

# Fixed-Income Toolbox™

## User's Guide

**R2012a**

**MATLAB®**

## How to Contact MathWorks



[www.mathworks.com](http://www.mathworks.com) Web  
[comp.soft-sys.matlab](mailto:comp.soft-sys.matlab) Newsgroup  
[www.mathworks.com/contact\\_TS.html](http://www.mathworks.com/contact_TS.html) Technical Support



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



508-647-7000 (Phone)



508-647-7001 (Fax)



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

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

*Fixed-Income Toolbox™ User's Guide*

© COPYRIGHT 2003–2012 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by, for, or through the federal government of the United States. By accepting delivery of the Program or Documentation, the government hereby agrees that this software or documentation qualifies as commercial computer software or commercial computer software documentation as such terms are used or defined in FAR 12.212, DFARS Part 227.72, and DFARS 252.227-7014. Accordingly, the terms and conditions of this Agreement and only those rights specified in this Agreement, shall pertain to and govern the use, modification, reproduction, release, performance, display, and disclosure of the Program and Documentation by the federal government (or other entity acquiring for or through the federal government) and shall supersede any conflicting contractual terms or conditions. If this License fails to meet the government's needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to The MathWorks, Inc.

### Trademarks

MATLAB and Simulink are registered trademarks of The MathWorks, Inc. See [www.mathworks.com/trademarks](http://www.mathworks.com/trademarks) for a list of additional trademarks. Other product or brand names may be trademarks or registered trademarks of their respective holders.

### Patents

MathWorks products are protected by one or more U.S. patents. Please see [www.mathworks.com/patents](http://www.mathworks.com/patents) for more information.

## Revision History

May 2003	Online only	New for Version 1.0 (Release 13)
November 2003	First printing	Unchanged
June 2004	Online only	Revised for Version 1.0.1 (Release 14)
August 2004	Online only	Revised for Version 1.1 (Release 14+)
September 2005	Online only	Revised for Version 1.1.1 (Release 14SP3)
March 2006	Online only	Revised for Version 1.1.2 (Release 2006a)
September 2006	Online only	Revised for Version 1.2 (Release 2006b)
March 2007	Online only	Revised for Version 1.3 (Release 2007a)
September 2007	Online only	Revised for Version 1.4 (Release 2007b)
March 2008	Online only	Revised for Version 1.5 (Release 2008a)
October 2008	Online only	Revised for Version 1.6 (Release 2008b)
March 2009	Online only	Revised for Version 1.7 (Release 2009a)
September 2009	Online only	Revised for Version 1.8 (Release 2009b)
March 2010	Online only	Revised for Version 1.9 (Release 2010a)
September 2010	Online only	Revised for Version 2.0 (Release 2010b)
April 2011	Online only	Revised for Version 2.1 (Release 2011a)
September 2011	Online only	Revised for Version 2.2 (Release 2011b)
March 2012	Online only	Revised for Version 2.3 (Release 2012a)



## Getting Started

### 1

<b>Product Description</b> .....	1-2
Key Features .....	1-2
<b>Expected Users</b> .....	1-3

## Mortgage-Backed Securities

### 2

<b>What Are Mortgage-Backed Securities?</b> .....	2-2
<b>Using Fixed-Rate Mortgage Pool Functions</b> .....	2-3
Introduction .....	2-3
Inputs to Functions .....	2-4
Generating Prepayment Vectors .....	2-4
Mortgage Prepayments .....	2-6
Risk Measurement .....	2-8
Mortgage Pool Valuation .....	2-9
Computing Option-Adjusted Spread .....	2-10
Prepayments with Fewer Than 360 Months Remaining ..	2-13
Pools with Different Numbers of Coupons Remaining ....	2-15
<b>Using Collateralized Mortgage Obligations (CMOs)</b> ...	2-17
What Are CMOs? .....	2-17
Prepayment Risk .....	2-17
CMO Workflow .....	2-28
<b>Create a PAC and Sequential CMO for Underlying Mortgage Pool</b> .....	2-32

## Debt Instruments

### 3

<b>Agency Option-Adjusted Spreads</b> .....	3-2
Computing the Agency OAS for Bonds .....	3-3
<b>Treasury Bills Defined</b> .....	3-7
<b>Computing Treasury Bill Price and Yield</b> .....	3-8
Introduction .....	3-8
Treasury Bill Repurchase Agreements .....	3-8
Treasury Bill Yields .....	3-10
<b>Using Zero-Coupon Bonds</b> .....	3-12
Introduction .....	3-12
Measuring Zero-Coupon Bond Function Quality .....	3-12
Pricing Treasury Notes .....	3-13
Pricing Corporate Bonds .....	3-15
<b>Stepped-Coupon Bonds</b> .....	3-17
Introduction .....	3-17
Cash Flows from Stepped-Coupon Bonds .....	3-17
Price and Yield of Stepped-Coupon Bonds .....	3-19
<b>Term Structure Calculations</b> .....	3-20
Introduction .....	3-20
Computing Spot and Forward Curves .....	3-20
Computing Spreads .....	3-22

## Derivative Securities

### 4

<b>Interest Rate Swaps</b> .....	4-2
Swap Pricing Assumptions .....	4-2
Swap Pricing Example .....	4-3
Portfolio Hedging .....	4-8

<b>Convertible Bond Valuation</b> .....	<b>4-10</b>
<b>Bond Futures</b> .....	<b>4-12</b>
Supported Bond Futures .....	<b>4-12</b>
Example Analysis of Bond Futures .....	<b>4-14</b>
Managing Interest-Rate Risk with Bond Futures .....	<b>4-16</b>

## Credit Derivatives

# 5

<b>Credit Default Swap (CDS)</b> .....	<b>5-2</b>
Bootstrapping a Default Probability Curve .....	<b>5-2</b>
Finding the Breakeven Spread for a New CDS Contract ..	<b>5-5</b>
Valuing an Existing CDS Contract .....	<b>5-8</b>
Converting from Running to Upfront and Vice Versa ....	<b>5-10</b>
Bootstrapping from Inverted Market Curves .....	<b>5-13</b>
 <b>Credit Default Swap Option</b> .....	 <b>5-17</b>

## Interest-Rate Curve Objects

# 6

<b>Introduction to Interest-Rate Curve Objects</b> .....	<b>6-2</b>
Class Structure .....	<b>6-2</b>
Supported Workflow Model Using Interest-Rate Curve Objects .....	<b>6-3</b>
 <b>Creating Interest-Rate Curve Objects</b> .....	 <b>6-4</b>
 <b>Creating an IRDataCurve Object</b> .....	 <b>6-6</b>
Using the IRDataCurve Constructor with Dates and Data .....	<b>6-6</b>
Using IRDataCurve bootstrap Method for Bootstrapping Based on Market Instruments .....	<b>6-7</b>

<b>Creating an IRFunctionCurve Object</b> .....	<b>6-13</b>
Using a Function Handle to Fit an IRFunctionCurve Object .....	<b>6-13</b>
Using the Nelson-Siegel Method to Fit an IRFunctionCurve Object .....	<b>6-14</b>
Using the Svensson Method to Fit an IRFunctionCurve Object .....	<b>6-16</b>
Using the Smoothing Spline Method to Fit an IRFunctionCurve Object .....	<b>6-18</b>
Using the fitFunction Method to Create a Custom Fitting Function for an IRFunctionCurve Object .....	<b>6-21</b>
<b>Converting an IRDataCurve or IRFunctionCurve Object</b> .....	<b>6-25</b>
Introduction .....	<b>6-25</b>
Using the toRateSpec Method .....	<b>6-25</b>
Using Vector of Dates and Data Methods .....	<b>6-27</b>

## Function Reference

# 7

<b>Bond Futures</b> .....	<b>7-2</b>
<b>Certificates of Deposit</b> .....	<b>7-3</b>
<b>Convertible Bonds</b> .....	<b>7-4</b>
<b>Credit Default Swaps</b> .....	<b>7-5</b>
<b>Derivative Securities</b> .....	<b>7-6</b>
<b>Interest-Rate Curve Objects</b> .....	<b>7-7</b>
<b>Mortgage-Backed Securities</b> .....	<b>7-9</b>
<b>Option-Adjusted Spread Computations</b> .....	<b>7-11</b>



<b>Stepped-Coupon Bonds</b> .....	<b>7-12</b>
<b>Treasury Bills</b> .....	<b>7-13</b>
<b>Zero-Coupon Instruments</b> .....	<b>7-14</b>

## Functions — Alphabetical List

# 8

## Class Reference

# A

<b>@IRBootstrapOptions</b> .....	<b>A-2</b>
Hierarchy .....	<b>A-2</b>
Constructor .....	<b>A-2</b>
Public Read-Only Properties .....	<b>A-2</b>
Methods .....	<b>A-3</b>
<b>@IRCurve</b> .....	<b>A-4</b>
Hierarchy .....	<b>A-4</b>
Description .....	<b>A-4</b>
Constructor .....	<b>A-4</b>
Public Read-Only Properties .....	<b>A-4</b>
Methods .....	<b>A-6</b>
<b>@IRDataCurve</b> .....	<b>A-7</b>
Hierarchy .....	<b>A-7</b>
Description .....	<b>A-7</b>
Constructor .....	<b>A-7</b>
Public Read-Only Properties .....	<b>A-8</b>
Methods .....	<b>A-9</b>
<b>@IRFitOptions</b> .....	<b>A-10</b>
Hierarchy .....	<b>A-10</b>
Description .....	<b>A-10</b>

Constructor .....	A-10
Public Read-Only Properties .....	A-11
Methods .....	A-11
<b>@IRFunctionCurve</b> .....	A-12
Hierarchy .....	A-12
Description .....	A-12
Constructor .....	A-12
Public Read-Only Properties .....	A-13
Methods .....	A-14

## Bibliography

### B

<b>Fitting Interest-Rate Curve Functions</b> .....	B-2
<b>Bootstrapping a Swap Curve</b> .....	B-3
<b>Bond Futures</b> .....	B-4
<b>Credit Derivatives</b> .....	B-5

## Examples

### C

<b>Agency Option Adjusted Spreads</b> .....	C-2
<b>Treasury Bills</b> .....	C-3
<b>Using Zero-Coupon Bonds</b> .....	C-4
<b>Stepped-Coupon Bonds</b> .....	C-5

<b>Pricing and Hedging</b> .....	<b>C-6</b>
<b>Bond Futures</b> .....	<b>C-7</b>
<b>Credit Default Swaps</b> .....	<b>C-8</b>

**Glossary**



**Index**





# Getting Started

---

- “Product Description” on page 1-2
- “Expected Users” on page 1-3

## Product Description

### **Model and analyze fixed-income securities**

Fixed-Income Toolbox™ provides functions for fixed-income modeling and analysis. The toolbox includes tools for fitting yield curves to market data using parametric fitting models and bootstrapping. You can calculate the price, rates, and sensitivities for interest rate swaps. You can also price and value other derivatives, including credit default swaps, bond futures, and convertible bonds.

Fixed-Income Toolbox also includes tools for determining the price, yield, and cash flow for many types of fixed-income securities, including mortgage-backed securities, corporate bonds, treasury bonds, municipal bonds, certificates of deposit, and treasury bills.

### **Key Features**

- Yield curve fitting with bootstrapping and parametric fitting models
- Price, rate, and sensitivity calculation for interest rate swaps
- Price and value calculation for credit default swaps
- Price, yield, discount rate, and cash-flow schedule calculation for debt instruments, including treasury bills, zero-coupon bonds, and stepped-coupon bonds
- Price and option adjusted spread calculation for bonds
- Price and rate calculation for convertible bonds, bond futures, and European call and put options
- Price and yield calculation for generic fixed-rate mortgage pools and balloon mortgages

## Expected Users

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

Your title is likely one of these:

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

Your background, education, training, and responsibilities likely match some aspects of this profile:

- Finance, economics, perhaps accounting
- Engineering, mathematics, physics, other quantitative sciences
- Bachelor's degree minimum; MS or MBA likely; Ph.D. perhaps; CFA
- Comfortable with probability theory, statistics, and algebra
- Understand linear or matrix algebra and calculus
- Perhaps new to MATLAB® software





# Mortgage-Backed Securities

---

- “What Are Mortgage-Backed Securities?” on page 2-2
- “Using Fixed-Rate Mortgage Pool Functions” on page 2-3
- “Using Collateralized Mortgage Obligations (CMOs)” on page 2-17
- “Create a PAC and Sequential CMO for Underlying Mortgage Pool” on page 2-32

## **What Are Mortgage-Backed Securities?**

Mortgage-backed securities (MBSs) are a type of investment that represents ownership in a group of mortgages. Principal and interest from the individual mortgages are used to pay principal and interest on the MBS.

Ownership in a group of mortgages is typically represented by a *pass-through certificate* (PC). Most pass-through certificates are issued by the Government National Mortgage Agency, a branch of the United States government, or by one of two private corporations: Fannie Mae or Freddie Mac. With these certificates, homeowners' payments pass from the originating bank through the issuing agency to holders of the certificates. These agencies also frequently guarantee that the certificate holder receives timely payment of principal and interest from the PCs.

## Using Fixed-Rate Mortgage Pool Functions

### In this section...

“Introduction” on page 2-3

“Inputs to Functions” on page 2-4

“Generating Prepayment Vectors” on page 2-4

“Mortgage Prepayments” on page 2-6

“Risk Measurement” on page 2-8

“Mortgage Pool Valuation” on page 2-9

“Computing Option-Adjusted Spread” on page 2-10

“Prepayments with Fewer Than 360 Months Remaining” on page 2-13

“Pools with Different Numbers of Coupons Remaining” on page 2-15

### Introduction

Fixed-Income Toolbox software supports calculations involved with generic fixed-rate mortgage pools and balloon mortgages. Pass-through certificates typically have embedded call options in the form of prepayment. Prepayment is an excess payment applied to the principal of a PC. These accelerated payments reduce the effective life of a PC.

The toolbox comes with a standard Bond Market Association (PSA) prepayment model and can generate multiples of standard prepayment speeds. The Public Securities Association provides a set of uniform practices for calculating the characteristics of mortgage-backed securities when there is an assumed prepayment function.

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

Alternatively, aside from the standard PSA implementation in this toolbox, you can supply your own projected prepayment vectors. At this time, however, custom prepayment functionality that incorporates pool-specific information and interest rate forecasts are not available in this toolbox. If you plan to use

custom prepayment vectors in your calculations, you presumably already own such a suite in MATLAB.

### Inputs to Functions

Because of the generic, all-purpose nature of the toolbox pass-through functions, you can fine-tune them to conform to a particular mortgage. Most functions require at least this set of inputs:

- Gross coupon rate
- Settlement date
- Issue (effective) date
- Maturity date

Typical optional inputs include standard prepayment speed (or customized vector), net coupon rate (if different from gross coupon rate), and payment delay in number of days.

All calculations are based on expected payment dates and actual cash flow to the investor. For example, when `GrossRate` and `CouponRate` differ as inputs to `mbsdurp`, the function returns a modified duration based on `CouponRate`. (A notable exception is `mbspassthrough`, which returns interest quantities based on the `GrossRate`.)

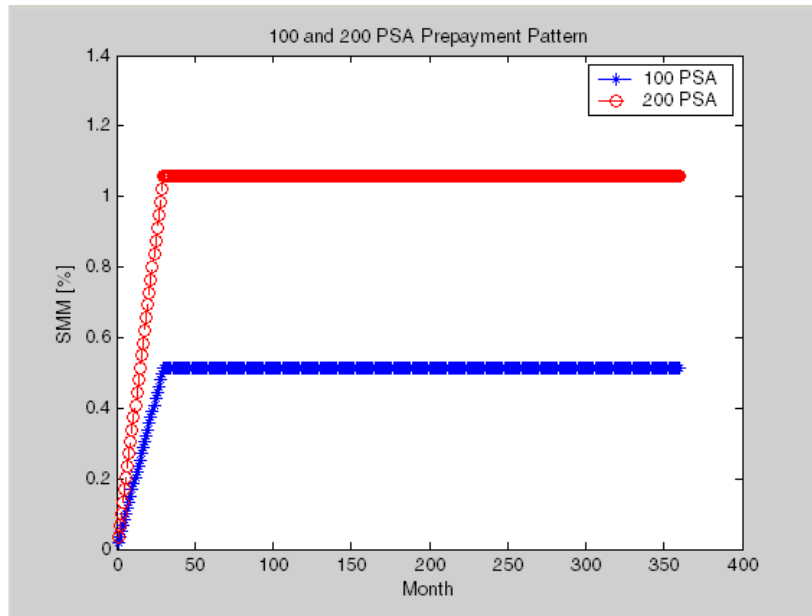
### Generating Prepayment Vectors

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

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

This function computes two prepayment values: conditional prepayment rate (CPR) and single monthly mortality (SMM) rate. CPR is the percentage of outstanding principal prepaid in 1 year. SMM is the percentage of outstanding principal prepaid in 1 month. In other words, CPR is an annual version of SMM.

Since the entire 360-by-2 array is too long to show in this document, observe the SMM (100 and 200 PSA) plots, spaced one month apart, instead.



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

Consider three mortgage pools that were sold on the issue date (which starts unamortized). The first two pools "balloon out" in 60 months, and the third is regularly amortized to the end. The prepayment speeds are assumed to be 100, 200, and 200 PSA, respectively.

```
Settle      = [datenum('1-Feb-2000');
              datenum('1-Feb-2000');
              datenum('1-Feb-2000')];

Maturity    = [datenum('1-Feb-2030')];

IssueDate   = datenum('1-Feb-2000');
```

```
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;

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

PrepayMatrix = ones(360,3);
PrepayMatrix(1:60,1:2) = SMM(1:60,1:2);
PrepayMatrix(:,3) = SMM(:,2);

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

The fourth output argument, `Factors`, indicates the fraction of the balance still outstanding at the beginning of each month. A snapshot of this argument in the MATLAB Variable Editor illustrates the 60-month life of the first two of the mortgages with balloon payments and the continuation of the third mortgage until the end (360 months).

	59	60	61	62	63
1	0.7627	0.7580	0.7533	0	0
2	0.6021	0.5951	0.5882	0	0
3	0.6021	0.5951	0.5882	0.5813	0.5746

You can readily see that `mbscfamounts` is the building block of most fixed rate and balloon pool cash flows.

## Mortgage Prepayments

Prepayment is beneficial to the pass-through owner when a mortgage pool has been purchased at discount. The next example compares mortgage yields (compounded monthly) versus the purchase clean price with constant prepayment speed. The example illustrates that when you have purchased

a pool at a discount, prepayment generates a higher yield with decreasing purchase price.

```
Price = [85; 90; 95];
Settle = datenum('15-Apr-2002');
Maturity = datenum('1 Jan 2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;
```

Compute the mortgage and bond-equivalent yields.

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

```
MYield =
```

```
    0.1018
    0.0918
    0.0828
```

```
BEMBSYield =
```

```
    0.1040
    0.0936
    0.0842
```

If for this same pool of mortgages, there was no prepayment (Speed = 0), the yields would decline to

```
MYield =
```

```
    0.0926
    0.0861
    0.0802
```

```
BEMBSYield =
```

```
    0.0944
```

0.0877  
0.0815

Likewise, if the rate of prepayment doubled (Speed = 200), the yields would increase to

MYield =

0.1124  
0.0984  
0.0858

BEMBSYield =

0.1151  
0.1004  
0.0873

For the same prepayment vector, deeper discount pools earn higher yields. For more information, see `mbsprice` and `mbsyield`.

### Risk Measurement

Fixed-Income Toolbox software provides the most basic risk measures of a pool portfolio:

- Modified duration
- Convexity
- Average life of pool

Consider the following example, which calculates the Macaulay and modified durations given the price of a mortgage pool.

```
Price = [95; 100; 105];  
Settle = datenum('15-Apr-2002');  
Maturity = datenum('1-Jan-2030');  
IssueDate = datenum('1-Jan-2000');  
GrossRate = 0.08125;  
CouponRate = 0.075;
```



```
Delay = 14;  
Speed = 100;  
  
[YearDuration, ModDuration] = mbsdurp(Price, Settle, ...  
Maturity, IssueDate, GrossRate, CouponRate, Delay, Speed)  
  
YearDuration =  
  
    6.1341  
    6.3882  
    6.6339  
  
ModDuration =  
  
    5.8863  
    6.1552  
    6.4159
```

Using Fixed-Income Toolbox functions, you can obtain modified duration and convexity from either price or yield, as long as you specify a prepayment vector or an assumed prepayment speed. The toolbox risk-measurement functions (`mbsdurp`, `mbsdury`, `mbsconvp`, `mbsconvy`, and `mbswal`) adhere to the guidelines listed in the *PSA Uniform Practices* manual.

## Mortgage Pool Valuation

For accurate valuation of a mortgage pool, you must generate interest rate paths and use them with mortgage pool characteristics to properly value the pool. A widely used methodology is the option-adjusted spread (OAS). OAS measures the yield spread that is not directly attributable to the characteristics of a fixed-income investment.

### Calculating OAS

Prepayment alters the cash flows of an otherwise regularly amortizing mortgage pool. A comprehensive option-adjusted spread calculation typically begins with the generation of a set of paths of spot rates to predict prepayment. A path is collection of  $i$  spot-rate paths, with corresponding  $j$  cash flows on each of those paths.

The effect of the OAS on pool pricing is shown mathematically in the following equation, where  $K$  is the option-adjusted spread.

$$PoolPrice = \frac{1}{NumberofPaths} \times \sum_i^{NumberofPaths} \sum_j^{CF_{ij}} \frac{CF_{ij}}{(1 + zerorates_{ij} + K)^{T_{ij}}}$$

### Calculating Effective Duration

Alternatively, if you are more interested in the sensitivity of a mortgage pool to interest rate changes, use effective duration, which is a more appropriate measure. Effective duration is defined mathematically with the following equation.

$$Effective\ Duration = \frac{P(y + \Delta y) - P(y - \Delta y)}{2P(y)\Delta y}$$

### Calculating Market Price

The toolbox has all the components required to calculate OAS and effective duration if you supply prepayment vectors or assumptions. For OAS, given a prepayment vector, you can generate a set of cash flows with `mbscf` amounts. Discounting these cash flows with the reference curve and then adding OAS produces the market price. See “Computing Option-Adjusted Spread” on page 2-10 for a discussion on the computation of option-adjusted spread.

Effective duration is a more difficult issue. While modified duration changes the discounting process (by changing the yield used to discount cash flows), effective duration must account for the change in cash flow because of the change in yield. A possible solution is to recompute prices using `mbsprice` for a small change in yield, in both the upwards and downwards directions. In this case, you must recompute the prepayment input. Internally, this alters the cash flows of the mortgage pool. Assuming that the OAS stays constant in all yield environments, you can apply a set of discounting factors to the cash flows in up and down yield environments to find the effective duration.

### Computing Option-Adjusted Spread

The option-adjusted spread (OAS) is an amount of extra interest added above (or below if negative) the reference zero curve. To compute the OAS, you must

provide the zero curve as an extra input. You can specify the zero curve in any intervals and with any compounding method. (To minimize any error due to interpolation, keep the intervals as regular and frequent as possible.) You must supply a prepayment vector or specify a speed corresponding to a standard PSA prepayment vector.

One way to compute the appropriate zero curve for an agency is to look at its bond yields and bootstrap them from the shortest maturity onwards. You can do this with Financial Toolbox™ functions `zbtprice` and `zbtyield`.

The following example shows how to calculate an appropriate zero curve followed by computation of the pool's OAS. This example calculates the OAS of a 30-year fixed rate mortgage with about a 28-year weighted average maturity left, given an assumption of 0, 50, and 100 PSA prepayment speeds.

Create curve for zerorates.

```
Bonds = [datenum('11/21/2002') 0 100 0 2 1;
         datenum('02/20/2003') 0 100 0 2 1;
         datenum('07/31/2004') 0.03 100 2 3 1;
         datenum('08/15/2007') 0.035 100 2 3 1;
         datenum('08/15/2012') 0.04875 100 2 3 1;
         datenum('02/15/2031') 0.05375 100 2 3 1];

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

Since the above is Treasury data and not selected agency data, a term structure of spread is assumed. In this example, the spread declines proportionally from a maximum of 250 basis points at the shortest maturity.

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

Get parameters from Bonds matrix.

```
Settle = datenum('20-Aug-2002');
Maturity = Bonds(:,1);
```

```
CouponRate = Bonds(:,2);
Face = Bonds(:,3);
Period = Bonds(:,4);
Basis = Bonds(:,5);
EndMonthRule = Bonds(:,6);

[Prices, AccruedInterest] = bndprice(Yields, CouponRate, ...
Settle, Maturity, Period, Basis, EndMonthRule, [], [], [], [], ...
Face);
```

Use `zbtprice` to solve for zero rates.

```
[ZeroRatesP, CurveDatesP] = zbtprice(Bonds, Prices, Settle);
ZeroCompounding = 2*ones(size(ZeroRatesP));
ZeroMatrix = [CurveDatesP, ZeroRatesP, ZeroCompounding];
```

Use output from `zbtprice` to calculate the OAS.

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

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

```
OAS =

    26.0502
    28.6348
    31.2222
```

This example shows that one cash flow set is being discounted and solved for its OAS, as contrasted with the `NumberOfPaths` set of cash flows as shown

in “Mortgage Pool Valuation” on page 2-9. Averaging the sets of cash flows resulting from all simulations into one average cash flow vector and solving for the OAS, discounts the averaged cash flows to have a present value of today’s (average) price.

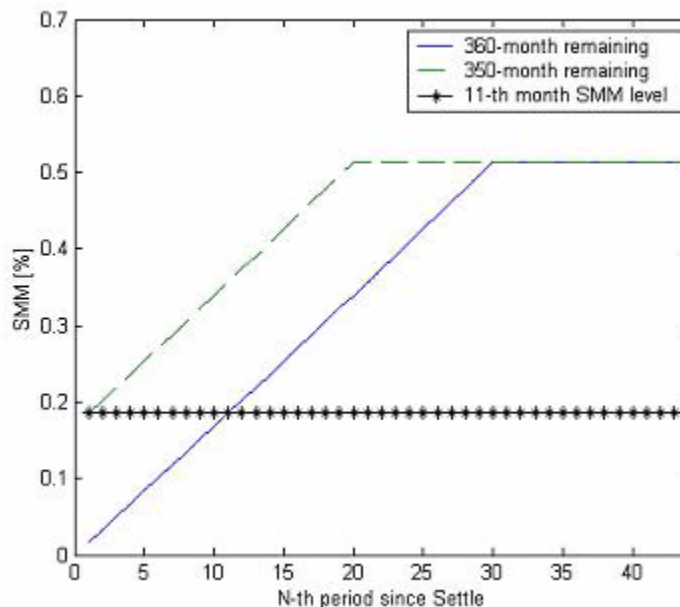
While this example uses the mortgage pool price (`mbsprice2oas`) to determine the OAS, you can also use yield to resolve it (`mbsyield2oas`). Also, there are reverse OAS functions that return prices and yields given OAS (`mbsoas2price` and `mbsoas2yield`).

The example also restates earlier examples that show discount securities benefit from higher level of prepayment, keeping everything else unchanged. The relation is reversed for premium securities.

## **Prepayments with Fewer Than 360 Months Remaining**

When fewer than 360 months remain in the pool, the applicable PSA prepayment vector is "seasoned" by the pool’s age. (Elements in the 360-element prepayment vector that represent past payments are skipped. For example, on a 30-year mortgage that is 10 months old, only the final 350 prepayments are applied.)

Assume, for example, that you have two 30-year loans, one new and another 10 months old. Both have the same PSA speed of 100 and prepay using the vectors plotted below.



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

If you supply a custom prepayment vector, you must account for the number of months remaining.

```
Price = 101;
Settle = datenum('1-Jan-2001');
Maturity = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
PrepayMatrix = 0.005*ones(348,1);
CouponRate = 0.075;
Delay = 14;
```

```
ImpliedSpeed = mbsprice2speed(Price, Settle, Maturity, ...
IssueDate, GrossRate, PrepayMatrix, CouponRate, Delay)
```

```
ImpliedSpeed =
```

```
104.2543
```

Examine the prepayment input. The remaining 29 years require 348 monthly elements in the prepayment vector. Suppose then, keeping everything the same, you change `Settle` to February 14, 2003.

```
Settle = datenum('14-Feb-2003');
```

You can use `cpncount` to count all incoming coupons received after `Settle` by invoking

```
NumCouponsRemaining = cpncount(Settle, Maturity, 12, 1, [], ...
IssueDate)
```

```
NumCouponsRemaining =
323
```

The input 12 defines the monthly payment frequency, 1 defines the 30/360 basis, and `IssueDate` defines aging and determination-of-holder date. Thus, you must supply a 323-element vector to account for a prepayment corresponding to each monthly payment.

## Pools with Different Numbers of Coupons Remaining

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

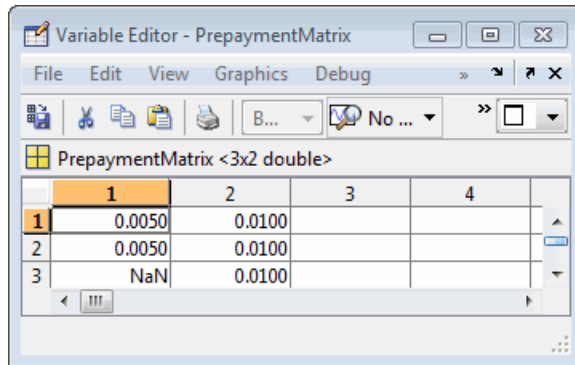
```
V11      V21
V12      V22
NaN      V23
```

$V_{i,j}$  denotes the single monthly mortality (SMM) rate for pool  $i$  during the  $j$ th coupon period since `Settle`.

The use of NaN to pad the prepayment matrix is necessary because MATLAB cannot concatenate vectors of different lengths into a matrix. Also, it can

serve as an error check against any unintended operation (any MATLAB operation that would return NaN).

For example, assume that the 2 month pool has a constant SMM of 0.5% and the 3 month has a constant SMM of 1% in every period. The prepayment matrix you would create is depicted below.



	1	2	3	4
1	0.0050	0.0100		
2	0.0050	0.0100		
3	NaN	0.0100		

Create this input in whatever manner is best for you.

### Summary of Prepayment Data Vector Representation

- When you specify a PSA prepayment speed, MATLAB "seasons" the pool according to its age.
- When you specify your own prepayment matrix, identify the maximum number of coupons remaining using `cpncount`. Then supply the matrix elements up to the point when cash flow ceases to exist.
- When different length pools must exist in the same matrix, pad the shorter one(s) with NaN. Each column of the prepayment matrix corresponds to a specific pool.



## Using Collateralized Mortgage Obligations (CMOs)

In this section...
“What Are CMOs?” on page 2-17
“Prepayment Risk” on page 2-17
“CMO Workflow” on page 2-28

### What Are CMOs?

Fixed-Income Toolbox supports collateralized mortgage obligations (CMOs) to provide investors with a greater range of risk and return characteristics than mortgage-backed securities (MBS). In contrast to an MBS, which simply redirects principal and interest cash flows to investors on a pro rata basis, a CMO structures cash flows to different tranches, or slices, to create securities that are better tailored to specific investors.

For example, banks might be primarily concerned with *extension risk*, or the risk that their investment lengthens in time due to increasing interest rates, given that they typically have short-term deposits as liabilities. Insurance companies and pension funds might be concerned primarily with *contraction risk*, or the risk that their investment will pay off too soon, with liabilities that have much longer lives. A CMO structure addresses the interest-rate risk of extension or contraction with a blend of short-term and long-term CMO securities, called tranches.

### Prepayment Risk

Prepayment risk is the risk that the term of the security varies according to differing rates of repayment of principal by borrowers (repayments from refinancings, sales, curtailments, or foreclosures). In a CMO, you can structure the principal (and associated coupon) stream from the underlying mortgage pool collateral to allocate prepayment risk. If principal is prepaid faster than expected (for example, if mortgage rates fall and borrowers refinance), then the overall term of the mortgage pool collateral shortens.

You cannot remove prepayment risk, but you can reallocate it among CMO tranches so that some tranches have some protection against this risk, and other tranches will absorb more of this risk. To facilitate this allocation of

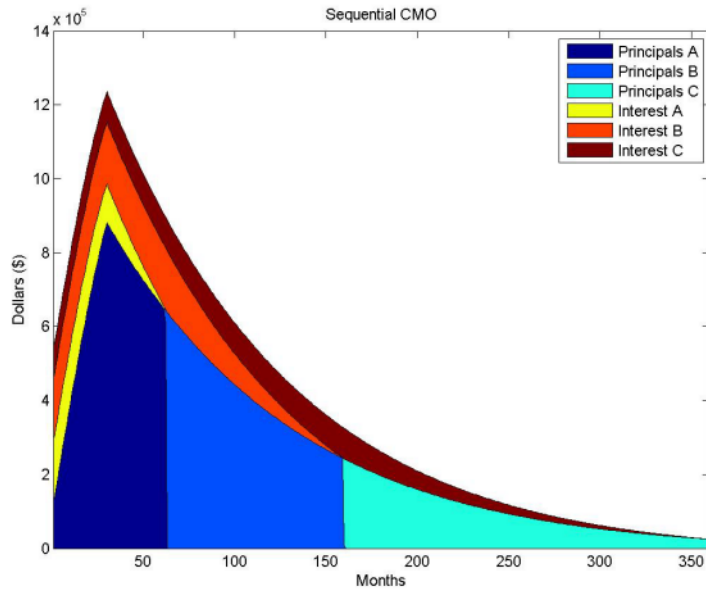
prepayment risk, CMOs are structured such that prepayments are allocated among tranches using a fixed set of rules. The most common schemes for prepayment tranching are:

- Sequential tranching, with or without, Z-bond tranching
- Schedule bond tranching
  - Planned amortization class (PAC) bonds
  - Target amortization class (TAC) bonds

Fixed-Income Toolbox supports these schemes for prepayment tranching for CMOs and tools for pricing and scheduling cash flows between the tranches, as well as analyzing the price and yield for CMOs. Fixed-Income Toolbox functionality for CMOs does not model credit risk. Therefore, this functionality is most appropriate for CMOs where credit risk is not an issue (for example, agency CMOs where the underlying mortgage pool collateral is insured for default by the agency Government-Sponsored Enterprises (GSEs), such as Fannie Mae and Freddie Mac).

### **Sequential Tranches Without a Z-Bond**

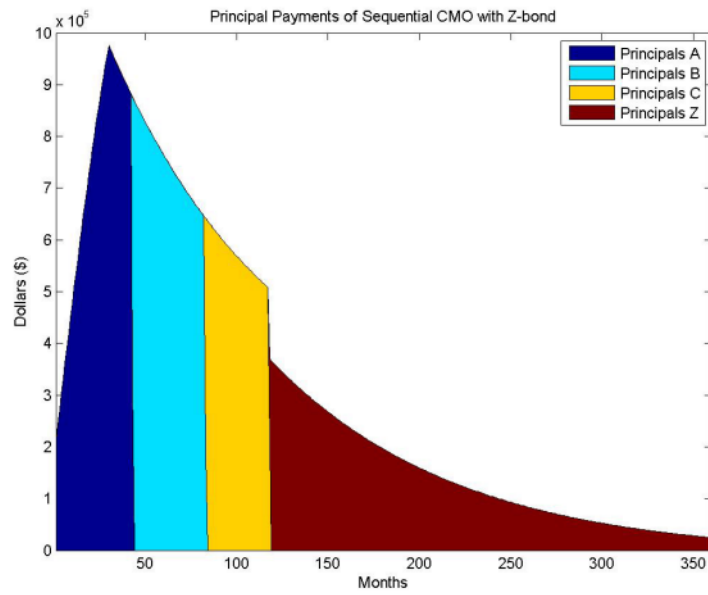
All available principal and interest payments go to the first sequential tranche, until its balance decrements to zero, then to the second, and so on. For example, consider the following example where all principal and interest from the underlying mortgage pool is repaid on tranche A first, then tranche B, then tranche C. Note that interest is paid on each tranche as long as the principal for the tranche has not been retired.



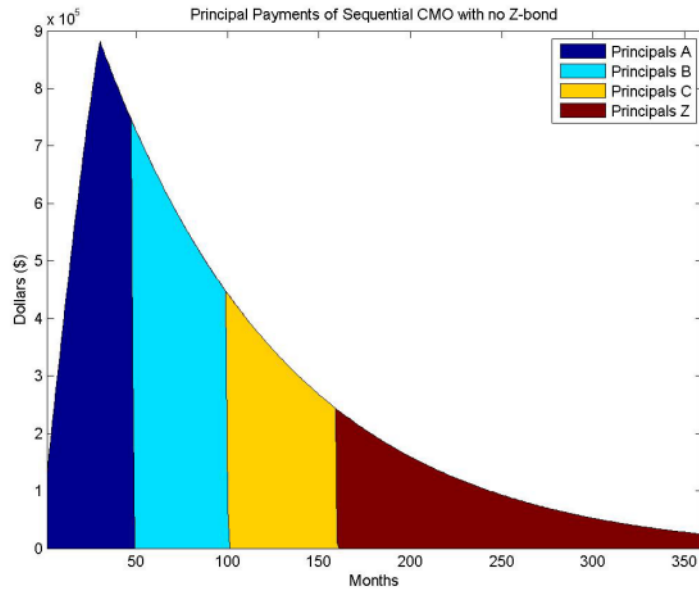
### Sequential Tranches With a Z-Bond

The Z-bond, also called an accrual bond, is a type of interest and principal pay rule. The Z-bond tranche supports other sequential pay tranches by not receiving an interest payment. The interest payment that would have accrued to the Z-bond tranche pays off the principal of other bonds, and the principal of the Z-bond tranche increases. The Z-bond tranche starts receiving interest and principal payments only after the other tranches in the CMO have been fully paid. The Z-bond tranche is used in a sequential-pay structure to accelerate the principal repayments of the sequential-pay bonds.

A Z-bond differs from other CMO instruments because it is not tranching principal but interest. The Z-bond receives no cash flows until all other securities have been paid off. In the interim, the interest that is owed to the Z-bond is accrued to its principal. The following chart demonstrates the difference between a Z-bond and a normal sequential pay tranche. Note that the C tranche pays off sooner with the Z-bond, because the interest cash flows to the Z-bond are being used to pay down the principal of the C tranche.



For comparison, the following graphic is the same sequential CMO with no Z-bond.



## PAC Tranches

Planned amortization class (PAC) bonds help reduce the effects of prepayment risk. They are designed to produce more stable cash flows by redirecting prepayments from the underlying mortgage collateral to other classes (tranches) called companion or support classes. PAC bonds have a principal payment rate over a predetermined period of time. The PAC bond payment schedule is determined by two different prepayment rates, which together form a band (also called a collar). Early in the life of the CMO, the prepayment at the lower PSA yields a lower prepayment. Later in its life, the principal in the higher PSA declines enough that it yields a lower prepayment. The PAC tranche receives whichever rate is lower, so it will change prepayment at one PSA for the first part of its life, then switch to the other rate. The ability to stay on this schedule is maintained by a support bond, which absorbs excess prepayments, and receives less prepayments to prevent extension of average life.

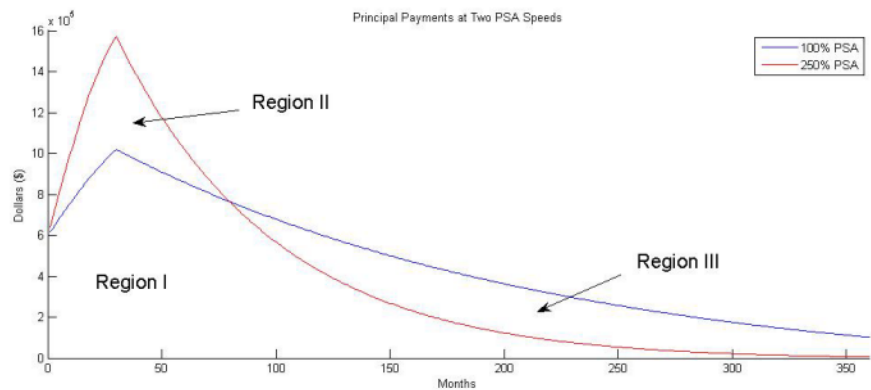
However, the PAC is only protected from extension to the amount that prepayments are made on the underlying MBSs. If there is a sustained

period of fast prepayments, then that might completely eliminate a PAC bond's outstanding support class. When the principal of the associated PAC bond is exhausted, the CMO is called a "busted PAC", or "busted collar". Alternatively, in times of slow prepayments, amortization of the support bonds is delayed if there is not enough principal for the currently paying PAC bond. This extends the average life of the class.

A PAC bond protects against both extension and contraction risk by:

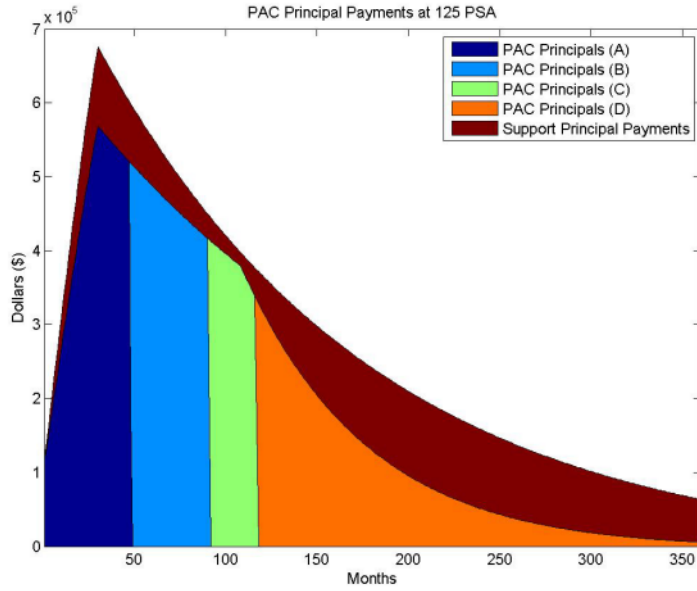
- Specifying a schedule of principal payments for the PAC bond
- Including support tranches that are allocated prepayments inside a specified prepayment band

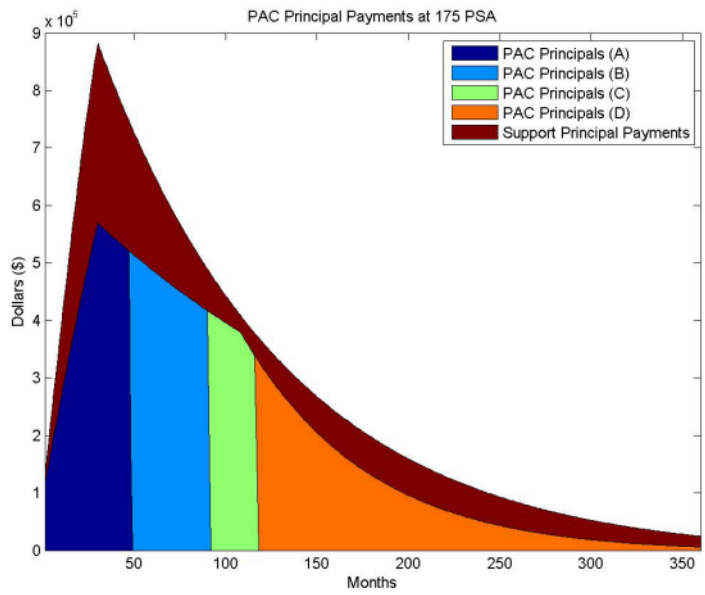
PAC bonds typically specify a band expressed using the PSA model. A PAC bond with a range of 100 to 250% has this principal schedule.



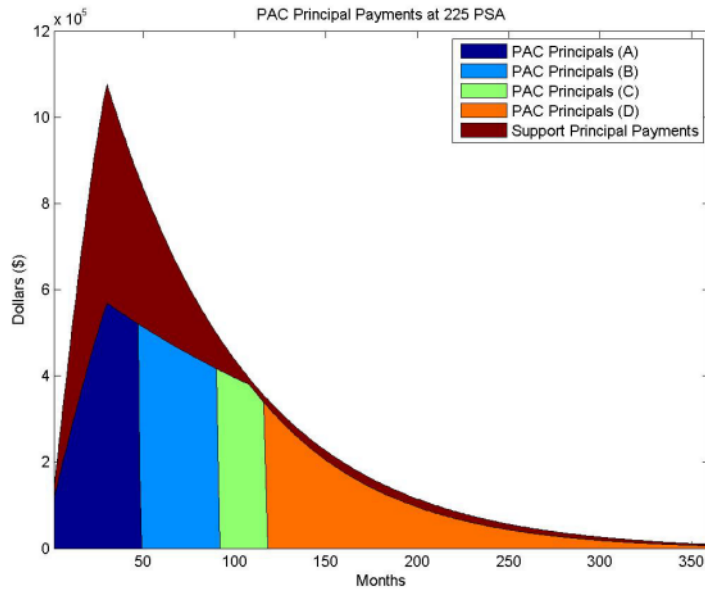
The principal repayment schedule is the minimum principal payment as Region 1 shows. Note that this is the principal payment schedule as long as the actual prepayment stays within the prepayment band of 100 to 250% PSA.

For example, for different prepayment speeds of 125%, 175%, and 225% PSA, the actual principal payments are shown in the following graphs. Note that at higher prepayment speeds, the support tranche is allocated principal earlier while the principal timing for the other tranches remains constant.





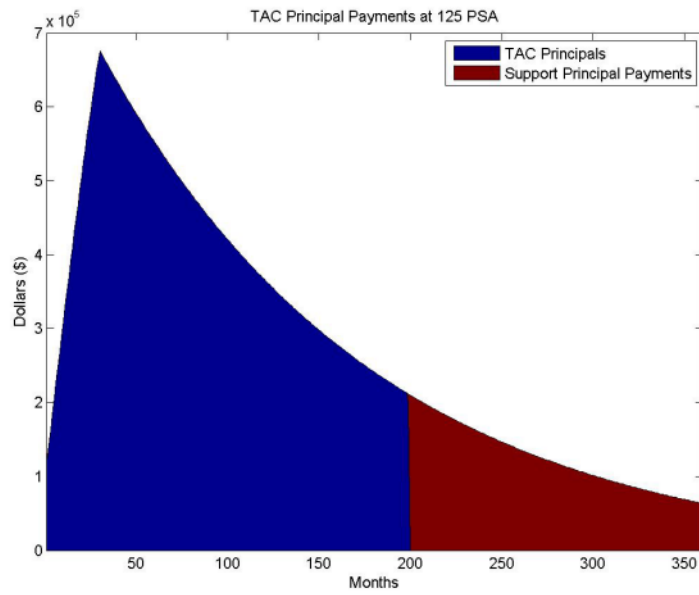


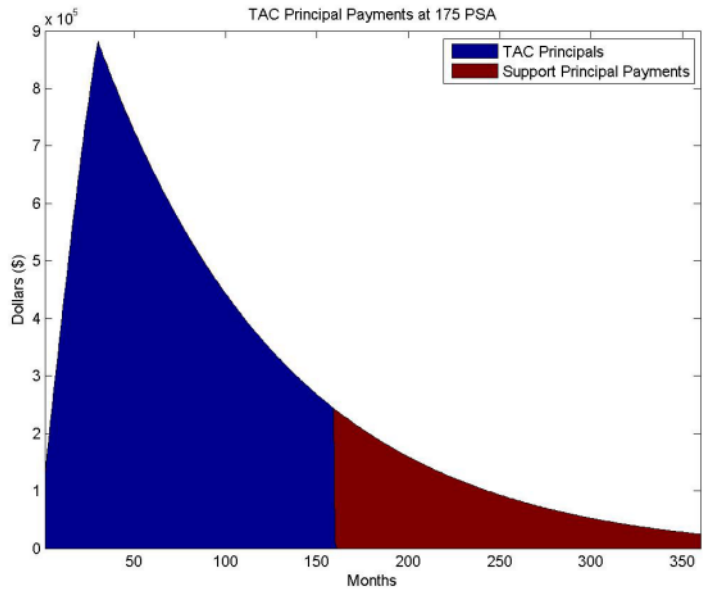


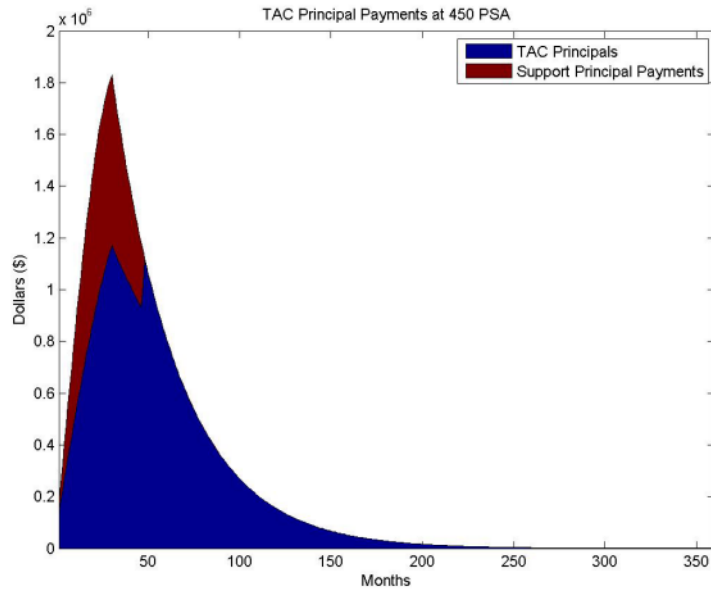
## TAC Tranches

Target amortization class (TAC) bonds are similar to PAC bonds, but they do not provide protection against extension of average life. Create the schedule of principal payments by using just a single PSA. TAC bonds pay a “targeted” principal payment schedule at a single, constant prepayment speed. As long as the underlying mortgage collateral does not prepay at a rate slower than this speed, the TAC bond payment schedule is met. TAC bonds can protect against increasing prepayments and early retirement of the TAC bond investment. If the principal cash flow from the mortgage collateral exceeds the TAC schedule, the excess is allocated to TAC companion (support) classes. Alternatively, if prepayments fall below the speed necessary to maintain the TAC schedule, the weighted average life of the TAC is extended. The TAC bond does not protect against low prepayment rates.

For example, here is a TAC structure rated for 125%, 175%, and 450% PSA.







Note that for prepayments below 175% PSA, the TAC bond extends like a normal sequential pay CMO. TAC bonds are appealing because they offer higher yields than comparable PAC bonds. The unaddressed risk from low prepayment rates generally does not concern investors as much as risk from high prepayment rates.

### CMO Workflow

In general, the CMO workflow is:

- 1 Calculate underlying mortgage cash flows.
- 2 Define CMO tranches
- 3 If using a PAC or TAC CMO, calculate the principal schedule.
- 4 Calculate cash flows for each tranche.
- 5 Analyze the CMO by computing price, yield, spread of CMO cash flows.

## Calculate Underlying Mortgage Cash Flows

Underlying mortgage pool pass-through cash flows are calculated by the existing function `mbspassthrough`. The CMO cash flow functions require the principal payments (including prepayments) calculated from existing functions `mbspassthrough` or `mbscfamounts`.

```
principal = 10000000;
coupon = 0.06;
terms = 360;
psa = 150;

[principal_balance, monthly_payments, sched_principal_payments,...
interest_payments, prepayments] = mbspassthrough(principal,...
coupon, terms, terms, psa, []);

principal_payments = sched_principal_payments.' + prepayments.';
```

After determining principal payments for the underlying mortgage collateral, you can generate cash flows for a sequential CMO, with or without a Z-bond, by using `cmoseqcf`. For a PAC or TAC CMO, the cash flows are generated using `cmoschedcf`

## Define CMO Tranches

Define CMO tranche; for example, define a CMO with 2 tranches:

```
TranchePrincipals = [500000; 500000];
TrancheCoupons = [0.06; 0.06];
```

## If Using a PAC or TAC CMO, Calculate Principal Schedule

Calculate the PAC/TAC principal balance schedule based on a band of PSA speeds. For scheduled CMOs (PAC/TAC), the CMO cash flow functions additionally take in the principal balance schedule calculated by the CMO schedule function `cmosched`.

```
speed = [100 300];
[balanceSchedule, initialBalance] = cmosched(principal, coupon,...
terms, terms, speed, TranchePrincipals(1));
```

## Calculate Cash Flows for Each Tranche

You can reuse the output from the cash flow generation functions to further divide the cash flows into tranches. For example, the output from `cmoschedcf` for a PAC tranche can be divided into sequential tranches by passing the principal cash flows of the PAC tranche into the `cmoschedcf` function. The output of the CMO cash flow functions are the principal and interest cash flows, as well as the principal balance.

```
[principal_balances, principal_cashflows, interest_cashflows] = cmoschedcf(principal_payments,...  
TranchePrincipals, TrancheCoupons, balanceSchedule);
```

## Analyze CMO by Computing Price, Yield, and Spread of CMO Cash Flows

The outputs from the CMO functions (`cmoseqcf` and `cmoschedcf`) are cash flows. The functions used to analyze a CMO are based on these cash flows. To that end, you can use `cfbyzero`, `cfspread`, `cfyield`, and `cfprice` to compute prices, yield, and spreads for the CMO cash flows. In addition, using the following, you can calculate a weighted average life (WAL) for each tranche in the CMO:

$$WAL = \sum_{i=1}^n \frac{P_i}{P} t_i$$

where:

$P$  is the total principal.

$P_i$  is the principal repayment of the coupon  $i$ .

$\frac{P_i}{P}$  is the fraction of the principal repaid in coupon  $i$ .

$t_i$  is the time in years from the start to coupon  $i$ .

### See Also

`cmoseqcf` | `cmosched` | `cmoschedcf` | `mbscfamounts` | `mbspassthrough`

## **Related Examples**

- “Create a PAC and Sequential CMO for Underlying Mortgage Pool” on page 2-32
- “Using Fixed-Rate Mortgage Pool Functions” on page 2-3

## **More About**

- “What Are Mortgage-Backed Securities?” on page 2-2

## Create a PAC and Sequential CMO for Underlying Mortgage Pool

This example shows how to use an underlying mortgage-backed security (MBS) pool for a 30-year fixed-rate mortgage of 6% to define a PAC bond, and then define a sequential CMO from the PAC bond. Analyze the CMO by comparing the CMO spread to a zero-rate curve for a 30-year Treasury bond and then calculate the weighted-average life (WAL) for the PAC bond.

