Fixed-Income Toolbox™ 1
User's Guide

MATLAB®

The MathWorks™

Accelerating the pace of engineering and science
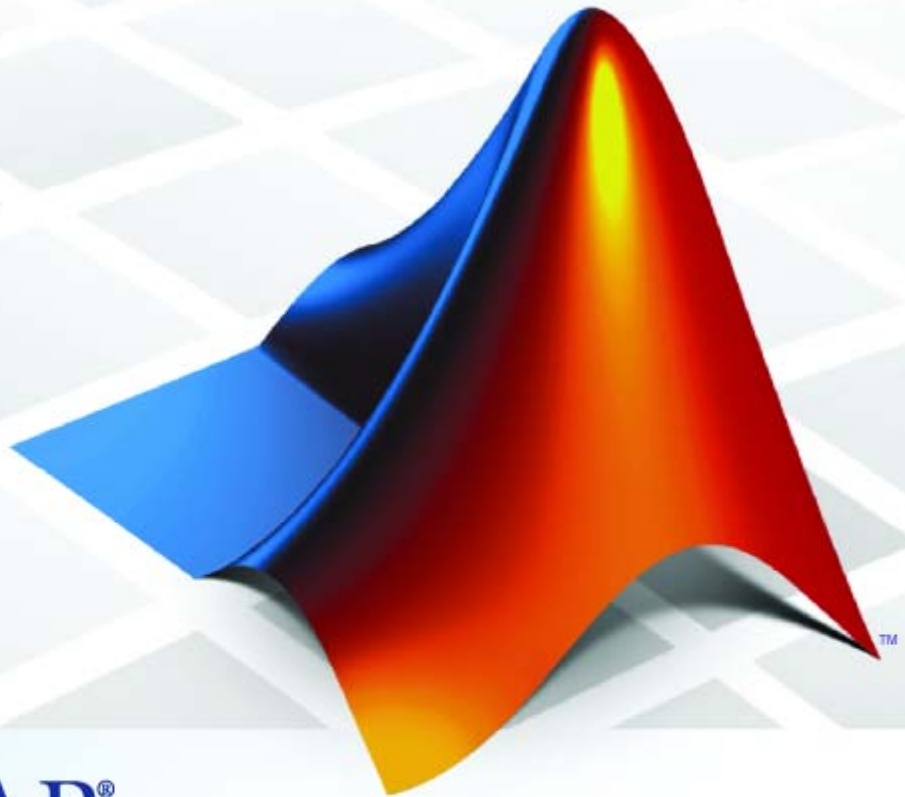
How to Contact The MathWorks

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.

Fixed-Income Toolbox™ User's Guide

© COPYRIGHT 2003–2010 by The MathWorks, Inc.

Trademarks

Patents

Revision History

Contents

3

Debt Instruments

4

Derivative Securities

Interest-Rate Curve Objects

5

6

7

A

Bibliography

B

Examples

C

Glossary

Index

Getting Started

- "Product Overview" on page 1-2
- "What This Product Provides" on page 1-4

Product Overview

In this section...
"Introduction" on page 1-2
"Expected Background" on page 1-2

Introduction

Fixed-Income Toolbox™ software extends MATLAB® with functions for fixed-income modeling and analysis. You can use the toolbox to determine the price, yield, and cash flow for many types of fixed-income securities, including mortgage-backed securities, corporate bonds, treasury bonds, municipal bonds, certificates of deposit, and treasury bills. Fixed-Income Toolbox functions also enable you to work with derivatives, including swaps, convertible bonds, and treasury futures. The built-in functions can be used to create customized fixed-income models based on mortgage-backed securities and debt instruments.

Expected Background

In general, this guide assumes experience working with fixed-income instruments and some familiarity with the underlying models.

Your title is likely one of these:

- Analyst, quantitative analyst

- Risk manager

- Portfolio manager

- Fund manager, asset manager

- Financial engineer

- Trader

- Student, professor, or other academic

Your background, education, training, and responsibilities likely 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 and calculus
- Perhaps new to MATLAB software

What This Product Provides

Supported Tasks

Fixed-Income Toolbox software extends MATLAB software with functions for fixed-income modeling and analysis. You can use the toolbox to determine the price, yield, and cash flow for many types of fixed-income securities, including mortgage-backed securities, corporate bonds, Treasury bonds, municipal bonds, certificates of deposit, and Treasury bills.

Fixed-Income Toolbox software also enables you to work with derivatives, including swaps, convertible bonds, and Treasury futures. You can use built-in functions to create customized fixed-income models based on mortgage-backed securities and debt instruments. You can then use Fixed-Income Toolbox software to perform the following:

- Calculate the price and yield for generic fixed-rate mortgage pools and balloon mortgages.

- Determine the price, yield, discount rate, and cash-flow schedule for debt instruments, including Treasury bills, zero-coupon bonds, and stepped-coupon bonds.

- Calculate swap rates and sensitivities.

- Analyze the term structure of interest rates, including bootstrapping and fitting the term structure to market data using parametric models.

Using Mortgage-Backed Securities

With Fixed-Income Toolbox software, you can model generic fixed-rate mortgage pools and balloon mortgages. Tools are provided for:

- Calculating the price and yield of mortgage-backed securities using prepayment options derived from uniform practices of the Public Securities Association (PSA)

- Determining the mortgage-pool price or effective duration using the option-adjusted spread (OAS) method

- Calculating basic risk measurements for a mortgage-pool portfolio using convexity, duration, and average life

Using Debt Instruments

You can also use Fixed-Income Toolbox software to work with a variety of debt instruments. You can calculate price, yield, discount rate, and break-even discount rate for treasury bills, and determine price, yield, and cash-flow schedules for corporate, treasury, and municipal bonds. The zero-coupon functions in Fixed-Income Toolbox software help with the extraction of present value from virtually any fixed-coupon instrument for any time period. Toolbox functions also let you calculate price, yield, and cash-flow schedules for stepped-coupon bonds. The next coupon dates are computed automatically from the last entered input end dates. The payment due on settlement represents the accrued interest due on that day.

Using Derivative Securities

In addition, Fixed-Income Toolbox software provides tools based on Black's option functions for working with fixed-income derivatives. These tools let you calculate swap price by computing par yields that equate the floating-rate side of a swap to the fixed-rate side. You can set the present value of the fixed side to the present value of the floating side without aligning and comparing fixed and floating periods. The duration-hedging capability in the toolbox lets you hedge a portfolio and address interest-rate risk exposure with a swap arrangement. Fixed-Income Toolbox software lets you use binomial and trinomial trees to value convertible bonds. The value of the convertible bond is determined by the uncertainty of the relative stock.

Using Interest-Rate Curve Objects

Fixed-Income Toolbox software provides tools for analyzing the term structure of interest rates, including bootstrapping and fitting the term structure to market data using parametric models (e.g., Nelson-Siegel and Svensson),

spline-based models, and user-defined functions. Fixed-Income Toolbox supports three class objects:

- "@IRCurve" on page A-4

  Creates an interest-rate curves and includes methods for extracting forward, zero, and discount factors curves.

  Supports a method to convert to a `RateSpec` structure, which is an acceptable input format for the Financial Derivatives Toolbox™ function `intenvset`.

- "@IRDataCurve" on page A-7

  Represents interest-rate curves based on vectors of dates and data. This class supports bootstrapping an interest-rate curve from market instruments with a range of interpolation methods.

- "@IRFunctionCurve" on page A-12

  Represents an interest-rate curve with a function; the function can be specified directly, or a form of the function can be specified and then the parameters are fit to available market data. In addition, you can determine which type of interest-rate curve (zero, forward, or discount curve) fits the market data, as well as, any custom functions.

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.

Using Fixed-Rate Mortgage Pool Functions

Introduction

Fixed-Income 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.

You can obtain more information about these uniform practices on the PSA Web site (`http://www.bondmarkets.com`).

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.
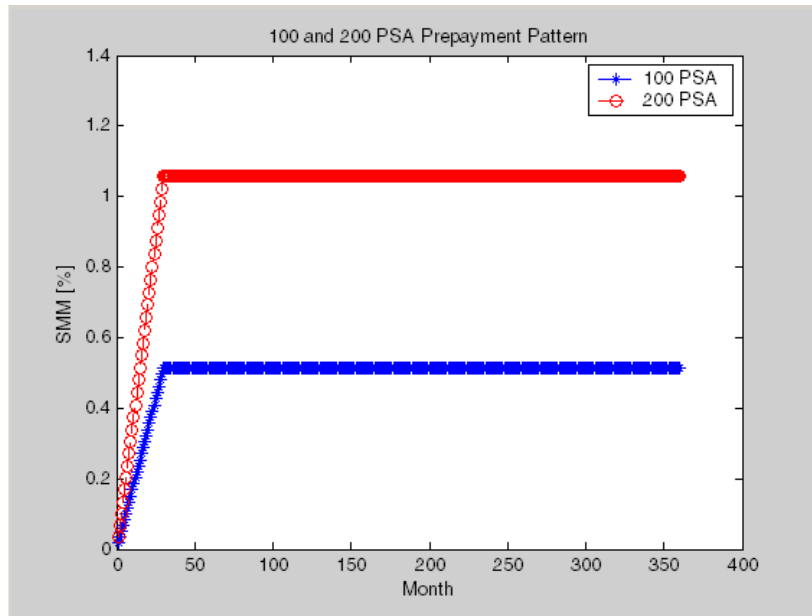
Prepayment assumptions form the basis upon which far more comprehensive MBS calculations are based. As an illustration observe the following example, which shows the use of the function mbscfamounts to generate cash flows and timings based on a set of standard prepayments.

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 Variable 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

Fixed-Income 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 Fixed-Income 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 2-10 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 2-9. 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.
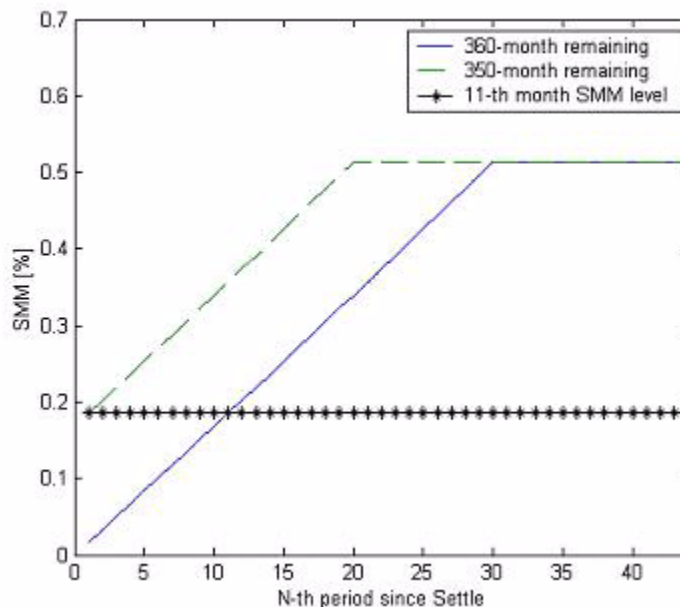
Still within the scope of relative valuation, you could also solve for the percentage of the standard PSA prepayment vector given the pool's arbitrary, user-supplied prepayment vector, such that the PSA speed gives the same Macaulay duration as the user-supplied prepayment vector.

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 =

   104.2526
```

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:
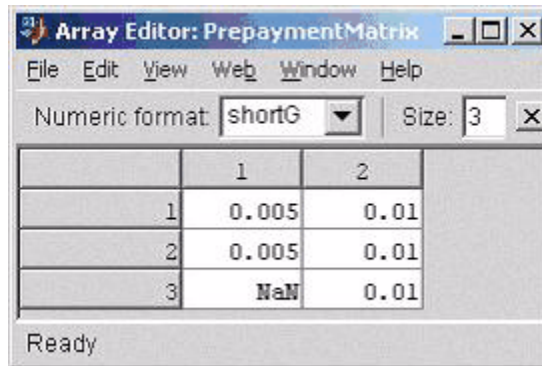
```
V11         V21
V12         V22
NaN         V23
```

V_{ij} denotes the single monthly mortality (SMM) rate for pool i during the jth 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.

3

Debt Instruments

- "Treasury Bills Defined" on page 3-2
- "Computing Treasury Bill Price and Yield" on page 3-3
- "Using Zero-Coupon Bonds" on page 3-7
- "Stepped-Coupon Bonds" on page 3-12
- "Term Structure Calculations" on page 3-15

Treasury Bills Defined

Treasury bills are short-term securities (issued with maturities of 1 year or less) sold by the United States Treasury. Sales of these securities are frequent, usually weekly. From time to time, the Treasury also offers longer duration securities called Treasury notes and Treasury bonds.

A Treasury bill is a discount security. The holder of the Treasury bill does not receive periodic interest payments. Instead, at the time of sale, a percentage discount is applied to the face value. At maturity, the holder redeems the bill for full face value.

The basis for Treasury bill interest calculation is actual/360. Under this system, interest accrues on the actual number of elapsed days between purchase and maturity, and each year contains 360 days.

Computing Treasury Bill Price and Yield

In this section...
"Introduction" on page 3-3
"Treasury Bill Repurchase Agreements" on page 3-3
"Treasury Bill Yields" on page 3-5

Introduction

Fixed-Income Toolbox software provides the following suite of functions for computing price and yield on Treasury bills.

Treasury Bill Functions

Function	Purpose
tbilldisc2yield	Convert discount rate to yield.
tbillprice	Price Treasury bill given its yield or discount rate.
tbillrepo	Break-even discount of repurchase agreement.
tbillyield	Yield and discount of Treasury bill given its price.
tbillyield2disc	Convert yield to discount rate.
tbillval01	The value of 1 basis point given the characteristics of the Treasury bill, as represented by its settlement and maturity dates. You can relate the basis point to discount, money-market, or bond-equivalent yield.

For all functions with yield in the computation, you can specify yield as money-market or bond-equivalent yield. The functions all assume a face value of $100 for each Treasury bill.

Treasury Bill Repurchase Agreements

The following example shows how to compute the break-even discount rate. This is the rate that correctly prices the Treasury bill such that the profit from selling the tail equals 0.

