread_Time = [1 2 3 5 7]';
Spread = [140 175 210 265 310]';
Market_Dates = daysadd(datenum(Settle),360*Spread_Time,1);
MarketData = [Market_Dates Spread];
Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
Zero_Dates = daysadd(datenum(Settle),360*Zero_Time,1);
```

```
ZeroData = [Zero_Dates Zero_Rate];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle)

ProbData =

  1.0e+005 *

    7.3434    0.0000
    7.3470    0.0000
    7.3507    0.0000
    7.3580    0.0000
    7.3653    0.0000

HazData =

  1.0e+005 *

    7.3434    0.0000
    7.3470    0.0000
    7.3507    0.0000
    7.3580    0.0000
    7.3653    0.0000
```

Algorithms   If the time to default is denoted by τ, the default probability curve, or function, *PD(t)*, and its complement, the survival function *Q(t)*, are given by:

$$PD(t) = P[\tau \le t] = 1 - P[\tau > t] = 1 - Q(t)$$

In the standard model, the survival probability is defined in terms of a piecewise constant hazard rate *h(t)*. For example, if *h(t)* =

λ_1, for $0 \le t \le t_1$

λ_2, for $t_1 < t \le t_2$

λ_3, for $t_2 < t$

cdsbootstrap

then the survival function is given by $Q(t) =$

$e^{-\lambda_1 t}$, for $0 \le t \le t_1$

$e^{-\lambda_1 t - \lambda_2 (t - t_1)}$, for $t_1 < t \le t_2$

$e^{-\lambda_1 t_1 - \lambda_2 (t_2 - t_1) - \lambda_3 (t - t_2)}$, for $t_2 < t$

Given n market dates $t_1,...,t_n$ and corresponding market CDS spreads $S_1,...,S_n$, cdsbootstrap calibrates the parameters $\lambda_1,...,\lambda_n$ and evaluates $PD(t)$ on the market dates, or an optional user-defined set of dates.

References Beumee, J., D. Brigo, D. Schiemert, and G. Stoyle. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research*, Global Special Report. April 7, 2009.

Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives* 8, 29-40.

O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps." *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.

See Also cdsspread | cdsprice | cdsrpv01"@IRDataCurve" on page A-7 |

Related Examples • "Credit Default Swap (CDS)" on page 8-2

Purpose   Price payer and receiver credit default swap options

Syntax   [Payer, Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, CDSMaturity, Strike, SpreadVol)
[Payer, Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, CDSMaturity, Strike, SpreadVol, Name,Value)

Description   [Payer, Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, CDSMaturity, Strike, SpreadVol) computes the price of payer and receiver credit default swap options.

[Payer, Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, CDSMaturity, Strike, SpreadVol, Name,Value) computes the price of payer and receiver credit default swap options with additional options specified by one or more Name,Value pair arguments.

Input Arguments

ZeroData

M-by-2 vector of dates and zero rates or an IRDataCurve object of zero rates.

ProbData

P-by-2 array of dates and default probabilities.

Settle

Settlement date is a serial date number or date string. Settle must be earlier than the maturity date.

OptionMaturity

N-by-1 vector of serial date numbers or date strings containing the option maturity dates.

CDSMaturity

N-by-1 vector of serial date numbers or date strings containing the CDS maturity dates.

Strike

N-by-1 vector of option strikes expressed in basis points.

SpreadVol

N-by-1 vector of annualized credit spread volatilities expressed as a positive decimal number.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Note Any optional input of size N-by-1 is also acceptable as an array of size 1-by-N, or as a single value applicable to all contracts. Single values are internally expanded to an array of size N-by-1.

'AdjustedForwardSpread'

N-by-1 vector of adjusted forward spreads (in basis points) to be used when pricing CDS index options.

> **Default:** unadjusted forward spread normally used for single-name CDS options

'Basis'

N-by-1 vector of contract day-count basis:

- 0 = actual/actual
- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 2 (actual/360)

'BusDayConvention'

String or N-by-1 cell array of strings of business day conventions. Values are:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

> **Default:** actual

'Knockout'

cdsoptprice

N-by-1 vector of Boolean flags. If the credit default swaptions is a knockout, the flag is True, otherwise it is False.

> **Default:** False

'PayAccruedPremium'

N-by-1 vector of Boolean flags. If accrued premiums are paid upon default, the flag is True, otherwise it is False.

> **Default:** True

'Period'

N-by-1 vector of the number of premiums per year of the CDS. Allowed values are 1, 2, 3, 4, 6, and 12.

> **Default:** 4

'RecoveryRate'

N-by-1 vector of recovery rates, expressed as a decimal from 0 to 1.

> **Default:** 0.4

'ZeroBasis'

Basis of the zero curve. Choices are identical to Basis.

> **Default:** 0 (actual/actual)

'ZeroCompounding'

Compounding frequency of the zero curve. Allowed values are:

- 1 — Annual compounding
- 2 — Semiannual compounding
- 3 — Compounding three times per year

- 4 — Quarterly compounding

- 6 — Bimonthly compounding

- 12 — Monthly compounding

- −1 — Continuous compounding

Note When ZeroData is an IRDataCurve object, the arguments ZeroCompounding and ZeroBasis are implicit in ZeroData and are redundant inside this function. In that case, specify these optional arguments when constructing the IRDataCurve object before calling this function.

Default: 2 (Semiannual compounding)

Output Arguments

Payer

N-by-1 vector of prices for payer swap options in Basis points.

Receiver

N-by-1 vector of prices for receiver swap options in Basis points.

Definitions

Credit Default Swap Option

A credit default swap (CDS) option, or credit default swaption, is a contract that provides the option holder with the right, but not the obligation, to enter into a credit default swap in the future. CDS options can either be payer swaptions or receiver swaptions. In a payer swaption, the option holder has the right to enter into a CDS in which they are paying premiums and in a receiver swaptions, the option holder is receiving premiums.

Examples

Obtain Payer and Receiver Values for a Credit Default Swap Option

Use cdsoptprice to generate Payer and Receiver values for a credit default swap option.

```
Settle = datenum('12-Jun-2012');
OptionMaturity = datenum('20-Sep-2012');
CDSMaturity = datenum('20-Sep-2017');
OptionStrike = 200;
SpreadVolatility = .4;

Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [.5 .75 1.5 1.7 1.9 2.2]'/100;
Zero_Dates = daysadd(Settle,360*Zero_Time,1);
ZeroData = [Zero_Dates Zero_Rate];

Market_Time = [1 2 3 5 7 10]';
Market_Rate = [100 120 145 220 245 270]';
Market_Dates = daysadd(Settle,360*Market_Time,1);
MarketData = [Market_Dates Market_Rate];

ProbData = cdsbootstrap(ZeroData, MarketData, Settle);

[Payer,Receiver] = cdsoptprice(ZeroData, ProbData, Settle,...
OptionMaturity, CDSMaturity, OptionStrike, SpreadVolatility)

Payer =

  223.5780


Receiver =

   22.7460
```

Algorithms   The payer and receiver credit default swap options are computed using the Black's model as described in O'Kane [1]:

$$V_{Pay(Knockout)} = RPV01(t,t_E,T)(F\Phi(d_1) - K\Phi(d_2))$$

$$V_{Rec(Knockout)} = RPV01(t,t_E,T)(K\Phi(-d_2) - F\Phi(-d_1))$$

$$d_1 = \frac{\ln\left(\dfrac{F}{K}\right) + \dfrac{1}{2}\sigma^2(t_E - t)}{\sigma\sqrt{t_E - t}}$$

$$d_2 = d_1 - \sigma\sqrt{t_E - t}$$

$$V_{Pay(Non-Knockout)} = V_{Pay(Knockout)} + FEP$$

$$V_{Pay(Non-Knockout)} = V_{Rec(Knockout)}$$

where

RPV01 is the risky present value of a basis point (see cdsrpv01).

Φ is the normal cumulative distribution function.

σ is the spread volatility.

t is the valuation date.

t_E is the option expiry date.

T is the CDS maturity date.

F is the forward spread (from option expiry to CDS maturity).

K is the strike spread.

FEP is the front end protection (from option initiation to option expiry).

References   [1] O'Kane, D., *Modelling Single-name and Multi-name Credit Derivatives*, Wiley, 2008, pp. 156–169.

cdsoptprice

See Also cdsbootstrap | cdsspread | cdsprice | cdsrpv01 "@IRDataCurve" on page A-7 **|**

Related Examples
- "Pricing a Single-Name CDS Option" on page 8-57
- "Pricing a CDS Index Option" on page 8-59

Concepts
- "Credit Default Swap Option" on page 8-57

Purpose   Determine price for credit default swap

Syntax
```
[Price, AccPrem, PaymentDates, PaymentTimes,
PaymentCF] = cdsprice(ZeroData, ProbData, Settle,
Maturity, ContractSpread)
[Price, AccPrem, PaymentDates, PaymentTimes,
PaymentCF] = cdsprice(ZeroData, ProbData,
Settle, Maturity, ContractSpread,
Name,Value)
```

Description   `[Price, AccPrem, PaymentDates, PaymentTimes, PaymentCF] = cdsprice(ZeroData, ProbData, Settle, Maturity, ContractSpread)` computes the price, or the mark-to-market value for CDS instruments.

`[Price, AccPrem, PaymentDates, PaymentTimes, PaymentCF] = cdsprice(ZeroData, ProbData, Settle, Maturity, ContractSpread, Name,Value)` computes the price, or the mark-to-market value for CDS instruments with additional options specified by one or more `Name,Value` pair arguments.

Input Arguments
ZeroData

M-by-2 vector of dates and zero rates or an `IRDataCurve` object of zero rates.

ProbData

P-by-2 array of dates and default probabilities.

Settle

Settlement date is a serial date number or date string. This must be earlier than or equal to the dates in `MarketData`.

Maturity

N-by-1 vector of serial date numbers or date strings containing the maturity dates.

ContractSpread

N-by-1 vector of contract spreads, expressed in basis points.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Note Any optional input of size N-by-1 is also acceptable as an array of size 1-by-N, or as a single value applicable to all contracts. Single values are internally expanded to an array of size N-by-1.

'Basis'

N-by-1 vector of day-count basis of the CDS:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

> **Default:** 2 (actual/360)

'BusDayConvention'

String or N-by-1 cell array of strings of business day conventions. Values are:

- actual
- follow
- modifiedfollow
- previous
- modifiedprevious

> **Default:** actual

'Notional'

N-by-1 vector of contract notional values. Use positive values for long positions and negative values for short positions.

> **Default:** 10MM

'PayAccruedPremium'

N-by-1 vector of Boolean flags. True, if accrued premiums are paid upon default, False otherwise.

> **Default:** True

cdsprice

'Period'

N-by-1 vector of number of premiums per year of the CDS. Allowed values are 1, 2, 3, 4, 6, and 12.

> **Default:** 4

'RecoveryRate'

N-by-1 vector of recovery rates, expressed as a decimal from 0 to 1.

> **Default:** 0.4

'TimeStep'

Positive integer indicating the number of days to take as time step for the numerical integration.

> **Default:** 10 (days)

'ZeroBasis'

Basis of the zero curve, where the choices are identical to Basis.

> **Default:** 0 (actual/actual)

'ZeroCompounding'

Compounding frequency of the zero curve. Allowed values are:

- 1 — Annual compounding
- 2 — Semiannual compounding
- 3 — Compounding three times per year
- 4 — Quarterly compounding
- 6 — Bimonthly compounding
- 12 — Monthly compounding
- -1 — Continuous compounding

> **Note** When ZeroData is an IRDataCurve object, the arguments
> ZeroCompounding and ZeroBasis are implicit in ZeroData and are
> redundant inside this function. In that case, specify these optional
> arguments when constructing the IRDataCurve object before calling
> this function.

Default: 2 (Semiannual compounding)

Output Arguments

Price

N-by-1 vector of CDS clean prices.

AccPrem

N-by-1 vector of accrued premiums.

PaymentDates

N-by-numCF matrix of payment dates.

PaymentTimes

N-by-numCF matrix of accrual fractions.

PaymentCF

N-by-numCF matrix of payments.

Definitions

CDS Price

The price or mark-to-market (MtM) value of an existing CDS contract is
computed using the following formula:

```
CDS price = Notional * (Current Spread - Contract Spread)
* RPV01
```

Current Spread is the current breakeven spread for a similar contract,
according to current market conditions. RPV01 is the 'risky present

cdsprice

value of a basis point,' the present value of the premium payments, taking into consideration the default probability. This formula assumes a long position, and the right side is multiplied by -1 for short positions.

Examples   Use cdsprice to compute the clean price for a CDS contract:

```
Settle = '17-Jul-2009';
Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
Zero_Dates = daysadd(Settle,360*Zero_Time,1);
ZeroData = [Zero_Dates Zero_Rate];

ProbData = [daysadd(datenum(Settle),360,1), 0.0247];
Maturity = '20-Sep-2010';
ContractSpread = 135;

[Price,AccPrem] = cdsprice(ZeroData,ProbData,Settle,Maturity,ContractSpread)

Price =

   1.5461e+04


AccPrem =

      10500
```

Algorithms   The premium leg is computed as the product of a spread S and the risky present value of a basis point (RPV01). The RPV01 is given by:

$$RPV01 = \sum_{j=1}^{N} Z(t_j)\Delta(t_{j-1},t_j,B)Q(t_j)$$

when no accrued premiums are paid upon default, and it can be approximated by

$$RPV01 \approx \frac{1}{2} \sum_{j=1}^{N} Z(t_j)\Delta(t_{j-1},t_j,B)(Q(t_{j-1}) + Q(t_j))$$

when accrued premiums are paid upon default. Here, $t_0 = 0$ is the valuation date, and $t_1,...,t_n = T$ are the premium payment dates over the life of the contract, T is the maturity of the contract, $Z(t)$ is the discount factor for a payment received at time t, and $\Delta(t_{j-1}, t_j, B)$ is a day count between dates t_{j-1} and t_j corresponding to a basis B.

The protection leg of a CDS contract is given by the following formula:

$$ProtectionLeg = \int_0^T Z(\tau)(1-R)dPD(\tau)$$

$$\approx (1-R)\sum_{i=1}^{M} Z(\tau_i)(PD(\tau_i) - PD(\tau_{i-1}))$$

$$= (1-R)\sum_{i=1}^{M} Z(\tau_i)(Q(\tau_{i-1}) - Q(\tau_i))$$

where the integral is approximated with a finite sum over the discretization $\tau_0 = 0, \tau_1,...,\tau_M = T$.

If the spread of an existing CDS contract is S_C, and the current breakeven spread for a comparable contract is S_0, the current price, or mark-to-market value of the contract is given by:

MtM = Notional $(S_0 - S_C)$RPV01

This assumes a long position from the protection standpoint (protection was bought). For short positions, the sign is reversed.

References
Beumee, J., D. Brigo, D. Schiemert, and G. Stoyle. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research*, Global Special Report. April 7, 2009.

cdsprice

Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives* 8, 29-40.

O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps." *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.

See Also   `cdsbootstrap` | `cdsspread` | `cdsoptprice`"@IRDataCurve" on page A-7 |

Related Examples

- "Credit Default Swap (CDS)" on page 8-2

Purpose	Compute risky present value of a basis point for credit default swap
Syntax	RPV01 = cdsrpv01(ZeroData,ProbData,Settle, Maturity) RPV01 = cdsrpv01(___ ,Name,Value) [RPV01,PaymentDates,PaymentTimes] = cdsrpv01(ZeroData,ProbData,Settle, Maturity) [RPV01,PaymentDates,PaymentTimes] = cdsrpv01(___ ,Name,Value)
Description	RPV01 = cdsrpv01(ZeroData,ProbData,Settle, Maturity) computes the risky present value of a basis point (RPV01) for a credit default swap (CDS). RPV01 = cdsrpv01(___ ,Name,Value) computes the risky present value of a basis point (RPV01) for a credit default swap (CDS) using optional name-value pair arguments. [RPV01,PaymentDates,PaymentTimes] = cdsrpv01(ZeroData,ProbData,Settle, Maturity) computes the risky present value of a basis point (RPV01), PaymentDates, and PaymentTimes for a credit default swap (CDS). [RPV01,PaymentDates,PaymentTimes] = cdsrpv01(___ ,Name,Value) computes the risky present value of a basis point (RPV01), PaymentDates, and PaymentTimes for a credit default swap (CDS) using optional name-value pair arguments.
Input Arguments	**ZeroData - Dates and zero rates** object from IRDataCurve or vector of dates and zero rates Dates and zero rates, specified by an M-by-2 vector of dates and zero rates or the object IRDataCurve for zero rates.

Data Types
struct | double

ProbData - Dates and default probabilities
vector of dates and default probabilities

Dates and default probabilities, specified by a P-by-2 array.

Data Types
double

Settle - Settlement date
string or cell array of dates or numeric dates

Settlement date, specified by a serial date number or date string. This must be earlier than or equal to the dates in Maturity.

Data Types
char | cell | double

Maturity - CDS maturity date
string or cell array of dates or numeric dates

CDS maturity date, specified by an N-by-1 vector of serial date numbers or date strings containing the maturity dates. The CDS premium payment dates occur at regular intervals, and the last payment occurs on these maturity dates.

Data Types
char | cell | double

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as Name1,Value1,...,NameN,ValueN.

Example: RPV01 =
cdsrpv01(ZeroData,ProbData,Settle,Maturity,'Period',1,'StartDate','20-
'BusDayConvention',actual,'CleanRPV01',true,'PayAccruedPremium',true,

'Period' - Number of premium payments per year
4 (default) | positive integer from the set [1,2,3,4,6,12] | vector of
positive integers from the set [1,2,3,4,6,12]

Number of premium payments per year, specified by an N-by-1 vector.
Values are 1, 2, 3, 4, 6, and 12.

Data Types
double

'StartDate' - Dates the CDS premium leg starts
Settle date (default) | string | cell array

Dates when the CDS premium leg actually starts, specified by an N-by-1
vector of serial date numbers or date strings. Must be on or between the
Settle and Maturity dates. For a forward-starting CDS, specify this
date as a future date after Settle.

Data Types
double

'Basis' - Day-count basis of contract
2 (actual/360) (default) | positive integers of the set [1...13] | vector
of positive integers of the set [1...13]

Day-count basis of the contract, specified as a positive integer using
a NINST-by-1 vector.

- 0 = actual/actual

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

11

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ISMA)

- 9 = actual/360 (ISMA)

- 10 = actual/365 (ISMA)

- 11 = 30/360E (ISMA)

- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
double

'BusDayConvention' - Business day conventions
actual (default) | string | cell array

Business day conventions, specified by a string or N-by-1 cell array of strings of business day conventions. Values are:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

Data Types
char | cell

'CleanRPV01' - Flag for premium accrual
true (default) | Boolean flag with value true or false

Flag for premium accrual, specified as an N-by-1 vector of Boolean flags, which is `true` if the premium accrued at `StartDate` is excluded in the RPV01, and `false` otherwise.

Data Types
`logical`

'PayAccruedPremium' - Flag for accrued premium payment
`true` (default) | Boolean flag with value `true` or `false`

Flag for accrued premium payment, specified as a N-by-1 vector of Boolean flags, `true` if accrued premiums are paid upon default, `false` otherwise.

Data Types
`logical`

'ZeroCompounding' - Compounding frequency of zero curve
2 semiannual compounding (default) | integer with acceptable value [1,2,3,4,6,12, −1]

Compounding frequency of the zero curve, specified with integer values:

- 1 — Annual compounding

- 2 — Semiannual compounding

- 3 — Compounding three times per year

- 4 — Quarterly compounding

- 6 — Bimonthly compounding

- 12 — Monthly compounding

- −1 — Continuous compounding

> **Note** When ZeroData is an IRDataCurve object, the arguments
> ZeroCompounding and ZeroBasis are implicit in ZeroData and are
> redundant inside this function. In that case, specify these optional
> arguments when constructing the IRDataCurve object before calling
> this function.

Data Types
double

'ZeroBasis' - Basis of zero curve
0 (actual/actual) (default) | positive integers of the set [1...13] |
vector of positive integers of the set [1...13]

Basis of the zero curve, specified as a positive integer using a NINST-by-1
vector.

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (PSA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ISMA)
- 9 = actual/360 (ISMA)
- 10 = actual/365 (ISMA)
- 11 = 30/360E (ISMA)
- 12 = actual/365 (ISDA)

- 13 = BUS/252

Data Types
double

Output Arguments

RPV01 - RPV01 value

scalar | vector

RPV01 value, returned as an N-by-1 vector.

PaymentDates - Payment dates

scalar | vector

Payment dates, returned as an N-by-numCF matrix of dates.

PaymentTimes - Payment times

scalar | vector

Payment times, returned as an N-by-numCF matrix of accrual fractions.

Definitions

RPV01

RPV01, associated with a CDS, is the value of a stream of 1-basis-point premiums according to the payment structure of the CDS contract, and taking into consideration the default probability over time.

For more information, see [3] and [4] for details.

Examples

Calculate the RPV01 Value for a CDS

Calculate the RPV01 value given the following specification for a CDS.

```
Settle = '17-Jul-2009';
Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
Zero_Dates = daysadd(Settle,360*Zero_Time,1);
ZeroData = [Zero_Dates Zero_Rate];
ProbData = [daysadd(datenum(Settle),360,1), 0.0247];
Maturity = '20-Sep-2010';
```

cdsrpv01

```
RPV01 = cdsrpv01(ZeroData,ProbData,Settle,Maturity)

RPV01 =

    1.1651
```

References

[1] Beumee, J., D. Brigo, D. Schiemert, and G. Stoyle. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research*, Global Special Report. April 7, 2009.

[2] Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives* 8, 29–40.

[3] O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps," *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.

[4] O'Kane, D., *Modelling Single-name and Multi-name Credit Derivatives*, Wiley Finance, 2008.

See Also	cdsbootstrap | cdsspread | cdsprice | cdsoptprice"@IRDataCurve" on page A-7 **|**
Related Examples	• "Pricing a CDS Index Option" on page 8-59
Concepts	• **basis** on page Glossary-1
External Web Sites	• www.cdsmodel.com

Purpose	Determine spread of credit default swap

Syntax

```
[Spread, PaymentDates, PaymentTimes] = cdsspread(ZeroData,
ProbData, Settle, Maturity)
[Spread, PaymentDates, PaymentTimes] = cdsspread(ZeroData,
 ProbData, Settle, Maturity,
Name,Value)
```

Description

[Spread, PaymentDates, PaymentTimes] = cdsspread(ZeroData, ProbData, Settle, Maturity) computes the spread of the CDS.

[Spread, PaymentDates, PaymentTimes] = cdsspread(ZeroData, ProbData, Settle, Maturity, Name,Value) computes the spread of the CDS with additional options specified by one or more Name,Value pair arguments.

Input Arguments

ZeroData

M-by-2 vector of dates and zero rates or IRCurve of zero rates.

ProbData

P-by-2 array of dates and default probabilities.

Settle

Settlement date is a serial date number or date string. This must be earlier than or equal to the dates in MarketData.

Maturity

N-by-1 vector of serial date numbers or date strings containing the maturity dates.

Name-Value Pair Arguments

Specify optional comma-separated pairs of Name,Value arguments. Name is the argument name and Value is the corresponding value. Name must appear inside single quotes (' '). You can

specify several name and value pair arguments in any order as
Name1,Value1,...,NameN,ValueN.

Note Any optional input of size N-by-1 is also acceptable as an array of
size 1-by-N, or as a single value applicable to all contracts. Single values
are internally expanded to an array of size N-by-1.

'Basis'

N-by-1 vector of day-count basis of the CDS:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Default: 2 (actual/360)

'BusDayConvention'

String or N-by-1 cell array of strings of business day conventions. Values are:

- actual

- follow

- modifiedfollow

- previous

- modifiedprevious

  Default: actual

'PayAccruedPremium'

N-by-1 vector of Boolean flags, True, if accrued premiums are paid upon default, False otherwise.

  Default: True

'Period'

N-by-1 vector of number of premiums per year of the CDS. Allowed values are 1, 2, 3, 4, 6, and 12.

  Default: 4

'RecoveryRate'

N-by-1 vector of recovery rates, expressed as a decimal from 0 to 1.

  Default: 0.4

'TimeStep'

Positive integer indicating the number of days to take as time step for the numerical integration.

Default: 10 (days)

'ZeroBasis'

Basis of the zero curve, where the choices are identical to `Basis`.

Default: 0 (actual/actual)

'ZeroCompounding'

Compounding frequency of the zero curve. Allowed values are:

- 1 — Annual compounding

- 2 — Semiannual compounding

- 3 — Compounding three times per year

- 4 — Quarterly compounding

- 6 — Bimonthly compounding

- 12 — Monthly compounding

- -1 — Continuous compounding

Note When `ZeroData` is an `IRCurve` object, the arguments `ZeroCompounding` and `ZeroBasis` are implicit in `ZeroData` and are redundant inside this function. In that case, specify these optional arguments when constructing the `IRCurve` object before calling this function.

Default: 2 (semiannual compounding)

Output Arguments

Spread

N-by-1 vector of spreads (in basis points).

PaymentDates

N-by-numCF matrix of payment dates.

PaymentTimes

N-by-numCF matrix of accrual fractions.

Definitions   **CDS Spread**

The market, or breakeven, spread value of a CDS can be computed by equating the value of the protection leg with the value of the premium leg:

Market Spread * RPV01 = Value of Protection Leg

The left side corresponds to the value of the premium leg, and this has been decomposed as the product of the market or breakeven spread times the RPV01 or 'risky present value of a basis point' of the contract. The latter is the present value of the premium payments, taking into consideration the default probability. The Market Spread can be computed as the ratio of the value of the protection leg, to the RPV01 of the contract. cdsspread returns the resulting spread in basis points.

Examples   Use cdsspread to compute the clean price for a CDS contract:

```
Settle = '17-Jul-2009';
Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
Zero_Dates = daysadd(Settle,360*Zero_Time,1);
ZeroData = [Zero_Dates Zero_Rate];
ProbData = [daysadd(datenum(Settle),360,1), 0.0247];
Maturity = '20-Sep-2010';

Spread = cdsspread(ZeroData,ProbData,Settle,Maturity)

Spread =

  148.2705
```

Algorithms   The premium leg is computed as the product of a spread S and the risky present value of a basis point (RPV01). The RPV01 is given by:

$$RPV01 = \sum_{j=1}^{N} Z(t_j)\Delta(t_{j-1}, t_j, B)Q(t_j)$$

when no accrued premium are paid upon default, and it can be approximated by

$$RPV01 \approx \frac{1}{2}\sum_{j=1}^{N} Z(t_j)\Delta(t_{j-1}, t_j, B)(Q(t_{j-1}) + Q(t_j))$$

when accrued premiums are paid upon default. Here, $t_0 = 0$ is the valuation date, and $t_1, ..., t_n = T$ are the premium payment dates over the life of the contract, T is the maturity of the contract, $Z(t)$ is the discount factor for a payment received at time t, and $\Delta(t_{j-1}, t_j, B)$ is a day count between dates t_{j-1} and t_j corresponding to a basis B.

The protection leg of a CDS contract is given by the following formula:

$$ProtectionLeg = \int_{0}^{T} Z(\tau)(1-R)dPD(\tau)$$

$$\approx (1-R)\sum_{i=1}^{M} Z(\tau_i)(PD(\tau_i) - PD(\tau_{i-1}))$$

$$= (1-R)\sum_{i=1}^{M} Z(\tau_i)(Q(\tau_{i-1}) - Q(\tau_i))$$

where the integral is approximated with a finite sum over the discretization $\tau_0 = 0, \tau_1, ..., \tau_M = T$.

A breakeven spread S_0 makes the value of the premium and protection legs equal. It follows that:

$$S_0 = \frac{ProtectionLeg}{RPV01}$$

References   Beumee, J., D. Brigo, D. Schiemert, and G. Stoyle. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research*, Global Special Report. April 7, 2009.

Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives* 8, 29-40.

O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps." *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.

See Also | cdsprice | cdsbootstrap

Tutorials   • "Credit Default Swap (CDS)" on page 8-2

cmosched

Purpose	Generate principal balance schedule for planned amortization class (PAC) or targeted amortization class (TAC) bond
Syntax	`[BalanceSchedule, InitialBalance] = cmosched(Principal, Coupon, OriginalTerm, TermRemaining, PrepaySpeed)` `[BalanceSchedule, InitialBalance] = cmosched(Principal, Coupon, OriginalTerm, TermRemaining, PrepaySpeed, TranchePrincipal)`
Description	`[BalanceSchedule, InitialBalance] = cmosched(Principal, Coupon, OriginalTerm, TermRemaining, PrepaySpeed)` generates a principal balance schedule for planned amortization class (PAC) bonds using two bands of Public Securities Association Prepayment Model (PSA) speeds or targeted amortization class (TAC) bonds using a single PSA speed. `[BalanceSchedule, InitialBalance] = cmosched(Principal, Coupon, OriginalTerm, TermRemaining, PrepaySpeed, TranchePrincipal)` with a specified tranche principal generates a principal balance schedule for planned amortization class (PAC) bonds using two bands of PSA speeds or targeted amortization class (TAC) bonds using a single PSA speed.

Input Arguments

Principal

Principal of the underlying mortgage pool.

Coupon

Coupon of the underlying mortgage pool.

OriginalTerm

Original term in months of the underlying mortgage pool.

TermRemaining

Terms remaining in months of the underlying mortgage pool.

PrepaySpeed

PSA speed. For a PAC, the speed is a 1-by-2 matrix where the first element is the lower band and the second element is the upper band. For a TAC, the speed is a scalar.

TranchePrincipal

(Optional) Principal of the scheduled tranche. If it is unspecified or empty [ ], the principal of the scheduled tranche is assumed to be the sum of the payment schedule calculated from the PSA prepayment speeds.

Output Arguments

BalanceSchedule

Matrix of size 1-by-NUMTERMS, where NUMTERMS is the number of terms remaining. Each column contains the scheduled principal balance for the time period corr-by-esponding to the column number.

InitialBalance

Scalar containing the initial principal balance of the scheduled tranche.

Definitions

Planned Amortization Class (PAC) Bond

PAC bonds are a type of CMO bond. They are designed to largely eliminate prepayment risk for investors. They do this by transferring essentially all prepayment risk to other bonds in the CMO that are called support bonds.

Targeted Amortization Class (TAC) Bond

TAC bonds are analogous to PAC bonds, but are structured differently. TAC bonds offer one-sided protection, shielding investors from high prepayment rates up to a specified PSA and do not protect against low prepayment rates.

Examples

Calculate the Principal Balance Schedule for a CMO PAC Bond

Define the mortgage pool under consideration and generate a principal balance schedule for planned amortization class (PAC) bonds using two bands of PSA speeds.

Calculate PAC bonds using cmosched.