### Step 1. Define the underlying mortgage pool.

```
principal = 100000000;  
grossrate = 0.06;  
coupon = 0.05;  
originalTerm = 360;  
termRemaining = 360;  
speed = 100;  
delay = 14;  
  
Settle      = datenum('1-Jan-2011');  
IssueDate  = datenum('1-Jan-2011');  
Maturity    = addtodate(IssueDate, 360, 'month');
```

### Step 2. Calculate underlying pool cash flow.

```
[CFlowAmounts, CFlowDates, ~, ~, ~, UnitPrincipal, UnitInterest, ...  
UnitPrepayment] = mbscfamounts(Settle, Maturity, IssueDate, grossrate, ...  
coupon, delay, speed, []);
```

### Step 3. Calculate prepayments.

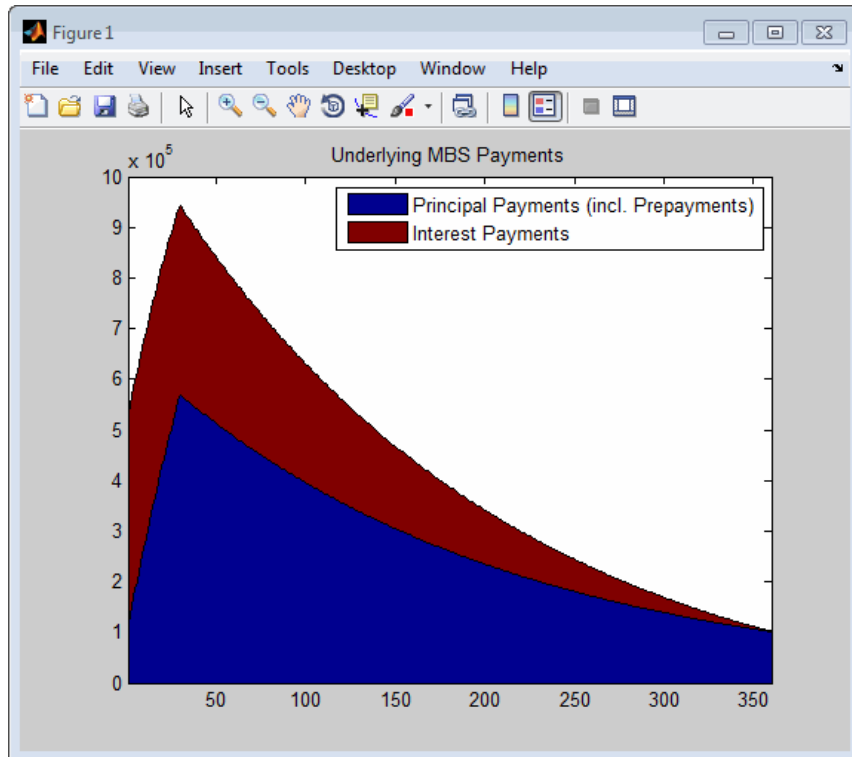
```
principalPayments = UnitPrincipal * principal;  
netInterest = UnitInterest * principal;  
prepayments = UnitPrepayment * principal;  
dates = CFlowDates' + delay;
```

Generate a plot for the underlying MBS payments:

```
area([principalPayments'+prepayments', netInterest'])
```



```
title('Underlying MBS Payments');
legend('Principal Payments (incl. Prepayments)', 'Interest Payments')
```

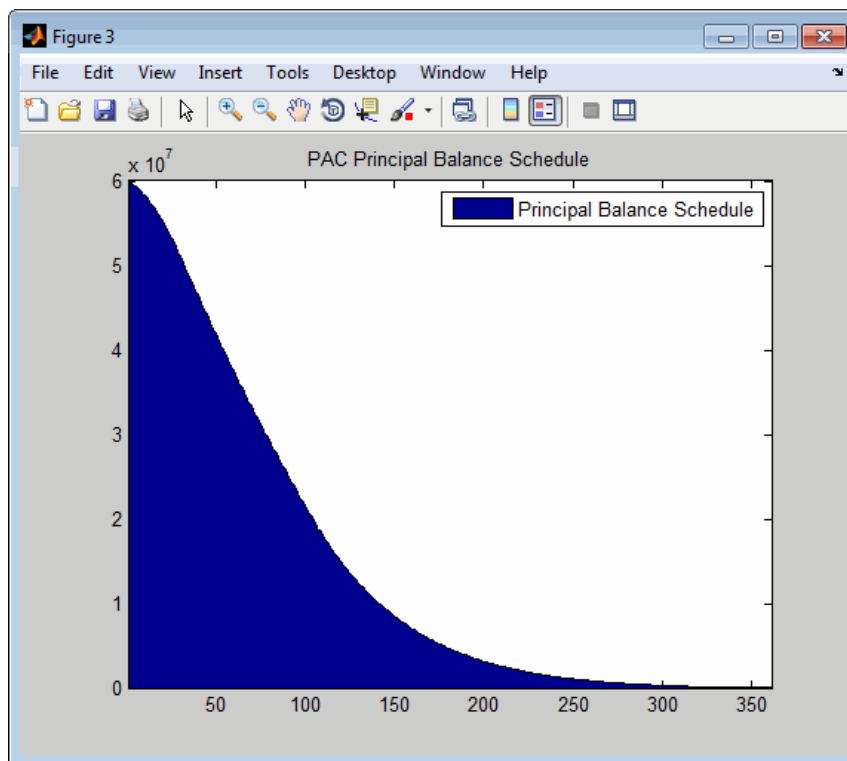


#### Step 4. Calculate the PAC schedule.

```
pacSpeed = [80 300];
[balanceSchedule, pacInitBalance] = ...
cmosched(principal, grossrate, originalTerm, termRemaining, ...
pacSpeed, []);
```

Generate a plot for the PAC principal balance schedule:

```
figure;
area([pacInitBalance'; balanceSchedule'])
title('PAC Principal Balance Schedule');
legend('Principal Balance Schedule');
```

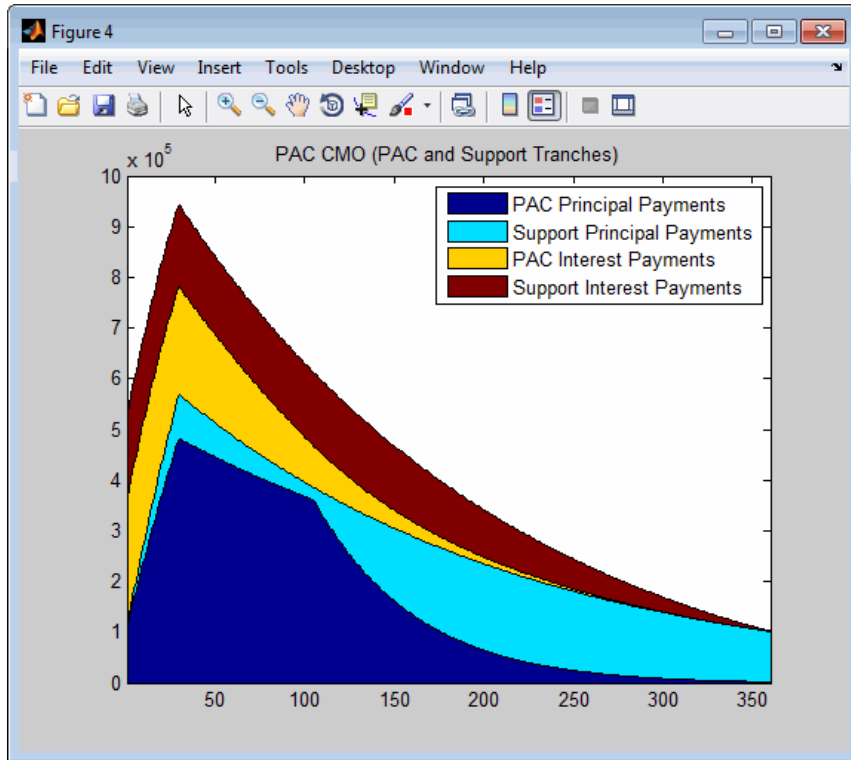


### Step 5. Calculate PAC cash flow.

```
pacTranchePrincipals = [pacInitBalance; principal-pacInitBalance];
pacTrancheCoupons = [0.05; 0.05];
[pacBalances, pacPrincipals, pacInterests] = ...
cmoschedcf(principalPayments+prepayments, ...
pacTranchePrincipals, pacTrancheCoupons, balanceSchedule);
```

Generate a plot for the PAC CMO tranches:

```
figure;
area([pacPrincipals' pacInterests']);
title('PAC CMO (PAC and Support Tranches)');
legend('PAC Principal Payments', 'Support Principal Payments', ...
'PAC Interest Payments', 'Support Interest Payments');
```



**Step 6. Create sequential CMO from the PAC bond.**

```
% CMO tranches, A, B, C, and D
seqTranchePrincipals = ...
[20000000; 20000000; 10000000; pacInitBalance-50000000];
seqTrancheCoupons = [0.05; 0.05; 0.05; 0.05];
```

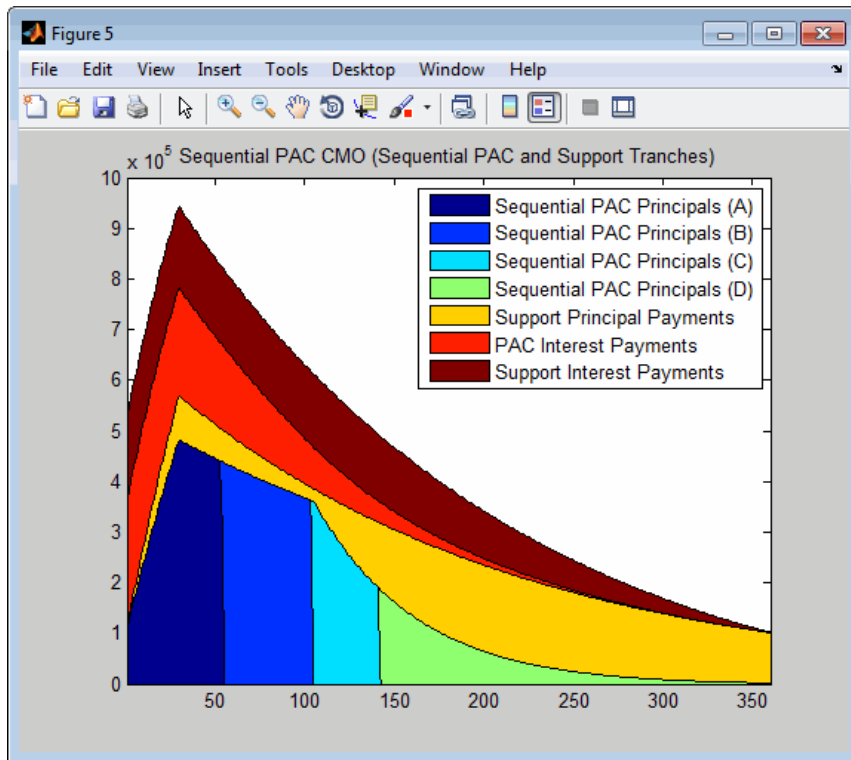
```
seqTrancheCoupons =
    0.0500
    0.0500
    0.0500
    0.0500
```

**Step 7. Calculate cash flows for each tranche.**

```
[seqBalances, seqPrincipals, seqInterests] = ...
cmoseqcf(pacPrincipals(1, :), seqTranchePrincipals, ...
seqTrancheCoupons, false);
```

Generate a plot for the sequential PAC CMO:

```
figure
area([seqPrincipals' pacPrincipals(2, :)' pacInterests']);
title('Sequential PAC CMO (Sequential PAC and Support Tranches)');
legend('Sequential PAC Principals (A)', 'Sequential PAC Principals (B)', ...
'Sequential PAC Principals (C)', 'Sequential PAC Principals (D)', ...
'Support Principal Payments', 'PAC Interest Payments', ...
'Support Interest Payments');
```



**Step 8. Create the discount curve.**

```

CurveSettle = datenum('1-Jan-2011');
ZeroRates = [0.01 0.03 0.10 0.19 0.45 0.81 1.76 2.50 3.18 4.09 4.38]'/100;
CurveTimes = [1/12 3/12 6/12 1 2 3 5 7 10 20 30]';
CurveDates = daysadd(CurveSettle, 360 * CurveTimes, 1);
zeroCurve = intenvset('Rates', ZeroRates, 'StartDates', CurveSettle, ...
'EndDates', CurveDates);

```

```

zeroCurve =

    FinObj: 'RateSpec'
  Compounding: 2
        Disc: [11x1 double]
        Rates: [11x1 double]
    EndTimes: [11x1 double]
  StartTimes: [11x1 double]
    EndDates: [11x1 double]
  StartDates: 734504
ValuationDate: 734504
        Basis: 0
  EndMonthRule: 1

```

### Step 9. Price the CMO cash flows.

The cash flow for the sequential PAC principal A tranche is calculated using the cash flow functions `cfbyzero`, `cfyield`, `cfprice`, and `cfsread`.

```

cflows = seqPrincipals(1, :)+seqInterests(1, :);
cfdates = dates(2:end)';
price1 = cfbyzero(zeroCurve, cflows, cfdates, Settle, 4)
yield = cfyield(cflows, cfdates, price1, Settle, 'Basis', 4)
price2 = cfprice(cflows, cfdates, yield, Settle, 'Basis', 4)
spread = cfsread(zeroCurve, price2, cflows, cfdates, Settle, 'Basis', 4)
WAL = sum(cflows .* yearfrac(Settle, cfdates, 4)) / sum(cflows)

```

```

price1 =

    2.2109e+07

```

```

yield =

```

0.0090

price2 =

2.2109e+07

spread =

-3.4093e-13

WAL =

2.5408

The weighted average life (WAL) for the sequential PAC principal A tranche is 2.54 years.

### See Also

[cmoseqcf](#) | [cmosched](#) | [cmoschedcf](#) | [mbscfamounts](#)

### Related Examples

- “Using Fixed-Rate Mortgage Pool Functions” on page 2-3

### More About

- “Using Collateralized Mortgage Obligations (CMOs)” on page 2-17

# Debt Instruments

---

- “Agency Option-Adjusted Spreads” on page 3-2
- “Treasury Bills Defined” on page 3-7
- “Computing Treasury Bill Price and Yield” on page 3-8
- “Using Zero-Coupon Bonds” on page 3-12
- “Stepped-Coupon Bonds” on page 3-17
- “Term Structure Calculations” on page 3-20

## Agency Option-Adjusted Spreads

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

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

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

where

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

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

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

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



**(Continued)**

<b>Agency OAS Functions</b>	<b>Purpose</b>
agencyoas	Compute the OAS of the callable bond using the Agency OAS model.
agencyprice	Price the callable bond OAS using Agency using the OAS model.

## Computing the Agency OAS for Bonds

To compute the Agency OAS using `agencyoas`, you must provide the zero curve as the input `ZeroData`. You can specify the zero curve in any intervals and with any compounding method. You can do this using Financial Toolbox™ functions `zbtprice` and `zbtyield`. Or, you can use `IRDataCurve` to construct an `IRDataCurve` object, and then use the `getZeroRates` to convert to dates and data for use in the `ZeroData` input.

After creating the `ZeroData` input for `agencyoas`, you can then:

- 1 Assign parameters for `CouponRate`, `Settle`, `Maturity`, `Vol`, `CallDate`, and `Price`.
- 2 Compute the option-adjusted spread using `agencyoas` to derive the OAS output.

If you have the Agency OAS for the callable bond, you can use the OAS value as an input to `agencyprice` to determine the price for a callable bond.

In the following example, the Agency OAS is computed using `agencyoas` for a range of bond prices and the spread of an identically priced noncallable bond is calculated using `bndspread`.

```
% Data
% Bond data -- note that there is only 1 call date
Settle = datenum('20-Jan-2010');
Maturity = datenum('30-Dec-2013');
Coupon = .022;
Vol = .5117;
```

```
CallDate = datenum('30-Dec-2010');
Period = 2;
Basis = 1;
Face = 100;

% Zero Curve data
ZeroTime = [.25 .5 1 2 3 4 5 7 10 20 30]';
ZeroDates = daysadd(Settle,360*ZeroTime,1);
ZeroRates = [.0008 .0017 .0045 .0102 .0169 .0224 .0274 .0347 .0414 .0530 .0740]';
ZeroData = [ZeroDates ZeroRates];
CurveCompounding = 2;
CurveBasis = 1;

Price = 94:104;
OAS = agencyoas(ZeroData, Price', Coupon, Settle,Maturity, Vol, CallDate,'Basis',Basis)
Spread = bndspread(ZeroData, Price', Coupon, Settle, Maturity)
plot(OAS,Price)
hold on
plot(Spread,Price,'r')
xlabel('Spread (bp)')
ylabel('Price')
title('A0AS and Spread for an Agency and Equivalent Noncallable Bond')
legend({'Callable Issue','Noncallable Issue'})

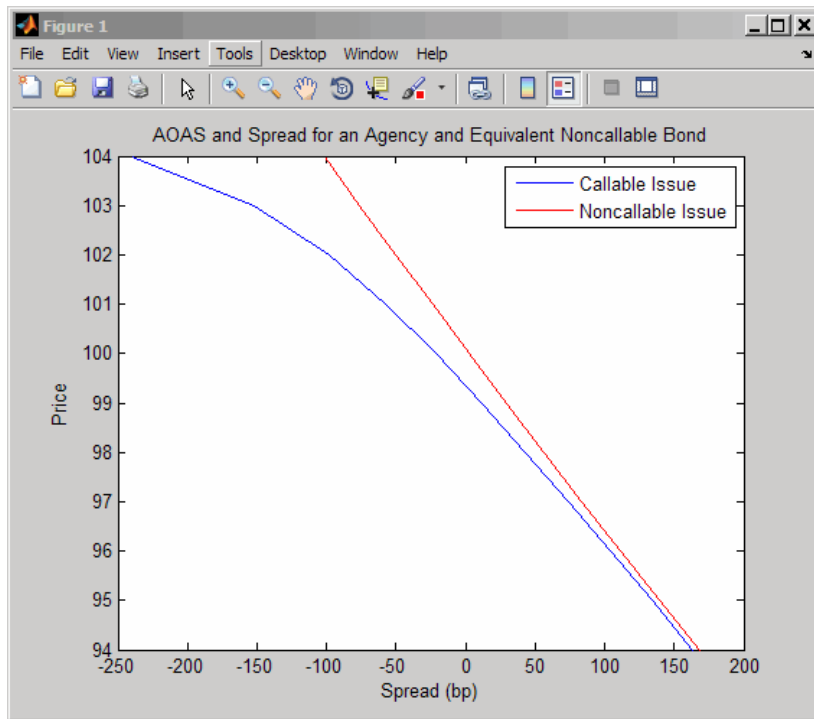
OAS =

    163.4942
    133.7306
    103.8735
     73.7505
     43.1094
     11.5608
    -21.5412
    -57.3869
    -98.5675
   -152.5226
   -239.6462

Spread =
```

168.1412  
139.7047  
111.6123  
83.8561  
56.4286  
29.3227  
2.5314  
-23.9523  
-50.1348  
-76.0226  
-101.6218

The following plot demonstrates as the price increases, the value of the embedded option in the Agency issue increases, and the value of the issue itself does not increase as much as it would for a noncallable bond, illustrating the negative convexity of this issue:



## Treasury Bills Defined

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

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

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

## Computing Treasury Bill Price and Yield

### In this section...

“Introduction” on page 3-8

“Treasury Bill Repurchase Agreements” on page 3-8

“Treasury Bill Yields” on page 3-10

### Introduction

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

### Treasury Bill Functions

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

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

### Treasury Bill Repurchase Agreements

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

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

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

```
BreakevenDiscount =
```

```
0.0167
```

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

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

```
PriceOnPurchaseDate =
```

```
99.5930
```

Next compute the interest due on the repurchase agreement.

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

```
RepoInterest =
```

```
0.1237
```

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

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

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

PriceOnSaleDate =

99.7167

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

**Cash Flows from Repurchase Agreement**

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

**Treasury Bill Yields**

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

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

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

BreakevenDiscount =



0.0167

BEYield =

0.0164

0.0170

MMYield =

0.0162

0.0168

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

## Using Zero-Coupon Bonds

In this section...
“Introduction” on page 3-12
“Measuring Zero-Coupon Bond Function Quality” on page 3-12
“Pricing Treasury Notes” on page 3-13
“Pricing Corporate Bonds” on page 3-15

### Introduction

A zero-coupon bond is a corporate, Treasury, or municipal debt instrument that pays no periodic interest. Typically, the bond is redeemed at maturity for its full face value. It will be a security issued at a discount from its face value, or it may be a coupon bond stripped of its coupons and repackaged as a zero-coupon bond.

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

### Measuring Zero-Coupon Bond Function Quality

Zero-coupon function quality is measured by how consistent the results are with coupon-bearing bonds. Because the zero coupon's yield is bond-equivalent, comparisons with coupon-bearing bonds are possible.

In the textbook case, where time ( $t$ ) is measured continuously and the rate ( $r$ ) is continuously compounded, the value of a zero bond is the principal multiplied by  $e^{-rt}$ . In reality, the rate quoted is continuous and the basis can be variable, requiring a more consistent approach to meet the stricter demands of accurate pricing.

The following two examples

- “Pricing Treasury Notes” on page 3-13
- “Pricing Corporate Bonds” on page 3-15

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.

<b>Cash Flows</b>	<b>Discount Factors</b>	<b>Discounted Cash Flows</b>
-1.2155	1.0000	-1.2155
2.7500	0.9908	2.7246
2.7500	0.9745	2.6799
2.7500	0.9585	2.6359
2.7500	0.9427	2.5925
2.7500	0.9272	2.5499
2.7500	0.9120	2.5080
2.7500	0.8970	2.4668
2.7500	0.8823	2.4263
2.7500	0.8678	2.3864
2.7500	0.8535	2.3472
2.7500	0.8395	2.3086
2.7500	0.8257	2.2706
102.7500	0.8121	83.4451
	Total	112.1263

Compare the quoted price and the calculated price based on zeros.

[QPriceACT PriceofZeros]

ans =

112.1270    112.1263

This example shows that zeroprice can satisfactorily price a Treasury note, a semiannual actual/actual basis bond, as if it were a composed of a series of zero-coupon bonds.

## Pricing Corporate Bonds

You can similarly price a corporate bond, for which there is no corresponding zero-coupon bond, as opposed to a Treasury note, for which corresponding zeros exist. You can create a synthetic zero-coupon bond and arrive at the quoted coupon-bond price when you later sum the zeros.

```
Settle = datenum('02-05-2003');
MaturityCpn = datenum('01-14-2009');
Period = 2;
Basis = 1;
% Quoted yield.
QYield = 0.05974;
% Quoted price.
QPrice30 = 99.382;
CouponRate = 0.05850;
```

Extract cash flow and compute price from the sum of zeros.

```
[CFlows, CDates] = cfamounts(CouponRate, Settle, MaturityCpn, ...
Period, Basis);

Maturity = CDates;
```

Compute the price of the coupon bond identically as a collection of zeros by multiplying the discount factors to the corresponding cash flows.

```
Price30 = CFlows * zeroprice(QYield, Settle, Maturity, Period, ...
Basis)/100;
```

Compare quoted price and calculated price based on zeros.

```
[QPrice30 Price30]
```

```
ans =
```

```
99.3820    99.3828
```

As a test of fidelity, intentionally giving the wrong basis, say actual/actual (Basis = 0) instead of 30/360, gives a price of 99.3972. Such a systematic

error, if recurring in a more complex pricing routine, quickly adds up to large inaccuracies.

In summary, the zero functions in MATLAB software facilitate extraction of present value from virtually any fixed-coupon instrument, up to any period in time.

## Stepped-Coupon Bonds

### In this section...

“Introduction” on page 3-17

“Cash Flows from Stepped-Coupon Bonds” on page 3-17

“Price and Yield of Stepped-Coupon Bonds” on page 3-19

### Introduction

A stepped-coupon bond has a fixed schedule of changing coupon amounts. Like fixed coupon bonds, stepped-coupon bonds could have different periodic payments and accrual bases.

The functions `stepcpnprice` and `stepcpnyield` compute prices and yields of such bonds. An accompanying function `stepcpncfamounts` produces the cash flow schedules pertaining to these bonds.

### Cash Flows from Stepped-Coupon Bonds

Consider a bond that has a schedule of two coupons. Suppose the bond starts out with a 2% coupon that steps up to 4% in 2 years and onward to maturity. Assume that the issue and settlement dates are both March 15, 2003. The bond has a 5 year maturity. Use `stepcpncfamounts` to generate the cash flow schedule and times.

```
Settle      = datenum('15-Mar-2003');
Maturity    = datenum('15-Mar-2008');
ConvDates   = [datenum('15-Mar-2005')];
CouponRates = [0.02, 0.04];

[CFlows, CDates, CTimes] = stepcpncfamounts(Settle, Maturity, ...
ConvDates, CouponRates)
```

Notably, `ConvDates` has 1 less element than `CouponRates` because MATLAB software assumes that the first element of `CouponRates` indicates the coupon schedule between `Settle` (March 15, 2003) and the first element of `ConvDates` (March 15, 2005), shown diagrammatically below.

	Pay 2% from March 15, 2003		Pay 4% from March 15, 2003
Effective 2% on March 15, 2003		Effective 4% on March 15, 2005	

<b>Coupon Dates</b>	<b>Semiannual Coupon Payment</b>
15-Mar-03	0
15-Sep-03	1
15-Mar-04	1
15-Sep-04	1
15-Mar-05	1
15-Sep-05	2
15-Mar-06	2
15-Sep-06	2
15-Mar-07	2
15-Sep-07	2
15-Mar-08	102

The payment on March 15, 2005 is still a 2% coupon. Payment of the 4% coupon starts with the next payment, September 15, 2005. March 15, 2005 is the end of first coupon schedule, not to be confused with the beginning of the second.

In summary, MATLAB takes user input as the end dates of coupon schedules and computes the next coupon dates automatically.

The payment due on settlement (zero in this case) represents the accrued interest due on that day. It is negative if such amount is nonzero. Comparison with `cfamounts` in Financial Toolbox software shows that the two functions operate identically.



## Price and Yield of Stepped-Coupon Bonds

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

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

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

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

Yield = 0.0333;

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

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

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

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

YTM =

0.0333

## Term Structure Calculations

### In this section...

“Introduction” on page 3-20

“Computing Spot and Forward Curves” on page 3-20

“Computing Spreads” on page 3-22

### Introduction

So far, a more formal definition of "yield" and its application has not been developed. In many situations when cash flow is available, discounting factors to the cash flows may not be immediately apparent. In other cases, what is relevant is often a *spread*, the difference between curves (also known as the term structure of spread).

All these calculations require one main ingredient, the Treasury spot, par-yield, or forward curve. Typically, the generation of these curves starts with a series of on-the-run and selected off-the-run issues as inputs.

MATLAB software uses these bonds to find spot rates one at a time, from the shortest maturity onwards, using bootstrap techniques. All cash flows are used to construct the spot curve, and rates between maturities (for these coupons) are interpolated linearly.

### Computing Spot and Forward Curves

For an illustration of how this works, observe the use of `zbtyield` (or equivalently `zbtprice`) on a portfolio of six Treasury bills and bonds.

Bills	Maturity Date	Current Yield
3 month	4/17/03	1.15
6 month	7/17/03	1.18

Notes/Bonds	Coupon	Maturity Date	Current Yield
2 year	1.750	12/31/04	1.68
5 year	3.000	11/15/07	2.97
10 year	4.000	11/15/12	4.01
30 year	5.375	2/15/31	4.92

You can specify prices or yields to the bonds above to infer the spot curve. The function `zbtyield` accepts yields (bond-equivalent yield, to be exact).

To proceed, first assemble the above table into a variable called `Bonds`. The first column contains maturities, the second contains coupons, and the third contains notionals or face values of the bonds. (Note that bills have zero coupons.)

```
Bonds = [datenum('04/17/2003')    0    100;
          datenum('07/17/2003')    0    100;
          datenum('12/31/2004')    0.0175 100;
          datenum('11/15/2007')    0.03   100;
          datenum('11/15/2012')    0.04   100;
          datenum('02/15/2031')    0.05375 100];
```

Then specify the corresponding yields.

```
Yields = [0.0115;
          0.0118;
          0.0168;
          0.0297;
          0.0401;
          0.0492];
```

You are now ready to compute the spot curve for each of these six maturities. The spot curve is based upon a settlement date of January 17, 2003.

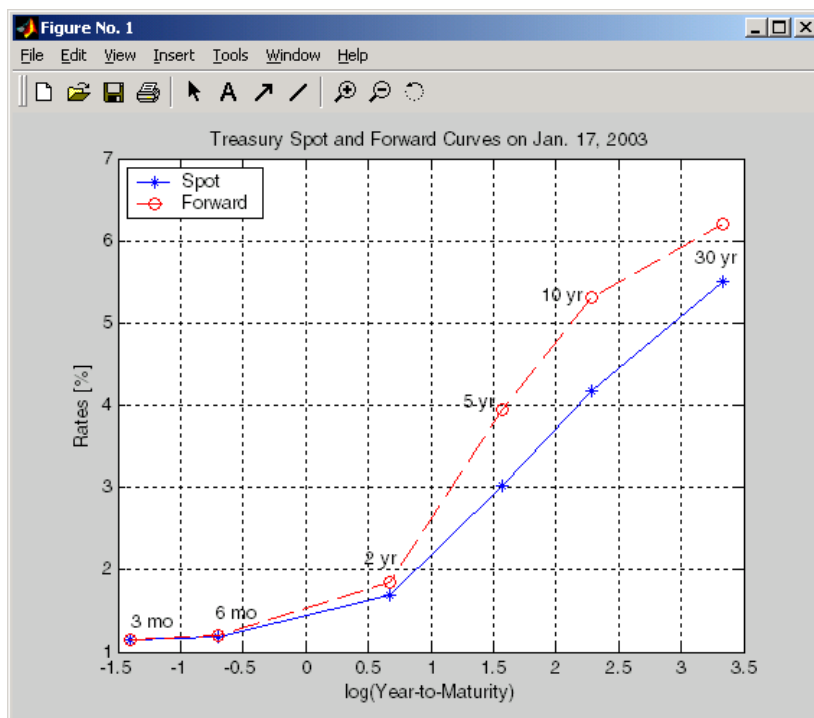
```
Settle = datenum('17-Jan-2003');
[ZeroRates, CurveDates] = zbtyield(Bonds, Yields, Settle)
```

This gets you the Treasury spot curve for the day.

You can compute the forward curve from this spot curve with `zero2fwd`.

```
[ForwardRates, CurveDates] = zero2fwd(ZeroRates, CurveDates, ...
Settle)
```

Here the notion of forward rates refers to rates between the maturity dates shown above, not to a certain period (forward 3 month rates, for example).



## Computing Spreads

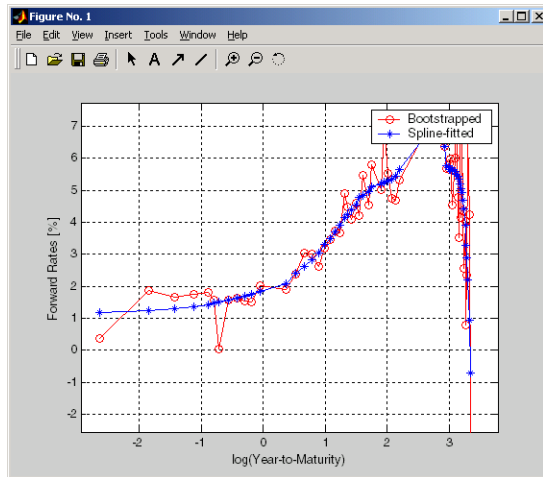
Calculating the spread between specific, fixed forward periods (such as the Treasury-Eurodollar spread) requires an extra step. Interpolate the zero rates (or zero prices, instead) for the corresponding maturities on the interval dates. Then use the interpolated zero rates to deduce the forward rates,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

Additionally, the variety of curve functions (including `zero2fwd`) helps to standardize such calculations. For instance, by making both rates quoted with quarterly compounding and on an actual/360 basis, the resulting spread structure is fully comparable. This avoids the small inconsistency that occurs when directly comparing the bond-equivalent yield of a Treasury bill to the quarterly forward rates implied by Eurodollar futures.

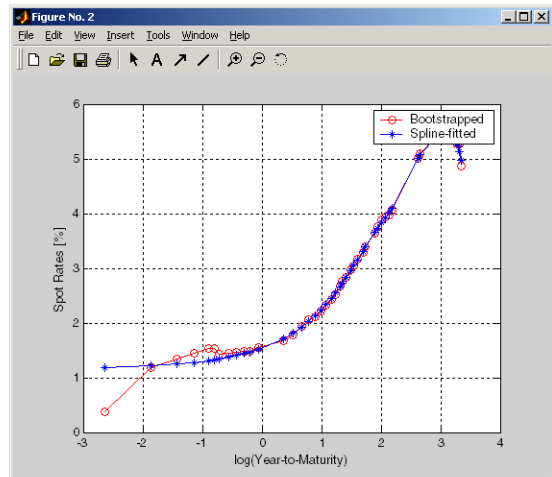
### **Noise in Curve Computations**

When introducing more bonds in constructing curves, noise may become a factor and may need some “smoothing” (with splines, for example); this helps obtain a smoother forward curve.

The following spot and forward curves are constructed from 67 Treasury bonds. The fitted and bootstrapped spot curve (bottom right figure) displays comparable stability. The forward curve (upper-left figure) contains significant noise and shows an improbable forward rate structure. The noise is not necessarily bad; it could uncover trading opportunities for a relative-value approach. Yet, a more balanced approach is desired when the bootstrapped forward curve oscillates this much and contains a negative rate as large as -10% (not shown in the plot because it is outside the limits).



**Implied Forward Curves.**  
The jagged curve comes from direct bootstrapping. The smooth curve shows the effect of smoothing with splines.



**Implied Spot Rate Curves.**  
These curves correspond to the forward curve above.

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

# Derivative Securities

---

- “Interest Rate Swaps” on page 4-2
- “Convertible Bond Valuation” on page 4-10
- “Bond Futures” on page 4-12

## Interest Rate Swaps

In this section...
“Swap Pricing Assumptions” on page 4-2
“Swap Pricing Example” on page 4-3
“Portfolio Hedging” on page 4-8

### Swap Pricing Assumptions

Fixed-Income Toolbox software contains the function `liborfloat2fixed`, which computes a fixed-rate par yield that equates the floating-rate side of a swap to the fixed-rate side. The solver sets the present value of the fixed side to the present value of the floating side without having to line up and compare fixed and floating periods.

### Assumptions on Floating-Rate Input

- Rates are quarterly, for example, that of Eurodollar futures.
- Effective date is the first third Wednesday after the settlement date.
- All delivery dates are spaced 3 months apart.
- All periods start on the third Wednesday of delivery months.
- All periods end on the same dates of delivery months, 3 months after the start dates.
- Accrual basis of floating rates is actual/360.
- Applicable forward rates are estimated by interpolation in months when forward-rate data is not available.

### Assumptions on Fixed-Rate Output

- Design allows you to create a bond of any coupon, basis, or frequency, based upon the floating-rate input.
- The start date is a valuation date, that is, a date when an agreement to enter into a contract by the settlement date is made.



- Settlement can be on or after the start date. If it is after, a forward fixed-rate contract results.
- Effective date is assumed to be the first third Wednesday after settlement, the same date as that of the floating rate.
- The end date of the bond is a designated number of years away, on the same day and month as the effective date.
- Coupon payments occur on anniversary dates. The frequency is determined by the period of the bond.
- Fixed rates are not interpolated. A fixed-rate bond of the same present value as that of the floating-rate payments is created.

## Swap Pricing Example

This example shows the use of the functions in computing the fixed rate applicable to a series of 2-, 5-, and 10-year swaps based on Eurodollar market data. According to the Chicago Mercantile Exchange (<http://www.cme.com>), Eurodollar data on Friday, October 11, 2002, was as shown in the following table.

---

**Note** This example illustrates swap calculations in MATLAB software. Timing of the data set used was not rigorously examined and was assumed to be the proxy for the swap rate reported on October 11, 2002.

---

### Eurodollar Data on Friday, October 11, 2002

Month	Year	Settle
10	2002	98.21
11	2002	98.26
12	2002	98.3
1	2003	98.3
2	2003	98.31
3	2003	98.275
6	2003	98.12

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

<b>Month</b>	<b>Year</b>	<b>Settle</b>
9	2003	97.87
12	2003	97.575
3	2004	97.26
6	2004	96.98
9	2004	96.745
12	2004	96.515
3	2005	96.33
6	2005	96.135
9	2005	95.955
12	2005	95.78
3	2006	95.63
6	2006	95.465
9	2006	95.315
12	2006	95.16
3	2007	95.025
6	2007	94.88
9	2007	94.74
12	2007	94.595
3	2008	94.48
6	2008	94.375
9	2008	94.28
12	2008	94.185
3	2009	94.1
6	2009	94.005
9	2009	93.925
12	2009	93.865

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

Month	Year	Settle
3	2010	93.82
6	2010	93.755
9	2010	93.7
12	2010	93.645
3	2011	93.61
6	2011	93.56
9	2011	93.515
12	2011	93.47
3	2012	93.445
6	2012	93.41
9	2012	93.39

Using this data, you can compute 1-, 2-, 3-, 4-, 5-, 7-, and 10-year swap rates with the toolbox function `liborfloat2fixed`. The function requires you to input only Eurodollar data, the settlement date, and tenor of the swap. MATLAB software then performs the required computations.

To illustrate how this function works, first load the data contained in the supplied Excel® worksheet `EDdata.xls`.

```
[EDRawData, textdata] = xlsread('EDdata.xls');
```

Extract the month from the first column and the year from the second column. The rate used as proxy is the arithmetic average of rates on opening and closing.

```
Month = EDRawData(:,1);
Year = EDRawData(:,2);
IMMData = (EDRawData(:,4)+EDRawData(:,6))/2;
EDFutData = [Month, Year, IMMData];
```

Next, input the current date.

```
Settle = datenum('11-Oct-2002');
```

To compute for the 2 year swap rate, set the tenor to 2.

```
Tenor = 2;
```

Finally, compute the swap rate with `liborfloat2fixed`.

```
[FixedSpec, ForwardDates, ForwardRates] = ...  
liborfloat2fixed(EDFutData, Settle, Tenor)
```

MATLAB returns a par-swap rate of 2.23% using the default setting (quarterly compounding and 30/360 accrual), and forward dates and rates data (quarterly compounded), comparable to 2.17% of Friday's average broker data in Table H15 of *Federal Reserve Statistical Release* (<http://www.federalreserve.gov/releases/h15/update/>).

```
FixedSpec =
```

```
    Coupon: 0.0223  
    Settle: '16-Oct-2002'  
    Maturity: '16-Oct-2004'  
    Period: 4  
    Basis: 1
```

```
ForwardDates =
```

```
    731505  
    731596  
    731687  
    731778  
    731869  
    731967  
    732058  
    732149
```

```
ForwardRates =
```

```
    0.0178  
    0.0168
```

```

0.0171
0.0189
0.0216
0.0250
0.0280
0.0306

```

In the `FixedSpec` output, note that the swap rate actually goes forward from the third Wednesday of October 2002 (October 16, 2002), 5 days after the original `Settle` input (October 11, 2002). This, however, is still the best proxy for the swap rate on `Settle`, as the assumption merely starts the swap's effective period and does not affect its valuation method or its length.

The correction suggested by Hull and White improves the result by turning on convexity adjustment as part of the input to `liborfloat2fixed`. (See Hull, J., *Options, Futures, and Other Derivatives*, 4th Edition, Prentice-Hall, 2000.) For a long swap, for example, 5 years or more, this correction could prove to be large.

The adjustment requires additional parameters:

- `StartDate`, which you make the same as `Settle` (the default) by providing an empty matrix `[]` as input.
- `ConvexAdj` to tell `liborfloat2fixed` to perform the adjustment.
- `RateParam`, which provides the parameters `a` and `S` as input to the Hull-White short rate process.
- Optional parameters `InArrears` and `Sigma`, for which you can use empty matrices `[]` to accept the MATLAB defaults.
- `FixedCompound`, with which you can facilitate comparison with values cited in Table H15 of *Federal Reserve Statistical Release* by turning the default quarterly compounding into semiannual compounding, with the (default) basis of 30/360.

```

StartDate = [];
Interpolation = [];
ConvexAdj = 1;
RateParam = [0.03; 0.017];
FixedCompound = 2;

```

```
[FixedSpec, ForwardDaates, ForwardRates] = ...
liborfloat2fixed(EDFutData, Settle, Tenor, StartDate, ...
Interpolation, ConvexAdj, RateParam, [], [], FixedCompound)
```

This returns 2.21% as the 2-year swap rate, quite close to the reported swap rate for that date.

Analogously, the following table summarizes the solutions for 1-, 3-, 5-, 7-, and 10-year swap rates (convexity-adjusted and unadjusted).

**Calculated and Market Average Data of Swap Rates on Friday, October 11, 2002**

Swap Length (Years)	Unadjusted	Adjusted	Table H15	Adjusted Error (Basis Points)
1	1.80%	1.79%	1.80%	-1
2	2.24%	2.21%	2.22%	-1
3	2.70%	2.66%	2.66%	0
4	3.12%	3.03%	3.04%	-1
5	3.50%	3.37%	3.36%	+1
7	4.16%	3.92%	3.89%	+3
10	4.87%	4.42%	4.39%	+3

**Portfolio Hedging**

You can use these results further, such as for hedging a portfolio. The `liborduration` function provides a duration-hedging capability. You can isolate assets (or liabilities) from interest-rate risk exposure with a swap arrangement.

Suppose you own a bond with these characteristics:

- \$100 million face value
- 7% coupon paid semiannually

- 5% yield to maturity
- Settlement on October 11, 2002
- Maturity on January 15, 2010
- Interest accruing on an actual/365 basis

Use of the `bnddury` function from Financial Toolbox software shows a modified duration of 5.6806 years.

To immunize this asset, you can enter into a pay-fixed swap, specifically a swap in the amount of notional principal ( $Ns$ ) such that  $Ns * SwapDuration + \$100M * 5.6806 = 0$  (or  $Ns = -100 * 5.6806 / SwapDuration$ ).

Suppose again, you choose to use a 5-, 7-, or 10-year swap (3.37%, 3.92%, and 4.42% from the previous table) as your hedging tool.

```
SwapFixRate = [0.0337; 0.0392; 0.0442];
Tenor = [5; 7; 10];
Settle = '11-Oct-2002';
PayFixDuration = liborduration(SwapFixRate, Tenor, Settle)
```

This gives a duration of -3.6835, -4.7307, and -6.0661 years for 5-, 7-, and 10-year swaps. The corresponding notional amount is computed by

```
Ns = -100*5.6806./PayFixDuration
```

```
Ns =
```

```
154.2163
120.0786
93.6443
```

The notional amount entered in pay-fixed side of the swap instantaneously immunizes the portfolio.

## Convertible Bond Valuation

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

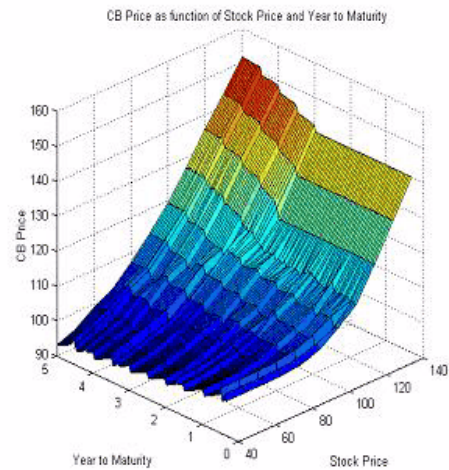
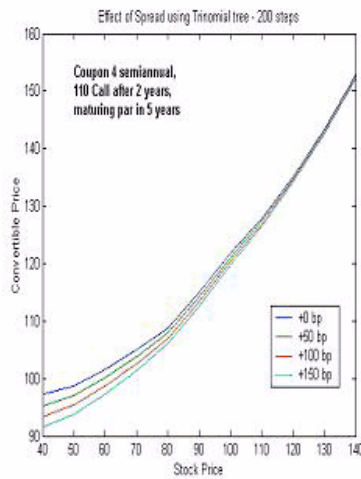
Fixed-Income Toolbox software uses a binomial and trinomial tree approach (cbprice) to value convertible bonds. The value of a convertible bond is determined by the uncertainty of the related stock. Once the stock evolution is modeled, backward discounting is computed.

The last column of such trees provides the data to decide which is more profitable: the debt notional (plus interest, if any) or the equivalent number of shares per the notional.

Where debt prevails, the toolbox discounts backward with the risk-free rate plus the spread reflecting the credit risk of the issuer. Where stock prevails, the toolbox discounts with the risk-free rate. The intrinsic value of a convertible bond is the sum of the (probability-adjusted) debt and stock portions from the last node. This is compared to current stock price, to see if voluntary or forced conversion may take place. Otherwise, its value is the intrinsic value. From here, the same discounting process is repeated after adjusting debt portion to be equal to 0 if any conversion takes place. Dividends and coupons are handled discretely, at the date they occur.

The approach is equivalent to solving a one-dimensional partial differential equation such as one described by Tsiveriotis and Fernandes. (See Tsiveriotis, K. and C. Fernandes (1998), "Valuing Convertible Bonds with Credit Risk," *The Journal of Fixed Income*, 8 (3), 95-102.) Using the same example of bond specifications that they use (4% annual coupon, payable twice a year, callable after 2 years at 110, and redeemable at par in 5 years), the toolbox gives results like theirs.





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

The change of curvature located at the end of the second year is due to the activation of the embedded (soft) call feature (visible on the surface plot in the right figure).

Finally, there is the flat section when time is nearing expiration and share prices are high, indicating a delta of unity, a characteristic of in-the-money equity options embedded in a bond.

## Bond Futures

**In this section...**

“Supported Bond Futures” on page 4-12

“Example Analysis of Bond Futures” on page 4-14

“Managing Interest-Rate Risk with Bond Futures” on page 4-16

### Supported Bond Futures

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

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

- 30-year U.S. Treasury bond
- 10-year U.S. Treasury bond
- 5-year U.S. Treasury bond
- 2-year U.S. Treasury bond

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

$$\text{InvoicePrice} = \text{FutPrice} \times CF + AI$$

where:

*FutPrice* is the price of the bond future.

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

*AI* is the accrued interest.

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

Fixed-Income Toolbox software supports the following bond futures:

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

The functions supporting all bond futures are:

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

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

<b>Function</b>	<b>Purpose</b>
tfutbyprice	Calculates future prices of Treasury bonds given the spot price.
tfutbyyield	Calculates future prices of Treasury bonds given current yield.
tfutimrepo	Calculates implied repo rates for the Treasury bond future given price.
tfutpricebyrepo	Calculates implied repo rates given the Treasury bond future price.
tfutyieldbyrepo	Calculates implied repo rates given the Treasury bond future yield.

### **Example Analysis of Bond Futures**

The following example demonstrates analyzing German Euro-Bund futures traded on Eurex. However, `convfactor`, `bndfutprice`, and `bndfutimrepo` apply to bond futures in the U.S., U.K., Germany, and Japan. The workflow for this analysis is:

- 1** Calculate bond conversion factors.
- 2** Calculate implied repo rates to find the CTD bond.
- 3** Price the bond future using the term implied repo rate.

### **Calculating Bond Conversion Factors**

Use conversion factors to normalize the price of a particular bond for delivery in a futures contract. When using conversion factors, the assumption is that a bond for delivery has a 6% coupon. Use `convfactor` to calculate conversion factors for all bond futures from the U.S., Germany, Japan, and U.K.

For example, conversion factors for Euro-Bund futures on Eurex are listed at [www.eurexchange.com](http://www.eurexchange.com). The delivery date for Euro-Bund futures is the 10th day of the month, as opposed to bond futures in the U.S., where the short position has the option of choosing when to deliver the bond.

For the 4% bond, compute the conversion factor with:

```
CF1 = convfactor('10-Sep-2009','04-Jul-2018',.04,.06,3)
CF1 =
```

```
0.8659
```

This syntax for `convfactor` works fine for bonds with standard coupon periods. However, some deliverable bonds have long or short first coupon periods. Compute the conversion factors for such bonds using the optional input parameters `StartDate` and `FirstCouponDate`. Specify all optional input arguments for `convfactor` as parameter/value pairs:

```
CF2 = convfactor('10-Sep-2009','04-Jan-2019',.0375,'Convention',3,'startdate',...
datenum('14-Nov-2008'))
CF2 =
```

```
0.8426
```

## Calculating Implied Repo Rates to Find the CTD Bond

To determine the availability of the cheapest bond for deliverable bonds against a futures contract, compute the implied repo rate for each bond. The bond with the highest repo rate is the cheapest because it has the lowest initial value, thus yielding a higher return, provided you deliver it with the stated futures price. Use `bndfutimprepo` to calculate repo rates:

```
% Bond Properties
CouponRate = [.0425;.0375;.035];
Maturity = [datenum('04-Jul-2018');datenum('04-Jan-2019');datenum('04-Jul-2019')];
CF = [0.882668;0.842556;0.818193];
Price = [105.00;100.89;98.69];

% Futures Properties
FutSettle = '09-Jun-2009';
FutPrice = 118.54;
Delivery = '10-Sep-2009';

% Note that the default for BDNFUTIMPREPO is for the bonds to be
% semi-annual with a day count basis of 0. Since these are German
% bonds, we need to have a Basis of 8 and a Period of 1
ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle, Delivery, CF, ...
CouponRate, Maturity, 'Basis',8, 'Period',1)
```

```
ImpRepo =  
  
    0.0261  
   -0.0022  
   -0.0315
```

### Pricing Bond Futures Using the Term Implied Repo Rate

Use `bndfutprice` to perform price calculations for all bond futures from the U.S., Germany, Japan, and U.K. To price the bond, given a term repo rate:

```
% Assume a term repo rate of .0091;  
RepoRate = .0091;  
[FutPrice,AccrInt] = bndfutprice(RepoRate, Price(1), FutSettle,...  
Delivery, CF(1), CouponRate(1), Maturity(1),...  
'Basis',8,'Period',1)  
  
FutPrice =  
  
    118.0126  
  
AccrInt =  
  
    0.7918
```

### Managing Interest-Rate Risk with Bond Futures

The Present Value of a Basis Point (PVBP) is used to manage interest-rate risk. PVBP is a measure that quantifies the change in price of a bond given a one-basis point shift in interest rates. The PVBP of a bond is computed with the following:

$$PVBP_{Bond} = \frac{Duration \times MarketValue}{100}$$

The PVBP of a bond futures contract can be computed with the following:

$$PVBP_{Futures} = \frac{PVBP_{CTDBond}}{CTDConversionFactor}$$

Use `bnddurp` and `bnddury` from Financial Toolbox software to compute the modified durations of CTD bonds. For more information, see the Fixed-Income Toolbox demo “Managing Interest Rate Risk with Bond Futures” at:

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





# Credit Derivatives

---

- “Credit Default Swap (CDS)” on page 5-2
- “Credit Default Swap Option” on page 5-17

## Credit Default Swap (CDS)

A credit default swap (CDS) is a contract that protects against losses resulting from credit defaults. The transaction involves two parties, the protection buyer and the protection seller, and also a reference entity, usually a bond. The protection buyer pays a stream of premiums to the protection seller, who in exchange offers to compensate the buyer for the loss in the bond's value if a credit event occurs. The stream of premiums is called the premium leg, and the compensation when a credit event occurs is called the protection leg. Credit events usually include situations in which the bond issuer goes bankrupt, misses coupon payments, or enters a restructuring process. Fixed-Income Toolbox software supports:

### CDS Functions

Function	Purpose
<code>cdsbootstrap</code>	Compute default probability parameters from CDS market quotes.
<code>cdsspread</code>	Compute breakeven spreads for the CDS contracts.
<code>cdsprice</code>	Compute the price for the CDS contracts.

### Bootstrapping a Default Probability Curve

In a typical workflow, pricing a new CDS contract involves first estimating a default probability term structure using `cdsbootstrap`. This requires market quotes of existing CDS contracts, or quotes of CDS indices (e.g., iTraxx). The estimated default probability curve is then used as input to `cdsspread` or `cdsprice`. If the default probability information is already known, `cdsbootstrap` can be bypassed and `cdsspread` or `cdsprice` can be called directly.

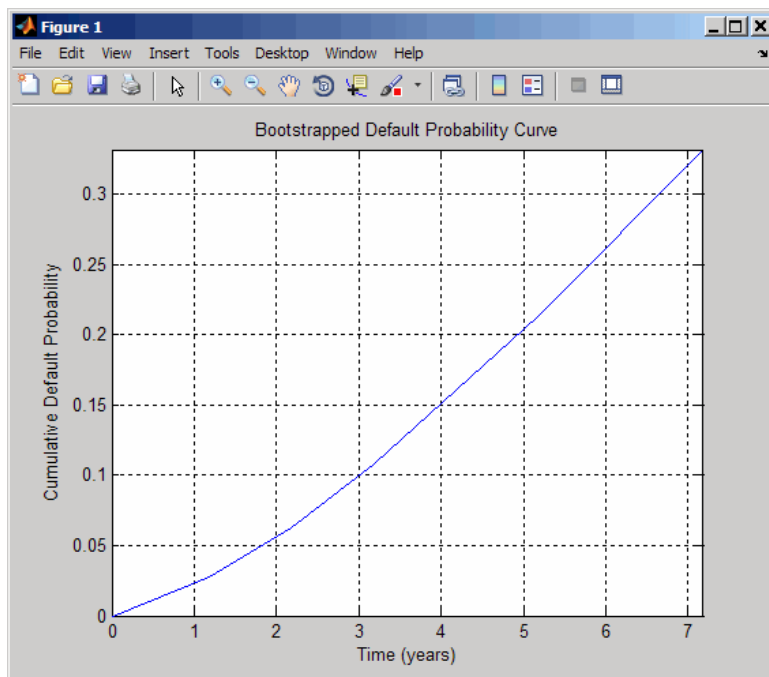
The market information in this example is provided in the form of running spreads of CDS contracts maturing on the CDS standard payment dates closest to 1, 2, 3, 5, and 7 years from the valuation date.

```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10', '20-Sep-11', '20-Sep-12', '20-Sep-14', ...
```

```
'20-Sep-16'});  
MarketSpreads = [140 175 210 265 310]';  
MarketData = [MarketDates MarketSpreads];  
ZeroDates = datenum({'17-Jan-10', '17-Jul-10', '17-Jul-11', '17-Jul-12', ...  
    '17-Jul-13', '17-Jul-14'});  
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;  
ZeroData = [ZeroDates ZeroRates];  
  
[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle);
```

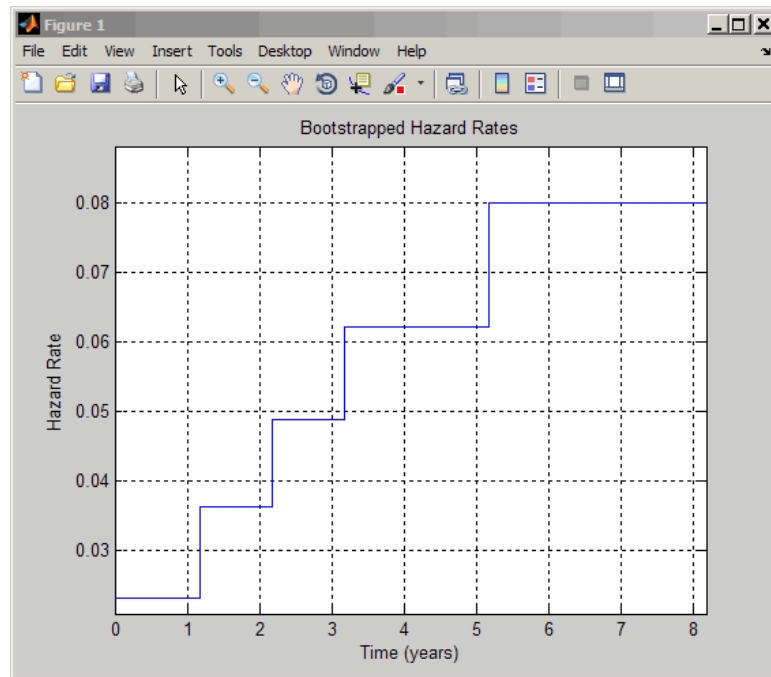
The bootstrapped default probability curve is plotted against time, in years, from the valuation date.

```
ProbTimes = yearfrac(Settle,ProbData(:,1));  
figure  
plot([0; ProbTimes],[0; ProbData(:,2)])  
grid on  
axis([0 ProbTimes(end,1) 0 ProbData(end,2)])  
xlabel('Time (years)')  
ylabel('Cumulative Default Probability')  
title('Bootstrapped Default Probability Curve')
```



The associated hazard rates are returned as an optional output. The convention is that the first hazard rate applies from the settlement date to the first market date, the second hazard rate from the first to the second market date, etc., and the last hazard rate applies from the second-to-last market date onwards. The following plot displays the bootstrapped hazard rates, plotted against time, in years, from the valuation date:

```
HazTimes = yearfrac(Settle,HazData(:,1));
figure
stairs([0; HazTimes(1:end-1,1); HazTimes(end,1)+1],...
[HazData(:,2);HazData(end,2)])
grid on
axis([0 HazTimes(end,1)+1 0.9*HazData(1,2) 1.1*HazData(end,2)])
xlabel('Time (years)')
ylabel('Hazard Rate')
title('Bootstrapped Hazard Rates')
```



## Finding the Breakeven Spread for a New CDS Contract

The breakeven, or running, spread is the premium a protection buyer must pay, with no upfront payments involved, to receive protection for credit events associated to a given reference entity. Spreads are expressed in basis points (bp). There is a notional amount associated to the CDS contract to determine the monetary amounts of the premium payments.

New quotes for CDS contracts can be obtained with `cdsspread`. After obtaining a default probability curve using `cdsbootstrap`, you get quotes that are consistent with current market conditions.

In this example, instead of standard CDS payment dates, define new maturity dates. Using the period from 3 and 5 years (CDS standard dates), maturities are defined within this range spaced at quarterly intervals (measuring time from the valuation date):

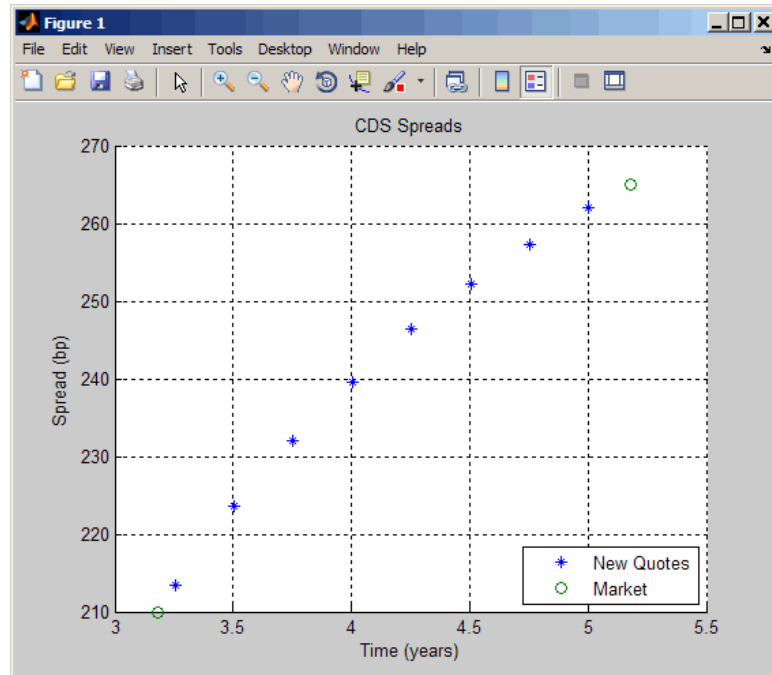
```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10', '20-Sep-11', '20-Sep-12', '20-Sep-14', ...
    '20-Sep-16'});
MarketSpreads = [140 175 210 265 310]';
MarketData = [MarketDates MarketSpreads];
ZeroDates = datenum({'17-Jan-10', '17-Jul-10', '17-Jul-11', '17-Jul-12', ...
    '17-Jul-13', '17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle);

Maturity1 = datestr(daysadd('17-Jul-09',360*(3.25:0.25:5),1));
Spread1 = cdsspread(ZeroData,ProbData,Settle,Maturity1);

figure
scatter(yearfrac(Settle,Maturity1),Spread1,'*')
hold on
scatter(yearfrac(Settle,MarketData(3:4,1)),MarketData(3:4,2))
hold off
grid on
xlabel('Time (years)')
ylabel('Spread (bp)')
title('CDS Spreads')
legend('New Quotes','Market','location','SouthEast')
```

This plot displays the resulting spreads:

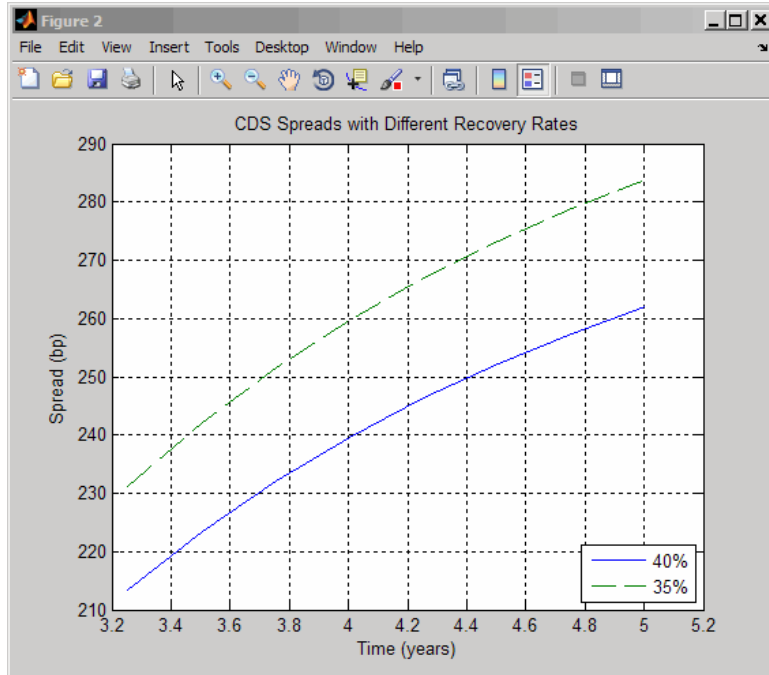


To evaluate the effect of the recovery rate parameter, instead of 40% (default value), use a recovery rate of 35%:

```
Spread1Rec35 = cdsspread(ZeroData,ProbData,Settle,Maturity1,...
'RecoveryRate',0.35);
```

```
figure
plot(yearfrac(Settle,Maturity1),Spread1,...
yearfrac(Settle,Maturity1),Spread1Rec35,'--')
grid on
xlabel('Time (years)')
ylabel('Spread (bp)')
title('CDS Spreads with Different Recovery Rates')
legend('40%','35%','location','SouthEast')
```

The resulting plot shows that smaller recovery rates produce higher premia, as expected, since in the event of default, the protection payments will be higher:



## Valuing an Existing CDS Contract

The current value, or mark-to-market, of an existing CDS contract is the amount of money the contract holder would receive (if positive) or pay (if negative) to unwind this contract. The upfront of the contract is the current value expressed as a fraction of the notional amount of the contract, and it is commonly used to quote market values.

The value of existing CDS contracts is obtained with `cdsprice`. By default, `cdsprice` treats contracts as long positions. Whether a contract position is long or short is determined from a protection standpoint, that is, long means protection was bought, and short means protection was sold. In the following example, an existing CDS contract pays a premium that is lower than current market conditions. The price is positive, as expected, since it would be more costly to buy the same type of protection today.

```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10', '20-Sep-11', '20-Sep-12', '20-Sep-14', ...
```



```

'20-Sep-16'});
MarketSpreads = [140 175 210 265 310]';
MarketData = [MarketDates MarketSpreads];

ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
'17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle);

Maturity2 = '20-Sep-2012';
Spread2 = 196;

[Price,AccPrem,PaymentDates,PaymentTimes,PaymentCF] = cdsprice(ZeroData,...
ProbData,Settle,Maturity2,Spread2);

fprintf('Dirty Price: %8.2f\n',Price);
fprintf('Accrued Premium: %8.2f\n',AccPrem);
fprintf('Clean Price: %8.2f\n',Price-AccPrem);
fprintf('\nPayment Schedule:\n\n');
fprintf('Date \t\t Time Frac \t Amount\n');
for k = 1:length(PaymentDates)
    fprintf('%s \t %5.4f \t %8.2f\n',datestr(PaymentDates(k)),...
    PaymentTimes(k),PaymentCF(k));
end

```

This resulting payment schedule is:

```

Dirty Price: 41630.75
Accrued Premium: 15244.44
Clean Price: 26386.30

```

Payment Schedule:

Date	Time Frac	Amount
20-Sep-2009	0.1806	35388.89
20-Dec-2009	0.2528	49544.44
20-Mar-2010	0.2500	49000.00
20-Jun-2010	0.2556	50088.89

20-Sep-2010	0.2556	50088.89
20-Dec-2010	0.2528	49544.44
20-Mar-2011	0.2500	49000.00
20-Jun-2011	0.2556	50088.89
20-Sep-2011	0.2556	50088.89
20-Dec-2011	0.2528	49544.44
20-Mar-2012	0.2528	49544.44
20-Jun-2012	0.2556	50088.89
20-Sep-2012	0.2556	50088.89

Additionally, you can use `cdsprice` to value a portfolio of CDS contracts. In the following example, a simple hedged position with two vanilla CDS contracts, one long, one short, with slightly different spreads is priced in a single call and the value of the portfolio is the sum of the returned prices:

```
[Price2,AccPrem2] = cdsprice(ZeroData,ProbData,Settle,...
    repmat(datenum(Maturity2),2,1),[Spread2;Spread2+3],...
    'Notional',[1e7; -1e7]);

fprintf('Contract \t Dirty Price \t Acc Premium \t Clean Price\n');
fprintf('    Long \t $ %9.2f \t $ %9.2f \t $ %9.2f \t\n',...
    Price2(1), AccPrem2(1), Price2(1) - AccPrem2(1));
fprintf('    Short \t $ %8.2f \t $ %8.2f \t $ %8.2f \t\n',...
    Price2(2), AccPrem2(2), Price2(2) - AccPrem2(2));
fprintf('Mark-to-market of hedged position: $ %8.2f\n',sum(Price2));
```

This resulting value of the portfolio is:

Contract	Dirty Price	Acc Premium	Clean Price
Long	\$ 41630.75	\$ 15244.44	\$ 26386.30
Short	\$ -32709.87	\$ -15477.78	\$ -17232.10
Mark-to-market of hedged position:			\$ 8920.87

## Converting from Running to Upfront and Vice Versa

A CDS market quote is given in terms of a standard spread (usually 100 bp or 500 bp) and an upfront payment, or in terms of an equivalent running or breakeven spread, with no upfront payment. The functions `cdsbootstrap`, `cdsspread`, and `cdsprice` perform upfront to running or running to upfront conversions.

