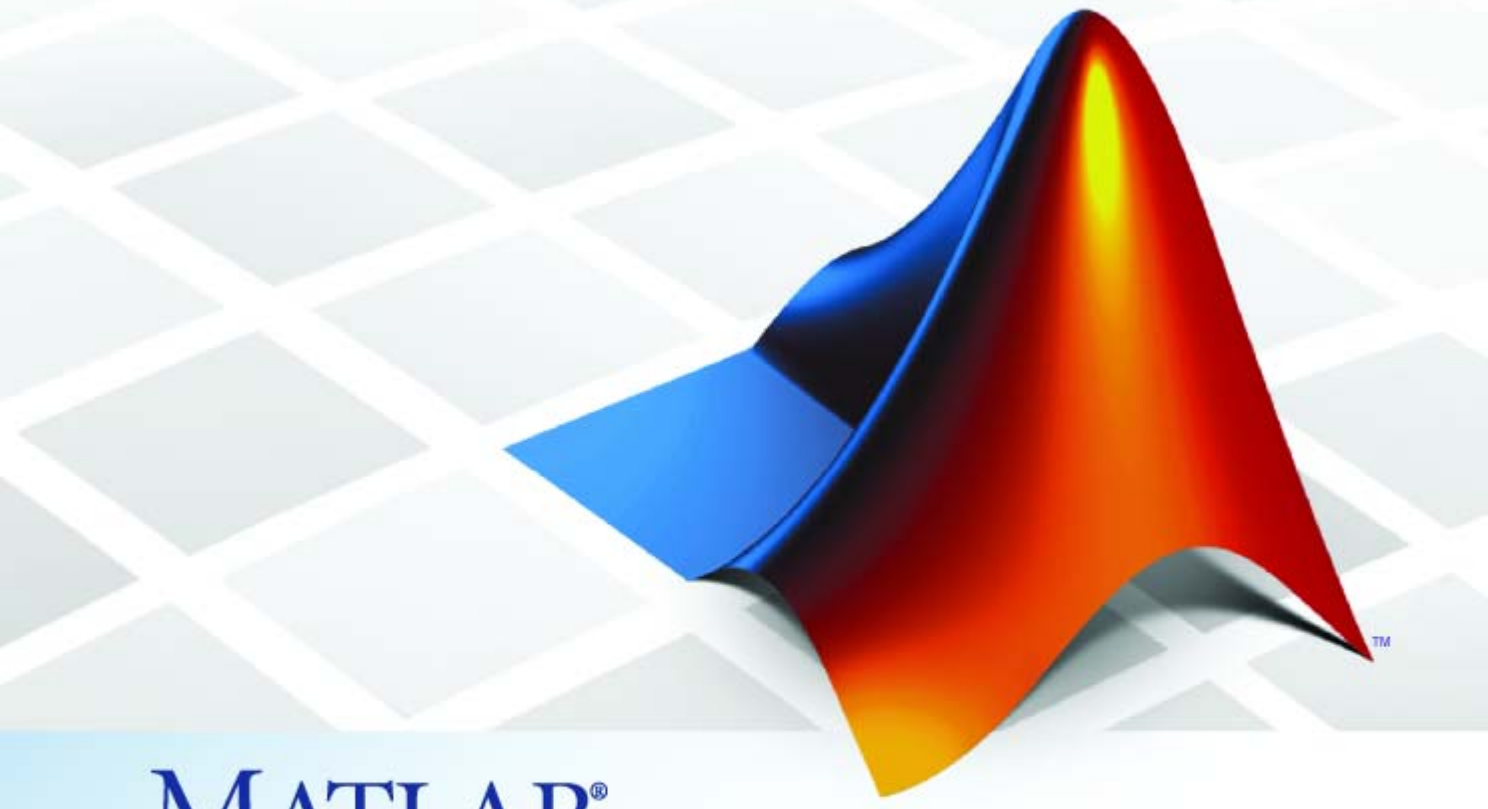
MATLAB® 7
Data Analysis



MATLAB®

The MathWorks™
Accelerating the pace of engineering and science

How to Contact The MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support

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

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

MATLAB® Data Analysis

© COPYRIGHT 2005–2009 by The MathWorks, Inc.

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

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

Trademarks

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

Patents

The MathWorks products are protected by one or more U.S. patents. Please see `www.mathworks.com/patents` for more information.

Revision History

September 2005	Online only	New for MATLAB 7.1 (Release 14SP3)
March 2006	Online only	Revised for Version 7.2 (Release 2006a)
September 2006	Online only	Revised for Version 7.3 (Release 2006b)
March 2007	Online only	Revised for Version 7.4 (Release 2007a)
September 2007	Online only	Revised for Version 7.5 (Release 2007b)
March 2008	Online only	Revised for Version 7.6 (Release 2008a)
October 2008	Online only	Revised for Version 7.7 (Release 2008b)
March 2009	Online Only	Revised for MATLAB 7.8 (Release 2009a)
September 2009	Online Only	Revised for MATLAB 7.9 (Release 2009b)

Contents

1 Data Processing

Importing and Exporting Data 1-2
Importing Data to the Workspace 1-2
Exporting Data from the Workspace 1-3

Plotting Data 1-4
Introduction 1-4
Example: Loading and Plotting Data 1-4

Missing Data 1-7
Representing Missing Data Values 1-7
Calculating with NaNs 1-7
Removing NaNs from Data 1-8
Interpolating Missing Data 1-9

Inconsistent Data 1-10

Filtering Data 1-12
Introduction 1-12
Filter Function 1-12
Example: Moving Average Filter 1-13
Example: Discrete Filter 1-14

Detrending Data 1-18
Introduction 1-18
Example: Removing Linear Trends from Data 1-18

Differencing Data 1-22

Descriptive Statistics 1-23
Functions for Calculating Descriptive Statistics 1-23
Example: Using MATLAB Data Statistics 1-26

2 Interactive Data Exploration

What Is Interactive Data Exploration? **2-2**
Interacting with MATLAB Data Graphs **2-2**

Marking Up Graphs with Data Brushing **2-4**
What Is Data Brushing? **2-4**
How to Brush Data **2-6**
Effects of Brushing on Data **2-8**
Other Data Brushing Aspects **2-11**

Making Graphs Responsive with Data Linking **2-13**
What Is Data Linking? **2-13**
Why Use Linked Plots? **2-14**
How to Link Plots **2-14**
How Linked Plots Behave **2-16**
Linking vs. Refreshing Plots **2-19**
Using Linked Plot Controls **2-21**

Interacting with Graphed Data **2-24**
Data Brushing with the Variable Editor **2-24**
Using Data Tips to Explore Graphs **2-25**
Example — Visually Exploring Demographic Statistics . . **2-26**

3 Regression Analysis

Linear Correlation **3-2**
Introduction **3-2**
Covariance **3-2**
Correlation Coefficients **3-5**

Linear Regression **3-7**
Introduction **3-7**
Residuals and Goodness of Fit **3-8**
Fitting Data with Curve Fitting Toolbox Functions **3-8**

Interactive Fitting **3-10**
The Basic Fitting GUI **3-10**
Preparing for Basic Fitting **3-10**
Opening the Basic Fitting GUI **3-11**
Example: Using Basic Fitting GUI **3-12**

Programmatic Fitting **3-23**
MATLAB Functions for Polynomial Models **3-23**
Linear Model with Nonpolynomial Terms **3-27**
Multiple Regression **3-29**
Example: Programmatic Fitting **3-30**

4 Time Series Analysis

Introduction **4-2**

Time Series Objects **4-3**
Introduction **4-3**
Time Series Data Sample **4-4**
Example: Time Series Objects and Methods **4-6**
Time Series Constructor **4-21**
Time Series Methods **4-29**
Time Series Collection Constructor **4-33**
Time Series Collection Methods **4-37**

Time Series Tools **4-39**
Introduction **4-39**
Importing and Exporting Data **4-44**
Plotting Time Series **4-50**
Selecting Data for Analysis **4-64**
Editing Data, Time, Attributes, and Events **4-67**
Processing and Manipulating Time Series **4-76**
Example: Time Series Tools **4-77**

Index

1

Data Processing

- "Importing and Exporting Data" on page 1-2
- "Plotting Data" on page 1-4
- "Missing Data" on page 1-7
- "Inconsistent Data" on page 1-10
- "Filtering Data" on page 1-12
- "Detrending Data" on page 1-18
- "Differencing Data" on page 1-22
- "Descriptive Statistics" on page 1-23

Importing and Exporting Data

Importing Data to the Workspace

The first step in analyzing data is to import it into the MATLAB® workspace. See the topics under "Importing Data" for detailed information about supported data formats and the functions you can use.

The easiest way to import data is to use the MATLAB Import Wizard. With the Import Wizard, you can create variables from the following types of data sources:

- Text files, such as `.txt` and `.dat`
- MAT-files
- Scientific data in CDF or HDF4 formats
- Spreadsheets, such as Microsoft® Excel® `.xls` files
- Graphics files, such as `.gif` and `.jpg`
- Audio and video files, such as `.avi` and `.wav`

The MATLAB Import Wizard processes the data source and recognizes data delimiters, as well as row or column headers, to facilitate the process of data selection. It also lets you choose which worksheets within an Excel workbook to import. Use the Import Wizard or `xlsfinfo` to identify names of worksheets contained in a workbook.

You can analyze data stored in scientific data formats, such as CDF, netCDF, FITS, HDF4 and HDF5. Import CDF files with `cdfread`, HDF4 files with `hdfread` and with the `hdftool` GUI, and HDF5 files with `hdf5read`. Use informational functions, such as `cdfinfo`, `hdfinfo`, and `hdf5info`, to determine what data these files contain before importing variables from them. For more information on using these and other scientific data formats, see "Importing Scientific Data Files" and the complete list of supported `file formats`.

You can also import time series data and generate time vectors for it. When you work with time series data, use the Time Series Tools GUI (`tstool`) to import the data and create `timeseries` objects. The Time Series Tools Import

Wizard makes it easy to import or define a time vector for your data. For more information, see “Time Series Tools” on page 4-39.

Exporting Data from the Workspace

When you analyze your data, you might create new variables or modified imported variables. You can export variables from the MATLAB workspace to a variety of file formats, both character-based and binary. You can, for example, create HDF and Excel files containing your data. File formats you can export to include MAT-files, text files, spreadsheets, and several types of scientific data files. For details, see the respective topics listed in “Exporting Data” and the complete list of supported `file formats`.

Plotting Data

In this section...

"Introduction" on page 1-4

"Example: Loading and Plotting Data" on page 1-4

Introduction

After you import data into the MATLAB workspace, it is a good idea to plot the data so that you can explore its features. An exploratory plot of your data enables you to identify discontinuities and potential outliers, as well as the regions of interest.

The "Plots and Plotting Tools" section of the MATLAB Graphics documentation fully describes the MATLAB figure window, which displays the plot, and the types of graphs you can create in figure windows. It also discusses the various interactive tools available for editing and customizingMATLAB graphics.

If you are working with time series data, see "Time Series Tools" on page 4-39 for detailed information about working with time series plots.

Example: Loading and Plotting Data

In this example, you perform the following tasks on the data in a space-delimited text file:

- "Loading the Data" on page 1-4
- "Plotting the Data" on page 1-5

This example uses sample data in `count.dat` that consists of three sets of hourly traffic counts, recorded at three different town intersections over a 24-hour period. Each data column in the file represents data for one intersection.

Loading the Data

Import data into the workspace using the `load` function:

```
load count.dat
```

Loading this data creates a 24-by-3 matrix called `count` in the MATLAB workspace.

You can get the size of the data matrix by

```
[n,p] = size(count)
n =
    24
p =
     3
```

where `n` represents the number of rows, and `p` represents the number of columns.

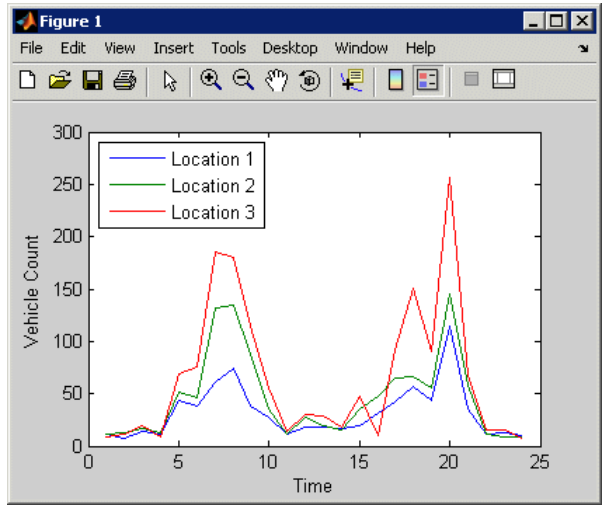
Plotting the Data

Create a time vector, `t`, containing integers from `1` to `n`:

```
t = 1:n;
```

Use the following commands to plot the data as a function of time, and to annotate the plot:

```
plot(t,count),
legend('Location 1','Location 2','Location 3',2)
xlabel('Time'), ylabel('Vehicle Count')
```



Traffic Counts at Three Intersections

Missing Data

In this section...
"Representing Missing Data Values" on page 1-7
"Calculating with NaNs" on page 1-7
"Removing NaNs from Data" on page 1-8
"Interpolating Missing Data" on page 1-9

Representing Missing Data Values

Users of MATLAB software often represent missing or unavailable data values by the special value `NaN`, which stands for *Not-a-Number*.

The IEEE® floating-point arithmetic convention defines `NaN` as the result of an undefined operation, such as 0/0.

Calculating with NaNs

When you perform calculations on a IEEE variable that contains `NaN`s, the `NaN` values are propagated to the final result. This might render the result useless.

For example, consider a matrix containing the 3-by-3 magic square with its center element replaced with `NaN`:

```
a = magic(3); a(2,2) = NaN

a =
     8     1     6
     3   NaN     7
     4     9     2
```

Compute the sum for each column in the matrix:

```
sum(a)

ans =
    15   NaN    15
```

Notice that the sum of the elements in the middle column is a NaN value because that column contains a NaN.

If you do not want to have NaNs in your final results, you must remove these values from your data. For more information, see "Removing NaNs from Data" on page 1-8.

Removing NaNs from Data

You can use the IEEE function isnan to identify NaNs in the data, and then remove them using the techniques in the following table.

Note You must use the function isnan to identify NaNs because, by IEEE arithmetic convention, the logical comparison NaN == NaN always produces 0 (i.e., it never evaluates to true). Therefore, you *cannot* use x(x==NaN) = [] to remove NaNs from your data.

Code	Description
`i = find(~isnan(x));` `x = x(i)`	Find the indices of elements in a vector x that are not NaNs. Keep only the non-NaN elements.
`x = x(~isnan(x));`	Remove NaNs from a vector x.
`x(isnan(x)) = [];`	Remove NaNs from a vector x (alternative method).
`X(any(isnan(X),2),:)  = [];`	Remove any rows containing NaNs from a matrix X.

If you frequently need to remove NaNs, you might want to write a short M-file function that you can call:

```
function X = exciseRows(X)
X(any(isnan(X),2),:) = [];
```

The following command computes the correlation coefficients of `X` after all rows containing `NaNs` are removed:

```
C = corrcoef(excise(X));
```

For more information about correlation coefficients, see “Linear Correlation” on page 3-2.

Interpolating Missing Data

You can use interpolation to find intermediate points in your data. The simplest function for performing interpolation is `interp1`, which is a 1-D interpolation function.

By default, the interpolation method is `'linear'`, which fits a straight line between a pair of existing data points to calculate the intermediate value. The complete set of available methods, which you can specify as arguments in the `interp1` function, includes the following:

- `'nearest'` — Nearest neighbor interpolation
- `'linear'` — Linear interpolation
- `'spline'` — Piecewise cubic spline interpolation
- `'pchip'` or `'cubic'` — Shape-preserving piecewise cubic interpolation
- `'v5cubic'` — Cubic interpolation from IEEE Version 5, which does not use `'extrapolate'` and uses `'spline'` when `X` is not equally spaced

For more information about `interp1`, see the IEEE documentation or type at the IEEE prompt

```
help interp1
```

Inconsistent Data

When you examine a data plot, you might find that some points appear to dramatically differ from the rest of the data. In some cases, it is reasonable to consider such points *outliers*, or data values that do not appear to be consistent with the rest of the data.

The following example illustrates how to remove outliers from three data sets in the 24-by-3 matrix `count`. In this case, an outlier is defined as a value that is more than three standard deviations away from the mean.