```
Maturity = '26-Dec-2002';
InitialDiscount = 0.0161;
PurchaseDate = '26-Sep-2002';
SaleDate = '26-Oct-2002';
RepoRate = 0.0149;

BreakevenDiscount = tbillrepo(RepoRate, InitialDiscount, ...
PurchaseDate, SaleDate, Maturity)

BreakevenDiscount =

    0.0167
```

You can check the result of this computation by examining the cash flows in and out from the repurchase transaction. First compute the price of the Treasury bill on the purchase date (September 26).

```
PriceOnPurchaseDate = tbillprice(InitialDiscount, ...
PurchaseDate, Maturity, 3)

PriceOnPurchaseDate =

   99.5930
```

Next compute the interest due on the repurchase agreement.

```
RepoInterest = ...
RepoRate*PriceOnPurchaseDate*days360(PurchaseDate,SaleDate)/360

RepoInterest =

    0.1237
```

RepoInterest for a 1.49% 30-day term repurchase agreement (30/360 basis) is 0.1237.

Finally, compute the price of the Treasury bill on the sale date (October 26).

```
PriceOnSaleDate = tbillprice(BreakevenDiscount, SaleDate, ...
Maturity, 3)
```

```
PriceOnSaleDate =

    99.7167
```

Examining the cash flows, observe that the break-even discount causes the sum of the price on the purchase date plus the accrued 30-day interest to be equal to the price on sale date. The next table shows the cash flows.

Cash Flows from Repurchase Agreement

Date	Cash Out Flow		Cash In Flow	
9/26/2002	Purchase T-bill	99.593	Repo money	99.593
10/26/2002	Payment of repo	99.593	Sell T-bill	99.7168
	Repo interest	0.1238		
	Total	199.3098		199.3098

Treasury Bill Yields

Using the same data as before, you can examine the money-market and bond-equivalent yields of the Treasury bill at the time of purchase and sale. The function tbilldisc2yield can perform both computations at one time.

```
Maturity = '26-Dec-2002';
InitialDiscount = 0.0161;
PurchaseDate = '26-Sep-2002';
SaleDate = '26-Oct-2002';
RepoRate = 0.0149;
BreakevenDiscount = tbillrepo(RepoRate, InitialDiscount, ...
PurchaseDate, SaleDate, Maturity)

[BEYield, MMYield] = ...
tbilldisc2yield([InitialDiscount; BreakevenDiscount], ...
[PurchaseDate; SaleDate], Maturity)

BreakevenDiscount =
```

```
     0.0167


BEYield =

     0.0164
     0.0170


MMYield =

     0.0162
     0.0168
```

For the short Treasury bill (fewer than 182 days to maturity), the money-market yield is 360/365 of the bond-equivalent yield, as this example shows.

Using Zero-Coupon Bonds

In this section...
"Introduction" on page 3-7
"Measuring Zero-Coupon Bond Function Quality" on page 3-7
"Pricing Treasury Notes" on page 3-8
"Pricing Corporate Bonds" on page 3-10

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.

Fixed-Income 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 3-8

- "Pricing Corporate Bonds" on page 3-10

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 3-12
"Cash Flows from Stepped-Coupon Bonds" on page 3-12
"Price and Yield of Stepped-Coupon Bonds" on page 3-14

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 software 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:

$$\frac{(2\times2)+(4\times3)}{5}$$

.

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" on page 3-15
"Computing Spot and Forward Curves" on page 3-15
"Computing Spreads" on page 3-17

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).
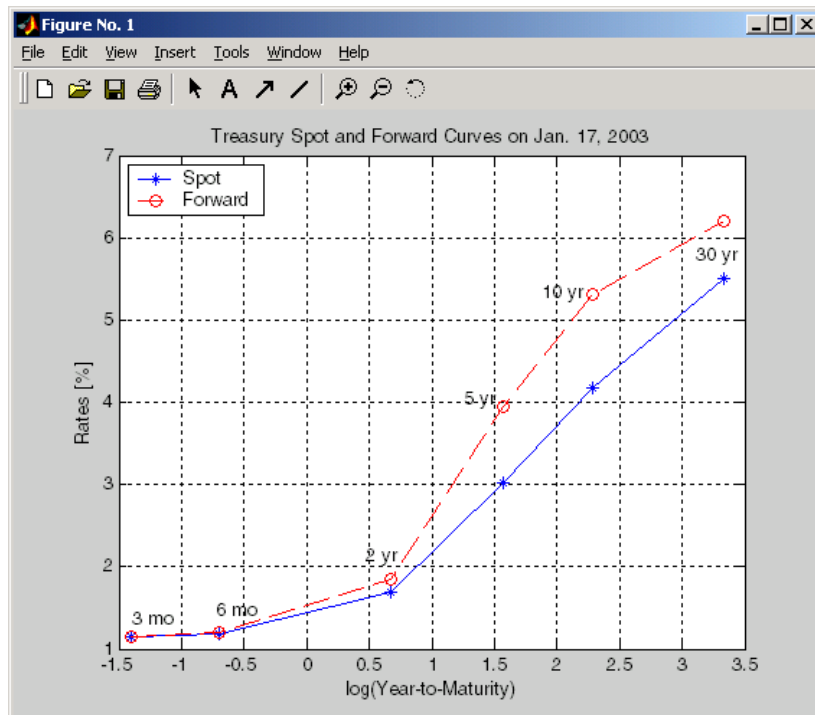
Computing Spreads

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).
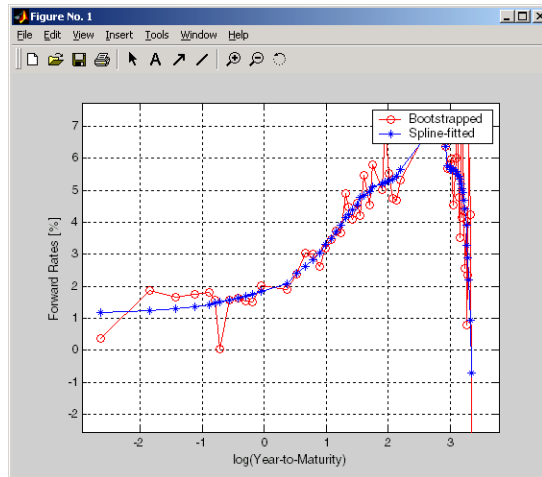
Implied Forward Curves. The jagged curve comes from direct bootstrapping. The smooth curve shows the effect of smoothing with splines.
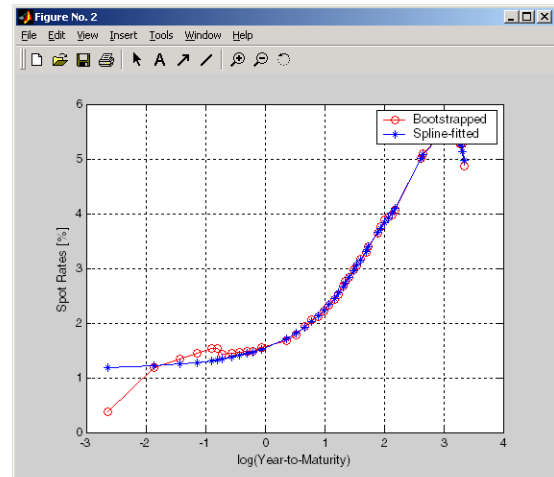
Implied Spot Rate Curves. These curves correspond to the forward curve above.

This example uses `termfit`, a demonstration function from Financial Toolbox software that also requires the use of Spline Toolbox™ software.

4

Derivative Securities

Pricing and Hedging

In this section...

Swap Pricing Assumptions

Fixed-Income 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.cme.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), comparable to 2.17% of Friday's average broker data in Table H15 of *Federal Reserve Statistical Release* (`http://www.federalreserve.gov/releases/h15/update/`).

```
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.

Fixed-Income 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.
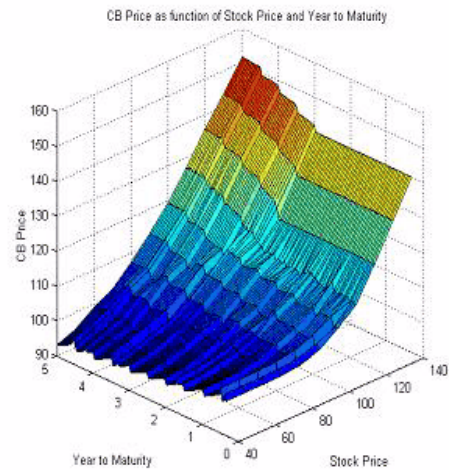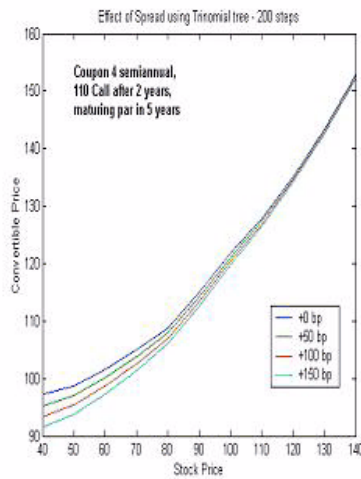
The figure on the left shows the bond "floor" of the convertible (a 5% yield, given a 4% coupon at about 97% par) when share prices are low.

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.

Fixed-Income 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 implied repo rates given the Treasury bond future price.
tfutyieldbyrepo	Calculates implied repo rates given the Treasury bond future yield.

Example 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', .0425,.06,3)
CF1 =

    0.8827
```

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 Interest-Rate Risk 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 the Fixed-Income Toolbox demo "Managing Interest Rate Risk with Bond Futures" at:

```
demo toolbox 'fixed-income'
```

5

Interest-Rate Curve Objects

Introduction to Interest-Rate Curve Objects

In this section...
"Class Structure" on page 5-2
"Supported Workflow Model Using Interest-Rate Curve Objects" on page 5-3

Class Structure

Fixed-Income 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.

Supported Workflow Model 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 Derivatives Toolbox function `intenvset`. Using the `RateSpec` for an interest-rate curve object, you can then use Financial Derivatives Toolbox functions to model an interest-rate structure and price. For more information, see "Interest-Rate Derivatives".

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 **Nelson-Siegel model** on page Glossary-8 to market data for bonds.

    - fitSvensson fits a **Svensson model** on page Glossary-12 to market data for bonds.

    - fitSmoothingSpline fits a **smoothing spline** on page Glossary-11 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 5-25.

Creating an IRDataCurve Object

In this section...

Using the 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.
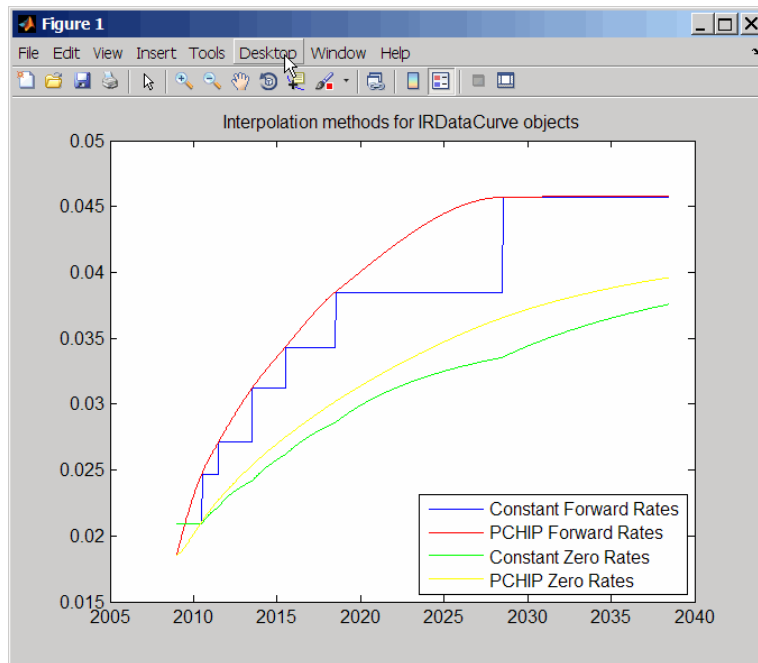
Using IRDataCurve bootstrap Method for Bootstrapping Based on Market Instruments

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, contsant, 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: 0 (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.
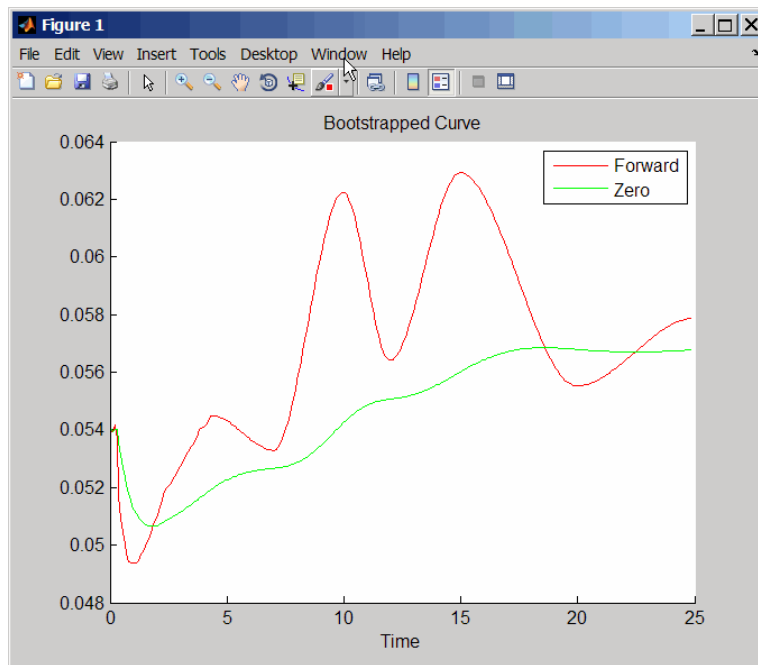
Example 2

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: 0 (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.
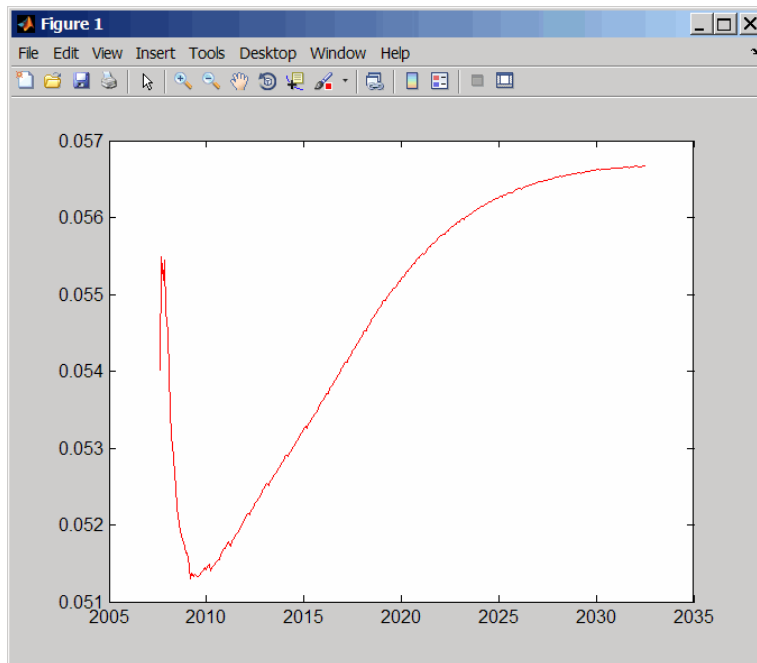
Creating an IRFunctionCurve Object

Using a Function Handle to Fit an IRFunctionCurve Object

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
```

Using the Nelson-Siegel Method to Fit an IRFunctionCurve Object

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);
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');
```

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')
```
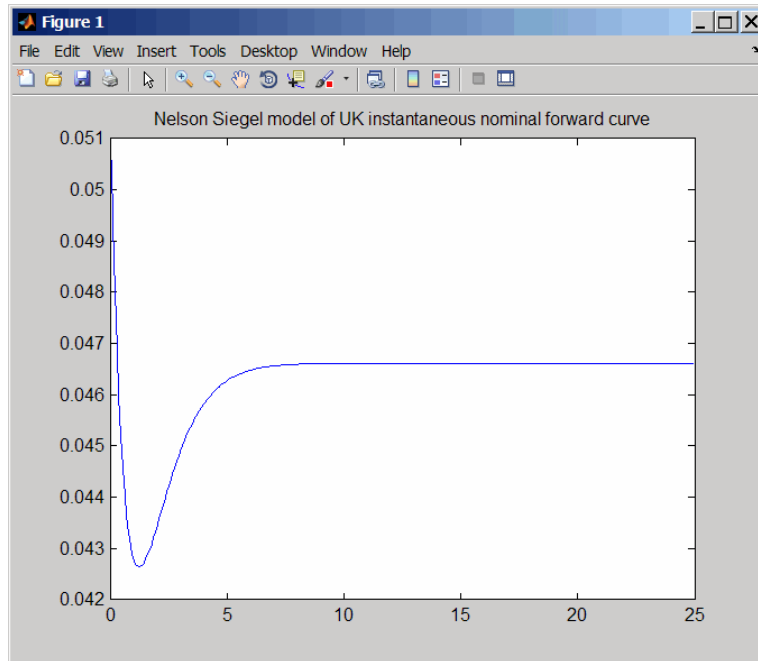
Using the Svensson Method to Fit an IRFunctionCurve Object

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, $\beta3$ and $\tau2$, 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 = optimset('lsqnonlin');
OptOptions = optimset(OptOptions,'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 selected 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')
```
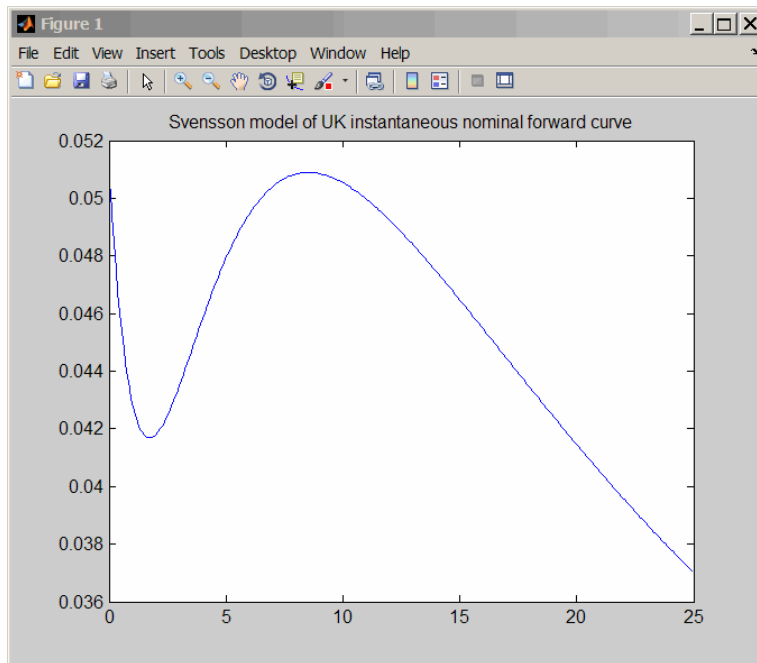

Using the Smoothing Spline Method to Fit an IRFunctionCurve Object

Use the method, fitSmoothingSpline, to model the term structure with a spline, specifically, the term structure represents the forward curve with a cubic spline.

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);
```

The plot demonstrates the interest-rate curve with the penalty function.
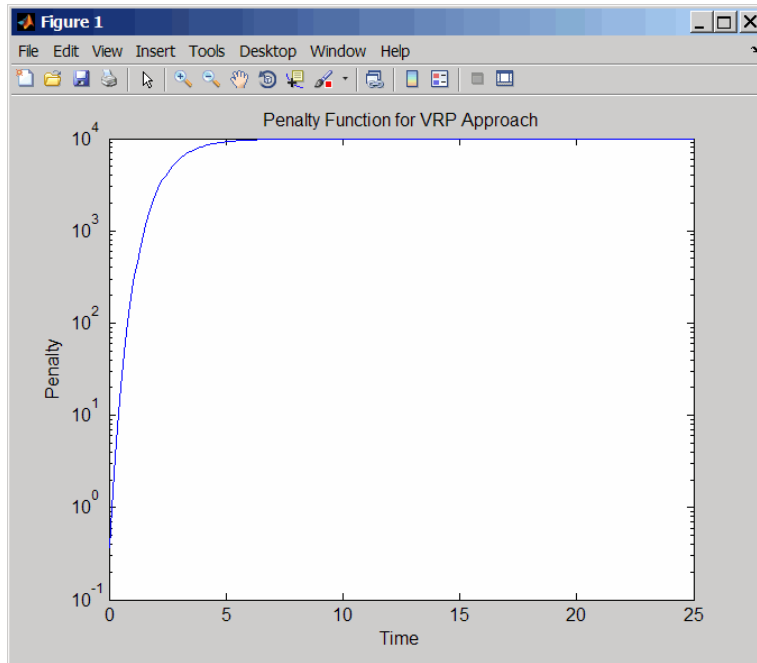
Plot the smoothing spline interest-rate curve for forward rates:

```
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')
```
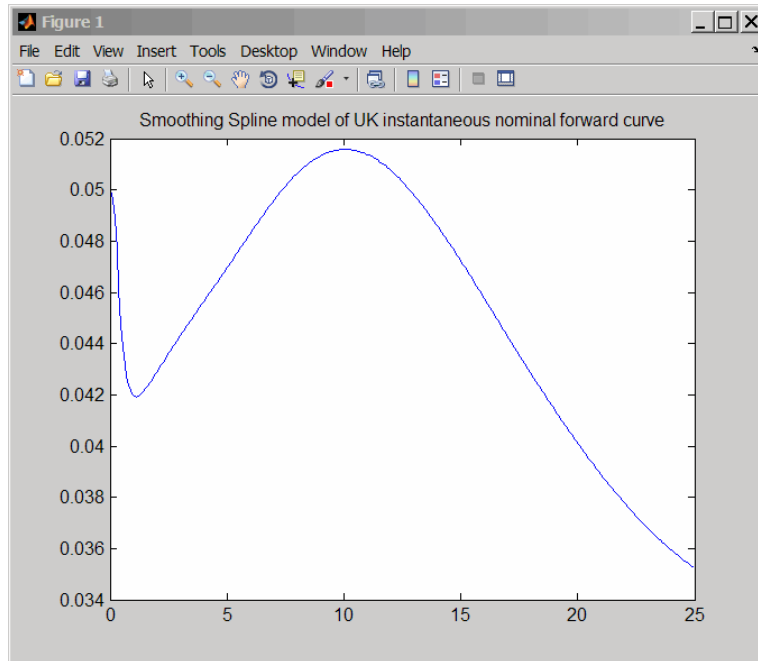
Using the fitFunction to Create a Custom Fitting Function for an IRFunctionCurve Object

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 = optimset('lsqnonlin');
OptOptions = optimset(OptOptions,'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+004	
1	8	38036.7	10	4.92e+004	0
2	12	38036.7	2.5	4.92e+004	0
3	16	38036.7	0.625	4.92e+004	0
4	20	38036.7	0.15625	4.92e+004	0
5	24	30741.5	0.0390625	1.72e+005	0
6	28	30741.5	0.078125	1.72e+005	0
7	32	30741.5	0.0195312	1.72e+005	0
8	36	28713.6	0.00488281	2.33e+005	0
9	40	20323.3	0.00976562	9.47e+005	0
10	44	20323.3	0.0195312	9.47e+005	0
11	48	20323.3	0.00488281	9.47e+005	0
12	52	20323.3	0.0012207	9.47e+005	0
13	56	19698.8	0.000305176	1.08e+006	0
14	60	17493	0.000610352	7e+006	0
15	64	17493	0.0012207	7e+006	0
16	68	17493	0.000305176	7e+006	0
17	72	15455.1	7.62939e-005	2.25e+007	0