For example, to convert the market quotes to upfront quotes for a standard spread of 100 bp:

```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10','20-Sep-11','20-Sep-12','20-Sep-14',...
'20-Sep-16'});
MarketSpreads = [140 175 210 265 310]';
MarketData = [MarketDates MarketSpreads];

ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
'17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle);

Maturity3 = MarketData(:,1);
Spread3Run = MarketData(:,2);
Spread3Std = 100*ones(size(Maturity3));
Price3 = cdsprice(ZeroData,ProbData,Settle,Maturity3,Spread3Std);
Upfront3 = Price3/10000000; % Standard notional of 10MM
display(Upfront3);
```

This resulting value is:

```
Upfront3 =

    0.0047
    0.0158
    0.0327
    0.0737
    0.1182
```

The conversion can be reversed to convert upfront quotes to market quotes:

```
ProbData3Upf = cdsbootstrap(ZeroData,[Maturity3 Upfront3 Spread3Std],Settle);
Spread3RunFromUpf = cdsspread(ZeroData,ProbData3Upf,Settle,Maturity3);
display([Spread3Run Spread3RunFromUpf]);
```

Comparing the results of this conversion to the original market spread demonstrates the reversal:

```
ans =

    140.0000    140.0000
    175.0000    175.0000
    210.0000    210.0000
    265.0000    265.0000
    310.0000    310.0000
```

Under the flat-hazard rate (FHR) quoting convention, a single market quote is used to calibrate a probability curve. This convention yields a single point in the probability curve, and a single hazard rate value. For example, assume a 4-year (standard dates) CDS contract with a current FHR-based running spread of 550 bp needs conversion to a CDS contract with a standard spread of 500 bp:

```
Maturity4 = datenum('20-Sep-13');
Spread4Run = 550;
ProbData4Run = cdsbootstrap(ZeroData,[Maturity4 Spread4Run],Settle);
Spread4Std = 500;
Price4 = cdsprice(ZeroData,ProbData4Run,Settle,Maturity4,Spread4Std);
Upfront4 = Price4/10000000;
fprintf('A running spread of %5.2f is equivalent to\n',Spread4Run);
fprintf('    a standard spread of %5.2f with an upfront of %8.7f\n',...
        Spread4Std,Upfront4);
```

```
A running spread of 550.00 is equivalent to
    a standard spread of 500.00 with an upfront of 0.0167583
```

To reverse the conversion:

```
ProbData4Upf = cdsbootstrap(ZeroData,[Maturity4 Upfront4 Spread4Std],Settle);
Spread4RunFromUpf = cdsspread(ZeroData,ProbData4Upf,Settle,Maturity4);
fprintf('A standard spread of %5.2f with an upfront of %8.7f\n',...
        Spread4Std,Upfront4);
fprintf('    is equivalent to a running spread of %5.2f\n',Spread4RunFromUpf);
```

```
A standard spread of 500.00 with an upfront of 0.0167583
    is equivalent to a running spread of 550.00
```

As discussed in Beumee et. al., 2009 (see “Credit Derivatives” on page B-5), the FHR approach is a quoting convention only, and leads to quotes

inconsistent with market data. For example, calculating the upfront for the 3-year (standard dates) CDS contract with a standard spread of 100 bp using the FHR approach and comparing the results to the upfront amounts previously calculated, demonstrates that the FHR-based approach yields a different upfront amount:

```
Maturity5 = MarketData(3,1);
Spread5Run = MarketData(3,2);
ProbData5Run = cdsbootstrap(ZeroData,[Maturity5 Spread5Run],Settle);
Spread5Std = 100;
Price5 = cdsprice(ZeroData,ProbData5Run,Settle,Maturity5,Spread5Std);
Upfront5 = Price5/10000000;
fprintf('Relative error of FHR-based upfront amount: %3.1f%%\n',...
        ((Upfront5-Upfront3(3))/Upfront3(3))*100);
```

Relative error of FHR-based upfront amount: -0.8%

## Bootstrapping from Inverted Market Curves

The following two examples demonstrate the behavior of bootstrapping with inverted CDS market curves, that is, market quotes with higher spreads for short-term CDS contracts. The first example is handled normally by `cdsbootstrap`:

```
Settle = '17-Jul-2009';
MarketDates = datenum({'20-Sep-10','20-Sep-11','20-Sep-12','20-Sep-14',...
    '20-Sep-16'});

ZeroDates = datenum({'17-Jan-10','17-Jul-10','17-Jul-11','17-Jul-12',...
    '17-Jul-13','17-Jul-14'});
ZeroRates = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
ZeroData = [ZeroDates ZeroRates];

MarketSpreadsInv1 = [750 650 550 500 450]';
MarketDataInv1 = [MarketDates MarketSpreadsInv1];
[ProbDataInv1,HazDataInv1] = cdsbootstrap(ZeroData,MarketDataInv1,Settle);
```

In the second example, `cdsbootstrap` generates a warning:

```
MarketSpreadsInv2 = [800 550 400 250 100]';
MarketDataInv2 = [MarketDates MarketSpreadsInv2];
```

```
[ProbDataInv2,HazDataInv2] = cdsbootstrap(ZeroData,MarketDataInv2,Settle);
```

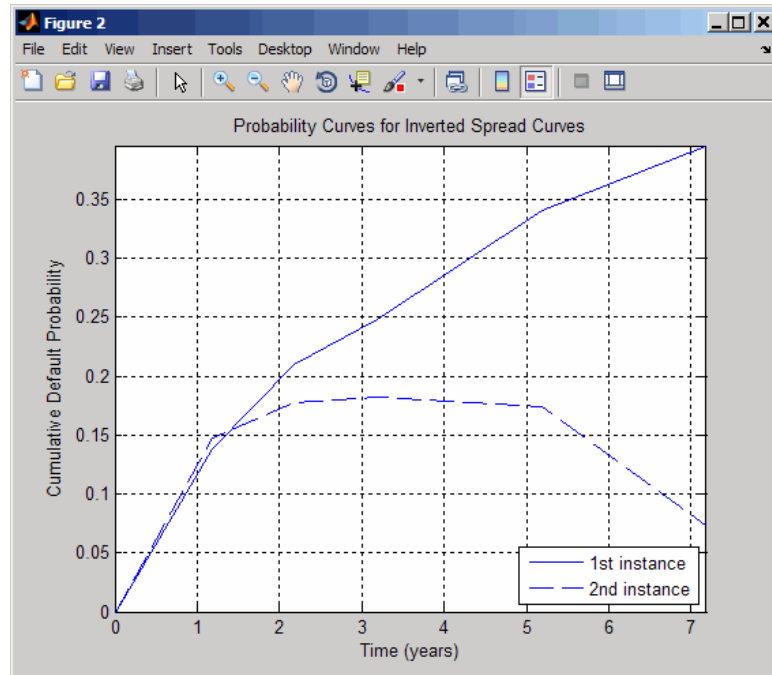
```
Warning: Found non-monotone default probabilities (negative hazard rates)
```

A non-monotone bootstrapped probability curve implies negative default probabilities and negative hazard rates for certain time intervals. Extreme market conditions can lead to these types of situations. In these cases, you must assess the reliability and usefulness of the bootstrapped results.

The following plot illustrates these bootstrapped probability curves. The curves are concave, meaning that the marginal default probability decreases with time. This result is consistent with the market information that indicates a higher default risk in the short term. The second bootstrapped curve is non-monotone, as indicated by the warning.

```
ProbTimes = yearfrac(Settle, MarketDates);  
figure  
plot([0; ProbTimes],[0; ProbDataInv1(:,2)])  
hold on  
plot([0; ProbTimes],[0; ProbDataInv2(:,2)],'--')  
hold off  
grid on  
axis([0 ProbTimes(end,1) 0 ProbDataInv1(end,2)])  
xlabel('Time (years)')  
ylabel('Cumulative Default Probability')  
title('Probability Curves for Inverted Spread Curves')  
legend('1st instance','2nd instance','location','SouthEast')
```

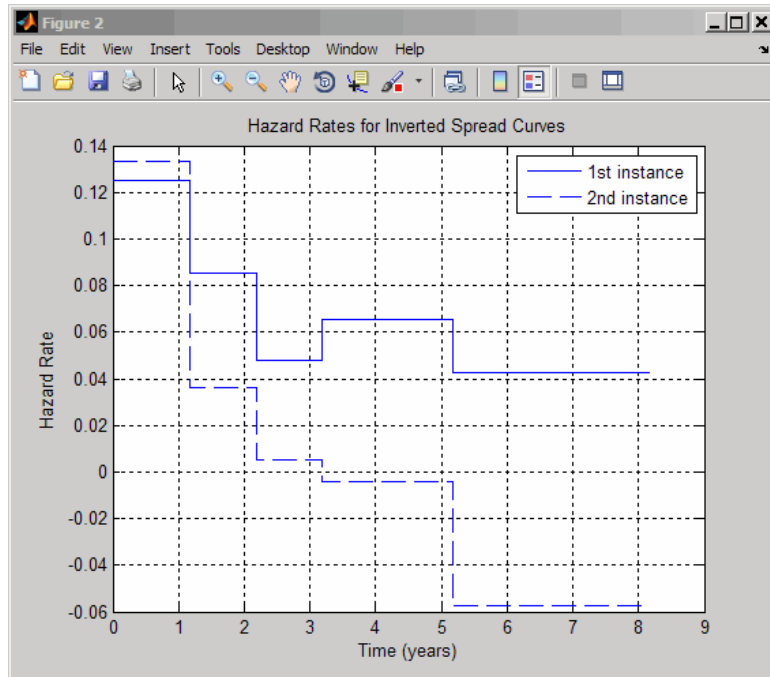
The resulting plot



Also in line with the previous plot, the hazard rates for these bootstrapped curves are decreasing because the short-term risk is higher. Some bootstrapped parameters in the second curve are negative, as indicated by the warning.

```
HazTimes = yearfrac(Settle, MarketDates);
figure
stairs([0; HazTimes(1:end-1,1); HazTimes(end,1)+1],...
       [HazDataInv1(:,2);HazDataInv1(end,2)])
hold on
stairs([0; HazTimes(1:end-1,1); HazTimes(end,1)+1],...
       [HazDataInv2(:,2);HazDataInv2(end,2)],'--')
hold off
grid on
xlabel('Time (years)')
ylabel('Hazard Rate')
title('Hazard Rates for Inverted Spread Curves')
legend('1st instance','2nd instance','location','NorthEast')
```

The resulting plot shows the hazard rates for both bootstrapped curves:



For further discussion on inverted curves, and their relationship to arbitrage, see O’Kane and Turnbull, 2003 (“Credit Derivatives” on page B-5).



## Credit Default Swap Option

A credit default swap (CDS) option, or credit default swaption, is a contract that provides the holder with the right, but not the obligation, to enter into a credit default swap in the future. CDS options can either be payer swaptions or receiver swaptions. If a payer swaption, the option holder has the right to enter into a CDS where they pay premiums; and, if a receiver swaption, the option holder receives premiums. Fixed-Income Toolbox software provides `cdsoptprice` for pricing payer and receiver credit default swaptions.

In addition, the optional `knockout` argument for `cdsoptprice` supports two variations of the mechanics of a CDS option. CDS options can be knockout or non-knockout options.

- A knockout option cancels with no payments if there is a credit event before the option expiry date.
- A non-knockout option does not cancel if there is a credit event before the option expiry date. In this case, the option holder of a non-knockout payer swaption can take delivery of the underlying long protection CDS on the option expiry date and exercise the protection, delivering a defaulted obligation in return for par.



# Interest-Rate Curve Objects

---

- “Introduction to Interest-Rate Curve Objects” on page 6-2
- “Creating Interest-Rate Curve Objects” on page 6-4
- “Creating an IRDataCurve Object” on page 6-6
- “Creating an IRFunctionCurve Object” on page 6-13
- “Converting an IRDataCurve or IRFunctionCurve Object” on page 6-25

## Introduction to Interest-Rate Curve Objects

In this section...
“Class Structure” on page 6-2
“Supported Workflow Model Using Interest-Rate Curve Objects” on page 6-3

### Class Structure

Fixed-Income Toolbox class structure supports interest-rate curve objects. The class structure supports five classes.

Class Name	Description
“@IRCurve” on page A-4	Base abstract class for interest-rate curves. <code>IRCurve</code> is an abstract class; you cannot create instances of it directly. You can create <code>IRFunctionCurve</code> and <code>IRDataCurve</code> objects that are derived from this class.
“@IRDataCurve” on page A-7	Creates a representation of an interest-rate curve with dates and data. <code>IRDataCurve</code> is constructed directly by specifying dates and corresponding interest rates or discount factors, or you can bootstrap an <code>IRDataCurve</code> object from market data.
“@IRFunctionCurve” on page A-12	Creates a representation of an interest-rate curve with a function. <code>IRFunctionCurve</code> is constructed directly by specifying a function handle, or you can fit a function to market data using methods of the <code>IRFunctionCurve</code> object.
“@IRBootstrapOptions” on page A-2	The <code>IRBootstrapOptions</code> object lets you specify options relating to the bootstrapping of an <code>IRDataCurve</code> object.
“@IRFitOptions” on page A-10	The <code>IRFitOptions</code> object lets you specify options relating to the fitting process for an <code>IRFunctionCurve</code> object.

## Supported Workflow Model Using Interest-Rate Curve Objects

The supported workflow model for using interest-rate curve objects is:

- 1** Create an interest-rate curve based on an `IRDataCurve` object or an `IRFunctionCurve` object.
  - To create an `IRDataCurve` object:
    - Use vectors of dates and data with interpolation methods.
    - Use bootstrapping based on market instruments.
  - To create an `IRFunctionCurve` object:
    - Specify a function handle.
    - Fit a function using the Nelson-Siegel model, Svensson model, or smoothing spline model.
    - Fit a custom function.
- 2** Use methods of the `IRDataCurve` or `IRFunctionCurve` objects to extract forward, zero, discount factor, or par yield curves for the interest-rate curve object.
- 3** Convert an interest-rate curve from an `IRDataCurve` or `IRFunctionCurve` object to a `RateSpec` structure. This `RateSpec` structure is identical to the `RateSpec` produced by the Financial Derivatives Toolbox™ function `intenvset`. Using the `RateSpec` for an interest-rate curve object, you can then use Financial Derivatives Toolbox functions to model an interest-rate structure and price. For more information, see “Interest-Rate Derivatives”.

## Creating Interest-Rate Curve Objects

Depending on your data and purpose for analysis, you can create an interest-rate curve object by using an `IRDataCurve` or `IRFunctionCurve` object.

To create an `IRDataCurve` object, you can:

- Use the `IRDataCurve` constructor.
- Use the `IRDataCurve` method `bootstrap`.

Using an `IRDataCurve` object, you can use the following methods to determine:

- Forward rate curve — `getForwardRates`
- Zero rate curve — `getZeroRates`
- Discount rate curve — `getDiscountFactors`
- Par yield curve — `getParYields`

Alternatively, to create an `IRFunctionCurve` object, you can:

- Use the `IRFunctionCurve` constructor and directly specify a function handle.
- Use `IRFunctionCurve` methods:
  - `fitNelsonSiegel` fits a **Nelson-Siegel model** on page Glossary-8 to market data for bonds.
  - `fitSvensson` fits a **Svensson model** on page Glossary-12 to market data for bonds.
  - `fitSmoothingSpline` fits a **smoothing spline** on page Glossary-12 function to market data for bonds.
  - `fitFunction` custom fits an interest-rate curve object to market data for bonds.

Using an `IRFunctionCurve` object, you can use the following method to determine:

- Forward rate curve — `getForwardRates`
- Zero rate curve — `getZeroRates`
- Discount rate curve — `getDiscountFactors`
- Par yield curve — `getParYields`

In addition, you can convert an `IRDataCurve` or `IRFunctionCurve` to a `RateSpec` structure. For more information, see “Converting an `IRDataCurve` or `IRFunctionCurve` Object” on page 6-25.

## Creating an IRDataCurve Object

### In this section...

“Using the IRDataCurve Constructor with Dates and Data” on page 6-6

“Using IRDataCurve bootstrap Method for Bootstrapping Based on Market Instruments” on page 6-7

### Using the IRDataCurve Constructor with Dates and Data

Use the IRDataCurve constructor with vectors of dates and data to create an interest-rate curve object. When constructing the IRDataCurve object, you can also use optional inputs to define how the interest-rate curve is constructed from the dates and data.

#### Example

In this example, you create the vectors for Dates and Data for an interest-rate curve:

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
```

Use the IRDataCurve constructor to build interest-rate objects based on the constant and pchip interpolation methods:

```
irdc_const = IRDataCurve('Forward',today,Dates,Data,'InterpMethod','constant');
irdc_pchip = IRDataCurve('Forward',today,Dates,Data,'InterpMethod','pchip');
```

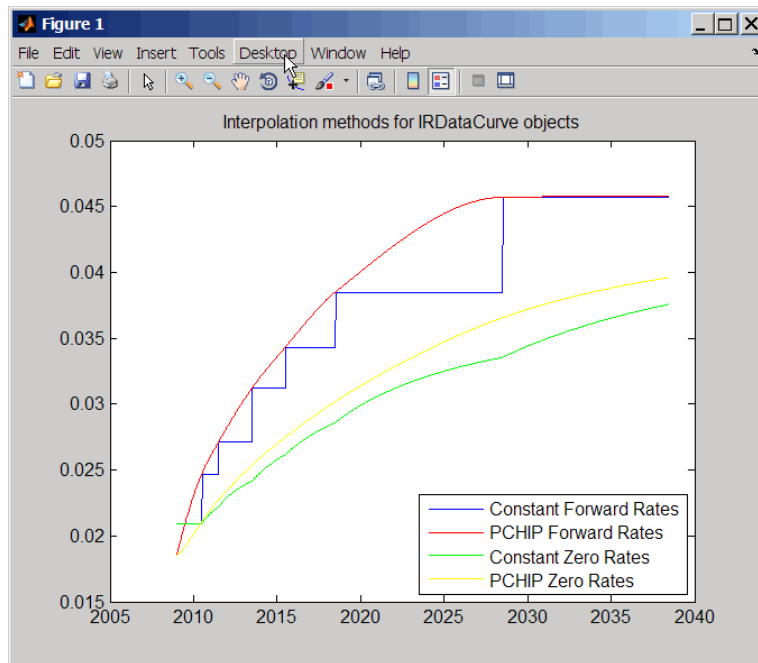
Plot the forward and zero rate curves for the two IRDataCurve objects based on constant and pchip interpolation methods:

```
PlottingDates = daysadd(today,180:10:360*30,1);
plot(PlottingDates,irdc_const.getForwardRates(PlottingDates),'b')
hold on
plot(PlottingDates,irdc_pchip.getForwardRates(PlottingDates),'r')
plot(PlottingDates,irdc_const.getZeroRates(PlottingDates),'g')
plot(PlottingDates,irdc_pchip.getZeroRates(PlottingDates),'yellow')
legend({'Constant Forward Rates','PCHIP Forward Rates','Constant Zero Rates',...
```



```
'PCHIP Zero Rates'}, 'location', 'SouthEast')
title('Interpolation methods for IRDataCurve objects')
datetick
```

The plot demonstrates the relationship of the forward and zero rate curves.



## Using IRDataCurve bootstrap Method for Bootstrapping Based on Market Instruments

Use the bootstrapping method, based on market instruments, to create an interest-rate curve object. When bootstrapping, you also have the option to define a range of interpolation methods (linear, spline, constant, and pchip).

### Example 1

In this example, you bootstrap a swap curve from deposits, Eurodollar Futures and swaps. The input market data for this example is hard-coded

and specified as two cell arrays of data; one cell array indicates the type of instrument and the other contains the `Settle`, `Maturity` values and a market quote for the instrument. For deposits and swaps, the quote is a rate; for the EuroDollar futures, the quote is a price. Although bonds are not used in this example, a bond would also be quoted with a price.

```
InstrumentTypes = {'Deposit';'Deposit';'Deposit';'Deposit';'Deposit'; ...
    'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Futures';'Futures';'Futures'; ...
    'Swap';'Swap';'Swap';'Swap';'Swap';'Swap';'Swap'};

Instruments = [datenum('08/10/2007'),datenum('08/17/2007'),.0532063; ...
    datenum('08/10/2007'),datenum('08/24/2007'),.0532000; ...
    datenum('08/10/2007'),datenum('09/17/2007'),.0532000; ...
    datenum('08/10/2007'),datenum('10/17/2007'),.0534000; ...
    datenum('08/10/2007'),datenum('11/17/2007'),.0535866; ...
    datenum('08/08/2007'),datenum('19-Dec-2007'),9485; ...
    datenum('08/08/2007'),datenum('19-Mar-2008'),9502; ...
    datenum('08/08/2007'),datenum('18-Jun-2008'),9509.5; ...
    datenum('08/08/2007'),datenum('17-Sep-2008'),9509; ...
    datenum('08/08/2007'),datenum('17-Dec-2008'),9505.5; ...
    datenum('08/08/2007'),datenum('18-Mar-2009'),9501; ...
    datenum('08/08/2007'),datenum('17-Jun-2009'),9494.5; ...
    datenum('08/08/2007'),datenum('16-Sep-2009'),9489; ...
    datenum('08/08/2007'),datenum('16-Dec-2009'),9481.5; ...
    datenum('08/08/2007'),datenum('17-Mar-2010'),9478; ...
    datenum('08/08/2007'),datenum('16-Jun-2010'),9474; ...
    datenum('08/08/2007'),datenum('15-Sep-2010'),9469.5; ...
    datenum('08/08/2007'),datenum('15-Dec-2010'),9464.5; ...
    datenum('08/08/2007'),datenum('16-Mar-2011'),9462.5; ...
    datenum('08/08/2007'),datenum('15-Jun-2011'),9456.5; ...
    datenum('08/08/2007'),datenum('21-Sep-2011'),9454; ...
    datenum('08/08/2007'),datenum('21-Dec-2011'),9449.5; ...
    datenum('08/08/2007'),datenum('08/08/2014'),.0530; ...
    datenum('08/08/2007'),datenum('08/08/2017'),.0545; ...
    datenum('08/08/2007'),datenum('08/08/2019'),.0551; ...
```

```

datenum('08/08/2007'),datenum('08/08/2022'),.0559; ...
datenum('08/08/2007'),datenum('08/08/2027'),.0565; ...
datenum('08/08/2007'),datenum('08/08/2032'),.0566; ...
datenum('08/08/2007'),datenum('08/08/2037'),.0566];

```

The `bootstrap` method is called as a static method of the “@IRDataCurve” on page A-7 class. Inputs to this method include the curve Type (zero or forward), Settle date, InstrumentTypes, and Instrument data. The `bootstrap` method also supports optional arguments, including an interpolation method, compounding, basis, and an options structure for bootstrapping. For example, you are passing in an “@IRBootstrapOptions” on page A-2 object which includes information for the ConvexityAdjustment to forward rates.

```

IRsigma = .01;
CurveSettle = datenum('08/10/2007');
bootModel = IRDataCurve.bootstrap('Forward', CurveSettle, ...
InstrumentTypes, Instruments,'InterpMethod','pchip',...
'Compounding',-1,'IRBootstrapOptions',...
IRBootstrapOptions('ConvexityAdjustment',@(t) .5*IRsigma^2.*t.^2))

```

```
bootModel =
```

```
IRDataCurve
```

```

Type: Forward
Settle: 733264 (10-Aug-2007)
Compounding: -1
Basis: 0 (actual/actual)
InterpMethod: pchip
Dates: [29x1 double]
Data: [29x1 double]

```

The `bootstrap` method uses an Optimization Toolbox™ function to solve for any bootstrapped rates.

Plot the forward and zero curves:

```

PlottingDates = (CurveSettle+20:30:CurveSettle+365*25)';
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);

```

```

BootstrappedForwardRates = bootModel.getForwardRates(PlottingDates);
BootstrappedZeroRates = bootModel.getZeroRates(PlottingDates);

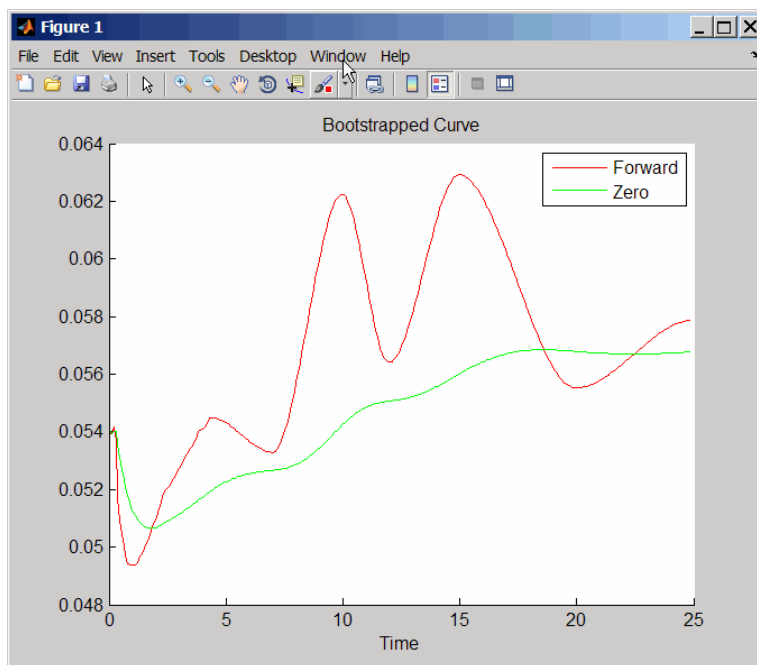
```

```

figure
hold on
plot(TimeToMaturity,BootstrappedForwardRates,'r')
plot(TimeToMaturity,BootstrappedZeroRates,'g')
title('Bootstrapped Curve')
xlabel('Time')
legend({'Forward','Zero'})

```

The plot demonstrates the forward and zero rate curves for the market data.



## Example 2

In this example, you bootstrap a swap curve from deposits, Eurodollar futures and swaps. The input market data for this example is hard-coded and specified as two cell arrays of data; one cell array indicates the type of

instrument and the other cell array contains the `Settle`, `Maturity` values and a market quote for the instrument. This example of bootstrapping also demonstrates the use of an `InstrumentBasis` for each `Instrument` type:

```
InstrumentTypes = {'Deposit';'Deposit';...
'Futures';'Futures';'Futures';'Futures';'Futures';'Futures';...
'Swap';'Swap';'Swap';'Swap';};

Instruments = [datenum('08/10/2007'),datenum('09/17/2007'),.0532000; ...
datenum('08/10/2007'),datenum('11/17/2007'),.0535866; ...
datenum('08/08/2007'),datenum('19-Dec-2007'),9485; ...
datenum('08/08/2007'),datenum('19-Mar-2008'),9502; ...
datenum('08/08/2007'),datenum('18-Jun-2008'),9509.5; ...
datenum('08/08/2007'),datenum('17-Sep-2008'),9509; ...
datenum('08/08/2007'),datenum('17-Dec-2008'),9505.5; ...
datenum('08/08/2007'),datenum('18-Mar-2009'),9501; ...
datenum('08/08/2007'),datenum('08/08/2014'),.0530; ...
datenum('08/08/2007'),datenum('08/08/2019'),.0551; ...
datenum('08/08/2007'),datenum('08/08/2027'),.0565; ...
datenum('08/08/2007'),datenum('08/08/2037'),.0566];

CurveSettle = datenum('08/10/2007');
```

The `bootstrap` method is called as a static method of the “@IRDataCurve” on page A-7 class. Inputs to this method include the curve `Type` (zero or forward), `Settle` date, `InstrumentTypes`, and `Instrument` data. The `bootstrap` method also supports optional arguments, including an interpolation method, compounding, basis, and an options structure for bootstrapping. In this example, you are passing an additional `Basis` value for each instrument type:

```
bootModel=IRDataCurve.bootstrap('Forward',CurveSettle,InstrumentTypes, ...
Instruments,'InterpMethod','pchip','InstrumentBasis',[repmat(2,8,1);repmat(0,4,1)])

bootModel =

IRDataCurve

Type: Forward
Settle: 733264 (10-Aug-2007)
```

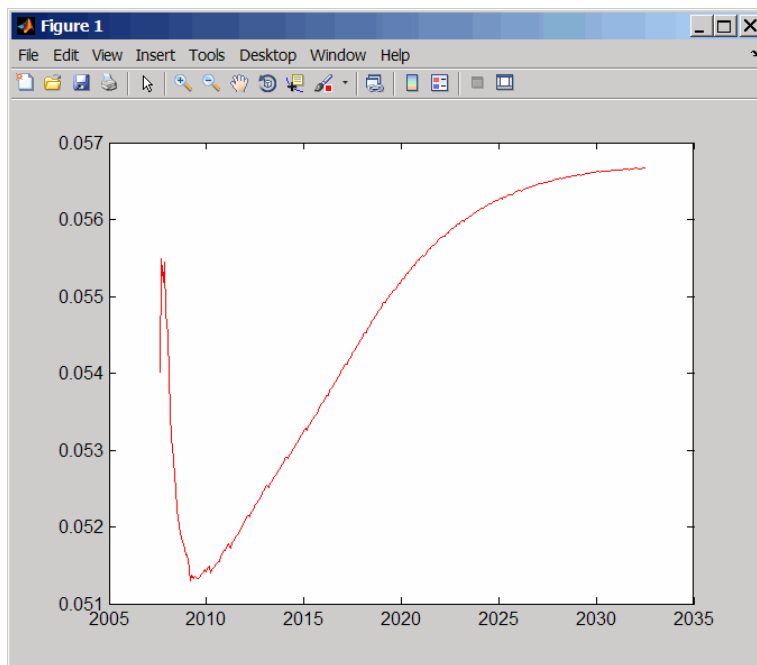
```
Compounding: 2  
Basis: 0 (actual/actual)  
InterpMethod: pchip  
Dates: [12x1 double]  
Data: [12x1 double]
```

The bootstrap method uses an Optimization Toolbox function to solve for any bootstrapped rates.

Plot the par yields curve using the `getParYields` method:

```
PlottingDates = (datenum('08/11/2007'):30:CurveSettle+365*25)';  
plot(PlottingDates,bootModel.getParYields(PlottingDates),'r')  
datetick
```

The plot demonstrates the par yields curve for the market data.



## Creating an IRFunctionCurve Object

### In this section...

“Using a Function Handle to Fit an IRFunctionCurve Object” on page 6-13

“Using the Nelson-Siegel Method to Fit an IRFunctionCurve Object” on page 6-14

“Using the Svensson Method to Fit an IRFunctionCurve Object” on page 6-16

“Using the Smoothing Spline Method to Fit an IRFunctionCurve Object” on page 6-18

“Using the fitFunction Method to Create a Custom Fitting Function for an IRFunctionCurve Object” on page 6-21

## Using a Function Handle to Fit an IRFunctionCurve Object

You can use the constructor `IRFunctionCurve` with a MATLAB function handle to define an interest-rate curve. For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.

### Example

This example uses a `FunctionHandle` argument with a value `@(t) t.^2` to construct an interest-rate curve:

```
rr = IRFunctionCurve('Zero',today,@(t) t.^2);  
rr =
```

Properties:

```
FunctionHandle: @(t)t.^2  
Type: 'Zero'  
Settle: 733600  
Compounding: 2  
Basis: 0
```

## Using the Nelson-Siegel Method to Fit an IRFunctionCurve Object

Use the method, `fitNelsonSiegel`, for the Nelson-Siegel model that fits the empirical form of the yield curve with a prespecified functional form of the spot rates which is a function of the time to maturity of the bonds.

The Nelson-Siegel model represents a dynamic three-factor model: level, slope, and curvature. However, the Nelson-Siegel factors are unobserved, or latent, which allows for measurement error, and the associated loadings have economic restrictions (forward rates are always positive, and the discount factor approaches zero as maturity increases). For more information, see “Zero-coupon yield curves: technical documentation,” *BIS Papers*, Bank for International Settlements, Number 25, October, 2005.

### Example

This example uses `IRFunctionCurve` to model the default-free term structure of interest rates in the United Kingdom.

Load the data:

```
load ukdata20080430
```

Convert repo rates to be equivalent zero coupon bonds:

```
RepoCouponRate = repmat(0,size(RepoRates));  
RepoPrice = bndprice(RepoRates, RepoCouponRate, RepoSettle, RepoMaturity);
```

Aggregate the data:

```
Settle = [RepoSettle;BondSettle];  
Maturity = [RepoMaturity;BondMaturity];  
CleanPrice = [RepoPrice;BondCleanPrice];  
CouponRate = [RepoCouponRate;BondCouponRate];  
Instruments = [Settle Maturity CleanPrice CouponRate];  
InstrumentPeriod = [repmat(0,6,1);repmat(2,31,1)];  
CurveSettle = datenum('30-Apr-2008');
```

The `IRFunctionCurve` object provides the capability to fit a Nelson-Siegel curve to observed market data with the `fitNelsonSiegel` method. The fitting

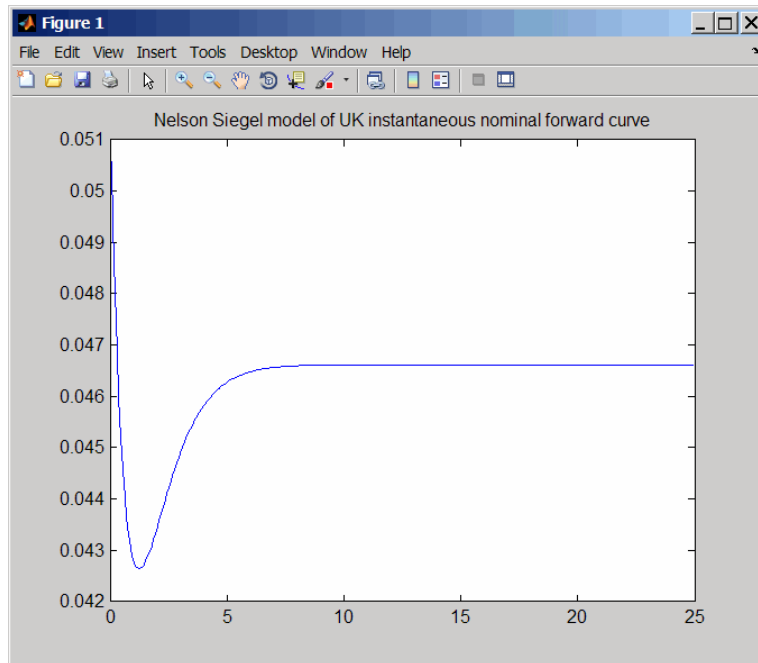


is done by calling the Optimization Toolbox function `lsqnonlin`. This method has required inputs of `Type`, `Settle`, and a matrix of instrument data.

```
NSModel = IRFunctionCurve.fitNelsonSiegel('Zero',CurveSettle,...  
Instruments,'Compounding',-1,'InstrumentPeriod',InstrumentPeriod);
```

Plot the Nelson-Siegel interest-rate curve for forward rates:

```
PlottingDates = CurveSettle+20:30:CurveSettle+365*25;  
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);  
NSForwardRates = NSModel.getForwardRates(PlottingDates);  
plot(TimeToMaturity,NSForwardRates)  
title('Nelson Siegel model of UK instantaneous nominal forward curve')
```



## Using the Svensson Method to Fit an IRFunctionCurve Object

Use the method, `fitSvensson`, for the Svensson model to improve the flexibility of the curves and the fit for a Nelson-Siegel model. In 1994, Svensson extended Nelson and Siegel's function by adding a further term that allows for a second "hump." The extra precision is achieved at the cost of adding two more parameters,  $\beta_3$  and  $\tau_2$ , which have to be estimated.

### Example

In this example of using the `fitSvensson` method, an `IRFitOptions` structure, previously defined using the `IRFitOptions` constructor, is used. Thus, you must specify `FitType`, `InitialGuess`, `UpperBound`, `LowerBound`, and the `OptOptions` optimization parameters for `lsqnonlin`.

Load the data:

```
load ukdata20080430
```

Convert repo rates to be equivalent zero coupon bonds:

```
RepoCouponRate = repmat(0,size(RepoRates));
RepoPrice = bndprice(RepoRates, RepoCouponRate, RepoSettle, RepoMaturity);
```

Aggregate the data:

```
Settle = [RepoSettle;BondSettle];
Maturity = [RepoMaturity;BondMaturity];
CleanPrice = [RepoPrice;BondCleanPrice];
CouponRate = [RepoCouponRate;BondCouponRate];
Instruments = [Settle Maturity CleanPrice CouponRate];
InstrumentPeriod = [repmat(0,6,1);repmat(2,31,1)];
CurveSettle = datenum('30-Apr-2008');
```

Define `OptOptions` for the `IRFitOptions` constructor:

```
OptOptions = optimset('lsqnonlin');
OptOptions = optimset(OptOptions,'MaxFunEvals',1000);
fIRFitOptions = IRFitOptions([5.82 -2.55 -.87 0.45 3.9 0.44],...
'FitType','durationweightedprice','OptOptions',OptOptions,...
'LowerBound',[0 -Inf -Inf -Inf 0 0],'UpperBound',[Inf Inf Inf Inf Inf Inf]);
```

Fit the interest-rate curve using a Svensson model:

```
SvenssonModel = IRFunctionCurve.fitSvensson('Zero',CurveSettle,...  
Instruments,'IRFitOptions',fIRFitOptions, 'Compounding',-1,...  
'InstrumentPeriod',InstrumentPeriod);
```

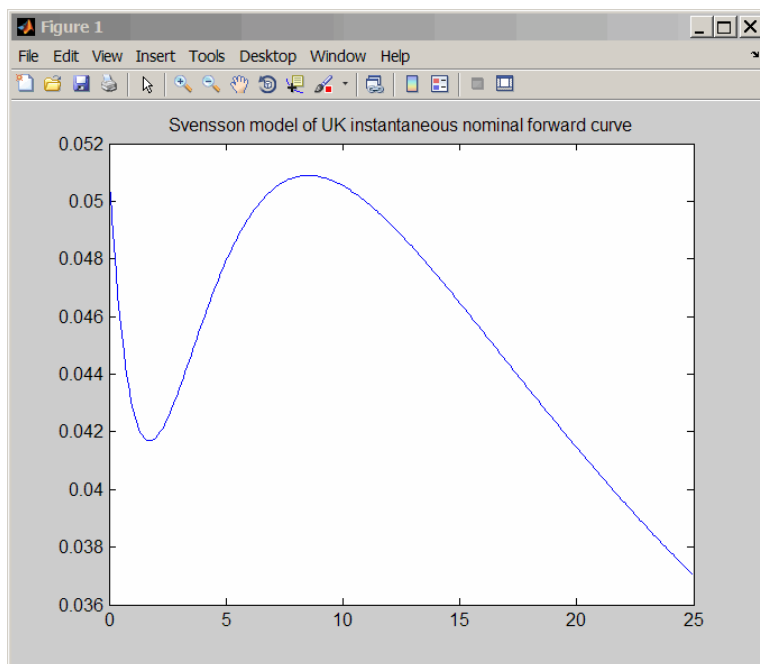
Local minimum possible.

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

The status message, output from `lsqnonlin`, indicates that the optimization to find parameters for the Svensson equation terminated successfully.

Plot the Svensson interest-rate curve for forward rates:

```
PlottingDates = CurveSettle+20:30:CurveSettle+365*25;  
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);  
SvenssonForwardRates = SvenssonModel.getForwardRates(PlottingDates);  
plot(TimeToMaturity,SvenssonForwardRates)  
title('Svensson model of UK instantaneous nominal forward curve')
```



### Using the Smoothing Spline Method to Fit an IRFunctionCurve Object

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

---

**Note** You must have a license for Curve Fitting Toolbox software to use the `fitSmoothingSpline` method.

---

### Example

The `IRFunctionCurve` object is used to fit a smoothing spline representation of the forward curve with a penalty function. Required inputs are `Type`, `Settle`, the matrix of Instruments, and `LambdaFun`, a function handle containing the penalty function

Load the data:

```
load ukdata20080430
```

Convert repo rates to be equivalent zero coupon bonds:

```
RepoCouponRate = repmat(0,size(RepoRates));
RepoPrice = bndprice(RepoRates, RepoCouponRate, RepoSettle, RepoMaturity);
```

Aggregate the data:

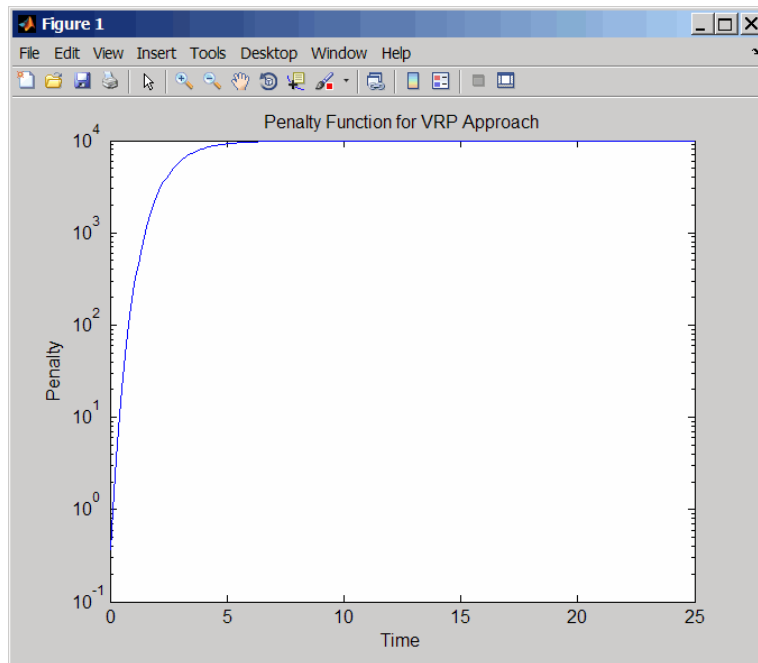
```
Settle = [RepoSettle;BondSettle];
Maturity = [RepoMaturity;BondMaturity];
CleanPrice = [RepoPrice;BondCleanPrice];
CouponRate = [RepoCouponRate;BondCouponRate];
Instruments = [Settle Maturity CleanPrice CouponRate];
InstrumentPeriod = [repmat(0,6,1);repmat(2,31,1)];
CurveSettle = datenum('30-Apr-2008');
```

Choose parameters for Lambdafun:

```
L = 9.2;
S = -1;
mu = 1;
```

Define the Lambdafun penalty function:

```
lambdafun = @(t) exp(L - (L-S)*exp(-t/mu));
t = 0:.1:25;
y = lambdafun(t);
figure
semilogy(t,y);
title('Penalty Function for VRP Approach')
ylabel('Penalty')
xlabel('Time')
```

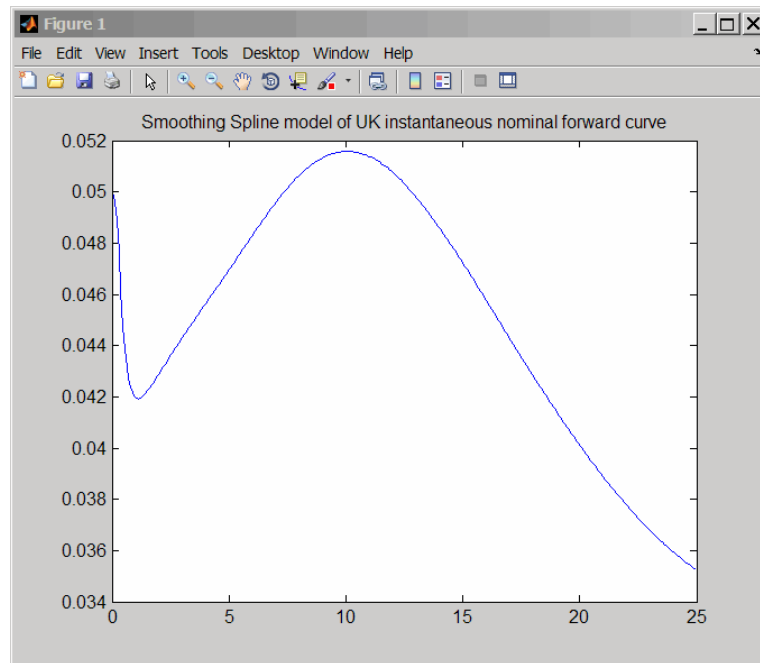


Use the `fitSmoothingSpline` method to fit the interest-rate curve and model the `Lambdafun` penalty function:

```
VRPModel = IRFunctionCurve.fitSmoothingSpline('Forward',CurveSettle,...  
Instruments,lambdafun,'Compounding',-1, 'InstrumentPeriod',InstrumentPeriod);
```

Plot the smoothing spline interest-rate curve for forward rates:

```
PlottingDates = CurveSettle+20:30:CurveSettle+365*25;  
TimeToMaturity = yearfrac(CurveSettle,PlottingDates);  
VRPForwardRates = VRPModel.getForwardRates(PlottingDates);  
plot(TimeToMaturity,VRPForwardRates)  
title('Smoothing Spline model of UK instantaneous nominal forward curve')
```



## Using the fitFunction Method to Create a Custom Fitting Function for an IRFunctionCurve Object

When using an IRFunctionCurve object, you can create a custom fitting function with the fitFunction method. To use fitFunction, you must define a FunctionHandle. In addition, you must also use the constructor IRFitOptions to define IRFitOptionsObj to support an InitialGuess for the parameters of the curve function.

### Example

The following example demonstrates the use of fitFunction with a FunctionHandle and an IRFitOptionsObj:

```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];
```

```
CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
CurveSettle = datenum('30-Apr-2008');
```

Define the FunctionHandle:

```
functionHandle = @(t,theta) polyval(theta,t);
```

Define the OptOptions for IRFitOptions:

```
OptOptions = optimset('lsqnonlin');
OptOptions = optimset(OptOptions,'display','iter');
```

Define fitFunction:

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

Iteration	Func-count	f(x)	Norm of step	First-order optimality	CG-iterations
0	4	38036.7		4.92e+004	
1	8	38036.7	10	4.92e+004	0
2	12	38036.7	2.5	4.92e+004	0
3	16	38036.7	0.625	4.92e+004	0
4	20	38036.7	0.15625	4.92e+004	0
5	24	30741.5	0.0390625	1.72e+005	0
6	28	30741.5	0.078125	1.72e+005	0
7	32	30741.5	0.0195312	1.72e+005	0
8	36	28713.6	0.00488281	2.33e+005	0
9	40	20323.3	0.00976562	9.47e+005	0
10	44	20323.3	0.0195312	9.47e+005	0
11	48	20323.3	0.00488281	9.47e+005	0
12	52	20323.3	0.0012207	9.47e+005	0
13	56	19698.8	0.000305176	1.08e+006	0
14	60	17493	0.000610352	7e+006	0
15	64	17493	0.0012207	7e+006	0
16	68	17493	0.000305176	7e+006	0
17	72	15455.1	7.62939e-005	2.25e+007	0



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

Local minimum possible.

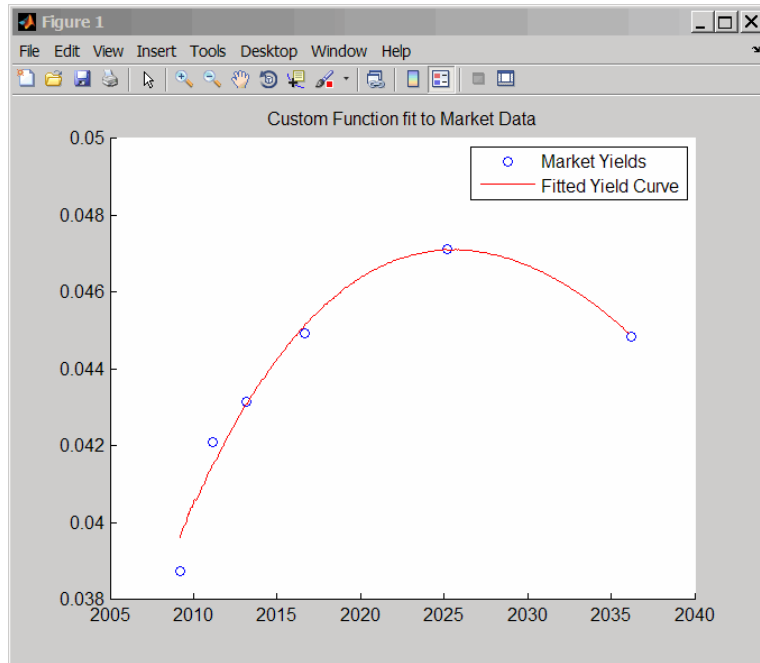
lsqnonlin stopped because the final change in the sum of squares relative to its initial value is less than the selected value of the function tolerance.

Plot the custom function that is defined using fitFunction:

```

Yields = bndyield(CleanPrice,CouponRate,Settle(1),Maturity);
scatter(Maturity,Yields);
PlottingPoints = min(Maturity):30:max(Maturity);
hold on;
plot(PlottingPoints,CustomModel.getParYields(PlottingPoints),'r');
datetick
legend('Market Yields','Fitted Yield Curve')
title('Custom Function fit to Market Data')

```



## Converting an IRDataCurve or IRFunctionCurve Object

### In this section...

“Introduction” on page 6-25

“Using the toRateSpec Method” on page 6-25

“Using Vector of Dates and Data Methods” on page 6-27

### Introduction

The IRDataCurve and IRFunctionCurve objects for interest-rate curves support conversion to:

- A RateSpec structure. The RateSpec generated from an IRDataCurve or IRFunctionCurve object, using the toRateSpec method, is identical to the RateSpec structure created with intenvset using Financial Derivatives Toolbox software.
- A vector of dates and data from an IRDataCurve object acceptable to prbyzero, bkcall, bkput, tfutbyprice, and tfutbyyield or any function that requires a term structure of interest rates.

### Using the toRateSpec Method

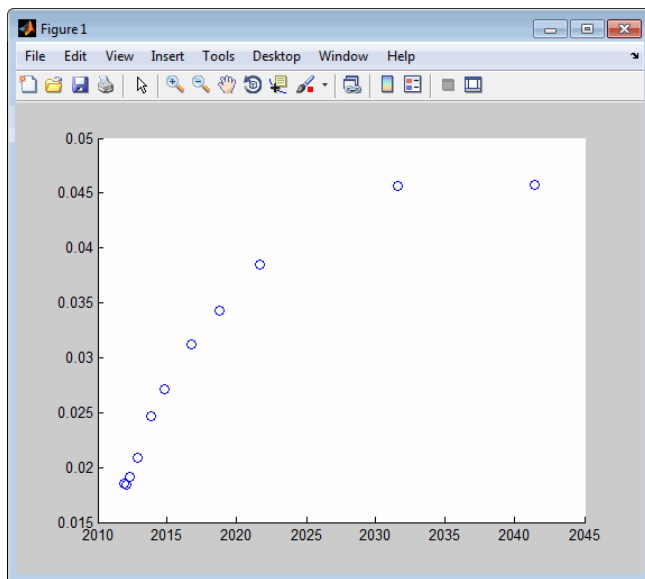
To convert an IRDataCurve or IRFunctionCurve object to a RateSpec structure, you must first create an interest-rate curve object. Then, use the toRateSpec method for an IRDataCurve object or the toRateSpec method for an IRFunctionCurve object.

### Example

Create a data vector from the following data:

<http://www.ustreas.gov/offices/domestic-finance/debt-management/interest-rate/yield.shtml>:

```
Data = [1.85 1.84 1.91 2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[30 90 180 360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],2);
scatter(Dates,Data)
datetick
```



Create an `IRDataCurve` interest-rate curve object:

```
rr = IRDataCurve('Zero',today,Dates,Data);
```

Convert to a `RateSpec`:

```
rr.toRateSpec(today+30:30:today+365)  
ans =
```

```
    FinObj: 'RateSpec'  
    Compounding: 2  
           Disc: [12x1 double]  
           Rates: [12x1 double]  
           EndTimes: [12x1 double]  
           StartTimes: [12x1 double]  
           EndDates: [12x1 double]  
           StartDates: 733569  
    ValuationDate: 733569  
           Basis: 0  
           EndMonthRule: 1
```

## Using Vector of Dates and Data Methods

You can use the `getZeroRates` method for an `IRDataCurve` object with a `Dates` property to create a vector of dates and data acceptable for `prbyzero` in Financial Toolbox software and `bkcall`, `bkput`, `tfutbyprice`, and `tfutbyyield` in Fixed-Income Toolbox software.

### Example

This is an example of using the `IRDataCurve` method `getZeroRates` with `prbyzero`:

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data,'InterpMethod','pchip');
Maturity = daysadd(today,8*360,1);
CouponRate = .055;
ZeroDates = daysadd(today,180:180:8*360,1);
ZeroRates = irdc.getZeroRates(ZeroDates);
BondPrice = prbyzero([Maturity CouponRate], today, ZeroRates, ZeroDates)
BondPrice =
    113.9221
```



# Function Reference

---

Bond Futures (p. 7-2)	Work with bond futures
Certificates of Deposit (p. 7-3)	Work with certificates of deposit
Convertible Bonds (p. 7-4)	Work with convertible bonds
Credit Default Swaps (p. 7-5)	Work with credit default swaps
Derivative Securities (p. 7-6)	Work with derivative securities
Interest-Rate Curve Objects (p. 7-7)	Work with interest-rate curve objects
Mortgage-Backed Securities (p. 7-9)	Work with mortgage-backed securities
Option-Adjusted Spread Computations (p. 7-11)	Work with option-adjusted spread computations
Stepped-Coupon Bonds (p. 7-12)	Work with stepped-coupon bonds
Treasury Bills (p. 7-13)	Work with Treasury bills
Zero-Coupon Instruments (p. 7-14)	Work with zero-coupon instruments

## Bond Futures

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



## Certificates of Deposit

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

## **Convertible Bonds**

cbprice

Price convertible bond

## Credit Default Swaps

<code>cdsbootstrap</code>	Bootstrap default probability curve from credit default swap market quotes
<code>cdsoptprice</code>	Price payer and receiver credit default swaptions
<code>cdsprice</code>	Determine price for credit default swap
<code>cdsspread</code>	Determine spread of credit default swap

## Derivative Securities

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

## Interest-Rate Curve Objects

<code>bootstrap</code>	Bootstrap interest-rate curve from market data
<code>fitFunction</code>	Custom fit interest-rate curve object to bond market data
<code>fitNelsonSiegel</code>	Fit Nelson-Siegel function to bond market data
<code>fitSmoothingSpline</code>	Fit smoothing spline to bond market data
<code>fitSvensson</code>	Fit Svensson function to bond market data
<code>getDiscountFactors</code>	Get discount factors for input dates for <code>IRDataCurve</code>
<code>getDiscountFactors</code>	Get discount factors for input dates for <code>IRFunctionCurve</code>
<code>getForwardRates</code>	Get forward rates for input dates for <code>IRDataCurve</code>
<code>getForwardRates</code>	Get forward rates for input dates for <code>IRFunctionCurve</code>
<code>getParYields</code>	Get par yields for input dates for <code>IRDataCurve</code>
<code>getParYields</code>	Get par yields for input dates for <code>IRFunctionCurve</code>
<code>getZeroRates</code>	Get zero rates for input dates for <code>IRDataCurve</code>
<code>getZeroRates</code>	Get zero rates for input dates for <code>IRFunctionCurve</code>
<code>IRBootstrapOptions</code>	Construct specific options for bootstrapping interest-rate curve object
<code>IRDataCurve</code>	Construct interest-rate curve object from dates and data

<code>IRFitOptions</code>	Construct specific options for fitting interest-rate curve object
<code>IRFunctionCurve</code>	Construct interest-rate curve object from function handle or function and fit to market data
<code>toRateSpec</code>	Convert <code>IRDataCurve</code> object to <code>RateSpec</code>
<code>toRateSpec</code>	Convert <code>IRFunctionCurve</code> object to <code>RateSpec</code>

## Mortgage-Backed Securities

cmosched	Generate principal balance schedule for planned amortization class (PAC) or targeted amortization class (TAC) bond
cmoschedcf	Generate cash flows for scheduled collateralized mortgage obligation (CMO) using PAC or TAC model
cmoseqcf	Generate cash flows for sequential collateralized mortgage obligation (CMO)
mbscfamounts	Cash flow and time mapping for mortgage pool
mbsconvp	Convexity of mortgage pool given price
mbsconvy	Convexity of mortgage pool given yield
mbsdurp	Duration of mortgage pool given price
mbsdury	Duration of mortgage pool given yield
mbsnoprepay	End-of-month mortgage cash flows and balances without prepayment
mbspassthrough	Mortgage pool cash flows and balances with prepayment
mbsprice	Mortgage-backed security price given yield
mbsprice2speed	Implied PSA prepayment speeds given price
mbswal	Weighted average life of mortgage pool

mbsyield	Mortgage-backed security yield given price
mbsyield2speed	Implied PSA prepayment speeds given yield
psaspeed2default	Benchmark default
psaspeed2rate	Single monthly mortality rate given PSA speed



## Option-Adjusted Spread Computations

agencyoas	Determine option-adjusted spread of callable bond using Agency OAS model
agencyprice	Price callable bond using Agency OAS model
mboas2price	Price given option-adjusted spread
mboas2yield	Yield given option-adjusted spread
mboprice2oas	Option-adjusted spread given price
mboyield2oas	Option-adjusted spread given yield

## **Stepped-Coupon Bonds**

stepcpncfamounts

Cash flow amounts and times for bonds and stepped coupons

stepcpnprice

Price bond with stepped coupons

stepcpnyield

Yield to maturity of bond with stepped coupons

## Treasury Bills

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

## **Zero-Coupon Instruments**

zeroprice

Price zero-coupon instruments given yield

zeroyield

Yield of zero-coupon instruments given price

# Functions — Alphabetical List

---

# agencyoas

---

<b>Purpose</b>	Determine option-adjusted spread of callable bond using Agency OAS model
<b>Syntax</b>	<pre>OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate) OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate, Name,Value)</pre>
<b>Description</b>	<p>OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate) computes OAS of a callable bond given price using the Agency OAS model.</p> <p>OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate, Name,Value) computes OAS of a callable bond given price using the Agency OAS model with additional options specified by one or more Name,Value pair arguments.</p>
<b>Input Arguments</b>	<p><b>ZeroData</b> Zero curve represented as a numRates-by-2 matrix where the first column is zero dates and the second column is the accompanying zero rates.</p> <p><b>Price</b> numBonds-by-1 vector of prices.</p> <p><b>CouponRate</b> numBonds-by-1 vector of coupon rates in decimal form.</p> <p><b>Settle</b> Scalar MATLAB date number for the settlement date for all bonds and the zero data.</p>

---

**Note** The `Settle` date must be an identical settlement date for all the bonds and the zero curve.

---

#### Maturity

`numBonds-by-1` vector of maturity dates.

#### Vol

`numBonds-by-1` vector of volatilities in decimal form. This is the volatility of interest rates corresponding to the time of the `CallDate`.

#### CallDate

`numBonds-by-1` vector of call dates.

### **Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as `Name1, Value1, . . . , NameN, ValueN`.

#### Basis

`N-by-1` vector of day-count basis:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**Default:** 0 (actual/actual)

#### CurveBasis

Basis of the zero curve, where the choices are identical to Basis.

**Default:** 0 (actual/actual)

#### CurveCompounding

Compounding frequency of the zero curve. Possible values include: -1, 0, 1, 2, 3, 4, 6, 12.

**Default:** 2 (Semi-annual)

#### EndMonthRule

End-of-month rule; 1, indicating in effect, and 0, indicating rule not in effect for the bond(s). When 1, the rule is in effect for the bond(s), this means that a security that pays coupon interest on the last day of the month will always make payment on the last day of the month.

**Default:** 1 — Indicates in effect

#### Face



Face value of the bond.

**Default:** 100

FirstCouponDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

**Default:** If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

InterpMethod

Interpolation method used to obtain points from the zero curve. Values are:

- linear — linear interpolation
- cubic — piecewise cubic spline interpolation
- pchip — piecewise cubic Hermite interpolation

**Default:** linear

IssueDate

Bond issue date.

**Default:** If you do not specify an IssueDate, the cash flow payment dates are determined from other inputs.

LastCouponDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at

the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

**Default:** If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

`Period`

Number of coupon payments per year. Possible values include: 0, 1, 2, 3, 4, 6, 12.

**Default:** 2

`StartDate`

Forward starting date of payments.

**Default:** If you do not specify a `StartDate`, the effective start date is the `Settle` date.

## Output Arguments

OAS

numBonds-by-1 matrix of option-adjusted spreads.

## Definitions

### Agency OAS Model

The BMA European Callable Securities Formula provides a standard methodology for computing price and option-adjusted spread for European Callable Securities (ECS).

