Financial Time Series Toolbox

For Use with MATLAB®

- Computation
- Visualization
- Programming

User's Guide

Version 2

How to Contact The MathWorks:

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

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

508-647-7000 Phone

508-647-7001 Fax

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

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

Financial Time Series Toolbox User's Guide

Printing History:

July 1999	First printing	New for Version 1.0
December 2000	Online only	Updated for Version 1.1
May 2001	Second printing	For Version 1.1
May 2002	Third printing	For Version 2.0
June 2004	Online only	For Release 14

Contents

1 Getting Started

2 Using Financial Time Series

3 Technical Analysis

4 Graphical User Interface

5 Function Reference

Index

1

Getting Started

What is the Financial Time Series Toolbox?

The Financial Time Series Toolbox for MATLAB® is a collection of tools for the analysis of time series data in the financial markets. The toolbox contains a financial time series object constructor and several methods that operate on and analyze the object. Financial engineers working with time series data, such as equity prices or daily interest fluctuations, can use this toolbox for more intuitive data management than by using regular vectors or matrices.

This chapter discusses how to create a financial time series object in one of two ways:

- "Using the Constructor" on page 1-3
- "Transforming a Text File" on page 1-14

The chapter also discusses `chartfts`, a graphical tool for visualizing financial time series objects. You can find this discussion in "Visualizing Financial Time Series Objects" on page 1-17.

Creating Financial Time Series Objects

The Financial Time Series Toolbox provides two ways to create a financial time series object:

- At the command line using the object constructor `fints`
- From a text data file through the function `ascii2fts`

The structure of the object minimally consists of a description field, a frequency indicator field, the date vector field, and at least one data series vector. The names for the fields are fixed for the first three fields: `desc`, `freq`, and `dates`. You can specify names of your choice for any data series vectors. If you do not specify names, the object uses the default names `series1`, `series2`, `series3`, etc.

If time-of-day information is incorporated in the date vector, the object contains an additional field named `times`.

Using the Constructor

The object constructor function `fints` has five different syntaxes. These forms exist to simplify object construction. The syntaxes vary according to the types of input arguments presented to the constructor. The syntaxes are

- Single Matrix Input
  - See “Time-of-Day Information Excluded” on page 1-4.
  - See “Time-of-Day Information Included” on page 1-7.
- Separate Vector Input
  - See “Time-of-Day Information Excluded” on page 1-7.
  - See “Time-of-Day Information Included” on page 1-8.
- See “Data Name Input” on page 1-10.
- See “Frequency Indicator Input” on page 1-12.
- See “Description Field Input” on page 1-13.

Single Matrix Input

The date information provided with this syntax must be in serial date number format. The date number may or may not include time-of-day information.

Note If you are unfamiliar with the concepts of date strings and serial date numbers, consult the section "Handling and Converting Dates" in the Financial Toolbox documentation.

Time-of-Day Information Excluded.

```
fts = fints(dates_and_data)
```

In this simplest form of syntax, the input must be at least a two-column matrix. The first column contains the dates in serial date format; the second column is the data series. The input matrix can have more than two columns, each additional column representing a different data series or set of observations.

If the input is a two-column matrix, the output object contains four fields: `desc`, `freq`, `dates`, and `series1`. The description field, `desc`, defaults to blanks `''`, and the frequency indicator field, `freq`, defaults to `0`. The dates field, `dates`, contains the serial dates from the first column of the input matrix, while the data series field, `series1`, has the data from the second column of the input matrix.

The first example makes two financial time series objects. The first one has only one data series, while the other has more than one. A random vector provides the values for the data series. The range of dates is arbitrarily chosen using the `today` function:

```
date_series = (today:today+100)';
data_series = exp(randn(1, 101))';
dates_and_data = [date_series data_series];
fts1 = fints(dates_and_data);
```

Examine the contents of the object `fts1` just created. The actual date series you observe will vary according to the day when you run the example (the value of `today`). Also, your values in `series1` will differ from those shown, depending upon the sequence of random numbers generated:

```
fts1 =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (101)'    'series1:  (101)'
    '12-Jul-1999'      [         0.3124]
    '13-Jul-1999'      [         3.2665]
    '14-Jul-1999'      [         0.9847]
    '15-Jul-1999'      [         1.7095]
    '16-Jul-1999'      [         0.4885]
    '17-Jul-1999'      [         0.5192]
    '18-Jul-1999'      [         1.3694]
    '19-Jul-1999'      [         1.1127]
    '20-Jul-1999'      [         6.3485]
    '21-Jul-1999'      [         0.7595]
    '22-Jul-1999'      [         9.1390]
    '23-Jul-1999'      [         4.5201]
    '24-Jul-1999'      [         0.1430]
    '25-Jul-1999'      [         0.1863]
    '26-Jul-1999'      [         0.5635]
    '27-Jul-1999'      [         0.8304]
    '28-Jul-1999'      [         1.0090]...
```

The output is truncated for brevity. There are actually 101 data points in the object.

Note that the `desc` field displays as `(none)` instead of `''`, and that the contents of the object display as cell array elements. Although the object displays as such, it should be thought of as a MATLAB structure containing the default field names for a single data series object: `desc`, `freq`, `dates`, and `series1`.

Now create an object with more than one data series in it:

```
date_series = (today:today+100)';
data_series1 = exp(randn(1, 101))';
data_series2 = exp(randn(1, 101))';
dates_and_data = [date_series data_series1 data_series2];
fts2 = fints(dates_and_data);
```

Now look at the object created (again in abbreviated form):

```
fts2 =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (101)'    'series1:  (101)'    'series2:  (101)'
    '12-Jul-1999'      [         0.5816]    [         1.2816]
    '13-Jul-1999'      [         5.1253]    [         0.9262]
    '14-Jul-1999'      [         2.2824]    [         5.6869]
    '15-Jul-1999'      [         1.2596]    [         5.0631]
    '16-Jul-1999'      [         1.9574]    [         1.8709]
    '17-Jul-1999'      [         0.6017]    [         1.0962]
    '18-Jul-1999'      [         2.3546]    [         0.4459]
    '19-Jul-1999'      [         1.3080]    [         0.6304]
    '20-Jul-1999'      [         1.8682]    [         0.2451]
    '21-Jul-1999'      [         0.3509]    [         0.6876]
    '22-Jul-1999'      [         4.6444]    [         0.6244]
    '23-Jul-1999'      [         1.5441]    [         5.7621]
    '24-Jul-1999'      [         0.1470]    [         2.1238]
    '25-Jul-1999'      [         1.5999]    [         1.0671]
    '26-Jul-1999'      [         3.5764]    [         0.7462]
    '27-Jul-1999'      [         1.8937]    [         1.0863]
    '28-Jul-1999'      [         3.9780]    [         2.1516]...
```

The second data series name defaults to `series2`, as expected.

Before you can perform any operations on the object, you must set the frequency indicator field `freq` to the valid frequency of the data series contained in the object. You can leave the description field `desc` blank.

To set the frequency indicator field to a daily frequency, enter

`fts2.freq = 1`, or

```
fts2.freq = 'daily'
```

See the `fints` function description in the “Function Reference” for a list of frequency indicators.

Time-of-Day Information Included.

The serial date number used with this form of the `fints` function can incorporate time-of-day information. When time-of-day information is present, the output of the function contains a field `times` that indicates the time of day.

If you recode a previous example to include time-of-day information, you can see the additional column present in the output object:

```
time_series = (now:now+100)';
data_series = exp(randn(1, 101))';
times_and_data = [time_series data_series];
fts1 = fints(times_and_data);

fts1 =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (101)'    'times:  (101)'    'series1:  (101)'
    '29-Nov-2001'      '14:57'            [         0.5816]
    '30-Nov-2001'      '14:57'            [         5.1253]
    '01-Dec-2001'      '14:57'            [         2.2824]
    '02-Dec-2001'      '14:57'            [         1.2596]...
```

Separate Vector Input

The date information provided with this syntax can be in serial date number or date string format. The date information may or may not include time-of-day information.

Time-of-Day Information Excluded.

```
fts = fints(dates, data)
```

In this second syntax the dates and data series are entered as separate vectors to `fints`, the financial time series object constructor function. The `dates` vector must be a column vector, while the data series `data` can be a column vector (if there is only one data series) or a column-oriented matrix (for multiple data series). A column-oriented matrix, in this context, indicates that each column is a set of observations. Different columns are different sets of data series.

Here is an example:

```
dates = (today:today+100)';
data_series1 = exp(randn(1, 101))';
data_series2 = exp(randn(1, 101))';
data = [data_series1 data_series2];
fts = fints(dates, data)

fts =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (101)'    'series1:  (101)'    'series2:  (101)'
    '12-Jul-1999'      [         0.5816]    [         1.2816]
    '13-Jul-1999'      [         5.1253]    [         0.9262]
    '14-Jul-1999'      [         2.2824]    [         5.6869]
    '15-Jul-1999'      [         1.2596]    [         5.0631]
    '16-Jul-1999'      [         1.9574]    [         1.8709]
    '17-Jul-1999'      [         0.6017]    [         1.0962]
    '18-Jul-1999'      [         2.3546]    [         0.4459]
    '19-Jul-1999'      [         1.3080]    [         0.6304]
    '20-Jul-1999'      [         1.8682]    [         0.2451]
    '21-Jul-1999'      [         0.3509]    [         0.6876]
    '22-Jul-1999'      [         4.6444]    [         0.6244]
    '23-Jul-1999'      [         1.5441]    [         5.7621]
    '24-Jul-1999'      [         0.1470]    [         2.1238]
    '25-Jul-1999'      [         1.5999]    [         1.0671]
    '26-Jul-1999'      [         3.5764]    [         0.7462]
    '27-Jul-1999'      [         1.8937]    [         1.0863]
    '28-Jul-1999'      [         3.9780]    [         2.1516]...
```

The result is exactly the same as the first syntax. The only difference between the first and second syntax is the way the inputs are entered into the constructor function.

Time-of-Day Information Included.

With this form of the function you can enter the time-of-day information either as a serial date number or as a date string. If more than one serial date and time are present, the entry must be in the form of a column-oriented matrix. If more than one string date and time are present, the entry must be a column-oriented cell array of dates and times.

With date string input the dates and times can initially be separate column-oriented date and time series, but you must concatenate them into a single column-oriented cell array before entering them as the first input to `fints`.

For date string input the allowable formats are

- `'ddmmmyy hh:mm'` or `'ddmmmyyyy hh:mm'`
- `'mm/dd/yy hh:mm'` or `'mm/dd/yyyy hh:mm'`
- `'dd-mmm-yy hh:mm'` or `'dd-mmm-yyyy hh:mm'`
- `'mmm.dd,yy hh:mm'` or `'mmm.dd,yyyy hh:mm'`

The next example shows time-of-day information input as serial date numbers in a column-oriented matrix:

```
f = fints([now;now+1],(1:2)')

f =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (2)'    'times:  (2)'    'series1:  (2)'
    '29-Nov-2001'    '15:22'          [            1]
    '30-Nov-2001'    '15:22'          [            2]
```

If the time-of-day information is in date string format, you must provide it to `fints` as a column-oriented cell array:

```
f = fints({'01-Jan-2001 12:00';'02-Jan-2001 12:00'},(1:2)')

f =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (2)'    'times:  (2)'    'series1:  (2)'
    '01-Jan-2001'    '12:00'          [            1]
    '02-Jan-2001'    '12:00'          [            2]
```

If the dates and times are in date string format and contained in separate matrices, you must concatenate them before using the date and time information as input to `fints`:

```
dates = ['01-Jan-2001'; '02-Jan-2001'; '03-Jan-2001'];
times = ['12:00';'12:00';'12:00'];
dates_time = cellstr([dates,repmat(' ',size(dates,1),1),times]);
f = fints(dates_time,(1:3)')

f =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (3)'    'times:  (3)'    'series1:  (3)'
    '01-Jan-2001'    '12:00'          [            1]
    '02-Jan-2001'    '12:00'          [            2]
    '03-Jan-2001'    '12:00'          [            3]
```

Data Name Input

```
fts = fints(dates, data, datanames)
```

The third syntax lets you specify the names for the data series with the argument `datanames`. The `datanames` argument can be a MATLAB string for a single data series. For multiple data series names, it must be a cell array of strings.

Look at two examples, one with a single data series and a second with two. The first example sets the data series name to the specified name `First`:

```
dates = (today:today+100)';
data = exp(randn(1, 101))';
fts1 = fints(dates, data, 'First')

fts1 =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (101)'    'First:  (101)'
    '12-Jul-1999'      [       0.4615]
    '13-Jul-1999'      [       1.1640]
```

```
    '14-Jul-1999'      [       0.7140]
    '15-Jul-1999'      [       2.6400]
    '16-Jul-1999'      [       0.8983]
    '17-Jul-1999'      [       2.7552]
    '18-Jul-1999'      [       0.6217]
    '19-Jul-1999'      [       1.0714]
    '20-Jul-1999'      [       1.4897]
    '21-Jul-1999'      [       3.0536]
    '22-Jul-1999'      [       1.8598]
    '23-Jul-1999'      [       0.7500]
    '24-Jul-1999'      [       0.2537]
    '25-Jul-1999'      [       0.5037]
    '26-Jul-1999'      [       1.3933]
    '27-Jul-1999'      [       0.3687]...
```

The second example provides two data series named `First` and `Second`:

```
dates = (today:today+100)';
data_series1 = exp(randn(1, 101))';
data_series2 = exp(randn(1, 101))';
data = [data_series1 data_series2];
fts2 = fints(dates, data, {'First', 'Second'})

fts2 =
    desc:  (none)
    freq:  Unknown (0)

    'dates:  (101)'    'First:  (101)'    'Second:  (101)'
    '12-Jul-1999'      [       1.2305]    [        0.7396]
    '13-Jul-1999'      [       1.2473]    [        2.6038]
    '14-Jul-1999'      [       0.3657]    [        0.5866]
    '15-Jul-1999'      [       0.6357]    [        0.4061]
    '16-Jul-1999'      [       4.0530]    [        0.4096]
    '17-Jul-1999'      [       0.6300]    [        1.3214]
    '18-Jul-1999'      [       1.0333]    [        0.4744]
    '19-Jul-1999'      [       2.2228]    [        4.9702]
    '20-Jul-1999'      [       2.4518]    [        1.7758]
    '21-Jul-1999'      [       1.1479]    [        1.3780]
    '22-Jul-1999'      [       0.1981]    [        0.8595]
    '23-Jul-1999'      [       0.1927]    [        1.3713]
    '24-Jul-1999'      [       1.5353]    [        3.8332]
```

```
'25-Jul-1999'      [       0.4784]    [       0.1067]
'26-Jul-1999'      [       1.7593]    [       3.6434]
'27-Jul-1999'      [       0.2505]    [       0.6849]
'28-Jul-1999'      [       1.5845]    [       1.0025]...
```

Note Data series names must be valid MATLAB variable names. The only allowed nonalphanumeric character is the underscore (_) character.

Because `freq` for `fts2` has not been explicitly indicated, the frequency indicator for `fts2` is set to `Unknown`. Set the frequency indicator field `freq` before you attempt any operations on the object. You will not be able to use the object until the frequency indicator field is set to a valid indicator.

Frequency Indicator Input

```
fts = fints(dates, data, datanames, freq)
```

With the fourth syntax you can set the frequency indicator field when you create the financial time series object. The frequency indicator field `freq` is set as the fourth input argument. You will not be able to use the financial time series object until `freq` is set to a valid indicator. Valid frequency indicators are

```
UNKNOWN, Unknown, unknown, U, u,0
DAILY, Daily, daily, D, d,1
WEEKLY, Weekly, weekly, W, w,2
MONTHLY, Monthly, monthly, M, m,3
QUARTERLY, Quarterly, quarterly, Q, q,4
SEMIANNUAL, Semiannual, semiannual, S, s,5
ANNUAL, Annual, annual, A, a,6
```

The previous example contained sets of daily data. The `freq` field displayed as `Unknown (0)` because the frequency indicator was not explicitly set. The command

```
fts = fints(dates, data, {'First', 'Second'}, 1);
```

sets the `freq` indicator to `Daily(1)` when creating the financial time series object:

```
fts =

    desc:  (none)
    freq:  Daily (1)

    'dates:  (101)'    'First:  (101)'    'Second:  (101)'
    '12-Jul-1999'      [       1.2305]    [        0.7396]
    '13-Jul-1999'      [       1.2473]    [        2.6038]
    '14-Jul-1999'      [       0.3657]    [        0.5866]
    '15-Jul-1999'      [       0.6357]    [        0.4061]
    '16-Jul-1999'      [       4.0530]    [        0.4096]
    '17-Jul-1999'      [       0.6300]    [        1.3214]
    '18-Jul-1999'      [       1.0333]    [        0.4744]...
```

When you create the object using this syntax, you can use the other valid frequency indicators for a particular frequency. For a daily data set you can use `DAILY`, `Daily`, `daily`, `D`, or `d`. Similarly, with the other frequencies, you can use the valid string indicators or their numeric counterparts.

Description Field Input

```
fts = fints(dates, data, datanames, freq, desc)
```

With the fifth syntax you can explicitly set the description field as the fifth input argument. The description can be anything you want. It is not used in any operations performed on the object.

This example sets the `desc` field to `'Test TS'`.

```
dates = (today:today+100)';
data_series1 = exp(randn(1, 101))';
data_series2 = exp(randn(1, 101))';
data = [data_series1 data_series2];
fts = fints(dates, data, {'First', 'Second'}, 1, 'Test TS')

fts =
    desc:  Test TS
    freq:  Daily (1)

    'dates:  (101)'    'First:  (101)'    'Second:  (101)'
    '12-Jul-1999'      [       0.5428]    [        1.2491]
    '13-Jul-1999'      [       0.6649]    [        6.4969]
    '14-Jul-1999'      [       0.2428]    [        1.1163]
```

```
'15-Jul-1999'      [       1.2550]    [       0.6628]
'16-Jul-1999'      [       1.2312]    [       1.6674]
'17-Jul-1999'      [       0.4869]    [       0.3015]
'18-Jul-1999'      [       2.1335]    [       0.9081]...
```

Now the description field is filled with the specified string `'Test TS'` when the constructor is called.

Transforming a Text File

The function `ascii2fts` creates a financial time series object from a text (ASCII) data file provided that the data file conforms to a general format. The general format of the text data file is

- Can contain header text lines.
- Can contain column header information. The column header information must immediately precede the data series columns unless the `skiprows` argument (see below) is specified.
- Leftmost column must be the date column.
- Dates must be in a valid date string format.
  - `'ddmmmyy'` or `'ddmmmyyyy'`
  - `'mm/dd/yy'` or `'mm/dd/yyyy'`
  - `'dd-mmm-yy'` or `'dd-mmm-yyyy'`
  - `'mmm.dd,yy'` or `'mmm.dd,yyyy'`
- Each column must be separated either by spaces or a tab.

Several example text data files are included with the toolbox. These files are in the `ftsdata` subdirectory within the Financial Time Series Toolbox directory `<matlab>/toolbox/ftseries`.

The syntax of the function

```
fts = ascii2fts(filename, descrow, colheadrow, skiprows);
```

takes in the data filename (`filename`), the row number where the text for the description field is (`descrow`), the row number of the column header information (`colheadrow`), and the row numbers of rows to be skipped (`skiprows`). For example, rows need to be skipped when there are intervening rows between the column head row and the start of the time series data.

Look at the beginning of the ASCII file `disney.dat` in the `ftsdata` subdirectory:

```
Walt Disney Company (DIS)
Daily prices (3/29/96 to 3/29/99)
DATE     OPEN     HIGH     LOW      CLOSE     VOLUME
3/29/99  33.0625  33.188   32.75    33.063    6320500
3/26/99  33.3125  33.375   32.75    32.938    5552800
3/25/99  33.5     33.625   32.875   33.375    7936000
3/24/99  33.0625  33.25    32.625   33.188    6025400...
```

The command line

```
disfts = ascii2fts('disney.dat', 1, 3, 2)
```

uses `disney.dat` to create time series object `disfts`. This example

- Reads the text data file `disney.dat`
- Uses the first line in the file as the content of the description field
- Skips the second line
- Parses the third line in the file for column header (or data series names)
- Parses the rest of the file for the date vector as well as the data series values

The resulting financial time series object looks like this.

```
disfts =

 desc:  Walt Disney Company (DIS)
 freq:  Unknown (0)

 'dates: (782)'   'OPEN: (782)'   'HIGH: (782)'   'LOW: (782)'
'29-Mar-1996'     [    21.1938]    [    21.6250]   [   21.2920]
'01-Apr-1996'     [    21.1120]    [    21.6250]   [   21.4170]
'02-Apr-1996'     [    21.3165]    [    21.8750]   [   21.6670]
'03-Apr-1996'     [    21.4802]    [    21.8750]   [   21.7500]
'04-Apr-1996'     [    21.4393]    [    21.8750]   [   21.5000]
'05-Apr-1996'     [         NaN]    [         NaN]   [        NaN]
'09-Apr-1996'     [    21.1529]    [    21.5420]   [   21.2080]
'10-Apr-1996'     [    20.7387]    [    21.1670]   [   20.2500]
'11-Apr-1996'     [    20.0829]    [    20.5000]   [   20.0420]
'12-Apr-1996'     [    19.9189]    [    20.5830]   [   20.0830]
```

```
'15-Apr-1996'     [    20.2878]    [    20.7920]    [    20.3750]
'16-Apr-1996'     [    20.3698]    [    20.9170]    [    20.1670]
'17-Apr-1996'     [    20.4927]    [    20.9170]    [    20.7080]
'18-Apr-1996'     [    20.4927]    [    21.0420]    [    20.7920]
```

There are 782 data points in this object. Only the first few lines are shown here. Also, this object has two other data series, the `CLOSE` and `VOLUME` data series, that are not shown here. Note that in creating the financial time series object, `ascii2fts` sorts the data into ascending chronological order.

The frequency indicator field, `freq`, is set to `0` for `Unknown` frequency. You can manually reset it to the appropriate frequency using structure syntax `disfts.freq = 1` for `Daily` frequency.

With a slightly different syntax, the function `ascii2fts` can create a financial time series object when time-of-day data is present in the ASCII file. The new syntax has the form

```
fts = ascii2fts(filename, timedata, descrow, colheadrow,
skiprows);
```

Set `timedata` to `'T'` when time-of-day data is present and to `'NT'` when there is no time data. For an example using this function with time-of-day data, see the reference page for `ascii2fts`.

Visualizing Financial Time Series Objects

The Financial Time Series Toolbox contains the function `chartfts`, which provides a visual representation of a financial time series object. `chartfts` is an interactive charting and graphing utility for financial time series objects. With this function you can observe time series values on the entire range of dates covered by the time series. The function additionally provides two specialized tools for extracting additional information about the displayed data series:

- "Zoom Tool" for focus on a specific time period within the time frame covered by the time series
- "Combine Axes Tool" to look for patterns among the various data series

---

Note Interactive charting is also available from the **Graphs** menu of the Financial Time Series Toolbox graphical user interface. See "Interactive Chart" on page 4-17 for additional information.

Using chartfts

`chartfts` requires a single input argument, `tsobj`, where `tsobj` is the name of the financial time series object you want to explore. Most equity financial time series objects contain four price series, such as opening, closing, highest, and lowest prices, plus an additional series containing the volume traded. However, `chartfts` is not limited to a time series of equity prices and volume traded. It can be used to display any time series data you may have.

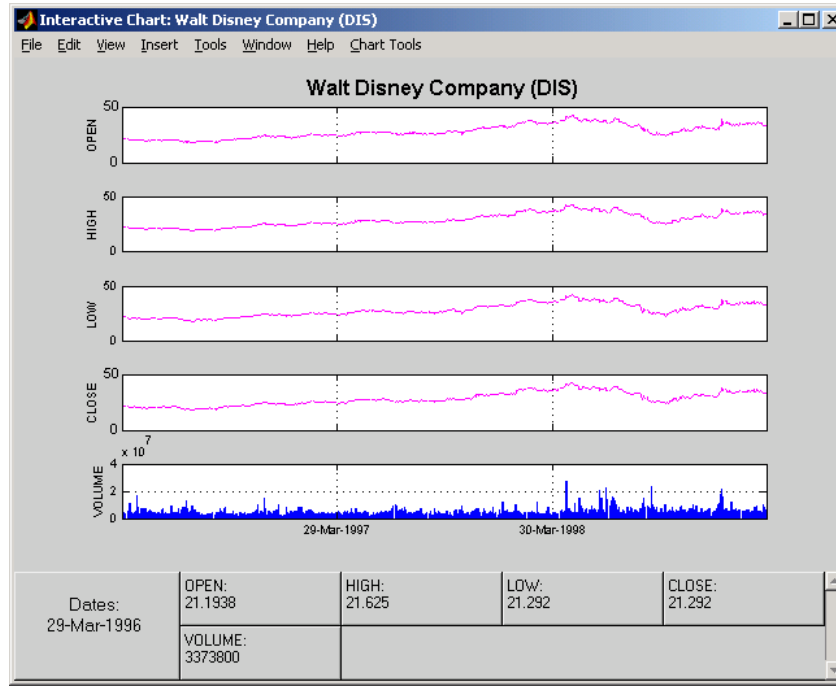
To illustrate the use of `chartfts`, use the equity price and volume traded data for the Walt Disney Corporation (NYSE: DIS) provided in the file `disney.mat`:

```
load disney.mat

whos

  Name             Size         Bytes  Class

  dis            782x5          39290  fints object
  dis_CLOSE      782x1           6256  double array
```

```
dis_HIGH        782x1           6256  double array
dis_LOW         782x1           6256  double array
dis_OPEN        782x1           6256  double array
dis_VOLUME      782x1           6256  double array
dis_nv          782x4          32930  fints object
q_dis            13x4           2196  fints object
```

For charting purposes look only at the objects `dis` (daily equity data including volume traded) and `dis_nv` (daily data without volume traded). Both objects contain the series `OPEN`, `HIGH`, `LOW`, and `CLOSE`, but only `dis` contains the additional `VOLUME` series.

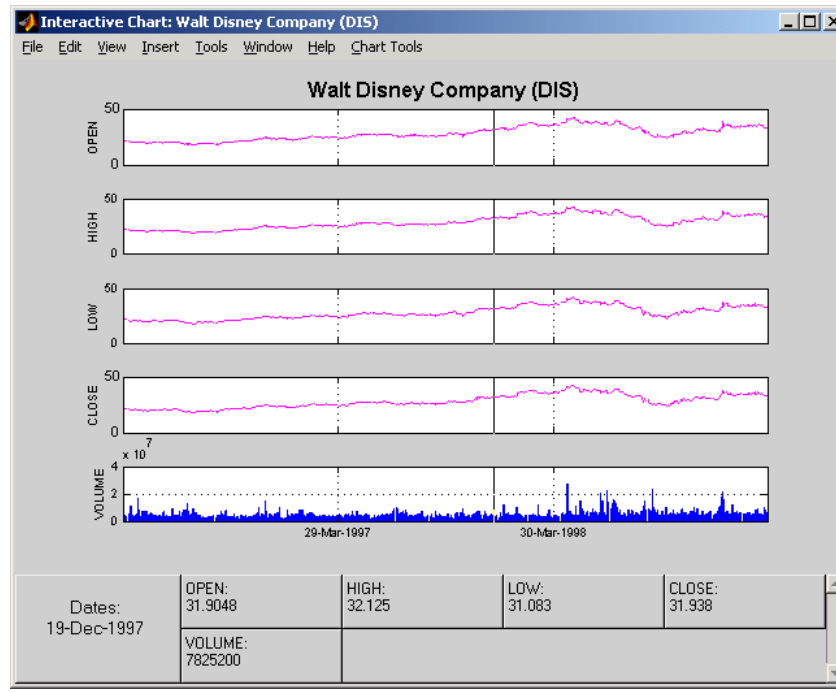
Use `chartfts(dis)` to observe the values.



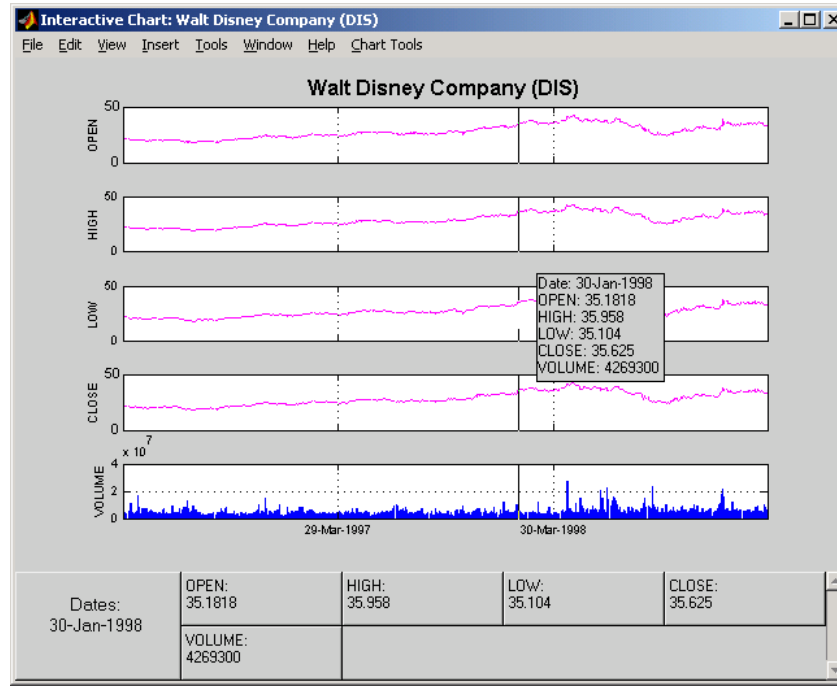
The chart contains five plots, each representing one of the series in the time series object. Boxes indicate the value of each individual plot. The date box is always on the left. The number of data boxes on the right depends upon the number of data series in the time series object, five in this case. The order in

which these boxes are arranged (left to right) matches the plots from top to bottom. With more than eight data series in the object, the scroll bar on the right is activated so that additional data from the other series can be brought into view.

Slide the mouse cursor over the chart. A vertical bar appears across all plots. This bar selects the set of data shown in the boxes below. Move this bar horizontally and the data changes accordingly.

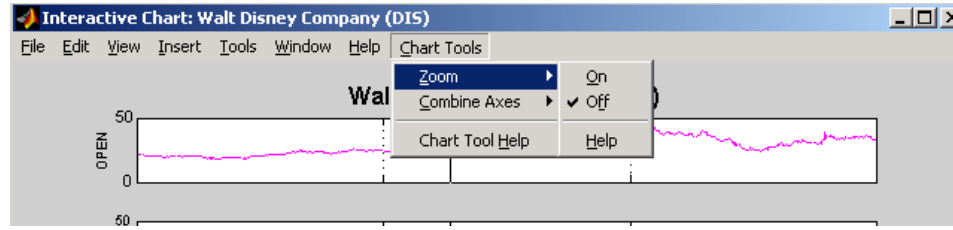


Click the plot. A small information box displays the data at the point where you click the mouse button.



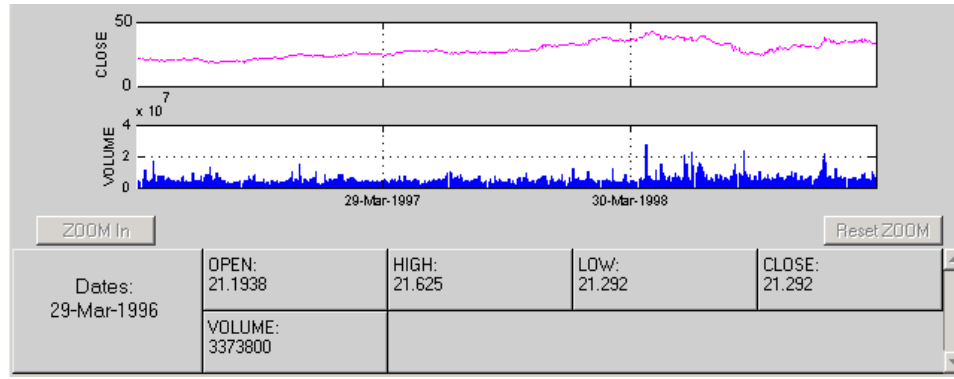
Zoom Tool

The zoom feature of `chartfts` enables a more detailed look at the data during a selected time frame. The Zoom tool is found under the **Chart Tools** menu.

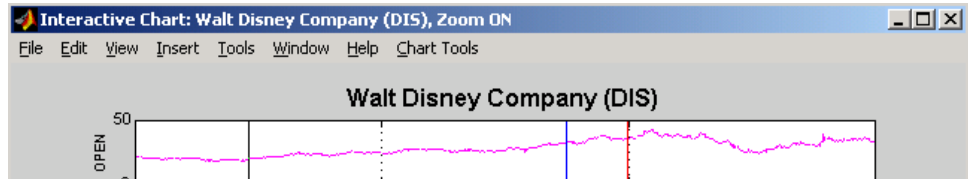


Note Due to the specialized nature of this feature, do not use the MATLAB `zoom` command or **Zoom In** and **Zoom Out** from the **Tools** menu.

When the feature is turned on, you will see two inactive buttons (**ZOOM In** and **Reset ZOOM**) above the boxes. The buttons become active later after certain actions have been performed.

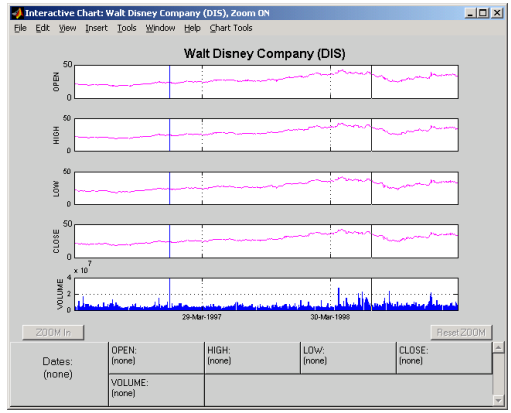


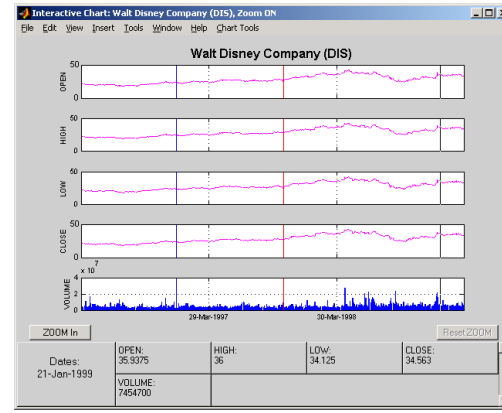
The figure window title bar displays the status of the chart tool that you are using. With the Zoom tool turned on, you see **Zoom ON** in the title bar in addition to the name of the time series you are working with. When the tool is off, no status is displayed.



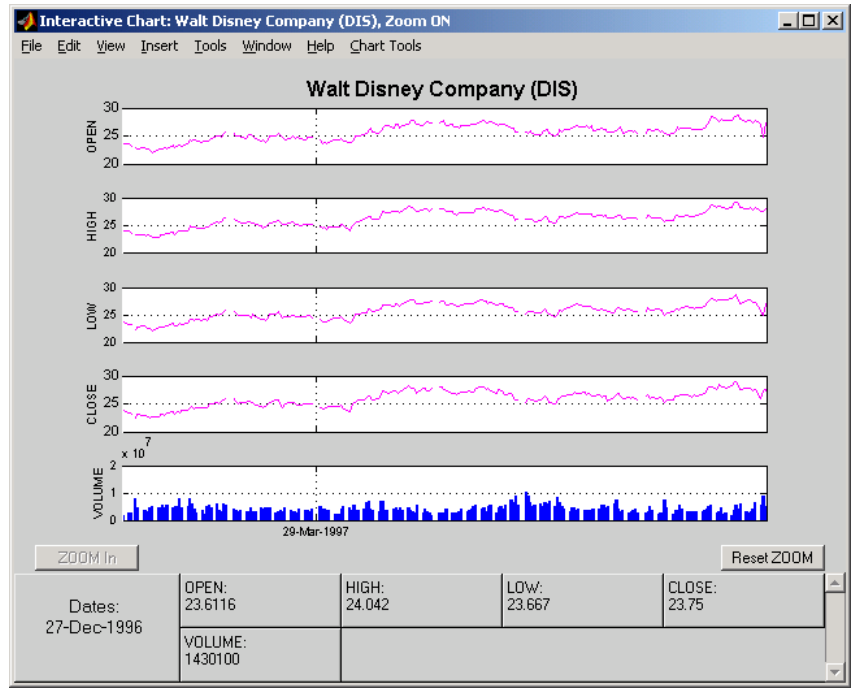
To zoom into the chart, you need to define the starting and ending dates. Define the starting date by moving the cursor over the chart until the desired date appears at the bottom left box and click the mouse button. A blue vertical line indicates the starting date you have selected. Next, again move the cursor over the chart until the desired ending date appears in the box and click the mouse

once again. This time, a red vertical line appears and the **ZOOM In** button is activated.



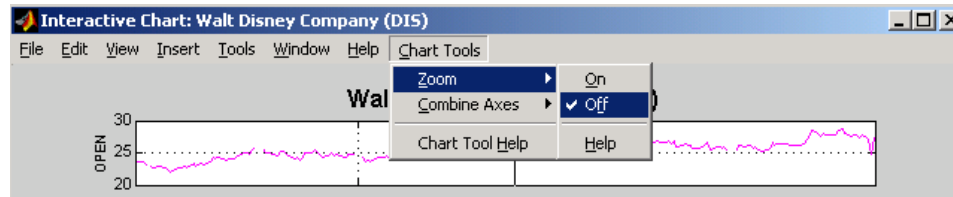


To zoom into the chart, click the **ZOOM In** button.



The chart is zoomed in. Note that the **Reset ZOOM** button now becomes active while the **ZOOM In** button becomes inactive again. To return the chart to its original state (not zoomed), click the **Reset ZOOM** button. To zoom into the chart even further, repeat the steps above for zooming into the chart.

Turn the Zoom tool off by going back to the **Chart Tools** menu and choosing **Zoom Off**.



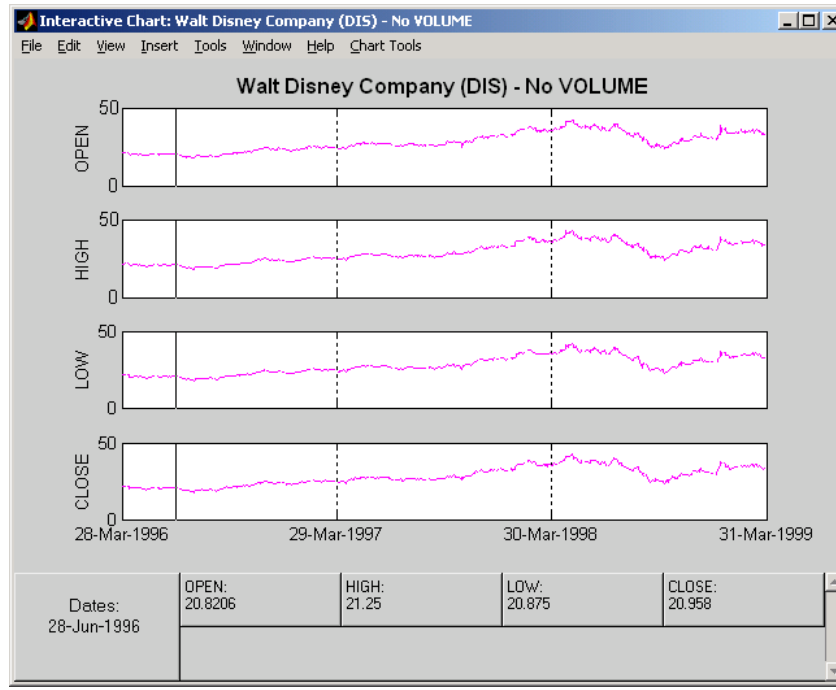
With the tool turned off, the chart stays at the last state that it was in. If you turn it off when the chart is zoomed in, the chart stays zoomed in. If you reset the zoom before turning it off, the chart becomes the original (not zoomed).

Combine Axes Tool

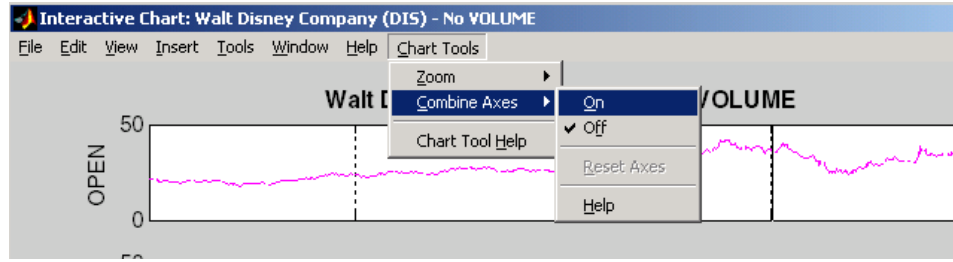
The Combine Axes tool allows you to combine all axes or specific axes into one. With axes combined you can visually spot any trends that can occur among the data series in a financial time series object.

To illustrate this tool, use `dis_nv`, the financial time series object that does not contain volume traded data:

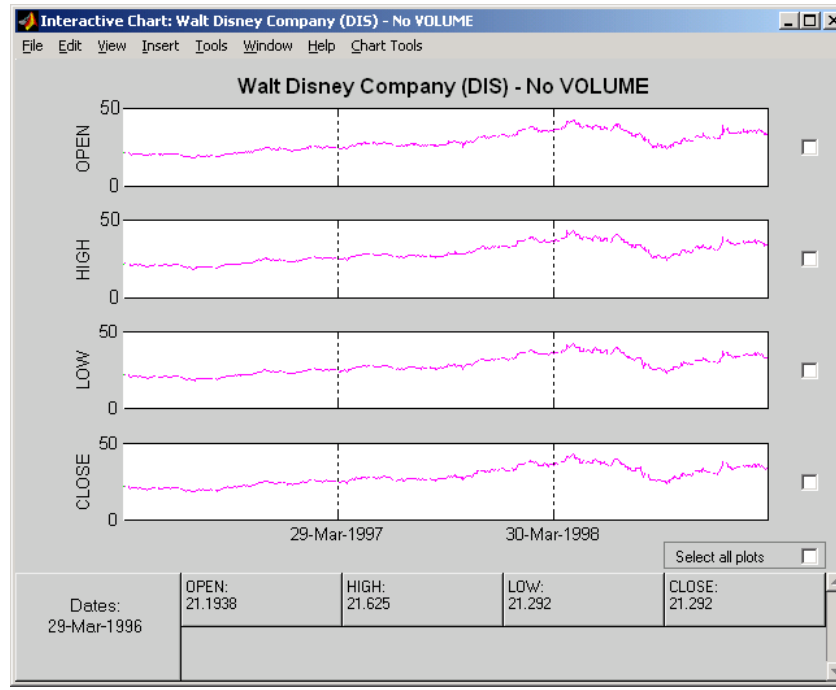
```
chartfts(dis_nv)
```



To combine axes, choose the **Chart Tools** menu, followed by **Combine Axes** and **On**.