Caution Be cautious about changing data unless you are confident that you understand the source of the problem you want to correct. Removing an outlier has a greater effect on the standard deviation than on the mean of the data. Deleting one such point leads to a smaller new standard deviation, which might result in making some remaining points appear to be outliers!

```
% Import the sample data
load count.dat;
% Calculate the mean and the standard deviation
% of each data column in the matrix
mu = mean(count)
sigma = std(count)
```

The Command Window displays

```
mu =
       32.0000   46.5417   65.5833

sigma =
       25.3703   41.4057   68.0281
```

When an *outlier* is considered to be more than three standard deviations away from the mean, you can use the following syntax to determine the number of outliers in each column of the `count` matrix:

```
[n,p] = size(count);
% Create a matrix of mean values by
% replicating the mu vector for n rows
MeanMat = repmat(mu,n,1);
% Create a matrix of standard deviation values by
% replicating the sigma vector for n rows
SigmaMat = repmat(sigma,n,1);
% Create a matrix of zeros and ones, where ones indicate
% the location of outliers
outliers = abs(count - MeanMat) > 3*SigmaMat;
% Calculate the number of outliers in each column
nout = sum(outliers)
```

The procedure returns the following number of outliers in each column:

```
nout =
       1     0     0
```

There is one outlier in the first data column of `count` and none in the other two columns.

To remove an entire row of data containing the outlier, type

```
count(any(outliers,2),:) = [];
```

Here, `any(outliers,2)` returns a `1` when any of the elements in the `outliers` vector is a nonzero number, and the argument `2` specifies that `any` works down the second dimension of the count matrix—its columns.

Filtering Data

In this section...

"Introduction" on page 1-12

"Filter Function" on page 1-12

"Example: Moving Average Filter" on page 1-13

"Example: Discrete Filter" on page 1-14

Introduction

A variety of MATLAB IEEE functions help you work with difference equations and filters to shape the variations in the raw data. These functions operate on both vectors and matrices. You can filter data to smooth out high-frequency fluctuations or remove periodic trends of a specific frequency.

A vector input represents a single, sampled data signal (or *sequence*). For a matrix input, each signal corresponds to a column in the matrix and each data sample is a row.

Filter Function

The function

```
y = filter(b,a,x)
```

creates filtered data `y` by processing the data in vector `x` with the filter described by vectors `a` and `b`.

The `filter` function is a general tapped delay-line filter, described by the difference equation

$$\begin{aligned} a(1)y(n) = b(1)x(n) + b(2)x(n-1) + \ldots + b(N_b)x(n-N_b+1) \\ - a(2)y(n-1) - \ldots - a(N_a)y(n-N_a+1) \end{aligned}$$

Here, n is the index of the current sample, N_a is the order of the polynomial described by vector a, and N_b is the order of the polynomial described by

vector b. The output $y(n)$ is a linear combination of current and previous inputs, $x(n)$ $x(n-1)$..., and previous outputs, $y(n-1)$ $y(n-2)$... .

Example: Moving Average Filter

You can smooth the data in `count.dat` using a moving-average filter to see the average traffic flow over a 4-hour window (covering the current hour and the previous 3 hours). This is represented by the following difference equation:

$$y(n) = \frac{1}{4}x(n) + \frac{1}{4}x(n-1) + \frac{1}{4}x(n-2) + \frac{1}{4}x(n-3)$$

The corresponding vectors are

```
a = 1;
b = [1/4 1/4 1/4 1/4];
```

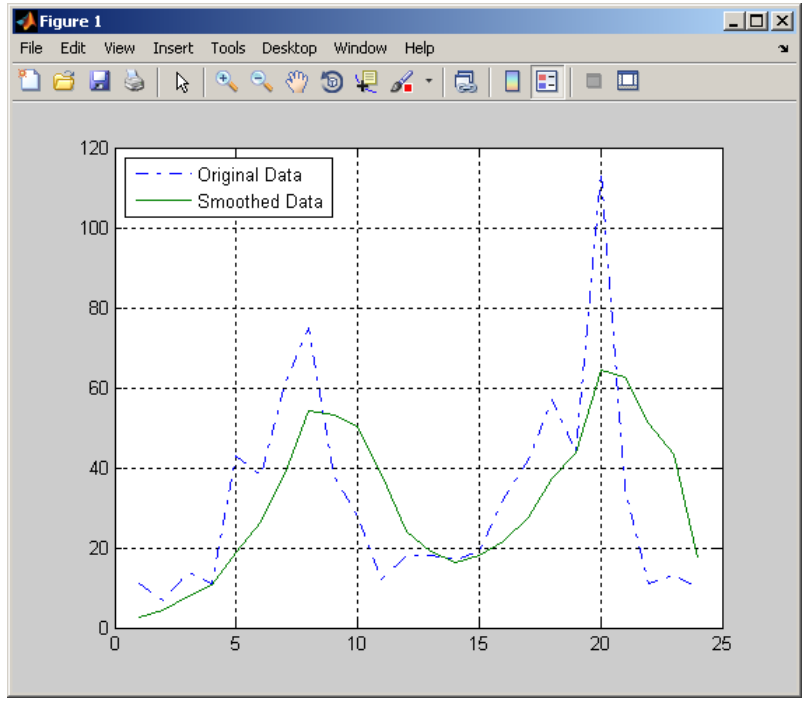
1 Extract the first column of `count` and assign it to the vector `x`:

```
x = count(:,1);
```

2 The 4-hour moving average of the data is calculated by

```
y = filter(b,a,x);
```

3 The filtered data, represented by the solid line in the plot, is the 4-hour moving average of the `count` data. The original data is represented by the dashed line.



Plot of Original and Smoothed Data

Example: Discrete Filter

You use the discrete filter to shape the data by applying a transfer function to the input signal.

Depending on your objectives, the transfer function you choose might alter both the amplitude and the phase of the variations in the data at different frequencies to produce either a smoother or a rougher output.

Taking the *z*-transform of the following difference equation

$$a(1)y(n) = b(1)x(n) + b(2)x(n-1) + \ldots + b(N_b)x(n-N_b+1) - a(2)y(n-1) - \ldots - a(N_a)y(n-N_a+1)$$

results in the following transfer function:

$$Y(z) = H(z^{-1})X(z) = \frac{b(1) + b(2)z^{-1} + \ldots b(N_b)z^{-N_b+1}}{a(1) + a(2)z^{-1} + \ldots a(N_a)z^{-N_a+1}}X(z)$$

Here $Y(z)$ is the z-transform of the filtered output $y(n)$. The coefficients b and a are unchanged by the z-transform.

In digital signal processing (DSP), it is customary to write transfer functions as rational expressions in z^{-1} and to order the numerator and denominator terms in ascending powers of z^{-1}.

Consider the following transfer function:

$$H(z^{-1}) = \frac{b(z^{-1})}{a(z^{-1})} = \frac{2 + 3z^{-1}}{1 + 0.2z^{-1}}$$

To apply this transfer function to the data in `count.dat`:

1 Load the matrix count into the workspace:

```
load count.dat;
```

2 Extract the first column and assign it to `x`:

```
x = count(:,1);
```

3 Enter the coefficients of the denominator ordered in ascending powers of z^{-1} to represent $1 + 0.2z^{-1}$:

```
a = [1 0.2];
```

4 Enter the coefficients of the numerator to represent $2+3z^{-1}$:

```
b = [2 3];
```

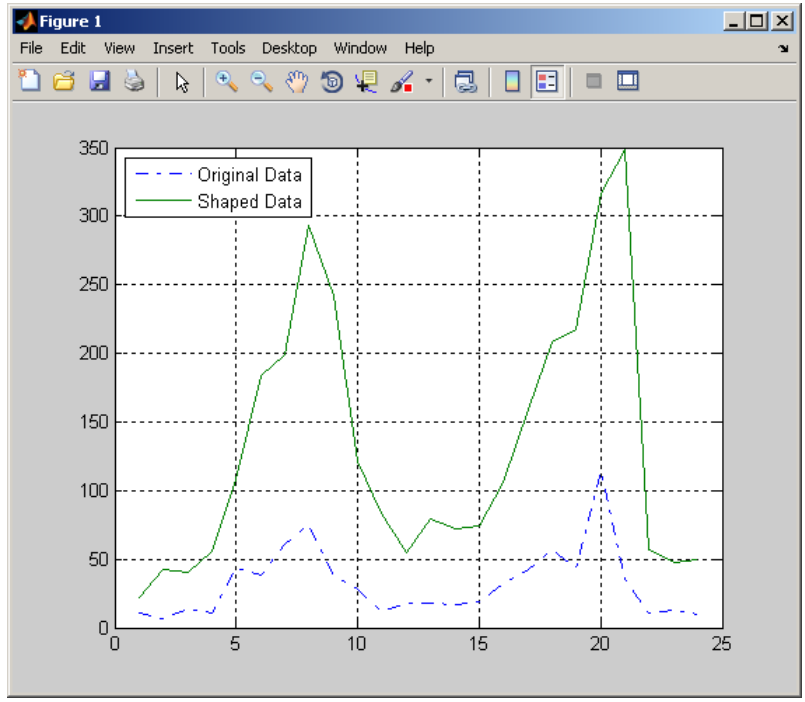
5 Call the filter function:

```
y = filter(b,a,x);
```

6 Compare the original data and the shaped data with an overlaid plot of the two curves:

```
t = 1:length(x);
plot(t,x,'-.',t,y,'-'), grid on
legend('Original Data','Shaped Data',2)
```

As you can see from the plot, this filter primarily modifies the amplitude of the original data.



Plot of Original and Shaped Data

Detrending Data

In this section...

"Introduction" on page 1-18

"Example: Removing Linear Trends from Data" on page 1-18

Introduction

The MATLAB function `detrend` subtracts the mean or a best-fit line (in the least-squares sense) from your data. If your data contains several data columns, each data column is detrended separately.

Removing a trend from the data enables you to focus your analysis on the fluctuations in the data about the trend. A linear trend typically indicates a systematic increase or decrease in the data. This might be caused by sensor drift, for example.

You must decide whether it makes sense to remove trend effects in the data based on the objectives of your analysis.

Example: Removing Linear Trends from Data

This example shows how to remove a linear trend from daily closing stock prices to emphasize the price fluctuations about the overall increase. This data is available in the Financial Toolbox™ `predict_ret_data.mat` file.

You can follow along with the steps in this example to perform the following tasks:

- "Loading and Plotting Data" on page 1-19
- "Detrending Data and Plotting Results" on page 1-20

Loading and Plotting Data

1. Load the sample data:

   ```
   load predict_ret_data.mat
   ```

   This adds the variable `sdata` to the workspace, which contains the daily stock prices.

2. View the contents of the column vector `sdata`:

   ```
   sdata
   ```

   The last data value is a `NaN`, which must be removed before detrending the data.

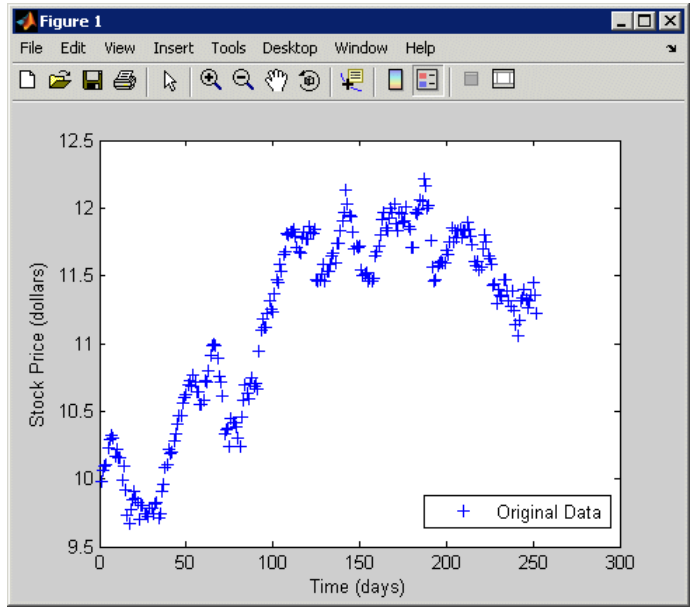
3. Identify and remove the `NaN` value from `sdata`:

   ```
   sdata(any(isnan(sdata),2),:) = []
   ```

   For more information about removing `NaN`s, see “Removing NaNs from Data” on page 1-8.

4. Plot the data:

   ```
   plot(sdata,'+')
   legend('Original Data',1);
   xlabel('Time (days)');
   ylabel('Stock Price (dollars)');
   ```



Daily Closing Stock Prices

Notice the systematic increase in the stock prices when this data was collected.

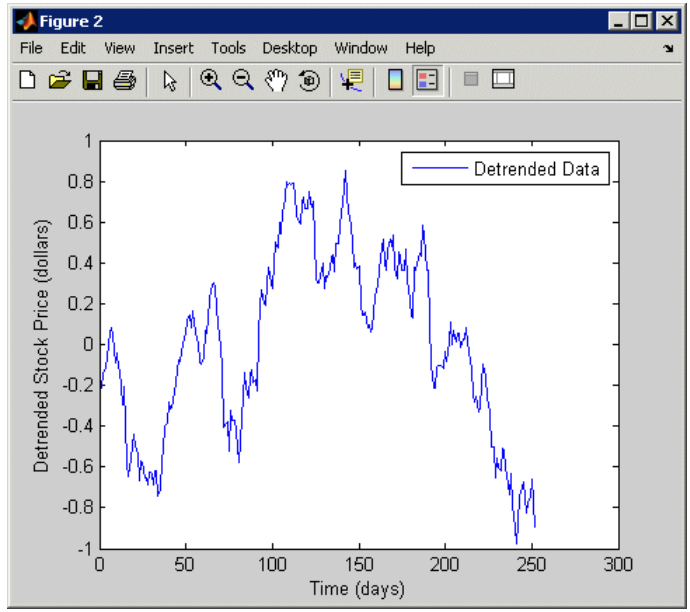
Detrending Data and Plotting Results

1. Remove a best-fit line (in the least-squares sense) from `sdata` and save the results to a new variable, `detrend_sdata`:

```
detrend_sdata=detrend(sdata);
```

2. Plot the detrended data in a new MATLAB Figure window:

```
figure
plot(detrend_sdata,'-')
legend('Detrended Data',2)
xlabel('Time (days)');
ylabel('Detrended Stock Price (dollars)');
```



Stock Prices with the Removed Linear Trend

Notice that the data is now centered about 0 and the linear drift is removed from the data.

Differencing Data

Three MATLAB functions perform finite difference calculations.

Function	Description
`del2`	Discrete Laplacian of a matrix
`diff`	Differences between successive elements of a vector; numerical partial derivatives of a vector
`gradient`	Numerical partial derivatives of a matrix

The `diff` function computes the difference between successive elements in a numeric vector. That is, `diff(X)` is `[X(2)-X(1) X(3)-X(2)...X(n)-X(n-1)]`. You might want to perform this operation on your data if you are more interested in analyzing the changes in the values, rather than the absolute values.

For a vector `A`,

```
A = [9 -2 3 0 1 5 4];
diff(A)

ans =
   -11     5    -3     1     4    -1
```

Besides computing the first difference, you can use `diff` to determine certain characteristics of vectors. For example, you can use `diff` to determine whether the vector values are monotonically increasing or decreasing, or whether a vector has equally spaced elements.

The following table provides examples for using `diff` with a vector `x`.

Test	Description
`any(diff(x)==0)`	Tests whether there are any repeated elements in X
`all(diff(x)>0)`	Tests whether the values are monotonically increasing
`all(diff(diff(x))==0)`	Tests for equally spaced vector elements

Descriptive Statistics

In this section...

"Functions for Calculating Descriptive Statistics" on page 1-23

"Example: Using MATLAB Data Statistics" on page 1-26

If you need more advanced statistics functionality, you might want to use the Statistics Toolbox™ software. For more information see the Statistics Toolbox documentation.

Functions for Calculating Descriptive Statistics

You can use the following MATLAB functions to calculate the descriptive statistics for your data.

Note For matrix data, descriptive statistics for each column are calculated independently.

Statistics Function Summary

Function	Description
`max`	Maximum value
`mean`	Average or mean value
`median`	Median value
`min`	Smallest value
`mode`	Most frequent value
`std`	Standard deviation
`var`	Variance, which measures the spread or dispersion of the values

The following examples apply MATLAB functions to calculate descriptive statistics:

- "Example 1 — Calculating Maximum, Mean, and Standard Deviation" on page 1-24
- "Example 2 — Subtracting the Mean" on page 1-26

Example 1 — Calculating Maximum, Mean, and Standard Deviation

This example shows how to use MATLAB functions to calculate the maximum, mean, and standard deviation values for a 24-by-3 matrix called `count`. MATLAB computes these statistics independently for each column in the matrix.

```
% Load the sample data
load count.dat
% Find the maximum value in each column
mx = max(count)
% Calculate the mean of each column
mu = mean(count)
% Calculate the standard deviation of each column
sigma = std(count)
```

The results are

```
mx =
          114          145          257

mu =
      32.0000      46.5417      65.5833

sigma =
      25.3703      41.4057      68.0281
```

To get the row numbers where the maximum data values occur in each data column, you can specify a second output parameter `indx` to return the row index. For example:

```
[mx,indx] = max(count)
```

These results are

```
mx =
      114     145     257

indx =
      20    20    20
```

Here, the variable `mx` is a row vector that contains the maximum value in each of the three data columns. The variable `indx` contains the row indices in each column that correspond to the maximum values.