## Examples

Compute the agency OAS value:

```
Settle = datenum('20-Jan-2010');
ZeroRates = [.07 .164 .253 1.002 1.732 2.226 2.605 3.316 ...
3.474 4.188 4.902]'/100;
ZeroDates = daysadd(Settle,360* [.25 .5 1 2 3 4 5 7 10 20 30],1);
ZeroData = [ZeroDates ZeroRates];

Maturity = datenum('30-Dec-2013');
```

```
CouponRate = .022;  
Price = 99.155;  
Vol = .5117;  
CallDate = datenum('30-Dec-2010');  
OAS = agencyoas(ZeroData, Price, CouponRate, Settle, Maturity, Vol, CallDate)  
OAS =  
  
8.6279
```

## References

SIFMA, The BMA European Callable Securities Formula,  
<http://www.sifma.org>.

## See Also

| [agencyprice](#) |

## Tutorials

- “Agency Option-Adjusted Spreads” on page 3-2

# agencyprice

---

<b>Purpose</b>	Price callable bond using Agency OAS model
<b>Syntax</b>	<pre>Price = agencyprice(ZeroData, OAS, CouponRate, Settle,     Maturity,     Vol, CallDate) Price = agencyprice(ZeroData, OAS, CouponRate, Settle,     Maturity,     Vol, CallDate, Name,Value)</pre>
<b>Description</b>	<p>Price = agencyprice(ZeroData, OAS, CouponRate, Settle, Maturity, Vol, CallDate) computes the price for a callable bond, given OAS, using the Agency OAS model.</p> <p>Price = agencyprice(ZeroData, OAS, CouponRate, Settle, Maturity, Vol, CallDate, Name,Value) computes the price for a callable bond, given OAS, using the Agency OAS model with additional options specified by one or more Name,Value pair arguments.</p>
<b>Input Arguments</b>	<p><b>ZeroData</b> Zero curve represented as a numRates-by-2 matrix where the first column is zero dates and the second column is the accompanying zero rates.</p> <p><b>OAS</b> numBonds-by-1 vector of option-adjusted spreads, expressed as a decimal (i.e., 50 basis points is entered as .005).</p> <p><b>CouponRate</b> numBonds-by-1 vector of coupon rates in decimal form.</p> <p><b>Settle</b> Scalar MATLAB date number for the settlement date for all the bonds and the zero data.</p>

---

**Note** The `Settle` date must be an identical settlement date for all bonds and the zero curve.

---

#### Maturity

`numBonds-by-1` vector of maturity dates.

#### Vol

`numBonds-by-1` vector of volatilities in decimal form. This is the volatility of interest rates corresponding to the time of the `CallDate`.

#### CallDate

`numBonds-by-1` vector of call dates.

### **Name-Value Pair Arguments**

Specify optional comma-separated pairs of `Name`, `Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as `Name1, Value1, ..., NameN, ValueN`.

#### Basis

`N-by-1` vector of day-count basis:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)

- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**Default:** 0 (actual/actual)

#### CurveBasis

Basis of the zero curve, where the choices are identical to Basis.

**Default:** 0 (actual/actual)

#### CurveCompounding

Compounding frequency of the curve. Possible values include: -1, 0, 1, 2, 3, 4, 6, 12.

**Default:** 2 (Semi-annual)

#### EndMonthRule

End-of-month rule; 1, indicating in effect, and 0, indicating rule not in effect for the bond(s). When 1, the rule is in effect for the bond(s). This means that a security that pays coupon interest on the last day of the month will always make payment on the last day of the month.

**Default:** 1 — Indicates in effect

#### Face

Face value of the bond.

**Default:** 100

FirstCouponDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When FirstCouponDate and LastCouponDate are both specified, FirstCouponDate takes precedence in determining the coupon payment structure.

**Default:** If you do not specify a FirstCouponDate, the cash flow payment dates are determined from other inputs.

InterpMethod

Interpolation method used to obtain points from the zero curve. Values are:

- linear — linear interpolation
- cubic — piecewise cubic spline interpolation
- pchip — piecewise cubic Hermite interpolation

**Default:** linear

IssueDate

Bond issue date.

**Default:** If you do not specify an IssueDate, the cash flow payment dates are determined from other inputs.

LastCouponDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at

the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

**Default:** If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

`Period`

Number of coupon payments per year. Possible values include: 0, 1, 2, 3, 4, 6, 12.

**Default:** 2

`StartDate`

Forward starting date of payments.

**Default:** If you do not specify a `StartDate`, the effective start date is the `Settle` date.

## Output Arguments

`Price`

numBonds-by-1 matrix of the price.

## Definitions

### Agency OAS Model

The BMA European Callable Securities Formula provides a standard methodology for computing price and option-adjusted spread for European Callable Securities (ECS).

## Examples

Compute the agency Price:

```
Settle = datenum('20-Jan-2010');
ZeroRates = [.07 .164 .253 1.002 1.732 2.226 2.605 3.316 ...
3.474 4.188 4.902]'/100;
ZeroDates = daysadd(Settle,360* [.25 .5 1 2 3 4 5 7 10 20 30],1);
ZeroData = [ZeroDates ZeroRates];
```

```
Maturity = datenum('30-Dec-2013');
```



```
CouponRate = .022;  
OAS = 6.53/10000;  
Vol = .5117;  
CallDate = datenum('30-Dec-2010');  
Price = agencyprice(ZeroData, OAS, CouponRate, Settle, Maturity, Vol, CallDate)  
Price =  
  
99.4226
```

## References

SIFMA, The BMA European Callable Securities Formula,  
<http://www.sifma.org>.

## See Also

| agencyoas |

## Tutorials

- “Agency Option-Adjusted Spreads” on page 3-2

# bkcall

---

**Purpose** Price European call option on bonds using Black's model

**Syntax** `CallPrice = bkcall(Strike, ZeroData, Sigma, BondData, Settle, Expiry, Period, Basis, EndMonthRule, InterpMethod, StrikeConvention)`

**Arguments**

Strike	Scalar or number of options (NOPT)-by-1 vector of strike prices.
ZeroData	Two-column (optionally three-column) matrix containing zero (spot) rate information used to discount future cash flows. <ul style="list-style-type: none"><li>• Column 1: Serial maturity date associated with the zero rate in the second column.</li><li>• Column 2: Annualized zero rates, in decimal form, appropriate for discounting cash flows occurring on the date specified in the first column. All dates must occur after <code>Settle</code> (dates must correspond to future investment horizons) and must be in ascending order.</li><li>• Column 3 (optional): Annual compounding frequency. Values are 1 (annual), 2 (semiannual, default), 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).</li></ul>
Sigma	Scalar or NOPT-by-1 vector of annualized price volatilities required by Black's model.

<b>BondData</b>	<p>Row vector with three (optionally four) columns or NOPT-by-3 (optionally NOPT-by-4) matrix specifying characteristics of underlying bonds in the form:</p> <p>[CleanPrice CouponRate Maturity Face]</p> <p>CleanPrice is the price excluding accrued interest.</p> <p>CouponRate is the decimal coupon rate.</p> <p>Maturity is the bond maturity date in serial date number format.</p> <p>Face is the face value of the bond. If unspecified, the face value is assumed to be 100.</p>
<b>Settle</b>	<p>Settlement date of the options. May be a serial date number or date string. Settle also represents the starting reference date for the input zero curve.</p>
<b>Expiry</b>	<p>Scalar or NOPT-by-1 vector of option maturity dates. May be a serial date number or date string.</p>
<b>Period</b>	<p>(Optional) Number of coupons per year for the underlying bond. Default = 2 (semiannual). Supported values are 0, 1, 2, 3, 4, 6, and 12.</p>

**Basis** (Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**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.

<b>InterpMethod</b>	(Optional) Scalar integer zero curve interpolation method. For cash flows that do not fall on a date found in the ZeroData spot curve, indicates the method used to interpolate the appropriate zero discount rate. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See <code>interp1</code> for more information.
<b>StrikeConvention</b>	<p>(Optional) Scalar or NOPT-by-1 vector of option contract strike price conventions.</p> <p><b>StrikeConvention = 0</b> (default) defines the strike price as the cash (dirty) price paid for the underlying bond.</p> <p><b>StrikeConvention = 1</b> defines the strike price as the quoted (clean) price paid for the underlying bond. When evaluating Black's model, the accrued interest of the bond at option expiration is added to the input strike price.</p>

## Description

`CallPrice = bkcall(Strike, ZeroData, Sigma, BondData, Settle, Expiry, Period, Basis, EndMonthRule, InterpMethod, StrikeConvention)` using Black's model, derives an NOPT-by-1 vector of prices of European call options on bonds.

If cash flows occur beyond the dates spanned by `ZeroData`, the input zero curve, the appropriate zero rate for discounting such cash flows is obtained by extrapolating the nearest rate on the curve (that is, if a cash flow occurs before the first or after the last date on the input zero curve, a flat curve is assumed).

In addition, you can use the Fixed-Income Toolbox method `getZeroRates` for an `IRDataCurve` object with a `Dates` property to create a vector of dates and data acceptable for `bkcall`. For more information, see "Converting an `IRDataCurve` or `IRFunctionCurve` Object" on page 6-25.

## Examples

This example is based on Example 22.1, page 512, of Hull. (See References below.)

Consider a European call option on a bond maturing in 9.75 years. The underlying bond has a clean price of \$935, a face value of \$1000, and pays 10% semiannual coupons. Since the bond matures in 9.75 years, a \$50 coupon will be paid in 3 months and again in 9 months. Also, assume that the annualized volatility of the forward bond price is 9%. Furthermore, suppose the option expires in 10 months and has a strike price of \$1000, and that the annualized continuously compounded risk-free discount rates for maturities of 3, 9, and 10 months are 9%, 9.5%, and 10%, respectively.

```
% Specify the option information.
Settle      = '15-Mar-2004';
Expiry      = '15-Jan-2005'; % 10 months from settlement
Strike      = 1000;
Sigma       = 0.09;
Convention  = [0 1]';

% Specify the interest-rate environment.
ZeroData    = [datenum('15-Jun-2004') 0.09 -1; % 3 months
               datenum('15-Dec-2004') 0.095 -1; % 9 months
               datenum(Expiry)        0.10 -1]; % 10 months

% Specify the bond information.
CleanPrice  = 935;
CouponRate  = 0.1;
Maturity    = '15-Dec-2013'; % 9.75 years from settlement
Face        = 1000;
BondData    = [CleanPrice CouponRate datenum(Maturity) Face];
Period      = 2;
Basis       = 1;

% Call Black's model.
CallPrices = bkcall(Strike, ZeroData, Sigma, BondData, Settle,...
Expiry, Period, Basis, [], [], Convention)
```

CallPrices =

9.4873

7.9686

When the strike price is the dirty price (`Convention = 0`), the call option value is \$9.49. When the strike price is the clean price (`Convention = 1`), the call option value is \$7.97.

## References

[1] Hull, John C., *Options, Futures, and Other Derivatives*, Prentice Hall, 5th edition, 2003, pp. 287-288, 508-515.

## See Also

bkput

# bkcaplet

---

**Purpose** Price interest-rate caplet using Black's model

**Syntax** `CapPrices = bkcaplet(CapData, FwdRates, ZeroPrice, Settle, StartDate, EndDate, Sigma)`

**Arguments**

CapData	Number of caps (NCAP)-by-2 matrix containing cap rates and bases: [CapRates Basis]. Values for bases may be: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul> For more information, see basis.
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)
```

# bkcaplet

---

CapPrice =

519.0046

## See Also

bkfloorlet

---

<b>Purpose</b>	Price interest-rate floorlet using Black's model				
<b>Syntax</b>	<code>FloorPrices = bkkfloorlet(FloorData, FwdRates, ZeroPrice, Settle, StartDate, EndDate, Sigma)</code>				
<b>Arguments</b>	<table><tr><td><code>FloorData</code></td><td>Number of floors (NFLR)-by-2 matrix containing floor rates and bases: [FloorRate Basis]. Values for bases may be:<ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul>For more information, see basis.</td></tr><tr><td><code>FwdRates</code></td><td>Scalar or NFLR-by-1 vector containing forward rates in decimal. <code>FwdRates</code> accrue on the same basis as <code>FloorRates</code>.</td></tr></table>	<code>FloorData</code>	Number of floors (NFLR)-by-2 matrix containing floor rates and bases: [FloorRate Basis]. Values for bases may be: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul> For more information, see basis.	<code>FwdRates</code>	Scalar or NFLR-by-1 vector containing forward rates in decimal. <code>FwdRates</code> accrue on the same basis as <code>FloorRates</code> .
<code>FloorData</code>	Number of floors (NFLR)-by-2 matrix containing floor rates and bases: [FloorRate Basis]. Values for bases may be: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul> For more information, see basis.				
<code>FwdRates</code>	Scalar or NFLR-by-1 vector containing forward rates in decimal. <code>FwdRates</code> accrue on the same basis as <code>FloorRates</code> .				

# bkfloorlet

---

ZeroPrice	Scalar or NFLR-by-1 vector containing zero coupon prices with maturities corresponding to those of each floor in FloorData, per \$100 nominal value.
Settle	Scalar or NFLR-by-1 vector of identical elements containing settlement date of floorlets.
StartDate	Scalar or NFLR-by-1 vector containing start dates of the floorlets.
EndDate	Scalar or NFLR-by-1 vector containing maturity dates of floorlets.
Sigma	Scalar or NFLR-by-1 vector containing volatility of forward rates in decimal, corresponding to each floorlet.

## Description

FloorPrices = bkfloorlet(FloorData, FwdRates, ZeroPrice, Settle, StartDate, EndDate, Sigma) computes the prices of interest-rate floorlets for every \$100 of notional value.

## Examples

Given a notional amount of \$1,000,000, compute the value of a floorlet on October 15, 2002 that starts on October 15, 2003 and ends on January 15, 2004.

```
FloorData = [0.08, 1];
FwdRates = 0.07;
ZeroPrice = 100*exp(-0.065*1.25);
Settle = datenum('15-Oct-2002');
BeginDates = datenum('15-Oct-2003');
EndDates = datenum('15-Jan-2004');
Sigma = 0.20;

% Because floorlet is $100 notional, divide $1,000,000 by $100.
Notional = 1000000/100;

FloorPrice = Notional*bkfloorlet(FloorData, FwdRates, ...
ZeroPrice, Settle, BeginDates, EndDates, Sigma)
```

```
FloorPrice =  
    2823.91
```

**See Also**      `bkcaplet`

# bkput

---

**Purpose** Price European put option on bonds using Black's model

**Syntax** `PutPrice = bkput(Strike, ZeroData, Sigma, BondData, Settle, Expiry, Period, Basis, EndMonthRule, InterpMethod, StrikeConvention)`

**Arguments**

Strike	Scalar or number of options (NOPT)-by-1 vector of strike prices.
ZeroData	Two-column (optionally three-column) matrix containing zero (spot) rate information used to discount future cash flows. <ul style="list-style-type: none"><li>• Column 1: Serial maturity date associated with the zero rate in the second column.</li><li>• Column 2: Annualized zero rates, in decimal form, appropriate for discounting cash flows occurring on the date specified in the first column. All dates must occur after <code>Settle</code> (dates must correspond to future investment horizons) and must be in ascending order.</li><li>• Column 3 (optional): Annual compounding frequency. Values are 1 (annual), 2 (semiannual, default), 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).</li></ul>
Sigma	Scalar or NOPT-by-1 vector of annualized price volatilities required by Black's model.

---

<b>BondData</b>	<p>Row vector with three (optionally four) columns or NOPT-by-3 (optionally NOPT-by-4) matrix specifying characteristics of underlying bonds in the form [CleanPrice CouponRate Maturity Face] where:</p> <ul style="list-style-type: none"><li>• <b>CleanPrice</b> is the price excluding accrued interest.</li><li>• <b>CouponRate</b> is the decimal coupon rate.</li><li>• <b>Maturity</b> is the bond maturity date in serial date number format.</li><li>• <b>Face</b> is the face value of the bond. If unspecified, the face value is assumed to be 100.</li></ul>
<b>Settle</b>	<p>Settlement date of the options. May be a serial date number or date string. <b>Settle</b> also represents the starting reference date for the input zero curve.</p>
<b>Expiry</b>	<p>Scalar or NOPT-by-1 vector of option maturity dates. May be a serial date number or date string.</p>
<b>Period</b>	<p>(Optional) Number of coupons per year for the underlying bond. Default = 2 (semiannual). Supported values are 0, 1, 2, 3, 4, 6, and 12.</p>

Basis	<p>(Optional) Day-count basis of the bond. A vector of integers.</p> <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul>
EndMonthRule	<p>For more information, see basis.</p> <p>(Optional) End-of-month rule. This rule applies only when <code>Maturity</code> 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.</p>



<b>InterpMethod</b>	(Optional) Scalar integer zero curve interpolation method. For cash flows that do not fall on a date found in the <code>ZeroData</code> spot curve, indicates the method used to interpolate the appropriate zero discount rate. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See <code>interp1</code> for more information.
<b>StrikeConvention</b>	(Optional) Scalar or NOPT-by-1 vector of option contract strike price conventions.  <b>StrikeConvention = 0</b> (default) defines the strike price as the cash (dirty) price paid for the underlying bond.  <b>StrikeConvention = 1</b> defines the strike price as the quoted (clean) price paid for the underlying bond. The accrued interest of the bond at option expiration is added to the input strike price when evaluating Black's model.

## Description

`PutPrice = bkput(Strike, ZeroData, Sigma, BondData, Settle, Expiry, Period, Basis, EndMonthRule, InterpMethod, StrikeConvention)` using Black's model, derives an NOPT-by-1 vector of prices of European put options on bonds.

If cash flows occur beyond the dates spanned by `ZeroData`, the input zero curve, the appropriate zero rate for discounting such cash flows is obtained by extrapolating the nearest rate on the curve (that is, if a cash flow occurs before the first or after the last date on the input zero curve, a flat curve is assumed).

In addition, you can use the Fixed-Income Toolbox method `getZeroRates` for an `IRDataCurve` object with a `Dates` property to create a vector of dates and data acceptable for `bkput`. For more information, see "Converting an `IRDataCurve` or `IRFunctionCurve` Object" on page 6-25.

## Examples

This example is based on example 22.2, page 514, of Hull. (See References below.)

Consider a European put option on a bond maturing in 10 years. The underlying bond has a clean price of \$122.82, a face value of \$100, and pays 8% semiannual coupons. Also, assume that the annualized volatility of the forward bond yield is 20%. Furthermore, suppose the option expires in 2.25 years and has a strike price of \$115, and that the annualized continuously compounded risk free zero (spot) curve is flat at 5%. For a hypothetical settlement date of March 15, 2004, the following code illustrates the use of Black's model to duplicate the put prices in Example 22.2 of the Hull reference. In particular, it illustrates how to convert a broker's yield volatility to a price volatility suitable for Black's model.

```
% Specify the option information.
Settle      = '15-Mar-2004';
Expiry      = '15-Jun-2006'; % 2.25 years from settlement
Strike      = 115;
YieldSigma  = 0.2;
Convention  = [0; 1];

% Specify the interest-rate environment. Since the
% zero curve is flat, interpolation into the curve always returns
% 0.05. Thus, the following curve is not unique to the solution.
ZeroData    = [datenum('15-Jun-2004') 0.05 -1;
               datenum('15-Dec-2004') 0.05 -1;
               datenum(Expiry)      0.05 -1];

% Specify the bond information.
CleanPrice  = 122.82;
CouponRate  = 0.08;
Maturity    = '15-Mar-2014'; % 10 years from settlement
Face        = 100;
BondData    = [CleanPrice CouponRate datenum(Maturity) Face];
Period      = 2; % semiannual coupons
Basis       = 1; % 30/360 day-count basis
```

```

% Convert a broker's yield volatility quote to a price volatility
% required by Black's model. To duplicate Example 22.2 in Hull,
% first compute the periodic (semiannual) yield to maturity from
% the clean bond price.
Yield = bndyield(CleanPrice, CouponRate, Settle, Maturity,...
Period, Basis);

% Compute the duration of the bond at option expiration. Most
% fixed-income sensitivity analyses use the modified duration
% statistic to examine the impact of small changes in periodic
% yields on bond prices. However, Hull's example operates in
% continuous time (annualized instantaneous volatilities and
% continuously compounded zero yields for discounting coupons).
% To duplicate Hull's results, use the second output of BNDDURY,
% the Macaulay duration.
[Modified, Macaulay] = bnddury(Yield, CouponRate, Expiry,...
Maturity, Period, Basis);

% Convert the yield-to-maturity from a periodic to a
% continuous yield.
Yield = Period .* log(1 + Yield./Period);

% Finally, convert the yield volatility to a price volatility via
% Hull's Equation 22.6 (page 514).
PriceSigma = Macaulay .* Yield .* YieldSigma;

% Finally, call Black's model.
PutPrices = bkput(Strike, ZeroData, PriceSigma, BondData,...
Settle, Expiry, Period, Basis, [], [], Convention)
PutPrices =

    1.7838
    2.4071

```

When the strike price is the dirty price (Convention = 0), the call option value is \$1.78. When the strike price is the clean price (Convention = 1), the call option value is \$2.41.

## References

[1] Hull, John C., *Options, Futures, and Other Derivatives*, Prentice Hall, 5th edition, 2003, pp. 287-288, 508-515.

## See Also

bkcall

<b>Purpose</b>	Implied repo rates for bond future given price
<b>Syntax</b>	<pre>ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle, Delivery, ConvFactor, CouponRate, Maturity) ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle, Delivery, ConvFactor, CouponRate, Maturity, 'ParameterName', 'ParameterValue ...)</pre>
<b>Description</b>	<p>ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle, Delivery, ConvFactor, CouponRate, Maturity) computes the implied repo rate for a bond future given the price of a bond, the bond properties, the price of the bond future, and the bond conversion factor.</p> <p>ImpRepo = bndfutimprepo(Price, FutPrice, FutSettle, Delivery, ConvFactor, CouponRate, Maturity, 'ParameterName', 'ParameterValue ... ) accepts optional inputs as one or more comma-separated parameter/value pairs. 'ParameterName' is the name of the parameter inside single quotes. ParameterValue is the value corresponding to 'ParameterName'. Specify parameter-value pairs in any order. Names are case-insensitive.</p>
<b>Input Arguments</b>	<p>Price numBonds-by-1 vector of bond prices.</p> <p>FutPrice numBonds-by-1 vector of future prices</p> <p>FutSettle numBonds-by-1 vector of future settle dates.</p> <p>Delivery numBonds-by-1 vector of future delivery dates.</p>

## ConvFactor

numBonds-by-1 vector of bond conversion factors. For more information, see `convfactor`.

## CouponRate

numBonds-by-1 vector of coupon rates in decimal form.

## Maturity

numBonds-by-1 vector of coupon rates in decimal form.

## Parameter-Value Pairs

### Basis

Day-count basis. Possible values include

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**Default:** 0

EndMonthRule

End-of-month rule. Values are:

- 0 — Rule is not in effect for the bond.
- 1 — Rule is in effect for the bond. This means that a security that pays coupon interest on the last day of the month always makes payment on the last day of the month.

**Default:** 1

Face

Face value of the bond. Face has no impact on key rate duration. This calling sequence is preserved for consistency.

**Default:** 100

FirstCouponDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

**Default:** If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

IssueDate

Issue date for a bond.

LastCouponDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

**Default:** If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.

Period

Number of coupons payments per year. Possible values include:

- 0
- 1
- 2
- 3
- 4
- 6
- 12

**Default:** 2

ReinvestBasis

Day count basis for reinvestment rate.

**Default:** Identical to RepoBasis.

ReinvestRate

Rate for reinvesting intermediate coupons from the bond.

**Default:** Identical to ImpRepo.



RepoBasis

Day count basis for ImpRepo.

**Default:** 2

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify StartDate, the effective start date is the Settle date.

## Output Arguments

ImpRepo

Implied repo rate, or the repo rate that would produce the price input.

## Definitions

bndfutimprepo computes the implied repo rate for a bond future given:

- Price of a bond
- Bond properties
- Price of the bond future
- Bond conversion factor

The default behavior is that the coupon reinvestment rate matches the repo rate. However, you can specify a separate reinvestment rate using optional inputs.

## Examples

Compute the repro rate for a bond future:

```
bndfutimprepo(129,98, '9/21/2000', '12/29/2000', 1.3136, .0875, '8/15/2020')
```

This returns:

```
ans =  
    0.0584
```

## References

Burghardt, G., T. Belton, M. Lane, and J. Papa, *The Treasury Bond Basis*, McGraw-Hill, 2005.

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

## See Also

bndfutprice | convfactor

## How To

- “Bond Futures” on page 4-12

<b>Purpose</b>	Price bond future given repo rates
<b>Syntax</b>	<pre>[FutPrice,AccrInt] = bndfutprice(RepoRate, Price, FutSettle, Delivery, ConvFactor, CouponRate, Maturity) FutPrice,AccrInt] = bndfutprice(RepoRate, FutPrice, FutSettle, Delivery, ConvFactor, CouponRate, Maturity, 'ParameterName','ParameterValue ...)</pre>
<b>Description</b>	<p>[FutPrice,AccrInt] = bndfutprice(RepoRate, Price, FutSettle, Delivery, ConvFactor, CouponRate, Maturity) computes the price of a bond futures contract for one or more bonds given a repo rate, and bond properties, including the bond conversion factor.</p> <p>FutPrice,AccrInt] = bndfutprice(RepoRate, FutPrice, FutSettle, Delivery, ConvFactor, CouponRate, Maturity, 'ParameterName','ParameterValue ...) accepts optional inputs as one or more comma-separated parameter/value pairs. 'ParameterName' is the name of the parameter inside single quotes. ParameterValue is the value corresponding to 'ParameterName'. Specify parameter-value pairs in any order. Names are case-insensitive.</p>
<b>Input Arguments</b>	<p>RepoRate numBonds-by-1 vector of repo rates.</p> <p>Price numBonds-by-1 vector of bond prices</p> <p>FutSettle numBonds-by-1 vector of future settle dates.</p> <p>Delivery numBonds-by-1 vector of future delivery dates.</p>

## ConvFactor

numBonds-by-1 vector of bond conversion factors. For more information, see `convfactor`.

## CouponRate

numBonds-by-1 vector of coupon rates in decimal form.

## Maturity

numBonds-by-1 vector of coupon rates in decimal form.

## Parameter-Value Pairs

### Basis

Day-count basis. Possible values include

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**Default:** 0

EndMonthRule

End-of-month rule. Values are:

- 0 — Rule is not in effect for the bond.
- 1 — Rule is in effect for the bond. This means that a security that pays coupon interest on the last day of the month always makes payment on the last day of the month.

**Default:** 1

IssueDate

Issue date for a bond.

Face

Face value of the bond. Face has no impact on key rate duration. This calling sequence is preserved for consistency.

**Default:** 100

FirstCouponDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

**Default:** If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

LastCouponDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

**Default:** If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.

Period

Number of coupons payments per year. Possible values include:

- 0
- 1
- 2
- 3
- 4
- 6
- 12

**Default:** 2

ReinvestBasis

Day count basis for reinvestment rate.

**Default:** Identical to RepoBasis.

ReinvestRate

Compounding convention for reinvestment rate.

**Default:** Identical to RepoRate.

RepoBasis

Day count basis for RepoRate.

**Default:** 2

StartDate

Date when a bond actually starts (the date from which a bond cash flow is considered). To make an instrument forward-starting, specify this date as a future date. If you do not specify StartDate, the effective start date is the Settle date.

## Output Arguments

FutPrice

Quoted futures price, per \$100 notional.

AccrInt

Accrued interest due at delivery date, per \$100 notional.

## Definitions

bndfutprice computes the price of a bond futures contract for one or more bonds, given a repo rate, and bond properties, including the bond conversion factor. The default behavior is that the coupon reinvestment rate matches the repo rate. However, you can specify a separate reinvestment rate using optional inputs.

## Examples

Compute the price for a bond future:

```
bndfutprice(.064, 129, '9/21/2000', '12/29/2000', 1.3136, .0875, '8/15/2020')
```

The returns:

```
ans =  
    98.1516
```

## References

Burghardt, G., T. Belton, M. Lane, and J. Papa, *The Treasury Bond Basis*, McGraw-Hill, 2005.

# bndfutprice

---

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

## **See Also**

bndfutimprepo | convfactor

## **How To**

- “Bond Futures” on page 4-12



---

<b>Purpose</b>	Bootstrap interest-rate curve from market data	
<b>Class</b>	@IRDataCurve	
<b>Syntax</b>	<pre>Dcurve = IRDataCurve.bootstrap(Type, Settle, InstrumentTypes, Instruments) Dcurve = IRDataCurve.bootstrap(Type, Settle, InstrumentTypes, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...)</pre>	
<b>Arguments</b>	Type	Type of interest-rate curve. Acceptable values are: discount, forward, or zero.
	Settle	Scalar or column vector of settlement dates.
	InstrumentTypes	N-by-1 cell array (where N is the number of instruments) indicating what kind of instrument is in the Instruments matrix. Acceptable values are deposit, futures, swap, and bond.
	Instruments	N-by-3 data matrix for Instruments where the first column is Settle date, the second column is Maturity, and the third column is the market quote (dates must be MATLAB date numbers).

---

**Note** The market quote represents the following for each instrument:

- deposit: rate
  - futures: price (e.g., 9628.54)
  - swap: rate
  - bond: clean price
- 

## Compounding

(Optional) Scalar that sets the compounding frequency per year for an IRDataCurve object:

- -1 = Continuous compounding
- 1 = Annual compounding
- 2 = Semiannual compounding (default)
- 3 = Compounding three times per year
- 4 = Quarterly compounding
- 6 = Bimonthly compounding
- 12 = Monthly compounding

**Basis**

(Optional) Day-count basis of the interest-rate curve. A scalar of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**InterpMethod**

(Optional) Values are:

- 'linear' — Linear interpolation (default).
- 'constant' — Piecewise constant interpolation.
- 'pchip' — Piecewise cubic Hermite interpolation.

- 'spline' — Cubic spline interpolation.

IRBootstrapOptionsObj (Optional) An IRBootstrapOptions object.

## Instrument Parameters

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

<code>CouponRate</code>	(Optional) Decimal number indicating the annual percentage rate used to determine the coupons payable on an instrument.
<code>Period</code>	(Optional) Coupons per year of the instrument. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.
<code>Basis</code>	(Optional) Day-count basis of the instrument. A vector of integers. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li></ul>

- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**EndMonthRule**

(Optional) End-of-month rule. A vector. This rule applies only when **Maturity** is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that an instrument's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that an instrument's coupon payment date is always the last actual day of the month.

**IssueDate**

(Optional) Date when an instrument was issued.

**FirstCouponDate**

(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When **FirstCouponDate** and **LastCouponDate** are both specified, **FirstCouponDate** takes precedence in determining the coupon payment structure. If you do not specify a **FirstCouponDate**, the cash flow payment dates are determined from other inputs.

LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.
Face	(Optional) Face or par value. Default = 100.

---

**Note** When using Instrument parameter/value pairs, you can specify simple interest for an Instrument by specifying the InstrumentPeriod value as 0. If InstrumentBasis and InstrumentPeriod are not specified for an Instrument, the following default values are used:

- deposit instrument uses Basis as 2 (act/360) and Period is 0 (simple interest).
  - futures instrument uses Basis as 2 (act/360) and Period is 4 (quarterly).
  - swap instrument uses Basis as 2 (act/360) and Period is 2.
  - bond instrument uses Basis as 0 (act/act) and Period is 2.
- 

## Description

Dcurve = IRDataCurve.bootstrap(Type, Settle, InstrumentTypes, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...) bootstraps an interest-rate curve from market data. The dates of the bootstrapped curve correspond to the maturity dates of the input instruments. You must enter the optional arguments for Basis, Compounding, Interpmethod, and IRBootstrapOptionsObj as parameter/value pairs.

## Examples

In this bootstrapping example, InstrumentTypes, Instruments, and a Settle date are defined:

```
InstrumentTypes = {'Deposit';'Deposit';...
'Futures';'Futures';'Futures';'Futures';'Futures';'Futures';...
'Swap';'Swap';'Swap';'Swap'};

Instruments = [datenum('08/10/2007'),datenum('09/17/2007'),.0532000; ...
datenum('08/10/2007'),datenum('11/17/2007'),.0535866; ...
datenum('08/08/2007'),datenum('19-Dec-2007'),9485; ...
datenum('08/08/2007'),datenum('19-Mar-2008'),9502; ...
datenum('08/08/2007'),datenum('18-Jun-2008'),9509.5; ...
datenum('08/08/2007'),datenum('17-Sep-2008'),9509; ...
datenum('08/08/2007'),datenum('17-Dec-2008'),9505.5; ...
datenum('08/08/2007'),datenum('18-Mar-2009'),9501; ...
datenum('08/08/2007'),datenum('08/08/2014'),.0530; ...
datenum('08/08/2007'),datenum('08/08/2019'),.0551; ...
datenum('08/08/2007'),datenum('08/08/2027'),.0565; ...
datenum('08/08/2007'),datenum('08/08/2037'),.0566];

CurveSettle = datenum('08/10/2007');
```

Use the bootstrap method to create an IRDataCurve object.

```
bootModel = IRDataCurve.bootstrap('Forward', CurveSettle, ...
InstrumentTypes, Instruments,'InterpMethod','pchip')
```

```
bootModel =
```

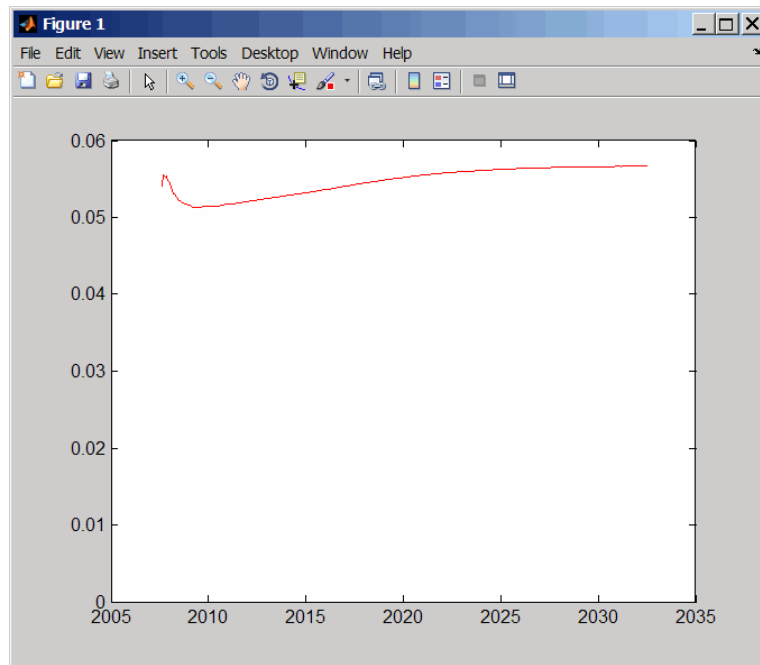
```
IRDataCurve
```

```

Type: Forward
Settle: 733264 (10-Aug-2007)
Compounding: 2
Basis: 0 (actual/actual)
InterpMethod: pchip
Dates: [12x1 double]
Data: [12x1 double]
```

To create the plot for the bootstrapped market data:

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



For an example of bootstrapping using instrument parameters support for prepending the word `Instrument` to the parameter field, see “Using `IRDataCurve` bootstrap Method for Bootstrapping Based on Market Instruments” on page 6-7.

## How To

- “`@IRDataCurve`” on page A-7
- “`@IRBootstrapOptions`” on page A-2



<b>Purpose</b>	Price convertible bond
<b>Syntax</b>	<pre>[CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] = cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ConvRatio, NumSteps, IssueDate, Settle, Maturity, CouponRate) [CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] = cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ConvRatio, NumSteps, IssueDate, Settle, Maturity, CouponRate, Name,Value)</pre>
<b>Description</b>	<pre>[CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] = cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ConvRatio, NumSteps, IssueDate, Settle, Maturity, CouponRate) price a convertible bond with a one-factor lattice method.  [CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] = cbprice(RiskFreeRate, StaticSpread, Sigma, Price, ConvRatio, NumSteps, IssueDate, Settle, Maturity, CouponRate, Name,Value) price a convertible bond with a one-factor lattice method with additional options specified by one or more Name,Value pair arguments.</pre>
<b>Input Arguments</b>	<p><b>RiskFreeRate</b> Annual yield of the risk-free bond with the same maturity as the convertible, compounded continuously. Scalar value of risk-free rates is in decimal. (Recommended value is the yield of a risk-free bond with the same maturity as the convertible.)</p> <p><b>StaticSpread</b> Scalar value of the constant spread to risk-free rate. Adding StaticSpread to the RiskFreeRate produces the issuer's yield, which reflects the credit risk.</p> <p><b>Sigma</b> Scalar value of the annual volatility environment in decimal.</p>

## Price

Scalar value of the price of the asset at the settlement or valuation date.

## ConvRatio

Scalar value of the number of assets convertible to one bond.

## NumSteps

Scalar value of the number of steps within the binomial tree.

## IssueDate

Scalar value of the issue date of the convertible bond.

## Settle

Scalar value of the settlement date of the convertible bond.

## Maturity

Scalar value of the maturity date of the convertible bond.

## CouponRate

Scalar value of the coupon rate in decimal form or a C-by-2 vector of dates and associated coupon rates.

## **Name-Value Pair Arguments**

Specify optional comma-separated pairs of **Name**, **Value** arguments, where **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1, Value1, ..., NameN, ValueN**.

## Basis

Day-count basis of the bond. A vector of integers.

- 0 = actual/actual

- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**Default:** 0 (actual/actual)

#### CallType

Scalar value for the call type. Values are 0 for a call on cash price, or 1 for a call on clean price.

**Default:** 0 (cash price)

#### CallInfo

Two-column matrix containing the call information. The first column is the call dates and the second column is the call prices for every \$100 face of the bond. The call, in the amount of call prices, is activated *after* the corresponding call date.

**Default:** No call feature

## ConvInfo

Two-column matrix containing convertible information. The first column is the convertible dates and the second column is whether the issue is convertible or not.

**Default:** Bond is always convertible

## DividendInfo

Two-column matrix of dividend information. The first column is the ex-dividend date and the second column is the corresponding amount. Enter any amount known at any time; only the amounts that are within the lifespan of the option are used. If the `DividendType` is 2, `DividendInfo` is a 1-by-2 matrix where the first entry is the `Settle` date and the second entry is the continuous dividend yield.

**Default:** No dividend

## DividendType

Scalar value for dividend type. Values are:

- 0 — Dollar dividend
- 1 — Dividend yield
- 2 — Continuous dividend yield

**Default:** 0 (Dollar dividend)

## EndMonthRule

NINST-by-1 vector for end-of-month rule. Values are 1 (on, in effect) and 0 (off, not in effect).

**Default:** 1 (on, in effect)

## Period

Scalar value for number of coupon payments. Values are:

- 1 — One coupon per year
- 2 — Semiannual
- 3 — Three times a year
- 4 — Quarterly
- 6 — Bimonthly compounding
- 12 — Monthly

**Default:** 2 (Semiannual)

#### IssueDate

NINST-by-1 vector of bond issue date.

**Default:** If you do not specify an `IssueDate`, the cash flow payment date is determined from other inputs.

#### FirstCouponDate

Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

**Default:** If you do not specify a `FirstCouponDate`, the cash flow payment dates are determined from other inputs.

#### LastCouponDate

Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate`, regardless of where it falls, and is followed only by the bond's maturity cash flow date.

**Default:** If you do not specify a `LastCouponDate`, the cash flow payment dates are determined from other inputs.

`Period`

NINST-by-1 vector for coupons per year.

**Default:** 2 per year

`PutInfo`

Two-column matrix containing put information. The first column is the put dates and the second column is the put prices for every \$100 face of the bond. The put, in the amount of put prices, is activated *after* the corresponding put date.

**Default:** No put feature

`PutType`

Scalar value for put type. Value are 0 for a put on cash price or 1 for a put on clean price.

**Default:** 0 (put on cash price)

`TreeType`

Scalar value for tree type. Values are 0 for binomial lattice or 1 for trinomial lattice.

**Default:** 0 (binomial lattice)

## Output Arguments

`CbMatrix`

Matrix of CB prices in binomial format. Price of convertible is `CbMatrix(1,1)`.

`UndMatrix`

Matrix of stock prices in binomial format.

DebtMatrix

Matrix of CB debt component in binomial format.

EqtyMatrix

Matrix of CB equity component in binomial format.

## Definitions

### Convertible Bond

A convertible bond (CB) is a debt instrument that you can convert into a predetermined amount of the issuing company's equity at certain times before the bond's maturity. In addition to standard bond features (for example, maturity date, face value, coupon), a convertible bond often has callable and puttable features.

## Examples

Perform a spread effect analysis of a 4% coupon convertible bond callable at 110 at the end of the second year, maturing at par in 5 years, with yield to maturity of 5%, and spread (of yield to maturity versus 5-year treasury) of 0, 50, 100, and 150 basis points. The underlying stock pays no dividend.

```
RiskFreeRate = 0.05;
Sigma        = 0.3;
ConvRatio    = 1;
NumSteps     = 200;
IssueDate    = '2-Jan-2002';
Settle       = '2-Jan-2002';
Maturity     = '2-Jan-2007';
CouponRate   = 0.04;
Period       = 2;
Basis        = 1;
EndMonthRule = 1;
DividendType = 0;
DividendInfo = [];
CallInfo     = [datenum('2-Jan-2004') , 110];
CallType     = 1;
TreeType     = 1;
```

```
Spreads = 0:0.005:0.015;
Prices = 40:10:140;
stock = repmat(Prices',1,length(Spreads));

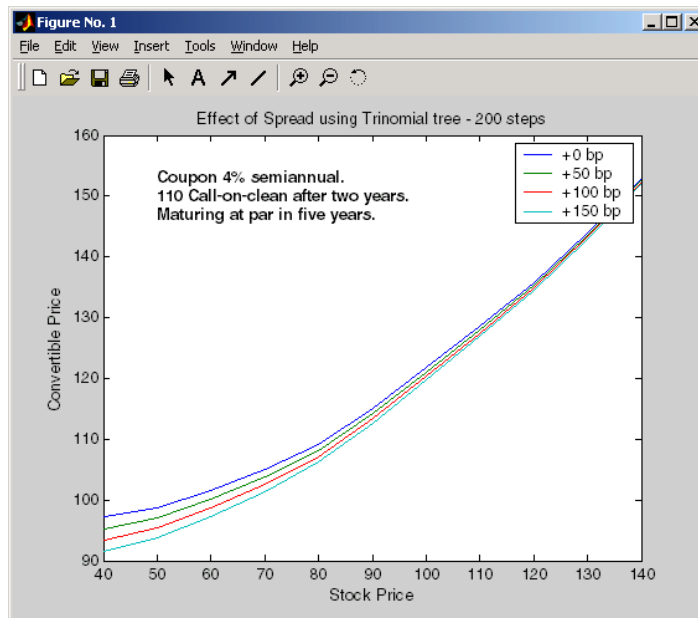
convprice = zeros(length(Prices),length(Spreads));

for spreadidx = 1:length(Spreads)
    for priceidx = 1:length(Prices)
        [CbMatrix, UndMatrix, DebtMatrix, EqtyMatrix] = ...
            cbprice(RiskFreeRate, Spreads(spreadidx), Sigma, Prices(priceidx), ...
                ConvRatio, NumSteps, IssueDate, Settle, ...
                Maturity, CouponRate, Period, Basis, EndMonthRule, ...
                DividendType, DividendInfo, CallType, CallInfo, TreeType);

        convprice(priceidx,spreadidx) = CbMatrix(1,1);
    end
end

plot(stock,convprice);
legend({'+0 bp'; '+50 bp'; '+100 bp'; '+150 bp'});
title ('Effect of Spread using Trinomial tree - 200 steps')
xlabel('Stock Price');
ylabel('Convertible Price');
text(50, 150, ['Coupon 4 semiannual,', sprintf('\n'), ...
    '110 Call-on-clean after 2 years,' sprintf('\n'), ...
    'maturing par in 5 years'],'fontweight','Bold')
```





## References

Andersen, L. and D. Buffum, "Calibration and implementation of convertible bonds models," Working paper, Banc of America Securities, 2003.

Ayache, E., P.A. Forsyth, and K.R. Vetzal, "Valuation of Convertible Bonds with Credit Risk," *Journal of Derivatives*, 11 (Fall 2003), 9-29.

Tsiveriotis, K. and C. Fernandes, "Valuing Convertible Bonds with Credit Risk," *Journal of Fixed Income* 8, 95-102, 1998

Zabolotnyuk, Yuriy, Jones, Robert A. and Veld, Chris H., "An Empirical Comparison of Convertible Bond Valuation Models," (October 15, 2009). Available at SSRN: <http://ssrn.com/abstract=994805>.

## Tutorials

- "Convertible Bond Valuation" on page 4-10

# cdai

---

**Purpose** Accrued interest on certificate of deposit

**Syntax** `AccrInt = cdai(CouponRate, Settle, Maturity, IssueDate, Basis)`

## Arguments

CouponRate	Annual interest rate in decimal.
Settle	Settlement date. <code>Settle</code> must be earlier than <code>Maturity</code> .
Maturity	Maturity date.
IssueDate	Issue date.
Basis	(Optional) Day-count basis of the instrument. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul> For more information, see basis.

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

## Description

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

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

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

## Examples

Given a certificate of deposit with these characteristics, compute the accrued interest due.

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

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

```
AccrInt =
```

```
    1.2917
```

## See Also

`accrfrac` | `bndyield` | `stepcpnyield` | `tbillyield` | `zeroyield`

# cdprice

---

**Purpose** Price of certificate of deposit

**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 Maturity.
Maturity	Maturity date.
IssueDate	Issue date.
Basis	(Optional) Day-count basis of the instrument. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li></ul>

- 13 = BUS/252

For more information, see basis.

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

## Description

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

`Price` is the clean price of the certificate of deposit per \$100 of face value.

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

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

## Examples

Given a certificate of deposit with these characteristics, compute the price 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
```

# cdprice

---

AccruedInt =

1.2917

## See Also

bndprice | cdai | cdyield | stepcpnprice | tbillprice

**Purpose** Bootstrap default probability curve from credit default swap market quotes

**Syntax** `[ProbData, HazData] = cdsbootstrap(ZeroData, MarketData, Settle)`  
`[ProbData, HazData] = cdsbootstrap(ZeroData, MarketData, Settle, Name, Value)`

**Description** `[ProbData, HazData] = cdsbootstrap(ZeroData, MarketData, Settle)` bootstraps the default probability curve using credit default swap (CDS) market quotes. The market quotes can be expressed as a list of maturity dates and corresponding CDS market spreads, or as a list of maturities and corresponding upfronts and standard spreads for standard CDS contracts. The estimation uses the standard model of the survival probability.

`[ProbData, HazData] = cdsbootstrap(ZeroData, MarketData, Settle, Name, Value)` bootstraps the default probability curve using CDS market quotes with additional options specified by one or more `Name, Value` pair arguments. The market quotes can be expressed as a list of maturity dates and corresponding CDS market spreads, or as a list of maturities and corresponding upfronts and standard spreads for standard CDS contracts. The estimation uses the standard model of the survival probability.

**Input Arguments** `ZeroData`  
M-by-2 vector of dates and zero rates or `IRCurve` of zero rates.

`MarketData`  
N-by-2 matrix of dates and corresponding market spreads or N-by-2 matrix of dates, upfronts, and standard spreads of CDS contracts.

`Settle`  
Settlement date is a serial date number or date string. This must be earlier than or equal to the dates in `MarketData`.

## Name-Value Pair Arguments

Specify optional comma-separated pairs of `Name`, `Value` arguments, where `Name` is the argument name and `Value` is the corresponding value. `Name` must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as `Name1,Value1,...,NameN,ValueN`.

---

**Note** Any optional input of size N-by-1 is also acceptable as an array of size 1-by-N, or as a single value applicable to all contracts. Single values are internally expanded to an array of size N-by-1.

---

### Basis

N-by-1 vector of day-count basis of the CDS:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252



For more information, see basis.

**Default:** 2 (actual/360)

BusDayConvention

String or N-by-1 cell array of strings of business day conventions.

Values are:

- actual
- follow
- modifiedfollow
- previous
- modifiedprevious

**Default:** actual

PayAccruedPremium

N-by-1 vector of Boolean flags, True (default), if accrued premiums are paid upon default, False otherwise.

**Default:** True

Period

N-by-1 vector of the number of premiums per year of the CDS. Allowed values are 1, 2, 3, 4, 6, and 12.

**Default:** 4

ProbDates

P-by-1 vector of dates for ProbData.

**Default:** Column of dates in MarketData

## RecoveryRate

N-by-1 vector of recovery rates, expressed as a decimal from 0 to 1.

**Default:** 0.4

## TimeStep

Positive integer indicating the number of days to take as time step for the numerical integration.

**Default:** 10 (days)

## ZeroBasis

Basis of the zero curve. Choices are identical to **Basis**.

**Default:** 0 (actual/actual)

## ZeroCompounding

Compounding frequency of the zero curve. Allowed values are:

- 1 — Annual compounding
- 2 — Semiannual compounding
- 3 — Compounding three times per year
- 4 — Quarterly compounding
- 6 — Bimonthly compounding
- 12 — Monthly compounding
- -1 — Continuous compounding

---

**Note** When ZeroData is an IRCurve object, the arguments ZeroCompounding and ZeroBasis are implicit in ZeroData and are redundant inside this function. In that case, specify these optional arguments when constructing the IRCurve object before calling this function.

---

**Default:** 2 (Semiannual compounding)

## Output Arguments

ProbData

P-by-2 matrix with dates and corresponding cumulative default probability values. The dates match those in MarketData, unless the optional input parameter ProbDates is provided.

HazData

N-by-2 matrix with dates and corresponding hazard rate values for the standard survival probability model. The dates match those in MarketData.

---

**Note** A warning is displayed when non-monotone default probabilities (i.e., negative hazard rates) are found.

---

## Examples

Use cdsbootstrap with market quotes for CDS contracts to generate ProbData and HazData values:

```
Settle = '17-Jul-2009';
Spread_Time = [1 2 3 5 7]';
Spread = [140 175 210 265 310]';
Market_Dates = daysadd(datenum(Settle),360*Spread_Time,1);
MarketData = [Market_Dates Spread];
Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
Zero_Dates = daysadd(datenum(Settle),360*Zero_Time,1);
```

```
ZeroData = [Zero_Dates Zero_Rate];

[ProbData,HazData] = cdsbootstrap(ZeroData,MarketData,Settle)
ProbData =

    1.0e+005 *

    7.3434    0.0000
    7.3470    0.0000
    7.3507    0.0000
    7.3580    0.0000
    7.3653    0.0000

HazData =

    1.0e+005 *

    7.3434    0.0000
    7.3470    0.0000
    7.3507    0.0000
    7.3580    0.0000
    7.3653    0.0000
```

## Algorithms

If the time to default is denoted by  $\tau$ , the default probability curve, or function,  $PD(t)$ , and its complement, the survival function  $Q(t)$ , are given by:

$$PD(t) = P[\tau \leq t] = 1 - P[\tau > t] = 1 - Q(t)$$

In the standard model, the survival probability is defined in terms of a piecewise constant hazard rate  $h(t)$ . For example, if  $h(t) =$

$$\lambda_1, \text{ for } 0 \leq t \leq t_1$$

$$\lambda_2, \text{ for } t_1 < t \leq t_2$$

$$\lambda_3, \text{ for } t_2 < t$$

then the survival function is given by  $Q(t) =$

$$e^{-\lambda_1 t}, \text{ for } 0 \leq t \leq t_1$$

$$e^{-\lambda_1 t - \lambda_2(t-t_1)}, \text{ for } t_1 < t \leq t_2$$

$$e^{-\lambda_1 t_1 - \lambda_2(t_2-t_1) - \lambda_3(t-t_2)}, \text{ for } t_2 < t$$

Given  $n$  market dates  $t_1, \dots, t_n$  and corresponding market CDS spreads  $S_1, \dots, S_n$ , `cdsbootstrap` calibrates the parameters  $\lambda_1, \dots, \lambda_n$  and evaluates  $PD(t)$  on the market dates, or an optional user-defined set of dates.

## References

Beumee, J., D. Brigo, D. Schiemert, and G. Stoyale. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research, Global Special Report*. April 7, 2009.

Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives* 8, 29-40.

O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps." *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.

## See Also

| `cdsspread` | `cdsprice`

## Tutorials

- "Credit Default Swap (CDS)" on page 5-2

# cdsoptprice

---

<b>Purpose</b>	Price payer and receiver credit default swaptions
<b>Syntax</b>	<pre>[Payer, Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, CDSMaturity, Strike, SpreadVol) [Payer, Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, CDSMaturity, Strike, SpreadVol, Name,Value)</pre>
<b>Description</b>	<p>[Payer, Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, CDSMaturity, Strike, SpreadVol) computes the price of payer and receiver credit default swaptions.</p> <p>[Payer, Receiver] = cdsoptprice(ZeroData, ProbData, Settle, OptionMaturity, CDSMaturity, Strike, SpreadVol, Name,Value) computes the price of payer and receiver credit default swaptions with additional options specified by one or more Name,Value pair arguments.</p>
<b>Input Arguments</b>	<p><b>ZeroData</b> M-by-2 vector of dates and zero rates or IRCurve of zero rates.</p> <p><b>ProbData</b> P-by-2 array of dates and default probabilities.</p> <p><b>Settle</b> Settlement date is a serial date number or date string. Settle must be earlier than the maturity date.</p> <p><b>OptionMaturity</b> N-by-1 vector of serial date numbers or date strings containing the option maturity dates.</p> <p><b>CDSMaturity</b> N-by-1 vector of serial date numbers or date strings containing the CDS maturity dates.</p>

**Strike**

N-by-1 vector of option strikes expressed in basis points.

**SpreadVol**

N-by-1 vector of annualized credit spread volatilities expressed as a positive decimal number.

**Name-Value Pair Arguments**

Specify optional comma-separated pairs of **Name**, **Value** arguments, where **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes (' '). You can specify several name and value pair arguments in any order as **Name1, Value1, . . . , NameN, ValueN**.

---

**Note** Any optional input of size N-by-1 is also acceptable as an array of size 1-by-N, or as a single value applicable to all contracts. Single values are internally expanded to an array of size N-by-1.

---

**Basis**

N-by-1 vector of contract day-count basis:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)

- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**Default:** 2 (actual/360)

BusDayConvention

String or N-by-1 cell array of strings of business day conventions.  
Values are:

- actual
- follow
- modifiedfollow
- previous
- modifiedprevious

**Default:** actual

Knockout

N-by-1 vector of Boolean flags. If the credit default swaptions is a knockout, the flag is True, otherwise it is False.

**Default:** True

PayAccruedPremium

N-by-1 vector of Boolean flags. If accrued premiums are paid upon default, the flag is True, otherwise it is False.



**Default:** True

Period

N-by-1 vector of the number of premiums per year of the CDS. Allowed values are 1, 2, 3, 4, 6, and 12.

**Default:** 4

RecoveryRate

N-by-1 vector of recovery rates, expressed as a decimal from 0 to 1.

**Default:** 0.4

ZeroBasis

Basis of the zero curve. Choices are identical to Basis.

**Default:** 0 (actual/actual)

ZeroCompounding

Compounding frequency of the zero curve. Allowed values are:

- 1 — Annual compounding
- 2 — Semiannual compounding
- 3 — Compounding three times per year
- 4 — Quarterly compounding
- 6 — Bimonthly compounding
- 12 — Monthly compounding
- -1 — Continuous compounding

---

**Note** When ZeroData is an IRCurve object, the arguments ZeroCompounding and ZeroBasis are implicit in ZeroData and are redundant inside this function. In that case, specify these optional arguments when constructing the IRCurve object before calling this function.

---

**Default:** 2 (Semiannual compounding)

## Output Arguments

Payer

N-by-1 vector of prices for payer swaptions in Basis points.

Receiver

N-by-1 vector of prices for receiver swaptions in Basis points.

## Definitions

### Credit Default Swap Option

A credit default swap (CDS) option, or credit default swaption, is a contract that provides the option holder with the right, but not the obligation, to enter into a credit default swap in the future. CDS options can either be payer swaptions or receiver swaptions. In a payer swaption, the option holder has the right to enter into a CDS in which they are paying premiums and in a receiver swaptions, the option holder is receiving premiums.

## Examples

Use cdsoptprice to generate Payer and Receiver values:

```
Settle = datenum('08-Sep-2010');
OptionMaturity = datenum('08-Sep-2011');
CDSMaturity = datenum('08-Sep-2015');
OptionStrike = 200;
SpreadVolatility = .4;

Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [4.65 5.02 5.019 5.008 5.002 5.03]'/100;
```

```
Zero_Dates = daysadd(Settle,360*Zero_Time,1);
ZeroData = [Zero_Dates Zero_Rate];

Market_Time = [1 2 3 5 7 10]';
Market_Rate = [100 120 145 220 245 270]';
Market_Dates = daysadd(Settle,360*Market_Time,1);
MarketData = [Market_Dates Market_Rate];

ProbData = cdsbootstrap(ZeroData, MarketData, Settle);

[Payer,Receiver] = cdsoptprice(ZeroData, ProbData, Settle,...
OptionMaturity, CDSMaturity, OptionStrike, SpreadVolatility)

Payer =

    323.4717

Receiver =

    47.2247
```

## References

O’Kane, D., *Modelling Single-name and Multi-name Credit Derivatives*, Wiley, 2008.

## See Also

| [cdsbootstrap](#) | [cdsspread](#) | [cdsprice](#)

## Tutorials

- “Credit Default Swap Option” on page 5-17

# cdsprice

---

## **Purpose**

Determine price for credit default swap

## **Syntax**

```
[Price, AccPrem, PaymentDates, PaymentTimes,  
PaymentCF] = cdsprice(ZeroData, ProbData, Settle,  
Maturity, ContractSpread)  
[Price, AccPrem, PaymentDates, PaymentTimes,  
PaymentCF] = cdsprice(ZeroData, ProbData,  
Settle, Maturity, ContractSpread, Name, Value)
```

## **Description**

[Price, AccPrem, PaymentDates, PaymentTimes, PaymentCF] = cdsprice(ZeroData, ProbData, Settle, Maturity, ContractSpread) computes the price, or the mark-to-market value for CDS instruments.

[Price, AccPrem, PaymentDates, PaymentTimes, PaymentCF] = cdsprice(ZeroData, ProbData, Settle, Maturity, ContractSpread, Name, Value) computes the price, or the mark-to-market value for CDS instruments with additional options specified by one or more Name, Value pair arguments.

## **Input Arguments**

ZeroData

M-by-2 vector of dates and zero rates or IRCurve of zero rates.

ProbData

P-by-2 array of dates and default probabilities.

Settle

Settlement date is a serial date number or date string. This must be earlier than or equal to the dates in MarketData.

Maturity

N-by-1 vector of serial date numbers or date strings containing the maturity dates.

ContractSpread

---

N-by-1 vector of contract spreads, expressed in basis points.

### **Name-Value Pair Arguments**

Specify optional comma-separated pairs of **Name**, **Value** arguments, where **Name** is the argument name and **Value** is the corresponding value. **Name** must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as **Name1**, **Value1**, . . . , **NameN**, **ValueN**.

---

**Note** Any optional input of size N-by-1 is also acceptable as an array of size 1-by-N, or as a single value applicable to all contracts. Single values are internally expanded to an array of size N-by-1.

---

### **Basis**

N-by-1 vector of day-count basis of the CDS:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)

- 13 = BUS/252

For more information, see basis.

**Default:** 2 (actual/360)

BusDayConvention

String or N-by-1 cell array of strings of business day conventions.  
Values are:

- actual
- follow
- modifiedfollow
- previous
- modifiedprevious

**Default:** actual

Notional

N-by-1 vector of contract notional values. Use positive values for long positions and negative values for short positions.

**Default:** 10MM

PayAccruedPremium

N-by-1 vector of Boolean flags. True, if accrued premiums are paid upon default, False otherwise.

**Default:** True

Period

N-by-1 vector of number of premiums per year of the CDS. Allowed values are 1, 2, 3, 4, 6, and 12.

**Default:** 4

RecoveryRate

N-by-1 vector of recovery rates, expressed as a decimal from 0 to 1.

**Default:** 0.4

TimeStep

Positive integer indicating the number of days to take as time step for the numerical integration.

**Default:** 10 (days)

ZeroBasis

Basis of the zero curve, where the choices are identical to Basis.

**Default:** 0 (actual/actual)

ZeroCompounding

Compounding frequency of the zero curve. Allowed values are:

- 1 — Annual compounding
- 2 — Semiannual compounding
- 3 — Compounding three times per year
- 4 — Quarterly compounding
- 6 — Bimonthly compounding
- 12 — Monthly compounding
- -1 — Continuous compounding

---

**Note** When ZeroData is an IRCurve object, the arguments ZeroCompounding and ZeroBasis are implicit in ZeroData and are redundant inside this function. In that case, specify these optional arguments when constructing the IRCurve object before calling this function.