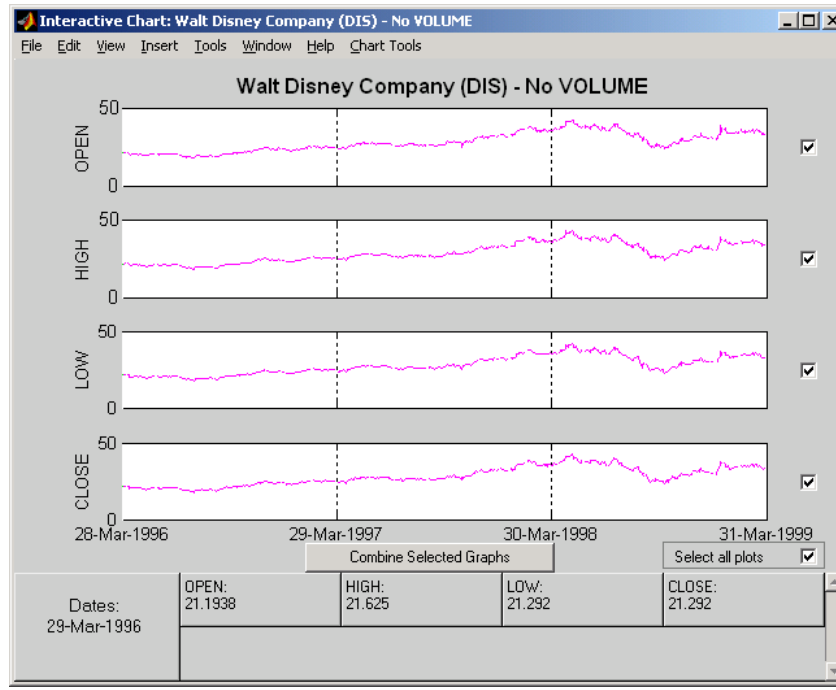


When the Combine Axes tool is on, check boxes appear beside each individual plot. An additional check box enables the combination of all plots.

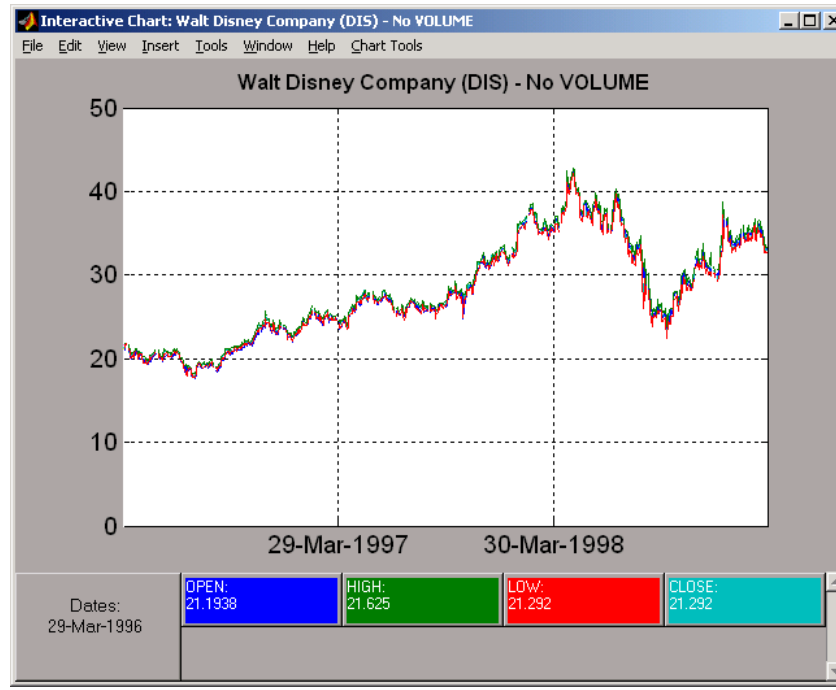


Combining All Axes

To combine all plots, click the check box for **Select all plots.**



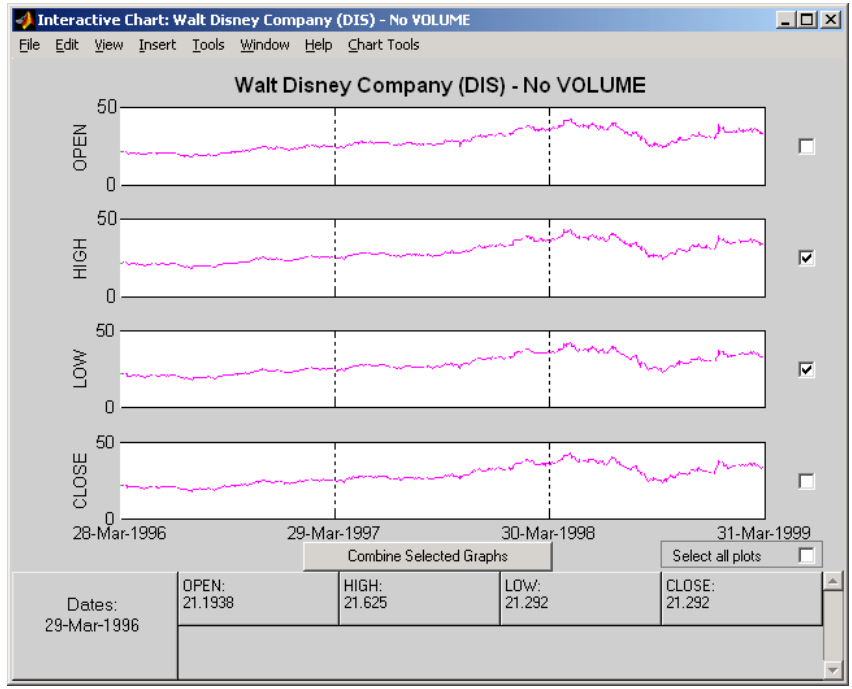
Now click the **Combine Selected Graphs** button to combine the chosen plots. In this case, all plots are combined.



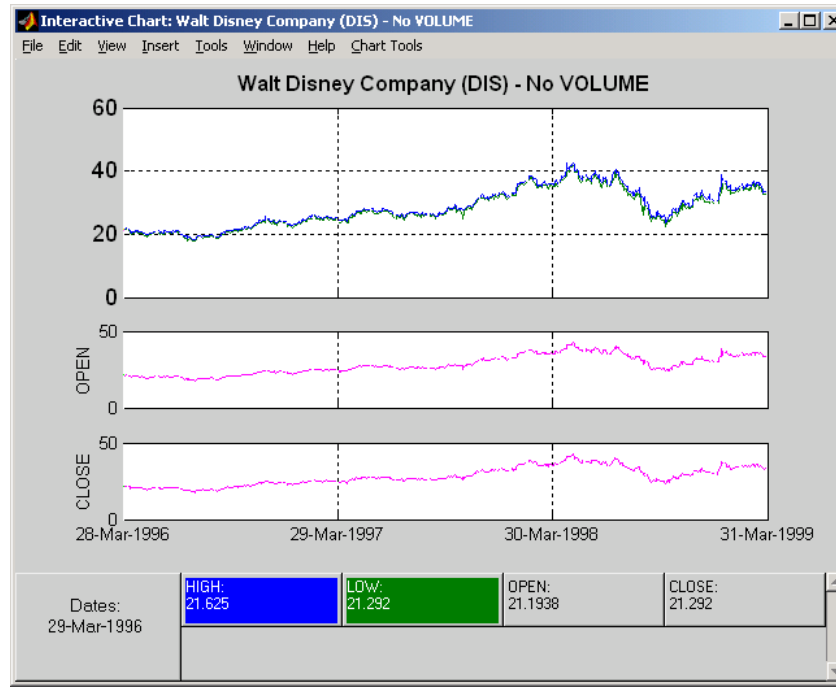
The combined plots have a single plot axis with all data series traced. The background of each data box has changed to the color corresponding to the color of the trace that represents the data series. After the axes are combined, the tool is turned off.

Combining Selected Axes

You can choose any combination of the available axes to combine. For example, combine the `HIGH` and `LOW` price series of the Disney time series. Click the check boxes next to the corresponding plots. The **Combine Selected Graphs** button appears and is active.



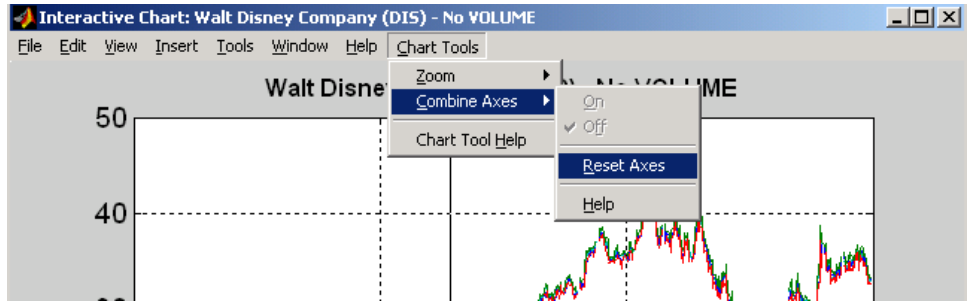
Click the **Combine Selected Graphs** button. The chart with the combined plots looks like the next figure.



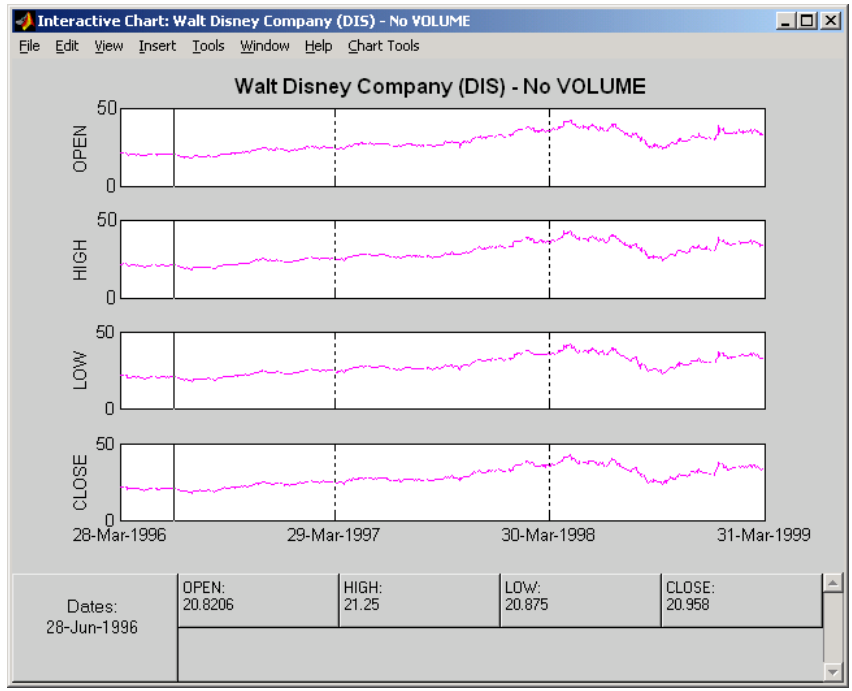
The plot with the combined axes is located at the top of the chart while the remaining plots follow it. The data boxes have also been changed. The boxes that correspond to the combined axes are relocated to the beginning, and the background colors are set to the color of the respective traces. The data boxes for the remaining axes retain their original formats.

Resetting Axes

If you have altered the chart by combining axes, you must reset the axes before you can visualize additional combinations. Reset the axes with the **Reset Axes** menu item under **Chart Tools -> Combine Axes**. Note that now the **On** and **Off** features are turned off.



With axes reset, the interactive chart appears in its original format, and you can proceed with additional axes combinations.



2

Using Financial Time Series

Introduction

This chapter discusses how to manipulate and analyze financial time series data. The major topics discussed include

Much of this information is summarized in the "Demonstration Program" on page 2-24.

Working with Financial Time Series Objects

A financial time series object is designed to be used as if it were a MATLAB structure. (See the MATLAB documentation for a description of MATLAB structures or how to use MATLAB in general.)

This part of the tutorial assumes that you know how to use MATLAB and are familiar with MATLAB structures. The terminology is similar to that of a MATLAB structure. The financial time series object term *component* is interchangeable with the MATLAB structure term *field*.

Financial Time Series Object Structure

A financial time series object always contains three component names: `desc` (description field), `freq` (frequency indicator field), and `dates` (date vector). If you build the object using the constructor `fints`, the default value for the description field is a blank string (`''`). If you build the object from a text data file using `ascii2fts`, the default is the name of the text data file. The default for the frequency indicator field is `0` (`Unknown` frequency). Objects created from operations can default the setting to `0`. For example, if you decide to pick out values selectively from an object, the frequency of the new object might not be the same as that of the object from which it came.

The date vector `dates` does not have a default set of values. When you create an object, you have to supply the date vector. You can change the date vector afterwards but, at object creation time, you must provide a set of dates.

The final component of a financial time series object is one or more data series vectors. If you do not supply a name for the data series, the default name is `series1`. If you have multiple data series in an object and do not supply the names, the default is the name series followed by a number, for example, `series1`, `series2`, and `series3`.

Data Extraction

Here is an exercise on how to extract data from a financial time series object. As mentioned before, you can think of the object as a MATLAB structure. Highlight each line in the exercise in the MATLAB Help browser, press the right mouse key, and select **Evaluate Selection** to execute it.

To begin, create a financial time series object called `myfts`:

```
dates = (datenum('05/11/99'):datenum('05/11/99')+100)';
data_series1 = exp(randn(1, 101))';
data_series2 = exp(randn(1, 101))';
data = [data_series1 data_series2];
myfts = fints(dates, data);
```

The `myfts` object looks like this:

```
myfts =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (101)'    'series1:  (101)'    'series2:  (101)'
    '11-May-1999'      [         2.8108]    [         0.9323]
    '12-May-1999'      [         0.2454]    [         0.5608]
    '13-May-1999'      [         0.3568]    [         1.5989]
    '14-May-1999'      [         0.5255]    [         3.6682]
    '15-May-1999'      [         1.1862]    [         5.1284]
    '16-May-1999'      [         3.8376]    [         0.4952]
    '17-May-1999'      [         6.9329]    [         2.2417]
    '18-May-1999'      [         2.0987]    [         0.3579]
    '19-May-1999'      [         2.2524]    [         3.6492]
    '20-May-1999'      [         0.8669]    [         1.0150]
    '21-May-1999'      [         0.9050]    [         1.2445]
    '22-May-1999'      [         0.4493]    [         5.5466]
    '23-May-1999'      [         1.6376]    [         0.1251]
    '24-May-1999'      [         3.4472]    [         1.1195]
    '25-May-1999'      [         3.6545]    [         0.3374]...
```

There are more dates in the object; only the first few lines are shown here.

Note The actual data in your `series1` and `series2` will differ from the above because of the use of random numbers.

Now create another object with only the values for `series2`:

```
srs2 = myfts.series2

srs2 =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (101)'    'series2:  (101)'
    '11-May-1999'      [         0.9323]
    '12-May-1999'      [         0.5608]
    '13-May-1999'      [         1.5989]
    '14-May-1999'      [         3.6682]
    '15-May-1999'      [         5.1284]
    '16-May-1999'      [         0.4952]
    '17-May-1999'      [         2.2417]
    '18-May-1999'      [         0.3579]
    '19-May-1999'      [         3.6492]
    '20-May-1999'      [         1.0150]
    '21-May-1999'      [         1.2445]
    '22-May-1999'      [         5.5466]
    '23-May-1999'      [         0.1251]
    '24-May-1999'      [         1.1195]
    '25-May-1999'      [         0.3374]...
```

The new object `srs2` contains all the dates in `myfts`, but the only data series is `series2`. The name of the data series retains its name from the original object, `myfts`.

Note The output from referencing a data series field or indexing a financial time series object is always another financial time series object. The exceptions are referencing the description, frequency indicator, and dates fields, and indexing into the dates field.

Object to Matrix Conversion

The function `fts2mat` extracts the dates and/or the data series values from an object and places them into a vector or a matrix. The default behavior extracts just the values into a vector or a matrix. Look at the next example:

```
srs2_vec = fts2mat(myfts.series2)

srs2_vec =

    0.9323
    0.5608
    1.5989
    3.6682
    5.1284
    0.4952
    2.2417
    0.3579
    3.6492
    1.0150
    1.2445
    5.5466
    0.1251
    1.1195
    0.3374...
```

If you want to include the dates in the output matrix, provide a second input argument and set it to 1. This results in a matrix whose first column is a vector of serial date numbers:

```
format long g

srs2_mtx = fts2mat(myfts.series2, 1)

srs2_mtx =

           730251      0.932251754559576
           730252      0.560845677519876
           730253      1.59888712183914
           730254      3.6681500883527
           730255      5.12842215360269
           730256      0.49519254119977
           730257      2.24174134286213
           730258      0.357918065917634
           730259      3.64915665824198
           730260      1.01504236943148
           730261      1.24446420606078
```

```
730262      5.54661849025711
730263      0.12507959735904
730264      1.11953883096805
730265      0.337398214166607
```

The vector `srs2_vec` contains just `series2` values. The matrix `srs2_mtx` contains dates in the first column and the values of the `series2` data series in the second. Dates in the first column are in serial date format. Serial date format is a representation of the date string format (for example, serial date = 1 is equivalent to 01-Jan-0000). (The serial date vector can include time-of-day information.)

The `long g` display format displays the numbers without exponentiation. (To revert to the default display format, use `format short`. (See the `format` command in the MATLAB documentation for a description of MATLAB display formats.) Remember that both the vector and the matrix have 101 rows of data as in the original object `myfts` but are shown truncated here.

Indexing a Financial Time Series Object

You can also index into the object as with any other MATLAB variable or structure. A financial time series object lets you use a date string, a cell array of date strings, a date string range, or normal integer indexing. *You cannot, however, index into the object using serial dates.* If you have serial dates, you must first use the MATLAB `datestr` command to convert them into date strings.

When indexing by date string, note that

- Each date string must contain the day, month, and year. Valid formats are
  - `'ddmmmyy hh:mm'` or `'ddmmmyyyy hh:mm'`
  - `'mm/dd/yy hh:mm'` or `'mm/dd/yyyy hh:mm'`
  - `'dd-mmm-yy hh:mm'` or `'dd-mmm-yyyy hh:mm'`
  - `'mmm.dd,yy hh:mm'` or `'mmm.dd,yyyy hh:mm'`
- All data falls at the end of the indicated time period, that is, weekly data falls on Fridays, monthly data falls on the end of each month, etc., whenever the data has gone through a frequency conversion.

Indexing with Date Strings

With date string indexing you get the values in a financial time series object for a specific date using a date string as the index into the object. Similarly, if you want values for multiple dates in the object, you can put those date strings into a cell array and use the cell array as the index to the object. Here are some examples.

This example extracts all values for May 11, 1999 from `myfts`:

```
format short
myfts('05/11/99')

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (1)'    'series1:  (1)'    'series2:  (1)'
    '11-May-1999'    [       2.8108]    [       0.9323]
```

The next example extracts only `series2` values for May 11, 1999 from `myfts`:

```
myfts.series2('05/11/99')

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (1)'    'series2:  (1)'
    '11-May-1999'    [       0.9323]
```

The third example extracts all values for three different dates:

```
myfts({'05/11/99', '05/21/99', '05/31/99'})

ans =

    desc:  (none)
    freq:  Unknown (0)
```

```
    'dates:  (3)'    'series1:  (3)'    'series2:  (3)'
    '11-May-1999'    [       2.8108]    [       0.9323]
    '21-May-1999'    [       0.9050]    [       1.2445]
    '31-May-1999'    [       1.4266]    [       0.6470]
```

The next example extracts only `series2` values for the same three dates:

```
myfts.series2({'05/11/99', '05/21/99', '05/31/99'})

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (3)'    'series2:  (3)'
    '11-May-1999'    [       0.9323]
    '21-May-1999'    [       1.2445]
    '31-May-1999'    [       0.6470]
```

Indexing with Date String Range

A financial time series is unique because it allows you to index into the object using a date string range. A date string range consists of two date strings separated by two colons (`::`). In MATLAB this separator is called the double-colon operator. An example of a MATLAB date string range is `'05/11/99::05/31/99'`. The operator gives you all data points available between those dates, including the start and end dates.

Here are some date string range examples:

```
myfts ('05/11/99::05/15/99')

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (5)'    'series1:  (5)'    'series2:  (5)'
    '11-May-1999'    [       2.8108]    [       0.9323]
    '12-May-1999'    [       0.2454]    [       0.5608]
    '13-May-1999'    [       0.3568]    [       1.5989]
```

```
    '14-May-1999'    [       0.5255]    [       3.6682]
    '15-May-1999'    [       1.1862]    [       5.1284]

myfts.series2('05/11/99::05/15/99')

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (5)'    'series2:  (5)'
    '11-May-1999'    [       0.9323]
    '12-May-1999'    [       0.5608]
    '13-May-1999'    [       1.5989]
    '14-May-1999'    [       3.6682]
    '15-May-1999'    [       5.1284]
```

As with any other MATLAB variable or structure, you can assign the output to another object variable:

```
nfts = myfts.series2('05/11/99::05/20/99');
```

`nfts` is the same as `ans` in the second example.

If one of the dates does not exist in the object, an error message indicates that one or both date indexes are out of the range of the available dates in the object. You can either display the contents of the object or use the command `ftsbound` to determine the first and last dates in the object.

Indexing with Integers

Integer indexing is the normal form of indexing in MATLAB. Indexing starts at 1 (not 0); index = 1 corresponds to the first element, index = 2 to the second element, index = 3 to the third element, and so on. Here are some examples with and without data series reference.

Get the first item in `series2`:

```
myfts.series2(1)

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (1)'    'series2:  (1)'
    '11-May-1999'    [       0.9323]
```

Get the first, third, and fifth items in `series2`:

```
myfts.series2([1, 3, 5])

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (3)'    'series2:  (3)'
    '11-May-1999'    [       0.9323]
    '13-May-1999'    [       1.5989]
    '15-May-1999'    [       5.1284]
```

Get items 16 through 20 in `series2`:

```
myfts.series2(16:20)

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (5)'    'series2:  (5)'
    '26-May-1999'    [       0.2105]
    '27-May-1999'    [       1.8916]
    '28-May-1999'    [       0.6673]
    '29-May-1999'    [       0.6681]
    '30-May-1999'    [       1.0877]
```

Get items 16 through 20 in the financial time series object `myfts`:

```
myfts(16:20)

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (5)'    'series1:  (5)'    'series2:  (5)'
    '26-May-1999'    [       0.7571]    [       0.2105]
    '27-May-1999'    [       1.2425]    [       1.8916]
    '28-May-1999'    [       1.8790]    [       0.6673]
    '29-May-1999'    [       0.5778]    [       0.6681]
    '30-May-1999'    [       1.2581]    [       1.0877]
```

Get the last item in `myfts`:

```
myfts(end)

ans =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (1)'    'series1:  (1)'    'series2:  (1)'
    '19-Aug-1999'    [       1.4692]    [       3.4238]
```

This example uses the MATLAB special variable `end`, which points to the last element of the object when used as an index. The example returns an object whose contents are the values in the object `myfts` on the last date entry.

Indexing When Time-of-Day Data Is Present

Both integer and date string indexing are permitted when time-of-day information is present in the financial time series object. You can index into the object with both date and time specifications, but not with time of day alone. To show how indexing works with time-of-day data present, create a financial time series object called `timeday` containing a time specification:

```
dates = ['01-Jan-2001';'01-Jan-2001'; '02-Jan-2001'; ...
         '02-Jan-2001'; '03-Jan-2001';'03-Jan-2001'];
times = ['11:00';'12:00';'11:00';'12:00';'11:00';'12:00'];
```

```
dates_times = cellstr([dates, repmat(' ',size(dates,1),1),...
                       times]);
timeday = fints(dates_times,(1:6)',{'Data1'},1,'My first FINTS')

timeday =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [          5]
    '     "     '    '12:00'          [          6]
```

Use integer indexing to extract the second and third data items from `timeday`:

```
timeday(2:3)

ans =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (2)'    'times:  (2)'    'Data1:  (2)'
    '01-Jan-2001'    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
```

For date string indexing enclose the date and time string in one pair of quotation marks. If there is one date with multiple times, indexing with only the date returns the data for all the times for that specific date. For example, the command `timeday('01-Jan-2001')` returns the data for all times on January 1, 2001:

```
ans =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (2)'    'times:  (2)'    'Data1:  (2)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
```

You can also indicate a specific date and time:

```
timeday('01-Jan-2001 12:00')

ans =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (1)'    'times:  (1)'    'Data1:  (1)'
    '01-Jan-2001'    '12:00'          [          2]
```

Use the double-colon operator `::` to specify a range of dates and times:

```
timeday('01-Jan-2001 12:00::03-Jan-2001 11:00')

ans =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (4)'    'times:  (4)'    'Data1:  (4)'
    '01-Jan-2001'    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [          5]
```

Treat `timeday` as a MATLAB structure if you want to obtain the contents of a specific field. For example, to find the times of day included in this object, enter

```
datestr(timeday.times)

ans =

11:00 AM
12:00 PM
11:00 AM
12:00 PM
11:00 AM
12:00 PM
```

Operations

Several MATLAB functions have been overloaded to work with financial time series objects. The overloaded functions include basic arithmetic functions such as addition, subtraction, multiplication, and division as well as other functions such as arithmetic average, filter, and difference. Also, specific methods have been designed to work with the financial time series object. For a list of functions grouped by type, refer to "Functions - Categorical List" or enter

```
help ftseries
```

at the MATLAB command prompt.

Basic Arithmetic

Financial time series objects permit you to do addition, subtraction, multiplication, and division, either on the entire object or on specific object fields. This is a feature that MATLAB structures do not allow. You cannot do arithmetic operations on entire MATLAB structures, only on specific fields of a structure.

You can perform arithmetic operations on two financial time series objects as long as they are compatible. (All contents are the same except for the description and the values associated with the data series.)

Note *Compatible* time series are not the same as *equal* time series. Two time series objects are equal when everything but the description fields is the same.

Here are some examples of arithmetic operations on financial time series objects.

Load a MAT-file that contains some sample financial time series objects:

```
load dji30short
```

One of the objects in `dji30short` is called `myfts1`:

```
myfts1 =

desc:  DJI30MAR94.dat
freq:  Daily (1)

'dates: (20)'  'Open: (20)' 'High: (20)' 'Low: (20)' 'Close: (20)'
'04-Mar-1994'  [ 3830.90]   [ 3868.04]  [ 3800.50]   [ 3832.30]
'07-Mar-1994'  [ 3851.72]   [ 3882.40]  [ 3824.71]   [ 3856.22]
'08-Mar-1994'  [ 3858.48]   [ 3881.55]  [ 3822.45]   [ 3851.72]
'09-Mar-1994'  [ 3853.97]   [ 3874.52]  [ 3817.95]   [ 3853.41]
'10-Mar-1994'  [ 3852.57]   [ 3865.51]  [ 3801.63]   [ 3830.62]...
```

Create another financial time series object that is identical to `myfts1`:

```
newfts = fints(myfts1.dates, fts2mat(myfts1)/100,...
{'Open','High','Low', 'Close'}, 1, 'New FTS')

newfts =

desc:  New FTS
freq:  Daily (1)

'dates: (20)' 'Open: (20)' 'High: (20)' 'Low: (20)' 'Close:(20)'
'04-Mar-1994'  [ 38.31]     [ 38.68]     [ 38.01]    [ 38.32]
'07-Mar-1994'  [ 38.52]     [ 38.82]     [ 38.25]    [ 38.56]
'08-Mar-1994'  [ 38.58]     [ 38.82]     [ 38.22]    [ 38.52]
'09-Mar-1994'  [ 38.54]     [ 38.75]     [ 38.18]    [ 38.53]
'10-Mar-1994'  [ 38.53]     [ 38.66]     [ 38.02]    [ 38.31]...
```

Perform an addition operation on both time series objects:

```
addup = myfts1 + newfts

addup =

desc:  DJI30MAR94.dat
freq:  Daily (1)

'dates: (20)'  'Open: (20)' 'High: (20)' 'Low: (20)' 'Close: (20)'
'04-Mar-1994'  [ 3869.21]   [ 3906.72]   [ 3838.51]  [ 3870.62]
'07-Mar-1994'  [ 3890.24]   [ 3921.22]   [ 3862.96]  [ 3894.78]
'08-Mar-1994'  [ 3897.06]   [ 3920.37]   [ 3860.67]  [ 3890.24]
'09-Mar-1994'  [ 3892.51]   [ 3913.27]   [ 3856.13]  [ 3891.94]
'10-Mar-1994'  [ 3891.10]   [ 3904.17]   [ 3839.65]  [ 3868.93]...
```

Now, perform a subtraction operation on both time series objects:

```
subout = myfts1 - newfts

subout =

desc:  DJI30MAR94.dat
freq:  Daily (1)

'dates: (20)'  'Open: (20)' 'High: (20)' 'Low: (20)' 'Close: (20)'
'04-Mar-1994'  [ 3792.59]   [ 3829.36]   [ 3762.49]  [ 3793.98]
'07-Mar-1994'  [ 3813.20]   [ 3843.58]   [ 3786.46]  [ 3817.66]
'08-Mar-1994'  [ 3819.90]   [ 3842.73]   [ 3784.23]  [ 3813.20]
'09-Mar-1994'  [ 3815.43]   [ 3835.77]   [ 3779.77]  [ 3814.88]
'10-Mar-1994'  [ 3814.04]   [ 3826.85]   [ 3763.61]  [ 3792.31]...
```

Operations with Objects and Matrices

You can also perform operations involving a financial time series object and a matrix or scalar:

```
  addscalar = myfts1 + 10000

  addscalar =

  desc:  DJI30MAR94.dat
  freq:  Daily (1)

'dates: (20)'  'Open: (20)' 'High: (20)' 'Low: (20)' 'Close: (20)'
'04-Mar-1994'  [ 13830.90]  [ 13868.04]  [ 13800.50] [ 13832.30]
'07-Mar-1994'  [ 13851.72]  [ 13882.40]  [ 13824.71] [ 13856.22]
'08-Mar-1994'  [ 13858.48]  [ 13881.55]  [ 13822.45] [ 13851.72]
'09-Mar-1994'  [ 13853.97]  [ 13874.52]  [ 13817.95] [ 13853.41]
'10-Mar-1994'  [ 13852.57]  [ 13865.51]  [ 13801.63] [ 13862.70]...
```

For operations with both an object and a matrix, the size of the matrix must match the size of the object. For example, a matrix to be subtracted from `myfts1` must be 20-by-4, since `myfts1` has 20 dates and four data series:

```
  submtx = myfts1 - randn(20, 4)

  submtx =

  desc:  DJI30MAR94.dat
  freq:  Daily (1)

'dates: (20)'  'Open: (20)' 'High: (20)' 'Low: (20)' 'Close: (20)'
'04-Mar-1994'  [ 3831.33]   [ 3867.75]   [ 3802.10]  [ 3832.63]
'07-Mar-1994'  [ 3853.39]   [ 3883.74]   [ 3824.45]  [ 3857.06]
'08-Mar-1994'  [ 3858.35]   [ 3880.84]   [ 3823.51]  [ 3851.22]
'09-Mar-1994'  [ 3853.68]   [ 3872.90]   [ 3816.53]  [ 3851.92]
'10-Mar-1994'  [ 3853.72]   [ 3866.20]   [ 3802.44]  [ 3831.17]...
```

Arithmetic Operations with Differing Data Series Names

Arithmetic operations on two objects that have the same size but contain different data series names require the function `fts2mat`. This function extracts the values in an object and puts them into a matrix or vector, whichever is appropriate.

To see an example, create another financial time series object the same size as `myfts1` but with different values and data series names:

```
newfts2 = fints(myfts1.dates, fts2mat(myfts1/10000),...
{'Rat1','Rat2', 'Rat3','Rat4'}, 1, 'New FTS')
```

If you attempt to add (or subtract, etc.) this new object to `myfts1`, an error indicates that the objects are not identical. Although they contain the same dates, number of dates, number of data series, and frequency, the two time series objects do not have the same data series names. Use `fts2mat` to bypass this problem:

```
addother = myfts1 + fts2mat(newfts2);
```

This operation adds the matrix that contains the contents of the data series in the object `newfts2` to `myfts1`. You should carefully consider the effects on your data before deciding to combine financial time series objects in this manner.

Other Arithmetic Operations

In addition to the basic arithmetic operations, several other mathematical functions operate directly on financial time series objects. These functions include exponential (`exp`), natural logarithm (`log`), common logarithm (`log10`), and many more. See the "Function Reference" chapter for more details.

Data Transformation and Frequency Conversion

The data transformation and the frequency conversion functions convert a data series into a different format.

Table 2-1: Data Transformation Functions

Function	Purpose
`boxcox`	Box-Cox transformation
`diff`	Differencing
`fillts`	Fill missing values
`filter`	Filter
`lagts`	Lag time series object
`leadts`	Lead time series object
`peravg`	Periodic average

Table 2-1: Data Transformation Functions (Continued)

Function	Purpose
`smoothts`	Smooth data
`tsmovavg`	Moving average

Table 2-2: Frequency Conversion Functions

Function	New Frequency
`convertto`	As specified
`resamplets`	As specified
`toannual`	Annual
`todaily`	Daily
`tomonthly`	Monthly
`toquarterly`	Quarterly
`tosemi`	Semiannually
`toweekly`	Weekly

As an example look at `boxcox`, the Box-Cox transformation function. This function transforms the data series contained in a financial time series object into another set of data series with relatively normal distributions.

First create a financial time series object from the supplied `whirlpool.dat` data file.

```
whrl = ascii2fts('whirlpool.dat', 1, 2, []);
```

Fill any missing values denoted with `NaN`s in `whrl` with values calculated using the linear method:

```
f_whrl = fillts(whrl);
```

Transform the nonnormally distributed filled data series `f_whrl` into a normally distributed one using Box-Cox transformation:

```
bc_whrl = boxcox(f_whrl);
```

Compare the result of the `Close` data series with a normal (Gaussian) probability distribution function as well as the nonnormally distributed `f_whrl`:

```
subplot(2, 1, 1);
hist(f_whrl.Close);
grid; title('Nonnormally Distributed Data');
subplot(2, 1, 2);
hist(bc_whrl.Close);
grid; title('Box-Cox Transformed Data');
```

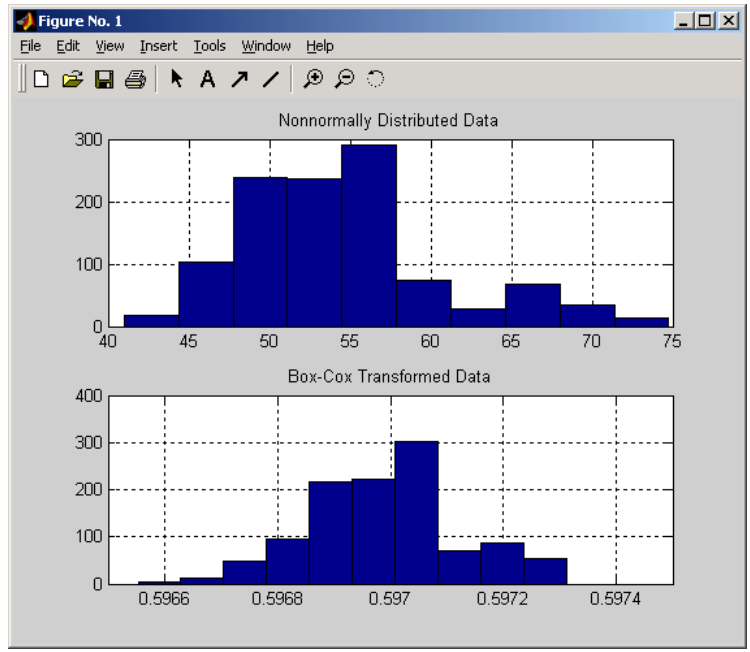


Figure 2-1: Box-Cox Transformation

The bar chart on the top represents the probability distribution function of the filled data series, `f_whrl`, which is the original data series `whrl` with the missing values interpolated using the linear method. The distribution is skewed towards the left (not normally distributed). The bar chart on the bottom is less skewed to the left. If you plot a Gaussian probability distribution

function (PDF) with similar mean and standard deviation, the distribution of the transformed data is very close to normal (Gaussian).

When you examine the contents of the resulting object `bc_whrl`, you find an identical object to the original object `whrl` but the contents are the transformed data series. If you have the Statistics Toolbox, you can generate a Gaussian PDF with mean and standard deviation equal to those of the transformed data series and plot it as an overlay to the second bar chart. In the next figure you can see that it is an approximately normal distribution.

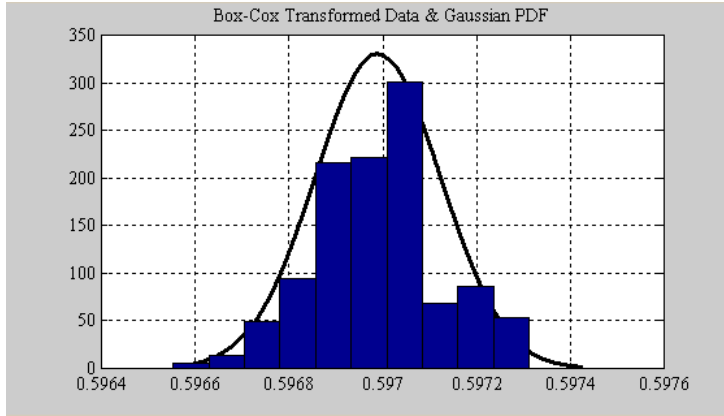


Figure 2-2: Overlay of Gaussian PDF

The next example uses the `smoothts` function to smooth a time series.

To begin, transform `ibm9599.dat`, a supplied data file, into a financial time series object:

```
ibm = ascii2fts('ibm9599.dat', 1, 3, 2);
```

Fill the missing data for holidays with data interpolated using the `fillts` function and the `Spline` fill method:

```
f_ibm = fillts(ibm, 'Spline');
```

Smooth the filled data series using the default Box (rectangular window) method:

```
sm_ibm = smoothts(f_ibm);
```

Now, plot the original and smoothed closing price series for IBM:

```
plot(f_ibm.CLOSE('11/01/97::02/28/98'), 'r')
datetick('x', 'mmmyy')
hold on
plot(sm_ibm.CLOSE('11/01/97::02/28/98'), 'b')
hold off
datetick('x', 'mmmyy')
legend('Filled', 'Smoothed')
title('Filled IBM Close Price vs. Smoothed Series')
```

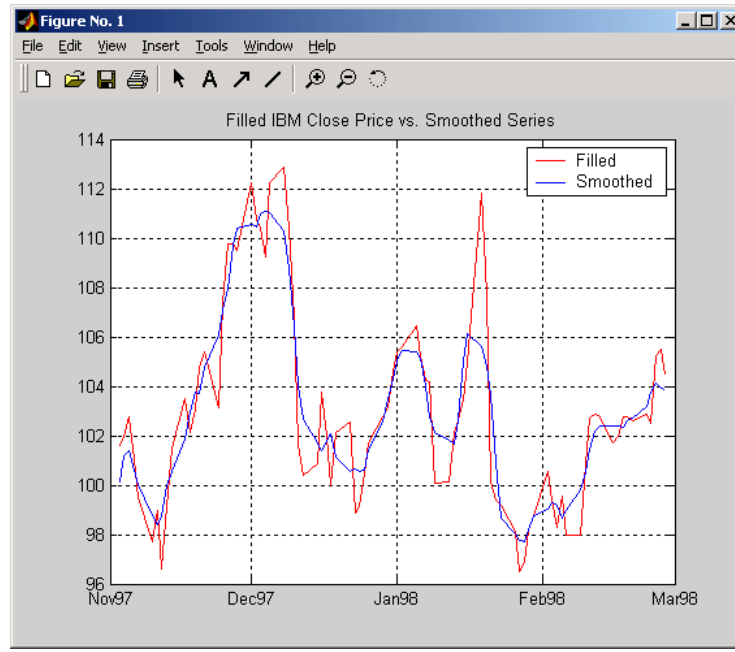


Figure 2-3: Smoothed Data Series

These examples give you an idea of what you can do with a financial time series object. This toolbox provides some MATLAB functions that have been overloaded to work directly with the these objects. The overloaded functions are those most commonly needed to work with time series data.

Demonstration Program

This example demonstrates a practical use of the Financial Time Series Toolbox, predicting the return of a stock from a given set of data. The data is a series of closing stock prices, a series of dividend payments from the stock, and an explanatory series (in this case a market index). Additionally, the example calculates the dividend rate from the stock data provided.

Note You can find a script M-file for this demonstration program in the directory `<matlab>/toolbox/ftseries/ftsdemos` on your MATLAB path. The script is named `predict_ret.m`.

To perform these computations follow these steps:

1. Load the data.
2. Create financial time series objects from the loaded data.
3. Create the series from dividend payment for adjusting the closing prices.
4. Adjust the closing prices and make them the spot prices.
5. Create the return series.
6. Regress the return series against the metric data (e.g., a market index) using the MATLAB \ operator.
7. Plot the results.
8. Calculate the dividend rate.

Load the Data

The data for this demonstration is found in the MAT-file `predict_ret_data.mat`:

```
load predict_ret_data.mat
```

The MAT-file contains six vectors:

- Dates corresponding to the closing stock prices, `sdates`
- Closing stock prices, `sdata`
- Dividend dates, `divdates`
- Dividend paid, `divdata`
- Dates corresponding to the metric data, `expdates`
- Metric data, `expdata`

Use the `whos` command to see the variables in your MATLAB workspace.

Create Financial Time Series Objects

It is advantageous to work with financial time series objects rather than with the vectors now in the workspace. By using objects, you can easily keep track of the dates. Also, you can easily manipulate the data series based on dates because the object keeps track of the administration of time series for you.

Use the object constructor `fints` to construct three financial time series objects.

```
t0 = fints(sdates, sdata, {'Close'}, 'd', 'Inc');
d0 = fints(divdates, divdata, {'Dividends'}, 'u', 'Inc');
x0 = fints(expdates, expdata, {'Metric'}, 'w', 'Index');
```

The variables `t0`, `d0`, and `x0` are financial time series objects containing the stock closing prices, dividend payments, and the explanatory data, respectively. To see the contents of an object, type its name at the MATLAB command prompt and press **Enter**. For example:

```
d0
```

```
d0 =
    'desc:'          'Inc'
    'freq:'          'Unknown (0)'
                ''                    ''
    'dates:  (4)'    'Dividends:  (4)'
    '04/15/99'       '0.2000'
    '06/30/99'       '0.3500'
    '10/02/99'       '0.2000'
    '12/30/99'       '0.1500'
```

Create Closing Prices Adjustment Series

The price of a stock is affected by the dividend payment. On the day before the dividend payment date, the stock price reflects the amount of dividend to be paid the next day. On the dividend payment date, the stock price is decreased by the amount of dividend paid. Create a time series that reflects this adjustment factor:

```
dadj1       = d0;
dadj1.dates = dadj1.dates-1;
```

Now create the series that adjust the prices at the day of dividend payment; this is an adjustment of `0`. You also need to add the previous dividend payment date since the stock price data reflect the period subsequent to that day; the previous dividend date was December 31, 1998:

```
dadj2             = d0;
dadj2.Dividends   = 0;
dadj2             = fillts(dadj2,'linear','12/31/98');
dadj2('12/31/98') = 0;
```

Combining the two objects above gives the data needed to adjust the prices. However, since the stock price data is daily data and the effect of the dividend is linearly divided during the period, use the `fillts` function to make a daily time series from the adjustment data. Use the dates from the stock price data to make the dates of the adjustment the same:

```
dadj3 = [dadj1; dadj2];
dadj3 = fillts(dadj3, 'linear', t0.dates);
```

Adjust Closing Prices and Make Them Spot Prices

The stock price recorded already reflects the dividend effect. To obtain the "correct" price, subtract the dividend amount from the closing prices. Put the result inside the same object `t0` with the data series name `Spot`.