To find the minimum value in the entire `count` matrix, you can reshape this 24-by-3 matrix into a 72-by-1 column vector by using the syntax `count(:)`. Then, to find the minimum value in the single column, you can use the following syntax:

```
min(count(:))

ans =
      7
```

Example 2 — Subtracting the Mean

You can subtract the mean from each column of the matrix by using the following syntax:

```
% Get the size of the count matrix
[n,p] = size(count)
% Compute the mean of each column
mu = mean(count)
% Create a matrix of mean values by
% replicating the mu vector for n rows
MeanMat = repmat(mu,n,1)
% Subtract the column mean from each element
% in that column
x = count - MeanMat
```

Note Subtracting the mean from the data is also called *detrending*. For more information about removing the mean or the best-fit line from the data, see "Detrending Data" on page 1-18.

Example: Using MATLAB Data Statistics

The Data Statistics dialog box helps you calculate and plot descriptive statistics with the data. This example shows how to use MATLAB Data Statistics to calculate and plot statistics for a 24-by-3 matrix, called `count`.

This section contains the following topics:

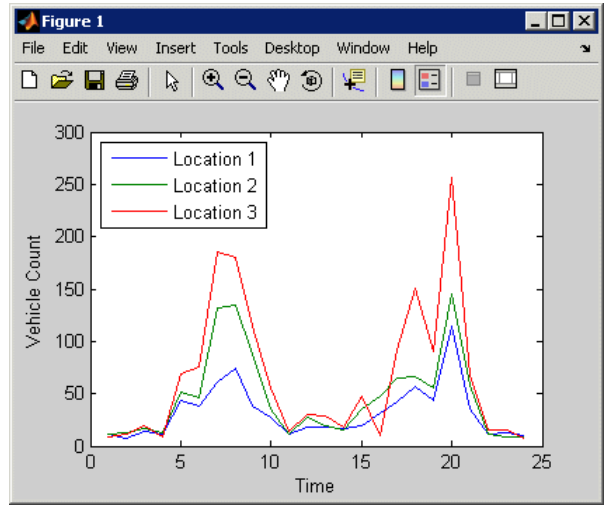
- "Calculating and Plotting Descriptive Statistics" on page 1-27
- "Formatting Data Statistics on Plots" on page 1-30
- "Saving Statistics to the MATLAB Workspace" on page 1-32
- "Generating an M-file" on page 1-33

Note MATLAB Data Statistics is available only for 2-D plots.

Calculating and Plotting Descriptive Statistics

1 Load and plot the data:

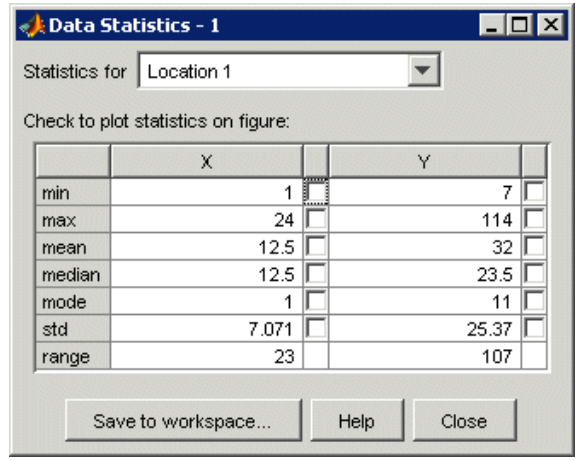
```
load count.dat
[n,p] = size(count);
% Define the x-values
t = 1:n;
% Plot the data and annotate the graph
plot(t,count)
legend('Location 1','Location 2','Location 3',2)
xlabel('Time'), ylabel('Vehicle Count')
```



Note The legend contains the name of each data set, as specified by the `legend` function: `Location 1`, `Location 2`, and `Location 3`. A *data set* refers to each column of data in the array you plotted. If you do not name the data sets, default names are assigned: `data 1`, `data 2`, and so on.

2 In the Figure window, select **Tools > Data Statistics** .

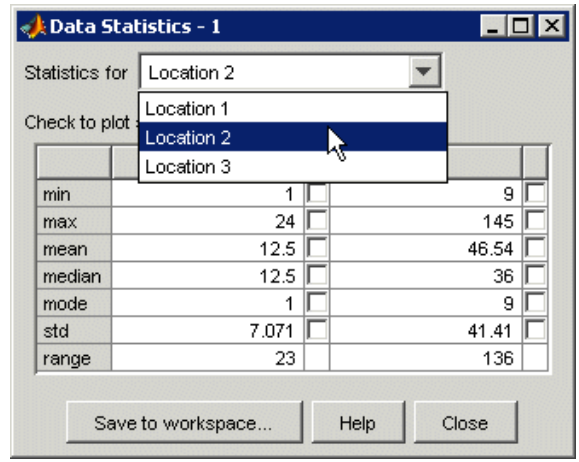
This opens the Data Statistics dialog box, which displays descriptive statistics for the `X`- and `Y`-data of the `Location 1` data set.



Note The Data Statistics GUI calculates the *range*, which is the difference between the minimum and maximum values in the selected data set. The Data Statistics GUI does not display the range on the plot.

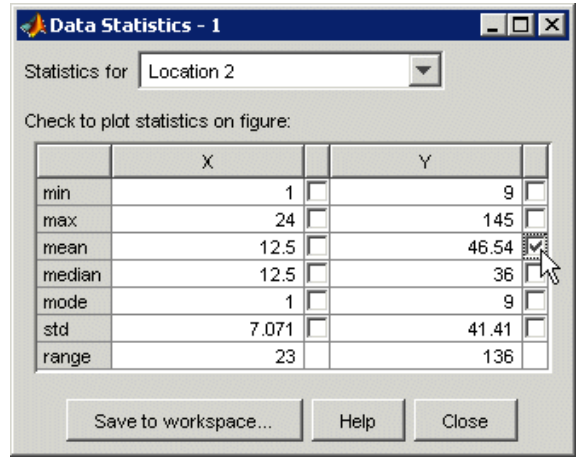
3 Select a different data set in the **Statistics for** list: `Location 2`.

This displays the statistics for the `X` and `Y` data of the `Location 2` data set.

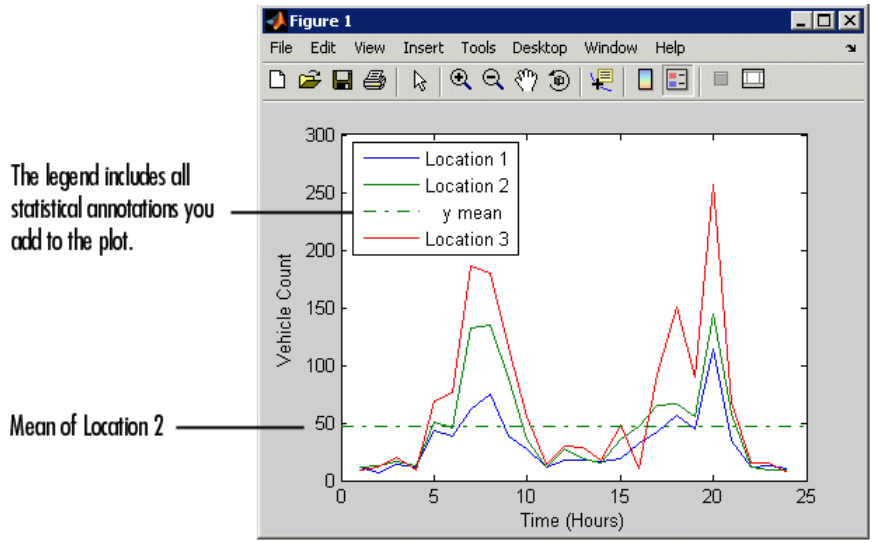


4 Select the check box for each statistic you want to display on the plot.

For example, to plot the mean of `Location 2`, select the **mean** check box in the **Y** column.



This plots a horizontal line to represent the mean of `Location 2` and updates the plot legend to include this statistic.



Formatting Data Statistics on Plots

The Data Statistics GUI uses colors and line styles to distinguish statistics from the data on the plot. This portion of the example shows how to customize the display of descriptive statistics on a plot, such as the color, line width, line style, or marker.

Note Do not edit display properties of statistics until you finish plotting all the statistics with the data. If you add or remove statistics after editing plot properties, the changes to plot properties are lost.

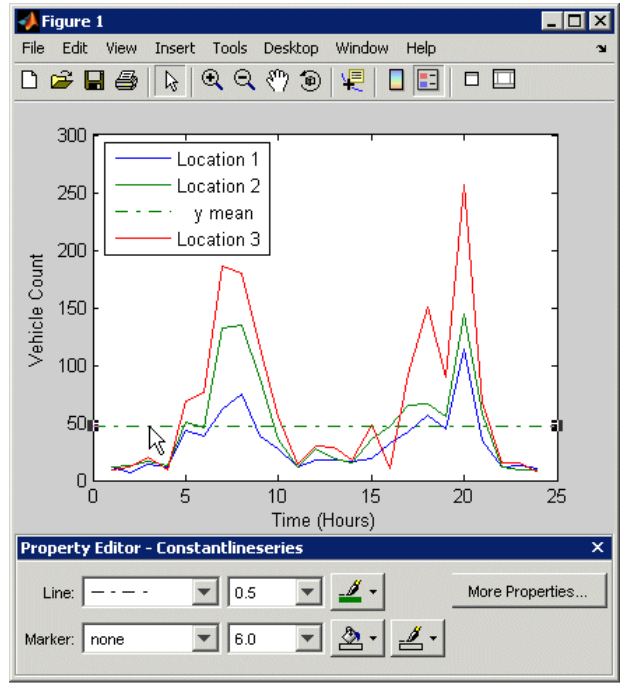
To modify the display of data statistics on a plot:

1. In the MATLAB Figure window, click the (**Edit Plot**) button in the toolbar.

   This enables plot editing.

2 Double-click the statistic on the plot for which you want to edit display properties. For example, double-click the horizontal line representing the mean of `Location 2`.

This opens the Property Editor below the MATLAB Figure window, where you can modify the appearance of the line used to represent this statistic.



3 In the Property Editor, specify the **Line** and **Marker** styles, sizes, and colors.

Tip Alternatively, right-click the statistic on the plot, and select an option from the shortcut menu.

Saving Statistics to the MATLAB Workspace

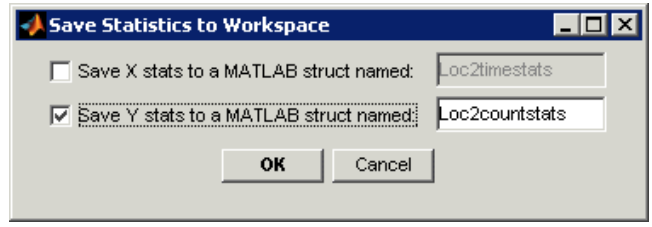
This portion of the example shows how to save statistics in the Data Statistics GUI to the MATLAB workspace.

Note When your plot contains multiple data sets, you must save statistics for each data set individually. To display statistics for a different data set, select it from the **Statistics for** list in the Data Statistics GUI.

---

1. In the Data Statistics dialog box, click the **Save to workspace** button.
2. In the Save Statistics to Workspace dialog box, specify to save statistics for either `X` data, `Y` data, or both. Then, enter the corresponding variable names.

   In this example, save only the `Y` data. Enter the variable name as `Loc2countstats`.



3. Click **OK**.

   This saves the descriptive statistics to a structure. The new variable is added to the MATLAB workspace.

To view the new structure variable, type the variable name at the MATLAB prompt:

```
Loc2countstats

Loc2countstats =

       min: 9
       max: 145
      mean: 46.5417
    median: 36
      mode: 9
       std: 41.4057
     range: 136
```

Generating an M-file

This portion of the example shows how to generate an M-file that reproduces the format of the plot and the plotted statistics with new data.

1. In the Figure window, select **File > Generate M-File**.

   This creates a function M-file and displays it in the MATLAB Editor. The code in the M-file shows you how to programmatically reproduce what you did interactively with the Data Statistics GUI and the Property Editor.

2. Change the name of the function on the first line of the M-file from `createfigure` to something more specific, like `countplot`. Save the file to your current folder with the file name `countplot.m`.

3. Generate some new, random count data:

   ```
   randcount = 300*rand(24,3);
   ```

4. Reproduce the plot with the new data and the recomputed statistics:

   ```
   countplot(t,randcount)
   ```

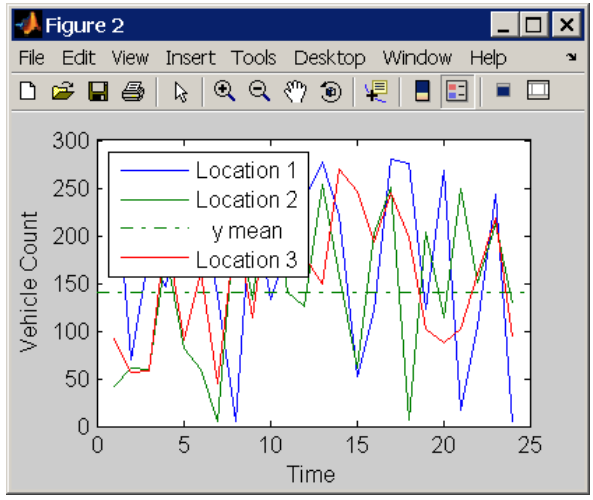


Figure 2
File Edit View Insert Tools Desktop Window Help
Location 1
Location 2
y mean
Location 3
Vehicle Count
Time
300
250
200
150
100
50
0
0
5
10
15
20
25

2

Interactive Data Exploration

- "What Is Interactive Data Exploration?" on page 2-2
- "Marking Up Graphs with Data Brushing" on page 2-4
- "Making Graphs Responsive with Data Linking" on page 2-13
- "Interacting with Graphed Data" on page 2-24

What Is Interactive Data Exploration?

Interacting with MATLAB Data Graphs

The MATLAB data analysis and graphics tools for visual data exploration leverage its Handle Graphics® capabilities. In addition to the presentation techniques described in the following section, they include:

- Highlighting and editing observations on graphs with data brushing
- Connecting data graphs with variables with data linking
- Finding, adding, removing, and changing data values with the Variable Editor
- Describing observations on graphs with data tips

Used alone or together, these tools help you to perceive trends, noise, and relationships in data sets, and understand aspects of the phenomena you model. Ways to use them are presented in the following sections. To learn more, you can also view a video tutorial that describes these and related features.

Understanding Data Using Graphic Presentations

Finding patterns in numbers is a mathematical and an intuitive undertaking. When people collect data to analyze, they often want to see how models, variables, and constants explain hypotheses. Sometimes they see by scanning tables or sets of statistics, other times by contemplating graphical representations of models and data. An analyst's powers of pattern recognition can lead to insights into data's distribution, outliers, curvilinearity, associations between variables, goodness-of-fit to models, and more. Computers amplify those powers greatly.

Graphically exploring digital data interactively generally requires:

- Data displays for charts, graphs, and maps
- A graphical user interface (GUI) capable of directly manipulating the displays

- Software that categorizes selected data performs operations on the categories, and then updates or creates new data displays

This approach to understanding is often called *exploratory data analysis* (EDA), a term coined during the infancy of computer graphics in the 1970s and generally attributed to statistician John Tukey (who also invented the box plot). EDA complements statistical methods and tools to help analysts check hypotheses and validate models. An EDA GUI usually lets analysts divide observations of variables on data plots into subsets using mouse gestures, and then analyze further or eliminate selected observations.

Part of EDA is simply looking at data graphics with an informed eye to observe patterns or lack of them. What makes EDA especially powerful, however, are interactive tools that let analysts probe, drill down, map, and spin data sets around, and select observations and trace them through plots, tables, and models.

Well before digital tool sets like the MATLAB environment developed, curious quantitative types plotted graphs, maps, and other data diagrams to trigger insights into what their collections of numbers might mean. If you are curious about what data might mean and like to reflect on data graphics, MATLAB provides many options:

- Plotting data — scatter, line, area, bar, histogram and other types of graphs
- Plotting thematic maps to show spatial relationships of point, lines and area data
- Plotting N-D point, vector, contour, surface, and volume shapes
- Overlaying other variables on points, lines, and surfaces (e.g. texture-maps)
- Rendering portions of a 3-D display with transparency
- Animating any of the above

All of these options generate static or dynamic displays that may reveal meaning in data. In many environments, however, users cannot interact with them; they can only change data or parameters and redisplay the same or different data graphics. MATLAB tools enable users to directly manipulate data displays to explore correlations and anomalies in data sets, as the following sections explain.

