Fixed-Income Toolbox

For Use with MATLAB®

- Computation
- Visualization
- Programming

User's Guide

Version 1

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	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. Mail
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

Printing History: May 2003 Online only New for Version 1.0 (Release 13)

Contents

Preface

1 Mortgage-Backed Securities

2 Debt Instruments

3 Derivative Securities

Function Reference

4

Glossary

A

Index

Preface

About This Book

This book describes the Fixed-Income Toolbox for MATLAB®, a collection of tools for evaluating mortgage-backed securities, short-term securities such as Treasury bills and certificates of deposit, and coupon-paying bond instruments.

The toolbox functions support fixed-rate mortgage pool construction and analysis, bond futures and conversion factors, convertible bond analysis, and LIBOR-based swap agreements.

You can compile and deploy applications you develop with this toolbox using MATLAB Excel Builder or MATLAB COM Builder. Also, if you have installed the Datafeed Toolbox, you can use Fixed-Income Toolbox functions to analyze data from Bloomberg and several other data servers.

Organization of the Document

Chapter	Description
"Mortgage-Backed Securities"	Describes the capabilities of this toolbox to support calculations involving the valuation of pools of fixed-rate mortgages. The effect of prepayment on pool prices and yields is extensively discussed.
"Debt Instruments"	Describes toolbox capabilities for handling various types of fixed-income securities, including short-term Treasury instruments, corporate bonds, and bonds with stepped coupons.
"Derivative Securities"	Describes some of the derivative processing capabilities of this toolbox and their use in such activities as portfolio hedging.

Chapter	Description
“Function Reference”	Provides alphabetic and categorized lists of functions that support computations involved with the above security types.
“Glossary”	Provides definitions of some commonly encountered financial terms and some terms specific to fixed-income instruments.

Expected Background

In designing the Fixed-Income Toolbox and this manual, we assume that your role is similar to one of these:

- Quantitative analyst
- Trader
- Student, professor, or other academic
- Financial engineer (with specific interest in mortgage-backed securities or fixed-income derivatives)

Product Requirements

To use the Fixed-Income Toolbox, you must install these additional products:

- MATLAB (Version 6.5 or later)
- Financial Toolbox (Version 2.3 or later)
- Optimization Toolbox
- Statistics Toolbox

Related Products

The MathWorks provides several products relevant to the tasks you can perform with the Fixed-Income Toolbox.

For more information about any of these products, see either

- The online documentation for that product if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site at `http://www.mathworks.com`; see the "products" section

Note The toolboxes and products listed below all include functions that extend the capabilities of MATLAB.

Product	Description
Datafeed Toolbox	Acquire real-time financial data from data service providers
Financial Toolbox	Model financial data and develop financial analysis algorithms
GARCH Toolbox	Analyze financial volatility using univariate GARCH models
MATLAB COM Builder	Create MATLAB based COM objects
MATLAB Excel Builder	Create MATLAB based add-ins for Excel
Optimization Toolbox	Solve standard and large-scale optimization problems
Statistics Toolbox	Apply statistical algorithms and probability models

Typographical Conventions

Item	Convention	Example
Example code	`Monospace` font	To assign the value 5 to `A`, enter `A = 5`
Function names, syntax, filenames, directory/folder names, and user input	`Monospace` font	The `cos` function finds the cosine of each array element. Syntax line example is `MLGetVar ML_var_name`
Buttons and keys	**Boldface** with book title caps	Press the **Enter** key.
Literal strings (in syntax descriptions in reference chapters)	**`Monospace bold`** for literals	`f = freqspace(n,'whole')`
Mathematical expressions	*Italics* for variables Standard text font for functions, operators, and constants	This vector represents the polynomial $p = x^2 + 2x + 3$.

MATLAB output	`Monospace` font	MATLAB responds with
		`A =`
		`5`

Menu and dialog box titles	**Boldface** with book title caps	Choose the **File Options** menu.
New terms and for emphasis	*Italics*	An *array* is an ordered collection of information.
Omitted input arguments	(...) ellipsis denotes all of the input/output arguments from preceding syntaxes.	`[c,ia,ib] = union(...)`
String variables (from a finite list)	*`Monospace italics`*	`sysc = d2c(sysd,'method')`

1

Mortgage-Backed Securities

What Are Mortgage-Backed Securities?

Mortgage-backed securities (MBS) 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 *passthrough certificate* (PC). Most passthrough 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 will receive timely payment of principal and interest from the PCs.

Using Fixed-Rate Mortgage Pool Functions

The Fixed-Income Toolbox supports calculations involved with generic fixed-rate mortgage pools and balloon mortgages. Passthrough 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 Public Securities 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/UP/default.shtml`).

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 passthrough functions, users 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`.)

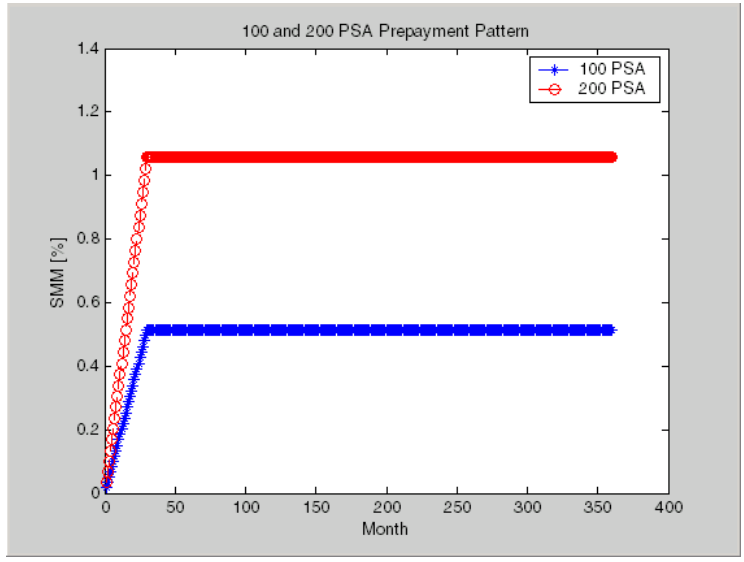
Generating Prepayment Vectors

You can generate PSA multiple prepayment vectors very quickly. To generate prepayment vectors of 100 and 200 PSA, type

```
PSASpeed = [100, 200];
[CPR, SMM] = psaspeed2rate(PSASpeed);
```

This function computes two prepayment values: constant prepayment rate (CPR) and single monthly mortality (SMM) rate. CPR is the percentage of outstanding principal prepaid in one year. SMM is the percentage of outstanding principal prepaid in one 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 book, observe the SMM (100 and 200 PSA) plots, spaced one month apart, instead.



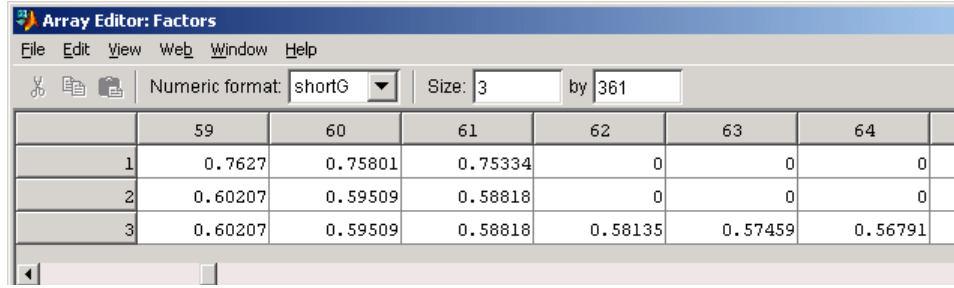
Prepayment assumptions form the basis upon which far more comprehensive MBS calculations are based. As an illustration observe the following example,

which demonstrates 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 array 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).



	59	60	61	62	63	64
1	0.7627	0.75801	0.75334	0	0	0
2	0.60207	0.59509	0.58818	0	0	0
3	0.60207	0.59509	0.58818	0.58135	0.57459	0.56791

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 passthrough 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 the descriptions of `mbsprice` and `mbsyield`.

Risk Measurement

The Fixed-Income Toolbox 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 in conjunction 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 represents 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, you should 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 needed 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 (OAS)” on page 1-10 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 needs to 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. You need to recompute the prepayment input because of this. 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 (OAS)

The option-adjusted spread 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 the Financial Toolbox functions `zbtprice` and `zbtyield`.