To make sure that adjustments correspond, index into the adjustment series using the dates from the stock price series `t0`. Use the `datestr` command because `t0.dates` returns the dates in serial date format. Also, since the data series name in the adjustment series `dadj3` does not match the one in `t0`, use the function `fts2mat`:

```
t0.Spot = t0.Close - fts2mat(dadj3(datestr(t0.dates)));
```

Create Return Series

Now calculate the return series from the stock price data. A stock return is calculated by dividing the difference between the current closing price and the previous closing price by the previous closing price.

```
tret = (t0.Spot - lagts(t0.Spot, 1)) ./ lagts(t0.Spot, 1);
tret = chfield(tret, 'Spot', 'Return');
```

Ignore any warnings you receive during this sequence. Since the operation on the first line above preserves the data series name `Spot`, it has to be changed with the `chfield` command to reflect the contents correctly.

Regress Return Series Against Metric Data

The explanatory (metric) data set is a weekly data set while the stock price data is a daily data set. The frequency needs to be the same. Use `todaily` to convert the weekly series into a daily series. The constant needs to be included here to get the constant factor from the regression:

```
x1 = todaily(x0);
x1.Const = 1;
```

Get all the dates common to the return series calculated above and the explanatory (metric) data. Then combine the contents of the two series that have dates in common into a new time series:

```
dcommon = intersect(tret.dates, x1.dates);
regts0  = [tret(datestr(dcommon)), x1(datestr(dcommon))];
```

Remove the contents of the new time series that are not finite:

```
finite_regts0 = find(all(isfinite( fts2mat(regts0)), 2));
regts1        = regts0( finite_regts0 );
```

Now, place the data to be regressed into a matrix using the function `fts2mat`. The first column of the matrix corresponds to the values of the first data series in the object, the second column to the second data series, and so on. In this case, the first column is regressed against the second and third column:

```
DataMatrix = fts2mat(regts1);
XCoeff     = DataMatrix(:, 2:3) \ DataMatrix(:, 1);
```

Using the regression coefficients, calculate the predicted return from the stock price data. Put the result into the return time series `tret` as the data series `PredReturn`:

```
RetPred = DataMatrix(:,2:3) * XCoeff;
tret.PredReturn(datestr(regts1.dates)) = RetPred;
```

Plot the Results

Plot the results in a single figure window. The top plot in the window has the actual closing stock prices and the dividend-adjusted stock prices (spot prices). The bottom plot shows the actual return of the stock and the predicted stock return through regression:

```
subplot(2, 1, 1);
plot(t0);
title('Spot and Closing Prices of Stock');
subplot(2, 1, 2);
plot(tret);
title('Actual and Predicted Return of Stock');
```

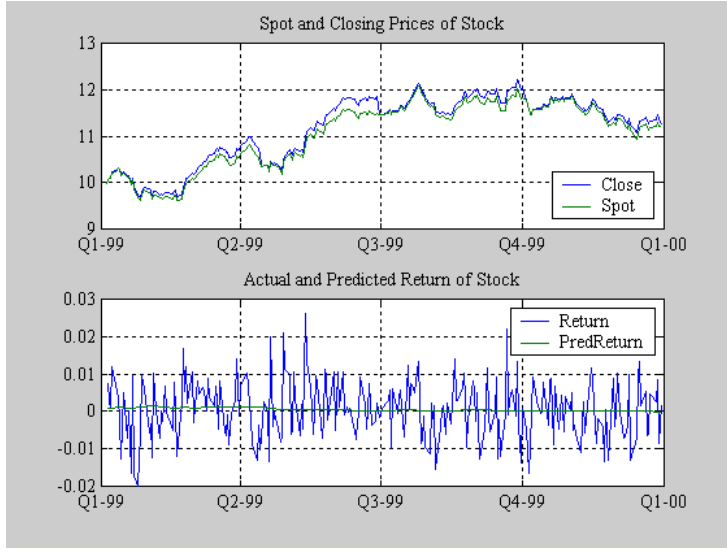


Figure 2-4: Closing Prices and Returns

Calculate the Dividend Rate

The last part of the task is to calculate the dividend rate from the stock price data. Calculate the dividend rate by dividing the dividend payments by the corresponding closing stock prices.

First check to see if you have the stock price data on all the dividend dates:

```
datestr(d0.dates, 2)

ans =

04/15/99
06/30/99
10/02/99
12/30/99

t0(datestr(d0.dates))

ans =

    'desc:'          'Inc'                       ''
    'freq:'          'Daily (1)'                 ''
                ''                ''             ''
    'dates:  (3)'    'Close:  (3)'    'Spot:  (3)'
    '04/15/99'       '10.3369'        '10.3369'
    '06/30/99'       '11.4707'        '11.4707'
    '12/30/99'       '11.2244'        '11.2244'
```

Note that stock price data for October 2, 1999 does not exist. The `fillts` function can overcome this situation; `fillts` allows you to insert a date and interpolate a value for the date from the existing values in the series. There are a number of interpolation methods. See `fillts` in the "Function Reference" for details.

Use `fillts` to create a new time series containing the missing date from the original data series. Then set the frequency indicator to daily:

```
t1 = fillts(t0,'nearest',d0.dates);
t1.freq = 'd';
```

Calculate the dividend rate:

```
tdr = d0./fts2mat(t1.Close(datestr(d0.dates)))

tdr =

    'desc:'          'Inc'
    'freq:'          'Unknown (0)'
                ''                    ''
    'dates:  (4)'    'Dividends:  (4)'
    '04/15/99'       '0.0193'
    '06/30/99'       '0.0305'
    '10/02/99'       '0.0166'
    '12/30/99'       '0.0134'
```

3

Technical Analysis

Introduction

Technical analysis (or charting) is used by some investment managers to help manage portfolios. Technical analysis relies heavily on the availability of historical data. Investment managers calculate different indicators from available data and plot them as charts. Observations of price, direction, and volume on the charts assist managers in making decisions on their investment portfolios.

The technical analysis functions in this toolbox are tools to help analyze your investments. The functions in themselves will not make any suggestions or perform any qualitative analysis of your investment.

Table 3-1: Technical Analysis: Oscillators

Function	Type
`adosc`	Accumulation/distribution oscillator
`chaikosc`	Chaikin oscillator
`macd`	Moving Average Convergence/Divergence
`stochosc`	Stochastic oscillator
`tsaccel`	Acceleration
`tsmom`	Momentum

Table 3-2: Technical Analysis: Stochastics

Function	Type
`chaikvolat`	Chaikin volatility
`fpctkd`	Fast stochastics
`spctkd`	Slow stochastics
`willpctr`	Williams %R

Table 3-3: Technical Analysis: Indexes

Function	Type
`negvolidx`	Negative volume index
`posvolidx`	Positive volume index
`rsindex`	Relative strength index

Table 3-4: Technical Analysis: Indicators

Function	Type
`adline`	Accumulation/distribution line
`bollinger`	Bollinger band
`hhigh`	Highest high
`llow`	Lowest low
`medprice`	Median price
`onbalvol`	On balance volume
`prcroc`	Price rate of change
`pvtrend`	Price-volume trend
`typprice`	Typical price
`volroc`	Volume rate of change
`wclose`	Weighted close
`willad`	Williams accumulation/distribution

The chapter provides examples for several types of technical analysis:

Examples

To illustrate some the technical analysis functions, this section uses the IBM stock price data contained in the supplied file `ibm9599.dat`. First create a financial time series object from the data using `ascii2fts`:

```
ibm = ascii2fts('ibm9599.dat', 1, 3, 2);
```

The time series data contains the open, close, high, and low prices, as well as the volume traded on each day. The time series dates start on January 3, 1995, and end on April 1, 1999, with some values missing for weekday holidays; weekend dates are not included.

Moving Average Convergence/Divergence (MACD)

Moving Average Convergence/Divergence (MACD) is an oscillator function used by technical analysts to spot overbought and oversold conditions. Look at the portion of the time series covering the three-month period between October 1, 1995 and December 31, 1995. At the same time fill any missing values due to holidays within the time period specified:

```
part_ibm = fillts(ibm('10/01/95::12/31/95'));
```

Now calculate the MACD, which when plotted produces two lines; the first line is the MACD line itself and the second is the nine-period moving average line:

```
macd_ibm = macd(part_ibm);
```

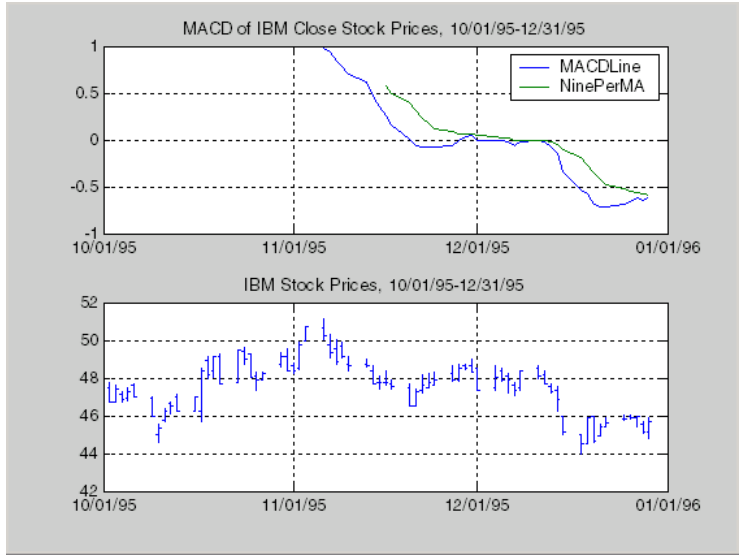
Note When you call `macd` without giving it a second input argument to specify a particular data series name, it searches for a closing price series named `Close` (in all combinations of letter cases). For more detail on the `macd` function, see `macd` in the "Function Reference."

Plot the MACD lines and the High-Low plot of the IBM stock prices in two separate plots in one window.

```
subplot(2, 1, 1);
plot(macd_ibm);
title('MACD of IBM Close Stock Prices, 10/01/95-12/31/95');
datetick('x', 'mm/dd/yy');
```

```
subplot(2, 1, 2);
highlow(part_ibm);
title('IBM Stock Prices, 10/01/95-12/31/95');
datetick('x', 'mm/dd/yy')
```

The following figure shows the result.



Williams %R

Williams %R is an indicator that measures overbought and oversold levels. The function `willpctr` is from the stochastics category. All the technical analysis functions can accept a different name for a required data series. If, for example, a function needs the high, low, and closing price series but your time series object does not have the data series names exactly as `High`, `Low`, and `Close`, you can specify the correct names as follows.

```
wpr = willpctr(tsobj, 14, 'HighName', 'Hi', 'LowName', 'Lo',...
'CloseName', 'Closing')
```

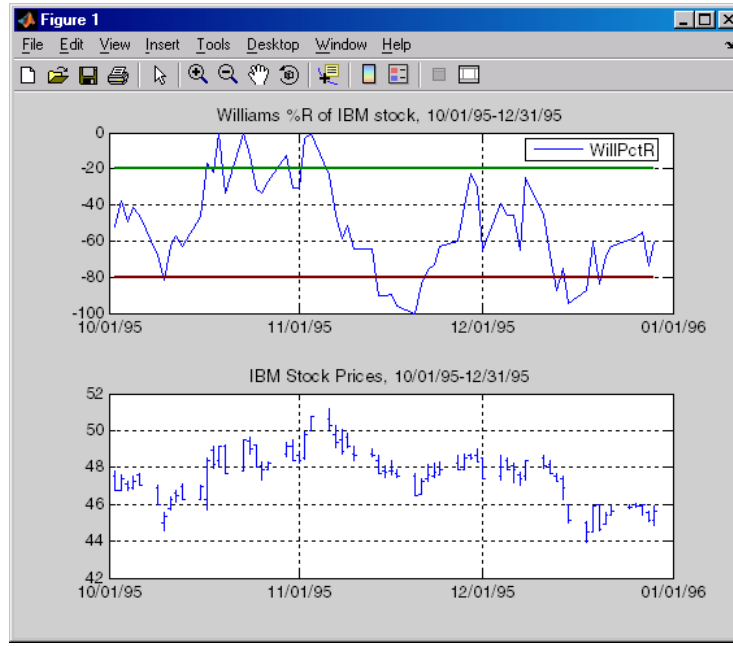
The function `willpctr` now assumes that your high price series is named `Hi`, low price series is named `Lo`, and closing price series is named `Closing`.

Since the time series object `part_ibm` has its data series names identical to the required names, name adjustments are not needed. The input argument to the function is only the name of the time series object itself.

Calculate and plot the Williams %R indicator for IBM along with the price range using these commands:

```
wpctr_ibm = willpctr(part_ibm);
subplot(2, 1, 1);
plot(wpctr_ibm);
title('Williams %R of IBM stock, 10/01/95-12/31/95');
datetick('x', 'mm/dd/yy');
hold on;
plot(wpctr_ibm.dates, -80*ones(1, length(wpctr_ibm)),...
'color', [0.5 0 0], 'linewidth', 2)
plot(wpctr_ibm.dates, -20*ones(1, length(wpctr_ibm)),...
'color', [0 0.5 0], 'linewidth', 2)
subplot(2, 1, 2);
highlow(part_ibm);
title('IBM Stock Prices, 10/01/95-12/31/95');
datetick('x', 'mm/dd/yy');
```

The next figure shows the results. The top plot has the Williams %R line plus two lines at -20% and -80%. The bottom plot is the High-Low plot of the IBM stock price for the corresponding time period.



Relative Strength Index (RSI)

The Relative Strength Index (RSI) is a momentum indicator that measures an equity's price relative to itself and its past performance. The function name is `rsindex`.

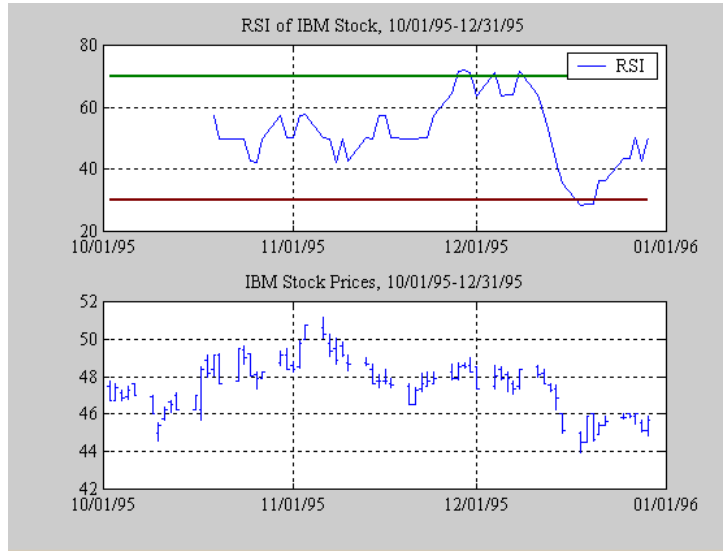
The `rsindex` function needs a series that contains the closing price of a stock. The default period length for the RSI calculation is 14 periods. This length can be changed by providing a second input argument to the function. Similar to the previous commands, if your closing price series is not named `Close`, you can provide the correct name.

Calculate and plot the RSI for IBM along with the price range using these commands:

```
rsi_ibm = rsindex(part_ibm);
subplot(2, 1, 1);
plot(rsi_ibm);
title('RSI of IBM stock, 10/01/95-12/31/95');
datetick('x', 'mm/dd/yy');
```

```
hold on;
plot(rsi_ibm.dates, 30*ones(1, length(wpctr_ibm)),...
'color', [0.5 0 0], 'linewidth', 2)
plot(rsi_ibm.dates, 70*ones(1, length(wpctr_ibm)),...
'color',[0 0.5 0], 'linewidth', 2)
subplot(2, 1, 2);
highlow(part_ibm);
title('IBM Stock Prices, 10/01/95-12/31/95');
datetick('x', 'mm/dd/yy');
```

The next figure shows the result.



On-Balance Volume (OBV)

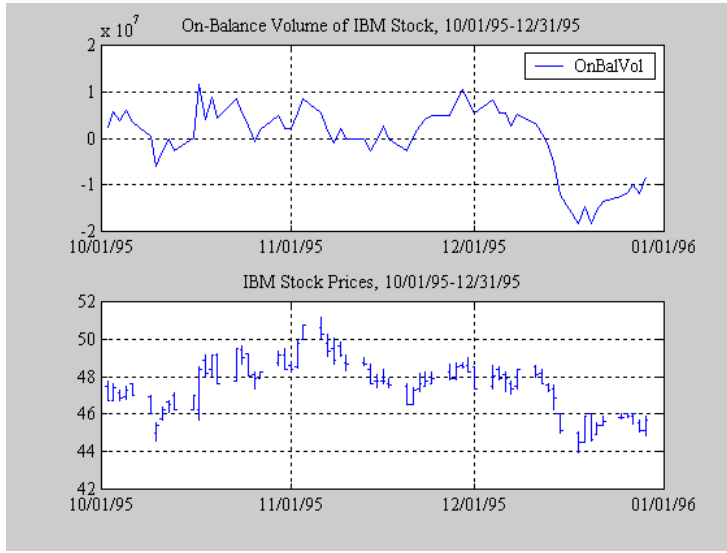
On-Balance Volume (OBV) relates volume to price change. The function `onbalvol` requires you to have the closing price (`Close`) series as well as the volume traded (`Volume`) series.

Calculate and plot the OBV for IBM along with the price range using these commands:

```
obv_ibm = onbalvol(part_ibm);
subplot(2, 1, 1);
```

```
plot(obv_ibm);
title('On-Balance Volume of IBM Stock, 10/01/95-12/31/95');
datetick('x', 'mm/dd/yy');
subplot(2, 1, 2);
highlow(part_ibm);
title('IBM Stock Prices, 10/01/95-12/31/95');
datetick('x', 'mm/dd/yy');
```

The next figure shows the result.



4

Graphical User Interface

Financial Time Series Graphical User Interface (GUI)

Use the financial time series graphical user interface (GUI) to analyze your time series data and display the results graphically without resorting to the command line. The GUI lets you visualize the data and the results at the same time. Through the GUI you have access to the full functionality of the Financial Time Series Toolbox.

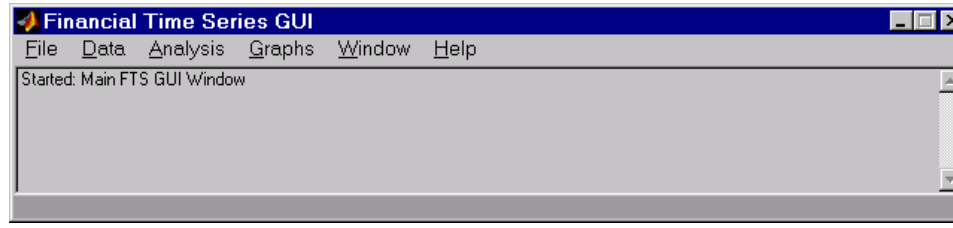
"Using the Financial Time Series GUI" on page 4-8 provides a discussion about how to use this GUI.

Main Window

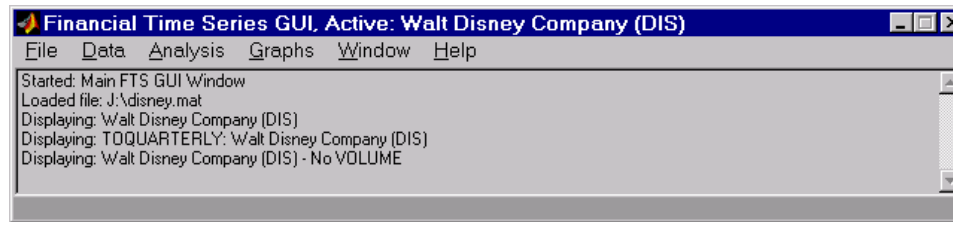
Start the financial time series GUI with the command

```
ftsgui
```

The main financial time series GUI window appears.

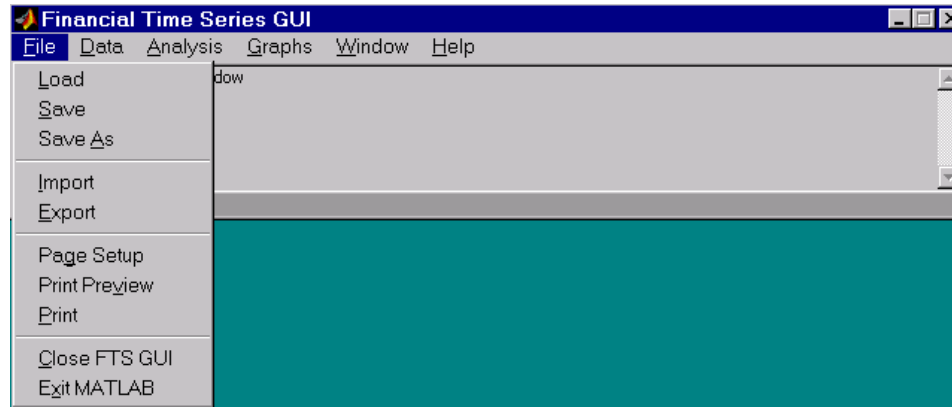


The title bar acts as an active time series object indicator (indicates the currently active financial time series object). For example, if you load the file `disney.mat` and want to use the time series data in the file `dis`, the title bar on the main GUI would read as shown.



The menu bar consists of six menu items: **File**, **Data**, **Analysis**, **Graphs**, **Window**, and **Help**. Under the menu bar is a status box that displays the steps you are doing.

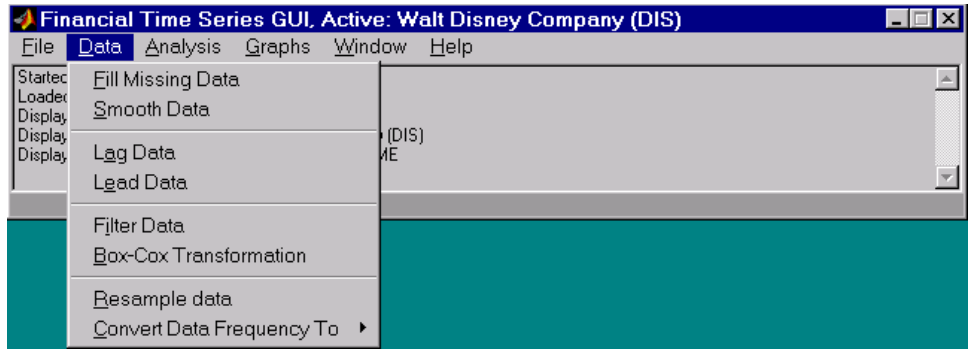
File Menu



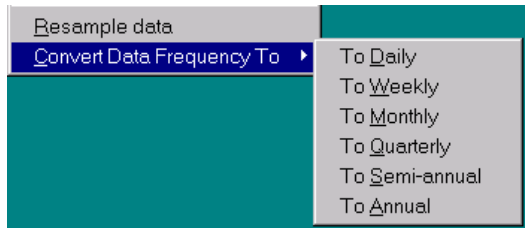
The **File** menu contains the commands for input and output. You can read and save (**Load**, **Save**, and **Save As**) MATLAB MAT-files, ASCII (text) data files, as well as import (**Import**) Microsoft Excel XLS files. MATLAB does not support the export of XLS files at this time.

The **File** menu also contains the printing suite (**Page Setup**, **Print Preview**, and **Print**). Lastly, from this menu you can close the GUI itself (**Close FTS GUI**) and quit MATLAB (**Exit MATLAB**).

Data Menu

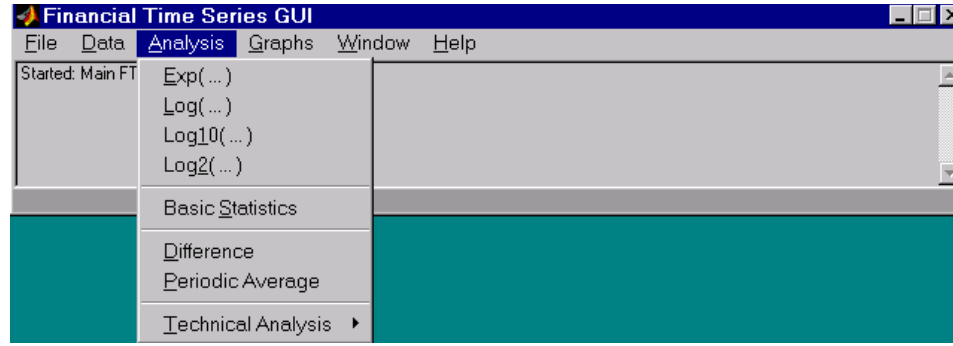


The **Data** menu item provides a collection of data manipulation functions and data conversion functions.



To use any of the functions here, make sure that the correct financial time series object is displayed in the title bar of the main GUI window.

Analysis Menu

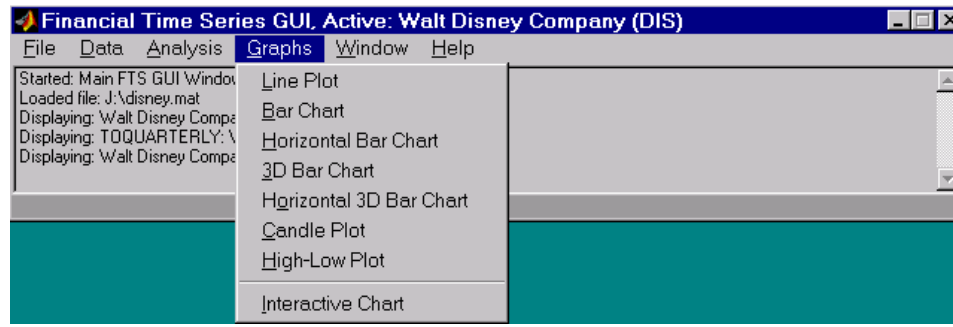


The **Analysis** menu provides

- A set of exponentiation and logarithmic functions.
- Statistical tools (**Basic Statistics**), which calculate and display the minimum, maximum, average (mean), standard deviation, and variance of the current (active) time series object; these basic statistics numbers are displayed in a dialog window.
- Data difference (**Difference**) and periodic average (**Periodic Average**) calculations. Data difference generates a vector of data that is the difference between the first data point and the second, the second and the third, etc. The periodic average function calculates the average per defined length period, for example, averages of every five days.
- Technical analysis functions. See Chapter 3, “Technical Analysis,” for a list of the provided technical analysis functions.

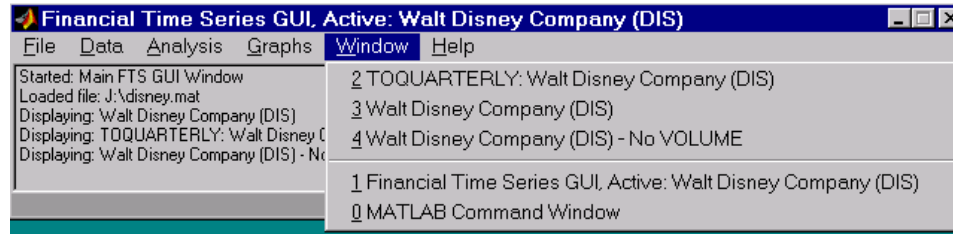
As with the **Data** menu, to use any of the **Analysis** menu functions, make sure that the correct financial time series object is displayed in the title bar of the main GUI window.

Graphs Menu



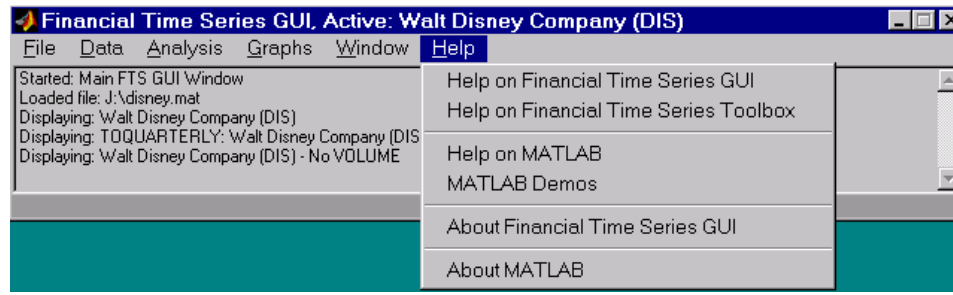
The **Graphs** menu contains functions that graphically display the current (active) financial time series object. You can also invoke the interactive charting function (`chartfts`) from this menu.

Window Menu



The **Window** menu lists open windows under the current MATLAB session.

Help Menu

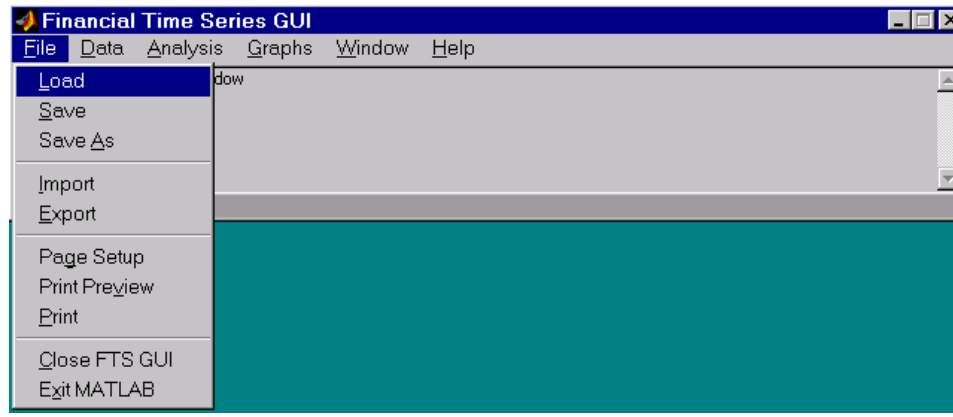


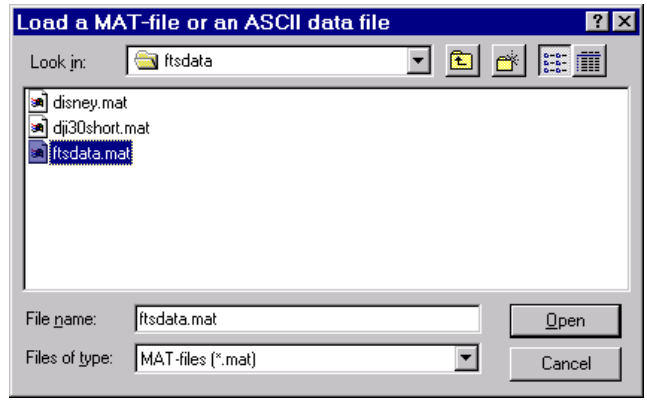
The **Help** menu provides a standard set of Help menu links.

Using the Financial Time Series GUI

Getting Started

To use the Financial Time Series GUI, first load (or import) the time series data. For example, if your data is in a MATLAB MAT-file, select **Load** from the **File** menu.



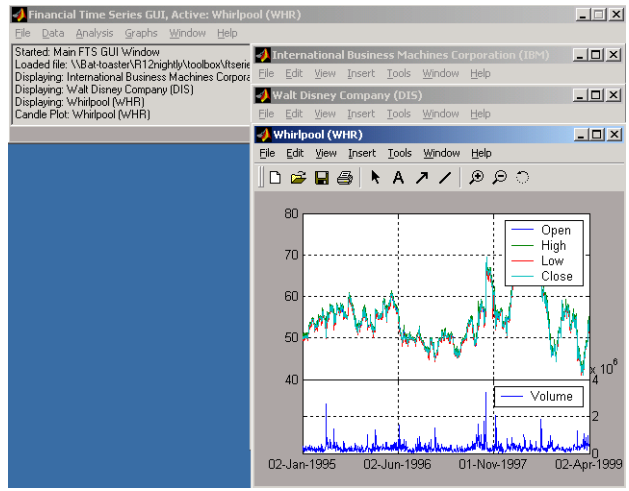


For illustration purposes, choose the file `ftsdata.mat` from the dialog presented.

If you don't see the MAT-file, look in the directory `<matlab>\toolbox\ftseries\ftsdata`, where `<matlab>` is the MATLAB root directory (the directory where MATLAB is installed).

Note Data loaded through the Financial Time Series GUI is not available in the MATLAB workspace. You can access this data only through the GUI itself, not with any MATLAB command-line functions.

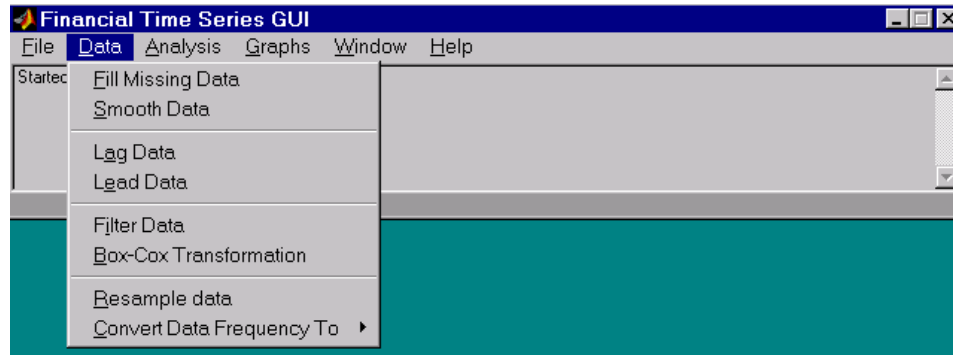
Each financial time series object inside the MAT-file is presented as a line plot in a separate window. The status window is updated accordingly.



Whirlpool (WHR) is the last plot displayed, as indicated on the title bar of the main window.

Data Menu

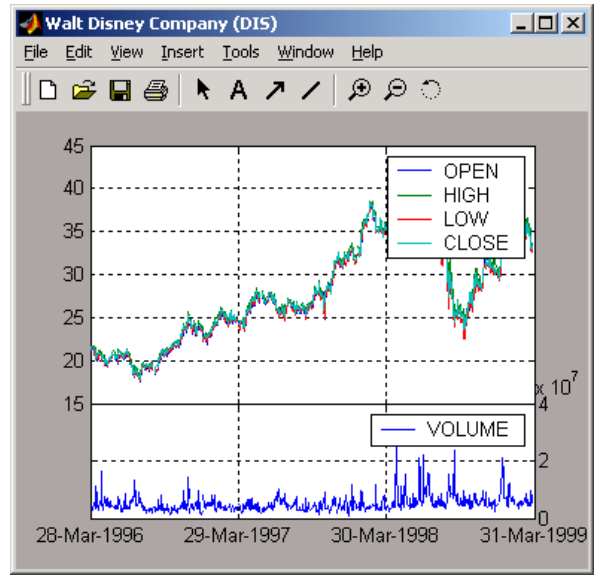
The **Data** menu provides functions that manipulate time series data.



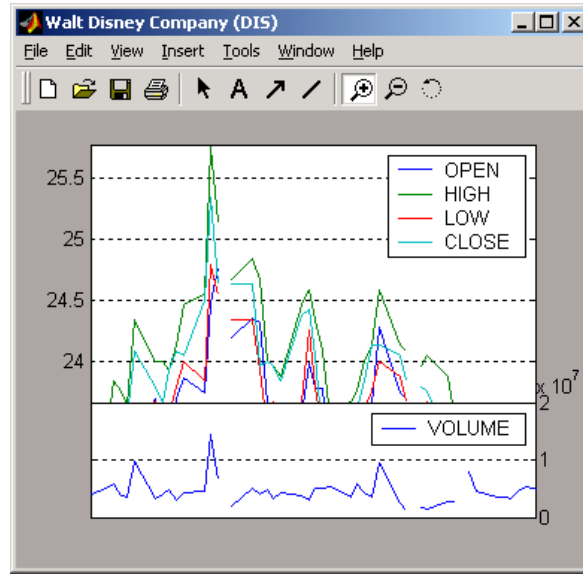
Here are some example tasks that illustrate the use of the functions on this menu.

Fill Missing Data

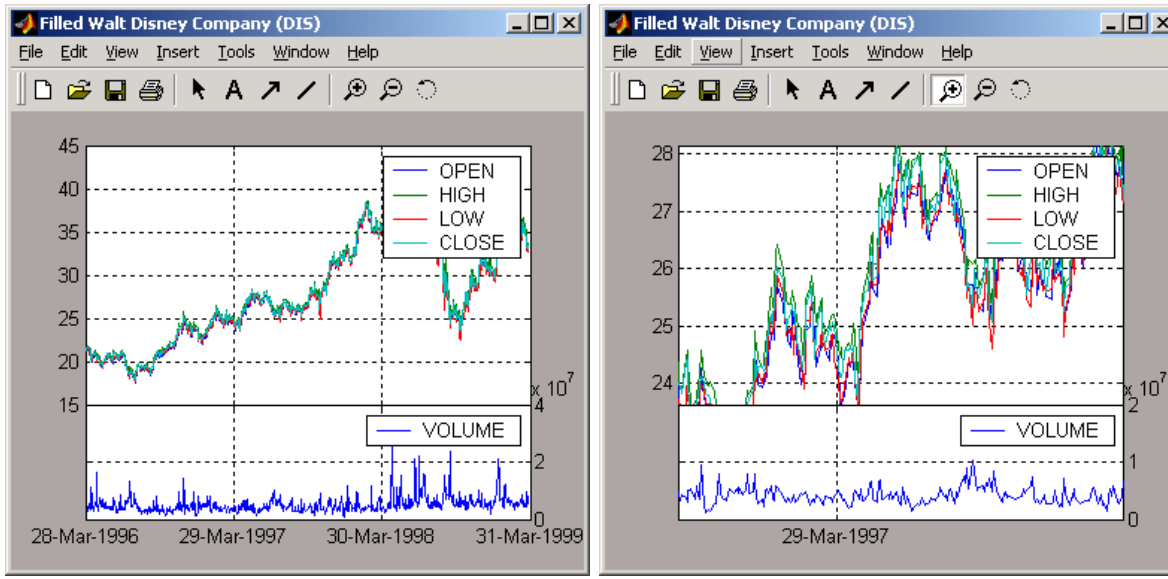
First, look at filling missing data. The **Fill Missing Data** item uses the toolbox function `fillts`. With the data loaded from the file `ftsdata`, you have three time series: IBM Corp. (IBM), Walt Disney Co. (DIS), and Whirlpool (WHR). Click on the window that shows the time series data for Walt Disney Co. (DIS).



To view any missing data in this time series data set, zoom into the plot using the Zoom tool (the magnifying glass icon with the plus sign) from the toolbar and select a region.



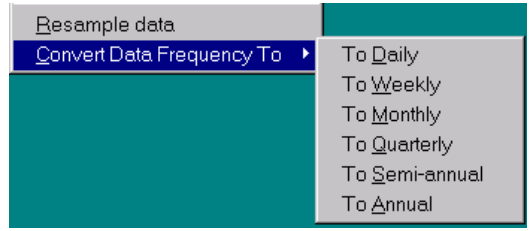
The gaps represent the missing data in the series. To fill these gaps, go to the **Data** menu and choose **Fill Missing Data**. This selection automatically fills the gaps and generates a new plot that displays the filled time series data.



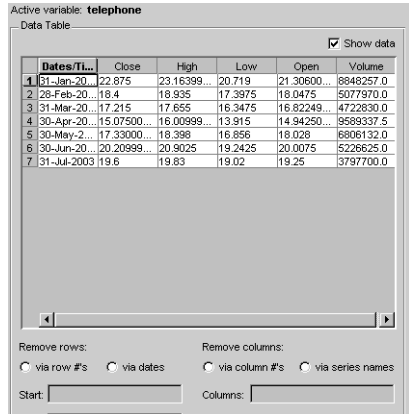
You cannot see the filled gaps when you display the entire data set. However, when you zoom into the plot, you see that the gaps have been eliminated. Note that the title bar has changed; the title has been prefixed with the word **Filled** to reflect the filled time series data.

Frequency Conversion

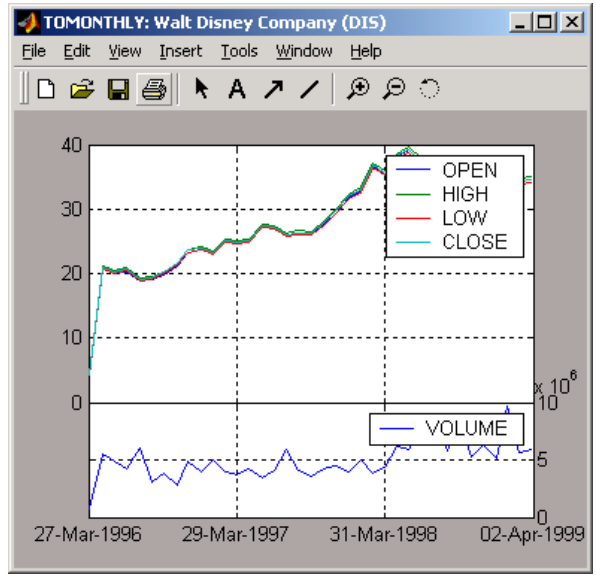
The **Data** menu also provides access to frequency conversion functions.



This example changes the DIS time series data frequency from daily to monthly. Close the Filled Walt Disney Company (DIS) window, and click on the Walt Disney Company (DIS) window to make it active (current) again. Then, from the **Data** menu, choose **Convert Data Frequency To** and **To Monthly**.



A new figure window displays the result of this conversion.

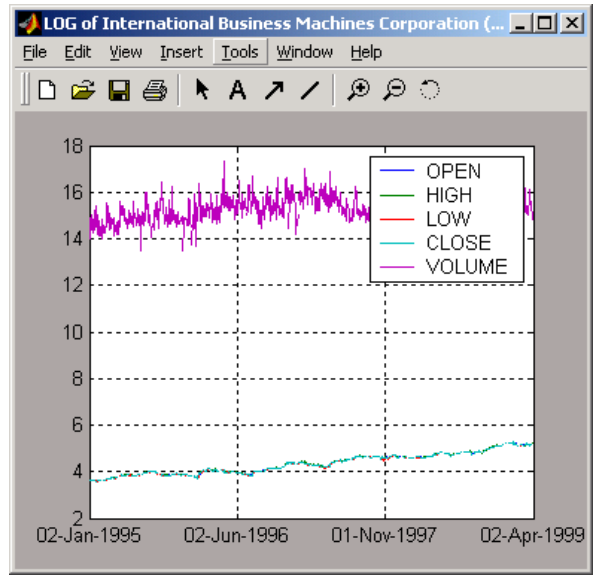


The title reflects that the data displayed had its frequency changed to monthly.

Analysis Menu

The **Analysis** menu provides functions that analyze time series data, including the technical analysis functions. (See Chapter 3, “Technical Analysis,” for a complete list of the technical analysis functions and several usage examples.)

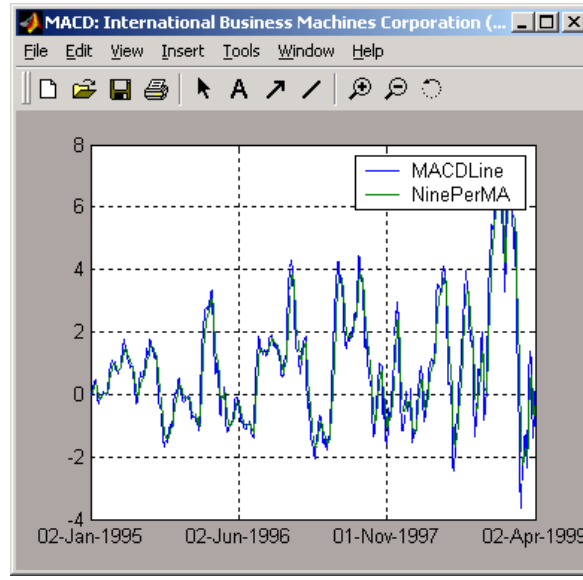
For example, you can use the **Analysis** menu to calculate the natural logarithm (`log`) of the data contained within the data set `ftsdata.mat`. This data file provides time series data for IBM (IBM), Walt Disney (DIS), and Whirlpool (WHR). Click on the window displaying the data for IBM Corporation (IBM) to make it active (current). Then choose the **Analysis** menu, followed by the **Log(...)** menu item. The result appears in its own window.