---

**Default:** 2 (Semiannual compounding)

## Output Arguments

Price

N-by-1 vector of CDS prices.

AccPrem

N-by-1 vector of accrued premiums.

PaymentDates

N-by-numCF matrix of payment dates.

PaymentTimes

N-by-numCF matrix of accrual fractions.

PaymentCF

N-by-numCF matrix of payments.

## Definitions

### CDS Price

The price or mark-to-market (MtM) value of an existing CDS contract is computed using the following formula:

$$\text{CDS price} = \text{Notional} * (\text{Current Spread} - \text{Contract Spread}) * \text{RPV01}$$

Current Spread is the current breakeven spread for a similar contract, according to current market conditions. RPV01 is the 'risky present



value of a basis point,' the present value of the premium payments, taking into consideration the default probability. This formula assumes a long position, and the right side is multiplied by -1 for short positions.

## Examples

Use `cdsprice` to compute the clean price for a CDS contract:

```
Settle = '17-Jul-2009';
Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
Zero_Dates = daysadd(Settle,360*Zero_Time,1);
ZeroData = [Zero_Dates Zero_Rate];

ProbData = [daysadd(datenum(Settle),360,1), 0.0247];
Maturity = '20-Sep-2010';
ContractSpread = 135;

[Price,AccPrem] = cdsprice(ZeroData,ProbData,Settle,Maturity,ContractSpread);
CleanPrice = Price - AccPrem
CleanPrice =

    4.9381e+003
```

## Algorithms

The premium leg is computed as the product of a spread  $S$  and the risky present value of a basis point (RPV01). The RPV01 is given by:

$$RPV01 = \sum_{j=1}^N Z(t_j) \Delta(t_{j-1}, t_j, B) Q(t_j)$$

when no accrued premiums are paid upon default, and it can be approximated by

$$RPV01 \approx \frac{1}{2} \sum_{j=1}^N Z(t_j) \Delta(t_{j-1}, t_j, B) (Q(t_{j-1}) + Q(t_j))$$

when accrued premiums are paid upon default. Here,  $t_0 = 0$  is the valuation date, and  $t_1, \dots, t_n = T$  are the premium payment dates over the

life of the contract,  $T$  is the maturity of the contract,  $Z(t)$  is the discount factor for a payment received at time  $t$ , and  $\Delta(t_{j-1}, t_j, B)$  is a day count between dates  $t_{j-1}$  and  $t_j$  corresponding to a basis  $B$ .

The protection leg of a CDS contract is given by the following formula:

$$\begin{aligned} \text{ProtectionLeg} &= \int_0^T Z(\tau)(1-R)dPD(\tau) \\ &\approx (1-R) \sum_{i=1}^M Z(\tau_i)(PD(\tau_i) - PD(\tau_{i-1})) \\ &= (1-R) \sum_{i=1}^M Z(\tau_i)(Q(\tau_{i-1}) - Q(\tau_i)) \end{aligned}$$

where the integral is approximated with a finite sum over the discretization  $\tau_0 = 0, \tau_1, \dots, \tau_M = T$ .

If the spread of an existing CDS contract is  $S_C$ , and the current breakeven spread for a comparable contract is  $S_\theta$ , the current price, or mark-to-market value of the contract is given by:

$$\text{MtM} = \text{Notional} (S_\theta - S_C) \text{RPV01}$$

This assumes a long position from the protection standpoint (protection was bought). For short positions, the sign is reversed.

## References

- Beumee, J., D. Brigo, D. Schiemert, and G. Stoye. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research, Global Special Report*. April 7, 2009.
- Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives* 8, 29-40.
- O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps." *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.

**See Also**

| [cdsspread](#) | [cdsbootstrap](#)

**Tutorials**

- “Credit Default Swap (CDS)” on page 5-2

# cdsspread

---

<b>Purpose</b>	Determine spread of credit default swap
<b>Syntax</b>	<pre>[Spread, PaymentDates, PaymentTimes] = cdsspread(ZeroData, ProbData, Settle, Maturity) [Spread, PaymentDates, PaymentTimes] = cdsspread(ZeroData, ProbData, Settle, Maturity, Name, Value)</pre>
<b>Description</b>	<p>[Spread, PaymentDates, PaymentTimes] = cdsspread(ZeroData, ProbData, Settle, Maturity) computes the spread of the CDS.</p> <p>[Spread, PaymentDates, PaymentTimes] = cdsspread(ZeroData, ProbData, Settle, Maturity, Name, Value) computes the spread of the CDS with additional options specified by one or more Name, Value pair arguments.</p>
<b>Input Arguments</b>	<p><b>ZeroData</b> M-by-2 vector of dates and zero rates or IRCurve of zero rates.</p> <p><b>ProbData</b> P-by-2 array of dates and default probabilities.</p> <p><b>Settle</b> Settlement date is a serial date number or date string. This must be earlier than or equal to the dates in MarketData.</p> <p><b>Maturity</b> N-by-1 vector of serial date numbers or date strings containing the maturity dates.</p> <p><b>Name-Value Pair Arguments</b> Specify optional comma-separated pairs of Name, Value arguments, where Name is the argument name and Value is the corresponding value. Name must appear inside single quotes ( ' '). You can specify several name and value pair arguments in any order as Name1, Value1, ..., NameN, ValueN.</p>

---

**Note** Any optional input of size N-by-1 is also acceptable as an array of size 1-by-N, or as a single value applicable to all contracts. Single values are internally expanded to an array of size N-by-1.

---

### Basis

N-by-1 vector of day-count basis of the CDS:

- 0 = actual/actual
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**Default:** 2 (actual/360)

BusDayConvention

String or N-by-1 cell array of strings of business day conventions.  
Values are:

- actual
- follow
- modifiedfollow
- previous
- modifiedprevious

**Default:** actual

PayAccruedPremium

N-by-1 vector of Boolean flags, True, if accrued premiums are paid upon default, False otherwise.

**Default:** True

Period

N-by-1 vector of number of premiums per year of the CDS. Allowed values are 1, 2, 3, 4, 6, and 12.

**Default:** 4

RecoveryRate

N-by-1 vector of recovery rates, expressed as a decimal from 0 to 1.

**Default:** 0.4

TimeStep

Positive integer indicating the number of days to take as time step for the numerical integration.

**Default:** 10 (days)

**ZeroBasis**

Basis of the zero curve, where the choices are identical to **Basis**.

**Default:** 0 (actual/actual)

**ZeroCompounding**

Compounding frequency of the zero curve. Allowed values are:

- 1 — Annual compounding
- 2 — Semiannual compounding
- 3 — Compounding three times per year
- 4 — Quarterly compounding
- 6 — Bimonthly compounding
- 12 — Monthly compounding
- -1 — Continuous compounding

---

**Note** When **ZeroData** is an **IRCurve** object, the arguments **ZeroCompounding** and **ZeroBasis** are implicit in **ZeroData** and are redundant inside this function. In that case, specify these optional arguments when constructing the **IRCurve** object before calling this function.

---

**Default:** 2 (semiannual compounding )

**Output  
Arguments****Spread**

N-by-1 vector of spreads (in basis points).

**PaymentDates**

N-by-numCF matrix of payment dates.

PaymentTimes

N-by-numCF matrix of accrual fractions.

## Definitions

### CDS Spread

The market, or breakeven, spread value of a CDS can be computed by equating the value of the protection leg with the value of the premium leg:

$$\text{Market Spread} * \text{RPV01} = \text{Value of Protection Leg}$$

The left side corresponds to the value of the premium leg, and this has been decomposed as the product of the market or breakeven spread times the RPV01 or 'risky present value of a basis point' of the contract. The latter is the present value of the premium payments, taking into consideration the default probability. The Market Spread can be computed as the ratio of the value of the protection leg, to the RPV01 of the contract. `cdsspread` returns the resulting spread in basis points.

## Examples

Use `cdsspread` to compute the clean price for a CDS contract:

```
Settle = '17-Jul-2009';
Zero_Time = [.5 1 2 3 4 5]';
Zero_Rate = [1.35 1.43 1.9 2.47 2.936 3.311]'/100;
Zero_Dates = daysadd(Settle,360*Zero_Time,1);
ZeroData = [Zero_Dates Zero_Rate];
ProbData = [daysadd(datenum(Settle),360,1), 0.0247];
Maturity = '20-Sep-2010';
```

```
Spread = cdsspread(ZeroData,ProbData,Settle,Maturity)
```

```
Spread =
```

```
148.2485
```

## Algorithms

The premium leg is computed as the product of a spread  $S$  and the risky present value of a basis point (RPV01). The RPV01 is given by:



$$RPV01 = \sum_{j=1}^N Z(t_j) \Delta(t_{j-1}, t_j, B) Q(t_j)$$

when no accrued premium are paid upon default, and it can be approximated by

$$RPV01 \approx \frac{1}{2} \sum_{j=1}^N Z(t_j) \Delta(t_{j-1}, t_j, B) (Q(t_{j-1}) + Q(t_j))$$

when accrued premiums are paid upon default. Here,  $t_0 = 0$  is the valuation date, and  $t_1, \dots, t_n = T$  are the premium payment dates over the life of the contract,  $T$  is the maturity of the contract,  $Z(t)$  is the discount factor for a payment received at time  $t$ , and  $\Delta(t_{j-1}, t_j, B)$  is a day count between dates  $t_{j-1}$  and  $t_j$  corresponding to a basis  $B$ .

The protection leg of a CDS contract is given by the following formula:

$$\begin{aligned} ProtectionLeg &= \int_0^T Z(\tau)(1-R)dPD(\tau) \\ &\approx (1-R) \sum_{i=1}^M Z(\tau_i)(PD(\tau_i) - PD(\tau_{i-1})) \\ &= (1-R) \sum_{i=1}^M Z(\tau_i)(Q(\tau_{i-1}) - Q(\tau_i)) \end{aligned}$$

where the integral is approximated with a finite sum over the discretization  $\tau_0 = 0, \tau_1, \dots, \tau_M = T$ .

A breakeven spread  $S_0$  makes the value of the premium and protection legs equal. It follows that:

$$S_0 = \frac{ProtectionLeg}{RPV01}$$

## References

Beumee, J., D. Brigo, D. Schiemert, and G. Stoye. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research*, Global Special Report. April 7, 2009.

Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives* 8, 29-40.

O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps." *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.

## See Also

| `cdsprice` | `cdsbootstrap`

## Tutorials

- "Credit Default Swap (CDS)" on page 5-2

**Purpose** Yield on certificate of deposit (CD)

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

**Arguments**

Price	Clean price of the certificate of deposit per \$100 face. If you have a vector of dirty or cash prices of CDs, compute the accrued interest portion using <code>cdai</code> .
CouponRate	Annual interest rate in decimal.
Settle	Settlement date. <code>Settle</code> must be earlier than <code>Maturity</code> .
Maturity	Maturity date.
IssueDate	Issue date.
Basis	(Optional) Day-count basis of the instrument. <ul style="list-style-type: none"> <li>• 0 = actual/actual (default)</li> <li>• 1 = 30/360 (SIA)</li> <li>• 2 = actual/360</li> <li>• 3 = actual/365</li> <li>• 4 = 30/360 (BMA)</li> <li>• 5 = 30/360 (ISDA)</li> <li>• 6 = 30/360 (European)</li> <li>• 7 = actual/365 (Japanese)</li> <li>• 8 = actual/actual (ICMA)</li> <li>• 9 = actual/360 (ICMA)</li> <li>• 10 = actual/365 (ICMA)</li> </ul>

- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see `basis`.

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

## Description

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

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

## Examples

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

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

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

```
Yield =
```

```
0.0039
```

## See Also

`bndprice` | `cdai` | `cdprice` | `stepcpnprice` | `tbillprice`

<b>Purpose</b>	Generate principal balance schedule for planned amortization class (PAC) or targeted amortization class (TAC) bond
<b>Syntax</b>	<pre>[BalanceSchedule, InitialBalance] = cmosched(Principal, Coupon, OriginalTerm, TermRemaining, PrepaySpeed) [BalanceSchedule, InitialBalance] = cmosched(Principal, Coupon, OriginalTerm, TermRemaining, PrepaySpeed, TranchePrincipal)</pre>
<b>Description</b>	<p>[BalanceSchedule, InitialBalance] = cmosched(Principal,Coupon, OriginalTerm, TermRemaining, PrepaySpeed) generates a principal balance schedule for planned amortization class (PAC) bonds using two bands of Public Securities Association Prepayment Model (PSA) speeds or targeted amortization class (TAC) bonds using a single PSA speed.</p> <p>[BalanceSchedule, InitialBalance] = cmosched(Principal,Coupon, OriginalTerm, TermRemaining, PrepaySpeed,TranchePrincipal) with a specified tranche principal generates a principal balance schedule for planned amortization class (PAC) bonds using two bands of PSA speeds or targeted amortization class (TAC) bonds using a single PSA speed.</p>
<b>Input Arguments</b>	<p><b>Principal</b> Principal of the underlying mortgage pool.</p> <p><b>Coupon</b> Coupon of the underlying mortgage pool.</p> <p><b>OriginalTerm</b> Original term in months of the underlying mortgage pool.</p> <p><b>TermRemaining</b> Terms remaining in months of the underlying mortgage pool.</p>

## PrepaySpeed

PSA speed. For a PAC, the speed is a 1-by-2 matrix where the first element is the lower band and the second element is the upper band. For a TAC, the speed is a scalar.

## TranchePrincipal

(Optional) Principal of the scheduled tranche. If it is unspecified or empty [], the principal of the scheduled tranche is assumed to be the sum of the payment schedule calculated from the PSA prepayment speeds.

## Output Arguments

### BalanceSchedule

Matrix of size 1-by-NUMTERMS, where NUMTERMS is the number of terms remaining. Each column contains the scheduled principal balance for the time period corresponding to the column number.

### InitialBalance

Scalar containing the initial principal balance of the scheduled tranche.

## Definitions

### Planned Amortization Class (PAC) Bond

PAC bonds are a type of CMO bond. They are designed to largely eliminate prepayment risk for investors. They do this by transferring essentially all prepayment risk to other bonds in the CMO that are called support bonds.

### Targeted Amortization Class (TAC) Bond

TAC bonds are analogous to PAC bonds, but are structured differently. TAC bonds offer one-sided protection, shielding investors from high prepayment rates up to a specified PSA and do not protect against low prepayment rates.

**Examples**

**Calculate the Principal Balance Schedule for a CMO PAC Bond**

Define the mortgage pool under consideration and generate a principal balance schedule for planned amortization class (PAC) bonds using two bands of PSA speeds.

Calculate PAC bonds using cmosched.

```
Principal = 128687000;
GrossRate = 0.0648;
OriginalTerm = 360;
TermRemaining = 325;
PrepaySpeed = [300 525];
PacPrincipal = 100250000;
```

```
[BalanceSchedule, InitialBalance] ...
= cmosched(Principal, GrossRate, OriginalTerm, TermRemaining, ...
PrepaySpeed, PacPrincipal);
```

BalanceSchedule =

1.0e+07 \*

Columns 1 through 10

9.7996	9.5780	9.3602	9.1461	8.9357	8.7289	8.5257	8.3259	8.1296	7.9366
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 11 through 20

7.7469	7.5605	7.3773	7.1972	7.0202	6.8463	6.6754	6.5073	6.3422	6.1799
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 21 through 30

6.0204	5.8637	5.7096	5.5582	5.4094	5.2632	5.1194	4.9782	4.8394	4.7030
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 31 through 40

4.5689	4.4372	4.3077	4.1804	4.0554	3.9325	3.8118	3.6931	3.5765	3.4619
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 41 through 50

3.3494	3.2406	3.1353	3.0334	2.9348	2.8394	2.7470	2.6576	2.5711	2.4873
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 51 through 60

2.4063	2.3279	2.2520	2.1786	2.1075	2.0387	1.9722	1.9078	1.8455	1.7852
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 61 through 70

1.7268	1.6703	1.6157	1.5628	1.5117	1.4622	1.4142	1.3679	1.3231	1.2797
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 71 through 80

1.2377	1.1970	1.1577	1.1197	1.0829	1.0473	1.0129	0.9795	0.9473	0.9161
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 81 through 90

0.8859	0.8567	0.8285	0.8011	0.7747	0.7491	0.7244	0.7004	0.6773	0.6549
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 91 through 100

0.6332	0.6122	0.5920	0.5723	0.5534	0.5350	0.5172	0.5001	0.4835	0.4674
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 101 through 110

0.4518	0.4368	0.4223	0.4082	0.3946	0.3814	0.3687	0.3564	0.3445	0.3330
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 111 through 120

0.3219	0.3111	0.3007	0.2906	0.2809	0.2715	0.2623	0.2535	0.2450	0.2368
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 121 through 130

0.2288	0.2211	0.2137	0.2065	0.1995	0.1928	0.1863	0.1800	0.1739	0.1680
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 131 through 140



0.1623 0.1568 0.1515 0.1464 0.1414 0.1366 0.1319 0.1275 0.1231 0.1189

Columns 141 through 150

0.1149 0.1109 0.1072 0.1035 0.0999 0.0965 0.0932 0.0900 0.0869 0.0839

Columns 151 through 160

0.0811 0.0783 0.0756 0.0730 0.0704 0.0680 0.0657 0.0634 0.0612 0.0591

Columns 161 through 170

0.0570 0.0550 0.0531 0.0513 0.0495 0.0478 0.0461 0.0445 0.0429 0.0414

Columns 171 through 180

0.0400 0.0386 0.0372 0.0359 0.0346 0.0334 0.0322 0.0311 0.0300 0.0289

Columns 181 through 190

0.0279 0.0269 0.0260 0.0250 0.0241 0.0233 0.0224 0.0216 0.0209 0.0201

Columns 191 through 200

0.0194 0.0187 0.0180 0.0174 0.0167 0.0161 0.0155 0.0150 0.0144 0.0139

Columns 201 through 210

0.0134 0.0129 0.0124 0.0120 0.0115 0.0111 0.0107 0.0103 0.0099 0.0096

Columns 211 through 220

0.0092 0.0089 0.0085 0.0082 0.0079 0.0076 0.0073 0.0070 0.0068 0.0065

Columns 221 through 230

0.0063	0.0060	0.0058	0.0056	0.0054	0.0052	0.0050	0.0048	0.0046	0.0044
Columns 231 through 240									
0.0042	0.0041	0.0039	0.0037	0.0036	0.0035	0.0033	0.0032	0.0031	0.0029
Columns 241 through 250									
0.0028	0.0027	0.0026	0.0025	0.0024	0.0023	0.0022	0.0021	0.0020	0.0019
Columns 251 through 260									
0.0018	0.0018	0.0017	0.0016	0.0016	0.0015	0.0014	0.0014	0.0013	0.0012
Columns 261 through 270									
0.0012	0.0011	0.0011	0.0010	0.0010	0.0009	0.0009	0.0009	0.0008	0.0008
Columns 271 through 280									
0.0007	0.0007	0.0007	0.0006	0.0006	0.0006	0.0005	0.0005	0.0005	0.0005
Columns 281 through 290									
0.0004	0.0004	0.0004	0.0004	0.0004	0.0003	0.0003	0.0003	0.0003	0.0003
Columns 291 through 300									
0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0002	0.0001	0.0001
Columns 301 through 310									
0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0000
Columns 311 through 320									
0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0	0

Columns 321 through 325

0 0 0 0 0

InitialBalance =

100250000

## References

Hayre, Lakhbir, ed., *Salomon Smith Barney Guide to Mortgage-Backed and Asset-Backed Securities*, John Wiley and Sons, New York, 2001.

Lyu, Yuh-Dah, *Financial Engineering and Computation*, Cambridge University Press, 2004.

## See Also

| cmoschedcf |

## Tutorials

- “Using Collateralized Mortgage Obligations (CMOs)” on page 2-17

## How To

- “Create a PAC and Sequential CMO for Underlying Mortgage Pool” on page 2-32

# cmoschedcf

---

**Purpose** Generate cash flows for scheduled collateralized mortgage obligation (CMO) using PAC or TAC model

**Syntax** [Balances, Principal, Interest] =  
cmoschedcf(PrincipalPayments, TranchePrincipals,  
TrancheCoupons, BalanceSchedule)

**Description** [Balances, Principal, Interest] =  
cmoschedcf(PrincipalPayments, TranchePrincipals,  
TrancheCoupons, BalanceSchedule) generate cash flows for a  
scheduled CMO such as the planned amortization class (PAC) or  
targeted amortization class (TAC), given the underlying mortgage  
pool payments (or payments from another CMO tranche). The output  
Balance, Principal, and Interest from this function can be used  
as input into cmoseqcf to further divide the PAC, TAC, or support  
tranche into sequential tranches.

## **Input Arguments**

**PrincipalPayments**

Matrix of size 1-by-NUMTERMS, where NUMTERMS is the number of terms remaining. Each column contains the underlying principal payment for the time period corresponding to the row number. Calculate underlying principal payments using mbscfamounts or mbspassthrough. The underlying principal payments can also be outputs from other CMO cash flow functions.

**TranchePrincipals**

Matrix of size 2-by-1 specifying the initial principal for the scheduled and the support tranche.

**TrancheCoupons**

Matrix of size 2-by-1 specifying the coupons for the schedule tranche and the support tranche. The weighted average coupon for the CMO should not exceed the coupon of the underlying mortgage.

**BalanceSchedule**

Matrix of size 1 -by-NUMTERMS, where NUMTERMS is the number of terms remaining. Each element represents the targeted balance schedule for the time period corresponding to that column.

## Output Arguments

### Balance

Matrix of size 2-by-NUMTERMS, where NUMTERMS is the number of terms remaining. The first row is the principal balances of the scheduled tranche, and the second row is the principal balances of the support tranche at the time period corresponding to the column.

### Principal

Matrix of size 2-by-NUMTERMS, where NUMTERMS is the number of terms remaining. The first row is the principal payments of the scheduled tranche, and the second row is the principal payments of the support tranche at the time period corresponding to the column.

### Interest

Matrix of size 2-by-NUMTERMS, where NUMTERMS is the number of terms remaining. The first row is the interest payments of the schedule tranche, and the second row is the interest payments of the support tranche at the time period corresponding to the column.

## Definitions

### Planned Amortization Class (PAC) Tranches

In a PAC CMO, there is a main tranche, known as the schedule tranche, and a support tranche. The main purpose of a schedule tranche is to give investors in the PAC tranche a more certain cash flow.

### Targeted Amortization Class (TAC) Tranches

TACs are like PACs, but principal payment is specified for only one prepayment rate. If prepayment rates are higher or lower, then the principal payment to TAC holders will be higher or lower accordingly.

### Schedule and Support Tranche

The main purpose of a PAC tranche is to give investors in the PAC tranche a more certain cash flow. The PAC tranche receives priority

for receiving payments of principal and interest that gives investors in the PAC tranche a steadier income. If prepayments differ from what was expected, then the support tranche gets the variable portion of the payments. While income to the support tranche is more variable, it is also higher yielding. Estimates of the yield, average life, and lockout periods of the PAC tranche is more certain.

## Examples

### Calculate Cash Flows for Each PAC Tranche

Define the mortgage pool under consideration for CMO structuring using `mbscfamounts` or `mbspassthrough`. Calculate the underlying mortgage cash flow, define the PAC schedule and CMO tranches, and calculate the cash flows for each tranche.

Calculate the underlying cash flow using `mbspassthrough`:

```
% Underlying mortgage
MortgagePrincipal = 1000000;
Coupon = 0.12;
Terms = 6; % months

[PrincipalBalance, MonthlyPayments, SchedPrincipalPayments, ...
InterestPayments, Prepayments] = ...
mbspassthrough(MortgagePrincipal, Coupon, Terms, Terms, 0, []);
PrincipalPayments = SchedPrincipalPayments.' + Prepayments.'

PrincipalPayments =

    1.0e+05 *

    1.6255    1.6417    1.6582    1.6747    1.6915    1.7084
```

Calculate the PAC schedule for CMO using `cmosched`.

```
PrepaySpeed = [100 300];
[BalanceSchedule, InitialBalance] ...
= cmosched(MortgagePrincipal, Coupon, Terms, Terms, PrepaySpeed, [])
```

BalanceSchedule =

```

1.0e+05 *
      8.3617    6.7180    5.0581    3.3828    1.6955    0

```

InitialBalance =

```
9.9886e+05
```

Define CMO tranches.

```

TranchePrincipals = ...
[InitialBalance; MortgagePrincipal-InitialBalance];
TrancheCoupons = [0.12; 0.12];

```

TrancheCoupons =

```

0.1200
0.1200

```

Calculate cash flows for each tranche.

```

[Balance, Principal, Interest] = ...
cmoschedcf(PrincipalPayments, TranchePrincipals, ...
TrancheCoupons, BalanceSchedule)

```

Balance =

```

1.0e+05 *
      8.3631    6.7213    5.0632    3.3885    1.6970    0
      0.0114    0.0114    0.0114    0.0114    0.0114    0.0000

```

Principal =

1.0e+05 *						
1.6255	1.6417	1.6582	1.6747	1.6915	1.6970	
0	0	0	0	0	0.0114	

Interest =

1.0e+03 *						
9.9886	8.3631	6.7213	5.0632	3.3885	1.6970	
0.0114	0.0114	0.0114	0.0114	0.0114	0.0114	

## References

Hayre, Lakhbir, ed., *Salomon Smith Barney Guide to Mortgage-Backed and Asset-Backed Securities*, John Wiley and Sons, New York, 2001.

Lyu, Yuh-Dah, *Financial Engineering and Computation*, Cambridge University Press, 2004.

## See Also

| [cmoseqcf](#) | [cmosched](#) | [mbscfamounts](#) | [mbspassthrough](#) |

## Tutorials

- “Using Collateralized Mortgage Obligations (CMOs)” on page 2-17

## How To

- “Create a PAC and Sequential CMO for Underlying Mortgage Pool” on page 2-32



---

<b>Purpose</b>	Generate cash flows for sequential collateralized mortgage obligation (CMO)
<b>Syntax</b>	<pre>[balances, principals, interests] = cmoseqcf( PrincipalPayments, TranchePrincipals, TrancheCoupons) [balances, principals, interests] = cmoseqcf( PrincipalPayments, TranchePrincipals, TrancheCoupons, HasZ)</pre>
<b>Description</b>	<p>[balances, principals, interests] = cmoseqcf(PrincipalPayments, TranchePrincipals, TrancheCoupons) generates cash flows for a sequential CMO without a Z-bond, given the underlying mortgage pool payments.</p> <p>[balances, principals, interests] = cmoseqcf(PrincipalPayments, TranchePrincipals, TrancheCoupons, HasZ) generates cash flows for a sequential CMO with a Z-bond, given the underlying mortgage pool payments.</p>
<b>Input Arguments</b>	<p><b>PrincipalPayments</b></p> <p>Matrix of size 1-by-NUMTERMS, where NUMTERMS is the number of terms remaining. Each row contains the underlying principal payment for the time period corresponding to the row number. The underlying principal payments can be calculated using mbscfamounts or mbspassthrough. The underlying principal payments can also be outputs from other CMO cash flow functions</p> <p><b>TranchePrincipals</b></p> <p>Matrix of size NUMTRANCHES-by-1, where NUMTRANCHES is the number of tranches in the sequential CMO. Each element of the matrix represents the initial principal for each tranche. If the sequential CMO includes a Z-bond (HasZ is true), the last element of this matrix is the principal of the Z-bond.</p> <p><b>TrancheCoupons</b></p>

Matrix of size NUMTRANCHES-by-1, where NUMTRANCHES is the number of tranches in the sequential CMO. Each element of the matrix represents the coupon for each tranche. If the sequential CMO includes a Z-bond (HasZ is true), the last element of this matrix is the coupon of the Z-bond. The weighted average coupon for the CMO should not exceed the coupon of the underlying mortgage.

HasZ

(Optional) Boolean (true or false). A value of true indicates that the sequential CMO contains a Z-bond, and the last element of TranchePrincipals and TrancheCoupons will be treated as that of the Z-bond. A value of false indicates that there is no Z-bond in the sequential CMO, and the last element of TranchePrincipals and TrancheCoupons will be treated as an ordinary tranche.

**Default:** false

## Output Arguments

Balance

Matrix of size NUMTRANCHES-by-NUMTERMS, where NUMTRANCHES is the number of terms remaining and NUMTRANCHES is the number of tranches. Each element represents the principal balance at the time period corresponding to the column, and for the tranche corresponding to the row.

Principal

Matrix of size NUMTRANCHES-by-NUMTERMS, where NUMTRANCHES is the number of terms remaining and NUMTRANCHES is the number of tranches. Each element represents the principal payments made at the time period corresponding to the column, and to the tranche corresponding to the row.

Interest

Matrix of size NUMTRANCHES-by-NUMTERMS, where NUMTRANCHES is the number of terms remaining and NUMTRANCHES is the number of tranches. Each element represents the interest payments made at the time period

corresponding to the column, and to the tranche corresponding to the row.

## Definitions

### Sequential Pay CMO

A sequential pay CMO involves tranches that pay off principal sequentially. For example, consider the following case, where all principal from the underlying mortgage pool is repaid on tranche A first, then tranche B, then tranche C. Note that interest is paid on each tranche as long as the principal for the tranche has not been retired.

### CMO Tranche

Tranche is a term often used to describe a specific class of bonds within an offering wherein each tranche offers varying degrees of risk to the investor.

## Examples

### Calculate Cash Flows for a Sequential Collateralized Mortgage Obligation (CMO)

Define the mortgage pool under consideration for CMO structuring using `mbscfamounts` or `mbspassthrough` and calculate the cash flows with an A and B tranche for a sequential CMO.

Calculate underlying cash flow using `mbspassthrough`:

```
MortgagePrincipal = 1000000;
Coupon = 0.12;
Terms = 6; % months
```

```
% Calculate underlying mortgage cash flows
[PrincipalBalance, MonthlyPayments, SchedPrincipalPayments, ...
InterestPayments, Prepayments] = ...
mbspassthrough(MortgagePrincipal, Coupon, Terms, Terms, 0, []);
PrincipalPayments = SchedPrincipalPayments.' + Prepayments.'
```

```
PrincipalPayments =
```

```
1.0e+05 *
```

1.6255    1.6417    1.6582    1.6747    1.6915    1.7084

Define CMO tranches, A and B.

TranchePrincipals = [500000; 500000];

TrancheCoupons = [0.12; 0.12];

Calculate cash flows for each tranche.

[Balance, Principal, Interest] = ...

cmoseqcf(PrincipalPayments, TranchePrincipals, TrancheCoupons, false)

Balance =

1.0e+05 \*

3.3745	1.7328	0.0746	0	0	0
5.0000	5.0000	5.0000	3.3999	1.7084	0.0000

Principal =

1.0e+05 \*

1.6255	1.6417	1.6582	0.0746	0	0
0	0	0	1.6001	1.6915	1.7084

Interest =

1.0e+03 \*

5.0000	3.3745	1.7328	0.0746	0	0
5.0000	5.0000	5.0000	5.0000	3.3999	1.7084

## References

Hayre, Lakhbir, ed., *Salomon Smith Barney Guide to Mortgage-Backed and Asset-Backed Securities*, John Wiley and Sons, New York, 2001.

Lyu, Yuh-Dah, *Financial Engineering and Computation*, Cambridge University Press, 2004.

**See Also**

| [cmoschedcf](#) | [mbscfamounts](#) | [mbspassthrough](#) | [cmosched](#) |

**Tutorials**

- “Using Collateralized Mortgage Obligations (CMOs)” on page 2-17

**How To**

- “Create a PAC and Sequential CMO for Underlying Mortgage Pool” on page 2-32

# convfactor

---

## Purpose

Bond conversion factors

## Syntax

```
CF = convfactor(RefDate, Maturity, CouponRate)
CF = convfactor(RefDate, Maturity, CouponRate,
'ParameterName',ParameterValue ...)
```

## Description

CF = convfactor(RefDate, Maturity, CouponRate) computes a conversion factor for a bond futures contract.

CF = convfactor(RefDate, Maturity, CouponRate, 'ParameterName',ParameterValue ...) accepts optional inputs as one or more comma-separated parameter-value pairs. 'ParameterName' is the name of the parameter inside single quotes. 'ParameterValue' is the value corresponding to 'ParameterName'. Specify parameter/value pairs in any order. Names are case-insensitive. convfactor computes a conversion factor for a bond futures contract, given a Convention value for a U.S. Treasury bond, German bond, U.K. Gilt, or Japanese Government Bond.

## Input Arguments

RefDate

Reference dates, for which conversion factor is computed (usually the first day of delivery months).

Maturity

Maturity date of the underlying bond.

CouponRate

Annual coupon rate of the underlying bond in decimal.

## Parameter-Value Pairs

Enter the following inputs only as parameter-value pairs.

Convention

Conversion factor convention. Scalar. Valid values are:

- 1 = U.S. Treasury bond (30-year) and Treasury note (10-year) futures contract
- 2 = U.S. 2-year and 5-year Treasury note futures contract
- 3 = German Bobl, Bund, Buxl, and Schatz
- 4 = U.K. gilts
- 5 = Japanese Government Bonds (JGBs)

**Default:** 1

FirstCouponDate

Irregular or normal first coupon date.

RefYield

Reference semiannual yield.

**Default:** 0.06 (6%)

StartDate

Forward starting date of payments.

## Output Arguments

CF

N-by1 vector of conversion factors against the 6% yield par-bond.

## Definitions

Conversion factors of U.S. Treasury bonds and other government bonds are based on a bond yielding 6%. Optionally, you can specify other types of bonds and yields using inputs for RefYield and Convention. For U.S. Treasury bonds, verify the output of convfactor by comparing the output against the quotations provided by the Chicago Board of Trade (<http://www.cbot.com>).

For German bonds, verify the output of convfactor by comparing the output against the quotations provided by Eurex (<http://www.eurexchange.com>).

For U.K. Gilts, verify the output of convfactor by comparing the output against the quotations provided by Euronext (<http://www.euronext.com>).

For Japanese Government Bonds, verify the output of convfactor by comparing the output against the quotations provided by the Tokyo Stock Exchange (<http://www.tse.or.jp/english/>).

## Examples

Calculate CF, given the following RefDate, Maturity, and CouponRate:

```
RefDate = {'1-Dec-2002';
           '1-Mar-2003';
           '1-Jun-2003';
           '1-Sep-2003';
           '1-Dec-2003';
           '1-Sep-2003';
           '1-Dec-2002';
           '1-Jun-2003'};

Maturity = {'15-Nov-2012';
           '15-Aug-2012';
           '15-Feb-2012';
           '15-Feb-2011';
           '15-Aug-2011';
           '15-Aug-2010';
           '15-Aug-2009';
           '15-Feb-2010'};

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

CF = convfactor(RefDate, Maturity, CouponRate)
```

This returns:

```
CF =
    0.8539
    0.8858
    0.9259
```



0.9418  
0.9403  
0.9862  
1.0000  
1.0266

---

Calculate cf, given the following RefDate, Maturity, and CouponRate for a German Bond:

```
cf = convfactor('3/10/2009', '1/04/2018', .04, .06, 3)
```

This returns:

```
cf =
```

```
0.8659
```

## References

Burghardt, G., T. Belton, M. Lane, and J. Papa, *The Treasury Bond Basis*, McGraw-Hill, 2005.

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

## See Also

tfutbyprice | tfutbyyield | tfutimprepo | bndfutimprepo |  
bndfutprice

## How To

- “Bond Futures” on page 4-12

# fitFunction

---

**Purpose** Custom fit interest-rate curve object to bond market data

**Class** @IRFunctionCurve

**Syntax**  
CurveObj = IRFunctionCurve.fitFunction(Type, Settle, FunctionHandle, Instruments, IRFitOptionsObj)  
CurveObj = IRFunctionCurve.fitFunction(Type, Settle, FunctionHandle, Instruments, IRFitOptionsObj, 'Parameter1', Value1, 'Parameter2', Value2, ...)

**Arguments**

Type	Type of interest-rate curve for a bond: zero, forward, or discount.
Settle	Scalar or column vector of settlement dates. Settle must be earlier than Maturity.
FunctionHandle	Function handle that defines the interest-rate curve. The function handle takes two numeric vectors (time-to-maturity and a vector of function coefficients) and returns one numeric output (interest rate or discount factor). For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.
Instruments	N-by-4 data matrix for Instruments where the first column is Settle date, the second column is Maturity, the third column is the clean price, and the fourth column is a CouponRate for the bond.
IRFitOptionsObj	Object constructed from IRFitOptions.

**Compounding**

(Optional) Scalar that sets the compounding frequency per year for the `IRFunctionCurve` object:

- -1 = Continuous compounding
- 1 = Annual compounding
- 2 = Semiannual compounding (default)
- 3 = Compounding three times per year
- 4 = Quarterly compounding
- 6 = Bimonthly compounding
- 12 = Monthly compounding

**Basis**

(Optional) Day-count basis of the bond. A scalar of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see `basis`.

## Instrument Parameters

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

Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.
Basis	(Optional) Day-count basis of the bond. A vector of integers. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul>

For more information, see basis.

---

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when <code>Maturity</code> is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when an instrument was issued.
FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When <code>FirstCouponDate</code> and <code>LastCouponDate</code> are both specified, <code>FirstCouponDate</code> takes precedence in determining the coupon payment structure. If you do not specify a <code>FirstCouponDate</code> , the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified <code>FirstCouponDate</code> , a specified <code>LastCouponDate</code> determines the coupon structure of the bond. The coupon structure of a bond is truncated at the <code>LastCouponDate</code> , regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a <code>LastCouponDate</code> , the cash flow payment dates are determined from other inputs.
Face	(Optional) Face or par value. Default = 100.

---

**Note** When using Instrument parameter/value pairs, you can specify simple interest for a bond by specifying the InstrumentPeriod value as 0. If InstrumentBasis and InstrumentPeriod are not specified for a bond, the following default values are used: Basis is 0 (act/act) and Period is 2.

---

## Description

CurveObj = IRFunctionCurve.fitFunction(Type, Settle, FunctionHandle, Instruments, IRFitOptionsObj, 'Parameter1', Value1, 'Parameter2', Value2, ...) fits a bond to a custom fitting function. You must enter the optional arguments for Basis and Compounding as parameter/value pairs.

## Examples

```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];
CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
CurveSettle = datenum('30-Apr-2008');
OptOptions = optimset('lsqnonlin');
OptOptions = optimset(OptOptions,'display','iter');
functionHandle = @(t,theta) polyval(theta,t);

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

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

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

Local minimum possible.

lsqnonlin stopped because the final change in the sum of squares relative to

# fitFunction

---

its initial value is less than the selected value of the function tolerance.

## How To

- “@IRFitOptions” on page A-10
- “@IRFunctionCurve” on page A-12



<b>Purpose</b>	Fit Nelson-Siegel function to bond market data	
<b>Class</b>	@IRFunctionCurve	
<b>Syntax</b>	CurveObj = IRFunctionCurve.fitNelsonSiegel(Type, Settle, Instruments) CurveObj = IRFunctionCurve.fitNelsonSiegel(Type, Settle, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...)	
<b>Arguments</b>	Type	Type of interest-rate curve for a bond: zero or forward.
	Settle	Scalar or column vector of settlement dates.
	Instruments	N-by-4 data matrix for Instruments where the first column is Settle date, the second column is Maturity, the third column is the clean price, and the fourth column is a CouponRate for the bond.
	Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRFunctionCurve object: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>

<b>Basis</b>	(Optional) Day-count basis of the interest-rate curve. A scalar of integers. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul> For more information, see <code>basis</code> .
<b>IRFitOptionsObj</b>	(Optional) Object constructed from <code>IRFitOptions</code> .

## Instrument Parameters

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

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

**Basis** (Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when <code>Maturity</code> is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when an instrument was issued.
FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When <code>FirstCouponDate</code> and <code>LastCouponDate</code> are both specified, <code>FirstCouponDate</code> takes precedence in determining the coupon payment structure. If you do not specify a <code>FirstCouponDate</code> , the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified <code>FirstCouponDate</code> , a specified <code>LastCouponDate</code> determines the coupon structure of the bond. The coupon structure of a bond is truncated at the <code>LastCouponDate</code> , regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a <code>LastCouponDate</code> , the cash flow payment dates are determined from other inputs.
Face	(Optional) Face or par value. Default = 100.

---

**Note** When using Instrument parameter/value pairs, you can specify simple for a bond by specifying the InstrumentPeriod value as 0. If InstrumentBasis and InstrumentPeriod are not specified for a bond, the following default values are used: Basis is 0 (act/act) and Period is 2.

---

## Description

CurveObj = IRFunctionCurve.fitNelsonSiegel(Type, Settle, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...) fits a Nelson-Siegel function to market data for a bond. You must enter the optional arguments for Basis, Compounding, and IRFitOptionsObj as parameter/value pairs.

## Examples

```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];

CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
PlottingPoints = datenum('07-Mar-2009'):180:datenum('07-Mar-2036');
Yield = bndyield(CleanPrice,CouponRate,Settle,Maturity);

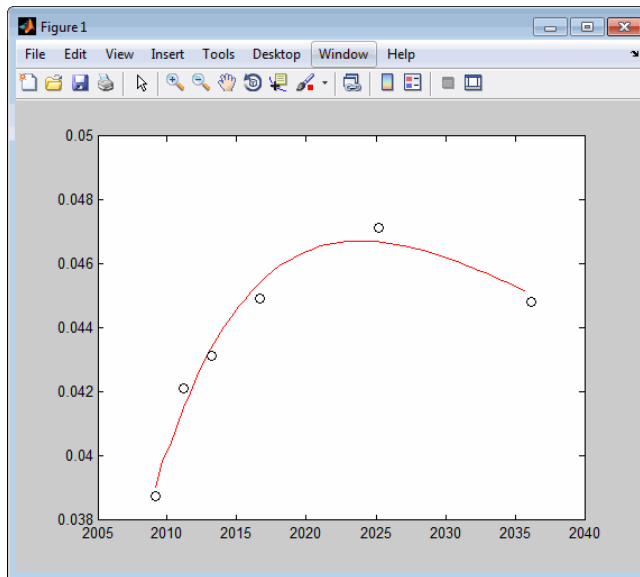
NSModel = IRFunctionCurve.fitNelsonSiegel('Zero',datenum('30-Apr-2008'),Instruments);
```

To create the plot:

```
plot(PlottingPoints,NSModel.getParYields(PlottingPoints),'r')
hold on
scatter(Maturity,Yield,'black')
datetick('x')
```

# fitNelsonSiegel

---



## How To

- “@IRFitOptions” on page A-10
- “@IRFunctionCurve” on page A-12

**Purpose** Fit smoothing spline to bond market data

**Class** @IRFunctionCurve

**Syntax**

```
CurveObj = IRFunctionCurve.fitSmoothingSpline(Type, Settle, Instruments, Lambdafun)
CurveObj = IRFunctionCurve.fitSmoothingSpline(Type, Settle, Instruments, Lambdafun, 'Parameter1', Value1, 'Parameter2', Value2, ...)
```

## Arguments

---

**Note** You must have a license for Curve Fitting Toolbox software to use the `fitSmoothingSpline` method.

---

Type	Type of interest-rate curve for a bond: forward.
Settle	Scalar or column vector of settlement dates.
Instruments	N-by-4 data matrix for Instruments where the first column is Settle date, the second column is Maturity, the third column is the clean price, and the fourth column is a CouponRate for the bond.
Lambdafun	Penalty function that takes as its input time and returns a penalty value. Use a function handle to support the penalty function. The function handle for the penalty function which takes one numeric input (time-to-maturity) and returns one numeric output (penalty to be applied to the curvature of the spline). For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.

# fitSmoothingSpline

---

---

**Note** The smoothing spline represents the forward curve. The spline is penalized for curvature by specifying a penalty function. This fit may only be done with a `FitType` of `DurationWeightedPrice`.

---

<b>Knots</b>	(Optional) Vector of knot locations (times-to-maturity); by default, knots is set to be a vector comprised of 0 and the time to maturity of all input instruments.
<b>Compounding</b>	(Optional) Scalar that sets the compounding frequency per year for the <code>IRFunctionCurve</code> object: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>
<b>Basis</b>	(Optional) Day-count basis of the interest-rate curve. A scalar of integers. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li></ul>



- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

## Instrument Parameters

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

<code>Period</code>	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2 (default), 3, 4, 6, and 12.
<code>Basis</code>	(Optional) Day-count basis of the bond. A vector of integers. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li></ul>

- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when <code>Maturity</code> is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when an instrument was issued.
FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When <code>FirstCouponDate</code> and <code>LastCouponDate</code> are both specified, <code>FirstCouponDate</code> takes precedence in determining the coupon payment structure. If you do not specify a <code>FirstCouponDate</code> , the cash flow payment dates are determined from other inputs.

LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate, regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a LastCouponDate, the cash flow payment dates are determined from other inputs.
Face	(Optional) Face or par value. Default = 100.

---

**Note** When using Instrument parameter/value pairs, you can specify simple interest for a bond by specifying the InstrumentPeriod value as 0. If InstrumentBasis and InstrumentPeriod are not specified for a bond, the following default values are used: Basis is 0 (act/act) and Period is 2.

---

## Description

Fcurve = IRFunctionCurve.fitSmoothingSpline(Type, Settle, Instruments, Lambdafun, 'Parameter1', Value1, 'Parameter2', Value2, ...) fits a smoothing spline to market data for a bond. You must enter the optional arguments for Basis, Compounding, and Knots as parameter/value pairs.

## Examples

```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];

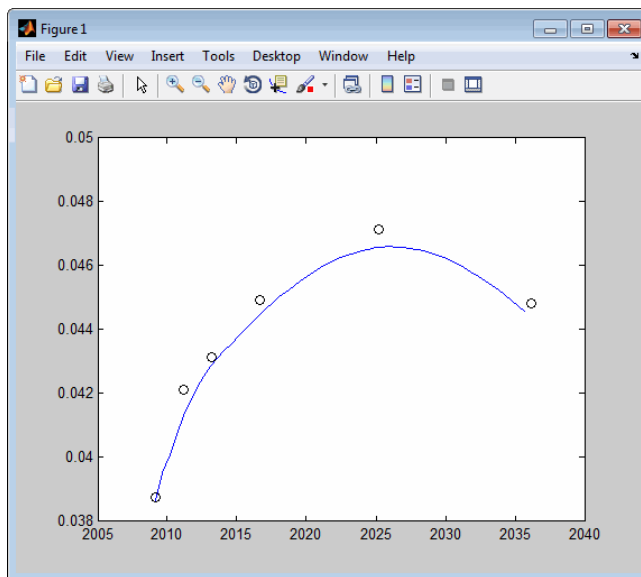
CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
```

# fitSmoothingSpline

```
PlottingPoints = datenum('07-Mar-2009'):180:datenum('07-Mar-2036');  
Yield = bndyield(CleanPrice,CouponRate,Settle,Maturity);  
  
SmoothingModel = IRFunctionCurve.fitSmoothingSpline('Forward',datenum('30-Apr-2008'),...  
Instruments,@(t) 1000);
```

To create the plot:

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



## How To

- “@IRFunctionCurve” on page A-12

---

<b>Purpose</b>	Fit Svensson function to bond market data
<b>Class</b>	@IRFunctionCurve
<b>Syntax</b>	<pre>CurveObj = IRFunctionCurve.fitSvensson(Type, Settle, Instruments) CurveObj = IRFunctionCurve.fitSvensson(Type, Settle, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...)</pre>

<b>Arguments</b>	Type	Type of interest-rate curve for a bond: zero or forward.
	Settle	Scalar or column vector of settlement dates.
	Instruments	N-by-4 data matrix for Instruments where the first column is Settle date, the second column is Maturity, the third column is the clean price, and the fourth column is a CouponRate for the bond.
	Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRFunctionCurve object: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>

**Basis** (Optional) Day-count basis of the interest-rate curve. A scalar of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see `basis`.

**IRFitOptionsObj** (Optional) Object constructed from `IRFitOptions`.

## Instrument Parameters

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

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

**Basis** (Optional) Day-count basis of the bond. A vector of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

EndMonthRule	(Optional) End-of-month rule. A vector. This rule applies only when <code>Maturity</code> is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
IssueDate	(Optional) Date when an instrument was issued.
FirstCouponDate	(Optional) Date when a bond makes its first coupon payment; used when bond has an irregular first coupon period. When <code>FirstCouponDate</code> and <code>LastCouponDate</code> are both specified, <code>FirstCouponDate</code> takes precedence in determining the coupon payment structure. If you do not specify a <code>FirstCouponDate</code> , the cash flow payment dates are determined from other inputs.
LastCouponDate	(Optional) Last coupon date of a bond before the maturity date; used when bond has an irregular last coupon period. In the absence of a specified <code>FirstCouponDate</code> , a specified <code>LastCouponDate</code> determines the coupon structure of the bond. The coupon structure of a bond is truncated at the <code>LastCouponDate</code> , regardless of where it falls, and is followed only by the bond's maturity cash flow date. If you do not specify a <code>LastCouponDate</code> , the cash flow payment dates are determined from other inputs.
Face	(Optional) Face or par value. Default = 100.



---

**Note** When using Instrument parameter/value pairs, you can specify simple interest for a bond by specifying the InstrumentPeriod value as 0. If InstrumentBasis and InstrumentPeriod are not specified for a bond, the following default values are used: Basis is 0 (act/act) and Period is 2.

---

## Description

CurveObj = IRFunctionCurve.fitSvensson(Type, Settle, Instruments, 'Parameter1', Value1, 'Parameter2', Value2, ...) fits the Svensson function to bond market data. You must enter the optional arguments for Basis, Compounding, and IRFitOptionsObj as parameter/value pairs.

## Examples

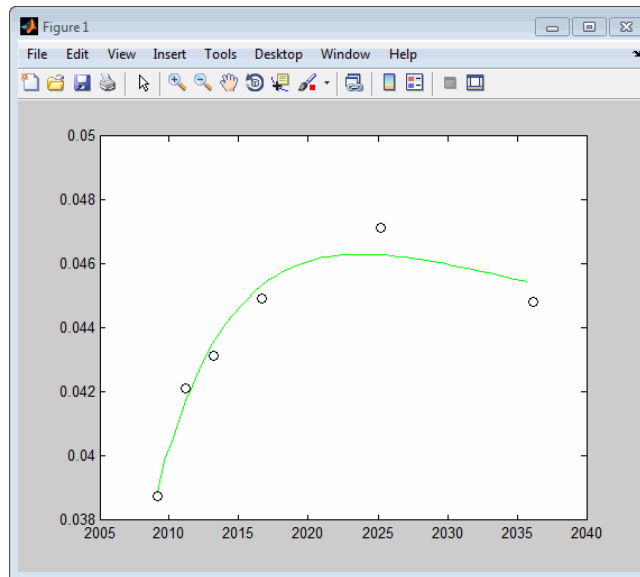
```
Settle = repmat(datenum('30-Apr-2008'),[6 1]);
Maturity = [datenum('07-Mar-2009');datenum('07-Mar-2011');...
datenum('07-Mar-2013');datenum('07-Sep-2016');...
datenum('07-Mar-2025');datenum('07-Mar-2036')];

CleanPrice = [100.1;100.1;100.8;96.6;103.3;96.3];
CouponRate = [0.0400;0.0425;0.0450;0.0400;0.0500;0.0425];
Instruments = [Settle Maturity CleanPrice CouponRate];
PlottingPoints = datenum('07-Mar-2009'):180:datenum('07-Mar-2036');
Yield = bndyield(CleanPrice,CouponRate,Settle,Maturity);

SvenssonModel = IRFunctionCurve.fitSvensson('Zero',datenum('30-Apr-2008'),Instruments);
```

To create a plot:

```
plot(PlottingPoints,SvenssonModel.getParYields(PlottingPoints),'g')
hold on
scatter(Maturity,Yield,'black')
datetick('x')
```



## How To

- “@IRFitOptions” on page A-10
- “@IRFunctionCurve” on page A-12

**Purpose** Get discount factors for input dates for IRDataCurve

**Class** @IRDataCurve

**Syntax** F = getDiscountFactors(CurveObj, InpDates)

**Arguments**

CurveObj	Interest-rate curve object that is constructed using IRDataCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.

**Description** F = getDiscountFactors(CurveObj, InpDates) returns discount factors for the input dates.

## Examples

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data);
irdc.getDiscountFactors(today+30:30:today+720)
ans =

    0.9986
    0.9971
    0.9956
    0.9940
    0.9924
    0.9907
    0.9890
    0.9873
    0.9855
    0.9836
    0.9817
    0.9798
    0.9778
    0.9757
```

# getDiscountFactors

---

0.9736  
0.9715  
0.9693  
0.9671  
0.9649  
0.9626  
0.9602  
0.9578  
0.9554  
0.9529

## How To

- “@IRDataCurve” on page A-7

<b>Purpose</b>	Get discount factors for input dates for IRFunctionCurve	
<b>Class</b>	@IRFunctionCurve	
<b>Syntax</b>	F = getDiscountFactors(CurveObj, InpDates)	
<b>Arguments</b>	CurveObj	Interest-rate curve object that is constructed using the IRFunctionCurve.
	InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
<b>Description</b>	F = getDiscountFactors(CurveObj, InpDates) returns discount factors for the input dates.	
<b>Examples</b>	<pre>irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t)); irfc.getDiscountFactors(today+30:30:today+720) ans =      0.9984     0.9967     0.9950     0.9933     0.9916     0.9899     0.9881     0.9864     0.9846     0.9828     0.9810     0.9792     0.9773     0.9755     0.9736     0.9717</pre>	

# getDiscountFactors

---

0.9698  
0.9679  
0.9660  
0.9641  
0.9621  
0.9602  
0.9582  
0.9562

## How To

- “@IRFunctionCurve” on page A-12

<b>Purpose</b>	Get forward rates for input dates for IRDataCurve
<b>Class</b>	@IRDataCurve
<b>Syntax</b>	<pre>F = getForwardRates(CurveObj, InpDates) F = getforwardrates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)</pre>
<b>Arguments</b>	
CurveObj	Interest-rate curve object that is constructed using IRDataCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for forward rates are: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>
Basis	(Optional) Day-count basis values for the forward rates: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li></ul>

# getForwardRates

---

- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

## Description

`F = getForwardRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)` returns forward rates for the input dates. You must enter the optional arguments for **Basis** and **Compounding** as parameter/value pairs.

## Examples

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data);
irdc.getForwardRates(today+30:30:today+720)
ans =

    0.0174
    0.0180
    0.0187
    0.0193
    0.0199
    0.0205
    0.0212
    0.0218
```



0.0224  
0.0230  
0.0237  
0.0243  
0.0249  
0.0255  
0.0262  
0.0268  
0.0274  
0.0280  
0.0287  
0.0293  
0.0299  
0.0305  
0.0312  
0.0318

## How To

- “@IRDataCurve” on page A-7

# getForwardRates

---

**Purpose** Get forward rates for input dates for IRFunctionCurve

**Class** @IRFunctionCurve

**Syntax**  
F = getForwardRates(CurveObj, InpDates)  
F = getforwardrates(CurveObj, InpDates, 'Parameter1',  
Value1, 'Parameter2', Value2, ...)

**Arguments**

CurveObj	Interest-rate curve object that is constructed using IRFunctionCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for the forward rates are: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>
Basis	(Optional) Day-count basis for the forward rates: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li></ul>

- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

## Description

`F = getForwardRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)` returns forward rates for the input dates. You must enter the optional arguments for **Basis** and **Compounding** as parameter/value pairs.

## Examples

```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));  
irfc.getForwardRates(today+30:30:today+720)  
ans =
```

```
0.0202  
0.0205  
0.0207  
0.0210  
0.0212  
0.0215  
0.0217  
0.0219  
0.0222  
0.0224  
0.0226  
0.0229  
0.0231
```

# getForwardRates

---

0.0233  
0.0235  
0.0238  
0.0240  
0.0242  
0.0244  
0.0247  
0.0249  
0.0251  
0.0253  
0.0255

## How To

- “@IRFunctionCurve” on page A-12

<b>Purpose</b>	Get par yields for input dates for IRDataCurve	
<b>Class</b>	@IRDataCurve	
<b>Syntax</b>	F = getParYields(CurveObj, InpDates) F = getParYields(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)	
<b>Arguments</b>	CurveObj	Interest-rate curve object that is constructed using IRDataCurve.
	InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
	Compounding	(Optional) Scalar that sets the compounding frequency per year for the par yield rates are: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>
	Basis	(Optional) Day-count basis values for the par yield rates: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li></ul>

# getParYields

---

- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

## Description

`F = getParYields(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)` returns par yields for the input dates. You must enter the optional arguments for **Basis** and **Compounding** as parameter/value pairs.

## Examples

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data);
irdc.getParYields(today+30:30:today+720)ans =
```

```
0.0174
0.0179
0.0181
0.0185
0.0187
0.0191
0.0194
0.0195
0.0199
0.0202
```

0.0205  
0.0208  
0.0212  
0.0215  
0.0218  
0.0221  
0.0224  
0.0228  
0.0231  
0.0233  
0.0236  
0.0239  
0.0242  
0.0245

## How To

- “@IRDataCurve” on page A-7

# getParYields

---

**Purpose** Get par yields for input dates for IRFunctionCurve

**Class** @IRFunctionCurve

**Syntax**  
F = getParYields(CurveObj, InpDates)  
F = getParYields(CurveObj, InpDates, 'Parameter1',  
Value1, 'Parameter2', Value2, ...)

**Arguments**

CurveObj	Interest-rate curve object that is constructed using IRFunctionCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for par yield rates are: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>
Basis	(Optional) Day-count basis values for par yield rates: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li></ul>



- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

## Description

`F = getParYields(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)` returns par yields for the input dates. You must enter the optional arguments for **Basis** and **Compounding** as parameter/value pairs.

## Examples

```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));
irfc.getParYields(today+30:30:today+720)
ans =

    0.0202
    0.0205
    0.0205
    0.0207
    0.0207
    0.0209
    0.0210
    0.0209
    0.0211
    0.0212
    0.0213
```

# getParYields

---

0.0214  
0.0216  
0.0217  
0.0218  
0.0220  
0.0220  
0.0222  
0.0223  
0.0223  
0.0225  
0.0226  
0.0227  
0.0228

## How To

- “@IRFunctionCurve” on page A-12

<b>Purpose</b>	Get zero rates for input dates for IRDataCurve
<b>Class</b>	@IRDataCurve
<b>Syntax</b>	<pre>F = getZeroRates(CurveObj, InpDates) F = getZeroRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)</pre>
<b>Arguments</b>	
CurveObj	Interest-rate curve object that is constructed using IRDataCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for zero rates are: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>
Basis	(Optional) Day-count basis values for zero rates: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li></ul>

# getZeroRates

---

- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

## Description

`F = getZeroRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)` returns zero rates for the input dates. You must enter the optional arguments for **Basis** and **Compounding** as parameter/value pairs.