Marking Up Graphs with Data Brushing

In this section...

"What Is Data Brushing?" on page 2-4

"How to Brush Data" on page 2-6

"Effects of Brushing on Data" on page 2-8

"Other Data Brushing Aspects" on page 2-11

What Is Data Brushing?

When you *brush* data, you manually select observations on an interactive data display in the course of assessing validity, testing hypotheses, or segregating observations for further processing. You can brush data on 2-D graphs, 3-D graphs, and surfaces. Most of the MATLAB high-level plotting functions allow you to brush on their displays. For a list of restrictions, see "Plot Types You Cannot Brush" in the `brush` function reference page, which also illustrates the types of graphs you can brush.

Note Data brushing is a MATLAB figure interactive mode like zooming, panning or plot editing. Unlike plot edit mode, in which you can affect a graph's appearance and compose annotations but cannot manipulate its data, in data brushing mode you can select, remove, and replace individual data values.

Data brushing mode applies to an entire figure and all axes within it that contain brushable data.

Activate data brushing in any of these ways:

- Click the Data Brushing tool button on the figure toolbar.
- Click the Data Brushing tool button in the Variable Editor toolbar.
- Select Brush from the figure **Tools** menu.
- Call the `brush` function.

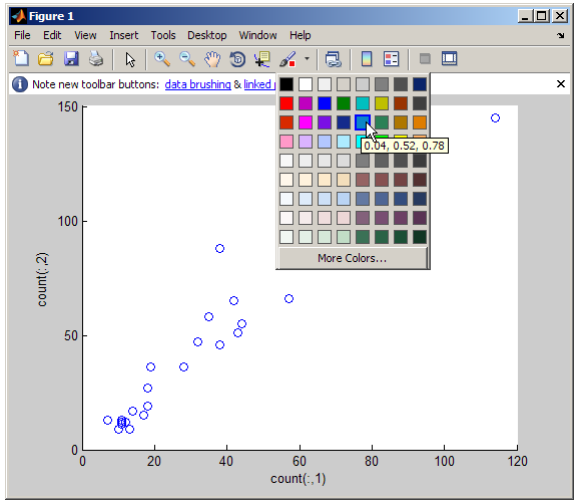
By default, data brushing is `off`. The Data Brushing tool button contains two parts and has a dual role:

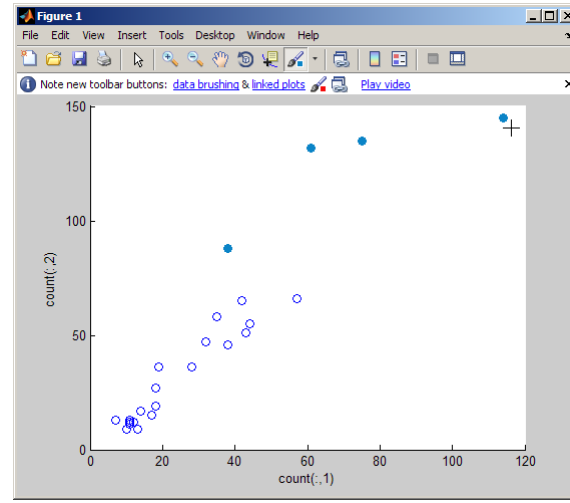
- When you click the tool icon on its left side, it toggles data brushing mode on and off.
- When you click the down arrow on its right side, it displays a drop-down menu for choosing a color for brushing data.

You can set the color with the `brush` function as well; it accepts colorspec names and RGB triplets. For example:

```
brush magenta
brush([.1 .3 .5])
```

The figures below show a scatter plot before and after brushing some outlying observations; the left-hand plot displays the Data Brushing tool palette for choosing a brush color. The information bar informs you that data brushing and linking are available for figures and contains hyperlinks to this page and other documentation. Once you dismiss this information bar by clicking the X on its right-hand side, it only reappears on subsequent plots if you select **Show linking and brushing message bar** in the MATLAB Preferences Confirmation Dialogs pane.





How to Brush Data

To brush observations on graphs and surface plots,

1. To enter brushing mode, select the Data Brushing tool; click the icon on the left side to activate the mode, and optionally select a brushing color by clicking the arrow on its right side.

2. Drag a selection rectangle to highlight observations on a graph in the current brushing color.
   Instead of dragging out a rectangle, you can click any observation to select it. Double-clicking selects all the observations in a series.

3. To add other observations to the highlighted set, hold down the **Shift** key and brush them.

4. **Shift**+clicking or **Shift**+dragging highlighted observations eliminates their highlighting and removes them from the selection set; this lets you select any set of observations.

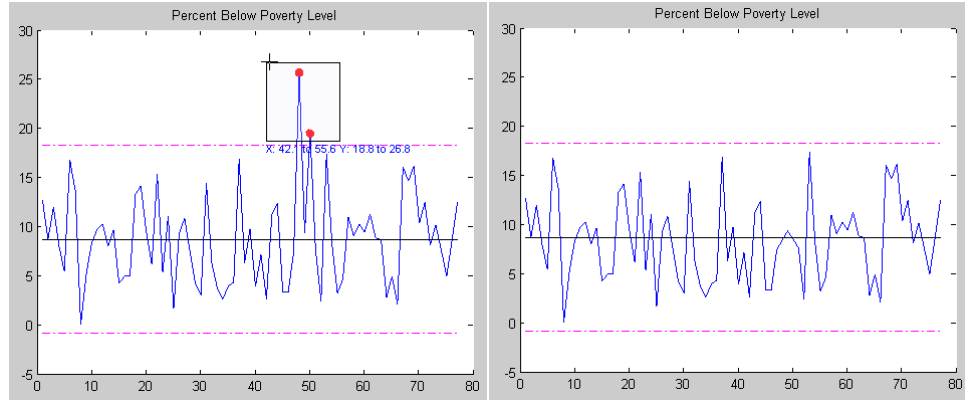
Brushed observations remain brushed even in other modes (pan, zoom, edit) until you deselect them by brushing an empty area or by selecting **Clear all**

brushing from the context menu. You can add and remove data tips to a brushed plot without disturbing its brushing.

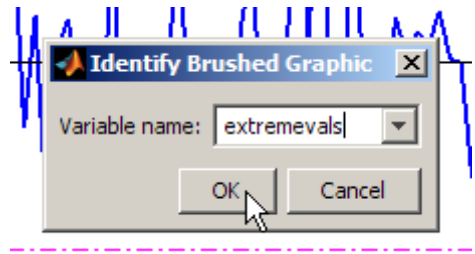
Once you have brushed observations from one or more graphed variables, you can perform several tasks with the brushing set, either from the **Tools** menu or by right-clicking any brushed observation:

- Remove all brushed observations from the plot.
- Remove all unbrushed observations from the plot.
- Replace the brushed observations with NaN or constant values.
- Copy the brushed data values to the clipboard.
- Paste the brushed data values to the command window
- Create a variable to hold the brushed data values
- Clear brushing marks from the plot (context menu only)

The two following figures show a lineseries plot of a variable, along with constant lines showing its mean and two standard deviations. On the left, the user is brushing observations that lie beyond two standard deviations from the mean. On the right, the user has eliminated these extreme values by selecting **Brushing > Remove brushed** from the **Tools** (or context) menu. The plot immediately redisplays with two fewer *x*- and *y*-values. The original workspace variable, however, remains unchanged.



Before removing the extreme values, you can save them as a new workspace variable with **Tools > Brushing > Create new variable**. Doing this opens a dialog box for you to declare a variable name.



Typing `extremevals` to name the variable and pressing **OK** to dismiss the dialog produces

```
extremevals =
   48.0000   25.7000
   50.0000   19.5000
```

The new variable contains one row per observation selected. The first column contains the *x*-values and the second column contains the *y*-values, copied from the lineseries' `XData` and `YData`. In graphs where multiple series are brushed, the Create New Variable dialog box helps you identify what series the new variable should represent, allowing you to select and name one at a time.

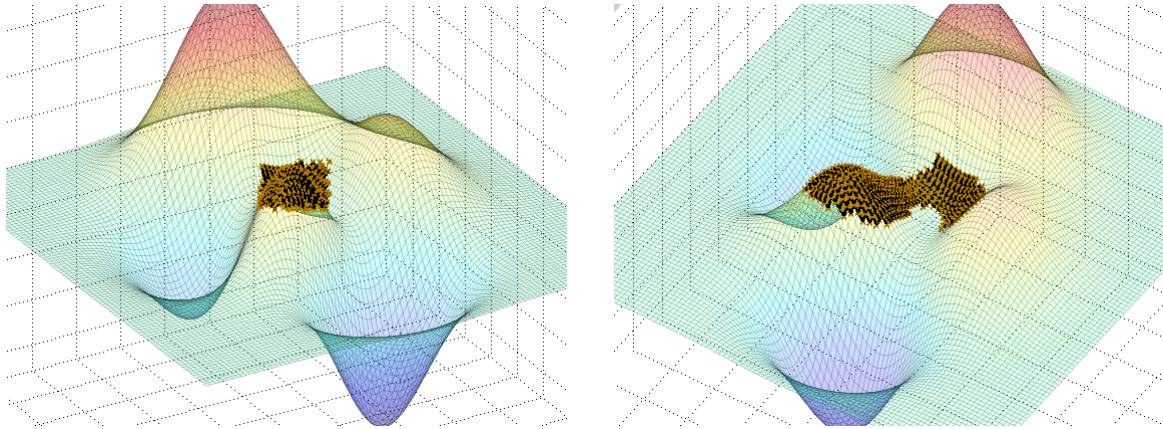
Effects of Brushing on Data

Brushing simply highlights data points in a graph, without affecting data on which the plot is based. If you remove brushed or unbrushed observations or replace them with NaN values, the change applies to the `XData`, `YData`, and possibly `ZData` properties of the plot itself, but not to variables in the workspace. You can undo such changes. However, if you replot a brushed graph using the same workspace variables, not only do its brushing marks go away, all removed or replaced values are restored and you cannot undo it. If you want brushing to affect the underlying workspace data, you must *link* the plot to the variables it displays. See "Making Graphs Responsive with Data Linking" on page 2-13 for more information.

Brushed 3-D Plots

When an axes displays three-dimensional graphics, brushing defines a region of interest (ROI) as an unbounded rectangular prism. The central axis of the prism is a line perpendicular to the plane of the screen. Opposite corners of the prism pass through points defined by the `CurrentPoint` associated with the initial mouse click and the value of `CurrentPoint` during the drag. All vertices lying within the rectangular prism ROI highlight as you brush them, even those that are hidden from view.

The next figure contains two views of a brushed ROI on a peaks surfaceplot. On the left plot, only the cross-section of the rectangular prism is visible (the brown rectangle) because the central axis of the prism is perpendicular to the viewing plane. When the viewpoint rotates by about 90 degrees clockwise (right-hand plot), you see that the prism extends along the initial axis of view and that the brushed region conforms to the surface.



Brushed Multiple Plots

When the same *x*-, *y*- or *z*-variable appears in several plots, brushing observations in one plot highlights the related ones in the others whenever the plots are linked. If the brushed variables are open in the Variable Editor, rows of data containing the brushed observations are highlighted in the brushing color there as well. For more information, see “Data Brushing with the Variable Editor” on page 2-24.

Note You can see brushed observations highlighted in the Variable Editor if you activate the Data Brushing tool on its toolbar. In data brushing mode, you can directly brush data values in the Variable Editor to highlight them on plots. In addition, if you change any values in the Variable Editor, all linked graphs on which they appear reflect those changes.

Organizing Plots for Brushing. Data brushing usually involves creating multiple views of related variables on graphs and in tables. Just as computer users organize their virtual desktops in many different ways, you can use various strategies for viewing sets of plots:

- Multiple overlapping figure windows
- Tiled figure windows
- Tabbed figure windows
- Subplots presenting multiple views

When MATLAB figures are created, by default, they appear as separate windows. Many users keep them as such, arranging, overlapping, hiding and showing them as their work requires. Any figure, however, can dock inside a figure group, which itself can float or dock in the MATLAB desktop. Once docked in a figure group, you can float and overlap the individual plots, tile them in various arrangements, or use tabs to show and hide them.

Note For more in formation on managing figure windows, see "Floating (Cascaded) Figures in Desktop Example" in the MATLAB Desktop documentation. "Managing Plotting Tools" in the MATLAB Graphics documentation provides related details.

Another way of organizing plots is to arrange them as subplots within a single figure window, as illustrated in the example for "Linking vs. Refreshing Plots" on page 2-19. You create and organize subplots with the `subplot` function, for which there is no GUI as there is for figure groups. Subplots are useful when you have an idea of how many graphs you want to work with simultaneously and how you want to arrange them (they do not need to be all the same size).

Note You can easily set up M-files to create subplots; see "Setting Up Figures" in the MATLAB Graphics documentation.

Other Data Brushing Aspects

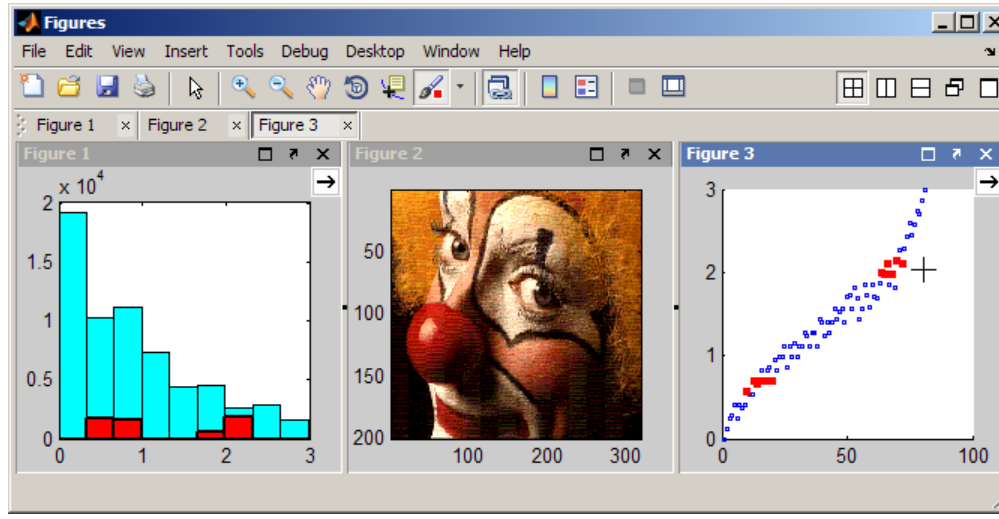
Not all types of graphs can be brushed, and each type that you can brush is marked up in a particular way. To be brushable, a graphic object must have `XDataSource`, `YDataSource`, and where applicable, `ZDataSource` properties. The one exception is the patch objects produced by the `hist` function, which are brushable due to the special handling they receive. In order to brush a histogram, you must put the figure containing it into a linked state. For related information, see "Plot Objects" in the MATLAB Graphics documentation.

The `brush` function reference page explains how to apply brushing to different graph types, describes how to use different mouse gestures for brushing, and lists graph types that you can and cannot brush. See the following sections:

- "Types of Plots You Can Brush"
- "Plot Types You Cannot Brush"
- "Mouse Gestures for Data Brushing"

Keep in mind that data brushing is a mode that operates on entire figures, like zoom, pan, or other modes. This means that some figures can be in data brushing mode at the same time other figures are in other modes. When you dock multiple figures into a figure group, there is only one toolbar, which reflects the state or mode of whatever figure docked in the group you happen to select. Thus, even when docked, some graphs may be in data brushing mode while others are not.

If an axes contains a plot type that cannot be brushed, you can select the figure's Data Brushing tool and trace out a rectangle by dragging it, but no brush marks appear. The following figure group contains a histogram and a scatter plot that describe intensity statistics for the image displayed in the middle. Although the graphs are brushable, the image itself is not. Here the graphs are shown brushed, after having linked to their data sources.



When you lay out graphs in subplots within a single figure and enter data brushing mode, all the subplot axes become brushable as long as the graphic objects they contain are brushable. If the figure is also in a linked state, brushing one subplot marks any other in the figure that shares a data source with it. Although this also happens when separate figures are linked and brushed, you can prevent individual figures from being brushed by unlinking them from data sources.

Making Graphs Responsive with Data Linking

In this section...

