Financial Toolbox

For Use with MATLAB®

Computation

Visualization

Programming

The MATH WORKS Inc.

User's Guide

Version 2

How to Contact The MathWorks:

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc.
24 Prime Park Way
Natick, MA 01760-1500 Mail

`http://www.mathworks.com` Web
`ftp.mathworks.com` Anonymous FTP server
`comp.soft-sys.matlab` Newsgroup

`support@mathworks.com` Technical support
`suggest@mathworks.com` Product enhancement suggestions
`bugs@mathworks.com` Bug reports
`doc@mathworks.com` Documentation error reports
`subscribe@mathworks.com` Subscribing user registration
`service@mathworks.com` Order status, license renewals, passcodes
`info@mathworks.com` Sales, pricing, and general information
`finance@mathworks.com` Financial products information

Financial Toolbox User's Guide

Printing History:

September 1995	Preliminary version	New for 1.0
October 1995	First printing	Revised for 1.0
January 1998	Second printing	Revised for 1.1
January 1999	Third printing	Revised for 2.0 (Release 11)

Preface

Before You Begin

Introducing the Financial Toolbox

MATLAB® and the Financial Toolbox provide a complete integrated computing environment for financial analysis and engineering. With 122 financial functions, including 42 functions for manipulating dates and time, the toolbox has everything you need to perform mathematical and statistical analysis of financial data and display the results with presentation-quality graphics. You can quickly ask, visualize, and answer complicated questions.

In traditional or spreadsheet programming you must deal with all sorts of housekeeping details: declaring, data typing, sizing, etc. MATLAB does all that for you. You just write expressions the way you think of problems. And there's no need to switch tools, convert files, or rewrite applications.

Setting Your Expectations

After reading this manual, you will understand Financial Toolbox concepts, content, functions, and uses. You will have successfully executed several examples, and you will be able to use the functions of choice.

Your Tasks

With MATLAB and the Financial Toolbox, you can:

- Compute and analyze prices, yields, and sensitivities for bonds, derivatives, and other securities, and for portfolios of securities
- Analyze or manage portfolios
- Design and evaluate hedging strategies
- Identify, measure, and control risk
- Analyze and compute cash flows, including rates of return and depreciation streams
- Analyze and predict economic activity
- Create structured financial instruments, including foreign-exchange instruments
- Teach or conduct academic research

Your Background

In designing the Financial Toolbox and this manual, we assume your title is similar to one of these:

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

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

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

Prerequisites

If MATLAB is new to you, please read *Getting Started with MATLAB* first. The Financial Toolbox also requires the Statistics and Optimization Toolboxes, but you need not read those manuals before reading this one. Some examples use functions in the Spline Toolbox, but that toolbox is not a prerequisite for the Financial Toolbox.

Installing the Financial Toolbox

To install the Financial Toolbox see the *MATLAB Installation Guide* for your computer system. Please also see any Financial Toolbox Release Notes. On some systems the Financial Toolbox may already be installed. If so, there will be a `finance` directory under the MATLAB `toolbox` directory.

Please note that the Statistics Toolbox and the Optimization Toolbox are prerequisites for the Financial Toolbox. Install MATLAB, the Statistics Toolbox, and the Optimization Toolbox when you install the Financial Toolbox.

Using MATLAB and the Toolbox

Starting and Stopping MATLAB

See "Starting MATLAB" in *Getting Started with MATLAB* or *Using MATLAB* for instructions on starting MATLAB. No additional steps are necessary to start the Financial Toolbox. If MATLAB is running, the toolboxes are available.

To stop MATLAB, type `quit` or `exit`, or click on **Exit MATLAB** in the **File** menu. Remember to save your workspace and variables first if you want them for later use.

Working with MATLAB

These reminders will help you use MATLAB efficiently:

- MATLAB is an interpreter. It executes entered commands as soon as you press the **Return** or **Enter** key.
- If the last character of a statement is a semicolon (;) it suppresses the display but still carries out the command.
- MATLAB variable names are case sensitive in all computer environments. For example, `bonds` and `Bonds` are different variables. On PCs and Macs, function names are not case sensitive. `PRBOND` and `prbond` call the same function.
- Variable names must begin with a letter. They can include any number of letters, numbers, or underscores. MATLAB remembers only the first 31 characters, however.

- To enter a comment line begin it with a percent sign (%), or use the percent sign to add comments to a line. MATLAB does not execute the remainder of the line.
- You can use spaces or tabs to separate variables in a command line. MATLAB ignores the separators, but they can help you format your expressions for readability. If a statement won't fit on one line, use an ellipsis (`...`) before pressing **Return** to tell MATLAB that the expression continues on the next line.
- MATLAB's built-in command-line editor is very useful for editing your commands. For example, you can recall the previous line by pressing the **Up-Arrow** key. See "Command-Line Editing" in *Getting Started with MATLAB* or *Using MATLAB*.
- If you are building a complicated script containing many commands, it helps to enter and test them as you go along rather than trying to debug the whole thing at once.
- You may prefer to create MATLAB command (script) files with your favorite ASCII text editor (e.g., Notepad in Windows). Save the text file as *`name`*`.m` (with the `.m` extension) in the MATLAB directory (e.g., `C:\MATLAB`) or anywhere in the MATLAB search path. Then in the MATLAB command window, type *`name`* to execute the commands in the file.
- The `format` command is useful for managing numeric output display. Type `help format` for full information. The commands `format bank`, `format short`, `format short g`, and `format long g` are especially useful for financial applications.
- The `clear` command is useful for managing your MATLAB workspace. You can clear all variables or just specific ones. Type `help clear` for full information.

Using Help

MATLAB provides extensive online help for functions and for your workspace. See "Help and Online Documentation" in *Getting Started with MATLAB* or *Using MATLAB*. Here are some reminders:

- Type `help function` for an online explanation of any MATLAB function, including those in the Financial Toolbox. Type `help help` for more information on using `help`.
- Type `who` to see a list of current variables in your workspace.
- Type `whos` to see sizes and other information for the current variables.
- Type `what` for a list of M-files, MAT-files, and MEX-files in the current working directory. Type `help what` for full information.
- Type `which` *`function`* to see the full pathname of the specified function. Type `help which` for full information.

Financial Demonstration Programs

The MATLAB Financial Toolbox Exposition ships with a large number of programs that illustrate many of the features of the toolbox, including charting, options pricing, portfolio analysis, and others. Type `help findemos` for a complete list of available financial demonstration programs.

Finding Additional Information

For additional information about MathWorks financial products and the Financial Toolbox, please visit our Web site

```
http://www.mathworks.com/finprod
```

or send e-mail to:

```
finance@mathworks.com
```

Using This Guide

Organization

Chapter 1: Tutorial Helps you learn to use MATLAB and the Financial Toolbox for financial analysis and engineering applications. It reviews key definitions and matrix algebra fundamentals in a financial context. It then

discusses the tools and their uses by tasks. The last two major sections show how the toolbox solves real-world financial problems and how it produces presentation-quality graphics. If you are already familiar with MATLAB, matrix algebra, and financial computations, you can turn immediately to those examples.

Chapter 2: Reference Groups the Financial Toolbox functions by task, then describes each toolbox function in alphabetical order. Purpose, syntax, description, examples, and related functions are given for each function.

Appendix A: Glossary Defines the financial terms used in this manual.

Appendix B: Bibliography Provides references for the toolbox formulas and concepts.

Examples

We encourage you to try the examples throughout the text. In addition, we provide M-files of the longer sample problems and graphics examples in the *Tutorial*. The five sample problems are `ftspex1.m` through `ftspex5.m` plus `discfunc.m`; the three graphics examples are `ftgex1.m` through `ftgex3.m`.

Typographic Conventions

This manual uses the standard MathWorks typographic conventions.

`Monospace`	Commands, function names, code fragments, and screen displays; for example, `prbond`.
`Monospace italics`	An item for which you must supply a value; for example, `which` *`function`*.
Italics	Book titles, chapter titles, and mathematical notation.
Bold Initial Caps	Key names, menu names, and items that are selected from menus; for example, the **Enter** key.

Contents

Preface

1 Tutorial

2 Reference

A Glossary

B Bibliography

1

Tutorial

This chapter helps you begin using the MATLAB Financial Toolbox for financial analysis and engineering applications. It contains these sections:

- Getting Started

  Reviews key definitions and some matrix algebra fundamentals, and provides a brief refresher on using matrix functions in financial analysis and engineering.

- Understanding the Financial Toolbox

  Explains name and argument conventions. Discusses the tools and their uses by tasks: handling and converting dates; formatting currency and charting financial data; pricing and computing yields, and analyzing the term structure for fixed-income securities; evaluating and computing cash flows including depreciation streams; analyzing portfolios; and analyzing derivatives using European and American option models.

- Solving Sample Problems with the Toolbox

  Shows how the toolbox solves real-world financial problems: analyzing sensitivity of bond and portfolio prices to interest-rate changes, hedging a bond portfolio and an option portfolio, and generating yield curves.

- Producing Graphics with the Toolbox

  Shows how the toolbox produces presentation-quality graphics: plotting an efficient frontier, plotting option sensitivities, and plotting option portfolio sensitivities.

Getting Started

If you are new to MATLAB please read *Getting Started with MATLAB*. Start MATLAB and try the commands and examples there and in this tutorial. This material explains some MATLAB concepts and operations using financial examples to help get you started.

Using Matrix Functions for Finance

Many financial analysis procedures involve *sets* of numbers; for example, a portfolio of securities at various prices and yields. Matrices, matrix functions, and matrix algebra are the most efficient ways to analyze sets of numbers and their relationships. Spreadsheets focus on individual cells and the relationships between cells. While you can think of a set of spreadsheet cells (a range of rows and columns) as a matrix, a matrix-oriented tool like MATLAB manipulates sets of numbers more quickly, easily, and naturally.

Key Definitions

Matrix. A rectangular array of numeric or algebraic quantities subject to mathematical operations; the regular formation of elements into rows and columns. Described as an "*m*-by-*n*" matrix, with *m* the number of rows and *n* the number of columns. The description is always "row-by-column." For example, here is a 2-by-3 matrix of two bonds (the rows) with different par values, coupon rates, and coupon payment frequencies per year (the columns) entered using MATLAB notation:

```
bonds = [1000   0.06   2
          500   0.055  4]
```

Vector. A matrix with only one row or column. Described as a "1-by-*n*" or "*m*-by-1" matrix. The description is always "row-by-column." Here is a 1-by-4 vector of cash flows in MATLAB notation:

```
cash = [1500    4470    5280    -1299]
```

Scalar. A 1-by-1 matrix; i.e., a single number.

Referencing Matrix Elements

To reference specific matrix elements use (row, column) notation. For example,

```
bonds(1,2)
ans =
          0.06

cash(3)
ans =
       5280.00
```

You can enlarge matrices using small matrices or vectors as elements. For example,

```
addbond = [1000   0.065   2];
bonds = [bonds; addbond]
```

adds another row to the matrix and creates

```
bonds =
       1000   0.06    2
        500   0.055   4
       1000   0.065   2
```

Likewise,

```
prices = [987.50
          475.00
          995.00]
bonds = [prices, bonds]
```

adds another column and creates

```
bonds =
    987.50   1000   0.06    2
    475.00    500   0.055   4
    995.00   1000   0.065   2
```

Finally, the colon (:) is important in generating and referencing matrix elements. For example, to reference the par value, coupon rate, and coupon frequency of the second bond:

```
bond_items = bonds(2, 2:4)
bond_items =
    500.00   0.055   4
```

See *Getting Started with MATLAB* and *Using MATLAB* for additional information on generating and referencing matrices.

Transposing Matrices

Sometimes matrices are in the wrong configuration for an operation. In MATLAB, the apostrophe or prime character (') transposes a matrix: columns become rows, rows become columns. For example,

```
cash = [1500    4470    5280    -1299]'
```

produces

```
cash =
        1500
        4470
        5280
       -1299
```

Matrix Algebra Refresher

Matrix algebra and matrix operations are fundamental to using MATLAB in financial analysis and engineering. These explanations should help refresh your skills.

William Sharpe's "electronic work-in-progress," *Macro-Investment Analysis*, also provides an excellent explanation of matrix algebra operations using MATLAB. It is available on the World Wide Web at:

```
http://www-sharpe.stanford.edu/mia.htm
```

Hint: When you are setting up a problem, it helps to "talk through" the units and dimensions associated with each input and output matrix. In the example under "Multiplying Two Matrices" below, one input matrix has "five days' closing prices for three stocks," the other input matrix has "shares of three stocks in two portfolios," and the output matrix therefore has "five days' closing values for two portfolios." It also helps to name variables using descriptive terms.

Adding and Subtracting Matrices

Matrix addition and subtraction operate element-by-element. The two input matrices must have the same dimensions. The result is a new matrix of the same dimensions where each element is the sum or difference of each corresponding input element. For example, consider combining portfolios of different quantities of the same stocks ("shares of stocks A, B, and C [the rows] in portfolios P and Q [the columns] plus shares of A, B, and C in portfolios R and S"):

```
portfolios_pq = [100   200
                 500   400
                 300   150];

portfolios_rs = [175   125
                 200   200
                 100   500];

new_portfolios = portfolios_pq + portfolios_rs

new_portfolios =
        275.00        325.00
        700.00        600.00
        400.00        650.00
```

Adding or subtracting a scalar and a matrix is allowed and also operates element-by-element:

```
smaller_portf = new_portfolios - 10
smaller_portf =
        265.00        315.00
        690.00        590.00
        390.00        640.00
```

Multiplying Matrices

Matrix multiplication does *not* operate element-by-element. It operates according to the rules of linear algebra. In multiplying matrices, it helps to remember this key rule: the inner dimensions must be the same. That is, if the first matrix is *m*-by-3, the second must be 3-by-*n*. The resulting matrix is *m*-by-*n*. It also helps to "talk through" the units of each matrix, as mentioned above.

Matrix multiplication also is *not* commutative; i.e., it is not independent of order. A*B does *not* equal B*A. The dimension rule illustrates this property. If A is 1-by-3 and B is 3-by-1, A*B yields a scalar (1-by-1) but B*A yields a 3-by-3 matrix.

Multiplying Vectors. Vector multiplication follows the same rules and helps illustrate the principles. For example, a stock portfolio has three different stocks and their closing prices today are:

```
close_prices = [42.5   15   78.875]
```

The portfolio contains these numbers of shares of each stock:

```
num_shares = [100
              500
              300]
```

To find the value of the portfolio, simply multiply the vectors

```
portf_value = close_prices * num_shares
```

which yields

```
portf_value =
      35412.50
```

The vectors are 1-by-3 and 3-by-1; the resulting vector is 1-by-1, a scalar. Multiplying these vectors thus means multiplying each closing price by its respective number of shares and summing the result.

To illustrate order dependence, switch the order of the vectors

```
values = num_shares * close_prices

values =
       4250.00       1500.00       7887.50
      21250.00       7500.00      39437.50
      12750.00       4500.00      23662.50
```

which shows the closing values of 100, 500, and 300 shares of each stock — not the portfolio value, and meaningless for this example.

Computing Dot Products of Vectors. In matrix algebra, if X and Y are vectors of the same length

$$Y = [y_1, y_2, \ldots, y_n]$$

$$X = [x_1, x_2, \ldots, x_n]$$

then the dot product

$$X \bullet Y = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n$$

is the scalar product of the two vectors. It is an exception to the commutative rule. To compute the dot product in MATLAB, use `sum(X .* Y)` or `sum(Y .* X)`. Just be sure the two vectors have the same dimensions. To illustrate, use the previous vectors:

```
value = sum(num_shares .* close_prices')
value =
      35412.50

value = sum(close_prices .* num_shares')
value =
      35412.50
```

Multiplying Vectors and Matrices. Multiplying vectors and matrices follows the matrix multiplication rules and process. For example, a portfolio matrix contains closing prices for a week. A second matrix (vector) contains the stock quantities in the portfolio:

```
week_close_pr = [42.5     15      78.875
                 42.125   15.5    78.75
                 42.125   15.125  79
                 42.625   15.25   78.875
                 43       15.25   78.625];

port_quan = [100
             500
             300];
```

To see the closing portfolio value for each day, simply multiply:

```
week_port_value = week_close_pr * port_quan

week_port_value =
      35412.50
      35587.50
      35475.00
      35550.00
      35512.50
```

The prices matrix is 5-by-3, the quantity matrix (vector) is 3-by-1, so the resulting matrix (vector) is 5-by-1.

Multiplying Two Matrices. Matrix multiplication also follows the rules of matrix algebra. In matrix algebra notation, if A is an m-by-n matrix and B is an n-by-p matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{i1} & a_{i2} & \cdots & a_{in} \\ \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & \cdots & b_{1j} & \cdots & b_{1p} \\ b_{21} & \cdots & b_{2j} & \cdots & b_{2p} \\ \vdots & & \vdots & & \vdots \\ b_{n1} & & b_{nj} & & b_{np} \end{bmatrix}$$

then $C = A*B$ is an m-by-p matrix; and the element c_{ij} in the ith row and jth column of C is

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \ldots + a_{in}b_{nj}$$

To illustrate, assume there are two portfolios of the same three stocks above but with different quantities:

```
portfolios = [100   200
              500   400
              300   150];
```

Multiplying the 5-by-3 week's closing prices matrix by the 3-by-2 portfolios matrix yields a 5-by-2 matrix showing each day's closing value for both portfolios:

```
portfolio_values = week_close_pr * portfolios

portfolio_values =
      35412.50      26331.25
      35587.50      26437.50
      35475.00      26325.00
      35550.00      26456.25
      35512.50      26493.75
```

Monday's values result from multiplying each Monday closing price by its respective number of shares and summing the result for the first portfolio, then doing the same for the second portfolio. Tuesday's values result from multiplying each Tuesday closing price by its respective number of shares and summing the result for the first portfolio, then doing the same for the second portfolio. And so on through the rest of the week. With one simple command, MATLAB quickly performs many calculations.

Multiplying a Matrix by a Scalar. Multiplying a matrix by a scalar is an exception to the dimension and commutative rules. It just operates element-by-element:

```
portfolios = [100   200
              500   400
              300   150];

double_port = portfolios * 2

double_port =
        200.00        400.00
       1000.00        800.00
        600.00        300.00
```

Dividing Matrices

Matrix division is useful primarily for solving equations, and especially for solving simultaneous linear equations (see the next section). For example, you want to solve for X in $A*X = B$.

In ordinary algebra, you would simply divide both sides of the equation by A, and X would equal B/A. However, since matrix algebra is not commutative $(A*X \neq X*A)$, different processes apply. In formal matrix algebra, the solution involves matrix inversion. MATLAB, however, simplifies the process by providing two matrix division symbols, left and right (\ and /). In general,

`X = A\B` solves for `X` in `A*X = B`

`X = B/A` solves for `X` in `X*A = B`.

In general, matrix `A` must be a nonsingular square matrix; i.e., it must be invertible and it must have the same number of rows and columns. (Generally, a matrix is invertible if the matrix times its inverse equals the identity matrix. To understand the theory and proofs, please consult a textbook on linear algebra such as the one by Hill listed in the *Bibliography*.) MATLAB gives a warning message if the matrix is singular or nearly so.

Solving Simultaneous Linear Equations

Matrix division is especially useful in solving simultaneous linear equations. Consider this problem: given two portfolios of mortgage-based instruments, each with certain yields depending on the prime rate, how do you weight the portfolios to achieve certain annual cash flows? The answer involves solving two linear equations.

A linear equation is any equation of the form

$$a_1 x + a_2 y = b$$

where a_1, a_2, and b are constants (with a_1 and a_2 not both zero), and x and y are variables. (It's a linear equation because it describes a line in the xy-plane. For example the equation $2x + y = 8$ describes a line such that if $x = 2$ then $y = 4$.)

A system of linear equations is a set of linear equations that we usually want to solve at the same time; i.e., simultaneously. A basic principle for exact answers in solving simultaneous linear equations requires that there be as many equations as there are unknowns. To get exact answers for x and y there must be two equations. For example, to solve for x and y in the system of linear equations:

$$2x + y = 13$$

$$x - 3y = -18$$

there must be two equations, which there are. Matrix algebra represents this system as an equation involving three matrices: A for the left-side constants, X for the variables, and B for the right-side constants:

$$A = \begin{bmatrix} 2 & 1 \\ 1 & -3 \end{bmatrix} \qquad X = \begin{bmatrix} x \\ y \end{bmatrix} \qquad B = \begin{bmatrix} 13 \\ -18 \end{bmatrix}$$

where $A*X = B$.

Solving the system simultaneously simply means solving for *X*. Using MATLAB,

```
A = [2   1
      1 –3];

B = [13
    –18];

X = A \ B
```

solves for `X` in `A * X = B`.

```
X = [3
      7]
```

So *x* = 3 and *y* = 7 in this example. In general, you can use matrix algebra to solve any system of linear equations such as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \end{aligned}$$

by representing them as matrices

$$A = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \qquad X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \qquad B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

and solving for *X* in *A*∗*X* = *B*.

To illustrate, consider this situation. There are two portfolios of mortgage-based instruments, M1 and M2. They have current annual cash payments of $100 and $70 per unit, respectively, based on today's prime rate. If the prime rate moves down one percentage point, their payments would be $80 and $40. An investor holds 10 units of M1 and 20 units of M2. The

investor's receipts equal cash payments times units, or R = C * U, for each prime-rate scenario. As word equations,

	M1	M2	
Prime flat:	\$100 * 10 units	+ \$70 * 20 units	= \$2400 receipts
Prime down:	\$80 * 10 units	+ \$40 * 20 units	= \$1600 receipts

As MATLAB matrices:

```
cash = [100  70
         80  40];

units =[10
        20];

receipts = cash * units

receipts =
       2400.00
       1600.00
```

Now the investor asks the question: given these two portfolios and their characteristics, how many units of each should I hold to receive \$7000 if the prime rate stays flat and \$5000 if the prime drops one percentage point? Find the answer by solving two linear equations:

	M1	M2	
Prime flat:	\$100 * *x* units	+ \$70 * *y* units	= \$7000 receipts
Prime down:	\$80 * *x* units	+ \$40 * *y* units	= \$5000 receipts

In other words, solve for U (units) in the equation R (receipts) = C (cash) * U (units). Using MATLAB left division:

```
cash = [100  70
         80  40];

receipts = [7000
            5000];

units = cash \ receipts
units =
         43.75
         37.50
```

The investor should hold 43.75 units of portfolio M1 and 37.5 units of portfolio M2 to achieve the annual receipts desired.

Operating Element-by-Element

Finally, element-by-element arithmetic operations are called *array* operations. To indicate an array operation in MATLAB, precede the operator with a period (.). Addition and subtraction, and matrix multiplication and division by a scalar, are already array operations so no period is necessary. When using array operations on two matrices, the dimensions of the matrices must be the same. For example, given vectors of stock dividends and closing prices:

```
dividends = [1.90  0.40  1.56  4.50];
prices = [25.625  17.75  26.125  60.50];

yields = dividends ./ prices
yields =
    0.0741    0.0225    0.0597    0.0744
```

Understanding the Financial Toolbox

The Financial Toolbox contains functions that perform many common financial tasks, including:

- Handling and converting dates
- Formatting currency and charting financial data
- Analyzing and computing cash flows
- Pricing and computing yields for fixed-income securities; analyzing the term structure of interest rates
- Pricing and analyzing equity derivatives
- Analyzing portfolios

The Tools and Their Uses

Handling and Converting Dates

Calendar functions convert dates among different formats (including Excel formats), determine future or past dates, find dates of holidays and business days, compute time differences between dates, find coupon dates and coupon periods for coupon bonds, and compute time periods based on 360-, 365-, or 366-day years.

Formatting Currency and Charting Financial Data

Charting and formatting functions produce a variety of financial charts including Bollinger bands, high-low-close charts, candlestick plots, point and figure plots, and moving-average plots. This set also includes functions for handling decimal values in bank (currency) formats and as fractional prices.

Analyzing and Computing Cash Flows

Cash-flow evaluation and financial accounting functions compute interest rates, rates of return, payments associated with loans and annuities, future and present values, depreciation, and other standard accounting calculations associated with cash-flow streams.

Pricing and Computing Yields for Fixed-Income Securities

Fixed-income functions compute prices, yields, accrued interest, and sensitivities for securities such as bonds, zero-coupon bonds, and Treasury

bills. They handle odd first and last periods in price/yield calculations, compute accrued interest and discount rates, and calculate convexity and duration. A set of functions analyzes term structure of interest rates, including zero curves, forward curves, discount curves, and yield curves.

Analyzing Portfolios

Portfolio analysis functions provide basic utilities to compute variances and covariance of portfolios, find combinations to minimize variance, compute Markowitz efficient frontiers, and calculate combined rates of return.

Pricing and Analyzing Equity Derivatives

Derivatives analysis functions compute prices, yields, and sensitivities for derivative securities. They deal with both European and American options.

Black-Scholes functions work with European options. They compute delta, gamma, lambda, rho, theta, and vega, as well as values of call and put options.

Binomial functions work with American options, computing put and call prices. The optional Simulink system provides powerful tools for constructing simulation models for pricing these kinds of options.

Name and Argument Conventions

The names of Financial Toolbox functions and arguments generally use mnemonic aids.

Function Names

The toolbox uses a generic naming scheme for functions within categories. For example, all of the Black-Scholes functions begin with `bls` followed by up to five characters indicating the nature of the computation; e.g., `blsgamma` computes the value of gamma for a Black-Scholes option. Prefixes indicate the function category.

Prefix	Function Category
`acru`	Accrued interest
`annu`	Annuities
`bin`	Binomial model options

Prefix	Function Category
bls	Black-Scholes model options
bond	Bond sensitivities
cf	Cash flows
cpn	Coupon-bond dating
cur	Currency conversions
date	Date handling
days	Days between dates in calendar functions
dep	Depreciation
fv	Future values
pay	Payment calculations
pc	Portfolio constraints
port	Portfolios
pr	Pricing fixed-income securities
pv	Present values
yld	Yields of fixed-income securities
zbt	Zero curves using bootstrap method
zero	Zero curves as input to term-structure functions

The toolbox also uses several generic abbreviations that may occur anywhere within a function name.

Abbreviation	Refers To
bond	Bonds
bus	Business days

Abbreviation	Refers To
conv	Convexity
date	Dates
days	Days between specified dates
disc	Discounted security, including zero-coupon bonds
dur	Duration (in interest rate sensitivities)
fix	Fixed values; the same every period
mat	Maturity
odd	Odd periods
per	Periods
price	Prices
rate	Rates: interest rates, discount rates, etc.
rr	Rate of return
str	Strings
tbill	Treasury bills
var	Varying values
week	Weekdays

Thus, for example, `pvfix` computes the present value of a cash flow with fixed periodic payments.

Finally, note that conversion functions (i.e., functions that convert one format to a different format) all are named `abc2xyz`, where object `abc` is converted to `xyz`. For example, `zero2fwd` converts a zero curve of interest rates to a forward curve.

Function Input Arguments

Almost all functions require input arguments, some of which are optional. Many functions also allow arguments to be matrices or vectors, rather than just single values.

Common Defaults. If you do not provide some of the optional arguments, the function uses default values. Common optional arguments and their default values are:

`basis` The day-count basis. Some functions ask you to specify whether years are 360, 365, or 366 days long. The default is `basis = 0` which means actual days (365 days normally and 366 days during leap years).

`due` When payments are made in periodic transactions. If payments occur at the start of each period, `due = 1`; at the end of each period, `due = 0`. The default is `due = 0`.

`eom` Treatment of end-of-month payment dates. When end-of-month maturity dates fall on a 28th, 29th, or 30th, this flag signals treatment of other payment dates. `eom = 1`, the default, forces all payment dates to the last day of the month. `eom = 0` treats the dates literally.

`hol` A vector of holidays and non-trading dates. (By definition, holidays and non-trading days are those that occur on weekdays.) The `holidays` function supplies the default vector.

`maxiter` Some functions solve a nonlinear equation, and they typically use Newton's method. `maxiter` specifies a maximum number of iterations for the computation. The default is `maxiter = 50`.

`per` The number of periods or coupons per year during the life of the financial instrument. The default is `per = 2` (semi-annual).

Matrix Input. Most functions allow matrices or vectors, rather than just scalar values, as input arguments. For example, the `irr` function computes the internal rate of return of a cash flow. It accepts a vector holding a cash flow and returns the rate of return. However, it also accepts a matrix of cash flows, where each column in the matrix represents a different cash flow. In this case `irr` returns a vector of rates of return, where each entry in the vector corresponds to a column of the input matrix. Many other toolbox functions work similarly.

For example, the function `yearfrac` computes the difference between two dates expressed as a fraction of a year. Set up two 2-by-3 matrices of serial date numbers `d1` and `d2`, and a corresponding matrix for the day-count basis `b`, then invoke the function

```
d1 = [729750  729751  729752
      729753  729754  729755];
d2 = [729773  729786  729799
      729793  729806  729819];
b =  [ 0  1  0
       0  1  3];

yf = yearfrac(d1, d2, b)
```

which returns

```
yf =
    0.0630    0.0944    0.1288
    0.1096    0.1389    0.1753
```

This 2-by-3 output matrix corresponds to the 2-by-3 input matrices. Each entry is the fraction of a year between the corresponding dates. Thus

```
yf(1,1)
ans =
    0.0630
```

is the difference between the serial dates `729750` and `729773` expressed as a fraction of the number of days in the year. (See “Handling and Converting Dates” to understand date representation.)

This simple example shows how to collect objects into a matrix and then use a toolbox function to compute answers for the entire collection. This feature can be useful in portfolio management, for example, where you might want to organize multiple assets into a single collection. Place data for each asset in a different column or row of a matrix, then pass the matrix to the Financial Toolbox function. MATLAB performs the same computation on all of the assets at once.

Matrices of String Input. Enter strings in MATLAB surrounded by single quotes (`'string'`).

Strings are stored as character arrays, one ASCII character per element. Thus the date string

```
ds = '9/16/1995'
```

is actually a 1-by-9 vector. Strings making up the rows of a matrix or vector all must have the same length. To enter several date strings, therefore, use a column vector and be sure all strings are the same length. Fill in with spaces or zeros. For example, to create a vector of dates corresponding to irregular cash flows:

```
df = ['01/12/1994'
      '02/14/1995'
      '03/03/1995'
      '06/14/1995'
      '12/01/1995'];
```

`df` actually becomes a 5-by-10 character array.

Don't mix numbers and strings in a matrix. If you do, MATLAB treats all entries as characters. For example,

```
item = [83  90  97  '14-Sep-1997']
```

becomes a 1-by-14 character array, not a 1-by-4 vector, and it contains

```
item =
SZa14-Sep-1997
```

Interest Rate Arguments

One common argument, both as input and output, is interest rate. All Financial Toolbox functions expect and return interest rates as decimal fractions. Thus an interest rate of 9.5% is `0.095`.

Function Output Arguments

Some functions return no arguments, some return just one, and some return multiple arguments. Functions that return multiple arguments use the syntax:

```
[A, B, C] = function(variables...)
```

to return arguments `A`, `B`, and `C`. If you omit all but one, the function returns the first argument. Thus for this example if you specify

```
X = function(variables...)
```

the *function* returns a value for A, but not for B or C.

Some functions that return vectors accept only scalars as arguments. Why could such functions not accept vectors as arguments and return matrices, where each column in the output matrix corresponds to an entry in the input vector? The answer is that the output vectors can be variable length and thus will not fit in a matrix without some convention to indicate that the shorter columns are missing data.

Functions that require asset life as an input, and return values corresponding to different periods over that life, cannot generally handle vectors or matrices as input arguments. Those functions are:

`amortize`	Amortization
`depfixdb`	Fixed declining-balance depreciation
`depgendb`	General declining-balance depreciation
`depsoyd`	Sum of years' digits depreciation

For example, suppose you have a collection of assets such as automobiles and you want to compute the depreciation schedules for them. The function `depfixdb` computes a stream of declining-balance depreciation values for an asset. You might want to set up a vector where each entry is the initial value of each asset. `depfixdb` also needs the lifetime of an asset. If you were to set up such a collection of automobiles as an input vector, and the lifetimes of those automobiles varied, then the resulting depreciation streams would differ in length according to the life of each automobile, and the output column lengths would vary. A matrix must have the same number of rows in each column.

Handling and Converting Dates

Since virtually all financial data is dated or derives from a time series, financial functions must have extensive date-handling capabilities. This section discusses date handling in the Financial Toolbox.

Date Formats

You most often work with date strings (14-Sep-1997) when dealing with dates. The Financial Toolbox works internally with *serial* date numbers (729647). A serial date represents a calendar date as the number of days that has passed since a fixed base date. In MATLAB, serial date number 1 is January 1, 0000 A.D. MATLAB also uses serial time to represent fractions of days beginning at

midnight; for example, 6 p.m. equals 0.75 serial days. So the string `'14-Sep-1997, 6:00 pm'` in MATLAB is date number 729647.75.

Many toolbox functions that require dates accept either date strings or serial date numbers. If you are dealing with a few dates at the MATLAB command-line level, date strings are more convenient. If you are using toolbox functions on large numbers of dates, as in analyzing large portfolios or cash flows, performance improves if you use date numbers.

The toolbox provides functions that convert date strings to serial date numbers, and vice versa.

Note: The date functions `datenum`, `datestr`, `datevec`, `eomday`, `now`, and `weekday` now ship with basic MATLAB. They originally shipped only with the Financial Toolbox, and their descriptions remain in this manual for your convenience.

Date Conversions

Functions that convert between date formats are:

`datenum`	Converts a date string to a serial date number
`datestr`	Converts a serial date number to a date string
`m2xdate`	Converts MATLAB serial date number to Excel serial date number
`x2mdate`	Converts Excel serial date number to MATLAB serial date number

Another function, `datevec`, converts a date number or date string to a date vector whose elements are `[year month day hour minute second]`. Date vectors are an internal format for some MATLAB functions and you will not normally use them in financial calculations.

Input Conversions. The `datenum` function is important for using the Financial Toolbox efficiently. `datenum` takes an input string in any of several formats, with `'dd-mmm-yyyy'`, `'mm/dd/yyyy'`, or `'dd-mmm-yyyy, hh:mm:ss.ss'` most

common. You can form up to six fields from letters and numbers separated by any other characters:

- The day field is an integer from 1 to 31.
- The month field is either an integer from 1 to 12 or an alphabetic string with at least three characters.
- The year field is a non-negative integer: if only two numbers are specified, then a year 19yy is assumed; if the year is omitted, then the current year is used as a default.
- The hours, minutes, and seconds fields are optional. They are integers separated by colons or followed by `'am'` or `'pm'`.

For example, if the current year is 1995, then these are all equivalent

```
'17-May-1995'
'17-May-95'
'17-may'
'May 17, 1995'
'5/17/95'
'5/17'
```

and both of these represent the same time:

```
'17-May-1995, 18:30'
'5/17/95/6:30 pm'
```

Note that the default format for numbers-only input follows the American convention. Thus `3/6` is March 6, not June 3.

With `datenum` you can convert dates and store them in a variable, then later pass the variable to a function, or you can use `datenum` directly in function input. For example, consider the function `prmat` that computes the price of a bond yielding interest only at maturity. You can first set up variables for the necessary dates, redemption value, coupon rate, and annual yield via

```
settledate = datenum('17-May-1994');
maturedate = datenum('1-Oct-1994');
issuedate = datenum('1-Jan-1994');
redeem = 100.0;
couprate = 0.08;
yield = 0.07;
```

then call the function:

```
prmat(settledate, maturedate, issuedate, redeem, ...
        couprate, yield)
```

Or you can call the function this way:

```
prmat(datenum('17-May-1994'), datenum('1-Oct-1994'), ...
        datenum('1-Jan-1994'), 100, 0.08, 0.07)
```

Or you can use date strings directly:

```
prmat('17-May-1994','1-Oct-1994','1-Jan-1994', 100, 0.08, 0.07)
```

Remember that if you create a vector of input date strings, use a column vector and be sure all strings are the same length. Fill in with spaces or zeros. See "Matrices of String Input" above.

Output Conversions. The function `datestr` converts a serial date number to one of 19 different date string output formats showing date, time, or both. The default output for dates is a day-month-year string: `24-Aug-1995`. This function is quite useful for preparing output reports.

Format	Description
`01-Mar-1995 15:45:17`	day-month-year hour:minute:second
`01-Mar-1995`	day-month-year
`03/01/95`	month/day/year
`Mar`	month, three letters
`M`	month, single letter
`3`	month
`03/01`	month/day
`1`	day of month
`Wed`	day of week, three letters
`W`	day of week, single letter

Format	Description
1995	year, four numbers
95	year, two numbers
Mar95	month year
15:45:17	hour:minute:second
03:45:17 PM	hour:minute:second AM or PM
15:45	hour:minute
03:45 PM	hour:minute AM or PM
Q1-95	calendar quarter-year
Q1	calendar quarter

Current Date and Time

The toolbox functions `today` and `now` return serial date numbers for the current date, and the current date and time, respectively.

```
today
ans =
      729647

now
ans =
     729647.51
```

The standard MATLAB function `date` returns a string for today's date.

```
date
ans =
14-Sep-1997
```

Day-Count Basis

Financial calculations generally work with dates on the basis of a 360- or 365-day year. Many toolbox functions that require dates also ask for a basis.

The default basis is actual days if none is specified. The toolbox basis choices are:

`basis = 0`	Actual days (default)
`basis = 1`	360-day year (assumes all months are 30 days)
`basis = 2`	Actual days/360
`basis = 3`	Actual days/365

Thus you can compute the difference between two dates in four different ways. The first way (`basis = 0`) simply returns the difference in actual days between two dates. The second (`basis = 1`) returns the difference, but adjusted to a 360-day year in which all months have 30 days. The third and fourth methods return a difference in dates as a fraction of either a 360- or 365-day year, respectively.

Determining Dates

The toolbox provides many functions for determining specific dates, including functions which account for holidays and other non-trading days.

For example, you schedule an accounting procedure for the last Friday of every month. The `lweekdate` function returns those dates for 1998; the `6` specifies Friday:

```
fridates = lweekdate(6, 1998, 1:12);

fridays = datestr(fridates)
fridays =
30-Jan-1998
27-Feb-1998
27-Mar-1998
24-Apr-1998
29-May-1998
26-Jun-1998
31-Jul-1998
28-Aug-1998
25-Sep-1998
30-Oct-1998
27-Nov-1998
25-Dec-1998
```

Or your company closes on Martin Luther King Jr. Day, which is the third Monday in January. The `nweekdate` function determines those dates for 1998 through 2001:

```
mlkdates = nweekdate(3, 2, 1998:2001, 1);

mlkdays = datestr(mlkdates)
mlkdays =
19-Jan-1998
18-Jan-1999
17-Jan-2000
15-Jan-2001
```

Accounting for holidays and other non-trading days is important when examining financial dates. The toolbox provides the `holidays` function, which contains holidays and special non-trading days for the New York Stock Exchange between 1950 and 2030, inclusive. You can edit the `holidays.m` file to customize it with your own holidays and non-trading days. In this example, use it to determine the standard holidays in the last half of 1998:

```
lhhdates = holidays('1-Jul-1998', '31-Dec-1998');

lhhdays = datestr(lhhdates)
lhhdays =
03-Jul-1998
07-Sep-1998
26-Nov-1998
25-Dec-1998
```

Now use the toolbox `busdate` function to determine the next business day after these holidays:

```
lhnextdates = busdate(lhhdates);

lhnextdays = datestr(lhnextdates)
lhnextdays =
06-Jul-1998
08-Sep-1998
27-Nov-1998
28-Dec-1998
```

The toolbox also provides the `cfdates` function to determine cash-flow dates for securities with periodic payments. This function accounts for the coupons per

year, the day-count basis, and the end-of-month rule. For example, to determine the cash-flow dates for a security that pays four coupons per year on the last day of the month, on an actual/365 day-count basis, just enter the settlement date, the maturity date, and the parameters:

```
paydates = cfdates('14-Mar-1997', '30-Nov-1998', 4, 3, 1);

paydays = datestr(paydates)
paydays =
31-May-1997
31-Aug-1997
30-Nov-1997
28-Feb-1998
31-May-1998
31-Aug-1998
30-Nov-1998
```

Formatting Currency and Charting Financial Data

The Financial Toolbox provides several functions to format currency and chart financial data.

Currency Formats

The currency formatting functions are:

`cur2frac`	Converts decimal currency values to fractional values
`cur2str`	Converts a value to Financial Toolbox bank format
`frac2cur`	Converts fractional currency values to decimal values

These examples show their use

```
dec = frac2cur('12.1', 8)
```

returns `dec = 12.125`, which is the decimal equivalent of 12-1/8. The second input variable is the denominator of the fraction.

```
str =  cur2str(-8264, 2)
```

returns the string `($8264.00)`. For this toolbox function, the output format is a numerical format with dollar sign prefix, two decimal places, and negative numbers in parentheses; e.g., `($123.45)` and `$6789.01`. The standard

MATLAB bank format uses two decimal places, no dollar sign, and a minus sign for negative numbers; e.g., –123.45 and 6789.01.

Financial Charts

The toolbox financial charting functions

`bolling`	Bollinger band chart
`candle`	Candlestick chart
`pointfig`	Point and figure chart
`highlow`	High, low, open, close chart
`movavg`	Leading and lagging moving averages chart

plot financial data and produce presentation-quality figures quickly and easily. They work with standard MATLAB functions that draw axes, control appearance, and add labels and titles.

Here are two plotting examples: a high-low-close chart of sample IBM stock price data, and a Bollinger band chart of the same data. These examples load data from an external file (`ibm.dat`), then call the functions using subsets of the data. `ibm` is an R-by-6 matrix where each row R is a trading day's data and where columns 2, 3, and 4 contain the high, low, and closing prices, respectively.

Note: The data in `ibm.dat` is fictional and for illustrative use only.

High-Low-Close Chart Example. First load the data and set up matrix dimensions. `load` and `size` are standard MATLAB functions.

```
load ibm.dat;
[ro, co] = size(ibm);
```

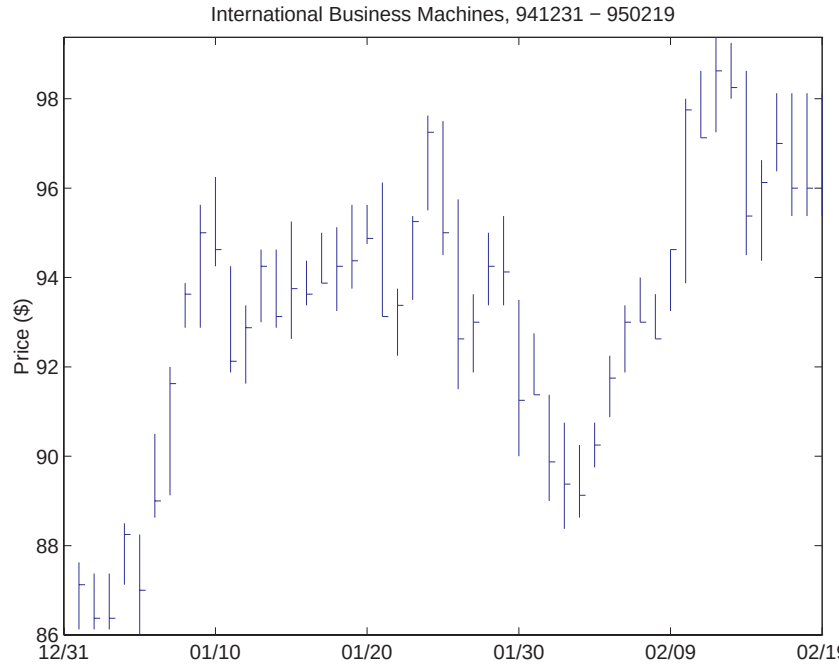
Open a figure window for the chart. Use the Financial Toolbox `highlow` function to plot high, low, and close prices for the last 50 trading days in the data file.

```
figure;

highlow(ibm(ro–50:ro,2),ibm(ro–50:ro,3),ibm(ro–50:ro,4),[],'b');
```

Add labels and title, and set axes with standard MATLAB functions. Use the Financial Toolbox `dateaxis` function to provide dates for the *x*-axis ticks.

```
xlabel('');
ylabel('Price ($)');
title('International Business Machines, 941231 - 950219');
axis([0 50 –inf inf]);
dateaxis('x',6)
```

MATLAB produces a figure similar to this. The plotted data and axes you see may differ. On a color monitor, the high-low-close bars are blue.



Bollinger Chart Example. Next the Financial Toolbox `bolling` function produces a Bollinger band chart using all the closing prices in the same IBM stock price matrix. A Bollinger band chart plots actual data along with three other bands of data. The upper band is two standard deviations above a moving average; the lower band is two standard deviations below that moving average; and the

middle band is the moving average itself. This example uses a 15-day moving average.

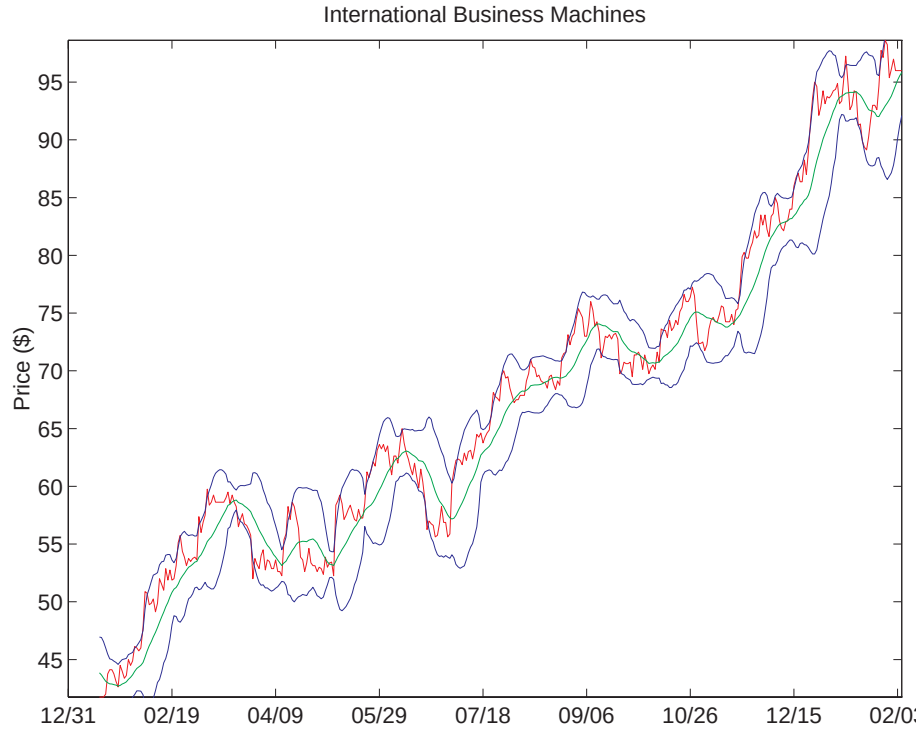
Assuming the previous IBM data is still loaded, simply execute the Financial Toolbox function.

```
bolling(ibm(:,4), 15, 0);
```

Specify the axes, labels, and titles. Again, use `dateaxis` to add the *x*-axis dates.

```
axis([0 ro min(ibm(:,4)) max(ibm(:,4))]);
ylabel('Price ($)');
title(['International Business Machines']);
dateaxis('x', 6)
```

MATLAB does the rest. On a color monitor, the red lines are the upper and lower bands, the green line is the 15-day simple moving average, and the blue line charts the actual price data. In this reproduction, the outer lines are the upper and lower bands, the middle smooth line is the moving average, and the middle jagged line is the actual price data. Again, the plotted data and axes you see may differ from this reproduction.