## Examples

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Zero',today,Dates,Data);
irdc.getZeroRates(today+30:30:today+720)
ans =

    0.0174
    0.0177
    0.0180
    0.0183
    0.0187
    0.0190
    0.0193
    0.0196
    0.0199
    0.0202
    0.0205
```

0.0208  
0.0212  
0.0215  
0.0218  
0.0221  
0.0224  
0.0227  
0.0230  
0.0233  
0.0237  
0.0240  
0.0243  
0.0246

## How To

- “@IRDataCurve” on page A-7

# getZeroRates

---

**Purpose** Get zero rates for input dates for IRFunctionCurve

**Class** @IRFunctionCurve

**Syntax**  
F = getZeroRates(CurveObj, InpDates)  
F = getZeroRates(CurveObj, InpDates, 'Parameter1',  
Value1, 'Parameter2', Value2, ...)

**Arguments**

CurveObj	Interest-rate curve object that is constructed using IRFunctionCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.
Compounding	(Optional) Scalar that sets the compounding frequency per year for zero rates are: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>
Basis	(Optional) Day-count basis value for zero rates: <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li></ul>

- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

## Description

`F = getZeroRates(CurveObj, InpDates, 'Parameter1', Value1, 'Parameter2', Value2, ...)` returns zero rates for the input dates. You must enter the optional arguments for **Basis** and **Compounding** as parameter/value pairs.

## Examples

```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));
irfc.getZeroRates(today+30:30:today+720)
ans =

    0.0202
    0.0204
    0.0205
    0.0206
    0.0207
    0.0209
    0.0210
    0.0211
    0.0212
    0.0213
    0.0214
```

# getZeroRates

---

0.0216  
0.0217  
0.0218  
0.0219  
0.0220  
0.0221  
0.0223  
0.0224  
0.0225  
0.0226  
0.0227  
0.0228  
0.0229

## How To

- “@IRFunctionCurve” on page A-12



**Purpose** Construct specific options for bootstrapping interest-rate curve object

**Class** @IRBootstrapOptions

**Syntax** `mybootoptions = IRBootstrapOptions('Param1', Value1)`

**Arguments**

ConvexityAdjustment	(Optional) Controls the convexity adjustment to interest-rate futures. This can be specified as a function handle that takes one numeric input (time-to-maturity) and returns one numeric output, ConvexityAdjustment. For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.  Alternatively, you can define ConvexityAdjustment as an N-by-1 vector of values, where N is the number of interest-rate futures.  In either case, the ConvexityAdjustment is subtracted from the futures rate.
---------------------	---

**Description** `mybootoptions = IRBootstrapOptions('Param1', Value1)` constructs an IRBootstrapOptionsObj structure. The IRBootstrapOptionsObj is used with the bootstrap method.

**Examples** `mybootoptions = IRBootstrapOptions('ConvexityAdjustment', repmat(.005,10,1))`

**How To**

- “@IRDataCurve” on page A-7

# IRDataCurve

---

**Purpose** Construct interest-rate curve object from dates and data

**Class** @IRDataCurve

**Syntax**  
CurveObj = IRDataCurve(Type, Settle)  
CurveObj = IRDataCurve(Type, Settle, Dates, Data, 'Parameter1', Value1, 'Parameter2', Value2, ...)

**Arguments**

Type	Type of interest-rate curve. Acceptable values are forward, zero, or discount.
Settle	Scalar of settlement dates.
Dates	(Optional) Dates corresponding to rate data.
Data	(Optional) Interest-rate data for the curve object.
Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRDataCurve object: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li><li>• 12 = Monthly compounding</li></ul>

**Basis** (Optional) Day-count basis of the interest-rate curve. A scalar of integers.

- 0 = actual/actual (default)
- 1 = 30/360 (SIA)
- 2 = actual/360
- 3 = actual/365
- 4 = 30/360 (BMA)
- 5 = 30/360 (ISDA)
- 6 = 30/360 (European)
- 7 = actual/365 (Japanese)
- 8 = actual/actual (ICMA)
- 9 = actual/360 (ICMA)
- 10 = actual/365 (ICMA)
- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**InterpMethod** (Optional) Values are:

- 'linear' — Linear interpolation (default).
- 'constant' — Piecewise constant interpolation.
- 'pchip' — Piecewise cubic Hermite interpolation.
- 'spline' — Cubic spline interpolation.

# IRDataCurve

---

## Description

`CurveObj = IRDataCurve(Type, Settle, Dates, Data, 'Parameter1', Value1, 'Parameter2', Value2, ...)` constructs an interest-rate curve with the optionally specified `Dates` and `Data`. You must enter the optional arguments for `Basis`, `Compounding`, and `InterpMethod` as parameter/value pairs.

Alternatively, an `IRDataCurve` object can be bootstrapped from market data using the `bootstrap` method.

After an `IRDataCurve` curve object is constructed, you can use the following methods to determine the forward rates, zero rates, and discount factors. In addition, you can use the `toRateSpec` method to convert the interest-rate curve object to a `RateSpec` structure.

Method	Description
<code>getForwardRates</code>	Returns forward rates for input dates.
<code>getZeroRates</code>	Returns zero rates for input dates.
<code>getDiscountFactors</code>	Returns discount factors for input dates.
<code>getParYields</code>	Returns par yields for input dates.
<code>toRateSpec</code>	Converts to be a <code>RateSpec</code> object; this structure is identical to the <code>RateSpec</code>
<code>bootstrap</code>	Bootstraps an interest rate curve from market data.

## Examples

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;  
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);  
irdc = IRDataCurve('Zero',today,Dates,Data)
```

```
irdc =
```

```
Properties:
```

```
Dates: [8x1 double]
Data: [8x1 double]
InterpMethod: 'linear'
Type: 'Zero'
Settle: 733599
Compounding: 2
Basis: 0
```

## How To

- “@IRCurve” on page A-4

# IRFitOptions

---

<b>Purpose</b>	Construct specific options for fitting interest-rate curve object
<b>Class</b>	@IRFitOptions
<b>Syntax</b>	<pre>myfitoptions = IRFitOptions(InitialGuess) myfitoptions = IRFitOptions(InitialGuess, 'Parameter1', Value1)</pre>

<b>Arguments</b>	<b>InitialGuess</b>	Initial guess for the parameters of the curve function. Vector of values for the starting point of the optimization.
	<b>FitType</b>	(Optional) Price, Yield, or DurationWeightedPrice determines which is minimized in the curve fitting process. The default is DurationWeightedPrice.
	<b>UpperBound</b>	(Optional) Lower bound for the parameters of the curve function.
	<b>LowerBound</b>	(Optional) Upper bound for the parameters of the curve function.
	<b>OptOptions</b>	(Optional) Optimization structure based on the output from the Optimization Toolbox function <code>optimset</code> . This optimization structure is evaluated by <code>lsqnonlin</code> .

**Description** `myfitoptions = IRFitOptions('Param1', Value1)` constructs the `IRFitOptions` structure with an initial guess or with an initial guess and bounds. You must enter the optional arguments for `FitType`, `UpperBound`, `LowerBound`, and `OptOptions` as parameter/value pairs.

---

**Note** `IRFitOptions` constructor must be used with `fitFunction` method when building a custom fitting function.

---

## Examples

```
myfitoptions = IRFitOptions([7 2 1 0], 'FitType', 'yield')
```

```
myfitoptions =
```

```
Properties:
```

```
    FitType: 'yield'  
InitialGuess: [7 2 1 0]  
    UpperBound: []  
    LowerBound: []  
    OptOptions: []
```

## How To

- “@IRFunctionCurve” on page A-12

# IRFunctionCurve

---

**Purpose** Construct interest-rate curve object from function handle or function and fit to market data

**Class** @IRFunctionCurve

**Syntax**  
CurveObj = IRFunctionCurve(Type, Settle, FunctionHandle)  
CurveObj = IRFunctionCurve(Type, Settle, FunctionHandle, 'Parameter1', Value1, 'Parameter2', Value2, ...)

**Arguments**

Type	Type of interest-rate curve: zero, forward, or discount.
Settle	Scalar of settlement dates.
FunctionHandle	Function handle that defines the interest-rate curve. The function handle requires one numeric input (time-to-maturity) and returns one numeric output (interest rate or discount factor). For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.
Compounding	(Optional) Scalar that sets the compounding frequency per year for the IRFunctionCurve object: <ul style="list-style-type: none"><li>• -1 = Continuous compounding</li><li>• 1 = Annual compounding</li><li>• 2 = Semiannual compounding (default)</li><li>• 3 = Compounding three times per year</li><li>• 4 = Quarterly compounding</li><li>• 6 = Bimonthly compounding</li></ul>



<b>Basis</b>	<ul style="list-style-type: none"><li>• 12 = Monthly compounding</li></ul> <p>(Optional) Day-count basis of the bond. A scalar of integers.</p> <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul> <p>For more information, see basis.</p>
--------------	--

## Description

`CurveObj = IRFunctionCurve(Type, Settle, FunctionHandle, 'Parameter1', Value1, 'Parameter2', Value2, ...)` constructs an interest-rate curve object directly by specifying a function handle. You must enter the optional arguments for **Basis** and **Compounding** as parameter/value pairs.

After you use the `IRFunctionCurve` constructor to create an `IRFunctionCurve` object, you can fit the bond using the following methods.

# IRFunctionCurve

Method	Description
getForwardRates	Returns forward rates for input dates.
getZeroRates	Returns zero rates for input dates.
getDiscountFactors	Returns discount factors for input dates.
getParYields	Returns par yields for input dates.
toRateSpec	Converts to be a RateSpec object.  This RateSpec structure is identical to the RateSpec produced by the Financial Derivatives Toolbox function <code>intenvset</code> .

Alternatively, you can construct an IRFunctionCurve object using the following static methods.

Static Method	Description
fitNelsonSiegel	Fits a Nelson-Siegel function to market data.
fitSvensson	Fits a Svensson function to market data.
fitSmoothingSpline	Fits a smoothing spline function to market data.
fitFunction	Fits a custom function to market data.

## Examples

```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t))
```

```
irfc =
```

```
Properties:
```

```
FunctionHandle: @(t)polyval([-0.0001,0.003,0.02],t)
```

```
Type: 'Forward'
```

Settle: 733599  
Compounding: 2  
Basis: 0

## How To

- “@IRCurve” on page A-4

# liborduration

---

**Purpose** Duration of LIBOR-based interest-rate swap

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

**Arguments**

<code>SwapFixRate</code>	Scalar or column vector of swap fixed rates in decimal.
<code>Tenor</code>	Scalar or column vector indicating life of the swap in years. Fractional numbers are rounded upward.
<code>Settle</code>	Scalar or column vector of settlement dates.

**Description** `[PayFixDuration GetFixDuration] = liborduration(SwapFixRate, Tenor, Settle)` computes the duration of LIBOR-based interest-rate swaps.

`PayFixDuration` is the modified duration, in years, realized when entering pay-fix side of the swap.

`GetFixDuration` is the modified duration, in years, realized when entering receive-fix side of the swap.

**Examples** Given the data

```
SwapFixRate = 0.0383;  
Tenor = 7;  
Settle = datenum('11-Oct-2002');
```

compute the swap durations.

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

```
PayFixDuration =
```

-4.7567

GetFixDuration =

4.7567

## See Also

[liborfloat2fixed](#) | [liborprice](#)

# liborfloat2fixed

---

**Purpose** Compute par fixed-rate of swap given 3-month LIBOR data

**Syntax** [FixedSpec, ForwardDates, ForwardRates] =  
liborfloat2fixed(ThreeMonthRates, Settle, Tenor, StartDate,  
Interpolation, ConvexAdj, RateParam, InArrears, Sigma,  
FixedCompound, FixedBasis)

**Arguments**

ThreeMonthRates	Three-month Eurodollar futures data or forward rate agreement data. (A forward rate agreement stipulates that a certain interest rate applies to a certain principal amount for a given future time period.) An n-by-3 matrix in the form of [month year IMMQuote]. The floating rate is assumed to compound quarterly and to accrue on an actual/360 basis.
Settle	Settlement date of the swap. Scalar.
Tenor	Life of the swap. Scalar.
StartDate	(Optional) Scalar value to denote reference date for valuation of (forward) swap. This in effect allows forward swap valuation. Default = Settle.
Interpolation	(Optional) Interpolation method to determine applicable forward rate for months when no Eurodollar data is available. Default is 'linear' or 1. Other possible values are 'Nearest' or 0, and 'Cubic' or 2.
ConvexAdj	(Optional) Default = 0 (off). 1 = on. Denotes whether futures/forward convexity adjustment is required. Pertains to forward rate adjustments when those rates are taken from Eurodollar futures data.

RateParam	<p>(Optional) Short-rate model's parameters (Hull-White) [a S], where the short-rate process is:</p> $dr = [\theta(t) - ar]dt + Sdz.$ <p>Default = [0.05 0.015].</p>
InArrears	<p>(Optional) Default = 0 (off). Set to 1 for on. If on, the routine does an automatic a convexity adjustment to forward rates.</p>
Sigma	<p>(Optional) Overall annual volatility of caplets.</p>
FixedCompound	<p>(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.</p>
FixedBasis	<p>(Optional) Scalar value. Basis of the fixed side.</p> <ul style="list-style-type: none"> <li>• 0 = actual/actual (default)</li> <li>• 1 = 30/360 (SIA)</li> <li>• 2 = actual/360</li> <li>• 3 = actual/365</li> <li>• 4 = 30/360 (BMA)</li> <li>• 5 = 30/360 (ISDA)</li> <li>• 6 = 30/360 (European)</li> <li>• 7 = actual/365 (Japanese)</li> <li>• 8 = actual/actual (ICMA)</li> <li>• 9 = actual/360 (ICMA)</li> <li>• 10 = actual/365 (ICMA)</li> <li>• 11 = 30/360E (ICMA)</li> <li>• 12 = actual/actual (ISDA)</li> </ul> <p>For more information, see basis.</p>

# liborfloat2fixed

---

## Description

[FixedSpec, ForwardDates, ForwardRates] = liborfloat2fixed(ThreeMonthRates, Settle, Tenor, StartDate, Interpolation, ConvexAdj, RateParam, InArrears, Sigma, FixedCompound, FixedBasis) computes forward rates, dates, and the swap fixed rate.

FixedSpec specifies the structure of the fixed-rate side of the swap:

- Coupon: Par-swap rate
- Settle: Start date
- Maturity: End date
- Period: Frequency of payment
- Basis: Accrual basis

ForwardDates are dates corresponding to ForwardRates (all third Wednesdays of the month, spread 3 months apart). The first element is the third Wednesday immediately after Settle.

ForwardRates are forward rates corresponding to the forward dates, quarterly compounded, and on the actual/360 basis.

---

**Note** To preserve input integrity, Tenor is rounded upward to the closest integer. Currently traded tenors are 2, 5, and 10 years.

---

The function assumes that floating-rate observations occur quarterly on the third Wednesday of a delivery month. The first delivery month is the month of the first third Wednesday after Settle. Floating-side payments occur on the third-month anniversaries of observation dates.

## Examples

Use the supplied EDdata.xls file as input to a liborfloat2fixed computation.

```
[EDFutData, textdata] = xlsread('EDdata.xls');  
Settle                = datenum('15-Oct-2002');  
Tenor                 = 2;
```



```
[FixedSpec, ForwardDates, ForwardRates] =...  
liborfloat2fixed(EDFutData(:,1:3), Settle, Tenor)
```

```
FixedSpec =
```

```
    Coupon: 0.0222  
    Settle: '16-Oct-2002'  
    Maturity: '16-Oct-2004'  
    Period: 4  
    Basis: 1
```

```
ForwardDates =
```

```
    731505  
    731596  
    731687  
    731778  
    731869  
    731967  
    732058  
    732149
```

```
ForwardRates =
```

```
    0.0177  
    0.0166  
    0.0170  
    0.0188  
    0.0214  
    0.0248  
    0.0279  
    0.0305
```

## See Also

[liborduration](#) | [liborprice](#)

# liborprice

---

**Purpose** Price swap given swap rate

**Syntax** Price = liborprice(ThreeMonthRates, Settle, Tenor, SwapRate, StartDate, Interpolation, ConvexAdj, RateParam, InArrears, Sigma, FixedCompound, FixedBasis)

**Arguments**

ThreeMonthRates	Three-month Eurodollar futures data or forward rate agreement data. (A forward rate agreement stipulates that a certain interest rate applies to a certain principal amount for a given future time period.) An n-by-3 matrix in the form of [month year IMMQuote]. The floating rate is assumed to compound quarterly and to accrue on an actual/360 basis.
Settle	Settlement date of swap. Scalar.
Tenor	Life of the swap. Scalar.
SwapRate	Swap rate in decimal.
StartDate	(Optional) Scalar value to denote reference date for valuation of (forward) swap. This in effect allows forward swap valuation. Default = Settle.
Interpolation	(Optional) Interpolation method to determine applicable forward rate for months when no Eurodollar data is available. Default is 'linear' or 1. Other possible values are 'Nearest' or 0, and 'Cubic' or 2.
ConvexAdj	(Optional) Default = 0 (off). 1 = on. Denotes whether futures/forward convexity adjustment is required. Pertains to forward rate adjustments when those rates are taken from Eurodollar futures data.

RateParam	<p>(Optional) Short-rate model's parameters (Hull-White) [a S], where the short-rate process is:</p> $dr = [\theta(t) - ar]dt + Sdz.$ <p>Default = [0.05 0.015].</p>
InArrears	<p>(Optional) Default = 0 (off). Set to 1 for on. If on, the routine does an automatic convexity adjustment to forward rates.</p>
Sigma	<p>(Optional) Overall annual volatility of caplets.</p>
FixedCompound	<p>(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.</p>
FixedBasis	<p>(Optional) Scalar value. Basis of the fixed side.</p> <ul style="list-style-type: none"> <li>• 0 = actual/actual (default)</li> <li>• 1 = 30/360 (SIA)</li> <li>• 2 = actual/360</li> <li>• 3 = actual/365</li> <li>• 4 = 30/360 (BMA)</li> <li>• 5 = 30/360 (ISDA)</li> <li>• 6 = 30/360 (European)</li> <li>• 7 = actual/365 (Japanese)</li> <li>• 8 = actual/actual (ICMA)</li> <li>• 9 = actual/360 (ICMA)</li> <li>• 10 = actual/365 (ICMA)</li> <li>• 11 = 30/360E (ICMA)</li> <li>• 12 = actual/actual (ISDA)</li> </ul> <p>For more information, see basis.</p>

# liborprice

---

## Description

`Price = liborprice(ThreeMonthRates, Settle, Tenor, SwapRate, StartDate, Interpolation, ConvexAdj, RateParam, InArrears, Sigma, FixedCompound, FixedBasis)` computes the price per \$100 notional value of a swap given the swap rate. A positive result indicates that fixed side is more valuable than the floating side.

Price is the present value of the difference between floating and fixed-rate sides of the swap per \$100 notional.

## Examples

This example shows that a swap paying the par swap rate has a value of 0.

Load the input data.

```
[EDFutData, textdata] = xlsread('EDdata.xls');  
Settle = datenum('15-Oct-2002');  
Tenor = 2;
```

Compute the fixed rate from the Eurodollar data.

```
FixedSpec = liborfloat2fixed(EDFutData(:,1:3), Settle, Tenor)
```

```
Coupon: 0.0222  
Settle: '16-Oct-2002'  
Maturity: '16-Oct-2004'  
Period: 4  
Basis: 1
```

Compute the price of a par swap.

```
Price = liborprice(EDFutData(:,1:3), Settle, Tenor, FixedSpec.Coupon)
```

```
Price =
```

```
4.1633e-015
```

MATLAB computes a value for Price that is effectively equal to 0.

**See Also**

[liborduration](#) | [liborfloat2fixed](#)

# mbscfamounts

---

**Purpose** Cash flow and time mapping for mortgage pool

**Syntax** [CFlowAmounts, CFlowDates, TFactors, Factors, Payment, Principal,... Interest, Prepayment] = mbscfamounts(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)

**Arguments**

Settle	Settlement date. A serial date number or date string. <b>Settle</b> must be earlier than <b>Maturity</b> .
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 = <b>GrossRate</b> .
Delay	(Optional) Delay in days. Default is 0 (no delay).
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. If you input a customized prepayment matrix, set <b>PrepaySpeed</b> to [].
PrepayMatrix	(Optional) Used only when <b>PrepaySpeed</b> is unspecified. Customized prepayment vector. A NaN-padded matrix of size $\max(\text{TermRemaining})$ -by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

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

## Description

[CFlowAmounts, CFlowDates, TFactors, Factors, Payment, Principal, Interest, Prepayment] = mbscfamounts(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes cash flows between settle and maturity dates, the corresponding time factors in months from settle, and the mortgage factor (the fraction of loan principal outstanding).

CFlowAmounts is a vector of cash flows starting from Settle through end of the last month (Maturity).

CFlowDates indicates when cash flows occur, including at Settle. A negative number at Settle indicates accrued interest is due.

TFactors is a vector of times in months from Settle, corresponding to each cash flow.

Factors is a vector of mortgage factors (the fraction of the balance still outstanding at the end of each month).

Payment is a NMBS-by-P matrix of total monthly payment.

Principal is a NMBS-by-P matrix of principal portion of the payment

Interest is a NMBS-by-P matrix of interest portion of the payment.

Prepayment is a NMBS-by-P matrix of unscheduled payment of principal.

## Examples

### Calculate Cash Flow Amounts and Dates, Time Factors, and Mortgage Factors for a Single Mortgage

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

Define the mortgage characteristics.

```
Settle      = datenum('17-April-2002');
Maturity    = datenum('1-Jan-2030');
IssueDate   = datenum('1-Jan-2000');
GrossRate   = 0.08125;
CouponRate  = 0.075;
Delay       = 14;
PrepaySpeed = 100;
```

# mbscfamounts

---

Use `mbscfamounts` to evaluate the mortgage.

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

CFlowAmounts =

Columns 1 through 7

-0.0033	0.0118	0.0120	0.0121	0.0120	0.0119	0.0119
---------	--------	--------	--------	--------	--------	--------

Columns 8 through 14

0.0118	0.0117	0.0117	0.0116	0.0115	0.0115	0.0114
--------	--------	--------	--------	--------	--------	--------

Columns 15 through 21

0.0114	0.0113	0.0112	0.0112	0.0111	0.0110	0.0110
--------	--------	--------	--------	--------	--------	--------

Columns 22 through 28

0.0109	0.0109	0.0108	0.0107	0.0107	0.0106	0.0106
--------	--------	--------	--------	--------	--------	--------

Columns 29 through 35

0.0105	0.0105	0.0104	0.0103	0.0103	0.0102	0.0102
--------	--------	--------	--------	--------	--------	--------

Columns 36 through 42

0.0101	0.0101	0.0100	0.0099	0.0099	0.0098	0.0098
--------	--------	--------	--------	--------	--------	--------

Columns 43 through 49

0.0097	0.0097	0.0096	0.0096	0.0095	0.0095	0.0094
--------	--------	--------	--------	--------	--------	--------

Columns 50 through 56



0.0094	0.0093	0.0093	0.0092	0.0092	0.0091	0.0090
Columns 57 through 63						
0.0090	0.0089	0.0089	0.0088	0.0088	0.0087	0.0087
Columns 64 through 70						
0.0087	0.0086	0.0086	0.0085	0.0085	0.0084	0.0084
Columns 71 through 77						
0.0083	0.0083	0.0082	0.0082	0.0081	0.0081	0.0080
Columns 78 through 84						
0.0080	0.0079	0.0079	0.0079	0.0078	0.0078	0.0077
Columns 85 through 91						
0.0077	0.0076	0.0076	0.0075	0.0075	0.0075	0.0074
Columns 92 through 98						
0.0074	0.0073	0.0073	0.0073	0.0072	0.0072	0.0071
Columns 99 through 105						
0.0071	0.0070	0.0070	0.0070	0.0069	0.0069	0.0068
Columns 106 through 112						
0.0068	0.0068	0.0067	0.0067	0.0066	0.0066	0.0066
Columns 113 through 119						

# mbscfamounts

---

0.0065 0.0065 0.0065 0.0064 0.0064 0.0063 0.0063

Columns 120 through 126

0.0063 0.0062 0.0062 0.0062 0.0061 0.0061 0.0061

Columns 127 through 133

0.0060 0.0060 0.0059 0.0059 0.0059 0.0058 0.0058

Columns 134 through 140

0.0058 0.0057 0.0057 0.0057 0.0056 0.0056 0.0056

Columns 141 through 147

0.0055 0.0055 0.0055 0.0054 0.0054 0.0054 0.0053

Columns 148 through 154

0.0053 0.0053 0.0052 0.0052 0.0052 0.0052 0.0051

Columns 155 through 161

0.0051 0.0051 0.0050 0.0050 0.0050 0.0049 0.0049

Columns 162 through 168

0.0049 0.0048 0.0048 0.0048 0.0048 0.0047 0.0047

Columns 169 through 175

0.0047 0.0046 0.0046 0.0046 0.0046 0.0045 0.0045

Columns 176 through 182

0.0045 0.0044 0.0044 0.0044 0.0044 0.0043 0.0043

Columns 183 through 189

0.0043	0.0043	0.0042	0.0042	0.0042	0.0041	0.0041
--------	--------	--------	--------	--------	--------	--------

Columns 190 through 196

0.0041	0.0041	0.0040	0.0040	0.0040	0.0040	0.0039
--------	--------	--------	--------	--------	--------	--------

Columns 197 through 203

0.0039	0.0039	0.0039	0.0038	0.0038	0.0038	0.0038
--------	--------	--------	--------	--------	--------	--------

Columns 204 through 210

0.0037	0.0037	0.0037	0.0037	0.0036	0.0036	0.0036
--------	--------	--------	--------	--------	--------	--------

Columns 211 through 217

0.0036	0.0035	0.0035	0.0035	0.0035	0.0035	0.0034
--------	--------	--------	--------	--------	--------	--------

Columns 218 through 224

0.0034	0.0034	0.0034	0.0033	0.0033	0.0033	0.0033
--------	--------	--------	--------	--------	--------	--------

Columns 225 through 231

0.0033	0.0032	0.0032	0.0032	0.0032	0.0031	0.0031
--------	--------	--------	--------	--------	--------	--------

Columns 232 through 238

0.0031	0.0031	0.0031	0.0030	0.0030	0.0030	0.0030
--------	--------	--------	--------	--------	--------	--------

Columns 239 through 245

0.0030	0.0029	0.0029	0.0029	0.0029	0.0029	0.0028
--------	--------	--------	--------	--------	--------	--------

# mbscfamounts

---

Columns 246 through 252

0.0028	0.0028	0.0028	0.0028	0.0027	0.0027	0.0027
--------	--------	--------	--------	--------	--------	--------

Columns 253 through 259

0.0027	0.0027	0.0026	0.0026	0.0026	0.0026	0.0026
--------	--------	--------	--------	--------	--------	--------

Columns 260 through 266

0.0025	0.0025	0.0025	0.0025	0.0025	0.0024	0.0024
--------	--------	--------	--------	--------	--------	--------

Columns 267 through 273

0.0024	0.0024	0.0024	0.0024	0.0023	0.0023	0.0023
--------	--------	--------	--------	--------	--------	--------

Columns 274 through 280

0.0023	0.0023	0.0023	0.0022	0.0022	0.0022	0.0022
--------	--------	--------	--------	--------	--------	--------

Columns 281 through 287

0.0022	0.0021	0.0021	0.0021	0.0021	0.0021	0.0021
--------	--------	--------	--------	--------	--------	--------

Columns 288 through 294

0.0020	0.0020	0.0020	0.0020	0.0020	0.0020	0.0020
--------	--------	--------	--------	--------	--------	--------

Columns 295 through 301

0.0019	0.0019	0.0019	0.0019	0.0019	0.0019	0.0018
--------	--------	--------	--------	--------	--------	--------

Columns 302 through 308

0.0018	0.0018	0.0018	0.0018	0.0018	0.0017	0.0017
--------	--------	--------	--------	--------	--------	--------

Columns 309 through 315

0.0017 0.0017 0.0017 0.0017 0.0017 0.0016 0.0016

Columns 316 through 322

0.0016 0.0016 0.0016 0.0016 0.0016 0.0015 0.0015

Columns 323 through 329

0.0015 0.0015 0.0015 0.0015 0.0015 0.0014 0.0014

Columns 330 through 334

0.0014 0.0014 0.0014 0.0014 0.0014

CFLowDates =

Columns 1 through 6

731323 731337 731368 731398 731429 731460

Columns 7 through 12

731490 731521 731551 731582 731613 731641

Columns 13 through 18

731672 731702 731733 731763 731794 731825

Columns 19 through 24

731855 731886 731916 731947 731978 732007

Columns 25 through 30

732038 732068 732099 732129 732160 732191

# mbscfamounts

---

Columns 31 through 36

732221	732252	732282	732313	732344	732372
--------	--------	--------	--------	--------	--------

Columns 37 through 42

732403	732433	732464	732494	732525	732556
--------	--------	--------	--------	--------	--------

Columns 43 through 48

732586	732617	732647	732678	732709	732737
--------	--------	--------	--------	--------	--------

Columns 49 through 54

732768	732798	732829	732859	732890	732921
--------	--------	--------	--------	--------	--------

Columns 55 through 60

732951	732982	733012	733043	733074	733102
--------	--------	--------	--------	--------	--------

Columns 61 through 66

733133	733163	733194	733224	733255	733286
--------	--------	--------	--------	--------	--------

Columns 67 through 72

733316	733347	733377	733408	733439	733468
--------	--------	--------	--------	--------	--------

Columns 73 through 78

733499	733529	733560	733590	733621	733652
--------	--------	--------	--------	--------	--------

Columns 79 through 84

733682	733713	733743	733774	733805	733833
--------	--------	--------	--------	--------	--------

Columns 85 through 90

733864	733894	733925	733955	733986	734017
--------	--------	--------	--------	--------	--------

Columns 91 through 96

734047	734078	734108	734139	734170	734198
--------	--------	--------	--------	--------	--------

Columns 97 through 102

734229	734259	734290	734320	734351	734382
--------	--------	--------	--------	--------	--------

Columns 103 through 108

734412	734443	734473	734504	734535	734563
--------	--------	--------	--------	--------	--------

Columns 109 through 114

734594	734624	734655	734685	734716	734747
--------	--------	--------	--------	--------	--------

Columns 115 through 120

734777	734808	734838	734869	734900	734929
--------	--------	--------	--------	--------	--------

Columns 121 through 126

734960	734990	735021	735051	735082	735113
--------	--------	--------	--------	--------	--------

Columns 127 through 132

735143	735174	735204	735235	735266	735294
--------	--------	--------	--------	--------	--------

Columns 133 through 138

735325	735355	735386	735416	735447	735478
--------	--------	--------	--------	--------	--------

Columns 139 through 144

# mbscfamounts

---

735508	735539	735569	735600	735631	735659
Columns 145 through 150					
735690	735720	735751	735781	735812	735843
Columns 151 through 156					
735873	735904	735934	735965	735996	736024
Columns 157 through 162					
736055	736085	736116	736146	736177	736208
Columns 163 through 168					
736238	736269	736299	736330	736361	736390
Columns 169 through 174					
736421	736451	736482	736512	736543	736574
Columns 175 through 180					
736604	736635	736665	736696	736727	736755
Columns 181 through 186					
736786	736816	736847	736877	736908	736939
Columns 187 through 192					
736969	737000	737030	737061	737092	737120
Columns 193 through 198					



737151	737181	737212	737242	737273	737304
Columns 199 through 204					
737334	737365	737395	737426	737457	737485
Columns 205 through 210					
737516	737546	737577	737607	737638	737669
Columns 211 through 216					
737699	737730	737760	737791	737822	737851
Columns 217 through 222					
737882	737912	737943	737973	738004	738035
Columns 223 through 228					
738065	738096	738126	738157	738188	738216
Columns 229 through 234					
738247	738277	738308	738338	738369	738400
Columns 235 through 240					
738430	738461	738491	738522	738553	738581
Columns 241 through 246					
738612	738642	738673	738703	738734	738765
Columns 247 through 252					
738795	738826	738856	738887	738918	738946

# mbscfamounts

---

Columns 253 through 258

738977	739007	739038	739068	739099	739130
--------	--------	--------	--------	--------	--------

Columns 259 through 264

739160	739191	739221	739252	739283	739312
--------	--------	--------	--------	--------	--------

Columns 265 through 270

739343	739373	739404	739434	739465	739496
--------	--------	--------	--------	--------	--------

Columns 271 through 276

739526	739557	739587	739618	739649	739677
--------	--------	--------	--------	--------	--------

Columns 277 through 282

739708	739738	739769	739799	739830	739861
--------	--------	--------	--------	--------	--------

Columns 283 through 288

739891	739922	739952	739983	740014	740042
--------	--------	--------	--------	--------	--------

Columns 289 through 294

740073	740103	740134	740164	740195	740226
--------	--------	--------	--------	--------	--------

Columns 295 through 300

740256	740287	740317	740348	740379	740407
--------	--------	--------	--------	--------	--------

Columns 301 through 306

740438	740468	740499	740529	740560	740591
--------	--------	--------	--------	--------	--------

Columns 307 through 312

740621 740652 740682 740713 740744 740773

Columns 313 through 318

740804 740834 740865 740895 740926 740957

Columns 319 through 324

740987 741018 741048 741079 741110 741138

Columns 325 through 330

741169 741199 741230 741260 741291 741322

Columns 331 through 334

741352 741383 741413 741444

TFactors =

Columns 1 through 7

0 0.9333 1.9333 2.9333 3.9333 4.9333 5.9333

Columns 8 through 14

6.9333 7.9333 8.9333 9.9333 10.9333 11.9333 12.9333

Columns 15 through 21

13.9333 14.9333 15.9333 16.9333 17.9333 18.9333 19.9333

Columns 22 through 28

# mbscfamounts

---

20.9333 21.9333 22.9333 23.9333 24.9333 25.9333 26.9333

Columns 29 through 35

27.9333 28.9333 29.9333 30.9333 31.9333 32.9333 33.9333

Columns 36 through 42

34.9333 35.9333 36.9333 37.9333 38.9333 39.9333 40.9333

Columns 43 through 49

41.9333 42.9333 43.9333 44.9333 45.9333 46.9333 47.9333

Columns 50 through 56

48.9333 49.9333 50.9333 51.9333 52.9333 53.9333 54.9333

Columns 57 through 63

55.9333 56.9333 57.9333 58.9333 59.9333 60.9333 61.9333

Columns 64 through 70

62.9333 63.9333 64.9333 65.9333 66.9333 67.9333 68.9333

Columns 71 through 77

69.9333 70.9333 71.9333 72.9333 73.9333 74.9333 75.9333

Columns 78 through 84

76.9333 77.9333 78.9333 79.9333 80.9333 81.9333 82.9333

Columns 85 through 91

83.9333 84.9333 85.9333 86.9333 87.9333 88.9333 89.9333

Columns 92 through 98

90.9333 91.9333 92.9333 93.9333 94.9333 95.9333 96.9333

Columns 99 through 105

97.9333 98.9333 99.9333 100.9333 101.9333 102.9333 103.9333

Columns 106 through 112

104.9333 105.9333 106.9333 107.9333 108.9333 109.9333 110.9333

Columns 113 through 119

111.9333 112.9333 113.9333 114.9333 115.9333 116.9333 117.9333

Columns 120 through 126

118.9333 119.9333 120.9333 121.9333 122.9333 123.9333 124.9333

Columns 127 through 133

125.9333 126.9333 127.9333 128.9333 129.9333 130.9333 131.9333

Columns 134 through 140

132.9333 133.9333 134.9333 135.9333 136.9333 137.9333 138.9333

Columns 141 through 147

139.9333 140.9333 141.9333 142.9333 143.9333 144.9333 145.9333

Columns 148 through 154

146.9333 147.9333 148.9333 149.9333 150.9333 151.9333 152.9333

# mbscfamounts

---

Columns 155 through 161

153.9333 154.9333 155.9333 156.9333 157.9333 158.9333 159.9333

Columns 162 through 168

160.9333 161.9333 162.9333 163.9333 164.9333 165.9333 166.9333

Columns 169 through 175

167.9333 168.9333 169.9333 170.9333 171.9333 172.9333 173.9333

Columns 176 through 182

174.9333 175.9333 176.9333 177.9333 178.9333 179.9333 180.9333

Columns 183 through 189

181.9333 182.9333 183.9333 184.9333 185.9333 186.9333 187.9333

Columns 190 through 196

188.9333 189.9333 190.9333 191.9333 192.9333 193.9333 194.9333

Columns 197 through 203

195.9333 196.9333 197.9333 198.9333 199.9333 200.9333 201.9333

Columns 204 through 210

202.9333 203.9333 204.9333 205.9333 206.9333 207.9333 208.9333

Columns 211 through 217

209.9333 210.9333 211.9333 212.9333 213.9333 214.9333 215.9333

Columns 218 through 224

216.9333 217.9333 218.9333 219.9333 220.9333 221.9333 222.9333

Columns 225 through 231

223.9333 224.9333 225.9333 226.9333 227.9333 228.9333 229.9333

Columns 232 through 238

230.9333 231.9333 232.9333 233.9333 234.9333 235.9333 236.9333

Columns 239 through 245

237.9333 238.9333 239.9333 240.9333 241.9333 242.9333 243.9333

Columns 246 through 252

244.9333 245.9333 246.9333 247.9333 248.9333 249.9333 250.9333

Columns 253 through 259

251.9333 252.9333 253.9333 254.9333 255.9333 256.9333 257.9333

Columns 260 through 266

258.9333 259.9333 260.9333 261.9333 262.9333 263.9333 264.9333

Columns 267 through 273

265.9333 266.9333 267.9333 268.9333 269.9333 270.9333 271.9333

Columns 274 through 280

272.9333 273.9333 274.9333 275.9333 276.9333 277.9333 278.9333

Columns 281 through 287

# mbscfamounts

---

279.9333 280.9333 281.9333 282.9333 283.9333 284.9333 285.9333

Columns 288 through 294

286.9333 287.9333 288.9333 289.9333 290.9333 291.9333 292.9333

Columns 295 through 301

293.9333 294.9333 295.9333 296.9333 297.9333 298.9333 299.9333

Columns 302 through 308

300.9333 301.9333 302.9333 303.9333 304.9333 305.9333 306.9333

Columns 309 through 315

307.9333 308.9333 309.9333 310.9333 311.9333 312.9333 313.9333

Columns 316 through 322

314.9333 315.9333 316.9333 317.9333 318.9333 319.9333 320.9333

Columns 323 through 329

321.9333 322.9333 323.9333 324.9333 325.9333 326.9333 327.9333

Columns 330 through 334

328.9333 329.9333 330.9333 331.9333 332.9333

Factors =

Columns 1 through 7

1.0000 0.9944 0.9887 0.9828 0.9769 0.9711 0.9653



Columns 8 through 14

0.9595 0.9538 0.9481 0.9424 0.9368 0.9311 0.9255

Columns 15 through 21

0.9199 0.9144 0.9089 0.9034 0.8979 0.8925 0.8871

Columns 22 through 28

0.8817 0.8763 0.8710 0.8657 0.8604 0.8552 0.8499

Columns 29 through 35

0.8447 0.8396 0.8344 0.8293 0.8242 0.8191 0.8140

Columns 36 through 42

0.8090 0.8040 0.7990 0.7941 0.7892 0.7842 0.7794

Columns 43 through 49

0.7745 0.7697 0.7649 0.7601 0.7553 0.7506 0.7458

Columns 50 through 56

0.7411 0.7365 0.7318 0.7272 0.7226 0.7180 0.7134

Columns 57 through 63

0.7089 0.7044 0.6999 0.6954 0.6910 0.6865 0.6821

Columns 64 through 70

0.6777 0.6734 0.6690 0.6647 0.6604 0.6561 0.6519

Columns 71 through 77

# mbscfamounts

---

0.6476 0.6434 0.6392 0.6350 0.6309 0.6267 0.6226

Columns 78 through 84

0.6185 0.6144 0.6104 0.6063 0.6023 0.5983 0.5943

Columns 85 through 91

0.5903 0.5864 0.5825 0.5785 0.5747 0.5708 0.5669

Columns 92 through 98

0.5631 0.5593 0.5555 0.5517 0.5479 0.5442 0.5405

Columns 99 through 105

0.5368 0.5331 0.5294 0.5257 0.5221 0.5185 0.5149

Columns 106 through 112

0.5113 0.5077 0.5042 0.5006 0.4971 0.4936 0.4901

Columns 113 through 119

0.4866 0.4832 0.4797 0.4763 0.4729 0.4695 0.4661

Columns 120 through 126

0.4628 0.4594 0.4561 0.4528 0.4495 0.4462 0.4430

Columns 127 through 133

0.4397 0.4365 0.4333 0.4301 0.4269 0.4237 0.4205

Columns 134 through 140

0.4174	0.4143	0.4111	0.4080	0.4049	0.4019	0.3988
Columns 141 through 147						
0.3958	0.3927	0.3897	0.3867	0.3837	0.3808	0.3778
Columns 148 through 154						
0.3748	0.3719	0.3690	0.3661	0.3632	0.3603	0.3574
Columns 155 through 161						
0.3546	0.3517	0.3489	0.3461	0.3433	0.3405	0.3377
Columns 162 through 168						
0.3350	0.3322	0.3295	0.3267	0.3240	0.3213	0.3186
Columns 169 through 175						
0.3160	0.3133	0.3106	0.3080	0.3054	0.3027	0.3001
Columns 176 through 182						
0.2975	0.2950	0.2924	0.2898	0.2873	0.2847	0.2822
Columns 183 through 189						
0.2797	0.2772	0.2747	0.2722	0.2698	0.2673	0.2649
Columns 190 through 196						
0.2624	0.2600	0.2576	0.2552	0.2528	0.2504	0.2480
Columns 197 through 203						
0.2457	0.2433	0.2410	0.2386	0.2363	0.2340	0.2317

# mbscfamounts

---

Columns 204 through 210

0.2294 0.2271 0.2249 0.2226 0.2204 0.2181 0.2159

Columns 211 through 217

0.2137 0.2115 0.2093 0.2071 0.2049 0.2027 0.2005

Columns 218 through 224

0.1984 0.1962 0.1941 0.1920 0.1899 0.1877 0.1856

Columns 225 through 231

0.1836 0.1815 0.1794 0.1773 0.1753 0.1732 0.1712

Columns 232 through 238

0.1692 0.1671 0.1651 0.1631 0.1611 0.1591 0.1572

Columns 239 through 245

0.1552 0.1532 0.1513 0.1493 0.1474 0.1455 0.1436

Columns 246 through 252

0.1416 0.1397 0.1378 0.1359 0.1341 0.1322 0.1303

Columns 253 through 259

0.1285 0.1266 0.1248 0.1229 0.1211 0.1193 0.1175

Columns 260 through 266

0.1157 0.1139 0.1121 0.1103 0.1085 0.1068 0.1050

Columns 267 through 273

0.1032	0.1015	0.0998	0.0980	0.0963	0.0946	0.0929
--------	--------	--------	--------	--------	--------	--------

Columns 274 through 280

0.0912	0.0895	0.0878	0.0861	0.0844	0.0827	0.0811
--------	--------	--------	--------	--------	--------	--------

Columns 281 through 287

0.0794	0.0778	0.0761	0.0745	0.0728	0.0712	0.0696
--------	--------	--------	--------	--------	--------	--------

Columns 288 through 294

0.0680	0.0664	0.0648	0.0632	0.0616	0.0600	0.0584
--------	--------	--------	--------	--------	--------	--------

Columns 295 through 301

0.0568	0.0553	0.0537	0.0522	0.0506	0.0491	0.0476
--------	--------	--------	--------	--------	--------	--------

Columns 302 through 308

0.0460	0.0445	0.0430	0.0415	0.0400	0.0385	0.0370
--------	--------	--------	--------	--------	--------	--------

Columns 309 through 315

0.0355	0.0340	0.0325	0.0311	0.0296	0.0281	0.0267
--------	--------	--------	--------	--------	--------	--------

Columns 316 through 322

0.0252	0.0238	0.0223	0.0209	0.0195	0.0180	0.0166
--------	--------	--------	--------	--------	--------	--------

Columns 323 through 329

0.0152	0.0138	0.0124	0.0110	0.0096	0.0082	0.0068
--------	--------	--------	--------	--------	--------	--------

Columns 330 through 334

# mbscfamounts

---

```
0.0055    0.0041    0.0027    0.0014    0
```

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

## Compute Cash Flow Amounts and Dates, Time Factors, and Mortgage Factors for a Mortgage Portfolio

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

Define characteristics for a mortgage portfolio.

```
Settle    = datenum(['13-Jan-2000'; '17-Apr-2002'; '17-May-2002']);
Maturity  = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = [0.075; 0.07875; 0.0775];
Delay     = 14;
PrepaySpeed = 100;
```

Use `mbscfamounts` to evaluate the mortgage.

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

```
CFlowAmounts =
```

```
Columns 1 through 10
```

```
-0.0033    0.0118    0.0120    0.0121    0.0120    0.0119    0.0119    0.0118    0.0117    0.0117
```

```
Columns 11 through 20
```

```
0.0116    0.0115    0.0115    0.0114    0.0114    0.0113    0.0112    0.0112    0.0111    0.0110
```

```
Columns 21 through 30
```

0.0110 0.0109 0.0109 0.0108 0.0107 0.0107 0.0106 0.0106 0.0105 0.0105

Columns 31 through 40

0.0104 0.0103 0.0103 0.0102 0.0102 0.0101 0.0101 0.0100 0.0099 0.0099

Columns 41 through 50

0.0098 0.0098 0.0097 0.0097 0.0096 0.0096 0.0095 0.0095 0.0094 0.0094

Columns 51 through 60

0.0093 0.0093 0.0092 0.0092 0.0091 0.0090 0.0090 0.0089 0.0089 0.0088

Columns 61 through 70

0.0088 0.0087 0.0087 0.0087 0.0086 0.0086 0.0085 0.0085 0.0084 0.0084

Columns 71 through 80

0.0083 0.0083 0.0082 0.0082 0.0081 0.0081 0.0080 0.0080 0.0079 0.0079

Columns 81 through 90

0.0079 0.0078 0.0078 0.0077 0.0077 0.0076 0.0076 0.0075 0.0075 0.0075

Columns 91 through 100

0.0074 0.0074 0.0073 0.0073 0.0073 0.0072 0.0072 0.0071 0.0071 0.0070

Columns 101 through 110

0.0070 0.0070 0.0069 0.0069 0.0068 0.0068 0.0068 0.0067 0.0067 0.0066

Columns 111 through 120

0.0066 0.0066 0.0065 0.0065 0.0065 0.0064 0.0064 0.0063 0.0063 0.0063

# mbscfamounts

---

Columns 121 through 130

0.0062	0.0062	0.0062	0.0061	0.0061	0.0061	0.0060	0.0060	0.0059	0.0059
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 131 through 140

0.0059	0.0058	0.0058	0.0058	0.0057	0.0057	0.0057	0.0056	0.0056	0.0056
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 141 through 150

0.0055	0.0055	0.0055	0.0054	0.0054	0.0054	0.0053	0.0053	0.0053	0.0052
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 151 through 160

0.0052	0.0052	0.0052	0.0051	0.0051	0.0051	0.0050	0.0050	0.0050	0.0049
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 161 through 170

0.0049	0.0049	0.0048	0.0048	0.0048	0.0048	0.0047	0.0047	0.0047	0.0046
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 171 through 180

0.0046	0.0046	0.0046	0.0045	0.0045	0.0045	0.0044	0.0044	0.0044	0.0044
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 181 through 190

0.0043	0.0043	0.0043	0.0043	0.0042	0.0042	0.0042	0.0041	0.0041	0.0041
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 191 through 200

0.0041	0.0040	0.0040	0.0040	0.0040	0.0039	0.0039	0.0039	0.0039	0.0038
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 201 through 210

0.0038	0.0038	0.0038	0.0037	0.0037	0.0037	0.0037	0.0036	0.0036	0.0036
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------



Columns 211 through 220

0.0036 0.0035 0.0035 0.0035 0.0035 0.0035 0.0034 0.0034 0.0034 0.0034

Columns 221 through 230

0.0033 0.0033 0.0033 0.0033 0.0033 0.0032 0.0032 0.0032 0.0032 0.0031

Columns 231 through 240

0.0031 0.0031 0.0031 0.0031 0.0030 0.0030 0.0030 0.0030 0.0030 0.0029

Columns 241 through 250

0.0029 0.0029 0.0029 0.0029 0.0028 0.0028 0.0028 0.0028 0.0028 0.0027

Columns 251 through 260

0.0027 0.0027 0.0027 0.0027 0.0026 0.0026 0.0026 0.0026 0.0026 0.0025

Columns 261 through 270

0.0025 0.0025 0.0025 0.0025 0.0024 0.0024 0.0024 0.0024 0.0024 0.0024

Columns 271 through 280

0.0023 0.0023 0.0023 0.0023 0.0023 0.0023 0.0022 0.0022 0.0022 0.0022

Columns 281 through 290

0.0022 0.0021 0.0021 0.0021 0.0021 0.0021 0.0021 0.0020 0.0020 0.0020

Columns 291 through 300

0.0020 0.0020 0.0020 0.0020 0.0019 0.0019 0.0019 0.0019 0.0019 0.0019

Columns 301 through 310

# mbscfamounts

---

0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0017 0.0017 0.0017 0.0017

Columns 311 through 320

0.0017 0.0017 0.0017 0.0016 0.0016 0.0016 0.0016 0.0016 0.0016 0.0016

Columns 321 through 330

0.0015 0.0015 0.0015 0.0015 0.0015 0.0015 0.0015 0.0014 0.0014 0.0014

Columns 331 through 334

0.0014 0.0014 0.0014 0.0014

CFLowDates =

Columns 1 through 8

731323 731337 731368 731398 731429 731460 731490 731521

Columns 9 through 16

731551 731582 731613 731641 731672 731702 731733 731763

Columns 17 through 24

731794 731825 731855 731886 731916 731947 731978 732007

Columns 25 through 32

732038 732068 732099 732129 732160 732191 732221 732252

Columns 33 through 40

732282 732313 732344 732372 732403 732433 732464 732494

Columns 41 through 48

732525	732556	732586	732617	732647	732678	732709	732737
--------	--------	--------	--------	--------	--------	--------	--------

Columns 49 through 56

732768	732798	732829	732859	732890	732921	732951	732982
--------	--------	--------	--------	--------	--------	--------	--------

Columns 57 through 64

733012	733043	733074	733102	733133	733163	733194	733224
--------	--------	--------	--------	--------	--------	--------	--------

Columns 65 through 72

733255	733286	733316	733347	733377	733408	733439	733468
--------	--------	--------	--------	--------	--------	--------	--------

Columns 73 through 80

733499	733529	733560	733590	733621	733652	733682	733713
--------	--------	--------	--------	--------	--------	--------	--------

Columns 81 through 88

733743	733774	733805	733833	733864	733894	733925	733955
--------	--------	--------	--------	--------	--------	--------	--------

Columns 89 through 96

733986	734017	734047	734078	734108	734139	734170	734198
--------	--------	--------	--------	--------	--------	--------	--------

Columns 97 through 104

734229	734259	734290	734320	734351	734382	734412	734443
--------	--------	--------	--------	--------	--------	--------	--------

Columns 105 through 112

734473	734504	734535	734563	734594	734624	734655	734685
--------	--------	--------	--------	--------	--------	--------	--------

# mbscfamounts

---

Columns 113 through 120

734716	734747	734777	734808	734838	734869	734900	734929
--------	--------	--------	--------	--------	--------	--------	--------

Columns 121 through 128

734960	734990	735021	735051	735082	735113	735143	735174
--------	--------	--------	--------	--------	--------	--------	--------

Columns 129 through 136

735204	735235	735266	735294	735325	735355	735386	735416
--------	--------	--------	--------	--------	--------	--------	--------

Columns 137 through 144

735447	735478	735508	735539	735569	735600	735631	735659
--------	--------	--------	--------	--------	--------	--------	--------

Columns 145 through 152

735690	735720	735751	735781	735812	735843	735873	735904
--------	--------	--------	--------	--------	--------	--------	--------

Columns 153 through 160

735934	735965	735996	736024	736055	736085	736116	736146
--------	--------	--------	--------	--------	--------	--------	--------

Columns 161 through 168

736177	736208	736238	736269	736299	736330	736361	736390
--------	--------	--------	--------	--------	--------	--------	--------

Columns 169 through 176

736421	736451	736482	736512	736543	736574	736604	736635
--------	--------	--------	--------	--------	--------	--------	--------

Columns 177 through 184

736665	736696	736727	736755	736786	736816	736847	736877
--------	--------	--------	--------	--------	--------	--------	--------

Columns 185 through 192

736908	736939	736969	737000	737030	737061	737092	737120
Columns 193 through 200							
737151	737181	737212	737242	737273	737304	737334	737365
Columns 201 through 208							
737395	737426	737457	737485	737516	737546	737577	737607
Columns 209 through 216							
737638	737669	737699	737730	737760	737791	737822	737851
Columns 217 through 224							
737882	737912	737943	737973	738004	738035	738065	738096
Columns 225 through 232							
738126	738157	738188	738216	738247	738277	738308	738338
Columns 233 through 240							
738369	738400	738430	738461	738491	738522	738553	738581
Columns 241 through 248							
738612	738642	738673	738703	738734	738765	738795	738826
Columns 249 through 256							
738856	738887	738918	738946	738977	739007	739038	739068
Columns 257 through 264							

# mbscfamounts

---

739099	739130	739160	739191	739221	739252	739283	739312
Columns 265 through 272							
739343	739373	739404	739434	739465	739496	739526	739557
Columns 273 through 280							
739587	739618	739649	739677	739708	739738	739769	739799
Columns 281 through 288							
739830	739861	739891	739922	739952	739983	740014	740042
Columns 289 through 296							
740073	740103	740134	740164	740195	740226	740256	740287
Columns 297 through 304							
740317	740348	740379	740407	740438	740468	740499	740529
Columns 305 through 312							
740560	740591	740621	740652	740682	740713	740744	740773
Columns 313 through 320							
740804	740834	740865	740895	740926	740957	740987	741018
Columns 321 through 328							
741048	741079	741110	741138	741169	741199	741230	741260
Columns 329 through 334							
741291	741322	741352	741383	741413	741444		

TFactors =

Columns 1 through 10

0 0.9333 1.9333 2.9333 3.9333 4.9333 5.9333 6.9333 7.9333 8.9333

Columns 11 through 20

9.9333 10.9333 11.9333 12.9333 13.9333 14.9333 15.9333 16.9333 17.9333 18.9333

Columns 21 through 30

19.9333 20.9333 21.9333 22.9333 23.9333 24.9333 25.9333 26.9333 27.9333 28.9333

Columns 31 through 40

29.9333 30.9333 31.9333 32.9333 33.9333 34.9333 35.9333 36.9333 37.9333 38.9333

Columns 41 through 50

39.9333 40.9333 41.9333 42.9333 43.9333 44.9333 45.9333 46.9333 47.9333 48.9333

Columns 51 through 60

49.9333 50.9333 51.9333 52.9333 53.9333 54.9333 55.9333 56.9333 57.9333 58.9333

Columns 61 through 70

59.9333 60.9333 61.9333 62.9333 63.9333 64.9333 65.9333 66.9333 67.9333 68.9333

Columns 71 through 80

69.9333 70.9333 71.9333 72.9333 73.9333 74.9333 75.9333 76.9333 77.9333 78.9333

Columns 81 through 90

# mbscfamounts

---

79.9333 80.9333 81.9333 82.9333 83.9333 84.9333 85.9333 86.9333 87.9333 88.9333

Columns 91 through 100

89.9333 90.9333 91.9333 92.9333 93.9333 94.9333 95.9333 96.9333 97.9333 98.9333

Columns 101 through 110

99.9333 100.9333 101.9333 102.9333 103.9333 104.9333 105.9333 106.9333 107.9333 108.9333

Columns 111 through 120

109.9333 110.9333 111.9333 112.9333 113.9333 114.9333 115.9333 116.9333 117.9333 118.9333

Columns 121 through 130

119.9333 120.9333 121.9333 122.9333 123.9333 124.9333 125.9333 126.9333 127.9333 128.9333

Columns 131 through 140

129.9333 130.9333 131.9333 132.9333 133.9333 134.9333 135.9333 136.9333 137.9333 138.9333

Columns 141 through 150

139.9333 140.9333 141.9333 142.9333 143.9333 144.9333 145.9333 146.9333 147.9333 148.9333

Columns 151 through 160

149.9333 150.9333 151.9333 152.9333 153.9333 154.9333 155.9333 156.9333 157.9333 158.9333

Columns 161 through 170

159.9333 160.9333 161.9333 162.9333 163.9333 164.9333 165.9333 166.9333 167.9333 168.9333

Columns 171 through 180



169.9333 170.9333 171.9333 172.9333 173.9333 174.9333 175.9333 176.9333 177.9333 178.9333

Columns 181 through 190

179.9333 180.9333 181.9333 182.9333 183.9333 184.9333 185.9333 186.9333 187.9333 188.9333

Columns 191 through 200

189.9333 190.9333 191.9333 192.9333 193.9333 194.9333 195.9333 196.9333 197.9333 198.9333

Columns 201 through 210

199.9333 200.9333 201.9333 202.9333 203.9333 204.9333 205.9333 206.9333 207.9333 208.9333

Columns 211 through 220

209.9333 210.9333 211.9333 212.9333 213.9333 214.9333 215.9333 216.9333 217.9333 218.9333

Columns 221 through 230

219.9333 220.9333 221.9333 222.9333 223.9333 224.9333 225.9333 226.9333 227.9333 228.9333

Columns 231 through 240

229.9333 230.9333 231.9333 232.9333 233.9333 234.9333 235.9333 236.9333 237.9333 238.9333

Columns 241 through 250

239.9333 240.9333 241.9333 242.9333 243.9333 244.9333 245.9333 246.9333 247.9333 248.9333

Columns 251 through 260

249.9333 250.9333 251.9333 252.9333 253.9333 254.9333 255.9333 256.9333 257.9333 258.9333

Columns 261 through 270

259.9333 260.9333 261.9333 262.9333 263.9333 264.9333 265.9333 266.9333 267.9333 268.9333

# mbscfamounts

---

Columns 271 through 280

269.9333 270.9333 271.9333 272.9333 273.9333 274.9333 275.9333 276.9333 277.9333 278.9333

Columns 281 through 290

279.9333 280.9333 281.9333 282.9333 283.9333 284.9333 285.9333 286.9333 287.9333 288.9333

Columns 291 through 300

289.9333 290.9333 291.9333 292.9333 293.9333 294.9333 295.9333 296.9333 297.9333 298.9333

Columns 301 through 310

299.9333 300.9333 301.9333 302.9333 303.9333 304.9333 305.9333 306.9333 307.9333 308.9333

Columns 311 through 320

309.9333 310.9333 311.9333 312.9333 313.9333 314.9333 315.9333 316.9333 317.9333 318.9333

Columns 321 through 330

319.9333 320.9333 321.9333 322.9333 323.9333 324.9333 325.9333 326.9333 327.9333 328.9333

Columns 331 through 334

329.9333 330.9333 331.9333 332.9333

Factors =

Columns 1 through 10

1.0000 0.9944 0.9887 0.9828 0.9769 0.9711 0.9653 0.9595 0.9538 0.9481

Columns 11 through 20

0.9424 0.9368 0.9311 0.9255 0.9199 0.9144 0.9089 0.9034 0.8979 0.8925

Columns 21 through 30

0.8871 0.8817 0.8763 0.8710 0.8657 0.8604 0.8552 0.8499 0.8447 0.8396

Columns 31 through 40

0.8344 0.8293 0.8242 0.8191 0.8140 0.8090 0.8040 0.7990 0.7941 0.7892

Columns 41 through 50

0.7842 0.7794 0.7745 0.7697 0.7649 0.7601 0.7553 0.7506 0.7458 0.7411

Columns 51 through 60

0.7365 0.7318 0.7272 0.7226 0.7180 0.7134 0.7089 0.7044 0.6999 0.6954

Columns 61 through 70

0.6910 0.6865 0.6821 0.6777 0.6734 0.6690 0.6647 0.6604 0.6561 0.6519

Columns 71 through 80

0.6476 0.6434 0.6392 0.6350 0.6309 0.6267 0.6226 0.6185 0.6144 0.6104

Columns 81 through 90

0.6063 0.6023 0.5983 0.5943 0.5903 0.5864 0.5825 0.5785 0.5747 0.5708

Columns 91 through 100

0.5669 0.5631 0.5593 0.5555 0.5517 0.5479 0.5442 0.5405 0.5368 0.5331

Columns 101 through 110

# mbscfamounts

---

0.5294	0.5257	0.5221	0.5185	0.5149	0.5113	0.5077	0.5042	0.5006	0.4971
Columns 111 through 120									
0.4936	0.4901	0.4866	0.4832	0.4797	0.4763	0.4729	0.4695	0.4661	0.4628
Columns 121 through 130									
0.4594	0.4561	0.4528	0.4495	0.4462	0.4430	0.4397	0.4365	0.4333	0.4301
Columns 131 through 140									
0.4269	0.4237	0.4205	0.4174	0.4143	0.4111	0.4080	0.4049	0.4019	0.3988
Columns 141 through 150									
0.3958	0.3927	0.3897	0.3867	0.3837	0.3808	0.3778	0.3748	0.3719	0.3690
Columns 151 through 160									
0.3661	0.3632	0.3603	0.3574	0.3546	0.3517	0.3489	0.3461	0.3433	0.3405
Columns 161 through 170									
0.3377	0.3350	0.3322	0.3295	0.3267	0.3240	0.3213	0.3186	0.3160	0.3133
Columns 171 through 180									
0.3106	0.3080	0.3054	0.3027	0.3001	0.2975	0.2950	0.2924	0.2898	0.2873
Columns 181 through 190									
0.2847	0.2822	0.2797	0.2772	0.2747	0.2722	0.2698	0.2673	0.2649	0.2624
Columns 191 through 200									
0.2600	0.2576	0.2552	0.2528	0.2504	0.2480	0.2457	0.2433	0.2410	0.2386

Columns 201 through 210

0.2363 0.2340 0.2317 0.2294 0.2271 0.2249 0.2226 0.2204 0.2181 0.2159

Columns 211 through 220

0.2137 0.2115 0.2093 0.2071 0.2049 0.2027 0.2005 0.1984 0.1962 0.1941

Columns 221 through 230

0.1920 0.1899 0.1877 0.1856 0.1836 0.1815 0.1794 0.1773 0.1753 0.1732

Columns 231 through 240

0.1712 0.1692 0.1671 0.1651 0.1631 0.1611 0.1591 0.1572 0.1552 0.1532

Columns 241 through 250

0.1513 0.1493 0.1474 0.1455 0.1436 0.1416 0.1397 0.1378 0.1359 0.1341

Columns 251 through 260

0.1322 0.1303 0.1285 0.1266 0.1248 0.1229 0.1211 0.1193 0.1175 0.1157

Columns 261 through 270

0.1139 0.1121 0.1103 0.1085 0.1068 0.1050 0.1032 0.1015 0.0998 0.0980

Columns 271 through 280

0.0963 0.0946 0.0929 0.0912 0.0895 0.0878 0.0861 0.0844 0.0827 0.0811

Columns 281 through 290

0.0794 0.0778 0.0761 0.0745 0.0728 0.0712 0.0696 0.0680 0.0664 0.0648

Columns 291 through 300

0.0632	0.0616	0.0600	0.0584	0.0568	0.0553	0.0537	0.0522	0.0506	0.0491
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 301 through 310

0.0476	0.0460	0.0445	0.0430	0.0415	0.0400	0.0385	0.0370	0.0355	0.0340
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 311 through 320

0.0325	0.0311	0.0296	0.0281	0.0267	0.0252	0.0238	0.0223	0.0209	0.0195
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 321 through 330

0.0180	0.0166	0.0152	0.0138	0.0124	0.0110	0.0096	0.0082	0.0068	0.0055
--------	--------	--------	--------	--------	--------	--------	--------	--------	--------

Columns 331 through 334

0.0041	0.0027	0.0014	0
--------	--------	--------	---

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

## **Calculate Payment, Principal, Interest, and Prepayment for a Single Mortgage**

Given a mortgage with the following characteristics, compute payments, principal, interest, and prepayment.