"What Is Data Linking?" on page 2-13
"Why Use Linked Plots?" on page 2-14
"How to Link Plots" on page 2-14
"How Linked Plots Behave" on page 2-16
"Linking vs. Refreshing Plots" on page 2-19
"Using Linked Plot Controls" on page 2-21

What Is Data Linking?

Linked plots are graphs in figure windows that visibly respond to changes in the current workspace variables they display and vice versa. This differs from the default behavior of graphs, which contain copies of variables they represent (their `XData`/`YData`/`ZData`) and must be explicitly replotted in order to update them when a displayed variable changes. For example, if variable *y* in the workspace appears in a linked plot and *y* is modified in the Command Window, the graphic representation of *y* in the linked plot updates within half a second to reflect the change.

The concept of linked data is familiar to users of the Variable Editor. When variables change or go out of scope, the Variable Editor updates itself. It continuously updates variables in the workspace when you add, change, or delete values. The Variable Editor works the same way with linked plots.

Linking graphs to workspace data has long been possible using the plot selectors in the Workspace Browser, the Variable Editor, the Plot Catalog, and MATLAB function calls. For example, you can use code such as

```
x = 0:.1:8*pi;
y = sin(x);
h = plot(x,y)
set(h,'XDataSource','x');
set(h,'YDataSource','y');
y = sin(x.^3);
refreshdata
```

to manually update the line plot of `y` versus `x` when `y` changes in the workspace. For more information on this manual technique, see the `refreshdata` reference page and "Linking Graphs to Variables — Data Source Properties" in the MATLAB Graphics documentation. Prior to data linking, users needed to explicitly update plots to reflect changes in workspace variables, as illustrated in "Linking vs. Refreshing Plots" on page 2-19.

Why Use Linked Plots?

If the same variable appears in plots in multiple figures, you can link any of the plots to the variable. You can use linked plots in concert with "Marking Up Graphs with Data Brushing" on page 2-4, but also on their own. Linking plots lets you

- Make graphs respond to changes in variables in the base workspace or within a function
- Make graphs respond when you change variables in the Variable Editor and Command Line
- Modify variables through data brushing that affect different graphical representations of them at once
- Create graphical "watch windows" for debugging purposes

Watch windows are useful if you program in the MATLAB language. For example, when refining a data processing algorithm to step through your code, you can see graphs respond to changes in variables as a function executes statements.

How to Link Plots

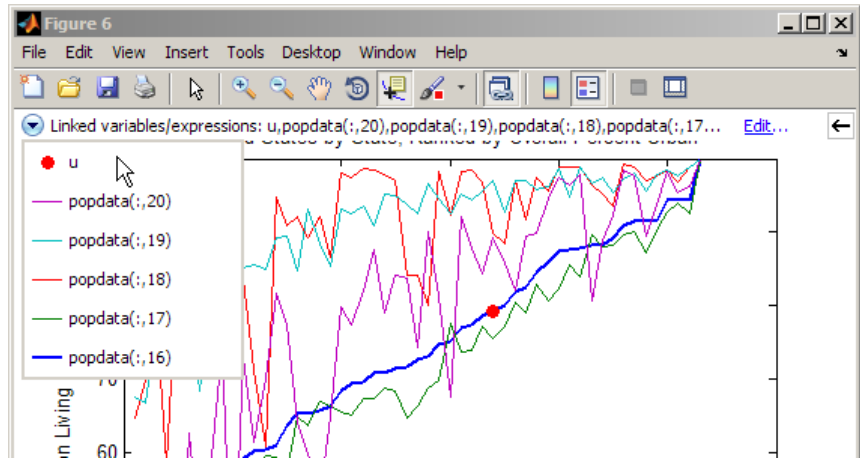
When you create a figure, by default, data linking is off. You can put a figure into a linked state in any of three ways:

- Click the Data Linking tool button on the figure toolbar.
- Select **Link** from the figure **Tools** menu.
- Call the `linkdata` MATLAB function, e.g., `linkdata on`.

- To disable data linking, click the Data Linking tool button, deselect **Tools > Link**, or type `linkdata off`.

Once a figure is linked, its appearance changes; an information bar, called the Linked Plot information bar, appears beneath the figure toolbar to reflect its new linked state. It identifies all linked variables and gives you an opportunity to unlink or relink any of them. The information bar looks like this.

The linked plot information bar identifies a figure as being linked and displays relationships between graphic objects and the workspace variables they represent. Click the circular down arrow icon on its left side to display a legend that identifies the data source for each graphic object in a graph, as in the following example.

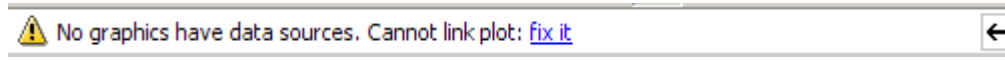


Dropping down the linked plot legend is useful when many data sources are linked to a graph at once. Like legends created with the `legend` function, it identifies graph components with variable expressions.

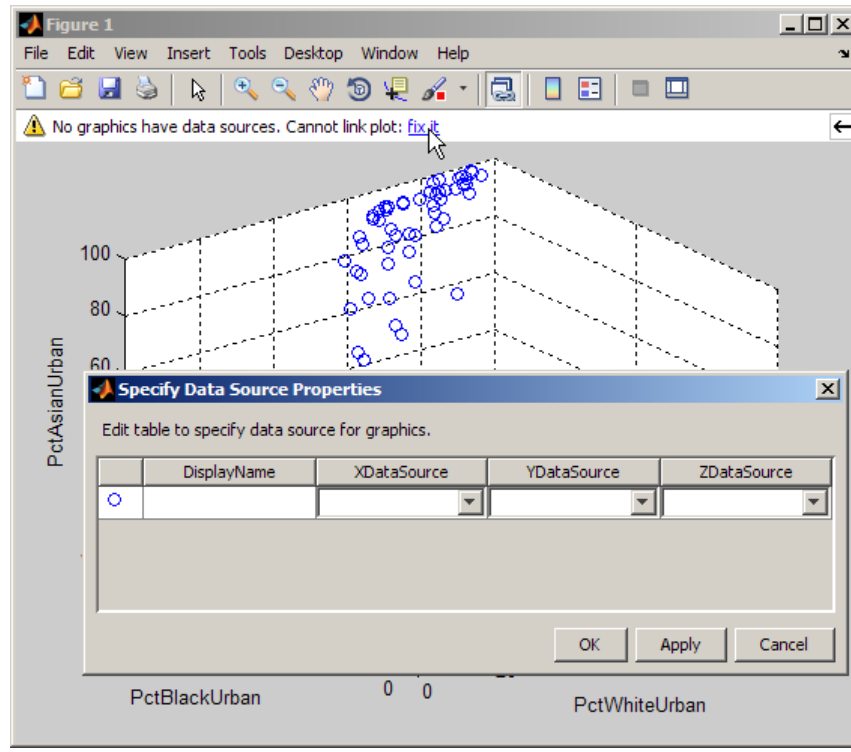
How Linked Plots Behave

Once linked to its data source(s), a figure acts as if you called the MATLAB function `refreshdata` every time a workspace variable it displays changes. That is, any series or group graphic objects contained in the figure can update its own `XData`, `YData`, or `ZData` properties and redraw itself when one of its data sources is modified. If the linked state is set to `'off'` using the `linkdata` function, by deselecting the **Data Linking** toolbar button, or by deselecting **Link** on the figure's **Tools** menu, automatic refreshing stops.

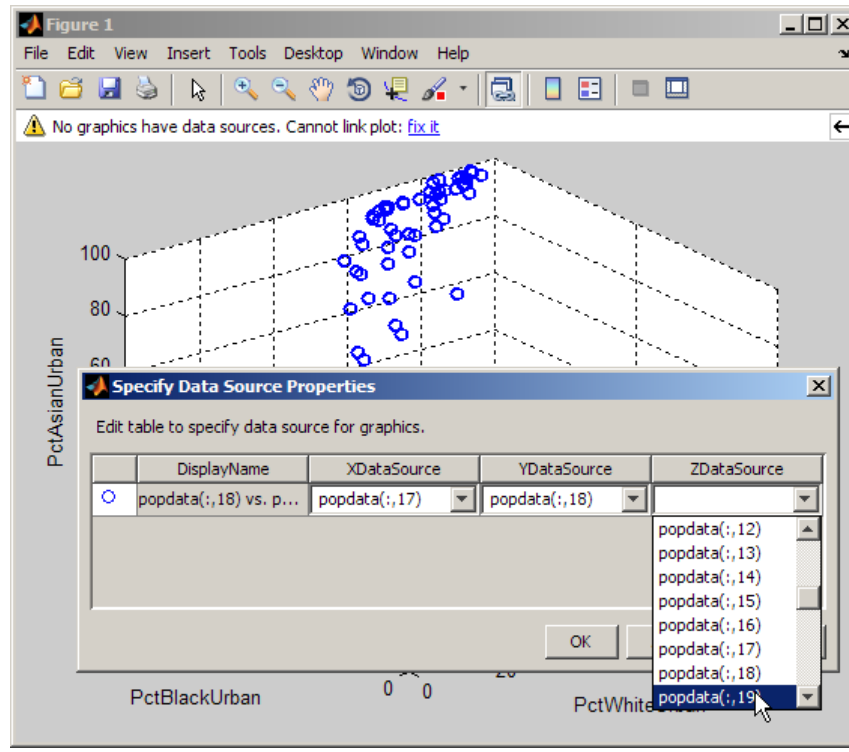
When you turn linking on for a figure, the linking mechanism can usually identify the data sources for displayed graphs, but sometimes ambiguity exists about what variable or range of a variable has been plotted. At such times, the Linked Plot information bar informs you that graphics have no data sources and gives you a chance to identify them, as you can see here.



Click **fix it** to open a dialog box where you can specify the variables and ranges of any or all plotted variables, shown in the following image for a 3-D scatter plot.



In the Specify Data Source Properties dialog box, choose a source for `XData`, `YData`, and/or `ZData` from drop-down menus or type an expression. For 2-D plots, usually you must specify at least `YData` and for 3-D plots, `ZData`. In the next image, the expressions `popdata(1:end-1,17)`, `popdata(1:end-1,18)`, and `popdata(1:end-1,19)` are typed in, in order to identify the appropriate columns and to exclude the final row of the data matrix from the plot. The `DisplayName` property (used by the `legend` function) is also set to `'Pct Urban'`.



Tip Save time by using the drop-down lists to select data sources unless you need to specify ranges of data or other expressions.

Note You can create graphs that have no data sources. For example, `plot(randn(100,1))` generates a line graph that has neither an `XDataSource` (the *x*-values are implicit) nor a `YDataSource` (no variable for *y*-values exists). Therefore, while you can brush such graphs, you cannot link them to data sources, because linking requires workspace data. Similarly, if you create a variable, graph it, and then clear the variable from the workspace you will be unable to link that plot.

When you brush a graph that is not linked to data sources, you brush the graphics only. The brushing affects only the figure you interact with. However, when you brush a linked plot, you are brushing the underlying variables. In this case, your brush marks also display on all linked plots that have the same data sources you brushed, as well as any display of that data which you have opened in the Variable Editor. The color of the brush marks in all displays is the brush color you have selected for the figure in which you are brushing. This color can differ from the brush colors you have chosen to use in others display, and overrides those colors.

Linking vs. Refreshing Plots

Besides the linked plots feature, other MATLAB mechanisms connect graphic objects to data sources (workspace variables). The main techniques are:

- Directly update the `XData/YData/ZData` properties of a graph.
- Set a graph's `XDataSource/YDataSource/ZDataSource` and indirectly update `XData/YData/ZData` by calling `refreshdata`.

For an example of using these techniques to animate graphs, see "Updating Plot Object Axis and Color Data" in the MATLAB Graphics documentation. That section explains that data linking is not a method intended for animating data graphs.

Linking plots automates these tasks and keeps graphs continuously in sync with the variables they depict, making it the easiest technique to use. Data sources must still exist in the workspace, but you do not need to explicitly declare them for linked plots unless some ambiguity exists. The following code examples iteratively approximate *pi*, and illustrate the difference between declaring and refreshing data sources yourself and letting the `linkdata` function handle it for you.

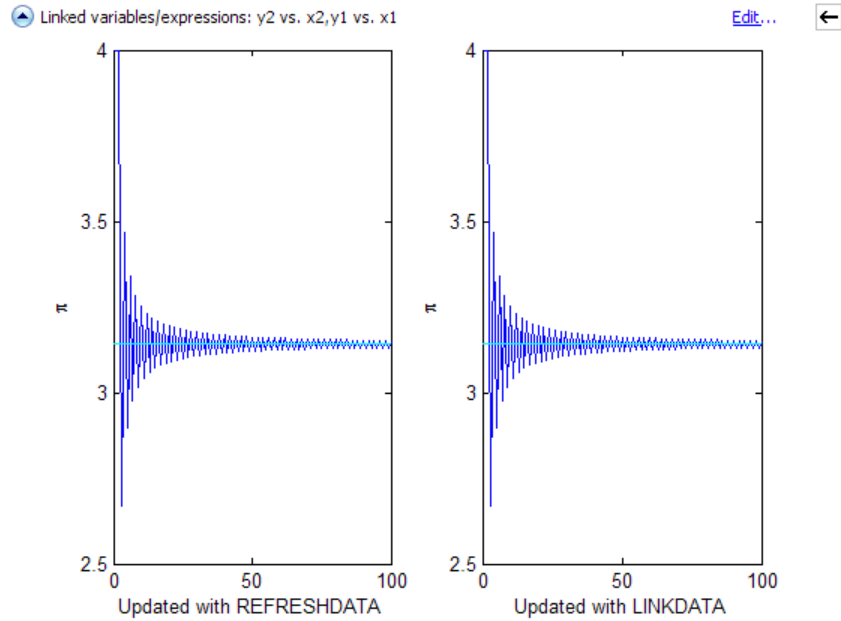
Updating a Graph with refreshdata

```
x1= [1 2];
y1 = [4 4];
ntimes = 100;
denom = 1;
k = -1;
subplot(1,2,1)
hp1 = plot(x1,y1);
xlabel('Updated with REFRESHDATA')
ylabel('\pi')
set(gca,'Xlim',[0 ntimes],...
        'Ylim',[2.5 4])
set(hp1,'XDataSource', 'x1')
set(hp1,'YDataSource', 'y1')
for t = 3:ntimes
    denom = denom + 2;
    x1(t) = t;
    y1(t) = 4*(y1(t-1)/4 + k/denom);
    refreshdata
    drawnow
    k = -k;
end
line([0 ntimes], [pi pi],'color','c')
```

Updating a Graph with linkdata

```
x2= [1 2];
y2 = [4 4];
ntimes = 100;
denom = 1;
k = -1;
subplot(1,2,2)
plot(x2,y2);
xlabel('Updated with LINKDATA')
ylabel('\pi')
set(gca,'Xlim',[0 ntimes],...
        'Ylim',[2.5 4])
linkdata on
for t = 3:ntimes
    denom = denom + 2;
    x2(t) = t;
    y2(t) = 4*(y2(t-1)/4 + k/denom);
    k = -k;
end
line([0 ntimes], [pi pi],'color','c')
```

Differences are shown in italics. When you execute the code on the left, which uses `refreshdata`, it animates the approximation process. The code on the right uses `linkdata` and does not animate; it runs much faster. (A `drawnow` command is not needed, because data linking buffers updates and refreshes the graph is at half-second intervals.) The graphic results, shown in the next image, are identical. Because both plots are in axes in the same figure, linking the second graph also links the first graph to its variables.



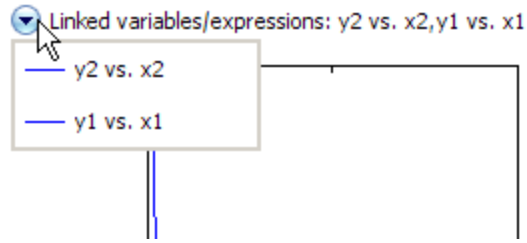
Using Linked Plot Controls

To minimize the Linked Plot information bar while remaining in linked mode, click the hide/show button on its right side; the button flips direction and the bar is hidden. Clicking the button again flips the arrow back and restores the Linked Plot information bar. Turning off linking cuts all data source connections and removes the Linked Plot information bar from the figure. However, the data source properties remain set, and the bar reappears whenever a linked state is restored by selecting **Tools > Link**, depressing the **Linked Plot** button, or calling the `linkdata` function. Whatever data sources were established previously will then reconnect (assuming those variables still exist in the same form).

The Data Source Button

The down arrow button on the left side of the Linked Plot information bar drops down a legend (similar to what the `legend` function produces but without Display Names). The legend identifies workspace variables associated

with plot objects for the entire figure (`legend` works on a per-axes basis), such as these linked lineseries from the previous example, shown in the next image.