```
Principal = 128687000;
GrossRate = 0.0648;
OriginalTerm = 360;
TermRemaining = 325;
PrepaySpeed = [300 525];
PacPrincipal = 100250000;

[BalanceSchedule, InitialBalance] ...
= cmosched(Principal, GrossRate, OriginalTerm, TermRemaining, ...
PrepaySpeed, PacPrincipal);

BalanceSchedule =

  1.0e+07 *

  Columns 1 through 10

    9.7996    9.5780    9.3602    9.1461    8.9357    8.7289    8.5257    8.3259    8.1296    7.9366
  Columns 11 through 20

    7.7469    7.5605    7.3773    7.1972    7.0202    6.8463    6.6754    6.5073    6.3422    6.1799

  Columns 21 through 30

    6.0204    5.8637    5.7096    5.5582    5.4094    5.2632    5.1194    4.9782    4.8394    4.7030

  Columns 31 through 40

    4.5689    4.4372    4.3077    4.1804    4.0554    3.9325    3.8118    3.6931    3.5765    3.4619
```

```
Columns 41 through 50

  3.3494    3.2406    3.1353    3.0334    2.9348    2.8394    2.7470    2.6576    2.5711    2.4873

Columns 51 through 60

  2.4063    2.3279    2.2520    2.1786    2.1075    2.0387    1.9722    1.9078    1.8455    1.7852

Columns 61 through 70

  1.7268    1.6703    1.6157    1.5628    1.5117    1.4622    1.4142    1.3679    1.3231    1.2797

Columns 71 through 80

  1.2377    1.1970    1.1577    1.1197    1.0829    1.0473    1.0129    0.9795    0.9473    0.9161

Columns 81 through 90

  0.8859    0.8567    0.8285    0.8011    0.7747    0.7491    0.7244    0.7004    0.6773    0.6549

Columns 91 through 100

  0.6332    0.6122    0.5920    0.5723    0.5534    0.5350    0.5172    0.5001    0.4835    0.4674

Columns 101 through 110

  0.4518    0.4368    0.4223    0.4082    0.3946    0.3814    0.3687    0.3564    0.3445    0.3330

Columns 111 through 120

  0.3219    0.3111    0.3007    0.2906    0.2809    0.2715    0.2623    0.2535    0.2450    0.2368

Columns 121 through 130

  0.2288    0.2211    0.2137    0.2065    0.1995    0.1928    0.1863    0.1800    0.1739    0.1680

Columns 131 through 140
```

```
   0.1623     0.1568     0.1515     0.1464     0.1414     0.1366     0.1319     0.1275     0.1231     0.1189

Columns 141 through 150

   0.1149     0.1109     0.1072     0.1035     0.0999     0.0965     0.0932     0.0900     0.0869     0.0839

Columns 151 through 160

   0.0811     0.0783     0.0756     0.0730     0.0704     0.0680     0.0657     0.0634     0.0612     0.0591

Columns 161 through 170

   0.0570     0.0550     0.0531     0.0513     0.0495     0.0478     0.0461     0.0445     0.0429     0.0414

Columns 171 through 180

   0.0400     0.0386     0.0372     0.0359     0.0346     0.0334     0.0322     0.0311     0.0300     0.0289

Columns 181 through 190

   0.0279     0.0269     0.0260     0.0250     0.0241     0.0233     0.0224     0.0216     0.0209     0.0201

Columns 191 through 200

   0.0194     0.0187     0.0180     0.0174     0.0167     0.0161     0.0155     0.0150     0.0144     0.0139

Columns 201 through 210

   0.0134     0.0129     0.0124     0.0120     0.0115     0.0111     0.0107     0.0103     0.0099     0.0096

Columns 211 through 220

   0.0092     0.0089     0.0085     0.0082     0.0079     0.0076     0.0073     0.0070     0.0068     0.0065

Columns 221 through 230
```

```
0.0063    0.0060    0.0058    0.0056    0.0054    0.0052    0.0050    0.0048    0.0046    0.0044

Columns 231 through 240

0.0042    0.0041    0.0039    0.0037    0.0036    0.0035    0.0033    0.0032    0.0031    0.0029

Columns 241 through 250

0.0028    0.0027    0.0026    0.0025    0.0024    0.0023    0.0022    0.0021    0.0020    0.0019

Columns 251 through 260

0.0018    0.0018    0.0017    0.0016    0.0016    0.0015    0.0014    0.0014    0.0013    0.0012

Columns 261 through 270

0.0012    0.0011    0.0011    0.0010    0.0010    0.0009    0.0009    0.0009    0.0008    0.0008

Columns 271 through 280

0.0007    0.0007    0.0007    0.0006    0.0006    0.0006    0.0005    0.0005    0.0005    0.0005

Columns 281 through 290

0.0004    0.0004    0.0004    0.0004    0.0004    0.0003    0.0003    0.0003    0.0003    0.0003

Columns 291 through 300

0.0002    0.0002    0.0002    0.0002    0.0002    0.0002    0.0002    0.0002    0.0001    0.0001

Columns 301 through 310

0.0001    0.0001    0.0001    0.0001    0.0001    0.0001    0.0001    0.0001    0.0001    0.0000

Columns 311 through 320

0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000         0         0
```

cmosched

```
    Columns 321 through 325

         0         0         0         0         0


InitialBalance =

   100250000
```

References   Hayre, Lakhbir, ed., *Salomon Smith Barney Guide to Mortgage-Backed and Asset-Backed Securities*, John Wiley and Sons, New York, 2001.

Lyuu, Yuh-Dah, *Financial Engineering and Computation*, Cambridge University Press, 2004.

See Also | cmoschedcf

Tutorials   • "Using Collateralized Mortgage Obligations (CMOs)" on page 5-51

How To   • "Create PAC and Sequential CMO" on page 5-66

Purpose	Generate cash flows for scheduled collateralized mortgage obligation (CMO) using PAC or TAC model

Syntax
```
[Balances, Principal, Interest] =
cmoschedcf(PrincipalPayments, TranchePrincipals,
TrancheCoupons, BalanceSchedule)
```

Description [Balances, Principal, Interest] =
cmoschedcf(PrincipalPayments, TranchePrincipals,
TrancheCoupons, BalanceSchedule) generate cash flows for a
scheduled CMO such as the planned amortization class (PAC) or
targeted amortization class (TAC), given the underlying mortgage
pool payments (or payments from another CMO tranche). The output
Balance, Principal, and Interest from this function can be used as
input into cmoseqcf to further divide the PAC, TAC, or support tranche
into sequential tranches.

Input Arguments

PrincipalPayments

Matrix of size 1-by-NUMTERMS, where NUMTERMS is the number of terms
remaining. Each column contains the underlying principal payment for
the time period corresponding to the row number. Calculate underlying
principal payments using mbscfamounts or mbspassthrough. The
underlying principal payments can also be outputs from other CMO
cash flow functions.

TranchePrincipals

Matrix of size 2-by-1 specifying the initial principal for the scheduled
and the support tranche.

TrancheCoupons

Matrix of size 2-by-1 specifying the coupons for the schedule tranche
and the support tranche. The weighted average coupon for the CMO
should not exceed the coupon of the underlying mortgage.

BalanceSchedule

cmoschedcf

Matrix of size 1 -by-NUMTERMS, where NUMTERMS is the number of terms remaining. Each element represents the targeted balance schedule for the time period corresponding to that column.

Output Arguments

Balance

Matrix of size 2-by-NUMTERMS, where NUMTERMS is the number of terms remaining. The first row is the principal balances of the scheduled tranche, and the second row is the principal balances of the support tranche at the time period corresponding to the column.

Principal

Matrix of size 2-by-NUMTERMS, where NUMTERMS is the number of terms remaining. The first row is the principal payments of the scheduled tranche, and the second row is the principal payments of the support tranche at the time period corresponding to the column.

Interest

Matrix of size 2-by-NUMTERMS, where NUMTERMS is the number of terms remaining. The first row is the interest payments of the schedule tranche, and the second row is the interest payments of the support tranche at the time period corresponding to the column.

Definitions

Planned Amortization Class (PAC) Tranches

In a PAC CMO, there is a main tranche, known as the schedule tranche, and a support tranche. The main purpose of a schedule tranche is to give investors in the PAC tranche a more certain cash flow.

Targeted Amortization Class (TAC) Tranches

TACs are like PACs, but principal payment is specified for only one prepayment rate. If prepayment rates are higher or lower, then the principal payment to TAC holders will be higher or lower accordingly.

Schedule and Support Tranche

The main purpose of a PAC tranche is to give investors in the PAC tranche a more certain cash flow. The PAC tranche receives priority

for receiving payments of principal and interest that gives investors in the PAC tranche a steadier income. If prepayments differ from what was expected, then the support tranche gets the variable portion of the payments. While income to the support tranche is more variable, it is also higher yielding. Estimates of the yield, average life, and lockout periods of the PAC tranche is more certain.

Examples   **Calculate Cash Flows for Each PAC Tranche**

Define the mortgage pool under consideration for CMO structuring using mbscfamounts or mbspassthrough. Calculate the underlying mortgage cash flow, define the PAC schedule and CMO tranches, and calculate the cash flows for each tranche.

Calculate the underlying cash flow using mbspassthrough.

```
% Underlying mortgage
MortgagePrincipal = 1000000;
Coupon = 0.12;
Terms = 6; % months

[PrincipalBalance, MonthlyPayments, SchedPrincipalPayments, ...
InterestPayments, Prepayments] = ...
mbspassthrough(MortgagePrincipal, Coupon, Terms, Terms, 0, []);
PrincipalPayments = SchedPrincipalPayments.' + Prepayments.'

PrincipalPayments =

   1.0e+05 *

    1.6255    1.6417    1.6582    1.6747    1.6915    1.7084
```

Calculate the PAC schedule for CMO using cmosched.

```
PrepaySpeed = [100 300];
[BalanceSchedule, InitialBalance] ...
= cmosched(MortgagePrincipal, Coupon, Terms, Terms, PrepaySpeed, [])
```

```
BalanceSchedule =

   1.0e+05 *

    8.3617    6.7180    5.0581    3.3828    1.6955         0


InitialBalance =

   9.9886e+05
```

Define CMO tranches.

```
TranchePrincipals = ...
[InitialBalance; MortgagePrincipal-InitialBalance];
TrancheCoupons = [0.12; 0.12];

TrancheCoupons =

    0.1200
    0.1200
```

Calculate cash flows for each tranche.

```
[Balance, Principal, Interest] = ...
cmoschedcf(PrincipalPayments, TranchePrincipals, ...
TrancheCoupons, BalanceSchedule)

Balance =

   1.0e+05 *

    8.3631    6.7213    5.0632    3.3885    1.6970         0
    0.0114    0.0114    0.0114    0.0114    0.0114    0.0000


Principal =
```

```
1.0e+05 *

1.6255    1.6417    1.6582    1.6747    1.6915    1.6970
     0         0         0         0         0    0.0114


Interest =

1.0e+03 *

9.9886    8.3631    6.7213    5.0632    3.3885    1.6970
0.0114    0.0114    0.0114    0.0114    0.0114    0.0114
```

References   Hayre, Lakhbir, ed., *Salomon Smith Barney Guide to Mortgage-Backed and Asset-Backed Securities*, John Wiley and Sons, New York, 2001.

Lyuu, Yuh-Dah, *Financial Engineering and Computation*, Cambridge University Press, 2004.

See Also | cmoseqcf | cmosched | mbscfamounts | mbspassthrough

Tutorials   • "Using Collateralized Mortgage Obligations (CMOs)" on page 5-51

How To   • "Create PAC and Sequential CMO" on page 5-66

cmoseqcf

Purpose	Generate cash flows for sequential collateralized mortgage obligation (CMO)
Syntax	`[balances, principals, interests] = cmoseqcf(PrincipalPayments, TranchePrincipals, TrancheCoupons)` `[balances, principals, interests] = cmoseqcf(PrincipalPayments, TranchePrincipals, TrancheCoupons, HasZ)`

Description `[balances, principals, interests] = cmoseqcf(PrincipalPayments, TranchePrincipals, TrancheCoupons)` generates cash flows for a sequential CMO without a Z-bond, given the underlying mortgage pool payments.

`[balances, principals, interests] = cmoseqcf(PrincipalPayments, TranchePrincipals, TrancheCoupons, HasZ)` generates cash flows for a sequential CMO with a Z-bond, given the underlying mortgage pool payments.

Input Arguments

PrincipalPayments

Matrix of size 1-by-NUMTERMS, where NUMTERMS is the number of terms remaining. Each row contains the underlying principal payment for the time period corresponding to the row number. The underlying principal payments can be calculated using `mbscfamounts` or `mbspassthrough`. The underlying principal payments can also be outputs from other CMO cash flow functions

TranchePrincipals

Matrix of size NUMTRANCHES-by-1, where NUMTRANCHES is the number of tranches in the sequential CMO. Each element of the matrix represents the initial principal for each tranche. If the sequential CMO includes a Z-bond (HasZ is true), the last element of this matrix is the principal of the Z-bond.

TrancheCoupons

Matrix of size NUMTRANCHES-by-1, where NUMTRANCHES is the number of tranches in the sequential CMO. Each element of the matrix represents the coupon for each tranche. If the sequential CMO includes a Z-bond (HasZ is true), the last element of this matrix is the coupon of the Z-bond. The weighted average coupon for the CMO should not exceed the coupon of the underlying mortgage.

HasZ

(Optional) Boolean (true or false). A value of true indicates that the sequential CMO contains a Z-bond, and the last element of TranchePrincipals and TrancheCoupons will be treated as that of the Z-bond. A value of false indicates that there is no Z-bond in the sequential CMO, and the last element of TranchePrincipals and TrancheCoupons will be treated as an ordinary tranche.

Default: false

Output Arguments

Balance

Matrix of size NUMTRANCHES-by-NUMTERMS, where NUMTRANCHES is the number of terms remaining and NUMTRANCHES is the number of tranches. Each element represents the principal balance at the time period corresponding to the column, and for the tranche corresponding to the row.

Principal

Matrix of size NUMTRANCHES-by-NUMTERMS, where NUMTRANCHES is the number of terms remaining and NUMTRANCHES is the number of tranches. Each element represents the principal payments made at the time period corresponding to the column, and to the tranche corresponding to the row.

Interest

Matrix of size NUMTRANCHES-by-NUMTERMS, where NUMTRANCHES is the number of terms remaining and NUMTRANCHES is the number of tranches. Each element represents the interest payments made at the time period

corresponding to the column, and to the tranche corresponding to the row.

Definitions   **Sequential Pay CMO**

A sequential pay CMO involves tranches that pay off principal sequentially. For example, consider the following case, where all principal from the underlying mortgage pool is repaid on tranche A first, then tranche B, then tranche C. Note that interest is paid on each tranche as long as the principal for the tranche has not been retired.

CMO Tranche

Tranche is a term often used to describe a specific class of bonds within an offering wherein each tranche offers varying degrees of risk to the investor.

Examples   **Calculate Cash Flows for a Sequential Collateralized Mortgage Obligation (CMO)**

Define the mortgage pool under consideration for CMO structuring using `mbscfamounts` or `mbspassthrough` and calculate the cash flows with an A and B tranche for a sequential CMO.

Calculate underlying cash flow using `mbspassthrough`.

```
MortgagePrincipal = 1000000;
Coupon = 0.12;
Terms = 6; % months

% Calculate underlying mortgage cash flows
[PrincipalBalance, MonthlyPayments, SchedPrincipalPayments, ...
InterestPayments, Prepayments] = ...
mbspassthrough(MortgagePrincipal, Coupon, Terms, Terms, O, []);
PrincipalPayments = SchedPrincipalPayments.' + Prepayments.'

PrincipalPayments =

   1.0e+05 *
```

```
      1.6255     1.6417     1.6582     1.6747     1.6915     1.7084
```

Define CMO tranches, A and B.

```
TranchePrincipals = [500000; 500000];
TrancheCoupons = [0.12; 0.12];
```

Calculate cash flows for each tranche.

```
[Balance, Principal, Interest] = ...
cmoseqcf(PrincipalPayments, TranchePrincipals, TrancheCoupons, false)

Balance =

   1.0e+05 *

   3.3745    1.7328    0.0746         0         0         0
   5.0000    5.0000    5.0000    3.3999    1.7084    0.0000


Principal =

   1.0e+05 *

   1.6255    1.6417    1.6582    0.0746         0         0
        0         0         0    1.6001    1.6915    1.7084


Interest =

   1.0e+03 *

   5.0000    3.3745    1.7328    0.0746         0         0
   5.0000    5.0000    5.0000    5.0000    3.3999    1.7084
```

References   Hayre, Lakhbir, ed., *Salomon Smith Barney Guide to Mortgage-Backed and Asset-Backed Securities*, John Wiley and Sons, New York, 2001.

cmoseqcf

Lyuu, Yuh-Dah, *Financial Engineering and Computation*, Cambridge University Press, 2004.

See Also | cmoschedcf | mbscfamounts | mbspassthrough | cmosched

Tutorials   • "Using Collateralized Mortgage Obligations (CMOs)" on page 5-51

How To   • "Create PAC and Sequential CMO" on page 5-66

Purpose　　Bond conversion factors

Syntax　　CF = convfactor(RefDate, Maturity, CouponRate)
CF = convfactor(RefDate, Maturity, CouponRate,
'ParameterName',*ParameterValue* ...)

Description　　CF = convfactor(RefDate, Maturity, CouponRate) computes a
conversion factor for a bond futures contract.

CF = convfactor(RefDate, Maturity, CouponRate,
'ParameterName',*ParameterValue* ...) accepts optional inputs as
one or more comma-separated parameter-value pairs. *'ParameterName'*
is the name of the parameter inside single quotes. *'ParameterValue* is
the value corresponding to *'ParameterName'*. Specify parameter/value
pairs in any order. Names are case-insensitive. convfactor computes
a conversion factor for a bond futures contract, given a Convention
value for a U.S. Treasury bond, German bond, U.K. Gilt, or Japanese
Government Bond.

**Input
Arguments**
RefDate

Reference dates, for which conversion factor is computed (usually the
first day of delivery months).

Maturity

Maturity date of the underlying bond.

CouponRate

Annual coupon rate of the underlying bond in decimal.

**Parameter–Value
Pairs**　　Enter the following inputs only as parameter–value pairs.

Convention

Conversion factor convention. Scalar. Valid values are:

- 1 = U.S. Treasury bond (30-year) and Treasury note (10-year) futures contract

- 2 = U.S. 2-year and 5-year Treasury note futures contract

- 3 = German Bobl, Bund, Buxl, and Schatz

- 4 = U.K. gilts

- 5 = Japanese Government Bonds (JGBs)

   Default: 1

FirstCouponDate

Irregular or normal first coupon date.

RefYield

Reference semiannual yield.

   Default: 0.06 (6%)

StartDate

Forward starting date of payments.

Output Arguments

CF

N-by1 vector of conversion factors against the 6% yield par-bond.

Definitions

Conversion factors of U.S. Treasury bonds and other government bonds are based on a bond yielding 6%. Optionally, you can specify other types of bonds and yields using inputs for RefYield and Convention. For U.S. Treasury bonds, verify the output of convfactor by comparing the output against the quotations provided by the Chicago Board of Trade (http://www.cbot.com).

For German bonds, verify the output of convfactor by comparing the output against the quotations provided by Eurex (http://www.eurexchange.com).

For U.K. Gilts, verify the output of `convfactor` by comparing the output against the quotations provided by Euronext (`http://www.euronext.com`).

For Japanese Government Bonds, verify the output of `convfactor` by comparing the output against the quotations provided by the Tokyo Stock Exchange (`http://www.tse.or.jp/english/`).

Examples
Calculate `CF`, given the following `RefDate`, `Maturity`, and `CouponRate`:

```
RefDate  = {'1-Dec-2002';
            '1-Mar-2003';
            '1-Jun-2003';
            '1-Sep-2003';
            '1-Dec-2003';
            '1-Sep-2003';
            '1-Dec-2002';
            '1-Jun-2003'};

Maturity = {'15-Nov-2012';
            '15-Aug-2012';
            '15-Feb-2012';
            '15-Feb-2011';
            '15-Aug-2011';
            '15-Aug-2010';
            '15-Aug-2009';
            '15-Feb-2010'};

CouponRate = [0.04; 0.04375; 0.04875; 0.05; 0.05; 0.0575; 0.06; 0.065];

    CF = convfactor(RefDate, Maturity, CouponRate)
```

This returns:

```
CF =
        0.8539
        0.8858
        0.9259
```

```
0.9418
0.9403
0.9862
1.0000
1.0266
```

Calculate `cf`, given the following `RefDate`, `Maturity`, and `CouponRate` for a German Bond:

```
cf = convfactor('3/10/2009','1/04/2018', .04,.06,3)
```

This returns:

```
cf =

    0.8659
```

References   Burghardt, G., T. Belton, M. Lane, and J. Papa, *The Treasury Bond Basis*, McGraw-Hill, 2005.

Krgin, Dragomir, *Handbook of Global Fixed Income Calculations*, John Wiley & Sons, 2002.

See Also   `tfutbyprice` | `tfutbyyield` | `tfutimprepo` | `bndfutimprepo` | `bndfutprice`

How To   • "Bond Futures" on page 7-12

fitFunction

Purpose	Custom fit interest-rate curve object to bond market data
Class	@IRFunctionCurve
Syntax	CurveObj = IRFunctionCurve.fitFunction(Type, Settle, FunctionHandle, Instruments, IRFitOptionsObj) CurveObj = IRFunctionCurve.fitFunction(Type, Settle, FunctionHandle, Instruments, IRFitOptionsObj, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

Type	Type of interest-rate curve for a bond: zero, forward, or discount.
Settle	Scalar or column vector of settlement dates. Settle must be earlier than Maturity.
FunctionHandle	Function handle that defines the interest-rate curve. The function handle takes two numeric vectors (time-to-maturity and a vector of function coefficients) and returns one numeric output (interest rate or discount factor). For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.
Instruments	N-by-4 data matrix for Instruments where the first column is Settle date, the second column is Maturity, the third column is the clean price, and the fourth column is a CouponRate for the bond.
IRFitOptionsObj	Object constructed from IRFitOptions.

fitFunction

Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRFunctionCurve object:

- -1 = Continuous compounding
- 1 = Annual compounding
- 2 = Semiannual compounding (default)
- 3 = Compounding three times per year
- 4 = Quarterly compounding
- 6 = Bimonthly compounding
- 12 = Monthly compounding

Basis	(Optional) Day-count basis of the bond. A scalar of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Instrument Parameters

For each bond Instrument, you can specify the following additional instrument parameters as parameter/value pairs. For example, InstrumentBasis distinguishes a bond instrument's Basis value from the curve's Basis value.

InstrumentPeriod   (Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.

InstrumentBasis   (Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page
Glossary-1.

InstrumentEndMonthRule (Optional) End-of-month rule. A vector.
This rule applies only when `Maturity` is an
end-of-month date for a month having 30 or
fewer days. 0 = ignore rule, meaning that a
bond's coupon payment date is always the
same numerical day of the month. 1 = set rule
on (default), meaning that a bond's coupon
payment date is always the last actual day of
the month.

InstrumentIssueDate (Optional) Date when an instrument was
issued.

InstrumentFirstCouponDate (Optional) Date when a bond makes its
first coupon payment; used when bond has
an irregular first coupon period. When
`FirstCouponDate` and `LastCouponDate`
are both specified, `FirstCouponDate` takes
precedence in determining the coupon
payment structure. If you do not specify a
`FirstCouponDate`, the cash flow payment dates
are determined from other inputs.

InstrumentLastCouponDate (Optional) Last coupon date of a bond before the
maturity date; used when bond has an irregular
last coupon period. In the absence of a specified
`FirstCouponDate`, a specified `LastCouponDate`
determines the coupon structure of the bond.
The coupon structure of a bond is truncated
at the `LastCouponDate`, regardless of where
it falls, and is followed only by the bond's
maturity cash flow date. If you do not specify a
`LastCouponDate`, the cash flow payment dates
are determined from other inputs.

InstrumentFace (Optional) Face or par value. Default = 100.

> **Note** When using `Instrument` parameter/value pairs, you can specify simple interest for a bond by specifying the `InstrumentPeriod` value as `0`. If `InstrumentBasis` and `InstrumentPeriod` are not specified for a bond, the following default values are used: `Basis` is `0` (act/act) and `Period` is `2`.

Description `CurveObj = IRFunctionCurve.fitFunction(Type, Settle, FunctionHandle, Instruments, IRFitOptionsObj, 'Parameter1', Value1, 'Parameter2', Value2, ...)` fits a bond to a custom fitting function. You must enter the optional arguments for `Basis` and `Compounding` as parameter/value pairs.

Examples
```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];
CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
CurveSettle = datenum('30-Apr-2008');
OptOptions = optimoptions('lsqnonlin','display','iter');
functionHandle = @(t,theta) polyval(theta,t);

CustomModel = IRFunctionCurve.fitFunction('Zero', CurveSettle, ...
functionHandle,Instruments, ...
IRFitOptions([.05 .05 .05],'FitType','price',...
'OptOptions',OptOptions));
```

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
0	4	38036.7		4.92e+04	
1	8	38036.7	10	4.92e+04	0
2	12	38036.7	2.5	4.92e+04	0
3	16	38036.7	0.625	4.92e+04	0
4	20	38036.7	0.15625	4.92e+04	0

```
 5     24    30741.5    0.0390625    1.72e+05    0
 6     28    30741.5     0.078125    1.72e+05    0
 7     32    30741.5    0.0195312    1.72e+05    0
 8     36    28713.6   0.00488281    2.33e+05    0
 9     40    20323.3   0.00976562    9.47e+05    0
10     44    20323.3    0.0195312    9.47e+05    0
11     48    20323.3   0.00488281    9.47e+05    0
12     52    20323.3    0.0012207    9.47e+05    0
13     56    19698.8  0.000305176    1.08e+06    0
14     60      17493  0.000610352       7e+06    0
15     64      17493    0.0012207       7e+06    0
16     68      17493  0.000305176       7e+06    0
17     72    15455.1  7.62939e-05    2.25e+07    0
18     76    15455.1  0.000177499    2.25e+07    0
19     80    13317.1   3.8147e-05    3.18e+07    0
20     84    12865.3  7.62939e-05    7.83e+07    0
21     88    11779.8  7.62939e-05    7.58e+06    0
22     92    11747.6  0.000152588    1.45e+05    0
23     96    11720.9  0.000305176    2.33e+05    0
24    100    11667.2  0.000610352    1.48e+05    0
25    104    11558.6    0.0012207    3.55e+05    0
26    108    11335.5   0.00244141    1.57e+05    0
27    112    10863.8   0.00488281    6.36e+05    0
28    116    9797.14   0.00976562    2.53e+05    0
29    120    6882.83    0.0195312    9.18e+05    0
30    124    6882.83    0.0373993    9.18e+05    0
31    128    3218.45   0.00934981    1.96e+06    0
32    132    612.703    0.0186996    3.01e+06    0
33    136    13.0998    0.0253882    3.05e+06    0
34    140  0.0762922   0.00154002    5.05e+04    0
35    144  0.0731652  3.61102e-06        29.9    0
36    148  0.0731652  6.32335e-08       0.063    0

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to
its initial value is less than the default value of the function tolerance.
```

How To

fitNelsonSiegel

Purpose	Fit Nelson-Siegel function to bond market data
Class	@IRFunctionCurve
Syntax	CurveObj = IRFunctionCurve.fitNelsonSiegel(Type, Settle, Instruments) CurveObj = IRFunctionCurve.fitNelsonSiegel(Type, Settle, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

Type	Type of interest-rate curve for a bond: zero or forward.
Settle	Scalar or column vector of settlement dates.
Instruments	N-by-4 data matrix for Instruments where the first column is Settle date, the second column is Maturity, the third column is the clean price, and the fourth column is a CouponRate for the bond.
Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRFunctionCurve object: • -1 = Continuous compounding
Basis	(Optional) Day-count basis of the interest-rate curve. A scalar of integers. • 0 = actual/actual (default) • 1 = 30/360 (SIA) • 2 = actual/360 • 3 = actual/365 • 4 = 30/360 (BMA) • 5 = 30/360 (ISDA) • 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

IRFitOptionsObj (Optional) Object constructed from IRFitOptions.

Instrument Parameters

For each bond Instrument, you can specify the following additional instrument parameters as parameter/value pairs. For example, InstrumentBasis distinguishes a bond instrument's Basis value from the curve's Basis value.

InstrumentPeriod (Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.

InstrumentBasis (Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see basis.

InstrumentEndMonthRule (Optional) End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

InstrumentIssueDate (Optional) Date when an instrument was issued.

InstrumentFirstCouponDate (Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

<table>
<tr>
<td>InstrumentLastCouponDate</td>
<td>(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.</td>
</tr>
<tr>
<td>InstrumentFace</td>
<td>(Optional) Face or par value. Default = 100.</td>
</tr>
</table>

Note When using Instrument parameter/value pairs, you can specify simple for a bond by specifying the InstrumentPeriod value as 0. If InstrumentBasis and InstrumentPeriod are not specified for a bond, the following default values are used: Basis is 0 (act/act) and Period is 2.

Description   CurveObj = IRFunctionCurve.fitNelsonSiegel(Type, Settle, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...) fits a Nelson-Siegel function to market data for a bond. You must enter the optional arguments for Basis, Compounding, and IRFitOptionsObj as parameter/value pairs. After creating a Nelson-Siegel model, you can view the model parameters using:

CurveObj.Parameters

Examples
```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];

CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
```

```
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
PlottingPoints = datenum('07-Mar-2009'):180:datenum('07-Mar-2036');
Yield = bndyield(CleanPrice,CouponRate,Settle,Maturity);


NSModel = IRFunctionCurve.fitNelsonSiegel('Zero',datenum('30-Apr-2008'),Instruments);


NSModel.Parameters

ans =

   4.6617   -1.0227   -0.3484    1.2386
```

To create the plot:

```
plot(PlottingPoints,NSModel.getParYields(PlottingPoints),'r')
hold on
scatter(Maturity,Yield,'black')
datetick('x')
```
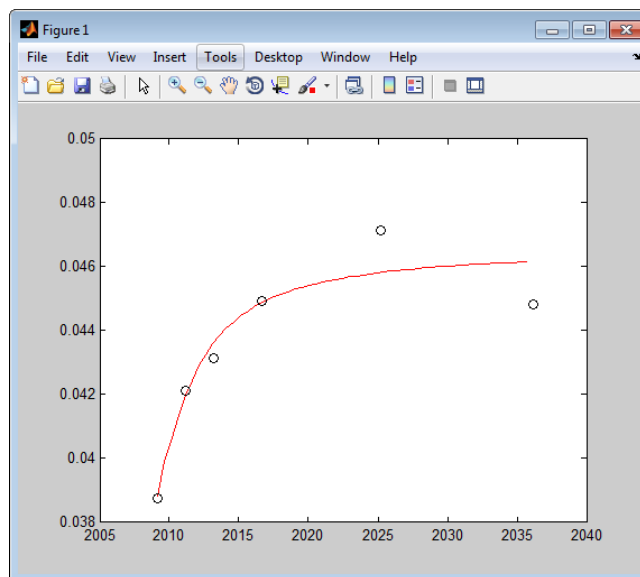
Algorithms   The Nelson-Siegel model proposes that the instantaneous forward curve can be modeled with the following:

$$f = \beta_0 + \beta_1 e^{\frac{-m}{\tau}} + \beta_2 \frac{m}{\tau} e^{\frac{-m}{\tau}}$$

This can be integrated to derive an equation for the zero curve (see [6] for more information on the equations and the derivation):

$$s = \beta_0 + (\beta_1 + \beta_2)\frac{\tau}{m}(1 - e^{\frac{-m}{\tau}}) - \beta_2 e^{\frac{-m}{\tau}}$$

See [1] for more information.

References

[1] Nelson, C.R., Siegel, A.F., (1987), "Parsimonious modelling of yield curves", Journal of Business, 60, pp 473-89

[2] Svensson, L.E.O. (1994), "Estimating and interpreting forward interest rates: Sweden 1992-4", International Monetary Fund, IMF Working Paper, 1994/114

[3] Fisher, M., Nychka, D., Zervos, D. (1995), "Fitting the term structure of interest rates with smoothing splines", Board of Governors of the Federal Reserve System, Federal Reserve Board Working Paper 95-1

[4] Anderson, N., Sleath, J. (1999), "New estimates of the UK real and nominal yield curves", Bank of England Quarterly Bulletin, November, pp 384-92

[5] Waggoner, D. (1997), "Spline Methods for Extracting Interest Rate Curves from Coupon Bond Prices", Federal Reserve Board Working Paper 97-10

[6] "Zero-coupon yield curves: technical documentation", BIS Papers No. 25, October 2005

fitNelsonSiegel

[7] Bolder, D.J., Gusba,S (2002), "Exponentials, Polynomials, and Fourier Series: More Yield Curve Modelling at the Bank of Canada," Working Papers 02-29, Bank of Canada

[8] Bolder, D.J., Streliski, D (1999), "Yield Curve Modelling at the Bank of Canada," Technical Reports 84, Bank of Canada

See Also   "@IRFitOptions" on page A-10 **|** "@IRFunctionCurve" on page A-12 **|**

Related Examples
* "Fitting Interest Rate Curve Functions" on page 9-29

fitSmoothingSpline

Purpose	Fit smoothing spline to bond market data
Class	@IRFunctionCurve
Syntax	CurveObj = IRFunctionCurve.fitSmoothingSpline(Type, Settle, Instruments, Lambdafun) CurveObj = IRFunctionCurve.fitSmoothingSpline(Type, Settle, Instruments, Lambdafun, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

Note You must have a license for Curve Fitting Toolbox software to use the fitSmoothingSpline method.

Type	Type of interest-rate curve for a bond: Forward, Zero, or Discount.
Settle	Scalar or column vector of settlement dates.
Instruments	N-by-4 data matrix for Instruments where the first column is Settle date, the second column is Maturity, the third column is the clean price, and the fourth column is a CouponRate for the bond.
Lambdafun	Penalty function that takes as its input time and returns a penalty value. Use a function handle to support the penalty function. The function handle for the penalty function which takes one numeric input (time-to-maturity) and returns one numeric output (penalty to be applied to the curvature of the spline). For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.

fitSmoothingSpline

> **Note** The smoothing spline represents the forward
> curve. The spline is penalized for curvature by
> specifying a penalty function. This fit may only be
> done with a `FitType` of `DurationWeightedPrice`.

Knots   (Optional) Vector of knot locations
(times-to-maturity); by default, knots is set to
be a vector comprised of `0` and the time to maturity
of all input instruments. The default is for the spline
type to be cubic but you can specify any spline type
by explicitly specifying the knots. User-defined
knots can be specified using the following command,
where k is the order: `augknt(knots,k)`.

Compounding   (Optional) Scalar that sets the compounding
frequency per year for the `IRFunctionCurve` object:

- -1 = Continuous compounding (default)

- 1 = Annual compounding

- 2 = Semiannual compounding

- 3 = Compounding three times per year

- 4 = Quarterly compounding

- 6 = Bimonthly compounding

- 12 = Monthly compounding

Basis   (Optional) Day-count basis of the interest-rate curve.
A scalar of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Instrument Parameters

For each bond Instrument, you can specify the following additional instrument parameters as parameter/value pairs. For example, InstrumentBasis distinguishes a bond instrument's Basis value from the curve's Basis value.

InstrumentPeriod   (Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.

InstrumentBasis   (Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

InstrumentEndMonthRule (Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

InstrumentIssueDate (Optional) Date when an instrument was issued.

InstrumentFirstCouponDate   (Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

InstrumentLastCouponDate   (Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

InstrumentFace   (Optional) Face or par value. Default = 100.

Note When using `Instrument` parameter/value pairs, you can specify simple interest for a bond by specifying the `InstrumentPeriod` value as `0`. If `InstrumentBasis` and `InstrumentPeriod` are not specified for a bond, the following default values are used: `Basis` is `0` (act/act) and `Period` is `2`.

Description   `Fcurve = IRFunctionCurve.fitSmoothingSpline(Type, Settle, Instruments, Lambdafun, 'Parameter1', Value1, 'Parameter2', Value2, ...)` fits a smoothing spline to market data for a bond. You must enter the optional arguments for `Basis`, `Compounding`, and `Knots` as parameter/value pairs.

fitSmoothingSpline

Examples
```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];

CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
PlottingPoints = datenum('07-Mar-2009'):180:datenum('07-Mar-2036');
Yield = bndyield(CleanPrice,CouponRate,Settle,Maturity);

% Use the AUGKNT function to construct the knots for a cubic spline at every 5 years

CustomKnots = augknt(0:5:30,4);
SmoothingModel = IRFunctionCurve.fitSmoothingSpline('Zero',datenum('30-Apr-2008'),...
Instruments,@(t) 1000,'knots', CustomKnots)
```

To create the plot:

```
plot(PlottingPoints,SmoothingModel.getParYields(PlottingPoints),'b')
hold on
scatter(Maturity,Yield,'black')
datetick('x')
```
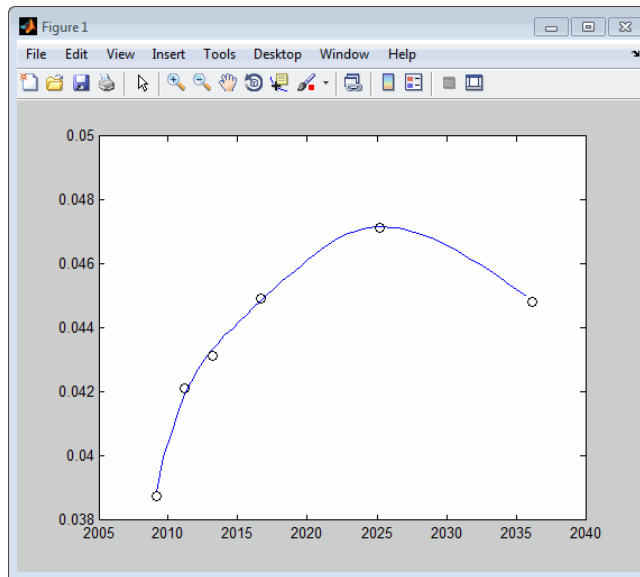
Algorithms   The term structure can be modeled with a spline — specifically, one way to model the term structure is by representing the forward curve with a cubic spline. To ensure that the spline is sufficiently smooth, a penalty is imposed relating to the curvature (second derivative) of the spline:

$$\sum_{i=1}^{N} [\frac{P_i - \hat{P}_i(f)}{D_i}]^2 + \int_0^M \lambda_t(m)[f''(m)]^2 dm$$

where the first term is the difference between the observed price *P* and the predicted price, $\check{P}$, (weighted by the bond's duration, *D*) summed over all bonds in our data set and the second term is the penalty term (where λ is a penalty function and *f* is the spline).

See [3], [4], [5] below.

There have been different proposals for the specification of the penalty function λ. One approach, advocated by [4], and currently used by the UK Debt Management Office, is a penalty function of the following form:

$$log(\lambda(m)) = L - (L - S)e^{\frac{-m}{r}}$$

References

[1] Nelson, C.R., Siegel, A.F., (1987), "Parsimonious modelling of yield curves", Journal of Business, 60, pp 473-89

[2] Svensson, L.E.O. (1994), "Estimating and interpreting forward interest rates: Sweden 1992-4", International Monetary Fund, IMF Working Paper, 1994/114

[3] Fisher, M., Nychka, D., Zervos, D. (1995), "Fitting the term structure of interest rates with smoothing splines", Board of Governors of the Federal Reserve System, Federal Reserve Board Working Paper 95-1

[4] Anderson, N., Sleath, J. (1999), "New estimates of the UK real and nominal yield curves", Bank of England Quarterly Bulletin, November, pp 384-92

[5] Waggoner, D. (1997), "Spline Methods for Extracting Interest Rate Curves from Coupon Bond Prices", Federal Reserve Board Working Paper 97-10

[6] "Zero-coupon yield curves: technical documentation", BIS Papers No. 25, October 2005

[7] Bolder, D.J., Gusba,S (2002), "Exponentials, Polynomials, and Fourier Series: More Yield Curve Modelling at the Bank of Canada," Working Papers 02-29, Bank of Canada

[8] Bolder, D.J., Streliski, D (1999), "Yield Curve Modelling at the Bank of Canada," Technical Reports 84, Bank of Canada

See Also   "@IRFunctionCurve" on page A-12 |

Related Examples
- "Fitting Interest Rate Curve Functions" on page 9-29

Purpose	Fit Svensson function to bond market data
Class	@IRFunctionCurve
Syntax	CurveObj = IRFunctionCurve.fitSvensson(Type, Settle, Instruments) CurveObj = IRFunctionCurve.fitSvensson(Type, Settle, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

Type	Type of interest-rate curve for a bond: zero or forward.
Settle	Scalar or column vector of settlement dates.
Instruments	N-by-4 data matrix for Instruments where the first column is Settle date, the second column is Maturity, the third column is the clean price, and the fourth column is a CouponRate for the bond.
Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRFunctionCurve object: • -1 = Continuous compounding

fitSvensson

Basis	(Optional) Day-count basis of the interest-rate curve. A scalar of integers.	

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

IRFitOptions	(Optional) Object constructed from IRFitOptions.

Instrument Parameters

For each bond Instrument, you can specify the following additional instrument parameters as parameter/value pairs. For example, InstrumentBasis distinguishes a bond instrument's Basis value from the curve's Basis value.

InstrumentPeriod	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.
InstrumentBasis	(Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see basis.

InstrumentEndMonthRule (Optional) End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

InstrumentIssueDate (Optional) Date when an instrument was issued.

InstrumentFirstCouponDate (Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure. If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

InstrumentLastCouponDate (Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

InstrumentFace (Optional) Face or par value. Default = 100.

> **Note** When using `Instrument` parameter/value pairs, you can specify simple interest for a bond by specifying the `InstrumentPeriod` value as `0`. If `InstrumentBasis` and `InstrumentPeriod` are not specified for a bond, the following default values are used: `Basis` is `0` (act/act) and `Period` is `2`.

Description `CurveObj = IRFunctionCurve.fitSvensson(Type, Settle, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...)` fits the Svensson function to bond market data. You must enter the optional arguments for `Basis`, `Compounding`, and `IRFitOptions` as parameter/value pairs. After creating a Svensson model, you can view the model parameters using:

`CurveObj.Parameters`

Examples
```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];

CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
PlottingPoints = datenum('07-Mar-2009'):180:datenum('07-Mar-2036');
Yield = bndyield(CleanPrice,CouponRate,Settle,Maturity);

SvenssonModel = IRFunctionCurve.fitSvensson('Zero',datenum('30-Apr-2008'),Instruments);

SvenssonModel.Parameters

ans =

    0.0002    3.7616   -24.5489   34.3165   10.8867   11.9282
```

To create a plot:

```
plot(PlottingPoints,SvenssonModel.getParYields(PlottingPoints),'g')
hold on
scatter(Maturity,Yield,'black')
datetick('x')
```
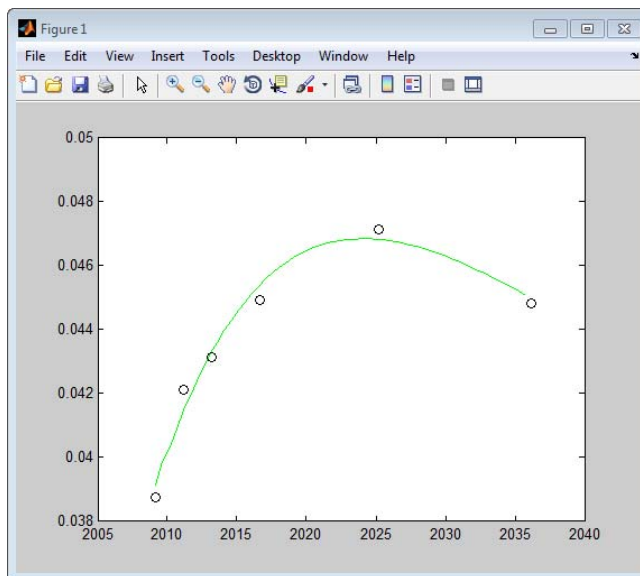

Algorithms   A very similar model to the Nelson-Siegel is the Svensson model, which adds two additional parameters to account for greater flexibility in the term structure. This model proposes that the forward rate can be modeled with the following form:

$$f = \beta_0 + \beta_1 e^{\frac{-m}{\tau_1}} + \beta_2 e^{\frac{-m}{\tau_1}} \frac{m}{\tau_1} + \beta_3 e^{\frac{-m}{\tau_2}} \frac{m}{\tau_2}$$

As above, this can be integrated to derive an equation for the zero curve:

$$s = \beta_0 + \beta_1 (1 - e^{\frac{-m}{\tau_1}})(-\frac{\tau_1}{m}) + \beta_2 ((1 - e^{\frac{-m}{\tau_1}})\frac{\tau_1}{m} - e^{\frac{m}{\tau_1}}) + \beta_3 ((1 - e^{\frac{-m}{\tau_2}})\frac{\tau_2}{m} - e^{\frac{m}{\tau_2}})$$

References

[1] Nelson, C.R., Siegel, A.F., (1987), "Parsimonious modelling of yield curves", Journal of Business, 60, pp 473-89

[2] Svensson, L.E.O. (1994), "Estimating and interpreting forward interest rates: Sweden 1992-4", International Monetary Fund, IMF Working Paper, 1994/114

[3] Fisher, M., Nychka, D., Zervos, D. (1995), "Fitting the term structure of interest rates with smoothing splines", Board of Governors of the Federal Reserve System, Federal Reserve Board Working Paper 95-1

[4] Anderson, N., Sleath, J. (1999), "New estimates of the UK real and nominal yield curves", Bank of England Quarterly Bulletin, November, pp 384-92

[5] Waggoner, D. (1997), "Spline Methods for Extracting Interest Rate Curves from Coupon Bond Prices", Federal Reserve Board Working Paper 97-10

[6] "Zero-coupon yield curves: technical documentation", BIS Papers No. 25, October 2005

[7] Bolder, D.J., Gusba,S (2002), "Exponentials, Polynomials, and Fourier Series: More Yield Curve Modelling at the Bank of Canada," Working Papers 02-29, Bank of Canada

[8] Bolder, D.J., Streliski, D (1999), "Yield Curve Modelling at the Bank of Canada," Technical Reports 84, Bank of Canada

See Also   "@IRFunctionCurve" on page A-12 **|** "@IRFitOptions" on page A-10 **|**

Related Examples   • "Fitting Interest Rate Curve Functions" on page 9-29

getDiscountFactors

Purpose	Get discount factors for input dates for `IRDataCurve`
Class	`@IRDataCurve`
Syntax	`F = getDiscountFactors(CurveObj, InpDates)`

Arguments

CurveObj	Interest-rate curve object that is constructed using `IRDataCurve`.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.

Description   `F = getDiscountFactors(CurveObj, InpDates)` returns discount factors for the input dates.

Examples

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data);
irdc.getDiscountFactors(today+30:30:today+720)

ans =

    0.9986
    0.9971
    0.9956
    0.9940
    0.9924
    0.9907
    0.9890
    0.9873
    0.9855
    0.9836
    0.9817
    0.9798
    0.9778
```

```
0.9757
0.9736
0.9715
0.9693
0.9671
0.9649
0.9626
0.9602
0.9578
0.9554
0.9529
```

How To
- "@IRDataCurve" on page A-7

getDiscountFactors

Purpose	Get discount factors for input dates for IRFunctionCurve
Class	@IRFunctionCurve
Syntax	F = getDiscountFactors(CurveObj, InpDates)

Arguments

CurveObj	Interest-rate curve object that is constructed using the IRFunctionCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.

Description   F = getDiscountFactors(CurveObj, InpDates) returns discount factors for the input dates.

Examples
```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));
irfc.getDiscountFactors(today+30:30:today+720)

ans =

    0.9984
    0.9967
    0.9950
    0.9933
    0.9916
    0.9899
    0.9881
    0.9864
    0.9846
    0.9828
    0.9810
    0.9792
    0.9773
    0.9755
    0.9736
```

```
    0.9717
    0.9698
    0.9679
    0.9660
    0.9641
    0.9621
    0.9602
    0.9582
    0.9562
```

How To   • "@IRFunctionCurve" on page A-12

getForwardRates

Purpose	Get forward rates for input dates for IRDataCurve
Class	@IRDataCurve
Syntax	`F = getForwardRates(CurveObj, InpDates)` `F = getforwardrates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)`

Arguments

CurveObj	Interest-rate curve object that is constructed using IRDataCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for forward rates are:

- -1 = Continuous compounding

- 1 = Annual compounding

- 2 = Semiannual compounding (default)

- 3 = Compounding three times per year

- 4 = Quarterly compounding

- 6 = Bimonthly compounding

- 12 = Monthly compounding

Basis	(Optional) Day-count basis values for the forward rates:

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Description   F = getForwardRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...) returns forward rates for the input dates. You must enter the optional arguments for Basis and Compounding as parameter/value pairs.

Examples
```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data);
irdc.getForwardRates(today+30:30:today+720)

ans =

    0.0174
    0.0180
    0.0187
    0.0193
    0.0199
    0.0205
    0.0212
    0.0218
```

```
0.0224
0.0230
0.0237
0.0243
0.0249
0.0255
0.0262
0.0268
0.0274
0.0280
0.0287
0.0293
0.0299
0.0305
0.0312
0.0318
```

How To   • "@IRDataCurve" on page A-7

getForwardRates

Purpose	Get forward rates for input dates for `IRFunctionCurve`
Class	`@IRFunctionCurve`
Syntax	`F = getForwardRates(CurveObj, InpDates)`
	`F = getforwardrates(CurveObj, InpDates, 'Parameter1',`
	`Value1, 'Parameter2', Value2, ...)`

Arguments

CurveObj	Interest-rate curve object that is constructed using `IRFunctionCurve`.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for the forward rates are:

- -1 = Continuous compounding

- 1 = Annual compounding

- 2 = Semiannual compounding (default)

- 3 = Compounding three times per year

- 4 = Quarterly compounding

- 6 = Bimonthly compounding

- 12 = Monthly compounding

Basis	(Optional) Day-count basis for the forward rates:

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Description   F = getForwardRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...) returns forward rates for the input dates. You must enter the optional arguments for Basis and Compounding as parameter/value pairs.

Examples
```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));
irfc.getForwardRates(today+30:30:today+720)

ans =

    0.0202
    0.0205
    0.0207
    0.0210
    0.0212
    0.0215
    0.0217
    0.0219
    0.0222
    0.0224
    0.0226
```

```
0.0229
0.0231
0.0233
0.0235
0.0238
0.0240
0.0242
0.0244
0.0247
0.0249
0.0251
0.0253
0.0255
```

How To   • "@IRFunctionCurve" on page A-12

getParYields

Purpose	Get par yields for input dates for `IRDataCurve`
Class	`@IRDataCurve`

Syntax	`F = getParYields(CurveObj, InpDates)`
	`F = getParYields(CurveObj, InpDates, 'Parameter1',`
	`Value1, 'Parameter2', Value2, ...)`

Arguments		
	CurveObj	Interest-rate curve object that is constructed using `IRDataCurve`.
	InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
	Compounding	(Optional) Scalar that sets the compounding frequency per year for the par yield rates are:

- -1 = Continuous compounding

- 1 = Annual compounding

- 2 = Semiannual compounding (default)

- 3 = Compounding three times per year

- 4 = Quarterly compounding

- 6 = Bimonthly compounding

- 12 = Monthly compounding

	Basis	(Optional) Day-count basis values for the par yield rates:

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Description   F = getParYields(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...) returns par yields for the input dates. You must enter the optional arguments for Basis and Compounding as parameter/value pairs.

Examples
```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data);
irdc.getParYields(today+30:30:today+720)

ans =

    0.0174
    0.0179
    0.0181
    0.0185
    0.0187
    0.0191
    0.0194
    0.0195
```

```
0.0199
0.0202
0.0205
0.0208
0.0212
0.0215
0.0218
0.0221
0.0224
0.0228
0.0231
0.0233
0.0236
0.0239
0.0242
0.0245
```

How To   • "@IRDataCurve" on page A-7

Purpose	Get par yields for input dates for IRFunctionCurve
Class	@IRFunctionCurve
Syntax	F = getParYields(CurveObj, InpDates) F = getParYields(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

CurveObj	Interest-rate curve object that is constructed using IRFunctionCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for par yield rates are:

- -1 = Continuous compounding

- 1 = Annual compounding

- 2 = Semiannual compounding (default)

- 3 = Compounding three times per year

- 4 = Quarterly compounding

- 6 = Bimonthly compounding

- 12 = Monthly compounding

Basis	(Optional) Day-count basis values for par yield rates:

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

getParYields

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

Description F = getParYields(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...) returns par yields for the input dates. You must enter the optional arguments for Basis and Compounding as parameter/value pairs.

Examples

```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));
irfc.getParYields(today+30:30:today+720)

ans =

    0.0202
    0.0205
    0.0205
    0.0207
    0.0207
    0.0209
    0.0210
    0.0209
    0.0211
    0.0212
```

```
0.0213
0.0214
0.0216
0.0217
0.0218
0.0220
0.0220
0.0222
0.0223
0.0223
0.0225
0.0226
0.0227
0.0228
```

How To • "@IRFunctionCurve" on page A-12

getZeroRates

Purpose	Get zero rates for input dates for IRDataCurve
Class	@IRDataCurve
Syntax	F = getZeroRates(CurveObj, InpDates) F = getZeroRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

CurveObj	Interest-rate curve object that is constructed using IRDataCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for zero rates are:

- -1 = Continuous compounding

- 1 = Annual compounding

- 2 = Semiannual compounding (default)

- 3 = Compounding three times per year

- 4 = Quarterly compounding

- 6 = Bimonthly compounding

- 12 = Monthly compounding

Basis	(Optional) Day-count basis values for zero rates:

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Description F = getZeroRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...) returns zero rates for the input dates. You must enter the optional arguments for Basis and Compounding as parameter/value pairs.

Examples
```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data);
irdc.getZeroRates(today+30:30:today+720)


ans =

    0.0174
    0.0177
    0.0180
    0.0183
    0.0187
    0.0190
    0.0193
    0.0196
    0.0199
```

```
0.0202
0.0205
0.0208
0.0212
0.0215
0.0218
0.0221
0.0224
0.0227
0.0230
0.0233
0.0237
0.0240
0.0243
0.0246
```

How To   • "@IRDataCurve" on page A-7

getZeroRates

Purpose	Get zero rates for input dates for `IRFunctionCurve`
Class	`@IRFunctionCurve`
Syntax	`F = getZeroRates(CurveObj, InpDates)`
	`F = getZeroRates(CurveObj, InpDates, 'Parameter1',`
	`Value1, 'Parameter2', Value2, ...)`

Arguments

CurveObj	Interest-rate curve object that is constructed using `IRFunctionCurve`.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for zero rates are:

- -1 = Continuous compounding

- 1 = Annual compounding

- 2 = Semiannual compounding (default)

- 3 = Compounding three times per year

- 4 = Quarterly compounding

- 6 = Bimonthly compounding

- 12 = Monthly compounding

Basis	(Optional) Day-count basis value for zero rates:

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

getZeroRates

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Description F = getZeroRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...) returns zero rates for the input dates. You must enter the optional arguments for Basis and Compounding as parameter/value pairs.

Examples
```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));
irfc.getZeroRates(today+30:30:today+720)


ans =

    0.0202
    0.0204
    0.0205
    0.0206
    0.0207
    0.0209
    0.0210
    0.0211
    0.0212
    0.0213
    0.0214
```

```
0.0216
0.0217
0.0218
0.0219
0.0220
0.0221
0.0223
0.0224
0.0225
0.0226
0.0227
0.0228
0.0229
```

How To   • "@IRFunctionCurve" on page A-12

IRBootstrapOptions

Purpose	Construct specific options for bootstrapping interest-rate curve object
Class	@IRBootstrapOptions
Syntax	mybootoptions = IRBootstrapOptions('Param1', Value1)
Arguments	ConvexityAdjustment (Optional) Controls the convexity adjustment to interest-rate futures. This can be specified as a function handle that takes one numeric input (time-to-maturity) and returns one numeric output, ConvexityAdjustment. For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation. Alternatively, you can define ConvexityAdjustment as an N-by-1 vector of values, where N is the number of interest-rate futures. In either case, the ConvexityAdjustment is subtracted from the futures rate.
Description	mybootoptions = IRBootstrapOptions('Param1', Value1) constructs an IRBootstrapOptionsObj structure. The IRBootstrapOptionsObj is used with the bootstrap method.
Examples	mybootoptions = IRBootstrapOptions('ConvexityAdjustment',repmat(.005,10,1))
How To	• "@IRDataCurve" on page A-7

Purpose	Construct interest-rate curve object from dates and data
Class	@IRDataCurve
Syntax	CurveObj = IRDataCurve(Type, Settle, Dates, Data) CurveObj = IRDataCurve(Type, Settle, Dates, Data, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

Type	Type of interest-rate curve. Acceptable values are forward, zero, or discount.
Settle	Scalar of settlement dates.
Dates	Dates corresponding to rate data.
Data	Interest-rate data for the curve object.
Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRDataCurve object:

- -1 = Continuous compounding
- 1 = Annual compounding
- 2 = Semiannual compounding (default)
- 3 = Compounding three times per year
- 4 = Quarterly compounding
- 6 = Bimonthly compounding
- 12 = Monthly compounding

Basis	(Optional) Day-count basis of the interest-rate curve. A scalar of integers.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

InterpMethod	(Optional) Values are:

- 'linear' — Linear interpolation (default).

- 'constant' — Piecewise constant interpolation.

- 'pchip' — Piecewise cubic Hermite interpolation.

- 'spline' — Cubic spline interpolation.

Description   CurveObj = IRDataCurve(Type, Settle, Dates, Data,
'Parameter1', Value1, 'Parameter2', Value2, ...) constructs an
interest-rate curve with the specified Dates and Data. You must enter
the optional arguments for Basis, Compounding, and InterpMethod
as parameter/value pairs.

Alternatively, an IRDataCurve object can be bootstrapped from market
data using the bootstrap method.

After an IRDataCurve curve object is constructed, you can use the
following methods to determine the forward rates, zero rates, and
discount factors. In addition, you can use the toRateSpec method to
convert the interest-rate curve object to a RateSpec structure.

Method	Description
getForwardRates	Returns forward rates for input dates.
getZeroRates	Returns zero rates for input dates.
getDiscountFactors	Returns discount factors for input dates.
getParYields	Returns par yields for input dates.
toRateSpec	Converts to be a RateSpec object; this structure is identical to the RateSpec
bootstrap	Bootstraps an interest rate curve from market data.

Examples
```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data)

irdc =

  Properties:
          Dates: [8x1 double]
```

```
            Data: [8x1 double]
      InterpMethod: 'linear'
            Type: 'Zero'
          Settle: 733599
      Compounding: 2
           Basis: O
```

How To   • "@IRCurve" on page A-4

Purpose	Construct specific options for fitting interest-rate curve object
Class	@IRFitOptions
Syntax	myfitoptions = IRFitOptions(InitialGuess) myfitoptions = IRFitOptions(InitialGuess, 'Parameter1', Value1)

Arguments

InitialGuess	Initial guess for the parameters of the curve function. Vector of values for the starting point of the optimization.
FitType	(Optional) Price, Yield, or DurationWeightedPrice determines which is minimized in the curve fitting process. The default is DurationWeightedPrice.
UpperBound	(Optional) Lower bound for the parameters of the curve function.
LowerBound	(Optional) Upper bound for the parameters of the curve function.
OptOptions	(Optional) Optimization solver options constructed with optimoptions from the Optimization Toolbox (optimset is also supported).

Description myfitoptions = IRFitOptions('Param1', Value1) constructs the IRFitOptions structure with an initial guess or with an initial guess and bounds. You must enter the optional arguments for FitType, UpperBound, LowerBound, and OptOptions as parameter/value pairs.

Note IRFitOptions constructor must be used with fitFunction method when building a custom fitting function.

IRFitOptions

Examples

```
myfitoptions = IRFitOptions([7 2 1 0],'FitType','yield')

myfitoptions =

  Properties:
         FitType: 'yield'
     InitialGuess: [7 2 1 0]
       UpperBound: []
       LowerBound: []
        OptOptions: []
```

How To

- "@IRFunctionCurve" on page A-12

Purpose	Construct interest-rate curve object from function handle or function and fit to market data
Class	@IRFunctionCurve
Syntax	CurveObj = IRFunctionCurve(Type, Settle, FunctionHandle) CurveObj = IRFunctionCurve(Type, Settle, FunctionHandle, 'Parameter1', Value1, 'Parameter2', Value2, ...)

Arguments

Type	Type of interest-rate curve: zero, forward, or discount.
Settle	Scalar of settlement dates.
FunctionHandle	Function handle that defines the interest-rate curve. The function handle requires one numeric input (time-to-maturity) and returns one numeric output (interest rate or discount factor). For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.
Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRFunctionCurve object: • -1 = Continuous compounding • 1 = Annual compounding • 2 = Semiannual compounding (default) • 3 = Compounding three times per year • 4 = Quarterly compounding • 6 = Bimonthly compounding

	• 12 = Monthly compounding
Basis	(Optional) Day-count basis of the bond. A scalar of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

Description   CurveObj = IRFunctionCurve(Type, Settle, FunctionHandle, 'Parameter1', Value1, 'Parameter2', Value2, ...) constructs an interest-rate curve object directly by specifying a function handle. You must enter the optional arguments for Basis and Compounding as parameter/value pairs.

After you use the IRFunctionCurve constructor to create an IRFunctionCurve object, you can fit the bond using the following methods.

Method	Description
getForwardRates	Returns forward rates for input dates.
getZeroRates	Returns zero rates for input dates.
getDiscountFactors	Returns discount factors for input dates.
getParYields	Returns par yields for input dates.
toRateSpec	Converts to be a RateSpec object.
	This RateSpec structure is identical to the RateSpec produced by the Financial Instruments Toolbox function intenvset.

Alternatively, you can construct an IRFunctionCurve object using the following static methods.

Static Method	Description
fitNelsonSiegel	Fits a Nelson-Siegel function to market data.
fitSvensson	Fits a Svensson function to market data.
fitSmoothingSpline	Fits a smoothing spline function to market data.
fitFunction	Fits a custom function to market data.

Examples

```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t))

irfc =

  Properties:
    FunctionHandle: @(t)polyval([-0.0001,0.003,0.02],t)
              Type: 'Forward'
            Settle: 733599
```

```
Compounding: 2
      Basis: 0
```

How To   • "@IRCurve" on page A-4

Purpose   Duration of LIBOR-based interest-rate swap

Syntax   [PayFixDuration GetFixDuration] = liborduration(SwapFixRate,
                Tenor, Settle)

Arguments

SwapFixRate	Scalar or column vector of swap fixed rates in decimal.
Tenor	Scalar or column vector indicating life of the swap in years. Fractional numbers are rounded upward.
Settle	Scalar or column vector of settlement dates.

Description   [PayFixDuration GetFixDuration] =
                liborduration(SwapFixRate, Tenor, Settle) computes the
                duration of LIBOR-based interest-rate swaps.

                PayFixDuration is the modified duration, in years, realized when
                entering pay-fix side of the swap.

                GetFixDuration is the modified duration, in years, realized when
                entering receive-fix side of the swap.

Examples   Given the data

```
SwapFixRate = 0.0383;
Tenor = 7;
Settle = datenum('11-Oct-2002');
```

compute the swap durations.

```
[PayFixDuration GetFixDuration] = liborduration(SwapFixRate,...
Tenor, Settle)

PayFixDuration =

   -4.7567
```

```
GetFixDuration =

    4.7567
```

See Also   liborfloat2fixed | liborprice

Purpose	Compute par fixed-rate of swap given 3-month LIBOR data

Syntax	`[FixedSpec, ForwardDates, ForwardRates] = liborfloat2fixed(ThreeMonthRates, Settle, Tenor, StartDate, Interpolation, ConvexAdj, RateParam, InArrears, Sigma, FixedCompound, FixedBasis)`

Arguments	ThreeMonthRates	Three-month Eurodollar futures data or forward rate agreement data. (A forward rate agreement stipulates that a certain interest rate applies to a certain principal amount for a given future time period.) An n-by-3 matrix in the form of [month year IMMQuote]. The floating rate is assumed to compound quarterly and to accrue on an actual/360 basis.
	Settle	Settlement date of the swap. Scalar.
	Tenor	Life of the swap. Scalar.
	StartDate	(Optional) Scalar value to denote reference date for valuation of (forward) swap. This in effect allows forward swap valuation. Default = Settle.
	Interpolation	(Optional) Interpolation method to determine applicable forward rate for months when no Eurodollar data is available. Default is 'linear' or 1. Other possible values are 'Nearest' or 0, and 'Cubic' or 2.
	ConvexAdj	(Optional) Default = 0 (off). 1 = on. Denotes whether futures/forward convexity adjustment is required. Pertains to forward rate adjustments when those rates are taken from Eurodollar futures data.

liborfloat2fixed

RateParam	(Optional) Short-rate model's parameters (Hull-White) [a S], where the short-rate process is: $dr = \left[\theta(t) - ar\right]dt + Sdz.$ Default = [0.05 0.015].
InArrears	(Optional) Default = 0 (off). Set to 1 for on. If on, the routine does an automatic a convexity adjustment to forward rates.
Sigma	(Optional) Overall annual volatility of caplets.
FixedCompound	(Optional) Scalar value. Compounding or frequency of payment on the fixed side. Also, the reset frequency. Default = 4 (quarterly). Other values are 1, 2, and 12.
FixedBasis	(Optional) Scalar value. Basis of the fixed side. • 0 = actual/actual (default) • 1 = 30/360 (SIA) • 2 = actual/360 • 3 = actual/365 • 4 = 30/360 (BMA) • 5 = 30/360 (ISDA) • 6 = 30/360 (European) • 7 = actual/365 (Japanese) • 8 = actual/actual (ICMA) • 9 = actual/360 (ICMA) • 10 = actual/365 (ICMA) • 11 = 30/360E (ICMA) • 12 = actual/actual (ISDA)

For more information, see **basis** on page Glossary-1.

Description   [FixedSpec, ForwardDates, ForwardRates] = liborfloat2fixed(ThreeMonthRates, Settle, Tenor, StartDate, Interpolation, ConvexAdj, RateParam, InArrears, Sigma, FixedCompound, FixedBasis) computes forward rates, dates, and the swap fixed rate.

FixedSpec specifies the structure of the fixed-rate side of the swap:

- Coupon: Par-swap rate
- Settle: Start date
- Maturity: End date
- Period: Frequency of payment
- Basis: Accrual basis

ForwardDates are dates corresponding to ForwardRates (all third Wednesdays of the month, spread 3 months apart). The first element is the third Wednesday immediately after Settle.

ForwardRates are forward rates corresponding to the forward dates, quarterly compounded, and on the actual/360 basis.

Note To preserve input integrity, Tenor is rounded upward to the closest integer. Currently traded tenors are 2, 5, and 10 years.

The function assumes that floating-rate observations occur quarterly on the third Wednesday of a delivery month. The first delivery month is the month of the first third Wednesday after Settle. Floating-side payments occur on the third-month anniversaries of observation dates.

Examples   Use the supplied EDdata.xls file as input to a liborfloat2fixed computation.

liborfloat2fixed

```
[EDFutData, textdata] = xlsread('EDdata.xls');
Settle                = datenum('15-Oct-2002');
Tenor                 = 2;

[FixedSpec, ForwardDates, ForwardRates] =...
liborfloat2fixed(EDFutData(:,1:3), Settle, Tenor)

FixedSpec =

      Coupon: 0.0222
      Settle: '16-Oct-2002'
    Maturity: '16-Oct-2004'
      Period: 4
       Basis: 1

ForwardDates =

      731505
      731596
      731687
      731778
      731869
      731967
      732058
      732149

ForwardRates =

      0.0177
      0.0166
      0.0170
      0.0188
      0.0214
      0.0248
      0.0279
      0.0305
```

See Also   liborduration | liborprice

liborprice

Purpose	Price swap given swap rate

Syntax

```
Price = liborprice(ThreeMonthRates, Settle, Tenor,
SwapRate, StartDate, Interpolation, ConvexAdj, RateParam,
InArrears, Sigma, FixedCompound, FixedBasis)
```

Arguments

ThreeMonthRates	Three-month Eurodollar futures data or forward rate agreement data. (A forward rate agreement stipulates that a certain interest rate applies to a certain principal amount for a given future time period.) An n-by-3 matrix in the form of [month year IMMQuote]. The floating rate is assumed to compound quarterly and to accrue on an actual/360 basis.
Settle	Settlement date of swap. Scalar.
Tenor	Life of the swap. Scalar.
SwapRate	Swap rate in decimal.
StartDate	(Optional) Scalar value to denote reference date for valuation of (forward) swap. This in effect allows forward swap valuation. Default = Settle.
Interpolation	(Optional) Interpolation method to determine applicable forward rate for months when no Eurodollar data is available. Default is 'linear' or 1. Other possible values are 'Nearest' or 0, and 'Cubic' or 2.
ConvexAdj	(Optional) Default = 0 (off). 1 = on. Denotes whether futures/forward convexity adjustment is required. Pertains to forward rate adjustments when those rates are taken from Eurodollar futures data.

RateParam	(Optional) Short-rate model's parameters (Hull-White) [a S], where the short-rate process is:

$$dr = \left[\theta(t) - ar \right] dt + Sdz.$$

Default = [0.05 0.015].

InArrears	(Optional) Default = 0 (off). Set to 1 for on. If on, the routine does an automatic convexity adjustment to forward rates.
Sigma	(Optional) Overall annual volatility of caplets.
FixedCompound	(Optional) Scalar value. Compounding or frequency of payment on the fixed side. Also, the reset frequency. Default = 4 (quarterly). Other values are 1, 2, and 12.
FixedBasis	(Optional) Scalar value. Basis of the fixed side.

- 0 = actual/actual (default)

- 1 = 30/360 (SIA)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (BMA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)

- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)

- 10 = actual/365 (ICMA)

- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

liborprice

For more information, see **basis** on page Glossary-1.

Description   `Price = liborprice(ThreeMonthRates, Settle, Tenor, SwapRate, StartDate, Interpolation, ConvexAdj, RateParam, InArrears, Sigma, FixedCompound, FixedBasis)` computes the price per $100 notional value of a swap given the swap rate. A positive result indicates that fixed side is more valuable than the floating side.