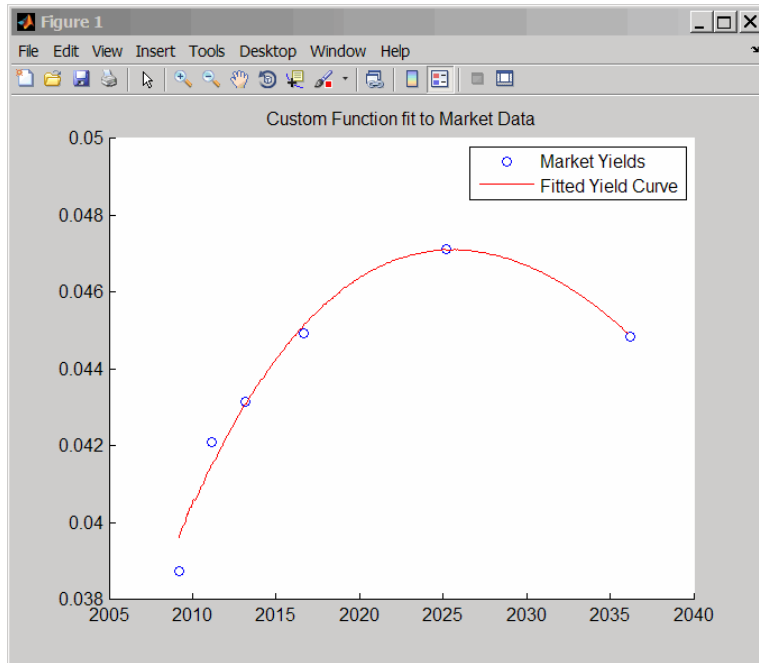| 18	76	15455.1	0.000177558	2.25e+007	0
19	80	13317.1	3.8147e-005	3.18e+007	0
20	84	12867.9	7.62939e-005	7.84e+007	0
21	88	11779.8	7.62939e-005	7.58e+006	0
22	92	11747.6	0.000152588	1.46e+005	0
23	96	11720.9	0.000305176	2.48e+005	0
24	100	11667.2	0.000610352	1.48e+005	0
25	104	11558.5	0.0012207	4.47e+005	0
26	108	11335.4	0.00244141	1.58e+005	0
27	112	10864	0.00488281	1.61e+005	0
28	116	9797.68	0.00976562	6.85e+005	0
29	120	6884.03	0.0195312	5.79e+005	0
30	124	6884.03	0.037498	5.79e+005	0
31	128	3216.51	0.00937449	1.75e+006	0
32	132	607.317	0.018749	2.94e+006	0
33	136	12.7284	0.0253662	3e+006	0
34	140	0.0760939	0.00153457	4.88e+004	0
35	144	0.0731652	3.58678e-006	24.6	0
36	148	0.0731652	6.04329e-008	0.0213	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.
```

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')
```

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 Derivatives
  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 thetoRateSpec 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 Fixed-Income 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
```

6

Function Reference

Bond Futures

bndfutimprepo	Implied repo rates for bond future given price
bndfutprice	Price bond future given repo rates
convfactor	Bond conversion factors
tfutbyprice	Future prices of Treasury bonds given spot price
tfutbyyield	Future prices of Treasury bonds given current yield
tfutimprepo	Implied repo rates for Treasury bond future given price
tfutpricebyrepo	Implied repo rates given Treasury bond future price
tfutyieldbyrepo	Implied repo rates given Treasury bond future yield

Certificates of Deposit

cdai	Accrued interest on certificate of deposit (CD)
cdprice	Price of certificate of deposit (CD)
cdyield	Yield on certificate of deposit (CD)

Convertible Bonds

cbprice	Price convertible bond

Derivative Securities

bkcall	Price European call option on bonds using Black's model
bkcaplet	Price interest-rate caplet using Black's model
bkfloorlet	Price interest-rate floorlet using Black's model
bkput	Price European put option on bonds using Black's model
liborduration	Duration of LIBOR-based interest-rate swap
liborfloat2fixed	Compute par fixed-rate of swap given 3-month LIBOR data
liborprice	Price swap given swap rate

Interest-Rate Curve Objects

bootstrap	Bootstrap interest-rate curve from market data
fitFunction	Custom fit interest-rate curve object to bond market data
fitNelsonSiegel	Fit Nelson-Siegel function to bond market data

`fitSmoothingSpline`	Fit smoothing spline to bond market data
`fitSvensson`	Fit Svensson function to bond market data
`getDiscountFactors`	Get discount factors for input dates for `IRDataCurve`
`getDiscountFactors`	Get discount factors for input dates for `IRFunctionCurve`
`getForwardRates`	Get forward rates for input dates for `IRDataCurve`
`getForwardRates`	Get forward rates for input dates for `IRFunctionCurve`
`getParYields`	Get par yields for input dates for `IRDataCurve`
`getParYields`	Get par yields for input dates for `IRFunctionCurve`
`getZeroRates`	Get zero rates for input dates for `IRDataCurve`
`getZeroRates`	Get zero rates for input dates for `IRFunctionCurve`
`IRBootstrapOptions`	Construct specific options for bootstrapping interest-rate curve object
`IRDataCurve`	Construct interest-rate curve object from dates and data
`IRFitOptions`	Construct specific options for fitting interest-rate curve object
`IRFunctionCurve`	Construct interest-rate curve object from function handle or function by fitting to market data using object methods

toRateSpec	Convert IRDataCurve object to RateSpec
toRateSpec	Convert IRFunctionCurve object to RateSpec

Mortgage-Backed Securities

mbscfamounts	Cash flow and time mapping for mortgage pool
mbsconvp	Convexity of mortgage pool given price
mbsconvy	Convexity of mortgage pool given yield
mbsdurp	Duration of mortgage pool given price
mbsdury	Duration of mortgage pool given yield
mbsnoprepay	End-of-month mortgage cash flows and balances without prepayment
mbspassthrough	Mortgage pool cash flows and balances with prepayment
mbsprice	Mortgage-backed security price given yield
mbsprice2speed	Implied PSA prepayment speeds given price
mbswal	Weighted average life of mortgage pool
mbsyield	Mortgage-backed security yield given price
mbsyield2speed	Implied PSA prepayment speeds given yield
psaspeed2default	Benchmark default
psaspeed2rate	Single monthly mortality rate given PSA speed

Option-Adjusted Spread Computations

`mbsoas2price`	Price given option-adjusted spread
`mbsoas2yield`	Yield given option-adjusted spread
`mbsprice2oas`	Option-adjusted spread given price
`mbsyield2oas`	Option-adjusted spread given yield

Stepped-Coupon Bonds

stepcpncfamounts	Cash flow amounts and times for bonds and stepped coupons
stepcpnprice	Price bond with stepped coupons
stepcpnyield	Yield to maturity of bond with stepped coupons

Treasury Bills

`tbilldisc2yield`	Convert Treasury bill discount to equivalent yield
`tbillprice`	Price Treasury bill
`tbillrepo`	Break-even discount of repurchase agreement
`tbillval01`	Value of one basis point
`tbillyield`	Yield on Treasury bill
`tbillyield2disc`	Convert Treasury bill yield to equivalent discount

Zero-Coupon Instruments

zeroprice	Price zero-coupon instruments given yield
zeroyield	Yield of zero-coupon instruments given price

Functions — Alphabetical List

bkcall

Purpose	Price European call option on bonds using Black's 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.

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

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 Fixed-Income 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 5-25.

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's 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

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's 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

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's 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

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 Fixed-Income 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 5-25.

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.

bkput

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

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

Irregular or normal first coupon date.

IssueDate

Issue date for a bond.

LastCouponDate

Irregular or normal last coupon date.

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
```

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 4-12

bndfutprice

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

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

> Irregular or normal first coupon date.

LastCouponDate

> Irregular or normal last coupon date.

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.

bndfutprice

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.

Krgin, Dragomir, *Handbook of Global Fixed Income Calculations*, John Wiley & Sons, 2002.

See Also   bndfutimprepo | convfactor

How To   • "Bond Futures" on page 4-12

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 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 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

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 of the supported `InstrumentTypes`, you can specify the following additional instrument parameters as parameter/value pairs by prepending the word `Instrument` to the parameter field. For example, prepending `InstrumentBasis` distinguishes an instrument's `Basis` value from the curve's `Basis` value.

CouponRate	(Optional) Decimal number indicating the annual percentage rate used to determine the coupons payable on an instrument.
Period	(Optional) Coupons per year of the instrument. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.
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

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 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.
IssueDate	(Optional) Date when an instrument was issued.
FirstCouponDate	(Optional) Date when an instrument makes its first coupon payment. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.
LastCouponDate	(Optional) Last coupon date of an instrument before the maturity date. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the instrument. The coupon structure of an instrument is truncated at the LastCouponDate regardless of where it falls and will be followed only by the instrument's maturity cash flow date.
Face	(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 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]
```
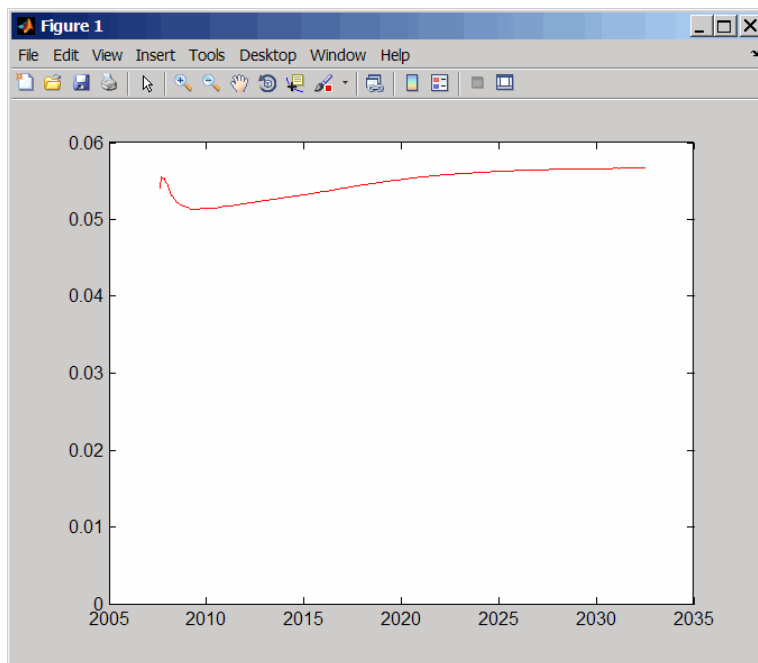
To create the plot for the bootstrapped market data:

```
PlottingDates = (datenum('08/11/2007'):30:CurveSettle+365*25)';
plot(PlottingDates,bootModel.getParYields(PlottingDates),'r')
set(gca,'ylim',[0 .06])
datetick
```

bootstrap

For an example of bootstrapping using instrument parameters support for prepending the word `Instrument` to the parameter field, see "Using IRDataCurve bootstrap Method for Bootstrapping Based on Market Instruments" on page 5-7.

See Also   "@IRDataCurve" on page A-7, "@IRBootstrapOptions" on page A-2

Purpose   Price convertible bond

Syntax   [CBMatrix, UndMatrix, DebtMatrix, EqtyMatrix] =
                     cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ConvRatio,
                     NumSteps, IssueDate, Settle, Maturity, CouponRate, Period, Basis,
                     EndMonthRule, DividendType, DividendInfo, CallType, CallInfo,
                     TreeType)

Arguments

RiskFreeRate	Annual yield of risk-free bond with the same maturity as the convertible, compounded continuously. (Recommended value is the yield of a risk-free bond with the same maturity as the convertible.)
StaticSpread	Value of constant spread to the risk free rate. Adding this to the RiskFreeRate produces the issuer's yield, which reflects its credit risk.
Sigma	Annual volatility in decimal.
Price	Price of asset at settlement or valuation date.
ConvRatio	Scalar. Number of assets convertible to a single bond.
NumSteps	Number of steps in binomial tree.
IssueDate	Issue date of bond.
Settle	Settlement date of bond.
Maturity	Maturity date of bond.
CouponRate	Coupon rate payable per unit of face value.
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) Scalar value for day-count basis of the instrument.

- 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

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.
DividendType	(Optional) 0 = dollar dividend (default). 1 = dividend yield.

DividendInfo	(Optional) Two-column matrix of dividend information. First column contains the ex-dividend date corresponding to the amount in the second column. Default = no dividend.
CallType	0 = call on cash price (default). 1 = call on clean price.
CallInfo	(Optional) Two-column matrix of call information. First column contains the call dates. Second column contains call prices for every $100 face of bond. A call in the amount of call prices is activated *after* the corresponding call date. Default = no call feature.
TreeType	(Optional) 0 = binomial tree (default). 1 = trinomial tree.

All inputs are scalars except for DividendInfo and CallInfo.

Description [CBMatrix, UndMatrix, DebtMatrix, EqtyMatrix] = cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ConvRatio, NumSteps, IssueDate, Settle, Maturity, CouponRate, Period, Basis, EndMonthRule, DividendType, DividendInfo, CallType, CallInfo, TreeType) computes the price of a convertible bond using a Cox-Ross-Rubinstein binomial tree or, optionally, a trinomial tree.

CBMatrix is a matrix of convertible bond prices.

UndMatrix is a matrix of stock prices in binomial format.

DebtMatrix is a matrix of the debt portion of the convertible bond.

EqtyMatrix is a matrix of the equity portion of the convertible bond.

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 YTM 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    = datenum('2-Jan-2002');
Settle       = datenum('2-Jan-2002');
Maturity     = datenum('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;

% Nested loop accross prices and static spread dimensions
% to compute convertible prices.

for j = 0:0.005:0.015;
StaticSpread = j;
     for i = 0:10:100
         Price = 40+i;
         [CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] = ...
          cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ...
          ConvRatio, NumSteps, IssueDate, Settle, ...
          Maturity, CouponRate, Period, Basis, EndMonthRule, ...
          DividendType, DividendInfo, CallType, CallInfo, ...
          TreeType);

         convprice(i/10+1,j*200+1) =  CbMatrix(1,1);
         stock(i/10+1,j*200+1)     =  Price;
     end
```

```
end

    plot(stock, convprice);
    legend({'+O 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 two years.' sprintf('\n'), ...
            'Maturing at par in five years.'],'fontweight','Bold')
```
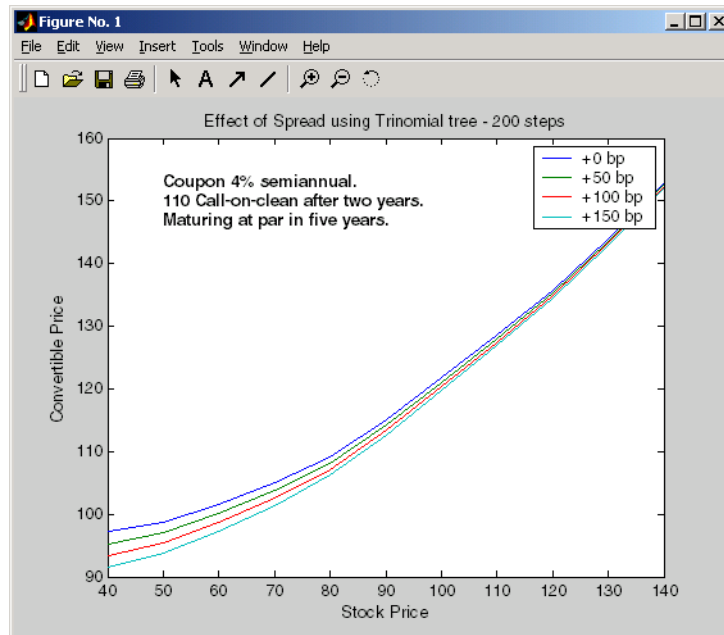
cdai