The example below demonstrates how to calculate an appropriate zero curve followed by computation of the pool’s OAS. This examples calculates the OAS of a 30-year fixed rate mortgage with approximately 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 =

  312.6026
  343.6172
  374.6668
```

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 Prepayments" on page 1-6. 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.

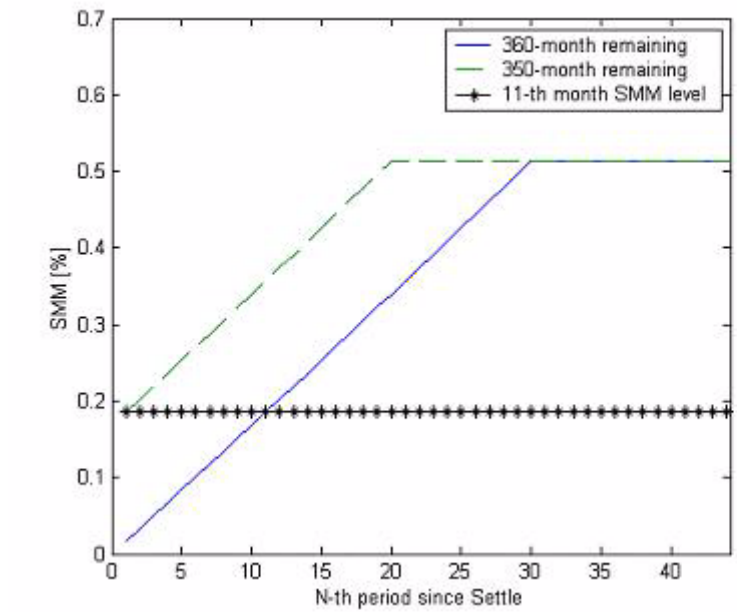
While this example uses the mortgage pool price (`mbsprice2oas`) to determine the OAS, you can also use yield to resolve it (`mbsyield2oas`). Also, there are reverse OAS functions that return prices and yields given OAS (`mbsoas2price` and `mbsoas2yield`).

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 need to 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, 4, [], ...
IssueDate)

NumCouponsRemaining =
323
```

The input `12` defines the monthly payment frequency, `4` defines the 30/360 (PSA) basis, and `IssueDate` defines aging and determination-of-holder date. Thus, you need to supply a 323-element vector to properly account for prepayment corresponding to each monthly payment.

Pools with Different Numbers of Coupons Remaining

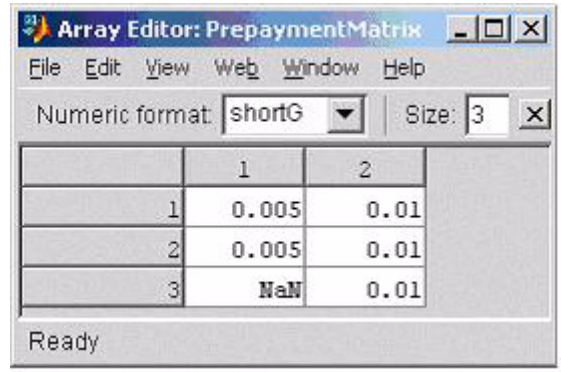
Suppose one pool has two remaining coupons, and the other has three. MATLAB expects the prepayment matrix to be in the following format:

```
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 two-month pool has a constant SMM of 0.5% and the three-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 most appropriate 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.

2

Debt Instruments

Treasury Bills Defined

Treasury bills are short-term (usually six months) securities 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. 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 interest accrual 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. These assumptions result in a slight increase in the actual discount applied to the notional.

Computing Treasury Bill Price and Yield

The Fixed-Income Toolbox provides a suite of functions for computing price and yield on Treasury bills. These functions are shown below.

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 one 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 example below 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 zero.

```
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 above data, 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.

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

BEYield =

    0.01639
    0.01700

MMYield =

    0.01617
    0.01677
```

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

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.

The Fixed Income Toolbox provides functions for valuing zero-coupon debt instruments. These functions supplement existing coupon bond functions such as `bndprice` and `bndyield` that are available in the Financial Toolbox.

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's yield is essentially 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 simply the principal multiplied by $e^{-r \cdot t}$. In reality, the rate quoted is very seldom 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 2-6
- "Pricing Corporate Bonds" on page 2-8

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

Cash Flows	Discount Factors	Discounted Cash Flows
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 facilitate extraction of present value from virtually any fixed-coupon instrument, up to any period in time.

Stepped-Coupon Bonds

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 two years and onwards to maturity. Assume that the issue and settlement dates are both March 15, 2003. The bond has a five-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 one less element than `CouponRates` because MATLAB 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	

CouponDates	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 the Financial Toolbox shows that the two functions operate identically.

Price and Yield of Stepped-Coupon Bonds

The toolbox provides two basic analytical functions to compute price and yield for stepped-coupon bonds. Using the above bond as an example, you can compute the price when the yield is known.

You can estimate the yield to maturity as a number-of-year weighted average of coupon rates. For this bond the estimated yield is

$$\frac{(2 \times 2) + (4 \times 3)}{5}$$

or 3.33%. While definitely not exact (due to nonlinear relation of price and yield), this estimate suggests close to par valuation and serves as a quick first check on the function.

```
Yield = 0.0333;

[Price, AccruedInterest] = stepcpnprice(Yield, Settle, ...
Maturity, ConvDates, CouponRates)
```

The price returned is 99.2237 (per $100 notional), and the accrued interest is zero, consistent with our earlier assertions.

To validate that there is consistency among the stepped-coupon functions, you can use the above price and see if indeed it implies a 3.33% yield by using `stepcpnyield`.

```
YTM = stepcpnyield(Price, Settle, Maturity, ConvDates, ...
CouponRates)

YTM =

    0.0333
```

Term Structure Calculations

So far we have avoided a more formal definition of "yield" and its application. 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 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.

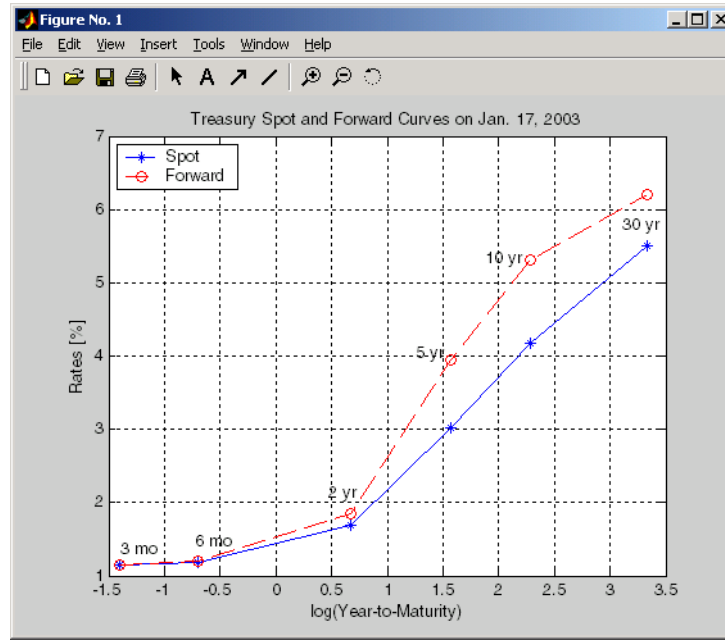
```
Settle = datenum('17-Jan-2003');
[ZeroRates, CurveDates] = zbtyield(Bonds, Yields, Settle)
```

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 9forward three-month rates, for example).



Computing Spreads

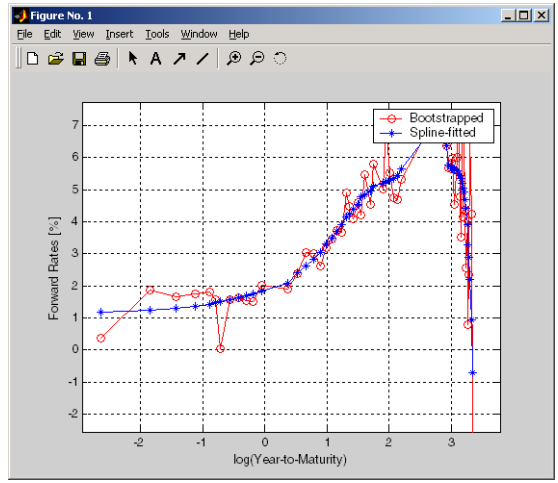
Calculating the spread between specific, fixed forward periods (such as the Treasury-Eurodollar spread) requires an extra step. Interpolate the zero rates (or zero prices, instead) for the corresponding maturities on the interval dates. Then use the interpolated zero rates to deduce the forward rates, 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.

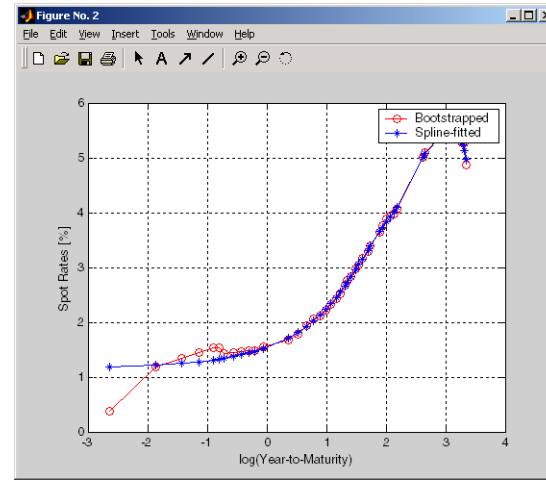
Noise in Curve Computations

When introducing more bonds in constructing curves, noise may become a factor and may need some “smoothing” (with splines, for example). This will help 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 certainly 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 the Financial Toolbox that also requires the use of the Spline Toolbox.

3

Derivative Securities

Pricing and Hedging

The Fixed-Income Toolbox contains functions that perform swap pricing and portfolio hedging.

Swap Pricing Assumptions

The Fixed-Income Toolbox 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 three months apart.
- All periods start on the third Wednesday of delivery months.
- All periods end on the same dates of delivery months, three 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 demonstrates 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 table below.

Note This example illustrates swap calculations in MATLAB. 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
9	2003	97.87
12	2003	97.575
3	2004	97.26

Eurodollar Data on Friday, October 11, 2002 (Continued)

Month	Year	Settle
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

Eurodollar Data on Friday, October 11, 2002 (Continued)

Month	Year	Settle
12	2009	93.865
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 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 two-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 =

    0.0179
    0.0170
    0.0177
```

```
0.0196
0.0222
0.0255
0.0285
0.0311
```

In the `FixedSpec` output, note that the swap rate actually goes forward from the third Wednesday of October 2002 (October 16, 2002), five 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, e.g., five years or more, this correction could prove to be substantial.