`Price` is the present value of the difference between floating and fixed-rate sides of the swap per $100 notional.

Examples   This example shows that a swap paying the par swap rate has a value of 0.

Load the input data.

```
[EDFutData, textdata] = xlsread('EDdata.xls');
Settle = datenum('15-Oct-2002');
Tenor = 2;
```

Compute the fixed rate from the Eurodollar data.

```
FixedSpec = liborfloat2fixed(EDFutData(:,1:3), Settle, Tenor)


Coupon: 0.0222
Settle: '16-Oct-2002'
Maturity: '16-Oct-2004'
Period: 4
Basis: 1
```

Compute the price of a par swap.

```
Price = liborprice(EDFutData(:,1:3), Settle, Tenor, FixedSpec.Coupon)

Price =

  4.1633e-015
```

MATLAB computes a value for `Price` that is effectively equal to 0.

See Also   liborduration | liborfloat2fixed

mbscfamounts

Purpose   Cash flow and time mapping for mortgage pool

Syntax   `[CFlowAmounts, CFlowDates, TFactors, Factors, Payment, Principal, Interest, Prepayment] = mbscfamounts(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)`

Arguments

Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay in days. Default is 0 (no delay).
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. If you input a customized prepayment matrix, set PrepaySpeed to [].
PrepayMatrix	(Optional) Used only when PrepaySpeed is unspecified. Customized prepayment vector. A NaN-padded matrix of size max(TermRemaining)-by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS)-by-1 vectors.

Description   [CFlowAmounts, CFLowDates, TFactors, Factors, Payment, Principal, Interest, Prepayment] = mbscfamounts(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes cash flows between settle and maturity dates, the corresponding time factors in months from settle, and the mortgage factor (the fraction of loan principal outstanding).

CFlowAmounts is a vector of cash flows starting from Settle through end of the last month (Maturity).

CFlowDates indicates when cash flows occur, including at Settle. A negative number at Settle indicates accrued interest is due.

TFactors is a vector of times in months from Settle, corresponding to each cash flow.

Factors is a vector of mortgage factors (the fraction of the balance still outstanding at the end of each month).

Payment is a NMBS-by-P matrix of total monthly payment.

Principal is a NMBS-by-P matrix of principal portion of the payment

Interest is a NMBS-by-P matrix of interest portion of the payment.

Prepayment is a NMBS-by-P matrix of unscheduled payment of principal.

Examples   **Calculate Cash Flow Amounts and Dates, Time Factors, and Mortgage Factors for a Single Mortgage**

Given a mortgage with the following characteristics, compute the cash flow amounts and dates, the time factors, and the mortgage factors.

Define the mortgage characteristics.

```
Settle      = datenum('17-April-2002');
Maturity    = datenum('1-Jan-2030');
IssueDate   = datenum('1-Jan-2000');
GrossRate   = 0.08125;
CouponRate  = 0.075;
Delay       = 14;
PrepaySpeed = 100;
```

Use mbscfamonts to evaluate the mortgage.

```
[CFlowAmounts, CFLowDates, TFactors, Factors] = ...
mbscfamounts(Settle, Maturity, IssueDate, GrossRate, ...
CouponRate, Delay, PrepaySpeed)

CFlowAmounts =

  Columns 1 through 7

   -0.0033    0.0118    0.0120    0.0121    0.0120    0.0119    0.0119
  Columns 8 through 14

    0.0118    0.0117    0.0117    0.0116    0.0115    0.0115    0.0114

  Columns 15 through 21

    0.0114    0.0113    0.0112    0.0112    0.0111    0.0110    0.0110

  Columns 22 through 28

    0.0109    0.0109    0.0108    0.0107    0.0107    0.0106    0.0106

  Columns 29 through 35

    0.0105    0.0105    0.0104    0.0103    0.0103    0.0102    0.0102

  Columns 36 through 42

    0.0101    0.0101    0.0100    0.0099    0.0099    0.0098    0.0098

  Columns 43 through 49

    0.0097    0.0097    0.0096    0.0096    0.0095    0.0095    0.0094

  Columns 50 through 56
```

```
    0.0094    0.0093    0.0093    0.0092    0.0092    0.0091    0.0090

  Columns 57 through 63

    0.0090    0.0089    0.0089    0.0088    0.0088    0.0087    0.0087

  Columns 64 through 70

    0.0087    0.0086    0.0086    0.0085    0.0085    0.0084    0.0084

  Columns 71 through 77

    0.0083    0.0083    0.0082    0.0082    0.0081    0.0081    0.0080

  Columns 78 through 84

    0.0080    0.0079    0.0079    0.0079    0.0078    0.0078    0.0077

  Columns 85 through 91

    0.0077    0.0076    0.0076    0.0075    0.0075    0.0075    0.0074

  Columns 92 through 98

    0.0074    0.0073    0.0073    0.0073    0.0072    0.0072    0.0071

  Columns 99 through 105

    0.0071    0.0070    0.0070    0.0070    0.0069    0.0069    0.0068

  Columns 106 through 112

    0.0068    0.0068    0.0067    0.0067    0.0066    0.0066    0.0066

  Columns 113 through 119
```

```
      0.0065    0.0065    0.0065    0.0064    0.0064    0.0063    0.0063

Columns 120 through 126

      0.0063    0.0062    0.0062    0.0062    0.0061    0.0061    0.0061

Columns 127 through 133

      0.0060    0.0060    0.0059    0.0059    0.0059    0.0058    0.0058

Columns 134 through 140

      0.0058    0.0057    0.0057    0.0057    0.0056    0.0056    0.0056

Columns 141 through 147

      0.0055    0.0055    0.0055    0.0054    0.0054    0.0054    0.0053

Columns 148 through 154

      0.0053    0.0053    0.0052    0.0052    0.0052    0.0052    0.0051

Columns 155 through 161

      0.0051    0.0051    0.0050    0.0050    0.0050    0.0049    0.0049

Columns 162 through 168

      0.0049    0.0048    0.0048    0.0048    0.0048    0.0047    0.0047

Columns 169 through 175

      0.0047    0.0046    0.0046    0.0046    0.0046    0.0045    0.0045

Columns 176 through 182

      0.0045    0.0044    0.0044    0.0044    0.0044    0.0043    0.0043
```

```
Columns 183 through 189

  0.0043    0.0043    0.0042    0.0042    0.0042    0.0041    0.0041

Columns 190 through 196

  0.0041    0.0041    0.0040    0.0040    0.0040    0.0040    0.0039

Columns 197 through 203

  0.0039    0.0039    0.0039    0.0038    0.0038    0.0038    0.0038

Columns 204 through 210

  0.0037    0.0037    0.0037    0.0037    0.0036    0.0036    0.0036

Columns 211 through 217

  0.0036    0.0035    0.0035    0.0035    0.0035    0.0035    0.0034

Columns 218 through 224

  0.0034    0.0034    0.0034    0.0033    0.0033    0.0033    0.0033

Columns 225 through 231

  0.0033    0.0032    0.0032    0.0032    0.0032    0.0031    0.0031

Columns 232 through 238

  0.0031    0.0031    0.0031    0.0030    0.0030    0.0030    0.0030

Columns 239 through 245

  0.0030    0.0029    0.0029    0.0029    0.0029    0.0029    0.0028
```

```
Columns 246 through 252

  0.0028    0.0028    0.0028    0.0028    0.0027    0.0027    0.0027

Columns 253 through 259

  0.0027    0.0027    0.0026    0.0026    0.0026    0.0026    0.0026

Columns 260 through 266

  0.0025    0.0025    0.0025    0.0025    0.0025    0.0024    0.0024

Columns 267 through 273

  0.0024    0.0024    0.0024    0.0024    0.0023    0.0023    0.0023

Columns 274 through 280

  0.0023    0.0023    0.0023    0.0022    0.0022    0.0022    0.0022

Columns 281 through 287

  0.0022    0.0021    0.0021    0.0021    0.0021    0.0021    0.0021

Columns 288 through 294

  0.0020    0.0020    0.0020    0.0020    0.0020    0.0020    0.0020

Columns 295 through 301

  0.0019    0.0019    0.0019    0.0019    0.0019    0.0019    0.0018

Columns 302 through 308

  0.0018    0.0018    0.0018    0.0018    0.0018    0.0017    0.0017

Columns 309 through 315
```

```
      0.0017    0.0017    0.0017    0.0017    0.0017    0.0016    0.0016

  Columns 316 through 322

      0.0016    0.0016    0.0016    0.0016    0.0016    0.0015    0.0015

  Columns 323 through 329

      0.0015    0.0015    0.0015    0.0015    0.0015    0.0014    0.0014

  Columns 330 through 334

      0.0014    0.0014    0.0014    0.0014    0.0014


CFLowDates =

  Columns 1 through 6

       731323      731337      731368      731398      731429      731460

  Columns 7 through 12

       731490      731521      731551      731582      731613      731641

  Columns 13 through 18

       731672      731702      731733      731763      731794      731825

  Columns 19 through 24

       731855      731886      731916      731947      731978      732007

  Columns 25 through 30

       732038      732068      732099      732129      732160      732191
```

mbscfamounts

```
Columns 31 through 36

    732221       732252       732282       732313       732344       732372

Columns 37 through 42

    732403       732433       732464       732494       732525       732556

Columns 43 through 48

    732586       732617       732647       732678       732709       732737

Columns 49 through 54

    732768       732798       732829       732859       732890       732921

Columns 55 through 60

    732951       732982       733012       733043       733074       733102

Columns 61 through 66

    733133       733163       733194       733224       733255       733286

Columns 67 through 72

    733316       733347       733377       733408       733439       733468

Columns 73 through 78

    733499       733529       733560       733590       733621       733652

Columns 79 through 84

    733682       733713       733743       733774       733805       733833
```

```
Columns 85 through 90

   733864      733894      733925      733955      733986      734017

Columns 91 through 96

   734047      734078      734108      734139      734170      734198

Columns 97 through 102

   734229      734259      734290      734320      734351      734382

Columns 103 through 108

   734412      734443      734473      734504      734535      734563

Columns 109 through 114

   734594      734624      734655      734685      734716      734747

Columns 115 through 120

   734777      734808      734838      734869      734900      734929

Columns 121 through 126

   734960      734990      735021      735051      735082      735113

Columns 127 through 132

   735143      735174      735204      735235      735266      735294

Columns 133 through 138

   735325      735355      735386      735416      735447      735478

Columns 139 through 144
```

```
     735508      735539      735569      735600      735631      735659

Columns 145 through 150

     735690      735720      735751      735781      735812      735843

Columns 151 through 156

     735873      735904      735934      735965      735996      736024

Columns 157 through 162

     736055      736085      736116      736146      736177      736208

Columns 163 through 168

     736238      736269      736299      736330      736361      736390

Columns 169 through 174

     736421      736451      736482      736512      736543      736574

Columns 175 through 180

     736604      736635      736665      736696      736727      736755

Columns 181 through 186

     736786      736816      736847      736877      736908      736939

Columns 187 through 192

     736969      737000      737030      737061      737092      737120

Columns 193 through 198
```

```
  737151      737181      737212      737242      737273      737304

Columns 199 through 204

  737334      737365      737395      737426      737457      737485

Columns 205 through 210

  737516      737546      737577      737607      737638      737669

Columns 211 through 216

  737699      737730      737760      737791      737822      737851

Columns 217 through 222

  737882      737912      737943      737973      738004      738035

Columns 223 through 228

  738065      738096      738126      738157      738188      738216

Columns 229 through 234

  738247      738277      738308      738338      738369      738400

Columns 235 through 240

  738430      738461      738491      738522      738553      738581

Columns 241 through 246

  738612      738642      738673      738703      738734      738765

Columns 247 through 252

  738795      738826      738856      738887      738918      738946
```

```
Columns 253 through 258

    738977      739007      739038      739068      739099      739130

Columns 259 through 264

    739160      739191      739221      739252      739283      739312

Columns 265 through 270

    739343      739373      739404      739434      739465      739496

Columns 271 through 276

    739526      739557      739587      739618      739649      739677

Columns 277 through 282

    739708      739738      739769      739799      739830      739861

Columns 283 through 288

    739891      739922      739952      739983      740014      740042

Columns 289 through 294

    740073      740103      740134      740164      740195      740226

Columns 295 through 300

    740256      740287      740317      740348      740379      740407

Columns 301 through 306

    740438      740468      740499      740529      740560      740591
```

```
Columns 307 through 312

    740621      740652      740682      740713      740744      740773

Columns 313 through 318

    740804      740834      740865      740895      740926      740957

Columns 319 through 324

    740987      741018      741048      741079      741110      741138

Columns 325 through 330

    741169      741199      741230      741260      741291      741322

Columns 331 through 334

    741352      741383      741413      741444


TFactors =

  Columns 1 through 7

         0    0.9333    1.9333    2.9333    3.9333    4.9333    5.9333

  Columns 8 through 14

    6.9333    7.9333    8.9333    9.9333   10.9333   11.9333   12.9333

  Columns 15 through 21

   13.9333   14.9333   15.9333   16.9333   17.9333   18.9333   19.9333

  Columns 22 through 28
```

```
    20.9333   21.9333   22.9333   23.9333   24.9333   25.9333   26.9333

Columns 29 through 35

    27.9333   28.9333   29.9333   30.9333   31.9333   32.9333   33.9333

Columns 36 through 42

    34.9333   35.9333   36.9333   37.9333   38.9333   39.9333   40.9333

Columns 43 through 49

    41.9333   42.9333   43.9333   44.9333   45.9333   46.9333   47.9333

Columns 50 through 56

    48.9333   49.9333   50.9333   51.9333   52.9333   53.9333   54.9333

Columns 57 through 63

    55.9333   56.9333   57.9333   58.9333   59.9333   60.9333   61.9333

Columns 64 through 70

    62.9333   63.9333   64.9333   65.9333   66.9333   67.9333   68.9333

Columns 71 through 77

    69.9333   70.9333   71.9333   72.9333   73.9333   74.9333   75.9333

Columns 78 through 84

    76.9333   77.9333   78.9333   79.9333   80.9333   81.9333   82.9333

Columns 85 through 91

    83.9333   84.9333   85.9333   86.9333   87.9333   88.9333   89.9333
```

```
Columns 92 through 98

 90.9333    91.9333    92.9333    93.9333    94.9333    95.9333    96.9333

Columns 99 through 105

 97.9333    98.9333    99.9333   100.9333   101.9333   102.9333   103.9333

Columns 106 through 112

104.9333   105.9333   106.9333   107.9333   108.9333   109.9333   110.9333

Columns 113 through 119

111.9333   112.9333   113.9333   114.9333   115.9333   116.9333   117.9333

Columns 120 through 126

118.9333   119.9333   120.9333   121.9333   122.9333   123.9333   124.9333

Columns 127 through 133

125.9333   126.9333   127.9333   128.9333   129.9333   130.9333   131.9333

Columns 134 through 140

132.9333   133.9333   134.9333   135.9333   136.9333   137.9333   138.9333

Columns 141 through 147

139.9333   140.9333   141.9333   142.9333   143.9333   144.9333   145.9333

Columns 148 through 154

146.9333   147.9333   148.9333   149.9333   150.9333   151.9333   152.9333
```

```
Columns 155 through 161

153.9333   154.9333   155.9333   156.9333   157.9333   158.9333   159.9333

Columns 162 through 168

160.9333   161.9333   162.9333   163.9333   164.9333   165.9333   166.9333

Columns 169 through 175

167.9333   168.9333   169.9333   170.9333   171.9333   172.9333   173.9333

Columns 176 through 182

174.9333   175.9333   176.9333   177.9333   178.9333   179.9333   180.9333

Columns 183 through 189

181.9333   182.9333   183.9333   184.9333   185.9333   186.9333   187.9333

Columns 190 through 196

188.9333   189.9333   190.9333   191.9333   192.9333   193.9333   194.9333

Columns 197 through 203

195.9333   196.9333   197.9333   198.9333   199.9333   200.9333   201.9333

Columns 204 through 210

202.9333   203.9333   204.9333   205.9333   206.9333   207.9333   208.9333

Columns 211 through 217

209.9333   210.9333   211.9333   212.9333   213.9333   214.9333   215.9333

Columns 218 through 224
```

```
216.9333  217.9333  218.9333  219.9333  220.9333  221.9333  222.9333

Columns 225 through 231

223.9333  224.9333  225.9333  226.9333  227.9333  228.9333  229.9333

Columns 232 through 238

230.9333  231.9333  232.9333  233.9333  234.9333  235.9333  236.9333

Columns 239 through 245

237.9333  238.9333  239.9333  240.9333  241.9333  242.9333  243.9333

Columns 246 through 252

244.9333  245.9333  246.9333  247.9333  248.9333  249.9333  250.9333

Columns 253 through 259

251.9333  252.9333  253.9333  254.9333  255.9333  256.9333  257.9333

Columns 260 through 266

258.9333  259.9333  260.9333  261.9333  262.9333  263.9333  264.9333

Columns 267 through 273

265.9333  266.9333  267.9333  268.9333  269.9333  270.9333  271.9333

Columns 274 through 280

272.9333  273.9333  274.9333  275.9333  276.9333  277.9333  278.9333

Columns 281 through 287
```

mbscfamounts

```
        279.9333   280.9333   281.9333   282.9333   283.9333   284.9333   285.9333

   Columns 288 through 294

        286.9333   287.9333   288.9333   289.9333   290.9333   291.9333   292.9333

   Columns 295 through 301

        293.9333   294.9333   295.9333   296.9333   297.9333   298.9333   299.9333

   Columns 302 through 308

        300.9333   301.9333   302.9333   303.9333   304.9333   305.9333   306.9333

   Columns 309 through 315

        307.9333   308.9333   309.9333   310.9333   311.9333   312.9333   313.9333

   Columns 316 through 322

        314.9333   315.9333   316.9333   317.9333   318.9333   319.9333   320.9333

   Columns 323 through 329

        321.9333   322.9333   323.9333   324.9333   325.9333   326.9333   327.9333

   Columns 330 through 334

        328.9333   329.9333   330.9333   331.9333   332.9333


Factors =

  Columns 1 through 7

     1.0000    0.9944    0.9887    0.9828    0.9769    0.9711    0.9653
```

```
Columns 8 through 14

  0.9595    0.9538    0.9481    0.9424    0.9368    0.9311    0.9255

Columns 15 through 21

  0.9199    0.9144    0.9089    0.9034    0.8979    0.8925    0.8871

Columns 22 through 28

  0.8817    0.8763    0.8710    0.8657    0.8604    0.8552    0.8499

Columns 29 through 35

  0.8447    0.8396    0.8344    0.8293    0.8242    0.8191    0.8140

Columns 36 through 42

  0.8090    0.8040    0.7990    0.7941    0.7892    0.7842    0.7794

Columns 43 through 49

  0.7745    0.7697    0.7649    0.7601    0.7553    0.7506    0.7458

Columns 50 through 56

  0.7411    0.7365    0.7318    0.7272    0.7226    0.7180    0.7134

Columns 57 through 63

  0.7089    0.7044    0.6999    0.6954    0.6910    0.6865    0.6821

Columns 64 through 70

  0.6777    0.6734    0.6690    0.6647    0.6604    0.6561    0.6519

Columns 71 through 77
```

```
      0.6476    0.6434    0.6392    0.6350    0.6309    0.6267    0.6226

Columns 78 through 84

      0.6185    0.6144    0.6104    0.6063    0.6023    0.5983    0.5943

Columns 85 through 91

      0.5903    0.5864    0.5825    0.5785    0.5747    0.5708    0.5669

Columns 92 through 98

      0.5631    0.5593    0.5555    0.5517    0.5479    0.5442    0.5405

Columns 99 through 105

      0.5368    0.5331    0.5294    0.5257    0.5221    0.5185    0.5149

Columns 106 through 112

      0.5113    0.5077    0.5042    0.5006    0.4971    0.4936    0.4901

Columns 113 through 119

      0.4866    0.4832    0.4797    0.4763    0.4729    0.4695    0.4661

Columns 120 through 126

      0.4628    0.4594    0.4561    0.4528    0.4495    0.4462    0.4430

Columns 127 through 133

      0.4397    0.4365    0.4333    0.4301    0.4269    0.4237    0.4205

Columns 134 through 140
```

```
  0.4174    0.4143    0.4111    0.4080    0.4049    0.4019    0.3988

Columns 141 through 147

  0.3958    0.3927    0.3897    0.3867    0.3837    0.3808    0.3778

Columns 148 through 154

  0.3748    0.3719    0.3690    0.3661    0.3632    0.3603    0.3574

Columns 155 through 161

  0.3546    0.3517    0.3489    0.3461    0.3433    0.3405    0.3377

Columns 162 through 168

  0.3350    0.3322    0.3295    0.3267    0.3240    0.3213    0.3186

Columns 169 through 175

  0.3160    0.3133    0.3106    0.3080    0.3054    0.3027    0.3001

Columns 176 through 182

  0.2975    0.2950    0.2924    0.2898    0.2873    0.2847    0.2822

Columns 183 through 189

  0.2797    0.2772    0.2747    0.2722    0.2698    0.2673    0.2649

Columns 190 through 196

  0.2624    0.2600    0.2576    0.2552    0.2528    0.2504    0.2480

Columns 197 through 203

  0.2457    0.2433    0.2410    0.2386    0.2363    0.2340    0.2317
```

```
Columns 204 through 210

  0.2294    0.2271    0.2249    0.2226    0.2204    0.2181    0.2159

Columns 211 through 217

  0.2137    0.2115    0.2093    0.2071    0.2049    0.2027    0.2005

Columns 218 through 224

  0.1984    0.1962    0.1941    0.1920    0.1899    0.1877    0.1856

Columns 225 through 231

  0.1836    0.1815    0.1794    0.1773    0.1753    0.1732    0.1712

Columns 232 through 238

  0.1692    0.1671    0.1651    0.1631    0.1611    0.1591    0.1572

Columns 239 through 245

  0.1552    0.1532    0.1513    0.1493    0.1474    0.1455    0.1436

Columns 246 through 252

  0.1416    0.1397    0.1378    0.1359    0.1341    0.1322    0.1303

Columns 253 through 259

  0.1285    0.1266    0.1248    0.1229    0.1211    0.1193    0.1175

Columns 260 through 266

  0.1157    0.1139    0.1121    0.1103    0.1085    0.1068    0.1050
```

```
Columns 267 through 273

  0.1032    0.1015    0.0998    0.0980    0.0963    0.0946    0.0929

Columns 274 through 280

  0.0912    0.0895    0.0878    0.0861    0.0844    0.0827    0.0811

Columns 281 through 287

  0.0794    0.0778    0.0761    0.0745    0.0728    0.0712    0.0696

Columns 288 through 294

  0.0680    0.0664    0.0648    0.0632    0.0616    0.0600    0.0584

Columns 295 through 301

  0.0568    0.0553    0.0537    0.0522    0.0506    0.0491    0.0476

Columns 302 through 308

  0.0460    0.0445    0.0430    0.0415    0.0400    0.0385    0.0370

Columns 309 through 315

  0.0355    0.0340    0.0325    0.0311    0.0296    0.0281    0.0267

Columns 316 through 322

  0.0252    0.0238    0.0223    0.0209    0.0195    0.0180    0.0166

Columns 323 through 329

  0.0152    0.0138    0.0124    0.0110    0.0096    0.0082    0.0068

Columns 330 through 334
```

```
    0.0055    0.0041    0.0027    0.0014         0
```

The result is contained in four 334-element row vectors.

Compute Cash Flow Amounts and Dates, Time Factors, and Mortgage Factors for a Mortgage Portfolio

Given a portfolio of mortgage-backed securities, use `mbscfamounts` to compute the cash flows and other factors from the portfolio.

Define characteristics for a mortgage portfolio.

```
Settle     = datenum(['13-Jan-2000';'17-Apr-2002';'17-May-2002']);
Maturity   = datenum('1-Jan-2030');
IssueDate  = datenum('1-Jan-2000');
GrossRate  = 0.08125;
CouponRate = [0.075; 0.07875; 0.0775];
Delay      = 14;
PrepaySpeed = 100;
```

Use `mbscfamonts` to evaluate the mortgage.

```
[CFlowAmounts, CFlowDates, TFactors, Factors] = ...
mbscfamounts(Settle, Maturity, IssueDate, GrossRate, ...
CouponRate, Delay, PrepaySpeed)

CFlowAmounts =

  Columns 1 through 10

  -0.0033    0.0118    0.0120    0.0121    0.0120    0.0119    0.0119    0.0118    0.0117    0.0117
  Columns 11 through 20

   0.0116    0.0115    0.0115    0.0114    0.0114    0.0113    0.0112    0.0112    0.0111    0.0110


  Columns 21 through 30
```

```
    0.0110    0.0109    0.0109    0.0108    0.0107    0.0107    0.0106    0.0106    0.0105    0.0105

Columns 31 through 40

    0.0104    0.0103    0.0103    0.0102    0.0102    0.0101    0.0101    0.0100    0.0099    0.0099

Columns 41 through 50

    0.0098    0.0098    0.0097    0.0097    0.0096    0.0096    0.0095    0.0095    0.0094    0.0094

Columns 51 through 60

    0.0093    0.0093    0.0092    0.0092    0.0091    0.0090    0.0090    0.0089    0.0089    0.0088

Columns 61 through 70

    0.0088    0.0087    0.0087    0.0087    0.0086    0.0086    0.0085    0.0085    0.0084    0.0084

Columns 71 through 80

    0.0083    0.0083    0.0082    0.0082    0.0081    0.0081    0.0080    0.0080    0.0079    0.0079

Columns 81 through 90

    0.0079    0.0078    0.0078    0.0077    0.0077    0.0076    0.0076    0.0075    0.0075    0.0075

Columns 91 through 100

    0.0074    0.0074    0.0073    0.0073    0.0073    0.0072    0.0072    0.0071    0.0071    0.0070

Columns 101 through 110

    0.0070    0.0070    0.0069    0.0069    0.0068    0.0068    0.0068    0.0067    0.0067    0.0066

Columns 111 through 120

    0.0066    0.0066    0.0065    0.0065    0.0065    0.0064    0.0064    0.0063    0.0063    0.0063
```

```
Columns 121 through 130

 0.0062    0.0062    0.0062    0.0061    0.0061    0.0061    0.0060    0.0060    0.0059    0.0059

Columns 131 through 140

 0.0059    0.0058    0.0058    0.0058    0.0057    0.0057    0.0057    0.0056    0.0056    0.0056

Columns 141 through 150

 0.0055    0.0055    0.0055    0.0054    0.0054    0.0054    0.0053    0.0053    0.0053    0.0052

Columns 151 through 160

 0.0052    0.0052    0.0052    0.0051    0.0051    0.0051    0.0050    0.0050    0.0050    0.0049

Columns 161 through 170

 0.0049    0.0049    0.0048    0.0048    0.0048    0.0048    0.0047    0.0047    0.0047    0.0046

Columns 171 through 180

 0.0046    0.0046    0.0046    0.0045    0.0045    0.0045    0.0044    0.0044    0.0044    0.0044

Columns 181 through 190

 0.0043    0.0043    0.0043    0.0043    0.0042    0.0042    0.0042    0.0041    0.0041    0.0041

Columns 191 through 200

 0.0041    0.0040    0.0040    0.0040    0.0040    0.0039    0.0039    0.0039    0.0039    0.0038

Columns 201 through 210

 0.0038    0.0038    0.0038    0.0037    0.0037    0.0037    0.0037    0.0036    0.0036    0.0036
```

```
Columns 211 through 220

 0.0036    0.0035    0.0035    0.0035    0.0035    0.0035    0.0034    0.0034    0.0034    0.0034

Columns 221 through 230

 0.0033    0.0033    0.0033    0.0033    0.0033    0.0032    0.0032    0.0032    0.0032    0.0031

Columns 231 through 240

 0.0031    0.0031    0.0031    0.0031    0.0030    0.0030    0.0030    0.0030    0.0030    0.0029

Columns 241 through 250

 0.0029    0.0029    0.0029    0.0029    0.0028    0.0028    0.0028    0.0028    0.0028    0.0027

Columns 251 through 260

 0.0027    0.0027    0.0027    0.0027    0.0026    0.0026    0.0026    0.0026    0.0026    0.0025

Columns 261 through 270

 0.0025    0.0025    0.0025    0.0025    0.0024    0.0024    0.0024    0.0024    0.0024    0.0024

Columns 271 through 280

 0.0023    0.0023    0.0023    0.0023    0.0023    0.0023    0.0022    0.0022    0.0022    0.0022

Columns 281 through 290

 0.0022    0.0021    0.0021    0.0021    0.0021    0.0021    0.0021    0.0020    0.0020    0.0020

Columns 291 through 300

 0.0020    0.0020    0.0020    0.0020    0.0019    0.0019    0.0019    0.0019    0.0019    0.0019

Columns 301 through 310
```

```
   0.0018    0.0018    0.0018    0.0018    0.0018    0.0018    0.0017    0.0017    0.0017    0.0017

  Columns 311 through 320

   0.0017    0.0017    0.0017    0.0016    0.0016    0.0016    0.0016    0.0016    0.0016    0.0016

  Columns 321 through 330

   0.0015    0.0015    0.0015    0.0015    0.0015    0.0015    0.0015    0.0014    0.0014    0.0014

  Columns 331 through 334

   0.0014    0.0014    0.0014    0.0014


CFLowDates =

  Columns 1 through 8

     731323       731337       731368       731398       731429       731460       731490       731521

  Columns 9 through 16

     731551       731582       731613       731641       731672       731702       731733       731763

  Columns 17 through 24

     731794       731825       731855       731886       731916       731947       731978       732007

  Columns 25 through 32

     732038       732068       732099       732129       732160       732191       732221       732252

  Columns 33 through 40

     732282       732313       732344       732372       732403       732433       732464       732494
```

```
Columns 41 through 48

   732525      732556      732586      732617      732647      732678      732709      732737

Columns 49 through 56

   732768      732798      732829      732859      732890      732921      732951      732982

Columns 57 through 64

   733012      733043      733074      733102      733133      733163      733194      733224

Columns 65 through 72

   733255      733286      733316      733347      733377      733408      733439      733468

Columns 73 through 80

   733499      733529      733560      733590      733621      733652      733682      733713

Columns 81 through 88

   733743      733774      733805      733833      733864      733894      733925      733955

Columns 89 through 96

   733986      734017      734047      734078      734108      734139      734170      734198

Columns 97 through 104

   734229      734259      734290      734320      734351      734382      734412      734443

Columns 105 through 112

   734473      734504      734535      734563      734594      734624      734655      734685
```

```
Columns 113 through 120

   734716      734747      734777      734808      734838      734869      734900      734929

Columns 121 through 128

   734960      734990      735021      735051      735082      735113      735143      735174

Columns 129 through 136

   735204      735235      735266      735294      735325      735355      735386      735416

Columns 137 through 144

   735447      735478      735508      735539      735569      735600      735631      735659

Columns 145 through 152

   735690      735720      735751      735781      735812      735843      735873      735904

Columns 153 through 160

   735934      735965      735996      736024      736055      736085      736116      736146

Columns 161 through 168

   736177      736208      736238      736269      736299      736330      736361      736390

Columns 169 through 176

   736421      736451      736482      736512      736543      736574      736604      736635

Columns 177 through 184

   736665      736696      736727      736755      736786      736816      736847      736877

Columns 185 through 192
```

```
     736908        736939        736969        737000        737030        737061        737092        737120
```

Columns 193 through 200

```
     737151        737181        737212        737242        737273        737304        737334        737365
```

Columns 201 through 208

```
     737395        737426        737457        737485        737516        737546        737577        737607
```

Columns 209 through 216

```
     737638        737669        737699        737730        737760        737791        737822        737851
```

Columns 217 through 224

```
     737882        737912        737943        737973        738004        738035        738065        738096
```

Columns 225 through 232

```
     738126        738157        738188        738216        738247        738277        738308        738338
```

Columns 233 through 240

```
     738369        738400        738430        738461        738491        738522        738553        738581
```

Columns 241 through 248

```
     738612        738642        738673        738703        738734        738765        738795        738826
```

Columns 249 through 256

```
     738856        738887        738918        738946        738977        739007        739038        739068
```