Define the mortgage characteristics.

```
Settle      = datenum('17-April-2002');
Maturity    = datenum('1-Jan-2030');
IssueDate   = datenum('1-Jan-2000');
GrossRate   = 0.08125;
CouponRate  = 0.075;
Delay       = 14;
PrepaySpeed = 100;
```

Use mbscfamounts to evaluate the mortgage.

```
[Payment, Principal, Interest, Prepayment] = ...  
mbscfamounts(Settle, Maturity, IssueDate, GrossRate, ...  
CouponRate, Delay, PrepaySpeed)
```

Payment =

Columns 1 through 8

-0.0033	0.0118	0.0120	0.0121	0.0120	0.0119	0.0119	0.0118
---------	--------	--------	--------	--------	--------	--------	--------

Columns 9 through 16

0.0117	0.0117	0.0116	0.0115	0.0115	0.0114	0.0114	0.0113
--------	--------	--------	--------	--------	--------	--------	--------

Columns 17 through 24

0.0112	0.0112	0.0111	0.0110	0.0110	0.0109	0.0109	0.0108
--------	--------	--------	--------	--------	--------	--------	--------

Columns 25 through 32

0.0107	0.0107	0.0106	0.0106	0.0105	0.0105	0.0104	0.0103
--------	--------	--------	--------	--------	--------	--------	--------

Columns 33 through 40

0.0103	0.0102	0.0102	0.0101	0.0101	0.0100	0.0099	0.0099
--------	--------	--------	--------	--------	--------	--------	--------

Columns 41 through 48

0.0098	0.0098	0.0097	0.0097	0.0096	0.0096	0.0095	0.0095
--------	--------	--------	--------	--------	--------	--------	--------

Columns 49 through 56

# mbscfamounts

---

0.0094	0.0094	0.0093	0.0093	0.0092	0.0092	0.0091	0.0090
Columns 57 through 64							
0.0090	0.0089	0.0089	0.0088	0.0088	0.0087	0.0087	0.0087
Columns 65 through 72							
0.0086	0.0086	0.0085	0.0085	0.0084	0.0084	0.0083	0.0083
Columns 73 through 80							
0.0082	0.0082	0.0081	0.0081	0.0080	0.0080	0.0079	0.0079
Columns 81 through 88							
0.0079	0.0078	0.0078	0.0077	0.0077	0.0076	0.0076	0.0075
Columns 89 through 96							
0.0075	0.0075	0.0074	0.0074	0.0073	0.0073	0.0073	0.0072
Columns 97 through 104							
0.0072	0.0071	0.0071	0.0070	0.0070	0.0070	0.0069	0.0069
Columns 105 through 112							
0.0068	0.0068	0.0068	0.0067	0.0067	0.0066	0.0066	0.0066
Columns 113 through 120							
0.0065	0.0065	0.0065	0.0064	0.0064	0.0063	0.0063	0.0063
Columns 121 through 128							
0.0062	0.0062	0.0062	0.0061	0.0061	0.0061	0.0060	0.0060



Columns 129 through 136

0.0059	0.0059	0.0059	0.0058	0.0058	0.0058	0.0057	0.0057
--------	--------	--------	--------	--------	--------	--------	--------

Columns 137 through 144

0.0057	0.0056	0.0056	0.0056	0.0055	0.0055	0.0055	0.0054
--------	--------	--------	--------	--------	--------	--------	--------

Columns 145 through 152

0.0054	0.0054	0.0053	0.0053	0.0053	0.0052	0.0052	0.0052
--------	--------	--------	--------	--------	--------	--------	--------

Columns 153 through 160

0.0052	0.0051	0.0051	0.0051	0.0050	0.0050	0.0050	0.0049
--------	--------	--------	--------	--------	--------	--------	--------

Columns 161 through 168

0.0049	0.0049	0.0048	0.0048	0.0048	0.0048	0.0047	0.0047
--------	--------	--------	--------	--------	--------	--------	--------

Columns 169 through 176

0.0047	0.0046	0.0046	0.0046	0.0046	0.0045	0.0045	0.0045
--------	--------	--------	--------	--------	--------	--------	--------

Columns 177 through 184

0.0044	0.0044	0.0044	0.0044	0.0043	0.0043	0.0043	0.0043
--------	--------	--------	--------	--------	--------	--------	--------

Columns 185 through 192

0.0042	0.0042	0.0042	0.0041	0.0041	0.0041	0.0041	0.0040
--------	--------	--------	--------	--------	--------	--------	--------

Columns 193 through 200

0.0040	0.0040	0.0040	0.0039	0.0039	0.0039	0.0039	0.0038
--------	--------	--------	--------	--------	--------	--------	--------

# mbscfamounts

---

Columns 201 through 208

0.0038	0.0038	0.0038	0.0037	0.0037	0.0037	0.0037	0.0036
--------	--------	--------	--------	--------	--------	--------	--------

Columns 209 through 216

0.0036	0.0036	0.0036	0.0035	0.0035	0.0035	0.0035	0.0035
--------	--------	--------	--------	--------	--------	--------	--------

Columns 217 through 224

0.0034	0.0034	0.0034	0.0034	0.0033	0.0033	0.0033	0.0033
--------	--------	--------	--------	--------	--------	--------	--------

Columns 225 through 232

0.0033	0.0032	0.0032	0.0032	0.0032	0.0031	0.0031	0.0031
--------	--------	--------	--------	--------	--------	--------	--------

Columns 233 through 240

0.0031	0.0031	0.0030	0.0030	0.0030	0.0030	0.0030	0.0029
--------	--------	--------	--------	--------	--------	--------	--------

Columns 241 through 248

0.0029	0.0029	0.0029	0.0029	0.0028	0.0028	0.0028	0.0028
--------	--------	--------	--------	--------	--------	--------	--------

Columns 249 through 256

0.0028	0.0027	0.0027	0.0027	0.0027	0.0027	0.0026	0.0026
--------	--------	--------	--------	--------	--------	--------	--------

Columns 257 through 264

0.0026	0.0026	0.0026	0.0025	0.0025	0.0025	0.0025	0.0025
--------	--------	--------	--------	--------	--------	--------	--------

Columns 265 through 272

0.0024	0.0024	0.0024	0.0024	0.0024	0.0024	0.0023	0.0023
--------	--------	--------	--------	--------	--------	--------	--------

Columns 273 through 280

0.0023 0.0023 0.0023 0.0023 0.0022 0.0022 0.0022 0.0022

Columns 281 through 288

0.0022 0.0021 0.0021 0.0021 0.0021 0.0021 0.0021 0.0020

Columns 289 through 296

0.0020 0.0020 0.0020 0.0020 0.0020 0.0020 0.0019 0.0019

Columns 297 through 304

0.0019 0.0019 0.0019 0.0019 0.0018 0.0018 0.0018 0.0018

Columns 305 through 312

0.0018 0.0018 0.0017 0.0017 0.0017 0.0017 0.0017 0.0017

Columns 313 through 320

0.0017 0.0016 0.0016 0.0016 0.0016 0.0016 0.0016 0.0016

Columns 321 through 328

0.0015 0.0015 0.0015 0.0015 0.0015 0.0015 0.0015 0.0014

Columns 329 through 334

0.0014 0.0014 0.0014 0.0014 0.0014 0.0014

Principal =

Columns 1 through 6

731323 731337 731368 731398 731429 731460

# mbscfamounts

---

Columns 7 through 12

731490	731521	731551	731582	731613	731641
--------	--------	--------	--------	--------	--------

Columns 13 through 18

731672	731702	731733	731763	731794	731825
--------	--------	--------	--------	--------	--------

Columns 19 through 24

731855	731886	731916	731947	731978	732007
--------	--------	--------	--------	--------	--------

Columns 25 through 30

732038	732068	732099	732129	732160	732191
--------	--------	--------	--------	--------	--------

Columns 31 through 36

732221	732252	732282	732313	732344	732372
--------	--------	--------	--------	--------	--------

Columns 37 through 42

732403	732433	732464	732494	732525	732556
--------	--------	--------	--------	--------	--------

Columns 43 through 48

732586	732617	732647	732678	732709	732737
--------	--------	--------	--------	--------	--------

Columns 49 through 54

732768	732798	732829	732859	732890	732921
--------	--------	--------	--------	--------	--------

Columns 55 through 60

732951	732982	733012	733043	733074	733102
--------	--------	--------	--------	--------	--------

Columns 61 through 66

733133	733163	733194	733224	733255	733286
--------	--------	--------	--------	--------	--------

Columns 67 through 72

733316	733347	733377	733408	733439	733468
--------	--------	--------	--------	--------	--------

Columns 73 through 78

733499	733529	733560	733590	733621	733652
--------	--------	--------	--------	--------	--------

Columns 79 through 84

733682	733713	733743	733774	733805	733833
--------	--------	--------	--------	--------	--------

Columns 85 through 90

733864	733894	733925	733955	733986	734017
--------	--------	--------	--------	--------	--------

Columns 91 through 96

734047	734078	734108	734139	734170	734198
--------	--------	--------	--------	--------	--------

Columns 97 through 102

734229	734259	734290	734320	734351	734382
--------	--------	--------	--------	--------	--------

Columns 103 through 108

734412	734443	734473	734504	734535	734563
--------	--------	--------	--------	--------	--------

Columns 109 through 114

734594	734624	734655	734685	734716	734747
--------	--------	--------	--------	--------	--------

Columns 115 through 120

# mbscfamounts

---

734777	734808	734838	734869	734900	734929
Columns 121 through 126					
734960	734990	735021	735051	735082	735113
Columns 127 through 132					
735143	735174	735204	735235	735266	735294
Columns 133 through 138					
735325	735355	735386	735416	735447	735478
Columns 139 through 144					
735508	735539	735569	735600	735631	735659
Columns 145 through 150					
735690	735720	735751	735781	735812	735843
Columns 151 through 156					
735873	735904	735934	735965	735996	736024
Columns 157 through 162					
736055	736085	736116	736146	736177	736208
Columns 163 through 168					
736238	736269	736299	736330	736361	736390
Columns 169 through 174					

736421 736451 736482 736512 736543 736574

Columns 175 through 180

736604 736635 736665 736696 736727 736755

Columns 181 through 186

736786 736816 736847 736877 736908 736939

Columns 187 through 192

736969 737000 737030 737061 737092 737120

Columns 193 through 198

737151 737181 737212 737242 737273 737304

Columns 199 through 204

737334 737365 737395 737426 737457 737485

Columns 205 through 210

737516 737546 737577 737607 737638 737669

Columns 211 through 216

737699 737730 737760 737791 737822 737851

Columns 217 through 222

737882 737912 737943 737973 738004 738035

Columns 223 through 228

738065 738096 738126 738157 738188 738216

# mbscfamounts

---

Columns 229 through 234

738247	738277	738308	738338	738369	738400
--------	--------	--------	--------	--------	--------

Columns 235 through 240

738430	738461	738491	738522	738553	738581
--------	--------	--------	--------	--------	--------

Columns 241 through 246

738612	738642	738673	738703	738734	738765
--------	--------	--------	--------	--------	--------

Columns 247 through 252

738795	738826	738856	738887	738918	738946
--------	--------	--------	--------	--------	--------

Columns 253 through 258

738977	739007	739038	739068	739099	739130
--------	--------	--------	--------	--------	--------

Columns 259 through 264

739160	739191	739221	739252	739283	739312
--------	--------	--------	--------	--------	--------

Columns 265 through 270

739343	739373	739404	739434	739465	739496
--------	--------	--------	--------	--------	--------

Columns 271 through 276

739526	739557	739587	739618	739649	739677
--------	--------	--------	--------	--------	--------

Columns 277 through 282

739708	739738	739769	739799	739830	739861
--------	--------	--------	--------	--------	--------



Columns 283 through 288

739891 739922 739952 739983 740014 740042

Columns 289 through 294

740073 740103 740134 740164 740195 740226

Columns 295 through 300

740256 740287 740317 740348 740379 740407

Columns 301 through 306

740438 740468 740499 740529 740560 740591

Columns 307 through 312

740621 740652 740682 740713 740744 740773

Columns 313 through 318

740804 740834 740865 740895 740926 740957

Columns 319 through 324

740987 741018 741048 741079 741110 741138

Columns 325 through 330

741169 741199 741230 741260 741291 741322

Columns 331 through 334

741352 741383 741413 741444

# mbscfamounts

---

Interest =

Columns 1 through 8

0 0.9333 1.9333 2.9333 3.9333 4.9333 5.9333 6.9333

Columns 9 through 16

7.9333 8.9333 9.9333 10.9333 11.9333 12.9333 13.9333 14.9333

Columns 17 through 24

15.9333 16.9333 17.9333 18.9333 19.9333 20.9333 21.9333 22.9333

Columns 25 through 32

23.9333 24.9333 25.9333 26.9333 27.9333 28.9333 29.9333 30.9333

Columns 33 through 40

31.9333 32.9333 33.9333 34.9333 35.9333 36.9333 37.9333 38.9333

Columns 41 through 48

39.9333 40.9333 41.9333 42.9333 43.9333 44.9333 45.9333 46.9333

Columns 49 through 56

47.9333 48.9333 49.9333 50.9333 51.9333 52.9333 53.9333 54.9333

Columns 57 through 64

55.9333 56.9333 57.9333 58.9333 59.9333 60.9333 61.9333 62.9333

Columns 65 through 72

63.9333 64.9333 65.9333 66.9333 67.9333 68.9333 69.9333 70.9333

Columns 73 through 80

71.9333 72.9333 73.9333 74.9333 75.9333 76.9333 77.9333 78.9333

Columns 81 through 88

79.9333 80.9333 81.9333 82.9333 83.9333 84.9333 85.9333 86.9333

Columns 89 through 96

87.9333 88.9333 89.9333 90.9333 91.9333 92.9333 93.9333 94.9333

Columns 97 through 104

95.9333 96.9333 97.9333 98.9333 99.9333 100.9333 101.9333 102.9333

Columns 105 through 112

103.9333 104.9333 105.9333 106.9333 107.9333 108.9333 109.9333 110.9333

Columns 113 through 120

111.9333 112.9333 113.9333 114.9333 115.9333 116.9333 117.9333 118.9333

Columns 121 through 128

119.9333 120.9333 121.9333 122.9333 123.9333 124.9333 125.9333 126.9333

Columns 129 through 136

127.9333 128.9333 129.9333 130.9333 131.9333 132.9333 133.9333 134.9333

Columns 137 through 144

135.9333 136.9333 137.9333 138.9333 139.9333 140.9333 141.9333 142.9333

# mbscfamounts

---

Columns 145 through 152

143.9333 144.9333 145.9333 146.9333 147.9333 148.9333 149.9333 150.9333

Columns 153 through 160

151.9333 152.9333 153.9333 154.9333 155.9333 156.9333 157.9333 158.9333

Columns 161 through 168

159.9333 160.9333 161.9333 162.9333 163.9333 164.9333 165.9333 166.9333

Columns 169 through 176

167.9333 168.9333 169.9333 170.9333 171.9333 172.9333 173.9333 174.9333

Columns 177 through 184

175.9333 176.9333 177.9333 178.9333 179.9333 180.9333 181.9333 182.9333

Columns 185 through 192

183.9333 184.9333 185.9333 186.9333 187.9333 188.9333 189.9333 190.9333

Columns 193 through 200

191.9333 192.9333 193.9333 194.9333 195.9333 196.9333 197.9333 198.9333

Columns 201 through 208

199.9333 200.9333 201.9333 202.9333 203.9333 204.9333 205.9333 206.9333

Columns 209 through 216

207.9333 208.9333 209.9333 210.9333 211.9333 212.9333 213.9333 214.9333

Columns 217 through 224

215.9333 216.9333 217.9333 218.9333 219.9333 220.9333 221.9333 222.9333

Columns 225 through 232

223.9333 224.9333 225.9333 226.9333 227.9333 228.9333 229.9333 230.9333

Columns 233 through 240

231.9333 232.9333 233.9333 234.9333 235.9333 236.9333 237.9333 238.9333

Columns 241 through 248

239.9333 240.9333 241.9333 242.9333 243.9333 244.9333 245.9333 246.9333

Columns 249 through 256

247.9333 248.9333 249.9333 250.9333 251.9333 252.9333 253.9333 254.9333

Columns 257 through 264

255.9333 256.9333 257.9333 258.9333 259.9333 260.9333 261.9333 262.9333

Columns 265 through 272

263.9333 264.9333 265.9333 266.9333 267.9333 268.9333 269.9333 270.9333

Columns 273 through 280

271.9333 272.9333 273.9333 274.9333 275.9333 276.9333 277.9333 278.9333

Columns 281 through 288

279.9333 280.9333 281.9333 282.9333 283.9333 284.9333 285.9333 286.9333

Columns 289 through 296

# mbscfamounts

---

287.9333 288.9333 289.9333 290.9333 291.9333 292.9333 293.9333 294.9333

Columns 297 through 304

295.9333 296.9333 297.9333 298.9333 299.9333 300.9333 301.9333 302.9333

Columns 305 through 312

303.9333 304.9333 305.9333 306.9333 307.9333 308.9333 309.9333 310.9333

Columns 313 through 320

311.9333 312.9333 313.9333 314.9333 315.9333 316.9333 317.9333 318.9333

Columns 321 through 328

319.9333 320.9333 321.9333 322.9333 323.9333 324.9333 325.9333 326.9333

Columns 329 through 334

327.9333 328.9333 329.9333 330.9333 331.9333 332.9333

Prepayment =

Columns 1 through 8

1.0000 0.9944 0.9887 0.9828 0.9769 0.9711 0.9653 0.9595

Columns 9 through 16

0.9538 0.9481 0.9424 0.9368 0.9311 0.9255 0.9199 0.9144

Columns 17 through 24

0.9089 0.9034 0.8979 0.8925 0.8871 0.8817 0.8763 0.8710

Columns 25 through 32

0.8657 0.8604 0.8552 0.8499 0.8447 0.8396 0.8344 0.8293

Columns 33 through 40

0.8242 0.8191 0.8140 0.8090 0.8040 0.7990 0.7941 0.7892

Columns 41 through 48

0.7842 0.7794 0.7745 0.7697 0.7649 0.7601 0.7553 0.7506

Columns 49 through 56

0.7458 0.7411 0.7365 0.7318 0.7272 0.7226 0.7180 0.7134

Columns 57 through 64

0.7089 0.7044 0.6999 0.6954 0.6910 0.6865 0.6821 0.6777

Columns 65 through 72

0.6734 0.6690 0.6647 0.6604 0.6561 0.6519 0.6476 0.6434

Columns 73 through 80

0.6392 0.6350 0.6309 0.6267 0.6226 0.6185 0.6144 0.6104

Columns 81 through 88

0.6063 0.6023 0.5983 0.5943 0.5903 0.5864 0.5825 0.5785

Columns 89 through 96

0.5747 0.5708 0.5669 0.5631 0.5593 0.5555 0.5517 0.5479

Columns 97 through 104

# mbscfamounts

---

0.5442	0.5405	0.5368	0.5331	0.5294	0.5257	0.5221	0.5185
Columns 105 through 112							
0.5149	0.5113	0.5077	0.5042	0.5006	0.4971	0.4936	0.4901
Columns 113 through 120							
0.4866	0.4832	0.4797	0.4763	0.4729	0.4695	0.4661	0.4628
Columns 121 through 128							
0.4594	0.4561	0.4528	0.4495	0.4462	0.4430	0.4397	0.4365
Columns 129 through 136							
0.4333	0.4301	0.4269	0.4237	0.4205	0.4174	0.4143	0.4111
Columns 137 through 144							
0.4080	0.4049	0.4019	0.3988	0.3958	0.3927	0.3897	0.3867
Columns 145 through 152							
0.3837	0.3808	0.3778	0.3748	0.3719	0.3690	0.3661	0.3632
Columns 153 through 160							
0.3603	0.3574	0.3546	0.3517	0.3489	0.3461	0.3433	0.3405
Columns 161 through 168							
0.3377	0.3350	0.3322	0.3295	0.3267	0.3240	0.3213	0.3186
Columns 169 through 176							



0.3160 0.3133 0.3106 0.3080 0.3054 0.3027 0.3001 0.2975

Columns 177 through 184

0.2950 0.2924 0.2898 0.2873 0.2847 0.2822 0.2797 0.2772

Columns 185 through 192

0.2747 0.2722 0.2698 0.2673 0.2649 0.2624 0.2600 0.2576

Columns 193 through 200

0.2552 0.2528 0.2504 0.2480 0.2457 0.2433 0.2410 0.2386

Columns 201 through 208

0.2363 0.2340 0.2317 0.2294 0.2271 0.2249 0.2226 0.2204

Columns 209 through 216

0.2181 0.2159 0.2137 0.2115 0.2093 0.2071 0.2049 0.2027

Columns 217 through 224

0.2005 0.1984 0.1962 0.1941 0.1920 0.1899 0.1877 0.1856

Columns 225 through 232

0.1836 0.1815 0.1794 0.1773 0.1753 0.1732 0.1712 0.1692

Columns 233 through 240

0.1671 0.1651 0.1631 0.1611 0.1591 0.1572 0.1552 0.1532

Columns 241 through 248

0.1513 0.1493 0.1474 0.1455 0.1436 0.1416 0.1397 0.1378

# mbscfamounts

---

Columns 249 through 256

0.1359	0.1341	0.1322	0.1303	0.1285	0.1266	0.1248	0.1229
--------	--------	--------	--------	--------	--------	--------	--------

Columns 257 through 264

0.1211	0.1193	0.1175	0.1157	0.1139	0.1121	0.1103	0.1085
--------	--------	--------	--------	--------	--------	--------	--------

Columns 265 through 272

0.1068	0.1050	0.1032	0.1015	0.0998	0.0980	0.0963	0.0946
--------	--------	--------	--------	--------	--------	--------	--------

Columns 273 through 280

0.0929	0.0912	0.0895	0.0878	0.0861	0.0844	0.0827	0.0811
--------	--------	--------	--------	--------	--------	--------	--------

Columns 281 through 288

0.0794	0.0778	0.0761	0.0745	0.0728	0.0712	0.0696	0.0680
--------	--------	--------	--------	--------	--------	--------	--------

Columns 289 through 296

0.0664	0.0648	0.0632	0.0616	0.0600	0.0584	0.0568	0.0553
--------	--------	--------	--------	--------	--------	--------	--------

Columns 297 through 304

0.0537	0.0522	0.0506	0.0491	0.0476	0.0460	0.0445	0.0430
--------	--------	--------	--------	--------	--------	--------	--------

Columns 305 through 312

0.0415	0.0400	0.0385	0.0370	0.0355	0.0340	0.0325	0.0311
--------	--------	--------	--------	--------	--------	--------	--------

Columns 313 through 320

0.0296	0.0281	0.0267	0.0252	0.0238	0.0223	0.0209	0.0195
--------	--------	--------	--------	--------	--------	--------	--------

Columns 321 through 328

0.0180 0.0166 0.0152 0.0138 0.0124 0.0110 0.0096 0.0082

Columns 329 through 334

0.0068 0.0055 0.0041 0.0027 0.0014 0

## References

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

## See Also

mbspassthrough | mbsnoprepay | cmoseqcf | cmoschedcf | cmosched

# mbsconvp

---

**Purpose** Convexity of mortgage pool given price

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

<b>Arguments</b>	Price	Clean price for every \$100 face value.
	Settle	Settlement date. A serial date number or date string. <b>Settle</b> must be earlier than <b>Maturity</b> .
	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 = <b>GrossRate</b> .
	Delay	Delay in days.
	PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. Set <b>PrepaySpeed</b> to [] if you input a customized prepayment matrix.
	PrepayMatrix	(Optional) Used only when <b>PrepaySpeed</b> is unspecified. Customized prepayment vector. A NaN-padded matrix of size $\max(\text{TermRemaining})$ -by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

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

## Description

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

---

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

---

## Examples

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

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

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

```
Convexity =
    71.6299
```

## References

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

## See Also

mbsconvy | mbsdurp | mbspassthrough | mbsnoprepay | mbsdury

# mbsconvy

---

**Purpose** Convexity of mortgage pool given yield

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

**Arguments**

Yield	Mortgage yield, compounded monthly (in decimal).
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	Net coupon rate, in decimal. Default = GrossRate.
Delay	Delay in days.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Used only when PrepaySpeed is unspecified. Customized prepayment vector. A NaN-padded matrix of size max(TermRemaining)-by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

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

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

computes mortgage-backed security convexity, given time information, semiannual mortgage yield, and optionally, a prepayment model.

---

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

---

## Examples

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

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

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

```
Convexity =
```

```
72.8263
```

## References

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

## See Also

mbsconvp | mbsdurp | mbsdury

# mbsdurp

---

**Purpose** Duration of mortgage pool given price

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

**Arguments**

Price	Clean price for every \$100 face value.
Settle	Settlement date. A serial date number or date string. <b>Settle</b> must be earlier than or equal to <b>Maturity</b> .
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 = <b>GrossRate</b> .
Delay	Delay in days.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. Set <b>PrepaySpeed</b> to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Used only when <b>PrepaySpeed</b> is unspecified. Customized prepayment vector. A NaN-padded matrix of size $\max(\text{TermRemaining})$ -by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

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



## Description

[YearDuration, ModDuration] = mbsdurp(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes the mortgage-backed security Macaulay (YearDuration) and modified (ModDuration) durations, given time information, price at settlement, and optionally, a prepayment model.

---

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

---

## Examples

Given a mortgage-backed security with the following characteristics, compute the Macaulay and modified durations of the security.

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

[YearDuration, ModDuration] = mbsdurp(Price, Settle, Maturity,...
IssueDate, GrossRate, CouponRate, Delay, Speed)

YearDuration =

    6.4380

ModDuration =

    6.2080
```

## References

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

# mbsdurp

---

## See Also

[mbsconvp](#) | [mbsconvy](#) | [mbsdury](#)

**Purpose** Duration of mortgage pool given yield

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

**Arguments**

Yield	Mortgage yield, compounded monthly, in decimal.
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	Net coupon rate, in decimal. Default = GrossRate.
Delay	Delay in days.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0. Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Used only when PrepaySpeed is unspecified. Customized prepayment vector. A NaN-padded matrix of size max(TermRemaining)-by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

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

**Description** [YearDuration, ModDuration] = mbsdury(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay,

PrepaySpeed, PrepayMatrix) computes the mortgage-backed security Macaulay (YearDuration) and Modified (ModDuration) durations, given time information, yield to maturity, and optionally, a prepayment model.

---

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

---

## Examples

Given a mortgage-backed security with the following characteristics, compute the Macaulay and Modified durations of the security.

```
Yield = 0.07298413;
Settle = '15-Apr-2002';
Maturity = '1 Jan 2030';
IssueDate = '1-Jan-2000';
GrossRate = 0.08125;
Speed = 100;
CouponRate = 0.075;
Delay = 14;

[YearDuration, ModDuration] = mbsdury(Yield, Settle, Maturity,...
IssueDate, GrossRate, CouponRate, Delay, Speed)

YearDuration =

    6.4380

ModDuration =

    6.2080
```

## References

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

## See Also

mbsconvp | mbsconvy | mbsdurp

**Purpose** End-of-month mortgage cash flows and balances without prepayment

**Syntax** [Balance, Interest, Payment, Principal] =  
mbsnoprepay(OriginalBalance, GrossRate, Term)

**Arguments**

OriginalBalance	Original face value in dollars.
GrossRate	Gross coupon rate (including fees), in decimal.
Term	Term of the mortgage in months.

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

**Description** [Balance, Interest, Payment, Principal] =  
mbsnoprepay(OriginalBalance, GrossRate, Term) computes end-of-month mortgage balance, interest payments, principal payments, and cash flow payments with zero prepayment rate.

The function returns amortizing cash flows and balances over a specified term with no prepayment. When the lengths of pass-throughs are not the same, MATLAB software pads the shorter ones with NaN.

**Balance** lists the end-of-month balances over the life of the pass-through.

**Interest** lists all end-of-month interest payments over the life of the pass-through.

**Payment** lists all end-of-month payments over the life of the pass-through.

**Principal** lists all scheduled end-of-month principal payments over the life of the pass-through.

All outputs are Term-by-1 vectors.

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

# mbsnoprepay

---

```
OriginalBalance = 400000000;  
CouponRate = 0.08125;  
Term = [357; 355]; % Three- and five-month old mortgage pools.  
  
[Balance, Interest, Payment, Principal] = ...  
mbsnoprepay(OriginalBalance, CouponRate, Term);
```

## Purpose

Price given option-adjusted spread

## Syntax

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

## Arguments

ZeroCurve	A matrix of three columns: <ul style="list-style-type: none"><li>• Column 1: Serial date numbers.</li><li>• Column 2: Spot rates with maturities corresponding to the dates in Column 1, in decimal (for example, 0.075).</li><li>• Column 3: Compounding of the rates in Column 2. (This is the agency spot rate on the settlement date.)</li></ul>
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. <b>Settle</b> must be earlier than <b>Maturity</b> .
Maturity	Maturity date. Scalar or vector in serial date number or date string format.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).

# mbsoas2price

---

- Interpolation** Interpolation method. Computes the corresponding spot rates for the bond's cash flow. Available methods are (0) nearest, (1) linear, and (2) cubic spline. Default = 1. See `interp1` for more information.
- PrepaySpeed** (Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set `PrepaySpeed` to [] if you input a customized prepayment matrix.
- PrepayMatrix** (Optional) Customized prepayment matrix. A matrix of size `max(TermRemaining)`-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

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

## Description

`Price = mbsoas2price(ZeroCurve, OAS, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)` computes the clean price of a pass-through security for each \$100 face value of outstanding principal.

## Examples

Given an option-adjusted spread, a spot curve, and a prepayment assumption, compute theoretical price of a mortgage pool.

Create the bonds matrix.

```
Bonds = [datenum('11/21/2002') 0      100 0 2 1;
          datenum('02/20/2003') 0      100 0 2 1;
          datenum('07/31/2004') 0.03   100 2 3 1;
          datenum('08/15/2007') 0.035  100 2 3 1;
          datenum('08/15/2012') 0.04875 100 2 3 1;
          datenum('02/15/2031') 0.05375 100 2 3 1];
```



Choose a settlement date.

```
Settle = datenum('20-Aug-2002');
```

Assume these clean prices for the bonds.

```
Prices = [ 98.97467;  
          98.58044;  
          100.10534;  
          98.18054;  
          101.38136;  
          99.25411];
```

Use this formula to compute spot compounding for the bonds.

```
SpotCompounding = 2*ones(size(Prices));
```

Use compute the zero curve.

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

Assign parameters.

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

Calculate the price from the option-adjusted spread.

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

# mbsoas2price

---

Price =

95.0000

95.0000

95.0000

## See Also

[mbsprice2oas](#) | [mbsyield2oas](#) | [mbsoas2yield](#)

## Purpose

Yield given option-adjusted spread

## Syntax

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

## Arguments

ZeroCurve	A matrix of three columns: <ul style="list-style-type: none"><li>• Column 1: Serial date numbers.</li><li>• Column 2: Spot rates with maturities corresponding to the dates in Column 1, in decimal (for example, 0.075).</li><li>• Column 3: Compounding of the rates in Column 2. (This is the agency spot rate on the settlement date.)</li></ul>
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. <b>Settle</b> must be earlier than <b>Maturity</b> .
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 = <code>GrossRate</code> .
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).

# mbsoas2yield

---

Interpolation	Interpolation method. Computes the corresponding spot rates for the bond's cash flow. Available methods are (0) nearest, (1) linear, and (2) cubic spline. Default = 1. See <code>interp1</code> for more information.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set <code>PrepaySpeed</code> to <code>[]</code> if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size <code>max(TermRemaining)</code> -by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

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

## Description

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

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

0.075 (7.5%)

`BEMBSYield` is the corresponding bond equivalent yield of the mortgage-backed security. This yield is compounded semiannually (two times per year). For example:

0.0761 (7.61%)

## Examples

Given an option-adjusted spread, a spot curve, and a prepayment assumption, compute the theoretical yield to maturity of a mortgage pool.

Create the bonds matrix.

```
Bonds = [datenum('11/21/2002') 0 100 0 2 1;
         datenum('02/20/2003') 0 100 0 2 1;
         datenum('07/31/2004') 0.03 100 2 3 1;
         datenum('08/15/2007') 0.035 100 2 3 1;
         datenum('08/15/2012') 0.04875 100 2 3 1;
         datenum('02/15/2031') 0.05375 100 2 3 1];
```

Choose a settlement date.

```
Settle = datenum('08/20/2002');
```

Assume these clean prices for the bonds.

```
Prices = [ 98.97467;
          98.58044;
          100.10534;
          98.18054;
          101.38136;
          99.25411];
```

Use this formula to compute spot compounding for the bonds.

```
SpotCompounding = 2*ones(size(Prices));
```

Compute the zero curve.

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

Assign parameters.

```
OAS = [26.0502; 28.6348; 31.2222];
Maturity = datenum('02-Jan-2030');
```

# mbsoas2yield

---

```
IssueDate      = datenum('02-Jan-2000');
GrossRate      = 0.08125;
CouponRate     = 0.075;
Delay          = 14;
Interpolation  = 1;
PrepaySpeed    = [0 50 100];
```

Compute the mortgage yield and bond equivalent mortgage yield.

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

```
MYield =
```

```
0.0802
0.0814
0.0828
```

```
BEMBSYield =
```

```
0.0816
0.0828
0.0842
```

## See Also

[mbsprice2oas](#) | [mbsyield2oas](#) | [mbsoas2price](#)

## Purpose

Mortgage pool cash flows and balances with prepayment

## Syntax

```
[Balance, Payment, Principal, Interest, Prepayment] =  
mbspassthrough(OriginalBalance, GrossRate, OriginalTerm,  
TermRemaining, PrepaySpeed, PrepayMatrix)
```

## Arguments

OriginalBalance	Original balance value in dollars (balance at the beginning of each TermRemaining).
GrossRate	Gross coupon rate (including fees), in decimal.
OriginalTerm	Term of the mortgage in months.
TermRemaining	(Optional) Number of full months between settlement and maturity.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0 (no prepayment). Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Used only when PrepaySpeed is unspecified. Customized prepayment vector. A NaN-padded matrix of size max(TermRemaining)-by-NMBS. Each column corresponds to each mortgage-backed security, and each row corresponds to each month after settlement.

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

## Description

```
[Balance, Payment, Principal, Prepayment, Interest] =  
mbspassthrough(OriginalBalance, GrossRate, OriginalTerm,  
TermRemaining, PrepaySpeed, PrepayMatrix) computes the cash  
flow of principal, interest, and prepayment of pass-through securities.
```

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

Balance is a matrix for the principal balance at end of month.

Payment is a matrix for the total monthly payment.

Principal is a matrix for the principal portion of the payment.

Interest is a matrix for the interest portion of the payment.

Prepayment is a matrix that indicates any unscheduled principal payment.

By default, the securities are seasoned. The applicable CPR depends upon `TermRemaining` based on a 30-year prepayment model (PSA or FHA). You may supply a different CPR vector of size `TermRemaining-by-1`.

## Examples

Compute the cash flows and balances of a 3-month old mortgage pool with original term of 360 months, assuming a prepayment speed of 100.

```
OriginalBalance = 100000;  
GrossRate = 0.08125;  
OriginalTerm = 360;  
TermRemaining = 357;  
PrepaySpeed = 100;
```

```
[Balance, Payment, Principal, Interest, Prepayment] =...  
mbspassthrough(OriginalBalance, GrossRate, OriginalTerm,...  
TermRemaining, PrepaySpeed);
```

## See Also

mbswal



**Purpose** Mortgage-backed security price given yield

**Syntax** [Price, AccrInt] = mbsprice(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)

**Arguments**

- Yield Mortgage yield, compounded monthly (in decimal).
- Settle Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
- Maturity Maturity date. A serial date number or date string.
- IssueDate Issue date. A serial date number or date string.
- GrossRate Gross coupon rate (including fees), in decimal.
- CouponRate (Optional) Net coupon rate, in decimal. Default = GrossRate.
- Delay (Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).
- PrepaySpeed (Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0 (no prepayment). Set PrepaySpeed to [] if you input a customized prepayment matrix.
- PrepayMatrix (Optional) Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

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

## Description

[Price, AccrInt] = mbsprice(Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes a mortgage-backed security price, given time information, mortgage yield at settlement, and optionally, a prepayment model.

All outputs are scalar values.

Price is the clean price for every \$100 face value of the securities.

AccrInt is the accrued interest of the mortgage-backed securities.

## Examples

**Example 1.** Given a mortgage-backed security with the following characteristics, compute the price and the accrued interest due on the security.

```
Yield = 0.0725;
Settle = datenum('15-Apr-2002');
Maturity = datenum('1 Jan 2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;

[Price AccrInt] = mbsprice(Yield, Settle, Maturity, IssueDate,...
GrossRate, CouponRate, Delay, Speed)

Price =

    101.3147

AccrInt =

    0.2917
```

**Example 2.** Given a portfolio of mortgage-backed securities, compute the clean prices and accrued interest.

```
Yield = 0.075;
Settle = datenum(['13-Feb-2000'; '17-Apr-2002'; '17-May-2002'; ...
'13-Jan-2000']);
Maturity = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = [0.075; 0.07875; 0.0775; 0.08125];
Delay = 14;
Speed = 100;

[Price AccrInt] = mbsprice(Yield, Settle, Maturity, IssueDate, ...
GrossRate, CouponRate, Delay, Speed)

Price =

    99.7085
   102.0678
   101.2792
   104.0175

AccrInt =

    0.2500
    0.3500
    0.3444
    0.2708
```

## References

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

## See Also

mbsyield

# mbsprice2oas

---

**Purpose** Option-adjusted spread given price

**Syntax** OAS = mbsprice2oas(ZeroCurve, Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation PrepaySpeed, PrepayMatrix)

**Arguments**

ZeroCurve	A matrix of three columns: <ul style="list-style-type: none"><li>• Column 1: Serial date numbers.</li><li>• Column 2: Spot rates with maturities corresponding to the dates in Column 1, in decimal (for example, 0.075).</li><li>• Column 3: Compounding of the rates in Column 2. Values are 1 (annual), 2 (semiannual, 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).</li></ul>
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. <b>Settle</b> must be earlier than <b>Maturity</b> .
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 = <b>GrossRate</b> .

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 <code>interp1</code> for more information.
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set <code>PrepaySpeed</code> to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size <code>max(TermRemaining)</code> -by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

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

## Description

`OAS = mbsprice2oas(ZeroCurve, Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)` computes the monthly option-adjusted spread in basis points.

## Examples

Calculate the option-adjusted spread of a 30-year fixed-rate mortgage with about a 28-year weighted average maturity remaining, given assumptions of 0, 50, and 100 PSA prepayments.

Create the bonds matrix.

```
Bonds = [datenum('11/21/2002') 0      100 0 2 1;
          datenum('02/20/2003') 0      100 0 2 1;
```

## mbsprice2oas

---

```
          datenum('07/31/2004')    0.03      100  2  3  1;  
          datenum('08/15/2007')    0.035     100  2  3  1;  
          datenum('08/15/2012')    0.04875  100  2  3  1;  
          datenum('02/15/2031')    0.05375  100  2  3  1];
```

Choose a settlement date.

```
Settle= datenum('20-Aug-2002');
```

Assume these clean prices for the bonds.

```
Prices = [ 98.97467;  
          98.58044;  
          100.10534;  
          98.18054;  
          101.38136;  
          99.25411];
```

Use this formula to compute spot compounding for the bonds.

```
SpotCompounding = 2*ones(size(Prices));
```

Compute the zero curve.

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

Assign parameters.

```
Price          = 95;  
Maturity       = datenum('02-Jan-2030');  
IssueDate      = datenum('02-Jan-2000');  
GrossRate      = 0.08125;  
CouponRate     = 0.075;  
Delay          = 14;  
Interpolation  = 1;  
PrepaySpeed    = [0; 50; 100];  
Interpolation  = 1;
```

Compute the option-adjusted spread.

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

OAS =

26.0502

28.6348

31.2222

## See Also

[mbssoas2price](#) | [mbssoas2yield](#) | [mbsyield2oas](#)

# mbsprice2speed

---

**Purpose** Implied PSA prepayment speeds given price

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

## Arguments

Price	Clean price for every \$100 face value.
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
PrepayMatrix	Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).

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

## Description

[ImpSpdOnPrc, ImpSpdOnDur, ImpSpdOnCnv] = mbsprice2speed(Price, Settle, Maturity, IssueDate, GrossRate, PrepayMatrix, CouponRate, Delay) computes PSA prepayment speeds implied by pool prices and projected (user-defined) prepayment vectors. The calculated PSA speed produces the same



price, modified duration, or modified convexity, depending upon the output requested.

ImpSpdOnPrc calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same price.

ImpSpdOnDur calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same modified duration.

ImpSpdOnCnv calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same modified convexity.

All outputs are NMBS-by-1 vectors.

## Examples

Calculate the equivalent PSA benchmark prepayment speeds for a mortgage pool with these characteristics and prepayment matrix.

```
Price          = 101;
Settle         = datenum('1-Jan-2000');
Maturity       = datenum('1-Jan-2030');
IssueDate     = datenum('1-Jan-2000');
GrossRate     = 0.08125;
PrepayMatrix  = 0.005*ones(360,1);
CouponRate    = 0.075;
Delay         = 14;
```

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

```
ImpSpdOnPrc =
```

```
    118.5980
```

```
ImpSpdOnDur =
```

```
    118.3946
```

```
ImpSpdOnCnv =
```

# mbsprice2speed

---

109.5115

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

**See Also** mbsprice | mbsyield2speed

**Purpose** Weighted average life of mortgage pool

**Compatibility** PSA

**Syntax** `WAL = mbswal(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix)`

<b>Arguments</b>	<b>Settle</b>	Settlement date. A serial date number or date string. <b>Settle</b> must be earlier than <b>Maturity</b> .
	<b>Maturity</b>	Maturity date. A serial date number or date string.
	<b>IssueDate</b>	Issue date. A serial date number or date string.
	<b>GrossRate</b>	Gross coupon rate (including fees), in decimal.
	<b>CouponRate</b>	(Optional) Net coupon rate, in decimal. Default = <b>GrossRate</b> .
	<b>Delay</b>	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).
	<b>PrepaySpeed</b>	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set <b>PrepaySpeed</b> to [] if you input a customized prepayment matrix.
	<b>PrepayMatrix</b>	(Optional) Customized prepayment matrix. A matrix of size <code>max(TermRemaining)</code> -by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

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

# mbswal

---

## Description

WAL = mbswal(Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed, PrepayMatrix) computes the weighted average life, in number of years, of a mortgage pool, as measured from the settlement date.

## Examples

Given a pass-through security with the following characteristics, compute the weighted average life of the security.

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

```
WAL = mbswal(Settle, Maturity, IssueDate, GrossRate, ...
CouponRate, Delay, Speed)
```

```
WAL =
    10.5477
```

## References

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

## See Also

mbspassthrough

**Purpose** Mortgage-backed security yield given price

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

**Arguments**

Price	Clean price for every \$100 face value.
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).
PrepaySpeed	(Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = 0 (no prepayment). Set PrepaySpeed to [] if you input a customized prepayment matrix.
PrepayMatrix	(Optional) Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

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

**Description** [MYield, BEMBSYield] = mbsyield(Price, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, PrepaySpeed,

PrepayMatrix) computes a mortgage-backed security yield to maturity and the bond equivalent yield, given time information, price at settlement, and optionally, a prepayment model.

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

BEMBSYield is the corresponding bond equivalent yield of the mortgage-backed security. This yield is compounded semiannually (two times a year).

## Examples

**Example 1.** Given a mortgage-backed security with the following characteristics, compute the mortgage yield and the bond equivalent yield of the security.

```
Price = 102;
Settle = '15-Apr-2002';
Maturity = '1 Jan 2030';
IssueDate = '1-Jan-2000';
GrossRate = 0.08125;
CouponRate = 0.075;
Delay = 14;
Speed = 100;

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

```
MYield =

    0.0715
```

```
BEMBSYield =

    0.0725
```

**Example 2.** Given a portfolio of mortgage-backed securities, compute the mortgage yields and the bond equivalent yields.

```

Price = 102;
Settle = datenum(['13-Feb-2000'; '17-Apr-2002'; '17-May-2002'; ...
'13-Jan-2000']);
Maturity = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
CouponRate = [0.075; 0.07875; 0.0775; 0.08125];
Delay = 14;
Speed = 100;

```

```

[MYield, BEMBSYield] = mbsyield(Price, Settle, Maturity, ...
IssueDate, GrossRate, CouponRate, Delay, Speed)

```

MYield =

```

    0.0717
    0.0751
    0.0739
    0.0779

```

BEMBSYield =

```

    0.0728
    0.0763
    0.0750
    0.0791

```

## References

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

## See Also

mbsprice

# mbsyield2oas

---

**Purpose** Option-adjusted spread given yield

**Syntax** OAS = mbsyield2oas(ZeroCurve, Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation PrepaySpeed, PrepayMatrix)

**Arguments**

ZeroCurve	A matrix of three columns: <ul style="list-style-type: none"><li>• Column 1: Serial date numbers.</li><li>• Column 2: Spot rates with maturities corresponding to the dates in Column 1, in decimal (for example, 0.075).</li><li>• Column 3: Compounding of the rates in Column 2. Values are 1 (annual), 2 (semiannual), 3 (three times per year), 4 (quarterly), 6 (bimonthly), 12 (monthly), and -1 (continuous).</li></ul>
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. <b>Settle</b> must be earlier than <b>Maturity</b> .
Maturity	Maturity date. Scalar or vector in serial date number or date string format.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).



- Interpolation** Interpolation method. Computes the corresponding spot rates for the bond's cash flow. Available methods are (0) nearest, (1) linear, and (2) cubic spline. Default = 1. See `interp1` for more information.
- PrepaySpeed** (Optional) Relation of the conditional payment rate (CPR) to the benchmark model. Default = end of month's CPR. Set `PrepaySpeed` to [] if you input a customized prepayment matrix.
- PrepayMatrix** (Optional) Customized prepayment matrix. A matrix of size `max(TermRemaining)`-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.

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

## Description

`OAS = mbsyield2oas(ZeroCurve, Yield, Settle, Maturity, IssueDate, GrossRate, CouponRate, Delay, Interpolation, PrepaySpeed, PrepayMatrix)` computes the option-adjusted spread in basis points.

## Examples

Calculate the option-adjusted spread of a 30-year, fixed-rate mortgage pool with about 28-year weighted average maturity left, given assumptions of 0, 50, and 100 PSA prepayments.

Create a bonds matrix.

```
Bonds = [datenum('11/21/2002') 0      100 0 2 1;
         datenum('02/20/2003') 0      100 0 2 1;
         datenum('07/31/2004') 0.03  100 2 3 1;
         datenum('08/15/2007') 0.035 100 2 3 1;
         datenum('08/15/2012') 0.04875 100 2 3 1;
         datenum('02/15/2031') 0.05375 100 2 3 1];
```

Choose a settlement date.

```
Settle = datenum('08/20/2002');
```

Assume these clean prices for the bonds.

```
Prices = [ 98.97467;  
          98.58044;  
          100.10534;  
          98.18054;  
          101.38136;  
          99.25411];
```

Use this formula to compute spot compounding for the bonds.

```
SpotCompounding = 2*ones(size(Prices));
```

Compute the zero curve.

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

Assign parameters.

```
Price          = 95;  
Maturity       = datenum('02-Jan-2030');  
IssueDate      = datenum('02-Jan-2000');  
GrossRate      = 0.08125;  
CouponRate     = 0.075;  
Delay          = 14;  
Interpolation  = 1;  
PrepaySpeed    = [0 50 100];
```

Compute the yield, and from the yield, compute the option-adjusted spread.

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

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

```
OAS =
```

```
26.0502
```

```
28.6348
```

```
31.2222
```

## See Also

```
mbsoas2price | mbsoas2yield | mbsprice2oas
```

# mbsyield2speed

---

**Purpose** Implied PSA prepayment speeds given yield

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

**Arguments**

Yield	Mortgage yield, compounded monthly, in decimal.
Settle	Settlement date. A serial date number or date string. Settle must be earlier than Maturity.
Maturity	Maturity date. A serial date number or date string.
IssueDate	Issue date. A serial date number or date string.
GrossRate	Gross coupon rate (including fees), in decimal.
PrepayMatrix	Customized prepayment matrix. A matrix of size max(TermRemaining)-by-NMBS. Missing values are padded with NaNs. Each column corresponds to a mortgage-backed security, and each row corresponds to each month after settlement.
CouponRate	(Optional) Net coupon rate, in decimal. Default = GrossRate.
Delay	(Optional) Delay (in days) between payment from homeowner and receipt by bondholder. Default = 0 (no delay between payment and receipt).

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

**Description** [ImpSpdOnYld, ImpSpdOnDur, ImpSpdOnCnv] = mbsyield2speed(Yield, Settle, Maturity, IssueDate, GrossRate, PrepayMatrix, CouponRate, Delay) computes PSA prepayment speeds implied by pool yields and projected (user-defined) prepayment vectors. The calculated PSA speed produces the same

yield, modified duration, or modified convexity, depending upon the output requested.

`ImpSpdOnPrc` calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same price.

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

`ImpSpdOnCnv` calculates the equivalent PSA benchmark prepayment speed for the pass-through to carry the same modified convexity.

All outputs are NMBS-by-1 vectors.

## Examples

Calculate the equivalent PSA benchmark prepayment speeds for a security with these characteristics and prepayment matrix.

```
Yield = 0.065;
Settle = datenum('1-Jan-2000');
Maturity = datenum('1-Jan-2030');
IssueDate = datenum('1-Jan-2000');
GrossRate = 0.08125;
PrepayMatrix = 0.005*ones(360,1);
CouponRate = 0.075;
Delay = 14;

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

ImpSpdOnYld =

    117.7644

ImpSpdOnDur =

    116.7436

ImpSpdOnCnv =
```

# mbsyield2speed

---

108.3309

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

**See Also** mbsyield | mbsprice2speed

**Purpose** Benchmark default

**Syntax** [ADRPSA, MDRPSA] = psaspeed2default(DefaultSpeed)

**Arguments**

DefaultSpeed	Annual speed relative to the benchmark. PSA benchmark = 100.
--------------	--

**Description** [ADRPSA, MDRPSA] = psaspeed2default(DefaultSpeed) computes the benchmark default on the performing balance of mortgage-backed securities per PSA benchmark speed.

ADRPSA is the PSA default rate, in decimal (360-by-1).

MDRPSA is the PSA monthly default rate, in decimal (360-by-1).

**Examples** Given a mortgage-backed security with annual speed set at the PSA default benchmark, compute the default rates.

```
DefaultSpeed = 100;
```

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

**See Also** psaspeed2rate

# psaspeed2rate

---

**Purpose** Single monthly mortality rate given PSA speed

**Syntax** [CPRPSA, SMMPSA]= psaspeed2rate(PSASpeed)

**Arguments**

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

**Description** [CPRPSA, SMMPSA]= psaspeed2rate(PSASpeed) calculates vectors of PSA prepayments, each containing 360 prepayment elements, to represent the 360 months in a 30-year mortgage pool.

CPRPSA is the PSA conditional prepayment rate, in decimal [360-by-1].

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

**Examples** Given a mortgage-backed security with annual speed set at the PSA default benchmark, compute the prepayment and mortality rates.

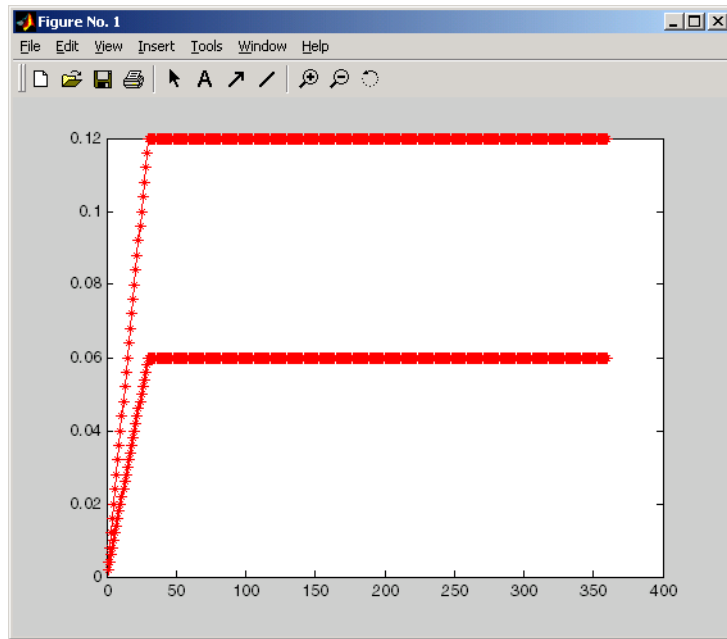
```
PSASpeed = [100 200];
```

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

View a plot of the output.

```
psaspeed2rate(PSASpeed)
```





**See Also** [psaspeed2default](#)

# stepcpncfamounts

---

**Purpose** Cash flow amounts and times for bonds and stepped coupons

**Syntax** [CFlows, CDates, CTimes] = stepcpncfamounts(Settle, Maturity, ConvDates, CouponRates, Period, Basis, EndMonthRule, Face)

**Arguments**

Settle	Settlement date. A scalar or vector of serial date numbers. <b>Settle</b> must be earlier than <b>Maturity</b> .
Maturity	Maturity date. A scalar or vector of serial date numbers.
ConvDates	Matrix of serial date numbers representing conversion dates after <b>Settle</b> . Size = number of instruments by maximum number of conversions. Fill unspecified entries with NaN.
CouponRates	Matrix indicating the coupon rates for each bond in decimal form. Size = number of instruments by maximum number of conversions + 1. First column of this matrix contains rates applicable between <b>Settle</b> and the first conversion date (date in the first column of <b>ConvDates</b> ). Fill unspecified entries with NaN. See Note below.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2, 3, 4, 6, and 12. Default = 2.

<b>Basis</b>	<p>(Optional) Day-count basis of the instrument. A vector of integers.</p> <ul style="list-style-type: none"> <li>• 0 = actual/actual (default)</li> <li>• 1 = 30/360 (SIA)</li> <li>• 2 = actual/360</li> <li>• 3 = actual/365</li> <li>• 4 = 30/360 (BMA)</li> <li>• 5 = 30/360 (ISDA)</li> <li>• 6 = 30/360 (European)</li> <li>• 7 = actual/365 (Japanese)</li> <li>• 8 = actual/actual (ICMA)</li> <li>• 9 = actual/360 (ICMA)</li> <li>• 10 = actual/365 (ICMA)</li> <li>• 11 = 30/360E (ICMA)</li> <li>• 12 = actual/actual (ISDA)</li> <li>• 13 = BUS/252</li> </ul>
<b>EndMonthRule</b>	<p>For more information, see basis.</p> <p>(Optional) End-of-month rule. A vector. This rule applies only when <code>Maturity</code> 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.</p>
<b>Face</b>	<p>(Optional) Face value of each bond in the portfolio. Default = 100.</p>

# stepcpncfamounts

---

All arguments must be scalars or number of bonds (NUMBONDS)-by-1 vectors, except for `ConvDates` and `CouponRates`.

---

**Note** `ConvDates` has the same number of rows as `CouponRates` to reflect the same number of bonds. However, `ConvDates` has one less column than `CouponRates`. This situation is illustrated by

```
Settle-----ConvDate1-----ConvDate2-----Maturity
                                     Rate1           Rate2           Rate3
```

---

## Description

`[CFloWS, CDates, CTimes] = stepcpncfamounts(Settle, Maturity, ConvDates, CouponRates, Period, Basis, EndMonthRule, Face)` returns matrices of cash flow amounts, cash flow dates, and time factors for a portfolio of NUMBONDS stepped-coupon bonds.

`CFloWS` is a matrix of cash flow amounts. The first entry in each row vector is a negative number indicating the accrued interest due at settlement. If no accrued interest is due, the first column is 0.

`CDates` is a matrix of cash flow dates in serial date number form. At least two columns are always present, one for settlement and one for maturity.

`CTimes` is a matrix of time factors for the SIA semiannual price/yield conversion.

$$\text{DiscountFactor} = (1 + \text{Yield}/2)^{(-\text{TFactor})}$$

Time factors are in units of semiannual coupon periods. In computing time factors, use SIA actual/actual conventions for all time factor calculations.

---

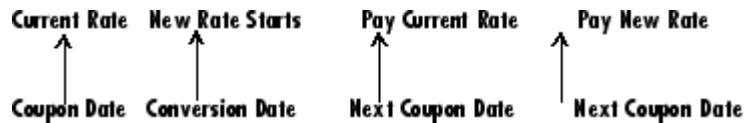
**Note** For bonds with fixed coupons, use `cfamounts`. If you use a fixed-coupon bond with `stepcpncfamounts`, MATLAB software generates an error.

---

## Examples

This example generates stepped cash flows for three different bonds, all paying interest semiannually. Their life span is about 18 to 19 years each:

- Bond A has two conversions, but the first one occurs on the settlement date and immediately expires.
- Bond B has three conversions, with conversion dates exactly on the coupon dates.
- Bond C has three conversions, with some conversion dates not on the coupon dates. It has the longest maturity. This case illustrates that only cash flows for full periods after conversion dates are affected, as illustrated below.