For help using MATLAB plotting functions, see the “Graphics” section of *Getting Started with MATLAB*. See *Using MATLAB Graphics* for details on the `axis`, `title`, `xlabel`, and `ylabel` functions.

Analyzing and Computing Cash Flows

The Financial Toolbox cash-flow functions compute interest rates, payments, present and future values, annuities, rates of return, and depreciation streams.

Some examples in this section use this income stream: an initial investment of $20,000 followed by three annual return payments, a second investment of $5,000, then four more returns. Investments are negative cash flows, return payments are positive cash flows.

```
stream = [-20000,  2000,  2500,  3500,  -5000,  6500,  ...
           9500,  9500,  9500];
```

Interest Rates / Rates of Return

Several functions calculate interest rates involved with cash flows. To compute the internal rate of return of the cash stream, simply execute the toolbox function `irr`

```
ror = irr(stream)
```

which gives a rate of return of 11.72%.

Note that the internal rate of return of a cash flow may not have a unique value. Every time the sign changes in a cash flow, the equation defining `irr` can give up to two additional answers. An `irr` computation requires solving a polynomial equation, and the number of real roots of such an equation can depend on the number of sign changes in the coefficients. The equation for internal rate of return is

$$\frac{cf_1}{(1+r)} + \frac{cf_2}{(1+r)^2} + \ldots + \frac{cf_n}{(1+r)^n} + Investment = 0$$

where *Investment* is a (negative) initial cash outlay at time 0, cf_n is the cash flow in the *n*th period, and *n* is the number of periods. Basically, `irr` finds the rate *r* such that the net present value of the cash flow equals the initial investment. If all of the cf_ns are positive there is only one solution. Every time there is a change of sign between coefficients, up to two additional real roots are possible. There is usually only one answer that makes sense, but it is possible to get returns of both 5% and 11% (for example) from one income stream.

Another toolbox rate function, `effrr`, calculates the effective rate of return given an annual interest rate (also known as nominal rate or annual percentage rate, APR) and number of compounding periods per year. To ask for the effective rate of a 9% APR compounded monthly, simply enter:

```
rate = effrr(0.09, 12)
```

The answer is 9.38%.

A companion function `nomrr` computes the nominal rate of return given the effective annual rate and the number of compounding periods.

Present / Future Values

The toolbox includes functions to compute the present or future value of cash flows at regular or irregular time intervals with equal or unequal payments: `fvfix`, `fvvar`, `pvfix`, and `pvvar`. The `-fix` functions assume equal cash flows at regular intervals, while the `-var` functions allow irregular cash flows at irregular periods.

Now compute the net present value of the sample income stream for which you computed the internal rate of return. This exercise also serves as a check on that calculation because the net present value of a cash stream at its internal rate of return should be zero. Enter

```
npv = pvvar(stream, ror)
```

which returns 2.6830e-011, or 0.000000000026830— very close to zero. The answer usually is not *exactly* zero due to rounding errors and the computational precision of the computer.

Note: Some other toolbox functions behave similarly. The functions that compute a bond's yield, for example, often must solve a nonlinear equation. If you then use that yield to compute the net present value of the bond's income stream, it usually does not *exactly* equal the purchase price — but the difference is negligible for practical applications.

Sensitivity

The Financial Toolbox can also compute the Macaulay duration for a cash flow where all values in the stream are not necessarily the same. The Macaulay duration measures how long, on average, the owner waits before receiving a payment. `cfdur` computes the Macaulay duration and modified duration (volatility) of a cash stream.

Returning to the sample cash flow and using a discount rate of 6%, this example

```
rate = 0.06;
[duration, moddur] = cfdur(stream, rate)
```

gives a duration of 23.49 periods and a modified duration (volatility) of 22.16 periods. The length of these periods is the same as the length of the periods in

the cash flow. Note that using the computed internal rate of return as the discount rate is not appropriate, since a cash flow with net present value of 0 has infinite duration.

Depreciation

The toolbox includes functions to compute standard depreciation schedules: straight line, general declining-balance, fixed declining-balance, and sum of years' digits. Functions also compute a complete amortization schedule for an asset, and return the remaining depreciable value after a depreciation schedule has been applied.

This example depreciates an automobile worth $15,000 over five years with a salvage value of $1,500. It computes the general declining balance using two different depreciation rates: 50% (or 1.5), and 100% (or 2.0, also known as double declining balance). Enter

```
decline1 = depgendb(15000, 1500, 5, 1.5)
decline2 = depgendb(15000, 1500, 5, 2.0)
```

which returns

```
decline1 =
      4500.00      3150.00      2205.00      1543.50      2101.50
decline2 =
      6000.00      3600.00      2160.00      1296.00       444.00
```

These functions return the actual depreciation amount for the first four years and the remaining depreciable value as the entry for the fifth year.

Annuities

Several toolbox functions deal with annuities. This first example shows how to compute the interest rate associated with a series of loan payments when only the payment amounts and principal are known. For a loan whose original value was $5000.00 and which was paid back monthly over four years at $130.00/month:

```
rate = annurate(4*12, 130, 5000, 0, 0)
```

The function returns a rate of 0.0094 monthly, or approximately 11.28% annually.

The next example uses a present-value function to show how to compute the initial principal when the payment and rate are known. For a loan paid at $300.00/month over four years at 11% annual interest:

```
principal = pvfix(0.11/12, 4*12, 300, 0, 0)
```

The function returns the original principal value of $11,607.43.

The final example computes an amortization schedule for a loan or annuity. The original value was $5000.00 and was paid back over 12 months at an annual rate of 9%.

```
[prpmt, intpmt, balance, payment] = ...
        amortize(0.09/12, 12, 5000, 0, 0);
```

This function returns vectors containing the amount of principal paid,

```
prpmt = [402.76  405.78  408.82  411.89  414.97  418.09 ...
         421.22  424.38  427.56  430.77  434.00  437.26]
```

the amount of interest paid,

```
intpmt = [34.50  31.48  28.44  25.37  22.28  19.17 ...
          16.03  12.88   9.69   6.49   3.26   0.00]
```

the remaining balance for each period of the loan,

```
balance = [4600.24  4197.49  3791.71  3382.89  2971.01 ...
           2556.03  2137.94  1716.72  1292.34   864.77 ...
            434.00    0.00]
```

and a scalar for the monthly payment.

```
payment = 437.26
```

Pricing and Computing Yields for Fixed-Income Securities

The Financial Toolbox includes many functions to compute accrued interest, determine prices and yields, calculate convexity and duration of fixed-income securities, and generate and analyze term structure of interest rates.

Terminology

Since terminology varies among texts on this subject, here are some basic definitions that apply to these Financial Toolbox functions. You can also find these and other definitions in the *Glossary*.

The **settlement date** of a bond is the date when money first changes hands; i.e., when a buyer pays for a bond. It need not coincide with the issue date. The **issue date** is the date a bond is first offered for sale. That date usually determines when interest payments, known as **coupons**, are made. The first and last **coupon dates** are the dates when the first and last coupons are paid. Typically a bond pays coupons annually or semi-annually.

Fixed-income securities can 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 coupon period, and thus the bond owner is not entitled to the full value of the coupon for that period. Instead, the coupon is pro-rated according to how long the bond is held during that period. The toolbox includes price and yield functions that handle odd first period, odd last period, and odd first and last periods.

The **maturity date** of a bond is the date when the issuer returns the final face value, also known as the **redemption value** or **par value**, to the buyer. Typically the **purchase price**, the price actually paid for a bond, is not the same as the redemption value. The **yield** of a bond is determined by the ratio of redemption value to purchase price over the life of the bond. It is the nominal annual interest rate that gives a **future value** of the purchase price equal to the redemption value of the bond. The coupon payments determine part of that yield.

In the *Reference* chapter, these fixed-income functions use standard abbreviations for variables.

Abbreviation	Refers To
`p`	Price
`ai`	Accrued interest
`sd`	Settlement date
`md`	Maturity date

Abbreviation	Refers To
id	Issue date
fd	First coupon date
lcd	Last coupon date
rv	Redemption value
cpn	Coupon rate
yld	Bond yield
per	Coupons per year
basis	Day-count basis
eom	End of month rule

Pricing Functions

This example shows how easily you can compute the price of a bond with an odd first period. It uses the standard abbreviations to set up the input variables.

```
sd = '11/11/1992';
md = '03/01/2005';
id = '10/15/1992';
fd = '03/01/1993';
cpn = 0.0785;
yld = 0.0625;
per = 2;
basis = 0;
eom = 1;
```

Simply executing the function `bndprice`

```
[pr, ai] = bndprice(yld, cpn, sd, md, per, basis, eom, id, fd)
```

returns a price, `pr = 113.60` and accrued interest, `ai = 0.59`.

Similar functions compute prices with regular payments, payment at maturity, and odd first and last periods, as well as prices of Treasury bills and discounted securities such as zero-coupon bonds.

Yield Functions

To illustrate toolbox yield functions, this example computes the yield of a bond that has odd first and last periods and settlement in the first period. It uses descriptive variable names rather than the standard abbreviations. First set up variables for settlement date, maturity date, issue date, first coupon date (`fcoup`), and a last coupon date (`lcoup`).

```
settledate = '12-Jan-1994';
maturedate = '1-Oct-1995';
issuedate = '1-Jan-1994';
fcoup = '15-Jan-1994';
lcoup = '15-Apr-1994';
```

Now for a redemption value of $100.00, supply a purchase price of $95.70, coupon rate of 4%, paying coupons four times a year, and assume a 360-day year of equal 30-day months (`basis = 1`). To find the answer the function solves a nonlinear equation using at most 50 iterations.

```
purch = 95.7;
coupr = 0.04;
freq = 4;
basis = 1;
eom = 1;
```

Call the function

```
bndyield(purch, coupr, settledate, maturedate, freq, basis,...
eom, issuedate, fcoup, lcoup)
```

which returns the yield of 6.73%.

You can also use vectors for each of these arguments but be careful that each vector has the same number of arguments. You receive a vector of yields in return. Using vectors computes yields for several securities at once.

Toolbox functions also compute yields of Treasury bills and fixed-income securities with payments at regular times or at maturity. For example, using the previous dates and redemption value, a price of $95.00 and a 6% coupon rate, this function

```
price = 95.0;
cpn = 0.06;
yldmat(settledate, maturedate, issuedate, redeem, price, cpn)
```

calculates a 13.62% yield for a security whose interest is paid at maturity.

Fixed-Income Sensitivities

The toolbox includes functions to perform sensitivity analysis such as convexity and the Macaulay and modified durations for fixed-income securities. 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. The modified duration is the Macaulay duration discounted by the per-period interest rate; i.e., divided by (1+rate/frequency). Modified duration is also known as volatility.

To illustrate, this example computes the Macaulay and modified durations for a bond with settlement and maturity dates as above, a redemption or par value of $100.00, a 5% coupon rate, a 4.5% yield, semi-annual coupons, and date basis = 0 (actual days).

```
redeem = 100;
cpn = 0.05;
freq = 2;
yld = 0.045;

[Mac, Mod] = bonddur(settledate, maturedate, redeem, cpn, ...
        yld, freq, 0);
```

The durations are (in years):

```
[Mac, Mod] = [0.7049    0.6894]
```

Term Structure of Interest Rates

The toolbox contains several functions to derive and analyze interest rate curves, including data conversion and extrapolation, bootstrapping, and interest-rate curve conversion functions.

One of the first problems in analyzing the term structure of interest rates is dealing with market data reported in different formats. Treasury bills, for example, are quoted with bid and asked bank-discount rates. Treasury notes and bonds, on the other hand, are quoted with bid and asked prices based on $100 face value. To examine the full spectrum of Treasury securities, analysts must convert data to a single format. Toolbox functions ease this conversion.

In this brief example, we use only one security each; analysts often use 30, 100, or more of each.

First, capture Treasury bill quotes in their reported format

```
                Maturity                  Days  Bid      Ask      AskYield
tbill = [datenum('12/26/1997')  53     0.0503   0.0499   0.0510];
```

then capture Treasury bond quotes in their reported format

```
                Coupon   Maturity              Bid        Ask        AskYield
tbond = [0.08875  datenum(1998,11,4) 103+4/32  103+6/32  0.0564];
```

and note that these quotes are based on a November 3, 1997 settlement date.

```
settle = datenum('3-Nov-1997');
```

Next use the toolbox `tbl2bond` function to convert the Treasury bill data to Treasury bond format:

```
tbtbond = tbl2bond(tbill)

tbtbond =
         0         729750          99.259        99.265      0.052091
```

(The second element of `tbtbond` is the serial date number for December 26, 1997.)

Now combine short-term (Treasury bill) with long-term (Treasury bond) data to set up the overall term structure.

```
tbondsall = [tbtbond; tbond]

tbondsall =
             0  729750        99.259        99.265      0.052091
       0.08875  730063       103.125       103.1875     0.0564
```

The toolbox provides a second data-preparation function, `tr2bonds`, to convert the bond data into a form ready for the bootstrapping functions. `tr2bonds` generates a matrix of bond information sorted by maturity date, plus vectors of prices and yields.

```
[bonds, prices, yields] = tr2bonds(tbondsall);
```

With this market data, we are now ready to use one of the toolbox bootstrapping functions to derive an implied zero curve. Bootstrapping is a

process whereby you begin with known data points and solve for unknown data points using an underlying arbitrage theory. Every coupon bond can be valued as a package of zero-coupon bonds which mimic its cash flow and risk characteristics. By mapping yields-to-maturity for each theoretical zero-coupon bond, to the dates spanning the investment horizon, we can create a theoretical zero-rate curve. The toolbox provides two bootstrapping functions: `zbtprice` derives a zero curve from bond data and *prices*, and `zbtyield` derives a zero curve from bond data and *yields*. We use `zbtprice`

```
[zerorates, curvedates] = zbtprice(bonds, prices, settle)

zerorates =
     0.051107
     0.054501
curvedates =
      729750
      730063
```

where `curvedates` gives the investment horizon.

```
datestr(curvedates)
ans =
26-Dec-1997
04-Nov-1998
```

Additional toolbox functions construct discount, forward, and par yield curves from the zero curve, and vice-versa:

```
[discrates, curvedates] = zero2disc(zerorates, curvedates, settle);
[fwdrates, curvedates] = zero2fwd(zerorates, curvedates, settle);
[pyldrates, curvedates] = zero2pyld(zerorates, curvedates, settle);
```

Pricing and Analyzing Equity Derivatives

These toolbox functions compute prices, sensitivities, and profits for portfolios of options or other equity derivatives. They use the Black-Scholes model for European options and the binomial model for American options. Such measures are useful for managing portfolios and for executing collars, hedges, and straddles.

Sensitivity Measures

There are six basic sensitivity measures associated with option pricing: delta, gamma, lambda, rho, theta, and vega — the "greeks." The toolbox provides functions for calculating each sensitivity and for implied volatility.

Delta

Delta of a derivative security is the rate of change of its price relative to the price of the underlying asset. It is the first derivative of the curve that relates the price of the derivative to the price of the underlying security. When delta is large, the price of the derivative is sensitive to small changes in the price of the underlying security.

Gamma

Gamma of a derivative security is the rate of change of delta relative to the price of the underlying asset; i.e., the second derivative of the option price relative to the security price. When gamma is small, the change delta is small. This sensitivity measure is important for deciding how much to adjust a hedge position.

Lambda

Lambda, also known as the elasticity of an option, represents the percentage change in the price of an option relative to a 1% change in the price of the underlying security.

Rho

Rho is the rate of change in option price relative to the underlying security's risk-free interest rate.

Theta

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

Vega

Vega is 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. For example, options traders often

must decide whether to buy an option to hedge against vega or gamma. The hedge selected usually depends upon how frequently one rebalances a hedge position and also upon the variance of the price of the underlying asset (the volatility). If the variance is changing rapidly, balancing against vega is usually preferable.

Implied Volatility

The implied volatility of an option is the variance that makes a call option price equal to the market price. It is used to determine a market estimate for the future volatility of a stock and provides the input volatility (when needed) to the other Black-Scholes functions.

Analysis Models

Toolbox functions for analyzing equity derivatives use the Black-Scholes model for European options and the binomial model for American options. The Black-Scholes model makes several assumptions about the underlying securities and their behavior. The binomial model, on the other hand, makes far fewer assumptions about the processes underlying an option. For further explanation, please see the book by John Hull listed in the *Bibliography*.

Black-Scholes Model

Using the Black-Scholes model entails several assumptions:

- The option price follows an Ito process.
- The option can be exercised only on its expiration date.
- Short selling is permitted.
- There are no transaction costs.
- All securities are divisible and pay no dividends.
- There is no riskless arbitrage.
- Trading is a continuous process.
- The risk-free interest rate is constant and remains the same for all maturities.

If any of these assumptions is untrue, Black-Scholes may not be an appropriate model.

To illustrate toolbox Black-Scholes functions, this example computes the call and put prices of a European option and its delta, gamma, lambda, and implied volatility. The asset price is $100.00, the exercise price is $95.00, the risk-free interest rate is 10%, the time to maturity is 0.25 years, the volatility is 0.50, and the dividend rate is 0. Simply executing the toolbox functions:

```
[optcall, optput] = blsprice(100, 95, 0.10, 0.25, 0.50, 0);
[callval, putval] = blsdelta(100, 95, 0.10, 0.25, 0.50, 0);
gammaval = blsgamma(100, 95, 0.10, 0.25, 0.50, 0);
vegaval = blsvega(100, 95, 0.10, 0.25, 0.50, 0);
[lamcall, lamput] = blslambda(100, 95, 0.10, 0.25, 0.50, 0);
```

yields

- The option call price `optcall` = $13.70
- The option put price `optput` = $6.35
- delta for a call `callval` = 0.6665 and delta for a put `putval` = –0.3335
- gamma `gammaval` = 0.0145
- vega `vegaval` = 18.1843
- lambda for a call `lamcall` = 4.8664 and lambda for a put `lamput` = –5.2528

Now as a computation check, find the implied volatility of the option using the call option price from `blsprice`.

```
volatility = blsimpv(100, 95, 0.10, 0.25, optcall);
```

The function returns an implied volatility of 0.500, the original `blsprice` input.

Binomial Model

The binomial model for pricing options or other equity derivatives assumes that 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 up and one down, over any short time period. Plotting the two 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.” This model applies to American options, which can be exercised any time up to and including their expiration date.

This example prices an American call option using a binomial model. Again, the asset price is $100.00, the exercise price is $95.00, the risk-free interest rate is 10%, and the time to maturity is 0.25 years. It computes the tree in increments of 0.05 years, so there are 0.25/0.05 = 5 periods in the example. The volatility is 0.50, this is a call (`flag = 1`), the dividend rate is 0, and it pays a dividend of $5.00 after three periods (an ex-dividend date). Executing the toolbox function

```
[optionpr, optionval] = binprice(100, 95, 0.10, 0.25, 0.05, ...
        0.50, 1, 0, 5.0, 3);
```

yields the option prices `optionpr =`

```
100.00    111.27    123.87    137.96    148.69    166.28
     0     89.97    100.05    111.32    118.90    132.96
     0         0     81.00     90.02     95.07    106.32
     0         0         0     72.98     76.02     85.02
     0         0         0         0     60.79     67.98
     0         0         0         0         0     54.36
```

and the option values `optionval =`

```
 12.10     19.17     29.35     42.96     54.17     71.28
     0      5.31      9.41     16.32     24.37     37.96
     0         0      1.35      2.74      5.57     11.32
     0         0         0         0         0         0
     0         0         0         0         0         0
     0         0         0         0         0         0
```

The output from the binomial function is a binary tree. Read the option prices matrix this way: column 1 shows the price for period 0, column 2 shows the up and down prices for period 1, column 3 shows the up-up, up-down, and down-down prices for period 2, etc. Ignore the zeros. The option value matrix gives the associated option value for each node in the price tree. Ignore the zeros that correspond to a zero in the price tree.

Analyzing Portfolios

Portfolio managers concentrate their efforts on achieving the best possible trade-off between risk and return. For portfolios constructed from a fixed set of assets, the risk/return profile varies with the portfolio composition. Portfolios that maximize the return, given the risk, or, conversely, minimize the risk for

the given return, are called *optimal*. Optimal portfolios define a line in the risk/return plane called the *efficient frontier*.

A portfolio may also have to meet additional requirements to be considered. Different investors have different levels of risk tolerance. Selecting the adequate portfolio for a particular investor is a difficult process. The portfolio manager can hedge the risk related to a particular portfolio along the efficient frontier with partial investment in risk-free assets. The definition of the capital allocation line, and finding where the final portfolio falls on this line, if at all, is a function of:

- The risk/return profile of each asset
- The risk-free rate
- The borrowing rate
- The degree of risk aversion characterizing an investor

The Financial Toolbox includes a set of functions designed to solve the problem of finding the portfolio that best meets the requirements of investors, while considering all parameters.

Portfolio Optimization Functions

The portfolio optimization functions assist portfolio managers in constructing portfolios that optimize risk and return.

Capital Allocation	
`portalloc`	Computes the optimal risky portfolio on the efficient frontier, based on the risk-free rate, the borrowing rate, and the investor's degree of risk aversion. Also generates the capital allocation line, which provides the optimal allocation of funds between the risky portfolio and the risk-free asset.

Efficient Frontier Computation	
`frontcon`	Computes portfolios along the efficient frontier for a given group of assets. The computation is based on sets of constraints representing the maximum and minimum weights for each asset, and the maximum and minimum total weight for specified groups of assets.
`portopt`	Computes portfolios along the efficient frontier for a given group of assets. The computation is based on a set user-specified linear constraints. Typically, these constraints are generated using the constraint specification functions described below.

Constraint Specification		
`portcons`	Generates the portfolio constraints matrix, a matrix of constraints for a portfolio of asset investments using linear inequalities. The inequalities are of the type `A*Wts' <= b`, where `Wts` is a row vector of weights. The capabilities of `portcons` are also provided individually by the following functions.	
	`pcalims`	Asset minimum and maximum allocation. Generates a constraint set to fix the minimum and maximum weight for each individual asset.
	`pcgcomp`	Group-to-group ratio constraint. Generates a constraint set specifying the maximum and minimum ratios between pairs of groups.
	`pcglims`	Asset group minimum and maximum allocation. Generates a constraint set to fix the minimum and maximum total weight for each defined group of assets.
	`pcpval`	Total portfolio value. Generates a constraint set to fix the total value of the portfolio.

Asset Data Specification

The efficient frontier computation functions require information about each of the assets in the portfolio. This data is entered into the function via two matrices: an expected return vector and a covariance matrix. The expected return vector contains the average expected return for each asset in the portfolio. The covariance matrix is a square matrix representing the interrelationships between pairs of assets. This information can be directly specified or can be estimated from an asset return time series with the function `ewstats`.

Example 1: Efficient Frontier

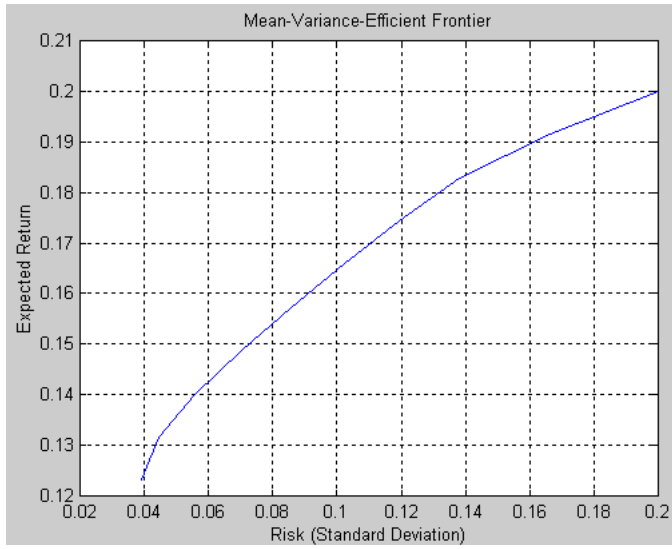
This example computes the efficient frontier of portfolios consisting of three different assets, using the function `frontcon`. To visualize the efficient frontier curve clearly, consider 10 different evenly spaced portfolios.

Assume that the expected return of the first asset is 10%, the second is 20%, and the third is 15%. The covariance is defined in the matrix `ExpCovariance`.

```
ExpReturn = [0.1 0.2 0.15];

ExpCovariance  = [0.005   -0.010    0.004 ;
                  -0.010    0.040   -0.002 ;
                   0.004   -0.002    0.023];

NumPorts = 10;
```

Since there are no constraints, you can call `frontcon` directly with the data you already have. If you call `frontcon` without specifying any output arguments, you get a graph representing the efficient frontier curve:

```
frontcon (ExpReturn, ExpCovariance, NumPorts);
```



Calling `frontcon` while specifying the output arguments returns the corresponding vectors and arrays representing the risk, return, and weights for each of the 10 points computed along the efficient frontier.

```
[PortRisk, PortReturn, PortWts] = frontcon(ExpReturn,...
ExpCovariance, NumPorts);
PortRisk =
    0.0392
    0.0445
    0.0559
    0.0701
    0.0858
    0.1023
    0.1192
    0.1383
    0.1661
    0.2000

PortReturn =
    0.1231
    0.1316
    0.1402
    0.1487
    0.1573
    0.1658
    0.1744
    0.1829
    0.1915
    0.2000

PortWts =
    0.7692    0.2308    0.0000
    0.6667    0.2991    0.0342
    0.5443    0.3478    0.1079
    0.4220    0.3964    0.1816
    0.2997    0.4450    0.2553
    0.1774    0.4936    0.3290
    0.0550    0.5422    0.4027
         0    0.6581    0.3419
         0    0.8291    0.1709
         0    1.0000    0.0000
```

The output data is represented row-wise. Each portfolio's risk, rate of return, and associated weights are identified as corresponding rows in the vectors and matrix.

For example, you can see from these results that the second portfolio has a risk of 0.0445, an expected return of 13.16%, and allocations of about 67% in the first asset, 30% in the second, and 3% in the third.

Portfolio Selection and Risk Aversion

One of the factors to consider when selecting the optimal portfolio for a particular investor is degree of risk aversion. This level of aversion to risk can be characterized by defining the investor's indifference curve. This curve consists of the family of risk/return pairs defining the trade-off between the expected return and the risk. It establishes the increment in return that a particular investor will require in order to make an increment in risk worthwhile. Typical risk aversion coefficients range between 2.0 and 4.0, with the higher number representing lesser tolerance to risk. The equation used to represent risk aversion in the Financial Toolbox is

```
U = E(r) – 0.005*A*sig^2
```

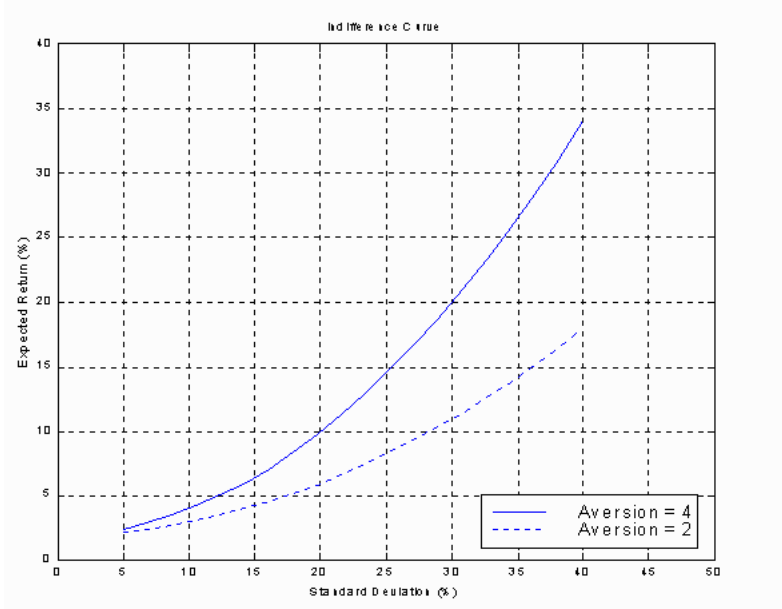
where:

`U` is the utility value.

`E(r)` is the expected return.

`A` is the index of investor's aversion.

`sig` is the standard deviation.



Example 2: Optimal Risky Portfolio

This example computes the optimal risky portfolio on the efficient frontier based upon the risk-free rate, the borrowing rate, and the investor's degree of risk aversion. You do this with the function `portalloc`.

First generate the efficient frontier data using either `portopt` or `frontcon`. This example uses `portopt` and the same asset data from Example 1:

```
ExpReturn = [0.1 0.2 0.15];

ExpCovariance  = [0.005   -0.010    0.004 ;
                 -0.010    0.040   -0.002 ;
                  0.004   -0.002    0.023];
```

This time consider 20 different points along the efficient frontier:

```
NumPorts = 20;

[PortRisk, PortReturn, PortWts] = portopt(ExpReturn,...
ExpCovariance, NumPorts);
```

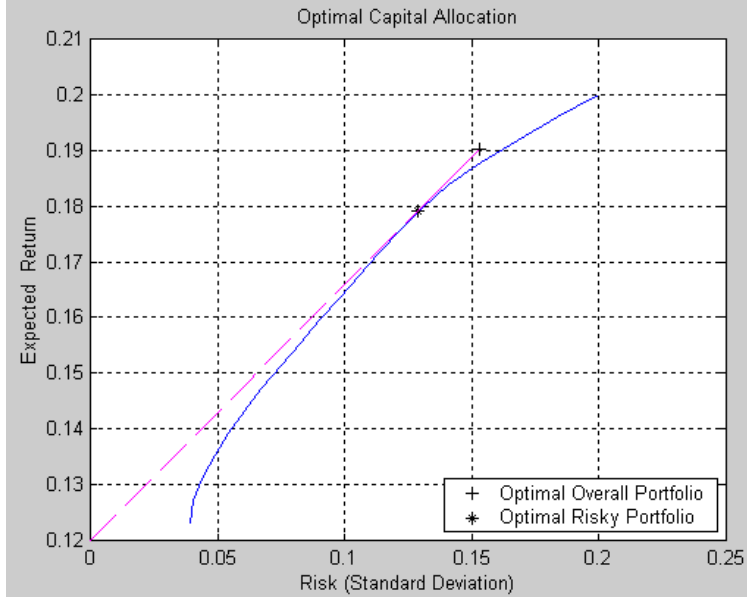
As with `frontcon`, calling `portopt` while specifying output arguments returns the corresponding vectors and arrays representing the risk, return, and weights for each of the portfolios along the efficient frontier. Use them as the first three input arguments to the function `portalloc`.

Now find the optimal risky portfolio and the optimal allocation of funds between the risky portfolio and the risk-free asset, using these values for the risk-free rate, borrowing rate and investor's degree of risk aversion:

```
RisklessRate  =  0.08
BorrowRate    =  0.12
RiskAversion  =  3
```

Calling `portalloc` without specifying any output arguments gives a graph displaying the critical points:

```
portalloc (PortRisk, PortReturn, PortWts, RisklessRate,...
BorrowRate, RiskAversion);
```



Calling `portalloc` while specifying the output arguments returns the variance (`RiskyRisk`), the expected return (`RiskyReturn`), and the weights (`RiskyWts`) allocated to the optimal risky portfolio. It also returns the fraction

(RiskyFraction) of the complete portfolio allocated to the risky portfolio, and the variance (OverallRisk) and expected return (OverallReturn) of the optimal overall portfolio. The overall portfolio combines investments in the risk-free asset and in the risky portfolio. The actual proportion assigned to each of these two investments is determined by the degree of risk aversion characterizing the investor.

```
[RiskyRisk, RiskyReturn, RiskyWts,RiskyFraction, OverallRisk,...
OverallReturn] = portalloc (PortRisk, PortReturn, PortWts,...
RisklessRate, BorrowRate, RiskAversion)
```

```
RiskyRisk = 0.1288
RiskyReturn = 0.1791
RiskyWts = 0.0057 0.5879 0.4064
RiskyFraction = 1.1869
OverallRisk = 0.1529
OverallReturn = 0.1902
```

The value of RiskyFraction exceeds 1 (100%), implying that the risk tolerance specified allows borrowing money to invest in the risky portfolio, and that no money will be invested in the risk-free asset. This borrowed capital is added to the original capital available for investment. In this example the customer will tolerate borrowing 18.69% of the original capital amount.

Example 3: Constraint Specification

This example computes the efficient frontier of portfolios consisting of three different assets, INTC, XON, and RD, given a list of constraints. The expected returns for INTC, XON, and RD are respectively as follows:

```
ExpReturn = [0.1 0.2 0.15];
```

The covariance matrix is:

```
ExpCovariance  =  [ 0.005   -0.010    0.004 ;
                   -0.010    0.040   -0.002 ;
                    0.004   -0.002    0.023];
```

Constraint 1: Allow short selling up to 10% of the portfolio value in any asset but limit the investment in any one asset to 110% of the portfolio value.

Constraint 2: Consider two different sectors, technology and energy, with the table below indicating the sector each asset belongs to.

Asset	INTC	XON	RD
Sector	Technology	Energy	Energy

Constrain the investment in the Energy sector to 80% of the portfolio value, and the investment in the Technology sector to 70%.

To solve this problem, use `frontcon`, passing in a list of asset constraints. Consider eight different portfolios along the efficient frontier:

```
NumPorts = 8;
```

To introduce the asset bounds constraints specified in Constraint 1, create the matrix `AssetBounds`, where each column represents an asset. The upper row represents the lower bounds, and the lower row represents the upper bounds:

```
AssetBounds = [-0.10, -0.10, -0.10;
                1.10,  1.10,  1.10];
```

Constraint 2 needs to be entered in two parts, the first part defining the groups, and the second part defining the constraints for each group. Given the information above, you can build a matrix of 1s and 0s indicating whether a specific asset belongs to a group. Each column represents an asset, and each row represents a group. This example has two groups: the technology group, and the energy group. Create the matrix `Groups` as follows:

```
Groups =  [ 0   1   1  ;
            1   0   0 ];
```

The `GroupBounds` matrix allows you to specify an upper and lower bound for each group. Each row in this matrix represents a group. The first column represents the minimum allocation, and the second column represents the maximum allocation to each group. Since the investment in the Energy sector is capped at 80% of the portfolio value, and the investment in the Technology sector is capped at 70%, create the `GroupBounds` matrix using this information.

```
GroupBounds = [ 0   0.80;
                0   0.70 ];
```

Now use frontcon to obtain the vectors and arrays representing the risk, return, and weights for each of the eight portfolios computed along the efficient frontier.

```
[PortRisk, PortReturn, PortWts] =  frontcon(ExpReturn,...
ExpCovariance,  NumPorts, [], AssetBounds, Groups, GroupBounds)

 PortRisk =
    0.0416
    0.0499
    0.0624
    0.0767
    0.0920
    0.1100
    0.1378
    0.1716

PortReturn  =
   0.1279
   0.1361
   0.1442
   0.1524
   0.1605
   0.1687
   0.1768
   0.1850

PortWts  =
    0.7000    0.2582    0.0418
    0.6031    0.3244    0.0725
    0.4864    0.3708    0.1428
    0.3696    0.4172    0.2132
    0.2529    0.4636    0.2835
    0.2000    0.5738    0.2262
    0.2000    0.7369    0.0631
    0.2000    0.9000   -0.1000
```

The output data is represented row-wise, where each portfolio's risk, rate of return, and associated weight is identified as corresponding rows in the vectors and matrix.

Linear Constraint Equations

While `frontcon` allows you to enter a fixed set of constraints related to minimum and maximum values for groups and individual assets, you often need to specify a larger and more general set of constraints when finding the optimal risky portfolio. The function `portopt` addresses this need, by accepting an arbitrary set of constraints as an input matrix.

The auxiliary function `portcons` can be used to create the matrix of constraints, with each row representing an inequality. These inequalities are of the type `A*Wts' <= b`, where `A` is a matrix, `b` is a vector, and `Wts` is a row vector of asset allocations. The number of columns of the matrix `A`, and the length of the vector `Wts` correspond to the number of assets. The number of rows of the matrix `A`, and the length of vector `b` correspond to the number of constraints. This method allows you to specify any number of linear inequalities to the function `portopt`.

In actuality, `portcons` is an entry point to a set of functions that generate matrices for specific types of constraints. `portcons` allows you to specify all the constraints data at once, while the specific portfolio constraint functions allow you to build the constraints incrementally. These constraint functions are `pcpval`, `pcalims`, `pcglims`, and `pcgcomp`.

Consider an example to help understand how to specify constraints to `portopt` while bypassing the use of `portcons`. This example requires specifying the minimum and maximum investment in various groups.

Table 1-1: Maximum and Minimum Group Exposure

Group	Minimum Exposure	Maximum Exposure
North America	0.30	0.75
Europe	0.10	0.55
Latin America	0.20	0.50
Asia	0.50	0.50

Note that the minimum and maximum exposure in Asia is the same. This means that you require a fixed exposure for this group.

Also assume that the portfolio consists of three different funds. The correspondence between funds and groups is shown in Table 1-2.

Table 1-2: Group Membership

Group	Fund 1	Fund 2	Fund 3
North America	X	X	
Europe			X
Latin America	X		
Asia		X	X

Using the information in these two tables, build a mathematical representation of the constraints represented. Assume that the vector of weights representing the exposure of each asset in a portfolio is called `Wts = [W1 W2 W3]`.

Specifically:

1. $W1 + W2 \geq 0.30$
2. $W1 + W2 \leq 0.75$
3. $W3 \geq 0.10$
4. $W3 \leq 0.55$
5. $W1 \geq 0.20$
6. $W1 \leq 0.50$
7. $W2 + W3 = 0.50$

Since you need to represent the information in the form `A*Wts <= b`, multiply equations 1, 3 and 5 by –1. Also turn equation 7 into a set of two inequalities: $W2 + W3 \geq 0.50$ and $W2 + W3 \leq 0.50$ (The intersection of these two inequalities is the equality itself.). Thus:

1. $-W1 - W2 \leq -0.30$
2. $W1 + W2 \leq 0.75$
3. $-W3 \leq -0.10$

4. $W3 \le 0.55$
5. $-W1 \le -0.20$
6. $W1 \le 0.50$
7. $-W2 - W3 \le -0.50$
8. $W2 + W3 \le 0.50$

Bringing these equations into matrix notation gives:

```
A =
[ −1    −1     0;
   1     1     0;
   0     0    −1;
   0     0     1;
  −1     0     0;
   1     0     0;
   0    −1    −1;
   0     1     1]

b =
[−0.30;
  0.75;
 −0.10;
  0.55;
 −0.20;
  0.50;
 −0.50;
  0.50]
```

Build the constraint matrix `ConSet` by concatenating the matrix `A` to the vector `b`:

```
ConSet = [A, b]
```

Specifying Additional Constraints

The example above defined a constraints matrix that specified a set of typical scenarios. It defined groups of assets, specified upper and lower bounds for total allocation in each of these groups, and it set the total allocation of one of the groups to a fixed value. Constraints like these are common occurrences. The function `portcons` was created to simplify the creation of the constraint

matrix for these and other common portfolio requirements. portcons takes as input arguments a list of constraint-specifier strings, followed by the data necessary to build the constraint specified by the strings.

Assume that you need to add some more constraints to the previous example. Specifically, add a constraint indicating that the sum of weights in any portfolio should be equal to 1, and another set of constraints (one per asset) indicating that the weight for each asset must greater than 0. This translates into five more constraint rows: two for the new equality, and three indicating that each weight must be greater or equal to 0. The total number of inequalities in the example is now 13. Clearly, creating the constraint matrix can turn into a tedious task.

To create the new constraint matrix using portcons, use two separate constraint-specifier strings:'Default', which indicates that each weight is greater than 0 and that the total sum of the weights adds to 1; and 'GroupLims', which defines the minimum and maximum allocation on each group. The only data requirement for the constraint-specifier string 'Default' is NumAssets (the total number of assets). The constraint-specifier string 'GroupLims' requires three different arguments: a Groups matrix indicating the assets that belong to each group, the GroupMin vector indicating the minimum bounds for each group, and the GroupMax vector indicating the maximum bounds for each group. Based on Table 1-2, build the Group matrix, with each row representing a group, and each column representing an asset:

```
Group =
[1    1    0;
 0    0    1;
 1    0    0;
 0    1    1]
```

Table 1-1 has the information to build GroupMin and GroupMax:

```
GroupMin = [0.30  0.10  0.20  0.50];
GroupMax = [0.75  0.55  0.50  0.50];
```

Given that the number of assets is three, build the constraint matrix by calling portcons:

```
ConSet = portcons('Default', 3, 'GroupLims', Group, GroupMin,...
GroupMax);
```

In most cases, `portcons('Default')` returns the minimal set of constraints required for calling `portopt`. If `ConSet` is not specified in the call to `portopt`, the function calls `portcons` passing `'Default'` as its only specifier.

Now use `portopt` to obtain the vectors and arrays representing the risk, return, and weights for the portfolios computed along the efficient frontier.

```
[PortRisk, PortReturn, PortWts] = portopt(ExpReturn,...
ExpCovariance, [],  [],  ConSet);
```

```
PortRisk = 0.0586
Port Return = 0.1375
PortWts = 0.5 0.25 0.25
```

In this case the constraints allow only one optimal portfolio.

Solving Sample Problems with the Toolbox

This section shows how Financial Toolbox functions solve real-world problems. The examples ship with the toolbox as M-files. Try them by entering the commands directly or by executing the M-files.

Example 1: Sensitivity of Bond Prices to Changes in Interest Rates

Macaulay and **modified duration** measure the sensitivity of a bond's price to changes in the level of interest rates. **Convexity** measures the change in duration for small shifts in the yield curve, and thus measures the second-order price sensitivity of a bond. Both measures can gauge the vulnerability of a bond portfolio's value to changes in the level of interest rates.

Alternatively, analysts can use duration and convexity to construct a bond portfolio that is partly hedged against small shifts in the term structure. If one chooses bonds and combines them in a portfolio whose duration is zero, then the portfolio is insulated, to some extent, against interest rate changes. If the portfolio convexity is also zero, then this insulation is even better. However, since hedging costs money or reduces expected return, it is important to know how much protection results from hedging duration alone compared with hedging both duration and convexity.

This example demonstrates a way to analyze the relative importance of duration and convexity for a bond portfolio. Using duration, it constructs a first-order approximation of the change in portfolio price to a level shift in interest rates. Then, using convexity, it calculates a second-order approximation. Finally it compares the two approximations with the true price change resulting from a change in the yield curve. This example M-file is `ftspex1.m`.

Step 1. Define three bonds using values for these parameters: settlement date, maturity date, face value, coupon rate, number of coupons per year, and day-count basis. In the `Bonds` matrix, each row represents a bond and each column represents a parameter. You can't mix numbers and strings in a

matrix, so today gives a date number for the settlement date and datenum gives a date number for the maturity date. Pick yields from the yield curve.

```
          Settle   Maturity                Face Rate   # Basis

Bonds = [today datenum('17-jun-2010')  100 0.07  2 0
         today datenum('09-jun-2015')  100 0.06  2 0
         today datenum('14-may-2025') 1000 0.045 2 0];

yields = [0.05 0.06 0.065];
format bank;
```

Step 2. Use Financial Toolbox functions to calculate the price, modified duration in years, and convexity in years of each bond. Store these numbers in preallocated vectors to improve performance. The true price is quoted price plus accrued interest, the two values prbond returns.

```
temp = zeros(1,2);
prices = zeros(1,3);
durations = zeros(1,3);
convexities = zeros(1,3);

for i = 1:3
    [temp(1), temp(2)] = prbond(Bonds(i,1), Bonds(i,2),...
        Bonds(i,3), Bonds(i,4), yields(i),...
        Bonds(i,5), Bonds(i,6));

    prices(i) = temp(1) + temp(2);

    [temp(1), temp(2)] = bonddur(Bonds(i,1), Bonds(i,2),...
        Bonds(i,3), Bonds(i,4), yields(i),...
        Bonds(i,5), Bonds(i,6));

    durations(i) = temp(2);

    [temp(1), temp(2)] = bondconv(Bonds(i,1),Bonds(i,2),...
        Bonds(i,3), Bonds(i,4), yields(i),...
        Bonds(i,5), Bonds(i,6));

    convexities(i) = temp(2);
end
```

Step 3: Choose a hypothetical amount by which to shift the yield curve (here, 0.2 percentage point or 20 basis points).

```
dY = 0.002;
```

Weight the three bonds equally, and calculate the actual dollar amount of each bond in the portfolio, which has a total value of $100,000.

```
portf_price = 100000;
portf_weights = ones(1,3)/3.0;
portf_amnts = portf_price * portf_weights ./ prices;
```

Step 4: Calculate the modified duration and convexity of the portfolio. Use the fact that the portfolio duration or convexity is a weighted average of the durations or convexities of the individual bonds, where the weights are the same as the bonds in the portfolio. Calculate the first- and second-order approximations of the percent price change as a function of the change in the level of interest rates.

```
portf_duration = sum(portf_weights .* durations)
portf_convexity = sum(portf_weights .* convexities)

perc_approx1 = –portf_duration * dY * 100;
perc_approx2 = perc_approx1 + portf_convexity * dY^2 * 100/2.0;
```

Step 5: Estimate the new portfolio price using the two estimates for the percent price change.

```
price_approx1 = portf_price + perc_approx1 * portf_price /100
price_approx2 = portf_price + perc_approx2 * portf_price /100
```

Step 6: Calculate the true new portfolio price.

```
new_prices = zeros(1,3);

for i = 1:3
    [temp(1), temp(2)] = prbond(Bonds(i,1), Bonds(i,2), ...
        Bonds(i,3), Bonds(i,4), yields(i)+dY,...
        Bonds(i,5), Bonds(i,6));

    new_prices(i) = temp(1) + temp(2);
end

new_price = sum(portf_amnts .* new_prices)
```

The analysis results are as follows: (If you run this example, the answers will differ since the settlement date is today's date.)