Purpose	Accrued interest on certificate of deposit (CD)
Syntax	AccrInt = cdai(CouponRate, Settle, Maturity, IssueDate, Basis)

Arguments

CouponRate	Annual interest rate in decimal.
Settle	Settlement date. Settle must be earlier than or equal to Maturity.
Maturity	Maturity date.
IssueDate	Issue date.
Basis	(Optional) Day-count basis of the instrument.

- 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

Each required input must be some certificates of deposit (NCDS)-by-1 or 1-by-NCDS conforming vector or scalar. The optional Basis argument may be either a NCDS-by-1 or a 1-by-NCDS vector, a scalar, or the empty matrix ([ ]).

Description   AccrInt = cdai(CouponRate, Settle, Maturity, IssueDate, Basis) computes the accrued interest on a certificate of deposit.

AccrInt represents the accrued interest per $100 of face value.

This function assumes that the certificates of deposit pay interest at maturity. Because of the simple interest treatment of these securities, the function is best used for short-term maturities (less than 1 year). The default simple interest calculation is the actual/360 convention (SIA).

Examples   Given a certificate of deposit (CD) with these characteristics, compute the accrued interest due on the CD.

```
CouponRate       =  0.05;
Settle           =  '02-Jan-02';
Maturity         =  '31-Mar-02';
IssueDate        =  '1-Oct-01';

AccrInt = cdai(CouponRate, Settle, Maturity, IssueDate)

AccrInt =

    1.2917
```

See Also   accrfrac, bndyield, stepcpnyield, tbillyield, zeroyield

cdprice

Purpose	Price of certificate of deposit (CD)
Syntax	[Price, AccrInt] = cdprice(Yield, CouponRate, Settle, Maturity, IssueDate, Basis)

Arguments

Yield	Simple yield to maturity over the basis denominator.
CouponRate	Coupon interest rate in decimal.
Settle	Settlement date. Settle must be earlier than or equal to Maturity.
Maturity	Maturity date.
IssueDate	Issue date.
Basis	(Optional) Day-count basis of the instrument.

- 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

Each required input must be some certificates of deposit (NCDS)-by-1 or 1-by-NCDS conforming vector or scalar. The optional Basis argument may be either a NCDS-by-1 or a 1-by-NCDS vector, a scalar, or the empty matrix ([ ]).

Description [Price, AccrInt] = cdprice(Yield, CouponRate, Settle, Maturity, IssueDate, Basis) computes the price of a certificate of deposit given its yield.

Price is the clean price of the CD per $100 of face value.

AccruedInt is the accrued interest payable at settlement per unit of face value.

This function assumes that the certificates of deposit pay interest at maturity. Because of the simple interest treatment of these securities, the function is best used for short-term maturities (less than 1 year). The default simple interest calculation is the actual/360 convention.

Examples Given a certificate of deposit (CD) with these characteristics, compute the price of the CD and the accrued interest due on the settlement date.

```
Yield          =  0.0525;
CouponRate     =  0.05;
Settle         =  '02-Jan-02';
Maturity       =  '31-Mar-02';
IssueDate      =  '1-Oct-01';

[Price, AccruedInt] = cdprice(Yield, CouponRate, Settle, ...
Maturity, IssueDate)

Price =

    99.9233

AccruedInt =
```

cdprice

        1.2917

See Also   bndprice, cdai, cdyield, stepcpnprice, tbillprice

Purpose   Yield on certificate of deposit (CD)

Syntax   Yield = cdyield(Price, CouponRate, Settle, Maturity, IssueDate, Basis)

Arguments

Price	Clean price of the certificate of deposit per $100 face. If you have a vector of dirty or cash prices of CDs, compute the accrued interest portion using cdai.
CouponRate	Annual interest rate in decimal.
Settle	Settlement date. Settle must be earlier than or equal to Maturity.
Maturity	Maturity date.
IssueDate	Issue date.
Basis	(Optional) Day-count basis of the instrument.

- 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

Each required input must be some certificates of deposit (NCDS)-by-1 or 1-by-NCDS conforming vector or scalar. The optional `Basis` argument may be either a NCDS-by-1 or a 1-by-NCDS vector, a scalar, or the empty matrix ([ ]).

Description   `Yield = cdyield(Price, CouponRate, Settle, Maturity, IssueDate, Basis)` computes the yield to maturity of a certificate of deposit given its clean price.

This function assumes that the certificates of deposit pay interest at maturity. Because of the simple interest treatment of these securities, the function is best used for short-term maturities (less than 1 year). The default simple interest calculation is the actual/360 convention.

Examples   Given a certificate of deposit (CD) with these characteristics, compute the yield on the CD.

```
Price      = 101.125;
CouponRate = 0.05;
Settle     = '02-Jan-02';
Maturity   = '31-Mar-02';
IssueDate  = '1-Oct-01';

Yield = cdyield(Price, CouponRate, Settle, Maturity, IssueDate)

Yield =

     0.0039
```

See Also   bndprice, cdai, cdprice, stepcpnprice, tbillprice

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   RefDate
Arguments
                        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 Enter the following inputs only as parameter/value pairs.
Pairs
                    Convention

                        Conversion factor convention. Scalar. Valid values are:

                        • 1 = U.S. Treasury bond (30-year) and Treasury note (10-year)
                          futures contract

convfactor

- `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 4-12

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.
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 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

Instrument Parameters

For each bond Instrument, you can specify the following additional instrument parameters as parameter/value pairs by prepending the word Instrument to the parameter field. For example, prepending InstrumentBasis distinguishes a bond instrument's Basis value from the curve's Basis value.

CouponRate   (Optional) Decimal number indicating the annual percentage rate used to determine the coupons payable on a bond.

Period   (Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.

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

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 an instrument was issued.
FirstCouponDate	(Optional) Date when a bond makes its first coupon payment. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date. 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 will be followed only by the bond's maturity cash flow date.
Face	(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 = optimset('lsqnonlin');
OptOptions = optimset(OptOptions,'display','iter');
functionHandle = @(t,theta) polyval(theta,t);

CustomModel = IRFunctionCurve.fitFunction('Zero', CurveSettle, ...
functionHandle,Instruments, ...
IRFitOptions([.05 .05 .05],'FitType','price',...
'OptOptions',OptOptions));
```

		Norm of	First-order		
Iteration	Func-count	f(x)	step	optimality	CG-iterations
0	4	38036.7		4.92e+004	
1	8	38036.7	10	4.92e+004	0
2	12	38036.7	2.5	4.92e+004	0
3	16	38036.7	0.625	4.92e+004	0

4	20	38036.7	0.15625	4.92e+004	0
5	24	30741.5	0.0390625	1.72e+005	0
6	28	30741.5	0.078125	1.72e+005	0
7	32	30741.5	0.0195312	1.72e+005	0
8	36	28713.6	0.00488281	2.33e+005	0
9	40	20323.3	0.00976562	9.47e+005	0
10	44	20323.3	0.0195312	9.47e+005	0
11	48	20323.3	0.00488281	9.47e+005	0
12	52	20323.3	0.0012207	9.47e+005	0
13	56	19698.8	0.000305176	1.08e+006	0
14	60	17493	0.000610352	7e+006	0
15	64	17493	0.0012207	7e+006	0
16	68	17493	0.000305176	7e+006	0
17	72	15455.1	7.62939e-005	2.25e+007	0
18	76	15455.1	0.000177558	2.25e+007	0
19	80	13317.1	3.8147e-005	3.18e+007	0
20	84	12867.9	7.62939e-005	7.84e+007	0
21	88	11779.8	7.62939e-005	7.58e+006	0
22	92	11747.6	0.000152588	1.46e+005	0
23	96	11720.9	0.000305176	2.48e+005	0
24	100	11667.2	0.000610352	1.48e+005	0
25	104	11558.5	0.0012207	4.47e+005	0
26	108	11335.4	0.00244141	1.58e+005	0
27	112	10864	0.00488281	1.61e+005	0
28	116	9797.68	0.00976562	6.85e+005	0
29	120	6884.03	0.0195312	5.79e+005	0
30	124	6884.03	0.037498	5.79e+005	0
31	128	3216.51	0.00937449	1.75e+006	0
32	132	607.317	0.018749	2.94e+006	0
33	136	12.7284	0.0253662	3e+006	0
34	140	0.0760939	0.00153457	4.88e+004	0
35	144	0.0731652	3.58678e-006	24.6	0
36	148	0.0731652	6.04329e-008	0.0213	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.

See Also   "@IRFitOptions" on page A-10, "@IRFunctionCurve" on page A-12

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 • 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 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

IRFitOptionsObj	(Optional) Object constructed from IRFitOptions.

Instrument Parameters

For each bond Instrument, you can specify the following additional instrument parameters as parameter/value pairs by prepending the word Instrument to the parameter field. For example, prepending InstrumentBasis distinguishes a bond instrument's Basis value from the curve's Basis value.

CouponRate	(Optional) Decimal number indicating the annual percentage rate used to determine the coupons payable on a bond.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.
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

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 an instrument was issued.
FirstCouponDate	(Optional) Date when a bond makes its first coupon payment. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date. 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 will be followed only by the bond's maturity cash flow date.
Face	(Optional) Face or par value. Default = 100.

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.

fitNelsonSiegel

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.

Examples
```
NSModel = IRFunctionCurve.fitNelsonSiegel('Zero',datenum('30-Apr-2008'),Instruments);
Optimization terminated: first-order optimality less than OPTIONS.TolFun,
 and no negative/zero curvature detected in trust region model.
```

To create the plot:

```
plot(PlottingPoints,NSModel.getParYields(PlottingPoints),'r')
hold on
scatter(Maturity,Yield,'black')
```
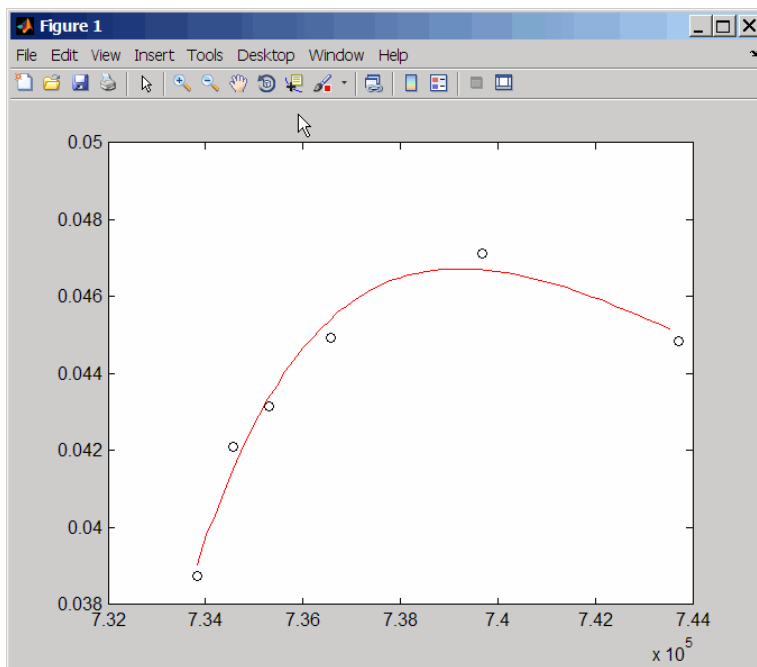
See Also   "@IRFitOptions" on page A-10, "@IRFunctionCurve" on page A-12

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 Spline Toolbox software to use the fitSmoothingSpline method.

Type	Type of interest-rate curve for a bond: 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.
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.

> **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 O and the time to maturity of all input instruments.
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 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

Instrument Parameters

For each bond Instrument, you can specify the following additional instrument parameters as parameter/value pairs by prepending the word Instrument to the parameter field. For example, prepending InstrumentBasis distinguishes a bond instrument's Basis value from the curve's Basis value.

CouponRate	(Optional) Decimal number indicating the annual percentage rate used to determine the coupons payable on a bond.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.

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

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 an instrument was issued.

FirstCouponDate	(Optional) Date when a bond makes its first coupon payment. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.	
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date. 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 will be followed only by the bond's maturity cash flow date.	
Face	(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.

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);

SmoothingModel = IRFunctionCurve.fitSmoothingSpline('Forward',datenum('30-Apr-2008'),...
Instruments,@(t) 1000);
```