Close the above window and click again on the IBM data window to make it active (current).

Note Before proceeding with any time series analysis, make certain that the title bar confirms that the active data series is the correct one.

From the **Analysis** menu on the main window, choose **Technical Analysis**, and the **MACD** item. The result, again, is displayed in its own window.



Other analysis functions work similarly.

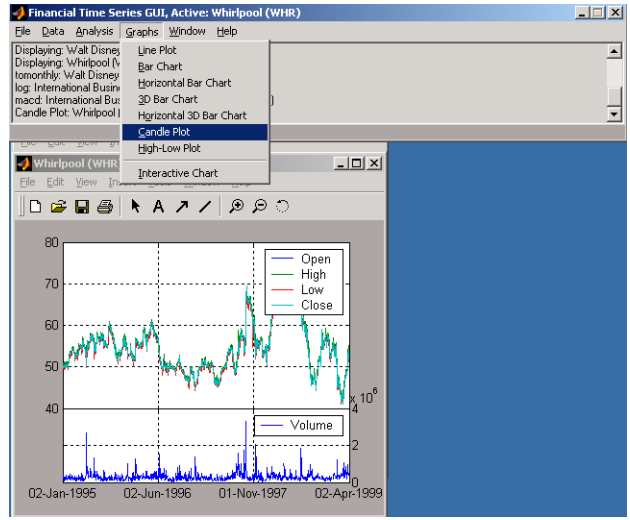
Graphs Menu

The **Graphs** menu displays time series data using the provided graphics functions. Included in the **Graphs** menu are several types of bar charts (`bar`, `barh`, `bar3`, `bar3h`), line plot (`plot`), candle plot (`candle`), and High-Low plot (`highlow`). The **Graphs** menu also provides access to the interactive charting function, `chartfts`.

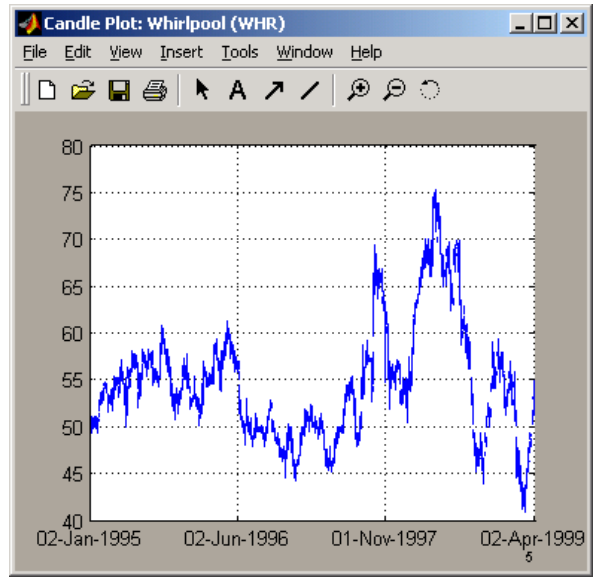
Candle Plot

For example, you can display the candle plot of a set of time series data and invoke the interactive chart on the same data set.

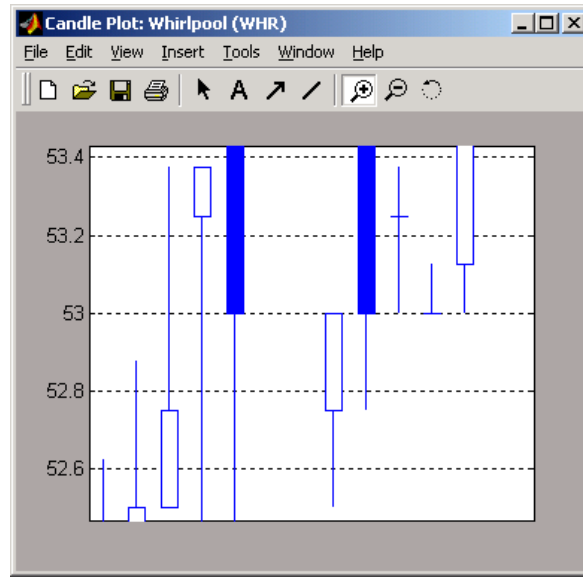
Load the `ftsdata.mat` data set, and click on the window that displays the Whirlpool (WHR) time series data to make it active (current). From the main window choose the **Graphs** menu and **Candle Plot** menu item.



The result is shown below.

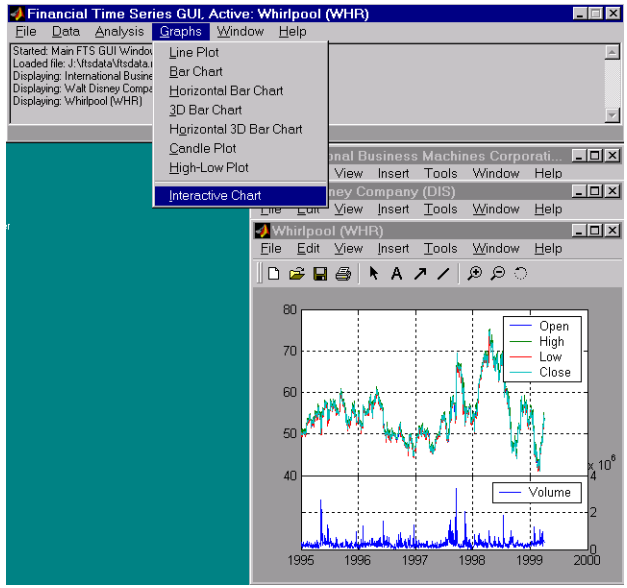


This does not look much like a candle plot because there are too many data points in the data set. All the candles are too compressed for effective viewing. However, when you zoom into a region of this plot, the candles become apparent.

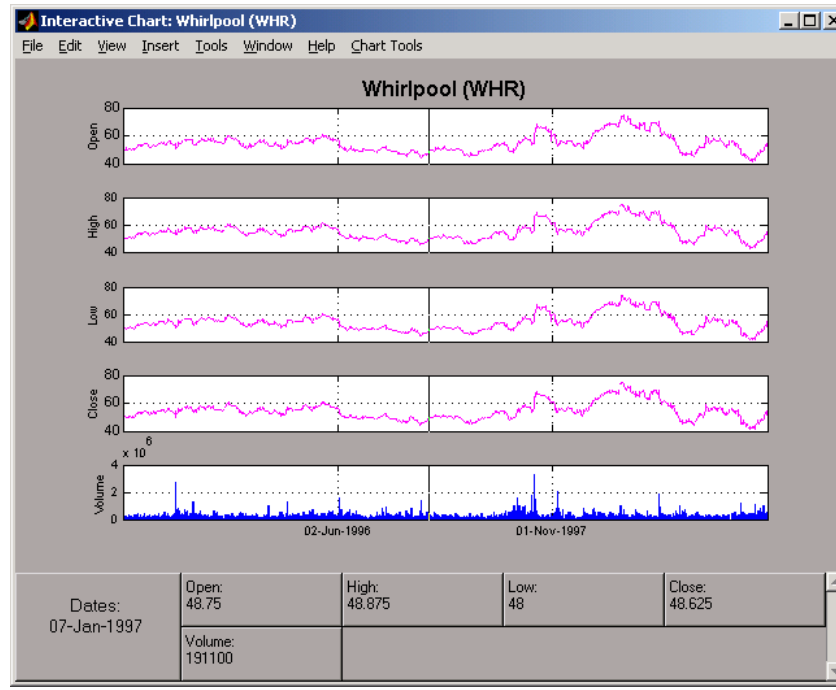


Interactive Chart

To create an interactive chart (`chartfts`) on the Whirlpool data, click on the window that displays the Whirlpool (WHR) data to make it active (current). Then, go to the **Graphs** menu and choose **Interactive Chart**.



The chart that results is shown below.



You can use this interactive chart as if you had invoked it with the `chartfts` command from the MATLAB command line. For a tutorial on the use of `chartfts`, see "Visualizing Financial Time Series Objects" on page 1-17.

Saving Time Series Data

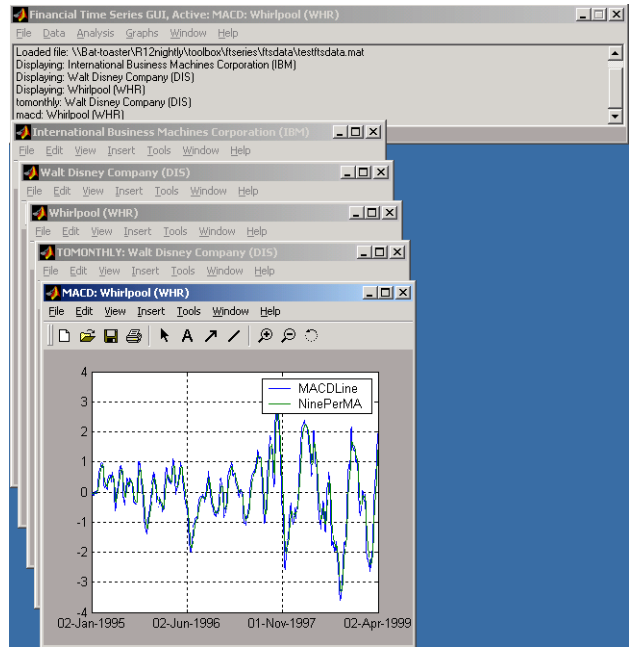
The **Save** and **Save As** items on the main window **File** menu let you save the time series data that results from your analyses and computations. These items save *all* time series data that has been loaded or processed during the current session, even if the window displaying the results of a computation has previously been dismissed.

Note The **Save** and **Save As** items on the **File** menu of individual plot windows are not available for use.

You can save your time series data in two ways:

- Into the latest MAT-file loaded (**Save**)
- Into a MAT-file chosen (or named) from the dialog window (**Save As**)

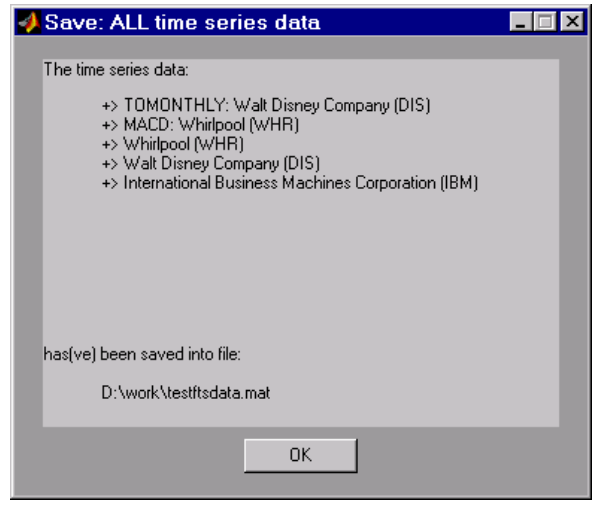
To illustrate this, start by loading the data file `testftsdata.mat` (located in `<matlab>/toolbox/ftseries/ftsdata`). Then, convert the Disney (DIS) data from daily (the original frequency) to monthly data. Next, run the MACD analysis on the Whirlpool (WHR) data. You now have a set of five open figure windows.



Saving into the Original File (Save)

To save the data back into the original file (`testftsdata.mat`), choose **Save** on the **File** menu.

A confirmation window appears. It confirms that the data has been saved in the latest MAT-file loaded (`testftsdata.mat` in this example).

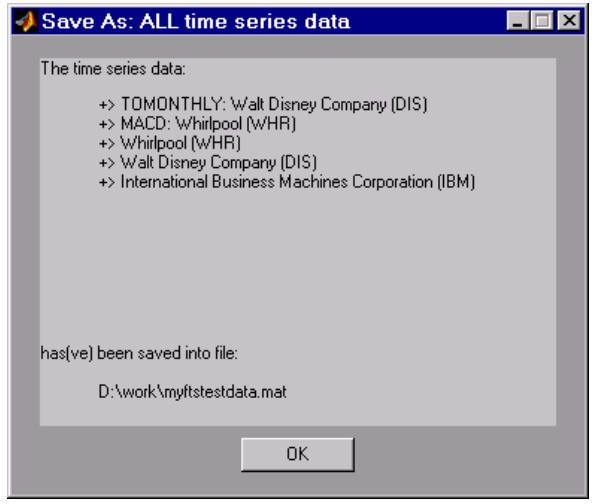


Saving into a New File (Save As)

To save the data in a different file, choose **Save As** from the **File** menu.

The dialog box that appears lets you choose an existing MAT-file from a list or type in the name of a new MAT-file you want to create.

After you click the **Save** button, another confirmation window appears.



This confirmation window indicates that the data has been saved in a new file named `myftstestdata.mat`.

5

Function Reference

Functions - Categorical List

This section provides detailed descriptions of the functions in the Financial Time Series Toolbox. The categories of functions described are:

- "Financial Time Series Object and File Construction"
- "Arithmetic Functions"
- "Mathematical Functions"
- "Utility Functions"
- "Data Transformation Functions"
- "Indicator Functions"
- "Calendar Functions"
- "Plotting Functions"
- "Graphical User Interface Function"
- "Financial Time Series Object Management Function"
- "Information Retrieval Functions"
- "Obsolete Functions"

Financial Time Series Object and File Construction

`ascii2fts`	Create financial time series object from ASCII data file
`fints`	Construct financial time series object
`fts2ascii`	Write elements of time series data into an ASCII file
`ftsnew2old`	Convert Version 2 time series object to Version 1
`fts2mat`	Convert to matrix
`ftsold2new`	Convert Version 1 time series object to Version 2

Arithmetic Functions

`end`	Last date entry
`horzcat`	Concatenate financial time series objects horizontally
`length`	Get number of dates (rows)
`minus`	Financial time series subtraction
`mrdivide`	Financial time series matrix division
`mtimes`	Financial time series matrix multiplication
`plus`	Financial time series addition
`power`	Financial time series power
`rdivide`	Financial time series division
`size`	Get number of dates and data series
`subsasgn`	Content assignment
`subsref`	Subscripted reference
`times`	Financial time series multiplication
`uminus`	Unary minus of financial time series object
`uplus`	Unary plus of financial time series object
`vertcat`	Concatenate financial time series objects vertically

Mathematical Functions

`cumsum`	Cumulative sum
`exp`	Exponential values
`hist`	Histogram
`log`	Natural logarithm
`log2`	Base 2 logarithm
`log10`	Common logarithm
`max`	Maximum value
`mean`	Arithmetic average
`min`	Minimum value
`std`	Standard deviation

Utility Functions

`chfield`	Change data series name
`extfield`	Extract data series
`fetch`	Extract data from financial time series object
`fieldnames`	Get names of fields
`freqnum`	Convert string frequency indicator to numeric frequency indicator
`freqstr`	Convert numeric frequency indicator to string representation
`ftsbound`	Start and end dates
`getfield`	Get content of a specific field
`getnameidx`	Find name in list
`iscompatible`	Structural equality
`isequal`	Multiple object equality
`isfield`	Check if a string is a field name

`rmfield`	Remove data series
`setfield`	Set content of a specific field
`sortfts`	Sort financial time series

Data Transformation Functions

`boxcox`	Box-Cox transformation
`convertto`	Convert to specified frequency
`diff`	Differencing
`fillts`	Fill missing values in time series
`filter`	Linear filtering
`lagts`	Lag time series object
`leadts`	Lead time series object
`peravg`	Periodic average
`resamplets`	Downsample data
`smoothts`	Smooth data
`toannual`	Convert to annual
`todaily`	Convert to daily
`todecimal`	Fractional to decimal conversion
`tomonthly`	Convert to monthly
`toquarterly`	Convert to quarterly
`toquoted`	Decimal to fractional conversion
`tosemi`	Convert to semiannual
`toweekly`	Convert to weekly
`tsmovavg`	Moving average

Indicator Functions

`adline`	Accumulation/Distribution line
`adosc`	Accumulation/Distribution oscillator
`bollinger`	Bollinger band
`chaikosc`	Chaikin oscillator
`chaikvolat`	Chaikin volatility
`fpctkd`	Fast stochastics
`hhigh`	Highest high
`llow`	Lowest low
`macd`	Moving Average Convergence/Divergence (MACD)
`medprice`	Median price
`negvolidx`	Negative volume index
`onbalvol`	On-Balance Volume (OBV)
`posvolidx`	Positive volume index
`prcroc`	Price rate of change
`pvtrend`	Price and Volume Trend (PVT)
`rsindex`	Relative Strength Index (RSI)
`spctkd`	Slow stochastics
`stochosc`	Stochastic oscillator
`tsaccel`	Acceleration between periods
`tsmom`	Momentum between periods
`typprice`	Typical price
`volroc`	Volume rate of change
`wclose`	Weighted close
`willad`	Williams Accumulation/Distribution line
`willpctr`	Williams %R

Calendar Functions

`busdays`	Business days in serial date format

Plotting Functions

`bar`	Bar chart
`bar3`	Three-dimensional bar chart
`bar3h`	Three-dimensional bar chart (horizontal)
`barh`	Bar chart (horizontal)
`candle`	Candle plot
`chartfts`	Interactive display
`highlow`	High-Low plot
`plot`	Plot data series

Graphical User Interface Function

`ftsgui`	Financial time series graphical user interface

Financial Time Series Object Management Function

`ftsmanager`	Create, display, and modify financial time series objects

Information Retrieval Functions

`display`	Display financial time series object
`fintsver`	Determine version
`ftsinfo`	Financial time series object information
`ftsuniq`	Determine uniqueness
`issorted`	Check if dates and times are monotonically increasing

Obsolete Functions

The function `flipud` is obsolete, and its description has been removed from the documentation. The function `fts2mtx` has been renamed `fts2mat`. For compatibility purposes the original functions remain in the product.

Type `help @fints/`*`function_name`* at the MATLAB command line for a description.

Functions — Alphabetical List

This section contains function reference pages listed alphabetically.

adline

Purpose

Accumulation/Distribution line

Syntax

```
adln = adline(highp, lowp, closep, tvolume)
adln = adline([highp lowp  closep  tvolume])
adlnts = adline(tsobj)
adlnts = adline(tsobj, ParameterName, ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`closep`	Closing price (vector)
`tvolume`	Volume traded (vector)
`tsobj`	Time series object

Description

`adln = adline(highp, lowp, closep, tvolume)` computes the Accumulation/Distribution line for a set of stock price and volume traded data. The prices required for this function are the high (`highp`), low (`lowp`), and closing (`closep`) prices.

`adln = adline([highp lowp closep tvolume])` accepts a four-column matrix as input. The first column contains the high prices, the second contains the low prices, the third contains the closing prices, and the fourth contains the volume traded.

`adlnts = adline(tsobj)` computes the Williams Accumulation/Distribution line for a set of stock price data contained in the financial time series object `tsobj`. The object must contain the high, low, and closing prices plus the volume traded. The function assumes that the series are named `High`, `Low`, `Close`, and `Volume`. All are required. `adlnts` is a financial time series object with the same dates as `tsobj` but with a single series named `ADLine`.

`adlnts = adline(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

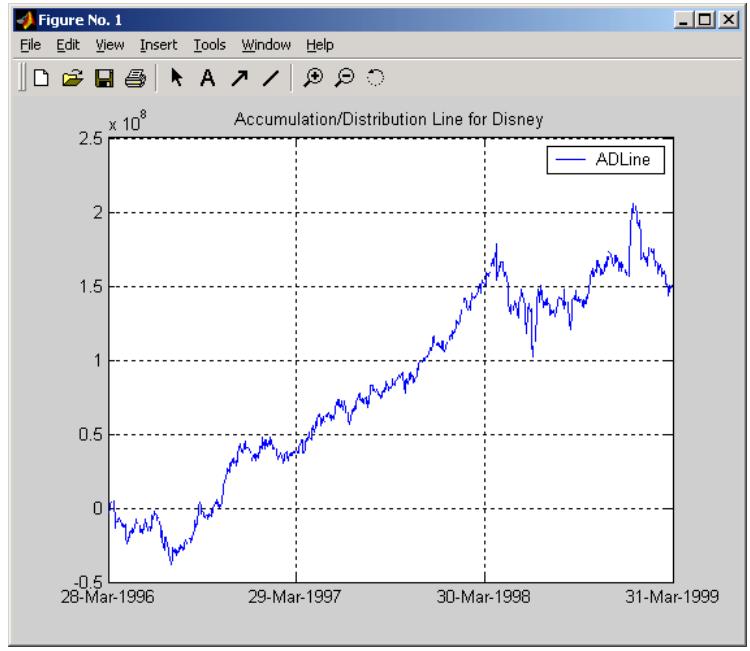
- `HighName`: high prices series name
- `LowName`: low prices series name
- `CloseName`: closing prices series name
- `VolumeName`: volume traded series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the Accumulation/Distribution line for Disney stock and plot the results:

```
load disney.mat
dis_ADLine = adline(dis)
plot(dis_ADLine)
title('Accumulation/Distribution Line for Disney')
```



See Also

`adosc`, `willad`, `willpctr`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 56 - 58.

Purpose

Accumulation/Distribution oscillator

Syntax

```
ado = adosc(highp, lowp, openp, closep)
ado = adosc([highp lowp openp closep])
adots = adosc(tsobj)
adots = adosc(tsojb, ParameterName, ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`openp`	Opening price (vector)
`closep`	Closing price (vector)
`tsobj`	Time series object

Description

`ado = adosc(highp, lowp, openp, closep)` returns a vector, `ado`, that represents the Accumulation/Distribution (A/D) oscillator. The A/D oscillator is calculated based on the high, low, opening, and closing prices of each period. Each period is treated individually.

`ado = adosc([highp lowp openp closep])` accepts a four column matrix as input. The order of the columns must be high, low, opening, and closing prices.

`adots = adosc(tsobj)` calculates the Accumulation/Distribution (A/D) oscillator, `adots`, for the set of stock price data contained in the financial time series object `tsobj`. The object must contain the high, low, opening, and closing prices. The function assumes that the series are named `High`, `Low`, `Open`, and `Close`. All are required. `adots` is a financial time series object with similar dates to `tsobj` and only one series named `ADOsc`.

`adots = adosc(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name- parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

- `HighName`: high prices series name
- `LowName`: low prices series name

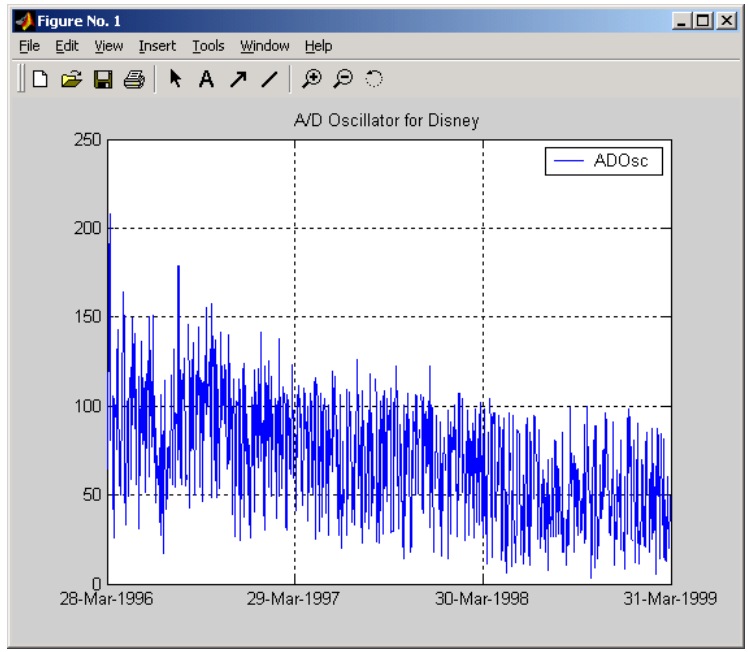
- `OpenName`: opening prices series name
- `CloseName`: closing prices series name

Parameter values are the strings that represents the valid parameter names.

Examples

Compute the Accumulation/Distribution oscillator for Disney stock and plot the results:

```
load disney.mat
dis_ADOsc = adosc(dis)
plot(dis_ADOsc)
title('A/D Oscillator for Disney')
```



See Also

`adline`, `willad`

Purpose

Create financial time series object from ASCII data file

Syntax

```
tsobj = ascii2fts(filename, descrow, colheadrow, skiprows)
tsobj = ascii2fts(filename, timedata, descrow, colheadrow, skiprows)
```

Arguments

`filename`	ASCII data file
`descrow`	(Optional) Row number in the data file that contains the description to be used for the description field of the financial time series object
`colheadrow`	(Optional) Row number that has the column headers/ names
`skiprows`	(Optional) Scalar or vector of row numbers to be skipped in the data file
`timedata`	Set to `'T'` if time-of-day data is present in the ASCII data file or to `'NT'` if no time-of-day data is present.

Description

`tsobj = ascii2fts(filename, descrow, colheadrow, skiprows)` creates a financial time series object `tsobj` from the ASCII file named `filename`. This form of the function can only read a data file without time-of-day information and create a financial time series object without time information. If time information is present in the ASCII file, an error message appears.

The general format of the text data file is

- Can contain header text lines.
- Can contain column header information. The column header information must immediately precede the data series columns unless `skiprows` is specified.
- Leftmost column must be the date column.
- Dates must be in a valid date string format:
  - `'ddmmmyy'` or `'ddmmmyyyy'`
  - `'mm/dd/yy'` or `'mm/dd/yyyy'`
  - `'dd-mmm-yy'` or `'dd-mmm-yyyy'`
  - `'mmm.dd,yy'` or `'mmm.dd,yyyy'`

- Each column must be separated either by spaces or a tab.

`tsobj = ascii2fts(filename, timedata, descrow, colheadrow, skiprows)` creates a financial time series object containing time-of-day data. Set `timedata` to `'T'` to create a financial time series object containing time-of-day data.

Examples

Example 1. If your data file contains no description or column header rows,

```
1/3/95   36.75   36.9063   36.6563   36.875    1167900
1/4/95   37      37.2813   36.625    37.1563   1994700  ...
```

you can create a financial time series object from it with the simplest form of the `ascii2fts` function:

```
myinc = ascii2fts('my_inc.dat');

myinc =

desc:  my_inc.dat
freq:  Unknown (0)

'dates:  (2)'  'series1: (2)'  'series2: (2)'   'series3: (2)'...
'03-Jan-1995'  [   36.7500]    [    36.9063]    [   36.6563]
'04-Jan-1995'  [        37]    [    37.2813]    [   36.6250]
```

Example 2: If your data file contains description and column header information with the data series immediately following the column header row,

```
International Business Machines Corporation (IBM)
Daily prices (1/3/95 to 4/5/99)
DATE     OPEN    HIGH      LOW       CLOSE     VOLUME
1/3/95   36.75   36.9063   36.6563   36.875    1167900
1/4/95   37      37.2813   36.625    37.1563   1994700  ...
```

you must specify the row numbers containing the description and column headers:

```
ibm = ascii2fts('ibm9599.dat', 1, 3);
```

```
ibm =

desc:  International Business Machines Corporation (IBM)
freq:  Unknown (0)
'dates: (2)'  'OPEN: (2)'    'HIGH: (2)'    'LOW: (2)' ...
'03-Jan-1995'  [  36.7500]    [   36.9063]    [  36.6563]
'04-Jan-1995'  [       37]    [   37.2813]    [  36.6250]
```

Example 3: If your data file contains rows between the column headers and the data series, e.g.,

```
Staples, Inc. (SPLS)
Daily prices
DATE     OPEN    HIGH     LOW      CLOSE    VOLUME
Starting date: 04/08/1996
Ending date:   04/07/1999
4/8/96   19.50   19.75    19.25    19.375   548500
4/9/96   19.75   20.125   19.375   20       1135900  ...
```

you need to indicate to `ascii2fts` the rows in the file that must be skipped. Assume that you have called the data file containing the Staples data above `staples.dat`. The command

```
spls = ascii2fts('staples.dat', 1, 3, [4 5]);
```

indicates that the fourth and fifth rows in the file should be skipped in creating the financial time series object:

```
spls =

desc:  Staples, Inc. (SPLS)
freq:  Unknown (0)

'dates:  (2)'  'OPEN:  (2)'    'HIGH:  (2)'    'LOW:  (2)'
'08-Apr-1996'  [   19.5000]    [  19.7500]     [19.2500]
'09-Apr-1996'  [   19.7500]    [  20.1250]     [19.3750]
```

Example 4. Create a financial time series object containing time-of-day information.

First create a data file with time information:

```
dates = ['01-Jan-2001';'01-Jan-2001'; '02-Jan-2001'; ...
'02-Jan-2001'; '03-Jan-2001';'03-Jan-2001'];
times = ['11:00';'12:00';'11:00';'12:00';'11:00';'12:00'];
serial_dates_times = [datenum(dates), datenum(times)];
data = round(10*rand(6,2));
stat = fts2ascii('myfts_file2.txt',serial_dates_times,data, ...
{'dates';'times';'Data1';'Data2'},'My FTS with Time');
```

Now read the data file back and create a financial time series object:

```
MyFts = ascii2fts('myfts_file2.txt','t',1,2,1)

MyFts =

    desc:  My FTS with Time
    freq:  Unknown (0)

    'dates:  (6)'    'times:  (6)'   'Data1:  (6)'  'Data2:  (6)'
    '01-Jan-2001'    '11:00'         [         9]   [         4]
    '     "     '    '12:00'         [         7]   [         9]
    '02-Jan-2001'    '11:00'         [         2]   [         1]
    '     "     '    '12:00'         [         4]   [         4]
    '03-Jan-2001'    '11:00'         [         9]   [         8]
    '     "     '    '12:00'         [         9]   [         0]
```

See Also

fints, fts2ascii

Purpose

Bar chart

Syntax

```
bar(tsobj)
bar(tsobj, width)
bar(..., 'style')
hbar = bar(...)

barh(...)
hbarh = barh(...)
```

Arguments

`tsobj`	Financial time series object
`width`	Width of the bars and separation of bars within a group. (Default = 0.8.) If width is 1, the bars within a group touch one another. Values > 1 produce overlapping bars.
`style`	`'grouped'` (default) or `'stacked'`

Description

`bar` and `barh` draw vertical and horizontal bar charts.

`bar(tsobj)` draws the columns of data series of the object `tsobj`. The number of data series dictates the number of vertical bars per group. Each group is the data for one particular date.

`bar(tsobj, width)` specifies the width of the bars.

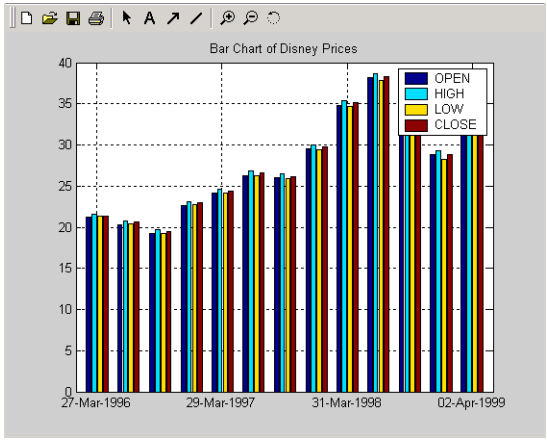
`bar(..., 'style')` changes the style of the bar chart.

`hbar = bar(...)` returns a vector of bar handles.

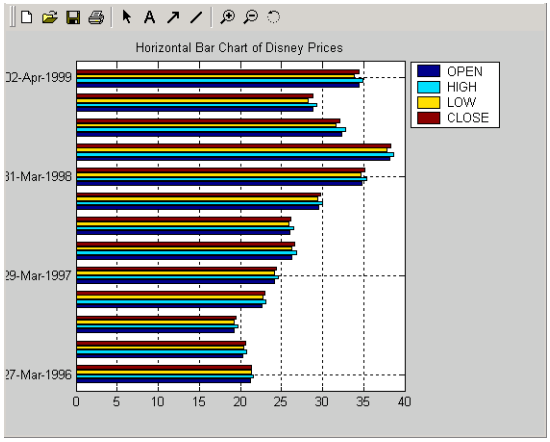
Use the MATLAB command `shading faceted` to put edges on the bars. Use `shading flat` to turn edges off.

Examples

Create bar charts for Disney stock showing high, low, opening, and closing prices.




```
load disney
bar(q_dis)
title('Bar Chart of Disney Prices')
```




```
load disney
bar(q_dis)
title('Horizontal Bar Chart of Disney Prices')
```

See Also bar3, bar3h, candle, highlow

bar3, bar3h

Purpose

Three-dimensional bar chart

Syntax

```
bar3(tsobj)
bar3(tsobj, width)
bar3(..., 'style')
hbar3 = bar3(...)

bar3h(...)
hbar3h = bar3h(...)
```

Arguments

`tsobj`	Financial time series object
`width`	Width of the bars and separation of bars within a group. (Default = 0.8.) If width is 1, the bars within a group touch one another. Values > 1 produce overlapping bars.
`style`	`'detached'` (default), `'grouped'`, or `'stacked'`

Description

`bar3` and `bar3h` draw three-dimensional vertical and horizontal bar charts.

`bar3(tsobj)` draws the columns of data series of the object `tsobj`. The number of data series dictates the number of vertical bars per group. Each group is the data for one particular date.

`bar3(tsobj, width)` specifies the width of the bars.

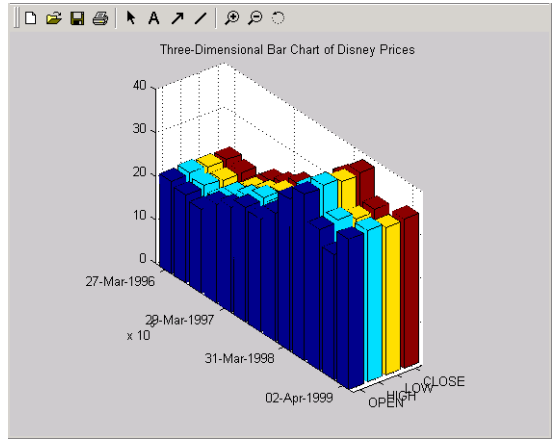
`bar3(..., 'style')` changes the style of the bar chart.

`hbar3 = bar3(...)` returns a vector of bar handles.

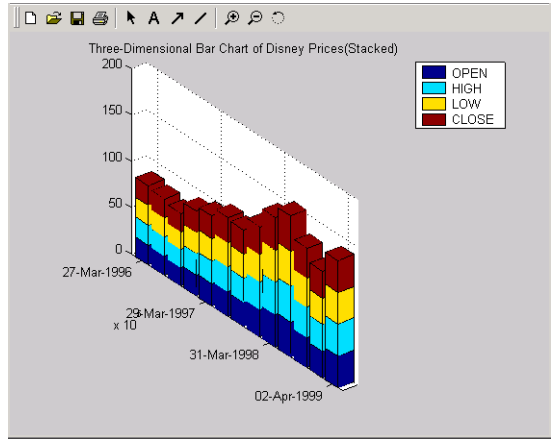
Use the MATLAB command `shading faceted` to put edges on the bars. Use `shading flat` to turn edges off.

Examples

Create three-dimensional bar charts for Disney stock showing high, low, opening, and closing prices.




```
load disney
bar3(q_dis, 'stacked')
title('Three-Dimensional Bar Chart of Disney Prices')
```




```
load disney
bar3(q_dis, 'stacked')
title('Three-Dimensional Bar Chart of Disney Prices (Stacked)')
```

See Also bar, barh, candle, highlow

Purpose

Bollinger band

Syntax

```
[mid, uppr, lowr] = bollinger(data, wsize, wts, nstd)
[midfts, upprfts, lowrfts] = bollinger(tsobj, wsize, wts, nstd)
```

Arguments

`data`	Data vector
`wsize`	(Optional) Window size. Default = 20.
`wts`	(Optional) Weight factor. Determines the type of moving average used. Default = 0 (box). 1 = linear.
`nstd`	(Optional) Number of standard deviations for upper and lower bands. Default = 2.
`tsobj`	Financial time series object

Description

`[mid, uppr, lowr] = bollinger(data, wsize, wts, nstd)` calculates the middle, upper, and lower bands that make up the Bollinger bands from the vector `data`.

`mid` is the vector that represents the middle band, a simple moving average with default window size of 20. `uppr` and `lowr` are vectors that represent the upper and lower bands. These bands are +2 times and -2 times moving standard deviations away from the middle band.

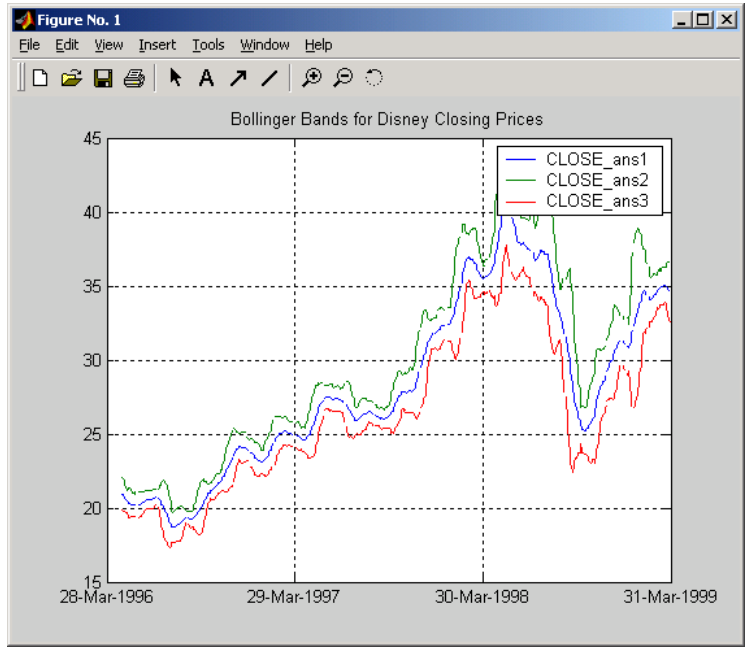
`[midfts, upprfts, lowrfts] = bollinger(tsobj, wsize, wts, nstd)` calculates the middle, upper, and lower bands that make up the Bollinger bands from a financial time series object `tsobj`.

`midfts` is a financial time series object that represents the middle band for all series in `tsobj`. Both `upprfts` and `lowrfts` are financial time series objects that represent the upper and lower bands of all series, which are +2 times and -2 times moving standard deviations away from the middle band.

Examples

Compute the Bollinger bands for Disney stock closing prices and plot the results:

```
load disney.mat
[dis_Mid,dis_Uppr,dis_Lowr]= bollinger(dis);
dis_CloseBolling = [dis_Mid.CLOSE, dis_Uppr.CLOSE,...
dis_Lowr.CLOSE];
plot(dis_CloseBolling)
title('Bollinger Bands for Disney Closing Prices')
```



See Also

`tsmovavg`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 72 - 74.

Purpose

Box-Cox transformation

Syntax

```
[transdat, lambda] = boxcox(data)
[transfts, lambdas] = boxcox(tsobj)
transdat = boxcox(lambda, data)
transfts = boxcox(lambda, tsobj)
```

Arguments

`data`	Data vector. Must be positive.
`tsobj`	Financial time series object

Description

`boxcox` transforms nonnormally distributed data to a set of data that has approximately normal distribution. The Box-Cox transformation is a family of power transformations defined by

$$data(\lambda) = \begin{cases} \dfrac{data^{\lambda} - 1}{\lambda} & \text{if } \lambda \neq 0 \\ \log(data) & \text{if } \lambda = 1 \end{cases}$$

The logarithm is the natural logarithm (log base e). The algorithm calls for finding the λ value that maximizes the Log-Likelihood Function (LLF). The search is conducted using `fminsearch`.

`[transdat, lambda] = boxcox(data)` transforms the data vector `data` using the Box-Cox transformation method into `transdat`. It also calculates the transformation parameter λ.

`[transfts, lambda] = boxcox(tsojb)` transforms the financial time series object `tsobj` using the Box-Cox transformation method into `transfts`. It also calculates the transformation parameter λ.

If the input data is a vector, `lambda` is a scalar. If the input is a financial time series object, `lambda` is a structure with fields similar to the components of the object, e.g., if the object contains series names `Open` and `Close`, `lambda` has fields `lambda.Open` and `lambda.Close`.

`transdat = boxcox(lambda, data)` and `transfts = boxcox(lambda, tsobj)` transform the data using a certain specified λ for the Box-Cox transformation. This syntax does not find the optimum λ that maximizes the LLF.

See Also

`fminsearch`

Purpose

Business days in serial date format

Syntax

```
bdates = busdays(sdate, edate, bdmode)
bdates = busdays(sdate, edate, bdmode, holvec)
```

Arguments

`sdate`	Start date in string or serial date format
`edate`	End date in string or serial date format

`bdmode`	(Optional) Frequency of business days:
	`DAILY`, `Daily`, `daily`, `D`, `d`, `1` (default)
	`WEEKLY`, `Weekly`, `weekly`, `W`, `w`, `2`
	`MONTHLY`, `Monthly`, `monthly`, `M`, `m`, `3`
	`QUARTERLY`, `Quarterly`, `quarterly`, `Q`, `q`, `4`
	`SEMIANNUAL`, `Semiannual`, `semiannual`, `S`, `s`, `5`
	`ANNUAL`, `Annual`, `annual`, `A`, `a`, `6`
	Strings must be enclosed in single quotation marks.

`holvec`	(Optional) Holiday dates vector in string or serial date format

Description

`bdates = busdays(sdate, edate, bdmode)` generates a vector of business days, `bdates`, in serial date format between the start date, `sdate`, and end date, `edate`, with frequency, `bdmode`. The dates are generated based on United States holidays. If you do not supply `bdmode`, `busdays` generates a daily vector.

`bdates = busdays(sdate, edate, bdmode, holvec)` lets you supply a vector of holidays, `holvec`, used to generate business days. `holvec` can either be in serial date format or date string format. If you use this syntax, you need to supply the frequency `bdmode`.

The output, `bdates`, is a column vector of business dates in serial date format.

If you want a weekday vector without the holidays, set `holvec` to `''` (empty string) or `[]` (empty vector).

candle

Purpose

Candle plot

Syntax

```
candle(tsobj)
candle(tsobj, color)
candle(tsobj, color, dateform)
candle(tsobj, color, dateform, ParameterName, ParameterValue, ...)
hcdl = candle(tsobj, color, dateform, ParameterName, ParameterValue,
   ...)
```

Arguments

`tsobj`	Financial time series object
`color`	(Optional) A three-element row vector representing RGB or a color identifier. (See `plot` in the MATLAB documentation.)
`dateform`	(Optional) Date string format used as the *x*-axis tick labels. (See `datetick` in the MATLAB documentation.) You can specify a `dateform` only when `tsobj` does not contain time-of-day data. If `tsobj` contains time-of-day data, `dateform` is restricted to `'dd-mmm-yyyy HH:MM'`.

Description

`candle(tsobj)` generates a candle plot of the data in the financial time series object `tsobj`. `tsobj` must contain at least four data series representing the high, low, open, and closing prices. These series must have the names `High`, `Low`, `Open`, and `Close` (case-insensitive).

`candle(tsobj, color)` additionally specifies the color of the candle box.

`candle(tsobj, color, dateform)` additionally specifies the date string format used as the *x*-axis tick labels. See `datestr` in the Financial Toolbox documentation for a list of date string formats.