- The original portfolio price was $100,000.
- The yield curve shifted up by 0.2 percentage point or 20 basis points.
- The first-order approximation, based on modified duration, predicts the new portfolio price will be $97,693.78.
- The second-order approximation, based on duration and convexity, predicts the new portfolio price will be $97,733.83.
- The true new portfolio price for this yield curve shift is $97,733.28.

The estimate using duration and convexity is quite good (for this fairly small shift in the yield curve), but only about 0.04% better than the estimate using duration alone. The importance of convexity increases as the magnitude of the yield curve shift increases. Try a larger shift to see this effect.

The approximation formulas in this example consider only parallel shifts in the term structure, because both formulas are functions of `dY`, the change in yield. The formulas are not well-defined unless each yield changes by the same amount. In actual financial markets, changes in yield curve level typically explain a substantial portion of bond price movements. However, other changes in the yield curve, such as slope, may also be important and are not captured here. Also, both formulas give local approximations whose accuracy deteriorates as `dY` increases in size. You can demonstrate this by running the program with larger values of `dY`.

Example 2: Constructing a Bond Portfolio to Hedge Against Duration and Convexity

This example constructs a bond portfolio to hedge the portfolio of Example 1. It assumes a long position in (holding) the portfolio of Example 1, and that three other bonds are available for hedging. It chooses weights for these three other bonds in a new portfolio so that the duration and convexity of the new portfolio match those of the original portfolio. Taking a short position in the new portfolio, in an amount equal to the value of the first portfolio, partially hedges against parallel shifts in the yield curve.

This example also uses the fact that the portfolio duration or convexity is a weighted average of the durations or convexities of the individual bonds, where the weights are the same as the bonds in the portfolio. As in Example 1, it uses modified duration in years and convexity in years. The hedging problem therefore becomes one of solving a system of linear equations, which is very easy in MATLAB. The M-file for this example is `ftspex2.m`.

Step 1: Set up three bonds in the new portfolio. Use three since you want to hedge against duration and convexity and constrain total portfolio price. As before create a matrix where the rows are the bonds and the columns are settlement date, maturity date, face value, coupon rate, number of coupons per year, and day-count basis. Since you can't mix numbers and strings in a matrix, use `today` to give a date number for settlement date, and `datenum` to give a date number for maturity date. Define a yield curve corresponding to the three bonds.

```
            Settle   Maturity                Face Rate   # Basis

Bonds = [today datenum('15-jun-2005')   500 0.07  2 0
         today datenum('2-oct-2010')   1000 0.066 2 0
         today datenum('1-mar-2025')    250 0.08  2 0];

yields = [0.06 0.07 0.075];
format bank;
```

Step 2: Use Financial Toolbox functions to compute price, modified duration, and convexity for all three bonds. Store these numbers in preallocated vectors

to improve performance. The true price is quoted price plus accrued interest, the two values prbond returns.

```
temp = zeros(1,2);
prices = zeros(1,3);
durations = zeros(1,3);
convexities = zeros(1,3);

for i = 1:3
    [temp(1), temp(2)] = prbond(Bonds(i,1), Bonds(i,2),...
        Bonds(i,3), Bonds(i,4), yields(i),...
        Bonds(i,5), Bonds(i,6));
    prices(i) = temp(1) + temp(2);

    [temp(1), temp(2)] = bonddur(Bonds(i,1), Bonds(i,2),...
        Bonds(i,3), Bonds(i,4), yields(i),...
        Bonds(i,5), Bonds(i,6));
    durations(i) = temp(2);

    [temp(1), temp(2)] = bondconv(Bonds(i,1),Bonds(i,2),...
        Bonds(i,3), Bonds(i,4), yields(i),...
        Bonds(i,5), Bonds(i,6));
    convexities(i) = temp(2);
end
```

Step 3: Set up and solve the system of linear equations whose solution is the weights of the new bonds in a new portfolio with the same duration and convexity as in Example 1. In addition, scale the weights to sum to one; that is, force them to be portfolio weights. Then you can scale up this "unit portfolio" to have the same price as the original portfolio. (If you run this example, your values will differ since today's date is the settlement date.)

```
A = [durations
     convexities
     1 1 1];

b = [ 11.53
     200.27
       1];

weights = A\b;
```

Step 4: Check that the duration and convexity of the hedge portfolio match those of the portfolio of Example 1.

```
portf_duration = sum(weights .* durations')
portf_convexity = sum(weights .* convexities')
```

Step 5: Finally, scale up the unit portfolio to match the value of the portfolio in Example 1.

```
portf_value = 100000;

hedge_amounts = weights' ./ prices * portf_value
```

The result

```
hedge_amounts = [-98.82  101.00  201.31]
```

is a vector containing the amount of each bond to combine in a new portfolio worth $100,000 today, and whose duration and convexity match those of the portfolio in Example 1. If you are holding that first portfolio, you can hedge by taking a short position in the new portfolio.

Just as the approximation formulas of Example 1 are appropriate only for small parallel shifts in the yield curve, the hedge portfolio of Example 2 is appropriate only for reducing the impact of small level changes in the term structure.

Example 3: Visualizing the Sensitivity of a Bond Portfolio's Price to Parallel Shifts in the Yield Curve

Often bond portfolio managers want to consider more than just the sensitivity of a portfolio's price to a small shift in the yield curve, particularly if the investment horizon is long. This example shows how MATLAB can visualize the price behavior of a portfolio of bonds over a wide range of yield curve scenarios, and as time progresses toward maturity.

This example uses the Financial Toolbox bond pricing functions in a routine that takes time-to-maturity and yield as input arguments and returns the price of a portfolio of bonds. It plots the price and shows the behavior of bond prices as yield and time vary. This example M-file is `ftspex3.m`.

Step 1: Set up a matrix of four bonds where the rows are the bonds and the columns give settlement date, maturity date, face value, coupon rate, number of coupons per year, and day-count basis. You can't mix numbers and strings in a matrix, so `datenum` supplies numbers for the dates.

```
   Settle                Maturity                Face   Rate   # Basis

Bonds = ...
[datenum('15-jan-95') datenum('3-apr-2020')   1000  0      1  3
 datenum('15-jan-95') datenum('14-may-2025')  1000  0.05   2  3
 datenum('15-jan-95') datenum('9-jun-2019')   1000  0      1  3
 datenum('15-jan-95') datenum('25-feb-2019')  1000  0.055  2  3];
```

Set yields for the bonds, and preallocate space for vectors to improve performance.

```
yields = [0.078  0.09  0.075  0.085];

temp = zeros(1,2);
prices = zeros(1,4);
```

Use the Financial Toolbox `prbond` function to compute the bond prices. Note that the actual prices are quoted prices plus accrued interest, the two values returned by `prbond`.

```
[temp1, temp2] = prbond(Bonds(:,1),Bonds(:,2),...
                 Bonds(:,3),Bonds(:,4),yields(:),...
                 Bonds(:,5),Bonds(:,6));

prices = temp1 + temp2;
```

Step 2: Determine the quantity of each bond (`bond_amnts`), so that each bond's cash value is $25,000, in a portfolio with a $100,000 total value:

```
bond_amnts = 25000 ./ prices;
```

Step 3: Plot the price of the portfolio as a function of settlement date and a range of yields, and as a function of the change in yield (`dY`). This plot illustrates the interest rate sensitivity of the portfolio as time progresses (`dT`), under a range of interest rate scenarios.

Set the yield range from -5% to +5% in 0.5% increments, and set a vector for the time horizon in 18-month increments from the settlement date.

```
dYvect = –0.05:0.005:0.05;

i = 1:15;
dTvect = datemnth(datenum('14-May-1995')*ones(size(i)), 18*i);
```

Note the length of these yield and time vectors, and allocate space for the price matrix.

```
num_dY = length(dYvect);
num_dT = length(dTvect);

price_mat = zeros(num_dY, num_dT);
```

Loop over the yield range and loop over the time horizon to compute the portfolio price, which is the sum of individual prices, and save the values in the price matrix.

```
for i = 1:num_dY
    for j = 1:num_dT

        [temp1, temp2] =...
           prbond(dTvect(j),Bonds(:,2),Bonds(:,3),...
            Bonds(:,4),yields(:)+dYvect(i),...
            Bonds(:,5),Bonds(:,6));

        prices = temp1 + temp2;

        price_mat(i,j) = sum(bond_amnts .* prices);
    end
end
```

Create a new figure window, and plot the price as a surface plot.

```
figure;
surf(dTvect, dYvect, price_mat);
```

Add the base price plotted as a plane, and compute the surface,

```
hold on
basemesh = mesh(dTvect, dYvect, 100000*ones(num_dY, num_dT));
```

make it transparent, plot it so the price surface shows through, and draw a box around the plot.

```
set(basemesh,'facecolor','none');
set(basemesh,'edgecolor','m');
set(gca,'box','on');
```

Set up a vector for the *x*-axis, set the *x*-axis tick spacing to three-year intervals, starting from the settlement date until the farthest maturity date,

```
dl = [datenum('14-May-1995'):3*365:datenum('14-May-2025')];
set(gca,'xlim',[min(dl),max(dl)]);
```

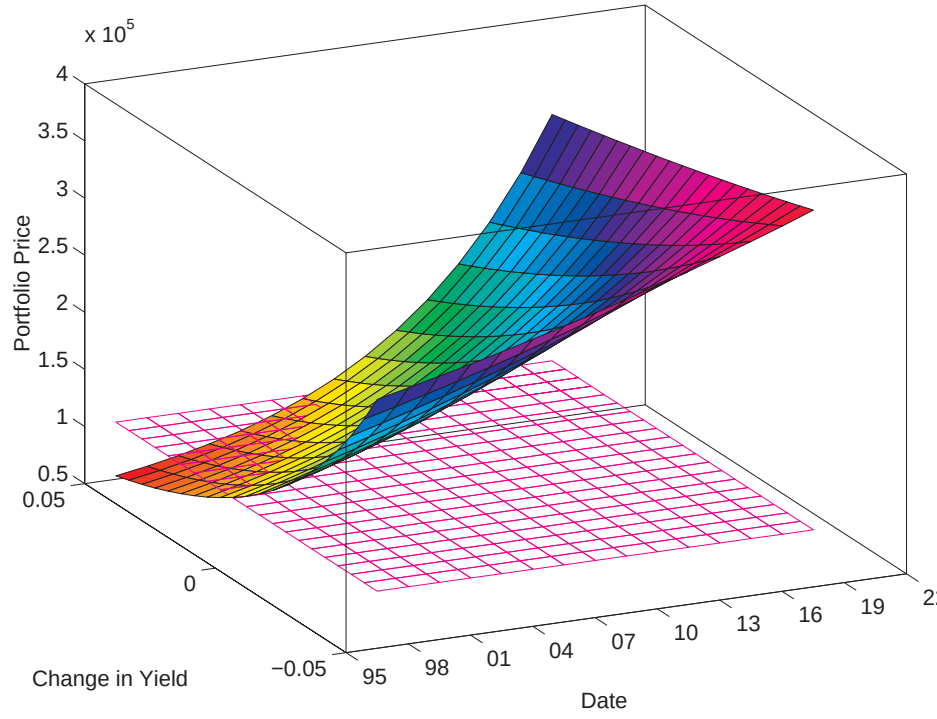
and plot the *x*-axis using two-digit year labels for ticks.

```
set(gca,'xtick',dl);
dateaxis('x',11);
```

Add axis labels and set the viewpoint. MATLAB produces the figure. On a color monitor, it is quite striking. The black-and-white reproduction here cannot show the color gradations and surface curvature.

```
xlabel('Date');
ylabel('Change in Yield');
zlabel('Portfolio Price');
hold off

view(-25,25);
```

Note: Because of the large nested loops, it takes about 2 minutes to run this example on a 200 MHz Pentium-based PC.



MATLAB's 3-D graphics have visualized the landscape of interest rate risk, or repricing risk, experienced by a bond portfolio over time. This example assumed parallel shifts in the term structure, but it might similarly have allowed the level and slope to vary, etc.

Example 4: Constructing Greek-Neutral Portfolios of European Stock Options

The six price sensitivity measures that most option traders use are named for Greek letters, and are called the "greeks": delta, gamma, lambda, rho, theta, and vega. Delta is the sensitivity of an option portfolio's price to changes in the price of the underlying asset. Vega is sensitivity to changes in the volatility of the underlying asset. See "Pricing and Analyzing Equity Derivatives" or the *Glossary* for other definitions.

The greeks of a particular option are a function of the model used to price the option. However, given enough different options to work with, a trader can construct a portfolio with any desired values for its greeks.

To reduce the vulnerability of a portfolio to changes in the price of the underlying asset, one trader might construct a portfolio whose delta is zero. The portfolio is then said to be "delta neutral." Another trader might believe that the volatility of a certain stock will increase beyond market expectations, and will thus try to construct a portfolio with a very large vega, and buy it.

This example creates two equity option portfolios, each hedged (according to the Black-Scholes model for European options) against one or more greeks. It uses the fact that the value of a particular greek for an option portfolio is a weighted average of the value of the same greek for each individual option, where the weights are those of the options in the portfolio. Hedging thus becomes a case of solving a system of linear equations, which is very easy in MATLAB. This example M-file is `ftspex4.m`.

Part I: Create a delta-neutral portfolio of two options with a $1,000 total value.

Step 1: Create a matrix of option data. Each row contains the standard inputs to the Financial Toolbox Black-Scholes functions: current price, strike price, risk-free interest rate, years to maturity, volatility, and dividend rate. Column 7 indicates a put (0) or a call (1) option. Set MATLAB to its standard bank format.

```
              Price      Strike   Rate  Yrs   Vol    Div      Put/Call
optmat = [100       100     0.1   0.2  0.3   0        1
          119.1     125     0.25  0.2  0.2   0.025    0
           87.2      85     0.19  0.1  0.23  0        1
          301.125   315     0.1   0.5  0.25  0.0333   0];

format bank
```

Step 2: Calculate the Black-Scholes prices of the first two options in `optmat`. The standard MATLAB function `diag` gets the call price of the first option and the put price of the second.

```
i = 1:2;

[temp1, temp2] = blsprice(optmat(i,1),optmat(i,2),...
           optmat(i,3),optmat(i,4),optmat(i,5),optmat(i,6));

prices = diag([temp1, temp2])';
```

Step 3: Calculate the Black-Scholes deltas of the two options used to build our hedged portfolio. `diag` gets the call delta for option one and the put delta for option two.

```
[temp1, temp2] = blsdelta(optmat(i,1),optmat(i,2),...
           optmat(i,3),optmat(i,4),optmat(i,5),optmat(i,6));

deltas = diag([temp1, temp2])';
```

Step 4: Set up and solve the matrix equation that expresses the hedging problem. Determine the amounts of each option for a portfolio consisting of the first two options in optmat so that the portfolio is delta-neutral (locally hedged against delta) and has a value of $1,000. Calculate the option values:

```
A = [deltas
     prices];

b = [0
     1000];

amounts = A\b

values = amounts .* prices'
```

which yields

```
amounts =
         86.24
        102.09
values =
        547.10
        452.90
```

Part II: Using all four options in optmat, construct a portfolio that is hedged locally against delta, gamma, and vega, and has a total value of $17,000.

Step 5: Calculate prices and greeks for all four options.

```
[temp1, temp2] = blsprice(optmat(:,1),optmat(:,2),...
          optmat(:,3),optmat(:,4),optmat(:,5),optmat(:,6));
prices = [temp1(1) temp2(2) temp1(3) temp2(4)];

[temp1, temp2] = blsdelta(optmat(:,1),optmat(:,2),...
          optmat(:,3),optmat(:,4),optmat(:,5),optmat(:,6));
deltas = [temp1(1) temp2(2) temp1(3) temp2(4)];

gammas = blsgamma(optmat(:,1),optmat(:,2),...
          optmat(:,3),optmat(:,4),optmat(:,5),optmat(:,6));

vegas = blsvega(optmat(:,1),optmat(:,2),...
          optmat(:,3),optmat(:,4),optmat(:,5),optmat(:,6));
```

Step 6: Set up and solve the matrix equations.

```
A = [deltas
     gammas'
     vegas'
     prices];

b = [0
     0
     0
     17000];

portf_amounts = [A\b]';
```

Step 7: Generate and display the matrix.

```
portf_weights = portf_amounts .* prices / 17000;
portf_values = portf_weights * 17000;

portf_infomat = [portf_weights
                 portf_values
                 portf_amounts]
```

The first row is the desired portfolio weights, the second row is the actual dollar amounts of each option, and the third row is the number of units of each option. Negative numbers indicate sales.

```
portf_infomat =
        -11.88          -2.24           6.67           8.45
    -201924.23      -38076.19      113411.08      143589.33
     -31828.60       -8583.28       23507.90        6306.25
```

Note that delta and vega are first-order sensitivity measures. Gamma, the sensitivity of delta to changes in stock price, is a second-order sensitivity measure. Hedges based on these measures are effective only for small changes in the respective variables.

Producing Graphics with the Toolbox

The Financial Toolbox and MATLAB graphics functions work together to produce presentation quality graphics, as these examples show. The examples ship with the toolbox as M-files. Try them by entering the commands directly or by executing the M-files. (These examples assume you have a color monitor to display output. Black-and-white printing limits the reproductions here.) For more information on using the graphics commands (axis labels, color, line width, plotting points as circles, titles, etc.) please see the "Graphics" section of *Getting Started with MATLAB*. See *Using MATLAB Graphics* for details of specific graphics functions.

Example 1: Plotting an Efficient Frontier

This example plots the efficient frontier from a portfolio analysis using the same data as "Analyzing Portfolios" on page 1-48, except that this example enters asset returns as decimal percentages, not decimal fractions. It produces a graph showing a set of portfolios that maximizes expected return at each level of portfolio risk. This example M-file is `ftgex1.m`.

Create a portfolio of six fictional assets having random returns for 50 time periods. First initialize the MATLAB random number generator and create the six assets with 50 observations and mean return = 25%.

```
rand('seed', 65537);
asset = rand(50,6) * 50;
```

Create an associated set of random expected rates of return for each asset, with mean rate = 10%.

```
raret = rand(1,6) * 20;
```

Execute the efficient frontier function to return 10 different points along that frontier.

```
[risk, ror] = frontier(asset,  raret,  10);
```

Find the minimum risk point and its rate of return along the efficient frontier.

```
minr = min(risk);
minror = max(ror(find(risk == minr)));
```

Plot the frontier as a cyan line from the minimum yield to the maximum, and hold it to add to the plot.

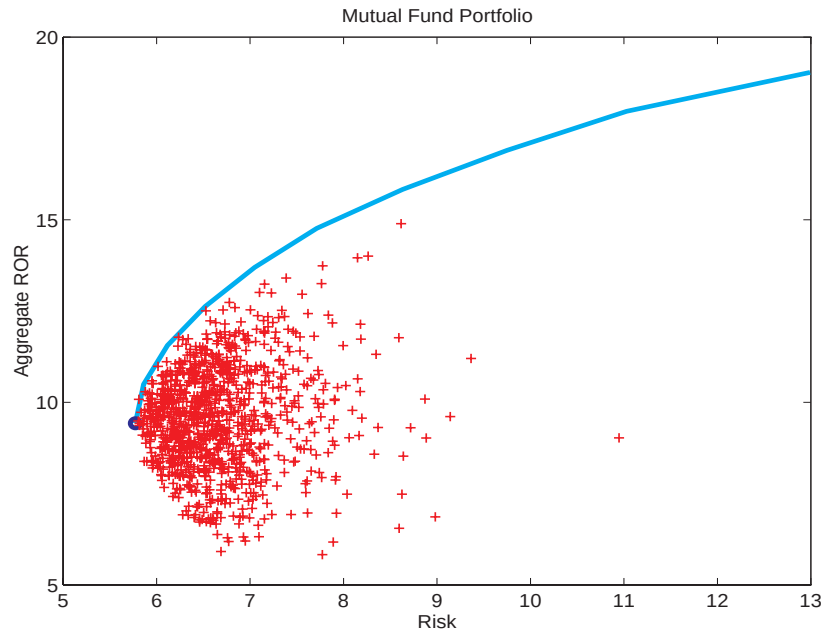
```
plot(risk, ror, 'c', 'linewidth',3, 'erase','none');
hold on;
```

Find the minimum risk point and its rate of return along the efficient frontier, and plot that point as a blue circle.

```
minr = min(risk);
minror = max(ror(find(risk == minr)));
plot(minr, minror, 'bo', 'linewidth',3, 'erasemode','normal');
```

Finally generate 1000 random portfolio configurations, plot them as red '+' signs, add axis labels and a title, and show how the efficient frontier compares with those random choices.

```
[randrisk,  randror] = portrand(asset, raret, 1000);
plot(randrisk, randror, 'r+', 'linewidth',1);
xlabel('Risk')
ylabel('Aggregate ROR');
title('Mutual Fund Portfolio');
```



Example 2: Plotting Sensitivities of an Option

This example creates a three-dimensional plot showing how gamma changes relative to price for a Black-Scholes option. Recall that gamma is the second derivative of the option price relative to the underlying security price. The plot shows a three-dimensional surface whose *z*-value is the gamma of an option as price (*x*-axis) and time (*y*-axis) vary. It adds yet a fourth dimension by showing option delta (the first derivative of option price to security price) as the color of the surface. This example M-file is `ftgex2.m`.

First set the price range of the options, and set the time range to one year divided into half-months and expressed as fractions of a year.

```
range = 10:70;
span = length(range);
j = 1:0.5:12;
newj = j(ones(span,1),:)'/12;
```

For each time period create a vector of prices from 10 to 70 and create a matrix of all ones.

```
jspan = ones(length(j),1);
newrange = range(jspan,:);
pad = ones(size(newj));
```

Call the toolbox gamma and delta sensitivity functions. Exercise price is $40, risk-free interest rate is 10%, and volatility is 0.35 for all prices and periods. Gamma is the *z*-axis, delta is the color.

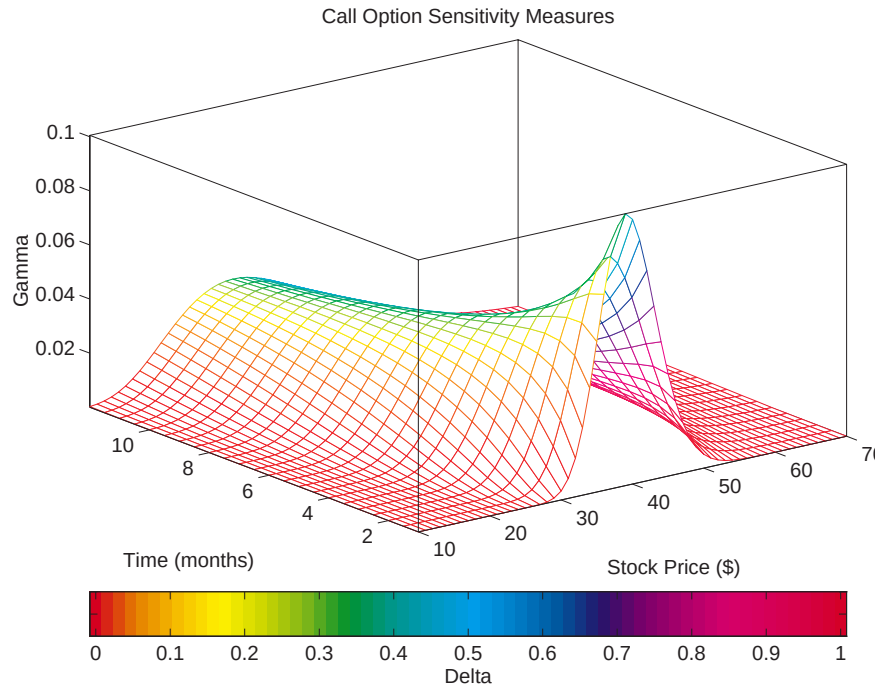
```
zval = blsgamma(newrange, 40*pad, 0.1*pad, newj, 0.35*pad);
color = blsdelta(newrange, 40*pad, 0.1*pad, newj, 0.35*pad);
```

Draw the surface as a mesh, add axis labels and a title. The axes range from 10 to 70, 1 to 12, and -∞ to ∞.

```
mesh(range, j, zval, color);
xlabel('Stock Price ($)');
ylabel('Time (months)');
zlabel('Gamma');
title('Call Option Sensitivity Measures');
axis([10 70  1 12  -inf inf]);
```

Finally add a box around the whole plot, annotate the colors with a bar and label the colorbar. This black-and-white reproduction only hints at the output you see on a color monitor.

```
set(gca, 'box', 'on');
colorbar('horiz');
a = findobj(gcf, 'type', 'axes');
set(get(a(2), 'xlabel'), 'string', 'Delta');
```



Example 3: Plotting Sensitivities of a Portfolio of Options

This example plots gamma as a function of price and time for a portfolio of 10 Black-Scholes options. The plot shows a three-dimensional surface. For each point on the surface, the height (*z*-value) represents the sum of the gammas for each option in the portfolio weighted by the amount of each option. The *x*-axis represents changing price, and the *y*-axis represents time. The plot adds a fourth dimension by showing delta as surface color. This example M-file is `ftgex3.m`.

First set up the portfolio with arbitrary data. Current prices range from $20 to $90 for each option. Set corresponding exercise prices for each option.

```
range = 20:90;
plen = length(range);
exprice = [75 70 50 55 75 50 40 75 60 35];
```

Set all risk-free interest rates to 10%, and set times to maturity in days. Set all volatilities to 0.35. Set the number of options of each instrument, and allocate space for matrices.

```
rate = 0.1*ones(10,1);
time = [36  36  36  27  18  18  18  9  9  9];
sigma = 0.35*ones(10,1);
numopt = 1000*[4  8  3  5  5.5  2  4.8  3  4.8  2.5];
zval = zeros(36, plen);
color = zeros(36, plen);
```

For each instrument, create a matrix (of size `time` by `plen`) of prices for each period.

```
for i = 1:10
    pad = ones(time(i),plen);
    newr = range(ones(time(i),1),:);
```

Create a vector of time periods 1 to `time`; and a matrix of times, one column for each price.

```
    t = (1:time(i))';
    newt = t(:,ones(plen,1));
```

Call the toolbox gamma and delta sensitivity functions to compute gamma and delta.

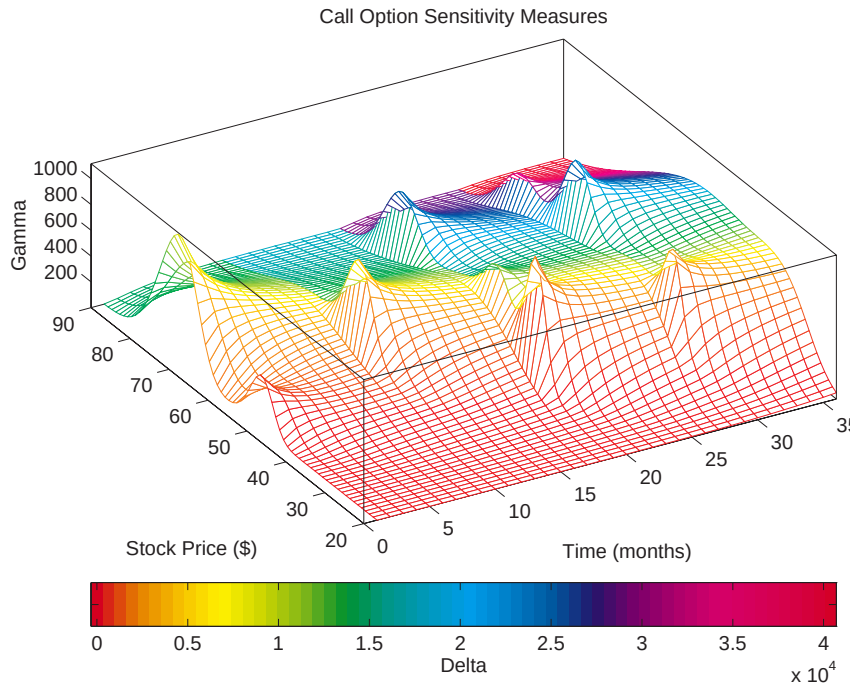
```
        zval(36–time(i)+1:36,:) = zval(36–time(i)+1:36,:) ...
            + numopt(i) * blsgamma(newr, exprice(i)*pad, ...
            rate(i)*pad, newt/36, sigma(i)*pad);

        color(36–time(i)+1:36,:) = color(36–time(i)+1:36,:) ...
            + numopt(i) * blsdelta(newr, exprice(i)*pad, ...
            rate(i)*pad, newt/36, sigma(i)*pad);
    end
```

Draw the surface as a mesh, set the viewpoint, and reverse the *x*-axis because of the viewpoint. The axes range from 20 to 90, 0 to 36, and -∞ to ∞.

```
    mesh(range, 1:36, zval, color);
    view(60,60);
    set(gca, 'xdir','reverse');
    axis([20 90  0 36  –inf inf]);
```

Add a title and axis labels and draw a box around the plot. Annotate the colors with a bar and label the colorbar. A color monitor shows all the details; this reproduction is only approximate.

```
title('Call Option Sensitivity Measures');
xlabel('Stock Price ($)');
ylabel('Time (months)');
zlabel('Gamma');
set(gca, 'box', 'on');
colorbar('horiz');
a = findobj(gcf, 'type', 'axes');
set(get(a(2), 'xlabel'), 'string', 'Delta');
```



2

Reference

This chapter contains detailed descriptions of all the functions in the Financial Toolbox. It first groups the functions in task categories and then explains each function in alphabetical order. You may also access the reference material online by typing

```
help function_name
```

Handling and Converting Dates

	Current Time and Date
now	Current date and time.
today	Current date.

	Date and Time Components
datefind	Indices of date numbers in matrix.
datevec	Date components.
day	Day of month.
eomdate	Last date of month.
eomday	Last day of month.
hour	Hour of date or time.
lweekdate	Date of last occurrence of weekday in month.
minute	Minute of date or time.
month	Month of date.
months	Number of whole months between dates.
nweekdate	Date of specific occurrence of weekday in month.
second	Second of date or time.
weekday	Day of the week.
year	Year of date.
yeardays	Number of days in year.

	Date Conversion
datenum	Create date number.
datestr	Create date string.
m2xdate	MATLAB serial date number to Excel serial date number.
x2mdate	Excel serial date number to MATLAB serial date number.

	Financial Dates
busdate	Next or previous business day.
datemnth	Date of day in future or past month.
datewrkdy	Date of future or past workday.
days360	Days between dates based on 360-day year.
days365	Days between dates based on 365-day year.
daysact	Actual number of days between dates.
daysdif	Days between dates for any day-count basis.
fbusdate	First business date of month.
holidays	Holidays and non-trading days.
isbusday	True for dates that are business days.
lbusdate	Last business date of month.
wrkdydif	Number of working days between dates.
yearfrac	Fraction of year between dates.

Note: The date functions datenum, datestr, datevec, eomday, now, and weekday now ship with basic MATLAB. They originally shipped only with the Financial Toolbox, and their descriptions remain in this manual for your convenience.

	Coupon Bond Dates
accrfrac	Fraction of coupon period before settlement.
cfamounts	Cash flow and time mapping for bond portfolio.
cfdates	Cash flow dates for a fixed-income security with periodic payments.
cftimes	Time factors corresponding to bond cash flow dates.
cpncount	Coupon payments remaining until maturity.
cpndaten	Next coupon date after settlement date.
cpndatenq	Next quasi coupon date for fixed income security.
cpndatep	Previous coupon date before settlement date.
cpndatepq	Previous quasi coupon date for fixed income security.
cpndaysn	Number of days between settlement date and next coupon date.
cpndaysp	Number of days between previous coupon date and settlement date.
cpnpersz	Number of days in coupon period containing settlement date.

Formatting Currency and Charting Financial Data

	Currency Formats
cur2frac	Decimal currency value to fractional value.
cur2str	Bank formatted text.
frac2cur	Fractional currency value to decimal value.

	Financial Charts
bolling	Bollinger band chart.
candle	Candlestick chart.
dateaxis	Convert serial-date axis labels to calendar-date axis labels.
highlow	High, low, open, close chart.
movavg	Leading and lagging moving averages chart.
pointfig	Point and figure chart.

Analyzing and Computing Cash Flows

	Annuities
annurate	Periodic interest rate of annuity.
annuterm	Number of periods to obtain value.

	Amortization and Depreciation
amortize	Amortization.
depfixdb	Fixed declining-balance depreciation.
depgendb	General declining-balance depreciation.
deprdv	Remaining depreciable value.
depsoyd	Sum of years' digits depreciation.
depstln	Straight-line depreciation.

	Present Value
pvfix	Present value with fixed periodic payments.
pvvar	Present value of varying cash flow.

	Future Value
fvdisc	Future value of discounted security.
fvfix	Future value with fixed periodic payments.
fvvar	Future value of varying cash flow.

	Payment Calculations
payadv	Periodic payment given number of advance payments.
payodd	Payment of loan or annuity with odd first period.
payper	Periodic payment of loan or annuity.
payuni	Uniform payment equal to varying cash flow.

	Rates of Return
effrr	Effective rate of return.
irr	Internal rate of return.
mirr	Modified internal rate of return.
nomrr	Nominal rate of return.
taxedrr	After-tax rate of return.
xirr	Internal rate of return for nonperiodic cash flow.

	Cash Flow Sensitivities
cfconv	Cash flow convexity.
cfdur	Cash flow duration and modified duration.

Fixed-Income Securities

	Accrued Interest
acrubond	Accrued interest of security with periodic interest payments.
acrudisc	Accrued interest of discount security paying at maturity.

	Prices
bndprice	Price a fixed income security from yield to maturity.
prbond	Price of security with regular periodic interest payments.
prdisc	Price of discounted security.
prmat	Price with interest at maturity.
proddf	Price with odd first period.
proddfl	Price with odd first and last periods and settlement in first period.
proddl	Price with odd last period.
prtbill	Price of Treasury bill.

	Term Structure of Interest Rates
disc2zero	Zero curve given a discount curve.
fwd2zero	Zero curve given a forward curve.
pyld2zero	Zero curve given a par yield curve.
tbl2bond	Treasury bond parameters given Treasury bill parameters.
tr2bonds	Term-structure parameters given Treasury bond parameters.
zbtprice	Zero curve from coupon bond prices, using bootstrap method.
zbtyield	Zero curve from coupon bond yields, using bootstrap method.
zero2disc	Discount curve given a zero curve.
zero2fwd	Forward curve given a zero curve.
zero2pyld	Par yield curve given a zero curve.

	Yields
beytbill	Bond equivalent yield for Treasury bill.
bndyield	Yield to maturity for fixed income security.
discrate	Discount rate of a security.
yldbond	Yield to maturity of bond.
ylddisc	Yield of discounted security.
yldmat	Yield of security with interest at maturity.
yldoddf	Yield of security with odd first period.
yldoddfl	Yield of security with odd first and last periods and settlement in first period.
yldoddl	Yield of security with odd last period.
yldtbill	Yield of Treasury bill.

	Interest Rate Sensitivities
bondconv	Convexity.
bonddur	Macaulay and modified durations.

Analyzing Portfolios

	Portfolio Analysis
corr2cov	Convert standard deviation and correlation to covariance.
cov2corr	Convert covariance to standard deviation and correlation coefficient.
ewstats	Expected return and covariance from return time series.
frontcon	Mean-variance efficient frontier.
pcalims	Linear inequalities for individual asset allocation.
pcgcomp	Linear inequalities for asset group comparison constraints.
pcglims	Linear inequalities for asset group minimum and maximum allocation.
pcpval	Linear inequalities for fixing total portfolio value.
portalloc	Optimal capital allocation.
portcons	Portfolio constraints.
portopt	Portfolios on constrained efficient frontier.
portrand	Randomized portfolio risks, returns, and weights.
portstats	Portfolio expected return and risk.
portsim	Random simulation of correlated asset returns.
portvrisk	Portfolio value at risk
ret2tick	Price tick series from incremental returns and initial price.
tick2ret	Incremental return series from a tick price series.

Pricing and Analyzing Derivatives

	Option Valuation and Sensitivity
bdtbond	Black-Derman-Toy pricing of option-embedded bonds.
bdttrans	Translate a tree returned by bdtbond.
binprice	Binomial put and call pricing.
blkprice	Black's option pricing.
blsdelta	Black-Scholes sensitivity to underlying price change.
blsgamma	Black-Scholes sensitivity to underlying delta change.
blsimpv	Black-Scholes implied volatility.
blslambda	Black-Scholes elasticity.
blsprice	Black-Scholes put and call pricing.
blsrho	Black-Scholes sensitivity to interest rate change.
blstheta	Black-Scholes sensitivity to time-until-maturity change.
blsvega	Black-Scholes sensitivity to underlying price volatility.
opprofit	Option profit.

GARCH Processes

	Univariate GARCH Processes
ugarch	GARCH parameter estimation.
ugarchllf	Log-likelihood objective function.
ugarchpred	Forecast conditional variance.
ugarchsim	Simulate GARCH process.

Purpose

Fraction of coupon period before settlement.

Syntax

```
Fraction = accrfrac(Settle, Maturity, Period, Basis, EndMonthRule,
   IssueDate, FirstCouponDate, LastCouponDate, StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are 1, 2, 3, 4, 6, and 12. Default = 2.
`Basis`	Day-count basis of the bond. A vector of integers. 0 = actual/actual (default), 1 = 30/360, 2 = actual/360, 3 = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.

`Settle` and `Maturity` are required arguments. All others are optional. Vector arguments must have consistent dimensions, or they must be scalars.

Description

`Fraction = accrfrac(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` returns the fraction of the coupon period before settlement. This function is used for computing accrued interest.

Example

Given data for three bonds:

```
Settle = '14-Mar-1997';
Maturity = ['30-Nov-2000'
            '31-Dec-2000'
            '31-Jan-2001'];
Period = 2;
Basis = 0;
EndMonthRule = 1;
```

Execute the function.

```
Fraction = accrfrac(Settle, Maturity, Period, Basis,...
EndMonthRule)
```

```
Fraction =
    0.5714
    0.4033
    0.2320
```

See Also

cfdates, cpncount, cpndaten, cpndatep, cpndaysn, cpndaysp, cpnpersz

acrubond

Purpose

Accrued interest of security with periodic interest payments.

Syntax

```
ai = acrubond(id, sd, fd, rv, cpn, per, basis)
ai = acrubond(id, sd, fd, rv, cpn, per)
ai = acrubond(id, sd, fd, rv, cpn)
```

Arguments

`id`	Issue date. Enter as serial date number or date string.
`sd`	Settlement date. Enter as serial date number or date string.
`fd`	First coupon date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`cpn`	Coupon rate. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = 2.
`basis`	Day-count basis: 0 = actual/actual (default), 1 = 30/360, 2 = actual/360, 3 = actual/365.

Description

`ai = acrubond(id, sd, fd, rv, cpn, per, basis)` returns the accrued interest for a security with periodic interest payments. This function computes the accrued interest for securities with standard, short, and long first coupon periods.

Example

```
ai = acrubond('31-jan-1983', '1-mar-1993',...
              '31-jul-1983', 100, 0.1, 2, 0)

ai =
    0.8011
```

See Also

`acrudisc`, `datenum`, `prbond`, `yldbond`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas A, B, C.

Purpose

Accrued interest of discount security paying at maturity.

Syntax

```
ai = acrudisc(sd, md, rv, disc, per, basis)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`disc`	Discount rate of the security. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`ai = acrudisc(sd, md, rv, disc, per, basis)` returns the accrued interest of a discount security paid at maturity.

Example

```
ai = acrudisc('05/01/1992', '07/15/1992', ...
    100, 0.1, 2, 0)

ai =
    2.0604 (or $2.06)
```

See Also

`acrubond`, `prdisc`, `prmat`, `ylddisc`, `yldmat`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formula D.

amortize

Purpose

Amortization.

Syntax

```
[prinp, intp, bal, pmt] = amortize(rate, nper, pv, fv, due)
[prinp, intp, bal, pmt] = amortize(rate, nper, pv, fv)
[prinp, intp, bal, pmt] = amortize(rate, nper, pv)
```

Arguments

`rate` Interest rate per period, as a decimal fraction.

`nper` Number of payment periods.

`pv` Present value of the loan.

`fv` Future value of the loan. Default = `0`.

`due` When payments are due: `0` = end of period (default), or `1` = beginning of period.

Description

`[prinp, intp, bal, pmt] = amortize(rate, nper, pv, fv, due)` returns the principal and interest payments of a loan, the remaining balance of the original loan amount, and the periodic payment.

`prinp` Principal paid in each period. A 1-by-`nper` vector.

`intp` Interest paid in each period. A 1-by-`nper` vector.

`bal` Remaining balance of the loan in each payment period. A 1-by-`nper` vector.

`pmt` Payment per period. A scalar.

Example

A $500 loan paid in six installments at an annual interest rate of 9%:

```
[prinp, intp, bal, pmt] = amortize(0.09/6, 6, 500)

prinp =
   81.47   82.69   83.93   85.19   86.47   87.76
intp =
    6.30    5.07    3.83    2.57    1.30    0.00
bal =
  419.74  338.27  255.58  171.65   86.47    0.00
p =
   87.76
```

See Also

`annurate, annuterm, payadv, payodd, payper`

Purpose

Periodic interest rate of annuity.

Syntax

```
r = annurate(nper, pmt, pv, fv, due)
r = annurate(nper, pmt, pv, fv)
r = annurate(nper, pmt, pv)
```

Arguments

`nper` Number of payment periods.

`pmt` Payment per period.

`pv` Present value of the loan or annuity.

`fv` Future value of the loan or annuity. Default = `0`.

`due` When payments are due: `0` = end of period (default), or `1` = beginning of period.

Description

`r = annurate(nper, pmt, pv, fv, due)` returns the periodic interest rate paid of a loan or annuity.

Example

Find the periodic interest rate of a four-year, $5000 loan with a $130 monthly payment made at the end of each month:

```
r = annurate(4*12, 130, 5000, 0, 0)

r =
    0.0094
```

(`r` multiplied by 12 gives an annual interest rate of 11.32% on the loan.)

See Also

`amortize`, `annuterm`, `irr`, `yldbond`

annuterm

Purpose

Number of periods to obtain value.

Syntax

```
nper = annuterm(rate, pmt, pv, fv, due)
nper = annuterm(rate, pmt, pv, fv)
nper = annuterm(rate, pmt, pv)
```

Arguments

`rate` Interest rate per period, as a decimal fraction.

`pmt` Payment per period.

`pv` Present value.

`fv` Future value. Default = `0`.

`due` When payments are due: `0` = end of period (default), or `1` = beginning of period.

Description

`nper = annuterm(rate, pmt, pv, fv, due)` calculates the number of periods needed to obtain a future value. To calculate the number of periods needed to pay off a loan, enter the payment or the present value as a negative value.

Example

A savings account has a starting balance of $1500. $200 is added at the end of each month and the account pays 9% interest, compounded monthly. How many years will it take to save $5,000?

```
nper = annuterm(0.09/12, 200, 1500, 5000, 0)

nper =
   15.68  months or 1.31 years.
```

See Also

`annurate`, `amortize`, `fvfix`, `pvfix`

Purpose

Black-Derman-Toy pricing of option-embedded bonds.

Syntax

```
[Price, Sensitivities, DiscTree, PriceTree] = bdtbond(OptBond,
  ZeroCurve, VolatilityCurve, Accuracy, CreditCurve,
  ComputeSensitivity)
```

Arguments

All arguments except `Accuracy` are structures. The name of the argument variable can be substituted, but the fieldnames must be reproduced exactly. The variables and the fields are listed below in `Variable.fieldname` format. An optional field or variable may be set to the empty matrix `[]` to invoke defaults. An optional field in a structure may also be left unspecified.

`OptBond`

(required) specification of underlying bond with a possible call and put option. Fields are scalars or date strings.
Bond specification fields for the underlying bond. Type `help ftb` for detailed information on bond parameters.
`OptBond.Settle`: (required) Settlement date.
`OptBond.Maturity`: (required) Maturity date.
`OptBond.Period`: (optional) Coupon frequency. Default = `2`.
`OptBond.Basis`: (optional) Market Basis. Default = `0` (actual/actual).
`OptBond.EndMonthRule`: (optional) EOM rule. Default = `1` (in effect).
`OptBond.FirstCouponDate`: (optional) First coupon payment if the first coupon period is irregular.
`OptBond.LastCouponDate`: (optional) Last coupon payment before maturity if the last coupon period is irregular.
`OptBond.IssueDate`: (optional) Issue date of the bond if the first coupon period is irregular.
`OptBond.StartDate`: (optional) Forward start date of the security if not before settlement.
`OptBond.CouponRate`: (optional) Coupon payment rate.
`OptBond.Face`: (optional) Maturity payment of the bond. Default = `100`.

`OptBond` (continued)	Option specification fields: Specify fields for either a call or a put. Calls assumed held by the bond issuer; puts assumed held by the bondholder. A call therefore reduces the value of the bond to the bond holder, while a put increases the value of the bond. `OptBond.CallStrike`: (required) call option strike price. `OptBond.CallType`: (required) `1` (American) or `0` (European). Default = `1`. `OptBond.CallExpiryDate`: (required) last date of exercise for an American option, or only date for a European option. `OptBond.CallStartDate`: (optional) first date of exercise for an American option. Default is Settlement. `OptBond.PutStrike`: (required) put option strike price. `OptBond.PutType`: (required) `1` (American) or `0` (European). Default = `1`. `OptBond.PutExpiryDate`: (required) last date of exercise for an American option, or only date for a European option. Default is Maturity. `OptBond.PutStartDate`: (optional) first date of exercise for an American option. Default is Settlement.
`ZeroCurve`	(required) Zero curve of `NCURVE` (date, decimal rate) pairs is interpolated to cover the time span of the bond. Times before the first curve date use the first rate; times after the last curve date use the last rate. `ZeroCurve.CurveDates`: (required) `NCURVE`-by-`1` vector of serial dates. `ZeroCurve.ZeroRates`: (required) `NCURVE`-by-`1` vector of rates.