To create the plot:

```
plot(PlottingPoints,SmoothingModel.getParYields(PlottingPoints),'b')
hold on
scatter(Maturity,Yield,'black')
```
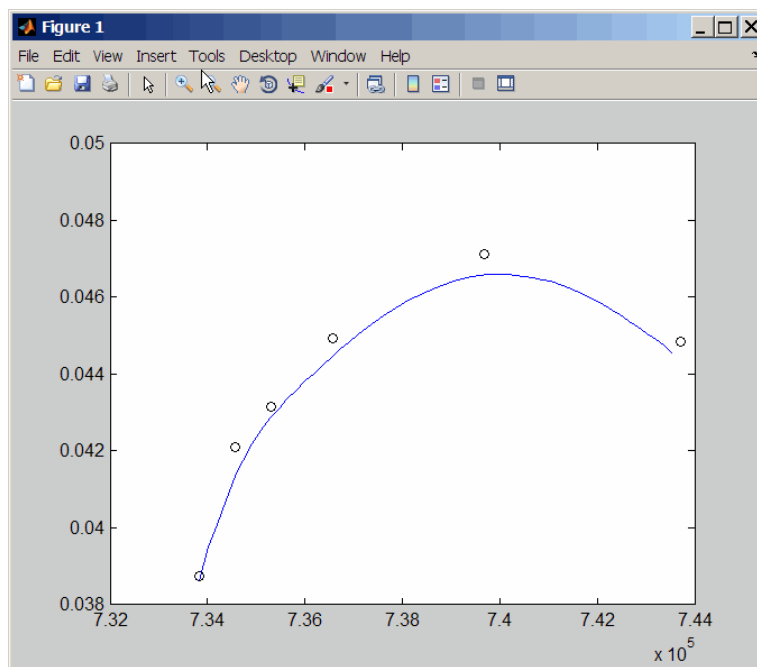

See Also   "@IRFunctionCurve" on page A-12

fitSvensson

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

- 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 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

IRFitOptionsObj	(Optional) Object constructed from IRFitOptions.

Instrument Parameters   For each bond Instrument, you can specify the following additional instrument parameters as parameter/value pairs by prepending the word Instrument to the parameter field. For example, prepending InstrumentBasis distinguishes a bond instrument's Basis value from the curve's Basis value.

fitSvensson

CouponRate	(Optional) Decimal number indicating the annual percentage rate used to determine the coupons payable on a bond.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.
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

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 an instrument was issued.
FirstCouponDate	(Optional) Date when a bond makes its first coupon payment. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date. 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 will be followed only by the bond's maturity cash flow date.
Face	(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.

---

fitSvensson

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 `IRFitOptionsObj` 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];
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);
```

To create a plot:

```
plot(PlottingPoints,SvenssonModel.getParYields(PlottingPoints),'g')
hold on
scatter(Maturity,Yield,'black')
```
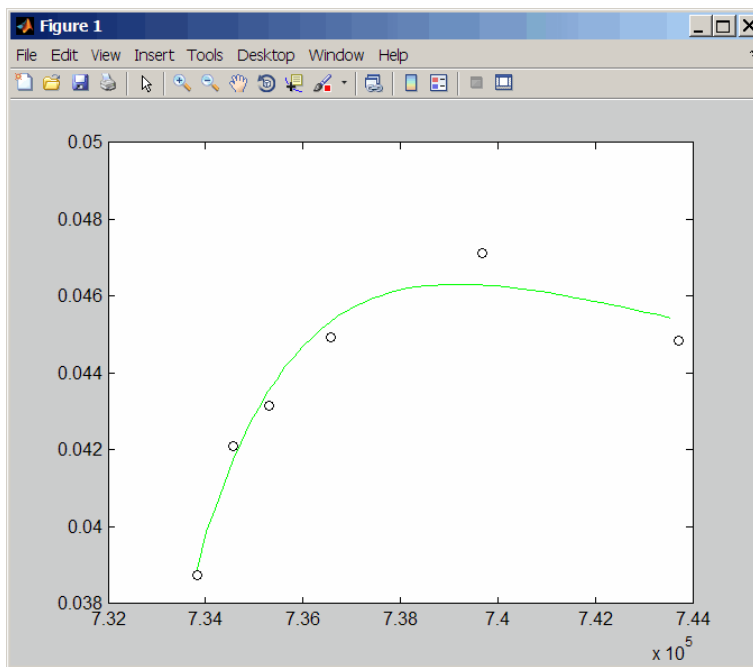
See Also "@IRFitOptions" on page A-10, "@IRFunctionCurve" on page A-12

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
```

See Also   "@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
```

See Also   "@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

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
```

See Also   "@IRDataCurve" on page A-7

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

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
```

See Also   "@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

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
```

See Also   "@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)

- 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

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
```

See Also   "@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

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
```

See Also   "@IRDataCurve" on page A-7

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)

- 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

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
```

See Also   "@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))
```

See Also   "@IRDataCurve" on page A-7

Purpose	Construct interest-rate curve object from dates and data

Class	@IRDataCurve

Syntax	CurveObj = IRDataCurve(Type, Settle) 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 or column vector of settlement dates.
Dates	(Optional) Dates corresponding to rate data.
Data	(Optional) 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 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

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 optionally 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: 0
```

See Also   "@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 structure based on the output from the Optimization Toolbox function optimset. This optimization structure is evaluated by lsqnonlin.

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: []
```

See Also   "@IRFunctionCurve" on page A-12

Purpose	Construct interest-rate curve object from function handle or function by fitting to market data using object methods
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 or column vector 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

IRFunctionCurve

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

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 Derivatives 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'
```

IRFunctionCurve

```
              Settle: 733599
         Compounding: 2
               Basis: 0
```

See Also "@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.

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

- 1 = 30/360 (SIA, default)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = act/365 (Japanese)

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.

```
[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

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

- 1 = 30/360 (SIA, default)

- 2 = actual/360

- 3 = actual/365

- 4 = 30/360 (PSA)

- 5 = 30/360 (ISDA)

- 6 = 30/360 (European)

- 7 = act/365 (Japanese)

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, Settle, Tenor);
```

Compute the price of a par swap.

```
Price = liborprice(EDFutData, 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] =
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 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	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. 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] =
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).

Examples **Example 1**. Given a mortgage with the following characteristics,
compute the cash flow amounts and dates, the time factors, and the
mortgage factors.

```
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;

[CFlowAmounts, CFLowDates, TFactors, Factors] = ...
mbscfamounts(Settle, Maturity, IssueDate, GrossRate, ...
CouponRate, Delay, PrepaySpeed)
```

The result is contained in four 334-element row vectors.

Example 2. Given a portfolio of mortgage-backed securities, use mbscfamounts to compute the cash flows and other factors from the 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;

[CFlowAmounts, CFlowDates, TFactors, Factors] = ...
mbscfamounts(Settle, Maturity, IssueDate, GrossRate, ...
CouponRate, Delay, PrepaySpeed)
```

Each output is a 3-by-361 element matrix padded with NaNs wherever elements are missing.

References   [1] *PSA Uniform Practices*, SF-4

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 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	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.

mbsconvp

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, mbsdury

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 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	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)

mbsconvy

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

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	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.

mbsdurp

Description [YearDuration, ModDuration] = mbsdurp(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes the mortgage-backed security Macaulay (YearDuration) and modified (ModDuration) durations, 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

See Also   `mbsconvp, mbsconvy, mbsdury`

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 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	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] = mbsdurvy(Yield, Settle,
Maturity, IssueDate, GrossRate, CouponRate, Delay,
PrepaySpeed, PrepayMatrix) computes the mortgage-backed security
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
```

mbsdury

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);
```

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.
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)
```

```
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.
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   [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

mbspassthrough

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, Interest, Prepayment] = 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.

mbspassthrough

Balance is the principal balance at end of month.

Payment is the total monthly payment.

Principal is the principal portion of the payment.

Interest is the interest portion of the payment.

Prepayment 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 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) 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.

mbsprice

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.
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.

<table>
<tr>
<td>Delay</td>
<td>(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt.</td>
</tr>
<tr>
<td>Interpolation</td>
<td>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.</td>
</tr>
<tr>
<td>PrepaySpeed</td>
<td>(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.</td>
</tr>
<tr>
<td>PrepayMatrix</td>
<td>(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.</td>
</tr>
</table>

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 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.
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

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 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) 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.

mbswal

Description	`WAL = mbswal(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)` computes the 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 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) 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 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.
	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];
```

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 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.
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
```

mbsyield2speed

```
ImpSpdOnCnv =

   108.3309
```

References   [1] *PSA Uniform Practices*, SF-49

See Also   mbsyield, mbsprice2speed

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].

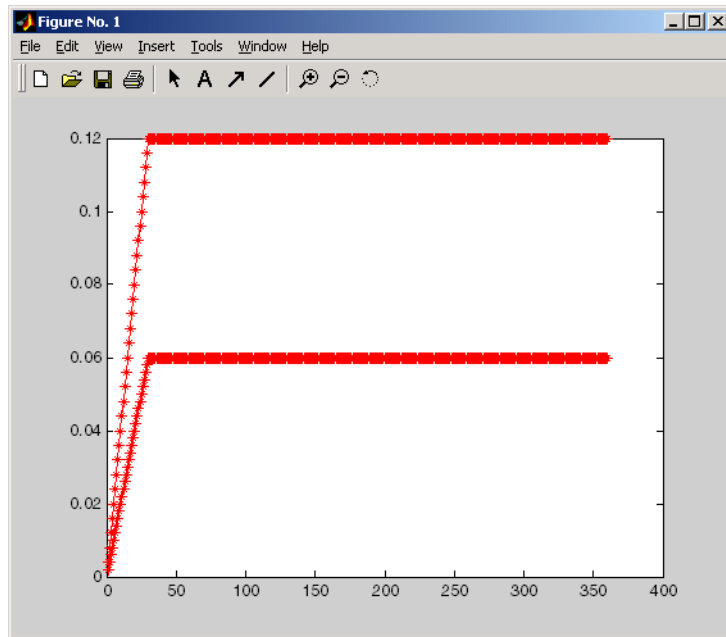`SMMPSA` is the PSA single monthly mortality rate, in decimal [360-by-1].

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 or equal to 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

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.
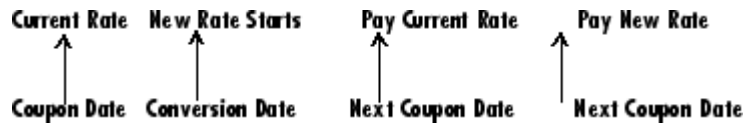
The following table illustrates the interest rate characteristics of this bond portfolio.

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). (`cfplot` is available at `/finance/findemos/cfplot.m`.)