`candle(tsobj, color, dateform, ParameterName, ParameterValue, ...)` indicates the actual name(s) of the required data series if the data series do not have the default names. `ParameterName` can be

- `HighName`: high prices series name
- `LowName`: low prices series name
- `OpenName`: open prices series name

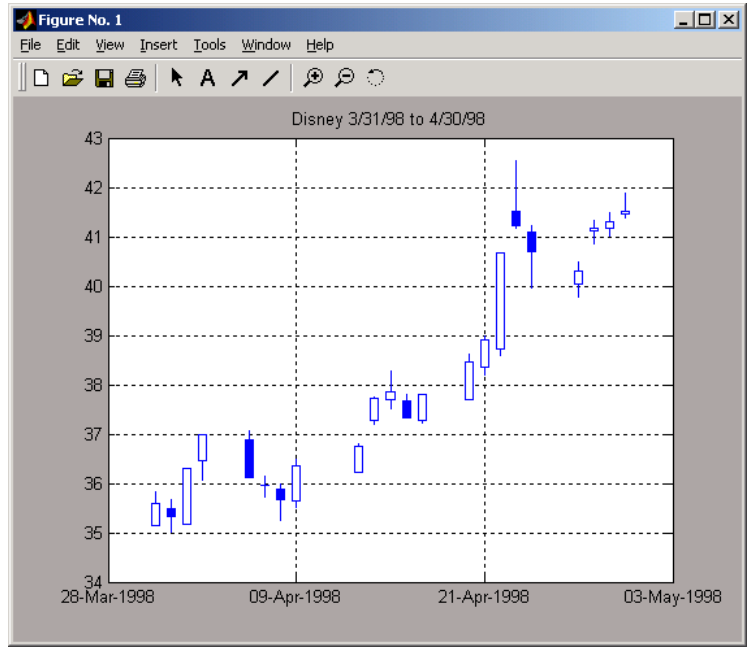
- `CloseName`: closing prices series name

`hcdl = candle(tsobj, color, dateform, ParameterName, ParameterValue, ...)` returns the handle to the patch objects and the line object that make up the candle plot. `hdcl` is a three-element column vector representing the handles to the two patches and one line that forms the candle plot.

Examples

Create a candle plot for Disney stock for the dates March 31, 1998 through April 30, 1998:

```
load disney.mat
candle(dis('3/31/98::4/30/98'))
title('Disney 3/31/98 to 4/30/98')
```



See Also

`chartfts`, `highlow`, `plot`

`candle` in the Financial Toolbox documentation

`datetick` and `plot` in the MATLAB documentation

chaikosc

Purpose

Chaikin oscillator

Syntax

```
chosc = chaikosc(highp, lowp, closep, tvolume)
chosc = chaikosc([highp lowp closep tvolume])
choscts = chaikosc(tsobj)
choscts = chaikosc(tsobj, ParameterName, ParameterValue, ... )
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`closep`	Closing price (vector)
`tvolume`	Volume traded (vector)
`tsobj`	Financial time series object

Description

The Chaikin oscillator is calculated by subtracting the 10-period exponential moving average of the Accumulation/Distribution (A/D) line from the three-period exponential moving average of the A/D line.

`chosc = chaikosc(highp, lowp, closep, tvolume)` calculates the Chaikin oscillator (vector), `chosc`, for the set of stock price and volume traded data (`tvolume`). The prices that must be included are the high (`highp`), low (`lowp`), and closing (`closep`) prices.

`chosc = chaikosc([highp lowp closep tvolume])` accepts a four-column matrix as input.

`choscts = chaikosc(tsobj)` calculates the Chaikin Oscillator, `choscts`, from the data contained in the financial time series object `tsobj`. `tsobj` must at least contain data series with names `High`, `Low`, `Close`, and `Volume`. These series must represent the high, low, and closing prices, plus the volume traded. `choscts` is a financial time series object with the same dates as `tsobj` but only one series named `ChaikOsc`.

`choscts = chaikosc(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

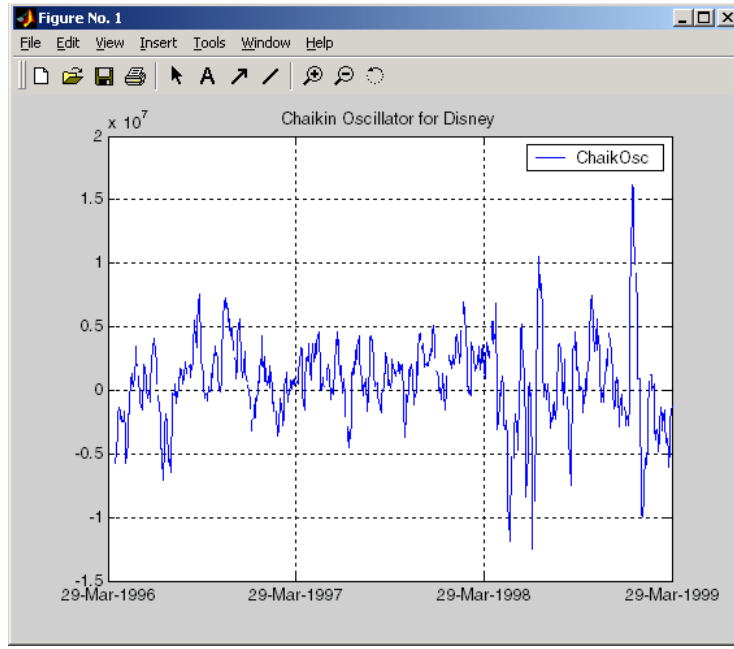
- `HighName`: high prices series name
- `LowName`: low prices series name
- `CloseName`: closing prices series name
- `VolumeName`: volume traded series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the Chaikin oscillator for Disney stock and plot the results.

```
load disney.mat
dis_CHAIKosc = chaikosc(dis)
plot(dis_CHAIKosc)
title('Chaikin Oscillator for Disney')
```



See Also

`adline`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 91 - 94.

chaikvolat

Purpose

Chaikin volatility

Syntax

```
chvol = chaikvolat(highp, lowp)
chvol = chaikvolat([highp lowp])
chvol = chaikvolat(high, lowp, nperdiff, manper)
chvol = chaikvolat([high lowp], nperdiff, manper)
chvts = chaikvolat(tsobj)
chvts = chaikvolat(tsobj, nperdiff, manper, ParameterName,
  ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`nperdiff`	Period difference (vector). Default = 10.
`manper`	Length of exponential moving average in periods (vector). Default = 10.
`tsobj`	Financial time series object

Description

`chvol = chaikvolat(highp, lowp)` calculates the Chaikin volatility from the series of stock prices, `highp` and `lowp`. The vector `chvol` contains the Chaikin volatility values, calculated on a 10-period exponential moving average and 10-period difference.

`chvol = chaikvolat([highp lowp])` accepts a two-column matrix as the input.

`chvol = chaikvolat(high, lowp, nperdiff, manper)` manually sets the period difference `nperdiff` and the length of the exponential moving average `manper` in periods.

`chvol = chaikvolat([high lowp], nperdiff, manper)` accepts a two-column matrix as the first input.

`chvts = chaikvolat(tsobj)` calculates the Chaikin volatility from the financial time series object `tsobj`. The object must contain at least two series named `High` and `Low`, representing the high and low prices per period. `chvts` is a financial time series object containing the Chaikin volatility values, based on

a 10-period exponential moving average and 10-period difference. `chvts` has the same dates as `tsobj` and a series called `ChaikVol`.

`chvts = chaikvolat(tsobj, nperdiff, manper, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

- `HighName`: high prices series name
- `LowName`: low prices series name

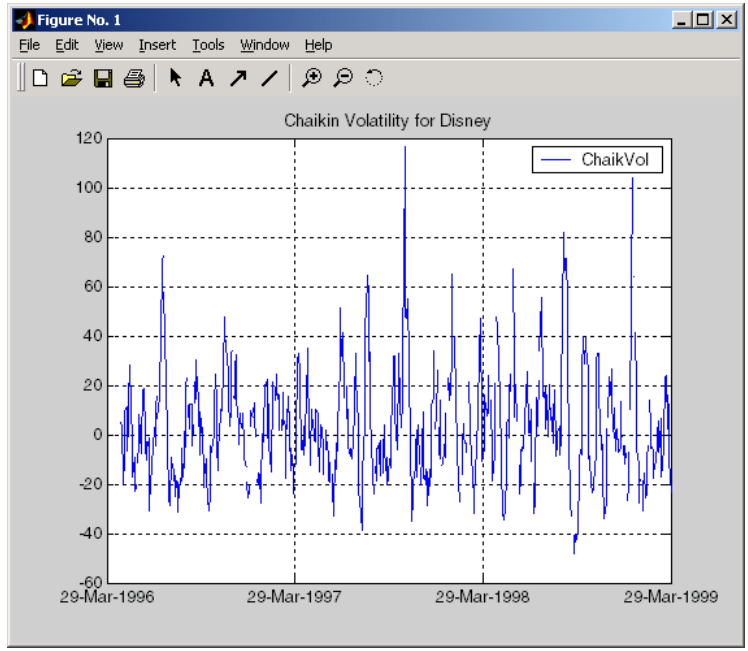
Parameter values are the strings that represent the valid parameter names.

`nperdiff`, the period difference, and `manper`, the length of the exponential moving average in periods, can also be set with this form of `chaikvolat`.

Examples

Compute the Chaikin volatility for Disney stock and plot the results:

```
load disney.mat
dis_CHAIKvol = chaikvolat(dis)
plot(dis_CHAIKvol)
title('Chaikin Volatility for Disney')
```



See Also

`chaikosc`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 304 - 305.

Purpose

Interactive display

Syntax

```
chartfts(tsobj)
```

Description

`chartfts(tsobj)` produces a figure window that contains one or more plots. You can use the mouse to observe the data at a particular time point of the plot.

Examples

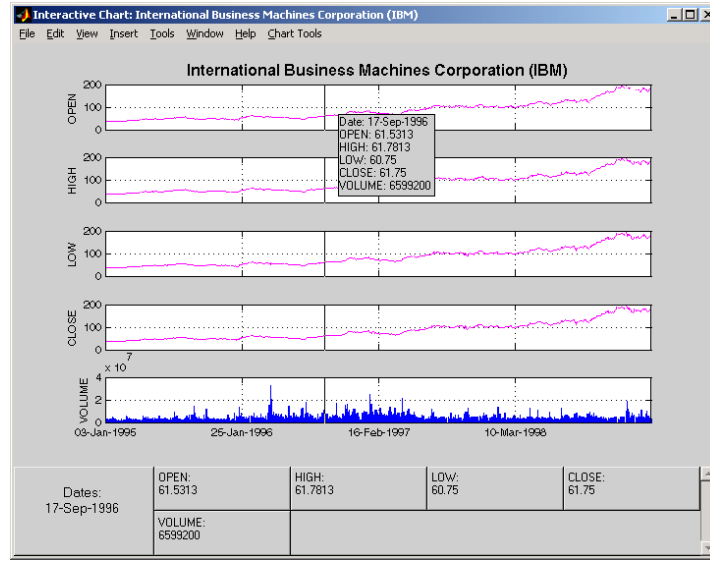
Create a financial time series object from the supplied data file `ibm9599.dat`:

```
ibmfts = ascii2fts('ibm9599.dat', 1, 3, 2);
```

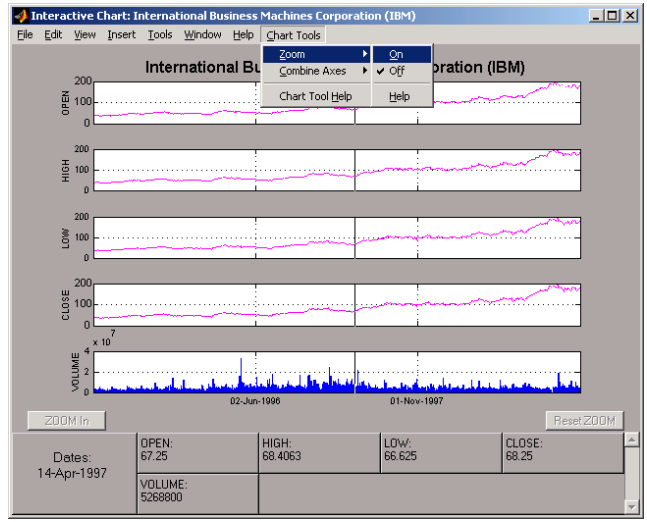
Chart the financial time series object `ibmfts`:

```
chartfts(ibmfts)
```

With the **Zoom** feature set `off`, a mouse click on the indicator line displays object data in a pop-up box.



With the **Zoom** feature set `on`, mouse clicks indicate the area of the chart to zoom.



You can find a tutorial on using `chartfts` in the section "Visualizing Financial Time Series Objects" on page 1-17. See "Zoom Tool" on page 1-20 for details on performing the zoom. Also see "Combine Axes Tool" on page 1-23 for information about combining axes for specified plots.

See Also

`candle`, `highlow`, `plot`

Purpose

Change data series name

Syntax

```
newfts = chfield(oldfts, oldname, newname)
```

Arguments

`oldfts`	Name of an existing financial time series object
`oldname`	Name of the existing component in `oldfts`. A MATLAB string or column cell array.
`newname`	New name for the component in `oldfts`. A MATLAB string or column cell array.

Description

`newfts = chfield(oldfts, oldname, newname)` changes the name of the financial time series object component from `oldname` to `newname`.

Set `newfts = oldfts` to change the name of an existing component without changing the name of the financial time series object.

To change the names of several components at once, specify the series of old and new component names in corresponding column cell arrays.

You cannot change the names of the object components `desc`, `freq`, and `dates`.

See Also

`fieldnames`, `isfield`, `rmfield`

convertto

Purpose

Convert to specified frequency

Syntax

```
newfts = convertto(oldfts, newfreq)
```

Arguments

`newfreq`	1, `DAILY`, `Daily`, `daily`, `D`, `d` 2, `WEEKLY`, `Weekly`, `weekly`, `W`, `w` 3, `MONTHLY`, `Monthly`, `monthly`, `M`, `m` 4, `QUARTERLY`, `Quarterly`, `quarterly`, `Q`, `q` 5, `SEMIANNUAL`, `Semiannual`, `semiannual`, `S`, `s` 6, `ANNUAL`, `Annual`, `annual`, `A`, `a`

Description

`convertto` converts a financial time series of any frequency to one of a specified frequency. It makes some assumptions regarding the dates in the resulting time series.

`newfts = convertto(oldfts, newfreq)` converts the object `oldfts` to the new time series object `newfts` with the frequency `newfreq`.

See Also

`toannual`, `todaily`, `tomonthly`, `toquarterly`, `tosemi`, `toweekly`

Purpose

Cumulative sum

Syntax

```
newfts = cumsum(oldfts)
```

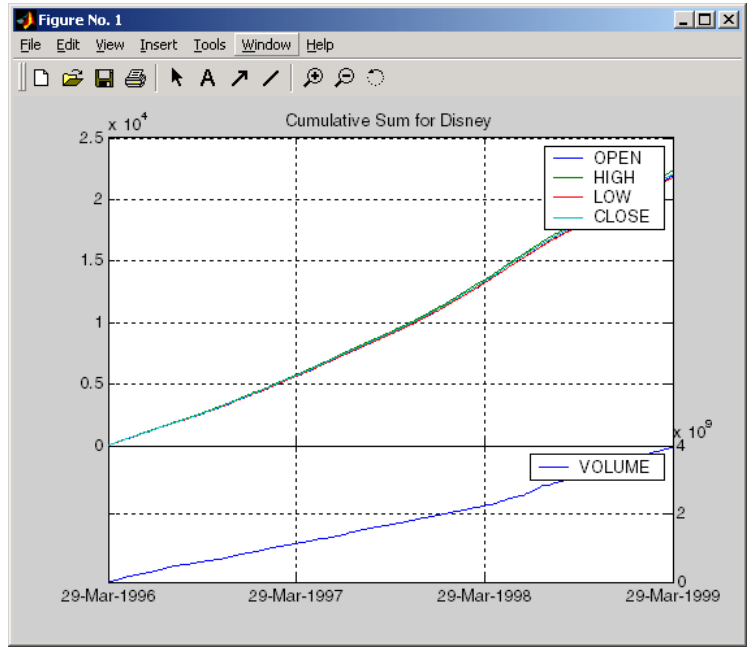
Description

`newfts = cumsum(oldfts)` calculates the cumulative sum of each individual time series data series in the financial time series object `oldfts` and returns the result in another financial time series object `newfts`. `newfts` contains the same data series names as `oldfts`.

Examples

Compute the cumulative sum for Disney stock and plot the results:

```
load disney.mat
cs_dis = cumsum(fillts(dis));
plot(cs_dis)
title('Cumulative Sum for Disney')
```



See Also

`cumsum` in the MATLAB documentation

diff

Purpose

Differencing

Syntax

```
newfts = diff(oldfts)
```

Description

`diff` computes the differences of the data series in a financial time series object. It returns another time series object containing the difference.

`newfts = diff(oldfts)` computes the difference of all the data in the data series of the object `oldfts` and returns the result in the object `newfts`. `newfts` is a financial time series object containing the same data series (names) as the input `oldfts`.

See Also

`diff` in the MATLAB documentation

Purpose

Display financial time series object

Syntax

```
display(tsobj)
```

Description

`display(tsobj)` displays a financial time series object in the command window. Numeric values inherit the format specified in MATLAB.

Note Although the contents of the object are displayed as cells of a cell array, the object itself is not a cell array.

See Also

`format` in the MATLAB documentation

end

Purpose

Last date entry

Syntax

```
end
```

Description

`end` returns the index to the last date entry in a financial time series object.

Examples

Consider a financial time series object called `fts`:

```
fts =

    desc:  DJI30MAR94.dat
    freq:  Daily (1)

    'dates:  (20)'    'Open:  (20)'
    '04-Mar-1994'     [        3830.9]
    '07-Mar-1994'     [        3851.7]
    '08-Mar-1994'     [        3858.5]
    '09-Mar-1994'     [          3854]
    '10-Mar-1994'     [        3852.6]
    '11-Mar-1994'     [        3832.6]
    '14-Mar-1994'     [        3870.3]
    '16-Mar-1994'     [          3851]
    '17-Mar-1994'     [        3853.6]
    '18-Mar-1994'     [        3865.4]
    '21-Mar-1994'     [        3878.4]
    '22-Mar-1994'     [        3865.7]
    '23-Mar-1994'     [        3868.9]
    '24-Mar-1994'     [        3849.9]
    '25-Mar-1994'     [        3827.1]
    '28-Mar-1994'     [        3776.5]
    '29-Mar-1994'     [        3757.2]
    '30-Mar-1994'     [        3688.4]
    '31-Mar-1994'     [        3639.7]
```

The command `fts(15:end)` returns

```
ans =

    desc:  DJI30MAR94.dat
    freq:  Daily (1)

    'dates:  (6)'     'Open:  (6)'
    '24-Mar-1994'     [        3849.9]
    '25-Mar-1994'     [        3827.1]
    '28-Mar-1994'     [        3776.5]
    '29-Mar-1994'     [        3757.2]
    '30-Mar-1994'     [        3688.4]
    '31-Mar-1994'     [        3639.7]
```

See Also

`subsasgn`, `subsref`

`end` in the MATLAB documentation

exp

Purpose

Exponential values

Syntax

```
newfts = exp(tsobj)
```

Description

`newfts = exp(tsobj)` calculates the natural exponential (base e) of all the data in the data series of the financial time series object `tsobj` and returns the result in the object `newfts`.

See Also

`log`, `log2`, `log10`

Purpose

Extract data series

Syntax

```
ftse = extfield(tsobj, fieldnames)
```

Arguments

`tsobj`	Financial time series object
`fieldnames`	Data series to be extracted. A cell array if a list of data series names (`fieldnames`) is supplied. A string if only one is wanted.

Description

`ftse = extfield(tsobj, fieldnames)` extracts from `tsobj` the dates and data series specified by `fieldnames` into a new financial time series object `ftse`. `ftse` has all the dates in `tsobj` but contains a smaller number of data series.

Examples

`extfield` is identical to referencing a field in the object. For example,

```
ftse = extfield(fts, 'Close')
```

is the same as

```
ftse = fts.Close
```

This function is the complement of the function `rmfield`.

See Also

`rmfield`

fetch

Purpose

Extract data from financial time series object

Syntax

```
newfts = fetch(oldfts, StartDate, StartTime, EndDate, EndTime,
   delta, dmy_specifier, time_ref)
```

Arguments

`oldfts`	Existing financial time series object
`StartDate`	First date in the range from which data is to be extracted.
`StartTime`	Beginning time on each day. If you do not require specific times or `oldfts` does not contain time information, use `[]`. If you specify `StartTime`, you must also specify `EndTime`.
`EndDate`	Last date in the range from which data is to be extracted.
`EndTime`	Ending time on each day. If you do not require specific times or `oldfts` does not contain time information, use `[]`. If you specify `EndTime`, you must also specify `StartTime`.
`delta`	Skip interval. Can be any positive integer. Units for the skip interval specified by `dmy_specifier`.
`dmy_specifier`	Specifies the units for `delta`. Can be `D`, `d` (Days) `M`, `m` (Months) `Y`, `y` (Years)
`time_ref`	Time reference intervals or specific times. Valid time reference intervals are 1, 5, 15, or 60 minutes. Enter specific times as `'hh:mm'`.

Description

`newfts = fetch(oldfts, StartDate, StartTime, EndDate, EndTime, delta, dmy_specifier, time_ref)` requests data from a financial time series object beginning from the start date and/or start time to the end date and/or end time, skipping a specified number of days, months, or years.

Note If time information is present in `oldfts`, using `[]` for start or end times results in `fetch` returning all instances of a specific date.

Examples

Example 1. Create a financial time series object containing both dates and times:

```
dates = ['01-Jan-2001';'01-Jan-2001'; '02-Jan-2001'; ...
 '02-Jan-2001'; '03-Jan-2001';'03-Jan-2001'];
times = ['11:00';'12:00';'11:00';'12:00';'11:00';'12:00'];
dates_times = cellstr([dates, repmat(' ',size(dates,1),1),...
times]);
myFts = fints(dates_times,(1:6)',{'Data1'},1,'My first FINTS')

myFts =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [          5]
    '     "     '    '12:00'          [          6]
```

To fetch all dates and times from this financial time series, enter

```
fetch(myFts,'01-Jan-2001',[],'03-Jan-2001',[],1,'d')
```

or

```
fetch(myFts,'01-Jan-2001','11:00','03-Jan-2001','12:00',1,'d')
```

These commands reproduce the entire time series shown above.

To fetch every other day's data, enter

```
fetch(myFts,'01-Jan-2001',[],'03-Jan-2001',[],2,'d')
```

This produces

```
ans =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (4)'    'times:  (4)'    'Data1:  (4)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
    '03-Jan-2001'    '11:00'          [          5]
    '     "     '    '12:00'          [          6]
```

Example 2. Create a financial time series object with time intervals of less than one hour:

```
dates2 = ['01-Jan-2001';'01-Jan-2001'; '01-Jan-2001';...
'02-Jan-2001'; '02-Jan-2001';'02-Jan-2001'];
times2 = ['11:00';'11:05';'11:06';'12:00';'12:05';'12:06'];
dates_times2 = cellstr([dates2, repmat(' ',size(dates2,1),1),...
times2]);
myFts2 = fints(dates_times2,(1:6)',{'Data1'},1,'My second FINTS')

myFts2 =

    desc:  My second FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '11:05'          [          2]
    '     "     '    '11:06'          [          3]
    '02-Jan-2001'    '12:00'          [          4]
    '     "     '    '12:05'          [          5]
    '     "     '    '12:06'          [          6]
```

Use `fetch` to extract data from this time series object at five-minute intervals for each day starting at 11:00 o'clock on January 1, 2001.

```
fetch(myFts2,'01-Jan-2001',[],'02-Jan-2001',[],1,'d',5)

    desc:  My second FINTS
    freq:  Daily (1)

    'dates:  (4)'    'times:  (4)'    'Data1:  (4)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '11:05'          [          2]
    '02-Jan-2001'    '12:00'          [          4]
    '     "     '    '12:05'          [          5]
```

You can use this version of `fetch` to extract data at specific times. For example, to fetch data only at 11:06 and 12:06 from `myFts2`, enter

```
fetch(myFts2,'01-Jan-2001',[],'02-Jan-2001',[],1,'d',...
{'11:06';'12:06'})

ans =

    desc:  My second FINTS
    freq:  Daily (1)

    'dates:  (2)'    'times:  (2)'    'Data1:  (2)'
    '01-Jan-2001'    '11:06'          [          3]
    '02-Jan-2001'    '12:06'          [          6]
```

See Also

`extfield`, `ftsbound`, `getfield`, `subsref`

fieldnames

Purpose

Get names of fields

Syntax

```
fnames = fieldnames(tsobj)
fnames = fieldnames(tsobj, srsnameonly)
```

Arguments

`tsobj`	Financial time series object
`srsnameonly`	Field names returned: 0 = all field names (default). 1 = data series names only.

Description

`fieldnames` gets field names in a financial time series object.

`fnames = fieldnames(tsobj)` returns the field names associated with the financial time series object `tsobj` as a cell array of strings, including the common fields: `desc`, `freq`, `dates` (and `times` if present).

`fnames = fieldnames(tsobj, srsnameonly)` returns field names depending upon the setting of `srsnameonly`. If `srsnameonly` is 0, the function returns all field names, including the common fields: `desc`, `freq`, `dates`, and `times`. If `srsnameonly` is set to 1, `fieldnames` returns only the data series in `fnames`.

See Also

`chfield`, `getfield`, `isfield`, `rmfield`, `setfield`

Purpose

Fill missing values in time series

Syntax

```
newfts = fillts(oldfts, fill_method)
newfts = fillts(oldfts, fill_method, newdates)
newfts = fillts(oldfts, fill_method, newdates, {'T1','T2',...})
newfts = fillts(oldfts, fill_method, newdates, 'SPAN', {'TS','TE'},
  delta)
newfts = fillts(... sortmode)
```

Arguments

`oldfts`	Financial time series object
`fill_method`	(Optional) Replaces missing values (`NaN`) in `oldfts` using an interpolation process, a constant, or a zero-order hold. Valid fill methods (interpolation methods) are: linear - `'linear '` - `'l'` (default) linear with extrapolation - `'linearExtrap'` - `'le'` cubic - `'cubic'` - `'c'` cubic with extrapolation - `'cubicExtrap'` - `'ce'` spline - `'spline'` - `'s'` spline with extrapolation - `'splineExtrap'` - `'se'` nearest - `'nearest'` - `'n'` nearest with extrapolation - `'nearestExtrap'` - `'ne'` pchip - `'pchip'` - `'p'` pchip with extrapolation - `'pchipExtrap'` - `'pe'` (See `interp1` for a discussion of extrapolation.) To fill with a constant, enter that constant. A zero-order hold (`'zero'`) fills a missing value with the value immediately preceding it. If the first value in the time series is missing, it remains a `NaN`.

`newdates`	(Optional) Column vector of serial dates, a date string, or a column cell array of date strings. If `oldfts` contains time of day information, `newdates` must be accompanied by a time vector (`newtimes`). Otherwise, `newdates` is assumed to have times of `'00:00'`.
`T1, T2, TS, TE`	First time, second time, start time, end time
`delta`	Time interval in minutes to span between the start time and end time
`sortmode`	(Optional) Default = 0 (unsorted). 1 = sorted.

Description

`newfts = fillts(oldfts, fill_method)` replaces missing values (represented by `NaN`) in the financial time series object `oldfts` with real values, using either a constant or the interpolation process indicated by *`fill_method`*.

`newfts = fillts(oldfts, fill_method, newdates)` replaces all the missing values on the specified dates `newdates` added to the financial time series `oldfts` with new values. The values can be a single constant or values obtained through the interpolation process designated by `fill_method`. If any of the dates in `newdates` exists in `oldfts`, the existing one has precedence.

`newfts = fillts(oldfts, fill_method, newdates, {'T1','T2',...})` additionally allows the designation of specific times of day for addition or replacement of data.

`newfts = fillts(oldfts, fill_method, newdates, 'SPAN', {'TS','TE'}, delta)` is similar to the previous format except that you designate only a start time and an end time. You follow these times with a spanning time interval, `delta`.

If you specify only one date for `newdates`, specifying a start and end time generates only times for that specific date.

`newfts = fillts(... sortmode)` additionally denotes whether you want the order of the dates in the output object to stay the same as in the input object or to be sorted chronologically.

`sortmode = 0` (unsorted) appends any new dates to the end. The interpolation and zero-order processes that calculate the values for the new dates work on a sorted object. Upon completion, the existing dates are reordered as they were originally, and the new dates are appended to the end.

`sortmode = 1` sorts the output. After interpolation, no reordering of the date sequence occurs.

Examples

Create a financial time series object with missing data in the fourth and fifth rows.

```
dates = ['01-Jan-2001';'01-Jan-2001'; '02-Jan-2001';...
         '02-Jan-2001'; '03-Jan-2001';'03-Jan-2001'];
times = ['11:00';'12:00';'11:00';'12:00';'11:00';'12:00'];
dates_times = cellstr([dates, repmat(' ',size(dates,1),1),...
                       times]);
OpenFts = fints(dates_times,[(1:3)'; nan; nan; 6],{'Data1'},1,...
'Open Financial Time Series');
```

`OpenFts` looks like

```
OpenFts =

    desc:  Open Financial Time Series
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [        NaN]
    '03-Jan-2001'    '11:00'          [        NaN]
    '     "     '    '12:00'          [          6]
```

Example 1. Fill the missing data in `OpenFts` using cubic interpolation.

```
FilledFts = fillts(OpenFts,'cubic')

FilledFts =

       desc:  Filled Open Financial Time Series
       freq:  Unknown (0)
```

```
    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [     3.0663]
    '03-Jan-2001'    '11:00'          [     5.8411]
    '     "     '    '12:00'          [     6.0000]
```

Example 2. Fill the missing data in `OpenFts` with a constant value.

```
FilledFts = fillts(OpenFts,0.3)

FilledFts =

    desc:  Filled Open Financial Time Series
    freq:  Unknown (0)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [     0.3000]
    '03-Jan-2001'    '11:00'          [     0.3000]

    '     "     '    '12:00'          [          6]
```

Example 3. You can use `fillts` to identify a specific time on a specific day for the replacement of missing data. This example shows how to replace missing data at 12:00 on January 2 and 11:00 on January 3.

```
FilltimeFts = fillts(OpenFts,'c',...
{'02-Jan-2001';'03-Jan-2001'}, {'12:00';'11:00'},0)

FilltimeFts =

       desc:  Filled Open Financial Time Series
       freq:  Unknown (0)

       'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
       '01-Jan-2001'    '11:00'          [          1]
       '     "     '    '12:00'          [          2]
```

```
'02-Jan-2001'    '11:00'          [          3]
'     "     '    '12:00'          [     3.0663]
'03-Jan-2001'    '11:00'          [     5.8411]
'     "     '    '12:00'          [     6.0000]
```

Example 4. Use a spanning time interval to add an additional day to `OpenFts`.

```
SpanFts = fillts(OpenFts,'c','04-Jan-2001','span',...
              {'11:00';'12:00'},60,0)

SpanFts =

       desc:  Filled Open Financial Time Series
       freq:  Unknown (0)

       'dates:  (8)'    'times:  (8)'    'Data1:  (8)'
       '01-Jan-2001'    '11:00'          [          1]
       '     "     '    '12:00'          [          2]
       '02-Jan-2001'    '11:00'          [          3]
       '     "     '    '12:00'          [     3.0663]
       '03-Jan-2001'    '11:00'          [     5.8411]
       '     "     '    '12:00'          [     6.0000]
       '04-Jan-2001'    '11:00'          [     9.8404]
       '     "     '    '12:00'          [     9.9994]
```

See Also

`interp1` in the MATLAB documentation

filter

Purpose

Linear filtering

Syntax

```
newfts = filter(B, A, oldfts)
```

Description

`filter` filters an entire financial time series object with certain filter specifications. The filter is specified in a transfer function expression.

`newfts = filter(B, A, oldfts)` filters the data in the financial time series object `oldfts` with the filter described by vectors `A` and `B` to create the new financial time series object `newfts`. The filter is a "Direct Form II Transposed" implementation of the standard difference equation. `newfts` is a financial time series object containing the same data series (names) as the input `oldfts`.

See Also

`filter`, `filter2` in the MATLAB documentation

Purpose

Construct financial time series object

Syntax

```
tsobj = fints(dates_and_data)
tsobj = fints(dates, data)
tsobj = fints(dates, data, datanames)
tsobj = fints(dates, data, datanames, freq)
tsobj = fints(dates, data, datanames, freq, desc)
```

Arguments

`dates_and_data`

Column-oriented matrix containing one column of dates and a single column for each series of data. In this format, dates must be entered in serial date number format. If the input serial date numbers encode time-of-day information, the output object contains a column labeled `'dates'` containing the date information and another labeled `'times'` containing the time information.

You can use the function `today` to enter date information or the function `now` to enter date with time information.

`dates`

Column vector of dates. Dates can be date strings or serial date numbers and can include time of day information. When entering time-of-day information as serial date numbers, the entry must be a column-oriented matrix when multiple entries are present. If the time-of-day information is in string format, the entry must be a column-oriented cell array of dates and times when multiple entries are present. Valid date and time string formats are

- `'ddmmmyy hh:mm'` or `'ddmmmyyyy hh:mm'`
- `'mm/dd/yy hh:mm'` or `'mm/dd/yyyy hh:mm'`
- `'dd-mmm-yy hh:mm'` or `'dd-mmm-yyyy hh:mm'`
- `'mmm.dd,yy hh:mm'` or `'mmm.dd,yyyy hh:mm'`

Dates and times can initially be separate column-oriented vectors, but they must be concatenated into a single column-oriented matrix before being passed to `fints`.

You can use the MATLAB functions `today` and `now` to assist in entering date and time information.

`data`

Column-oriented matrix containing a column for each series of data. The number of values in each data series must match the number of dates. If a mismatch occurs, MATLAB does not generate the financial time series object, and you receive an error message.

`datanames`

Cell array of data series names. Overrides the default data series names. Default data series names are `series1`, `series2`, ...

`freq`	Frequency indicator. Allowed values are
	`UNKNOWN`, `Unknown`, `unknown`, `U`, `u`, `0`
	`DAILY`, `Daily`, `daily`, `D`, `d`, `1`
	`WEEKLY`, `Weekly`, `weekly`, `W`, `w`, `2`
	`MONTHLY`, `Monthly`, `monthly`, `M`, `m`, `3`
	`QUARTERLY`, `Quarterly`, `quarterly`, `Q`, `q`, `4`
	`SEMIANNUAL`, `Semiannual`, `semiannual`, `S`, `s`, `5`
	`ANNUAL`, `Annual`, `annual`, `A`, `a`, `6`
	Default = `Unknown`.

`desc`	String providing descriptive name for financial time series object. Default = `''`.

Note The toolbox only supports hourly and minute time series. Seconds are disregarded when the object is created (e.g., 01-jan-2001 12:00:01 is considered to be 01-jan-2001 12:00). If there are duplicate dates and times, `fints` sorts the dates and times and chooses the first instance of the duplicate dates and times. The other duplicate dates and times are removed from the object along with their corresponding data.

Description

`fints` constructs a financial time series object. A financial time series object is a MATLAB object that contains a series of dates and one or more series of data. Before you perform an operation on the data, you must set the frequency indicator (`freq`). You can optionally provide a description (`desc`) for the time series.

`tsobj = fints(dates_and_data)` creates a financial time series object containing the dates and data from the matrix `dates_and_data`. If the dates contain time-of-day information, the object contains an additional series of times. The date series and each data series must each be a column in the input matrix. The names of the data series default to `series1, ..., seriesn`. The `desc` and `freq` fields are set to their defaults.

`tsobj = fints(dates, data)` generates a financial time series object containing dates from the `dates` column vector of dates and data from the matrix `data`. If the dates contain time-of-day information, the object contains

an additional series of times. The `data` matrix must be column-oriented, that is, each column in the matrix is a data series. The names of the series default to `series1, ..., seriesn`, where `n` is the total number of columns in `data`. The `desc` and `freq` fields are set to their defaults.

`tsobj = fints(dates, data, datanames)` additionally allows you to rename the data series. The names are specified in the `datanames` cell array. The number of strings in `datanames` must correspond to the number of columns in `data`. The `desc` and `freq` fields are set to their defaults.

`tsobj = fints(dates, data, datanames, freq)` additionally sets the frequency when you create the object. The `desc` field is set to its default `''`.

`tsobj = fints(dates, data, datanames, freq, desc)` provides a description string for the financial time series object.

Examples

Example 1: Create a financial time series containing days and data only:

```
data = [1:6]'

data =

     1
     2
     3
     4
     5
     6

dates = [today:today+5]'

dates =

      731132
      731133
      731134
      731135
      731136
      731137
```

```
tsobjkt = fints(dates, data)

tsobjkt =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (6)'    'series1:  (6)'
    '08-Oct-2001'    [            1]
    '09-Oct-2001'    [            2]
    '10-Oct-2001'    [            3]
    '11-Oct-2001'    [            4]
    '12-Oct-2001'    [            5]
    '13-Oct-2001'    [            6]
```

Example 2. Expand the above example to include time-of-day information:

```
dates = [now:now+5]';

tsobjkt = fints(dates, data)

tsobjkt =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (6)'    'times:  (6)'    'series1:  (6)'
    '08-Oct-2001'    '14:51'          [            1]
    '09-Oct-2001'    '14:51'          [            2]
    '10-Oct-2001'    '14:51'          [            3]
    '11-Oct-2001'    '14:51'          [            4]
    '12-Oct-2001'    '14:51'          [            5]
    '13-Oct-2001'    '14:51'          [            6]
```

Example 3. Create a financial time series object when dates and times are located in separate vectors.

Step 1. Create a column vector of times in date number format:

```
times = datenum(datestr(now:1/24+1/24/60:now+6/24+1/24/60,15))

times =
```

```
   0.43750000000000
   0.47986111111111
   0.52222222222222
   0.56458333333333
   0.60694444444444
   0.64930555555556
```

Step 2. Create a column vector of dates:

```
dates = [today:today+5]'

dates =

      731133
      731134
      731135
      731136
      731137
      731138
```

Step 3. Concatenate dates and times into a single matrix:

```
dates_times = [dates, times]

dates_times =

  1.0e+005 *

   7.31133000000000   0.00000437500000
   7.31134000000000   0.00000479861111
   7.31135000000000   0.00000522222222
   7.31136000000000   0.00000564583333
   7.31137000000000   0.00000606944444
   7.31138000000000   0.00000649305556
```

Step 4. Create column vector of data:

```
data = [1:6]'
```

Step 5. Create the financial time series object:

```
tsobj = fints(dates_times, data)

tsobj =

    desc:  (none)
    freq:  Unknown (0)

    'dates:  (6)'    'times:  (6)'    'series1:  (6)'
    '09-Oct-2001'    '10:30'          [            1]
    '10-Oct-2001'    '11:31'          [            2]
    '11-Oct-2001'    '12:32'          [            3]
    '12-Oct-2001'    '13:33'          [            4]
    '13-Oct-2001'    '14:34'          [            5]
    '14-Oct-2001'    '15:35'          [            6]
```

See Also

`datenum`, `datestr` in the Financial Toolbox documentation

fintsver

Purpose

Determine version

Syntax

```
ftsver = fintsver(tsobj)
[ftsver, timedata} = fintsver(tsobj)
```

Arguments

`tsobj`	Financial time series object

Description

`ftsver = fintsver(tsobj)` determines if `tsobj` is an object from the Financial Time Series Toolbox Version 2.0 or earlier. `ftsver = 1` indicates that `tsobj` is an object from Financial Time Series Toolbox Version 1.0 or 1.1. `ftsver = 2` indicates that `tsobj` is an object from Version 2 of the toolbox. Version 2 objects can contain time-of-day data.

`[ftsver, timedata] = fintsver(tsobj)` additionally indicates if `tsobj` contains time information. `timedata = 0` indicates no time information is present. `timedata = 1` indicates that time information is present.