The drop-down legend names variable linked to the graphic objects in the figure. For items to appear there, a graph must have an `XDataSource`, `YDataSource`, or a `ZDataSource` property that MATLAB can evaluate without error. The icon for each list entry reflects the `Color`, `Linestyle` and `Marker` of the corresponding graphic object, making clear which graphic objects link to which variables. The drop-down legend is informational only; you can only dismiss it after reading it by clicking anywhere else on the figure.

The Edit Button

Clicking the **Edit** link on the information bar opens the Specify Data Source Properties modal dialog box for you to set the `DisplayName`, `XDataSource`, `YDataSource`, and `ZDataSource` properties of plot objects in the figure to columns or vectors of workspace variables. Changing a `DisplayName` updates text on a legend, if present for the variable, and has no other effects. The three columns on the right contain drop-down lists of workspace variables. You can also type variable names and ranges, or an M-expression. When you change variables or their ranges on the fly with this dialog box, variables plotted against one another must be compatible types and have the same number of observations (as in any bivariate graph).

If you attempt to link a plot and `linkdata` can identify more than one possible workspace variable for one or more plot objects, the Specify Data Source Properties dialog box appears for you to resolve the ambiguity. If you choose not to or are unable to do so and cancel the dialog box, data linking is not established for those graphic objects.

When Data Links Fail

Updating a linked plot can fail if the strings in the `XDataSource`, `YDataSource`, or `ZDataSource` properties are incompatible with what is in the current workspace. Consequently, the corresponding `XData`, `YData`, and `ZData` cannot be updated. This happens most often because variables are cleared or no longer exist when the workspace changes (e.g., when you are debugging).

However, failing links do not affect the visual appearance of the object in the graph. Instead, a warning icon and message appears on the Linked Plot information bar when this occurs for any plotted data in the figure. The failing link warning is general, but you can identify which variables are affected by clicking the Data Source button. If you hide the Linked Plot information bar (by clicking its Hide button), the bar reappears when a data links fails, alerting you to the issue.

Interacting with Graphed Data

In this section...

"Data Brushing with the Variable Editor" on page 2-24

"Using Data Tips to Explore Graphs" on page 2-25

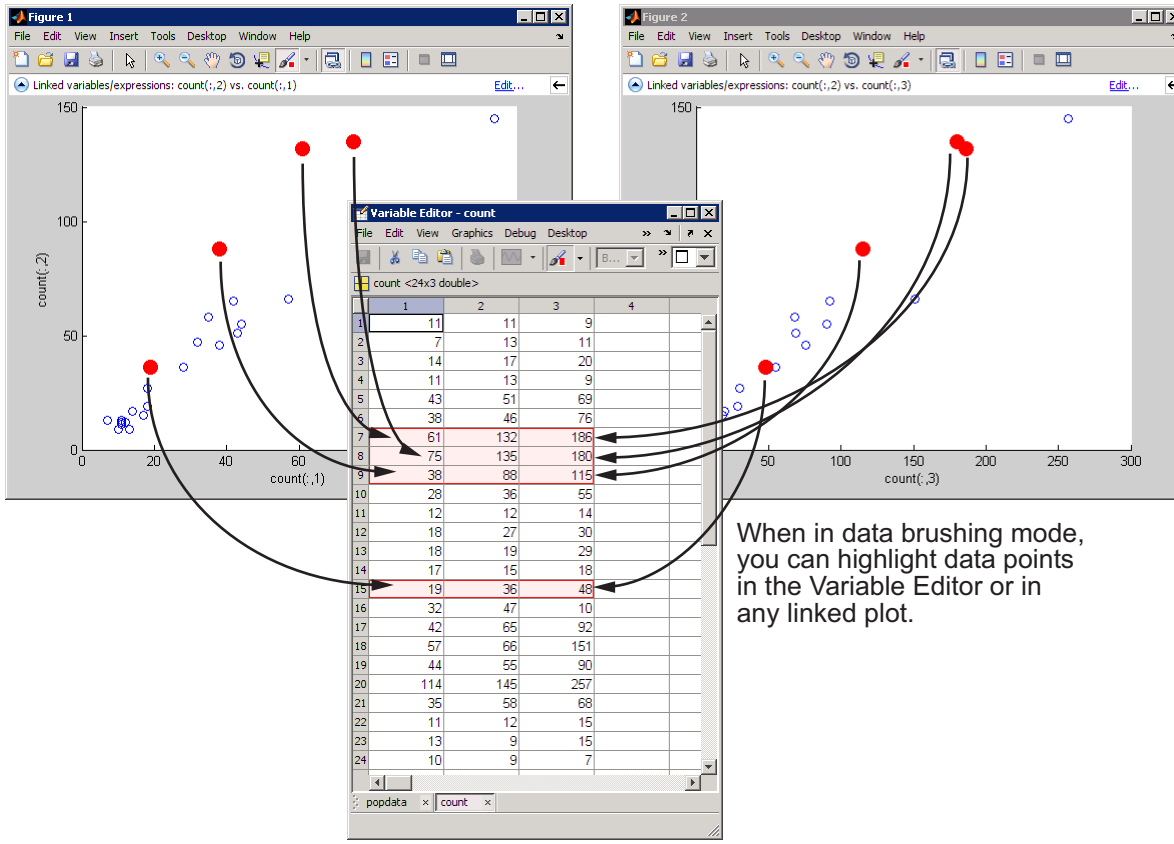
"Example — Visually Exploring Demographic Statistics" on page 2-26

Data Brushing with the Variable Editor

Shared variables in linked figures are highlighted in all figures when data in one is brushed. They also highlight when you open the variables in the Variable Editor.

The Variable Editor also has a Data Brushing tool. It has no Data Linking tool, however, because in the Variable Editor, variables are always "live," and their data sources therefore respond immediately to any changes you make in the Variable Editor. This means that whenever you place it in data brushing mode, brush marks and changes to data values you make in the Variable Editor appear in linked plots.

If you have linked plots of matrix data with observations across rows and where each column represents a distinct, related quantity, brushing any observation—whether in a graph or the Variable Editor—highlights all observations in the same row, as you can see in the next image.



For more information about the using the Variable Editor, see “Viewing and Editing Workspace Variables with the Variable Editor” in the MATLAB Desktop Tools and Development Environment documentation and see the reference page for `openvar`.

Using Data Tips to Explore Graphs

A data tip is a small display associated with an axes that reads out individual data observation values from a 2-D or 3-D graph. You create data tips by mouse clicks on graphs using the **Data Cursor tool** from the figure toolbar. When you select this tool, you are in data cursor mode—signified by a hollow cross-hair cursor—in which you identify *x*-, *y*-, and *z*-values of

data points you click. Like data points you brush, export such values to the workspace.

For descriptions of data cursor properties and how to use them, see

- "Data Cursor — Displaying Data Values Interactively" and Using Data Cursors with Histograms in the in the MATLAB Graphics documentation
- The MATLAB function reference page for `datacursormode`

The default behavior of data tips is to simply display the `XData`, `YData`, and `ZData` values of the selected observations as text in a box. Sometimes this information is not helpful by itself, and you might want to replace or augment it with other information. You can modify this behavior to display other facts connected to observations. You customize data tip behavior by constructing a data tip text update function (in M-code) to construct text strings for display in data tips and then instructing data cursor mode to use your function instead of the default one.

Customize data cursor update functions to display information such as

- Names associated with *x*-, *y*-, and *z*-values
- Weights associated with *x*-, *y*-, and *z*-values
- Differences in *x*-, *y*-, and *z*-values from the mean or their neighbors
- Transformations of values (e.g., normalizations or to different units of measure)
- Related variables

You can create data tip text update functions to display such information and change their behavior on the fly. You can even make the update function behave differently for distinct observations in the same graph if your update function or the code calling it can distinguish groups of them. The next section contains an example of coding and using a customized data cursor update function.

Example — Visually Exploring Demographic Statistics

- "The Data Tip Text Update Function" on page 2-27

- "Preparing, Plotting, and Annotating the Data" on page 2-29
- "Explore the Graph with the Custom Data Cursor" on page 2-32
- "Plot and Link a Histogram of a Related Variable" on page 2-34
- "Explore the Linked Graphs with Data Brushing" on page 2-36
- "Plot the Observations on a Linked Map" on page 2-37

The extended example that follows begins by using data tips to explore the incidence of fatal traffic accidents tabulated for U.S. states, with respect to state populations. The example extends this analysis to brush, link, and map the data to discover spatial patterns in the data. Each section of the example has four or fewer steps. By executing them all, you gain insight into the data set and become familiar with useful graphical data exploration techniques.

Censuses of population and other national government statistics are valuable sources of demographic and socioeconomic data. An important aspect of census data is its geography, i.e., the regions to which a given set of statistics applies, and at what level of granularity. When exploring census data, you frequently need to identify what geographic unit any given observation represents.

This example uses data tips to show place names and statistics for individual observations. You pass place names and the data matrix to a custom text update function to enable this. The place names are for U.S. states and the District of Columbia. If all these names were placed as labels on the *x*-axis, they would be too small or too crowded to be legible, but they are readable one at a time as data tips.

The example also illustrates how sorting a data matrix by rows can enhance interpretation when the original ordering (in this case alphabetical by state) provides no special insight into relationships among observations and variables.

The Data Tip Text Update Function

Data tips can present other information beyond *x*-, *y*- and *z*-values. Read through the example function `labeldtips`, which takes three more parameters than a default callback, and displays the following information:

- Its *y*-value

- Deviation from an expected *y*-value
- Percent deviation from the expected *y*-value
- The observation's label (state name)

Because it customizes data tips, the function must be an M-file that you invoke from the Command Window or from a script. This file, `labeldtips.m`, and the MAT-files `accidents.mat` and `usapolygon.mat` that the following examples also use, exist on the MATLAB path. Here is the code for the `labeldtips` data cursor callback function.

```
function output_txt = labeldtips(obj,event_obj,...
                      xydata,labels,xymean)
% Display an observation's Y-data and label for a data tip
% obj          Currently not used (empty)
% event_obj    Handle to event object
% xydata       Entire data matrix
% labels       State names identifying matrix row
% xymean       Ratio of y to x mean (avg. for all obs.)
% output_txt   Datatip text (string or string cell array)
% This datacursor callback calculates a deviation from the
% expected value and displays it, Y, and a label taken
% from the cell array 'labels'; the data matrix is needed
% to determine the index of the x-value for looking up the
% label for that row. X values could be output, but are not.

pos = get(event_obj,'Position');
x = pos(1); y = pos(2);
output_txt = {['Y: ',num2str(y,4)]};
ydev = round((y - x*xymean));
ypct = round((100 * ydev) / (x*xymean));
output_txt{end+1} = ['Yobs-Yexp: ' num2str(ydev) ...
                     '; Pct. dev: ' num2str(ypct)];
idx = find(xydata == x,1);  % Find index to retrieve obs. name
% The find is reliable only if there are no duplicate x values
[row,col] = ind2sub(size(xydata),idx);
output_txt{end+1} = cell2mat(labels(row));
```

The portion of the example called "Explore the Graph with the Custom Data Cursor" on page 2-32 sets up data cursor mode and declares this function as a callback using the following code:

```
hdt = datacursormode;
set(hdt,'UpdateFcn',{@labeldtips,hwydata,statelabel,usmean})
```

The call to `datacursormode` puts the current figure in data cursor mode. `hdt` is the handle of a data cursor mode object for the figure you want to explore. The function name and its three formal arguments are a cell array.

Preparing, Plotting, and Annotating the Data

The following steps show how you load statistical data for U.S. states, plot some of it, and enter data cursor mode to explore the data:

Note To help you interpret graphs created in this example, the `hwydata` data matrix and its row labels have been presorted by rows to be in ascending order by total state population. The 51-by-1 vector `hwyidx` contains indices from the presorting (the data were originally in alphabetic order)

If you ever want to resort the data array and state labels alphabetically, you can sort on the first column of the `hwydata` matrix, which contains Census Bureau state IDs that ascend in alphabetical order, as follows:

```
[hwydata hwyidx] = sortrows(hwydata,1);
statelabel = statelabel(hwyidx);
```

If you do resort the data, to make the graph easier to interpret you might plot it using markers rather than lines. To do this, change the call to `plot` in section 2, below, to the following:

```
plot(hwydata(:,14),hwydata(:,4),'.')
```

---

1 Load U.S. state data statistics from the National Transportation Safety Highway Administration and the Bureau of the Census and look at the variables:

```
load 'accidents.mat'
whos
  Name             Size             Bytes  Class

  datasources       3x1              2568  cell
  hwycols           1x1                 8  double
  hwydata          51x17             6936  double
  hwyheaders        1x17              1874  cell
  hwyidx           51x1                408  double
  hwyrows           1x1                 8  double
  statelabel       51x1              3944  cell
  ushwydata         1x17               136  double
  uslabel           1x1                 86  cell
```

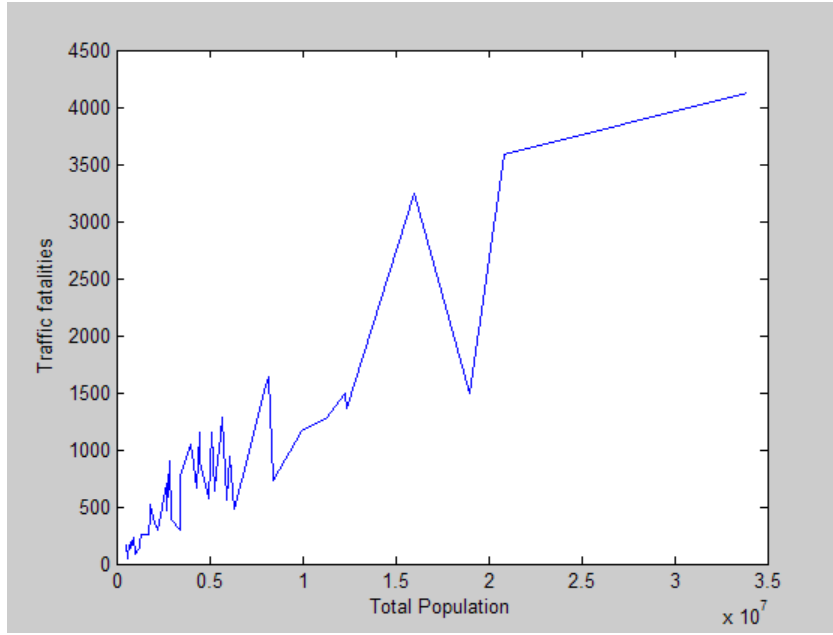
The data set has 51 observations for 17 variables.

- The state-by-state statistics; the double 51-by-17 matrix `hwydata`
- The variable (column) names; the 1-by-17 text cell array `hwyheaders`
- The state names; the 51-by-1 text cell array `statelabel`
- Values for the entire United States for the 17 variables; the 1-by-17 matrix `ushwydata`
- The label for the US values; the 1-by-1 cell array `uslabel`
- Metadata describing data sources; the 3-by-1 cell array `datasources`

2 Plot a line graph of the population by state as *x* versus the number of traffic fatalities per state as *y*:

```
hf1 = figure;
plot(hwydata(:,14),hwydata(:,4));
xlabel(hwyheaders(14))
ylabel(hwyheaders(4))
```

Because the state observations are sorted by population size, the graph is monotonic in *x*. The larger a population a state has, the more variation in traffic accident fatalities it tends to show.



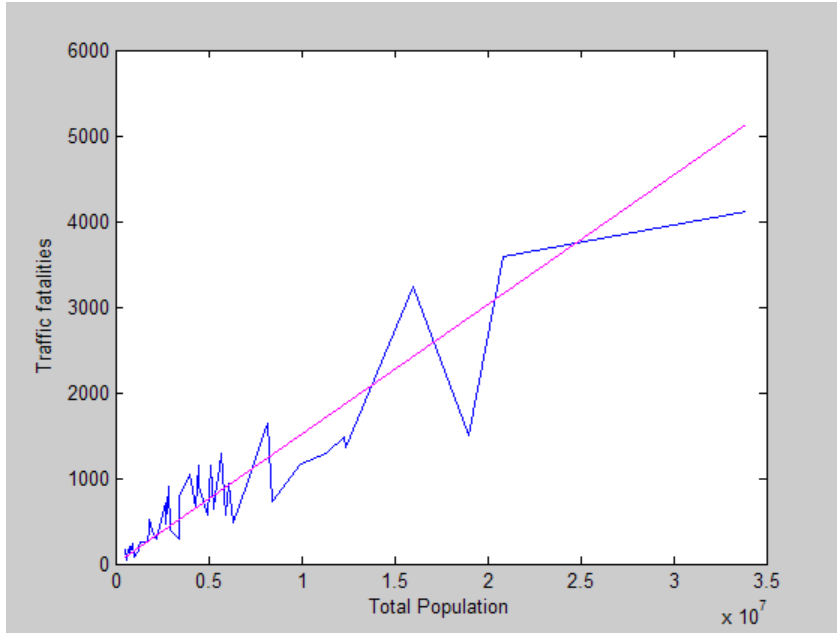
3 Compute the per capita rate of traffic fatalities for the entire United States; in the next part of this example, the data cursor update function uses this average to compute an expected value for each state you query:

```
usmean = ushwydata(4)/ushwydata(14)

usmean =
  1.5150e-004
```

The statistic shows that nationally, about 150 per 100,0000 people die in traffic accidents every year.