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 two-year swap rate, quite close to the reported swap rate for that date.

Analogously, the table below 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 the Financial Toolbox 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 employ 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

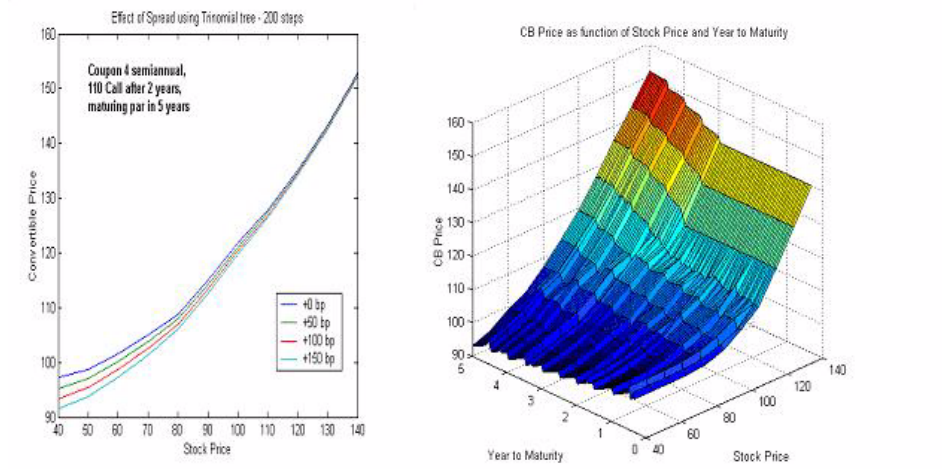
A convertible bond (CB) is a debt instrument that can be converted into a predetermined amount of a company's equity at certain times prior to the bond's maturity.

The Fixed-Income Toolbox 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, backwards 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 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 with 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 zero 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," T*he 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 two years at 110, and redeemable at par in five years), the toolbox gives results similar to 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 very 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.

Treasury Bond Futures

The Fixed-Income Toolbox also provides new functions that compute Treasury futures conversion factors and theoretical Treasury futures prices.

Theoretical Prices

This example shows how you can provide an input of eligible bonds and obtain its conversion factor to a 6% coupon rate (or against any other desired coupon rate). The example assumes no knowledge of the repo rate and instead uses the spot curve as the funding rate (`tfutbyprice` and `tfutbyyield`).

```
RefDate  = [datenum('1-Dec-2002');
            datenum('1-Mar-2003');
            datenum('1-Jun-2003');
            datenum('1-Sep-2003');
            datenum('1-Dec-2003');
            datenum('1-Sep-2003');
            datenum('1-Dec-2002');
            datenum('1-Jun-2003')];

Maturity = [datenum('15-Nov-2012');
            datenum('15-Aug-2012');
            datenum('15-Feb-2012');
            datenum('15-Feb-2011');
            datenum('15-Aug-2011');
            datenum('15-Aug-2010');
            datenum('15-Aug-2009');
            datenum('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)

CF =

    0.8539
    0.8858
    0.9259
```

```
0.9418
0.9403
0.9862
1.0000
1.0266
```

The results can be checked against the Chicago Board of Trade (`http://www.cbot.com`) 10-year futures contract table.

Coupon	Issue Date	Maturity Date	6% Conversion Factors					
			Dec-02	Mar-03	Jun-03	Sep-03	Dec-03	Mar-04
4.00	11/15/02	11/15/12	0.8539	0.8568	0.8595	0.8625	0.8653	0.8683
4 3/8	08/15/02	08/15/12	0.8836	0.8858	0.8883	0.8905	0.893	0.8954
4 7/8	02/15/02	02/15/12	0.9226	0.9242	0.9259	0.9275	0.9293	0.931
5	02/15/01	02/15/11	0.9372	0.9386	0.9403	0.9418	0.9435	0.9451
5	08/15/01	08/15/11	0.9342	0.9356	0.9372	0.9386	0.9403	0.9418
5 1/2	05/17/99	05/15/09	---	---	---	---	---	---
5 3/4	08/15/00	08/15/10	0.9851	0.9854	0.9859	0.9862	0.9867	---
6	08/16/99	08/15/09	1	---	---	---	---	---
6 1/2	02/15/00	02/15/10	1.0282	1.0273	1.0266	---	---	---

This computation can be incorporated into other functions that use conversion factors, such as routines to find the cheapest-to-deliver bonds. This is also equivalent to calculating the theoretical price of a particular set of bond futures prices.

A Treasury spot curve is necessary to discount the issue properly. MATLAB takes into account the actual/actual accrual basis and any intermittent coupons between the settlement and delivery dates. You can generate the spot curve using any set of Treasury bonds as long as the bonds cover the entire life of the futures in question.

```
% Computing the reference spot curve.
  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);

% Computing theoretical futures T-bonds price.
SpotCurve = [CurveDates, ZeroRates];
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];
CF = convfactor(RefDate, Maturity, CouponRate);
Price = [114.416; 113.171];
Interpolation = 1;

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

QtdFutPrice =

  113.8129
  112.4986
```

These quoted prices for December 2002 and March 2003 are comparable with futures prices of 113.93 and 112.68 traded that same hour at the Chicago Board of Trade. Without a live data feed, the data timings are asynchronous, but the results compare favorably for illustration purposes. These methods are well documented (such as in Hull 2000) and require some assumptions about the intended delivery date. Because of the short-term maturities on most bond futures, no convexity adjustment is usually needed.

When you know the repo rate, you can calculate theoretical prices with `tfutpricebyrepo` or `tfutyieldbyrepo`. Effectively, the known repo rate substitutes for the segment of the spot curve between the settlement and delivery dates.

Implied Repo

Alternatively, you can calculate the cheapest-to-deliver (CTD) bonds by comparing the repo rates implied by bond current and future prices. A higher-cost bond, obviously, is not a candidate for the CTD bond, and vice versa.

Within a low-yield environment, you would like to compare implied repos on two bonds on the extremes of a deliverable three-month contract for 10-year Treasury bonds. You expect that the CTD bond has a higher coupon and shorter maturity, and that this bond implies lower funding cost.

For example, two bonds on the extremes are a 6.5% coupon maturing February 15, 2010, and a 3.875% coupon maturing February 15, 2013. You would expect the 6.5% to be significantly cheaper to deliver as implied by its lower funding rate. The price of the 6.5% is 118.439, and price of the 3.875% is 99.601.

Quoted futures price for June 2003 is 113.9219 (113-295). Today is March 27, 2003, and you intend to deliver on June 27, 2003 (92 days repo). There are no coupons to reinvest during this period.

The code that provides this information to the toolbox `tfutimprepo` function is

```
ReinvestData = [0.025  3];
Price = [118.300;
          99.601];
QtdFutPrice = [113.9219;
               113.9219];
Settle = datenum('03/27/2003');
MatFut = [datenum('27-Jun-2003');
          datenum('27-Jun-2003')];
CF = [1.0266;
      0.8478];
CouponRate = [0.06500;
              0.03875];
Maturity = [datenum('15-Feb-2010');
            datenum('15-Feb-2013')];
```

```
ImpliedRepo = tfutimprepo(ReinvestData, Price, QtdFutPrice, ...
Settle, MatFut, CF, CouponRate, Maturity)

ImpliedRepo =

    0.0100
   -0.0795
```

This result confirms that the CTD bond is indeed the 6.5% February 2010 bond, as it has the higher implied repo (ignoring transaction cost) before arbitrage can occur. The implied repo in MATLAB is always returned on an actual/360 accrual basis.

Remember that this data is likely to be asynchronous and is useful for illustration purpose only.

Hedge Parameters

Treasury futures hedge parameters are frequently measured with DV01, the dollar value when there is a one-basis-point shift. This is easily calculated by computing the duration of the underlying bond for the contract (the CTD bond), dividing it by its conversion factor, and multiplying it by its cash price. Use `bnddurp` and `bnddury` from the Financial Toolbox to compute modified durations of fixed-coupon bonds. Again, the facility provided by the Treasury bond futures functions easily leverages such tasks and lets you focus on more qualitative assessment of your routines.

4

Function Reference

Functions — By Category

Note The functions in this toolbox use the MATLAB `double` data type almost exclusively. There is virtually no use of MATLAB structures, objects, or strings as input to these functions.