Examples

Determine the version number and whether time information is present in the Disney stock price financial time series object:

```
load disney.mat
[ftsver, timedata] = fintsver(dis)

ftsver =

     1

timedata =

     0
```

Purpose

Fast stochastics

Syntax

```
[pctk, pctd] = fpctkd(highp, lowp, closep)
[pctk, pctd] = fpctkd([highp lowp closep])
[pctk, pctd] = fpctkd(highp, lowp, closep, kperiods, dperiods,
  dmamethod)
[pctk, pctd] = fpctkd([highp lowp closep], kperiods, dperiods,
  dmamethod)
pkdts = fpctkd(tsobj, kperiods, dperiods, dmamethod)
pkdts = fpctkd(tsobj, kperiods, dperiods, dmamethod, ParameterName,
  ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`closep`	Closing price (vector)
`kperiods`	(Optional) %K periods. Default = 10.
`dperiods`	(Optional) %D periods. Default = 3.
`damethod`	(Optional) %D moving average method. Default = `'e'` (exponential).
`tsobj`	Financial time series object

Description

`fpctkd` calculates the stochastic oscillator.

`[pctk, pctd] = fpctkd(highp, lowp, closep)` calculates the fast stochastics F%K and F%D from the stock price data `highp` (high prices), `lowp` (low prices), and `closep` (closing prices).

`[pctk, pctd] = fpctkd([highp lowp closep])` accepts a three-column matrix of high (`highp`), low (`lowp`), and closing prices (`closep`), in that order.

`[pctk, pctd] = fpctkd(highp, lowp, closep, kperiods, dperiods, dmamethod)` calculates the fast stochastics F%K and F%D from the stock price data `highp` (high prices), `lowp` (low prices), and `closep` (closing prices). `kperiods` sets the %K period. `dperiods` sets the %D period.

`damethod` specifies the %D moving average method. Valid moving average methods for %D are Exponential (`'e'`) and Triangular (`'t'`). See `tsmovavg` for explanations of these methods.

`[pctk, pctd]= fpctkd([highp lowp closep], kperiods, dperiods, dmamethod)` accepts a three-column matrix of high (`highp`), low (`lowp`), and closing prices (`closep`), in that order.

`pkdts = fpctkd(tsobj, kperiods, dperiods, dmamethod)` calculates the fast stochastics F%K and F%D from the stock price data in the financial time series object `tsobj`. `tsobj` must minimally contain the series `High` (high prices), `Low` (low prices), and `Close` (closing prices). `pkdts` is a financial time series object with similar dates to `tsobj` and two data series named `PercentK` and `PercentD`.

`pkdts = fpctkd(tsobj, kperiods, dperiods, dmamethod, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are:

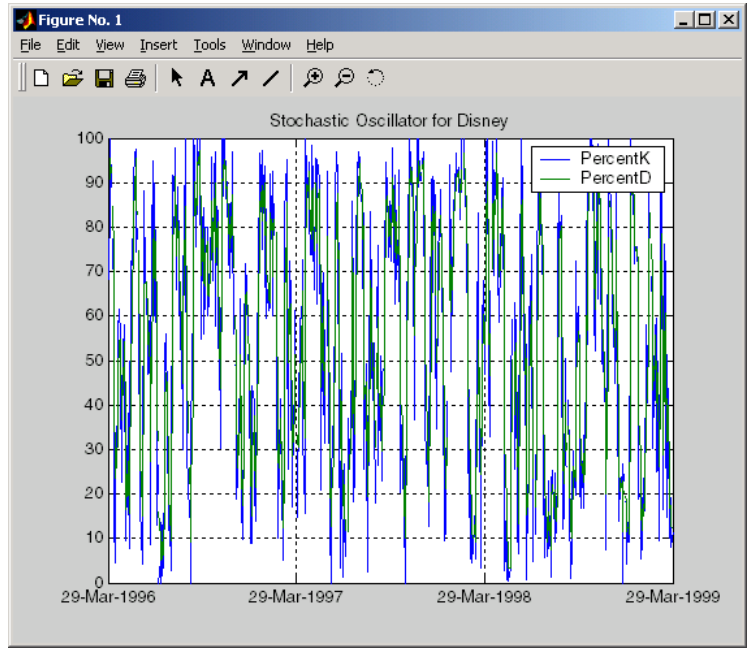
- `HighName`: high prices series name
- `LowName`: low prices series name
- `CloseName`: closing prices series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the stochastic oscillator for Disney stock and plot the results:

```
load disney.mat
dis_FastStoc = fpctkd(dis)
plot(dis_FastStoc)
title('Stochastic Oscillator for Disney')
```



See Also

`spctkd`, `stochosc`, `tsmovavg`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 268 - 271.

freqnum

Purpose

Convert string frequency indicator to numeric frequency indicator

Syntax

```
nfreq = freqnum(sfreq)
```

Arguments

`sfreq`	`UNKNOWN`, `Unknown`, `unknown`, `U`, `u`
	`DAILY`, `Daily`, `daily`, `D`, `d`
	`WEEKLY`, `Weekly`, `weekly`, `W`, `w`
	`MONTHLY`, `Monthly`, `monthly`, `M`, `m`
	`QUARTERLY`, `Quarterly`, `quarterly`, `Q`, `q`
	`SEMIANNUAL`, `Semiannual`, `semiannual`, `S`, `s`
	`ANNUAL`, `Annual`, `annual`, `A`, `a`

Description

`nfreq = freqnum(sfreq)` converts a string frequency indicator into a numeric value.

String Frequency Indicator	Numeric Representation
`UNKNOWN`, `Unknown`, `unknown`, `U`, `u`	0
`DAILY`, `Daily`, `daily`, `D`, `d`	1
`WEEKLY`, `Weekly`, `weekly`, `W`, `w`	2
`MONTHLY`, `Monthly`, `monthly`, `M`, `m`	3
`QUARTERLY`, `Quarterly`, `quarterly`, `Q`, `q`	4
`SEMIANNUAL`, `Semiannual`, `semiannual`, `S`, `s`	5
`ANNUAL`, `Annual`, `annual`, `A`, `a`	6

See Also

`freqstr`

Purpose

Convert numeric frequency indicator to string representation

Syntax

```
sfreq = freqstr(nfreq)
```

Arguments

`nfreq`	0 1 2 3 4 5 6

Description

`sfreq = freqstr(nfreq)` converts a numeric frequency indicator into a string representation.

Numeric Frequency Indicator	String Representation
0	`Unknown`
1	`Daily`
2	`Weekly`
3	`Monthly`
4	`Quarterly`
5	`Semiannual`
6	`Annual`

See Also

`freqnum`

fts2ascii

Purpose

Write elements of time series data into an ASCII file

Syntax

```
stat = fts2ascii(filename, tsobj, exttext)
stat = fts2ascii(filename, dates, data, colheads, desc, exttext)
```

Arguments

`filename`	Name of an ASCII file
`tsobj`	Financial time series object
`exttext`	(Optional) Extra text. A string written after the description line (line 2 in the file).
`dates`	Column vector containing dates. Dates must be in serial date number format and can specify time of day.
`data`	Column-oriented matrix. Each column is a series.
`colheads`	(Optional) Cell array of column headers (names); first cell must always be the one for the dates column. `colheads` will be written to the file just before the data.
`desc`	(Optional) Description string, which will be the first line in the file.

Description

`stat = fts2ascii(filename, tsobj, exttext)` writes the financial time series object `tsobj` into an ASCII file `filename`. The data in the file is tab delimited.

`stat = fts2ascii(filename, dates, data, colheads, desc, exttext)` writes into an ASCII file `filename` the dates, times, and data contained in the column vector `dates` and the column-oriented matrix `data`. The first column in `filename` contains the `dates`, followed by `times` (if specified). Subsequent columns contain the `data`. The data in the file is tab delimited.

`stat` indicates whether file creation is successful (`1`) or not (`0`).

See Also

`ascii2fts`

Purpose

Convert to matrix

Syntax

```
tsmat = fts2mat(tsobj)
tsmat = fts2mat(tsobj, datesflag)
tsmat = fts2mat(tsobj, seriesnames)
tsmat = fts2mat(tsobj, datesflag, seriesnames)
```

Arguments

`tsobj`	Financial time series object
`datesflag`	(Optional) Specifies inclusion of dates vector: `datesflag = 0` (default) excludes dates. `datesflag = 1` includes dates vector.
`seriesnames`	(Optional) Specifies the data series to be included in the matrix. Can be a cell array of strings.

Description

`tsmat = fts2mat(tsobj)` takes the data series in the financial time series object `tsobj` and puts them into the matrix `tsmat` as columns. The order of the columns is the same as the order of the data series in the object `tsobj`.

`tsmat = fts2mat(tsobj, datesflag)` specifies whether or not you want the dates vector included. The dates vector will be the first column. The dates are represented as serial date numbers. Dates can include time-of-day information.

`tsmat = fts2mat(tsobj, seriesnames)` extracts the data series named in `seriesnames` and puts its values into `tsmat`. The `seriesnames` argument can be a cell array of strings.

`tsmat = fts2mat(tsobj, datesflag, seriesnames)` puts into `tsmat` the specific data series named in `seriesnames`. The `datesflag` argument must be specified. If `datesflag` is set to `1`, the dates vector is included. If you specify an empty matrix (`[]`) for `datesflag`, the default behavior is adopted.

See Also

`subsref`

ftsbound

Purpose

Start and end dates

Syntax

```
datesbound = ftsbound(tsobj)
datesbound = ftsbound(tsobj, dateform)
```

Arguments

`tsobj`	Financial time series object
`dateform`	`dateform` is an integer representing the format of a date string. See `datestr` for a description of these formats.

Description

`ftsbound` returns the start and end dates of a financial time series object. If the object contains time-of-day data, `ftsbound` additionally returns the starting time on the first date and the ending time on the last date.

`datesbound = ftsbound(tsobj)` returns the start and end dates contained in `tsobj` as serial dates in the column matrix `datesbound`. The first row in `datesbound` corresponds to the start date, and the second corresponds to the end date.

`datesbound = ftsbound(tsobj, dateform)` returns the starting and ending dates contained in the object, `tsobj`, as date strings in the column matrix, `datesbound`. The first row in `datesbound` corresponds to the start date, and the second corresponds to the end date. The `dateform` argument controls the format of the output dates.

See Also

`datestr` in the Financial Toolbox documentation

Purpose

Financial time series graphical user interface

Syntax

```
ftsgui
```

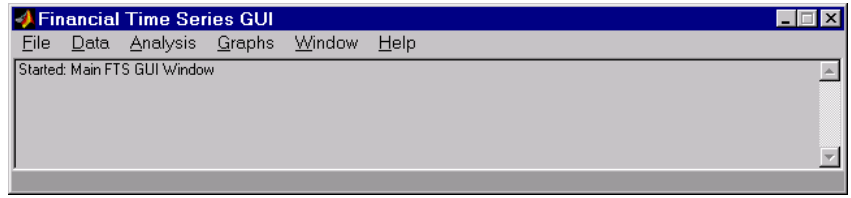
Description

`ftsgui` displays the financial time series graphical user interface (GUI) main window.

The use of the Financial Time Series GUI is described in Chapter 4, “Graphical User Interface.”

Example

```
ftsgui
```



ftsinfo

Purpose

Financial time series object information

Syntax

```
ftsinfo(tsobj)
infofts = ftsinfo(tsobj)
```

Arguments

`tsobj`	Financial time series object

Description

`ftsinfo(tsobj)` displays information about the financial time series object `tsobj`.

`infofts = ftsinfo(tsobj)` stores information about the financial time series object `tsobj` in the structure `infofts`.

`infofts` has these fields.

Field	Contents
`version`	Financial time series object version
`desc`	Description of the time series object (`tsobj.desc`)
`freq`	Numeric representation of the time series data frequency (`tsobj.freq`). See `freqstr` for list of numeric frequencies and what they represent.
`startdate`	Earliest date in the time series
`enddate`	Latest date in the time series
`seriesnames`	Cell array containing the time series data column names
`ndata`	Number of data points in the time series
`nseries`	Number of columns of time series data

Examples

Convert the supplied file `disney.dat` into a financial time series object named `dis`:

```
dis = ascii2fts('disney.dat', 1, 3);
```

Now use ftsinfo to obtain information about dis:

```
ftsinfo(dis)

 FINTS version:   2.0
   Description:  Walt Disney Company (DIS)
     Frequency:  Unknown
    Start date:  29-Mar-1996
      End date:  29-Mar-1999
  Series names:  OPEN
                 HIGH
                 LOW
                 CLOSE
                 VOLUME
     # of data:  782
   # of series:  5
```

Then, executing

```
infodis = ftsinfo(dis)
```

creates the structure infodis containing the values

```
infodis =

            ver: '2.0'
           desc: 'Walt Disney Company (DIS)'
           freq: 0
      startdate: '29-Mar-1996'
        enddate: '29-Mar-1999'
    seriesnames: {5x1 cell}
          ndata: 782
        nseries: 5
```

See Also

fints, freqnum, freqstr, ftsbound

ftsnew2old

Purpose

Convert Version 2 time series object to Version 1

Syntax

```
ftsno = ftsnew2old(tsobj2)
```

Arguments

`tsobj2`	Financial Time Series Toolbox (Version 2) object. (Version 2 objects can contain a time data field.)

Description

`ftsno = ftsnew2old(tsobj2)` converts a financial time series object from a Financial Time Series Toolbox Version 2 object to an object compatible with Version 1.

See Also

`ftsold2new`

Purpose

Convert Version 1 time series object to Version 2

Syntax

```
ftsno = ftsnew2old(tsobj1)
```

Arguments

`tsobj1`	Financial Time Series Toolbox (Version 1) object. (Version 1 objects cannot contain a time data field.)

Description

`ftsno = ftsnew2old(tsobj1)` converts a financial time series object from a Financial Time Series Toolbox Version 1 object to an object compatible with Version 2.

See Also

`ftsnew2old`

ftstool

Purpose

Create, display, and modify financial time series objects

Syntax

```
ftstool
```

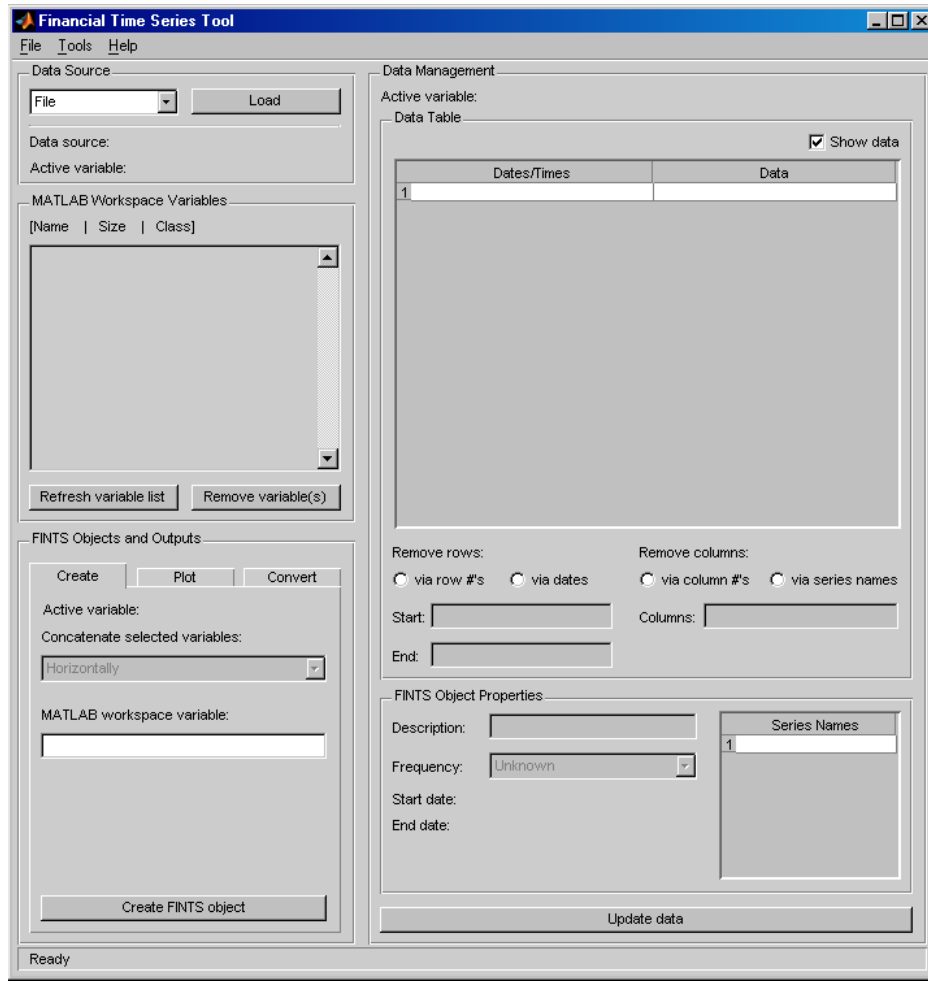
Description

`ftstool` displays the financial time series manager graphical user interface (GUI).

The Financial Time Series Tool is described in Chapter 5, “Financial Time Series Tool (FTSTool)”.

Example

```
ftstool
```



ftsuniq

Purpose

Determine uniqueness

Syntax

```
uniq = ftsuniq(dates_and_times)
[uniq, dup] = ftsuniq(dates_and_times)
```

Arguments

`dates_and_times`	A single column vector of serial date numbers. The serial date numbers can include time-of-day information.

Description

`uniq = ftsuniq(dates_and_times)` returns `1` if the dates and times within the financial time series object are unique and `0` if duplicates exist.

`[uniq, dup] = ftsuniq(dates_and_times)` additionally returns a structure `dup`. In the structure

- `dup.dt` contains the strings of the duplicate dates and times and their locations in the object.
- `dup.intidx` contains the integer indices of duplicate dates and times in the object.

See Also

`fints`

Purpose

Get content of a specific field

Syntax

```
fieldval = getfield(tsobj, field)
fieldval = getfield(tsobj, field, {dates})
```

Arguments

`tsobj`	Financial time series object
`field`	Field name within `tsobj`
`dates`	Date range. Dates can be expanded to include time-of-day information.

Description

`getfield` treats the contents of a financial times series object `tsobj` as fields in a structure.

`fieldval = getfield(tsobj, field)` returns the contents of the specified field. This is equivalent to the syntax `fieldval = tsobj.field`.

`fieldval = getfield(tsobj, field, {dates})` returns the contents of the specified field for the specified dates. `dates` can be individual cells of date strings or a cell of a date string range using the `::` operator, such as `'03/01/99::03/31/99'`.

Examples

Create a financial time series object containing both date and time-of-day information:

```
dates = ['01-Jan-2001';'01-Jan-2001'; '02-Jan-2001'; ...
         '02-Jan-2001'; '03-Jan-2001';'03-Jan-2001'];
times = ['11:00';'12:00';'11:00';'12:00';'11:00';'12:00'];
dates_times = cellstr([dates, repmat(' ',size(dates,1),1),...
times]);
AnFts = fints(dates_times,[(1:4)'; nan; 6],{'Data1'},1,...
              'Yet Another Financial Time Series')
```

```
AnFts =

    desc:  Yet Another Financial Time Series
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [        NaN]
    '     "     '    '12:00'          [          6]
```

Example 1. Get the contents of the `times` field in `AnFts`:

```
F = datestr(getfield(AnFts, 'times'))

F =

11:00 AM
12:00 PM
11:00 AM
12:00 PM
11:00 AM
12:00 PM
```

Example 2. Extract the contents of specific data fields within `AnFts`:

```
FF = getfield(AnFts,'Data1',...
              '01-Jan-2001 12:00::02-Jan-2001 12:00')

FF =

     2
     3
     4
```

See Also

`chfield`, `fieldnames`, `isfield`, `rmfield`, `setfield`

Purpose

Find name in list

Syntax

```
nameidx = getnameidx(list, name)
```

Arguments

`list`	A cell array of name strings
`name`	A string or cell array of name strings

Description

`nameidx = getnameidx(list, name)` finds the occurrence of a name or set of names in a list. It returns an index (order number) indicating where the specified names are located within the list. If `name` is not found, `nameidx` returns `0`.

If `name` is a cell array of names, `getnameidx` returns a vector containing the indices (order number) of the `name` strings within `list`. If none of the names in the `name` cell array is in `list`, it returns zero. If some of the names in `name` are not found, the indices for these names will be zeros.

`getnameidx` finds only the first occurrence of the name in the list of names. This function is meant to be used on a list of unique names (strings) only. It does not find multiple occurrences of a name or a list of names within `list`.

Examples

Given

```
poultry = {'duck', 'chicken'}
animals = {'duck', 'cow', 'sheep', 'horse', 'chicken'}
nameidx = getnameidx(animals, poultry)

ans =
    1   5
```

Given

```
poultry = {'duck', 'goose', 'chicken'}
animals = {'duck', 'cow', 'sheep', 'horse', 'chicken'}
nameidx = getnameidx(animals, poultry)

ans =
    1  0  5
```

See Also

`findstr`, `strcmp`, `strfind`

hhigh

Purpose

Highest high

Syntax

```
hhv = hhigh(data)
hhv = hhigh(data, nperiods, dim)
hhvts = hhigh(tsobj, nperiods)
hhvts = hhigh(tsobj, nperiods, ParameterName, ParameterValue)
```

Arguments

`data`	Data series matrix
`nperiods`	(Optional) Number of periods. Default = 14.
`dim`	(Optional) Dimension
`tsobj`	Financial time series object

Description

`hhv = hhigh(data)` generates a vector of highest high values the past 14 periods from the matrix `data`.

`hhv = hhigh(data, nperiods, dim)` generates a vector of highest high values the past `nperiods` periods. `dim` indicates the direction in which the highest high is to be searched. If you input `[]` for `nperiods`, the default is 14.

`hhvts = hhigh(tsobj, nperiods)` generates a vector of highest high values from `tsobj`, a financial time series object. `tsobj` must include at least the series `High`. The output `hhvts` is a financial time series object with the same dates as `tsobj` and data series named `HighestHigh`. If `nperiods` is specified, `hhigh` generates a financial time series object of highest high values for the past `nperiods` periods.

`hhvts = hhigh(tsobj, nperiods, ParameterName, ParameterValue)` specifies the name for the required data series when it is different from the default name. The valid parameter name is:

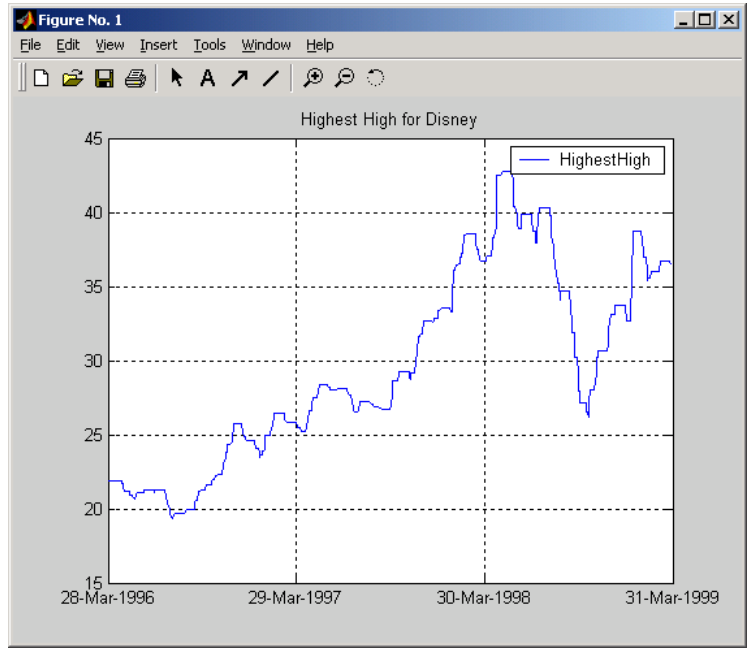
- `HighName`: high prices series name

The parameter value is a string that represents the valid parameter name .

Example

Compute the highest high prices for Disney stock and plot the results:

```
load disney.mat
dis_HHigh = hhigh(dis)
plot(dis_HHigh)
title('Highest High for Disney')
```



See Also

`llow`

highlow

Purpose

High-Low plot

Syntax

```
highlow(tsobj)
highlow(tsobj, color)
highlow(tsobj, color, dateform)
highlow(tsobj, color, dateform, ParameterName, ParameterValue, ...)
hhll = highlow(tsobj, color, dateform, ParameterName,
  ParameterValue, ...)
```

Arguments

`tsobj`	Financial time series object
`color`	(Optional) A three-element row vector representing RGB or a color identifier. (See `plot` in the MATLAB documentation.)
`dateform`	(Optional) Date string format used as the *x*-axis tick labels. (See `datetick` in the MATLAB documentation.) You can specify a `dateform` only when `tsobj` does not contain time-of-day data. If `tsobj` contains time-of-day data, `dateform` is restricted to `'dd-mmm-yyyy HH:MM'`.

Description

`highlow(tsobj)` generates a High-Low plot of the data in the financial time series object `tsobj`. `tsobj` must contain at least four data series representing the high, low, open, and closing prices. These series must have the names `High`, `Low`, `Open`, and `Close` (case-insensitive).

`highlow(tsobj, color)` additionally specifies the color of the plot.

`highlow(tsobj, color, dateform)` additionally specifies the date string format used as the *x*-axis tick labels. See `datestr` in the Financial Toolbox documentation for a list of date string formats.

`highlow(tsobj, color, dateform, ParameterName, ParameterValue,...)` indicates the actual name(s) of the required data series if the data series do not have the default names. `ParameterName` can be

- `HighName`: high prices series name
- `LowName`: low prices series name
- `OpenName`: open prices series name

- `CloseName`: closing prices series name

You can specifiy open prices as optional by providing the parameter name `'OpenName'` and the parameter value `''` (empty string).

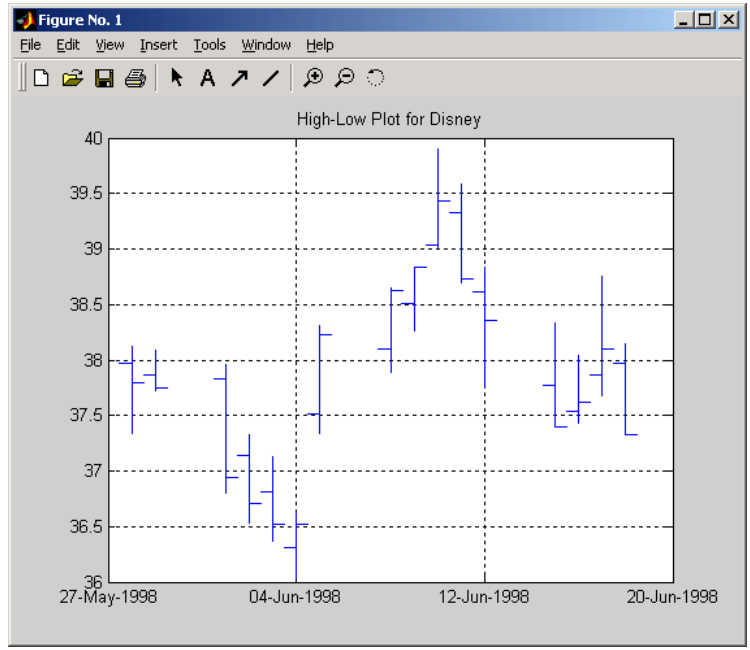
```
highlow(tsobj, color, dateform, 'OpenName', '')
```

`hhll = highlow(tsobj, color, dateform, ParameterName, ParameterValue, ...)` returns the handle to the line object that makes up the High-Low plot.

Examples

Generate a High-Low plot for Disney stock for the dates from May 28 to June 18, 1998:

```
load disney.mat
highlow(dis('28-May-1998::18-Jun-1998'))
title(`High-Low Plot for Disney')
```



See Also

`candle`

`highlow` in the Financial Toolbox documentation

`datetick` and `plot` in the MATLAB documentation

Purpose

Histogram

Syntax

```
hist(tsobj, numbins)
ftshist = hist(tsobj, numbins)
[ftshist, binpos] = hist(tsobj, numbins)
```

Arguments

`tsobj`	Financial time series object
`numbins`	(Optional) Number of histogram bins. Default = 10.

Description

`hist(tsobj, numbins)` calculates and displays the histogram of the data series contained in the financial time series object `tsobj`.

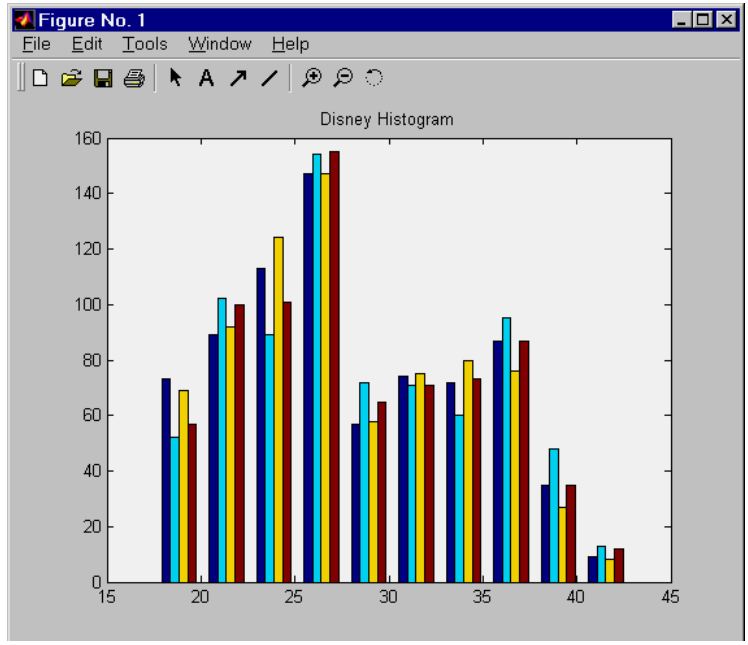
`ftshist = hist(tsobj, numbins)` calculates, but does not display, the histogram of the data series contained in the financial time series object `tsobj`. The output `ftshist` is a structure with field names similar to the data series names of `tsobj`.

`[ftshist, binpos] = hist(tsobj, numbins)` additionally returns the bin positions `binpos`. The positions are the centers of each bin. `binpos` is a column vector.

Example

Create a histogram of Disney open, high, low, and close prices:

```
load disney.mat
dis = rmfield(dis,'VOLUME') % Remove VOLUME field
hist(dis)
title('Disney Histogram')
```



See Also

`mean`, `std`

`hist` in the MATLAB documentation

Purpose

Concatenate financial time series objects horizontally

Description

`horzcat` implements horizontal concatenation of financial time series objects. `horzcat` essentially merges the data columns of the financial time series objects. The time series objects must contain the exact same dates and times.

When multiple instances of a data series name occur, concatenation adds a suffix to the current names of the data series. The suffix has the format `_objectname<n>`, where `n` is a number indicating the position of the time series, from left to right, in the concatenation command. The `n` part of the suffix appears only when there is more than one instance of a particular data series name.

The description fields are concatenated as well. They are separated by two forward slashes (`//`).

Examples

Construct three financial time series, each containing a data series named `DataSeries`:

```
firstfts  = fints((today:today+4)', (1:5)','DataSeries','d');
secondfts = fints((today:today+4)', (11:15)','DataSeries','d');
thirdfts  = fints((today:today+4)', (21:25)','DataSeries','d');
```

Concatenate the time series horizontally into a new financial time series `newfts`.

```
newfts  = [firstfts secondfts thirdfts secondfts];
```

The resulting object `newfts` has data series names `DataSeries_firstfts`, `DataSeries_secondfts2`, `DataSeries_thirdfts`, and `DataSeries_secondfts4`.

Verify this with the command

```
fieldnames(newfts)

ans =

     'desc'
     'freq'
     'dates'
     'DataSeries_firstfts'
```

```
'DataSeries_secondfts2'
'DataSeries_thirdfts'
'DataSeries_secondfts4'
'times'
```

Use `chfield` to change the data series names.

Note If all input objects have the same frequency, the new object has that frequency as well. However, if one of the objects concatenated has a different frequency from the others, the frequency indicator of the resulting object is set to `Unknown (0)`.

See Also

`vertcat`

Purpose

Structural equality

Syntax

```
iscomp = iscompatible(tsobj_1, tsobj_2)
```

Arguments

`tsobj_1`, `tsobj_2`	A pair of financial time series objects

Description

`iscomp = iscompatible(tsobj_1, tsobj_2)` returns `1` if both financial time series objects `tsobj_1` and `tsobj_2` have the same dates and data series names. It returns `0` if any component is different.

`iscomp = 1` indicates that the two objects contain the same number of data points as well as equal number of data series. However, the values contained in the data series can be different.

Note Data series names are case sensitive.

See Also

`isequal`

isequal

Purpose

Multiple object equality

Syntax

```
iseq = isequal(tsobj_1, tsobj_2, ...)
```

Arguments

`tsobj_1 ...`	A list of financial time series objects

Description

`iseq = isequal(tsobj_1, tsobj_2, ...)` returns `1` if all listed financial time series objects have the same dates, data series names, and values contained in the data series. It returns `0` if any of those components is different.

Note Data series names are case sensitive.

`iseq = 1` implies that each object contains the same number of dates and the same data. Only the descriptions can differ.

See Also

`iscompatible`

Purpose

Check if string is a field name

Syntax

```
F = isfield(tsobj, name)
```

Description

`F = isfield(tsobj, name)` returns true (`1`) if `name` is the name of a data series in `tsobj`. Otherwise, `isfield` returns false (`0`).

See Also

`fieldnames`, `getfield`, `setfield`

issorted

Purpose

Check if dates and times are monotonically increasing

Syntax

```
monod = issorted(tsobj)
```

Arguments

`tsobj`	Financial time series object

Description

`monod = issorted(tsobj)` returns `1` if the dates and times in `tsobj` are monotonically increasing or `0` if they are not.

See Also

`sortfts`

Purpose

Lag time series object

Syntax

```
newfts = lagts(oldfts)
newfts = lagts(oldfts, lagperiod)
newfts = lagts(oldfts, lagperiod, padmode)
```

Arguments

`oldfts`	Financial time series object
`lagperiod`	Number of lag periods expressed in the frequency of the time series object
`padmode`	Data padding value

Description

`lagts` delays a financial time series object by a specified time step.

`newfts = lagts(oldfts)` delays the data series in `oldfts` by one time series date entry and returns the result in the object `newfts`. The end will be padded with zeros, by default.

`newfts = lagts(oldfts, lagperiod)` shifts time series values to the right on an increasing time scale. `lagts` delays the data series to happen at a later time. `lagperiod` is the number of lag periods expressed in the frequency of the time series object `oldfts`. For example, if `oldfts` is a daily time series, `lagperiod` is specified in days. `lagts` pads the data with zeros (default).

`newfts = lagts(oldfts, lagperiod, padmode)` lets you pad the data with an arbitrary value, `NaN`, or `Inf` rather than zeros by setting `padmode` to the desired value.

See Also

`leadts`

leadts

Purpose

Lead time series object

Syntax

```
newfts = leadts(oldfts)
newfts = leadts(oldfts, leadperiod)
newfts = leadts(oldfts, leadperiod, padmode)
```

Arguments

`oldfts`	Financial time series object
`leadperiod`	Number of lead periods expressed in the frequency of the time series object
`padmode`	Data padding value

Description

`leadts` advances a financial time series object by a specified time step.

`newfts = leadts(oldfts)` advances the data series in `oldfts` by one time series date entry and returns the result in the object `newfts`. The end will be padded with zeros, by default.

`newfts = leadts(oldfts, leadperiod)` shifts time series values to the left on an increasing time scale. `leadts` advances the data series to happen at an earlier time. `leadperiod` is the number of lead periods expressed in the frequency of the time series object `oldfts`. For example, if `oldfts` is a daily time series, `leadperiod` is specified in days. `leadts` pads the data with zeros (default).

`newfts = leadts(oldfts, leadperiod, padmode)` lets you pad the data with an arbitrary value, `NaN`, or `Inf` rather than zeros by setting `padmode` to the desired value.

See Also

`lagts`

Purpose

Get number of dates (rows)

Syntax

```
lenfts = length(tsobj)
```

Description

`lenfts = length(tsobj)` returns the number of dates (rows) in the financial time series object `tsobj`. This is the same as issuing `lenfts = size(tsobj, 1)`.

See Also

`size`

`length` in the MATLAB documentation

llow

Purpose

Lowest low

Syntax

```
llv = llow(data)
llv = llow(data, nperiods, dim)
llvts = llow(tsobj, nperiods)
llvts = llow(tsobj, nperiods, ParameterName, ParameterValue)
```

Arguments

`data`	Data series matrix
`nperiods`	(Optional) Number of periods. Default = 14.
`dim`	Dimension
`tsobj`	Financial time series object

Description

`llv = llow(data)` generates a vector of lowest low values for the past 14 periods from the matrix `data`.

`llv = llow(data, nperiods, dim)` generates a vector of lowest low values for the past `nperiods` periods. `dim` indicates the direction in which the lowest low is to be searched. If you input `[]` for `nperiods`, the default is 14.

`llvts = llow(tsobj, nperiods)` generates a vector of lowest low values from `tsobj`, a financial time series object. `tsobj` must include at least the series `Low`. The output `llvts` is a financial time series object with the same dates as `tsobj` and data series named `LowestLow`. If `nperiods` is specified, `llow` generates a financial time series object of lowest low values for the past `nperiods` periods.

`llvts = llow(tsobj, nperiods, ParameterName, ParameterValue)` specifies the name for the required data series when it is different from the default name. The valid parameter name is

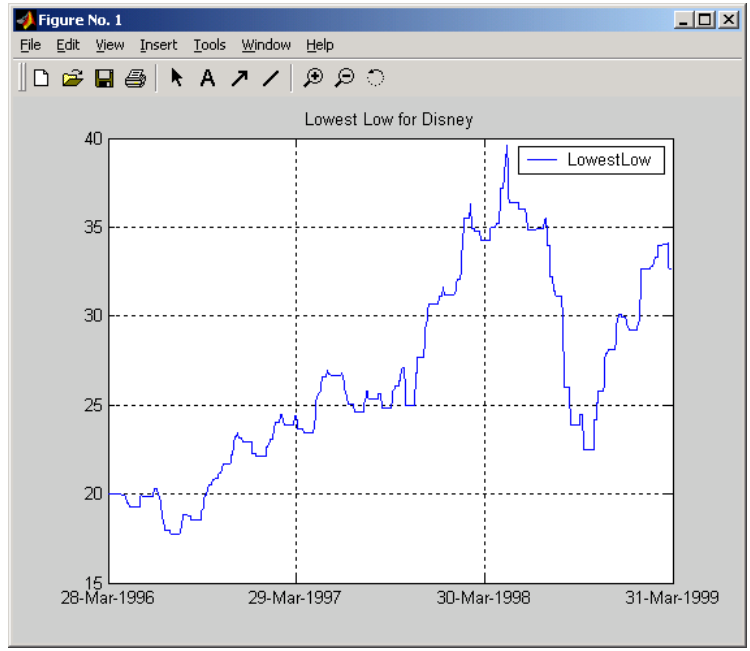
- `LowName`: low prices series name

The parameter value is a string that represents the valid parameter name.

Examples

Compute the lowest low prices for Disney stock and plot the results.

```
load disney.mat
dis_LLow = llow(dis)
plot(dis_LLow)
title('Lowest Low for Disney')
```



See Also

`hhigh`

log

Purpose

Natural logarithm

Syntax

```
newfts = log(tsobj)
```

Description

`newfts = log(tsobj)` calculates the natural logarithm (log base e) of the data series in a financial time series object `tsobj`. It returns another time series object `newfts` containing the natural logarithms.

See Also

`exp`, `log2`, `log10`

Purpose

Base 2 logarithm

Syntax

```
newfts = log2(tsobj)
```

Description

`newfts = log2(tsobj)` calculates the base 2 logarithm of the data series in a financial time series object `tsobj`. It returns another time series object `newfts` containing the logarithms.

See Also

`exp`, `log`, `log10`

log10

Purpose

Common logarithm

Syntax

```
newfts = log10(tsobj)
```

Description

`newfts = log10(tsobj)` calculates the common logarithm (base 10) of all the data in the data series of the financial time series object `tsobj` and returns the result in the object `newfts`.

See Also

`exp`, `log`, `log2`

Purpose

Moving Average Convergence/Divergence (MACD)

Syntax

```
[macdvec, nineperma] = macd(data)
[macdvec, nineperma] = macd(data, dim)
macdts = macd(tsobj, series_name)
```

Arguments

`data`	Data matrix
`dim`	Dimension. Default = 1 (column orientation).
`tsobj`	Financial time series object
`series_name`	Data series name

Description

`[macdvec, nineperma] = macd(data)` calculates the Moving Average Convergence/Divergence (MACD) line, `macdvec`, from the data matrix, `data`, as well as the nine-period exponential moving average, `nineperma`, from the MACD line.

When the two lines are plotted, they can give you an indication of whether to buy or sell a stock, when an overbought or oversold condition is occurring, and when the end of a trend might occur.

The MACD is calculated by subtracting the 26-period (7.5%) exponential moving average from the 12-period (15%) moving average. The 9-day (20%) exponential moving average of the MACD line is used as the *signal* line. For example, when the MACD and the 20% moving average line have just crossed and the MACD line falls below the other line, it is time to sell.

`[macdvec, nineperma] = macd(data, dim)` lets you specify the orientation direction for the input. If the input data is a matrix, you need to indicate whether each row is a set of observations (`dim = 2`) or each column is a set of observations (`dim = 1`, the default).

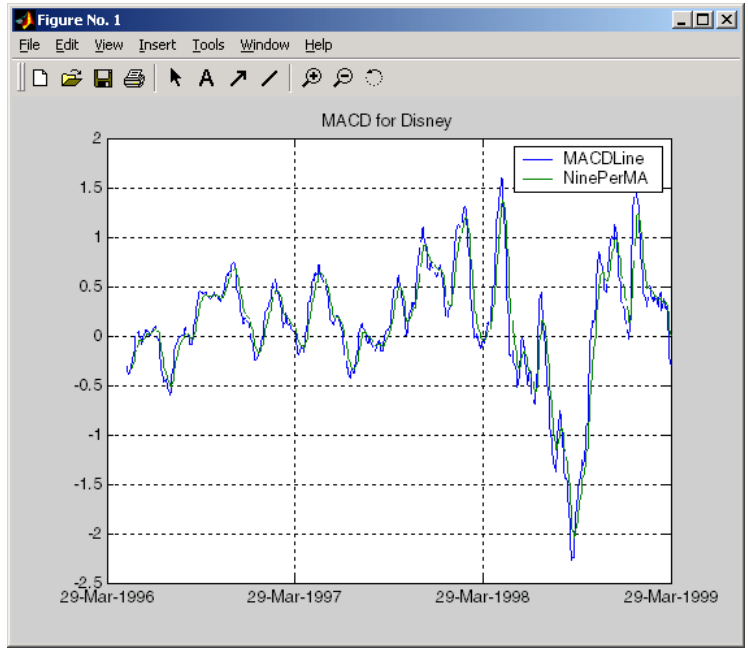
`macdts = macd(tsobj, series_name)` calculates the MACD line from the financial time series `tsobj`, as well as the nine-period exponential moving average from the MACD line. The MACD is calculated for the closing price series in `tsobj`, presumed to have been named `Close`. The result is stored in the financial time series object `macdts`. The `macdts` object has the same dates as the input object `tsobj` and contains only two series, named `MACDLine` and

`NinePerMA`. The first series contains the values representing the MACD line and the second is the nine-period exponential moving average of the MACD line.

Examples

Compute the MACD for Disney stock and plot the results:

```
load disney.mat
dis_CloseMACD = macd(dis);
dis_OpenMACD = macd(dis, 'OPEN');
plot(dis_CloseMACD);
plot(dis_OpenMACD);
title('MACD for Disney')
```



See Also

`adline`, `willad`

Purpose

Maximum value

Syntax

```
tsmax = max(tsobj)
```

Description

`tsmax = max(tsobj)` finds the maximum value in each data series in the financial time series object `tsobj` and returns it in a structure `tsmax`. The `tsmax` structure contains field name(s) identical to the data series name(s).

Note `tsmax` returns only the values and does not return the dates associated with the values. The maximum values are not necessarily from the same date.

See Also

`min`

mean

Purpose

Arithmetic average

Syntax

```
tsmean = mean(tsobj)
```

Description

`tsmean = mean(tsobj)` computes the arithmetic mean of all data in all series in `tsobj` and returns it in a structure `tsmean`. The `tsmean` structure contains field name(s) identical to the data series name(s).

See Also

`peravg`, `tsmovavg`

Purpose

Median price

Syntax

```
mprc = medprice(highp, lowp)
mprc = medprice([highp lowp])
mprcts = medprice(tsobj)
mprcts = medprice(tsobj, ParameterName, ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`tsobj`	Financial time series object

Description

`mprc = medprice(highp, lowp)` calculates the median prices `mprc` from the high (`highp`) and low (`lowp`) prices. The median price is the average of the high and low price for each period.

`mprc = medprice([highp lowp])` accepts a two-column matrix as the input rather than two individual vectors. The columns of the matrix represent the high and low prices, in that order.

`mprcts = medprice(tsobj)` calculates the median prices of a financial time series object `tsobj`. The object must minimally contain the series `High` and `Low`. The median price is the average of the high and low price each period. `mprcts` is a financial time series object with the same dates as `tsobj` and the data series `MedPrice`.

`mprcts = medprice(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

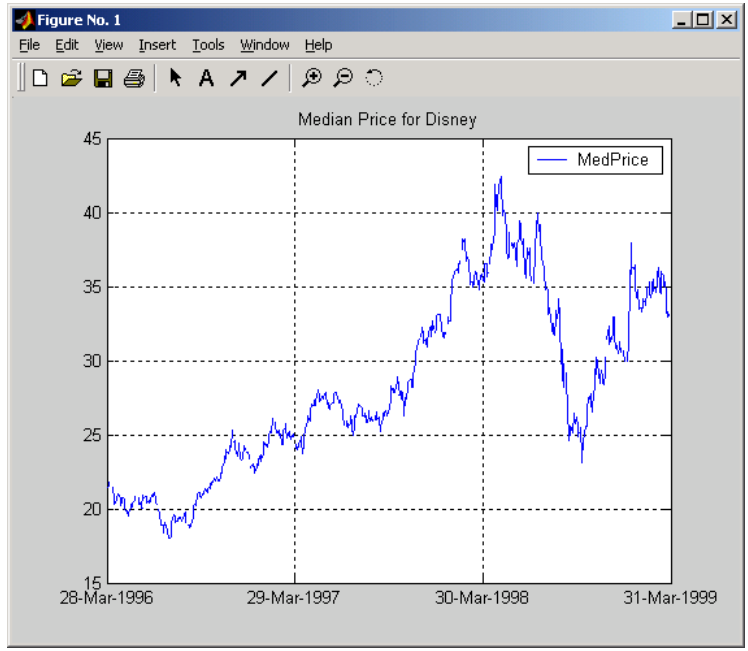
- `HighName`: high prices series name
- `LowName`: low prices series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the median price for Disney stock and plot the results:

```
load disney.mat
dis_MedPrice = medprice(dis)
plot(dis_MedPrice)
title('Median Price for Disney')
```



Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 177 -178.

Purpose

Minimum value

Syntax

```
tsmin = min(tsobj)
```

Description

`tsmin = min(tsobj)` finds the minimum value in each data series in the financial time series object `tsobj` and returns it in the structure `tsmin`. The `tsmin` structure contains field name(s) identical to the data series name(s).

Note `tsmin` returns only the values and does not return the dates associated with the values. The minimum values are not necessarily from the same date.