Columns 257 through 264

```
   739099      739130      739160      739191      739221      739252      739283      739312

Columns 265 through 272

   739343      739373      739404      739434      739465      739496      739526      739557

Columns 273 through 280

   739587      739618      739649      739677      739708      739738      739769      739799

Columns 281 through 288

   739830      739861      739891      739922      739952      739983      740014      740042

Columns 289 through 296

   740073      740103      740134      740164      740195      740226      740256      740287

Columns 297 through 304

   740317      740348      740379      740407      740438      740468      740499      740529

Columns 305 through 312

   740560      740591      740621      740652      740682      740713      740744      740773

Columns 313 through 320

   740804      740834      740865      740895      740926      740957      740987      741018

Columns 321 through 328

   741048      741079      741110      741138      741169      741199      741230      741260

Columns 329 through 334

   741291      741322      741352      741383      741413      741444
```

```
TFactors =

  Columns 1 through 10

       0    0.9333    1.9333    2.9333    3.9333    4.9333    5.9333    6.9333    7.9333    8.9333

  Columns 11 through 20

  9.9333   10.9333   11.9333   12.9333   13.9333   14.9333   15.9333   16.9333   17.9333   18.9333

  Columns 21 through 30

 19.9333   20.9333   21.9333   22.9333   23.9333   24.9333   25.9333   26.9333   27.9333   28.9333

  Columns 31 through 40

 29.9333   30.9333   31.9333   32.9333   33.9333   34.9333   35.9333   36.9333   37.9333   38.9333

  Columns 41 through 50

 39.9333   40.9333   41.9333   42.9333   43.9333   44.9333   45.9333   46.9333   47.9333   48.9333

  Columns 51 through 60

 49.9333   50.9333   51.9333   52.9333   53.9333   54.9333   55.9333   56.9333   57.9333   58.9333

  Columns 61 through 70

 59.9333   60.9333   61.9333   62.9333   63.9333   64.9333   65.9333   66.9333   67.9333   68.9333

  Columns 71 through 80

 69.9333   70.9333   71.9333   72.9333   73.9333   74.9333   75.9333   76.9333   77.9333   78.9333

  Columns 81 through 90
```

```
        79.9333    80.9333    81.9333    82.9333    83.9333    84.9333    85.9333    86.9333    87.9333    88.9333

  Columns 91 through 100

        89.9333    90.9333    91.9333    92.9333    93.9333    94.9333    95.9333    96.9333    97.9333    98.9333

  Columns 101 through 110

  99.9333  100.9333  101.9333  102.9333  103.9333  104.9333  105.9333  106.9333  107.9333  108.9333

  Columns 111 through 120

109.9333  110.9333  111.9333  112.9333  113.9333  114.9333  115.9333  116.9333  117.9333  118.9333

  Columns 121 through 130

119.9333  120.9333  121.9333  122.9333  123.9333  124.9333  125.9333  126.9333  127.9333  128.9333

  Columns 131 through 140

129.9333  130.9333  131.9333  132.9333  133.9333  134.9333  135.9333  136.9333  137.9333  138.9333

  Columns 141 through 150

139.9333  140.9333  141.9333  142.9333  143.9333  144.9333  145.9333  146.9333  147.9333  148.9333

  Columns 151 through 160

149.9333  150.9333  151.9333  152.9333  153.9333  154.9333  155.9333  156.9333  157.9333  158.9333

  Columns 161 through 170

159.9333  160.9333  161.9333  162.9333  163.9333  164.9333  165.9333  166.9333  167.9333  168.9333

  Columns 171 through 180
```

```
169.9333  170.9333  171.9333  172.9333  173.9333  174.9333  175.9333  176.9333  177.9333  178.9333

  Columns 181 through 190

179.9333  180.9333  181.9333  182.9333  183.9333  184.9333  185.9333  186.9333  187.9333  188.9333

  Columns 191 through 200

189.9333  190.9333  191.9333  192.9333  193.9333  194.9333  195.9333  196.9333  197.9333  198.9333

  Columns 201 through 210

199.9333  200.9333  201.9333  202.9333  203.9333  204.9333  205.9333  206.9333  207.9333  208.9333

  Columns 211 through 220

209.9333  210.9333  211.9333  212.9333  213.9333  214.9333  215.9333  216.9333  217.9333  218.9333

  Columns 221 through 230

219.9333  220.9333  221.9333  222.9333  223.9333  224.9333  225.9333  226.9333  227.9333  228.9333

  Columns 231 through 240

229.9333  230.9333  231.9333  232.9333  233.9333  234.9333  235.9333  236.9333  237.9333  238.9333

  Columns 241 through 250

239.9333  240.9333  241.9333  242.9333  243.9333  244.9333  245.9333  246.9333  247.9333  248.9333

  Columns 251 through 260

249.9333  250.9333  251.9333  252.9333  253.9333  254.9333  255.9333  256.9333  257.9333  258.9333

  Columns 261 through 270

259.9333  260.9333  261.9333  262.9333  263.9333  264.9333  265.9333  266.9333  267.9333  268.9333
```

```
 Columns 271 through 280

269.9333  270.9333  271.9333  272.9333  273.9333  274.9333  275.9333  276.9333  277.9333  278.9333

 Columns 281 through 290

279.9333  280.9333  281.9333  282.9333  283.9333  284.9333  285.9333  286.9333  287.9333  288.9333

 Columns 291 through 300

289.9333  290.9333  291.9333  292.9333  293.9333  294.9333  295.9333  296.9333  297.9333  298.9333

 Columns 301 through 310

299.9333  300.9333  301.9333  302.9333  303.9333  304.9333  305.9333  306.9333  307.9333  308.9333

 Columns 311 through 320

309.9333  310.9333  311.9333  312.9333  313.9333  314.9333  315.9333  316.9333  317.9333  318.9333

 Columns 321 through 330

319.9333  320.9333  321.9333  322.9333  323.9333  324.9333  325.9333  326.9333  327.9333  328.9333

 Columns 331 through 334

329.9333  330.9333  331.9333  332.9333


Factors =

 Columns 1 through 10

   1.0000   0.9944   0.9887   0.9828   0.9769   0.9711   0.9653   0.9595   0.9538   0.9481

 Columns 11 through 20
```

```
    0.9424    0.9368    0.9311    0.9255    0.9199    0.9144    0.9089    0.9034    0.8979    0.8925

  Columns 21 through 30

    0.8871    0.8817    0.8763    0.8710    0.8657    0.8604    0.8552    0.8499    0.8447    0.8396

  Columns 31 through 40

    0.8344    0.8293    0.8242    0.8191    0.8140    0.8090    0.8040    0.7990    0.7941    0.7892

  Columns 41 through 50

    0.7842    0.7794    0.7745    0.7697    0.7649    0.7601    0.7553    0.7506    0.7458    0.7411

  Columns 51 through 60

    0.7365    0.7318    0.7272    0.7226    0.7180    0.7134    0.7089    0.7044    0.6999    0.6954

  Columns 61 through 70

    0.6910    0.6865    0.6821    0.6777    0.6734    0.6690    0.6647    0.6604    0.6561    0.6519

  Columns 71 through 80

    0.6476    0.6434    0.6392    0.6350    0.6309    0.6267    0.6226    0.6185    0.6144    0.6104

  Columns 81 through 90

    0.6063    0.6023    0.5983    0.5943    0.5903    0.5864    0.5825    0.5785    0.5747    0.5708

  Columns 91 through 100

    0.5669    0.5631    0.5593    0.5555    0.5517    0.5479    0.5442    0.5405    0.5368    0.5331

  Columns 101 through 110
```

```
    0.5294    0.5257    0.5221    0.5185    0.5149    0.5113    0.5077    0.5042    0.5006    0.4971

  Columns 111 through 120

    0.4936    0.4901    0.4866    0.4832    0.4797    0.4763    0.4729    0.4695    0.4661    0.4628

  Columns 121 through 130

    0.4594    0.4561    0.4528    0.4495    0.4462    0.4430    0.4397    0.4365    0.4333    0.4301

  Columns 131 through 140

    0.4269    0.4237    0.4205    0.4174    0.4143    0.4111    0.4080    0.4049    0.4019    0.3988

  Columns 141 through 150

    0.3958    0.3927    0.3897    0.3867    0.3837    0.3808    0.3778    0.3748    0.3719    0.3690

  Columns 151 through 160

    0.3661    0.3632    0.3603    0.3574    0.3546    0.3517    0.3489    0.3461    0.3433    0.3405

  Columns 161 through 170

    0.3377    0.3350    0.3322    0.3295    0.3267    0.3240    0.3213    0.3186    0.3160    0.3133

  Columns 171 through 180

    0.3106    0.3080    0.3054    0.3027    0.3001    0.2975    0.2950    0.2924    0.2898    0.2873

  Columns 181 through 190

    0.2847    0.2822    0.2797    0.2772    0.2747    0.2722    0.2698    0.2673    0.2649    0.2624

  Columns 191 through 200

    0.2600    0.2576    0.2552    0.2528    0.2504    0.2480    0.2457    0.2433    0.2410    0.2386
```

```
Columns 201 through 210

  0.2363    0.2340    0.2317    0.2294    0.2271    0.2249    0.2226    0.2204    0.2181    0.2159

Columns 211 through 220

  0.2137    0.2115    0.2093    0.2071    0.2049    0.2027    0.2005    0.1984    0.1962    0.1941

Columns 221 through 230

  0.1920    0.1899    0.1877    0.1856    0.1836    0.1815    0.1794    0.1773    0.1753    0.1732

Columns 231 through 240

  0.1712    0.1692    0.1671    0.1651    0.1631    0.1611    0.1591    0.1572    0.1552    0.1532

Columns 241 through 250

  0.1513    0.1493    0.1474    0.1455    0.1436    0.1416    0.1397    0.1378    0.1359    0.1341

Columns 251 through 260

  0.1322    0.1303    0.1285    0.1266    0.1248    0.1229    0.1211    0.1193    0.1175    0.1157

Columns 261 through 270

  0.1139    0.1121    0.1103    0.1085    0.1068    0.1050    0.1032    0.1015    0.0998    0.0980

Columns 271 through 280

  0.0963    0.0946    0.0929    0.0912    0.0895    0.0878    0.0861    0.0844    0.0827    0.0811

Columns 281 through 290

  0.0794    0.0778    0.0761    0.0745    0.0728    0.0712    0.0696    0.0680    0.0664    0.0648
```

mbscfamounts

```
Columns 291 through 300

 0.0632    0.0616    0.0600    0.0584    0.0568    0.0553    0.0537    0.0522    0.0506    0.0491

Columns 301 through 310

 0.0476    0.0460    0.0445    0.0430    0.0415    0.0400    0.0385    0.0370    0.0355    0.0340

Columns 311 through 320

 0.0325    0.0311    0.0296    0.0281    0.0267    0.0252    0.0238    0.0223    0.0209    0.0195

Columns 321 through 330

 0.0180    0.0166    0.0152    0.0138    0.0124    0.0110    0.0096    0.0082    0.0068    0.0055

Columns 331 through 334

 0.0041    0.0027    0.0014         0
```

Each output is a 3-by-361 element matrix padded with NaNs wherever
elements are missing.

Calculate Payment, Principal, Interest, and Prepayment for a Single Mortgage

Given a mortgage with the following characteristics, compute payments,
principal, interest, and prepayment.

Define the mortgage characteristics.

```
Settle      = datenum('17-April-2002');
Maturity    = datenum('1-Jan-2030');
IssueDate   = datenum('1-Jan-2000');
GrossRate   = 0.08125;
CouponRate  = 0.075;
Delay       = 14;
PrepaySpeed = 100;
```

Use `mbscfamonts` to evaluate the mortgage.

```
[Payment, Principal, Interest, Prepayment] = ...
mbscfamounts(Settle, Maturity, IssueDate, GrossRate, ...
CouponRate, Delay, PrepaySpeed)
```

```
Payment =

  Columns 1 through 8

   -0.0033    0.0118    0.0120    0.0121    0.0120    0.0119    0.0119    0.0118

  Columns 9 through 16

    0.0117    0.0117    0.0116    0.0115    0.0115    0.0114    0.0114    0.0113

  Columns 17 through 24

    0.0112    0.0112    0.0111    0.0110    0.0110    0.0109    0.0109    0.0108

  Columns 25 through 32

    0.0107    0.0107    0.0106    0.0106    0.0105    0.0105    0.0104    0.0103

  Columns 33 through 40

    0.0103    0.0102    0.0102    0.0101    0.0101    0.0100    0.0099    0.0099

  Columns 41 through 48

    0.0098    0.0098    0.0097    0.0097    0.0096    0.0096    0.0095    0.0095

  Columns 49 through 56

    0.0094    0.0094    0.0093    0.0093    0.0092    0.0092    0.0091    0.0090
```

```
Columns 57 through 64

  0.0090    0.0089    0.0089    0.0088    0.0088    0.0087    0.0087    0.0087

Columns 65 through 72

  0.0086    0.0086    0.0085    0.0085    0.0084    0.0084    0.0083    0.0083

Columns 73 through 80

  0.0082    0.0082    0.0081    0.0081    0.0080    0.0080    0.0079    0.0079

Columns 81 through 88

  0.0079    0.0078    0.0078    0.0077    0.0077    0.0076    0.0076    0.0075

Columns 89 through 96

  0.0075    0.0075    0.0074    0.0074    0.0073    0.0073    0.0073    0.0072

Columns 97 through 104

  0.0072    0.0071    0.0071    0.0070    0.0070    0.0070    0.0069    0.0069

Columns 105 through 112

  0.0068    0.0068    0.0068    0.0067    0.0067    0.0066    0.0066    0.0066

Columns 113 through 120

  0.0065    0.0065    0.0065    0.0064    0.0064    0.0063    0.0063    0.0063

Columns 121 through 128

  0.0062    0.0062    0.0062    0.0061    0.0061    0.0061    0.0060    0.0060

Columns 129 through 136
```

```
   0.0059    0.0059    0.0059    0.0058    0.0058    0.0058    0.0057    0.0057

Columns 137 through 144

   0.0057    0.0056    0.0056    0.0056    0.0055    0.0055    0.0055    0.0054

Columns 145 through 152

   0.0054    0.0054    0.0053    0.0053    0.0053    0.0052    0.0052    0.0052

Columns 153 through 160

   0.0052    0.0051    0.0051    0.0051    0.0050    0.0050    0.0050    0.0049

Columns 161 through 168

   0.0049    0.0049    0.0048    0.0048    0.0048    0.0048    0.0047    0.0047

Columns 169 through 176

   0.0047    0.0046    0.0046    0.0046    0.0046    0.0045    0.0045    0.0045

Columns 177 through 184

   0.0044    0.0044    0.0044    0.0044    0.0043    0.0043    0.0043    0.0043

Columns 185 through 192

   0.0042    0.0042    0.0042    0.0041    0.0041    0.0041    0.0041    0.0040

Columns 193 through 200

   0.0040    0.0040    0.0040    0.0039    0.0039    0.0039    0.0039    0.0038

Columns 201 through 208
```

```
     0.0038    0.0038    0.0038    0.0037    0.0037    0.0037    0.0037    0.0036

  Columns 209 through 216

     0.0036    0.0036    0.0036    0.0035    0.0035    0.0035    0.0035    0.0035

  Columns 217 through 224

     0.0034    0.0034    0.0034    0.0034    0.0033    0.0033    0.0033    0.0033

  Columns 225 through 232

     0.0033    0.0032    0.0032    0.0032    0.0032    0.0031    0.0031    0.0031

  Columns 233 through 240

     0.0031    0.0031    0.0030    0.0030    0.0030    0.0030    0.0030    0.0029

  Columns 241 through 248

     0.0029    0.0029    0.0029    0.0029    0.0028    0.0028    0.0028    0.0028

  Columns 249 through 256

     0.0028    0.0027    0.0027    0.0027    0.0027    0.0027    0.0026    0.0026

  Columns 257 through 264

     0.0026    0.0026    0.0026    0.0025    0.0025    0.0025    0.0025    0.0025

  Columns 265 through 272

     0.0024    0.0024    0.0024    0.0024    0.0024    0.0024    0.0023    0.0023

  Columns 273 through 280

     0.0023    0.0023    0.0023    0.0023    0.0022    0.0022    0.0022    0.0022
```

```
Columns 281 through 288

  0.0022    0.0021    0.0021    0.0021    0.0021    0.0021    0.0021    0.0020

Columns 289 through 296

  0.0020    0.0020    0.0020    0.0020    0.0020    0.0020    0.0019    0.0019

Columns 297 through 304

  0.0019    0.0019    0.0019    0.0019    0.0018    0.0018    0.0018    0.0018

Columns 305 through 312

  0.0018    0.0018    0.0017    0.0017    0.0017    0.0017    0.0017    0.0017

Columns 313 through 320

  0.0017    0.0016    0.0016    0.0016    0.0016    0.0016    0.0016    0.0016

Columns 321 through 328

  0.0015    0.0015    0.0015    0.0015    0.0015    0.0015    0.0015    0.0014

Columns 329 through 334

  0.0014    0.0014    0.0014    0.0014    0.0014    0.0014


Principal =

  Columns 1 through 6

      731323      731337      731368      731398      731429      731460

  Columns 7 through 12
```

```
    731490      731521      731551      731582      731613      731641

Columns 13 through 18

    731672      731702      731733      731763      731794      731825

Columns 19 through 24

    731855      731886      731916      731947      731978      732007

Columns 25 through 30

    732038      732068      732099      732129      732160      732191

Columns 31 through 36

    732221      732252      732282      732313      732344      732372

Columns 37 through 42

    732403      732433      732464      732494      732525      732556

Columns 43 through 48

    732586      732617      732647      732678      732709      732737

Columns 49 through 54

    732768      732798      732829      732859      732890      732921

Columns 55 through 60

    732951      732982      733012      733043      733074      733102

Columns 61 through 66
```

```
   733133      733163      733194      733224      733255      733286

Columns 67 through 72

   733316      733347      733377      733408      733439      733468

Columns 73 through 78

   733499      733529      733560      733590      733621      733652

Columns 79 through 84

   733682      733713      733743      733774      733805      733833

Columns 85 through 90

   733864      733894      733925      733955      733986      734017

Columns 91 through 96

   734047      734078      734108      734139      734170      734198

Columns 97 through 102

   734229      734259      734290      734320      734351      734382

Columns 103 through 108

   734412      734443      734473      734504      734535      734563

Columns 109 through 114

   734594      734624      734655      734685      734716      734747

Columns 115 through 120

   734777      734808      734838      734869      734900      734929
```

```
Columns 121 through 126

   734960      734990      735021      735051      735082      735113

Columns 127 through 132

   735143      735174      735204      735235      735266      735294

Columns 133 through 138

   735325      735355      735386      735416      735447      735478

Columns 139 through 144

   735508      735539      735569      735600      735631      735659

Columns 145 through 150

   735690      735720      735751      735781      735812      735843

Columns 151 through 156

   735873      735904      735934      735965      735996      736024

Columns 157 through 162

   736055      736085      736116      736146      736177      736208

Columns 163 through 168

   736238      736269      736299      736330      736361      736390

Columns 169 through 174

   736421      736451      736482      736512      736543      736574
```

```
Columns 175 through 180

   736604      736635      736665      736696      736727      736755

Columns 181 through 186

   736786      736816      736847      736877      736908      736939

Columns 187 through 192

   736969      737000      737030      737061      737092      737120

Columns 193 through 198

   737151      737181      737212      737242      737273      737304

Columns 199 through 204

   737334      737365      737395      737426      737457      737485

Columns 205 through 210

   737516      737546      737577      737607      737638      737669

Columns 211 through 216

   737699      737730      737760      737791      737822      737851

Columns 217 through 222

   737882      737912      737943      737973      738004      738035

Columns 223 through 228

   738065      738096      738126      738157      738188      738216

Columns 229 through 234
```

```
    738247      738277      738308      738338      738369      738400

Columns 235 through 240

    738430      738461      738491      738522      738553      738581

Columns 241 through 246

    738612      738642      738673      738703      738734      738765

Columns 247 through 252

    738795      738826      738856      738887      738918      738946

Columns 253 through 258

    738977      739007      739038      739068      739099      739130

Columns 259 through 264

    739160      739191      739221      739252      739283      739312

Columns 265 through 270

    739343      739373      739404      739434      739465      739496

Columns 271 through 276

    739526      739557      739587      739618      739649      739677

Columns 277 through 282

    739708      739738      739769      739799      739830      739861

Columns 283 through 288
```

```
    739891       739922       739952       739983       740014       740042

Columns 289 through 294

    740073       740103       740134       740164       740195       740226

Columns 295 through 300

    740256       740287       740317       740348       740379       740407

Columns 301 through 306

    740438       740468       740499       740529       740560       740591

Columns 307 through 312

    740621       740652       740682       740713       740744       740773

Columns 313 through 318

    740804       740834       740865       740895       740926       740957

Columns 319 through 324

    740987       741018       741048       741079       741110       741138

Columns 325 through 330

    741169       741199       741230       741260       741291       741322

Columns 331 through 334

    741352       741383       741413       741444


Interest =
```

mbscfamounts

```
Columns 1 through 8

       0    0.9333    1.9333    2.9333    3.9333    4.9333    5.9333    6.9333

Columns 9 through 16

  7.9333    8.9333    9.9333   10.9333   11.9333   12.9333   13.9333   14.9333

Columns 17 through 24

 15.9333   16.9333   17.9333   18.9333   19.9333   20.9333   21.9333   22.9333

Columns 25 through 32

 23.9333   24.9333   25.9333   26.9333   27.9333   28.9333   29.9333   30.9333

Columns 33 through 40

 31.9333   32.9333   33.9333   34.9333   35.9333   36.9333   37.9333   38.9333

Columns 41 through 48

 39.9333   40.9333   41.9333   42.9333   43.9333   44.9333   45.9333   46.9333

Columns 49 through 56

 47.9333   48.9333   49.9333   50.9333   51.9333   52.9333   53.9333   54.9333

Columns 57 through 64

 55.9333   56.9333   57.9333   58.9333   59.9333   60.9333   61.9333   62.9333

Columns 65 through 72

 63.9333   64.9333   65.9333   66.9333   67.9333   68.9333   69.9333   70.9333

Columns 73 through 80
```

```
    71.9333    72.9333    73.9333    74.9333    75.9333    76.9333    77.9333    78.9333

  Columns 81 through 88

    79.9333    80.9333    81.9333    82.9333    83.9333    84.9333    85.9333    86.9333

  Columns 89 through 96

    87.9333    88.9333    89.9333    90.9333    91.9333    92.9333    93.9333    94.9333

  Columns 97 through 104

    95.9333    96.9333    97.9333    98.9333    99.9333   100.9333   101.9333   102.9333

  Columns 105 through 112

   103.9333   104.9333   105.9333   106.9333   107.9333   108.9333   109.9333   110.9333

  Columns 113 through 120

   111.9333   112.9333   113.9333   114.9333   115.9333   116.9333   117.9333   118.9333

  Columns 121 through 128

   119.9333   120.9333   121.9333   122.9333   123.9333   124.9333   125.9333   126.9333

  Columns 129 through 136

   127.9333   128.9333   129.9333   130.9333   131.9333   132.9333   133.9333   134.9333

  Columns 137 through 144

   135.9333   136.9333   137.9333   138.9333   139.9333   140.9333   141.9333   142.9333

  Columns 145 through 152
```

```
  143.9333  144.9333  145.9333  146.9333  147.9333  148.9333  149.9333  150.9333

Columns 153 through 160

  151.9333  152.9333  153.9333  154.9333  155.9333  156.9333  157.9333  158.9333

Columns 161 through 168

  159.9333  160.9333  161.9333  162.9333  163.9333  164.9333  165.9333  166.9333

Columns 169 through 176

  167.9333  168.9333  169.9333  170.9333  171.9333  172.9333  173.9333  174.9333

Columns 177 through 184

  175.9333  176.9333  177.9333  178.9333  179.9333  180.9333  181.9333  182.9333

Columns 185 through 192

  183.9333  184.9333  185.9333  186.9333  187.9333  188.9333  189.9333  190.9333

Columns 193 through 200

  191.9333  192.9333  193.9333  194.9333  195.9333  196.9333  197.9333  198.9333

Columns 201 through 208

  199.9333  200.9333  201.9333  202.9333  203.9333  204.9333  205.9333  206.9333

Columns 209 through 216

  207.9333  208.9333  209.9333  210.9333  211.9333  212.9333  213.9333  214.9333

Columns 217 through 224

  215.9333  216.9333  217.9333  218.9333  219.9333  220.9333  221.9333  222.9333
```

```
Columns 225 through 232

223.9333  224.9333  225.9333  226.9333  227.9333  228.9333  229.9333  230.9333

Columns 233 through 240

231.9333  232.9333  233.9333  234.9333  235.9333  236.9333  237.9333  238.9333

Columns 241 through 248

239.9333  240.9333  241.9333  242.9333  243.9333  244.9333  245.9333  246.9333

Columns 249 through 256

247.9333  248.9333  249.9333  250.9333  251.9333  252.9333  253.9333  254.9333

Columns 257 through 264

255.9333  256.9333  257.9333  258.9333  259.9333  260.9333  261.9333  262.9333

Columns 265 through 272

263.9333  264.9333  265.9333  266.9333  267.9333  268.9333  269.9333  270.9333

Columns 273 through 280

271.9333  272.9333  273.9333  274.9333  275.9333  276.9333  277.9333  278.9333

Columns 281 through 288

279.9333  280.9333  281.9333  282.9333  283.9333  284.9333  285.9333  286.9333

Columns 289 through 296

287.9333  288.9333  289.9333  290.9333  291.9333  292.9333  293.9333  294.9333
```

```
    Columns 297 through 304

  295.9333  296.9333  297.9333  298.9333  299.9333  300.9333  301.9333  302.9333

    Columns 305 through 312

  303.9333  304.9333  305.9333  306.9333  307.9333  308.9333  309.9333  310.9333

    Columns 313 through 320

  311.9333  312.9333  313.9333  314.9333  315.9333  316.9333  317.9333  318.9333

    Columns 321 through 328

  319.9333  320.9333  321.9333  322.9333  323.9333  324.9333  325.9333  326.9333

    Columns 329 through 334

  327.9333  328.9333  329.9333  330.9333  331.9333  332.9333


Prepayment =

  Columns 1 through 8

    1.0000    0.9944    0.9887    0.9828    0.9769    0.9711    0.9653    0.9595

  Columns 9 through 16

    0.9538    0.9481    0.9424    0.9368    0.9311    0.9255    0.9199    0.9144

  Columns 17 through 24

    0.9089    0.9034    0.8979    0.8925    0.8871    0.8817    0.8763    0.8710

  Columns 25 through 32
```

```
    0.8657    0.8604    0.8552    0.8499    0.8447    0.8396    0.8344    0.8293

Columns 33 through 40

    0.8242    0.8191    0.8140    0.8090    0.8040    0.7990    0.7941    0.7892

Columns 41 through 48

    0.7842    0.7794    0.7745    0.7697    0.7649    0.7601    0.7553    0.7506

Columns 49 through 56

    0.7458    0.7411    0.7365    0.7318    0.7272    0.7226    0.7180    0.7134

Columns 57 through 64

    0.7089    0.7044    0.6999    0.6954    0.6910    0.6865    0.6821    0.6777

Columns 65 through 72

    0.6734    0.6690    0.6647    0.6604    0.6561    0.6519    0.6476    0.6434

Columns 73 through 80

    0.6392    0.6350    0.6309    0.6267    0.6226    0.6185    0.6144    0.6104

Columns 81 through 88

    0.6063    0.6023    0.5983    0.5943    0.5903    0.5864    0.5825    0.5785

Columns 89 through 96

    0.5747    0.5708    0.5669    0.5631    0.5593    0.5555    0.5517    0.5479

Columns 97 through 104

    0.5442    0.5405    0.5368    0.5331    0.5294    0.5257    0.5221    0.5185
```

```
Columns 105 through 112

  0.5149    0.5113    0.5077    0.5042    0.5006    0.4971    0.4936    0.4901

Columns 113 through 120

  0.4866    0.4832    0.4797    0.4763    0.4729    0.4695    0.4661    0.4628

Columns 121 through 128

  0.4594    0.4561    0.4528    0.4495    0.4462    0.4430    0.4397    0.4365

Columns 129 through 136

  0.4333    0.4301    0.4269    0.4237    0.4205    0.4174    0.4143    0.4111

Columns 137 through 144

  0.4080    0.4049    0.4019    0.3988    0.3958    0.3927    0.3897    0.3867

Columns 145 through 152

  0.3837    0.3808    0.3778    0.3748    0.3719    0.3690    0.3661    0.3632

Columns 153 through 160

  0.3603    0.3574    0.3546    0.3517    0.3489    0.3461    0.3433    0.3405

Columns 161 through 168

  0.3377    0.3350    0.3322    0.3295    0.3267    0.3240    0.3213    0.3186

Columns 169 through 176

  0.3160    0.3133    0.3106    0.3080    0.3054    0.3027    0.3001    0.2975
```

```
Columns 177 through 184

  0.2950    0.2924    0.2898    0.2873    0.2847    0.2822    0.2797    0.2772

Columns 185 through 192

  0.2747    0.2722    0.2698    0.2673    0.2649    0.2624    0.2600    0.2576

Columns 193 through 200

  0.2552    0.2528    0.2504    0.2480    0.2457    0.2433    0.2410    0.2386

Columns 201 through 208

  0.2363    0.2340    0.2317    0.2294    0.2271    0.2249    0.2226    0.2204

Columns 209 through 216

  0.2181    0.2159    0.2137    0.2115    0.2093    0.2071    0.2049    0.2027

Columns 217 through 224

  0.2005    0.1984    0.1962    0.1941    0.1920    0.1899    0.1877    0.1856
Columns 225 through 232

  0.1836    0.1815    0.1794    0.1773    0.1753    0.1732    0.1712    0.1692

Columns 233 through 240

  0.1671    0.1651    0.1631    0.1611    0.1591    0.1572    0.1552    0.1532

Columns 241 through 248

  0.1513    0.1493    0.1474    0.1455    0.1436    0.1416    0.1397    0.1378

Columns 249 through 256
```

```
   0.1359    0.1341    0.1322    0.1303    0.1285    0.1266    0.1248    0.1229

Columns 257 through 264

   0.1211    0.1193    0.1175    0.1157    0.1139    0.1121    0.1103    0.1085

Columns 265 through 272

   0.1068    0.1050    0.1032    0.1015    0.0998    0.0980    0.0963    0.0946

Columns 273 through 280

   0.0929    0.0912    0.0895    0.0878    0.0861    0.0844    0.0827    0.0811

Columns 281 through 288

   0.0794    0.0778    0.0761    0.0745    0.0728    0.0712    0.0696    0.0680

Columns 289 through 296

   0.0664    0.0648    0.0632    0.0616    0.0600    0.0584    0.0568    0.0553

Columns 297 through 304

   0.0537    0.0522    0.0506    0.0491    0.0476    0.0460    0.0445    0.0430

Columns 305 through 312

   0.0415    0.0400    0.0385    0.0370    0.0355    0.0340    0.0325    0.0311

Columns 313 through 320

   0.0296    0.0281    0.0267    0.0252    0.0238    0.0223    0.0209    0.0195

Columns 321 through 328

   0.0180    0.0166    0.0152    0.0138    0.0124    0.0110    0.0096    0.0082
```

```
Columns 329 through 334

0.0068    0.0055    0.0041    0.0027    0.0014         0
```

References   [1] *PSA Uniform Practices*, SF-4

See Also   mbspassthrough | mbsnoprepay | cmoseqcf | cmoschedcf | cmosched

mbsconvp

Purpose	Convexity of mortgage pool given price	
Syntax	Convexity = mbsconvp(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)	
Arguments	Price	Clean price for every $100 face value.
	Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
	Maturity	Maturity date. A serial date number or date string.
	IssueDate	Issue date. A serial date number or date string.
	GrossRate	Gross coupon rate (including fees), in decimal.
	CouponRate	Net coupon rate, in decimal. Default = GrossRate.
	Delay	Delay in days.
	PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. Set PrepaySpeed to [ ] if you input a customized prepayment matrix.
	PrepayMatrix	(Optional) Used only when PrepaySpeed is unspecified. Customized prepayment vector. A NaN-padded matrix of size max(TermRemaining)-by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS) by 1 vectors.

Description   Convexity = mbsconvp(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)

computes mortgage-backed security convexity, given time information, price at settlement, and optionally, a prepayment model.

Note If you specify the PSA or FHA model, it will be seasoned with how long the debt has been outstanding (the loan's age).

Examples   Given a mortgage-backed security with the following characteristics, compute the convexity of the security.

```
Price      = 101;
Settle     = '15-Apr-2002';
Maturity   = '1 Jan 2030';
IssueDate  = '1-Jan-2000';
GrossRate  = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;

Convexity = mbsconvp(Price, Settle, Maturity, IssueDate,...
GrossRate, CouponRate, Delay, Speed)

Convexity =

    71.6299
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbsconvy | mbsdurp | mbspassthrough | mbsnoprepay | mbsdury

mbsconvy

Purpose	Convexity of mortgage pool given yield
Syntax	Convexity = mbsconvy(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)

Arguments

Yield	Mortgage yield, compounded monthly (in decimal).
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	Net coupon rate, in decimal. Default = GrossRate.
Delay	Delay in days.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Used only when PrepaySpeed is unspecified. Customized prepayment vector. A NaN-padded matrix of size max(TermRemaining)-by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS) by 1 vectors.

Description Convexity = mbsconvy(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes mortgage-backed security convexity, given time information, semiannual mortgage yield, and optionally, a prepayment model.

Note If you specify the PSA or FHA model, it will be seasoned with how long the debt has been outstanding (the loan's age).

Examples   Given a mortgage-backed security with the following characteristics, compute the convexity of the security.

```
Yield      = 0.07125;
Settle     = '15-Apr-2002';
Maturity   = '1 Jan 2030';
IssueDate  = '1-Jan-2000';
GrossRate  = 0.08125;
Speed      = 100;
CouponRate = 0.075;
Delay = 14;

Convexity = mbsconvy(Yield, Settle, Maturity, IssueDate, ...
GrossRate, CouponRate, Delay, Speed)

Convexity =

  72.8263
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbsconvp | mbsdurp | mbsdury

mbsdurp

Purpose   Duration of mortgage pool given price

Syntax   [YearDuration, ModDuration] = mbsdurp(Price, Settle, Maturity,
                 IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed,
                 PrepayMatrix)

Arguments

Price	Clean price for every $100 face value.
Settle	Settlement date. A serial date number or date string. `Settle` must be earlier than or equal to `Maturity`.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = `GrossRate`.
Delay	(Optional) Delay in days.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. Set `PrepaySpeed` to [ ] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Used only when `PrepaySpeed` is unspecified. Customized prepayment vector. A `NaN`-padded matrix of size `max(TermRemaining)`-by-`NMBS`. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by 1 vectors.

Description   [YearDuration, ModDuration] = mbsdurp(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes the mortgage-backed security Macaulay (YearDuration) in years and modified (ModDuration) durations in years, given time information, price at settlement, and optionally, a prepayment model.

Note If you specify the PSA or FHA model, it will be seasoned with how long the debt has been outstanding (the loan's age).