Cash Flows

`cfamounts`	Cash flow and time mapping for bond portfolio	SIA[1], PSA[2], ISDA[3], European, Japanese compliant

[1] Securities Industry Association

[2] Public Securities Association

[3] International Swap Dealers Association

Certificates of Deposit

`cdai`	Accrued interest on a certificate of deposit	SIA, PSA, ISDA, European, Japanese compliant
`cdprice`	Price a certificate of deposit	SIA, PSA, ISDA, European, Japanese compliant
`cdyield`	Yield on a certificate of deposit	SIA, PSA, ISDA, European, Japanese compliant

Convertible Bonds

`cbprice`	Price a convertible bond

Derivative Securities

`bkcall`	Price a call option using Black's model
`bkcaplet`	Price an interest rate caplet using Black's model
`bkfloorlet`	Price an interest rate floorlet using Black's model
`bkput`	Price a put option using Black's model
`liborduration`	Duration of a LIBOR-based interest rate swap
`liborfloat2fixed`	Compute par fixed-rate of swap given three-month LIBOR data
`liborprice`	Price a swap given the swap rate

Mortgage-Backed Securities

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

Option Adjusted Spread Computations

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

Stepped Coupon Bonds

`stepcpncfamounts`	Cash flow amounts and times for bonds with stepped coupons	SIA, PSA, ISDA, European, Japanese compliant
`stepcpnprice`	Price a bond with stepped coupons	SIA, PSA, ISDA, European, Japanese compliant
`stepcpnyield`	Yield to maturity of a bond with stepped coupons	SIA, PSA, ISDA, European, Japanese compliant

Treasury Bills

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

Treasury Bond Futures

`convfactor`	Treasury bond conversion factors	
`tfutbyprice`	Treasury bond future prices	
`tfutbyyield`	Treasury bond future yields	
`tfutpricebyrepo`	Theoretical futures bond price	
`tfutyieldbyrepo`	Theoretical futures bond yield	
`tfutimprepo`	Implied simple annual repurchase rate to prevent arbitrage	

Zero Coupon Instruments

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

Functions — Alphabetical List

Purpose

Price a call option using Black's model

Compliance

SIA, PSA, ISDA, European, Japanese

Syntax

```
CallPrices = bkcall(Strike, FwdRateData, Sigma, BondInfo, SettleOpt,
MatOpt, Period, Basis, EndMonthRule, Interpolation, CallType)
```

Arguments

`Strike`	Strike price for every $100 face of bond.
`FwdRateData`	Two-column (optionally three-column) matrix containing forward rate information. First column: Date when forward rate starts to apply. Second column: Annual rate applicable between time T_i and T_{i+1}. (The last forward rate can be any value. The only needed data is its date) Third column: (optional): Compounding method for rates. It has a minor effect, as the model returns parameters based on continuous compounding suitable for no-arbitrage models. Values are 1 (annual), 2 (semiannual, default), 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).
`Sigma`	Annualized volatility (365-day year).
`BondInfo`	Information about underlying bond in the form: `[CleanPrice CouponRate Maturity]`
`SettleOpt`	Settlement date of option.
`MatOpt`	Maturity date of option.
`Period`	(Optional) Number of coupons per year for the underlying bond. Default = 2 (semiannual). Supported values are 0, 1, 2, 4,6, and 12. 0 = zero coupon receiving principal at maturity.

`Basis`	(Optional) Day-count basis of the bond. A vector of integers. 0 = actual/actual (default), 1 = 30/360, 2 = actual/360 (default), 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).
`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.
`Interpolation`	(Optional) Interpolation method. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See `interp1` for more information.
`CallType`	(Optional) Scalar or `NOPT`-by-`1` vector of call types. 0 = cash bond price (default). 1 = clean bond price.

All input arguments are scalars or number of options (`NOPT`) by `1` vectors.

Description

`CallPrices = bkcall(Strike, FwdRateData, Sigma, BondInfo, SettleOpt, MatOpt, Period, Basis, EndMonthRule, Interpolation, CallType)` computes the price of a call for every $100 face value of a bond.

Examples

Compute the prices of a pair of bond call option using the supplied data.

```
Strike       = 100;
FwdRateData = [datenum('16-Oct-2002') 0.09    -1;
               datenum('16-Jan-2003') 0.0975  -1;
               datenum('16-Jul-2003') 0.145   -1;
               datenum('16-Aug-2003') 0.145   -1];

Sigma = 0.09;
BondInfo = [93.5  0.1  datenum('16-Jul-20012')];
SettleOpt = datenum('16-Oct-2002');
MatOpt = datenum('16-Aug-2003');
Period = 2;
Basis = 0;
```

```
EndMonthRule = 1;
Interpolation = 1;
CallType = [0;1];

CallPrices = bkcall(Strike, FwdRateData, Sigma, BondInfo,...
SettleOpt, MatOpt, Period, Basis, EndMonthRule, Interpolation,...
CallType)

CallPrices =

    0.9484

    0.7950
```

See Also

bkput

bkcaplet

Purpose

Price an 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, 2 = actual/360, 3 = actual/365.
`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

bkfloorlet

Purpose

Price an 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, 2 = actual/360, 3 = actual/365.
`FwdRates`	Scalar or `NFLR`-by-1 vector containing forward rates in decimal. `FwdRates` accrue on the same basis as `FloorRates`.
`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 a put option using Black's model

Compliance

SIA, PSA, ISDA, European, Japanese

Syntax

```
PutPrices = bkput(Strike, FwdRateData, Sigma, BondInfo, SettleOpt,
MatOpt, Period, Basis, EndMonthRule, Interpolation, PutType)
```

Arguments

`Strike`	A number of options (`NOPT`) by 1 vector of strike prices for every $100 face value of bonds.
`FwdRateData`	Two-column (optionally three-column) matrix containing forward rate information. First column: Date when forward rate starts to apply. Second column: Annual rate applicable between time T_i and T_{i+1}. (The last forward rate can be any value. The only needed data is its date.) Third column: (optional): Compounding method for rates. It has a minor effect, as the model returns parameters based on continuous compounding suitable for no-arbitrage models. Values are 1 (annual), 2 (semiannual, default), 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).
`Sigma`	Annualized volatility (365-day year).
`BondInfo`	Information about underlying bond in the form: `[CleanPrice CouponRate Maturity]`
`SettleOpt`	Scalar or `NOPT`-by-1 vector of settlement dates of options.
`MatOpt`	Scalar or `NOPT`-by-1 vector of maturity dates of options.
`Period`	(Optional) Number of coupons per year for the underlying bond. Default = 2 (semiannual). Supported values are 0, 1, 2, 4, 6, and 12. 0 = zero coupon receiving principal at maturity.

`Basis`	(Optional) Day-count basis of the bond. A vector of integers. 0 = actual/actual (default), 1 = 30/360, 2 = actual/360 (default), 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).
`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.
`Interpolation`	(Optional) Interpolation method. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See `interp1` for more information.
`PutType`	(Optional) Scalar or `NOPT`-by-1 vector of put types. 0 = cash bond price (default). 1 = clean bond price.

All input arguments are scalars or number of options (`NOPT`) by 1 vectors.

Description

`PutPrices = bkput(Strike, FwdRateData, Sigma, BondInfo, SettleOpt, MatOpt, Period, Basis, EndMonthRule, Interpolation, PutType)` produces a `NOPT`-by-1 vector of put prices for $100 notional value, based upon Black's model.

Examples

Compute the prices of two bond put options using the supplied data.

```
Strike        = 100;
FwdRateData   = [datenum('16-Oct-2002') 0.09    -1;
                 datenum('16-Jan-2003') 0.0975  -1;
                 datenum('16-Jul-2003') 0.145   -1;
                 datenum('16-Aug-2003') 0.145   -1];
Sigma         = 0.09;
BondInfo      = [93.5  0.1  datenum('16-Jul-20012')];
SettleOpt     = datenum('16-Oct-2002');
MatOpt        = datenum('16-Aug-2003');
Period        = 2;
Basis         = 0;
```

```
EndMonthRule  = 1;
Interpolation = 1;
PutType       = [0;1];

PutPrices = bkput(Strike, FwdRateData, Sigma, BondInfo, ...
SettleOpt, MatOpt, Period, Basis, EndMonthRule, ...
Interpolation, PutType)

PutPrices =

          7.6748
          8.4092
```

See Also

bkcall

Purpose

Price a 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) Number of coupons per year (1 to 4).
`Basis`	(Optional) Scalar value for day-count basis of the instrument. 0 = actual/actual, 1 = 30/360, 2 = actual/360 (default), 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).

`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-Rubenstein binomial tree or, optionally, a trinomial tree.

`CBMatrix` is a matrix of convertible bond prices.

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

`EqtyMatrix` is a matrix of the equity component of the convertible bond.

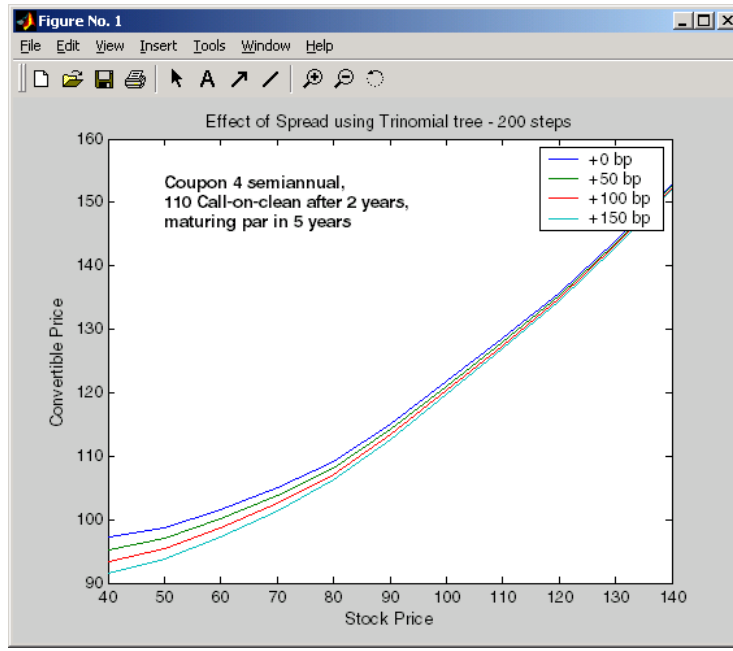
`EqtyMatrix` is a tree of the equity portion of the convertible bond prices.