```
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')
```
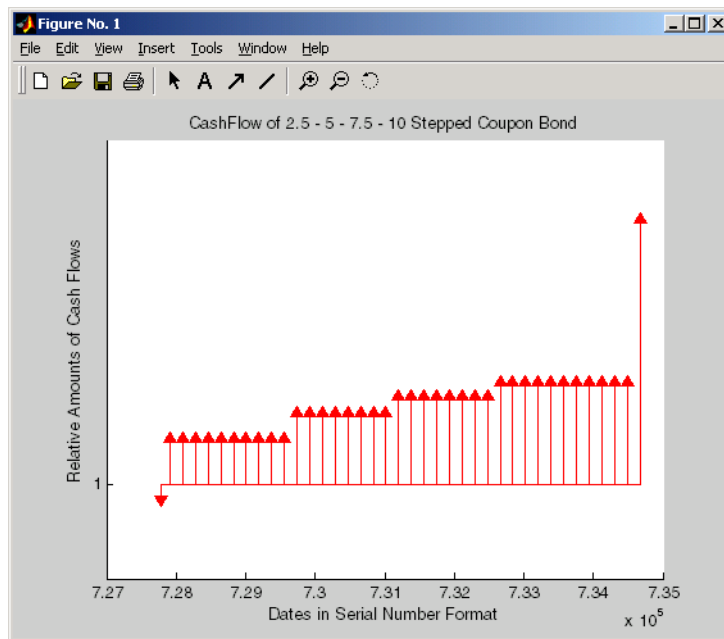

See Also   `stepcpnprice`, `stepcpnyield`

stepcpnprice

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 or equal to `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

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.

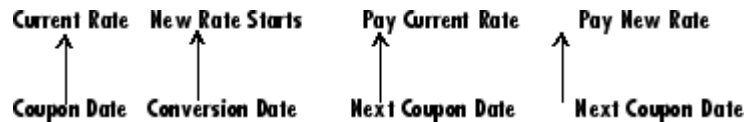**Note** For bonds with fixed coupons, use bndprice. If you use a fixed-coupon bond with stepcpnprice, you will receive the error: incorrect number of inputs.

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 bndprice, cdprice, stepcpncfamounts, stepcpnyield, tbillprice, zeroprice

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 or equal to 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

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 bndprice, cdprice, stepcpncfamounts, stepcpnprice, tbillprice, zeroprice

tbilldisc2yield

Purpose   Convert Treasury bill discount to equivalent yield

Syntax   [BEYield MMYield] = tbilldisc2yield(Discount, Settle, Maturity)

Arguments

Discount	Discount rate of Treasury bills in decimal. The discount rate basis is actual/360.
Settle	Settlement date. Settle must be earlier than or equal to Maturity.
Maturity	Maturity date.

Inputs must either be a scalar or a vector of size equal to the number of Treasury bills (NTBILLS) by 1 or 1-by-NTBILLS.

Description [BEYield MMYield] = tbilldisc2yield(Yield, Settle, Maturity) converts the discount rate on Treasury bills into their respective money-market or bond-equivalent yields.

BEYield is an NTBILLS-by-1 vector of bond-equivalent yields. The bond-equivalent yield basis is actual/365.

MMYield is an NTBILLS-by-1 vector of money-market yields. The money-market yield basis is actual/360.

Examples   Given a Treasury bill with these characteristics, compute the bond-equivalent and money-market yields.

```
Discount = 0.0497;
Settle = '01-Oct-02';
Maturity = '31-Mar-03';

[BEYield MMYield] = tbilldisc2yield(Discount, Settle, Maturity)

BEYield =
```

```
        0.0517

    MMYield =

        0.0510
```

References   This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 44 - 45 (on Treasury bills), and *Money Market and Bond Calculation* by Stigum and Robinson.

See Also   tbillyield2disc, zeroyield

tbillprice

Purpose	Price Treasury bill
Syntax	Price = tbillprice(Rate, Settle, Maturity, Type)

Arguments

Rate	Bond-equivalent yield, money-market yield, or discount rate in decimal.
Settle	Settlement date. Settle must be earlier than or equal to Maturity.
Maturity	Maturity date.
Type	(Optional) Rate type. Determines how to interpret values entered in Rate. 1 = money market (default). 2 = bond-equivalent. 3 = discount rate.

All arguments must be a scalar or some Treasury bills (NTBILLS) by 1 or 1-by-NTBILLS vector.

Note The bond-equivalent yield basis is actual/365. The money-market yield basis is actual/360. The discount rate basis is actual/360.

Description	Price = tbillprice(Rate, Settle, Maturity, Type) computes the price of a Treasury bill given a yield or discount rate.
	Price is an NTBILLS-by-1 vector of T-bill prices for every $100 face.
Examples	**Example 1**. Given a Treasury bill with these characteristics, compute the price of the Treasury bill using the bond-equivalent yield as input.

```
Rate = 0.045;
Settle = '01-Oct-02';
```

```
Maturity = '31-Mar-03';

Type = 2;

Price = tbillprice(Rate, Settle, Maturity, Type)

Price =

   97.8172
```

Example 2. Use `tbillprice` to price a portfolio of Treasury bills.

```
Rate = [0.045; 0.046];
Settle = {'02-Jan-02'; '01-Mar-02'};
Maturity = {'30-June-02'; '30-June-02'};
Type = [2 3];

Price = tbillprice(Rate, Settle, Maturity, Type)

Price =

   97.8408
   98.4539
```

References   This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 44 - 45 (on Treasury bills), and *Money Market and Bond Calculation* by Stigum and Robinson.

See Also   `tbillyield`, `zeroprice`

tbillrepo

Purpose	Break-even discount of repurchase agreement

Syntax TBEDiscount = tbillrepo(RepoRate, InitialDiscount, PurchaseDate, SaleDate, Maturity)

Arguments

RepoRate	The annualized, 360-day based repurchase rate, in decimal.
InitialDiscount	Discount on the Treasury bill on the day of purchase, in decimal.
PurchaseDate	Date the Treasury bill is purchased.
SaleDate	Date the Treasury bill repurchase term is due.
Maturity	Treasury bill maturity date.

All arguments must be a scalar or some Treasury bills (NTBILLS) by 1 or a 1-by-NTBILLS vector.

All dates must be in serial date number format.

Description TBEDiscount = tbillrepo(RepoRate, InitialDiscount, PurchaseDate, SaleDate, Maturity) computes the true break-even discount of a repurchase agreement. TBEDiscount can be a scalar or vector of size NTBills-by-1.

Examples Compute the true break-even discount on a Treasury bill repurchase agreement.

```
RepoRate = [0.045; 0.0475];
InitialDiscount = 0.0475;
PurchaseDate = '3-Jan-2002';
SaleDate = '3-Feb-2002';
Maturity = '3-Apr-2002';
```

```
TBEDiscount = tbillrepo(RepoRate, InitialDiscount,...
PurchaseDate, SaleDate, Maturity)

TBEdiscount =

    0.0491
    0.0478
```

References This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 44 - 45 (on Treasury bills), and *Money Market and Bond Calculation* by Stigum and Robinson.

tbillval01

Purpose	Value of one basis point
Syntax	[Val01Disc, Val01MMY, Val01BEY] = tbillval01(Settle, Maturity)

Arguments

Settle	Settlement date of Treasury bills. Settle must be earlier than or equal to Maturity.
Maturity	Maturity date of Treasury bills.

Description
[Val01Disc, Val01MMY, Val01BEY] = tbillval01(Settle, Maturity) calculates the value of one basis point of $100 Treasury bill face value on the discount rate, money-market yield, or bond-equivalent yield.

Val01Disc is the value of one basis point of discount rate.

Val01MMY is the value of one basis point of money-market yield.

Val01BEY is the value of one basis point of bond-equivalent yield.

All outputs are of size equal to the number of Treasury bills (NTBILLS) by 1.

Examples
Given a Treasury bill with these settle and maturity dates, compute the value of one basis point.

```
Settle = '01-Mar-03';
Maturity = '30-June-03';
[Val01Disc, Val01MMY, Val01BEY] = tbillval01(Settle, Maturity)

Val01Disc =

    0.0034
```

```
Val01MMY =

    0.0034

Val01BEY =

    0.0033
```

References   This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp 108 - 115, on zero coupon instrument pricing.

See Also   tbilldisc2yield, tbillprice, tbillyield, tbillyield2disc

tbillyield

Purpose

Yield on Treasury bill

Syntax

[MMYield, BEYield, Discount] = tbillyield(Price, Settle, Maturity)

Arguments

Price	Price of Treasury bills for every $100 face value.
Settle	Settlement date. Settle must be earlier than or equal to Maturity.
Maturity	Maturity date.

All arguments must be a scalar or some Treasury bills (NTBILLS) by 1 or 1-by-NTBILLS vector.

Description

[MMYield, BEYield, Discount] = tbillyield(Price, Settle, Maturity) computes the yield of U.S. Treasury bills given Price, Settle, and Maturity. MMYield is the money-market yields of the Treasury bills. BEYield is the bond equivalent yields of the Treasury bills. Discount is the discount rates of the Treasury bills.

All outputs are NTBILLS-by-1 vectors.

Note The money-market yield basis is actual/360. The bond-equivalent yield basis is actual/365. The discount rate basis is actual/360.

Examples

Given a Treasury bill with these characteristics, compute the money-market and bond-equivalent yields and the discount rate.

```
Price = 98.75;
Settle = '01-Oct-02';
Maturity = '31-Mar-03';
```

```
[MMYield, BEYield, Discount] = tbillyield(Price, Settle,...
Maturity)

MMYield =

    0.0252

BEYield =

    0.0255

Discount =

    0.0249
```

References   This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 44 - 45 (on Treasury bills), and *Money Market and Bond Calculation* by Stigum and Robinson.

See Also   tbilldisc2yield, tbillprice, tbillyield2disc, zeroyield

tbillyield2disc

Purpose	Convert Treasury bill yield to equivalent discount

Syntax   `Discount = tbillyield2disc(Yield, Settle, Maturity, Type)`

Arguments

Yield	Yield of Treasury bills in decimal.
Settle	Settlement date. `Settle` must be earlier than or equal to `Maturity`.
Maturity	Maturity date.
Type	(Optional) Yield type. Determines how to interpret values entered in `Yield`. 1 = money market (default). 2 = bond-equivalent.

Inputs must either be a scalar or a vector of size equal to the number of Treasury bills (`NTBILLS`) by 1 or 1-by-`NTBILLS`.

Note The money-market yield basis is actual/360. The bond-equivalent yield basis is actual/365. The discount rate basis is actual/360.

Description   `Discount = tbillyield2disc(Yield, Settle, Maturity, Type)` converts the yield on some Treasury bills into their respective discount rates.

`Discount` is a `NTBILLS`-by-1 vector of T-bill discount rates.

Examples   Given a Treasury bill with these characteristics, compute the discount rate on a money-market basis.

```
Yield = 0.0497;
Settle = '01-Oct-02';
Maturity = '31-Mar-03';

Discount = tbillyield2disc(Yield, Settle, Maturity)
```

```
Discount =

    0.0485
```

Now recompute the discount on a bond-equivalent basis.

```
Discount = tbillyield2disc(Yield, Settle, Maturity, 2)

Discount =

    0.0478
```

References   This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 44 - 45 (on Treasury bills), and *Money Market and Bond Calculation* by Stigum and Robinson.

See Also   tbilldisc2yield

tfutbyprice

Purpose	Future prices of Treasury bonds given spot price	
Syntax	QtdFutPrice = 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.
	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 = tfutbyprice(SpotCurve, Price, SettleFut,
MatFut, ConvFactor, CouponRate, Maturity, Interpolation)
computes future prices of Treasury notes and bonds given the spot price.

In addition, you can use the Fixed-Income 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 5-25.

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 = tfutbyprice(SpotCurve, Price, Settle, ...
MatFut, ConvFactor, CouponRate, Maturity, Interpolation)

QtdFutPrice =

  114.0409
  113.4029
```

This compares with closing prices of 113.93 and 112.68. The differences are expected due to the nature of the contract and data that is not directly comparable.

See Also   convfactor, tfutbyyield

Purpose   Future prices of Treasury bonds given current yield

Syntax   QtdFutPrice = tfutbyyield(SpotCurve, Yield, 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.
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 = tfutbyyield(SpotCurve, Yield, SettleFut, MatFut, ConvFactor, CouponRate, Maturity, Interpolation)

computes future prices of Treasury notes and bonds given current yields
of Treasury bonds/notes.

In addition, you can use the Fixed-Income 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 5-25.