Examples   Given a mortgage-backed security with the following characteristics, compute the Macaulay and modified durations of the security.

```
Price = 101;
Settle = datenum('15-Apr-2002');
Maturity = datenum('1 Jan 2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;;
Delay = 14;
Speed = 100;

[YearDuration, ModDuration] = mbsdurp(Price, Settle, Maturity,...
IssueDate, GrossRate, CouponRate, Delay, Speed)

YearDuration =

    6.4380

ModDuration =

    6.2080
```

References   [1] *PSA Uniform Practices*, SF-49

mbsdurp

mbsdury

Purpose   Duration of mortgage pool given yield

Syntax   [YearDuration, ModDuration] = mbsdury(Yield, Settle, Maturity,
                   IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed,
                   PrepayMatrix)

Arguments

Yield	Mortgage yield, compounded monthly, in decimal.
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	Net coupon rate, in decimal. Default = GrossRate.
Delay	Delay in days.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Used only when PrepaySpeed is unspecified. Customized prepayment vector. A NaN-padded matrix of size max(TermRemaining)-by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS) by 1 vectors.

Description   [YearDuration, ModDuration] = mbsdury(Yield, Settle,
                   Maturity, IssueDate, GrossRate, CouponRate, Delay,
                   PrepaySpeed, PrepayMatrix) computes the mortgage-backed security

11-1409

mbsdury

Macaulay (`YearDuration`) and Modified (`ModDuration`) durations, given time information, yield to maturity, and optionally, a prepayment model.

Note If you specify the PSA or FHA model, it will be seasoned with how long the debt has been outstanding (the loan's age).

Examples   Given a mortgage-backed security with the following characteristics, compute the Macaulay and Modified durations of the security.

```
Yield = 0.07298413;
Settle = '15-Apr-2002';
Maturity = '1 Jan 2030';
IssueDate = '1-Jan-2000';
GrossRate = 0.08125;
Speed = 100;
CouponRate = 0.075;
Delay = 14;

[YearDuration, ModDuration] = mbsdury(Yield, Settle, Maturity,...
IssueDate, GrossRate, CouponRate, Delay, Speed)

YearDuration =

    6.4380

ModDuration =

    6.2080
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbsconvp | mbsconvy | mbsdurp

Purpose   End-of-month mortgage cash flows and balances without prepayment

Syntax   [Balance, Interest, Payment, Principal] =
                  mbsnoprepay(OriginalBalance, GrossRate, Term)

Arguments

OriginalBalance	Original face value in dollars.
GrossRate	Gross coupon rate (including fees), in decimal.
Term	Term of the mortgage in months.

All inputs are number of mortgage-backed securities (NMBS)-by-1 vectors.

Description   [Balance, Interest, Payment, Principal] =
                  mbsnoprepay(OriginalBalance, GrossRate, Term) computes
                  end-of-month mortgage balance, interest payments, principal payments,
                  and cash flow payments with zero prepayment rate.

The function returns amortizing cash flows and balances over a
specified term with no prepayment. When the lengths of pass-throughs
are not the same, MATLAB software pads the shorter ones with NaN.

Balance lists the end-of-month balances over the life of the
pass-through.

Interest lists all end-of-month interest payments over the life of the
pass-through.

Payment lists all end-of-month payments over the life of the
pass-through.

Principal lists all scheduled end-of-month principal payments over
the life of the pass-through.

All outputs are Term-by-1 vectors.

Examples   Given mortgage pools with the following characteristics, compute an
                  amortization schedule.

```
OriginalBalance = 400000000;
CouponRate = 0.08125;
Term = [357; 355]; % Three- and five-month old mortgage pools.

[Balance, Interest, Payment, Principal] = ...
mbsnoprepay(OriginalBalance, CouponRate, Term);
```

mbsoas2price

Purpose   Price given option-adjusted spread

Syntax   Price = mbsoas2price(ZeroCurve, OAS, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)

Arguments

ZeroCurve	A matrix of three columns:
	• Column 1: Serial date numbers.
	• Column 2: Spot rates with maturities corresponding to the dates in Column 1, in decimal (for example, 0.075).
	• Column 3: Compounding of the rates in Column 2. (This is the agency spot rate on the settlement date.)
OAS	Option-adjusted spreads in basis points.
Settle	Settlement date (scalar only). A serial date number or date string. Date when option-adjusted spread is calculated. Settle must be earlier than Maturity.
Maturity	Maturity date. Scalar or vector in serial date number or date string format.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).

mbsoas2price

Interpolation	Interpolation method. Computes the corresponding spot rates for the bond's cash flow. Available methods are (0) nearest, (1) linear, and (2) cubic spline. Default = 1. See `interp1` for more information.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set `PrepaySpeed` to `[]` if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size `max(TermRemaining)`-by-NMBS. Missing values are padded with `NaN`s. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by 1 vectors.

Description   `Price = mbsoas2price(ZeroCurve, OAS, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)` computes the clean price of a pass-through security for each $100 face value of outstanding principal.

Examples   Given an option-adjusted spread, a spot curve, and a prepayment assumption, compute theoretical price of a mortgage pool.

Create the bonds matrix.

```
Bonds = [datenum('11/21/2002')  0        100  0  2  1;
         datenum('02/20/2003')  0        100  0  2  1;
         datenum('07/31/2004')  0.03     100  2  3  1;
         datenum('08/15/2007')  0.035    100  2  3  1;
         datenum('08/15/2012')  0.04875  100  2  3  1;
         datenum('02/15/2031')  0.05375  100  2  3  1];
```

Choose a settlement date.

```
Settle = datenum('20-Aug-2002');
```

Assume these clean prices for the bonds.

```
Prices = [ 98.97467;
            98.58044;
           100.10534;
            98.18054;
           101.38136;
            99.25411];
```

Use this formula to compute spot compounding for the bonds.

```
SpotCompounding = 2*ones(size(Prices));
```

Use compute the zero curve.

```
[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);
ZeroCurve = [CurveDatesP, ZeroRatesP, SpotCompounding];
```

Assign parameters.

```
OAS           = [26.0502; 28.6348; 31.2222];
Maturity      = datenum('02-Jan-2030');
IssueDate     = datenum('02-Jan-2000');
GrossRate     = 0.08125;
CouponRate    = 0.075;
Delay         = 14;
Interpolation = 1;
PrepaySpeed   = [0 50 100];
```

Calculate the price from the option-adjusted spread.

```
Price = mbsoas2price(ZeroCurve, OAS, Settle, Maturity, ...
IssueDate, GrossRate, CouponRate, Delay, Interpolation, ...
PrepaySpeed)
```

mbsoas2price

```
Price =

    95.0000
    95.0000
    95.0000
```

See Also   mbsprice2oas | mbsyield2oas | mbsoas2yield

mbsoas2yield

Purpose   Yield given option-adjusted spread

Syntax   [MYield, BEMBSYield] = mbsoas2yield(ZeroCurve, OAS, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)

Arguments

ZeroCurve	A matrix of three columns:
	• Column 1: Serial date numbers.
	• Column 2: Spot rates with maturities corresponding to the dates in Column 1, in decimal (for example, 0.075).
	• Column 3: Compounding of the rates in Column 2. (This is the agency spot rate on the settlement date.)
OAS	Option-adjusted spreads in basis points.
Settle	Settlement date (scalar only). A serial date number or date string. Date when option-adjusted spread is calculated. Settle must be earlier than Maturity.
Maturity	Maturity date. Scalar or vector in serial date number or date string format.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).

11-1417

mbsoas2yield

Interpolation	Interpolation method. Computes the corresponding spot rates for the bond's cash flow. Available methods are (0) nearest, (1) linear, and (2) cubic spline. Default = 1. See `interp1` for more information.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set `PrepaySpeed` to `[ ]` if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size `max(TermRemaining)`-by-`NMBS`. Missing values are padded with `NaN`s. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by 1 vectors.

Description [MYield, BEMBSYield] = mbsoas2yield(ZeroCurve, OAS, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix) computes the mortgage and bond-equivalent yields of a pass-through security.

MYield is the yield to maturity of the mortgage-backed security (the mortgage yield). This yield is compounded monthly (12 times per year). For example:

0.075 (7.5%)

BEMBSYield is the corresponding bond equivalent yield of the mortgage-backed security. This yield is compounded semiannually (two times per year). For example:

0.0761 (7.61%)

Examples   Given an option-adjusted spread, a spot curve, and a prepayment
assumption, compute the theoretical yield to maturity of a mortgage
pool.

Create the bonds matrix.

```
Bonds = [datenum('11/21/2002')   0       100 0 2 1;
         datenum('02/20/2003')   0       100 0 2 1;
         datenum('07/31/2004')   0.03    100 2 3 1;
         datenum('08/15/2007')   0.035   100 2 3 1;
         datenum('08/15/2012')   0.04875 100 2 3 1;
         datenum('02/15/2031')   0.05375 100 2 3 1];
```

Choose a settlement date.

```
Settle = datenum('08/20/2002');
```

Assume these clean prices for the bonds.

```
Prices =  [ 98.97467;
            98.58044;
           100.10534;
            98.18054;
           101.38136;
            99.25411];
```

Use this formula to compute spot compounding for the bonds.

```
SpotCompounding = 2*ones(size(Prices));
```

Compute the zero curve.

```
[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);
ZeroCurve = [CurveDatesP, ZeroRatesP, SpotCompounding];
```

Assign parameters.

```
OAS        = [26.0502; 28.6348; 31.2222];
Maturity   = datenum('02-Jan-2030');
```

```
IssueDate      = datenum('02-Jan-2000');
GrossRate      = 0.08125;
CouponRate     = 0.075;
Delay          = 14;
Interpolation  = 1;
PrepaySpeed    = [0 50 100];
```

Compute the mortgage yield and bond equivalent mortgage yield.

```
[MYield BEMBSYield] = mbsoas2yield(ZeroCurve, OAS, Settle, ...
Maturity, IssueDate, GrossRate, CouponRate, Delay, ...
Interpolation, PrepaySpeed)

MYield =

    0.0802
    0.0814
    0.0828


BEMBSYield =

    0.0816
    0.0828
    0.0842
```

See Also   mbsprice2oas | mbsyield2oas | mbsoas2price

Purpose	Mortgage pool cash flows and balances with prepayment

Syntax

```
[Balance, Payment, Principal, Interest, Prepayment] =
mbspassthrough(OriginalBalance, GrossRate, OriginalTerm,
TermRemaining,  PrepaySpeed, PrepayMatrix)
```

Arguments

OriginalBalance	Original balance value in dollars (balance at the beginning of each `TermRemaining`).
GrossRate	Gross coupon rate (including fees), in decimal.
OriginalTerm	Term of the mortgage in months.
TermRemaining	(Optional) Number of full months between settlement and maturity.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0 (no prepayment). Set `PrepaySpeed` to `[]` if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Used only when `PrepaySpeed` is unspecified. Customized prepayment vector. A `NaN`-padded matrix of size `max(TermRemaining)`-by-`NMBS`. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by 1 vectors.

Description   `[Balance, Payment, Principal, Prepayment, Interest] = mbspassthrough(OriginalBalance, GrossRate, OriginalTerm, TermRemaining, PrepaySpeed, PrepayMatrix)` computes the cash flow of principal, interest, and prepayment of pass-through securities.

All outputs are `TermRemaining`-by-1 vectors of end-of-month values.

Balance is a matrix for the principal balance at end of month.

Payment is a matrix for the total monthly payment.

Principal is a matrix for the principal portion of the payment.

Interest is a matrix for the interest portion of the payment.

Prepayment is a matrix that indicates any unscheduled principal payment.

By default, the securities are seasoned. The applicable CPR depends upon TermRemaining based on a 30-year prepayment model (PSA or FHA). You may supply a different CPR vector of size TermRemaining-by-1.

Examples   Compute the cash flows and balances of a 3-month old mortgage pool with original term of 360 months, assuming a prepayment speed of 100.

```
OriginalBalance = 100000;
GrossRate = 0.08125;
OriginalTerm = 360;
TermRemaining = 357;
PrepaySpeed = 100;

[Balance, Payment, Principal, Interest, Prepayment] =...
mbspassthrough(OriginalBalance, GrossRate, OriginalTerm,...
TermRemaining, PrepaySpeed);
```

See Also   mbswal

mbsprice

Purpose	Mortgage-backed security price given yield

Syntax   [Price, AccrInt] = mbsprice(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)

Arguments

Yield	Mortgage yield, compounded monthly (in decimal).
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0 (no prepayment). Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS) by 1 vectors.

Description   [Price, AccrInt] = mbsprice(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed,

PrepayMatrix) computes a mortgage-backed security price, given time information, mortgage yield at settlement, and optionally, a prepayment model.

All outputs are scalar values.

Price is the clean price for every $100 face value of the securities.

AccrInt is the accrued interest of the mortgage-backed securities.

Examples

Example 1. Given a mortgage-backed security with the following characteristics, compute the price and the accrued interest due on the security.

```
Yield = 0.0725;
Settle = datenum('15-Apr-2002');
Maturity = datenum('1 Jan 2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;

[Price AccrInt] = mbsprice(Yield, Settle, Maturity, IssueDate,...
GrossRate, CouponRate, Delay, Speed)

Price =

   101.3147


AccrInt =

     0.2917
```

Example 2. Given a portfolio of mortgage-backed securities, compute the clean prices and accrued interest.

```
Yield = 0.075;
```

```
Settle = datenum(['13-Feb-2000';'17-Apr-2002';'17-May-2002';...
'13-Jan-2000']);
Maturity  = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = [0.075; 0.07875; 0.0775; 0.08125];
Delay = 14;
Speed = 100;

[Price AccrInt] = mbsprice(Yield, Settle, Maturity, IssueDate,...
GrossRate, CouponRate, Delay, Speed)

Price =

   99.7085
  102.0678
  101.2792
  104.0175

AccrInt =

    0.2500
    0.3500
    0.3444
    0.2708
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbsyield

mbsprice2oas

Purpose   Option-adjusted spread given price

Syntax   OAS = mbsprice2oas(ZeroCurve, Price, Settle, Maturity, IssueDate,
                  GrossRate, CouponRate, Delay, Interpolation PrepaySpeed,
                  PrepayMatrix)

Arguments

ZeroCurve	A matrix of three columns:
	• Column 1: Serial date numbers.
	• Column 2: Spot rates with maturities corresponding to the dates in Column 1, in decimal (for example, 0.075).
	• Column 3: Compounding of the rates in Column 2. Values are 1 (annual), 2 (semiannual, 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).
Price	Clean price for every $100 face value of bond issue.
Settle	Settlement date (scalar only). A serial date number or date string. Date when option-adjusted spread is calculated. Settle must be earlier than Maturity.
Maturity	Maturity date. Scalar or vector in serial date number or date string format.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.

Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt.
Interpolation	Interpolation method. Computes the corresponding spot rates for the bond's cash flow. Available methods are (0) nearest, (1) linear, and (2) cubic spline. Default = 1. See interp1 for more information.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS) by 1 vectors.

Description OAS = mbsprice2oas(ZeroCurve, Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix) computes the monthly option-adjusted spread in basis points.

Examples Calculate the option-adjusted spread of a 30-year fixed-rate mortgage with about a 28-year weighted average maturity remaining, given assumptions of 0, 50, and 100 PSA prepayments.

Create the bonds matrix.

```
Bonds = [datenum('11/21/2002')   0        100 0 2 1;
         datenum('02/20/2003')   0        100 0 2 1;
```

```
            datenum('07/31/2004')   0.03     100  2  3  1;
            datenum('08/15/2007')   0.035    100  2  3  1;
            datenum('08/15/2012')   0.04875  100  2  3  1;
            datenum('02/15/2031')   0.05375  100  2  3  1];
```

Choose a settlement date.

```
Settle= datenum('20-Aug-2002');
```

Assume these clean prices for the bonds.

```
Prices =  [ 98.97467;
             98.58044;
            100.10534;
             98.18054;
            101.38136;
             99.25411];
```

Use this formula to compute spot compounding for the bonds.

```
SpotCompounding = 2*ones(size(Prices));
```

Compute the zero curve.

```
[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);
ZeroCurve = [CurveDatesP, ZeroRatesP, SpotCompounding];
```

Assign parameters.

```
Price         = 95;
Maturity      = datenum('02-Jan-2030');
IssueDate     = datenum('02-Jan-2000');
GrossRate     = 0.08125;
CouponRate    = 0.075;
Delay         = 14;
Interpolation = 1;
PrepaySpeed   = [0; 50; 100];
Interpolation = 1;
```

Compute the option-adjusted spread.

```
OAS = mbsprice2oas(ZeroCurve, Price, Settle, Maturity, ...
IssueDate, GrossRate, CouponRate, Delay, Interpolation, ...
PrepaySpeed)

OAS =

  26.0502
  28.6348
  31.2222
```

See Also   mbsoas2price | mbsoas2yield | mbsyield2oas

mbsprice2speed

Purpose	Implied PSA prepayment speeds given price
Syntax	[ImpSpdOnPrc, ImpSpdOnDur, ImpSpdOnCnv] = mbsprice2speed(Price, Settle, Maturity, IssueDate, GrossRate, PrepayMatrix, CouponRate, Delay)

Arguments

Price	Clean price for every $100 face value.
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
PrepayMatrix	Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS)-by-1 vectors.

Description [ImpSpdOnPrc, ImpSpdOnDur, ImpSpdOnCnv] = mbsprice2speed(Price, Settle, Maturity, IssueDate, GrossRate, PrepayMatrix, CouponRate, Delay) computes PSA prepayment speeds implied by pool prices and projected (user-defined) prepayment vectors. The calculated PSA speed produces the same

price, modified duration, or modified convexity, depending upon the output requested.

`ImpSpdOnPrc` calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same price.

`ImpSpdOnDur` calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same modified duration.

`ImpSpdOnCnv` calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same modified convexity.

All outputs are `NMBS`-by-1 vectors.

Examples Calculate the equivalent PSA benchmark prepayment speeds for a mortgage pool with these characteristics and prepayment matrix.

```
Price        = 101;
Settle       = datenum('1-Jan-2000');
Maturity     = datenum('1-Jan-2030');
IssueDate    = datenum('1-Jan-2000');
GrossRate    = 0.08125;
PrepayMatrix = 0.005*ones(360,1);
CouponRate   = 0.075;
Delay        = 14;

[ImpSpdOnPrc, ImpSpdOnDur, ImpSpdOnCnv] = ...
mbsprice2speed(Price,Settle, Maturity, IssueDate, ...
GrossRate, PrepayMatrix, CouponRate, Delay)

ImpSpdOnPrc =

  118.5980

ImpSpdOnDur =

  118.3946

ImpSpdOnCnv =
```

mbsprice2speed

```
     109.5115
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbsprice | mbsyield2speed

mbswal

Purpose	Weighted average life of mortgage pool
Compatibility	PSA
Syntax	WAL = mbswal(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)

Arguments

Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS) by 1 vectors.

Description WAL = mbswal(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes the

mbswal

weighted average life, in number of years, of a mortgage pool, as measured from the settlement date.

Examples   Given a pass-through security with the following characteristics, compute the weighted average life of the security.

```
Settle = datenum('15-Apr-2002');
Maturity = datenum('1 Jan 2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;

WAL = mbswal(Settle, Maturity, IssueDate, GrossRate, ...
CouponRate, Delay, Speed)

WAL =

   10.5477
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbspassthrough

Purpose	Mortgage-backed security yield given price	

Syntax [MYield, BEMBSYield] = mbsyield(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)

Arguments

Price	Clean price for every $100 face value.
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0 (no prepayment). Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS) by 1 vectors.

Description [MYield, BEMBSYield] = mbsyield(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes a mortgage-backed security yield to maturity

mbsyield

and the bond equivalent yield, given time information, price at settlement, and optionally, a prepayment model.

`MYield` is the yield to maturity of the mortgage-backed security (the mortgage yield). This yield is compounded monthly (12 times a year).

`BEMBSYield` is the corresponding bond equivalent yield of the mortgage-backed security. This yield is compounded semiannually (two times a year).

Examples

Example 1. Given a mortgage-backed security with the following characteristics, compute the mortgage yield and the bond equivalent yield of the security.

```
Price = 102;
Settle = '15-Apr-2002';
Maturity = '1 Jan 2030';
IssueDate = '1-Jan-2000';
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;

[MYield, BEMBSYield] = mbsyield(Price, Settle, Maturity, ...
IssueDate, GrossRate, CouponRate, Delay,  Speed)

MYield =

    0.0715

BEMBSYield =

    0.0725
```

Example 2. Given a portfolio of mortgage-backed securities, compute the mortgage yields and the bond equivalent yields.

```
Price = 102;
```

```
Settle = datenum(['13-Feb-2000';'17-Apr-2002';'17-May-2002';...
'13-Jan-2000']);
Maturity  = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = [0.075; 0.07875; 0.0775; 0.08125];
Delay = 14;
Speed = 100;

[MYield, BEMBSYield] = mbsyield(Price, Settle, Maturity,...
IssueDate, GrossRate, CouponRate, Delay,  Speed)

MYield =

    0.0717
    0.0751
    0.0739
    0.0779

BEMBSYield =

    0.0728
    0.0763
    0.0750
    0.0791
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbsprice

mbsyield2oas

Purpose	Option-adjusted spread given yield
Syntax	OAS = mbsyield2oas(ZeroCurve, Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation PrepaySpeed, PrepayMatrix)

Arguments

ZeroCurve	A matrix of three columns:
	• Column 1: Serial date numbers.
	• Column 2: Spot rates with maturities corresponding to the dates in Column 1, in decimal (for example, 0.075).
	• Column 3: Compounding of the rates in Column 2. Values are 1 (annual), 2 (semiannual, 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).
Yield	Mortgage yield, compounded monthly (in decimal).
Settle	Settlement date (scalar only). A serial date number or date string. Date when option-adjusted spread is calculated. Settle must be earlier than Maturity.
Maturity	Maturity date. Scalar or vector in serial date number or date string format.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).

	Interpolation	Interpolation method. Computes the corresponding spot rates for the bond's cash flow. Available methods are (0) nearest, (1) linear, and (2) cubic spline. Default = 1. See `interp1` for more information.
	PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set `PrepaySpeed` to `[]` if you input a customized prepayment matrix.
	PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size `max(TermRemaining)`-by-`NMBS`. Missing values are padded with `NaNs`. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by 1 vectors.

Description   `OAS = mbsyield2oas(ZeroCurve, Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)` computes the option-adjusted spread in basis points.

Examples   Calculate the option-adjusted spread of a 30-year, fixed-rate mortgage pool with about 28-year weighted average maturity left, given assumptions of 0, 50, and 100 PSA prepayments.

Create a bonds matrix.

```
Bonds = [datenum('11/21/2002') 0       100 0 2 1;
         datenum('02/20/2003') 0       100 0 2 1;
         datenum('07/31/2004') 0.03    100 2 3 1;
         datenum('08/15/2007') 0.035   100 2 3 1;
         datenum('08/15/2012') 0.04875 100 2 3 1;
         datenum('02/15/2031') 0.05375 100 2 3 1];
```

mbsyield2oas

Choose a settlement date.

```
Settle = datenum('08/20/2002');
```

Assume these clean prices for the bonds.

```
Prices =  [ 98.97467;
             98.58044;
            100.10534;
             98.18054;
            101.38136;
             99.25411];
```

Use this formula to compute spot compounding for the bonds.

```
SpotCompounding = 2*ones(size(Prices));
```

Compute the zero curve.

```
[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);
ZeroCurve = [CurveDatesP, ZeroRatesP, SpotCompounding];
```

Assign parameters.

```
Price         = 95;
Maturity      = datenum('02-Jan-2030');
IssueDate     = datenum('02-Jan-2000');
GrossRate     = 0.08125;
CouponRate    = 0.075;
Delay         = 14;
Interpolation = 1;
PrepaySpeed   = [0 50 100];
```

Compute the yield, and from the yield, compute the option-adjusted spread.

```
[mbsyld, beyld] = mbsyield(Price, Settle, ...
Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed);
```

```
OAS = mbsyield2oas(ZeroCurve, mbsyld, Settle, ...
Maturity, IssueDate, GrossRate, CouponRate, Delay, ...
Interpolation, PrepaySpeed)

OAS =

   26.0502
   28.6348
   31.2222
```

See Also mbsoas2price | mbsoas2yield | mbsprice2oas

mbsyield2speed

Purpose	Implied PSA prepayment speeds given yield
Syntax	[ImpSpdOnYld, ImpSpdOnDur, ImpSpdOnCnv] = mbsyield2speed(Yield, Settle, Maturity, IssueDate, GrossRate, PrepayMatrix, CouponRate, Delay)

Arguments

Yield	Mortgage yield, compounded monthly, in decimal.
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
PrepayMatrix	Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).

All inputs (except PrepayMatrix) are number of mortgage-backed securities (NMBS) by 1 vectors.

Description [ImpSpdOnYld, ImpSpdOnDur, ImpSpdOnCnv] = mbsyield2speed(Yield, Settle, Maturity, IssueDate, GrossRate, PrepayMatrix, CouponRate, Delay) computes PSA prepayment speeds implied by pool yields and projected (user-defined) prepayment vectors. The calculated PSA speed produces the same

yield, modified duration, or modified convexity, depending upon the output requested.

`ImpSpdOnPrc` calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same price.

`ImpSpdOnDur` calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same modified duration.

`ImpSpdOnCnv` calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same modified convexity.

All outputs are `NMBS`-by-1 vectors.

Examples   Calculate the equivalent PSA benchmark prepayment speeds for a security with these characteristics and prepayment matrix.

```
Yield = 0.065;
Settle = datenum('1-Jan-2000');
Maturity = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
PrepayMatrix = 0.005*ones(360,1);
CouponRate = 0.075;
Delay = 14;

[ImpSpdOnYld, ImpSpdOnDur, ImpSpdOnCnv] = ...
mbsyield2speed(Yield, Settle, Maturity, IssueDate, GrossRate, ...
PrepayMatrix, CouponRate, Delay)

ImpSpdOnYld =

  117.7644

ImpSpdOnDur =

  116.7436

ImpSpdOnCnv =
```

mbsyield2speed

```
108.3309
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbsyield | mbsprice2speed

psaspeed2default

Purpose	Benchmark default

Syntax   [ADRPSA, MDRPSA] = psaspeed2default(DefaultSpeed)

Arguments

DefaultSpeed	Annual speed relative to the benchmark. PSA benchmark = 100.

Description   [ADRPSA, MDRPSA] = psaspeed2default(DefaultSpeed) computes the benchmark default on the performing balance of mortgage-backed securities per PSA benchmark speed.

ADRPSA is the PSA default rate, in decimal (360-by-1).

MDRPSA is the PSA monthly default rate, in decimal (360-by-1).

Examples   Given a mortgage-backed security with annual speed set at the PSA default benchmark, compute the default rates.

```
DefaultSpeed = 100;

[ADRPSA, MDRPSA] = psaspeed2default(DefaultSpeed);
```

See Also   psaspeed2rate

psaspeed2rate

Purpose	Single monthly mortality rate given PSA speed

Syntax

```
[CPRPSA, SMMPSA]= psaspeed2rate(PSASpeed)
```

Arguments

PSASpeed	Any value > 0 representing the annual speed relative to the benchmark. PSA benchmark = 100.

Description `[CPRPSA, SMMPSA]= psaspeed2rate(PSASpeed)` calculates vectors of PSA prepayments, each containing 360 prepayment elements, to represent the 360 months in a 30-year mortgage pool.

CPRPSA is the PSA conditional prepayment rate, in decimal [360-by-1].

SMMPSA is the PSA single monthly mortality rate, in decimal [360-by-1].

Examples Given a mortgage-backed security with annual speed set at the PSA default benchmark, compute the prepayment and mortality rates.

```
PSASpeed = [100 200];

[CPRPSA, SMMPSA]= psaspeed2rate(PSASpeed);
```

View a plot of the output.

```
psaspeed2rate(PSASpeed)
```
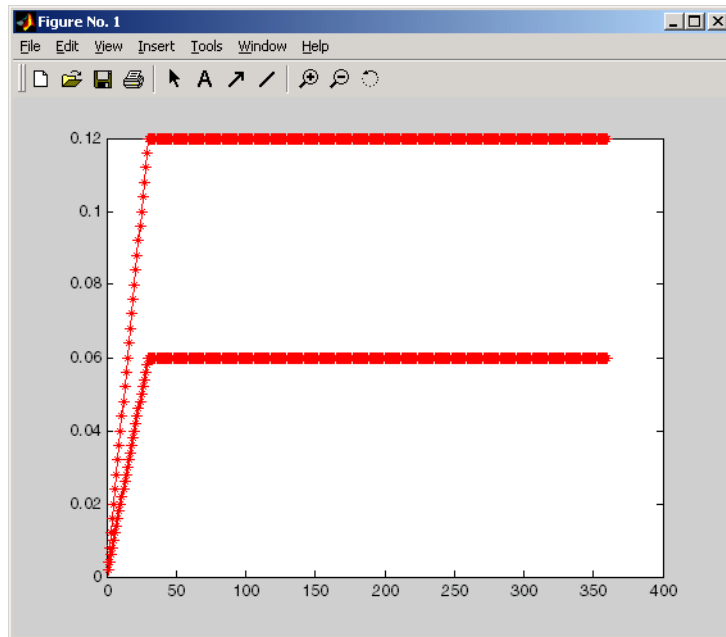
See Also   psaspeed2default

stepcpncfamounts

Purpose	Cash flow amounts and times for bonds and stepped coupons	
Syntax	[CFlows, CDates, CTimes] = stepcpncfamounts(Settle, Maturity, ConvDates, CouponRates, Period, Basis, EndMonthRule, Face)	
Arguments	Settle	Settlement date. A scalar or vector of serial date numbers. Settle must be earlier than Maturity.
	Maturity	Maturity date. A scalar or vector of serial date numbers.
	ConvDates	Matrix of serial date numbers representing conversion dates after Settle. Size = number of instruments by maximum number of conversions. Fill unspecified entries with NaN.
	CouponRates	Matrix indicating the coupon rates for each bond in decimal form. Size = number of instruments by maximum number of conversions + 1. First column of this matrix contains rates applicable between Settle and the first conversion date (date in the first column of ConvDates). Fill unspecified entries with NaN. See Note below.
	Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2, 3, 4, 6, and 12. Default = 2.

Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)
	• 10 = actual/365 (ICMA)
	• 11 = 30/360E (ICMA)
	• 12 = actual/actual (ISDA)
	• 13 = BUS/252
	For more information, see **basis** on page Glossary-1.
EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
Face	(Optional) Face value of each bond in the portfolio. Default = 100.

stepcpncfamounts

All arguments must be scalars or number of bonds (NUMBONDS)-by-1 vectors, except for ConvDates and CouponRates.

Note ConvDates has the same number of rows as CouponRates to reflect the same number of bonds. However, ConvDates has one less column than CouponRates. This situation is illustrated by

Settle---------ConvDate1-----------ConvDate2------------Maturity

      Rate1   Rate2   Rate3

Description   [CFlows, CDates, CTimes] = stepcpncfamounts(Settle, Maturity, ConvDates, CouponRates, Period, Basis, EndMonthRule, Face) returns matrices of cash flow amounts, cash flow dates, and time factors for a portfolio of NUMBONDS stepped-coupon bonds.

CFlows is a matrix of cash flow amounts. The first entry in each row vector is a negative number indicating the accrued interest due at settlement. If no accrued interest is due, the first column is 0.

CDates is a matrix of cash flow dates in serial date number form. At least two columns are always present, one for settlement and one for maturity.

CTimes is a matrix of time factors for the SIA semiannual price/yield conversion.

DiscountFactor = (1 + Yield/2).^(-TFactor)

Time factors are in units of semiannual coupon periods. In computing time factors, use SIA actual/actual conventions for all time factor calculations.

> **Note** For bonds with fixed coupons, use cfamounts. If you use a fixed-coupon bond with stepcpncfamounts, MATLAB software generates an error.

Examples   This example generates stepped cash flows for three different bonds, all paying interest semiannually. Their life span is about 18 to 19 years each:

- Bond A has two conversions, but the first one occurs on the settlement date and immediately expires.

- Bond B has three conversions, with conversion dates exactly on the coupon dates.

- Bond C has three conversions, with some conversion dates not on the coupon dates. It has the longest maturity. This case illustrates that only cash flows for full periods after conversion dates are affected, as illustrated below.

The following table illustrates the interest rate characteristics of this bond portfolio.