`VolatilityCurve`	(required) Curve of instantaneous yearly volatilities of the short rates.The curve is interpolated to cover the time span of the bond. `VolatilityCurve.CurveDates`: (required) `NCURVE`2-by-1 vector of serial dates. `VolatilityCurve.VolatilityRates`: (required) `NCURVE`2-by-1 vector of yearly volatilities in decimal form.
`Accuracy`	(required) Scalar that specifies the number of steps in the tree per coupon period. Larger numbers yield more accurate answers, but require more time and memory.
`CreditCurve`	(optional) Curve of rate spreads arising from default risk. The curve has `NCURVE`3 (date, basis point) pairs.The curve is interpolated to cover the time span of the bond. `CreditCurve.CurveDates`: (required) `NCURVE`3-by-1 vector of serial dates. `CreditCurve.CreditRates`: (required) `NCURVE`3-by-1 vector of credit spread values in basis points (not decimal rates). The effective change to the zero rate is `CreditRates/10000`.
`ComputeSensitivity`	(optional) Specify if bond sensitivity measures (with and without options) are to be computed. 1 indicates measure computed; 0 indicates not computed. Sensitivities found by a finite difference calculation. The default is no sensitivities, only prices returned. `ComputeSensitivity.Duration`: (required) scalar 1 or 0. `ComputeSensitivity.Convexity`: (required) scalar 1 or 0. `ComputeSensitivity.Vega`: (required) scalar 1 or 0.

Description

`[Price, Sensitivities, DiscTree, PriceTree] = bdtbond(OptBond, ZeroCurve, VolatilityCurve, Accuracy, CreditCurve, ComputeSensitivity)` computes price and sensitivity measures of a bond with embedded call or put options. Valuation is based on the Black-Derman-Toy model for pricing interest rate options given an input yield curve (and possibly a credit spread) and volatility curve.

`Price` is the value of the bond with and without the options.

`Price.OptionFreePrice`: Scalar price of the bond without any options.

`Price.OptionEmbedPrice`: Scalar price (value to the holder of the bond) of the bond with options.

`Price.OptionValue`: scalar value of the options to the holder of the bond .

`Sensitivities` refer to the effect that changes in the yield curve and volatility term structure have on option-free and option-embedded bond prices.

`Sensitivities.Duration`: Sensitivity of option-free bond price to parallel shifts of the yield curve.

`Sensitivities.EffDuration`: Sensitivity of the option-embedded price to shifts in the yield curve.

`Sensitivities.Convexity`: Sensitivity of `Duration` to shifts in the yield curve.

`Sensitivities.EffConvexity`: Sensitivity of `EffDuration` to shifts in the yield curve.

`Sensitivities.Vega`: Sensitivity of the option-embedded price to parallel shifts of the volatility curve.

`DiscTree` is the recombining binomial tree of the interest rate structure. The tree covers `NPERIODS` times from `Settlement` to `Maturity`, where there are `Accuracy` steps in each coupon period. The short rate at settlement and between settlement and the first time is deterministic.

`DiscTree.Values`: `NSTATES`-by-`NPERIODS` matrix of short discount factors. The `NPERIODS` columns of `Values` correspond to successive times. The `NSTATES` rows correspond to states in the rate process. Unused states are masked by `NaN`.

Multiplication of a cash amount at time `Dates(i)` by the discount `Values(j,i)` gives the price at `Dates(i-1)` after traversing the `(j,i)` edge of the tree. The short rate `R(j,i)` prevailing at node `(j,i)` satisfies:

```
( 1 + R(j,i)/Frequency)^(-(Times(j)-Times(j-1))) = Values(j,i)
```

`DiscTree.Times`: 1-by-`NPERIODS` vector of tree node times in units of coupon intervals. (Type `help ftbTFactors` for more information.)

`DiscTree.Dates`: 1-by-`NPERIODS` vector of tree node times as serial date numbers.

`DiscTree.Type`:`'Short Discount'`

`DiscTree.Frequency`: Compounding frequency of the input bond.

`DiscTree.ErrorFlag`: (0 or 1). Set to 1 if any short rate becomes negative.

`PriceTree` is the recombining binomial tree of cash amounts at tree nodes. `PriceTree` is computed from the bond cash flows and the option payoffs. The clean price of the bond is the `PriceTree` value minus the coupon payment and the accrued interest.

`PriceTree.Values`: `NSTATES`-by-`NPERIODS` matrix of price states.

`PriceTree.Times`: 1-by-`NPERIODS` vector of tree node times in units of coupon intervals. (Type `help ftbTFactors` for more information.)

`PriceTree.Dates`: 1-by-`NPERIODS` vector of tree node times as serial date numbers.

`PriceTree.AccrInt`: 1-by-`NPERIODS` vector of accrued interest payable at each time.

`PriceTree.Coupons`: 1-by-`NPERIODS` vector of coupon payments at each time.

`DiscTree.Type`:`'Price'`

Example

Given a bond with the characteristics

```
OptBond.Settle =   '15-Jul-1996';
OptBond.Maturity = '15-Jan-1998';
OptBond.CouponRate = 0.06;
OptBond.Period = 2
```

Specify an American put option on the bond. Make the bond putable by the holder between 15-Jan-1997 and maturity for a strike of 98.

```
OptBond.PutType = 1;
OptBond.PutStartDate =  '15-Jan-1997';
OptBond.PutExpiryDate = '15-Jan-1998';
OptBond.PutStrike = 98
```

Build zero curve term structure.

```
ZeroCurve.ZeroRates =  [0.05; 0.06; 0.065];
ZeroCurve.CurveDates = ['01-Jan-1996'; '01-Jan-1997';
'01-Jan-1998']
```

Build volatility curve term structure.

```
VolCurve.VolatilityRates = [0.15; 0.13];
VolCurve.CurveDates = [729025; 729756]
```

The coupon interval is 1/2 year; use 10 tree periods per year.

```
Accuracy = 5
```

Specify a constant credit spread of 200 basis points (0.02).

```
CreditCurve.CreditRates =  [200];
CreditCurve.CurveDates = ['01-Jan-1996']
```

Ask for duration and vega.

```
SensChoice.Duration = 1;
SensChoice.Convexity = 0;
SensChoice.Vega = 1;

[Price, Sensitivities, DiscTree, PriceTree] = ...
bdtbond(OptBond, ZeroCurve, VolCurve, Accuracy, CreditCurve,...
SensChoice);

Price =
OptionFreePrice: 96.5565
OptionEmbedPrice: 97.1769
OptionValue: 0.6204

Sensitivities =
Duration: 1.3959
EffDuration: 0.6848
Convexity: NaN
EffConvexity: NaN
Vega: -0.0194
```

To look at the rate and clean price trees, use the `bdttrans` function.

```
bdttrans(DiscTree)
bdttrans(PriceTree)
```

See Also

`bdttrans`

Purpose

Translate a tree returned by bdtbond.

Syntax

```
bdttrans(Tree)
[TreeMat, TreeTimes] = bdttrans(Tree)
```

Arguments

Tree	Tree structure returned by bdtbond.

Description

bdttrans unpacks and converts a Short Discount tree structure to a Short Rate tree, or a Price tree structure to a Clean Price tree. The output is a matrix of tree node values and a vector of node times.

TreeMat is an NSTATES-by-NTIMES converted matrix of Short Rate or Clean Price values at points on the tree. Time layers are columns containing the different states. Unused entries of the matrix are screened with NaNs

TreeTimes is a 1-by-NTIMES vector of times corresponding to layers of TreeMat.

If bdttrans is called without output arguments, it plots the translated tree against the time axis in coupon intervals.

Example

Specify a bond.

```
OptBond.Settle = '15-Jul-1996';
OptBond.Maturity = '15-Jan-1998';
OptBond.CouponRate = 0.07;
OptBond.Period = 2;
```

Specify an embedded American call option. Make the bond callable by the issuer between 15-Jan-1997 and maturity.

```
OptBond.CallType = 1;
OptBond.CallStartDate = '15-Jan-1997';
OptBond.CallExpiryDate = '15-Jan-1998';
OptBond.CallStrike = 101;
```

Build constant zero curve structure.

```
ZeroCurve.ZeroRates = 0.05;
ZeroCurve.CurveDates = '15-Jul-1996';
```

Build constant volatility curve structure.

```
VolCurve.VolatilityRates = 0.15;
VolCurve.CurveDates = '15-Jul-1996';
```

Choose two tree nodes per whole coupon period.

```
Accuracy = 2;
```

Run the bdtbond function.

```
[Price, Sensitivities, DiscTree, PriceTree] = bdtbond(OptBond,...
ZeroCurve, VolCurve, Accuracy);
```

Convert the discount tree to a short rate tree.

```
DiscTree =
Values: [6x7 double]
Times: [0 0.5000 1 1.5000 2 2.5000 3]
Dates: [729221 729313 729405 729496 729586 729678 729770]
ErrorFlag: 0
Type: 'Short Discount'
Frequency: 2
```

```
[ShortRateMat, TreeTimes] = bdttrans(DiscTree)
```

```
ShortRateMat =
```

0.0500	0.0500	0.0447	0.0399	0.0357	0.0319	0.0285
NaN	NaN	0.0554	0.0494	0.0442	0.0395	0.0353
NaN	NaN	NaN	0.0613	0.0548	0.0489	0.0437
NaN	NaN	NaN	NaN	0.0679	0.0607	0.0542
NaN	NaN	NaN	NaN	NaN	0.0753	0.0672
NaN	NaN	NaN	NaN	NaN	NaN	0.0835

```
TreeTimes =
0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
```

Convert the Price tree to a Clean Price tree.

```
PriceTree =
Values: [6x7 double]
Times: [0 0.5000 1 1.5000 2 2.5000 3]
Dates: [729221 729313 729405 729496 729586 729678 729770]
ErrorFlag: 0
AccrInt: [0 1.7500 0 1.7597 0 1.7500 0]
Coupons: [0 0 3.5000 0 3.5000 0 3.5000]
Type: 'Price'

[CleanPriceMat, TreeTimes] = bdttrans(PriceTree)

CleanPriceMat =
```

101.9512	101.4677	101.0000	101.0000	101.0000	100.9348	100.0000
NaN	NaN	101.0000	101.0000	101.0000	100.7429	100.0000
NaN	NaN	NaN	100.6019	100.7397	100.5060	100.0000
NaN	NaN	NaN	NaN	100.0978	100.2139	100.0000
NaN	NaN	NaN	NaN	NaN	99.8539	100.0000
NaN	NaN	NaN	NaN	NaN	NaN	100.0000

```
TreeTimes =

0     0.5000     1.0000     1.5000     2.0000     2.5000     3.0000
```

See Also

bdtbond

beytbill

Purpose

Bond equivalent yield for Treasury bill.

Syntax

```
y = beytbill(sd, md, disc)
```

Arguments

`sd` Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.

`md` Maturity date. Enter as serial date number or date string.

`disc` Discount rate of the Treasury bill. Enter as decimal fraction.

Description

`y = beytbill(sd, md, disc)` returns the bond equivalent yield for a Treasury bill.

Example

The settlement date of a Treasury bill is February 10, 1992, the maturity date is August 6, 1992, and the discount rate is 3.77%. The bond equivalent yield:

```
y = beytbill('2/10/1992', '8/6/1992', 0.0377)

y =
    0.0389
```

See Also

`datenum`, `prtbill`, `yldtbill`

Purpose

Binomial put and call pricing.

Syntax

```
[pr, opt] = binprice(so, x, r, t, dt, sig, flag, q, div, exdiv)
```

Arguments

`so`	Underlying asset price. A scalar.
`x`	Option exercise price. A scalar.
`r`	Risk-free interest rate. A scalar. Enter as a decimal fraction.
`t`	The option's time until maturity in years. A scalar.
`dt`	The time increment within `t`. A scalar. `dt` is adjusted so that the length of each interval is consistent with the maturity time of the option. (`dt` is adjusted so that `t` divided by `dt` equals an integer number of increments.)
`sig`	The asset's volatility. A scalar.
`flag`	Specifies whether the option is a call (`flag = 1`) or a put (`flag = 0`). A scalar.
`q`	The dividend rate, as a decimal fraction. A scalar. Default = `0`. If you enter a value for `q`, set `div` and `exdiv` = `0` or do not enter them. If you enter values for `div` and `exdiv`, set `q` = `0`.
`div`	The dividend payment at an ex-dividend date, `exdiv`. A 1-by-N vector. For each dividend payment, there must be a corresponding ex-dividend date. Default = `0`. If you enter values for `div` and `exdiv`, set `q` = `0`.
`exdiv`	Ex-dividend date, specified in number of periods. A 1-by-N vector. Default = `0`.

Description

`[pr, opt] = binprice(so, x, r, t, dt, sig, flag, q, div, exdiv)` prices an option using a binomial pricing model.

Example

For a put option, the asset price is $52, option exercise price is $50, risk-free interest rate is 10%, option matures in 5 months, volatility is 40%, and there is one dividend payment of $2.06 in 3-1/2 months:

```
[pr, opt] = binprice(52, 50, 0.1, 5/12, 1/12, 0.4, 0, 0, 2.06, 3.5)
```

returns the asset price and option value at each node of the binary tree:

```
pr =
   52.0000   58.1367   65.0226   72.7494   79.3515   89.0642
         0   46.5642   52.0336   58.1706   62.9882   70.6980
         0         0   41.7231   46.5981   49.9992   56.1192
         0         0         0   37.4120   39.6887   44.5467
         0         0         0         0   31.5044   35.3606
         0         0         0         0         0   28.0688
opt =
    4.4404    2.1627    0.6361         0         0         0
         0    6.8611    3.7715    1.3018         0         0
         0         0   10.1591    6.3785    2.6645         0
         0         0         0   14.2245   10.3113    5.4533
         0         0         0         0   18.4956   14.6394
         0         0         0         0         0   21.9312
```

See Also

`blkprice`, `blsprice`

Reference

Hull, *Options, Futures, and Other Derivative Securities*, 2nd edition, Chapter 14.

Purpose

Black's option pricing.

Syntax

```
[call, put] = blkprice(f, x, r, t, sig)
```

Arguments

`f` Forward price of underlying asset at time zero. Must be greater than 0. You can extend Black's model to interest-rate derivatives (call and put options embedded in bonds) by calculating the forward price from the equation

```
f = (B - I) * exp(r*t)
```

where `B` is the face value of the bond and `I` is the present value of the coupons during the life of the option.

`x` Strike or exercise price of the options. Must be greater than 0.

`r` Risk-free interest rate (plus storage costs less any convenience yield). Must be greater than or equal to 0.

`t` Time until maturity of option in years. Must be greater than 0.

`sig` Volatility of the price of the underlying asset. Must be greater than or equal to 0.

Description

`[call, put] = blkprice(f, x, r, t, sig)` uses Black's model to value an option and returns the `call` and `put` option prices.

Note: This function uses `normcdf`, the normal cumulative distribution function in the Statistics Toolbox.

Example

The forward price of a bond is $95, the exercise price of the option is $98, the risk-free interest rate is 11%, the time to maturity of the option is 3 years, and the volatility of the bond price is 2.5%.

```
[call, put] = blkprice(95, 98, 0.11, 3, 0.025)

call =
    0.4162 (or $0.42)
put =
    2.5729 (or $2.57)
```

References

Hull, *Options, Futures, and Other Derivative Securities*, 2nd edition, Formulas 15.7 and 15.8.

Black, "The Pricing of Commodity Contracts," *Journal of Financial Economics*, March 3, 1976, pp. 167-179.

See Also

`binprice`, `blsprice`

Purpose

Black-Scholes sensitivity to underlying price change.

Syntax

```
[cd, pd] = blsdelta(so, x, r, t, sig, q)
[cd, pd] = blsdelta(so, x, r, t, sig)
```

Arguments

`so` Current stock price.

`x` Exercise price.

`r` Risk-free interest rate. Enter as a decimal fraction.

`t` Time to maturity of the option, in years.

`sig` Standard deviation of the annualized continuously compounded rate of return of the stock, also known as volatility.

`q` Dividend rate or foreign interest rate where applicable. Enter as a decimal fraction. Default = `0`.

Description

`[cd, pd] = blsdelta(so, x, r, t, sig, q)` returns sensitivity in option value to change in the underlying security price. Delta is also known as the hedge ratio. `cd` is the delta of a call option, and `pd` is the delta of a put option.

Note: This function uses `normcdf`, the normal cumulative distribution function in the Statistics Toolbox.

Example

```
[cd, pd] = blsdelta(50, 50, 0.1, 0.25, 0.3, 0)

cd =
    0.5955
pd =
   -0.4045
```

See Also

`blsgamma`, `blslambda`, `blsprice`, `blsrho`, `blstheta`, `blsvega`

Reference

Hull, *Options, Futures, and Other Derivative Securities*, 2nd edition, Chapter 13.

blsgamma

Purpose

Black-Scholes sensitivity to underlying delta change.

Syntax

```
g = blsgamma(so, x, r, t, sig, q)
g = blsgamma(so, x, r, t, sig)
```

Arguments

`so` Current stock price.

`x` Exercise price.

`r` Risk-free interest rate. Enter as a decimal fraction.

`t` Time to maturity of the option in years.

`sig` Standard deviation of the annualized continuously compounded rate of return of the stock (also known as the volatility).

`q` Dividend rate. Enter as a decimal fraction. Default = 0.

Description

`g = blsgamma(so, x, r, t, sig, q)` returns gamma `g`, the sensitivity of delta to change in the underlying security price.

Note: This function uses `normpdf`, the normal probability density function in the Statistics Toolbox.

Example

```
g = blsgamma(50, 50, 0.12, 0.25, 0.3, 0)

g =
    0.0512
```

See Also

`blsdelta`, `blslambda`, `blsprice`, `blsrho`, `blstheta`, `blsvega`

Reference

Hull, *Options, Futures, and Other Derivative Securities*, 2nd edition, Chapter 13.

Purpose

Black-Scholes implied volatility.

Syntax

```
v = blsimpv(so, x, r, t, call, maxiter)
v = blsimpv(so, x, r, t, call)
```

Arguments

`so`	Current asset price.
`x`	Exercise price.
`r`	Risk-free interest rate. Enter as a decimal fraction.
`t`	Time to maturity in years.
`call`	Call option value.
`maxiter`	Maximum number of iterations used in solving for `v` using Newton's method. Default = `50`.

Description

`v = blsimpv(so, x, r, t, call, maxiter)` returns the implied volatility `v` of an underlying asset, using Newton's method.

Note: This function uses `normcdf` and `normpdf`, the normal cumulative distribution and normal probability density functions in the Statistics Toolbox.

Example

An asset has a current price of $100, an exercise price of $95, the risk free interest rate is 7.5%, the time to maturity of the option is 0.25 years, and the call option has a value of $10.00.

```
v = blsimpv(100, 95, 0.075, 0.25, 10)

v =
    0.3130  (or 31.3%)
```

See Also

`blsprice`

Reference

Bodie, Kane, and Marcus, *Investments*, page 681.

blslambda

Purpose

Black-Scholes elasticity.

Syntax

```
[lc, lp] = blslambda(so, x, r, t, sig, q)
[lc, lp] = blslambda(so, x, r, t, sig)
```

Arguments

`so` Current stock price.

`x` Exercise price.

`r` Risk-free interest rate. Enter as a decimal fraction.

`t` Time to maturity of the option in years.

`sig` Standard deviation of the annualized continuously compounded rate of return of the stock (also known as the volatility).

`q` Dividend rate. Enter as a decimal fraction. Default = 0.

Description

`[lc, lp] = blslambda(so, x, r, t, sig, q)` returns the elasticity of an option. `lc` is the call option elasticity or leverage factor, and `lp` is the put option elasticity or leverage factor. Elasticity (the leverage of an option position) measures the percent change in an option price per one percent change in the underlying stock price.

Note: This function uses `normcdf`, the normal cumulative distribution function in the Statistics Toolbox.

Example

```
[lc, lp] = blslambda(50, 50, 0.12, 0.25, 0.3)

lc =
    8.1274
lp =
   -8.6466
```

See Also

`blsdelta`, `blsgamma`, `blsprice`, `blsrho`, `blstheta`, `blsvega`

Reference

Daigler, *Advanced Options Trading*, Chapter 4.

Purpose

Black-Scholes put and call pricing.

Syntax

```
[call, put] = blsprice(so, x, r, t, sig, q)
[call, put] = blsprice(so, x, r, t, sig)
```

Arguments

`so` Current asset price.

`x` Exercise price.

`r` Risk-free interest rate. Enter as a decimal fraction.

`t` Time to maturity of the option in years.

`sig` Standard deviation of the annualized continuously compounded rate of return of the asset (also known as the volatility).

`q` Dividend rate of the asset. Enter as a decimal fraction. Default = 0.

Description

`[call, put] = blsprice(so, x, r, t, sig, q)` returns the value of call and put options using the Black-Scholes pricing formula.

Note: This function uses `normcdf`, the normal cumulative distribution function in the Statistics Toolbox.

Example

The current price of an asset is $100, the exercise price of the option is $95, the risk-free interest rate is 10%, the time to maturity of the option is 0.25 years, and the standard deviation of the asset is 50%.

```
[call, put] = blsprice(100, 95, 0.1, 0.25, 0.5)

call =
         13.70
put =
          6.35
```

See Also

`blkprice`, `blsdelta`, `blsgamma`, `blsimpv`, `blslambda`, `blsrho`, `blstheta`, `blsvega`

Reference

Bodie, Kane, and Marcus, *Investments*, page 681.

blsrho

Purpose

Black-Scholes sensitivity to interest rate change.

Syntax

```
[cr, pr]= blsrho(so, x, r, t, sig, q)
[cr, pr]= blsrho(so, x, r, t, sig, q)
```

Arguments

`so` Current security price.

`x` Exercise or strike price.

`r` Interest rate. Enter as a decimal fraction.

`t` Time to maturity of the option in years.

`sig` Standard deviation of the annualized continuously compounded rate of return of the security (also known as the volatility).

`q` Dividend rate of the security. Enter as a decimal fraction. Default = 0.

Description

`[cr, pr]= blsrho(so, x, r, t, sig, q)` returns the call option rho `cr`, and the put option rho `pr`. Rho is the rate of change in value of securities with respect to interest rates.

Note: This function uses `normcdf`, the normal cumulative distribution function in the Statistics Toolbox.

Example

```
[cr, pr] = blsrho(50, 50, 0.12, 0.25, 0.3, 0)

cr =
    6.6686
pr =
   -5.4619
```

See Also

`blsdelta`, `blsgamma`, `blslambda`, `blsprice`, `blstheta`, `blsvega`

Reference

Hull, *Options, Futures, and Other Derivative Securities*, 2nd edition, Chapter 13.

Purpose

Black-Scholes sensitivity to time-until-maturity change.

Syntax

```
[ct, pt] = blstheta(so, x, r, t, sig, q)
[ct, pt] = blstheta(so, x, r, t, sig)
```

Arguments

`so` Current stock price.

`x` Exercise price.

`r` Risk-free interest rate. Enter as a decimal fraction.

`t` Time to maturity of the option in years.

`sig` Standard deviation of the annualized continuously compounded rate of return of the stock (also known as the volatility).

`q` Dividend rate. Enter as a decimal fraction. Default = 0.

Description

`[ct, pt] = blstheta(so, x, r, t, sig, q)` returns the call option theta `ct`, and the put option theta `pt`. Theta is the sensitivity in option value with respect to time.

Note: This function uses `normpdf`, the normal probability density function and `normcdf`, the normal cumulative distribution function in the Statistics Toolbox.

Example

```
[ct, pt] = blstheta(50, 50, 0.12, 0.25, 0.3, 0)

ct =
   -8.9630
pt =
   -3.1404
```

See Also

`blsdelta`, `blsgamma`, `blslambda`, `blsprice`, `blsrho`, `blsvega`

Reference

Hull, *Options, Futures, and Other Derivative Securities*, 2nd edition, Chapter 13.

blsvega

Purpose

Black-Scholes sensitivity to underlying price volatility.

Syntax

```
vega = blsvega(so, x, r, t, sig, q)
vega = blsvega(so, x, r, t, sig)
```

Arguments

`so` Current stock price.

`x` Exercise price.

`r` Risk-free interest rate. Enter as a decimal fraction.

`t` Time to maturity of the option in years.

`sig` Standard deviation of the annualized continuously compounded rate of return of the stock (also known as the volatility).

`q` Dividend rate. Enter as a decimal fraction. Default = 0.

Description

`vega = blsvega(so, x, r, t, sig, q)` returns `vega`, the rate of change of the option value with respect to the volatility of the underlying asset.

Note: This function uses `normpdf`, the normal probability density function in the Statistics Toolbox.

Example

```
vega = blsvega(50, 50, 0.12, 0.25, 0.3, 0)

vega =
    9.6035
```

See Also

`blsdelta`, `blsgamma`, `blslambda`, `blsprice`, `blsrho`, `blstheta`

Reference

Hull, *Options, Futures, and Other Derivative Securities*, 2nd edition, Chapter 13.

Purpose

Price a fixed income security from yield to maturity.

Syntax

```
[Price, AccruedInt] = bndprice(Yield, CouponRate, Settle, Maturity,
  Period, Basis, EndMonthRule, IssueDate, FirstCouponDate,
  LastCouponDate, StartDate, Face)
```

Arguments

All inputs are scalar or `NumBonds`-by-1 vectors. Dates can be serial date numbers or date strings. Fill unspecified entries in input vectors with `NaN`. Optional arguments can be passed as the empty matrix `[]`.

`Yield`	(required) Bond equivalent yield to maturity with semi-annual compounding.
`CouponRate`	(required) Decimal number indicating the annual percentage rate used to determine the coupons payable on a bond.
`Settle`	(required) Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	(required) Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.

`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.
`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.
`Face`	Face or par value.

Description

`[Price, AccruedInt] = bndprice(Yield, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face)` given `NumBonds` with SIA date parameters and semi-annual yields to maturity, returns the clean prices and accrued interest due.

`Price` is the clean price of the bond (current price without accrued interest).

`AccruedInt` is the accrued interest payable at settlement.

`Price` and `Yield` are related by the formula:

```
Price + Accrued_Interest = sum(Cash_Flow*(1+Yield/2)^(–Time))
```

where the sum is over the bonds' cash flows and corresponding times in units of semi-annual coupon periods.

Example

Price a treasury bond at three different yield values.

```
Yield = [0.04; 0.05; 0.06];
CouponRate = 0.05;
Settle = '20-Jan-1997';
Maturity = '15-Jun-2002';
Period = 2;
Basis = 0;

[Price, AccruedInt] = bndprice(Yield, CouponRate, Settle, ...
      Maturity, Period, Basis)

Price =
  104.8106
  99.9951
  95.4384

AccruedInt =
  0.4945
  0.4945
  0.4945
```

See Also

cfamounts, bndyield

bndyield

Purpose

Yield to maturity for a fixed income security.

Syntax

```
Yield = bndyield(Price, CouponRate, Settle, Maturity, Period, Basis,
   EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
   StartDate, Face)
```

Arguments

All inputs are scalar or `NumBonds`-by-`1` vectors. Dates can be serial date numbers or date strings. Fill unspecified entries in input vectors with `NaN`. Optional arguments can be passed as the empty matrix `[]`.

`Price`	(required) Clean price of the bond (current price without accrued interest).
`CouponRate`	(required) Decimal number indicating the annual percentage rate used to determine the coupons payable on a bond.
`Settle`	(required) Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	(required) Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.

`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.
`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.
`Face`	Face or par value.

Description

`Yield = bndyield(Price, CouponRate, Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate, Face)` given `NumBonds` bonds with SIA date parameters and clean prices, returns the bond equivalent yields to maturity.

`Yield` is a `NumBonds`-by-1 vector of the bond equivalent yields to maturity with semi-annual compounding.

`Price` and `Yield` are related by the formula:

```
Price + Accrued_Interest = sum(Cash_Flow*(1+Yield/2)^(–Time))
```

where the sum is over the bonds' cash flows and corresponding times in units of semi-annual coupon periods.

Example

Compute the yield of a treasury bond at three different price values.

```
Price = [95; 100; 105];
CouponRate = 0.05;
Settle = '20-Jan-1997';
Maturity = '15-Jun-2002';
Period = 2;
Basis = 0;

[Yield] = bndyield(Price, CouponRate, Settle, ...
      Maturity, Period, Basis)

Yield =
    0.0610
    0.0500
    0.0396
```

See Also

bndprice, cfamounts

Purpose

Bollinger band chart.

Syntax

```
bolling(asset, samples, alpha)
[mav, uband, lband] = bolling(asset, samples, alpha)
```

Description

`bolling(asset, samples, alpha)` plots Bollinger bands for given `asset` data. `samples` specifies the number of samples to use in computing the moving average. `alpha` is the exponent used to compute the element weights of the moving average. This form of the function does not return any data.

`[mav, uband, lband] = bolling(asset, samples, alpha)` returns `mav` with the moving average of the `asset` data, `uband` with the upper band data, and `lband` with the lower band data. This form of the function does not plot any data.

Examples

If `asset` is an N-by-1 vector of closing stock prices:

```
bolling(asset, 20, 1)
```

plots linear 20-day moving average Bollinger bands based on the stock prices.

```
[mav, uband, lband] = bolling(asset, 20, 1)
```

returns `mav`, `uband`, and `lband` as (N-19)-by-1 vectors containing the moving average, upper band, and lower band data, without plotting the data.

See Also

`candle`, `dateaxis`, `highlow`, `movavg`, `pointfig`

bondconv

Purpose

Convexity.

Syntax

```
[pc, yc] = bondconv(sd, md, rv, cpn, yld, per, basis)
[pc, yc] = bondconv(sd, md, rv, cpn, yld, per)
[pc, yc] = bondconv(sd, md, rv, cpn, yld)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`cpn`	Coupon rate. Enter as a decimal fraction.
`yld`	Yield. Enter as a decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[pc, yc] = bondconv(sd, md, rv, cpn, yld, per, basis)` returns the convexity for a security in periods `pc` and years `yc`.

Example

Given this data for a security:

Settlement date	01-Dec-1994
Maturity date	01-Jan-2000
Par value	$100.00
Coupon rate	5%
Yield	4.34%
Coupons per year	2 (semi-annual)
Basis	actual/actual

Find the convexity:

```
[pc,yc] = bondconv('12/1/1994','1/1/2000',100, 0.05, 0.0434, 2,0)

pc =
          92.13 (periods)
yc =
          23.03 (years)
```

See Also

bonddur, cfconv, cfdur, datenum

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 8, 9.

bonddur

Purpose

Macaulay and modified durations.

Syntax

```
[d, m] = bonddur(sd, md, rv, cpn, yld, per, basis)
[d, m] = bonddur(sd, md, rv, cpn, yld, per)
[d, m] = bonddur(sd, md, rv, cpn, yld)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`cpn`	Coupon rate. Enter as a decimal fraction.
`yld`	Yield. Enter as a decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[d, m] = bonddur(sd, md, rv, cpn, yld, per, basis)` finds the Macaulay duration `d` and modified duration `m` in years for a security with periodic interest payments.

Example

Given this data for a security:

Settlement date	01-Dec-1994
Maturity date	01-Jan-2000
Par value	$100.00
Coupon rate	5%
Yield	4.34%
Coupons per year	2 (semi-annual)
Basis	actual/actual

Find the durations:

```
[d, m] = bonddur('12/1/1994','1/1/2000', 100, 0.05, 0.0434, 2, 0)

d =
    4.4720 (years)
m =
    4.3770 (years)
```

See Also

bondconv, cfconv, cfdur, datenum

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 5, 6.

busdate

Purpose

Next or previous business day.

Syntax

```
bd = busdate(d, direc, hol)
bd = busdate(d, direc)
bd = busdate(d)
```

Arguments

`d`	Reference date. Enter as serial date number or date string.
`direc`	Direction. `1` = next (default) or `-1` = previous business day.
`hol`	Vector of holidays and non-trading-day dates. All dates in `hol` must be the same format: either serial date numbers or date strings. (Using date numbers improves performance.) The `holidays` function supplies the default vector.

Description

`bd = busdate(d, direc, hol)` returns the serial date number `bd` of the next or previous business day from the reference date.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Example

```
bd = busdate('3-Jul-1997', 1, holidays)
bd =
      729578
datestr(bd)
ans =
07-Jul-1997
```

See Also

`holidays`, `isbusday`

Purpose

Candlestick chart.

Syntax

```
candle(hi, lo, cl, op, color)
candle(hi, lo, cl, op)
```

Arguments

`hi` High prices for a security. An N-by-1 vector.

`lo` Low prices for a security. An N-by-1 vector.

`cl` Closing prices for a security. An N-by-1 vector.

`op` Opening prices for a security. An N-by-1 vector.

`color` Candlestick color. A string. MATLAB supplies a default color if none is specified. The default color differs depending on the background color of the figure window. See "ColorSpec" in the online MATLAB Help Desk for `color` names.

Description

`candle(hi, lo, cl, op, color)` plots a candlestick chart given N-by-1 vectors with the high, low, closing, and opening, prices of a security.

If the closing price is greater than the opening price, the body (the region between the opening and closing price) is filled.

If the opening price is greater than the closing price, the body is empty.

Example

Given `hi`, `lo`, `cl`, and `op` as equal-size vectors of stock price data

```
candle(hi, lo, cl, op, 'cyan')
```

plots a candlestick chart with cyan candles.

See Also

`bolling`, `dateaxis`, `highlow`, `movavg`, `pointfig`

cfamounts

Purpose

Cash flow and time mapping for bond portfolio.

Syntax

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

Arguments

`CouponRate`	Decimal number indicating the annual percentage rate used to determine the coupons payable on a bond.
`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.
`Face`	Face or par value.

`Settle` and `Maturity` are required arguments. All others are optional. Required arguments must be N-by-1 or 1-by-N conforming vectors or scalars. Optional arguments must be either N-by-1 or 1-by-N conforming vectors, scalars, or empty matrices.

Description

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

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

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

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

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

where

`PV` = present value of a cash flow

`CF` = the cash flow amount

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

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

`CFlowFlags` is the matrix of cash flow flags for a portfolio of bonds. Each row corresponds to the vector of cash flow flags for each bond. Each element in a

column corresponds to the specific flag associated with each cash flow of a bond. Flags identify the type of each cash flow (e.g., nominal coupon cash flow, front or end partial or “stub” coupon, maturity cash flow). Possible values are shown in the table.

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

Examples

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

```
Settle = '01-Nov-1993'
Maturity = ['15-Dec-1994';'15-Jun-1995']
CouponRate= [0.06; 0.05]
Period = [4;2]
Basis = [1;0]
[CFlowAmounts, CFlowDates, TFactors, CFlowFlags] = ...
cfamounts(CouponRate,Settle, Maturity, Period, Basis)

CFlowAmounts =

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

CFlowDates =

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

TFactors =

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

CFlowFlags =

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

See Also

cfdates

Purpose

Cash flow convexity.

Syntax

```
c = cfconv(cf, yld)
```

Arguments

`cf` Cash flow. A vector of real numbers.

`yld` Periodic yield. A scalar. Enter as a decimal fraction.

Description

`c = cfconv(cf, yld)` returns the convexity `c` of a cash flow in periods.

Example

Given a cash flow of nine payments of $2.50 and a final payment $102.50, with a periodic yield of 2.5%:

```
cf=[2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 102.5];

c=cfconv(cf, 0.025)

c =
        90.4493 (periods)
```

See Also

`bondconv`, `bonddur`, `cfdur`

cfdates

Purpose

Cash flow dates for a fixed-income security.

Syntax

```
[CFlowDates, CFlowFlags] = cfdates(Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

LastCouponDate	Last coupon date of a bond prior to the maturity date. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate regardless of where it falls and will be followed only by the bond's maturity cash flow date.
StartDate	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If StartDate is not explicitly specified, the effective start date is the settlement date.

Settle and Maturity are required arguments. All others are optional. Required arguments must be N-by-1 or 1-by-N conforming vectors or scalars. Optional arguments must be either N-by-1 or 1-by-N conforming vectors, scalars, or empty matrices.

Any input can contain multiple values, but if so, all other inputs must contain the same number of values or a single value that applies to all. For example, if Maturity contains N dates, then Settle must contain N dates or a single date.

Description

`[CFlowDates, CFlowFlags] = cfdates(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` returns a matrix of cash flow dates for a bond or set of bonds. cfdates determines all cash flow dates for a bond whether or not the coupon payment structure is normal or the first and/or last coupon period is long or short.

CFlowDates is an N-row matrix of serial date numbers, padded with NaNs as necessary to ensure that all rows have the same number of elements. Use the function datestr to convert serial date numbers to formatted date strings.

CFlowFlags is the matrix of cash flow flags for a portfolio of bonds. Each row corresponds to the vector of cash flow flags for each bond. Each element in a column corresponds to the specific flag associated with each cash flow of a bond. Flags identify the type of each cash flow (e.g., nominal coupon cash flow, front

or end partial or “stub” coupon, maturity cash flow). Possible values are shown in the table.

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

Examples

```
CFlowDates = cfdates('14 Mar 1997', '30 Nov 1998', 2, 0, 1)
CFlowDates =
      729541      729724      729906      730089
datestr(CFlowDates)
ans =
31-May-1997
30-Nov-1997
31-May-1998
30-Nov-1998
```

Given three securities with different maturity dates and the same default arguments:

```
Maturity = ['30-Sep-1997'; '31-Oct-1998'; '30-Nov-1998'];
CFlowDates = cfdates('14-Mar-1997', Maturity)
CFlowDates =
      729480      729663         NaN         NaN
      729510      729694      729875      730059
      729541      729724      729906      730089
```

Look at the cash-flow dates for the last security:

```
datestr(CFlowDates(3,:))
ans =
31-May-1997
30-Nov-1997
31-May-1998
30-Nov-1998
```

See Also

accrfrac,cpncount,cpndaten,cpndatep,cpndaysn,cpndaysp,cpnpersz

cfdur

Purpose

Cash-flow duration and modified duration.

Syntax

```
[d, md] = cfdur(cf, yld)
```

Arguments

`cf` Cash flow. A vector of real numbers.

`yld` Periodic yield. A scalar. Enter as a decimal fraction.

Description

`[d, md] = cfdur(cf, yld)` calculates the duration `d` and modified duration (volatility) `md` of a cash flow in periods.

Example

Given a cash flow of nine payments of $2.50 and a final payment $102.50, with a periodic yield of 2.5%:

```
cf=[2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 2.5 102.5];

[d, md]=cfdur(cf, 0.025)

d =
       8.9709 (periods)
md =
       8.7521 (periods)
```

See Also

bondconv, bonddur, cfconv

Purpose

Time factors corresponding to bond cash flow dates.

Syntax

```
TFactors = cftimes(Settle, Maturity, Period, Basis,
  EndMonthRule,        IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are 1, 2, 3, 4, 6, and 12. Default = 2.
`Basis`	Day-count basis of the bond. (Time factors are computed on an actual/actual basis. `Basis` is included here as an input argument to maintain interface consistency with other coupon functions.)
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. 0 = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. 1 = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

LastCouponDate	Last coupon date of a bond prior to the maturity date. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate regardless of where it falls and will be followed only by the bond's maturity cash flow date.
StartDate	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If StartDate is not explicitly specified, the effective start date is the settlement date.

Settle and Maturity are required arguments. All others are optional. Vector arguments must have consistent dimensions, or they must be scalars.

Description

TFactors = cftimes(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate) determines the time factors corresponding to the cash flows of a bond or set of bonds. The time factor of a cash flow is the difference between the settlement date and the cash flow date in units of semi-annual coupon periods.

Example

```
Settle = '15-Mar-1997'
Maturity = '01-Sep-1999'
Period = 2
TFactors = cftimes(Settle, Maturity, Period)

TFactors =

    0.9239    1.9239    2.9239    3.9239    4.9239
```

See Also

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

Purpose

Convert standard deviation and correlation to covariance.

Syntax

```
ExpCovariance = corr2cov(ExpSigma, ExpCorrC)
```

Arguments

`ExpSigma`	Vector of length `N` with the standard deviations of each process.
`ExpCorrC`	`N`-by-`N` correlation coefficient matrix. If `ExpCorrC` is not specified, the processes are assumed to be uncorrelated, and the identity matrix is used.

Description

`corr2cov` converts standard deviation and correlation to covariance.

`ExpCovariance` is an `N`-by-`N` covariance matrix, where `N` is the number of processes.

```
ExpCov(i,j) = ExpCorrC(i,j)*(ExpSigma(i)*ExpSigma(j)
```

Example

```
ExpSigma = [0.5  2.0];

ExpCorrC = [1.0 –0.5
           –0.5  1.0];

[ExpCovariance] = corr2cov(ExpSigma, ExpCorrC)
```

Expected results:

```
ExpCovariance =
                    0.25  –0.5
                   –0.5    4.0
```

See Also

`corrcoef`, `cov`, `cov2corr`, `ewstats`, `std`

cov2corr

Purpose

Convert covariance to standard deviation and correlation coefficient.

Syntax

```
[ExpSigma, ExpCorrC] = cov2corr(ExpCovariance)
```

Arguments

`ExpCovariance`	`N`- by-`N` covariance matrix, e.g., from `cov` or `ewstats`. `N` is the number of random processes.

Description

`cov2corr` converts covariance to standard deviations and correlation coefficients.

`ExpSigma` is a 1-by-`N` vector with the standard deviation of each process.

`ExpCorrC` is an `N`-by-`N` matrix of correlation coefficients.

```
ExpSigma(i) = sqrt(ExpCovariance(i,i))
ExpCorrC(i,j) = ExpCovariance(i,j)/(ExpSigma(i)*ExpSigma(j))
```

Example

```
ExpCovariance = [0.25 -0.5
                 -0.5   4.0];

[ExpSigma, ExpCorrC] = cov2corr(ExpCovariance)
```

Expected results:

```
ExpSigma =
    0.5  2.0

ExpCorrC =
    1.0  -0.5
   -0.5   1.0
```

See Also

`corr2cov`, `corrcoef`, `cov`, `ewstats`, `std`

Purpose

Coupon payments remaining until maturity.

Syntax

```
NumCouponsRemaining = cpncount(Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

LastCouponDate	Last coupon date of a bond prior to the maturity date. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate regardless of where it falls and will be followed only by the bond's maturity cash flow date.
StartDate	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If StartDate is not explicitly specified, the effective start date is the settlement date.

Settle and Maturity are required arguments. All others are optional. Required arguments must be N-by-1 or 1-by-N conforming vectors or scalars. Optional arguments must be either N-by-1 or 1-by-N conforming vectors, scalars, or empty matrices.

Description

NumCouponsRemaining = cpncount(Settle, Maturity, Period, Basis, EndMonthRule) returns the whole number of coupon payments between the settlement and maturity dates for a coupon bond or set of bonds.

Examples

```
NumCouponsRemaining = cpncount('14 Mar 1997', '30 Nov 2000',...
2, 0, 0)
n =
     8
```

Given three coupon bonds with different maturity dates and the same default arguments:

```
Maturity = ['30 Sep 2000'; '31 Oct 2001'; '30 Nov 2002'];
NumCouponsRemaining = cpncount('14 Sep 1997', Maturity)
NumCouponsRemaining =
     7
     9
    11
```

See Also

accrfrac, cfamounts, cfdates, cftimes, cpndaten, cpndatenq, cpndatep, cpndatepq, cpndaysn, cpndaysp, cpnpersz

cpndaten

Purpose

Next coupon date for fixed-income security.

Syntax

```
NextCouponDate = cpndaten(Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.

`Settle` and `Maturity` are required arguments. All others are optional. Required arguments must be N-by-1 or 1-by-N conforming vectors or scalars. Optional arguments must be either N-by-1 or 1-by-N conforming vectors, scalars, or empty matrices.

Description

`NextCouponDate = cpndaten(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` returns the serial date number for the next coupon date after the settlement date. This function finds the next coupon date whether or not the coupon structure is synchronized with the maturity date.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

```
NextCouponDate = cpndaten('14 Mar 1997', '30 Nov 2000', 2, 0, 0);
datestr(NextCouponDate)
ans =
30-May-1997

NextCouponDate = cpndaten('14 Mar 1997', '30 Nov 2000', 2, 0, 1);
datestr(NextCouponDate)
ans =
31-May-1997

Maturity = ['30 Sep 2000'; '31 Oct 2000'; '30 Nov 2000'];
NextCouponDate = cpndaten('14 Mar 1997', Maturity);
datestr(NextCouponDate)
ans =
31-Mar-1997
30-Apr-1997
31-May-1997
```

See Also

accrfrac, cfamounts, cfdates, cftimes, cpncount, cpndatenq, cpndatep, cpndatepq, cpndaysn, cpndaysp, cpnpersz

Purpose

Next quasi coupon date for fixed income security.

Syntax

```
NextQuasiCouponDate = cpndatenq(Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

LastCouponDate	Last coupon date of a bond prior to the maturity date. In the absence of a specified FirstCouponDate, a specified LastCouponDate determines the coupon structure of the bond. The coupon structure of a bond is truncated at the LastCouponDate regardless of where it falls and will be followed only by the bond's maturity cash flow date.
StartDate	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If StartDate is not explicitly specified, the effective start date is the settlement date.

Settle and Maturity are required arguments. All others are optional. Required arguments must be NUMBONDS-by-1 or 1-by-NUMBONDS conforming vectors or scalars. Optional arguments must be either NUMBONDS-by-1 or 1-by-NUMBONDS conforming vectors, scalars, or empty matrices. Fill unspecified entries in input vectors with the value NaN. Dates can be serial date numbers or date strings.

Description

`NextQuasiCouponDate = cpndatenq(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` determines the next quasi coupon date for a set of NUMBONDS fixed income securities. The term "next quasi coupon date" refers to the next coupon date for a bond computed as if the first coupon date is unspecified. This function finds the next quasi coupon date for a bond with a coupon structure in which the first or last period is either normal, short, or long. For zero coupon bonds this function returns quasi coupon dates as if the bond had a semi-annual coupon structure.

NextQuasiCouponDate is the next coupon date for a bond computed as if no first coupon date has been explicitly specified.

Outputs are NUMBONDS-by-1 vectors

If Settle is a coupon date, this function never returns the settlement date. It returns the coupon date strictly after settlement.

Examples

The NextQuasiCouponDate indicates the dates on which coupons would normally be paid if a different FirstCouponDate is not specified.

For example, given a pair of bonds with the characteristics:

```
Settle = str2mat('30-May-1992','10-Dec-1992')
Maturity = str2mat('30-Nov-1997','10-Jun-1999')
```

Compute NextCouponDate for this pair of bonds.

```
NextCouponDate = cpndaten(Settle, Maturity)
```

```
ans =

31-May-1992
10-Jun-1993
```

Compute the next quasi coupon dates for these two bonds.

```
NextQuasiCouponDate = cpndatenq(Settle, Maturity)
```

```
ans =

31-May-1992
10-Jun-1993
```

Because no FirstCouponDate has been specified, the results are identical.