See Also

`max`

minus

Purpose

Financial time series subtraction

Syntax

```
newfts = tsobj_1 - tsobj_2
newfts = tsobj - array
newfts = array - tsobj
```

Arguments

`tsobj_1`, `tsobj_2`	A pair of financial time series objects
`array`	A scalar value or array with the number of rows equal to the number of dates in `tsobj` and the number of columns equal to the number of data series in `tsobj`

Description

`minus` is an element by element subtraction of the components.

`newfts = tsobj_1 - tsobj_2` subtracts financial time series objects. If an object is to be subtracted from another object, both objects must have the same dates and data series names, although the order need not be the same. The order of the data series, when one financial time series object is subtracted from another, follows the order of the first object.

`newfts = tsobj - array` subtracts an array element by element from a financial time series object.

`newfts = array - tsobj` subtracts a financial time series object element by element from an array.

See Also

`rdivide`, `plus`, `times`

Purpose

Financial time series matrix division

Syntax

```
newfts = tsobj_1 / tsobj_2
newfts = tsobj / array
newfts = array / tsobj
```

Arguments

`tsobj_1`, `tsobj_2`	A pair of financial time series objects
`array`	A scalar value or array with number of rows equal to the number of dates in `tsobj` and number of columns equal to the number of data series in `tsobj`.

Description

The `mrdivide` method divides element by element the components of one financial time series object by the components of the other. You can also divide the whole object by an array or divide a financial time series object into an array.

If an object is to be divided by another object, both objects must have the same dates and data series names, although the order need not be the same. The order of the data series, when an object is divided by another object, follows the order of the first object.

`newfts = tsobj_1 / tsobj_2` divides financial time series objects element by element.

`newfts = tsobj / array` divides a financial time series object element by element by an array.

`newfts = array / tsobj` divides an array element by element by a financial time series object.

For financial time series objects, the `mrdivide` operation is identical to the `rdivide` operation.

See Also

`minus`, `plus`, `rdivide`, `times`

mtimes

Purpose

Financial time series matrix multiplication

Syntax

```
newfts = tsobj_1 * tsobj_2
newfts = tsobj * array
newfts = array * tsobj
```

Arguments

`tsobj_1`, `tsobj_2`	A pair of financial time series objects
`array`	A scalar value or array with number of rows equal to the number of dates in `tsobj` and number of columns equal to the number of data series in `tsobj`.

Description

The `mtimes` method multiplies element by element the components of one financial time series object by the components of the other. You can also multiply the entire object by an array.

If an object is to be multiplied by another object, both objects must have the same dates and data series names, although the order need not be the same. The order of the data series, when an object is multiplied by another object, follows the order of the first object.

`newfts = tsobj_1 * tsobj_2` multiplies financial time series objects element by element.

`newfts = tsobj * array` multiplies a financial time series object element by element by an array.

`newfts = array * tsobj` `newfts = array / tsobj` multiplies an array element by element by a financial time series object.

For financial time series objects, the `mtimes` operation is identical to the `times` operation.

See Also

`mrdivide`, `minus`, `plus`, `times`

Purpose

Negative volume index

Syntax

```
nvi = negvolidx(closep, tvolume, initnvi)
nvi = negvolidx([closep tvolume], initnvi)
nvits = negvolidx(tsobj)
nvits = negvolidx(tsobj, initnvi, ParameterName, ParameterValue,
  ...)
```

Arguments

`closep`	Closing price (vector)
`tvolume`	Volume traded (vector)
`initnvi`	(Optional) Initial value for negative volume index (Default = 100).
`tsobj`	Financial time series object

Description

`nvi = negvolidx(closep, tvolume, initnvi)` calculates the negative volume index from a set of stock closing prices (`closep`) and volume traded (`tvolume`) data. `nvi` is a vector representing the negative volume index. If `initnvi` is specified, `negvolidx` uses that value instead of the default (100).

`nvi = negvolidx([closep tvolume], initnvi)` accepts a two-column matrix, the first column representing the closing prices (`closep`) and the second representing the volume traded (`tvolume`). If `initnvi` is specified, `negvolidx` uses that value instead of the default (100).

`nvits = negvolidx(tsobj)` calculates the negative volume index from the financial time series object `tsobj`. The object must contain, at least, the series `Close` and `Volume`. The `nvits` output is a financial time series object with dates similar to `tsobj` and a data series named `NVI`. The initial value for the negative volume index is arbitrarily set to 100.

`nvits = negvolidx(tsobj, initnvi, ParameterName, ParameterValue, ...)` accepts parameter name/ parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

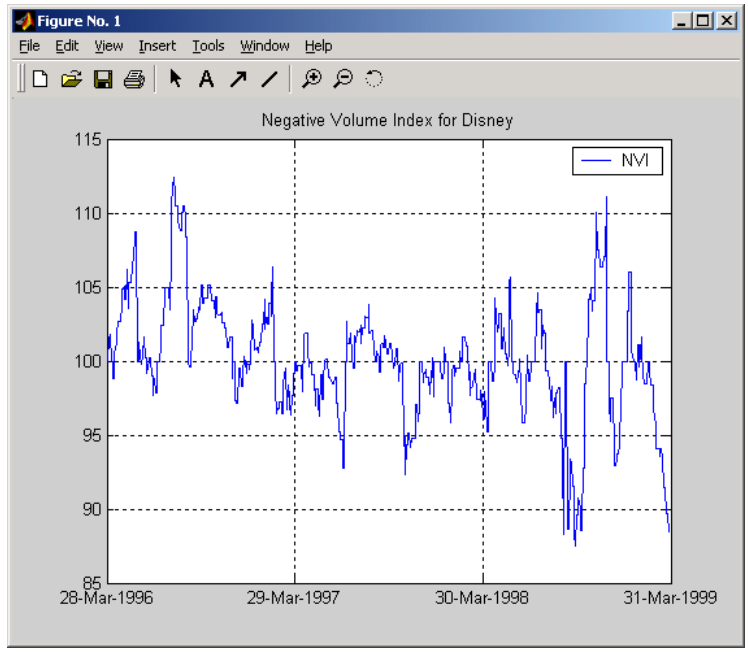
- `CloseName`: closing prices series name
- `VolumeName`: volume traded series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the negative volume index for Disney stock and plot the results:

```
load disney.mat
dis_NegVol = negvolidx(dis)
plot(dis_NegVol)
title('Negative Volume Index for Disney')
```



See Also

`onbalvol`, `posvolidx`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 193 - 194.

Purpose

On-Balance Volume (OBV)

Syntax

```
obv = onbalvol(closep, tvolume)
obv = onbalvol([closep tvolume])
obvts = onbalvol(tsobj)
obvts = onbalvol(tsobj, ParameterName, ParameterValue, ...)
```

Arguments

`closep`	Closing price (vector)
`tvolume`	Volume traded
`tsobj`	Financial time series object

Description

`obv = onbalvol(closep, tvolume)` calculates the On-Balance Volume (OBV) from the stock closing price (`closep`) and volume traded (`tvolume`) data.

`obv = onbalvol([closep tvolume])` accepts a two-column matrix representing the closing price (`closep`) and volume traded (`tvolume`), in that order.

`obvts = onbalvol(tsobj)` calculates the OBV from the stock data in the financial time series object `tsobj`. The object must minimally contain series names `Close` and `Volume`. The `obvts` output is a financial time series object with the same dates as `tsobj` and a series named `OnBalVol`.

`obvts = onbalvol(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name/ parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

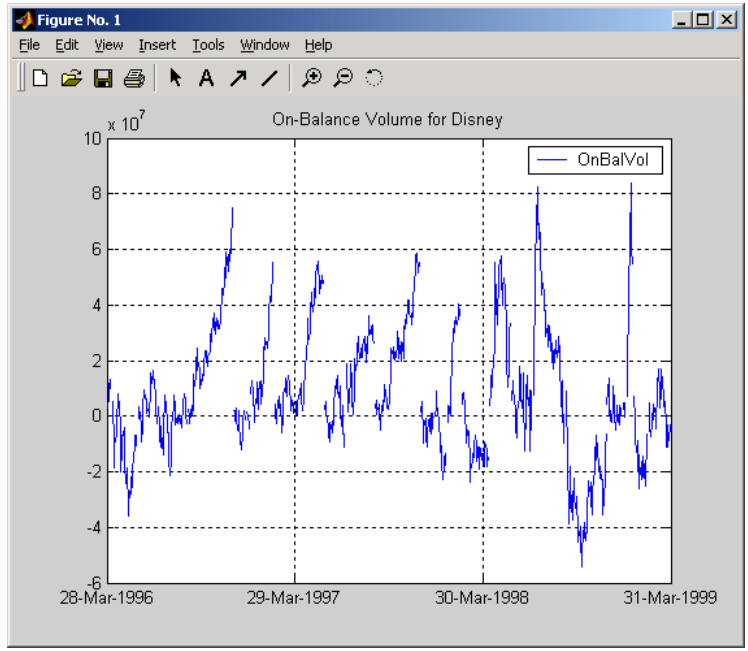
- `CloseName`: closing prices series name
- `VolumeName`: volume traded series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the OBV for Disney stock and plot the results:

```
load disney.mat
dis_OnBalVol = onbalvol(dis)
plot(dis_OnBalVol)
title('On-Balance Volume for Disney')
```



See Also

`negvolidx`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 207 - 209.

Purpose

Periodic average

Syntax

```
avgfts = peravg(tsobj, numperiod)
avgfts = peravg(tsobj, daterange)
```

Arguments

`tsobj`	Financial time series object
`numperiod`	Integer specifying the number of data points over which each periodic average should be averaged
`daterange`	Time period over which the data is averaged

Description

`peravg` calculates periodic averages of a financial time series object. Periodic averages are calculated from the values per period defined. If the period supplied is a string, it is assumed as a range of date string. If the period is entered as numeric, the number represents the number of data points (financial time series periods) to be included in a period for the calculation. For example, if you enter `'01/01/98::01/01/99'` as the period input argument, `peravg` returns the average of the time series between those dates, inclusive. However, if you enter the number `5` as the period input, `peravg` returns a series of averages from the time series data taken 5 date points (financial time series periods) at a time.

`avgfts = peravg(tsobj, numperiod)` returns a structure `avgfts` that contains the periodic (per `numperiod` periods) average of the financial time series object. `avgfts` has field names identical to the data series names of `tsobj`.

`avgfts = peravg(tsobj, daterange)` returns a structure `avgfts` that contains the periodic (as specified by `daterange`) average of the financial time series object. `avgfts` has field names identical to the data series names of `tsobj`.

See Also

`mean`, `tsmovavg`

`mean` in the MATLAB documentation

plot

Purpose

Plot data series

Syntax

```
plot(tsobj)
hp = plot(tsobj)
plot(tsobj, linefmt)
hp = plot(tsobj, linefmt)
plot(..., volumename, bar)
hp = plot(..., volumename, bar)
```

Arguments

`tsobj`	Financial time series object
`linefmt`	(Optional) Line format
`volumename`	(Optional) Specifies which data series is the volume series. `volumename` must be the exact data series name for the volume column (case sensitive).
`bar`	(Optional) `bar = 0` (default). Plot volume as a line. `bar = 1`. Plot volume as a bar chart. The width of each bar is the same as the default in `bar`.

Description

`plot(tsobj)` plots the data series contained in the object `tsobj`. Each data series will be a line. `plot` automatically generates a legend as well as dates on the *x*-axis. Grid is turned on by default. `plot` uses the default color order as if plotting a matrix.

The `plot` command automatically creates subplots when multiple time series are encountered, and they differ greatly on their decimal scales. For example, subplots are generated if one time series data set is in the 10s and another's is in the 10,000s.

`hp = plot(tsobj)` additionally returns the handle(s) to the object(s) inside the plot figure. If there are multiple lines in the plot, `hp` is a vector of multiple handles.

`plot(tsobj, linefmt)` plots the data series in `tsobj` using the line format specified. For a list of possible line formats, see `plot` in the MATLAB documentation. The plot legend is not generated, but the dates on the *x*-axis

and the plot grid are. The specified line format is applied to all data series; that is, all data series will have the same line type.

`hp = plot(tsobj, linefmt)` plots the data series in `tsobj` using the format specified. The plot legend is not generated, but the dates on the *x*-axis and the plot grid are. The specified line format is applied to all data series, that is, all data series can have the same line type. If there are multiple lines in the plot, `hp` is a vector of multiple handles.

`plot(..., volumename, bar)` additionally specifies which data series is the volume. The volume is plotted in a subplot below the other data series. If `bar = 1`, the volume is plotted as a bar chart. Otherwise, a line plot is used.

`hp = plot(..., volumename, bar)` returns handles for each line. If `bar = 1`, the handle to the patch for the bars is also returned.

Note To turn the legend off, enter `legend off` at the MATLAB command line. Once you turn it off, the legend is essentially deleted. To turn it back on, recreate it using the `legend` command as if you are creating it for the first time. To turn the grid off, enter `grid off`. To turn it back on, enter `grid on`.

See Also

`candle`, `chartfts`, `highlow`

`grid`, `legend`, and `plot` in the MATLAB documentation

plus

Purpose

Financial time series addition

Syntax

```
newfts = tsobj_1 + tsobj_2
newfts = tsobj + array
newfts = array + tsobj
```

Arguments

`tsobj_1`, `tsobj_2`	A pair of financial time series objects
`array`	A scalar value or array with the number of rows equal to the number of dates in `tsobj` and the number of columns equal to the number of data series in `tsobj`

Description

`plus` is an element by element addition of the components.

`newfts = tsobj_1 + tsobj_2` adds financial time series objects. If an object is to be added to another object, both objects must have the same dates and data series names, although the order need not be the same. The order of the data series, when one financial time series object is added to another, follows the order of the first object.

`newfts = tsobj + array` adds an array element by element to a financial time series object.

`newfts = array + tsobj` adds a financial time series object element by element to an array.

See Also

`minus`, `rdivide`, `times`

Purpose

Positive volume index

Syntax

```
pvi = posvolidx(closep, tvolume, initpvi)
pvi = posvolidx([closep tvolume], initpvi)
pvits = posvolidx(tsobj)
pvits = posvolidx(tsobj, initpvi, ParameterName, ParameterValue, ...)
```

Arguments

`closep`	Closing price (vector)
`tvolume`	Volume traded (vector)
`initpvi`	(Optional) Initial value for positive volume index Default = 100.
`tsobj`	Financial time series object

Description

`pvi = posvolidx(closep, tvolume, initpvi)` calculates the positive volume index from a set of stock closing prices (`closep`) and volume traded (`tvolume`) data. `pvi` is a vector representing the positive volume index. If `initpvi` is specified, `posvolidx` uses that value instead of the default (100).

`pvi = posvolidx([closep tvolume], initpvi)` accepts a two-column matrix, the first column representing the closing prices (`closep`) and the second representing the volume traded (`tvolume`). If `initpvi` is specified, `posvolidx` uses that value instead of the default (100).

`pvits = posvolidx(tsobj)` calculates the positive volume index from the financial time series object `tsobj`. The object must contain, at least, the series `Close` and `Volume`. The `pvits` output is a financial time series object with dates similar to `tsobj` and a data series named `PVI`. The initial value for the positive volume index is arbitrarily set to 100.

`pvits = posvolidx(tsobj, initpvi, ParameterName, ParameterValue,...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

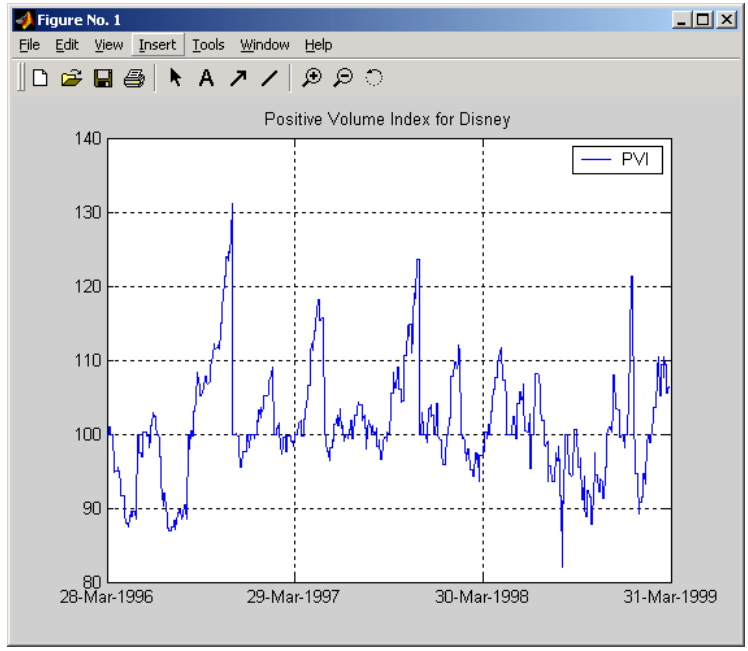
- `CloseName`: closing prices series name
- `VolumeName`: volume traded series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the positive volume index for Disney stock and plot the results:

```
load disney.mat
dis_PosVol = posvolidx(dis)
plot(dis_PosVol)
title('Positive Volume Index for Disney')
```



See Also

`onbalvol`, `negvolidx`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 236 - 238.

Purpose

Financial time series power

Syntax

```
newfts = tsobj .^ array
newfts = array .^tsobj
newfts = tsobj_1 .^ tsobj_2
```

Arguments

`tsobj`	Financial time series object
`array`	A scalar value or array with the number of rows equal to the number of dates in `tsobj` and the number of columns equal to the number of data series in `tsobj`.
`tsobj_1`, `tsobj_2`	A pair of financial time series objects

Description

`newfts = tsobj .^ array` raises all values in the data series of the financial time series object `tsobj` element by element to the power indicated by the array value. The results are stored in another financial time series object `newfts`. The `newfts` object contains the same data series names as `tsobj`.

`newfts = array .^ tsobj` raises the array values element by element to the values contained in the data series of the financial time series object `tsobj`. The results are stored in another financial time series object `newfts`. The `newfts` object contains the same data series names as `tsobj`.

`newfts = tsobj_1 .^ tsobj_2` raises the values in the object `tsobj_1` element by element to the values in the object `tsobj_2`. The data series names, the dates, and the number of data points in both series must be identical. `newfts` contains the same data series names as the original time series objects.

See Also

`minus`, `plus`, `rdivide`, `times`

prcroc

Purpose

Price rate of change

Syntax

```
proc = prcroc(closep, nperiods)
procts = prcroc(tsobj, nperiods)
procts = prcroc(tsobj, nperiods, ParameterName, ParameterValue)
```

Arguments

`closep`	Closing price
`nperiods`	(Optional) Period difference. Default = 12.
`tsobj`	Financial time series object

Description

`proc = prcroc(closep, nperiods)` calculates the price rate of change `proc` from the closing price `closep`. If `nperiods` periods is specified, the price rate of change is calculated between the current closing price and the closing price `nperiods` ago.

`procts = prcroc(tsobj, nperiods)` calculates the price rate of change `procts` from the financial time series object `tsobj`. `tsobj` must contain a data series named `Close`. The output `procts` is a financial time series object with similar dates as `tsobj` and a data series named `PriceROC`. If `nperiods` is specified, the price rate of change is calculated between the current closing price and the closing price `nperiods` ago.

`procts = prcroc(tsobj, nperiods, ParameterName, ParameterValue)` specifies the name for the required data series when it is different from the default name. The valid parameter name is

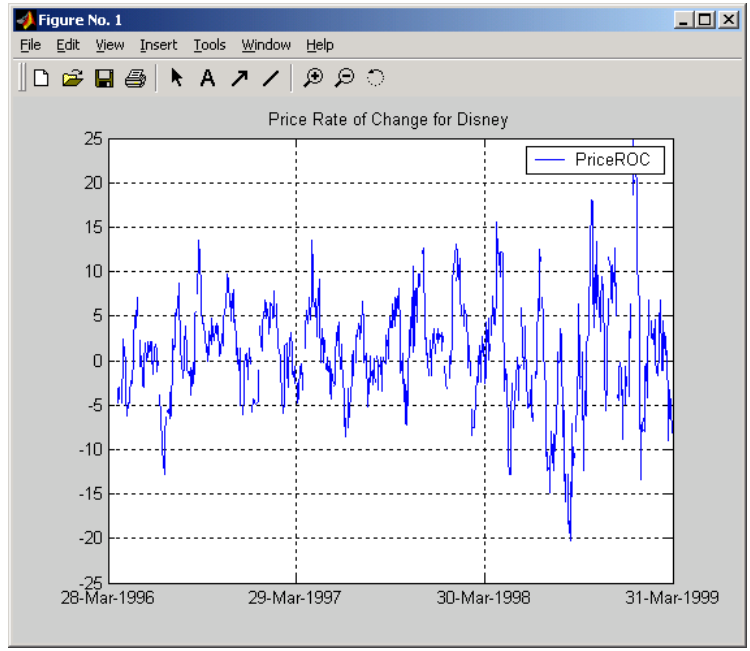
- `CloseName`: closing price series name

The parameter value is a string that represents the valid parameter name.

Examples

Compute the price rate of change for Disney stock and plot the results:

```
load disney.mat
dis_PriceRoc = prcroc(dis)
plot(dis_PriceRoc)
title('Price Rate of Change for Disney')
```



See Also

`volroc`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 243 - 245.

pvtrend

Purpose

Price and Volume Trend (PVT)

Syntax

```
pvt = pvtrend(closep, tvolume)
pvt = pvtrend([closep tvolume])
pvtts = pvtrend(tsobj)
pvtts = pvtrend(tsobj, ParameterName, ParameterValue, ...)
```

Arguments

`closep`	Closing price
`tvolume`	Volume traded
`tsobj`	Financial time series object

Description

`pvt = pvtrend(closep, tvolume)` calculates the Price and Volume Trend (PVT) from the stock closing price (`closep`) data and the volume traded (`tvolume`) data.

`pvt = pvtrend([closep tvolume])` accepts a two-column matrix in which the first column contains the closing prices (`closep`) and the second contains the volume traded (`tvolume`).

`pvtts = pvtrend(tsobj)` calculates the PVT from the stock data contained in the financial time series object `tsobj`. The object `tsobj` must contain the closing price series `Close` and the volume traded series `Volume`. The output `pvtts` is a financial time series object with dates similar to `tsobj` and a data series named `PVT`.

`pvtts = pvtrend(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name/ parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

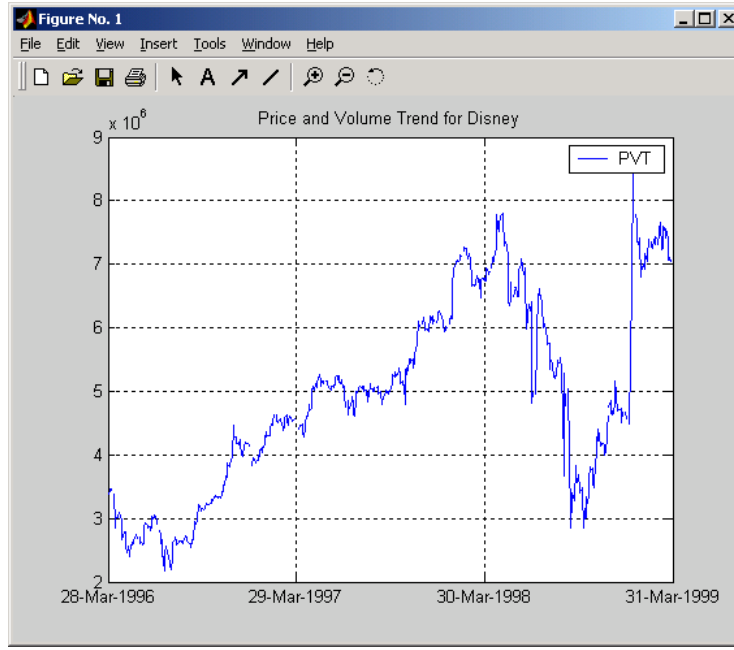
- `CloseName`: closing prices series name
- `VolumeName`: volume traded series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the PVT for Disney stock and plot the results:

```
load disney.mat
dis_PVTrend = pvtrend(dis)
plot(dis_PVTrend)
title('Price and Volume Trend for Disney')
```



Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 239 - 240.

rdivide

Purpose

Financial time series division

Syntax

```
newfts = tsobj_1 ./ tsobj_2
newfts = tsobj ./ array
newfts = array ./ tsobj
```

Arguments

`tsobj_1`, `tsobj_2`	A pair of financial time series objects
`array`	A scalar value or array with the number of rows equal to the number of dates in `tsobj` and the number of columns equal to the number of data series in `tsobj`

Description

The `rdivide` method divides, element by element, the components of one financial time series object by the components of the other. You can also divide the whole object by an array or divide a financial time series object into an array.

If an object is to be divided by another object, both objects must have the same dates and data series names, although the order need not be the same. The order of the data series, when an object is divided by another object, follows the order of the first object.

`newfts = tsobj_1 ./ tsobj_2` divides financial time series objects element by element.

`newfts = tsobj ./ array` divides a financial time series object element by element by an array.

`newfts = array ./ tsobj` divides an array element by element by a financial time series object.

For financial time series objects, the `rdivide` operation is identical to the `mrdivide` operation.

See Also

`minus`, `mrdivide`, `plus`, `times`

Purpose

Downsample data

Syntax

```
newfts = resamplets(oldfts, samplestep)
```

Description

`newfts = resamplets(oldfts, samplestep)` downsamples the data contained in the financial time series object `oldfts` every `samplestep` periods. For example, to have the new financial time series object contain every other data element from `oldfts`, set `samplestep` to `2`.

`newfts` is a financial time series object containing the same data series (names) as the input `oldfts`.

See Also

`filter`

rmfield

Purpose

Remove data series

Syntax

```
fts = rmfield(tsobj, fieldname)
```

Arguments

`tsobj`	Financial time series object
`fieldname`	String array containing the data series name to remove a single series from the object. Cell array of data series names to remove multiple data series from the object at the same time.

Description

`fts = rmfield(tsobj, fieldname)` removes the data series `fieldname` and its contents from the financial time series object `tsobj`.

See Also

`chfield`, `extfield`, `fieldnames`, `getfield`, `isfield`

Purpose

Relative Strength Index (RSI)

Syntax

```
rsi = rsindex(closep, nperiods)
rsits = rsindex(tsobj, nperiods)
rsits = rsindex(tsobj, nperiods, ParameterName, ParameterValue)
```

Arguments

`closep`	Vector of closing prices
`nperiods`	(Optional) Number of periods. Default = 14.
`tsobj`	Financial time series object

Description

`rsi = rsindex(closep, nperiods)` calculates the Relative Strength Index (RSI) from the closing price vector `closep`.

`rsits = rsindex(tsobj, nperiods)` calculates the RSI from the closing price series in the financial time series object `tsobj`. The object `tsobj` must contain at least the series `Close`, representing the closing prices. The output `rsits` is a financial time series object whose dates are the same as `tsobj` and whose data series name is `RSI`.

`rsits = rsindex(tsobj, nperiods, ParameterName, ParameterValue)` accepts a parameter name/parameter value pair as input. This pair specifies the name for the required data series if it is different from the expected default name. The valid parameter name is

- `CloseName`: closing prices series name

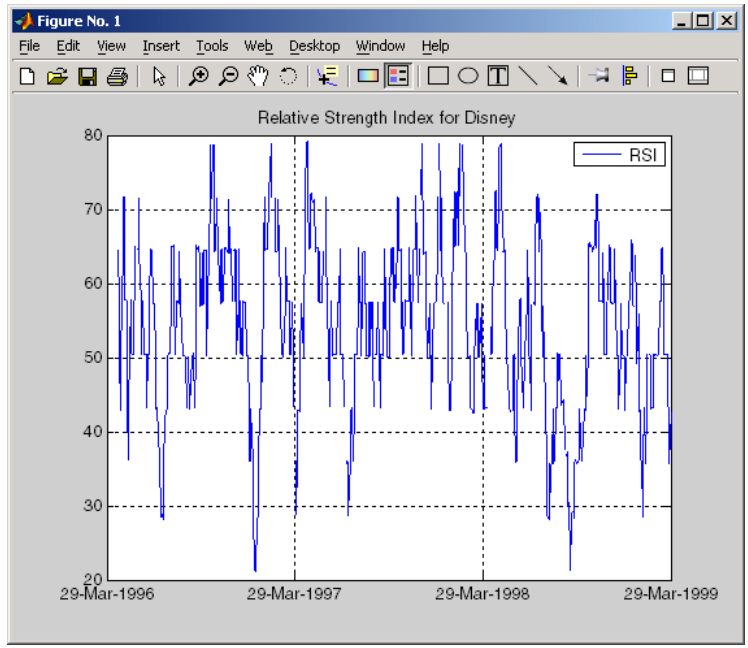
The parameter value is the string that represents the valid parameter name.

Note The relative strength index is calculated by dividing the sum of the closing values for the up days by the sum of the closing values for the down days: `RSI = sum(CLOSEP_up)/sum(CLOSEP_down)`.
Also, the first value of `RSI`, `RISI(1)`, is set as `NaN` to preserve the dimensions of `CLOSEP`.

Examples

Compute the RSI for Disney stock and plot the results:

```
load disney.mat
dis_RSI = rsindex(dis)
plot(dis_RSI)
title('Relative Strength Index for Disney')
```



See Also

`negvolidx`, `posvolidx`

Reference

Murphy, John J., *Technical Analysis of the Futures Market*, New York Institute of Finance, 1986, pp. 295 - 302.

Purpose

Set content of a specific field

Syntax

```
newfts = setfield(tsobj, field, V)
newfts = setfield(tsobj, field, {dates}, V)
```

Description

`setfield` treats the contents of fields in a time series object (`tsobj`) as fields in a structure.

`newfts = setfield(tsobj, field, V)` sets the contents of the specified field to the value `V`. This is equivalent to the syntax `S.field = V`.

`newfts = setfield(tsobj, field, {dates}, V)` sets the contents of the specified field for the specified dates. `dates` can be individual cells of date strings or a cell of a date string range using the `::` operator, e.g., `'03/01/99::03/31/99'`. Dates can contain time-of-day information.

Examples

Example 1. Set the closing value for all days to 3890.

```
load dji30short
format bank
myfts1 = setfield(myfts1, 'Close', 3890);
```

Example 2. Set values for specific times on specific days.

First create a financial time series containing time-of-day data.

```
dates = ['01-Jan-2001';'01-Jan-2001'; '02-Jan-2001'; ...
         '02-Jan-2001'; '03-Jan-2001';'03-Jan-2001'];
times = ['11:00';'12:00';'11:00';'12:00';'11:00';'12:00'];
dates_times = cellstr([dates, repmat(' ',size(dates,1),1),...
                       times]);
myfts = fints(dates_times,[(1:4)'; nan; 6],{'Data1'},1,...
              'My FINTS')

myfts =

    desc:  My FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
```

```
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [        NaN]
    '     "     '    '12:00'          [          6]
```

Now use `setfield` to replace the data in `myfts` with new data starting at 12:00 on January 1, 2001 and ending at 11:00 on January 3, 2001.

```
S = setfield(myfts,'Data1',...
            {'01-Jan-2001 12:00::03-Jan-2001 11:00'},(102:105)')

S =

    desc:  My FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [       1.00]
    '     "     '    '12:00'          [     102.00]
    '02-Jan-2001'    '11:00'          [     103.00]
    '     "     '    '12:00'          [     104.00]
    '03-Jan-2001'    '11:00'          [     105.00]
    '     "     '    '12:00'          [       6.00]
```

See Also

`chfield`, `fieldnames`, `getfield`, `isfield`, `rmfield`

Purpose

Get number of dates and data series

Syntax

```
szfts = size(tsobj)
szfts = size(tsobj, dim)
```

Arguments

`tsobj`	Financial time series object
`dim`	Dimension: `dim = 1` returns number of dates (rows). `dim = 2` returns number of data series (columns).

Description

`szfts = size(tsobj)` returns the number of dates (rows) and the number of data series (columns) in the financial time series object `tsobj`. The result is returned in the vector `szfts`, whose first element is the number of dates and second is the number of data series.

`szfts = size(tsobj, dim)` specifies the dimension you want to extract.

See Also

`length`

`size` in the MATLAB documentation

smoothts

Purpose

Smooth data

Syntax

```
output = smoothts(input)
output = smoothts(input, 'b', wsize)
output = smoothts(input, 'g', wsize, stdev)
output = smoothts(input, 'e', n)
```

Arguments

`input`	A financial time series object or a row-oriented matrix. In a row-oriented matrix each row represents an individual set of observations.
`'b'`, `'g'`, or `'e'`	Smoothing method (essentially the type of filter used). Can be Exponential (`e`), Gaussian (`g`), or Box (`b`). Default = `b`.
`wsize`	Window size (scalar). Default = 5.
`stdev`	Scalar that represents the standard deviation of the Gaussian window. Default = 0.65.
`n`	For Exponential method, specifies window size or exponential factor, depending upon value. `n > 1` (window size) or period length `n < 1` and `> 0` (exponential factor: alpha) `n = 1` (either window size or alpha) If `n` is not supplied, the defaults are `wsize = 5` and `alpha = 0.3333`.

Description

`smoothts` smooths the input data using the specified method.

`output = smoothts(input)` smooths the input data using the default Box method with window size, `wsize`, of `5`.

`output = smoothts(input, 'b', wsize)` smooths the input data using the Box (simple, linear) method. `wsize` specifies the width of the box to be used.

`output = smoothts(input, 'g', wsize, stdev)` smooths the input data using the Gaussian window method.

`output = smoothts(input, 'e', n)` smooths the input data using the Exponential method. `n` can represent the window size (period length) or alpha. If `n > 1`, `n` represents the window size. If `0 < n < 1`, `n` represents alpha, where

$$\alpha = \frac{2}{wsize + 1}$$

If `input` is a financial time series object, `output` is a financial time series object identical to `input` except for contents. If `input` is a row-oriented matrix, `output` is a row-oriented matrix of the same length.

See Also

`tsmovavg`

sortfts

Purpose

Sort financial time series

Syntax

```
sfts = sortfts(tsobj)
sfts = sortfts(tsobj, flag)
sfts = sortfts(tsobj, seriesnames, flag)
[sfts, sidx] = sortfts(...)
```

Arguments

`tsobj`	Financial time series object
`flag`	(Optional) Sort order: `flag = 1`; increasing order (default) `flag = -1`; decreasing order
`seriesnames`	(Optional) String containing a data series name or cell array containing a list of data series names

Description

`sfts = sortfts(tsobj)` sorts the financial time series object `tsobj` in increasing order based only upon the `'dates'` vector if `tsobj` does not contain time-of-day information. If the object includes time-of-day information, the sort is based upon a combination of the `'dates'` and `'times'` vectors. The `'times'` vector cannot be sorted individually.

`sfts = sortfts(tsobj, flag)` sets the order of the sort. `flag = 1`: increasing date and time order. `flag = -1`: decreasing date and time order.

`sfts = sortfts(tsobj, seriesnames, flag)` sorts the financial time series object `tsobj` based upon the data series name(s) `seriesnames`. The `seriesnames` argument can be a single string containing a data series name or a cell array containing a list of data series names. If the optional `flag` is set to `-1`, the sort is in decreasing order.

`[sfts, sidx] = sortfts(...)` additionally returns the index of the original object `tsobj` sorted based on `'dates'` or specified data series name(s).

See Also

`issorted`

`sort` and `sortrows` in the MATLAB documentation

Purpose

Slow stochastics

Syntax

```
[spctk, spctd] = spctkd(fastpctk, fastpctd)
[spctk, spctd] = spctkd([fastpctk  fastpctd])
[spctk, spctd] = spctkd(fastpctk, fastpctd, dperiods, dmamethod)
[spctk, spctd] = spctkd([fastpctk  fastpctd], dperiods, dmamethod)
skdts = spctkd(tsobj)
skdts = spctkd(tsobj, dperiods, dmamethod)
skdts = spctkd(tsobj, dperiods, dmamethod, ParameterName,
    ParameterValue, ...)
```

Arguments

`fastpctk`	Fast stochastic F%K (vector)
`fastpctk`	Fast stochastic F%D (vector)
`dperiods`	(Optional) %D periods. Default = 3.
`dmamethod`	(Optional) %D moving average method. Default = `'e'` (exponential).
`tsobj`	Financial time series object

Description

`[spctk, spctd] = spctkd(fastpctk, fastpctd)` calculates the slow stochastics S%K and S%D. `spctk` and `spctd` are column vectors representing the respective slow stochastics. The inputs must be single column-oriented vectors containing the fast stochastics F%K and F%D.

`[spctk, spctd] = spctkd([fastpctk fastpctd])` accepts a two-column matrix as input. The first column contains the fast stochastic F%K values, and the second contains the fast stochastic F%D values.

`[spctk, spctd] = spctkd(fastpctk, fastpctd, dperiods, dmamethod)` calculates the slow stochastics, S%K and S%D, using the value of `dperiods` to set the number of periods and `dmamethod` to indicate the moving average method. The inputs `fastpctk` and `fastpctk` must contain the fast stochastics, F%K and F%D, in column orientation. `spctk` and `spctd` are column vectors representing the respective slow stochastics.

Valid moving average methods for %D are exponential (`'e'`), triangular (`'t'`), and modified (`'m'`). See `tsmovavg` for explanations of these methods.

`[spctk, spctd] = spctkd([fastpctk  fastpctd], dperiods, dmamethod)` accepts a two-column matrix rather than two separate vectors. The first column contains the F%K values, and the second contains the F%D values.

`skdts = spctkd(tsobj)` calculates the slow stochastics, S%K and S%D. `tsobj` must contain the fast stochastics, F%K and F%D, in data series named `PercentK` and `PercentD`. The `skdts` output is a financial time series object with the same dates as `tsobj`. Within `tsobj` the two series `SlowPctK` and `SlowPctD` represent the respective slow stochastics.

`skdts = spctkd(tsobj, dperiods, dmamethod)` allows you to specify the length and the method of the moving average used to calculate S%D values.

`skdts = spctkd(tsobj, dperiods, dmamethod, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

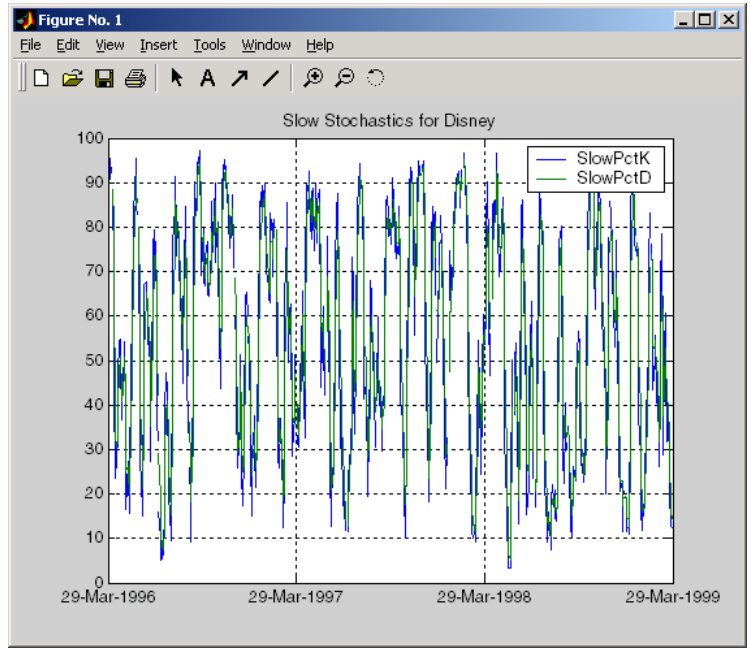
- `KName`: F%K series name
- `DName`: F%D series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the slow stochastics for Disney stock and plot the results:

```
load disney.mat
dis_FastStoch = fpctkd(dis);
dis_SlowStoch = spctkd(dis_FastStoch);
plot(dis_SlowStoch)
title('Slow Stochastics for Disney')
```



See Also

`fpctkd`, `stochosc`, `tsmovavg`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 268 - 271.

std

Purpose

Standard deviation

Syntax

```
tsstd = std(tsobj)
tsstd = std(tsobj, flag)
```

Arguments

`tsobj`	Financial time series object
`flag`	(Optional) Normalization factor: `flag = 1` normalizes by n (number of observations). `flag = 0` normalizes by n-1.

Description

`tsstd = std(tsobj)` computes the standard deviation of each data series in the financial time series object `tsobj` and returns the results in `tsstd`. The `tsstd` output is a structure with field name(s) identical to the data series name(s).

`tsstd = std(tsobj, flag)` normalizes the data as indicated by `flag`.

See Also

`hist`, `mean`

Purpose

Stochastic oscillator

Syntax

```
stosc = stochosc(highp, lowp, closep)
stosc = stochosc([highp lowp closep])
stosc = stochosc(highp, lowp, closep, kperiods, dperiods, dmamethod)
stosc = stochosc([highp lowp closep], kperiods, dperiods, dmamethod)
stoscts = stochosc(tsobj, kperiods, dperiods, dmamethod)
stoscts = stochosc(tsobj, kperiods, dperiods, dmamethod,
  ParameterName,  ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`closep`	Closing price (vector)
`kperiods`	(Optional) %K periods. Default = 10.
`dperiods`	(Optional) %D periods. Default = 3.
`damethod`	(Optional) %D moving average method. Default = `'e'` (exponential).
`tsobj`	Financial time series object

Description

`stosc = stochosc(highp, lowp, closep)` calculates the fast stochastics F%K and F%D from the stock price data `highp` (high prices), `lowp` (low prices), and `closep` (closing prices). `stosc` is a two-column matrix whose first column is the F%K values and second is the F%D values.

`stosc = stochosc([highp lowp closep])` accepts a three-column matrix of high (`highp`), low (`lowp`), and closing prices (`closep`), in that order.

`stosc = stochosc(highp, lowp, closep, kperiods, dperiods, dmamethod)` calculates the fast stochastics F%K and F%D from the stock price data `highp` (high prices), `lowp` (low prices), and `closep` (closing prices). `kperiods` sets the %K period. `dperiods` sets the %D period. `damethod` specifies the %D moving average method. Valid moving average methods for %D are exponential (`'e'`) and triangular (`'t'`). See `tsmovavg` for explanations of these methods.

`stosc= stochosc([highp lowp closep], kperiods, dperiods, dmamethod)` accepts a three-column matrix of high (`highp`), low (`lowp`), and closing prices (`closep`), in that order.

`stoscts = stochosc(tsobj, kperiods, dperiods, dmamethod)` calculates the fast stochastics F%K and F%D from the stock price data in the financial time series object `tsobj`. `tsobj` must minimally contain the series `High` (high prices), `Low` (low prices), and `Close` (closing prices). `stoscts` is a financial time series object with similar dates to `tsobj` and two data series named `SOK` and `SOD`.

`stoscts = stochosc(tsobj, kperiods, dperiods, dmamethod, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

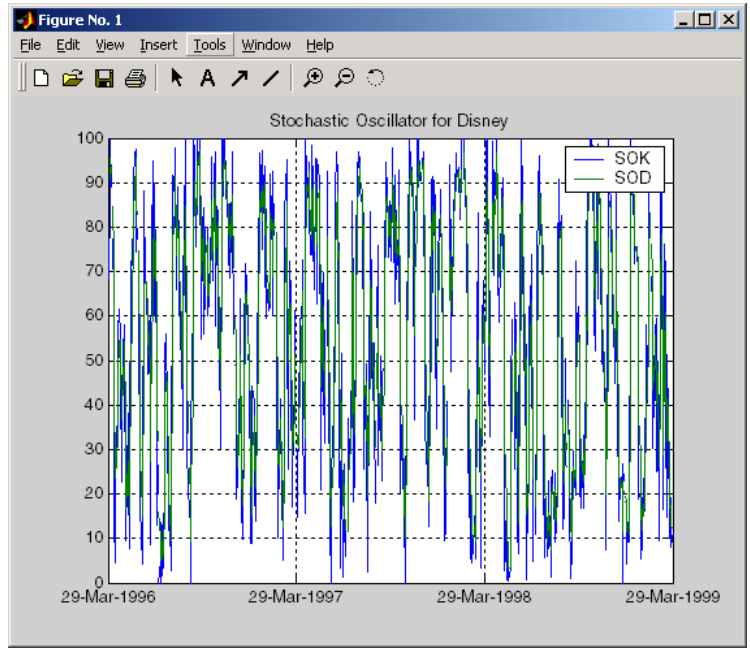
- `HighName`: high prices series name
- `LowName`: low prices series name
- `CloseName`: closing prices series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the stochastic oscillator for Disney stock and plot the results:

```
load disney.mat
dis_StochOsc = stochosc(dis)
plot(dis_StochOsc)
title('Stochastic Oscillator for Disney')
```



See Also

`fpctkd`, `spctkd`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 268 - 271.

subsasgn

Purpose

Content assignment

Description

`subasgn` assigns content to a component within a financial time series object. `subasgn` supports integer indexing or date string indexing into the time series object with values assigned to the designated components. *Serial date numbers cannot be used as indices.* To use date string indexing, enclose the date string(s) in a pair of single quotation marks `' '`.

You can use integer indexing on the object as in any other MATLAB matrix. It will return the appropriate entry(ies) from the object.

You must specify the component to which you want to assign values. An assigned value must be either a scalar or a column vector.

Examples

Given a time series `myfts` with a default data series name of `series1`,

```
myfts.series1('07/01/98::07/03/98') = [1 2 3]';
```

assigns the values 1, 2, and 3 corresponding to the first three days of July, 1998.