Bond A Dates	Bond A Rates	Bond B Dates	Bond B Rates	Bond C Dates	Bond C Rates
Settle (02-Aug-92)	7.5%	Settle (02-Aug-92)	7.5%	Settle (02-Aug-92)	2.5%
First Conversion (02-Aug-92)	8.875%	First Conversion (15-Jun-97))	8.875%	First Conversion (14-Jun-97))	5.0%

Bond A Dates	Bond A Rates	Bond B Dates	Bond B Rates	Bond C Dates	Bond C Rates
Second Conversion (15-Jun-03)	9.25%	Second Conversion (15-Jun-01)	9.25%	Second Conversion (14-Jun-01)	7.5%
Maturity (15-Jun-10)	NaN	Third Conversion (15-Jun-05)	10.0%	Third Conversion (14-Jun-05)	10.0%
		Maturity (15-Jun-10)	NaN	Maturity (15-Jun-11)	NaN

```
Settle   = datenum('02-Aug-1992');

ConvDates = [datenum('02-Aug-1992'), datenum('15-Jun-2003'),...
             nan;
             datenum('15-Jun-1997'), datenum('15-Jun-2001'),...
             datenum('15-Jun-2005');
             datenum('14-Jun-1997'), datenum('14-Jun-2001'),...
             datenum('14-Jun-2005')];

Maturity = [datenum('15-Jun-2010');
            datenum('15-Jun-2010');
            datenum('15-Jun-2011')];

CouponRates = [0.075 0.08875 0.0925 nan;
               0.075 0.08875 0.0925 0.1;
               0.025 0.05    0.0750 0.1];
Basis = 1;
Period = 2;
EndMonthRule = 1;
Face = 100;
```

Call stepcpncfamounts to compute cash flows and timings.

```
[CFlows, CDates, CTimes] = stepcpncfamounts(Settle, Maturity, ...
```

```
ConvDates, CouponRates);
```

Visualize the third bond cash flows (2.5 - 5 - 7.5 - 10) using the `cfplot` function.

```
cfplot(CDates(3,:),CFlows(3,:));
xlabel('Dates in Serial Number Format')
ylabel('Relative Amounts of Cash Flows')
title('CashFlow of 2.5 - 5 - 7.5 - 10 Stepped Coupon Bond')
```
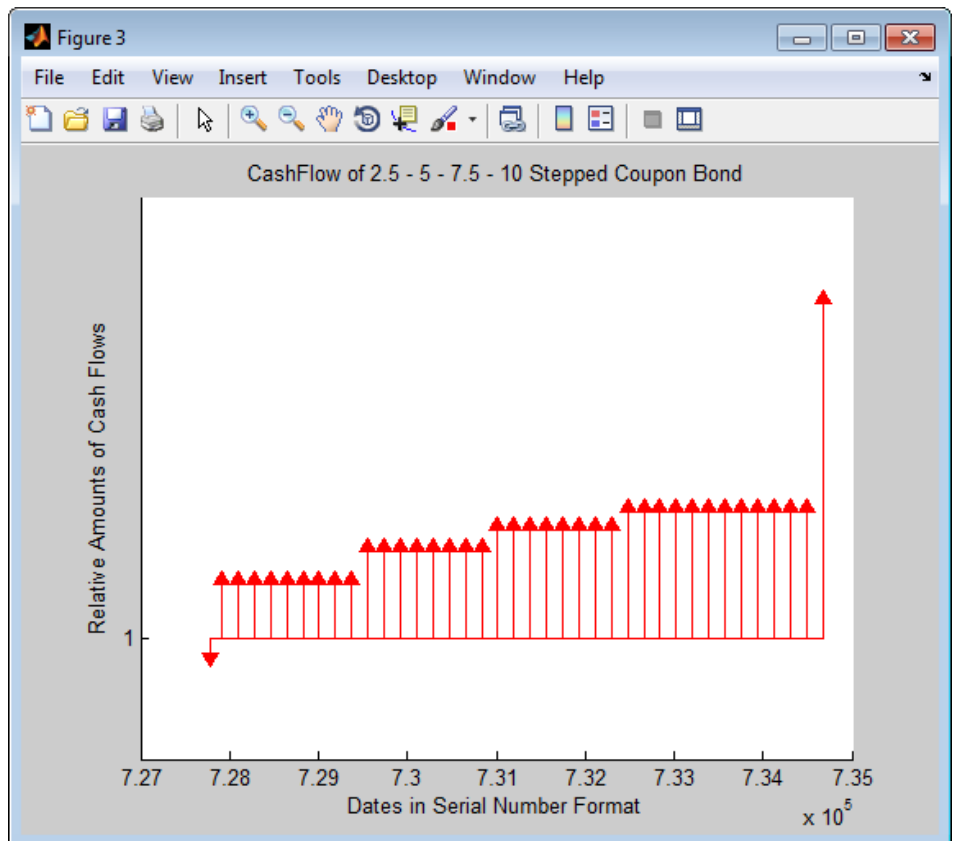
stepcpncfamounts

See Also   stepcpnprice | stepcpnyield | cfplot

Related Examples
- "Cash Flows from Stepped-Coupon Bonds" on page 6-12
- "Price and Yield of Stepped-Coupon Bonds" on page 6-14

Purpose   Price bond with stepped coupons

Syntax   [Price, AccruedInterest] = stepcpnprice(Yield, Settle, Maturity, ConvDates, CouponRates, Period, Basis, EndMonthRule, Face)

Arguments

Yield	Scalar or vector containing yield to maturity of instruments.
Settle	Settlement date. A scalar or vector of serial date numbers. Settle must be earlier than Maturity.
Maturity	Maturity date. A scalar or vector of serial date numbers.
ConvDates	Matrix of serial date numbers representing conversion dates after Settle. Size = number of instruments by maximum number of conversions. Fill unspecified entries with NaN.
CouponRates	Matrix indicating the coupon rates for each bond in decimal form. Size = number of instruments by maximum number of conversions + 1. First column of this matrix contains rates applicable between Settle and the first conversion date (date in the first column of ConvDates). Fill unspecified entries with NaN. See Note below.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2, 3, 4, 6, and 12. Default = 2.

Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)
	• 10 = actual/365 (ICMA)
	• 11 = 30/360E (ICMA)
	• 12 = actual/actual (ISDA)
	• 13 = BUS/252
	For more information, see **basis** on page Glossary-1.
EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
Face	(Optional) Face value of each bond in the portfolio. Default = 100.

All arguments must be scalars or number of bonds (NUMBONDS)-by-1 vectors, except for ConvDates and CouponRates.

Note ConvDates has the same number of rows as CouponRate to reflect the same number of bonds. However, ConvDates has one less column than CouponRate. This situation is illustrated by

```
Settle---------ConvDate1-----------ConvDate2------------Maturity

        Rate1              Rate2                Rate3
```

Description   [Price, AccruedInterest] = stepcpnprice(Yield, Settle, Maturity, ConvDates, CouponRates, Period, Basis, EndMonthRule, Face) computes the price of bonds with stepped coupons given the yield to maturity. The function supports any number of conversion dates.

Price is a NUMBONDS-by-1 vector of clean prices.

AccruedInterest is a NUMBONDS-by-1 vector of accrued interest payable at settlement dates.

Note For bonds with fixed coupons, use bndprice. If you use a fixed-coupon bond with stepcpnprice, you will receive the error: incorrect number of inputs.

Examples   Compute the price and accrued interest due on a portfolio of stepped-coupon bonds having a yield of 7.221%, given three conversion scenarios:

• Bond A has two conversions, the first one falling on the settle date and immediately expiring.

- Bond B has three conversions, with conversion dates exactly on the coupon dates.

- Bond C has three conversions, with one or more conversion dates not on coupon dates. This case illustrates that only cash flows for full periods after conversion dates are affected, as illustrated below.

The following table illustrates the interest rate characteristics of this bond portfolio.

Bond A Dates	Bond A Rates	Bond B Dates	Bond B Rates	Bond C Dates	Bond C Rates
Settle (02-Aug-92)	7.5%	Settle (02-Aug-92)	7.5%	Settle (02-Aug-92)	7.5%
First Conversion (02-Aug-92)	8.875%	First Conversion (15-Jun-97)	8.875%	First Conversion (14-Jun-97)	8.875%
Second Conversion (15-Jun-03)	9.25%	Second Conversion (15-Jun-01)	9.25%	Second Conversion (14-Jun-01)	9.25%
Maturity (15-Jun-10)	NaN	Third Conversion (15-Jun-05)	10.0%	Third Conversion (14-Jun-05)	10.0%
		Maturity (15-Jun-10)	NaN	Maturity (15-Jun-10)	NaN

```
Yield = 0.07221;
Settle  = datenum('02-Aug-1992');
ConvDates = [datenum('02-Aug-1992'), datenum('15-Jun-2003'),...
             nan;
            datenum('15-Jun-1997'), datenum('15-Jun-2001'),...
```

```
            datenum('15-Jun-2005');
            datenum('14-Jun-1997'), datenum('14-Jun-2001'),...
            datenum('14-Jun-2005')];
Maturity = datenum('15-Jun-2010');

CouponRates = [0.075 0.08875 0.0925 nan;
               0.075 0.08875 0.0925 0.1;
               0.075 0.08875 0.0925 0.1];
Basis = 1;
Period = 2;
EndMonthRule = 1;
Face = 100;

[Price, AccruedInterest] = ...
stepcpnprice(Yield, Settle, Maturity, ConvDates, CouponRates, ...
Period, Basis, EndMonthRule, Face)

Price =

  117.3824
  113.4339
  113.4339

AccruedInterest =

    1.1587
    0.9792
    0.9792
```

References This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 120 - 123, on zero-coupon instruments pricing.

See Also stepcpnprice | stepcpnyield | tbillprice | stepcpncfamounts | cdprice | bndprice

stepcpnprice

Related Examples

- "Cash Flows from Stepped-Coupon Bonds" on page 6-12
- "Price and Yield of Stepped-Coupon Bonds" on page 6-14

Purpose	Yield to maturity of bond with stepped coupons
Syntax	Yield = stepcpnyield(Price, Settle, Maturity, ConvDates, CouponRate, Period, Basis, EndMonthRule, Face)

Arguments

Price	Vector containing price of the bonds.
Settle	Settlement date. A vector of serial date numbers. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers.
ConvDates	Matrix of serial date numbers representing conversion dates after Settle. Size = number of instruments by maximum number of conversions. Fill unspecified entries with NaN.
CouponRates	Matrix indicating the coupon rates for each bond in decimal form. Size = number of instruments by maximum number of conversions + 1. First column of this matrix contains rates applicable between Settle and the first conversion date (date in the first column of ConvDates). Fill unspecified entries with NaN. See Note below.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2, 3, 4, 6, and 12. Default = 2.

Basis	(Optional) Day-count basis of the instrument. A vector of integers.
	• 0 = actual/actual (default)
	• 1 = 30/360 (SIA)
	• 2 = actual/360
	• 3 = actual/365
	• 4 = 30/360 (BMA)
	• 5 = 30/360 (ISDA)
	• 6 = 30/360 (European)
	• 7 = actual/365 (Japanese)
	• 8 = actual/actual (ICMA)
	• 9 = actual/360 (ICMA)
	• 10 = actual/365 (ICMA)
	• 11 = 30/360E (ICMA)
	• 12 = actual/actual (ISDA)
	• 13 = BUS/252
	For more information, see **basis** on page Glossary-1.
EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
Face	(Optional) Face value of each bond in the portfolio. Default = 100.

All arguments must be number of bonds (NUMBONDS)-by-1 vectors, except for ConvDates and CouponRate.

Note ConvDates has the same number of rows as CouponRate to reflect the same number of bonds. However, ConvDates has one less column than CouponRate. This situation is illustrated by

Settle---------ConvDate1-----------ConvDate2-----------Maturity

        Rate1   Rate2   Rate3

Description Yield = stepcpnyield(Price, Settle, Maturity, ConvDates, CouponRate, Period, Basis, EndMonthRule, Face) computes the yield to maturity of bonds with stepped coupons given the price. The function supports any number of conversion dates.

Yield is a NUMBONDS-by-1 vector of yields to maturity in decimal form.

Note For bonds with fixed coupons, use bndyield. You will receive the error incorrect number of inputs if you use a fixed-coupon bond with stepcpnyield.

Examples Find the yield to maturity of three stepped-coupon bonds of known price, given three conversion scenarios:

• Bond A has two conversions, the first one falling on the settle date and immediately expiring.

• Bond B has three conversions, with conversion dates exactly on the coupon dates.

• Bond C has three conversions, with one or more conversion dates not on coupon dates. This case illustrates that only cash flows for full periods after conversion dates are affected, as illustrated below.

The following table illustrates the interest rate characteristics of this bond portfolio.

Bond A Dates	Bond A Rates	Bond B Dates	Bond B Rates	Bond C Dates	Bond C Rates
Settle (02-Aug-92)	7.5%	Settle (02-Aug-92)	7.5%	Settle (02-Aug-92)	7.5%
First Conversion (02-Aug-92)	8.875%	First Conversion (15-Jun-97)	8.875%	First Conversion (14-Jun-97)	8.875%
Second Conversion (15-Jun-03)	9.25%	Second Conversion (15-Jun-01)	9.25%	Second Conversion (14-Jun-01)	9.25%
Maturity (15-Jun-10)	NaN	Third Conversion (15-Jun-05)	10.0%	Third Conversion (14-Jun-05)	10.0%
		Maturity (15-Jun-10)	NaN	Maturity (15-Jun-10)	NaN

```
format long
Price = [117.3824; 113.4339; 113.4339];
Settle = datenum('02-Aug-1992');

ConvDates = [datenum('02-Aug-1992'), datenum('15-Jun-2003'), nan;
datenum('15-Jun-1997'), datenum('15-Jun-2001'), datenum('15-Jun-2005');
datenum('14-Jun-1997'), datenum('14-Jun-2001'), datenum('14-Jun-2005')];

Maturity = datenum('15-Jun-2010');

CouponRates = [0.075 0.08875 0.0925 nan;
```

```
                        0.075 0.08875 0.0925 0.1;
                        0.075 0.08875 0.0925 0.1];
Basis = 1;
Period = 2;
EndMonthRule = 1;
Face = 100;

Yield = stepcpnyield(Price, Settle, Maturity, ConvDates, ...
CouponRates, Period, Basis, EndMonthRule, Face)

Yield =

0.07221440204915
0.07221426780036
0.07221426780036
```

References   This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 120 - 123, on zero-coupon instruments pricing.

See Also   `stepcpnprice` | `bndprice` | `cdprice` | `stepcpncfamounts` | `stepcpnprice` | `tbillprice` | `zeroprice`

Related Examples
- "Cash Flows from Stepped-Coupon Bonds" on page 6-12
- "Price and Yield of Stepped-Coupon Bonds" on page 6-14

tfutbyprice

Purpose	Future prices of Treasury bonds given spot price

Syntax

```
[QtdFutPrice, AccrInt] = tfutbyprice(SpotCurve, Price, SettleFut, MatFut,
ConvFactor, CouponRate, Maturity, Interpolation)
```

Arguments

SpotCurve	Treasury spot curve; a number of futures (NFUT) by 3 matrix in the form of [SpotDates SpotRates Compounding]. Allowed compounding values are -1, 1, 2 (default), 3, 4, and 12, where -1 is continuous compounding.
Price	Scalar or vector containing prices of Treasury bonds or notes per \$100 notional. Use bndprice for theoretical value of bond.
SettleFut	Scalar or vector of identical elements containing settlement date of futures contract.
MatFut	Scalar or vector containing maturity dates (or anticipated delivery dates) of futures contract.
ConvFactor	Conversion factor. See convfactor.
CouponRate	Scalar or vector containing underlying bond annual coupon in decimal.
Maturity	Scalar or vector containing underlying bond maturity.
Interpolation	(Optional) Interpolation method. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See interp1 for more information.

Inputs (except SpotCurve) must either be a scalar or a vector of size
equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

Description

```
[QtdFutPrice, AccrInt] = tfutbyprice(SpotCurve, Price,
SettleFut, MatFut, ConvFactor, CouponRate, Maturity,
```

Interpolation) computes future prices of Treasury notes and bonds given the spot price. The output arguments are:

- QtdFutPrice — Quoted futures price, per $100 notional.

- AccrInt — Accrued interest due at delivery date, per $100 notional.

In addition, you can use the Financial Instruments Toolbox method getZeroRates for an IRDataCurve object with a Dates property to create a vector of dates and data acceptable for tfutbyprice. For more information, see "Converting an IRDataCurve or IRFunctionCurve Object" on page 9-38.

Examples
Determine the future price of two Treasury bonds based upon a spot rate curve constructed from data for November 14, 2002.

```
% Constructing spot curve from Nov 14, data
Bonds = [datenum('02/13/2003'),        0;
         datenum('05/15/2003'),        0;
         datenum('10/31/2004'),  0.02125;
         datenum('11/15/2007'),     0.03;
         datenum('11/15/2012'),     0.04;
         datenum('02/15/2031'),  0.05375];

Yields  = [1.20; 1.25; 1.86; 2.99; 4.02; 4.93]/100;

Settle = datenum('11/15/2002');

[ZeroRates, CurveDates] = ...
zbtyield(Bonds, Yields, Settle);

SpotCurve  = [CurveDates, ZeroRates];

% Calculating a particular bond's future quoted price
RefDate    = [datenum('1-Dec-2002'); datenum('1-Mar-2003')];
MatFut     = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
Maturity   = [datenum('15-Aug-2009');datenum('15-Aug-2010')];
CouponRate = [0.06;0.0575];
```

```
ConvFactor = convfactor(RefDate, Maturity, CouponRate);
Price = [114.416; 113.171];
Interpolation = 1;

[QtdFutPrice, AccrInt] = tfutbyprice(SpotCurve, Price, Settle, ...
MatFut, ConvFactor, CouponRate, Maturity, Interpolation)

QtdFutPrice =

  114.0409
  113.4029

AccrInt =
        1.9891
        0.4448
```

See Also   convfactor | tfutbyyield

Related Examples
• "Computing Treasury Bill Price and Yield"

Concepts   • "Treasury Bills Defined"

Purpose	Future prices of Treasury bonds given current yield

Syntax   [QtdFutPrice, AccrInt] = tfutbyyield(SpotCurve, Yield, SettleFut, MatF
ConvFactor, CouponRate, Maturity, Interpolation)

Arguments

SpotCurve	Treasury spot curve. A number of futures (NFUT)-by-3 matrix in the form of [SpotDates SpotRates Compounding].
	Allowed compounding values are -1, 1, 2 (default), 3, 4, and 12, where -1 is continuous compounding.
Yield	Scalar or vector containing yield to maturity of bonds. Use bndyield for theoretical value of bond yield.
SettleFut	Scalar or vector of identical elements containing settlement date of futures contract.
MatFut	Scalar or vector containing maturity dates (or anticipated delivery dates) of futures contract.
ConvFactor	Conversion factor. See convfactor.
CouponRate	Scalar or vector containing underlying bond annual coupon in decimal.
Maturity	Scalar or vector containing underlying bond maturity.
Interpolation	(Optional) Interpolation method. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See interp1 for more information.

Inputs (except SpotCurve) must either be a scalar or a vector of size
equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

Description   [QtdFutPrice, AccrInt] = tfutbyyield(SpotCurve, Yield,
SettleFut, MatFut, ConvFactor, CouponRate, Maturity,
Interpolation) computes future prices of Treasury notes and bonds

given current yields of Treasury bonds/notes. The output arguments are:

- QtdFutPrice — Quoted futures price, per $100 notional.
- AccrInt — Accrued Interest due at delivery date, per $100 notional.

In addition, you can use the Financial Instruments Toolbox method getZeroRates for an IRDataCurve object with a Dates property to create a vector of dates and data acceptable for tfutbyyield. For more information, see "Converting an IRDataCurve or IRFunctionCurve Object" on page 9-38.

Examples   Determine the future price of two Treasury bonds based upon a spot rate curve constructed from data for November 14, 2002.

```
% Constructing spot curve from Nov 14, data
Bonds = [datenum('02/13/2003'),         0;
         datenum('05/15/2003'),         0;
         datenum('10/31/2004'),   0.02125;
         datenum('11/15/2007'),      0.03;
         datenum('11/15/2012'),      0.04;
         datenum('02/15/2031'),   0.05375];

Yields  = [1.20; 1.25; 1.86; 2.99; 4.02; 4.93]/100;

Settle = datenum('11/15/2002');

[ZeroRates, CurveDates] = ...
zbtyield(Bonds, Yields, Settle);

SpotCurve  = [CurveDates, ZeroRates];

% Calculating a particular bond's future quoted price
RefDate    = [datenum('1-Dec-2002'); datenum('1-Mar-2003')];
MatFut     = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
Maturity   = [datenum('15-Aug-2009');datenum('15-Aug-2010')];
CouponRate = [0.06;0.0575];
```

```
ConvFactor = convfactor(RefDate, Maturity, CouponRate);
Yield = [0.03576; 0.03773];
Interpolation = 1;

[QtdFutPrice, AccrInt] = tfutbyyield(SpotCurve, Yield, Settle, ...
MatFut, ConvFactor, CouponRate, Maturity, Interpolation)

QtdFutPrice =

  114.0416
  113.4034

AccrInt =
        1.9891
        0.4448
```

See Also   convfactor | tfutbyprice

**Related
Examples**   • "Computing Treasury Bill Price and Yield"

Concepts   • "Treasury Bills Defined"

tfutimprepo

Purpose   Implied repo rates for Treasury bond future given price

Syntax   ImpliedRepo = tfutimprepo(ReinvestData, Price, QtdFutPrice, Settle, MatFut, ConvFactor, CouponRate, Maturity)

Arguments

ReinvestData	Number of futures (NFUT) by 2 matrix of rates and bases for the reinvestment of intervening coupons in the form of [ReinvestRate ReinvestBasis]. ReinvestRate is the simple reinvestment rate, in decimal. Specify ReinvestBasis as 0 = not reinvested, 2 = actual/360, or 3 = actual/365.
Price	Current bond price per \$100 notional.
QtdFutPrice	Quoted bond futures price per \$100 notional.
Settle	Settlement/valuation date of futures contract.
MatFut	Maturity date (or anticipated delivery dates) of futures contract.
ConvFactor	Conversion factor. See convfactor.
CouponRate	Underlying bond annual coupon, in decimal.
Maturity	Underlying bond maturity date.

Inputs (except ReinvestData) must either be a scalar or a vector of size equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

Description   ImpliedRepo = tfutimprepo(ReinvestData, Price, QtdFutPrice, Settle, MatFut, ConvFactor, CouponRate, Maturity) computes the implied repo rate that prevents arbitrage of Treasury bond futures, given the clean price at the settlement and delivery dates.

ImpliedRepo is the implied annual repo rate, in decimal, with an actual/360 basis.

Examples   Compute the implied repo rate given the following set of data.

```
ReinvestData = [0.018  3];
Price = [114.4160; 113.1710];
QtdFutPrice = [114.1201; 113.7090];
Settle = datenum('11/15/2002');
MatFut = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
ConvFactor = [1; 0.9854];
CouponRate = [0.06; 0.0575];
Maturity = [datenum('15-Aug-2009'); datenum('15-Aug-2010')];

ImpliedRepo = tfutimprepo(ReinvestData, Price, QtdFutPrice, ...
Settle, MatFut, ConvFactor, CouponRate, Maturity)

ImpliedRepo =

    0.0200
    0.0200
```

See Also   tfutpricebyrepo | tfutyieldbyrepo

Related Examples   • "Computing Treasury Bill Price and Yield"

Concepts   • "Treasury Bills Defined"

tfutpricebyrepo

Purpose	Calculates Treasury bond futures price given the implied repo rates

Syntax

```
[QtdFutPrice AccrInt] = tfutpricebyrepo(RepoData, ReinvestData,
Price, Settle, MatFut, ConvFactor, CouponRate, Maturity)
```

Arguments

RepoData	Number of futures (NFUT) by 2 matrix of simple term repo/funding rates in decimal and their bases in the form of [RepoRate RepoBasis]. Specify RepoBasis as 2 = actual/360 or 3 = actual/365.
ReinvestData	Number of futures (NFUT) by 2 matrix of rates and bases for the reinvestment of intervening coupons in the form of [ReinvestRate ReinvestBasis]. ReinvestRate is the simple reinvestment rate, in decimal. Specify ReinvestBasis as 0 = not reinvested, 2 = actual/360, or 3 = actual/365.
Price	Quoted clean prices of Treasury bonds per $100 notional at Settle.
Settle	Settlement/valuation date of futures contract.
MatFut	Maturity date (or anticipated delivery dates) of futures contract.
ConvFactor	Conversion factor. See convfactor.
CouponRate	Underlying bond annual coupon, in decimal.
Maturity	Underlying bond maturity date.

Inputs (except RepoData and ReinvestData) must either be a scalar or a vector of size equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

Description

[QtdFutPrice AccrInt] = tfutpricebyrepo(RepoData, ReinvestData, Price, Settle, MatFut, ConvFactor,

CouponRate, Maturity) computes the theoretical futures bond price given the settlement price, the repo/funding rates, and the reinvestment rate.

`QtdFutPrice` is the quoted futures price, per $100 notional.

`AccrInt` is the accrued interest due at the delivery date, per $100 notional.

Examples Compute the quoted futures price and accrued interest due on the target delivery date, given the following data.

```
RepoData     = [0.020  2];
ReinvestData = [0.018  3];
Price        = [114.416; 113.171];
Settle       = datenum('11/15/2002');
MatFut       = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
ConvFactor   = [1 ; 0.9854];
CouponRate   = [0.06;0.0575];
Maturity     = [datenum('15-Aug-2009'); datenum('15-Aug-2010')];

[QtdFutPrice AccrInt] = tfutpricebyrepo(RepoData, ...
ReinvestData, Price, Settle, MatFut, ConvFactor, CouponRate, ...
Maturity)

QtdFutPrice =

  114.1201
  113.7090

AccrInt =

    1.9891
    0.4448
```

See Also tfutimprepo | tfutyieldbyrepo

tfutpricebyrepo

Related Examples	• "Computing Treasury Bill Price and Yield"
Concepts	• "Treasury Bills Defined"

Purpose	Calculates Treasury bond futures yield given the implied repo rates

Syntax

FwdYield = tfutyieldbyrepo(RepoData, ReinvestData, Yield, Settle, MatFut, ConvFactor, CouponRate, Maturity)

Arguments

RepoData	Number of futures (NFUT) by 2 matrix of simple term repo/funding rates in decimal and their bases in the form of [RepoRate RepoBasis]. Specify RepoBasis as 2 = actual/360 or 3 = actual/365.
ReinvestData	Number of futures (NFUT) by 2 matrix of rates and bases for the reinvestment of intervening coupons in the form of [ReinvestRate ReinvestBasis]. ReinvestRate is the simple reinvestment rate, in decimal. Specify ReinvestBasis as 0 = not reinvested, 2 = actual/360, or 3 = actual/365.
Yield	Yield to maturity of Treasury bonds per $100 notional at Settle.
Settle	Settlement/valuation date of futures contract.
MatFut	Maturity date (or anticipated delivery dates) of futures contract.
ConvFactor	Conversion factor. See convfactor.
CouponRate	Underlying bond annual coupon, in decimal.
Maturity	Underlying bond maturity date.

Inputs (except RepoData and ReinvestData) must either be a scalar or a vector of size equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

Description

FwdYield = tfutyieldbyrepo(RepoData, ReinvestData, Yield, Settle, MatFut, ConvFactor, CouponRate, Maturity) computes

tfutyieldbyrepo

the theoretical futures bond yield given the settlement yield, the repo/funding rate, and the reinvestment rate.

FwdYield is the forward yield to maturity, in decimal, compounded semiannually.

Examples   Compute the quoted futures bond yield, given the following data:

```
RepoData     = [0.020  2];
ReinvestData = [0.018  3];
Yield        = [0.0215; 0.0257];
Settle       = datenum('11/15/2002');
MatFut       = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
ConvFactor   = [1; 0.9854];
CouponRate   = [0.06; 0.0575];
Maturity     = [datenum('15-Aug-2009'); datenum('15-Aug-2010')];

FwdYield = tfutyieldbyrepo(RepoData, ReinvestData, Yield,...
Settle, MatFut, ConvFactor, CouponRate, Maturity)

FwdYield =

    0.0221
    0.0282
```

See Also   tfutimprepo | tfutpricebyrepo

Related Examples   • "Computing Treasury Bill Price and Yield"

Concepts   • "Treasury Bills Defined"

Purpose	Convert `IRDataCurve` object to `RateSpec`

Class   `@IRDataCurve`

Syntax   `F = toratespec(CurveObj, InpDates)`

Arguments

CurveObj	Interest-rate curve object that is constructed using `IRDataCurve`.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.

Description   `F = toratespec(CurveObj, InpDates)` returns a `RateSpec` object that is identical to the `RateSpec` structure created by the Financial Instruments Toolbox function `intenvset`.

Examples   This example creates an `IRDataCurve` object from the `IRDataCurve` constructor using `Dates` and `Data` and then is converted to a `RateSpec` structure using the `toRateSpec` method:

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Forward',today,Dates,Data)
irdc.toRateSpec(today+30:30:today+365)

irdc =

  IRDataCurve handle

  Properties:
          Dates: [8x1 double]
           Data: [8x1 double]
    InterpMethod: 'linear'
           Type: 'Forward'
         Settle: 733596
     Compounding: 2
```

```
                    Basis: 0

        Methods, Events, Superclasses

    ans =

              FinObj: 'RateSpec'
         Compounding: 2
                Disc: [12x1 double]
               Rates: [12x1 double]
            EndTimes: [12x1 double]
          StartTimes: [12x1 double]
            EndDates: [12x1 double]
          StartDates: 733596
       ValuationDate: 733596
               Basis: 0
         EndMonthRule: 1
```

How To   • "@IRDataCurve" on page A-7

Purpose	Convert `IRFunctionCurve` object to `RateSpec`

Class @IRFunctionCurve

Syntax F = toRateSpec(CurveObj, InpDates)

Arguments

CurveObj	Interest-rate curve object that is constructed using `IRFunctionCurve`.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.

Description F = toRateSpec(CurveObj, InpDates) returns a `RateSpec` object that is identical to the `RateSpec` structure created by the Financial Instruments Toolbox function `intenvset`.

Examples This example creates an `IRFunctionCurve` object using the `IRFunctionCurve` constructor and then a `RateSpec` structure is created using the `toRateSpec` method:

```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));
irfc.toRateSpec(today+30:30:today+365)

ans =

              FinObj: 'RateSpec'
         Compounding: 2
                Disc: [12x1 double]
               Rates: [12x1 double]
            EndTimes: [12x1 double]
          StartTimes: [12x1 double]
            EndDates: [12x1 double]
          StartDates: 733596
       ValuationDate: 733596
               Basis: 0
         EndMonthRule: 1
```

toRateSpec

How To   • "@IRFunctionCurve" on page A-12

Purpose	Price zero-coupon instruments given yield
Syntax	Price = zeroprice(Yield, Settle, Maturity, Period, Basis, EndMonthRule)

Arguments

Yield	Scalar or vector containing yield to maturity of instruments.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Scalar or vector specifying number of quasi-coupons per year. Default = 2.
Basis	(Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)

- 12 = actual/actual (ISDA)

- 13 = BUS/252

  For more information, see **basis** on page Glossary-1.

EndMonthRule   (Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

Description   Price = zeroprice(Yield, Settle, Maturity, Period, Basis, EndMonthRule) calculates the prices for a portfolio of general short and long term zero-coupon instruments given the yield of reference bonds. In other words, if the zero-coupon computed with this yield is used to discount the reference bond, the value of that reference bond will be equal its price.

Price is a column vector containing a price for each zero-coupon instrument per $100 notional.

When there is less than one quasi-coupon, the function uses a simple yield based upon (Period) x (Number of Days in quasi coupon period) day-year. The default period is 2 and the default number of days is 180, which makes the user-supplied yield a simple yield on a 360-day year.

For longer term computations (more than one quasi-coupon), use the bond equivalent yield based upon present value (or compounding).

Algorithms   To compute the price when there is 1 or 0 quasi-coupon periods to redemption, zeroprice uses the formula

$$Price = \frac{RV}{1 + \left(\dfrac{DSR}{E} \times \dfrac{Y}{M} \right)}$$

.

Quasi-coupon periods are the coupon periods that would exist if the bond were paying interest at a rate other than zero.

When there is more than one quasi-coupon period to the redemption date, zeroprice uses the formula

$$Price = \frac{RV}{\left(1 + \dfrac{Y}{M} \right)^{N_q - 1 + \frac{DSC}{E}}}$$

.

The elements of the equations are defined as follows.

Variable	Definition
DSC	Number of days from settlement date to next quasi-coupon date as if the security paid periodic interest.
DSR	Number of days from settlement date to the redemption date (call date, put date, and so on).
E	Number of days in quasi-coupon period.
M	Number of quasi-coupon periods per year (standard for the particular security involved).
Nq	Number of quasi-coupon periods between settlement date and redemption date. If this number contains a fractional part, raise it to the next whole number.
Price	Dollar price per $100 par value.
RV	Redemption value.
Y	Annual yield (decimal) when held to redemption.

zeroprice

Examples   **Example 1**. Compute the price of a short-term zero-coupon instrument.

```
Settle = '24-Jun-1993';
Maturity = '1-Nov-1993';
Period = 2;
Basis = 0;
Yield = 0.04;

Price = zeroprice(Yield, Settle, Maturity, Period, Basis)

Price =

   98.6066
```

Example 2. Compute the prices of a portfolio of two zero-coupon instruments, one short-term, and the other long-term.

```
Settle = '24-Jun-1993';
Maturity = ['01-Nov-1993'; '15-Jan-2024'];
Basis = [0; 1];
Yield = [0.04; 0.1];

Price = zeroprice(Yield, Settle, Maturity, [], Basis)

Price =

   98.6066
    5.0697
```

References   [1] Mayle, Jan. *Standard Securities Calculation Methods*. New York: Securities Industry Association, Inc. Vol. 1, 3rd ed., 1993, ISBN 1-882936-01-9. Vol. 2, 1994, ISBN 1-882936-02-7.

See Also   bndprice | cdprice | tbillprice | zeroyield

Related Examples
- "Computing Treasury Bill Price and Yield"
- "Pricing Treasury Notes" on page 6-8
- "Pricing Corporate Bonds" on page 6-10

Concepts
- "Measuring Zero-Coupon Bond Function Quality" on page 6-7

zeroyield

Purpose	Yield of zero-coupon instruments given price
Syntax	Yield = zeroyield(Price, Settle, Maturity, Period, Basis, EndMonthRule)

Arguments

Price	Scalar or vector containing prices of instruments.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Scalar or vector specifying number of quasi-coupons per year. Default = 2.
Basis	(Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see **basis** on page Glossary-1.

EndMonthRule (Optional) End-of-month rule. A vector. This rule applies only when Maturity is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

Description Yield = zeroyield(Price, Settle, Maturity, Period, Basis, EndMonthRule) calculates the bond-equivalent yield for a portfolio of general short and long term zero-coupon instruments given the price of the instruments. In other words, if the zero-coupon computed with this yield is used to discount the reference bond, the value of that reference bond will be equal its price.

Yield is a column vector containing a bond-equivalent yield for each zero-coupon instrument.

When the maturity date is fewer than 182 days away and the basis is actual/365, the function uses a simple-interest algorithm. If maturity is more than 182 days away, the function uses present value calculations.

When the basis is actual/360, the simple interest algorithm gives the money-market yield for short (1 to 6 months to maturity) Treasury bills.

The present value algorithm always gives the bond equivalent yield of the zero-coupon instrument. The algorithm is equivalent to calling bndyield with the zero-coupon information within one basis point.

Algorithms To compute the yield when there is zero or one quasi-coupon periods to redemption, zeroyield uses the formula

$$Yield = \left(\frac{RV - P}{P}\right) \times \left(\frac{M \times E}{DSR}\right)$$

.

Quasi-coupon periods are the coupon periods which would exist if the bond was paying interest at a rate other than zero. The first term calculates the yield on invested dollars. The second term converts this yield to a per annum basis.

When there is more than one quasi-coupon period to the redemption date, `zeroyield` uses the formula

$$Yield = \left(\left(\frac{RV}{P}\right)^{\frac{1}{Nq-1+\frac{DSC}{E}}} -1 \right) \times M$$

.

The elements of the equations are defined as follows.

Variable	Definition
DSC	Number of days from the settlement date to next quasi-coupon date as if the security paid periodic interest.
DSR	Number of days from the settlement date to redemption date (call date, put date, and so on).
E	Number of days in quasi-coupon period.
M	Number of quasi-coupon periods per year (standard for the particular security involved).
Nq	Number of quasi-coupon periods between the settlement date and redemption date. If this number contains a fractional part, raise it to the next whole number.
P	Dollar price per $100 par value.

Variable	Definition
RV	Redemption value.
Yield	Annual yield (decimal) when held to redemption.

Examples

Example 1. Compute the yield of a short-term zero-coupon instrument.