Now supply an explicit FirstCouponDate for each bond.

```
FirstCouponDate = str2mat('30-Nov-1992','10-Dec-1993')
```

Compute the next coupon dates.

```
NextCouponDate = cpndaten(Settle, Maturity, 2, 0, 1, [],...
FirstCouponDate)
```

```
ans =

30-Nov-1992
10-Dec-1993
```

The next coupon dates are identical to the specified first coupon dates.

Now recompute the next quasi coupon dates.

```
NextQuasiCouponDate = cpndatenq(Settle, Maturity, 2, 0, 1, [],...
FirstCouponDate)

ans =

31-May-1992
10-Jun-1993
```

The results indicate the dates on which coupons would normally be paid if a different first coupon date had not been specified.

See Also

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

Purpose

Previous coupon date for fixed-income security.

Syntax

```
PreviousCouponDate = cpndatep(Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.

`Settle` and `Maturity` are required arguments. All others are optional. Required arguments must be N-by-1 or 1-by-N conforming vectors or scalars. Optional arguments must be either N-by-1 or 1-by-N conforming vectors, scalars, or empty matrices.

Description

`PreviousCouponDate = cpndatep(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` returns the serial date number for the previous coupon date for a bond or set of bonds. This function finds the previous coupon date whether or not the coupon structure is synchronized with the maturity date. When the coupon frequency is 0 (a zero coupon bond), the previous coupon date is calculated as if the frequency were semi-annual.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

```
PreviousCouponDate = cpndatep('14 Mar 1997', '30 Jun 2000',...
2, 0, 0);
datestr(PreviousCouponDate)
ans =
30-Dec-1996

PreviousCouponDate = cpndatep('14 Mar 1997', '30 Jun 2000',...
2, 0, 1);
datestr(PreviousCouponDate)
ans =
31-Dec-1996

Maturity = ['30 Apr 2000'; '31 May 2000'; '30 Jun 2000'];
PreviousCouponDate = cpndatep('14 Mar 1997', Maturity);
datestr(PreviousCouponDate)
ans =
31-Oct-1996
30-Nov-1996
31-Dec-1996
```

See Also

accrfrac, cfamounts, cfdates, cftimes, cpncount, cpndaten, cpndatenq, cpndatepq, cpndaysn, cpndaysp, cpnpersz

cpndatepq

Purpose

Previous quasi coupon date for fixed income security.

Syntax

```
PreviousQuasiCouponDate = cpndatepq(Settle, Maturity, Period,
  Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.

`Settle` and `Maturity` are required arguments. All others are optional. Required arguments must be `NUMBONDS`-by-`1` or `1`-by-`NUMBONDS` conforming vectors or scalars. Optional arguments must be either `NUMBONDS`-by-`1` or `1`-by-`NUMBONDS` conforming vectors, scalars, or empty matrices. Fill unspecified entries in input vectors with the value `NaN`. Dates can be serial date numbers or date strings.

Description

`PreviousQuasiCouponDate = cpndatepq(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` determines the previous quasi coupon date for a set of `NUMBONDS` fixed income securities. This function finds the previous quasi coupon date for a bond with a coupon structure in which the first or last period is either normal, short, or long (whether or not the coupon structure is synchronized to maturity). For zero coupon bonds this function returns the theoretical previous coupon date that would prevail if the bond were a coupon bond.

The term "previous quasi coupon date" refers to the previous coupon date for a bond calculated as if no issue date were specified. Although the issue date is not actually a coupon date, when issue date is specified, the previous actual coupon date for a bond is normally calculated as being either the previous date or the issue date, whichever is greater. Also, the actual previous coupon date and the previous quasi coupon date can differ when the maturity date is not synchronized with the coupon structure and the settlement date is the maturity date. This function always returns the previous quasi coupon date whether or not the issue date is greater. This function only returns a maturity

date as a previous coupon date if it is synchronized with coupon structure of the bond. If the settlement date is a coupon date, this function always returns the settlement date.

PreviousQuasiCouponDate is an NUMBONDS- by-1 vector.

Examples

Given a pair of bonds with the characteristics:

```
Settle = str2mat('30-May-1992','10-Dec-1992')
Maturity = str2mat('30-Nov-1997','10-Jun-1999')
```

With no FirstCouponDate explicitly supplied, compute the PreviousCouponDate for this pair of bonds:

```
PreviousCouponDate = cpndatep(Settle, Maturity)

datestr(PreviousCouponDate)

ans =

30-Nov-1991
10-Dec-1992
```

Note that since the settlement date for the second bond is also a coupon date, cpndatep returns this date as the previous coupon date.

Now establish a FirstCouponDate and IssueDate for this pair of bonds:

```
FirstCouponDate = str2mat('30-Nov-1992','10-Dec-1993')
IssueDate = str2mat('30-May-1991', '10-Dec-1991')
```

Recompute the PreviousCouponDate for this pair of bonds:

```
PreviousCouponDate = cpndatep(Settle, Maturity, 2, 0, 1, ...
IssueDate, FirstCouponDate)

ans =

30-May-1991
10-Dec-1991
```

Since both of these bonds settled before the first coupon had been paid, cpndatep returns the IssueDate as the PreviousCouponDate.

Using the same data, compute PreviousQuasiCouponDate:

```
PreviousQuasiCouponDate = cpndatep(Settle, Maturity, 2, 0, 1, ...
IssueDate, FirstCouponDate)

ans =

30-Nov-1991
10-Dec-1992
```

For the first bond the settlement date is not a normal coupon date. For this bond PreviousQuasiCouponDate indicates the date (30-Nov-1991) on which a coupon would normally be paid if an explicit FirstCouponDate had not been specified. For the second bond the settlement date (10-Dec-1992) occurs on a date when a coupon would normally be paid in the absence of an explicit FirstCouponDate. cpndatepq returns this date as PreviousQuasiCouponDate.

See Also

accrfrac, cfamounts, cfdates, cftimes, cpncount, cpndaten, cpndatenq, cpndatep, cpndaysn, cpndaysp, cpnpersz

cpndaysn

Purpose

Number of days to next coupon date.

Syntax

```
NumDaysNext = cpndaysn(Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.

`Settle` and `Maturity` are required arguments. All others are optional. Required arguments must be N-by-1 or 1-by-N conforming vectors or scalars. Optional arguments must be either N-by-1 or 1-by-N conforming vectors, scalars, or empty matrices.

Description

`NumDaysNext = cpndaysn(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` returns the number of days from the settlement date to the next coupon date for a bond or set of bonds.

Examples

```
NumDaysNext = cpndaysn('14 Sep 1997', '30 Jun 1998', 2, 0, 0)
NumDaysNext =
   107

NumDaysNext = cpndaysn('14 Sep 1997', '30 Jun 1998', 2, 0, 1)
NumDaysNext =
   108

Maturity = ['30 Apr 1998'; '31 May 1998'; '30 Jun 1998'];
NumDaysNext = cpndaysn('14 Sep 1997', Maturity)
NumDaysNext =
    47
    77
   108
```

See Also

accrfrac, cfamounts, cftimes, cfdates, cpncount, cpndaten, cpndatenq, cpndatep, cpndatepq, cpndaysp, cpnpersz

Purpose

Number of days since previous coupon date.

Syntax

```
NumDaysPrevious = cpndaysp(Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.

`Settle` and `Maturity` are required arguments. All others are optional. Required arguments must be N-by-1 or 1-by-N conforming vectors or scalars. Optional arguments must be either N-by-1 or 1-by-N conforming vectors, scalars, or empty matrices.

Description

`NumDaysPrevious = cpndaysp(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` returns the number of days between the previous coupon date and the settlement date for a bond or set of bonds. When the coupon frequency is 0 (a zero coupon bond), the previous coupon date is calculated as if the frequency were semi-annual.

Examples

```
NumDaysPrevious = cpndaysp('14 Mar 1997', '30 Jun 1998', 2, 0, 0)
NumDaysPrevious =
    74

NumDaysPrevious = cpndaysp('14 Mar 1997', '30 Jun 1998', 2, 0, 1)
NumDaysPrevious =
    73

Maturity = ['30 Apr 1998'; '31 May 1998'; '30 Jun 1998'];
NumDaysPrevious = cpndaysp('14 Mar 1997', Maturity)
NumDaysPrevious =
   134
   104
    73
```

See Also

accrfrac, cfamounts, cfdates, cftimes, cpncount, cpndaten, cpndatenq, cpndatep, cpndatepq, cpndaysn, cpnpersz

cpnpersz

Purpose

Number of days in coupon period.

Syntax

```
NumDaysPeriod = cpnpersz(Settle, Maturity, Period, Basis,
  EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate,
  StartDate)
```

Arguments

`Settle`	Settlement date. A vector of serial date numbers or date strings. `Settle` must be earlier than or equal to `Maturity`.
`Maturity`	Maturity date. A vector of serial date numbers or date strings.
`Period`	Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
`Basis`	Day-count basis of the bond. A vector of integers. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`EndMonthRule`	End-of-month rule. A vector. This rule applies only when `Maturity` is an end-of-month date for a month having 30 or fewer days. `0` = ignore rule, meaning that a bond's coupon payment date is always the same numerical day of the month. `1` = set rule on (default), meaning that a bond's coupon payment date is always the last actual day of the month.
`IssueDate`	Date when a bond was issued.
`FirstCouponDate`	Date when a bond makes its first coupon payment. When `FirstCouponDate` and `LastCouponDate` are both specified, `FirstCouponDate` takes precedence in determining the coupon payment structure.

`LastCouponDate`	Last coupon date of a bond prior to the maturity date. In the absence of a specified `FirstCouponDate`, a specified `LastCouponDate` determines the coupon structure of the bond. The coupon structure of a bond is truncated at the `LastCouponDate` regardless of where it falls and will be followed only by the bond's maturity cash flow date.
`StartDate`	Date when a bond actually starts (the date from which a bond's cash flows can be considered). To make an instrument forward-starting, specify this date as a future date. If `StartDate` is not explicitly specified, the effective start date is the settlement date.

`Settle` and `Maturity` are required arguments. All others are optional. Required arguments must be N-by-1 or 1-by-N conforming vectors or scalars. Optional arguments must be either N-by-1 or 1-by-N conforming vectors, scalars, or empty matrices.

Description

`NumDaysPeriod = cpnpersz(Settle, Maturity, Period, Basis, EndMonthRule, IssueDate, FirstCouponDate, LastCouponDate, StartDate)` returns the number of days in the coupon period containing the settlement date.

Examples

```
NumDaysPeriod = cpnpersz('14 Sep 1997', '30 Jun 1998', 2, 0, 0)
NumDaysPeriod =
   183

NumDaysPeriod = cpnpersz('14 Sep 1997', '30 Jun 1998', 2, 0, 1)
NumDaysPeriod =
   184

Maturity = ['30 Apr 1998'; '31 May 1998'; '30 Jun 1998'];
NumDaysPeriod = cpnpersz('14 Sep 1997', Maturity)
NumDaysPeriod =
   184
   183
   184
```

See Also

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

Purpose

Decimal currency values to fractional values.

Syntax

```
f = cur2frac(p, d)
```

Description

`f = cur2frac(p, d)` converts decimal currency values to fractional values. `p` is the decimal currency value and `d` is the denominator of the fraction. `f` is returned as a string.

Example

```
f = cur2frac(12.125, 8)
```

returns `f = 12.1`, a string.

See Also

`cur2str`, `frac2cur`

Purpose

Bank formatted text.

Syntax

```
t = cur2str(p, d)
t = cur2str(p)
```

Description

`t = cur2str(p, d)` returns the given value in bank format. `p` is the given value to be formatted and `d` is the number of significant digits. By default, `d = 2`. A negative `d` rounds the value to the left of the decimal point. `t` is returned as a string with a leading dollar sign ($). Negative numbers are displayed in parentheses.

Example

```
t = cur2str(-8264, 2)
```

returns `t = ($8264.00)`

See Also

`cur2frac`, `frac2cur`

Purpose

Convert serial-date axis labels to calendar-date axis labels.

Syntax

```
dateaxis(aksis, dateform, startdate)
dateaxis(aksis, dateform)
dateaxis(aksis)
dateaxis
```

Arguments

`aksis` Determines which axis tick labels—*x*, *y*, or *z*—to replace. Enter as a string. Default = `'x'`.

`dateform` Specifies which date format to use. Enter as an integer from 0 to 16. If no `dateform` argument is entered, this function determines the date format based on the span of the axis limits. For example, if the difference between the axis minimum and maximum is less than 15, the tick labels are converted to three-letter day-of-the-week abbreviations (`dateform = 8`). See `dateform` format descriptions below.

`startdate` Assigns the date to the first axis tick value. Enter as a string. The tick values are treated as serial date numbers. The default `startdate` is the lower axis limit converted to the appropriate date number. For example, a tick value of 1 is converted to the date 01-Jan-0000. Entering `startdate` as `'06-apr-1995'` assigns the date April 6, 1995 to the first tick value and the axis tick labels are set accordingly.

Description

`dateaxis(aksis, dateform, startdate)` replaces axis tick labels with date labels on a graphic figure.

See *Using MATLAB Graphics* for information on using `set` to modify the axis tick values and other axis parameters.

dateform	Format	Description
0	`01-Mar-1995 15:45:17`	day-month-year hour:minute:second
1	`01-mar-1995`	day-month-year
2	`03/01/95`	month/day/year
3	`Mar`	month, three letters
4	`M`	month, single letter
5	`3`	month
6	`03/01`	month/day
7	`1`	day of month
8	`Wed`	day of week, three letters
9	`W`	day of week, single letter
10	`1995`	year, four digits
11	`95`	year, two digits
12	`Mar95`	month year
13	`15:45:17`	hour:minute:second
14	`03:45:17 PM`	hour:minute:second AM or PM
15	`15:45`	hour:minute
16	`03:45 PM`	hour:minute AM or PM

Examples

```
dateaxis('x') or dateaxis
```

converts the *x*-axis labels to an automatically determined date format.

```
dateaxis('y', 6)
```

converts the *y*-axis labels to the month/day format.

```
dateaxis('x', 2, '03/03/1995')
```

converts the *x*-axis labels to the month/day/year format. The minimum *x*-tick value is treated as March 3, 1995.

See Also

`bolling`, `candle`, `datenum`, `datestr`, `highlow`, `movavg`, `pointfig`

datefind

Purpose

Indices of date numbers in matrix.

Syntax

```
ind = datefind(sub, super, tol)
ind = datefind(sub, super)
```

Arguments

`sub` Subset matrix of date numbers used to find matching date numbers in `super`. These date numbers must be a non-repeating subset of those in `super`.

`super` Superset matrix of non-repeating date numbers whose elements are sought.

`tol` Tolerance (+/-) for matching the date numbers in `super`. A positive integer. Default = `0`.

Description

`ind = datefind(sub, super, tol)` returns a vector of indices to the date numbers in `super` that are present in `sub`, plus or minus the tolerance `tol`. If no date numbers match, `ind = []`.

Although this function was designed for use with sequential date numbers, you can use it with any non-repeating integers.

Example

```
super = datenum(1997, 7, 1:31);
sub = [datenum(1997, 7, 10); datenum(1997, 7, 20)];
ind = datefind(sub, super, 1)
ind =
     9
    10
    11
    19
    20
    21
```

See Also

`datenum`

Purpose

Date of day in future or past month.

Syntax

```
d = datemnth(d1, mm, df, basis, eom)
d = datemnth(d1, mm, df, basis)
d = datemnth(d1, mm, df)
d = datemnth(d1, mm)
```

Arguments

`d1` Start date. Enter as serial date number or date string.

`mm` Number of months in future (positive) or past (negative). An integer.

`df` Day flag: `0` = day of month in `d` corresponds to the day of month in the start date or the last day of `d`, as appropriate (default); `1` = day of month in `d` is the first day of month; `2` = day of month in `d` is last day of month. This flag has no effect if `eom` applies and is set to `1`.

`basis` Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

`eom` End-of-month flag. This flag applies only when the start date `d1` is an end-of-month date. `0` = ignore flag (default), or `1` = force `d` to the end-of-month. When `eom` applies and is set to `0`, the setting of `df` controls the day of month in `d`. When `eom` applies and is set to `1`, `d` is always the last day of the month.

Description

`d = datemnth(d1, mm, df, basis, eom)` returns the serial date number `d` for a date `mm` months in the future or past.

Any input can contain multiple values, but if so, all other inputs must contain the same number of values or a single value that applies to all. For example, if `d1` is an N-row character array of date strings, then `mm` must be an N-by-1 vector of integers or a single integer. `d` is then an N-by-1 vector of date numbers.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

```
d = datemnth('3 jun 1997', 6, 0, 0, 0)
d =
      729727
datestr(d)
ans =
03-Dec-1997

d = datemnth('3 jun 1997', 6, 1, 0, 1); datestr(d)
ans =
01-Dec-1997

d = datemnth('31 jan 1997', 5, 0, 0, 0); datestr(d)
ans =
30-Jun-1997

d = datemnth('31 jan 1997', 5, 1, 0, 0); datestr(d)
ans =
01-Jun-1997

d = datemnth('31 jan 1997', 5, 1, 0, 1); datestr(d)
ans =
30-Jun-1997

d = datemnth('31 jan 1997', 5, 2, 0, 1); datestr(d)
ans =
30-Jun-1997

mm = [1; 3; 5; 7; 9];
d = datemnth('31 jan 1997', mm); datestr(d)
ans =
28-Feb-1997
30-Apr-1997
30-Jun-1997
31-Aug-1997
31-Oct-1997
```

See Also

datestr, datevec, days360, days365, daysact, daysdif, wrkdydif

Purpose

Create date number.

Syntax

```
n = datenum(ds)
n = datenum(ds, p)
n = datenum(y, mo, d)
n = datenum(y, mo, d, h, mi, s)
n = datenum(dn)
```

Description

`n = datenum(ds)` returns a serial date number `n` given a date string `ds`. Date numbers are the number of days that has passed since a base date. In MATLAB, date number 1 is January 1, 0000 A.D. If the input includes time components, the date number includes a fractional component. If the input is only a time component, the date number is only a fractional time component.

The date string `ds` can be any of several forms:

```
'19-may-1995'
'may 19, 1995'
'19-may-95'
'19-may' (current year assumed)
'5/19/95'
'5/19' (current year assumed)
'19-may-1995, 18:37'
'19-may-1995, 6:37 pm'
'5/19/95/18:37'
'5/19/95/6:37 pm'
'18:37'
```

Date strings with two-character years, e.g., `12-june-12`, are assumed to lie within the 100-year period centered about the current year.

`n = datenum(ds,p)` assumes that two-character years lie within the 100-year period beginning with the pivot year `p` . The default pivot year is the current year minus 50 years.

`n = datenum(y, mo, d)` returns a serial date number `n` given year, month, and day integers `y`, `mo`, and `d`.

`n = datenum(y, mo, d, h, mi, s)` returns a serial date number `n` given year, month, day, hour, minute, and second integers `y`, `mo`, `d`, `h`, `mi`, and `s`.

`n = datenum(dn)` returns a serial date number `n` given a serial date number `dn`.

Note: This function now ships with basic MATLAB. It originally shipped only with the Financial Toolbox, and this description remains here for your convenience.

Examples

```
n = datenum('19-may-1995')
n = 728798

n = datenum('5/19/95')
n = 728798

n = datenum('19-may-1995, 6:37 pm')
n = 728798.78

n = datenum('5/19/95/18:37')
n = 728798.78

n = datenum('6:37 pm')
n = 0.78

n = datenum(1995, 5, 19)
n = 728798

n = datenum(1995, 1:6, 19)
n = [728678  728709  728737  728768  728798  728829]

n = datenum(1995, 5, 19, 18, 37, 0)
n = 728798.78

n = datenum(728798)
n = 728798
```

The next example demonstrates the use of the pivot year in interpreting date strings with two-character years.

```
n = datenum('12-june-12')
n =
       735032
datestr(735032)
ans =
12-Jun-2012

 n = datenum('12-june-12',1900)
n =
       698507
datestr(698507)
ans =
12-Jun-1912
```

See Also

datestr, datevec, daysact, now, today

Purpose

Create date string.

Syntax

```
ds = datestr(d, dateform)
ds = datestr(d, dateform, p)
ds = datestr(d)
```

Description

`ds = datestr(d, dateform)` converts a date number or a date string `d` to a date string `ds`. `dateform` specifies the format of `ds`. Date strings with two-character years, e.g., `12-june-12`, are assumed to lie within the 100-year period centered about the current year.

`ds = datestr(d, dateform, p)` assumes that two-character years lie within the 100-year period beginning with the pivot year `p` . The default pivot year is the current year minus 50 years.

`ds = datestr(d)` assumes `dateform` is `1`, `16`, or `0` depending on whether the date number `d` contains a date, time, or both, respectively. If `d` is a date string, the function assumes `dateform` is `1`.

dateform	Format	Description
0	01-Mar-1995 15:45:17	day-month-year hour:minute:second
1	01-Mar-1995	day-month-year
2	03/01/95	month/day/year
3	Mar	month, three letters
4	M	month, single letter
5	3	month
6	03/01	month/day
7	1	day of month
8	Wed	day of week, three letters
9	W	day of week, single letter
10	1995	year, four digits

dateform	Format	Description
11	95	year, two digits
12	Mar95	month year
13	15:45:17	hour:minute:second
14	03:45:17 PM	hour:minute:second AM or PM
15	15:45	hour:minute
16	03:45 PM	hour:minute AM or PM
17	Q1-95	calendar quarter-year
18	Q1	calendar quarter

Note: This function now ships with basic MATLAB. It originally shipped only with the Financial Toolbox, and this description remains here for your convenience.

Examples

```
ds = datestr(728798, 1)
ds = 19-May-1995

ds = datestr(728798, 2)
ds = 05/19/95

ds = datestr(728798, 12)
ds = May95

ds = datestr(728798.776, 0)
ds = 19-May-1995 18:37:26

ds = datestr('5/19', 1) (assuming the year is 1995)
ds = 19-May-1995

ds = datestr(728798)
ds = 19-May-1995

ds = datestr([728678 728709 728737 728768 728798 728829])
ds =
    19-Jan-1995
    19-Feb-1995
    19-Mar-1995
    19-Apr-1995
    19-May-1995
    19-Jun-1995

ds = datestr('5/19')
ds = 19-May-1995 (assuming the year is 1995)
```

See Also

dateaxis, datenum, datevec, daysact, now, today

Purpose

Date components.

Syntax

```
dv = datevec(d)
dv = datevec(d, p)
[y, mo, d, h, mi, s] = datevec(d)
```

Description

`dv = datevec(d)` converts a date number or a date string `d` to a date vector `dv` whose elements are [year month day hour minute second]. The first five elements are integers, the sixth is a floating-point number. Date strings with two-character years, e.g., `12-june-12`, are assumed to lie within the 100-year period centered about the current year.

`dv = datevec(d, p)` assumes that two-character years lie within the 100-year period beginning with the pivot year `p` . The default pivot year is the current year minus 50 years.

`[y, mo, d, h, mi, s] = datevec(d)` converts a date number or a date string `d` to a date vector and returns the components of the date vector as individual variables `y` year, `mo` month, `d` day, `h` hour, `mi` minute, and `s` second.

Note: This function now ships with basic MATLAB. It originally shipped only with the Financial Toolbox, and this description remains here for your convenience.

Examples

```
dv=datevec('19-may-1995')
dv =
        1995       5      19       0       0       0

dv=datevec(728798)
dv =
        1995       5      19       0       0       0

dv=datevec(728798.776)
dv =
        1995       5      19      18      37    26.4

[y, mo, d, h, mi, s] = datevec(728798.776)
y =
  1995
mo =
     5
d =
    19
h =
    18
mi =
    37
s =
    26.4

[y, mo, d] = datevec(728798:728800)
y =
    1995    1995    1995
mo =
       5       5       5
d =
      19      20      21
```

See Also

datenum, datestr, now, today

Purpose

Date of future or past workday.

Syntax

```
d = datewrkdy(start, nd, hols)
d = datewrkdy(start, nd)
```

Arguments

`start` Starting date. Enter as serial date number or date string.

`nd` Number of working days in future (positive) or past (negative), including the starting date. An integer.

`hols` Number of holidays within `nd`. An integer. `hols` and `nd` must have the same sign.

Description

`d = datewrkdy(start, nd, hols)` returns the serial number of the date a given number of workdays before or after the start date.

Any input can contain multiple values, but if so, all other inputs must contain the same number of values or a single value that applies to all. For example, if `start` is an N-row character array of date strings, then `nd` must be an N-by-1 vector of integers or a single integer. `d` is then an N-by-1 vector of date numbers.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

```
d = datewrkdy('12-dec-1997',  16,  2);
datestr(d)
ans =
06-Jan-1998

nd = [16; 20; 44];
d = datewrkdy('12-dec-1997',  nd,  2);
datestr(d)
ans =
06-Jan-1998
12-Jan-1998
13-Feb-1998
```

See Also

`busdate`, `holidays`, `isbusday`, `wrkdydif`

day

Purpose

Day of month.

Syntax

```
dom = day(d)
```

Description

`dom = day(d)` returns the day of the month given a serial date number or date string `d`.

Example

```
dom = day(728744)
```

or

```
dom = day('3/26/95')
```

returns `dom = 26`

See Also

`datevec`, `eomday`, `month`, `year`

Purpose

Days between dates based on 360-day year.

Syntax

```
nd = days360(d1, d2)
nd = days360(d1, d2, eom)
```

Description

`nd = days360(d1, d2)` returns the number of days `nd` between dates `d1` and `d2` based on a 360-day year. If `d2` is earlier than `d1`, `nd` is negative. Enter dates as serial date numbers or date strings.

Either input argument can contain multiple values, but if so, the other input must contain the same number of values or a single value that applies to all. For example, if `d1` is an N-row character array of date strings, then `d2` must be an N-row character array of date strings or a single date. `nd` is then an N-by-1 vector of numbers.

Note: The `eom` argument now has no effect. The argument was mistakenly included in the previous release due to a misinterpretation of SIA guidelines. (The end-of-month rule applies only to the fixing of coupon dates for bonds with a 30/360 basis, not to the actual measurement of days between two dates on a 30/360 basis.) The `eom` argument is still allowed to provide compatibility for applications that use it, but it will be removed in a future release.

Examples

```
nd = days360('15-jan-1997', '15-mar-1997')
nd =
     60

d2 = ['15-mar-1997'; '15-apr-1997'; '15-jun-1997'];
nd = days360('15-jan-1997', d2)
nd =
     60
     90
    150
```

See Also

`days365`, `daysact`, `daysdif`, `wrkdydif`, `yearfrac`

Reference

Addendum to Securities Industry Association, *Standard Securities Calculation Methods: Fixed Income Securities Formulas for Analytic Measures*, Vol. 2, Spring 1995.

days365

Purpose

Days between dates based on 365-day year.

Syntax

```
nd = days365(d1, d2)
```

Description

`nd = days365(d1, d2)` returns the number of days between dates `d1` and `d2` based on a 365-day year. If `d2` is earlier than `d1`, `nd` is negative. Enter dates as serial date numbers or date strings.

Either input argument can contain multiple values, but if so, the other input must contain the same number of values or a single value that applies to all. For example, if `d1` is an N-row character array of date strings, then `d2` must be an N-row character array of date strings or a single date. `nd` is then an N-by-1 vector of numbers.

Examples

```
nd = days365('15-jan-1997', '15-mar-1997')
nd =
    59

d2 = ['15-mar-1997'; '15-apr-1997'; '15-jun-1997'];
nd = days365('15-jan-1997', d2)
nd =
    59
    90
   151
```

See Also

`days360`, `daysact`, `daysdif`, `wrkdydif`, `yearfrac`

Purpose

Actual number of days between dates.

Syntax

```
nd = daysact(d1, d2)
nd = daysact(d1)
```

Description

`nd = daysact(d1, d2)` returns the actual number of days between two dates `d1` and `d2`. Enter dates as serial date numbers or date strings. `nd` is negative if `d2` is earlier than `d1`.

`nd = daysact(d1)` returns the actual number of days between the MATLAB base date and date `d1`. In MATLAB, the base date 1 is 1-Jan-0000 A.D. See `datenum` for a similar function.

Either input argument can contain multiple values, but if so, the other input must contain the same number of values or a single value that applies to all. For example, if `d1` is an N-row character array of date strings, then `d2` must be an N-row character array of date strings or a single date. `nd` is then an N-by-1 vector of numbers.

Examples

```
nd = daysact('7-sep-1997',  '25-dec-1997')
nd =
   109

nd = daysact('9/7/1997')
nd =
      729640

d1 = ['09/07/1997'; '10/22/1997'; '11/05/1997'];
nd = daysact(d1, '12/25/1997')
nd =
   109
    64
    50
```

See Also

`datenum`, `datevec`, `days360`, `days365`, `daysdif`

daysdif

Purpose

Days between dates for any day-count basis.

Syntax

```
nd = daysdif(d1, d2, basis)
nd = daysdif(d1, d2)
```

Description

`nd = daysdif(d1, d2, basis)` returns the number of days between dates `d1` and `d2` using the given day-count basis. Enter dates as serial date numbers or date strings.

`basis` is the day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Any input argument can contain multiple values, but if so, all other inputs must contain the same number of values or a single value that applies to all. For example, if `d1` is an N-row character array of date strings, then `d2` must be an N-row character array of date strings or a single date. `nd` is then an N-by-1 vector of numbers.

This function is a helper function for the bond pricing and yield functions, and is designed to make the code more readable and to eliminate redundant calls within `if` statements.

Examples

```
nd = daysdif('3/1/97', '3/1/98', 1)
nd =
   360

d2 = ['3/1/1998'; '3/1/1999'; '3/1/2000'];
nd = daysdif('3/1/97', d2)
nd =
         365
         730
        1096
```

See Also

`datenum`, `days360`, `days365`, `daysact`, `wrkdydif`, `yearfrac`

Purpose

Fixed declining-balance depreciation.

Syntax

```
d = depfixdb(cost, salvage, life, period, month)
d = depfixdb(cost, salvage, life, period)
```

Arguments

`cost` Initial value of the asset.

`salvage` Salvage value of the asset.

`life` Life of the asset in years.

`period` Number of years to calculate.

`month` Number of months in the first year of asset life. Default = 12.

Description

`d = depfixdb(cost, salvage, life, period, month)` calculates `d`, the fixed declining-balance depreciation for each period.

Example

A car is purchased for $11,000 with a salvage value of $1500 and a lifetime of eight years. To calculate the depreciation for the first five years:

```
d = depfixdb(11000, 1500, 8, 5)
```

returns

```
d =
       2425.08     1890.44     1473.67     1148.78      895.52
```

See Also

`depgendb`, `deprdv`, `depsoyd`, `depstln`

depgendb

Purpose

General declining-balance depreciation.

Syntax

```
d = depgendb(cost, salvage, life, factor)
```

Arguments

`cost` Cost of the asset.

`salvage` Estimated salvage value of the asset.

`life` Number of periods over which the asset is depreciated.

`factor` Depreciation factor. `factor = 2` uses the double-declining-balance method.

Description

`d = depgendb(cost, salvage, life, factor)` calculates `d`, the declining-balance depreciation for each period.

Example

A car is purchased for $11,000 and is to be depreciated over five years. The estimated salvage value is $1000. Using the double-declining-balance method, the function calculates the depreciation for each year and returns the remaining depreciable value at the end of the life of the car.

```
d = depgendb(11000, 1000, 5, 2)
```

returns

```
d =
       4400.00     2640.00     1584.00     950.40     425.60
```

See Also

`depfixdb`, `deprdv`, `depsoyd`, `depstln`

Purpose

Remaining depreciable value.

Syntax

```
r = deprdv(cost, salvage, accum)
```

Arguments

`cost` Cost of the asset.

`salvage` Salvage value of the asset.

`accum` Accumulated depreciation of the asset for prior periods.

Description

`r = deprdv(cost, salvage, accum)` returns the remaining depreciable value for an asset.

Example

The cost of an asset is $13,000 with a life of 10 years. The salvage value is $1000. First find the accumulated depreciation with the straight-line depreciation function, `depstln`. Then find the remaining depreciable value after six years.

```
accum = depstln(13000, 1000, 10) * 6
accum =
       7200.00
r = deprdv(13000, 1000, 7200)
r =
       4800.00
```

See Also

`depfixdb`, `depgendb`, `depsoyd`, `depstln`

depsoyd

Purpose

Sum of years' digits depreciation.

Syntax

```
sd = depsoyd(cost, salvage, life)
```

Arguments

`cost` Cost of the asset.

`salvage` Salvage value of the asset.

`life` Depreciable life of the asset in years.

Description

`sd = depsoyd(cost, salvage, life)` calculates the depreciation for an asset using the sum of years' digits method. `sd` is a 1-by-`life` vector of depreciation values with each element corresponding to a year of the asset's life.

Example

The cost of an asset is $13,000 with a life of 10 years. The salvage value of the asset is $1000.

```
sd = depsoyd(13000, 1000, 10)'
```

returns

```
sd =
       2181.82
       1963.64
       1745.45
       1527.27
       1309.09
       1090.91
        872.73
        654.55
        436.36
        218.18
```

See Also

`depfixdb`, `depgendb`, `deprdv`, `depstln`

Purpose

Straight-line depreciation.

Syntax

```
sl = depstln(cost, salvage, life)
```

Arguments

`cost` Cost of the asset.

`salvage` Salvage value of the asset.

`life` Depreciable life of the asset in years.

Description

`sl = depstln(cost, salvage, life)` calculates straight-line depreciation for an asset.

Example

The cost of an asset is $13,000 with a life of 10 years. The salvage value of the asset is $1000.

```
sl = depstln(13000, 1000, 10)
```

returns

```
sl =
        1200
```

See Also

`depfixdb`, `depgendb`, `deprdv`, `depsoyd`

disc2zero

Purpose

Zero curve given a discount curve.

Syntax

```
[zr, cd] = disc2zero(dr, cd, sd, ocomp, obasis)
[zr, cd] = disc2zero(dr, cd, sd, ocomp)
[zr, cd] = disc2zero(dr, cd, sd)
```

Arguments

`dr` Discount factors. An N-by-1 vector of discount factors, as decimal fractions. In aggregate, the factors in `dr` constitute a discount curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the discount factors in `dr`. Use `datenum` to convert date strings to serial date numbers.

`sd` Settlement date. A serial date number that is the common settlement date for the discount rates in `dr`.

`ocomp` Output compounding. A scalar that sets the compounding frequency per year for annualizing the output zero rates in `zr`. Allowed values are:

- `1` = annual compounding
- `2` = semi-annual compounding (default)
- `3` = compounding three times per year
- `4` = quarterly compounding
- `6` = bimonthly compounding
- `12` = monthly compounding
- `365` = daily compounding
- `–1` = continuous compounding

`obasis` Output day-count basis for annualizing the output zero rates in `zr`. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[zr, cd] = disc2zero(dr, cd, sd, ocomp, obasis)` returns a zero curve given a discount curve and its maturity dates.

`zr` Zero rates. An N-by-1 vector of decimal fractions. In aggregate, the rates in `zr` constitute a zero curve for the investment horizon represented by `cd`. The zero rates are the yields to maturity on theoretical zero-coupon bonds.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the zero rates in `zr`. This vector is the same as the input vector `cd`. Use `datestr` to convert serial date numbers to date strings.

Example

Given discount factors `dr` over a set of maturity dates `cd`, and a settlement date `sd`:

```
dr = [0.9996
      0.9947
      0.9896
      0.9866
      0.9826
      0.9786
      0.9745
      0.9665
      0.9552
      0.9466];

cd = [datenum('06-Nov-1997')
      datenum('11-Dec-1997')
      datenum('15-Jan-1998')
      datenum('05-Feb-1998')
      datenum('04-Mar-1998')
      datenum('02-Apr-1998')
      datenum('30-Apr-1998')
      datenum('25-Jun-1998')
      datenum('04-Sep-1998')
      datenum('12-Nov-1998')];

sd = datenum('03-Nov-1997');
```

Set daily compounding for the output zero curve, on an actual/365 basis.

```
ocomp = 365;
obasis = 3;
```

Execute the function

```
[zr, cd] = disc2zero(dr, cd, sd, ocomp, obasis)
```

which returns the zero curve zr at the maturity dates cd:

```
zr =
    0.0487
    0.0510
    0.0523
    0.0524
    0.0530
    0.0526
    0.0530
    0.0532
    0.0549
    0.0536
cd =
      729700
      729735
      729770
      729791
      729818
      729847
      729875
      729931
      730002
      730071
```

(For readability, dr and zr are shown here only to the basis point. However, MATLAB computed them at full precision. If you enter dr as shown, zr may differ due to rounding.)

See Also

zero2disc and other functions for Term Structure of Interest Rates

Purpose

Discount rate of a security.

Syntax

```
d = discrate(sd, md, rv, price, basis)
d = discrate(sd, md, rv, price)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`price`	Price of the security.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`d = discrate(sd, md, rv, price, basis)` finds the discount rate of a security.

Example

```
d = discrate('12-jan-1994', '25-jun-1994', 100, 97.74, 0)
```

returns

```
d =
    0.0503
```

a discount rate of 5.03%.

See Also

`acrudisc`, `fvdisc`, `prdisc`, `ylddisc`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formula 1.

effrr

Purpose

Effective rate of return.

Syntax

```
r = effrr(apr, per)
```

Arguments

`apr` Annual percentage rate. Enter as a decimal fraction.

`per` Number of compounding periods per year, an integer.

Description

`r = effrr(apr, per)` calculates the annual effective rate of return. Compounding continuously returns `r` equivalent to `(e^apr-1)`.

Example

Find the effective annual rate of return based on an annual percentage rate of 9% compounded monthly.

```
r = effrr(0.09, 12)
```

returns

```
r =
    0.0938 or 9.38%
```

See Also

`nomrr`

Purpose

Last date of month.

Syntax

```
d = eomdate(y, m)
```

Description

`d = eomdate(y, m)` returns the serial date number `d` of the last date of the month for the given year `y` and month `m`. Enter `y` as a four-digit integer; enter `m` as an integer from 1 to 12.

Either input argument can contain multiple values, but if so, the other input must contain the same number of values or a single value that applies to all. For example, if `y` is a 1-by-N vector of integers, then `m` must be a 1-by-N vector of integers or a single integer. `d` is then a 1-by-N vector of date numbers.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

```
d = eomdate(1997, 2)
d =
      729449
datestr(d)
ans =
28-Feb-1997

y = [1997 1998 1999 2000];
d = eomdate(y, 2)
d =
      729449      729814      730179      730545
datestr(d)
ans =
28-Feb-1997
28-Feb-1998
28-Feb-1999
29-Feb-2000
```

See Also

`day`, `eomday`, `lbusdate`, `month`, `year`

eomday

Purpose

Last day of month.

Syntax

```
x = eomday(y, m)
```

Description

`x = eomday(y, m)` returns the last day of the month for the given year `y`, and month `m`.

Note: This function now ships with basic MATLAB. It originally shipped only with the Financial Toolbox, and this description remains here for your convenience.

Example

```
x = eomday(96, 2)

x =
    29
```

See Also

`day`, `eomdate`, `month`

Purpose

Expected return and covariance from return time series.

Syntax

```
[ExpReturn, ExpCovariance, NumEffObs] = ewstats(RetSeries,
  DecayFactor, WindowLength)
```

Arguments

`RetSeries`	Return Series: number of observations (`NUMOBS`)-by-number of assets (`NASSETS`) matrix of equally spaced incremental return observations. The first row is the oldest observation, and the last row is the most recent.
`DecayFactor`	Controls how much less each observation is weighted than its successor. The `k`'th observation back in time has weight `DecayFactor^k`. `DecayFactor` must lie in the range: `0 < DecayFactor <= 1`. Default = `1`, the equally weighted linear moving average model (BIS).
`WindowLength`	Number of recent observations in the computation. Default = `NUMOBS`.

Description

`[ExpReturn, ExpCovariance, NumEffObs] = ewstats(RetSeries, DecayFactor, WindowLength)` computes estimated expected returns, estimated covariance matrix, and the number of effective observations.

`ExpReturn` is a `1`-by-`NASSETS` vector of estimated expected returns.

`ExpCovariance` is an `NASSETS`-by-`NASSETS` estimated covariance matrix.The standard deviations of the asset return processes are given by:

```
STDVec = sqrt(diag(ECov))
```

The correlation matrix is:

```
CorrMat = VarMat./( STDVec*STDVec' )
```

`NumEffObs` is the number of effective observations = `(1-DecayFactor^WindowLength)/(1-DecayFactor)`.

A smaller `DecayFactor` or `WindowLength` emphasizes recent data more strongly but uses less of the available data set.

Example

```
RetSeries = [ 0.24 0.08
              0.15 0.13
              0.27 0.06
              0.14 0.13 ];

  DecayFactor = 0.98;

  [ExpReturn, ExpCovariance] = ewstats(RetSeries, DecayFactor)

ExpReturn =

    0.1995    0.1002

ExpCovariance =

    0.0032   -0.0017
   -0.0017    0.0010
```

See Also

cov, mean

Purpose

First business date of month.

Syntax

```
d = fbusdate(y, m, hol)
d = fbusdate(y, m)
```

Arguments

`y` Year. Enter as four-digit integer.

`m` Month. Enter as integer from 1 to 12.

`hol` Vector of holidays and non-trading-day dates. All dates in `hol` must be the same format: either serial date numbers or date strings. (Using date numbers improves performance.) The `holidays` function supplies the default vector.

Description

`d = fbusdate(y, m, hol)` returns the serial date number for the first business date of the given year `y` and month `m` where `hol` specifies non-trading days.

`y` and `m` can contain multiple values. If one contains multiple values, the other must contain the same number of values or a single value that applies to all. For example, if `y` is a 1-by-N vector of integers, then `m` must be a 1-by-N vector of integers or a single integer. `d` is then a 1-by-N vector of date numbers.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

```
d = fbusdate(1997, 11)
d =
      729697
datestr(d)
ans =
03-Nov-1997

y = [1997 1998 1999];
d = fbusdate(y, 11); datestr(d)
ans =
03-Nov-1997
02-Nov-1998
01-Nov-1999
```

See Also

`busdate`, `eomdate`, `holidays`, `isbusday`, `lbusdate`

frac2cur

Purpose

Fractional currency value to decimal value.

Syntax

```
d = frac2cur(p, f)
```

Description

`d = frac2cur(p, f)` converts a fractional currency value to a decimal value. `p` is the fractional currency value input as a string, and `f` is the denominator of the fraction.

Example

```
d = frac2cur('12.1', 8)
```

returns

```
d =
  12.1250
```

See Also

`cur2frac`, `cur2str`

Purpose

Mean-variance efficient frontier.

Syntax

```
[PortRisk, PortReturn, PortWts] = frontcon(ExpReturn,
  ExpCovariance, NumPorts, PortReturn, AssetBounds, Groups,
  GroupBounds)
```

Arguments

`ExpReturn`	1-by-number of assets (NASSETS) vector specifying the expected (mean) return of each asset.
`ExpCovariance`	NASSETS-by-NASSETS matrix specifying the covariance of asset returns.
`NumPorts`	Number of portfolios generated along the efficient frontier. Returns are equally spaced between the maximum possible return and the minimum risk point. If `NumPorts` is empty (entered as `[]`), computes 10 equally spaced points. When entering a target rate of return (`PortReturn`), enter `NumPorts` as an empty matrix `[]`.
`PortReturn`	Vector of length equal to the number of portfolios (NPORTS) containing the target return values on the frontier. If `PortReturn` is not entered or `[]`, `NumPorts` equally spaced returns between the minimum and maximum possible values are used.
`AssetBounds`	2-by-NASSETS matrix containing the lower and upper bounds on the weight allocated to each asset in the portfolio. Default lower bound = all 0s (no short-selling). Default upper bound = all 1s (any asset may constitute the entire portfolio).

Groups	Number of groups (NGROUPS)-by-NASSETS matrix specifying NGROUPS asset groups or classes. Each row specifies a group. Groups(i,j) = 1 (j'th asset belongs in the i'th group). Groups(i,j) = 0 (j'th asset not a member of the i'th group).
GroupBounds	NGROUPS-by-2 matrix specifying, for each group, the lower and upper bounds of the total weights of all assets in that group. Default lower bound = all 0s. Default upper bound = all 1s.

Description

`[PortRisk, PortReturn, PortWts] = frontcon(ExpReturn, ExpCovariance, NumPorts, PortReturn, AssetBounds, Groups, GroupBounds)` returns the mean-variance efficient frontier with user-specified asset constraints, covariance, and returns. For a collection of NASSETS risky assets, computes a portfolio of asset investment weights that minimize the risk for given values of the expected return. The portfolio risk is minimized subject to constraints on the asset weights or on groups of asset weights.

PortRisk is an NPORTS-by-1 vector of the standard deviation of each portfolio.

PortReturn is a NPORTS-by-1 vector of the expected return of each portfolio.

PortWts is an NPORTS-by-NASSETS matrix of weights allocated to each asset. Each row represents a portfolio. The total of all weights in a portfolio is 1.

frontcon generates a plot of the efficient frontier if you invoke it without output arguments.

The asset returns are assumed to be jointly normal, with expected mean returns of ExpReturn and return covariance ExpCovariance. The variance of a portfolio with 1-by-NASSET weights PortWts is given by `PortVar = PortWts*ExpCovariance*PortWts'`. The portfolio expected return is `PortReturn = dot(ExpReturn, PortWts)`.

Examples

Given three assets with expected returns of

```
ExpReturn = [0.1 0.2 0.15]
```

and expected covariance of

```
ExpCovariance = [
   0.0100   -0.0061    0.0042
  -0.0061    0.0400   -0.0252
   0.0042   -0.0252    0.0225 ]
```

compute the mean-variance efficient frontier for four points (`NumPorts = 4`):

```
[PortRisk,PortReturn,PortWts] = frontcon...
(ExpReturn,ExpCovariance,NumPorts)

PortRisk =

    0.0426
    0.0483
    0.1089
    0.2000

PortReturn =

    0.1569
    0.1713
    0.1856
    0.2000

PortWts =

    0.2134    0.3518    0.4348
    0.0096    0.4352    0.5552
         0    0.7128    0.2872
         0    1.0000    0.0000
```

See Also

ewstats, portopt, portstats

fvdisc

Purpose

Future value of discounted security.

Syntax

```
rv = fvdisc(sd, md, price, disc, basis)
rv = fvdisc(sd, md, price, disc)
```

Arguments

`sd` Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.

`md` Maturity date. Enter as serial date number or date string.

`price` Price (present value) of the security.

`disc` Discount rate of the security. Enter as decimal fraction.

`basis` Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`rv = fvdisc(sd, md, price, disc, basis)` finds the amount received at maturity `rv` for a fully vested security.

Example

Using this data:

```
sd = '02/15/1991';
md = '05/15/1991';
price = 100;
disc = 0.0575;
basis = 2;

rv = fvdisc(sd, md, price, disc, basis)
```

returns

```
rv =
  101.44
```

See Also

`acrudisc`, `prdisc`, `ylddisc`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition.

Purpose

Future value with fixed periodic payments.

Syntax

```
f = fvfix(rate, nper, p, pv, due)
f = fvfix(rate, nper, p, pv)
f = fvfix(rate, nper, p)
```

Arguments

`rate` Periodic interest rate, as a decimal fraction.

`nper` Number of periods.

`p` Periodic payment.

`pv` Initial value. Default = `0`.

`due` When payments are due or made: `0` = end of period (default), or `1` = beginning of period.

Description

`f = fvfix(rate, nper, p, pv, due)` returns the future value of a series of equal payments.

Example

A savings account has a starting balance of $1500. $200 is added at the end of each month for 10 years and the account pays 9% interest compounded monthly. Using this data,

```
f = fvfix(0.09/12, 12*10, 200, 1500, 0)
```

returns

```
f =
      42379.89
```

See Also

`fvvar`, `pvfix`, `pvvar`

fvvar

Purpose

Future value of varying cash flow.

Syntax

```
fv = fvvar(cf, rate, df)
fv = fvvar(cf, rate)
```

Arguments

`cf` A vector of varying cash flows. Include the initial investment as the initial cash flow value (a negative number).

`rate` Periodic interest rate. Enter as a decimal fraction.

`df` For irregular (nonperiodic) cash flows, a vector of dates on which the cash flows occur. Enter dates as serial date numbers or date strings. Default assumes `cf` contains regular (periodic) cash flows.

Description

`fv = fvvar(cf, rate, df)` returns the future value `fv` of a varying cash flow.

Examples

This cash flow represents the yearly income from an initial investment of $10,000. The annual interest rate is 8%.

Year 1	$2000
Year 2	$1500
Year 3	$3000
Year 4	$3800
Year 5	$5000

For the future value of this regular (periodic) cash flow:

```
fv = fvvar([-10000 2000 1500 3000 3800 5000], 0.08)
```

returns

```
fv =
       2520.47
```

An investment of $10,000 returns this irregular cash flow. The original investment and its date are included. The periodic interest rate is 9%.

Cash flow	Dates
($10000)	January 12, 1987
$2500	February 14, 1988
$2000	March 3, 1988
$3000	June 14, 1988
$4000	December 1, 1988

To calculate the future value of this irregular (nonperiodic) cash flow:

```
cf = [-10000, 2500, 2000, 3000, 4000];
df = ['01/12/1987'
      '02/14/1988'
      '03/03/1988'
      '06/14/1988'
      '12/01/1988'];

fv = fvvar(cf, 0.09, df)
```

returns

```
fv =
        167.28
```

See Also

`fvfix`, `irr`, `payuni`, `pvfix`, `pvvar`

fwd2zero

Purpose

Zero curve given a forward curve.

Syntax

```
[zr, cd] = fwd2zero(fr, cd, sd, ocomp, obasis, icomp, ibasis)
[zr, cd] = fwd2zero(fr, cd, sd, ocomp, obasis, icomp)
[zr, cd] = fwd2zero(fr, cd, sd, ocomp, obasis)
[zr, cd] = fwd2zero(fr, cd, sd, ocomp)
[zr, cd] = fwd2zero(fr, cd, sd)
```

Arguments

`fr` Forward rates. An N-by-1 vector of annualized implied forward rates, as decimal fractions. In aggregate, the rates in `fr` constitute an implied forward curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the forward rates in `fr`. Use `datenum` to convert date strings to serial date numbers.

`sd` Settlement date. A serial date number that is the common settlement date for the forward rates in `fr`.

`ocomp` Output compounding. A scalar that sets the compounding frequency per year for annualizing the output zero rates in `zr`. Allowed values are:

- `1` = annual compounding
- `2` = semi-annual compounding (default)
- `3` = compounding three times per year
- `4` = quarterly compounding
- `6` = bimonthly compounding
- `12` = monthly compounding
- `365` = daily compounding
- `–1` = continuous compounding

`obasis` Output day-count basis for annualizing the output zero rates in `zr`. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

`icomp` Input compounding. A scalar that indicates the compounding frequency per year used for annualizing the input forward rates in `fr`. Allowed values are the same as for `ocomp`. Default = `ocomp`.

`ibasis` Input day-count basis used for annualizing the forward rates in `fr`. Allowed values are the same as for `obasis`. Default = `obasis`.

Description

`[zr, cd] = fwd2zero(fr, cd, sd, ocomp, obasis, icomp, ibasis)` returns a zero curve given an implied forward curve and its maturity dates.

`zr` Zero rates. An N-by-1 vector of decimal fractions. In aggregate, the rates in `zr` constitute a zero curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the zero rates in `zr`. This vector is the same as the input vector `cd`. Use `datestr` to convert serial date numbers to date strings.

Example

Given an implied forward curve `fr` over a set of maturity dates `cd`, and a settlement date `sd`:

```
fr = [0.0469
      0.0519
      0.0549
      0.0535
      0.0558
      0.0508
      0.0560
      0.0545
      0.0615
      0.0486];

cd = [datenum('06-Nov-1997')
      datenum('11-Dec-1997')
      datenum('15-Jan-1998')
      datenum('05-Feb-1998')
      datenum('04-Mar-1998')
      datenum('02-Apr-1998')
      datenum('30-Apr-1998')
      datenum('25-Jun-1998')
      datenum('04-Sep-1998')
      datenum('12-Nov-1998')];

sd = datenum('03-Nov-1997');
```

Set daily compounding for the zero curve, on an actual/365 basis. The forward curve was compounded annually on an actual/actual basis.