```
myfts('07/01/98::07/05/98')

ans =

      desc:  Data Assignment
      freq:  Daily (1)

      'dates:  (5)'    'series1:  (5)'
      '01-Jul-1998'    [            1]
      '02-Jul-1998'    [            2]
      '03-Jul-1998'    [            3]
      '04-Jul-1998'    [       4561.2]
      '05-Jul-1998'    [       5612.3]
```

When the financial time series object contains a time-of-day specification, you can assign data to a specific time on a specific day. For example, create a financial time series object called `timeday` containing both dates and times:

```
dates = ['01-Jan-2001';'01-Jan-2001'; '02-Jan-2001'; ...
'02-Jan-2001'; '03-Jan-2001';'03-Jan-2001'];
times = ['11:00';'12:00';'11:00';'12:00';'11:00';'12:00'];
```

```
dates_times = cellstr([dates, repmat(' ',size(dates,1),1),...
times]);
timeday = fints(dates_times,(1:6)',{'Data1'},1,'My first FINTS')

timeday =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [          5]
    '     "     '    '12:00'          [          6]
```

Use integer indexing to assign the value 999 to the first item in the object.

```
timeday(1) = 999

timeday =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [        999]
    '     "     '    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [          5]
    '     "     '    '12:00'          [          6]
```

For value assignment using date strings, enclose the string in single quotation marks. If a date has multiple times, designating only the date and assigning a value results in every element of that date taking on the assigned value. For example, to assign the value 0.5 to all times-of-day on January 1, 2001, enter

```
timedata('01-Jan-2001') = 0.5
```

The result is

```
timedata =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [     0.5000]
    '     "     '    '12:00'          [     0.5000]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [          5]
    '     "     '    '12:00'          [          6]
```

To access the individual components of the financial time series object, use the structure syntax. For example, to assign a range of data to all the data items in the series `Data1`, you can use

```
timedata.Data1 = (0: .1 : .5)'

timedata =

    desc:  My first FINTS
    freq:  Daily (1)

    'dates:  (6)'    'times:  (6)'    'Data1:  (6)'
    '01-Jan-2001'    '11:00'          [          0]
    '     "     '    '12:00'          [     0.1000]
    '02-Jan-2001'    '11:00'          [     0.2000]
    '     "     '    '12:00'          [     0.3000]
    '03-Jan-2001'    '11:00'          [     0.4000]
    '     "     '    '12:00'          [     0.5000]
```

See Also

`datestr` in the Financial Toolbox documentation

`subsref`

Purpose

Subscripted reference

Description

`subsref` implements indexing for a financial time series object. Integer indexing or date (and time) string indexing is allowed. *Serial date numbers cannot be used as indices.*

To use date string indexing, enclose the date string(s) in a pair of single quotation marks `' '`.

You can use integer indexing on the object as in any other MATLAB matrix. It returns the appropriate entry(ies) from the object.

Additionally, `subsref` lets you access the individual components of the object using the structure syntax.

Examples

Create a time series named `myfts`:

```
myfts = fints((datenum('07/01/98'):datenum('07/01/98')+4)',...
[1234.56; 2345.61; 3456.12; 4561.23; 5612.34], [], 'Daily',...
'Data Reference');
```

Extract the data for the single day July 1, 1998:

```
myfts('07/01/98')

ans =

     desc:  Data Reference
     freq:  Daily (1)

     'dates:  (1)'    'series1:  (1)'
     '01-Jul-1998'    [       1234.6]
```

Now, extract the data for the range of dates July 1, 1998, through July 5, 1998:

```
myfts('07/01/98::07/03/98')

ans =

    desc:  Data Reference
    freq:  Daily (1)

    'dates:  (3)'    'series1:  (3)'
    '01-Jul-1998'    [       1234.6]
    '02-Jul-1998'    [       2345.6]
    '03-Jul-1998'    [       3456.1]
```

You can use the MATLAB structure syntax to access the individual components of a financial time series object. To get the description field of `myfts`, enter

```
myfts.desc
```

at the command line, which returns

```
ans =
Data Reference
```

Similarly

```
myfts.series1
```

returns

```
 ans =

    desc:  Data Reference
    freq:  Daily (1)

    'dates:  (5)'    'series1:  (5)'
    '01-Jul-1998'    [       1234.6]
    '02-Jul-1998'    [       2345.6]
    '03-Jul-1998'    [       3456.1]
    '04-Jul-1998'    [       4561.2]
    '05-Jul-1998'    [       5612.3]
```

The syntax for integer indexing is the same as for any other MATLAB matrix. Create a new financial time series object containing both dates and times:

```
dates = ['01-Jan-2001';'01-Jan-2001'; '02-Jan-2001'; ...
         '02-Jan-2001'; '03-Jan-2001';'03-Jan-2001'];
times = ['11:00';'12:00';'11:00';'12:00';'11:00';'12:00'];
dates_times = cellstr([dates, repmat(' ',size(dates,1),1),...
                       times]);
anewfts = fints(dates_times,(1:6)',{'Data1'},1,'Another FinTs');
```

Use integer indexing to extract the second and third data items from the object.

```
anewfts(2:3)

ans =

    desc:  Another FinTs
    freq:  Daily (1)

    'dates:  (2)'    'times:  (2)'    'Data1:  (2)'
    '01-Jan-2001'    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
```

For date or string enclose the indexing string in a pair of single quotation marks.

If there is one date with multiple times, indexing with only the date returns all the times for that specific date:

```
anewfts('01-Jan-2001')

ans =

    desc:  Another FinTs
    freq:  Daily (1)

    'dates:  (2)'    'times:  (2)'    'Data1:  (2)'
    '01-Jan-2001'    '11:00'          [          1]
    '     "     '    '12:00'          [          2]
```

To specify one specific date and time, index with that date and time:

```
anewfts('01-Jan-2001 12:00')

ans =

    desc:  Another FinTs
    freq:  Daily (1)

    'dates:  (1)'    'times:  (1)'    'Data1:  (1)'
    '01-Jan-2001'    '12:00'          [          2]
```

To specify a range of dates and times, use the double colon (`::`) operator:

```
anewfts('01-Jan-2001 12:00::03-Jan-2001 11:00')

ans =

    desc:  Another FinTs
    freq:  Daily (1)

    'dates:  (4)'    'times:  (4)'    'Data1:  (4)'
    '01-Jan-2001'    '12:00'          [          2]
    '02-Jan-2001'    '11:00'          [          3]
    '     "     '    '12:00'          [          4]
    '03-Jan-2001'    '11:00'          [          5]
```

To request all the dates, times, and data, use the `::` operator without specifiying any specific date or time:

```
anewfts('::')
```

See Also

`fts2mat`, `subsasgn`

`datestr` in the Financial Toolbox documentation

Purpose

Financial time series multiplication

Syntax

```
newfts = tsobj_1 .* tsobj_2
newfts = tsobj .* array
newfts = array .* tsobj
```

Arguments

`tsobj_1`, `tsobj_2`	A pair of financial time series objects
`array`	A scalar value or array with the number of rows equal to the number of dates in `tsobj` and the number of columns equal to the number of data series in `tsobj`

Description

The `times` method multiplies element by element the components of one financial time series object by the components of the other. You can also multiply the entire object by an array.

If an object is to be multiplied by another object, both objects must have the same dates and data series names, although the order need not be the same. The order of the data series, when an object is multiplied by another object, follows the order of the first object.

`newfts = tsobj_1 .* tsobj_2` multiplies financial time series objects element by element.

`newfts = tsobj .* array` multiplies a financial time series object element by element by an array.

`newfts = array .* tsobj` `newfts = array / tsobj` multiplies an array element by element by a financial time series object.

For financial time series objects, the `times` operation is identical to the `mtimes` operation.

See Also

`minus`, `mtimes`, `plus`, `rdivide`

toannual

Purpose

Convert to annual

Syntax

```
newfts = toannual(oldfts)
```

Description

`newfts = toannual(oldfts)` converts a financial time series of any frequency to one of an annual frequency. `toannual` sets the dates to the end of the year (December 31).

`toannual` displays only the last date and last time of the end of the year.

If `oldfts` does not contain time-of-day data, `newfts` does not contain a `times` vector.

If `oldfts` contains time-of-day data, `newfts` contains a `times` vector that replicates the times in `oldfts`.

If December 31 for a particular year does not appear in `oldfts`, and `oldfts` contains time-of-day information, the time-of-day for that specific date is set to `00:00`.

See Also

`convertto`, `todaily`, `tomonthly`, `toquarterly`, `tosemi`, `toweekly`

Purpose

Convert to daily

Syntax

```
newfts = todaily(oldfts)
```

Description

`newfts = todaily(oldfts)` converts a financial time series of any frequency to one of a daily frequency. `todaily` assumes a five-day business week. If `oldfts` contains weekend data, `todaily` removes that data when creating `newfts`.

To create a daily time series from nondaily `oldfts`, `todaily` copies the periodic value for the number of days in the period of the input time series. For example, if `oldfts` is a weekly time series, the value for each week is replicated four additional times until the next week's value is encountered. The process is then repeated for the next week.

If `oldfts` does not contain time-of-day data, `newfts` does not contain a `times` vector.

If `oldfts` contains time-of-day data, `newfts` contains a `times` vector that replicates the times in `oldfts`.

If `newfts` contains a date (e.g., January 31) that does not appear in `oldfts`, and `oldfts` contains time-of-day information, the time-of-day for that date is set to `00:00`.

See Also

`convertto`, `toannual`, `tomonthly`, `toquarterly`, `tosemi`, `toweekly`

todecimal

Purpose

Fractional to decimal conversion

Syntax

```
usddec = todecimal(quote, fracpart)
```

Description

`usddec = todecimal(quote, fracpart)` returns the decimal equivalent, `usddec`, of a security whose price is normally quoted as a whole number and a fraction (`quote`). `fracpart` indicates the fractional base (denominator) with which the security is normally quoted (default = 32).

Examples

In the *Wall Street Journal,* bond prices are quoted in fractional form based on a denominator of 32. For example, if you see the quoted price is 100:05 it means 100 5/32. To find the equivalent decimal value, enter

```
usddec = todecimal(100.05)

usddec =
   100.1563

usddec = todecimal(97.04, 16)

usddec =
   97.2500
```

Note The convention of using `.` (period) as a substitute for `:` (colon) in the input is adopted from Microsoft Excel.

See Also

`toquoted`

Purpose

Convert to monthly

Syntax

```
newfts = tomonthly(oldfts)
```

Description

`newfts = tomonthly(oldfts)` converts a financial time series of any frequency to one of a monthly frequency. `tomonthly` assumes a five-day business week.

If `oldfts` is a daily or weekly time series, the monthly values in `newfts` are the averages of the input daily or weekly values. If `oldfts` is a quarterly, semiannual, or annual time series, the input values are replicated as many times as necessary to fill the monthly time series. Dates are set to the end of the month.

`tomonthly` displays only the last date and last time of the end of each month.

If `oldfts` does not contain time-of-day data, `newfts` does not contain a `times` vector.

If `oldfts` contains time-of-day data, `newfts` contains a `times` vector that replicates the times in `oldfts`.

If `newfts` contains a date (e.g., January 31) that does not appear in `oldfts`, and `oldfts` contains time-of-day information, the time-of-day for that date is set to `00:00`.

See Also

`convertto`, `toannual`, `todaily`, `toquarterly`, `tosemi`, `toweekly`

toquarterly

Purpose

Convert to quarterly

Syntax

```
newfts = toquarterly(oldfts)
```

Description

`newfts = toquarterly(oldfts)` converts a financial time series of any frequency to one of a quarterly frequency. `toquarterly` assumes a five-day business week.

If `oldfts` is a daily, weekly, or monthly time series, the quarterly values in `newfts` are the averages of the input values for the quarter. If `oldfts` is a semiannual or annual time series, the input values are replicated as many times as necessary to fill the quarterly time series.

Dates in `newfts` are set to the end of the quarters (March 31, June 30, September 30, and December 31).

If `oldfts` does not contain time-of-day data, `newfts` does not contain a `times` vector.

If `oldfts` contains time-of-day data, `newfts` contains a `times` vector that replicates the times in `oldfts`.

If `newfts` contains a date (e.g., March 31) that does not appear in `oldfts`, and `oldfts` contains time-of-day information, the time-of-day for that date is set to `00:00`.

See Also

`convertto`, `toannual`, `todaily`, `tomonthly`, `tosemi`, `toweekly`

Purpose

Decimal to fractional conversion

Syntax

```
quote = toquoted(usddec, fracpart)
```

Description

`quote = toquoted(usddec, fracpart)` returns the fractional equivalent, `quote`, of the decimal figure, `usddec`, based on the fractional base (denominator), `fracpart`. The fractional bases are the ones used for quoting equity prices in the United States (denominator 2, 4, 8, 16, or 32). If `fracpart` is not entered, the denominator 32 is assumed.

Examples

A United States equity price in decimal form is 101.625. To convert this to fractional form in eighths of a dollar:

```
quote = toquoted(101.625, 8)

quote =
        101.05
```

The answer is interpreted as 101 5/8.

Note The convention of using `.` (period) as a substitute for `:` (colon) in the output is adopted from Microsoft Excel.

See Also

`todecimal`

tosemi

Purpose

Convert to semiannual

Syntax

```
newfts = tosemi(oldfts)
```

Description

`newfts = tosemi(oldfts)` converts a financial time series of any frequency to one of a semiannual frequency. `tosemi` sets the dates to the end of each semiannual time period (June 30 and December 31).

`tosemi` displays only the last date and last time of the end of each semiannual period.

If `oldfts` does not contain time-of-day data, `newfts` does not contain a `times` vector.

If `oldfts` contains time-of-day data, `newfts` contains a `times` vector that replicates the times in `oldfts`.

If `newfts` contains a date (e.g., June 30) that does not appear in `oldfts`, and `oldfts` contains time-of-day information, the time-of-day for that date is set to `00:00`.

See Also

`convertto`, `toannual`, `todaily`, `tomonthly`, `toquarterly`, `toweekly`

Purpose

Convert to weekly

Syntax

```
newfts = toweekly(oldfts)
```

Description

`newfts = toweekly(oldfts)` converts a financial time series of any frequency to one of a weekly frequency. `toweekly` assumes a five-day business week. All days in `newfts` are set to Fridays.

If `oldfts` is a daily series, `newfts` is a financial time series containing data for Fridays only. If `oldfts` is a monthly, quarterly, semiannual, or annual time series, the input values are replicated as many times as there are Fridays to fill the weekly time series.

`toweekly` displays only the last date and last time of the Friday of each week.

If `oldfts` does not contain time-of-day data, `newfts` does not contain a `times` vector.

If `oldfts` contains time-of-day data, `newfts` contains a `times` vector that replicates the times in `oldfts`.

If `newfts` contains a date (e.g., January 31) that does not appear in `oldfts`, and `oldfts` contains time-of-day information, the time-of-day for that date is set to `00:00`.

See Also

`convertto`, `toannual`, `todaily`, `tomonthly`, `toquarterly`, `tosemi`

tsaccel

Purpose

Acceleration between periods

Syntax

```
acc = tsaccel(data, nperiods, datatype)
accts = tsaccel(tsobj, nperiods, datatype)
```

Arguments

`data`	Data series
`nperiods`	(Optional) Number of periods. Default = 12.
`datatype`	(Optional) Indicates whether `data` contains the data itself or the momentum of the data: `0` = `data` contains the data itself (default). `1` = `data` contains the momentum of the data.
`tsobj`	Name of an existing financial time series object

Description

Acceleration is the difference of two momentums separated by some number of periods.

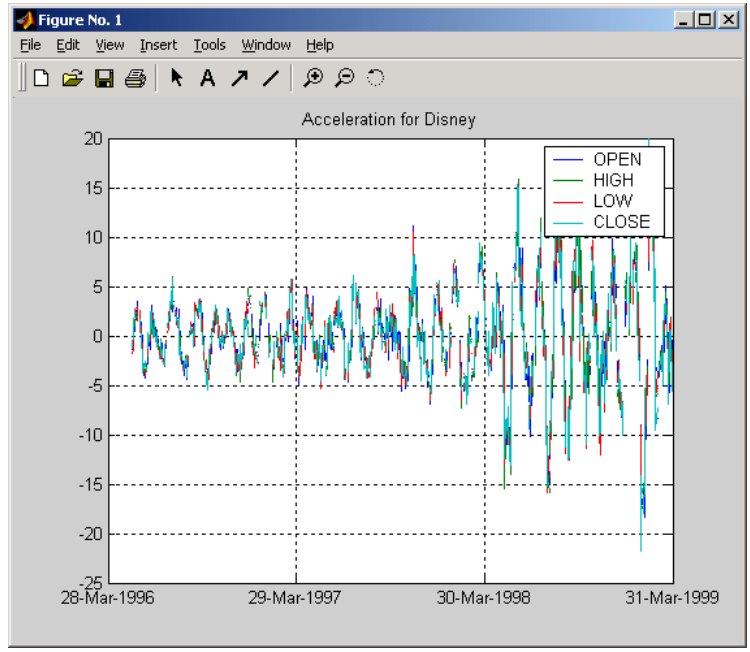
`acc = tsaccel(data, nperiods, datatype)` calculates the acceleration of a data series, essentially the difference of the current momentum with the momentum some number of periods ago. If `nperiods` is specified, `tsaccel` calculates the acceleration of a data series `data` with time distance of `nperiods` periods.

`accts = tsaccel(tsobj, nperiods, datatype)` calculates the acceleration of the data series in the financial time series object `tsobj`, essentially the difference of the current momentum with the momentum some number of periods ago. Each data series in `tsobj` is treated individually. `accts` is a financial time series object with similar dates and data series names as `tsobj`.

Examples

Compute the acceleration for Disney stock and plot the results:

```
load disney.mat
dis = rmfield(dis,'VOLUME') % remove VOLUME field
dis_Accel = tsaccel(dis);
plot(dis_Accel)
title('Acceleration for Disney')
```



See Also

`tsmom`

Reference

Kaufman, P. J., *The New Commodity Trading Systems and Methods*, New York: John Wiley & Sons, 1987.

tsmom

Purpose

Momentum between periods

Syntax

```
mom = tsmom(data, nperiods)
momts = tsmom(tsobj, nperiods)
```

Arguments

`data`	Data series. Column-oriented vector or matrix.
`nperiods`	(Optional) Number of periods. Default = 12.
`tsobj`	Financial time series object

Description

Momentum is the difference between two prices (data points) separated by a number of periods.

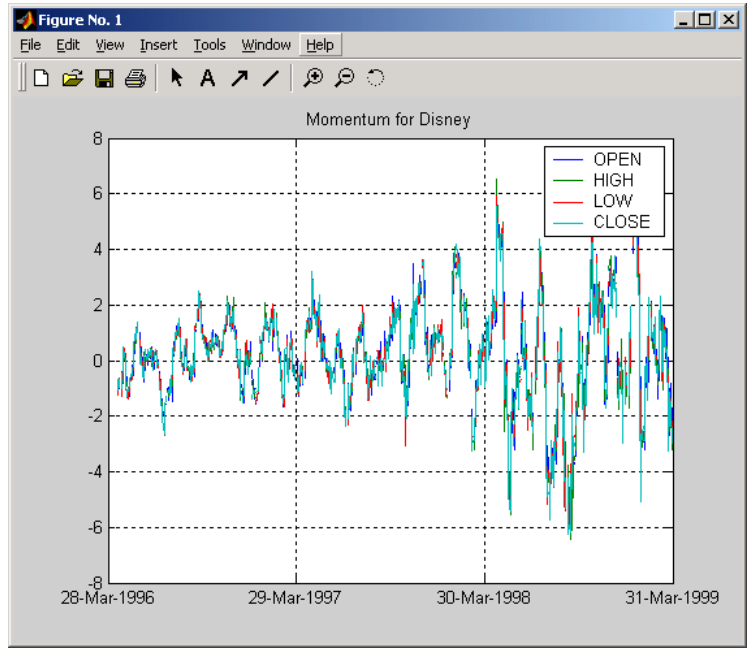
`mom = tsmom(data, nperiods)` calculates the momentum of a data series data. If `nperiods` is specified, `tsmom` uses that value instead of the default 12.

`momts = tsmom(tsobj, nperiods)` calculates the momentum of all data series in the financial time series object `tsobj`. Each data series in `tsobj` is treated individually. `momts` is a financial time series object with similar dates and data series names as `tsobj`. If `nperiods` is specified, `tsmom` uses that value instead of the default 12.

Examples

Compute the momentum for Disney stock and plot the results:

```
load disney.mat
dis = rmfield(dis,'VOLUME') % remove VOLUME field
dis_Mom = tsmom(dis);
plot(dis_Mom)
title('Momentum for Disney')
```



See Also

`tsaccel`

tsmovavg

Purpose

Moving average

Syntax

```
output = tsmovavg(tsobj, 's', lag)              (Simple)
output = tsmovavg(vector, 's', lag, dim)
output = tsmovavg(tsobj, 'e', timeperiod)       (Exponential)
output = tsmovavg(vector, 'e', timeperiod, dim)
output = tsmovavg(tsobj, 't', numperiod)        (Triangular)
output = tsmovavg(vector, 't', numperiod, dim)
output = tsmovavg(tsobj, 'w', weights)          (Weighted)
output = tsmovavg(vector, 'w', weights, dim)
output = tsmovavg(tsobj, 'm', numperiod)        (Modified)
output = tsmovavg(vector, 'm', numperiod, dim)
```

Arguments

`tsobj`	Financial time series object
`lag`	Number of previous data points
`vector`	Row vector or row-oriented matrix. Each row is a set of observations.
`dim`	(Optional) Specifies dimension when `input` is a vector or matrix. Default = 2 (Row-oriented matrix: each row is a variable, and each column is an observation.). If `dim = 1`, input is assumed to be a column vector or column-oriented matrix (each column is a variable and each row an observation). `output` is identical in format to `input`.
`timeperiod`	Length of time period
`numperiod`	Number of periods considered
`weights`	Weights for each element in the window

Description

`output = tsmovavg(tsobj, 's', lag)` and `output = tsmovavg(vector, 's', lag, dim)` compute the simple moving average. `lag` indicates the number of previous data points used in conjunction with the current data point when calculating the moving average.

`output = tsmovavg(tsobj, 'e', timeperiod)` and `output = tsmovavg(vector, 'e', timeperiod, dim)` compute the exponential weighted moving average. The exponential moving average is a weighted moving average, where `timeperiod` specifies the time period. Exponential moving averages reduce the lag by applying more weight to recent prices. For example, a 10-period exponential moving average weights the most recent price by 18.18%. (`2/(timeperiod + 1)`).

`output = tsmovavg(tsobj, 't', numperiod)` and `output = tsmovavg(vector, 't', numperiod, dim)` compute the triangular moving average. The triangular moving average double-smooths the data. `tsmovavg` calculates the first simple moving average with window width of `ceil(numperiod + 1)/2`. Then it calculates a second simple moving average on the first moving average with the same window size.

`output = tsmovavg(tsobj, 'w', weights)` and `output = tsmovavg(vector, 'w', weights, dim)` calculate the weighted moving average by supplying weights for each element in the moving window. The length of the weight vector determines the size of the window. If larger weight factors are used for more recent prices and smaller factors for previous prices, the trend is more responsive to recent changes.

`output = tsmovavg(tsobj, 'm', numperiod)` and `output = tsmovavg(vector, 'm', numperiod, dim)` calculate the modified moving average. The modified moving average is similar to the simple moving average. Consider the argument `numperiod` to be the lag of the simple moving average. The first modified moving average is calculated like a simple moving average. Subsequent values are calculated by adding the new price and subtracting the last average from the resulting sum.

See Also

`mean`, `peravg`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 184-192.

typprice

Purpose

Typical price

Syntax

```
tprc = typprice(highp, lowp, closep)
tprc = typprice([highp lowp closep])
tprcts = typprice(tsobj)
tprcts = typprice(tsobj, ParameterName, ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`closep`	Closing price (vector)
`tsobj`	Financial time series object

Description

`tprc = typprice(highp, lowp, closep)` calculates the typical prices `tprc` from the high (`highp`), low (`lowp`), and closing (`closep`) prices. The typical price is the average of the high, low, and closing prices for each period.

`tprc = typprice([highp lowp closep])` accepts a three-column matrix as the input rather than two individual vectors. The columns of the matrix represent the high, low, and closing prices, in that order.

`tprcts = typprice(tsobj)` calculates the typical prices from the stock data contained in the financial time series object `tsobj`. The object must contain, at least, the `High`, `Low`, and `Close` data series. The typical price is the average of the closing price plus the high and low prices. `tprcts` is a financial time series object of the same dates as `tsobj` containing the data series `TypPrice`.

`tprcts = typprice(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

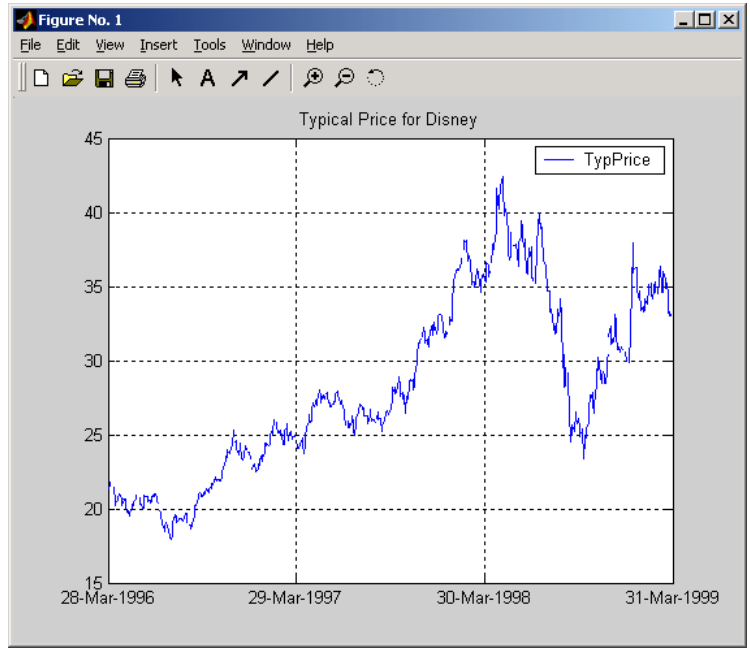
- `HighName`: high prices series name
- `LowName`: low prices series name
- `CloseName`: closing prices series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the typical price for Disney stock and plot the results:

```
load disney.mat
dis_Typ = typprice(dis);
plot(dis_Typ)
title('Typical Price for Disney')
```



See Also

`medprice`, `wclose`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 291 - 292.

uminus

Purpose Unary minus of financial time series object

Syntax `uminus`

Description `uminus` implements unary minus for a financial time series object.

See Also `uplus`

Purpose

Unary plus of financial time series object

Syntax

```
uplus
```

Description

`uplus` implements unary plus for a financial time series object.

See Also

`uminus`

vertcat

Purpose

Concatenate financial time series objects vertically

Description

`vertcat` implements vertical concatenation of financial time series objects. `vertcat` essentially adds data points to a time series object. Objects to be vertically concatenated must not have any duplicate dates and/or times or any overlapping dates and/or times. The description fields are concatenated as well. They are separated by `||`.

Examples

Create two financial time series objects with daily frequencies:

```
myfts   = fints((today:today+4)', (1:5)', 'DataSeries', 'd');
yourfts = fints((today+5:today+9)', (11:15)', 'DataSeries', 'd');
```

Use `vertcat` to concatenate them vertically:

```
newfts1 = [myfts; yourfts]

newfts1 =

    desc:   ||
    freq:  Daily (1)

    'dates:  (10)'    'DataSeries:  (10)'
    '11-Dec-2001'     [                1]
    '12-Dec-2001'     [                2]
    '13-Dec-2001'     [                3]
    '14-Dec-2001'     [                4]
    '15-Dec-2001'     [                5]
    '16-Dec-2001'     [               11]
    '17-Dec-2001'     [               12]
    '18-Dec-2001'     [               13]
    '19-Dec-2001'     [               14]
    '20-Dec-2001'     [               15]
```

Create two financial time series objects with different frequencies:

```
myfts   = fints((today:today+4)', (1:5)', 'DataSeries', 'd');
hisfts  = fints((today+5:7:today+34)', (11:15)', 'DataSeries',...
'w');
```

Concatenate these two objects vertically:

```
newfts2 = [myfts; hisfts]

 newfts2 =

    desc:   ||
    freq:  Unknown (0)

    'dates:  (10)'    'DataSeries:  (10)'
    '11-Dec-2001'     [                1]
    '12-Dec-2001'     [                2]
    '13-Dec-2001'     [                3]
    '14-Dec-2001'     [                4]
    '15-Dec-2001'     [                5]
    '16-Dec-2001'     [               11]
    '23-Dec-2001'     [               12]
    '30-Dec-2001'     [               13]
    '06-Jan-2002'     [               14]
    '13-Jan-2002'     [               15]
```

If all frequency indicators are the same, the new object has the same frequency indicator. However, if one of the concatenated objects has a different `freq` from the other(s), the frequency of the resulting object is set to `Unknown (0)`. In these examples, `newfts1` has `Daily` frequency, while `newfts2` has `Unknown (0)` frequency.

See Also

`horzcat`

volroc

Purpose

Volume rate of change

Syntax

```
vroc = volroc(tvolume nperiods)
vrocts = volroc(tsobj, nperiods)
vrocts = volroc(tsobj, nperiods, ParameterName, ParameterValue)
```

Arguments

`tvolume`	Volume traded
`nperiods`	(Optional) Period difference. Default = 12.
`tsobj`	Financial time series object

Description

`vroc = volroc(tvolume nperiods)` calculates the volume rate of change, `vroc`, from the volume traded data `tvolume`. If `nperiods` is specified, the volume rate of change is calculated between the current volume and the volume `nperiods` ago.

`vrocts = volroc(tsobj, nperiods)` calculates the volume rate of change, `vrocts`, from the financial time series object `tsobj`. The `vrocts` output is a financial time series object with similar dates as `tsobj` and a data series named `VolumeROC`. If `nperiods` is specified, the volume rate of change is calculated between the current volume and the volume `nperiods` ago.

`vrocts = volroc(tsobj, nperiods, ParameterName, ParameterValue)` specifies the name for the required data series when it is different from the default name. The valid parameter name is

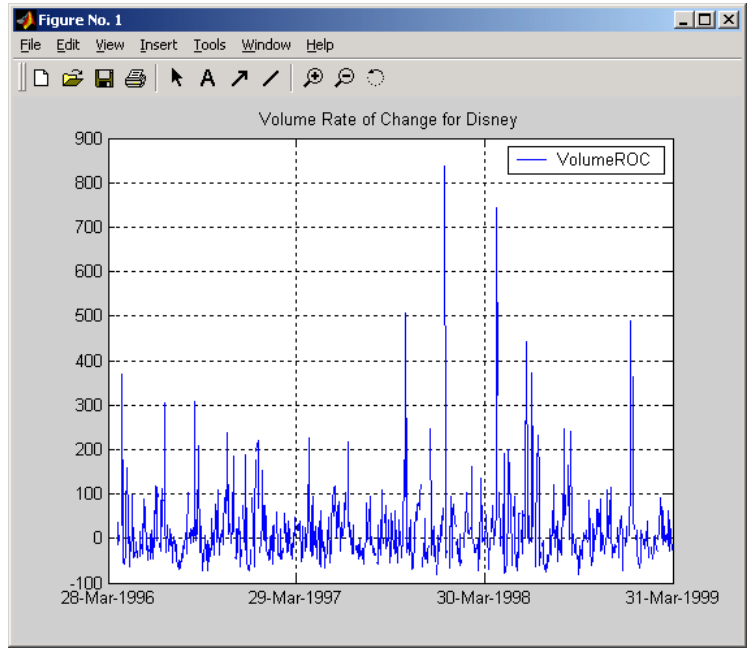
- `VolumeName`: volume traded series name

The parameter value is a string that represents the valid parameter name.

Examples

Compute the volume rate of change for Disney stock and plot the results:

```
load disney.mat
dis_VolRoc = volroc(dis)
plot(dis_VolRoc)
title('Volume Rate of Change for Disney')
```



See Also

`prcroc`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 310 - 311.

wclose

Purpose

Weighted close

Syntax

```
wcls = wclose(highp, lowp, closep)
wcls = wclose([highp lowp closep])
wclsts = wclose(tsobj)
wclsts = wclose(tsobj, ParameterName, ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`closep`	Closing price (vector)
`tsobj`	Financial time series object

Description

The weighted close price is the average of twice the closing price plus the high and low prices.

`wcls = wclose(highp, lowp, closep)` calculates the weighted close prices `wcls` based on the high (`highp`), low (`lowp`), and closing (`closep`) prices per period.

`wcls = wclose([highp lowp closep])` accepts a three-column matrix consisting of the high, low, and closing prices, in that order.

`wclsts = wclose(tsobj)` computes the weighted close prices for a set of stock price data contained in the financial time series object `tsobj`. The object must contain the high, low, and closing prices needed for this function. The function assumes that the series are named `High`, `Low`, and `Close`. All three are required. `wclsts` is a financial time series object of the same dates as `tsobj` and contains the data series named `WClose`.

`wclsts = wclose(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

- `HighName`: high prices series name
- `LowName`: low prices series name

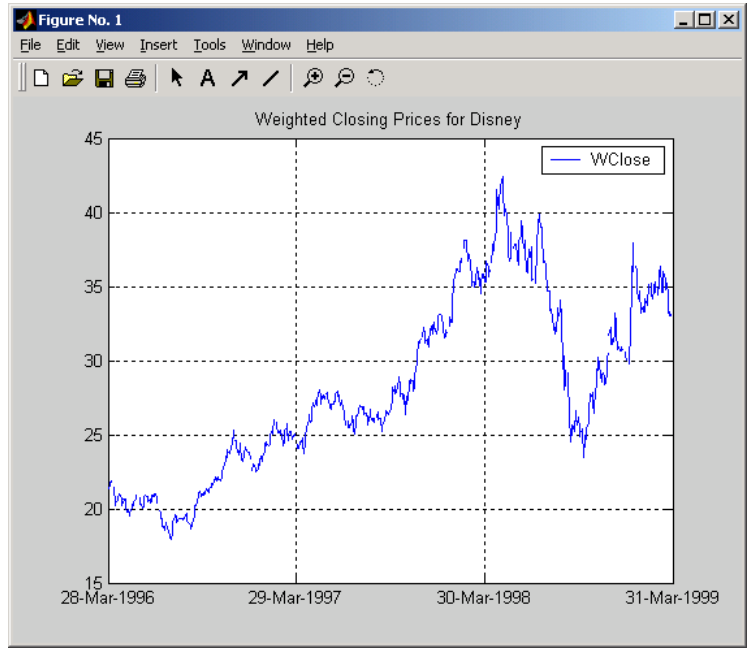
- `CloseName`: closing prices series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the weighted closing prices for Disney stock and plot the results:

```
load disney.mat
dis_Wclose = wclose(dis)
plot(dis_Wclose)
title('Weighted Closing Prices for Disney')
```



See Also

`medprice`, `typprice`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 312 - 313.

willad

Purpose

Williams Accumulation/Distribution line

Syntax

```
wadl = willad(highp, lowp, closep)
wadl = willad([highp lowp closep])
wadlts = willad(tsobj)
wadlts = willad(tsobj, ParameterName, ParameterValue, ...)
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`closep`	Closing price (vector)
`tsobj`	Time series object

Description

`wadl = willad(highp, lowp, closep)` computes the Williams Accumulation/Distribution line for a set of stock price data. The prices needed for this function are the high (`highp`), low (`lowp`), and closing (`closep`) prices. All three are required.

`wadl = willad([highp lowp closep])` accepts a three-column matrix of prices as input. The first column contains the high prices, the second contains the low prices, and the third contains the closing prices.

`wadlts = willad(tsobj)` computes the Williams Accumulation/Distribution line for a set of stock price data contained in the financial time series object `tsobj`. The object must contain the high, low, and closing prices needed for this function. The function assumes that the series are named `High`, `Low`, and `Close`. All three are required. `wadlts` is a financial time series object with the same dates as `tsobj` and a single data series named `WillAD`.

`wadlts = willad(tsobj, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

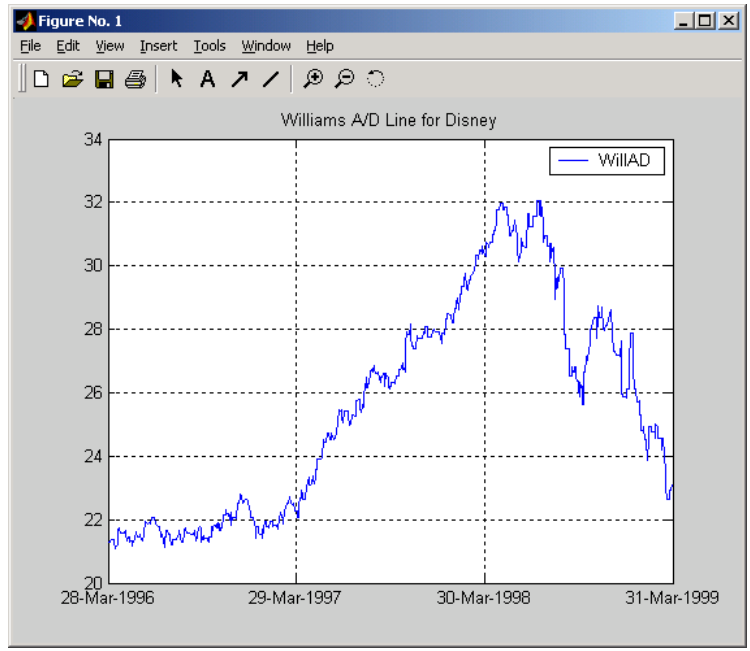
- `HighName`: high prices series name
- `LowName`: low prices series name
- `CloseName`: closing prices series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the Williams A/D line for Disney stock and plot the results:

```
load disney.mat
dis_Willad = willad(dis)
plot(dis_Willad)
title('Williams A/D Line for Disney')
```



See Also

`adline`, `adosc`, `willpctr`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 314 - 315.

willpctr

Purpose

Williams %R

Syntax

```
wpctr = willpctr(highp, lowp, closep, nperiods)
wpctr = willpctr([highp, lowp, closep], nperiods)
wpctrts = willpctr(tsobj)
wpctrts = willpctr(tsobj, nperiods)
wpctrts = willpctr(tsobj, nperiods, ParameterName, ParameterValue,
   ... )
```

Arguments

`highp`	High price (vector)
`lowp`	Low price (vector)
`closep`	Closing price (vector)
`nperiods`	Number of periods (scalar). Default = 14.
`tsobj`	Financial time series object

Description

`wpctr = willpctr(highp, lowp, closep, nperiods)` calculates the Williams %R values for the given set of stock prices for a specified number of periods `nperiods`. The stock prices needed are the high (`highp`), low (`lowp`), and closing (`closep`) prices. `wpctr` is a vector that represents the Williams %R values from the stock data.

`wpctr = willpctr([highp, lowp, closep], nperiods)` accepts the price input as a three-column matrix representing the high, low, and closing prices, in that order.

`wpctrts = willpctr(tsobj)` calculates the Williams %R values for the financial time series object `tsobj`. The object must contain at least three data series named `High` (high prices), `Low` (low prices), and `Close` (closing prices). `wpctrts` is a financial time series object with the same dates as `tsobj` and a single data series named `WillPctR`.

`wpctrts = willpctr(tsobj, nperiods)` calculates the Williams %R values for the financial time series object `tsobj` for `nperiods` periods.

`wpctrts = willpctr(tsobj, nperiods, ParameterName, ParameterValue, ...)` accepts parameter name/parameter value pairs as input. These pairs specify the name(s) for the required data series if it is different from the expected default name(s). Valid parameter names are

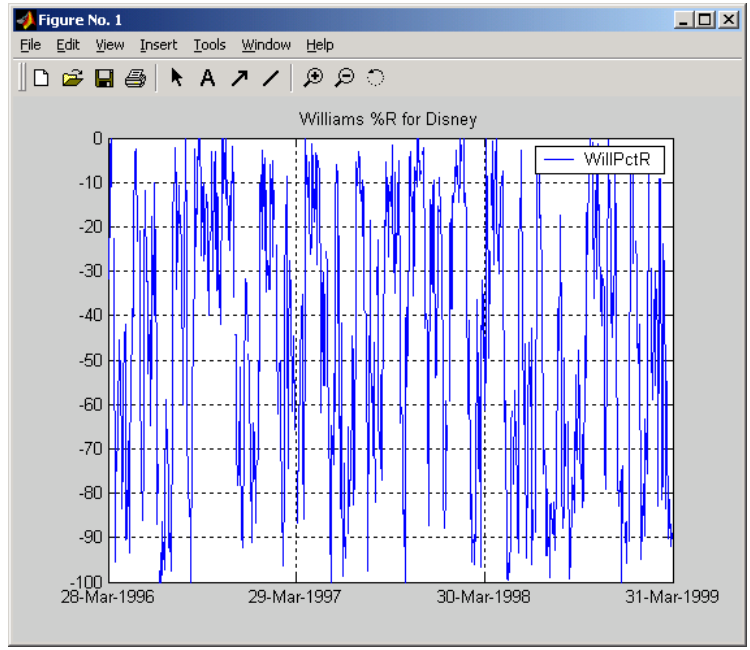
- `HighName`: high prices series name
- `LowName`: low prices series name
- `CloseName`: closing prices series name

Parameter values are the strings that represent the valid parameter names.

Examples

Compute the Williams %R values for Disney stock and plot the results:

```
load disney.mat
dis_Wpctr = willpctr(dis)
plot(dis_Wpctr)
title('Williams %R for Disney')
```



See Also

`stochosc`, `willad`

Reference

Achelis, Steven B., *Technical Analysis from A To Z*, Second printing, McGraw-Hill, 1995, pp. 316 - 317.

Index

G

H

I

L

M

V

W

Z