```
Settle   = '24-Jun-1993';
Maturity = '1-Nov-1993';
Basis    = 0;
Price    = 95;

Yield = zeroyield(Price, Settle, Maturity, [], Basis)

Yield =

    0.1490
```

Example 2. Recompute the yield of the same instrument using a different day-count basis.

```
Settle   = '24-Jun-1993';
Maturity = '1-Nov-1993';
Basis    = 1;
Price    = 95;

Yield = zeroyield(Price, Settle, Maturity, [], Basis)

Yield =

    0.1492
```

Example 3. Compute the yield of a long-term zero-coupon instrument.

```
Settle   = '24-Jun-1993';
Maturity = '15-Jan-2024';
Basis    = 0;
```

```
Price   = 9;

Yield = zeroyield(Price, Settle, Maturity, [], Basis)

Yield =

    0.0804
```

References   [1] Mayle, Jan. *Standard Securities Calculation Methods.* New York: Securities Industry Association, Inc. Vol. 1, 3rd ed., 1993, ISBN 1-882936-01-9. Vol. 2, 1994, ISBN 1-882936-02-7.

See Also   bndyield | cdyield | tbillyield | zeroprice

Related Examples
- "Computing Treasury Bill Price and Yield"
- "Pricing Treasury Notes" on page 6-8
- "Pricing Corporate Bonds" on page 6-10

Concepts
- "Measuring Zero-Coupon Bond Function Quality" on page 6-7

Derivatives Pricing Options

- "Pricing Options Structure" on page B-2
- "Customizing the Structure" on page B-5

Pricing Options Structure

In this section...
"Introduction" on page B-2
"Default Structure" on page B-2

Introduction

The MATLAB `Options` structure provides additional input to most pricing functions. The `Options` structure

- Tells pricing functions how to use the interest-rate tree to calculate instrument prices.

- Determines what additional information the Command Window displays along with instrument prices.

- Tells pricing functions which method to use in pricing barrier options.

The pricing options structure is primarily used in the pricing of interest-rate-based financial derivatives. However, the `BarrierMethod` field in the structure allows you to use it in pricing equity barrier options as well.

You provide pricing options in an optional `Options` argument passed to a pricing function. (See, for example, `bondbyhjm`, `bdtprice`, `barrierbycrr`, `barrierbyeqp`, or `barrierbyitt`.)

Default Structure

If you do not specify the `Options` argument in the call to a pricing function, the function uses a default structure. To observe the default structure, use `derivset` without any arguments.

```
Options = derivset

Options =

    Diagnostics: 'off'
       Warnings: 'on'
      ConstRate: 'on'
```

```
BarrierMethod: 'unenhanced'
```

The `Options` structure has four fields: `Diagnostics`, `Warnings`, `ConstRate`, and `BarrierMethod`.

Diagnostics Field

`Diagnostics` indicates whether additional information is displayed if the tree is modified. The default value for this option is `'off'`. If `Diagnostics` is set to `'on'` and `ConstRate` is set to `'off'`, the pricing functions display information such as the number of nodes in the last level of the tree generated for pricing purposes.

Warnings Field

`Warnings` indicates whether to display warning messages when the input tree is not adequate for accurately pricing the instruments. The default value for this option is `'on'`. If both `ConstRate` and `Warnings` are `'on'`, a warning is displayed if any of the instruments in the input portfolio has a cash flow date between tree dates. If `ConstRate` is `'off'`, and `Warnings` is `'on'`, a warning is displayed if the tree is modified to match the cash flow dates on the instruments in the portfolio.

ConstRate Field

`ConstRate` indicates whether the interest rates should be assumed constant between tree dates. By default this option is `'on'`, which is not an arbitrage-free assumption. Consequently the pricing functions return an approximate price for instruments featuring cash flows between tree dates. Instruments featuring cash flows only on tree nodes are not affected by this option and return exact (arbitrage-free) prices. When `ConstRate` is `'off'`, the pricing function finds the cash flow dates for all instruments in the portfolio. If these cash flows do not align exactly with the tree dates, a new tree is generated and used for pricing. This new tree features the same volatility and initial rate specifications of the input tree but contains tree nodes for each date in which at least one instrument in the portfolio has a cash flow. Keep in mind that the number of nodes in a tree grows exponentially with the number of tree dates. Consequently, setting `ConstRate` `'off'` dramatically increases the memory and processor demands on the computer.

BarrierMethod Field

When using binomial trees to price barrier options, you may require a large number of tree steps to achieve an accurate result when tree nodes do not align with the barrier level. With the `BarrierMethod` field, the toolbox provides an enhancement method that improves the accuracy of the results without having to use large trees.

The `BarrierMethod` field can be set to `'unenhanced'` (default) or `'interp'`. If you specify `'unenhanced'`, no correction calculation is used. Otherwise, if you specify `'interp'`, the toolbox provides an enhanced valuation by interpolating between nodes on barrier boundaries.

You specify the barrier method in the last input argument, `Options`, of the functions `barrierbycrr`, `barrierbyeqp`, `crrprice`, or `eqpprice`. `Options` is a structure that you create with the function `derivset`. Using `derivset`, you specify whether to use the enhanced or the unenhanced method.

For more information about this algorithm, see Derman, E., I. Kani, D. Ergener and I. Bardhan, "Enhanced Numerical Methods for Options with Barriers," *Financial Analysts Journal*, (Nov. - Dec. 1995), pp. 65-74.

Customizing the Structure

Customize the Options structure by passing property name/property value pairs to the `derivset` function.

As an example, consider an Options structure with `ConstRate` `'off'` and `Diagnostics` `'on'`.

```
Options = derivset('ConstRate', 'off', 'Diagnostics', 'on')

Options =

  Diagnostics: 'on'
     Warnings: 'on'
    ConstRate: 'off'
BarrierMethod: 'unenhanced'
```

To obtain the value of a specific property from the Options structure, use `derivget`.

```
CR = derivget(Options, 'ConstRate')

CR =
Off
```

Note Use `derivset` and `derivget` to construct the Options structure. These functions are guaranteed to remain unchanged, while the implementation of the structure itself may be modified in the future.

Now observe the effects of setting `ConstRate` `'off'`. Obtain the tree dates from the HJM tree.

```
TreeDates = [HJMTree.TimeSpec.ValuationDate;...
HJMTree.TimeSpec.Maturity]

TreeDates =

    730486
```

```
        730852
        731217
        731582
        731947

datedisp(TreeDates)

01-Jan-2000
01-Jan-2001
01-Jan-2002
01-Jan-2003
01-Jan-2004
```

All instruments in `HJMInstSet` settle on January 1, 2000, and all have cash flows once a year, with the exception of the second bond, which features a period of 2. This bond has cash flows twice a year, with every other cash flow consequently falling between tree dates. You can extract this bond from the portfolio to compare how its price differs by setting `ConstRate` to `'on'` and `'off'`.

```
BondPort = instselect(HJMInstSet, 'Index', 2);

instdisp(BondPort)

Index Type CouponRate Settle      Maturity    Period Basis...
1     Bond 0.04       01-Jan-2000 01-Jan-2004 2      NaN...
```

First price the bond with `ConstRate` `'on'` (default).

```
format long
[BondPrice, BondPriceTree] = hjmprice(HJMTree, BondPort)
Warning: Not all cash flows are aligned with the tree. Result will
be approximated.

BondPrice =

  97.52801411736377

BondPriceTree =
FinObj: 'HJMPriceTree'
```

```
 PBush: {1x5 cell}
AIBush: {[0]  [1x1x2 double] ... [1x4x2 double]  [1x8 double]}
  tObs: [0 1 2 3 4]
```

Now recalculate the price of the bond setting `ConstRate` `'off'`.

```
OptionsNoCR = derivset('ConstR', 'off')

OptionsNoCR =

Diagnostics: 'off'
   Warnings: 'on'
  ConstRate: 'off'

[BondPriceNoCR, BondPriceTreeNoCR] = hjmprice(HJMTree,...
BondPort, OptionsNoCR)
Warning: Not all cash flows are aligned with the tree. Rebuilding
tree.

BondPriceNoCR =

  97.53342361674437

BondPriceTreeNoCR =

FinObj: 'HJMPriceTree'
 PBush: {1x9 cell}
AIBush: {1x9 cell}
  tObs: [0 0.5000 1 1.5000 2 2.5000 3 3.5000 4]
```

As indicated in the last warning, because the cash flows of the bond did not
align with the tree dates, a new tree was generated for pricing the bond.
This pricing method returns more accurate results since it guarantees that
the process is arbitrage-free. It also takes longer to calculate and requires
more memory. The `tObs` field of the price tree structure indicates the
increased memory usage. `BondPriceTree.tObs` has only five elements, while
`BondPriceTreeNoCR.tObs` has nine. While this may not seem like a large
difference, it has a dramatic effect on the number of states in the last node.

```
size(BondPriceTree.PBush{end})
```

```
ans =

     1 8

size(BondPriceTreeNoCR.PBush{end})

ans =

     1 128
```

The differences become more obvious by examining the price trees with `treeviewer`.

```
treeviewer(BondPriceTree, BondPort)
```
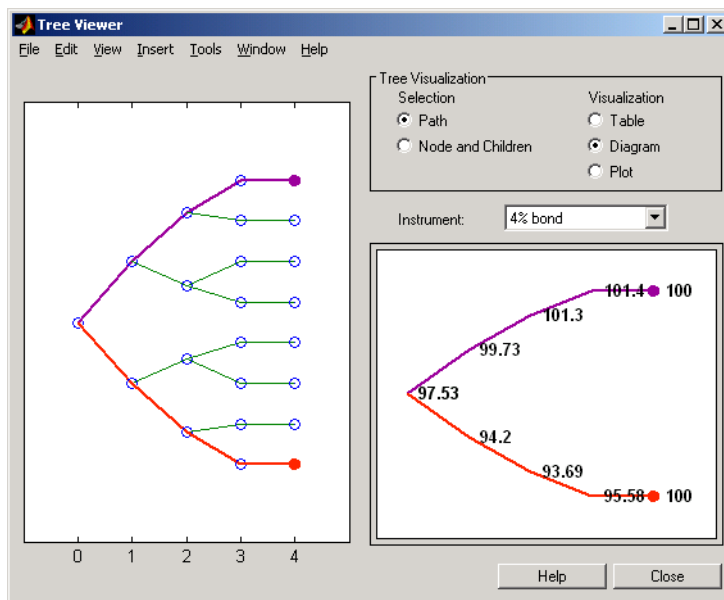


```
treeviewer(BondPriceTreeNoCR, BondPort)
```
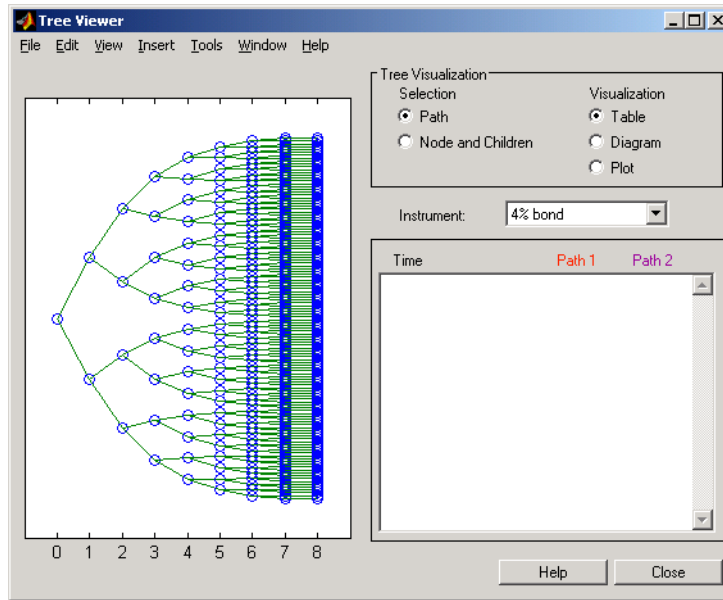
```
All = [Delta ./ Price, Gamma ./ Price, Vega ./ Price, Price]

All =

          -2.76         10.43          0.00         98.72
          -3.56         16.64         -0.00         97.53
        -166.18      13235.59        700.96          0.05
          -2.76         10.43          0.00         98.72
          -0.01          0.03             0        100.55
          46.95       1090.63         14.91          6.28
        -969.85     173969.77       1926.72          0.05
         -76.39        287.00          0.00         3.690
```

C

Bibliography

Black-Derman-Toy (BDT) Modeling

A description of the Black-Derman-Toy interest-rate model can be found in:

Black, Fischer, Emanuel Derman, and William Toy, "A One Factor Model of Interest Rates and its Application to Treasury Bond Options," *Financial Analysts Journal*, January - February 1990.

Heath-Jarrow-Morton (HJM) Modeling

An introduction to Heath-Jarrow-Morton modeling, used extensively in Financial Instruments Toolbox software, can be found in:

Jarrow, Robert A., *Modelling Fixed Income Securities and Interest Rate Options*, McGraw-Hill, 1996, ISBN 0-07-912253-1.

Hull-White (HW) and Black-Karasinski (BK) Modeling

A description of the Hull-White model and its Black-Karasinski modification can be found in:

Hull, John C., *Options, Futures, and Other Derivatives*, Prentice-Hall, 1997, ISBN 0-13-186479-3.

You can find additional information about the Hull-White single-factor model used in this toolbox in these papers:

Hull, J., and A. White, "Numerical Procedures for Implementing Term Structure Models I: Single-Factor Models," *Journal of Derivatives,* 1994.

Hull, J., and A. White, "Using Hull-White Interest Rate Trees," *Journal of Derivatives*, 1996.

Cox-Ross-Rubinstein (CRR) Modeling

To learn about the Cox-Ross-Rubinstein model, see:

Cox, J. C., S. A. Ross, and M. Rubinstein, "Option Pricing: A Simplified Approach," *Journal of Financial Economics*, Number 7, 1979, pp. 229-263.

Implied Trinomial Tree (ITT) Modeling

To learn about the Implied Trinomial Tree model, see:

Chriss, Neil A., E. Derman, and I. Kani, "Implied trinomial trees of the volatility smile," *Journal of Derivatives*, 1996.

Leisen-Reimer Tree (LR) Modeling

To learn about the Leisen-Reimer model, see:

Leisen D.P., M. Reimer, "Binomial Models for Option Valuation – Examining and Improving Convergence," *Applied Mathematical Finance*, Number 3, 1996, pp. 319-346.

Equal Probabilities Tree (EQP) Modeling

To learn about the Equal Probabilities model, see:

Chriss, Neil A.,*Black Scholes and Beyond: Option Pricing Models*, McGraw-Hill, 1996, ISBN 0-7863-1025-1.

Closed-Form Solutions Modeling

To learn about the Bjerksund-Stensland 2002 model, see:

Bjerksund, P. and G. Stensland, *Closed-Form Approximation of American Options*, Scandinavian Journal of Management, 1993, Vol. 9, Suppl., pp. S88-S99.

Bjerksund, P. and G. Stensland, *Closed Form Valuation of American Options*, Discussion paper 2002 (http://brage.bibsys.no/nhh/bitstream/URN:NBN:no-bibsys_brage_22301/1/bjerksund%20p

Financial Derivatives

You can find information on the creation of financial derivatives and their role in the marketplace in numerous sources. Among those consulted in the development of Financial Instruments Toolbox software are:

Chance, Don. M., *An Introduction to Derivatives*, The Dryden Press, 1998, ISBN 0-030-024483-8.

Fabozzi, Frank J., *Treasury Securities and Derivatives*, Frank J. Fabozzi Associates, 1998, ISBN 1-883249-23-6.

Wilmott, Paul, *Derivatives: The Theory and Practice of Financial Engineering*, John Wiley and Sons, 1998, ISBN 0-471-983-89-6.

Fitting Interest-Rate Curve Functions

Nelson, C.R., Siegel, A.F., "Parsimonious modelling of yield curves," *Journal of Business*, Number 60, 1987, pp 473-89.

Svensson, L.E.O., "Estimating and interpreting forward interest rates: Sweden 1992-4," International Monetary Fund, IMF Working Paper, 1994, p. 114.

Fisher, M., Nychka, D., Zervos, D., "Fitting the term structure of interest rates with smoothing splines," Board of Governors of the Federal Reserve System, Federal Reserve Board Working Paper, 1995.

Anderson, N., Sleath, J., "New estimates of the UK real and nominal yield curves," *Bank of England Quarterly Bulletin*, November, 1999, pp 384-92.

Waggoner, D., "Spline Methods for Extracting Interest Rate Curves from Coupon Bond Prices," Federal Reserve Board Working Paper, 1997, p. 10.

"Zero-coupon yield curves: technical documentation," *BIS Papers*, Bank for International Settlements, Number 25, October, 2005.

Bolder, D.J., Gusba,S, "Exponentials, Polynomials, and Fourier Series: More Yield Curve Modelling at the Bank of Canada," *Working Papers*, Bank of Canada, 2002, p. 29.

Bolder, D.J., Streliski, D., "Yield Curve Modelling at the Bank of Canada," *Technical Reports*, Number 84, 1999, Bank of Canada.

Interest-Rate Modeling Using Monte Carlo Simulation

Brigo, D. and F. Mercurio, *Interest Rate Models - Theory and Practice with Smile, Inflation and Credit* , Springer Finance, 2006.

Andersen, L. and V. Piterbarg, *Interest Rate Modeling*, Atlantic Financial Press. 2010.

Hull, J, *Options, Futures, and Other Derivatives*, Springer Finance, 2003.

Glasserman, P, *Monte Carlo Methods in Financial Engineering*, Prentice Hall, 2008.

Rebonato, R., K. McKay, and R. White, *The Sabr/Libor Market Model: Pricing, Calibration and Hedging for Complex Interest-Rate Derivatives*, John Wiley & Sons, 2010.

Bootstrapping a Swap Curve

Bootstrapping a Swap Curve

Hagan, P., West, G., "Interpolation Methods for Curve Construction," *Applied Mathematical Finance*, Vol. 13, Number 2, 2006.

Ron, Uri, "A Practical Guide to Swap Curve Construction," *Working Papers*, Bank of Canada, 2000, p. 17.

Bond Futures

Burghardt, G., T. Belton, M. Lane, and J. Papa, *The Treasury Bond Basis*, McGraw-Hill, 2005.

Krgin, Dragomir, *Handbook of Global Fixed Income Calculations*, John Wiley & Sons, 2002.

Credit Derivatives

Beumee, J., D. Brigo, D. Schiemert, and G. Stoyle. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research*, Global Special Report. April 7, 2009.

Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives* 8, 29-40.

O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps," *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.

O'Kane, D., *Modelling Single-name and Multi-name Credit Derivatives*, Wiley Finance, 2008, pp. 156–169.

American option

An option that can be exercised any time until its expiration date. Contrast with **European option** on page Glossary-6.

arbitrary cash flow instrument

A set of generic cash flow amounts for which a price needs to be established.

Asian option

An option whose payoff depends upon the average price of the underlying asset over a certain period of time.

asset-or-nothing option

A digital option that pays the value of the underlying security if the option expires in the money.

barrier option

An option that is activated or deactivated only if the price of the underlying asset crosses a barrier. See also **knock-in** on page Glossary-8 and **knock-out** on page Glossary-8. If the option fails to execute, the seller may pay to the purchaser a predetermined **rebate** on page Glossary-10.

barrier option

An option that is activated or deactivated only if the price of the underlying asset crosses a barrier. See also **knock-in** on page Glossary-8 and **knock-out** on page Glossary-8. If the option fails to execute, the seller may pay to the purchaser a predetermined **rebate** on page Glossary-10.

basis

Day count basis determines how interest accrues over time for various instruments and the amount transferred on interest payment dates. The calculation of accrued interest for dates between payments also uses day count basis. Day count basis is a fraction of `Number of interest accrual days` / `Days in the relevant coupon period`. Supported day count conventions and basis values are:

Basis Value	Day Count Convention
0	actual/actual (default) — Number of days in both a period and a year is the actual number of days.
1	30/360 SIA — Year fraction is calculated based on a 360 day year with 30-day months, after applying the following
2	actual/360 — Number of days in a period is equal to the actual number of days, however the number of days in a year is 360.
3	actual/365 — Number of days in a period is equal to the actual number of days, however the number of days in a year is 365 (even in a leap year).
4	30/360 PSA — Number of days in every month is set to 30 (including February). If the start date of the period is either the 31st of a month or the last day of February, the start date is set to the 30th, while if the start date is the 30th of a month and the end date is the 31st, the end date is set to the 30th. The number of days in a year is 360.
5	30/360 ISDA — Number of days in every month is set to 30, except for February where it is the actual number of days. If the start date of the period is the 31st of a month, the start date is set to the 30th while if the start date is the 30th of a month and the end date is the 31st, the end date is set to the 30th. The number of days in a year is 360.
6	30E /360 — Number of days in every month is set to 30 except for February where it is equal to the actual number of days. If the start date or the end date of the period is the 31st of a month, that date is set to the 30th. The number of days in a year is 360.
7	actual/365 Japanese — Number of days in a period is equal to the actual number of days, except for leap days (29th February) which are ignored. The number of days in a year is 365 (even in a leap year).

Basis Value	Day Count Convention
8	actual/actual ICMA — Number of days in both a period and a year is the actual number of days and the compounding frequency is annual.
9	actual/360 ICMA — Number of days in a period is equal to the actual number of days, however the number of days in a year is 360 and the compounding frequency is annual.
10	actual/365 ICMA — Number of days in a period is equal to the actual number of days, however the number of days in a year is 365 (even in a leap year) and the compounding frequency is annual.
11	30/360 ICMA — Number of days in every month is set to 30, except for February where it is equal to the actual number of days. If the start date or the end date of the period is the 31st of a month, that date is set to the 30th. The number of days in a year is 360 and the compounding frequency is annual.
12	actual/365 ISDA — The day count fraction is calculated using the following formula: `(Actual number of days in period that fall in a leap year / 366)` + `(Actual number of days in period that fall in a normal year / 365 )`.
13	bus/252 — The number of days in a period is equal to the actual number of business days. The number of business days in a year is 252.

basket option

An option that provides a payoff dependent on the value of a portfolio of assets.

beta

The price volatility of a financial instrument relative to the price volatility of a market or index as a whole. Beta is most commonly used with respect to equities. A high-beta instrument is riskier than a low-beta instrument.

binomial model

A method in which the probability over time of each possible price or rate follows a binomial distribution. The basic assumption is that prices or rates can move to only two values (one higher and one lower) over any short time period. See also **trinomial model** on page Glossary-12.

Black-Derman-Toy (BDT) model

A model for pricing interest rate derivatives where all security prices and rates depend upon the short rate (annualized one-period interest rate).

bond

A long-term debt security with fixed interest payments and fixed maturity date.

bond option

The right to sell a bond back to the issuer (put) or to redeem a bond from its current owner (call) at a specific price and on a specific date.

bushy tree

A tree of prices or interest rates in which the number of branches increases exponentially relative to observation times; branches never recombine. Opposite of a **recombining tree** on page Glossary-10.

call

1. An option to buy a certain quantity of a stock or commodity for a specified price within a specified time. See also **put** on page Glossary-9.

2. A demand to submit bonds to the issuer for redemption before the maturity date.

call swaption

Allows the option buyer to enter into an interest rate swap in which the buyer of the option pays the fixed rate and receives the floating rate.

callable bond

A bond that allows the issuer to buy back the bond at a predetermined price at specified future dates. The bond contains an embedded call option; that is, the holder has sold a call option to the issuer. See also **puttable bond** on page Glossary-9.

cap

Interest-rate option that guarantees that the rate on a floating-rate loan will not exceed a certain level.

caplet

An interim cap component in a multiperiod interest-rate cap agreement.

cash-or-nothing option

A digital option that pays some fixed amount of cash if the option expires in the money.

compound option

An option on an option, such as a call on a call, a put on a put, a call on a put, or a put on a call.

delta

The rate of change of the price of a derivative security relative to the price of the underlying asset; that is, the first derivative of the curve that relates the price of the derivative to the price of the underlying security.

derivative

A financial instrument that is based on some underlying asset. For example, an option is a derivative instrument based on the right to buy or sell an underlying instrument.

deterministic model

An interest rate model in which the values of the rates in the next time step are determined solely by the values of the rates in the current time step.

digital option

An option whose payout is fixed after the underlying stock exceeds the predetermined threshold or strike price.

discount factor

Coefficient used to compute the present value of future cash flows.

dollar sensitivity

Sensitivity reported as a dollar price change instead of a percentage price change.

down-and-in

A type of **barrier option** on page Glossary-1 that becomes active if the barrier is reached from above. See also **knock-in** on page Glossary-8.

down-and-out

A type of **barrier option** on page Glossary-1 that becomes deactivated if the barrier is reached from above. See also **knock-out** on page Glossary-8.

European option

An option that can be exercised only on its expiration date. Contrast with **American option** on page Glossary-1.

ex-dividend date

Date when a declared dividend belongs to the seller rather than the buyer.

exercise price

The price set for buying an asset (call) or selling an asset (put). The strike price.

exotic option

Any nonstandard option. Opposite of **vanilla option** on page Glossary-12.

fixed lookback option

Strike price is fixed at purchase. The underlying is priced at its highest or lowest level, depending whether it is a call or put, during the life of the option rather than expiring at market.

fixed-rate note

A long-term debt security with preset interest rate and maturity, by which the interest must be paid. The principal may or may not be paid at maturity.

floating lookback option

Strike price is fixed at maturity. For a call, the price is fixed at the lowest price during the life of the option; for a put it is fixed at the highest price.

floating-rate note

 A security similar to a bond, but in which the note's interest rate is reset periodically, relative to a reference index rate, to reflect fluctuations in market interest rates.

floor

 Interest-rate option that guarantees that the rate on a floating-rate loan will not fall below a certain level.

floorlet

 One of the interim period floors in a multiple period floor agreement.

forward curve

 The curve of forward interest rates vs. maturity dates for bonds.

forward rate

 The future interest rate of a bond inferred from the term structure, especially from the yield curve of zero-coupon bonds, calculated from the growth factor of an investment in a zero held until maturity.

gamma

 The rate of change of delta for a derivative security relative to the price of the underlying asset; that is, the second derivative of the option price relative to the security price.

gap option

 A digital option in which one strike decides if the option is in or out of money and another strike decides the size the size of the payoff.

Heath-Jarrow-Morton (HJM) model

 A model of the interest rate term structure that works with a type of interest rate tree called a **bushy tree** on page Glossary-4.

hedge

 A securities transaction that reduces or offsets the risk on an existing investment position.

instrument set

 A collection of financial assets. A portfolio.

inverse discount
> A factor by which the present value of an asset is multiplied to find its future value. The reciprocal of the discount factor.

irregular coupon
> A bond interest payment for more or less than six-months' interest. The first coupon on many bonds is irregular because payment is other than six months from the dated date.

knock-in
> A **barrier option** on page Glossary-1 that is activated when the price of the underlying asset achieves a designated target. There are two types: **up-and-in** on page Glossary-12 and **down-and-in** on page Glossary-6.

knock-out
> A **barrier option** on page Glossary-1 that is deactivated when the price of the underlying asset achieves a designated target. There are two types: **up-and-out** on page Glossary-12 and **down-and-out** on page Glossary-6.

Lambda
> The percentage change in an option price divided by the percentage change in an underlying price.

least-squares method
> A mathematical method of determining the best fit of a curve to a series of observations by choosing the curve that minimizes the sum of the squares of all deviations from the curve.

long rate
> The yield on a zero-coupon Treasury bond.

lookback option
> An option that reduces uncertainties associated with the timing of market entry. Lookback options can be either **fixed lookback option** on page Glossary-6 and **floating lookback option** on page Glossary-6.

mean reversion
> The tendency of a variable to return to its mean value after reaching a point of excessive positive or negative valuation relative to the mean.

option
> A right to buy or sell specific securities or commodities at a stated price (exercise or strike price) within a specified time. An option is a type of derivative.

per-dollar sensitivity
> The dollar **sensitivity** on page Glossary-10 divided by the corresponding instrument price.

portfolio
> A collection of financial assets. Also called an instrument set.

price tree structure
> A MATLAB structure that holds all pricing information.

price vector
> A vector of instrument prices.

pricing options structure
> A MATLAB structure that defines how the price tree is used to find the price of instruments in the portfolio, and how much additional information is displayed in the command window when the pricing function is called.

put
> An option to sell a stipulated amount of stock or securities within a specified time and at a fixed exercise price. See also **call** on page Glossary-4.

put swaption
> Allows the option buyer to enter into an interest rate swap in which the buyer of the option receives the fixed rate and pays the floating rate.

puttable bond
> A bond that allows the holder to redeem the bond at a predetermined price at specified future dates. The bond contains an embedded put option; that is, the holder has bought a put option. See also **callable bond** on page Glossary-4.

rainbow option
> A single option linked to two or more underlying assets. In order for the option to pay off, all the underlying assets must move in the intended direction.

rate specification
> A MATLAB structure that holds all information needed to identify completely the evolution of interest rates.

rebate
> A predetermined amount of money paid to the purchaser of a **barrier option** on page Glossary-1 if the option fails to execute.

recombining tree
> A tree of prices or interest rates whose branches recombine over time. Opposite of a **bushy tree** on page Glossary-4.

self-financing hedge
> A trading strategy whereby the value of a portfolio after rebalancing is equal to its value at any previous time.

sensitivity
> The "what if" relationship between variables; the degree to which changes in one variable cause changes in another variable. A specific synonym is volatility. See also **dollar sensitivity** on page Glossary-5.

short rate
> The annualized one-period interest rate.

sinking fund bond
> A sinking fund bond is a coupon bond with a sinking fund provision. This provision obligates the issuer to amortize portions of the principal prior to maturity, affecting bond prices since the time of the principal repayment changes.

spot curve, spot yield curve
> See **zero curve, zero-coupon yield curve** on page Glossary-13.

spot rate

  The current interest rate appropriate for discounting a cash flow of some given maturity.

spread

  For options, a combination of call or put options on the same stock with differing exercise prices or maturity dates.

stepped coupon bond

  A step-up and step-down bond is a debt security with a predetermined coupon structure over time.

stochastic model

  Involving or containing a random variable or variables; involving chance or probability.

strike

  Exercise a put or call option.

strike price

  See **exercise price** on page Glossary-6.

supershare option

  A digital option that pays out a proportion of the assets underlying a portfolio if the asset lies between a lower and an upper bound at the expiry of the option.

swap

  A contract between two parties to exchange cash flows in the future according to some formula.

swaption

  An option on an interest rate swap. It grants the option buyer the right to enter into an interest rate swap at a future date.

time specification

  A MATLAB structure that represents the mapping between times and dates for interest rate quoting.

trinomial model

A method in which the basic assumption is that prices or rates can move to one of three possible values over any short time period. At any time step the price or rate direction can be upward, neutral, or downward. See also **binomial model** on page Glossary-4.

under-determined system

A set of simultaneous equations in which the number of independent variables exceeds the number of equations in the set, leading to an infinite number of solutions.

up-and-in

A type of **barrier option** on page Glossary-1 that becomes active if the barrier is reached from below. See also **knock-in** on page Glossary-8.

up-and-out

A type of **barrier option** on page Glossary-1 that becomes deactivated if the barrier is reached from below. See also **knock-out** on page Glossary-8.

vanilla option

A common option, such as a put or call. Opposite of **exotic option** on page Glossary-6.

vanilla swap

A **swap** on page Glossary-11 agreement to exchange a fixed rate for a floating rate.

vega

The rate of change in the price of a derivative security relative to the volatility of the underlying security. When vega is large, the security is sensitive to small changes in volatility.

volatility specification

A MATLAB structure that specifies the forward rate volatility process.

yields

The zero coupon rate.

yield curve

The zero curve.

yield volatility

The zero coupon volatilities.

zero curve, zero-coupon yield curve

A yield curve for zero-coupon bonds; zero rates versus maturity dates. Since the maturity and duration (Macaulay duration) are identical for zeros, the zero curve is a pure depiction of supply/demand conditions for loanable funds across a continuum of durations and maturities. Also known as spot curve or spot yield curve.

zero-coupon bond, or zero

A bond that, instead of carrying a coupon, is sold at a discount from its face value, pays no interest during its life, and pays the principal only at maturity.

Index