```
ocomp = 365;
obasis = 3;
icomp = 1;
ibasis = 0;
```

Execute the function

```
[zr, cd] = fwd2zero(fr, cd, sd, ocomp, obasis, icomp, ibasis)
```

which returns the zero curve `zr` at the maturity dates `cd`:

```
zr =
    0.0458
    0.0502
    0.0518
    0.0518
    0.0524
    0.0518
    0.0523
    0.0525
    0.0541
    0.0529
cd =
       729700
       729735
       729770
       729791
       729818
       729847
       729875
       729931
       730002
       730071
```

(For readability, `fr` and `zr` are shown here only to the basis point. However, MATLAB computed them at full precision. If you enter `fr` as shown, `zr` may differ due to rounding.)

See Also

`zero2fwd` and other functions for Term Structure of Interest Rates

Purpose

High, low, open, close chart.

Syntax

```
highlow(hi, lo, cl, op, color)
highlow(hi, lo, cl, op)
highlow(hi, lo, cl)
highlow(hi, lo, cl, [], color)
h = highlow(hi, lo, cl, op, color)
```

Arguments

`hi` High prices for a security. An N-by-1 vector.

`lo` Low prices for a security. An N-by-1 vector.

`cl` Closing prices for a security. An N-by-1 vector.

`op` Opening prices for a security. An N-by-1 vector. Optional. To specify `color` when `op` is unknown, enter `op` as an empty matrix `[]`.

`color` Vertical line color. A string. MATLAB supplies a default color if none is specified. The default color differs depending on the background color of the figure window. See "ColorSpec" in the online MATLAB Help Desk for `color` names.

Description

`highlow(hi, lo, cl, op, color)` plots the high, low, opening, and closing prices of an asset. Plots are vertical lines whose top is the high, bottom is the low, open is a short horizontal tick to the left, and close is a short horizontal tick to the right.

`h = highlow(hi, lo, cl, op, color)` plots the figure and returns the handles `h` of the lines. See *Using MATLAB Graphics* for information on graphics and object handles.

Example

The high, low, and closing prices for an asset are stored in equal-length vectors `assethi`, `assetlo`, and `assetcl` respectively

```
highlow(assethi, assetlo, assetcl, [], 'cyan')
```

plots the price data using cyan lines.

See Also

`bolling`, `candle`, `dateaxis`, `movavg`, `pointfig`

holidays

Purpose

Holidays and non-trading days.

Syntax

```
h = holidays(d1, d2)
h = holidays
```

Description

`h = holidays(d1, d2)` returns a vector `h` of serial date numbers corresponding to the holidays and non-trading days between dates `d1` and `d2`, inclusive. Enter `d1` and `d2` as serial date numbers or date strings.

`h = holidays` returns a vector `h` of serial date numbers corresponding to all holidays and non-trading days.

As shipped, this function contains all holidays and special non-trading days for the New York Stock Exchange between 1950 and 2030, inclusive (681 dates). You can edit the `holidays.m` file to contain your own holidays and non-trading days. By definition, holidays and non-trading days are those that occur on weekdays.

Example

```
h = holidays('jan 1 1997', 'jun 23 1997') returns
h =
      729391
      729438
      729477
      729536
```

which are the serial date numbers for

```
01-Jan-1997   New Year's Day
17-Feb-1997   Washington's Birthday
28-Mar-1997   Good Friday
26-May-1997   Memorial Day
```

See Also

`busdate`, `fbusdate`, `isbusday`, `lbusdate`

Purpose

Hour of date or time.

Syntax

```
h = hour(d)
```

Description

`h = hour(d)` returns the hour of the day given a serial date number or a date string `d`.

Example

```
h = hour(728647.5590548427)
```

or

```
h = hour('19-dec-1994, 13:24:08.17')
```

returns

```
h =
    13
```

See Also

`datevec`, `minute`, `second`

irr

Purpose

Internal rate of return.

Syntax

```
r = irr(cf)
```

Description

`r = irr(cf)` calculates the internal rate of return for a series of periodic cash flows. `cf` is the cash flow vector. The first entry in `cf` is the initial investment. If the cash flow payments are monthly, multiply the resulting rate of return by 12 for the annual rate of return. This function calculates only positive rates of return; for negative rates of return, `r = NaN`.

Example

This cash flow represents the yearly income from an initial investment of $100,000:

Year 1	$10,000
Year 2	$20,000
Year 3	$30,000
Year 4	$40,000
Year 5	$50,000

To calculate the internal rate of return on the investment:

```
r = irr([-100000  10000  20000  30000  40000  50000])
```

returns

```
r =
    0.1201 (12.01%)
```

See Also

`effrr`, `mirr`, `nomrr`, `taxedrr`, `xirr`

Reference

Brealey and Myers, *Principles of Corporate Finance*, Chapter 5

Purpose

True for dates that are business days.

Syntax

```
t = isbusday(d, hol)
t = isbusday(d)
```

Arguments

`d` Date(s) being checked. Enter as a serial date number or date string. `d` can contain multiple dates, but they must all be in the same format.

`hol` Vector of holidays and non-trading-day dates. All dates in `hol` must be the same format: either serial date numbers or date strings. (Using date numbers improves performance.) The `holidays` function supplies the default vector.

Description

`t = isbusday(d, hol)` returns logical true (1) if `d` is a business day and logical false (0) otherwise.

Examples

```
t = isbusday('15 jun 1997')
t =
     0

d = ['15 feb 98'; '16 feb 98'; '17 feb 98'];
t = isbusday(d)
t =
     0
     0
     1
```

See Also

`busdate`, `fbusdate`, `holidays`, `lbusdate`

lbusdate

Purpose

Last business date of month.

Syntax

```
d = lbusdate(y, m, hol)
d = lbusdate(y, m)
```

Arguments

`y` Year. Enter as four-digit integer.

`m` Month. Enter as integer from 1 to 12.

`hol` Vector of holidays and non-trading-day dates. All dates in `hol` must be the same format: either serial date numbers or date strings. (Using date numbers improves performance.) The `holidays` function supplies the default vector.

Description

`d = lbusdate(y, m, hol)` returns the serial date number for the last business date of the given year `y` and month `m` where `hol` specifies non-trading days.

`y` and `m` can contain multiple values. If one contains multiple values, the other must contain the same number of values or a single value that applies to all. For example, if `y` is a 1-by-N vector of integers, them `m` must be a 1-by-N vector of integers or a single integer. `d` is then a 1-by-N vector of date numbers.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

```
d = lbusdate(1997, 5)
d =
      729540
datestr(d)
ans =
30-May-1997

y = [1997 1998 1999];
d = lbusdate(y, 5); datestr(d)
ans =
30-May-1997
29-May-1998
28-May-1999
```

See Also

`busdate`, `eomdate`, `fbusdate`, `holidays`, `isbusday`

Purpose

Date of last occurrence of weekday in month.

Syntax

```
d = lweekdate(wkd, y, m, g)
d = lweekdate(wkd, y, m)
```

Arguments

`wkd` Weekday whose date you seek. Enter as an integer from 1 through 7:

1 Sunday
2 Monday
3 Tuesday
4 Wednesday
5 Thursday
6 Friday
7 Saturday

`y` Year. Enter as a four-digit integer.

`m` Month. Enter as an integer from 1 through 12.

`g` Weekday that must occur after `wkd` in the same week. Enter as an integer from 0 through 7, where 0 = ignore (default) and 1 through 7 are as for `wkd`.

Description

`d = lweekdate(wkd, y, m, g)` returns the serial date number for the last occurrence of the weekday `wkd` in the given year `y` and month `m`, and in a week that also contains the weekday `g`.

Any input can contain multiple values, but if so, all other inputs must contain the same number of values or a single value that applies to all. For example, if `y` is a 1-by-N vector of integers, then `m` must be a 1-by-N vector of integers or a single integer. `d` is then a 1-by-N vector of date numbers.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

To find the last Monday in June 1997:

```
d = lweekdate(2, 1997, 6); datestr(d)
ans =
30-Jun-1997
```

To find the last Monday in a week that also contains a Friday in June 1997:

```
d = lweekdate(2, 1997, 6, 6); datestr(d)
ans =
23-Jun-1997
```

To find the last Monday in May for 1997, 1998, and 1999:

```
y = [1997:1999];
d = lweekdate(2, y, 5)
d =
      729536      729900      730271
datestr(d)
ans =
26-May-1997
25-May-1998
31-May-1999
```

See Also

eomdate, lbusdate, nweekdate

Purpose

MATLAB serial date number to Excel serial date number.

Syntax

```
xd = m2xdate(d, dsys)
xd = m2xdate(d)
```

Arguments

`d` MATLAB serial date number. A vector or scalar.

`dsys` Excel date system. A vector or scalar.
`0` = 1900 date system (default), in which 1 = January 1, 1900 A.D.
`1` = 1904 date system, in which 0 = January 1, 1904 A.D.

Vector arguments must have consistent dimensions.

Description

`xd = m2xdate(d, dsys)` converts MATLAB serial date numbers `d` to Excel serial date numbers `xd`. MATLAB date numbers start with 1 = January 1, 0000 A.D., hence there is a difference of 693960 relative to the 1900 date system, or 695422 relative to the 1904 date system. This function is useful with MATLAB Excel Link.

Example

Given MATLAB date numbers for Christmas 1997 through 2000:

```
d = datenum(1997:2000, 12, 25)
d =
      729749       730114       730479       730845
```

convert them to Excel date numbers in the 1904 system:

```
xd = m2xdate(d, 1)
xd =
       34327        34692        35057        35423
```

or the 1900 system:

```
xd = m2xdate(d)
xd =
       35789        36154        36519        36885
```

See Also

`datenum`, `datestr`, `x2mdate`

minute

Purpose

Minute of date or time.

Syntax

```
m = minute(d)
```

Description

`m = minute(d)` returns the minute given a serial date number or a date string `d`.

Example

```
m = minute(728647.559054842)
```

or

```
m = minute('19-dec-1994, 13:25:08.17')
```

returns

```
m =
   25
```

See Also

`datevec`, `hour`, `second`

Purpose

Modified internal rate of return.

Syntax

```
r = mirr(cf, srate, rrate)
```

Arguments

`cf` Vector of cash flows. The first entry in `cf` is the initial investment.

`srate` Finance rate for negative cash flow values. Enter as decimal fraction.

`rrate` Reinvestment rate for positive cash flow values, as a decimal fraction.

Description

`r = mirr(cf, srate, rrate)` calculates the modified internal rate of return for a series of periodic cash flows. This function calculates only positive rates of return; for negative rates of return, `r = 0`.

Example

This cash flow represents the yearly income from an initial investment of $100,000. The finance rate is 9% and the reinvestment rate is 12%.

Year 1	$20,000
Year 2	($10,000)
Year 3	$30,000
Year 4	$38,000
Year 5	$50,000

To calculate the modified internal rate of return on the investment:

```
r = mirr([−100000 20000 −10000 30000 38000 50000], 0.09, 0.12)
```

returns

```
r =
    0.0832 (8.32%)
```

See Also

`annurate, effrr, irr, nomrr, pvvar, xirr`

Reference

Brealey and Myers, *Principles of Corporate Finance*, Chapter 5

month

Purpose

Month of date.

Syntax

```
[n, m] = month(d)
```

Description

`[n, m] = month(d)` returns the month in numeric `n` and string form `m` given a serial date number or a date string `d`.

Example

```
[n, m] = month(728647)
```

or

```
[n, m] = month('19-dec-1994')
```

returns

```
n =
    12
m =
Dec
```

See Also

`datevec`, `day`, `year`

Purpose

Number of whole months between dates.

Syntax

```
mm = months(d1, d2, eom)
mm = months(d1, d2)
```

Description

`mm = months(d1, d2, eom)` returns the number of whole months between two dates `d1` and `d2`. If `d2` is earlier than `d1`, `mm` is negative. Enter dates as serial date numbers or date strings.

`eom` is the end-of-month flag. If `d1` and `d2` are end-of-month dates and `d2` has fewer days than `d1`, `eom = 1` (default) treats `d2` as the end of a whole month, while `eom = 0` does not.

Any input argument can contain multiple values, but if so, all other inputs must contain the same number of values or a single value that applies to all. For example, if `d1` is an N-row character array of date strings, then `d2` must be an N-row character array of date strings or a single date. `mm` is then an N-by-1 vector of numbers.

Examples

```
mm = months('may 31 1997', 'jun 30 1997', 1)
mm =
     1

mm = months('may 31 1997','jun 30 1997', 0)
mm =
     0

d1 = ['mar 31 1997'; 'apr 30 1997'; 'may 31 1997'];
mm = months(d1, 'jun 30 1997')
mm =
     3
     2
     1
```

See Also

`yearfrac`

movavg

Purpose

Leading and lagging moving averages chart.

Syntax

```
movavg(asset, lead, lag, alpha)
movavg(asset, lead, lag)
[short, long] = movavg(asset, lead, lag, alpha)
```

Arguments

`asset` Security data, usually a vector of time-series prices.

`lead` Number of samples to use in leading average calculation. A positive integer. `lead` must be less than or equal to `lag`.

`lag` Number of samples to use in the lagging average calculation. A positive integer.

`alpha` Control parameter that determines the type of moving averages. `0` = simple moving average (default), `0.5` = square root weighted moving average, `1` = linear moving average, `2` = square weighted moving average, etc. To calculate the exponential moving average, set `alpha ='e'`.

Description

`movavg(asset, lead, lag, alpha)` plots leading and lagging moving averages.

`[short, long] = movavg(asset, lead, lag, alpha)` returns the leading `short` and lagging `long` moving average data without plotting it.

Example

If `asset` is a vector of stock price data:

```
movavg(asset, 3, 20, 1)
```

plots linear three-sample leading and 20-sample lagging moving averages.

See Also

`bolling, candle, dateaxis, highlow, pointfig`

Purpose

Nominal rate of return.

Syntax

```
apr = nomrr(er, per)
```

Arguments

`er` Effective annual percentage rate. Enter as a decimal fraction.

`per` Number of compounding periods per year, an integer.

Description

`apr = nomrr(er, per)` calculates the nominal rate of return.

Example

To find the nominal annual rate of return based on an effective annual percentage rate of 9.38% compounded monthly:

```
apr = nomrr(0.0938, 12)
```

returns

```
apr =
    0.0900 (9.0%)
```

See Also

`effrr, irr, mirr, taxedrr, xirr`

now

Purpose

Current date and time.

Syntax

```
t = now
```

Description

`t = now` returns the current date and time as a serial date number.

Note: This function now ships with basic MATLAB. It originally shipped only with the Financial Toolbox, and this description remains here for your convenience.

Example

```
t = now
```

```
t =
    729647.5833
```

(on September 14, 1997 at 2:00 PM)

See Also

`date`, `datenum`, `today`

Purpose

Date of specific occurrence of weekday in month.

Syntax

```
d = nweekdate(n, wkd, y, m, g)
d = nweekdate(n, wkd, y, m)
```

Arguments

`n` Nth occurrence of the weekday in a month. Enter as an integer from 1 through 5.

`wkd` Weekday whose date you seek. Enter as an integer from 1 through 7:

- 1 Sunday
- 2 Monday
- 3 Tuesday
- 4 Wednesday
- 5 Thursday
- 6 Friday
- 7 Saturday

`y` Year. Enter as a four-digit integer.

`m` Month. Enter as an integer from 1 through 12.

`g` Weekday that must occur in the same week with `wkd`. Enter as an integer from 0 through 7, where 0 = ignore (default) and 1 through 7 are as for `wkd`.

Description

`d = nweekdate(n, wkd, y, m, g)` returns the serial date number for the specific occurrence of the weekday `wkd` in the given year `y` and month `m`, and in a week that also contains the weekday `g`.

If `n` is larger than the last occurrence of `wkd`, `d = 0`.

Any input can contain multiple values, but if so, all other inputs must contain the same number of values or a single value that applies to all. For example, if `y` is a 1-by-N vector of integers, then `m` must be a 1-by-N vector of integers or a single integer. `d` is then a 1-by-N vector of date numbers.

Use the function `datestr` to convert serial date numbers to formatted date strings.

Examples

To find the first Thursday in May 1997:

```
d = nweekdate(1, 5, 1997, 5); datestr(d)
ans =
01-May-1997
```

To find the first Thursday in a week that also contains a Wednesday in May 1997:

```
d = nweekdate(1, 5, 1997, 5, 4); datestr(d)
ans =
08-May-1997
```

To find the third Monday in February for 1997, 1998, and 1999:

```
y = [1997:1999];
d = nweekdate(3, 2, y, 2)
d =
      729438      729802      730166
datestr(d)
ans =
17-Feb-1997
16-Feb-1998
15-Feb-1999
```

See Also

`fbusdate`, `lbusdate`, `lweekdate`

Purpose

Option profit.

Syntax

```
p = opprofit(so, x, cost, flag, type)
```

Arguments

`so` Asset price.

`x` Strike or exercise price.

`cost` Cost of the option.

`flag` Option position. `0` = long, `1` = short.

`type` Option type. `0` = call option, `1` = put option.

Description

`p = opprofit(so, x, cost, flag, type)` returns the profit of an option.

Example

Buying (going long on) a call option with a strike price of $90 on an underlying asset with a current price of $100 for a cost of $4:

```
p = opprofit(100, 90, 4, 0, 0)
```

returns

```
p =
            6.00
```

a profit of $6 if the option is exercised under these conditions.

See Also

`binprice, blsprice`

payadv

Purpose

Periodic payment given number of advance payments.

Syntax

```
pmt = payadv(rate, nper, pv, fv, adv)
```

Arguments

`rate` Lending or borrowing rate per period. Enter as a decimal fraction. Must be greater than or equal to 0.

`nper` Number of periods in the life of the instrument.

`pv` Present value of the instrument.

`fv` Future value or target value to be attained after `nper` periods.

`adv` Number of advance payments. If the payments are made at the beginning of the period, add `1` to `adv`.

Description

`pmt = payadv(rate, nper, pv, fv, adv)` returns the periodic payment given a number of advance payments.

Example

The present value of a loan is $1000.00 and it will be paid in full in 12 months. The annual interest rate is 10% and three payments are made at closing time. Using this data:

```
pmt = payadv(0.1/12, 12, 1000, 0, 3)
```

returns

```
pmt =
          85.94
```

for the periodic payment.

See Also

`amortize, payodd, payper`

Purpose

Payment of loan or annuity with odd first period.

Syntax

```
pmt = payodd(rate, nper, pv, fv, dys)
```

Arguments

`rate` Interest rate per period. Enter as a decimal fraction.

`nper` Number of periods in the life of the instrument.

`pv` Present value of the instrument.

`fv` Future value or target value to be attained after `nper` periods.

`dys` Actual number of days until the first payment is made.

Description

`pmt = payodd(rate, nper, pv, fv, dys)` returns the payment for a loan or annuity with an odd first period.

Example

A two-year loan for $4000 has an annual interest rate of 11%. The first payment will be made in 36 days. To find the monthly payment:

```
pmt = payodd(0.11/12, 24, 4000, 0, 36)
```

returns

```
pmt =
        186.77
```

See Also

`amortize`, `payadv`, `payper`

payper

Purpose

Periodic payment of loan or annuity.

Syntax

```
pmt = payper(rate, nper, pv, fv, due)
pmt = payper(rate, nper, pv, fv)
pmt = payper(rate, nper, pv)
```

Arguments

`rate` Interest rate per period. Enter as a decimal fraction.

`nper` Number of payment periods in the life of the instrument.

`pv` Present value of the instrument.

`fv` Future value or remaining balance after `nper` periods. Default = `0`.

`due` When payments are due: `0` = end of period (default), or `1` = beginning of period.

Description

`pmt = payper(rate, nper, pv, fv, due)` returns the periodic payment of a loan or annuity.

Example

Find the monthly payment for a three-year loan of $9000 with an annual interest rate of 11.75%:

```
pmt = payper(0.1175/12, 36, 9000, 0, 0)
```

returns

```
pmt =
        297.86
```

See Also

`amortize, fvfix, payadv, payodd, pvfix`

Purpose

Uniform payment equal to varying cash flow.

Syntax

```
us = payuni(cf, rate)
```

Arguments

`cf` A vector of varying cash flows. Include the initial investment as the initial cash flow value (a negative number).

`rate` Periodic interest rate. Enter as a decimal fraction.

Description

`us = payuni(cf, rate)` returns the uniform series value us of a varying cash flow.

Example

This cash flow represents the yearly income from an initial investment of $10,000. The annual interest rate is 8%.

Year 1	$2000
Year 2	$1500
Year 3	$3000
Year 4	$3800
Year 5	$5000

To calculate the uniform series value:

```
us = payuni([-10000 2000 1500 3000 3800 5000], 0.08)
```

returns

```
us =
        429.63
```

See Also

`fvfix, fvvar, irr, pvfix, pvvar`

pcalims

Purpose

Linear inequalities for individual asset allocation.

Syntax

```
[A,b] = pcalims(AssetMin, AssetMax, NumAssets)
```

Arguments

`AssetMin`	Scalar or `NASSETS`-long vector of minimum allocations in each asset. `NaN` indicates no constraint.
`AssetMax`	Scalar or `NASSETS`-long vector of maximum allocations in each asset. `NaN` indicates no constraint
`NumAssets`	Optional. Number of assets. Default = length of `AssetMin` or `AssetMax`.

Description

`[A,b] = pcalims(AssetMin, AssetMax, NumAssets)` specifies the lower and upper bounds of portfolio allocations in each of `NumAssets` available asset investments.

`A` is a matrix and `b` a vector such that `A*PortWts' <= b`, where `PortWts` is a 1-by-`NASSETS` vector of asset allocations.

If `pcalims` is called with fewer than two output arguments, the function returns `A` concatenated with `b` `[A,b]`.

Example

Set the minimum weight in every asset to 0 (no short-selling), and set the maximum weight of IBM to 0.5 and CSCO to 0.8, while letting the maximum weight in INTC float.

Asset	IBM	INTC	CSCO
Min. Wt.	0	0	0
Max. Wt.	0.5		0.8

```
AssetMin = 0
AssetMax = [0.5 NaN 0.8]
[A,b] = pcalims(AssetMin, AssetMax)

A =

     1     0     0
     0     0     1
    -1     0     0
     0    -1     0
     0     0    -1

b =

    0.5000
    0.8000
         0
         0
         0
```

Portfolio weights of 50% in IBM and 50% in INTC satisfy the constraints.

Set the minimum weight in every asset to 0 and the maximum weight to 1.

Asset	IBM	INTC	CSCO
Min. Wt.	0	0	0
Max. Wt.	1	1	1

```
AssetMin = 0
AssetMax = 1
NumAssets = 3

[A,b] = pcalims(AssetMin, AssetMax, NumAssets)

A =

     1     0     0
     0     1     0
     0     0     1
    -1     0     0
     0    -1     0
     0     0    -1

b =

     1
     1
     1
     0
     0
     0
```

Portfolio weights of 50% in IBM and 50% in INTC satisfy the constraints.

See Also

pcgcomp, pcglims, pcpval, portcons, portopt

Purpose

Linear inequalities for asset group comparison constraints.

Syntax

```
[A,b] = pcgcomp(GroupA, AtoBmin, AtoBmax, GroupB)
```

Arguments

`GroupA` `GroupB`	Number of groups (`NGROUPS`) by number of assets (`NASSETS`) specifications of groups to compare. Each row specifies which assets are in a group: `Group(i,j) = 1` if group `i` contains asset `j`; otherwise, `Group(i,j) = 0`.

`AtoBmin`
`AtoBmax`
	Scalar or `NGROUPS`-long vectors of minimum and maximum ratios of allocations in group `A` to allocations in group `B`. `NaN` indicates no constraint between the two groups. Scalar bounds are applied to all group pairs.
	For each of the group pairs:
	`GroupA` total `>= GroupB` total `* AtoBmin`
	`GroupA` total `<= GroupB` total `* AtoBmax`

Description

`[A,b] = pcgcomp(GroupA, AtoBmin, AtoBmax, GroupB)` Specifies that the ratio of allocations in one group to allocations in another group is at least `AtoBmin` to 1 and at most `AtoBmax` to 1. Comparisons can be made between an arbitrary number of group pairs `NGROUPS` comprising subsets of `NASSETS` available investments.

`A` is a matrix and `b` a vector such that `A*PortWts' <= b`, where `PortWts` is a 1-by-`NASSETS` vector of asset allocations.

If `pcgcomp` is called with fewer than two output arguments, the function returns `A` concatenated with `b` `[A,b]`.

Example

Asset	INTC	XON	RD
Region	North America	North America	Europe
Sector	Technology	Energy	Energy

Group	Min. Exposure	Max. Exposure
North America	0.30	0.75
Europe	0.10	0.55
Technology	0.20	0.50
Energy	0.20	0.80

Make the North American energy sector compose exactly 20% of the North American investment.

```
            INTC  XON  RD
GroupA = [   0    1   0  ];  % North American Energy
GroupB = [   1    1   0  ];  % North America

AtoBmin = 0.20;
AtoBmax = 0.20;

[A,b] = pcgcomp(GroupA, AtoBmin, AtoBmax, GroupB)
A =

    0.2000   -0.8000     0
   -0.2000    0.8000     0

b =

   0
   0
```

Portfolio weights of 40% for INTC, 10% for XON, and 50% for RD satisfy the constraints.

See Also

pcalims, pcglims, pcpval, portcons, portopt

Purpose

Linear inequalities for asset group minimum and maximum allocation

Syntax

```
[A,b] = pcglims(Groups, GroupMin, GroupMax)
```

Arguments

`Groups`	Number of groups (`NGROUPS`) by number of assets (`NASSETS`) specifications of which assets belong to which group. Each row specifies a group: `Group(i,j) = 1` if group `i` contains asset `j`; otherwise, `Group(i,j) = 0`.
`GroupMin` `GroupMax`	Scalar or `NGROUPS`-long vectors of minimum and maximum combined allocations in each group. `NaN` indicates no constraint. Scalar bounds are applied to all groups.

Description

`[A,b] = pcglims(Groups, GroupMin, GroupMax)` Specifies minimum and maximum allocations to groups of assets. An arbitrary number of groups, `NGROUPS`, comprising subsets of `NASSETS` investments, is allowed.

`A` is a matrix and `b` a vector such that `A*PortWts' <= b`, where `PortWts` is a 1-by-`NASSETS` vector of asset allocations.

If `pcglims` is called with fewer than two output arguments, the function returns `A` concatenated with `b` `[A,b]`.

Examples

Asset	INTC	XON	RD
Region	North America	North America	Europe
Sector	Technology	Energy	Energy

Group	Min. Exposure	Max. Exposure
North America	0.30	0.75
Europe	0.10	0.55
Technology	0.20	0.50
Energy	0.50	0.50

Set the minimum and maximum investment in various groups.

```
%           INTC  XON  RD
Groups = [   1    1   0  ;  % North America
             0    0   1  ;  % Europe
             1    0   0  ;  % Technology
             0    1   1  ]  % Energy

GroupMin = [0.30
            0.10
            0.20
            0.50]

GroupMax = [0.75
            0.55
            0.50
            0.50]
```

```
[A,b] = pcglims(Groups, GroupMin, GroupMax)

A =

    −1    −1     0
     0     0    −1
    -1     0     0
     0    -1    −1
     1     1     0
     0     0     1
     1     0     0
     0     1     1

b =

   −0.3000
   −0.1000
   −0.2000
   −0.5000
    0.7500
    0.5500
    0.5000
    0.5000
```

Portfolio weights of 50% in INTC, 25% in XON, and 25% in RD satisfy the constraints.

See Also

pcalims, pcgcomp, pcpval, portcons, portopt

pcpval

Purpose

Linear inequalities for fixing total portfolio value.

Syntax

```
[A,b] = pcpval(PortValue, NumAssets)
```

Arguments

`PortValue`	Scalar total value of asset portfolio (sum of the allocations in all assets). `PortValue = 1` specifies weights as fractions of the portfolio and return and risk numbers as rates instead of value.
`NumAssets`	Number of available asset investments.

Description

`[A,b] = pcpval(PortValue, NumAssets)` Scales the total value of a portfolio of `NumAssets` assets to `PortValue`. All portfolio weights, bounds, return, and risk values except `ExpReturn` and `ExpCovariance` (see `portopt`) are in terms of `PortValue`.

`A` is a matrix and `b` a vector such that `A*PortWts' <= b`, where `PortWts` is a 1-by-`NASSETS` vector of asset allocations.

If `pcpval` is called with fewer than two output arguments, the function returns `A` concatenated with `b` `[A,b]`.

Examples

Scale the value of a portfolio of three assets to 1, so all return values are rates and all weight values are in fractions of the portfolio.

```
PortValue = 1;
NumAssets = 3;

[A,b] = pcpval(PortValue, NumAssets)
A =

     1     1     1
    -1    -1    -1

b =

     1
    -1
```

Portfolio weights of 40%, 10%, and 50% in the three assets satisfy the constraints.

See Also pcalims, pcgcomp, pcglims, portcons, portopt

pointfig

Purpose

Point and figure chart.

Syntax

```
pointfig(asset)
```

Description

`pointfig(asset)` plots a point and figure chart for a vector of price data `asset`. Upward price movements are plotted as X's and downward price movements are plotted as O's.

See Also

`bolling, candle, dateaxis, highlow, movavg`

Purpose

Optimal capital allocation.

Syntax

```
[RiskyRisk, RiskyReturn, RiskyWts, RiskyFraction, OverallRisk,
  OverallReturn] = portalloc(PortRisk, PortReturn, PortWts,
  RisklessRate, BorrowRate, RiskAversion)
```

Arguments

`PortRisk`	Variance of each portfolio. A number of portfolios (`NPORTS`)-by-`1` vector.
`PortReturn`	Expected return of each portfolio. A number of portfolios (`NPORTS`)-by-`1` vector.
`PortWts`	Weights allocated to each asset. A number of portfolios (`NPORTS`)-by-number of assets (`NASSETS`) matrix of weights allocated to each asset. Each row represents a different portfolio. Total of all weights in a portfolio is 1.
`RisklessRate`	Risk-free rate. A decimal number.
`BorrowRate`	Borrowing rate. A decimal number. If borrowing is not desired, or not an option, set to `NaN` (default)
`RiskAversion`	Coefficient of investor's degree of risk aversion. Higher numbers indicate greater risk aversion. Typical coefficients range between `2.0` and `4.0` (`3` = default).

Description

`[RiskyRisk, RiskyReturn, RiskyWts, RiskyFraction, OverallRisk, OverallReturn] = portalloc(PortRisk, PortReturn, PortWts, RisklessRate, BorrowRate, RiskAversion)` computes the optimal risky portfolio, and the optimal allocation of funds between the risky portfolio and the risk-free asset.

`RiskyRisk` is the variance of the optimal risky portfolio.

`RiskyReturn` is the expected return of the optimal risky portfolio.

`RiskyWts` is a 1-by-number of assets (`NASSETS`) vector of weights allocated to the optimal risky portfolio. The total of all weights in the portfolio is 1.

`RiskyFraction` is the fraction of the complete portfolio allocated in the risky portfolio.

`OverallRisk` is the variance of the optimal overall portfolio.

`OverallReturn` is the expected rate of return of the optimal overall portfolio.

Example

Generate the efficient frontier from the asset data:

```
ExpReturn = [0.1 0.2 0.15];

ExpCovariance = [
  0.005   -0.010    0.004
 -0.010    0.040   -0.002
  0.004   -0.002    0.023 ];

[PortRisk, PortReturn, PortWts] = portopt(ExpReturn,...
ExpCovariance);
```

Find the optimal risky portfolio and allocate capital. The risk free investment return is 8%, and the borrowing rate is 12%.

```
RisklessRate  = 0.08
BorrowRate    = 0.12
RiskAversion  = 3

[RiskyRisk, RiskyReturn, RiskyWts, RiskyFraction, ...
OverallRisk, OverallReturn] = portalloc(PortRisk, PortReturn,...
PortWts, RisklessRate, BorrowRate, RiskAversion)
```

```
RiskyRisk =

    0.1283

RiskyReturn =

    0.1788

RiskyWts =

    0.0265    0.6023    0.3712

RiskyFraction =

    1.1898

OverallRisk =

    0.1527

OverallReturn =

    0.1899
```

See Also

frontcon, portrand, portror

Reference

Bodie, Kane, and Marcus, *Investments*, Chapters 6 and 7.

portcons

Purpose

Portfolio constraints.

Syntax

```
ConSet = portcons(varargin)
```

Description

Using linear inequalities, `portcons` generates a matrix of constraints for a portfolio of asset investments. The matrix `ConSet` is defined as `ConSet = [A b]`. `A` is a matrix and `b` a vector such that `A*PortWts' <= b` sets the value, where `PortWts` is a 1-by-number of assets (`NASSETS`) vector of asset allocations.

`ConSet = portcons('ConstType', Data1, ..., DataN)` creates a matrix `ConSet`, based on the constraint type `ConstType`, and the constraint parameters `Data1, ..., DataN`.

`ConSet = portcons('ConstType1', Data11, ..., Data1N,'ConstType2', Data21, ..., Data2N, ...)` creates a matrix `ConSet`, based on the constraint types `ConstTypeN`, and the corresponding constraint parameters `DataN1, ..., DataNN`.

Constraint Type	Description	Values
Default	All allocations are >= 0; no short selling allowed. Combined value of portfolio allocations normalized to 1.	`NumAssets` (required). Scalar representing number of assets in portfolio.
`PortValue`	Fix total value of portfolio to `PVal`.	`PVal` (required). Scalar representing total value of portfolio. `NumAssets` (required). Scalar representing number of assets in portfolio. See `pcpval`.

Constraint Type	Description	Values
AssetLims	Minimum and maximum allocation per asset.	AssetMin (required). Scalar or vector of length NASSETS, specifying minimum allocation per asset. AssetMax (required). Scalar or vector of length NASSETS, specifying maximum allocation per asset. NumAssets (optional). See pcalims.
GroupLims	Minimum and maximum allocations to asset group.	Groups (required). NGROUPS-by-NASSETS matrix specifying which assets belong to each group. GroupMin (required). Scalar or a vector of length NGROUPS, specifying minimum combined allocations in each group. GroupMax (required). Scalar or a vector of length NGROUPS, specifying maximum combined allocations in each group. See pcglims.

Constraint Type	Description	Values
`GroupComparison`	Group-to-group comparison constraints.	`GroupA` (required). `GroupB` (required). `NGROUPS`-by-`NASSETS` matrices specifying groups to compare. `AtoBmin` (required). Scalar or vector of length `NGROUPS` specifying minimum ratios of allocations in `GroupA` to allocations in `GroupB`. `AtoBmax` (required). Scalar or vector of length `NGROUPS` specifying maximum ratios of allocations in `GroupA` to allocations in `GroupB`. See `pcgcomp` .
`Custom`	Custom linear inequality constraints `A*PortWts' <= b`.	`A` (required). `NCONSTRAINTS` by `NASSETS` matrix, specifying weights for each asset in each inequality equation. `b` (required). Vector of length `NCONSTRAINTS` specifying the right hand sides of the inequalities.

Example

Constrain a portfolio of three assets:

Asset	IBM	CPQ	XON
Group	A	A	B
Min. Wt.	0	0	0
Max. Wt.	0.5	0.9	0.8

```
NumAssets = 3
PVal = 1 % Scale portfolio value to 1.
AssetMin = 0
AssetMax = [0.5 0.9 0.8]
GroupA = [1 1 0]
GroupB = [0 0 1]
AtoBmax = 1.5 % Value of assets in Group A at most 1.5 times value
             % in group B.

ConSet = portcons('PortValue', PVal, NumAssets,'AssetLims',...
AssetMin, AssetMax, NumAssets, 'GroupComparison',GroupA, NaN,...
AtoBmax, GroupB)

ConSet =

    1.0000    1.0000    1.0000    1.0000
   -1.0000   -1.0000   -1.0000   -1.0000
    1.0000         0         0    0.5000
         0    1.0000         0    0.9000
         0         0    1.0000    0.8000
   -1.0000         0         0         0
         0   -1.0000         0         0
         0         0   -1.0000         0
    1.0000    1.0000   -1.5000         0
```

Portfolio weights of 30% in IBM, 30% in CPQ, and 40% in XON satisfy the constraints.

See Also

`pcalims`, `pcgcomp`, `pcglims`, `pcpval`, `portopt`

portopt

Purpose

Portfolios on constrained efficient frontier.

Syntax

```
[PortRisk, PortReturn, PortWts] = portopt(ExpReturn, ExpCovariance,
   NumPorts, PortReturn, ConSet)
```

Arguments

`ExpReturn`	1-by-number of assets (NASSETS) vector specifying the expected (mean) return of each asset.
`ExpCovariance`	NASSETS-by-NASSETS matrix specifying the covariance of the asset returns.
`NumPorts`	Number of portfolios generated along the efficient frontier. Returns are equally spaced between the maximum possible return and the minimum risk point. If `NumPorts` is empty (entered as `[]`), computes 10 equally spaced points.
`PortReturn`	Expected return of each portfolio. A number of portfolios (NPORTS)-by-1 vector. If not entered or empty, `NumPorts` equally spaced returns between the minimum and maximum possible values are used.
`ConSet`	Constraint matrix for a portfolio of asset investments, created using `portcons`. If not specified, a default is created.

Description

`[PortRisk, PortReturn, PortWts] = portopt(ExpReturn, ExpCovariance, NumPorts, PortReturn, ConSet)` returns the mean-variance efficient frontier with user-specified covariance, returns, and asset constraints (`ConSet`). Given a collection of NASSETS risky assets, computes a portfolio of asset investment weights that minimize the risk for given values of the expected return. The portfolio risk is minimized subject to constraints on the total portfolio value, the individual asset minimum and maximum allocation, the asset group minimum and maximum allocation, or the asset group-to-group comparison.

`PortRisk` is a number of portfolios (NPORTS)-by-1 vector of the variance of each portfolio.

`PortReturn` is an NPORTS-by-1 vector of the expected return of each portfolio.

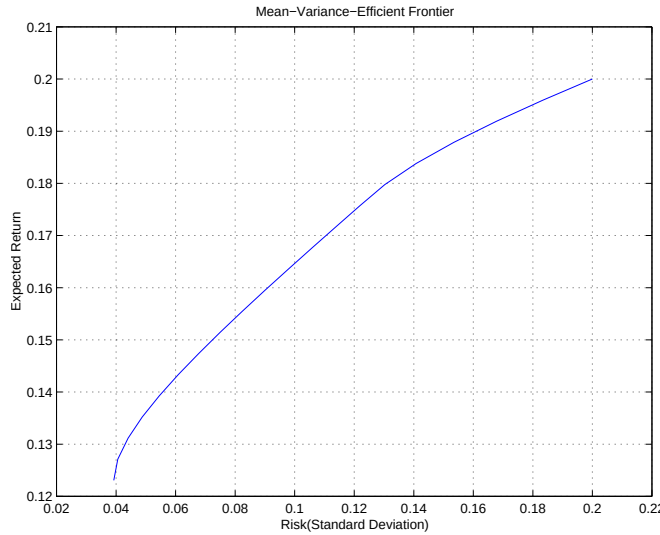
`PortWts` is an `NPORTS`-by-`NASSETS` matrix of weights allocated to each asset. Each row represents a portfolio. The total of all weights in a portfolio is 1.

If `portopt` is invoked without output arguments, it returns a plot of the efficient frontier.

Examples

Plot the risk-return efficient frontier of portfolios allocated among three assets. Connect 20 portfolios along the frontier having evenly spaced returns. By default, choose among portfolios without short-selling and scale the value of the portfolio to 1.

```
ExpReturn = [0.1 0.2 0.15];
ExpCovariance = [
  0.005   -0.010    0.004
 -0.010    0.040   -0.002
  0.004   -0.002    0.023 ];
NumPorts = 20;
portopt(ExpReturn, ExpCovariance, NumPorts)
```



Return the two efficient portfolios that have returns of 16% and 17%. Limit to portfolios that have at least 20% of the allocation in the first asset, and cap the total value in the first and third assets at 50% of the portfolio.