Examples

Perform a spread effect analysis of a 4%-coupon convertible bond callable at 110 at end of second year, maturing at par in five 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 convetible 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({'+0 bp'; '+50 bp'; '+100 bp'; '+150 bp'});
title ('Effect of Spread using Trinomial tree - 200 steps')
xlabel('Stock Price');
ylabel('Convertible Price');
text(50, 150, ['Coupon 4 semiannual,', sprintf('\n'), ...
     '110 Call-on-clean after 2 years,' sprintf('\n'), ...
     'maturing par in 5 years'],'fontweight','Bold')
```



Purpose

Accrued interest on certificate of deposit (CD)

Compliance

SIA, PSA, ISDA, European, Japanese

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. 2 = actual/360 (default), 3 = actual/365. These are the only bases allowable for certificates of deposit.

Each required input must be a number of 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 functions is best used for short term maturities (less than one 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

Purpose

Price a certificate of deposit (CD)

Compliance

SIA, PSA, ISDA, European, Japanese

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. 2 = actual/360 (default), 3 = actual/365. These are the only bases allowable for certificates of deposit.

Each required input must be a number of 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 one year). The default simple interest calculation is the actual/360 convention (SIA). Other bases are available.

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 =

    1.2917
```

See Also

bndprice, cdai, cdyield, stepcpnprice, tbillprice

Purpose

Yield on certificate of deposit (CD)

Compliance

SIA, PSA, ISDA, European, Japanese

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. 2 = actual/360 (default), 3 = actual/365. These are the only bases allowable for certificates of deposit.

Each required input must be a number of 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 functions is best used for short term maturities (less than one year). The default simple interest calculation is the actual/360 convention (SIA).

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

Cash flow and time mapping for bond portfolio

Compliance

SIA, PSA, ISDA, European, Japanese

Syntax

```
[CFlowAmounts, CFlowDates, TFactors, CFlowFlags] =
  cfamounts(CouponRate, Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate, Face)
```

Arguments

`CouponRate`	Decimal number indicating the annual percentage rate used to determine the coupons payable on a bond.
`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) 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 bond. A vector of integers. A scalar. 0 = actual/actual (default), 1 = 30/360, 2 = actual/360 (SIA compliant), 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).
`EndMonthRule`	(Optional) End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	(Optional) Date when a bond was issued.

`FirstCouponDate`	(Optional) Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.
`LastCouponDate`	(Optional) Last coupon date of a bond prior to 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.
`StartDate`	(Future implementation; optional) Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.
`Face`	(Optional) Face or par value. Default = 100.

Required arguments must be number of bonds (`NUMBONDS`) by `1` or `1`-by-`NUMBONDS` conforming vectors or scalars. Optional arguments must be either `NUMBONDS`-by-`1` or `1`-by-`NUMBONDS` conforming vectors, scalars, or empty matrices.

Description

`[CFlowAmounts, CFlowDates, TFactors, CFlowFlags] = cfamounts(CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face)` returns matrices of cash flow amounts, cash flow dates, time factors, and cash flow flags for a portfolio of `NUMBONDS` fixed income securities. The elements contained in the cash flow matrix, time factor matrix, and cash flow flag matrix correspond to the cash flow dates for each security. The first element of each row in the cash flow matrix is the accrued interest payable on each bond. This is zero in the case of all zero coupon bonds. This function determines all cash flows and time mappings for a bond whether or not the coupon structure contains odd first or last periods. All output matrices are padded with `NaN`s as necessary to ensure that all rows have the same number of elements.

`CFlowAmounts` is the cash flow matrix of a portfolio of bonds. Each row represents the cash flow vector of a single bond. Each element in a column represents a specific cash flow for that bond.

`CFlowDates` is the cash flow date matrix of a portfolio of bonds. Each row represents a single bond in the portfolio. Each element in a column represents a cash flow date of that bond.

`TFactors` is the matrix of time factors for a portfolio of bonds. Each row corresponds to the vector of time factors for each bond. Each element in a column corresponds to the specific time factor associated with each cash flow of a bond. Time factors are useful in determining the present value of a stream of cash flows. The term "time factor" refers to the exponent *TF* in the discounting equation

$$PV = CF/(1 + z/2)^{TF}$$

where:

PV = present value of a cash flow

CF = the cash flow amount

z = the risk-adjusted annualized rate or yield corresponding to given cash flow. The yield is quoted on a semiannual basis.

TF = time factor for a given cash flow. Time is measured in semiannual periods from the settlement date to the cash flow date.

`CFlowFlags` is the matrix of cash flow flags for a portfolio of bonds. Each row corresponds to the vector of cash flow flags for each bond. Each element in a column corresponds to the specific flag associated with each cash flow of a bond. Flags identify the type of each cash flow (e.g., nominal coupon cash flow, front or end partial or "stub" coupon, maturity cash flow). Possible values are shown in the table.

Flag	Cash Flow Type
0	Accrued interest due on a bond at settlement.
1	Initial cash flow amount smaller than normal due to “stub” coupon period. A stub period is created when the time from issue date to first coupon is shorter than normal.
2	Larger than normal initial cash flow amount because first coupon period is longer than normal.
3	Nominal coupon cash flow amount.
4	Normal maturity cash flow amount (face value plus the nominal coupon amount).
5	End “stub” coupon amount (last coupon period abnormally short and actual maturity cash flow is smaller than normal).
6	Larger than normal maturity cash flow because last coupon period longer than normal.
7	Maturity cash flow on a coupon bond when the bond has less than one coupon period to maturity.
8	Smaller than normal maturity cash flow when bond has less than one coupon period to maturity.
9	Larger than normal maturity cash flow when bond has less than one coupon period to maturity.
10	Maturity cash flow on a zero coupon bond.

Examples

Consider a portfolio containing a corporate bond paying interest quarterly and a treasury bond paying interest semiannually. Compute the cash flow structure and the time factors for each bond.

```
Settle = '01-Nov-1993';
Maturity = ['15-Dec-1994';'15-Jun-1995'];
CouponRate= [0.06; 0.05];
Period = [4;2];
```

```
Basis = [1;0];
[CFlowAmounts, CFlowDates, TFactors, CFlowFlags] = ...
cfamounts(CouponRate,Settle, Maturity, Period, Basis)

CFlowAmounts =

  -0.7667    1.5000    1.5000    1.5000    1.5000  101.5000
  -1.8989    2.5000    2.5000    2.5000  102.5000       NaN

CFlowDates =

728234     728278     728368     728460     728552     728643
728234     728278     728460     728643     728825        NaN

TFactors =

0    0.2404    0.7403    1.2404    1.7403    2.2404
0    0.2404    1.2404    2.2404    3.2404       NaN

CFlowFlags =

0     3     3     3     3     4
0     3     3     3     4   NaN
```

See Also

accrfrac, cfdates, cpncount, cpndaten, cpndatenq, cpndatep, cpndatepq, cpndaysn, cpndaysp, cpnpersz

convfactor

Purpose

Treasury bond conversion factors

Syntax

```
ConvFactor = convfactor(RefDate, Maturity, CouponRate, RefYield)
```

Arguments

`RefDate`	Reference dates, for which conversion factor is computed (usually the first day of delivery months).
`Maturity`	Maturity date of underlying bond.
`CouponRate`	Annual coupon rate of underlying bond in decimal.
`RefYield`	(Optional) Reference semiannual yield. Default = 0.06 (6%).

Description

`ConvFactor = convfactor(RefDate, Maturity, CouponRate, RefCpn)` computes conversion factors based upon a reference 6% semiannual yield.

Note You can verify the output of this function by comparing the output against the quotations provided by the Chicago Board of Trade (`http://www.cbot.com`).

Examples

```
RefDate  = [datenum('1-Dec-2002');
            datenum('1-Mar-2003');
            datenum('1-Jun-2003');
            datenum('1-Sep-2003');
            datenum('1-Dec-2003');
            datenum('1-Sep-2003');
            datenum('1-Dec-2002');
            datenum('1-Jun-2003')];

Maturity = [datenum('15-Nov-2012');
            datenum('15-Aug-2012');
            datenum('15-Feb-2012');
            datenum('15-Feb-2011');
            datenum('15-Aug-2011');
            datenum('15-Aug-2010');
            datenum('15-Aug-2009');
            datenum('15-Feb-2010')];
```

```
CouponRate = [0.04; 0.04375; 0.04875; 0.05;
              0.05; 0.0575;  0.06;    0.065];

ConvFactor = convfactor(RefDate, Maturity, CouponRate)

ConvFactor =

    0.8539
    0.8858
    0.9259
    0.9418
    0.9403
    0.9862
    1.0000
    1.0266
```

See Also

tfutbyprice, tfutbyyield

liborduration

Purpose

Duration of a LIBOR-based interest rate swap

Syntax

```
[PayFixDuration GetFixDuration] = liborduration(SwapFixRate, Tenor,
   Settle)
```

Arguments

`SwapFixRate`	Swap fixed rate in decimal
`Tenor`	Life of the swap in years. Fractional numbers are rounded upward.
`Settle`	Settlement date.