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 = tfutbyyield(SpotCurve, Yield, Settle, ...
MatFut, ConvFactor, CouponRate, Maturity, Interpolation)

QtdFutPrice =

  114.0416
  113.4034
```

This compares to closing prices of 113.93 and 112.68. The differences are expected because of the nature of the contract and data that are not directly comparable.

See Also   convfactor, tfutbyprice

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

tfutpricebyrepo

Purpose	Implied repo rates given Treasury bond future price

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

tfutyieldbyrepo

Purpose	Implied repo rates given Treasury bond future yield
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 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

toRateSpec

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 Derivatives 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
```

See Also   "@IRDataCurve" on page A-7

toRateSpec

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 Derivatives 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: O
               EndMonthRule: 1
```

See Also   "@IRFunctionCurve" on page A-12

zeroprice

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 or equal to 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

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 the instruments. Price is a column vector containing a price for each zero-coupon instrument.

When there is less than one quasi-coupon, the function uses a simple yield based upon "Period times 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).

Formulas   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.

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

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 or equal to `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

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. Yield is a column vector containing a 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.

Formulas   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.
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

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 Derivatives 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 Derivatives 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`. This optimization structure is evaluated by `lsqnonlin`.

Methods

There are no methods.

@IRFunctionCurve

Represent an interest-rate curve object using a function

In this section...
"Hierarchy" on page A-12
"Description" on page A-12
"Constructor" on page A-12
"Public Read-Only Properties" on page A-13
"Methods" on page A-14

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 Derivatives 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.

Bibliography

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.

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.

C

Examples

Use this list to find examples in the documentation.

Treasury Bills

Using Zero-Coupon Bonds

Stepped-Coupon Bonds

Pricing and Hedging

Bond Futures

Glossary

American option

    An option that can be exercised any time until its expiration date. Contrast with European option.

amortization

    Reduction in value of an asset over some period for accounting purposes. Generally used with intangible assets. Depreciation is the term used with fixed or tangible assets.

annuity

    A series of payments over a period of time. The payments are usually in equal amounts and usually at regular intervals such as quarterly, semiannually, or annually.

arbitrage

    The purchase of securities on one market for immediate resale on another market to profit from a price or currency discrepancy.

basis point

    One hundredth of one percentage point, or 0.0001.

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 of pricing options or other equity derivatives in which the probability over time of each possible price follows a binomial distribution. The basic assumption is that prices can move to only two values (one higher and one lower) over any short time period.

Black-Scholes model

    The first complete mathematical model for pricing options, developed by Fischer Black and Myron Scholes. It examines market price, strike price, volatility, time to expiration, and interest rates. It is limited to only certain kinds of options.

Bollinger band chart
    A financial chart that plots actual asset data along with three other bands of data: the upper band is two standard deviations above a user-specified moving average; the lower band is two standard deviations below that moving average; and the middle band is the moving average itself.

bootstrapping, bootstrap method
    A procedure for constructing a term structure from a set of market instruments by progressively deriving rates.

building a binomial tree
    For a binomial option model: plotting the two possible short-term price-changes values, and then the subsequent two values each, and then the subsequent two values each, and so on, over time, is known as "building a binomial tree." See also **binomial model** on page Glossary-1.

call
    a. An option to buy a certain quantity of a stock or commodity for a specified price within a specified time. See **put** on page Glossary-10. **b.** A demand to submit bonds to the issuer for redemption before the maturity date. **c.** A demand for payment of a debt. **d.** A demand for payment due on stock bought on margin.

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-10.

cap
    Interest-rate option that guarantees that the rate on a floating-rate loan will not exceed a certain level.

caplet
    A cap that is guaranteed for one particular date.

cash flow
    Cash received and paid over time.

cheapest to deliver

Cheapest to deliver represents the least expensive underlying product that can be delivered upon expiry to satisfy the requirements of a derivative contract.

clean price

The price of a bond excluding any interest that has accrued since issue or the most recent coupon payment.

collar

Interest-rate option that guarantees that the rate on a floating-rate loan will not exceed a certain upper level nor fall below a lower level. It is designed to protect an investor against wide fluctuations in interest rates.

conditional prepayment rate (CPR)

The fraction of mortgage principal that had not prepaid at the beginning of any year but does prepay during the year. CPR is an annualization of the single monthly mortality rate. See also **single monthly mortality (SMM)** on page Glossary-11.

conversion factor

The rate used to adjust differences in bond values for delivery on U.S. Treasury bond futures contracts.

convexity

A measure of the rate of change in duration; measured in time. The greater the rate of change, the more the duration changes as yield changes.

correlation

The simultaneous change in value of two random numeric variables.

correlation coefficient

A statistic in which the covariance is scaled to a value between minus one (perfect negative correlation) and plus one (perfect positive correlation).

coupon

Detachable certificate attached to a bond that shows the amount of interest payable at regular intervals, usually semiannually. Originally coupons were actually attached to the bonds and had to be cut off or "clipped" to redeem them and receive the interest payment.

coupon dates

The dates when the coupons are paid. Typically a bond pays coupons annually or semiannually.

coupon rate

The nominal interest rate that the issuer promises to pay the buyer of a bond.

covariance

A measure of the degree to which returns on two assets move in tandem. A positive covariance means that asset returns move together; a negative covariance means they vary inversely.

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.

depreciation

Reduction in value of fixed or tangible assets over some period for accounting purposes. See also **amortization** on page Glossary-1.

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.

dirty price

The price of a bond including the accrued interest.

discount curve

The curve of discount rates vs. maturity dates.

duration

The expected life of a fixed-income security considering its coupon yield, interest payments, maturity, and call features. As market interest rates rise, the duration of a financial instrument decreases. See also **Macaulay duration** on page Glossary-7.

efficient frontier

A graph representing a set of portfolios that maximizes expected return at each level of portfolio risk. See also **Markowitz model** on page Glossary-7.

elasticity

See **lambda** on page Glossary-7.

Eurodollar

U.S. dollar-denominated deposits at foreign banks or foreign branches of American banks.

European option

An option that can be exercised only on its expiration date. Contrast with American option.

exercise price

The price set for buying an asset (call) or selling an asset (put). The strike price.

face value

The maturity value of a security. Also known as par value, principal value, or redemption value.

fixed-income security

A security that pays a specified cash flow over a specific period. Bonds are typical fixed-income securities.

floor

Interest-rate option that guarantees that the rate on a floating-rate loan will not fall below a certain level.

forward curve

The curve of forward interest rates vs. maturity dates.

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.

forward rate agreement (FRA)

A forward contract that determines an interest rate to be paid or received on an obligation beginning at a start date sometime in the future.

future value

The value that a sum of money (the present value) earning compound interest will have in the future.

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.

Greeks

Collectively, "greeks" refer to the financial measures delta, gamma, lambda, rho, theta, and vega, which are sensitivity measures used in evaluating derivatives.

hedge

A securities transaction that reduces or offsets the risk on an existing investment position.

implied repo rate

Implied repo rate is the rate of return of borrowing money to buy an asset in the spot market and delivering it in the futures market where the notional is used to repay the loan.

implied volatility

For an option, the variance that makes a call option price equal to the market price. Given the option price, strike price, and other factors, the Black-Scholes model computes implied volatility.

internal rate of return

> **a.** The average annual yield earned by an investment during the period held. **b.** The effective rate of interest on a loan. **c.** The discount rate in discounted cash flow analysis. **d.** The rate that adjusts the value of future cash receipts earned by an investment so that interest earned equals the original cost. See also **yield to maturity** on page Glossary-14.

issue date

> The date a security is first offered for sale. That date usually determines when interest payments, known as coupons, are made.

lambda

> The percentage change in the price of an option relative to a 1% change in the price of the underlying security. Also known as elasticity.

LIBOR

> Abbreviation for London Interbank Offered Rate, an interest rate set daily in London. Applies to loans among large international banks.

long position

> Outright ownership of a security or financial instrument. The owner expects the price to rise to make a profit on some future sale.

long rate

> The yield on a zero-coupon Treasury bond.

Macaulay duration

> A widely used measure of price sensitivity to yield changes developed by Frederick Macaulay in 1938. It is measured in years and is a weighted average-time-to-maturity of an instrument. The Macaulay duration of an income stream, such as a coupon bond, measures how long, on average, the owner waits before receiving a payment. It is the weighted average of the times payments are made, with the weights at time T equal to the present value of the money received at time T.

Markowitz model

> A model for selecting an optimum investment portfolio, devised by H. M. Markowitz. It uses a discrete-time, continuous-outcome approach

for modeling investment problems, often called the mean-variance paradigm. See also **efficient frontier** on page Glossary-5.

maturity date

The date when the issuer returns the final face value of a bond to the buyer.

mean

a. A number that typifies a set of numbers, such as a geometric mean or an arithmetic mean. **b.** The average value of a set of numbers.

modified duration

The Macaulay duration discounted by the per-period interest rate; that is, divided by (1+rate/frequency).

Monte-Carlo simulation

A mathematical modeling process. For a model that has several parameters with statistical properties, pick a set of random values for the parameters and run a simulation. Then pick another set of values, and run it again. Run it many times (often 10,000 times) and build up a statistical distribution of outcomes of the simulation. This distribution of outcomes is then used to answer whatever question you are asking.

moving average

A price average that is adjusted by adding other parametrically determined prices over some time period.

moving-averages chart

A financial chart that plots leading and lagging moving averages for prices or values of an asset.

Nelson-Siegel model

A 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.

normal (bell-shaped) distribution

In statistics, a theoretical frequency distribution for a set of variable data, usually represented by a bell-shaped curve symmetrical about the mean.

notional

The nominal value used to calculate swap payments.

odd first or last period

Fixed-income securities may be purchased on dates that do not coincide with coupon or payment dates. The length of the first and last periods may differ from the regular period between coupons, and thus the bond owner is not entitled to the full value of the coupon for that period. Instead, the coupon is prorated according to how long the bond is held during that period.

off-the-run

All Treasury bonds and notes issued before the most recently issued bond or note of a particular maturity. These are the opposite of on-the-run treasuries.

on-the-run

The most recently issued U.S. Treasury bond or note of a particular maturity. These are the opposite of off-the-run treasuries.

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.

option-adjusted spread

A yield spread that is not directly attributable to the characteristics of a fixed income security.

pass-through

A type of mortgage-backed security in which the interest and principal payments on the underlying mortgages "pass through" to the holders, pro rata, minus a servicing fee.

par value

The maturity or face value of a security or other financial instrument.

par yield curve

The yield curve of bonds selling at par, or face, value.

piecewise constant interpolation

Interpolation where intermediate points take the value of the previous data point.

present value

Today's value of an investment that yields some future value when invested to earn compounded interest at a known interest rate; that is, the future value at a known period in time discounted by the interest rate over that time period.

principal value

See **par value** on page Glossary-9.

purchase price

Price paid for a security. Typically the purchase price of a bond is not the same as the redemption value.

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-2.

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-2.

redemption value

See **par value** on page Glossary-9.

regression analysis

Statistical analysis techniques that quantify the relationship between two or more variables. The intent is quantitative prediction or forecasting, particularly using a small population to forecast the behavior of a large population.

rho

The rate of change in a derivative's price relative to the underlying security's risk-free interest rate.

repo rate

   The discounted interest rate at which a central bank repurchases
   government securities.

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.

settlement date

   The date when money first changes hands; that is, when a buyer
   actually pays for a security. It need not coincide with the issue date.

short rate

   The annualized one-period interest rate.

short sale, short position

   The sale of a security or financial instrument not owned, in anticipation
   of a price decline and making a profit by purchasing the instrument
   later at a lower price, and then delivering the instrument to complete
   the sale. See **long position** on page Glossary-7.

single monthly mortality (SMM)

   The fraction of mortgage principal that had not prepaid at the
   beginning of a given month but does prepay during the month. See also
   conditional prepayment rate (CPR) on page Glossary-3.

smoothing spline

   Cubic spline that is smoothed by applying a penalty to the spline's
   second derivative.

spot curve, spot yield curve

   See **zero curve, zero-coupon yield curve** on page Glossary-14.

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.

standard deviation
> A measure of the variation in a distribution, equal to the square root of the arithmetic mean of the squares of the deviations from the arithmetic mean; the square root of the variance.

stochastic
> Involving or containing a random variable or variables; involving chance or probability.

straddle
> A strategy used in trading options or futures. It involves simultaneously purchasing put and call options with the same exercise price and expiration date, and it is most profitable when the price of the underlying security is volatile.

strike
> Exercise a put or call option.

strike price
> See **exercise price** on page Glossary-5.

Svensson model
> Extends the Nelson-Siegel model by adding a further term that allows for a second "hump." The extra precision is achieved by adding two more parameters, $\beta 3$ and $\tau 2$, which have to be estimated. See also **Nelson-Siegel model** on page Glossary-8.

swap
> A contract between two parties to exchange cash flows in the future according to some formula.

swap option
> A swap option; an option on an interest-rate swap. The option gives the holder the right to enter into a contracted interest-rate swap at a specified future date. See also **swap** on page Glossary-12.

tenor

Life of a swap.

term repo rate

Term repo rate is the rate of interest for a repurchase agreement that is structured to be in effect for a specific period of time. See also **implied repo rate** on page Glossary-6.

term structure

The relationship between the yields on fixed-interest securities and their maturity dates. Expectation of changes in interest rates affects term structure, as do liquidity preferences and hedging pressure. A yield curve is one representation in the term structure.

theta

The rate of change in the price of a derivative security relative to time. Theta is usually small or negative since the value of an option tends to drop as it approaches maturity.

Treasury bill

Short-term U.S. Government security issued at a discount from the face value and paying the face value at maturity.

Treasury bond

Long-term debt obligation of the U.S. Government that makes coupon payments semiannually and is sold at or near par value in $1000 denominations or higher. Face value is paid at maturity.

variance

The dispersion of a variable. The square of the standard deviation.

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

a. Another general term for sensitivity. **b.** The standard deviation of the annualized continuously compounded rate of return of an asset. **c.** A measure of uncertainty or risk.

yield

    a. Measure of return on an investment, stated as a percentage of price. Yield can be computed by dividing return by purchase price, current market value, or other measure of value. **b.** Income from a bond expressed as an annualized percentage rate. **c.** The nominal annual interest rate that gives a future value of the purchase price equal to the redemption value of the security. Any coupon payments determine part of that yield.

yield curve

    Graph of yields (vertical axis) of a particular type of security versus the time to maturity (horizontal axis). This curve usually slopes upward, indicating that investors usually expect to receive a premium for securities that have a longer time to maturity. The benchmark yield curve is for U.S. Treasury securities with maturities ranging from three months to 30 years. See **term structure** on page Glossary-13.

yield to maturity

    A measure of the average rate of return that will be earned on a bond if held to maturity.

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