```
ExpReturn = [0.1 0.2 0.15];
ExpCovariance = [
  0.005   -0.010    0.004
 -0.010    0.040   -0.002
  0.004   -0.002    0.023 ];

PortReturn = [0.16
              0.17];
NumAssets = 3;
AssetMin = [0.20 NaN NaN];

Group    = [ 1    0   1 ];

GroupMax = 0.50;

ConSet = portcons('Default', NumAssets, 'AssetLims', AssetMin,...
NaN,'GroupLims', Group, NaN, GroupMax);

[PortRisk, PortReturn, PortWts] = portopt(ExpReturn,...
ExpCovariance, [], PortReturn, ConSet)

PortRisk =

    0.0919
    0.1138

PortReturn =


    0.1600
    0.1700


PortWts =

    0.3000    0.5000    0.2000
    0.2000    0.6000    0.2000
```

See Also

ewstats, frontcon, portcons, portstats

Purpose

Randomized portfolio risks, returns, and weights.

Syntax

```
[risk, ror, weights] = portrand(asset, ret, pts)
[risk, ror, weights] = portrand(asset, ret)
[risk, ror, weights] = portrand(asset)
portrand(asset, ret, pts)
```

Arguments

`asset` M-by-N matrix of time series data. Rows (M) are observations, and each column (N) represents a single security.

`ret` 1-by-N vector where each column represents the rate of return for the corresponding security in `asset`. By default, `ret` is computed by taking the average value of each column of `asset`.

`pts` Scalar that specifies how many random points should be generated. Default = 1000.

Description

`[risk, ror, weights] = portrand(asset, ret, pts)` returns the risks, rates of return, and weights of random portfolio configurations.

`risk` A `pts`-by-1 vector of standard deviations.

`ror` A `pts`-by-1 vector of expected rates of return.

`weights` A `pts`-by-N matrix of asset weights. Each row of `weights` is a different portfolio configuration.

`portrand(asset, ret, pts)` plots the points representing each portfolio configuration. It does not return any data to the MATLAB workspace.

This function is used in the MATLAB Financial Expo and illustrates how multiple weighting combinations of the same portfolio will generate the same expected rate of return.

See Also

`frontier`

Reference

Bodie, Kane, and Marcus, *Investments*, Chapter 7.

portsim

Purpose

Random simulation of correlated asset returns.

Syntax

```
RetSeries = portsim(ExpReturn, ExpCovariance, NumObs, RetIntervals,
   NumSim)
```

Arguments

`ExpReturn`	1-by-number of assets (NASSETS) vector specifying the expected (mean) return of each asset.
`ExpCovariance`	NASSETS-by-NASSETS matrix of asset-asset covariances. The standard deviations of the returns are: `ExpSigma = sqrt(diag(ExpCovariance))`.
`NumObs`	Number of consecutive observations in the return time series. If `NumObs` is entered as the empty matrix `[]`, the length of `RetIntervals` is used.
`RetIntervals`	Scalar or number of observations (NUMOBS)-by-1 vector of interval times between observations. If `RetIntervals` is not specified, all intervals are assumed to have length 1.
`NumSim`	Number of separate simulations of the NUMOBS observations to perform. Default = 1.

Description

`portsim` simulates returns of NASSETS assets over consecutive observation intervals. Returns are simulated as the increments of constant drift and volatility Brownian processes.

`RetSeries` is a NUMOBS-by-NASSETS-by-NUMSIM array of incremental return observations. The return over an interval of length `DT` is given by `ExpReturn*DT + ExpSigma*sqrt(DT)*randn`, where `randn` provides a random scalar whose value changes each time `randn` is referenced.

The returns realized from portfolios listed in `PortWts` are given by: `PortReturn = PortWts * RetSeries(:,:,1)'`, where `PortWts` is a matrix in which each row contains the asset allocations of a portfolio. Each row of `PortReturn` corresponds to one of the portfolios identified in `PortWts`, and each column corresponds to one of the observations in `RetSeries`. See `portopt` and `portstats` for portfolio specification and optimization.

Example

Create sample returns for three stocks over 10 periods.

```
ExpReturn = [0.1 0.2 0.15];

ExpCovariance = [
  0.005   -0.010    0.004
 -0.010    0.040   -0.002
  0.004   -0.002    0.023 ];

NumObs = 10;

RetSeries = portsim(ExpReturn, ExpCovariance, NumObs)

RetSeries =

    0.1429    0.2626    0.2365
    0.0821    0.1599   -0.1796
    0.0054    0.6126    0.1072
    0.1719   -0.0669    0.1913
    0.1518   -0.0843    0.0442
    0.0112    0.2709    0.1501
    0.0409    0.1683    0.1932
    0.1485    0.2522    0.2774
    0.0463    0.3222    0.0954
    0.1990    0.1024    0.3843
```

See Also

ewstats, portopt, portstats, randn, ret2tick

portstats

Purpose

Portfolio expected return and risk.

Syntax

```
[PortRisk, PortReturn] = portstats(ExpReturn, ExpCovariance,
  PortWts)
```

Arguments

`ExpReturn`	1-by-number of assets (NASSETS) vector specifying the expected (mean) return of each asset.
`ExpCovariance`	NASSETS-by-NASSETS matrix specifying the covariance of the asset returns.
`PortWts`	Number of portfolios (NPORTS)-by-NASSETS matrix of weights allocated to each asset. Each row represents a different weighting combination. Default = 1/NASSETS.

Description

`[PortRisk, PortReturn] = portstats(ExpReturn, ExpCovariance, PortWts)` computes the expected rate of return and risk for a portfolio of assets.

`PortRisk` is an NPORTS-by-1 vector of the standard deviation of each portfolio.

`PortReturn` is an NPORTS-by-1 vector of the expected return of each portfolio.

Examples

```
ExpReturn = [0.1 0.2 0.15];

ExpCovariance = [
    0.0100   -0.0061    0.0042
   -0.0061    0.0400   -0.0252
    0.0042   -0.0252    0.0225 ];

PortWts=[0.4 0.2 0.4; 0.2 0.4 0.2];

[PortRisk, PortReturn] = portstats(ExpReturn, ExpCovariance,...
PortWts)

PortRisk =

    0.0560
    0.0550

PortReturn =

    0.1400
    0.1300
```

See Also

frontcon

portvrisk

Purpose

Portfolio value at risk

Syntax

```
ValueAtRisk = portvrisk(PortReturn, PortRisk, RiskThreshold,
  PortValue)
```

Arguments

`PortReturn`	Number of portfolios (`NPORTS`)-by-1 vector or scalar of the expected return of each portfolio over the period.
`PortRisk`	`NPORTS`-by-1 vector or scalar of the standard deviation of each portfolio over the period.
`RiskThreshold`	`NPORTS`-by-1 vector or scalar specifying the loss probability. Default = `0.05` (5%).
`PortValue`	`NPORTS`-by-1 vector or scalar specifying the total value of asset portfolio. Default = `1`.

Description

`ValueAtRisk = portvrisk(PortReturn, PortRisk, RiskThreshold, PortValue)` returns the maximum potential loss in the value of a portfolio over one period of time, given the loss probability level `RiskThreshold`.

`ValueAtRisk` is an `NPORTS`-by-1 vector of the estimated maximum loss in the portfolio, predicted with a confidence probability of `1– RiskThreshold`.

If `PortValue` is not given, `ValueAtRisk` is presented on a per-unit basis. A value of `0` indicates no losses.

Examples

This example computes `ValueAtRisk` on a per-unit basis.

```
PortReturn = 0.29/100;
PortRisk = 3.08/100;
RiskThreshold = [0.01;0.05;0.10];
PortValue = 1;
ValueAtRisk = portvrisk(PortReturn,PortRisk,...
RiskThreshold,PortValue)
ValueAtRisk =

    0.0688
    0.0478
    0.0366
```

This example computes ValueAtRisk with actual values.

```
PortReturn = [0.29/100;0.30/100];
PortRisk = [3.08/100;3.15/100];
RiskThreshold = 0.10;
PortValue = [1000000000;500000000];
ValueAtRisk = portvrisk(PortReturn,PortRisk,...
RiskThreshold,PortValue)
ValueAtRisk =

  1.0e+007 *
    3.6572
    1.8684
```

See Also

frontcon, portopt

prbond

Purpose

Price of security with regular periodic interest payments.

Syntax

```
[p, ai] = prbond(sd, md, rv, cpn, yld, per, basis)
[p, ai] = prbond(sd, md, rv, cpn, yld, per)
[p, ai] = prbond(sd, md, rv, cpn, yld)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`cpn`	Coupon rate. Enter as decimal fraction.
`yld`	Yield. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[p, ai] = prbond(sd, md, rv, cpn, yld, per, basis)` returns the price `p` and accrued interest `ai` of a security with regular periodic interest payments. This function also applies to zero-coupon bonds or pure discount securities by setting `cpn = 0`.

Example

Using this data:

```
sd = '02/01/1960';
md = '01/01/1990';
rv = 1000;
cpn = 0.08;
yld = 0.06;
per = 2;
basis = 0;

[p, ai] = prbond(sd, md, rv, cpn, yld, per, basis)
```

returns

```
p =
    1276.64e+003
ai =
    6.8132
```

See Also

acrubond, prdisc, prmat, proddf, proddfl, proddl, yldbond

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 6, 7.

prdisc

Purpose

Price of discounted security.

Syntax

```
p = prdisc(sd, md, rv, disc, basis)
p = prdisc(sd, md, rv, disc)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`disc`	Discount rate of the security. Enter as decimal fraction.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`p = prdisc(sd, md, rv, disc, basis)` returns the price of a discounted security.

Example

Using this data:

```
sd = '10/14/1988';
md = '03/17/1989';
rv = 100;
disc = 0.087;
basis = 2;

p = prdisc(sd, md, rv, disc, basis)
```

returns

```
 p =
    96.2783
```

See Also

`acrudisc`, `prbond`, `prmat`, `ylddisc`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formula 2.

Purpose

Price with interest at maturity.

Syntax

```
[p, ai] = prmat(sd, md, id, rv, cpn, yld, basis)
[p, ai] = prmat(sd, md, id, rv, cpn, yld)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`id`	Issue date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`cpn`	Coupon rate. Enter as decimal fraction.
`yld`	Yield. Enter as decimal fraction.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[p, ai] = prmat(sd, md, id, rv, cpn, yld, basis)` returns the price `p` and accrued interest `ai` of a security that pays interest at maturity. This function also applies to zero-coupon bonds or pure discount securities by setting `cpn = 0`.

Example

Using this data:

```
sd = '02/07/1992';
md = '04/13/1992';
id = '10/11/1991';
rv = 100;
cpn = 0.0608;
yld = 0.0608;
basis = 1;

[p, ai] = prmat(sd, md, id, rv, cpn, yld, basis)
```

returns

```
p =
    99.9784
ai =
    1.9591
```

See Also

acrubond, acrudisc, prbond, prdisc, yldmat

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formula 4.

Purpose

Price with odd first period.

Syntax

```
[p, ai] = proddf(sd, md, id, fd, rv, cpn, yld, per, basis)
[p, ai] = proddf(sd, md, id, fd, rv, cpn, yld, per)
[p, ai] = proddf(sd, md, id, fd, rv, cpn, yld)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`id`	Issue date. Enter as serial date number or date string.
`fd`	First coupon date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`cpn`	Coupon rate. Enter as decimal fraction.
`yld`	Yield. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[p, ai] = proddf(sd, md, id, fd, rv, cpn, yld, per, basis)` returns the price `p` and accrued interest `ai` of a security with an odd first period and the settlement date in the first period.

Example

Using this data:

```
sd = '11/11/1992';
md = '03/01/2005';
id = '10/15/1992';
fd = '03/01/1993';
rv = 100;
cpn = 0.0785;
yld = 0.0625;
per = 2;
basis = 0;

[p, ai] = proddf(sd, md, id, fd, rv, cpn, yld, per, basis)
```

returns

```
p =
    113.5977
ai =
    0.5855
```

See Also

acrubond, cfdates, prbond, proddfl, proddl, yldoddf

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 8, 9.

Purpose

Price with odd first and last periods and settlement in first period.

Syntax

```
[p, ai] = proddfl(sd, md, id, fd, lcd, rv, cpn, yld, per, basis)
[p, ai] = proddfl(sd, md, id, fd, lcd, rv, cpn, yld, per)
[p, ai] = proddfl(sd, md, id, fd, lcd, rv, cpn, yld)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`id`	Issue date. Enter as serial date number or date string.
`fd`	First coupon date. Enter as serial date number or date string.
`lcd`	Last coupon date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`cpn`	Coupon rate. Enter as decimal fraction.
`yld`	Yield. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[p, ai] = proddfl(sd, md, id, fd, lcd, rv, cpn, yld, per, basis)` returns the price `p` and accrued interest `ai` of a security with odd first and last periods and the settlement date in the first period.

Example

Using this data:

```
sd = '03/15/1993';
md = '03/01/2020';
id = '03/01/1993';
fd = '07/01/1993';
lcd = '01/01/2020';
rv = 100;
cpn = 0.04;
yld = 0.0427;
per = 2;
basis = 1;

[p, ai] = proddfl(sd, md, id, fd, lcd, rv, cpn, yld, per, basis)
```

returns

```
p =
    95.6939
ai =
    0.1556
```

See Also

`acrubond`, `cfdates`, `prbond`, `proddf`, `proddl`, `yldbond`, `yldoddfl`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 16, 17, 18, 19.

Purpose

Price with odd last period.

Syntax

```
[p, ai] = proddl(sd, md, lcd, rv, cpn, yld, per, basis)
[p, ai] = proddl(sd, md, lcd, rv, cpn, yld, per)
[p, ai] = proddl(sd, md, lcd, rv, cpn, yld)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`lcd`	Last coupon date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`cpn`	Coupon rate. Enter as decimal fraction.
`yld`	Yield. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[p, ai] = proddl(sd, md, lcd, rv, cpn, yld, per, basis)` returns the price `p` and accrued interest `ai` of a security with odd last period.

Example

Using this data:

```
sd = '02/07/1993';
md = '08/01/1993';
lcd = '02/04/1993';
rv = 100;
cpn = 0.0650;
yld = 0.0535;
per = 2;
basis = 1;

[p, ai] = proddl(sd, md, lcd, rv, cpn, yld, per, basis)
```

returns

```
p =
        100.5411
ai =
          0.0542
```

See Also

acrubond, cfdates, prbond, proddf, proddfl, yldoddl

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 11, 13, 14, 15.

Purpose

Price of Treasury bill.

Syntax

```
p = prtbill(sd, md, rv, disc)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`disc`	Discount rate of the Treasury bill. Enter as decimal fraction.

Description

`p = prtbill(sd, md, rv, disc)` returns the price `p` for a Treasury bill.

Example

The settlement date of a Treasury bill is February 10, 1992, the maturity date is August 6, 1992, the discount rate is 3.77%, and the par value is $1000. Using this data:

```
p = prtbill('2/10/1992', '8/6/1992', 1000, 0.0377)
```

returns

```
p =
        981.3594
```

See Also

`beytbill`, `yldtbill`

Reference

Bodie, Kane, and Marcus, *Investments*, pages 41-43.

pvfix

Purpose

Present value with fixed periodic payments.

Syntax

```
p = pvfix(rate, nper, p, fv, due)
p = pvfix(rate, nper, p, fv)
p = pvfix(rate, nper, p)
```

Arguments

`rate` Periodic interest rate, as a decimal fraction.

`nper` Number of periods.

`p` Periodic payment.

`fv` Payment received other than `p` in the last period. Default = `0`.

`due` When payments are due or made: `0` = end of period (default), or `1` = beginning of period.

Description

`p = pvfix(rate, nper, p, fv, due)` returns the present value of a series of equal payments.

Example

$200 is paid monthly into a savings account earning 6%. The payments are made at the end of the month for five years. To find the present value of these payments:

```
pv = pvfix(0.06/12, 5*12, 200, 0, 0)
```

returns

```
pv =
      10345.11
```

See Also

`fvfix`, `fvvar`, `payper`, `pvvar`

Purpose

Present value of varying cash flow.

Syntax

```
pv = pvvar(cf, rate, df)
pv = pvvar(cf, rate)
```

Arguments

`cf` A vector of varying cash flows. Include the initial investment as the initial cash flow value (a negative number).

`rate` Periodic interest rate. Enter as a decimal fraction.

`df` For irregular (nonperiodic) cash flows, a vector of dates on which the cash flows occur. Enter dates as serial date numbers or date strings. Default assumes `cf` contains regular (periodic) cash flows.

Description

`pv = pvvar(cf, rate, df)` returns the net present value `pv` of a varying cash flow.

Examples

This cash flow represents the yearly income from an initial investment of $10,000. The annual interest rate is 8%.

Year 1	$2000
Year 2	$1500
Year 3	$3000
Year 4	$3800
Year 5	$5000

To calculate the net present value of this regular cash flow:

```
pv = pvvar([-10000 2000 1500 3000 3800 5000], 0.08)
```

returns

```
pv =
       1715.39
```

An investment of $10,000 returns this irregular cash flow. The original investment and its date are included. The periodic interest rate is 9%.

Cash flow	Dates
($10000)	January 12, 1987
$2500	February 14, 1988
$2000	March 3, 1988
$3000	June 14, 1988
$4000	December 1, 1988

To calculate the net present value of this irregular cash flow:

```
cf = [-10000, 2500, 2000, 3000, 4000];
df = ['01/12/1987'
      '02/14/1988'
      '03/03/1988'
      '06/14/1988'
      '12/01/1988'];

pv = pvvar(cf, 0.09, df)
```

returns

```
pv =
        142.16
```

See Also

`fvfix`, `fvvar`, `irr`, `payuni`, `pvfix`

Purpose

Zero curve given a par yield curve.

Syntax

```
[zr, cd] = pyld2zero(pr,cd,sd,ocomp,obasis,icomp,ibasis,maxiter)
[zr, cd] = pyld2zero(pr,cd,sd,ocomp,obasis,icomp,ibasis)
[zr, cd] = pyld2zero(pr,cd,sd,ocomp,obasis,icomp)
[zr, cd] = pyld2zero(pr,cd,sd,ocomp,obasis)
[zr, cd] = pyld2zero(pr,cd,sd,ocomp)
[zr, cd] = pyld2zero(pr,cd,sd)
```

Arguments

`pr` Par yield rates. An N-by-1 vector of annualized par yields, as decimal fractions. (Par yields = coupon rates.) In aggregate, the yield rates in `pr` constitute a par yield curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the par rates in `pr`. Use `datenum` to convert date strings to serial date numbers.

`sd` Settlement date. A serial date number that is the common settlement date for the par rates in `pr`.

`ocomp` Output compounding. A scalar that sets the compounding frequency per year for annualizing the output zero rates in `zr`. Allowed values are:

- `1` = annual compounding
- `2` = semi-annual compounding (default)
- `3` = compounding three times per year
- `4` = quarterly compounding
- `6` = bimonthly compounding
- `12` = monthly compounding

`obasis` Output day-count basis for annualizing the output zero rates in `zr`. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

`icomp` Input compounding. A scalar that indicates the compounding frequency per year used for annualizing the input par rates in `pr`. Allowed values are the same as for `ocomp`. Default = `ocomp`.

`ibasis` Input day-count basis used for annualizing the input par rates in `pr`. Allowed values are the same as for `obasis`. Default = `obasis`.

maxiter Maximum number of iterations for deriving the zero rates in zr. A scalar. Default = 200.

Description

[zr, cd] = pyld2zero(pr,cd,sd,ocomp,obasis,icomp,ibasis,maxiter) returns a zero curve given a par yield curve and its maturity dates.

zr Zero rates. An N-by-1 vector of decimal fractions. In aggregate, the rates in zr constitute a zero curve for the investment horizon represented by cd.

cd Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) corresponding to the zero rates in zr. This vector is the same as the input vector cd. Use datestr to convert serial date numbers to date strings.

Example

Given a par yield curve pr over a set of maturity dates cd, and a settlement date sd:

```
pr = [0.0479
      0.0522
      0.0540
      0.0540
      0.0536
      0.0532
      0.0532
      0.0539
      0.0558
      0.0543];

cd = [datenum('06-Nov-1997')
      datenum('11-Dec-1997')
      datenum('15-Jan-1998')
      datenum('05-Feb-1998')
      datenum('04-Mar-1998')
      datenum('02-Apr-1998')
      datenum('30-Apr-1998')
      datenum('25-Jun-1998')
      datenum('04-Sep-1998')
      datenum('12-Nov-1998')];

sd = datenum('03-Nov-1997');
```

Set monthly compounding for the zero curve, on an actual/365 basis. The par yield curve was compounded annually on an actual/actual basis. Derive the zero curve within 50 iterations.

```
ocomp = 12;
obasis = 3;
icomp = 1;
ibasis = 0;
maxiter = 50;
```

Execute the function

```
[zr, cd] = pyld2zero(pr,cd,sd,ocomp,obasis,icomp,ibasis,maxiter)
```

which returns the zero curve `zr` at the maturity dates `cd`:

```
zr =
    0.0466
    0.0494
    0.0514
    0.0514
    0.0504
    0.0504
    0.0502
    0.0514
    0.0535
    0.0530
cd =
      729700
      729735
      729770
      729791
      729818
      729847
      729875
      729931
      730002
      730071
```

(For readability, `pr` and `zr` are shown only to the basis point. However, MATLAB computed them at full precision. If you enter `pr` as shown, `zr` may differ due to rounding.)

See Also `zero2pyld` and other functions for Term Structure of Interest Rates

Purpose

Price tick series from incremental returns and initial price.

Syntax

```
[TickSeries, TickTimes] = ret2tick(RetSeries, StartPrice,
  RetIntervals, StartTime)
```

Arguments

`RetSeries`	Number of observations (`NUMOBS`)-by-number of assets (`NASSETS`) matrix of incremental return observations. The `i`'th return is quoted for the period `TickTimes(i)` to `TickTimes(i+1)` and is not scaled to a yearly return.
`StartPrice`	1-by-`NASSETS` vector of initial asset prices. Prices start at 1 if `StartPrice` is not specified. (Optional)
`RetIntervals`	Scalar or `NUMOBS`-by-1 vector of interval times between observations. If not specified, all intervals are assumed to have length 1. (Optional)
`StartTime`	Starting time for first observation. (Optional)

Description

`ret2tick` generates price values from the starting prices of `NASSETS` investments and `NUMOBS` incremental return observations.

`TickSeries` is a `NUMOBS+1` by `NASSETS` matrix of prices of equity assets. The first row is the starting price of the assets.

`TickTimes` is a `NUMOBS+1` by 1 vector of tick observation times. The initial time is zero unless specified in `StartTime`.

Example

Compute the price increase of two stocks over a year's time based on three incremental return observations.

```
RetSeries = [0.10 0.12
             0.05 0.04
            -0.05 0.05];

RetIntervals = [182
                91
                92];

StartTime = datenum('18-Dec-1998')

[TickSeries,TickTimes] = ret2tick(RetSeries,[],RetIntervals,...
StartTime)

TickSeries =
    1.0000    1.0000
    1.1000    1.1200
    1.1550    1.1648
    1.0973    1.2230

datestr(TickTimes)

ans =
18-Dec-1998
18-Jun-1999
17-Sep-1999
18-Dec-1999
```

See Also

portsim, tick2ret

Purpose

Seconds of date or time.

Syntax

```
s = second(d)
```

Description

`s = second(d)` returns the seconds given a serial date number or a date string `d`.

Example

```
s = second(728647.558427893)
```

or

```
s = second('19-dec-1994, 13:24:08.17')
```

returns

```
s =
    8.1700
```

See Also

`datevec`, `hour`, `minute`

taxedrr

Purpose

After-tax rate of return.

Syntax

```
r = taxedrr(return, tax)
```

Arguments

`return` Nominal rate of return. Enter as a decimal fraction.

`tax` Tax rate. Enter as a decimal fraction.

Description

`r = taxedrr(return, tax)` calculates the after-tax rate of return.

Example

An investment has a 12% nominal rate of return and is taxed at a 30% rate. The after-tax rate of return is:

```
r = taxedrr(0.12, 0.30)

r =
    0.0840
```

or 8.4%

See Also

`effrr`, `irr`, `mirr`, `nomrr`, `xirr`

Purpose

Treasury bond parameters given Treasury bill parameters.

Syntax

```
tbond = tbl2bond(tbill)
```

Arguments

`tbill` Treasury bill parameters. An N-by-5 matrix where each row describes a Treasury bill. N is the number of Treasury bills.
Columns are `[md daysm bid ask askyld]` where

- `md` Maturity date, as a serial date number. Use `datenum` to convert date strings to serial date numbers.
- `daysm` Days to maturity, as an integer.
- `bid` Bid bank-discount rate: the percentage discount from face value at which the bill could be bought, annualized on a simple-interest basis. As a decimal fraction.
- `ask` Asked bank-discount rate, as a decimal fraction.
- `askyld` Asked yield: the bond-equivalent yield from holding the bill to maturity, annualized on a simple-interest basis and assuming a 365-day year. As a decimal fraction.

Description

`tbond = tbl2bond(tbill)` restates U.S. Treasury bill market parameters in U.S. Treasury bond form as zero-coupon bonds. This function makes Treasury bills directly comparable to Treasury bonds and notes.

`tbond` Treasury bond parameters. An N-by-5 matrix where each row describes an equivalent Treasury (zero-coupon) bond.
Columns are `[cpn md bidp askp askytm]` where

- `cpn` Coupon rate, which is always `0`.
- `md` Maturity date, as a serial date number. This date is the same as the `tbill` date `md`. Use `datestr` to convert serial date numbers to date strings.
- `bidp` Bid price based on $100 face value.
- `askp` Asked price based on $100 face value.
- `askytm` Asked yield to maturity: the effective return from holding the bond to maturity, annualized on a compound-interest basis.

Example

Given published Treasury bill market parameters for December 22, 1997:

```
tbill = [datenum('jan 02 1998')  10  0.0526  0.0522  0.0530
         datenum('feb 05 1998')  44  0.0537  0.0533  0.0544
         datenum('mar 05 1998')  72  0.0529  0.0527  0.0540];
```

Execute the function.

```
tbond = tbl2bond(tbill)
```

```
tbond =
            0  729757       99.854       99.855     0.054391
            0  729791       99.344       99.349     0.055714
            0  729819       98.942       98.946     0.055184
```

(Example output has been formatted for readability.)

See Also

`tr2bonds` and other functions for Term Structure of Interest Rates

Purpose

Incremental return series from a tick price series.

Syntax

```
[RetSeries, RetIntervals] = tick2ret(TickSeries, TickTimes)
```

Arguments

`TickSeries`	Number of observations (`NUMOBS`)-by-number of assets (`NASSETS`) matrix of prices of equity assets. First row is oldest observation. Last row is most recent.
`TickTimes`	`NUMOBS`-by-1 increasing vector of observation times. Times are taken as serial date numbers (day units) or as decimal numbers in arbitrary units (e.g., yearly). (Optional)

Description

`tick2ret` computes the asset returns realized between `NUMOBS` observations of prices of `NASSETS` assets.

`RetSeries` is a `NUMOBS-1` by `NASSETS` matrix of incremental return observations. The i'th return is quoted for the period `TickTimes(i)` to `TickTimes(i+1)` and is not scaled to a yearly return.

```
RetSeries(i) = TickSeries(i+1)/TickSeries(i) – 1
```

`RetIntervals` is a `NUMOBS-1` by 1 vector of interval times between observations. If `TickTimes` is not specified, all intervals are assumed to have length 1.

Example

Compute the periodic returns of two stocks observed in the first, second, third, and fourth quarters.

```
TickSeries = [100 80
              110 90
              115 88
              110 91];

TickTimes = [0
             6
             9
             12];

[RetSeries, RetIntervals] = tick2ret(TickSeries, TickTimes)

RetSeries =
    0.1000    0.1250
    0.0455   -0.0222
   -0.0435    0.0341

RetIntervals =
     6
     3
     3
```

See Also

ewstats, ret2tick

Purpose

Current date.

Syntax

```
t = today
```

Description

`t = today` returns the current date as a serial date number.

Example

```
t = today
```

returns

```
t =
      728911
```

on September 9, 1995.

See Also

`datenum`, `datestr`, `now`

tr2bonds

Purpose

Term-structure parameters given Treasury bond parameters.

Syntax

```
[bonds, p, y] = tr2bonds(tbond)
```

Arguments

`tbond` Treasury bond parameters. An N-by-5 matrix where each row describes a Treasury bond. N is the number of Treasury bonds. Columns are `[cpn md bidp askp askytm]` where

- `cpn` Coupon rate, as a decimal fraction.
- `md` Maturity date, as a serial date number. Use `datenum` to convert date strings to serial date numbers.
- `bidp` Bid price based on $100 face value.
- `askp` Asked price based on $100 face value.
- `askytm` Asked yield to maturity, as a decimal fraction.

Description

`[bonds, p, y] = tr2bonds(tbond)` returns term-structure parameters — bond information, prices, and yields — sorted by ascending maturity date `md`, given Treasury bond parameters. The formats of the output matrix and vectors meet requirements for input to the `zbtprice` and `zbtyield` zero-curve bootstrapping functions.

`bonds` Coupon bond information. An N-by-6 matrix where each row describes a bond. N is the number of bonds.
Columns are `[md cpn rv per basis eom]` where

- `md` Maturity date of the bond, as a serial date number. Use `datestr` to convert serial date numbers to date strings.
- `cpn` Coupon rate of the bond, as a decimal fraction.
- `rv` Redemption or face value of the bond, always `100`.
- `per` Coupons per year of the bond, always `2`.
- `basis` Day-count basis of the bond, always `0` (actual/actual).
- `eom` End-of-month flag, always `1`, meaning that a bond's coupon payment date is always the last day of the month.

p Prices. An N-by-1 vector containing the price of each bond in bonds, respectively. The number of rows (N) matches the number of rows in bonds.

y Yields. An N-by-1 vector containing the yield to maturity of each bond in bonds, respectively. The number of rows (N) matches the number of rows in bonds.

Example

Given published Treasury bond market parameters for December 22, 1997:

```
tbond=[0.0650  datenum('15-apr-1999') 101.03125 101.09375 0.0564
       0.05125 datenum('17-dec-1998')  99.4375   99.5     0.0563
       0.0625  datenum('30-jul-1998') 100.3125  100.375   0.0560
       0.06125 datenum('26-mar-1998') 100.09375 100.15625 0.0546];
```

Execute the function.

```
[bonds, p, y] = tr2bonds(tbond)

bonds =
  729840    0.06125    100    2    0    1
  729966    0.0625     100    2    0    1
  730106    0.05125    100    2    0    1
  730225    0.065      100    2    0    1
p =
  100.1563
  100.3750
   99.5000
  101.0938
y =
    0.0546
    0.056
    0.0563
    0.0564
```

(Example output has been formatted for readability.)

See Also

tbl2bond, zbtprice, zbtyield, and other functions for Term Structure of Interest Rates

ugarch

Purpose

Univariate GARCH(P,Q) parameter estimation with Gaussian innovations.

Syntax

```
[Kappa, Alpha, Beta] = ugarch(U, P, Q)
```

Arguments

`U`	Single column vector of random disturbances, i.e., the residuals, or innovations, of an econometric model representing a mean-zero, discrete-time stochastic process. The innovations time series `U` is assumed to follow a GARCH(P,Q) process.
`P`	Non-negative, scalar integer representing a model order of the GARCH process. `P` is the number of lags of the conditional variance. `P` can be zero; when `P = 0`, a GARCH(0,Q) process is actually an ARCH(Q) process.
`Q`	Positive, scalar integer representing a model order of the GARCH process. `Q` is the number of lags of the squared innovations.

Description

`[Kappa, Alpha, Beta] = ugarch(U, P, Q)` computes estimated univariate GARCH(P,Q) parameters with Gaussian innovations.

`Kappa` is the estimated scalar constant term of the GARCH process.

`Alpha` is a `P`-by-1 vector of estimated coefficients, where `P` is the number of lags of the conditional variance included in the GARCH process.

`Beta` is a `Q`-by-1 vector of estimated coefficients, where `Q` is the number of lags of the squared innovations included in the GARCH process.

GARCH(P,Q) coefficients {`Kappa`, `Alpha`, `Beta`} are subject to constraints:

```
Kappa > 0
Alpha(i) >= 0 for i = 1,2,...P
Beta(i)  >= 0 for i = 1,2,...Q
sum(Alpha(i) + Beta(j)) < 1 for i = 1,2,...P and j = 1,2,...Q
```

The time-conditional variance, `H(t)`, of a GARCH(P,Q) process is modeled as:

```
H(t) = Kappa + Alpha(1)*H(t–1) + Alpha(2)*H(t–2) +...+
Alpha(P)*H(t–P)+ Beta(1)*U^2(t–1)+ Beta(2)*U^2(t–2)+...+
Beta(Q)*U^2(t–Q)
```

Note that `U` is a vector of innovations, or regression residuals of an econometric model, representing a mean-zero, discrete-time stochastic process. That is, it is assumed that a regression model has already been run, and that `U(t) = y(t) – F(X(t),B)` is the time series of innovations derived from the model.

Although `H` is generated via the equation above, `U` and `H` are related as

```
U(t) = sqrt(H(t))*v(t)
```

where `v(t)` is an i.i.d. sequence ~ `N(0,1)`.

Example

See `ugarchsim` for an example of a GARCH(P,Q) process.

See Also

`ugarchpred`, `ugarchsim`

Reference

James D. Hamilton, *Time Series Analysis*, Princeton University Press, 1994

ugarchllf

Purpose

Log-likelihood objective function of univariate GARCH(P,Q) processes with Gaussian innovations.

Syntax

```
LogLikelihood = ugarchllf(Parameters, U, P, Q)
```

Arguments

`Parameters`	`(1 + P + Q)`- by-1 column vector of GARCH(P,Q) process parameters. The first element is the scalar constant term of the GARCH process; the next `P` elements are coefficients associated with the P lags of the conditional variance terms; the next `Q` elements are coefficients associated with the `Q` lags of the squared innovations terms.
`U`	Single column vector of random disturbances, i.e., the residuals, or innovations, of an econometric model representing a mean-zero, discrete-time stochastic process. The innovations time series `U` is assumed to follow a GARCH(P,Q) process.
`P`	Non-negative, scalar integer representing a model order of the GARCH process. `P` is the number of lags of the conditional variance. `P` can be zero; when `P = 0`, a GARCH(0,Q) process is actually an ARCH(Q) process.
`Q`	Positive, scalar integer representing a model order of the GARCH process. `Q` is the number of lags of the squared innovations.

Description

`LogLikelihood = ugarchllf(Parameters, U, P, Q)` computes the log-likelihood objective function of univariate GARCH(P,Q) processes with Gaussian innovations.

`LogLikelihood` is a scalar value of the GARCH(P,Q) log-likelihood objective function given the input arguments. This function is meant to be optimized via the `fmincon` function of the MATLAB Optimization Toolbox.

`fmincon` is a minimization routine. To maximize the log-likelihood function, the `LogLikelihood` output parameter is actually the negative of what is formally presented in most time series or econometrics references.

The time-conditional variance, `H(t)`, of a GARCH(P,Q) process is modeled as

```
H(t) = Kappa + Alpha(1)*H(t–1) + Alpha(2)*H(t–2) +...+
Alpha(P)*H(t–P)+ Beta(1)*U^2(t–1)+ Beta(2)*U^2(t–2)+...+
Beta(Q)*U^2(t–Q)
```

`U` is vector of innovations representing a mean-zero, discrete time stochastic process. Although `H` is generated via the equation above, `U` and `H` are related as

```
U(t) = sqrt(H(t))*v(t)
```

where `{v(t)}` is an i.i.d.sequence `~ N(0,1)`.

Since `ugarchllf` is really just a helper function, no argument checking is performed. This function is not meant to be called directly from the command line.

See Also

`ugarch`, `ugarchpred`, `ugarchsim`

ugarchpred

Purpose

Forecast conditional variance of univariate GARCH(P,Q) processes.

Syntax

```
[VarianceForecast, H] = ugarchpred(U, Kappa, Alpha, Beta,
  NumPeriods)
```

Arguments

`U`	Single column vector of random disturbances, i.e., the residuals, or innovations, of an econometric model representing a mean-zero, discrete-time stochastic process. The innovations time series `U` is assumed to follow a GARCH(P,Q) process.
`Kappa`	Scalar constant term of the GARCH process.
`Alpha`	`P`-by-1 vector of coefficients, where `P` is the number of lags of the conditional variance included in the GARCH process. `Alpha` can be an empty matrix, in which case `P` is assumed 0; when `P = 0`, a GARCH(0,Q) process is actually an ARCH(Q) process.
`Beta`	`Q`-by-1 vector of coefficients, where `Q` is the number of lags of the squared innovations included in the GARCH process.
`NumPeriods`	Positive, scalar integer representing the forecast horizon of interest, expressed in periods compatible with the sampling frequency of the input innovations column vector `U`.

Description

`[VarianceForecast, H] = ugarchpred(U, Kappa, Alpha, Beta, NumPeriods)` forecasts the conditional variance of univariate GARCH(P,Q) processes.

`VarianceForecast` is a number of periods (`NUMPERIODS`)-by-1 vector of the minimum mean-square error forecast of the conditional variance of the innovations time series vector `U`. The first element contains the 1-period-ahead forecast, the second element contains the 2-period-ahead forecast, and so on. Thus, if a forecast horizon greater than 1 is specified (`NUMPERIODS > 1`),the forecasts of all intermediate horizons are returned as well; in this case, the last element contains the variance forecast of the specified horizon, `NumPeriods` from the most recent observation in `U`.

H is a single column vector of the same length as the input innovations vector U. To model the GARCH(P,Q) process, you must construct the conditional variance time series, H(t), (see below). This represents the time series inferred from the innovations U, and is a reconstruction of the "past" conditional variances, whereas the VarianceForecast output represents the projection of conditional variances into the "future".This sequence is based on setting pre-sample values of H(t) to the unconditional variance of the U(t) process.

GARCH(P,Q) coefficients {Kappa, Alpha, Beta} are subject to constraints

```
Kappa > 0
Alpha(i) >= 0 for i = 1,2,...P
Beta(i)  >= 0 for i = 1,2,...Q
sum(Alpha(i) + Beta(j)) < 1 for i = 1,2,...P and j = 1,2,...Q
```

The time-conditional variance, H(t), of a GARCH(P,Q) process is modeled as:

```
H(t) = Kappa + Alpha(1)*H(t–1) + Alpha(2)*H(t–2) +...+
Alpha(P)*H(t–P)+ Beta(1)*U^2(t–1)+ Beta(2)*U^2(t–2)+...+
Beta(Q)*U^2(t–Q)
```

Note that U is a vector of innovations, or regression residuals of an econometric model, representing a mean-zero, discrete-time stochastic process. That is, it is assumed that a regression model has already been run, and that U(t) = y(t) – F(X(t),B) is the time series of innovations derived from the model.

Although H is generated via the equation above, U and H are related as:

```
U(t) = sqrt(H(t))*v(t)
```

where v(t) is an i.i.d. sequence.

Example

See ugarchsim for an example of forecasting the conditional variance of a univariate GARCH(P,Q) process.

See Also

ugarch, ugarchsim

ugarchsim

Purpose

Simulate a univariate GARCH(P,Q) process with Gaussian innovations.

Syntax

```
[U, H] = ugarchsim(Kappa, Alpha, Beta, NumSamples)
```

Arguments

`Kappa`	Scalar constant term of the GARCH process.
`Alpha`	`P`-by-1 vector of coefficients, where `P` is the number of lags of the conditional variance included in the GARCH process. `Alpha` can be an empty matrix, in which case `P` is assumed `0`; when `P = 0`, a GARCH(0,Q) process is actually an ARCH(Q) process.
`Beta`	`Q`-by-1 vector of coefficients, where `Q` is the number of lags of the squared innovations included in the GARCH process.
`NumSamples`	Positive, scalar integer indicating the number of samples of the innovations `U` and conditional variance `H` (see below) to simulate.

Description

`[U, H] = ugarchsim(Kappa, Alpha, Beta, NumSamples)` Simulates a univariate GARCH(P,Q) process with Gaussian innovations.

`U` is a number of samples (`NUMSAMPLES`)-by-1 vector of innovations, representing a mean-zero, discrete-time stochastic process. The innovations time series `U` is designed to follow the GARCH(P,Q) process specified by the inputs `Kappa`, `Alpha`, and `Beta`.

`H` is a `NUMSAMPLES`-by-1 vector of the conditional variances corresponding to the innovations vector `U`. Note that `U` and `H` are the same length, and form a "matching" pair of vectors. To model the GARCH(P,Q) process, the conditional variance time series, `H(t)`, must be constructed (see below). Thus, `H(t)` represents the time series inferred from the innovations time series vector `U`.

GARCH(P,Q) coefficients {`Kappa`, `Alpha`, `Beta`} are subject to constraints:

```
Kappa > 0
Alpha(i) >= 0 for i = 1,2,...P
Beta(i)  >= 0 for i = 1,2,...Q
sum(Alpha(i) + Beta(j)) < 1 for i = 1,2,...P and j = 1,2,...Q
```

The time-conditional variance, H(t), of a GARCH(P,Q) process is modeled as:

```
H(t) = Kappa + Alpha(1)*H(t–1) + Alpha(2)*H(t–2) +...+
Alpha(P)*H(t–P)+ Beta(1)*U^2(t–1)+ Beta(2)*U^2(t–2)+...+
Beta(Q)*U^2(t–Q)
```

Note that U is a vector of innovations, or regression residuals of an econometric model, representing a mean-zero, discrete-time stochastic process. That is, it is assumed that a regression model has already been run, and that U(t) = y(t) – F(X(t),B) is the time series of innovations derived from the model.

Although H is generated via the equation above, U and H are related as

```
U(t) = sqrt(H(t))*v(t)
```

where v(t) is an i.i.d. sequence ~ N(0,1).