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 three-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 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 $dr = [\theta(t) - ar]dt + Sdz$ Default = `[0.05 0.015]`.
`InArrears`	(Optional) Default = 0 (off). Set to 1 for on. If on, the routine does an automatic an 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 (default), 2 = actual/360, 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 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 three months apart). The first element is the third Wednesday immmediately 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, Settle, Tenor)

FixedSpec =

      Coupon: 0.0218
      Settle: '16-Oct-2002'
    Maturity: '21-Oct-2004'
      Period: 4
       Basis: 2

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 a swap given the 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 $dr = [\theta(t) - ar]dt + Sdz$ 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, 2 = actual/360 (default), 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 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 =

  3.2613e-014
```

See Also

liborduration, liborfloat2fixed

mbscfamounts

Purpose

Cash flow and time mapping for mortgage pool

Compliance

PSA

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. Set `PrepaySpeed` to `[]` if you input a customized prepayment matrix.
`PrepayMatrix`	(Optional) Used only when `PrepayModel` and `PrepaySpeed` are 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 `NaN`s wherever elements are missing.

References

PSA Uniform Practices, SF-4

Purpose

Convexity of mortgage pool given price

Syntax

```
Convexity = mbsconvp(Price, Settle, Maturity, IssueDate, GrossRate,
   CouponRate, Delay, PrepaySpeed, PrepayMatrix)
```

Compliance

PSA

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 `PrepayModel` and `PrepaySpeed` are unspecified.) Customized prepayment vector. A `NaN`-padded matrix of size `max(TermRemaining)`-by-`NMBS`. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by 1 vectors.

Description

`Convexity = mbsconvp(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)` computes mortgage-backed security convexity, given time information, price at settlement, and optionally, a prepayment model.

Note If you specify the PSA or FHA model, it will be seasoned with how long the debt has been outstanding (the loan's age).

Examples

Given a mortgage-backed security with the following characteristics, compute the convexity of the security.

```
Price      = 101;
Settle     = '15-Apr-2002';
Maturity   = '1 Jan 2030';
IssueDate  = '1-Jan-2000';
GrossRate  = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;

Convexity = mbsconvp(Price, Settle, Maturity, IssueDate,...
GrossRate, CouponRate, Delay, Speed)

Convexity =

   71.6299
```

See Also

`mbsconvy`, `mbsdurp`, `mbsdury`

References

PSA Uniform Practices, SF-49

Purpose

Convexity of mortgage pool given yield

Syntax

```
Convexity = mbsconvy(Yield, Settle, Maturity, IssueDate, GrossRate,
   CouponRate, Delay, PrepaySpeed, PrepayMatrix)
```

Compliance

PSA

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 `PrepayModel` and `PrepaySpeed` are unspecified.) Customized prepayment vector. A `NaN`-padded matrix of size `max(TermRemaining)`-by-`NMBS`. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by 1 vectors.

Description

`Convexity = mbsconvy(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)` computes mortgage-backed security convexity, given time information, semiannual mortgage yield, and optionally, a prepayment model.

Note If you specify the PSA or FHA model, it will be seasoned with how long the debt has been outstanding (the loan's age).

Examples

Given a mortgage-backed security with the following characteristics, compute the convexity of the security.

```
Yield      = 0.07125;
Settle     = '15-Apr-2002';
Maturity   = '1 Jan 2030';
IssueDate  = '1-Jan-2000';
GrossRate  = 0.08125;
Speed      = 100;
CouponRate = 0.075;
Delay = 14;

Convexity = mbsconvy(Yield, Settle, Maturity, IssueDate, ...
GrossRate, CouponRate, Delay, Speed)

Convexity =

  73.5509
```

See Also

`mbsconvp`, `mbsdurp`, `mbsdury`

References

PSA Uniform Practices, SF-49

Purpose

Duration of mortgage pool given price

Syntax

```
[YearDuration, ModDuration] = mbsdurp(Price, Settle, Maturity,
   IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed,
   PrepayMatrix)
```

Compliance

PSA

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 `PrepayModel` and `PrepaySpeed` are 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
```

See Also

`mbsconvp`, `mbsconvy`, `mbsdury`

References

PSA Uniform Practices, SF-49

Purpose

Duration of mortgage pool given yield

Syntax

```
[YearDuration, ModDuration] = mbsdury(Yield, Settle, Maturity,
   IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed,
   PrepayMatrix)
```

Compliance

PSA

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 `PrepayModel` and `PrepaySpeed` are 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.

mbsdury

Description

`[YearDuration, ModDuration] = mbsdurvy(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepayModel, 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
```

See Also

`mbsconvp`, `mbsconvy`, `mbsdurp`

References

PSA Uniform Practices, SF-49

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 passthroughs are not the same, MATLAB pads the shorter ones with `NaN`.

`Balance` lists the end-of-month balances over the life of the passthrough.

`Interest` lists all end-of-month interest payments over the life of the passthrough.

`Payment` lists all end-of-month payments over the life of the passthrough.

`Principal` lists all scheduled end-of-month principal payments over the life of the passthrough.

All outputs are `Term`-by-`1` vectors.

Examples

Given mortgage pools with the following characteristics, compute an amortization schedule.

```
OriginalBalance = 400000000;
CouponRate = 0.08125;
Term = [357; 355]; % Three and five month old mortgage pools.

[Balance, Interest, Payment, Principal] = ...
mbsnoprepay(OriginalBalance, CouponRate, Term);
```

mbsoas2price

Purpose

Price given an option-adjusted spread

Syntax

```
Price = mbsoas2price(ZeroMatrix, OAS, Settle, Maturity, IssueDate,
   GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed,
   PrepayMatrix)
```

Arguments

`ZeroMatrix`	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 (e.g., 0.075). Column 3: Compounding of the rates in the Column 1. (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 `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(ZeroMatrix, OAS, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)` computes the clean price of a passthrough 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 zero rates.

```
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, not "selected" agency data, an ad-hoc method of altering the yield for benchmark purposes has been selected.

```
Yields = Yields + 0.025 * (1./[1:6]');
SpotCompounding = 2*ones(size(Yields));
```

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 create a zero curve.

```
[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);
ZeroMatrix = [CurveDatesP, ZeroRatesP, SpotCompounding];

OAS       = [26.0502; 28.6348; 31.2222];
Settle    = datenum('20-Aug-2002');
Maturity  = datenum('02-Jan-2030');
IssueDate = datenum('02-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Interpolation = 1;
PrepaySpeed = [0 50 100];

Price = mbsoas2price(ZeroMatrix, 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 an option-adjusted spread

Syntax

```
[MYield, BEMBSYield] = mbsoas2yield(ZeroMatrix, OAS, Settle,
   Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation,
   PrepaySpeed, PrepayMatrix)
```

Arguments

`ZeroMatrix`	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 (e.g., 0.075). Column 3: Compounding of the rates in the Column 1. (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 `NaN`s. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by 1 vectors.

Description

`[MYield, BEMBSYield] = mbsoas2yield(ZeroMatrix, OAS, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)` computes the mortgage and bond-equivalent yields of a passthrough security.

`MYield` is the yield to maturity of the mortgage-backed security (the mortgage yield). This yield is compounded monthly (12 times per year).

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).

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 zero rates.

```
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, not "selected" agency data, an ad-hoc method of altering the yield for benchmark purposes has been selected.

```
Yields = Yields + 0.025 * (1./[1:6]');
SpotCompounding = 2*ones(size(Yields));
```

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);

[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);
ZeroMatrix = [CurveDatesP, ZeroRatesP, SpotCompounding];

OAS       = [312.6026; 343.6172; 374.6668];
Settle    = datenum('20-Aug-2002');
Maturity  = datenum('02-Jan-2030');
IssueDate = datenum('02-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Interpolation = 1;
PrepaySpeed = [0 50 100];

[MYield BEMBSYield] = mbsoas2yield(ZeroMatrix, 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] = passthrough(OriginalBalance, GrossRate, OriginalTerm, TermRemaining, PrepaySpeed, PrepayMatrix)` computes the cash flow of principal, interest, and prepayment of passthrough securities.

All outputs are `TermRemaining`-by-`1` vectors of end-of-month values.

`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 upon 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 three-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)
```

Compliance

PSA

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 `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, 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
```

See Also

mbsyield

References

PSA Uniform Practices, SF-49

Purpose

Option-adjusted spread given price

Syntax

```
OAS = mbsprice2oas(ZeroMatrix, Price, Settle, Maturity, IssueDate,
   GrossRate, CouponRate, Delay, Interpolation  PrepaySpeed,
   PrepayMatrix)
```

Arguments

`ZeroMatrix`	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 (e.g., 0.075). Column 3: Compounding of the rates in the Column 1. 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`.
`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 `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

`OAS = mbsprice2oas(ZeroMatrix, Price, 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 with about a 28-year weighted average maturity remaining, given assumptions of 0, 50, and 100 PSA prepayments.

Create zero 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];

Yields = [0.0162;
          0.0163;
          0.0211;
          0.0328;
          0.0420;
          0.0501];
```

Since the above is Treasury data, and not "selected" agency data, an ad-hoc method of altering the yield has been chosen for demonstration purposes.

```
Yields = Yields + 0.025 * (1./[1:6]');
SpotCompounding = 2*ones(size(Yields));
```

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);

[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);

ZeroMatrix = [CurveDatesP, ZeroRatesP, SpotCompounding];

Price = 95;
Settle     = '20-Aug-2002';
Maturity   = '2-Jan-2030';
IssueDate  = '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 =

  312.6026
  343.6172
  374.6668
```

See Also mbsoas2price, mbsoas2yield, mbsyield2oas

Purpose

Implied PSA prepayment speeds given price

Compliance

PSA

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 `NaN`s. 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 passthrough to carry the same price.