Use `usmean` to compute the smallest and largest expected values by multiplying it by the smallest and largest state populations, and draw a line connecting them:

```
line([min(hwydata(:,14)) max(hwydata(:,14))],...
     [min(hwydata(:,14))*usmean max(hwydata(:,14)*usmean)],...
      'Color','m');
```



Note The magenta line is not a regression line; it is a trend line that plots the number of traffic deaths that a state of a given size would have if all states obeyed the national average.

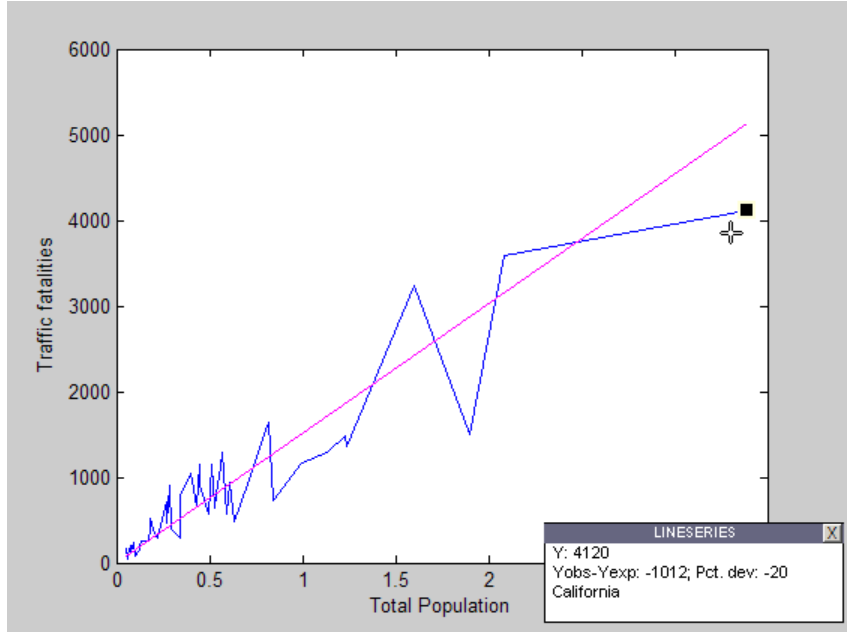
Explore the Graph with the Custom Data Cursor

You can now explore the graphed data with the example custom data cursor update function `labeldtips` (which must be on the MATLAB path or in the current folder). `labeldtips` displays state names and y-deviations.

1 Turn on data cursor mode and invoke the custom callback:

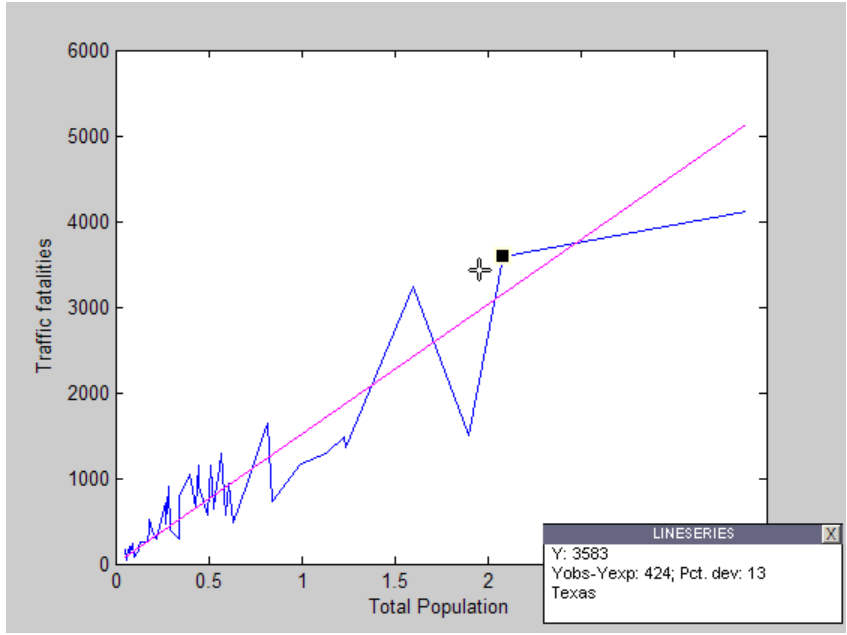
```
hdt = datacursormode;
set(hdt,'DisplayStyle','window');
% Declare a custom datatip update function to display state names
set(hdt,'UpdateFcn',{@labeldtips,hwydata,statelabel,usmean})
```

The data cursor `'window'` display style sends data tip output to a small window that you can move anywhere within the figure. This display style is best suited to data tips that contain more text than just *x*-, *y*-, and *z*-values. The `labeldtips` callback remains active for that figure until you use `set` to replace it with another function (or empty, to restore the default data cursor behavior). Click the right-most point on the blue graph.



The data tip shows that California has the largest population and the largest number of traffic fatalities, 4120. However, it had 1012, or 20%, fewer fatalities than predicted by the national average.

2 The next data point to the left depicts Texas. Click that data point or press the left arrow to show its data tip.



Texas had 3583 fatalities, which is 424 (13%) more than the expected value. To see results from other states, move the data tip by dragging the black square or using the left or right arrow to step it along the graph. If you know a little about U.S. geography, you might observe a pattern.

Plot and Link a Histogram of a Related Variable

The ninth column of `hwydata`, labeled "Fatalities per 100K Licensed Drivers," is related to population. Plot a histogram of this variable to see which states have fewer or more fatalities per driver. To do this, link the plots to their data, and brush either of them.

1 Open a new figure and plot a histogram of Fatalities per 100K Licensed Drivers in it:

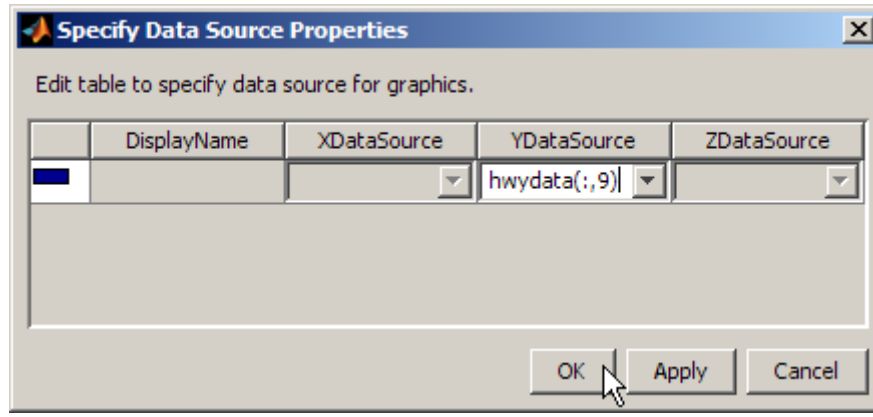
```
hf2 = figure
hist(hwydata(:,9),5)
xlabel(hwyheaders(9))
```

2 Link both the line graph and the histogram to their data sources in `hwydata`:

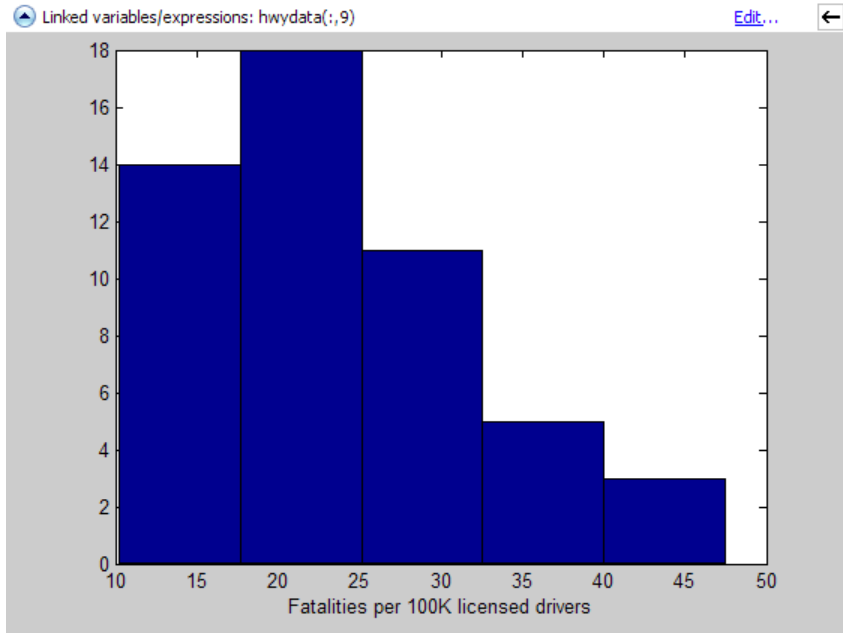
```
linkdata(hf1)
linkdata(hf2)
```

You can also click the **Data Linking** tool on the two figures. The first figure links automatically; the histogram does not because `linkdata` cannot determine with certainty the `YDataSource` for histograms. The Linked Plot information bar on top of the histogram informs you **No Graphics have data sources. Cannot link plot: fix it**.

3 Click **fix it** to open the Specify Data Source Properties dialog box. Type `hwydata(:,9)` into the `YDataSource` edit box and click **OK**.



The Linked Plot information bar displays the data source you identified. The histogram looks like this.

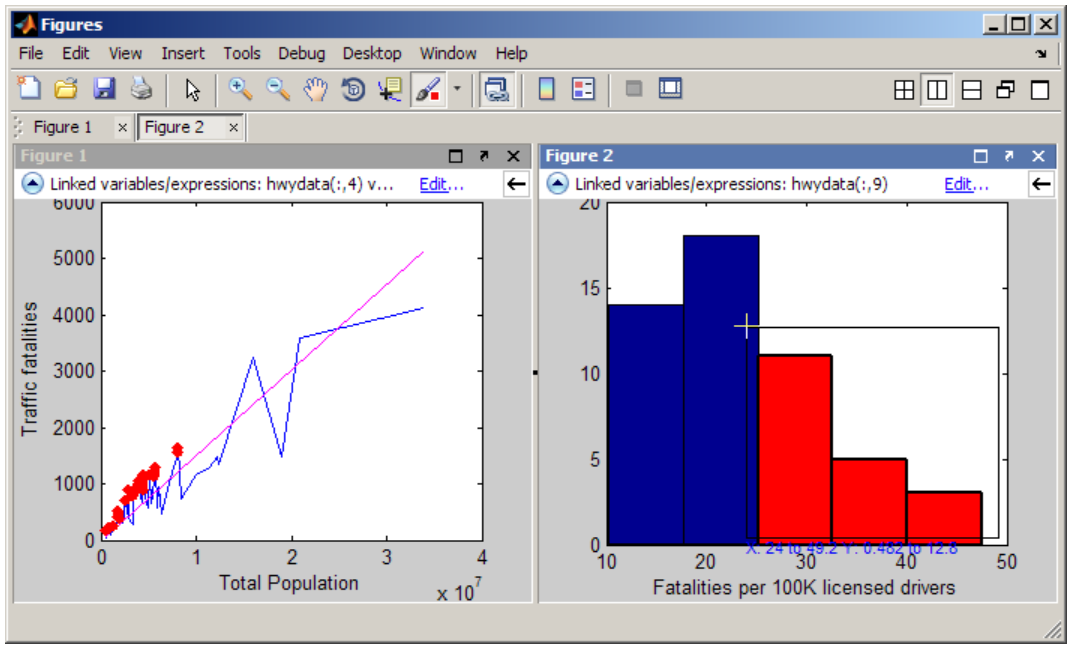


Explore the Linked Graphs with Data Brushing

Now that you have linked both graphs to a common data set, you can brush portions of one to see the effect on the other.

1. It isn't necessary, but you might want to dock the plots in a figure group so you can see them side by side.

2. Select the Data Brushing tool on the histogram plot. Brush the three right-most bars in the histogram; they represent higher values that range from 25 to 48 fatalities per 100,000 drivers.



Notice which observations light up on the line graph. Not only are these states with smaller populations, they are also states with above-average numbers of traffic fatalities.

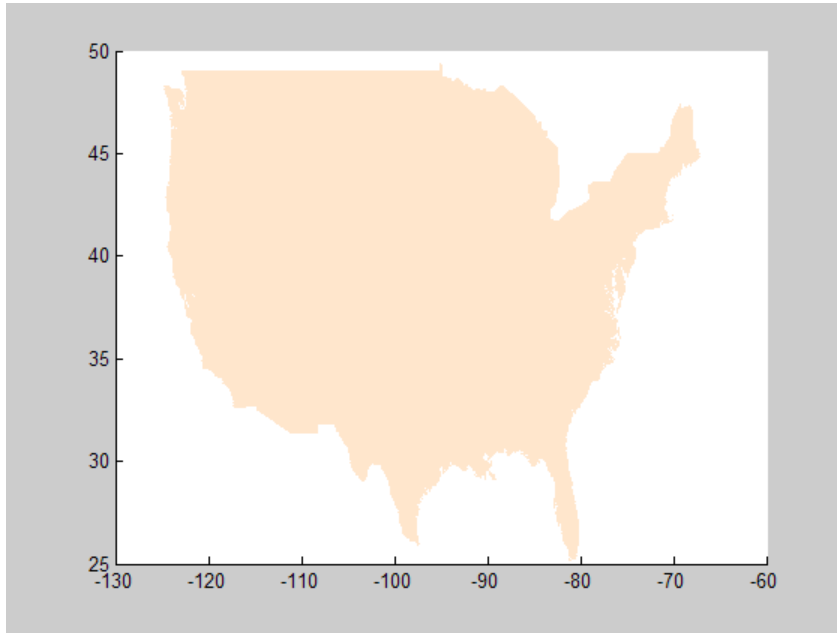
3 Click the line graph to make it the active figure and select its Data Brushing tool. Click all the observations you can that fall *below* the straight line average. You need to hold the **Shift** key down to make multiple selections, whether by clicking or dragging. You might want to zoom in on the left side of the graph to brush properly there. What do you see happening on the histogram?

Plot the Observations on a Linked Map

The `hwydata` matrix contains geographic location information in the form of latitude-longitude coordinates of a centroid for each state. You can make a crude map by generating a scatter plot of these coordinates, using longitude as *x* and latitude as *y*. If you link the scatter plot, you can brush all the plots at once.

1 To provide a context for the map, plot an outline map of the conterminous United State. Obtain the latitude and longitude coordinates required from the demo MAT-file `usapolygon.mat`:

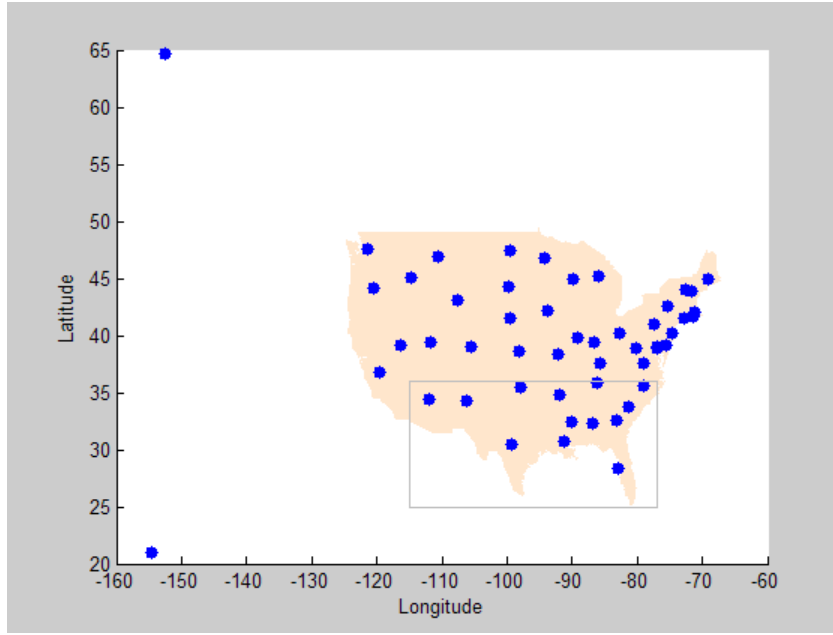
```
hf3 = figure;
load usapolygon
patch(uslon,uslat,[1 .9 .8],'Edgecolor','none');
hold on
```



When projected into the figure. the map is distorted to fit the aspect ratio of the axes.