The output vectors U and H are designed to be steady-state sequences; transients have arbitrarily small effect. The (arbitrary) metric used strips the first N samples of U and H such that the sum of the GARCH coefficients, excluding Kappa, raised to the N-th power, will not exceed 0.01:

```
0.01 = (sum(Alpha) + sum(Beta))^N
```

Thus

```
N = log(0.01)/log((sum(Alpha) + sum(Beta)))
```

Example

This example simulates a GARCH(P,Q) process with `P = 2` and `Q = 1`.

```
% Set the random number generator seed for reproducability.
randn('seed', 10)
% Set the simulation parameters of GARCH(P,Q) = GARCH(2,1) process.
Kappa = 0.25;       %a positive scalar.
Alpha = [0.2 0.1]'; %a column vector of non-negative numbers (P = 2).
Beta = 0.4;         % Q = 1.
NumSamples = 500;  % number of samples to simulate.
% Now simulate the process.
[U , H] = ugarchsim(Kappa, Alpha, Beta, NumSamples);
% Now estimate the process parameters.
P = 2;    % Model order P (P = length of Alpha).
Q = 1;    % Model order Q (Q = length of Beta).
[k, a, b] = ugarch(U , P , Q);
disp(' ')
disp(' Estimated Coefficients:')
disp(' ------------------------')
disp([k; a; b])
disp(' ')
% Now forecast the conditional variance using the estimated
% coefficients.
NumPeriods = 10;    % Forecast out to 10 periods.
[VarianceForecast, H1] = ugarchpred(U, k, a, b, NumPeriods);
disp(' Variance Forecasts:')
disp(' -------------------')
disp(VarianceForecast)
disp(' ')
```

When the above code is executed, the screen output looks like the display shown.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Diagnostic Information

Number of variables: 4

Functions
Objective:        ugarchllf
Gradient:         finite-differencing
Hessian:          finite-differencing (or Quasi-Newton)

Constraints
Nonlinear constraints:             do not exist
Number of linear inequality constraints:    1
Number of linear equality constraints:      0
Number of lower bound constraints:          4
Number of upper bound constraints:          0
Algorithm selected
      medium-scale


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  End diagnostic information
```

Iter	F-count	f(x)	max constraint	Step-size	Directional derivative	Procedure
1	5	699.185	-0.125	1	-2.97e+006	
2	22	658.224	-0.1249	0.000488	-64.6	
3	28	610.181	0	1	-49.4	
4	35	590.888	0	0.5	-38.9	
5	42	583.961	-0.03317	0.5	-29.8	
6	49	583.224	-0.02756	0.5	-31.8	
7	57	582.947	-0.02067	0.25	-7.28	
8	63	578.182	0	1	-2.43	
9	71	578.138	-0.09145	0.25	-0.55	
10	77	577.898	-0.04452	1	-0.148	
11	84	577.882	-0.06128	0.5	-0.0488	
12	90	577.859	-0.07117	1	-0.000758	
13	96	577.858	-0.07033	1	-0.000305	Hessian modified
14	102	577.858	-0.07042	1	-3.32e-005	Hessian modified
15	108	577.858	-0.0707	1	-1.29e-006	Hessian modified
16	114	577.858	-0.07077	1	-1.29e-007	Hessian modified
17	120	577.858	-0.07081	1	-1.97e-007	Hessian modified
18	127	577.858	-0.07079	0.5	-1.32e-008	Hessian modified

```
   Optimization Converged Successfully
   Magnitude of directional derivative in search direction
     less than 2*options.TolFun and maximum constraint violation
     is less than options.TolCon
No Active Constraints

Estimated Coefficients:
-----------------------
0.2520
0.0708
0.1623
0.4000

Variance Forecasts:
-------------------
1.3243
0.9594
0.9186
0.8402
0.7966
0.7634
0.7407
0.7246
0.7133
0.7054
```

See Also

ugarch, ugarchpred

Reference

James D. Hamilton, *Time Series Analysis*, Princeton University Press, 1994

Purpose

Day of the week.

Syntax

```
[d, w] = weekday(t)
```

Description

`[d, w] = weekday(t)` returns the day of the week in numeric `d` and string form `w` given a serial date number or date string `t`. The days of the week have these values:

d	w
1	Sun
2	Mon
3	Tue
4	Wed
5	Thu
6	Fri
7	Sat

Note: This function now ships with basic MATLAB. It originally shipped only with the Financial Toolbox, and this description remains here for your convenience.

Example

```
[d, w] = weekday(729736)
```

or

```
[d, w] = weekday('12-Dec-1997')
```

returns

```
d =
     6
w =
Fri
```

See Also

`datenum`, `datestr`, `datevec`, `day`

wrkdydif

Purpose

Number of working days between dates.

Syntax

```
n = wrkdydif(d1, d2, hols)
```

Description

`n = wrkdydif(d1, d2, hols)` returns the number of working days between dates `d1` and `d2`. `hols` is the number of holidays between the given dates, an integer. Enter dates as serial date numbers or date strings.

Example

```
n = wrkdydif('9/1/1995', '9/11/1995', 1)
```

or

```
n = wrkdydif(728903, 728913, 1)
```

returns

```
n =
     6
```

See Also

`busdate`, `datewrkdy`, `days360`, `days365`, `daysact`, `daysdif`, `holidays`, `yearfrac`

Purpose

Excel serial date number to MATLAB serial date number.

Syntax

```
d = x2mdate(xd, dsys)
d = x2mdate(xd)
```

Arguments

`xd` Excel serial date number. A vector or scalar.

`dsys` Excel date system. A vector or scalar.
`0` = 1900 date system (default), in which 1 = January 1, 1900 A.D.
`1` = 1904 date system, in which 0 = January 1, 1904 A.D.

Vector arguments must have consistent dimensions.

Description

`d = x2mdate(xd, dsys)` converts Excel serial date numbers `xd` to MATLAB serial date numbers `d`. MATLAB date numbers start with 1 = January 1, 0000 A.D., hence there is a difference of 693960 relative to the 1900 date system, or 695422 relative to the 1904 date system. This function is useful with MATLAB Excel Link.

Example

Given Excel date numbers in the 1904 system

```
xd = [34327  34692  35057  35423];
```

convert them to MATLAB date numbers

```
d = x2mdate(xd, 1)
d =
      729749       730114       730479       730845
```

and then to date strings:

```
datestr(d)
ans =
25-Dec-1997
25-Dec-1998
25-Dec-1999
25-Dec-2000
```

See Also

`datenum`, `datestr`, `m2xdate`

xirr

Purpose

Internal rate of return for nonperiodic cash flow.

Syntax

```
yld = xirr(cf, df, guess, maxiter)
yld = xirr(cf, df, guess)
yld = xirr(cf, df)
```

Arguments

`cf` A vector of nonperiodic cash flows. Include the initial investment as the initial cash flow value (a negative number).

`df` A vector of dates on which the cash flows occur. Enter dates as serial date numbers or date strings.

`guess` Initial estimate of the expected return. Default = `0.1` (10%).

`maxiter` Number of iterations used by Newton's method to solve for `yld`. Default = `50`.

Description

`yld = xirr(cf, df, guess, maxiter)` returns the internal rate of return for a schedule of nonperiodic cash flows.

Example

An investment of $10,000 returns this nonperiodic cash flow. The original investment and its date are included.

Cash flow	Dates
($10000)	January 12, 1987
$2500	February 14, 1988
$2000	March 3, 1988
$3000	June 14, 1988
$4000	December 1, 1988

To calculate the internal rate of return for this nonperiodic cash flow

```
cf = [−10000, 2500, 2000, 3000, 4000];
df = ['01/12/1987'
      '02/14/1988'
      '03/03/1988'
      '06/14/1988'
      '12/01/1988'];

yld = xirr(cf, df)
```

returns

```
yld =
    0.1006 (or 10.06%)
```

See Also

fvvar, irr, mirr, pvvar

Reference

Sharpe and Alexander, *Investments*, 4th edition, page 463.

year

Purpose

Year of date.

Syntax

```
y = year(d)
```

Description

`y = year(d)` returns the year of a serial date number or a date string `d`.

Example

```
y = year(728798.776)
```

or

```
y = year('19-may-1995')
```

returns

```
y =
        1995
```

See Also

`datevec`, `day`, `month`, `yeardays`

Purpose

Number of days in year.

Syntax

```
nd = yeardays(y)
```

Description

`nd = yeardays(y)` returns the actual number of days `nd` in the given year `y`. Enter `y` as a four-digit integer.

Example

```
nd = yeardays(2000)
nd =
   366
```

See Also

`days360`, `days365`, `daysact`, `year`, `yearfrac`

yearfrac

Purpose

Fraction of year between dates.

Syntax

```
f = yearfrac(d1, d2, basis)
f = yearfrac(d1, d2)
```

Description

`f = yearfrac(d1, d2, basis)` returns a fraction based on the number days between dates `d1` and `d2` using the given day-count basis. If `d2` is earlier than `d1`, `f` is negative. Enter dates as serial date numbers or date strings.

`basis` is the day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Examples

```
f = yearfrac('14 mar 97', '14 sep 97', 0)
f =
    0.5041

f = yearfrac('14 mar 97', '14 sep 97', 1)
f =
    0.5000
```

See Also

`days360`, `days365`, `daysact`, `daysdif`, `months`, `wrkdydif`, `yeardays`

Purpose

Yield to maturity of bond.

Syntax

```
yld = yldbond(sd, md, rv, price, cpn, per, basis, maxiter, eom)
yld = yldbond(sd, md, rv, price, cpn, per, basis, maxiter)
yld = yldbond(sd, md, rv, price, cpn, per, basis)
yld = yldbond(sd, md, rv, price, cpn, per)
yld = yldbond(sd, md, rv, price, cpn)
yld = yldbond(sd, md, rv, price)
```

Arguments

`sd` Settlement date. A vector of serial date numbers or date strings. `sd` must be earlier than or equal to `md`.

`md` Maturity date. A vector of serial date numbers or date strings.

`rv` Redemption or face value of the bond. A vector.

`price` Price of the bond. A vector.

`cpn` Coupon rate of the bond. A vector of decimal fractions. Default = `0`.

`per` Coupons per year of the bond. A vector of integers. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.

`basis` Day-count basis of the bond. A vector of integers.
`0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

`maxiter` Maximum number of iterations used in Newton's method to derive the yield to maturity. An integer. Default = `50`. Enter an empty matrix `[]` to accept the default and also enter values for `eom`.

`eom` End-of-month flag. A vector. This flag applies only when the maturity date `md` is an end-of-month date for a month having 30 or fewer days. `0` = ignore flag, meaning that a bond's coupon payment date is always the same day of the month. `1` = set flag (default), meaning that a bond's coupon payment date is always the last day of the month.

Vector arguments must have consistent dimensions, or they must be scalars.

yldbond

Description

`yld = yldbond(sd, md, rv, price, cpn, per, basis, maxiter, eom)` returns the yield to maturity of coupon and zero-coupon bonds using a Newton-Raphson iterative method. This function also applies to pure discount securities by setting `cpn = 0`. To calculate the spot rate of a security, set `per = 1`.

Example

Given data for three bonds with the same maturity date:

```
sd = '01/01/1960';
md = '01/01/1990';
rv = 1000;
price = [1276.76  1258.92  1197.43];
cpn = 0.08;
per = 2;
basis = 0;
maxiter = 30;
eom = 0;
```

Execute the function.

```
yld = yldbond(sd, md, rv, price, cpn, per, basis, maxiter, eom)
```

```
yld =
    0.0600    0.0611    0.0650
```

See Also

`acrubond`, `prbond`, `ylddisc`, `yldmat`, `yldoddf`, `yldoddfl`, `yldoddl`, `yldtbill` and functions for Term Structure of Interest Rates

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 5, 7.

Purpose

Yield of discounted security.

Syntax

```
y = ylddisc(sd, md, rv, price, basis)
y = ylddisc(sd, md, rv, price)
```

Arguments

`sd` Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.

`md` Maturity date. Enter as serial date number or date string.

`rv` Redemption (par, face) value.

`price` Discounted price of the security.

`basis` Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`y = ylddisc(sd, md, rv, price, basis)` finds the yield of a discounted security.

Example

Using the data

```
sd = '10/14/1988';
md = '03/17/1989';
rv = 100;
price = 96.28;
basis = 2;

y = ylddisc(sd, md, rv, price, basis)
```

returns

```
y =
    0.0903  (or 9.03%)
```

See Also

`acrudisc`, `prbond`, `prdisc`, `yldbond`, `yldmat`, `yldoddf`, `yldoddfl`, `yldoddl`, `yldtbill`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formula 1.

yldmat

Purpose

Yield with interest at maturity.

Syntax

```
y = yldmat(sd, md, id, rv, price, cpn, basis)
y = yldmat(sd, md, id, rv, price, cpn)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`id`	Issue date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`price`	Price of the security.
`cpn`	Coupon rate. Enter as decimal fraction.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`y = yldmat(sd, md, id, rv, price, cpn, basis)` returns the yield of a security paying interest at maturity.

Example

Using the data

```
sd = '02/07/1992';
md = '04/13/1992';
id = '10/11/1991';
rv = 100;
price = 99.98;
cpn = 0.0608;
basis = 1;

y = yldmat(sd, md, id, rv, price, cpn, basis)
```

returns

```
y =
    0.0607  (or 6.07%)
```

See Also

`acrubond`, `prbond`, `prmat`, `yldbond`, `ylddisc`, `yldoddf`, `yldoddfl`, `yldoddl`, `yldtbill`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formula 3.

yldoddf

Purpose

Yield of security with odd first period.

Syntax

```
yld = yldoddf(sd, md, id, fd, rv, price, cpn, per, basis, maxiter)
yld = yldoddf(sd, md, id, fd, rv, price, cpn, per, basis)
yld = yldoddf(sd, md, id, fd, rv, price, cpn, per)
yld = yldoddf(sd, md, id, fd, rv, price, cpn)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`id`	Issue date. Enter as serial date number or date string.
`fd`	First coupon date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`price`	Price of the security.
`cpn`	Coupon rate. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = 2.
`basis`	Day-count basis: 0 = actual/actual (default), 1 = 30/360, 2 = actual/360, 3 = actual/365.
`maxiter`	Number of iterations used by Newton's method to solve for `yld`. Default = 50.

Description

`yld = yldoddf(sd, md, id, fd, rv, price, cpn, per, basis, maxiter)` returns the yield of a security with an odd first period and the settlement date in the first period.

Example

Using the data:

```
sd = '11/11/1992';
md = '03/01/2005';
id = '10/15/1992';
fd = '03/01/1993';
rv = 100;
price = 113.60;
cpn = 0.0785;
per = 2;
basis = 0;

yld = yldoddf(sd, md, id, fd, rv, price, cpn, per, basis)
```

returns

```
yld =
    0.0625 (or 6.25%)
```

See Also

acrubond, cfdates, prbond, proddf, yldbond, yldmat, yldoddfl, yldoddl, yldtbill

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 8, 9.

yldoddfl

Purpose

Yield with odd first and last periods and settlement in first period.

Syntax

```
yld = yldoddfl(sd,md,id,fd,lcd,rv,price,cpn,per,basis,maxiter)
yld = yldoddfl(sd,md,id,fd,lcd,rv,price,cpn,per,basis)
yld = yldoddfl(sd,md,id,fd,lcd,rv,price,cpn,per)
yld = yldoddfl(sd,md,id,fd,lcd,rv,price,cpn)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`id`	Issue date. Enter as serial date number or date string.
`fd`	First coupon date. Enter as serial date number or date string.
`lcd`	Last coupon date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`price`	Price of the security.
`cpn`	Coupon rate. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`maxiter`	Number of iterations used by Newton's method to solve for `yld`. Default = `50`.

Description

`yld = yldoddfl(sd,md,id,fd,lcd,rv,price,cpn,per,basis,maxiter)` returns the yield of a security with odd first and last periods and the settlement date in the first period.

Example

Using this data:

```
sd = '03/15/1993';
md = '03/01/2020';
id = '03/01/1993';
fd = '07/01/1993';
lcd = '01/01/2020';
rv = 100;
price = 95.71;
cpn = 0.04;
per = 2;
basis = 1;

yld = yldoddfl(sd,md,id,fd,lcd,rv,price,cpn,per,basis)
```

returns

```
yld =
    0.0427  (or 4.27%)
```

See Also

`acrubond`, `cfdates`, `prbond`, `proddfl`, `yldbond`, `yldmat`, `yldoddf`, `yldoddl`, `yldtbill`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 16, 17, 18, 19.

yldoddl

Purpose

Yield with odd last period.

Syntax

```
yld = yldoddl(sd, md, lcd, rv, price, cpn, per, basis, maxiter)
yld = yldoddl(sd, md, lcd, rv, price, cpn, per, basis)
yld = yldoddl(sd, md, lcd, rv, price, cpn, per)
yld = yldoddl(sd, md, lcd, rv, price, cpn)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`lcd`	Last coupon date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`price`	Price of the security.
`cpn`	Coupon rate. Enter as decimal fraction.
`per`	Coupons per year. An integer. Default = `2`.
`basis`	Day-count basis: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
`maxiter`	Number of iterations used by Newton's method to solve for `yld`. Default = `50`.

Description

`yld = yldoddl(sd, md, lcd, rv, price, cpn, per, basis, maxiter)` returns the yield of a security with odd last period.

Example

Using this data:

```
sd = '02/07/1993';
md = '06/15/1993';
lcd = '10/15/1992';
rv = 100;
price = 99.878;
cpn = 0.0375;
per = 2;
basis = 1;

yld = yldoddl(sd, md, lcd, rv, price, cpn, per, basis)
```

returns

```
yld =
    0.0411  (or 4.11%)
```

See Also

`acrubond`, `cfdates`, `proddl`, `yldbond`, `yldmat`, `yldoddf`, `yldoddfl`, `yldtbill`

Reference

Mayle, *Standard Securities Calculation Methods*, Volumes I-II, 3rd edition. Formulas 10, 12, 14, 15.

yldtbill

Purpose

Yield of Treasury bill.

Syntax

```
y = yldtbill(sd, md, rv, price)
```

Arguments

`sd`	Settlement date. Enter as serial date number or date string. `sd` must be earlier than or equal to `md`.
`md`	Maturity date. Enter as serial date number or date string.
`rv`	Redemption (par, face) value.
`price`	Price of the Treasury bill.

Description

`y = yldtbill(sd, md, rv, price)` returns the yield for a Treasury bill.

Example

The settlement date of a Treasury bill is February 10, 1992, the maturity date is August 6, 1992, the par value is $1000, and the price is $981.36. Using this data:

```
y = yldtbill('2/10/1992', '8/6/1992', 1000, 981.36)
```

returns

```
y =
    0.0384  (or 3.84%)
```

See Also

`beytbill, prtbill, yldbond, yldmat, yldoddf, yldoddfl, yldoddl`

Reference

Bodie, Kane, and Marcus, *Investments*, pages 41-43.

Purpose

Zero curve from coupon bond prices, using bootstrap method.

Syntax

```
[zr, cd] = zbtprice(bonds, p, sd, ocomp, obasis, maxiter)
[zr, cd] = zbtprice(bonds, p, sd, ocomp, obasis)
[zr, cd] = zbtprice(bonds, p, sd, ocomp)
[zr, cd] = zbtprice(bonds, p, sd)
```

Arguments

`bonds` Coupon bond information used to generate the zero curve. An N-by-2 to N-by-6 matrix where each row describes a bond. N is the number of bonds. The first two columns are required; the rest are optional but must be added in order. All rows in `bonds` must have the same number of columns.

Columns are `[md cpn rv per basis eom]` where

- `md` Maturity date of the bond, as a serial date number. Use `datenum` to convert date strings to serial date numbers.
- `cpn` Coupon rate of the bond, as a decimal fraction.
- `rv` Redemption or face value of the bond. Default = `100`.
- `per` Coupons per year of the bond, as an integer. Allowed values are `1`, `2`, `3`, `4`, `6`, and `12`. Default = `2`.
- `basis` Day-count basis of the bond: `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.
- `eom` End-of-month flag. This flag applies only when the maturity date `md` is an end-of-month date for a month having 30 or fewer days. `0` = ignore flag, meaning that a bond's coupon payment date is always the same day of the month. `1` = set flag (default), meaning that a bond's coupon payment date is always the last day of the month.

`p` Prices. An N-by-1 vector containing the clean price (the price without accrued interest) of each bond in `bonds`, respectively. The number of rows (N) must match the number of rows in `bonds`.

`sd` Settlement date, as a scalar serial date number. This represents time zero for deriving the zero curve, and it is normally the common settlement date for all the bonds.

`ocomp` Output compounding. A scalar that sets the compounding frequency per year for the output zero rates in `zr`. Allowed values are:

- `1` = annual compounding
- `2` = semi-annual compounding (default)
- `3` = compounding three times per year
- `4` = quarterly compounding
- `6` = bimonthly compounding
- `12` = monthly compounding

`obasis` Output day-count basis for mapping cash-flow dates to years, in generating the output zero rates in `zr`. A scalar.
`0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

`maxiter` Maximum number of iterations for deriving the zero rates in `zr`. A scalar. Default = `50`. A value greater than 50 may slow processing.

Description

`[zr, cd] = zbtprice(bonds, p, sd, ocomp, obasis, maxiter)` uses the bootstrap method to return a zero curve given a portfolio of coupon bonds and their prices. A zero curve consists of the yields to maturity for a portfolio of theoretical zero-coupon bonds that are derived from the input `bonds` portfolio. The bootstrap method that this function uses does *not* require alignment among the cash-flow dates of the bonds in the input portfolio. It uses theoretical par bond arbitrage and yield interpolation to derive all zero rates. For best results, use a portfolio of at least 30 bonds evenly spaced across the investment horizon.

`zr` Zero rates. An M-by-1 vector of decimal fractions that are the implied zero rates for each point along the investment horizon represented by `cd`. In aggregate, the rates in `zr` constitute a zero curve.

If more than one bond has the same maturity date, `zbtprice` returns the mean zero rate for that maturity.

`cd` Curve dates. An M-by-1 vector of unique maturity dates (as serial date numbers) that correspond to the zero rates in `zr`. These dates begin with the earliest maturity date and end with the latest maturity date `md` in the `bonds` matrix. Use `datestr` to convert serial date numbers to date strings.

Example

Given data and prices for 12 coupon bonds, two with the same maturity date; and given the common settlement date:

```
bonds = [datenum('6/1/1998')   0.0475   100  2  0  0;
         datenum('7/1/2000')   0.06     100  2  0  0;
         datenum('7/1/2000')   0.09375  100  6  1  0;
         datenum('6/30/2001')  0.05125  100  1  3  1;
         datenum('4/15/2002')  0.07125  100  4  1  0;
         datenum('1/15/2000')  0.065    100  2  0  0;
         datenum('9/1/1999')   0.08     100  3  3  0;
         datenum('4/30/2001')  0.05875  100  2  0  0;
         datenum('11/15/1999') 0.07125  100  2  0  0;
         datenum('6/30/2000')  0.07     100  2  3  1;
         datenum('7/1/2001')   0.0525   100  2  3  0;
         datenum('4/30/2002')  0.07     100  2  0  0];

p = [ 99.375;
       99.875;
      105.75 ;
       96.875;
      103.625;
      101.125;
      103.125;
       99.375;
      101.0  ;
      101.25 ;
       96.375;
      102.75 ];

sd = datenum('12/18/1997');
```

Set semi-annual compounding for the zero curve, on an actual/365 basis. Derive the zero curve within 50 iterations.

```
ocomp = 2;
obasis = 3;
maxiter = 50;
```

Execute the function

```
[zr, cd] = zbtprice(bonds, p, sd, ocomp, obasis, maxiter)
```

which returns the zero curve `zr` at the maturity dates `cd`. Note the mean zero rate for the two bonds with the same maturity date*.

```
zr =
    0.0616
    0.0580
    0.0658
    0.0591
    0.0649
    0.0657*
    0.0605
    0.0601
    0.0643
    0.0584
    0.0627
cd =
       729907    (serial date number for 01-Jun-1998)
       730364    (01-Sep-1999)
       730439    (15-Nov-1999)
       730500    (15-Jan-2000)
       730667    (30-Jun-2000)
       730668    (01-Jul-2000)*
       730971    (30-Apr-2001)
       731032    (30-Jun-2001)
       731033    (01-Jul-2001)
       731321    (15-Apr-2002)
       731336    (30-Apr-2002)
```

See Also

`zbtyield` and other functions for Term Structure of Interest Rates

References

Fabozzi, Frank J. "The Structure of Interest Rates." Ch. 6 in Fabozzi, Frank J. and T. Dessa Fabozzi, eds. *The Handbook of Fixed Income Securities.* 4th ed. New York: Irwin Professional Publishing. 1995.

McEnally, Richard W. and James V. Jordan. "The Term Structure of Interest Rates." Ch. 37 in Fabozzi and Fabozzi, ibid.

Das, Satyajit. "Calculating Zero Coupon Rates." *Swap and Derivative Financing.* Appendix to Ch. 8, pp. 219-225. New York: Irwin Professional Publishing. 1994.

Purpose

Zero curve from coupon bond yields, using bootstrap method.

Syntax

```
[zr, cd] = zbtyield(bonds, y, sd, ocomp, obasis, maxiter)
[zr, cd] = zbtyield(bonds, y, sd, ocomp, obasis)
[zr, cd] = zbtyield(bonds, y, sd, ocomp)
[zr, cd] = zbtyield(bonds, y, sd)
```

Arguments

`bonds` Coupon bond information used to generate the zero curve. An N-by-2 to N-by-6 matrix where each row describes a bond. N is the number of bonds. The first two columns are required; the rest are optional but must be added in order. All rows in `bonds` must have the same number of columns.

Columns are `[md cpn rv per basis eom]` where

- `md` Maturity date of the bond, as a serial date number. Use `datenum` to convert date strings to serial date numbers.
- `cpn` Coupon rate of the bond, as a decimal fraction.
- `rv` Redemption or face value of the bond. Default = 100.
- `per` Coupons per year of the bond, as an integer. Allowed values are 1, 2, 3, 4, 6, and 12. Default = 2.
- `basis` Day-count basis of the bond: 0 = actual/actual (default), 1 = 30/360, 2 = actual/360, 3 = actual/365.
- `eom` End-of-month flag. This flag applies only when the maturity date `md` is an end-of-month date for a month having 30 or fewer days. 0 = ignore flag, meaning that a bond's coupon payment date is always the same day of the month. 1 = set flag (default), meaning that a bond's coupon payment date is always the last day of the month.

`y` Yields. An N-by-1 vector containing the yield to maturity of each bond in `bonds`, respectively. The number of rows (N) must match the number of rows in `bonds`.

`sd` Settlement date, as a scalar serial date number. This represents time zero for deriving the zero curve, and it is normally the common settlement date for all the bonds.

`ocomp` Output compounding. A scalar that sets the compounding frequency per year for the output zero rates in `zr`. Allowed values are:

- `1` = annual compounding
- `2` = semi-annual compounding (default)
- `3` = compounding three times per year
- `4` = quarterly compounding
- `6` = bimonthly compounding
- `12` = monthly compounding

`obasis` Output day-count basis for mapping cash-flow dates to years, in generating the output zero rates in `zr`. A scalar.
`0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

`maxiter` Maximum number of iterations for deriving the zero rates in `zr`. A scalar. Default = `50`. A value greater than 50 may slow processing.

Description

`[zr, cd] = zbtyield(bonds, y, sd, ocomp, obasis, maxiter)` uses the bootstrap method to return a zero curve given a portfolio of coupon bonds and their yields. A zero curve consists of the yields to maturity for a portfolio of theoretical zero-coupon bonds that are derived from the input `bonds` portfolio. The bootstrap method that this function uses does *not* require alignment among the cash-flow dates of the bonds in the input portfolio. It uses theoretical par bond arbitrage and yield interpolation to derive all zero rates. For best results, use a portfolio of at least 30 bonds evenly spaced across the investment horizon.

`zr` Zero rates. An M-by-1 vector of decimal fractions that are the implied zero rates for each point along the investment horizon represented by `cd`. In aggregate, the rates in `zr` constitute a zero curve.

If more than one bond has the same maturity date, `zbtyield` returns the mean zero rate for that maturity.

`cd` Curve dates. An M-by-1 vector of unique maturity dates (as serial date numbers) that correspond to the zero rates in `zr`. These dates begin with the earliest maturity date and end with the latest maturity date `md` in the `bonds` matrix. Use `datestr` to convert serial date numbers to date strings.

Example

Given data and yields to maturity for 12 coupon bonds, two with the same maturity date; and given the common settlement date:

```
bonds = [datenum('6/1/1998')   0.0475   100  2  0  0;
         datenum('7/1/2000')   0.06     100  2  0  0;
         datenum('7/1/2000')   0.09375  100  6  1  0;
         datenum('6/30/2001')  0.05125  100  1  3  1;
         datenum('4/15/2002')  0.07125  100  4  1  0;
         datenum('1/15/2000')  0.065    100  2  0  0;
         datenum('9/1/1999')   0.08     100  3  3  0;
         datenum('4/30/2001')  0.05875  100  2  0  0;
         datenum('11/15/1999') 0.07125  100  2  0  0;
         datenum('6/30/2000')  0.07     100  2  3  1;
         datenum('7/1/2001')   0.0525   100  2  3  0;
         datenum('4/30/2002')  0.07     100  2  0  0];

y = [0.048;
     0.06 ;
     0.089;
     0.053;
     0.069;
     0.064;
     0.078;
     0.059;
     0.071;
     0.069;
     0.057;
     0.068];

sd = datenum('12/18/1997');
```

Set semi-annual compounding for the zero curve, on an actual/365 basis. Derive the zero curve within 50 iterations.

```
ocomp = 2;
obasis = 3;
maxiter = 50;
```

Execute the function

```
[zr, cd] = zbtyield(bonds, y, sd, ocomp, obasis, maxiter)
```

which returns the zero curve `zr` at the maturity dates `cd`. Note the mean zero rate for the two bonds with the same maturity date*.

```
zr =
    0.0480
    0.0577
    0.0909
    0.0529
    0.0699
    0.0724*
    0.0584
    0.0716
    0.0696
    0.0526
    0.0687
cd =
       729907    (serial date number for 01-Jun-1998)
       730364    (01-Sep-1999)
       730439    (15-Nov-1999)
       730500    (15-Jan-2000)
       730667    (30-Jun-2000)
       730668    (01-Jul-2000)*
       730971    (30-Apr-2001)
       731032    (30-Jun-2001)
       731033    (01-Jul-2001)
       731321    (15-Apr-2002)
       731336    (30-Apr-2002)
```

See Also

`zbtprice` and other functions for Term Structure of Interest Rates

References

Fabozzi, Frank J. “The Structure of Interest Rates.” Ch. 6 in Fabozzi, Frank J. and T. Dessa Fabozzi, eds. *The Handbook of Fixed Income Securities.* 4th ed. New York: Irwin Professional Publishing. 1995.

McEnally, Richard W. and James V. Jordan. “The Term Structure of Interest Rates.” Ch. 37 in Fabozzi and Fabozzi, ibid.

Das, Satyajit. “Calculating Zero Coupon Rates.” *Swap and Derivative Financing.* Appendix to Ch. 8, pp. 219-225. New York: Irwin Professional Publishing. 1994.

Purpose

Discount curve given a zero curve.

Syntax

```
[dr, cd] = zero2disc(zr, cd, sd, icomp, ibasis)
[dr, cd] = zero2disc(zr, cd, sd, icomp)
[dr, cd] = zero2disc(zr, cd, sd)
```

Arguments

`zr` Zero rates. An N-by-1 vector of annualized zero rates, as decimal fractions. In aggregate, the rates in `zr` constitute an implied zero curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the zero rates in `zr`. Use `datenum` to convert date strings to serial date numbers.

`sd` Settlement date. A serial date number that is the common settlement date for the zero rates in `zr`; i.e., the settlement date for the bonds from which the zero curve was bootstrapped.

`icomp` Input compounding. A scalar that indicates the compounding frequency per year used for annualizing the input zero rates in `zr`. Allowed values are:

- `1` = annual compounding
- `2` = semi-annual compounding (default)
- `3` = compounding three times per year
- `4` = quarterly compounding
- `6` = bimonthly compounding
- `12` = monthly compounding
- `365` = daily compounding
- `–1` = continuous compounding

`ibasis` Input day-count basis used for annualizing the input zero rates in `zr`. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

Description

`[dr, cd] = zero2disc(zr, cd, sd, icomp, ibasis)` returns a discount curve given a zero curve and its maturity dates.

`dr` Discount factors. An N-by-1 vector of discount factors, as decimal fractions. In aggregate, the factors in `dr` constitute a discount curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the discount rates in `dr`. This vector is the same as the input vector `cd`. Use `datestr` to convert serial date numbers to date strings.

Example

Given a zero curve `zr` over a set of maturity dates `cd`, and a settlement date `sd`:

```
zr = [0.0464
      0.0509
      0.0524
      0.0525
      0.0531
      0.0525
      0.0530
      0.0531
      0.0549
      0.0536];

cd = [datenum('06-Nov-1997')
      datenum('11-Dec-1997')
      datenum('15-Jan-1998')
      datenum('05-Feb-1998')
      datenum('04-Mar-1998')
      datenum('02-Apr-1998')
      datenum('30-Apr-1998')
      datenum('25-Jun-1998')
      datenum('04-Sep-1998')
      datenum('12-Nov-1998')];

sd = datenum('03-Nov-1997');
```

The zero curve was compounded daily on an actual/365 basis.

```
icomp = 365;
ibasis = 3;
```

Execute the function:

```
[dr, cd] = zero2disc(zr, cd, sd, icomp, ibasis)
```

which returns the discount curve dr at the maturity dates cd:

```
dr =
    0.9996
    0.9947
    0.9896
    0.9866
    0.9826
    0.9787
    0.9745
    0.9665
    0.9552
    0.9466
cd =
      729700
      729735
      729770
      729791
      729818
      729847
      729875
      729931
      730002
      730071
```

(For readability, zr and dr are shown here only to the basis point. However, MATLAB computed them at full precision. If you enter zr as shown, dr may differ due to rounding.)

See Also

disc2zero and other functions for Term Structure of Interest Rates

zero2fwd

Purpose

Forward curve given a zero curve.

Syntax

```
[fr, cd] = zero2fwd(zr, cd, sd, ocomp, obasis, icomp, ibasis)
[fr, cd] = zero2fwd(zr, cd, sd, ocomp, obasis, icomp)
[fr, cd] = zero2fwd(zr, cd, sd, ocomp, obasis)
[fr, cd] = zero2fwd(zr, cd, sd, ocomp)
[fr, cd] = zero2fwd(zr, cd, sd)
```

Arguments

`zr` Zero rates. An N-by-1 vector of annualized zero rates, as decimal fractions. In aggregate, the rates in `zr` constitute an implied zero curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the zero rates in `zr`. Use `datenum` to convert date strings to serial date numbers.

`sd` Settlement date. A serial date number that is the common settlement date for the zero rates in `zr`.

`ocomp` Output compounding. A scalar that sets the compounding frequency per year for annualizing the output forward rates in `fr`. Allowed values are:

- `1` = annual compounding
- `2` = semi-annual compounding (default)
- `3` = compounding three times per year
- `4` = quarterly compounding
- `6` = bimonthly compounding
- `12` = monthly compounding
- `365` = daily compounding
- `–1` = continuous compounding

`obasis` Output day-count basis for annualizing the forward rates in `fr`. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

`icomp` Input compounding. A scalar that indicates the compounding frequency per year used for annualizing the input zero rates in `zr`. Allowed values are the same as for `ocomp`. Default = `ocomp`.

`ibasis` Input day-count basis used for annualizing the input zero rates in `zr`. Allowed values are the same as for `obasis`. Default = `obasis`.

Description

`[fr, cd] = zero2fwd(zr, cd, sd, ocomp, obasis, icomp, ibasis)` returns an implied forward rate curve given a zero curve and its maturity dates.

`fr` Forward rates. An N-by-1 vector of decimal fractions. In aggregate, the rates in `fr` constitute a forward curve over the dates in `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the forward rates in `fr`. This vector is the same as the input vector `cd`. Use `datestr` to convert serial date numbers to date strings.

Example

Given a zero curve `zr` over a set of maturity dates `cd`, and a settlement date `sd`:

```
zr = [0.0458
      0.0502
      0.0518
      0.0519
      0.0524
      0.0519
      0.0523
      0.0525
      0.0541
      0.0529];

cd = [datenum('06-Nov-1997')
      datenum('11-Dec-1997')
      datenum('15-Jan-1998')
      datenum('05-Feb-1998')
      datenum('04-Mar-1998')
      datenum('02-Apr-1998')
      datenum('30-Apr-1998')
      datenum('25-Jun-1998')
      datenum('04-Sep-1998')
      datenum('12-Nov-1998')];

sd = datenum('03-Nov-1997');
```

Set annual compounding for the forward curve, on an actual/actual basis. The zero curve was compounded daily on an actual/365 basis.

```
ocomp = 1;
obasis = 0;
icomp = 365;
ibasis = 3;
```

Execute the function

```
[fr, cd] = zero2fwd(zr, cd, sd, ocomp, obasis, icomp, ibasis)
```

which returns the forward rate curve `fr` at the maturity dates `cd`:

```
fr =
    0.0469
    0.0519
    0.0550
    0.0536
    0.0556
    0.0511
    0.0559
    0.0546
    0.0612
    0.0487
cd =
       729700
       729735
       729770
       729791
       729818
       729847
       729875
       729931
       730002
       730071
```

(For readability, `zr` and `fr` are shown here only to the basis point. However, MATLAB computed them at full precision. If you enter `zr` as shown, `fr` may differ due to rounding.)

See Also

`fwd2zero` and other functions for Term Structure of Interest Rates

Purpose

Par yield curve given a zero curve.

Syntax

```
[pr, cd] = zero2pyld(zr, cd, sd, ocomp, obasis, icomp, ibasis)
[pr, cd] = zero2pyld(zr, cd, sd, ocomp, obasis, icomp)
[pr, cd] = zero2pyld(zr, cd, sd, ocomp, obasis)
[pr, cd] = zero2pyld(zr, cd, sd, ocomp)
[pr, cd] = zero2pyld(zr, cd, sd)
```

Arguments

`zr` Zero rates. An N-by-1 vector of annualized zero rates, as decimal fractions. In aggregate, the rates in `zr` constitute an implied zero curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the zero rates in `zr`. Use `datenum` to convert date strings to serial date numbers.

`sd` Settlement date. A serial date number that is the common settlement date for the zero rates in `zr`.

`ocomp` Output compounding. A scalar that sets the compounding (coupon) frequency per year for annualizing the output par yield rates in `pr`. Allowed values are:

- `1` = annual compounding or one payment per year
- `2` = semi-annual compounding (default)
- `3` = compounding three times per year
- `4` = quarterly compounding
- `6` = bimonthly compounding
- `12` = monthly compounding

`obasis` Output day-count basis for annualizing the par yield rates in `pr`. `0` = actual/actual (default), `1` = 30/360, `2` = actual/360, `3` = actual/365.

`icomp` Input compounding. A scalar that indicates the compounding frequency per year used for annualizing the input zero rates in `zr`. Allowed values are the same as for `ocomp`. Default = `ocomp`.

`ibasis` Input day-count basis used for annualizing the input zero rates in `zr`. Allowed values are the same as for `obasis`. Default = `obasis`.

Description

`[pr, cd] = zero2pyld(zr, cd, sd, ocomp, obasis, icomp, ibasis)` returns a par yield curve given a zero curve and its maturity dates.

`pr` Par yield rates. An N-by-1 vector of annualized par yields, as decimal fractions. (Par yields = coupon rates.) In aggregate, the yield rates in `pr` constitute a par yield curve for the investment horizon represented by `cd`.

`cd` Curve dates. An N-by-1 vector of maturity dates (as serial date numbers) that correspond to the par yield rates in `pr`. This vector is the same as the input vector `cd`. Use `datestr` to convert serial date numbers to date strings.

Example

Given a zero curve `zr` over a set of maturity dates `cd`, and a settlement date `sd`:

```
zr = [0.0457
      0.0487
      0.0506
      0.0507
      0.0505
      0.0504
      0.0506
      0.0516
      0.0539
      0.0530];

cd = [datenum('06-Nov-1997')
      datenum('11-Dec-1997')
      datenum('15-Jan-1998')
      datenum('05-Feb-1998')
      datenum('04-Mar-1998')
      datenum('02-Apr-1998')
      datenum('30-Apr-1998')
      datenum('25-Jun-1998')
      datenum('04-Sep-1998')
      datenum('12-Nov-1998')];

sd = datenum('03-Nov-1997');
```

Set annual compounding for the par yield curve, on an actual/actual basis. The zero curve was compounded monthly, on an actual/365 basis.

```
ocomp = 1;
obasis = 0;
icomp = 12;
ibasis = 3;
```

Execute the function

```
[pr, cd] = zero2pyld(zr, cd, sd, ocomp, obasis, icomp, ibasis)
```

which returns the par yield curve `pr` at the maturity dates `cd`:

```
pr =
    0.0478
    0.0509
    0.0529
    0.0529
    0.0526
    0.0524
    0.0525
    0.0534
    0.0555
    0.0543
cd =
       729700
       729735
       729770
       729791
       729818
       729847
       729875
       729931
       730002
       730071
```

(For readability, `zr` and `pr` are shown only to the basis point. However, MATLAB computed them at full precision. If you enter `zr` as shown, `pr` may differ due to rounding.)

See Also

`pyld2zero` and other functions for Term Structure of Interest Rates

A

Glossary

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

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

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

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

Basis point - One hundredth of one percentage point, or 0.0001.

Beta - The price volatility of a financial instrument relative to the price volatility of a market or index as a whole. Beta is most commonly used with respect to equities. A high-beta instrument is riskier (and thus usually carries a premium price) than a low-beta instrument.

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

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

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

Bootstrapping, bootstrap method - An arithmetic method for backing an implied zero curve out of the par yield curve.

Building a binomial tree - For a binomial option model: plotting the two possible short-term price-changes values, and then the subsequent two values each, and then the subsequent two values each, and so on over time, is known as “building a binomial tree.” See Binomial model.

Call - a. An option to buy a certain quantity of a stock or commodity for a specified price within a specified time. See Put. **b.** A demand to submit bonds to the issuer for redemption before the maturity date. **c.** A demand for payment of a debt. **d.** A demand for payment due on stock bought on margin when the value has shrunk.

Callable bond - A bond that allows the issuer to buy back the bond at a predetermined price at specified future dates. The bond contains an embedded call option; i.e., the holder has sold a call option to the issuer. See Puttable bond.

Candlestick chart - A financial chart usually used to plot the high, low, open, and close price of a security over time. The body of the "candle" is the region between the open and close price of the security. Thin vertical lines extend up to the high and down to the low, respectively. If the open price is greater than the close price, the body is empty. If the close price is greater than the open price, the body is filled. See also High-low-close chart.

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

Cash flow - Cash received and paid over time.

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.

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 semi-annually. 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 semi-annually.

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

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

Delta - The rate of change of the price of a derivative security relative to the price of the underlying asset; i.e., the first derivative of the curve that relates the price of the derivative to the price of the underlying security.

Depreciation - Reduction in value of fixed or tangible assets over some period for accounting purposes. See Amortization.

Derivative - A financial instrument that is based on some underlying asset. For example, an option is a derivative instrument based on the right to buy or sell an underlying instrument.

Discount curve - The curve of discount rates vs. maturity dates for bonds.

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

Efficient frontier - A graph representing a set of portfolios that maximizes expected return at each level of portfolio risk. See Markowitz model.

Elasticity - See Lambda.

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

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

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

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

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

Forward curve - The curve of forward interest rates vs. maturity dates for bonds.

Forward rate - The future interest rate of a bond inferred from the term structure, especially from the yield curve of zero-coupon bonds, calculated from the growth factor of an investment in a zero held until maturity.

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

Gamma - The rate of change of delta for a derivative security relative to the price of the underlying asset; i.e., the second derivative of the option price relative to the security price.

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

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

High-low-close chart - A financial chart usually used to plot the high, low, open, and close price of a security over time. Plots are vertical lines whose top is the high, bottom is the low, open is a short horizontal tick to the left, and close is a short horizontal tick to the right.

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

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

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

Ito process - Statistical assumptions about the behavior of security prices. For details, see the book by Hull listed in the *Bibliography*.

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.

Long position - Outright ownership of a security or financial instrument. The owner expects the price to rise in order to make a profit on some future sale.

Long rate - The yield on a zero-coupon Treasury bond.

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

Markowitz model - A model for selecting an optimum investment portfolio, devised by H. M. Markowitz. It uses a discrete-time, continuous-outcome approach for modeling investment problems, often called the mean-variance paradigm. See Efficient frontier.

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

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

Modified duration - The Macaulay duration discounted by the per-period interest rate; i.e., divided by (1+rate/frequency). Modified duration is also known as volatility.

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

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

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

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

Odd first or last period - Fixed-income securities may be purchased on dates that do not coincide with coupon or payment dates. The length of the first and

last periods may differ from the regular period between coupons, and thus the bond owner is not entitled to the full value of the coupon for that period. Instead, the coupon is pro-rated according to how long the bond is held during that period.

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.

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.

Point and figure chart - A financial chart usually used to plot asset price data. Upward price movements are plotted as X's and downward price movements are plotted as O's.

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

Principal value - See Par value.

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

Put - An option to sell a stipulated amount of stock or securities within a specified time and at a fixed exercise price. See Call.

Puttable bond - A bond that allows the holder to redeem the bond at a predetermined price at specified future dates. The bond contains an embedded put option; i.e., the holder has bought a put option. See Callable bond.

Quant - A quantitative analyst; someone who does numerical analysis of financial information in order to detect relationships, disparities, or patterns that can lead to making money.

Redemption value - See Par value.

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

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

Sensitivity - The "what if" relationship between variables; the degree to which changes in one variable cause changes in another variable. A specific synonym is volatility.

Settlement date - The date when money first changes hands; i.e., when a buyer actually pays for a security. It need not coincide with the issue date.

Short rate - The annualized one-period interest rate.

Short sale, short position - The sale of a security or financial instrument not owned, in anticipation of a price decline and making a profit by purchasing the instrument later at a lower price, and then delivering the instrument to complete the sale. See Long position.

Spot curve, spot yield curve - See Zero curve.

Spot rate - The current interest rate appropriate for discounting a cash flow of some given maturity.

Spread - For options, a combination of call or put options on the same stock with differing exercise prices or maturity dates.

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

Stochastic - Involving or containing a random variable or variables; involving chance or probability.

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

Strike - Exercise a put or call option.

Strike price - See Exercise price.

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

Swaption - A swap option; an option on an interest-rate swap. The option gives the holder the right to enter into a contracted interest-rate swap at a specified future date. See Swap.

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

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

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

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

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

Vega - The rate of change in the price of a derivative security relative to the volatility of the underlying security. When vega is large the security is sensitive to small changes in volatility.

Volatility - a. Another general term for sensitivity. **b.** Another term for the specific sensitivity measure called modified duration. **c.** The standard deviation of the annualized continuously compounded rate of return of an asset.

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

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

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

Zero curve, zero-coupon yield curve - A yield curve for zero-coupon bonds; zero rates versus maturity dates. Since the maturity and duration (Macaulay duration) are identical for zeros, the zero curve is a pure depiction of supply/ demand conditions for loanable funds across a continuum of durations and maturities. Also known as spot curve or spot yield curve.

Zero-coupon bond, or Zero - A bond that, instead of carrying a coupon, is sold at a discount from its face value, pays no interest during its life, and pays the principal only at maturity.

B

Bibliography

For the well-known algorithms and formulas used in the Financial Toolbox (such as how to compute a loan payment given principal, interest rate, and length of the loan), no references are given here. The references here pertain to less common formulas.

The *Reference* chapter and the online help for each function also cite specific references when appropriate.

Bond Pricing and Yields

The pricing and yield formulas for fixed-income securities come from:

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.

In many cases these formulas compute the price of a security given yield, dates, rates, and other data. These formulas are nonlinear, however; so when solving for an independent variable within a formula, the Financial Toolbox uses Newton's method. See any elementary numerical methods textbook for the mathematics underlying Newton's method.

Term Structure of Interest Rates

The formulas and methodology for term structure functions come from:

Fabozzi, Frank J. "The Structure of Interest Rates." Ch. 6 in Fabozzi, Frank J. and T. Dessa Fabozzi, eds. *The Handbook of Fixed Income Securities.* 4th ed. New York: Irwin Professional Publishing. 1995. ISBN 0-7863-0001-9.

McEnally, Richard W. and James V. Jordan. "The Term Structure of Interest Rates." Ch. 37 in Fabozzi and Fabozzi, ibid.

Das, Satyajit. "Calculating Zero Coupon Rates." *Swap and Derivative Financing.* Appendix to Ch. 8, pp. 219-225. New York: Irwin Professional Publishing. 1994. ISBN 1-55738-542-4.

Derivatives Pricing and Yields

The pricing and yield formulas for derivative securities come from:

Hull, John, C. *Options, Futures, and Other Derivative Securities.* Englewood Cliffs, NJ: Prentice-Hall. 2nd ed., 1993, ISBN 0-13-639014-5.

Portfolio Analysis

The Markowitz model is used for portfolio analysis computations. For a discussion of this model see Chapter 7 of:

Bodie, Zvi, Alex Kane, and Alan J. Marcus. *Investments*. Burr Ridge, IL: Irwin. 2nd. ed., 1993, ISBN 0-256-08342-8.

To solve the quadratic minimization problem associated with finding the efficient frontier, the toolbox uses the `constr` function (finds the constrained minimum of a function of several variables) in the MATLAB Optimization Toolbox. Please see that toolbox documentation for more details.

Other References

Other references include:

Addendum to Securities Industry Association, *Standard Securities Calculation Methods: Fixed Income Securities Formulas for Analytic Measures*, Vol. 2, Spring 1995. This addendum explains and clarifies the end-of-month rule.

Brealey, Richard A., and Stewart C. Myers. *Principles of Corporate Finance.* New York: McGraw-Hill. 4th ed., 1991, ISBN 0-07-007405-4.

Daigler, Robert T. *Advanced Options Trading.* Chicago: Probus Publishing Co. 1994, ISBN 1-55738-552-1.

A Dictionary of Finance. Oxford: Oxford University Press. 1993, ISBN 0-19-285279-5.

Fabozzi, Frank J., and T. Dessa Fabozzi, eds. *The Handbook of Fixed-Income Securities.* Burr Ridge, IL: Irwin. 4th ed., 1995, ISBN 0-7863-0001-9.

Fitch, Thomas P. *Dictionary of Banking Terms.* Hauppauge, NY: Barron's. 2nd ed., 1993, ISBN 0-8120-1530-4.

Hill, Richard O., Jr. *Elementary Linear Algebra.* Orlando, FL: Academic Press. 1986, ISBN 0-12-348460-X

Marshall, John F., and Vipul K. Bansal. *Financial Engineering: A Complete Guide to Financial Innovation*. New York: New York Institute of Finance. 1992, ISBN 0-13-312588-2.

Sharpe, William F. *Macro-Investment Analysis*. An "electronic work-in-progress" published on the World Wide Web, 1995, at `http://www-sharpe.stanford.edu/mia.htm`.

Sharpe, William F., and Gordon J. Alexander. *Investments*. Englewood Cliffs, NJ: Prentice-Hall. 4th ed., 1990, ISBN 0-13-504382-4.

Stigum, Marcia, with John Mann. *Money Market Calculations: Yields, Break-Evens, and Arbitrage*. Burr Ridge, IL: Irwin. 1981, ISBN 0-87094-192-5.

Index

Numerics

A

B

C

D

E

F

G

H

I

L

M

N

Q

R

S

T

U

V

W

X

Y

Z