`ImpSpdOnDur` calculates the equivalent PSA benchmark prepayment speed for the passthrough to carry the same modified duration.

`ImpSpdOnCnv` calculates the equivalent PSA benchmark prepayment speed for the passthrough 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 = 0.02*ones(360,1);
CouponRate = 0.075;
Delay = 14;

[ImpSpdOnPrc, ImpSpdOnDur, ImpSpdOnCnv] = ...
mbsprice2speed(Price,Settle, Maturity, IssueDate, ...
GrossRate, PrepayMatrix, CouponRate, Delay)

ImpSpdOnPrc =

  530.2020

ImpSpdOnDur =

  527.9529

ImpSpdOnCnv =

  447.7260
```

See Also

`mbsprice`, `mbsyield2speed`

References *PSA Uniform Practices*, SF-49

mbswal

Purpose

Weighted average life of a 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 `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

`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 passthrough 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 =

   10.6070
```

See Also

`mbspassthrough`

References

PSA Uniform Practices, SF-49

mbsyield

Purpose

Mortgage-backed security yield given price

Syntax

```
[MYield, BEMBSYield] = mbsyield(Price, Settle, Maturity, IssueDate,
   GrossRate, CouponRate, Delay,  PrepaySpeed, PrepayMatrix)
```

Compliance

PSA

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 `NaN`s. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

All inputs (except `PrepayMatrix`) are number of mortgage-backed securities (`NMBS`) by `1` vectors.

Description

`[MYield, BEMBSYield] = 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
```

See Also

mbsprice

References

PSA Uniform Practices, SF-49

Purpose

Option-adjusted spread given yield

Syntax

```
OAS = mbsyield2oas(ZeroMatrix, Yield, Settle, Maturity, IssueDate,
   GrossRate, CouponRate, Delay, Interpolation  PrepaySpeed,
   PrepayMatrix)
```

Arguments

`ZeroMatrix`	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 (e.g., 0.075) Column 3: Compounding of the rates in the Column 1. 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 `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

`OAS = mbsyield2oas(ZeroMatrix, 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 with about 28-year weighted average maturity left, given assumptions of 0, 50, and 100 PSA prepayments.

Create zero 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];

Yields = [0.0162;
          0.0163;
          0.0211;
          0.0328;
          0.0420;
          0.0501];
```

Since the above is Treasury data and not "selected" agency data, an ad-hoc method of altering the yield has been selected for demonstration purposes.

```
Yields = Yields + 0.025 * (1./[1:6]');
SpotCompounding = 2*ones(size(Yields));
```

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);

[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);

ZeroMatrix = [CurveDatesP, ZeroRatesP, SpotCompounding];

Price = 95;
Settle    = '20-Aug-2002';
Maturity  = '2-Jan-2030';
IssueDate = '2-Jan-2000';
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Interpolation = 1;
PrepaySpeed = [0; 50; 100];

[mbsyld, beyld] = mbsyield(Price, Settle, ...
Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed);

OAS = mbsyield2oas(ZeroMatrix, mbsyld, Settle, ...
Maturity, IssueDate, GrossRate, CouponRate, Delay, ...
Interpolation, PrepaySpeed)
```

```
OAS =

  312.6026
  343.6172
  374.6668
```

See Also

mbsoas2price, mbsoas2yield, mbsprice2oas

Purpose

Implied PSA prepayment speeds given yield

Compliance

PSA

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 `NaN`s. 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 passthrough to carry the same price.

ImpSpdOnDur calculates the equivalent PSA benchmark prepayment speed for the passthrough to carry the same modified duration.

ImpSpdOnCnv calculates the equivalent PSA benchmark prepayment speed for the passthrough 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 = 0.02*ones(360,1);
CouponRate = 0.075;
Delay = 14;

[ImpSpdOnYld, ImpSpdOnDur, ImpSpdOnCnv] = ...
mbsyield2speed(Yield, Settle, Maturity, IssueDate, GrossRate, ...
PrepayMatrix, CouponRate, Delay)

ImpSpdOnYld =

  527.0321

ImpSpdOnDur =

  521.5559

ImpSpdOnCnv =

  443.4709
```

See Also

mbsyield, mbsprice2speed

References *PSA Uniform Practices*, SF-49

psaspeed2default

Purpose

Benchmark default

Syntax

```
[ADRPSA, MDRPSA] = psaspeed2default(DefaultSpeed)
```

Compliance

PSA

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`

Purpose

Single monthly mortality rate given PSA speed

Syntax

```
[CPRPSA, SMMPSA]= psaspeed2rate(PSASpeed)
```

Compliance

PSA

Arguments

`PSASpeed`	Any value > 0 representing the annual speed relative to the benchmark. PSA benchmark = 100.

Description

`[CPRPSA, SMMPSA]= psaspeed2rate(PSASpeed)` calculates the single monthly mortality rate given PSA speed.

`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].

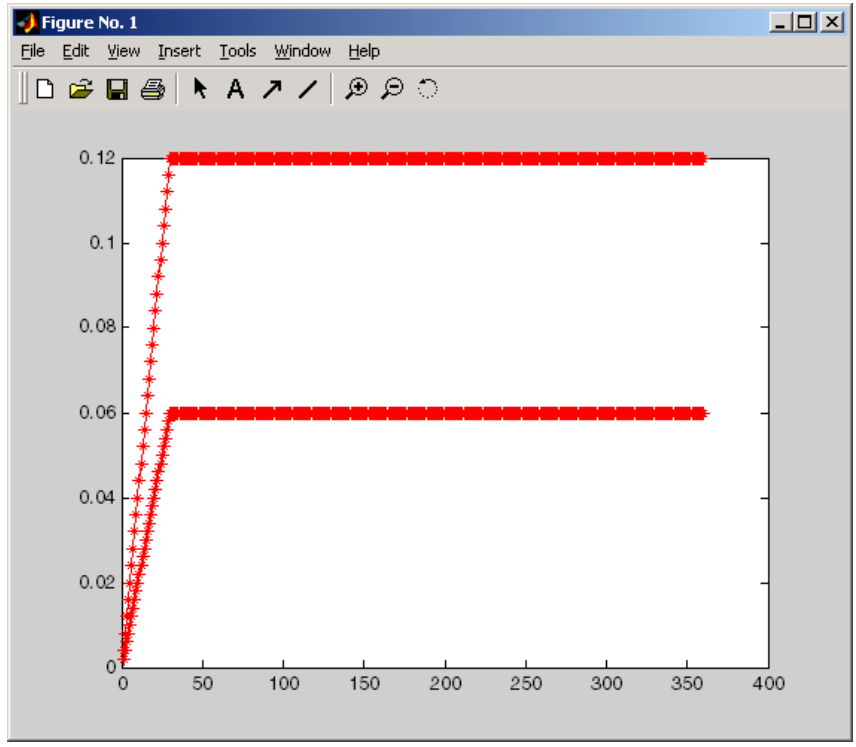
Examples

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`

Purpose

Cash flow amounts and times for bonds with stepped coupons

Compliance

SIA, PSA, ISDA, European, Japanese

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 containing conversion dates after `Settle`.
`CouponRates`	Matrix containing coupon rates for each bond in the portfolio in decimal form. First column of this matrix contains rates applicable between `Settle` and dates in the first column of `ConvDates`.
`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, 2 = actual/360, 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).
`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.

ConvDates and CouponRates must be matrices of size NUMBONDS by maximum number of distinct coupons. Fill unspecified entries with NaN.

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 zero.

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)
```

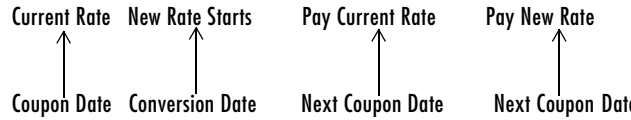
Time factors are in units of semiannual coupon periods.

Note For bonds with fixed coupons, use cfamounts. MATLAB generates an error if you use a fixed-coupon bond with stepcpncfamounts.

Examples

This example generates stepped cash flows for three different bonds, all paying interest semiannually. Their life span is about 18-19 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%
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

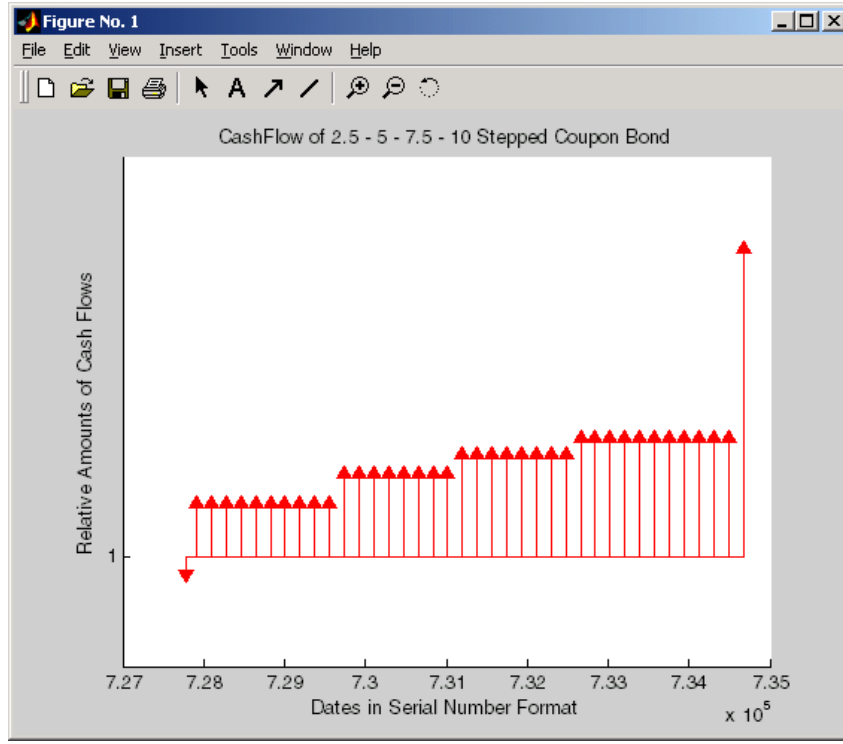
```
Settle   = datenum('02-Aug-1992');

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(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 a bond with stepped coupons

Compliance

SIA, PSA, ISDA, European, Japanese

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`	Conversion dates for the bonds. A matrix of serial date numbers.
`CouponRates`	Matrix indicating the initial coupon rate for each bond in decimal form.
`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, 2 = actual/360, 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).
`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.