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

<b>Bond A Dates</b>	<b>Bond A Rates</b>	<b>Bond B Dates</b>	<b>Bond B Rates</b>	<b>Bond C Dates</b>	<b>Bond C Rates</b>
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%

# stepcpncfamounts

Bond A Dates	Bond A Rates	Bond B Dates	Bond B Rates	Bond C Dates	Bond C Rates
Second Conversion (15-Jun-03)	9.25%	Second Conversion (15-Jun-01)	9.25%	Second Conversion (14-Jun-01)	7.5%
Maturity (15-Jun-10)	NaN	Third Conversion (15-Jun-05)	10.0%	Third Conversion (14-Jun-05)	10.0%
		Maturity (15-Jun-10)	NaN	Maturity (15-Jun-11)	NaN

```

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

ConvDates = [datenum('02-Aug-1992'), datenum('15-Jun-2003'),...
             nan;
             datenum('15-Jun-1997'), datenum('15-Jun-2001'),...
             datenum('15-Jun-2005');
             datenum('14-Jun-1997'), datenum('14-Jun-2001'),...
             datenum('14-Jun-2005')];

Maturity = [datenum('15-Jun-2010');
            datenum('15-Jun-2010');
            datenum('15-Jun-2011')];

CouponRates = [0.075 0.08875 0.0925 nan;
               0.075 0.08875 0.0925 0.1;
               0.025 0.05 0.0750 0.1];

Basis = 1;
Period = 2;
EndMonthRule = 1;
Face = 100;

```

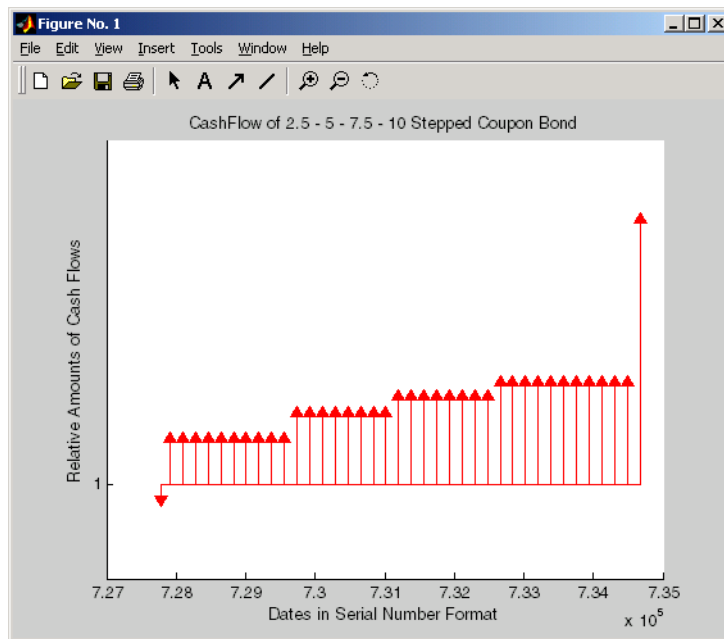
Call `stepcpncfamounts` to compute cash flows and timings.

```
[CFlows, CDates, CTimes] = stepcpncfamounts(Settle, Maturity, ...
```

```
ConvDates, CouponRates);
```

Visualize the third bond cash flows (2.5 - 5 - 7.5 - 10). (cfplot is available at /finance/findemos/cfplot.m.)

```
cfplot(CDates(3,:),CFFlows(3,:));
xlabel('Dates in Serial Number Format')
ylabel('Relative Amounts of Cash Flows')
title('CashFlow of 2.5 - 5 - 7.5 - 10 Stepped Coupon Bond')
```



**See Also** [stepcpnprice](#) | [stepcpnyield](#)

# stepcpnprice

---

**Purpose** Price bond with stepped coupons

**Syntax** [Price, AccruedInterest] = stepcpnprice(Yield, Settle, Maturity, ConvDates, CouponRates, Period, Basis, EndMonthRule, Face)

## Arguments

Yield	Scalar or vector containing yield to maturity of instruments.
Settle	Settlement date. A scalar or vector of serial date numbers. <b>Settle</b> must be earlier than <b>Maturity</b> .
Maturity	Maturity date. A scalar or vector of serial date numbers.
ConvDates	Matrix of serial date numbers representing conversion dates after <b>Settle</b> . Size = number of instruments by maximum number of conversions. Fill unspecified entries with NaN.
CouponRates	Matrix indicating the coupon rates for each bond in decimal form. Size = number of instruments by maximum number of conversions + 1. First column of this matrix contains rates applicable between <b>Settle</b> and the first conversion date (date in the first column of <b>ConvDates</b> ). Fill unspecified entries with NaN. See Note below.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2, 3, 4, 6, and 12. Default = 2.



<b>Basis</b>	<p>(Optional) Day-count basis of the instrument. A vector of integers.</p> <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul> <p>For more information, see basis.</p>
<b>EndMonthRule</b>	<p>(Optional) End-of-month rule. A vector. This rule applies only when <b>Maturity</b> 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.</p>
<b>Face</b>	<p>(Optional) Face value of each bond in the portfolio. Default = 100.</p>

# stepcpnprice

---

All arguments must be scalars or number of bonds (NUMBONDS)-by-1 vectors, except for `ConvDates` and `CouponRates`.

---

**Note** `ConvDates` has the same number of rows as `CouponRate` to reflect the same number of bonds. However, `ConvDates` has one less column than `CouponRate`. This situation is illustrated by

```
Settle-----ConvDate1-----ConvDate2-----Maturity
          Rate1           Rate2           Rate3
```

---

## Description

`[Price, AccruedInterest] = stepcpnprice(Yield, Settle, Maturity, ConvDates, CouponRates, Period, Basis, EndMonthRule, Face)` computes the price of bonds with stepped coupons given the yield to maturity. The function supports any number of conversion dates.

`Price` is a NUMBONDS-by-1 vector of clean prices.

`AccruedInterest` is a NUMBONDS-by-1 vector of accrued interest payable at settlement dates.

---

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

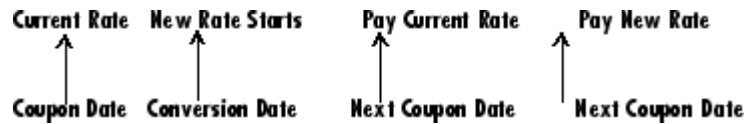
---

## Examples

Compute the price and accrued interest due on a portfolio of stepped-coupon bonds having a yield of 7.221%, given three conversion scenarios:

- Bond A has two conversions, the first one falling on the settle date and immediately expiring.

- Bond B has three conversions, with conversion dates exactly on the coupon dates.
- Bond C has three conversions, with one or more conversion dates not on coupon dates. This case illustrates that only cash flows for full periods after conversion dates are affected, as illustrated below.



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

<b>Bond A Dates</b>	<b>Bond A Rates</b>	<b>Bond B Dates</b>	<b>Bond B Rates</b>	<b>Bond C Dates</b>	<b>Bond C Rates</b>
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'),...

```

# stepcpnprice

---

```
        datenum('15-Jun-2005');
        datenum('14-Jun-1997'), datenum('14-Jun-2001'),...
        datenum('14-Jun-2005')];
Maturity = datenum('15-Jun-2010');

CouponRates = [0.075 0.08875 0.0925 nan;
               0.075 0.08875 0.0925 0.1;
               0.075 0.08875 0.0925 0.1];

Basis = 1;
Period = 2;
EndMonthRule = 1;
Face = 100;

[Price, AccruedInterest] = ...
stepcpnprice(Yield, Settle, Maturity, ConvDates, CouponRates, ...
Period, Basis, EndMonthRule, Face)

Price =

    117.3824
    113.4339
    113.4339

AccruedInterest =

    1.1587
    0.9792
    0.9792
```

## References

This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 120 - 123, on zero-coupon instruments pricing.

## See Also

bndprice | cdprice | stepcpncfamounts | stepcpnyield |  
tbillprice | zeroprice

**Purpose**

Yield to maturity of bond with stepped coupons

**Syntax**

Yield = stepcpnyield(Price, Settle, Maturity, ConvDates, CouponRate, Period, Basis, EndMonthRule, Face)

**Arguments**

Price	Vector containing price of the bonds.
Settle	Settlement date. A vector of serial date numbers. <b>Settle</b> must be earlier than <b>Maturity</b> .
Maturity	Maturity date. A vector of serial date numbers.
ConvDates	Matrix of serial date numbers representing conversion dates after <b>Settle</b> . Size = number of instruments by maximum number of conversions. Fill unspecified entries with NaN.
CouponRates	Matrix indicating the coupon rates for each bond in decimal form. Size = number of instruments by maximum number of conversions + 1. First column of this matrix contains rates applicable between <b>Settle</b> and the first conversion date (date in the first column of <b>ConvDates</b> ). Fill unspecified entries with NaN. See Note below.
Period	(Optional) Coupons per year of the bond. A vector of integers. Allowed values are 0, 1, 2, 3, 4, 6, and 12. Default = 2.

<b>Basis</b>	<p>(Optional) Day-count basis of the instrument. A vector of integers.</p> <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li><li>• 12 = actual/actual (ISDA)</li><li>• 13 = BUS/252</li></ul> <p>For more information, see basis.</p>
<b>EndMonthRule</b>	<p>(Optional) End-of-month rule. A vector. This rule applies only when <b>Maturity</b> 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.</p>
<b>Face</b>	<p>(Optional) Face value of each bond in the portfolio. Default = 100.</p>

All arguments must be number of bonds (NUMBONDS)-by-1 vectors, except for ConvDates and CouponRate.

---

**Note** ConvDates has the same number of rows as CouponRate to reflect the same number of bonds. However, ConvDates has one less column than CouponRate. This situation is illustrated by

```
Settle-----ConvDate1-----ConvDate2-----Maturity
           Rate1             Rate2             Rate3
```

---

## Description

Yield = stepcpnyield(Price, Settle, Maturity, ConvDates, CouponRate, Period, Basis, EndMonthRule, Face) computes the yield to maturity of bonds with stepped coupons given the price. The function supports any number of conversion dates.

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

---

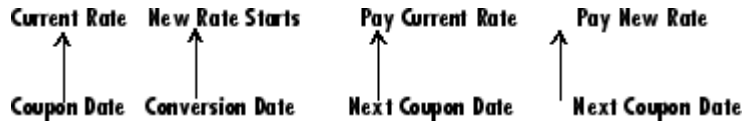
**Note** For bonds with fixed coupons, use bndyield. You will receive the error incorrect number of inputs if you use a fixed-coupon bond with stepcpnyield.

---

## Examples

Find the yield to maturity of three stepped-coupon bonds of known price, given three conversion scenarios:

- Bond A has two conversions, the first one falling on the settle date and immediately expiring.
- Bond B has three conversions, with conversion dates exactly on the coupon dates.
- Bond C has three conversions, with one or more conversion dates not on coupon dates. This case illustrates that only cash flows for full periods after conversion dates are affected, as illustrated below.



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

Bond A Dates	Bond A Rates	Bond B Dates	Bond B Rates	Bond C Dates	Bond C Rates
Settle (02-Aug-92)	7.5%	Settle (02-Aug-92)	7.5%	Settle (02-Aug-92)	7.5%
First Conversion (02-Aug-92)	8.875%	First Conversion (15-Jun-97)	8.875%	First Conversion (14-Jun-97)	8.875%
Second Conversion (15-Jun-03)	9.25%	Second Conversion (15-Jun-01)	9.25%	Second Conversion (14-Jun-01)	9.25%
Maturity (15-Jun-10)	NaN	Third Conversion (15-Jun-05)	10.0%	Third Conversion (14-Jun-05)	10.0%
		Maturity (15-Jun-10)	NaN	Maturity (15-Jun-10)	NaN

```
format long
Price = [117.3824; 113.4339; 113.4339];
Settle = datenum('02-Aug-1992');

ConvDates = [datenum('02-Aug-1992'), datenum('15-Jun-2003'), nan;
datenum('15-Jun-1997'), datenum('15-Jun-2001'), datenum('15-Jun-2005');
datenum('14-Jun-1997'), datenum('14-Jun-2001'), datenum('14-Jun-2005')];

Maturity = datenum('15-Jun-2010');

CouponRates = [0.075 0.08875 0.0925 nan];
```



```
0.075 0.08875 0.0925 0.1;  
0.075 0.08875 0.0925 0.1];  
Basis = 1;  
Period = 2;  
EndMonthRule = 1;  
Face = 100;  
  
Yield = stepcpnyield(Price, Settle, Maturity, ConvDates, ...  
CouponRates, Period, Basis, EndMonthRule, Face)  
  
Yield =  
  
0.07221440204915  
0.07221426780036  
0.07221426780036
```

## References

This function adheres to *SIA Fixed Income Securities Formulas for Price, Yield, and Accrued Interest*, Volume 1, 3rd edition, pp. 120 - 123, on zero-coupon instruments pricing.

## See Also

[bndprice](#) | [cdprice](#) | [stepcpncfamounts](#) | [stepcpnprice](#) | [tbillprice](#) | [zeroprice](#)

# tbilldisc2yield

---

**Purpose** Convert Treasury bill discount to equivalent yield

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

## Arguments

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

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

## Description

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

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

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

## Examples

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

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

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

```
BEYield =
```

0.0517

MMYield =

0.0510

## References

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

## See Also

[tbillyield2disc](#) | [zeroyield](#)

# tbillprice

---

**Purpose** Price Treasury bill

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

## Arguments

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

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

---

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

---

**Description** Price = tbillprice(Rate, Settle, Maturity, Type) computes the price of a Treasury bill given a yield or discount rate.

Price is an NTBILLS-by-1 vector of T-bill prices for every \$100 face.

**Examples** **Example 1.** Given a Treasury bill with these characteristics, compute the price of the Treasury bill using the bond-equivalent yield as input.

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

```
Maturity = '31-Mar-03';
Type = 2;
Price = tbillprice(Rate, Settle, Maturity, Type)
Price =
    97.8172
```

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

```
Rate = [0.045; 0.046];
Settle = {'02-Jan-02'; '01-Mar-02'};
Maturity = {'30-June-02'; '30-June-02'};
Type = [2 3];
Price = tbillprice(Rate, Settle, Maturity, Type)
Price =
    97.8408
    98.4539
```

## References

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

## See Also

`tbillyield` | `zeroprice`

**Purpose** Break-even discount of repurchase agreement

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

## Arguments

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

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

All dates must be in serial date number format.

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

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

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

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

```
TBEdiscount =
```

```
    0.0491
```

```
    0.0478
```

**References**

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

# tbillval01

---

**Purpose** Value of one basis point

**Syntax** [Val01Disc, Val01MMY, Val01BEY] = tbillval01(Settle, Maturity)

## Arguments

Settle	Settlement date of Treasury bills. Settle must be earlier than 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



Va101MMY =

0.0034

Va101BEY =

0.0033

**References**

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

**See Also**

tbilldisc2yield | tbillprice | tbillyield | tbillyield2disc

# tbillyield

---

**Purpose** Yield on Treasury bill

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

## Arguments

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

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

## Description

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

All outputs are NTBILLS-by-1 vectors.

---

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

---

## Examples

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

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

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

```
MMYield =
```

```
0.0252
```

```
BEYield =
```

```
0.0255
```

```
Discount =
```

```
0.0249
```

## References

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

## See Also

[tbilldisc2yield](#) | [tbillprice](#) | [tbillyield2disc](#) | [zeroyield](#)

# tbillyield2disc

---

**Purpose** Convert Treasury bill yield to equivalent discount

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

## Arguments

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

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

---

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

---

## Description

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

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

## Examples

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

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

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

Discount =

0.0485

Now recompute the discount on a bond-equivalent basis.

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

Discount =

0.0478

**References**

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

**See Also**

`tbilldisc2yield`

# tfutbyprice

---

**Purpose** Future prices of Treasury bonds given spot price

**Syntax** `QtdFutPrice = tfutbyprice(SpotCurve, Price, SettleFut, MatFut, ConvFactor, CouponRate, Maturity, Interpolation)`

**Arguments**

<code>SpotCurve</code>	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.
<code>Price</code>	Scalar or vector containing prices of Treasury bonds or notes per \$100 notional. Use <code>bndprice</code> for theoretical value of bond.
<code>SettleFut</code>	Scalar or vector of identical elements containing settlement date of futures contract.
<code>MatFut</code>	Scalar or vector containing maturity dates (or anticipated delivery dates) of futures contract.
<code>ConvFactor</code>	Conversion factor. See <code>convfactor</code> .
<code>CouponRate</code>	Scalar or vector containing underlying bond annual coupon in decimal.
<code>Maturity</code>	Scalar or vector containing underlying bond maturity.
<code>Interpolation</code>	(Optional) Interpolation method. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See <code>interp1</code> for more information.

Inputs (except `SpotCurve`) must either be a scalar or a vector of size equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

## Description

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

In addition, you can use the Fixed-Income Toolbox method `getZeroRates` for an `IRDataCurve` object with a `Dates` property to create a vector of dates and data acceptable for `tfutbyprice`. For more information, see “Converting an `IRDataCurve` or `IRFunctionCurve` Object” on page 6-25.

## Examples

Determine the future price of two Treasury bonds based upon a spot rate curve constructed from data for November 14, 2002.

```
% Constructing spot curve from Nov 14, data
Bonds = [datenum('02/13/2003'),      0;
         datenum('05/15/2003'),      0;
         datenum('10/31/2004'),      0.02125;
         datenum('11/15/2007'),      0.03;
         datenum('11/15/2012'),      0.04;
         datenum('02/15/2031'),      0.05375];

Yields = [1.20; 1.25; 1.86; 2.99; 4.02; 4.93]/100;

Settle = datenum('11/15/2002');

[ZeroRates, CurveDates] = ...
zbtyield(Bonds, Yields, Settle);

SpotCurve = [CurveDates, ZeroRates];

% Calculating a particular bond's future quoted price
RefDate   = [datenum('1-Dec-2002'); datenum('1-Mar-2003')];
MatFut    = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
Maturity  = [datenum('15-Aug-2009'); datenum('15-Aug-2010')];
CouponRate = [0.06;0.0575];
ConvFactor = convfactor(RefDate, Maturity, CouponRate);
Price = [114.416; 113.171];
```

# tfutbyprice

---

```
Interpolation = 1;
```

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

```
QtdFutPrice =
```

```
114.0409
```

```
113.4029
```

## See Also

[convfactor](#) | [tfutbyyield](#)



## Purpose

Future prices of Treasury bonds given current yield

## Syntax

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

## Arguments

<code>SpotCurve</code>	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.
<code>Yield</code>	Scalar or vector containing yield to maturity of bonds. Use <code>bndyield</code> for theoretical value of bond yield.
<code>SettleFut</code>	Scalar or vector of identical elements containing settlement date of futures contract.
<code>MatFut</code>	Scalar or vector containing maturity dates (or anticipated delivery dates) of futures contract.
<code>ConvFactor</code>	Conversion factor. See <code>convfactor</code> .
<code>CouponRate</code>	Scalar or vector containing underlying bond annual coupon in decimal.
<code>Maturity</code>	Scalar or vector containing underlying bond maturity.
<code>Interpolation</code>	(Optional) Interpolation method. Available methods are (0) nearest, (1) linear, and (2) cubic. Default = 1. See <code>interp1</code> for more information.

Inputs (except `SpotCurve`) must either be a scalar or a vector of size equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

## Description

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

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

In addition, you can use the Fixed-Income Toolbox method `getZeroRates` for an `IRDataCurve` object with a `Dates` property to create a vector of dates and data acceptable for `tfutbyyield`. For more information, see “Converting an `IRDataCurve` or `IRFunctionCurve` Object” on page 6-25.

## Examples

Determine the future price of two Treasury bonds based upon a spot rate curve constructed from data for November 14, 2002.

```
% Constructing spot curve from Nov 14, data
Bonds = [datenum('02/13/2003'),      0;
         datenum('05/15/2003'),      0;
         datenum('10/31/2004'),    0.02125;
         datenum('11/15/2007'),      0.03;
         datenum('11/15/2012'),      0.04;
         datenum('02/15/2031'),    0.05375];

Yields = [1.20; 1.25; 1.86; 2.99; 4.02; 4.93]/100;

Settle = datenum('11/15/2002');

[ZeroRates, CurveDates] = ...
zbtyield(Bonds, Yields, Settle);

SpotCurve = [CurveDates, ZeroRates];

% Calculating a particular bond's future quoted price
RefDate   = [datenum('1-Dec-2002'); datenum('1-Mar-2003')];
MatFut    = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
Maturity   = [datenum('15-Aug-2009'); datenum('15-Aug-2010')];
CouponRate = [0.06; 0.0575];
ConvFactor = convfactor(RefDate, Maturity, CouponRate);
Yield      = [0.03576; 0.03773];
Interpolation = 1;
```

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

```
QtdFutPrice =
```

```
114.0416
```

```
113.4034
```

## See Also

convfactor | tfutbyprice

# tfutimprepo

---

**Purpose** Implied repo rates for Treasury bond future given price

**Syntax** `ImpliedRepo = tfutimprepo(ReinvestData, Price, QtdFutPrice, Settle, MatFut, ConvFactor, CouponRate, Maturity)`

## Arguments

ReinvestData	Number of futures (NFUT) by 2 matrix of rates and bases for the reinvestment of intervening coupons in the form of [ReinvestRate ReinvestBasis]. ReinvestRate is the simple reinvestment rate, in decimal. Specify ReinvestBasis as 0 = not reinvested, 2 = actual/360, or 3 = actual/365.
Price	Current bond price per \$100 notional.
QtdFutPrice	Quoted bond futures price per \$100 notional.
Settle	Settlement/valuation date of futures contract.
MatFut	Maturity date (or anticipated delivery dates) of futures contract.
ConvFactor	Conversion factor. See convfactor.
CouponRate	Underlying bond annual coupon, in decimal.
Maturity	Underlying bond maturity date.

Inputs (except ReinvestData) must either be a scalar or a vector of size equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

**Description** `ImpliedRepo = tfutimprepo(ReinvestData, Price, QtdFutPrice, Settle, MatFut, ConvFactor, CouponRate, Maturity)` computes the implied repo rate that prevents arbitrage of Treasury bond futures, given the clean price at the settlement and delivery dates.

ImpliedRepo is the implied annual repo rate, in decimal, with an actual/360 basis.

## Examples

Compute the implied repo rate given the following set of data.

```
ReinvestData = [0.018 3];
Price = [114.4160; 113.1710];
QtdFutPrice = [114.1201; 113.7090];
Settle = datenum('11/15/2002');
MatFut = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
ConvFactor = [1; 0.9854];
CouponRate = [0.06; 0.0575];
Maturity = [datenum('15-Aug-2009'); datenum('15-Aug-2010')];

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

ImpliedRepo =

    0.0200
    0.0200
```

## See Also

tfutpricebyrepo | tfutyieldbyrepo

# tfutpricebyrepo

---

**Purpose** Implied repo rates given Treasury bond future price

**Syntax** [QtdFutPrice AccrInt] = tfutpricebyrepo(RepoData, ReinvestData, Price, Settle, MatFut, ConvFactor, CouponRate, Maturity)

**Arguments**

RepoData	Number of futures (NFUT) by 2 matrix of simple term repo/funding rates in decimal and their bases in the form of [RepoRate RepoBasis]. Specify RepoBasis as 2 = actual/360 or 3 = actual/365.
ReinvestData	Number of futures (NFUT) by 2 matrix of rates and bases for the reinvestment of intervening coupons in the form of [ReinvestRate ReinvestBasis]. ReinvestRate is the simple reinvestment rate, in decimal. Specify ReinvestBasis as 0 = not reinvested, 2 = actual/360, or 3 = actual/365.
Price	Quoted clean prices of Treasury bonds per \$100 notional at Settle.
Settle	Settlement/valuation date of futures contract.
MatFut	Maturity date (or anticipated delivery dates) of futures contract.
ConvFactor	Conversion factor. See convfactor.
CouponRate	Underlying bond annual coupon, in decimal.
Maturity	Underlying bond maturity date.

Inputs (except RepoData and ReinvestData) must either be a scalar or a vector of size equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.

## Description

[QtdFutPrice AccrInt] = tfutpricebyrepo(RepoData, ReinvestData, Price, Settle, MatFut, ConvFactor, CouponRate, Maturity) computes the theoretical futures bond price given the settlement price, the repo/funding rates, and the reinvestment rate.

QtdFutPrice is the quoted futures price, per \$100 notional.

AccrInt is the accrued interest due at the delivery date, per \$100 notional.

## Examples

Compute the quoted futures price and accrued interest due on the target delivery date, given the following data.

```
RepoData      = [0.020  2];
ReinvestData  = [0.018  3];
Price         = [114.416; 113.171];
Settle        = datenum('11/15/2002');
MatFut        = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
ConvFactor    = [1 ; 0.9854];
CouponRate    = [0.06;0.0575];
Maturity      = [datenum('15-Aug-2009'); datenum('15-Aug-2010')];
```

```
[QtdFutPrice AccrInt] = tfutpricebyrepo(RepoData, ...
ReinvestData, Price, Settle, MatFut, ConvFactor, CouponRate, ...
Maturity)
```

```
QtdFutPrice =
```

```
114.1201
113.7090
```

```
AccrInt =
```

```
1.9891
0.4448
```

## See Also

tfutimprepo | tfutyieldbyrepo

# tfutyieldbyrepo

---

**Purpose** Implied repo rates given Treasury bond future yield

**Syntax** `FwdYield = tfutyieldbyrepo(RepoData, ReinvestData, Yield, Settle, MatFut, ConvFactor, CouponRate, Maturity)`

## Arguments

<b>RepoData</b>	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.
<b>ReinvestData</b>	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.
<b>Yield</b>	Yield to maturity of Treasury bonds per \$100 notional at Settle.
<b>Settle</b>	Settlement/valuation date of futures contract.
<b>MatFut</b>	Maturity date (or anticipated delivery dates) of futures contract.
<b>ConvFactor</b>	Conversion factor. See convfactor.
<b>CouponRate</b>	Underlying bond annual coupon, in decimal.
<b>Maturity</b>	Underlying bond maturity date.

Inputs (except RepoData and ReinvestData) must either be a scalar or a vector of size equal to the number of Treasury futures (NFUT) by 1 or 1-by-NFUT.



## Description

FwdYield = tfutyieldbyrepo(RepoData, ReinvestData, Yield, Settle, MatFut, ConvFactor, CouponRate, Maturity) computes the theoretical futures bond yield given the settlement yield, the repo/funding rate, and the reinvestment rate.

FwdYield is the forward yield to maturity, in decimal, compounded semiannually.

## Examples

Compute the quoted futures bond yield, given the following data:

```
RepoData      = [0.020  2];
ReinvestData  = [0.018  3];
Yield         = [0.0215; 0.0257];
Settle        = datenum('11/15/2002');
MatFut        = [datenum('15-Dec-2002'); datenum('15-Mar-2003')];
ConvFactor    = [1; 0.9854];
CouponRate    = [0.06; 0.0575];
Maturity      = [datenum('15-Aug-2009'); datenum('15-Aug-2010')];
```

```
FwdYield = tfutyieldbyrepo(RepoData, ReinvestData, Yield,...
Settle, MatFut, ConvFactor, CouponRate, Maturity)
```

```
FwdYield =
```

```
    0.0221
    0.0282
```

## See Also

tfutimprepo | tfutpricebyrepo

# toRateSpec

---

**Purpose** Convert IRDataCurve object to RateSpec

**Class** @IRDataCurve

**Syntax** F = toratespec(CurveObj, InpDates)

**Arguments**

CurveObj	Interest-rate curve object that is constructed using IRDataCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.

**Description** F = toratespec(CurveObj, InpDates) returns a RateSpec object that is identical to the RateSpec structure created by the Financial Derivatives Toolbox function intenvset.

**Examples** This example creates an IRDataCurve object from the IRDataCurve constructor using Dates and Data and then is converted to a RateSpec structure using the toRateSpec method:

```
Data = [2.09 2.47 2.71 3.12 3.43 3.85 4.57 4.58]/100;
Dates = daysadd(today,[360 2*360 3*360 5*360 7*360 10*360 20*360 30*360],1);
irdc = IRDataCurve('Forward',today,Dates,Data)
irdc.toRateSpec(today+30:30:today+365)

irdc =

IRDataCurve handle

Properties:
    Dates: [8x1 double]
    Data: [8x1 double]
    InterpMethod: 'linear'
    Type: 'Forward'
```

```
Settle: 733596
Compounding: 2
Basis: 0

Methods, Events, Superclasses

ans =

    FinObj: 'RateSpec'
    Compounding: 2
        Disc: [12x1 double]
        Rates: [12x1 double]
        EndTimes: [12x1 double]
        StartTimes: [12x1 double]
        EndDates: [12x1 double]
        StartDates: 733596
    ValuationDate: 733596
        Basis: 0
        EndMonthRule: 1
```

## How To

- “@IRDataCurve” on page A-7

# toRateSpec

---

**Purpose** Convert IRFunctionCurve object to RateSpec

**Class** @IRFunctionCurve

**Syntax** F = toRateSpec(CurveObj, InpDates)

**Arguments**

CurveObj	Interest-rate curve object that is constructed using IRFunctionCurve.
InpDates	Vector of input dates using MATLAB date format. The input dates must be after the settle date.

**Description** F = toRateSpec(CurveObj, InpDates) returns a RateSpec object that is identical to the RateSpec structure created by the Financial Derivatives Toolbox function intenvset.

**Examples** This example creates an IRFunctionCurve object using the IRFunctionCurve constructor and then a RateSpec structure is created using the toRateSpec method:

```
irfc = IRFunctionCurve('Forward',today,@(t) polyval([-0.0001 0.003 0.02],t));  
irfc.toRateSpec(today+30:30:today+365)
```

```
ans =
```

```
    FinObj: 'RateSpec'  
Compounding: 2  
    Disc: [12x1 double]  
    Rates: [12x1 double]  
EndTimes: [12x1 double]  
StartTimes: [12x1 double]  
EndDates: [12x1 double]  
StartDates: 733596  
ValuationDate: 733596
```

Basis: 0  
EndMonthRule: 1

**How To**

- “@IRFunctionCurve” on page A-12

# zeroprice

---

**Purpose** Price zero-coupon instruments given yield

**Syntax** Price = zeroprice(Yield, Settle, Maturity, Period, Basis, EndMonthRule)

**Arguments**

Yield	Scalar or vector containing yield to maturity of instruments.
Settle	Settlement date. A vector of serial date numbers or date strings. Settle must be earlier than Maturity.
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Scalar or vector specifying number of quasi-coupons per year. Default = 2.
Basis	(Optional) Day-count basis of the bond. A vector of integers. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li></ul>

- 11 = 30/360E (ICMA)
- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**EndMonthRule** (Optional) End-of-month rule. A vector. This rule applies only when **Maturity** is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.

## Description

**Price** = `zeroprice(Yield, Settle, Maturity, Period, Basis, EndMonthRule)` calculates the prices for a portfolio of general short and long term zero-coupon instruments given the yield of the instruments. **Price** is a column vector containing a price for each zero-coupon instrument.

When there is less than one quasi-coupon, the function uses a simple yield based upon "Period times Number of Days in quasi coupon period" day-year. The default period is 2 and the default number of days is 180, which makes the user-supplied yield a simple yield on a 360-day year.

For longer term computations (more than one quasi-coupon), use the bond equivalent yield based upon present value (or compounding).

## Formulas

To compute the price when there is 1 or 0 quasi-coupon periods to redemption, `zeroprice` uses the formula

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

*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
<i>DSC</i>	Number of days from settlement date to next quasi-coupon date as if the security paid periodic interest.
<i>DSR</i>	Number of days from settlement date to the redemption date (call date, put date, and so on).
<i>E</i>	Number of days in quasi-coupon period.
<i>M</i>	Number of quasi-coupon periods per year (standard for the particular security involved).
<i>Nq</i>	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.
<i>Price</i>	Dollar price per \$100 par value.
<i>RV</i>	Redemption value.
<i>Y</i>	Annual yield (decimal) when held to redemption.

## Examples

**Example 1.** Compute the price of a short-term zero-coupon instrument.

```
Settle = '24-Jun-1993';
Maturity = '1-Nov-1993';
Period = 2;
```



```
Basis = 0;  
Yield = 0.04;  
  
Price = zeroprice(Yield, Settle, Maturity, Period, Basis)  
  
Price =  
  
    98.6066
```

**Example 2.** Compute the prices of a portfolio of two zero-coupon instruments, one short-term, and the other long-term.

```
Settle = '24-Jun-1993';  
Maturity = ['01-Nov-1993'; '15-Jan-2024'];  
Basis = [0; 1];  
Yield = [0.04; 0.1];  
  
Price = zeroprice(Yield, Settle, Maturity, [], Basis)  
  
Price =  
  
    98.6066  
     5.0697
```

## References

[1] Mayle, Jan. *Standard Securities Calculation Methods*. New York: Securities Industry Association, Inc. Vol. 1, 3rd ed., 1993, ISBN 1-882936-01-9. Vol. 2, 1994, ISBN 1-882936-02-7.

## See Also

bndprice | cdprice | tbillprice | zeroyield

# zeroyield

---

**Purpose** Yield of zero-coupon instruments given price

**Syntax** Yield = zeroyield(Price, Settle, Maturity, Period, Basis, EndMonthRule)

## Arguments

Price	Scalar or vector containing prices of instruments.
Settle	Settlement date. A vector of serial date numbers or date strings. <b>Settle</b> must be earlier than <b>Maturity</b> .
Maturity	Maturity date. A vector of serial date numbers or date strings.
Period	(Optional) Scalar or vector specifying number of quasi-coupons per year. Default = 2.
Basis	(Optional) Day-count basis of the bond. A vector of integers. <ul style="list-style-type: none"><li>• 0 = actual/actual (default)</li><li>• 1 = 30/360 (SIA)</li><li>• 2 = actual/360</li><li>• 3 = actual/365</li><li>• 4 = 30/360 (BMA)</li><li>• 5 = 30/360 (ISDA)</li><li>• 6 = 30/360 (European)</li><li>• 7 = actual/365 (Japanese)</li><li>• 8 = actual/actual (ICMA)</li><li>• 9 = actual/360 (ICMA)</li><li>• 10 = actual/365 (ICMA)</li><li>• 11 = 30/360E (ICMA)</li></ul>

- 12 = actual/actual (ISDA)
- 13 = BUS/252

For more information, see basis.

**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

$\text{Yield} = \text{zeroyield}(\text{Price}, \text{Settle}, \text{Maturity}, \text{Period}, \text{Basis}, \text{EndMonthRule})$  calculates the bond-equivalent yield for a portfolio of general short and long term zero-coupon instruments given the price of the instruments. **Yield** is a column vector containing a yield for each zero-coupon instrument.

When the maturity date is fewer than 182 days away and the basis is actual/365, the function uses a simple-interest algorithm. If maturity is more than 182 days away, the function uses present value calculations.

When the basis is actual/360, the simple interest algorithm gives the money-market yield for short (1 to 6 months to maturity) Treasury bills.

The present value algorithm always gives the bond equivalent yield of the zero-coupon instrument. The algorithm is equivalent to calling **bndyield** with the zero-coupon information within one basis point.

## Formulas

To compute the yield when there is zero or one quasi-coupon periods to redemption, **zeroyield** uses the formula

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

# zeroyield

*Quasi-coupon periods* are the coupon periods which would exist if the bond was paying interest at a rate other than zero. The first term calculates the yield on invested dollars. The second term converts this yield to a per annum basis.

When there is more than one quasi-coupon period to the redemption date, `zeroyield` uses the formula

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

The elements of the equations are defined as follows.

Variable Definition	
<i>DSC</i>	Number of days from the settlement date to next quasi-coupon date as if the security paid periodic interest.
<i>DSR</i>	Number of days from the settlement date to redemption date (call date, put date, and so on).
<i>E</i>	Number of days in quasi-coupon period.
<i>M</i>	Number of quasi-coupon periods per year (standard for the particular security involved).
<i>Nq</i>	Number of quasi-coupon periods between the settlement date and redemption date. If this number contains a fractional part, raise it to the next whole number.
<i>P</i>	Dollar price per \$100 par value.
<i>RV</i>	Redemption value.
<i>Yield</i>	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 =
```

# zeroyield

---

0.0804

## References

[1] Mayle, Jan. *Standard Securities Calculation Methods*. New York: Securities Industry Association, Inc. Vol. 1, 3rd ed., 1993, ISBN 1-882936-01-9. Vol. 2, 1994, ISBN 1-882936-02-7.

## See Also

bndyield | cdyield | tbillyield | zeroprice

# Class Reference

---

- “@IRBootstrapOptions” on page A-2
- “@IRCurve” on page A-4
- “@IRDataCurve” on page A-7
- “@IRFitOptions” on page A-10
- “@IRFunctionCurve” on page A-12

## @IRBootstrapOptions

Create specific options for bootstrapping an interest-rate curve object

In this section...
“Hierarchy” on page A-2
“Constructor” on page A-2
“Public Read-Only Properties” on page A-2
“Methods” on page A-3

### Hierarchy

Superclasses: None

Subclasses: None

### Constructor

IRBootstrapOptions

### Public Read-Only Properties

Name	Description
ConvexityAdjustment	<p>Controls the convexity adjustment to interest rate futures. This can be specified as a function handle that takes time to maturity as an input and returns a value which is ConvexityAdjustment. Alternatively, you can define ConvexityAdjustment as an N-by-1 vector of values, where N is the number of interest rate futures. In either case, the ConvexityAdjustment is subtracted from the futures rate.</p> <p>For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.</p>



## **Methods**

There are no methods.

## @IRCurve

Base abstract class for interest-rate curve objects

In this section...
“Hierarchy” on page A-4
“Description” on page A-4
“Constructor” on page A-4
“Public Read-Only Properties” on page A-4
“Methods” on page A-6

### Hierarchy

**Superclasses:** None

**Subclasses:** @IRDataCurve, @IRFunctionCurve

### Description

IRCurve is an abstract class; you cannot create instances of it directly. You can create IRDataCurve and IRFunctionCurve objects that are derived from this class.

### Constructor

@IRCurve is an abstract class. To construct an IRCurve object, use one of the subclass constructors, IRDataCurve or IRFunctionCurve.

### Public Read-Only Properties

Name	Description
Type	Type of interest-rate curve: zero, forward, or discount.
Settle	Scalar or column vector of settlement dates.

Name	Description
Compounding	<p>Scalar that sets the compounding frequency per year for the IRCurve object:</p> <ul style="list-style-type: none"> <li>• -1 = Continuous compounding</li> <li>• 1 = Annual compounding</li> <li>• 2 = Semiannual compounding (default)</li> <li>• 3 = Compounding three times per year</li> <li>• 4 = Quarterly compounding</li> <li>• 6 = Bimonthly compounding</li> <li>• 12 = Monthly compounding</li> </ul>
Basis	<p>Day-count basis of the interest-rate curve. A vector of integers.</p> <ul style="list-style-type: none"> <li>• 0 = actual/actual (default)</li> <li>• 1 = 30/360 (SIA)</li> <li>• 2 = actual/360</li> <li>• 3 = actual/365</li> <li>• 4 = 30/360 (BMA)</li> <li>• 5 = 30/360 (ISDA)</li> <li>• 6 = 30/360 (European)</li> <li>• 7 = actual/365 (Japanese)</li> <li>• 8 = actual/actual (ICMA)</li> <li>• 9 = actual/360 (ICMA)</li> <li>• 10 = actual/365 (ICMA)</li> <li>• 11 = 30/360E (ICMA)</li> <li>• 12 = actual/actual (ISDA)</li> <li>• 13 = BUS/252</li> </ul>

## Methods

Classes that inherit from the `IRCurve` abstract class must implement the following methods.

<b>Method</b>	<b>Description</b>
<code>getForwardRates</code>	Returns forward rates for input dates.
<code>getZeroRates</code>	Returns zero rates for input dates.
<code>getDiscountFactors</code>	Returns discount factors for input dates.
<code>getParYields</code>	Returns par yields for input dates.
<code>toRateSpec</code>	Converts to be a <code>RateSpec</code> object. This is identical to the <code>RateSpec</code> structure produced by the Financial Derivatives Toolbox function <code>intenvset</code> .

## @IRDataCurve

Represent interest-rate curve object based on vector of dates and data

### In this section...

“Hierarchy” on page A-7

“Description” on page A-7

“Constructor” on page A-7

“Public Read-Only Properties” on page A-8

“Methods” on page A-9

## Hierarchy

**Superclasses:** @IRCurve

**Subclasses:** None

## Description

IRDataCurve is a representation of an interest-rate curve object with dates and data. You can construct this object directly by specifying dates and corresponding interest rates or discount factors; alternatively, you can bootstrap the object from market data. After an interest-rate curve object is constructed, you can:

- Calculate forward and zero rates and determine par yields.
- Extract the discount factors.
- Convert to a RateSpec structure that is identical to the RateSpec structure produced by the Financial Derivatives Toolbox function `intenvset`.

## Constructor

IRDataCurve

## Public Read-Only Properties

Name	Description
Type	Type of interest-rate curve: zero, forward, or discount.
Settle	Scalar or column vector of settlement dates.
Compounding	<p>Scalar that sets the compounding frequency per year for the IRCurve object:</p> <ul style="list-style-type: none"> <li>• -1 = Continuous compounding</li> <li>• 1 = Annual compounding</li> <li>• 2 = Semiannual compounding (default)</li> <li>• 3 = Compounding three times per year</li> <li>• 4 = Quarterly compounding</li> <li>• 6 = Bimonthly compounding</li> <li>• 12 = Monthly compounding</li> </ul>
Basis	<p>Day-count basis of the financial curve. A vector of integers.</p> <ul style="list-style-type: none"> <li>• 0 = actual/actual (default)</li> <li>• 1 = 30/360 (SIA)</li> <li>• 2 = actual/360</li> <li>• 3 = actual/365</li> <li>• 4 = 30/360 (BMA)</li> <li>• 5 = 30/360 (ISDA)</li> <li>• 6 = 30/360 (European)</li> <li>• 7 = actual/365 (Japanese)</li> <li>• 8 = actual/actual (ICMA)</li> <li>• 9 = actual/360 (ICMA)</li> <li>• 10 = actual/365 (ICMA)</li> <li>• 11 = 30/360E (ICMA)</li> </ul>

Name	Description
	<ul style="list-style-type: none"> <li>• 12 = actual/actual (ISDA)</li> <li>• 13 = BUS/252</li> </ul>
Dates	Dates corresponding to rate data.
Data	Interest-rate data or discount factors for the curve object.
InterpMethod	Values are: <ul style="list-style-type: none"> <li>• 'linear' — Linear interpolation (default).</li> <li>• 'constant' — Piecewise constant interpolation.</li> <li>• 'pchip' — Piecewise cubic Hermite interpolation.</li> <li>• 'spline' — Cubic spline interpolation.</li> </ul>

## Methods

The following table contains links to methods with supporting reference pages, including examples.

Method	Description
<code>getForwardRates</code>	Returns forward rates for input dates.
<code>getZeroRates</code>	Returns zero rates for input dates.
<code>getDiscountFactors</code>	Returns discount factors for input dates.
<code>getParYields</code>	Returns par yields for input dates.
<code>toRateSpec</code>	Converts to be a <code>RateSpec</code> object. This structure is identical to the <code>RateSpec</code>
<code>bootstrap</code>	Bootstraps an interest rate curve from market data.

## @IRFitOptions

Object to specify fitting options for an IRFunctionCurve interest-rate curve object

In this section...
“Hierarchy” on page A-10
“Description” on page A-10
“Constructor” on page A-10
“Public Read-Only Properties” on page A-11
“Methods” on page A-11

### Hierarchy

**Superclasses:** None

**Subclasses:** None

### Description

The IRFitOptions object allows you to specify options relating to the fitting process for an IRFunctionCurve object. Input arguments are specified in parameter/value pairs. The IRFitOptions structure provides the capability to choose which quantity to be minimized and other optimization parameters.

### Constructor

IRFitOptions



## Public Read-Only Properties

Name	Description
FitType	Price, Yield, or DurationWeightedPrice determines which is minimized in the curve fitting process. DurationWeightedPrice is the default.
InitialGuess	Initial guess for the parameters of the curve function.
UpperBound	Upper bound for the parameters of the curve function.
LowerBound	Lower bound for the parameters of the curve function.
OptOptions	Optimization structure based on the output from the Optimization Toolbox function <code>optimset</code> . This optimization structure is evaluated by <code>lsqnonlin</code> .

## Methods

There are no methods.

## @IRFunctionCurve

Represent an interest-rate curve object using a function

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

### Hierarchy

**Superclasses:** @IRCurve

**Subclasses:** None

### Description

IRFunctionCurve is a representation of an interest-rate curve object. You can construct this object directly by specifying a function handle or a function can be fit to market data using methods of the object. After an interest-rate curve object is constructed; you can:

- Calculate forward and zero rates and determine par yields.
- Extract the discount factors.
- Convert to a RateSpec structure; this is identical to the RateSpec structure produced by the Financial Derivatives Toolbox function `intenvset`.

### Constructor

IRFunctionCurve

## Public Read-Only Properties

Name	Description
Type	Type of interest-rate curve: zero, forward, or discount.
Settle	Scalar or column vector of settlement dates.
Compounding	<p>Scalar that sets the compounding frequency per year for the IRCurve object:</p> <ul style="list-style-type: none"> <li>• -1 = Continuous compounding</li> <li>• 1 = Annual compounding</li> <li>• 2 = Semiannual compounding (default)</li> <li>• 3 = Compounding three times per year</li> <li>• 4 = Quarterly compounding</li> <li>• 6 = Bimonthly compounding</li> <li>• 12 = Monthly compounding</li> </ul>
Basis	<p>Day-count basis of the interest-rate curve. A vector of integers.</p> <ul style="list-style-type: none"> <li>• 0 = actual/actual (default)</li> <li>• 1 = 30/360 (SIA)</li> <li>• 2 = actual/360</li> <li>• 3 = actual/365</li> <li>• 4 = 30/360 (BMA)</li> <li>• 5 = 30/360 (ISDA)</li> <li>• 6 = 30/360 (European)</li> <li>• 7 = actual/365 (Japanese)</li> <li>• 8 = actual/actual (ICMA)</li> <li>• 9 = actual/360 (ICMA)</li> <li>• 10 = actual/365 (ICMA)</li> </ul>

Name	Description
	<ul style="list-style-type: none"> <li>• 11 = 30/360E (ICMA)</li> <li>• 12 = actual/actual (ISDA)</li> <li>• 13 = BUS/252</li> </ul>
FunctionHandle	Function handle that defines the interest-rate curve. For more information on defining a function handle, see the MATLAB Programming Fundamentals documentation.

## Methods

The following table contains links to methods with supporting reference pages, including examples.

Method	Description
getForwardRates	Returns forward rates for input dates.
getZeroRates	Returns zero rates for input dates.
getDiscountFactors	Returns discount factors for input dates.
getParYields	Returns par yields for input dates.
toRateSpec	Converts to be a RateSpec object. This is identical to the RateSpec structure
fitSvensson	Fits a Svensson function to market data.
fitNelsonSiegel	Fits a Nelson-Siegel function to market data.
fitSmoothingSpline	Fits a smoothing spline function to market data.
fitFunction	Fits a custom function to market data.

# Bibliography

---

- “Fitting Interest-Rate Curve Functions” on page B-2
- “Bootstrapping a Swap Curve” on page B-3
- “Bond Futures” on page B-4
- “Credit Derivatives” on page B-5

## Fitting Interest-Rate Curve Functions

Nelson, C.R., Siegel, A.F., "Parsimonious modelling of yield curves," *Journal of Business*, Number 60, 1987, pp 473-89.

Svensson, L.E.O., "Estimating and interpreting forward interest rates: Sweden 1992-4," International Monetary Fund, IMF Working Paper, 1994, p. 114.

Fisher, M., Nychka, D., Zervos, D., "Fitting the term structure of interest rates with smoothing splines," Board of Governors of the Federal Reserve System, Federal Reserve Board Working Paper, 1995.

Anderson, N., Sleath, J., "New estimates of the UK real and nominal yield curves," *Bank of England Quarterly Bulletin*, November, 1999, pp 384-92.

Waggoner, D., "Spline Methods for Extracting Interest Rate Curves from Coupon Bond Prices," Federal Reserve Board Working Paper, 1997, p. 10.

"Zero-coupon yield curves: technical documentation," *BIS Papers*, Bank for International Settlements, Number 25, October, 2005.

Bolder, D.J., Gusba, S., "Exponentials, Polynomials, and Fourier Series: More Yield Curve Modelling at the Bank of Canada," *Working Papers*, Bank of Canada, 2002, p. 29.

Bolder, D.J., Streliski, D., "Yield Curve Modelling at the Bank of Canada," *Technical Reports*, Number 84, 1999, Bank of Canada.

## **Bootstrapping a Swap Curve**

Hagan, P., West, G., "Interpolation Methods for Curve Construction," *Applied Mathematical Finance*, Vol. 13, Number 2, 2006.

Ron, Uri, "A Practical Guide to Swap Curve Construction," *Working Papers*, Bank of Canada, 2000, p. 17.

## **Bond Futures**

Burghardt, G., T. Belton, M. Lane, and J. Papa, *The Treasury Bond Basis*, McGraw-Hill, 2005.

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



## Credit Derivatives

Beumee, J., D. Brigo, D. Schiemert, and G. Stoye. "Charting a Course Through the CDS Big Bang," *Fitch Solutions, Quantitative Research*, Global Special Report. April 7, 2009.

Hull, J., and A. White, "Valuing Credit Default Swaps I: No Counterparty Default Risk," *Journal of Derivatives*8, 29-40.

O'Kane, D. and S. Turnbull, "Valuation of Credit Default Swaps." *Lehman Brothers, Fixed Income Quantitative Credit Research*, April, 2003.



# Examples

---

Use this list to find examples in the documentation.

## **Agency Option Adjusted Spreads**

“Computing the Agency OAS for Bonds” on page 3-3

## **Treasury Bills**

“Treasury Bill Repurchase Agreements” on page 3-8

“Treasury Bill Yields” on page 3-10

## **Using Zero-Coupon Bonds**

“Pricing Treasury Notes” on page 3-13

“Pricing Corporate Bonds” on page 3-15

## **Stepped-Coupon Bonds**

“Cash Flows from Stepped-Coupon Bonds” on page 3-17

“Price and Yield of Stepped-Coupon Bonds” on page 3-19

## **Pricing and Hedging**

“Swap Pricing Example” on page 4-3



## **Bond Futures**

“Example Analysis of Bond Futures” on page 4-14

## **Credit Default Swaps**

“Bootstrapping a Default Probability Curve” on page 5-2

“Finding the Breakeven Spread for a New CDS Contract” on page 5-5

“Valuing an Existing CDS Contract” on page 5-8

“Converting from Running to Upfront and Vice Versa” on page 5-10

“Bootstrapping from Inverted Market Curves” on page 5-13

**American option**

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

**amortization**

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

**annuity**

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

**arbitrage**

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

**basis point**

One hundredth of one percentage point, or 0.0001.

**beta**

The price volatility of a financial instrument relative to the price volatility of a market or index as a whole. Beta is most commonly used with respect to equities. A high-beta instrument is riskier than a low-beta instrument.

**binomial model**

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

**Black-Scholes model**

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

**Bollinger band chart**

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

**bootstrapping, bootstrap method**

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

**building a binomial tree**

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

**call**

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

**callable bond**

A bond that allows the issuer to buy back the bond at a predetermined price at specified future dates. The bond contains an embedded call option; that is, the holder has sold a call option to the issuer. See also **puttable bond** on page Glossary-10.

**cap**

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

**caplet**

A cap that is guaranteed for one particular date.

**cash flow**

Cash received and paid over time.

**cheapest to deliver**

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

**clean price**

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

**collar**

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

**conditional prepayment rate (CPR)**

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

**conversion factor**

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

**convexity**

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

**correlation**

The simultaneous change in value of two random numeric variables.

**correlation coefficient**

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

**coupon**

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

**coupon dates**

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

**coupon rate**

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

**covariance**

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

**credit default swap (CDS)**

The buyer of a credit default swap receives credit protection, whereas the seller of the credit default swap guarantees the credit worthiness of the product. By doing this, the risk of default is transferred from the holder of the fixed-income security to the seller of the credit default swap.

**delta**

The rate of change of the price of a derivative security relative to the price of the underlying asset; that is, the first derivative of the curve that relates the price of the derivative to the price of the underlying security.

**depreciation**

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

**derivative**

A financial instrument that is based on some underlying asset. For example, an option is a derivative instrument based on the right to buy or sell an underlying instrument.

**dirty price**

The price of a bond including the accrued interest.

**discount curve**

The curve of discount rates vs. maturity dates.

**duration**

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

**efficient frontier**

A graph representing a set of portfolios that maximizes expected return at each level of portfolio risk. See also **Markowitz model** on page Glossary-8.

**elasticity**

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

**Eurodollar**

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

**European option**

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

**exercise price**

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

**face value**

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

**fixed-income security**

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

**floor**

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

**forward curve**

The curve of forward interest rates vs. maturity dates.

**forward rate**

The future interest rate of a bond inferred from the term structure, especially from the yield curve of zero-coupon bonds, calculated from the growth factor of an investment in a zero held until maturity.

**forward rate agreement (FRA)**

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

**future value**

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

**gamma**

The rate of change of delta for a derivative security relative to the price of the underlying asset; that is, the second derivative of the option price relative to the security price.

**Greeks**

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

**hedge**

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

**implied repo rate**

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



**implied volatility**

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

**internal rate of return**

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

**issue date**

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

**lambda**

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

**LIBOR**

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

**long position**

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

**long rate**

The yield on a zero-coupon Treasury bond.

**Macaulay duration**

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

**Markowitz model**

A model for selecting an optimum investment portfolio, devised by H. M. Markowitz. It uses a discrete-time, continuous-outcome approach for modeling investment problems, often called the mean-variance paradigm. See also **efficient frontier** on page Glossary-5.

**maturity date**

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

**mean**

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

**modified duration**

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

**Monte-Carlo simulation**

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

**moving average**

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

**moving-averages chart**

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

**Nelson-Siegel model**

A model that fits the empirical form of the yield curve with a prespecified functional form of the spot rates, which is a function of the time to maturity of the bonds.

**normal (bell-shaped) distribution**

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

**notional**

The nominal value used to calculate swap payments.

**odd first or last period**

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

**off-the-run**

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

**on-the-run**

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

**option**

A right to buy or sell specific securities or commodities at a stated price (exercise or strike price) within a specified time. An option is a type of derivative.

**option-adjusted spread**

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

**pass-through**

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

**par value**

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

**par yield curve**

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

**piecewise constant interpolation**

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

**present value**

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

**principal value**

See **par value** on page Glossary-10.

**purchase price**

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

**put**

An option to sell a stipulated amount of stock or securities within a specified time and at a fixed exercise price. See also **call** on page Glossary-2.

**puttable bond**

A bond that allows the holder to redeem the bond at a predetermined price at specified future dates. The bond contains an embedded put option; that is, the holder has bought a put option. See also **callable bond** on page Glossary-2.

**redemption value**

See **par value** on page Glossary-10.

**regression analysis**

Statistical analysis techniques that quantify the relationship between two or more variables. The intent is quantitative prediction or

forecasting, particularly using a small population to forecast the behavior of a large population.

**rho**

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

**repo rate**

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

**running**

The breakeven, or running spread is the premium a protection buyer needs to pay, with no upfront payments involved, to receive protection for credit events associated to a given reference entity.

**sensitivity**

The "what if" relationship between variables; the degree to which changes in one variable cause changes in another variable. A specific synonym is volatility.

**settlement date**

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

**short rate**

The annualized one-period interest rate.

**short sale, short position**

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

**single monthly mortality (SMM)**

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

**smoothing spline**

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

**spot curve, spot yield curve**

See **zero curve, zero-coupon yield curve** on page Glossary-15.

**spot rate**

The current interest rate appropriate for discounting a cash flow of some given maturity.

**spread**

For options, a combination of call or put options on the same stock with differing exercise prices or maturity dates.

**standard deviation**

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

**stochastic**

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

**straddle**

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

**strike**

Exercise a put or call option.

**strike price**

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

**Svensson model**

Extends the Nelson-Siegel model by adding a further term that allows for a second "hump." The extra precision is achieved by adding two

more parameters,  $\beta_3$  and  $\tau_2$ , which have to be estimated. See also **Nelson-Siegel model** on page Glossary-8.

**swap**

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

**swap option**

A swap option; an option on an interest-rate swap. The option gives the holder the right to enter into a contracted interest-rate swap at a specified future date. See also **swap** on page Glossary-13.

**tenor**

Life of a swap.

**term repo rate**

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

**term structure**

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

**theta**

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

**Treasury bill**

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

**Treasury bond**

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

**upfront**

The upfront of the contract is the current value expressed as a fraction of the notional amount of the contract, and it is commonly used to quote market values.

**variance**

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

**vega**

The rate of change in the price of a derivative security relative to the volatility of the underlying security. When vega is large the security is sensitive to small changes in volatility.

**volatility**

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

**yield**

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

**yield curve**

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

**yield to maturity**

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



**zero curve, zero-coupon yield curve**

A yield curve for zero-coupon bonds; zero rates versus maturity dates. Since the maturity and duration (Macaulay duration) are identical for zeros, the zero curve is a pure depiction of supply/demand conditions for loanable funds across a continuum of durations and maturities. Also known as spot curve or spot yield curve.

**zero-coupon bond, or zero**

A bond that, instead of carrying a coupon, is sold at a discount from its face value, pays no interest during its life, and pays the principal only at maturity.



## A

actual/360 3-7  
Agency Option-Adjusted Spreads (AOAS)  
    defined 3-2  
agencyoas 8-2  
agencyprice 8-8

## B

bkcall 8-14  
bkcaplet 8-20  
bkfloorlet 8-23  
bkput 8-26  
bndfutimprepo 8-33  
bndfutprice 8-39  
bond equivalent yield 8-282  
bond futures 4-12  
    example analysis 4-14  
bootstrap (IRDataCurve) 8-45  
break-even discount rate 3-8

## C

cbprice 8-53  
cdai 8-62  
cdprice 8-64  
cdsbootstrap 8-67  
cdsoptprice 8-74  
cdsprice 8-80  
cdsspread 8-88  
cdyield 8-95  
cheapest to deliver (CTD) 4-15  
CMO 2-17  
CMO example 2-32  
cmosched 8-97  
cmoschedcf 8-104  
cmoseqcf 8-109  
conditional prepayment rate (CPR) 2-4  
conversion factors 4-14  
convertible bond 4-10

convfactor 8-114  
coupon bond functions 3-12  
CPR  
    (conditional payment rate) 2-4  
Credit Default Swap (CDS)  
    defined 5-2  
CTD  
    (cheapest to deliver) 4-15

## D

discount security 3-7  
duration  
    modified 2-8

## E

effective duration 2-10  
    defined mathematically 2-10

## F

fitFunction (IRFunctionCurve) 8-118  
fitNelsonSiegel (IRFunctionCurve) 8-125  
fitSmoothingSpline  
    (IRFunctionCurve) 8-131  
fitSvensson (IRFunctionCurve) 8-137  
forward rate agreement 8-178  
    defined 8-182

## G

getDiscountFactors  
    (IRFunctionCurve) 8-145  
getDiscountFactors(IRDataCurve) 8-143  
getForwardRates (IRDataCurve) 8-147  
getForwardRates (IRFunctionCurve) 8-150  
getParYields (IRDataCurve) 8-153  
getParYields (IRFunctionCurve) 8-156  
getZeroRates (IRDataCurve) 8-159  
getZeroRates (IRFunctionCurve) 8-162

**I**

- implied repo 4-15
- interest-rate curve objects
  - class objects 6-2
  - creating 6-4
  - workflow 6-3
- IRBootstrapOptions 8-165
- IRDataCurve 8-166
  - bootstrapping 6-7
  - constructor 6-6
  - converting to RateSpec 6-25
- IRFitOptions 8-170
- IRFunctionCurve 8-172
  - converting to RateSpec 6-25
  - customizing using fitFunction 6-21
  - using function handle 6-13
  - using Nelson-Siegel model 6-14
  - using smoothing spline model 6-18
  - using Svensson model 6-16

**L**

- liborduration 8-176
- liborfloat2fixed 8-178
- liborprice 8-182

**M**

- mbscfamounts 8-186
- mbsconvp 8-248
- mbsconvy 8-250
- mbsdurp 8-252
- mbsdury 8-255
- mbsnoprepay 8-257
- mbsoas2price 8-259
- mbsoas2yield 8-263
- mbspassthrough 8-267
- mbsprice 8-269
- mbsprice2oas 8-272
- mbsprice2speed 8-276

- mbswal 8-279
- mbsyield 8-281
- mbsyield2oas 8-284
- mbsyield2speed 8-288
- modified duration 2-8
- mortgage yield 8-282
- mortgage-backed securities 2-2

**O**

- OAS
  - (option-adjusted spread) 2-9
- off-the-run 3-20
- on-the-run 3-20
- option-adjusted spread
  - defined 2-10
- option-adjusted spread (OAS) 2-9
  - effect on pool pricing 2-10

**P**

- pass-through certificate 2-2
- prepayment 2-3
- prepayment summary 2-16
- psaspeed2default 8-291
- psaspeed2rate 8-292
- Public Securities Association (PSA) 2-3
- PVBP 4-16

**Q**

- quasi-coupon periods
  - zeroprice 8-340
  - zeroyield 8-344

**S**

- seasoned prepayment vector 2-13
- single monthly mortality (SMM) rate 2-4
- SMM
  - single monthly mortality rate 2-4

spread 3-20  
    term structure of 3-20  
stepcpncfamounts 8-294  
stepcpnprice 8-300  
stepcpnyield 8-305

## **T**

tbilldisc2yield 8-310  
tbillprice 8-312  
tbillrepo 8-314  
tbillval01 8-316  
tbillyield 8-318  
tbillyield2disc 8-320  
tenor 8-176  
tfutbyprice 8-322  
tfutbyyield 8-325

tfutimrepo 8-328  
tfutpricebyrepo 8-330  
toRateSpec (IRDataCurve) 8-334  
toRateSpec (IRFunctionCurve) 8-336  
Treasury bills  
    defined 3-7  
Treasury bonds 3-7  
Treasury notes 3-7

## **Z**

zero-coupon bond  
    defined 3-12  
    quality of measurement 3-12  
zeroprice 8-338  
zeroyield 8-342