ConvDates and CouponRates must be matrices of size NUMBONDS by maximum number of distinct coupons. Fill unspecified entries with NAN.

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.

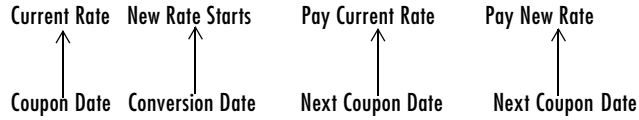
Note For bonds with fixed coupons, use bndprice. You will receive the error `incorrect number of inputs` if you use a fixed-coupon bond with stepcpnprice.

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
```

See Also

`bndprice`, `cdprice`, `stepcpncfamounts`, `stepcpnyield`, `tbillprice`, `zeroprice`

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.

stepcpnyield

Purpose

Yield to maturity of a bond with stepped coupons

Compliance

SIA, PSA, ISDA, European, Japanese

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`	Conversion dates for the bonds. A vector of serial date numbers.
`CouponRate`	Decimal number indicating the initial coupon rate for each bond.
`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, 2 = actual/360, 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).
`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.

ConvDates and CouponRates must be matrices of size NUMBONDS by the maximum number of distinct coupons. Fill unspecified entries with NaN.

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.

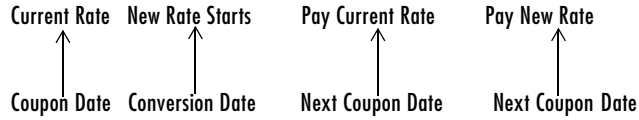
Yield is a NUMBONDS-by-1 vector of yields to maturity in decimal form.

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

```
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 =

    0.0722
    0.0722
    0.0722
```

See Also

`bndprice`, `cdprice`, `stepcpncfamounts`, `stepcpnprice`, `tbillprice`, `zeroprice`

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.

tbilldisc2yield

Purpose

Convert Treasury bill discount to equivalent yield

Compliance

SIA

Syntax

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

Arguments

`Yield`	Yield of Treasury bill in decimal.
`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 the bond-equivalent yield. Note that Treasury bond basis is actual/365.

`MMYield` is the money-market yield. The U.S. money-market basis is 30/360. This is equivalent to the algorithm used in `zeroyield` when pricing short-term instruments.

Use this function only when `Maturity` is 182 days or less. For longer maturities use `zeroyield`.

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(Yield, Settle, Maturity)

BEYield =

    0.0517

MMYield =

    0.0510
```

See Also

`tbillyield2disc`, `zeroyield`

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.

tbillprice

Purpose

Price a Treasury bill

Compliance

SIA

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) Yield type. 1 = money market (default). 2 = bond-equivalent. 3 = discount rate.

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

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.

Note Use this function only when there are 182 days or fewer to maturity. Use `zeroprice` for longer-maturity discount zero-coupon instruments.

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
   98.4980
```

See Also

`tbillyield`, `zeroprice`

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.

tbillrepo

Purpose

Break-even discount of repurchase agreement

Compliance

SIA

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 a number of 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

Compliance

SIA

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
```

See Also

`tbilldisc2yield`, `tbillprice`, `tbillyield`, `tbillyield2disc`

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.

tbillyield

Purpose

Yield on a Treasury bill

Compliance

SIA

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 a number of 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`. The U.S. Treasury bill basis is actual/360.

All outputs are `NTBILLS`-by-`1` vectors.

`MMYield` is the money-market yield of the Treasury bills.

`BEYield` is the bond equivalent yield of the Treasury bills.

`Discount` is the discount rate (annual) of the Treasury bills.

Note Use this function only when there are 182 days or fewer to maturity. Use `zeroyield` for longer-maturity discount zero-coupon instruments. To get the yield from the discount rate or vice versa, use `tbilldisc2yield` or `tbillyield2disc`.

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
```

See Also

`tbilldisc2yield`, `tbillprice`, `tbillyield2disc`, `zeroyield`

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.

tbillyield2disc

Purpose

Convert Treasury bill yield to equivalent discount

Compliance

SIA

Syntax

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

Arguments

`Yield`	Yield of Treasury bill in decimal.
`Settle`	Settlement date. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date.
`Type`	(Optional) Yield type. 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`.

Description

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

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

Use this function only when `Maturity` is 182 days or fewer.

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
```

See Also

`tbilldisc2yield`

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.

tfutbyprice

Purpose

Future prices of Treasury bonds given the 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.

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] = ...
bootstrapbyyield(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 =

    1.1381
    1.1250
```

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`

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.

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] = ...
bootstrapbyyield(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 =

    1.1381
    1.1250
```

This compares with 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 simple annual repurchase rate to prevent arbitrage

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

Theoretical futures bond 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

Theoretical futures bond 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]` `ReinvestmentRate` 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, Price, 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, Price, ...
Settle, MatFut, ConvFactor, CouponRate, Maturity)
```

See Also

`tfutimprepo`, `tfutpricebyrepo`

zeroprice

Purpose

Price zero-coupon instruments given yield

Compliance

SIA

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 coupons per year. Default = 2.
`Basis`	(Optional) Day-count basis of the bond. A vector of integers. 0 = actual/actual (default), 1 = 30/360, 2 = actual/360 (default), 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).
`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) you should use bond equivalent yield based upon present value (or compounding).

Formulas

To compute the price when there is one or zero quasi-coupon periods to redemption, `zeroprice` uses the formula

$$Price = \frac{RV}{1 + \left(\frac{DSR}{E} \cdot \frac{Y}{M}\right)}$$

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

$$Price = \frac{RV}{\left(1 + \frac{Y}{M}\right)^{Nq - 1 + \frac{DSC}{E}}}$$

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, etc.).
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
```

See Also

`bndprice`, `cdprice`, `tbillprice`, `zeroyield`

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.

zeroyield

Purpose

Yield of zero-coupon instruments given price

Compliance

SIA

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 coupons per year. Default = 2.
`Basis`	(Optional) Day-count basis of the bond. A vector of integers. 0 = actual/actual (default), 1 = 30/360, 2 = actual/360 (default), 3 = actual/365, 4 = 30/360 (PSA compliant), 5 = 30/360 (ISDA compliant), 6 = 30/360 (European), 7 = act/365 (Japanese).
`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 yields 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

$$Yield = \left(\frac{RV - P}{P}\right) \cdot \left(\frac{M \cdot E}{DSR}\right)$$

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

$$Yield = \left(\left(\frac{RV}{P}\right)^{\frac{1}{Nq - 1 + \frac{DSC}{E}}} - 1 \right) \cdot M$$

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 redemption date (call date, put date, etc.).
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.
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
```

See Also

bndyield, cdyield, tbillyield, zeroprice

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.

E

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 in order 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 An arithmetic method for backing an implied zero curve out of the par yield curve.

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 Binomial model.

Call **a.** An option to buy a certain quantity of a stock or commodity for a specified price within a specified time. See Put. **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; i.e., the holder has sold a call option to the issuer. See Puttable bond.
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.
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.
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; i.e., 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 Amortization.
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.
Discount curve	The curve of discount rates vs. maturity dates for bonds.
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 Macaulay duration.
Efficient frontier	A graph representing a set of portfolios that maximizes expected return at each level of portfolio risk. See Markowitz model.
Elasticity	See Lambda.
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 for bonds.
Forward rate	The future interest rate of a bond inferred from the term structure, especially from the yield curve of zero-coupon bonds, calculated from the growth factor of an investment in a zero held until maturity.

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; i.e., 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 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 Yield to maturity.
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 in order 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 Efficient frontier.
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; i.e., 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.
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 pro-rated 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.
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.
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.
Passthrough	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.
Present value	Today's value of an investment that yields some future value when invested to earn compounded interest at a known interest rate.; i.e., the future value at a known period in time discounted by the interest rate over that time period.
Principal value	See Par value.
Purchase price	Price actually 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 Call.
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; i.e., the holder has bought a put option. See Callable bond.
Redemption value	See Par value.
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.
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; i.e., 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.
Spot curve, spot yield curve	See Zero curve.
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 very volatile.
Strike	Exercise a put or call option.
Strike price	See Exercise price.
Swap	A contract between two parties to exchange cash flows in the future according to some formula.

Swaption	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 Swap.
Tenor	Life of a swap.
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 very 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.

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

O

P

Q

S

T

Z