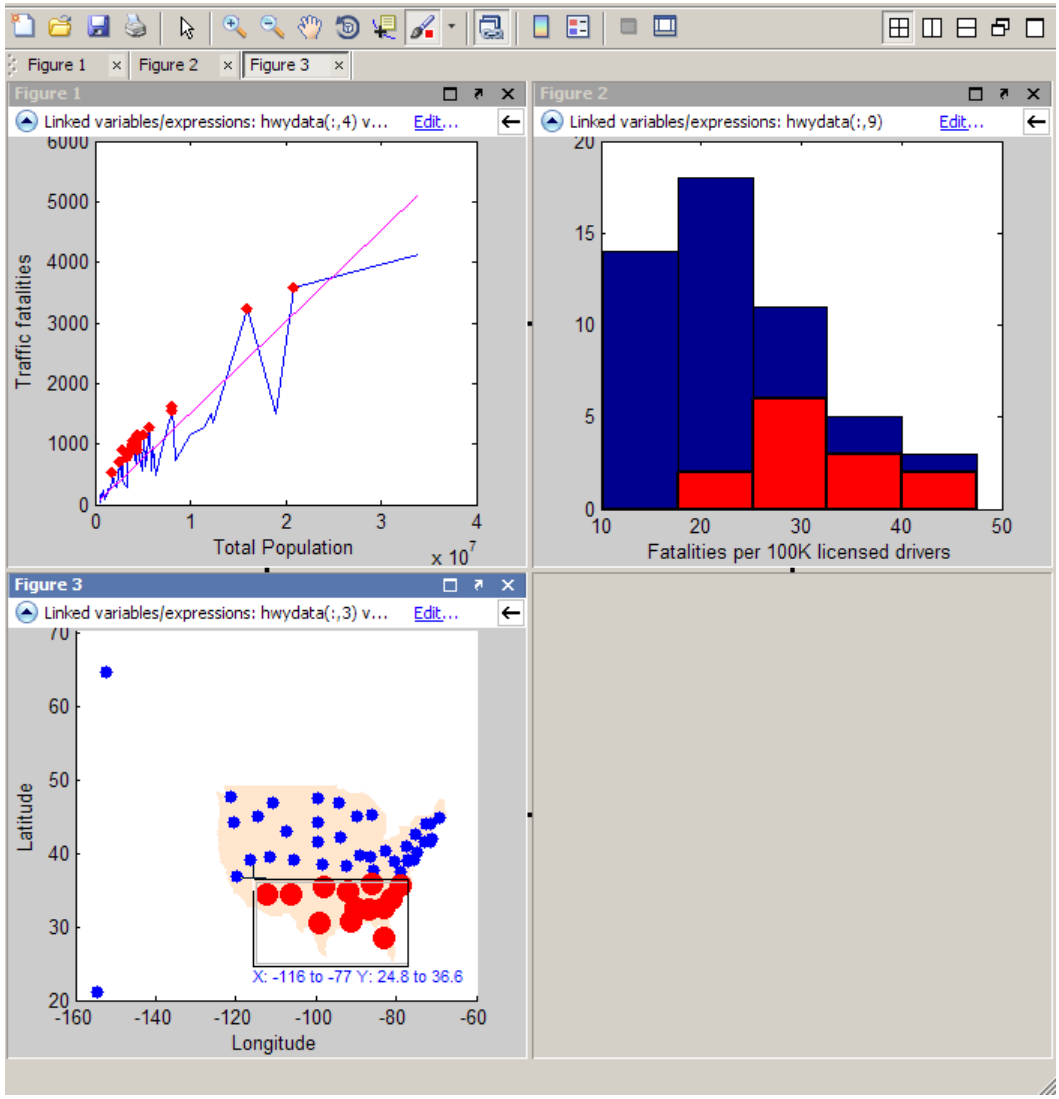
2 Map the centroid longitude and latitude as a scatter plot with filled circles. Plot a rectangle over part of the map, as follows:

```
scatter(hwydata(:,2),hwydata(:,3),36,'b','filled');
xlabel('Longitude')
ylabel('Latitude')
rectangle('Position',[-115,25,115-77,36-25],...
    'EdgeColor',[.75 .75 .75])
```



The *x*- and *y*-limits change, shrinking the map, because the data matrix contains observations for Alaska and Hawaii, but the map outline file does not include these states.

3 Dock the map underneath the other two figures. Brush the map after turning on the Data Linking and Data Brushing tools for its figure. Drag across the gray rectangle with the Data Brushing tool to highlight just the southeastern and southwestern states. What you see should look like this.



Data brushing and linking reveals that almost all the states with above-average traffic fatality rates are in the southern part of the U.S.

Using graphic data exploration, you have identified some intriguing regularities in this data. However, you have not identified any causes for the

patterns you found. That will take more work on with the data, and possibly additional data sets, along with some hypotheses and models.

3

Regression Analysis

- "Linear Correlation" on page 3-2
- "Linear Regression" on page 3-7
- "Interactive Fitting" on page 3-10
- "Programmatic Fitting" on page 3-23

Linear Correlation

In this section...
"Introduction" on page 3-2
"Covariance" on page 3-2
"Correlation Coefficients" on page 3-5

Introduction

Before you fit a function to model the relationship between two measured quantities, it is a good idea to determine if a relationship exists between these quantities.

Correlation quantifies the strength of a linear relationship between two variables. When there is no correlation between the two quantities, then there is no tendency for the values of one quantity to increase or decrease with the values of the second quantity.

The following three MATLAB functions compute correlation coefficients and covariance. In typical data analysis applications, where you are mostly interested in the degree of relationship between variables, you might only calculate correlation coefficients, but these are derived from covariances.

Function	Description
`corrcoef`	Correlation coefficient matrix
`cov`	Covariance matrix
`xcorr` (a Signal Processing Toolbox™ function)	Cross-correlation sequence of a random process (includes autocorrelation)

Covariance

Use the MATLAB `cov` function to explicitly calculate the covariance matrix for a data matrix (where each column represents a separate quantity).

The covariance matrix has the following properties:

- `cov(X)` is symmetrical.
- `diag(cov(X))` is a vector of variances for each data column, which represent a measure of the spread or dispersion of data in the corresponding column.
- `sqrt(diag(cov(X)))` is a vector of standard deviations.
- The off-diagonal elements of the covariance matrix represent the covariance between the individual data columns.

Here, `X` can be a vector or a matrix. For an *m*-by-*n* matrix, the covariance matrix is *n*-by-*n*.

For an example of calculating the covariance, load the sample data in `count.dat` that contains a 24-by-3 matrix:

```
load count.dat
```

Calculate the covariance matrix for this data:

```
cov(count)
```

MATLAB responds with the following result:

```
ans =
    1.0e+003 *
       0.6437  0.9802  1.6567
       0.9802  1.7144  2.6908
       1.6567  2.6908  4.6278
```

The covariance matrix for this data has the following form:

$$\begin{bmatrix} \sigma^2_{11} & \sigma^2_{12} & \sigma^2_{13} \\ \sigma^2_{21} & \sigma^2_{22} & \sigma^2_{23} \\ \sigma^2_{31} & \sigma^2_{32} & \sigma^2_{33} \end{bmatrix}$$

$$\sigma^2 ij = \sigma^2{}_{ji}$$

Here, σ^2_{ij} is the covariance between column i and column j of the data. Because the `count` matrix contains three columns, the covariance matrix is 3-by-3.

Note In the special case when a vector is the argument of `cov`, the function returns the variance.

Correlation Coefficients

The correlation coefficient matrix represents the normalized measure of the strength of linear relationship between variables.

The correlation coefficient $r_{X,Y}$ between two random variables X and Y with expected values μ_X and μ_Y and standard deviations σ_X and σ_Y is their covariance normalized by their standard deviations, as follows

$$r_{X,Y} = \frac{\mathrm{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E((X-\mu X)(Y-\mu Y))}{\sigma_X \sigma_Y}$$

where E is the expected value operator and cov means covariance. Since μX = E(X), σX2 = E(X2) − E2(X), and likewise for Y, $r_{X,Y}$ is also

$$r_{X,Y} = \frac{E(XY)-E(X)E(Y)}{\sqrt{E(X^2)-E^2(X)}\sqrt{E(Y^2)-E^2(Y)}}$$

The correlation is defined only if both of the standard deviations are finite and both of them are nonzero.

For time series, correlation coefficients r_k are given by

$$r_k = \frac{\sum_{t=1}^{N}(x_t-\bar{x})(x_{t+k}-\bar{x})}{\sum_{t=1}^{N}(x_t-\bar{x})^2}$$

where x_t is a data value at time step t, k is the lag, and the overall mean is given by

$$\bar{x} = \sum_{t=1}^{N}\frac{x_t}{N}$$

The MATLAB function `corrcoef` produces a matrix of correlation coefficients for a data matrix (where each column represents a separate quantity). The correlation coefficients range from -1 to 1, where

- Values close to 1 suggest that there is a positive linear relationship between the data columns.
- Values close to -1 suggest that one column of data has a negative linear relationship to another column of data (*anticorrelation*).
- Values close to or equal to 0 suggest there is no linear relationship between the data columns.

For an *m*-by-*n* matrix, the correlation-coefficient matrix is *n*-by-*n*. The arrangement of the elements in the correlation coefficient matrix corresponds to the location of the elements in the covariance matrix, as described in "Covariance" on page 3-2.

For an example of calculating correlation coefficients, load the sample data in `count.dat` that contains a 24-by-3 matrix:

```
load count.dat
```

Type the following syntax to calculate the correlation coefficients:

```
corrcoef(count)
```

This results in the following 3-by-3 matrix of correlation coefficients:

```
ans =
    1.0000    0.9331    0.9599
    0.9331    1.0000    0.9553
    0.9599    0.9553    1.0000
```

Because all correlation coefficients are close to 1, there is a strong correlation between each pair of data columns in the `count` matrix.

Linear Regression

In this section...

"Introduction" on page 3-7

"Residuals and Goodness of Fit" on page 3-8

"Fitting Data with Curve Fitting Toolbox Functions" on page 3-8

Introduction

A data *model* explicitly describes a relationship between *predictor* and *response* variables. Linear regression fits a data model that is linear in the model coefficients. The most common type of linear regression is a *least-squares fit*, which can fit both lines and polynomials, among other linear models.

Before you model the relationship between pairs of quantities, it is a good idea to perform correlation analysis to establish if a linear relationship exists between these quantities. When you do so, be aware that variables can have nonlinear relationships, which correlation analysis cannot detect. For more information, see "Linear Correlation" on page 3-2.

The MATLAB Basic Fitting GUI helps you to fit your data, which enables you to calculate model coefficients and plot the model on top of the data. For an example of using this GUI, see "Example: Using Basic Fitting GUI" on page 3-12. You can also use the MATLAB functions `polyfit` and `polyval` to fit your data to a model that is linear in the coefficients. For an example of using these functions, see "Example: Programmatic Fitting" on page 3-30.

If you need to fit data with a nonlinear model, you can try transforming the variables to make them linear. You can use the Statistics Toolbox `nlinfit` function, or use Curve Fitting Toolbox™ functions.

In this chapter, you learn how to do the following:

- Use correlation analysis to determine whether two quantities are related to justify fitting the data.
- Fit a linear model to the data.

- Plot the model and the data on the same plot.
- Evaluate the goodness of fit using a plot of the residuals.

Residuals and Goodness of Fit

Residuals are defined as the difference between the *observed* values of the response variable and the values that are *predicted* by the model. When you fit a model that is appropriate for your data, the residuals approximate independent random errors.

To calculate fit parameters for a linear model, the sum of the squares of the residuals are minimized to produce a good fit. This is called a least-squares fit.

You can gain insight into the "goodness" of a fit by visually examining a plot of the residuals: if the residual plot has a pattern (i.e., does not appear to have a random scatter), this indicates that the model does not properly fit the data.

Notice that the "goodness" of a fit must be determined in the context of your data. For example, if your goal of fitting the data is to extract coefficients that have physical meaning, then it is important that your model reflect the physics of the data. Understanding what your data represents and how it was measured is important when evaluating the goodness of fit.

Fitting Data with Curve Fitting Toolbox Functions

The Curve Fitting Toolbox software extends core MATLAB functionality by enabling the following data-fitting capabilities:

- Linear and nonlinear parametric fitting, including standard linear least squares, nonlinear least squares, weighted least squares, constrained least squares, and robust fitting procedures
- Nonparametric fitting
- Statistics for determining the goodness of fit
- Extrapolation, differentiation, and integration
- GUI that facilitates data sectioning and smoothing
- Saving fit results in various formats, including M-files, MAT-files, and workspace variables

For more information, see the Curve Fitting Toolbox documentation.

Interactive Fitting

In this section...

"The Basic Fitting GUI" on page 3-10
"Preparing for Basic Fitting" on page 3-10
"Opening the Basic Fitting GUI" on page 3-11
"Example: Using Basic Fitting GUI" on page 3-12

The Basic Fitting GUI

The MATLAB Basic Fitting GUI allows you to interactively:

- Model data using a spline interpolant, a shape-preserving interpolant, or a polynomial up to the tenth degree
- Plot one or more fits together with data
- Plot the residuals of the fits
- Compute model coefficients
- Compute the norm of the residuals (a measure of the goodness of fit)
- Use the model to interpolate or extrapolate outside of the data
- Save coefficients and computed values to the MATLAB workspace for use outside of the GUI
- Generate an M-file to recompute fits and reproduce plots with new data

---

Note The Basic Fitting GUI is only available for 2-D plots. For more advanced fitting and regression analysis, see the Curve Fitting Toolbox documentation and the Statistics Toolbox documentation.

Preparing for Basic Fitting

The Basic Fitting GUI sorts your data in ascending order before fitting. If your data set is large and the values are not sorted in ascending order, it will take longer for the Basic Fitting GUI to preprocess your data before fitting.

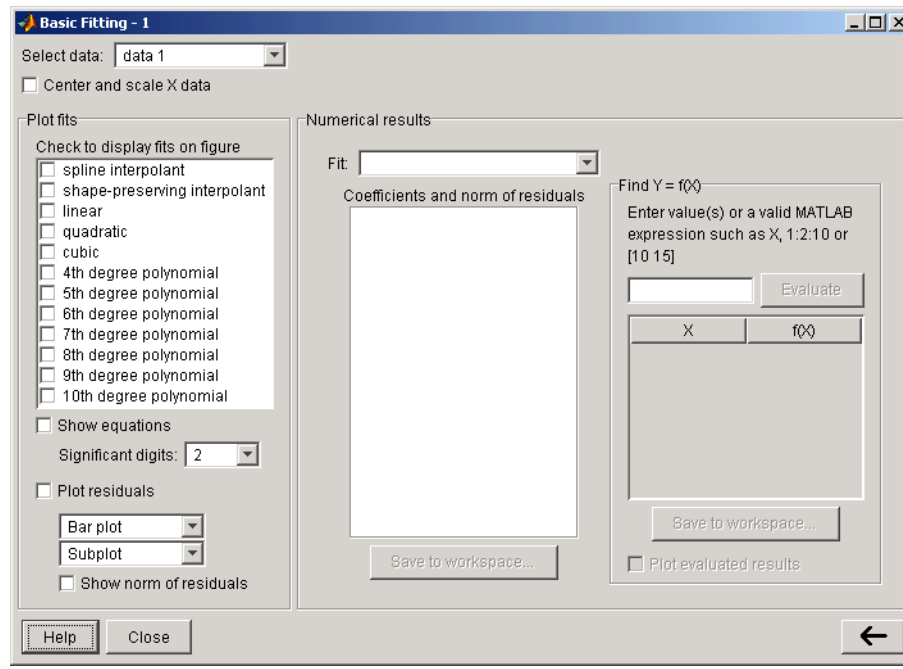
You can speed up the Basic Fitting GUI by first sorting your data. To create sorted vectors `x_sorted` and `y_sorted` from data vectors `x` and `y`, use the MATLAB `sort` function:

```
[x_sorted, i] = sort(x);
y_sorted = y(i);
```

Opening the Basic Fitting GUI

To use the Basic Fitting GUI, you must first plot your data in a figure window, using any MATLAB plotting command that produces (only) *x* and *y* data.

To open the Basic Fitting GUI, select **Tools > Basic Fitting** from the menus at the top of the figure window.



The GUI consists of three panels:

- For selecting a model and plotting options

- For examining and exporting model coefficients and norms of residuals
- For examining and exporting interpolated and extrapolated values

To expand or collapse the panels, use the arrow button in the lower right corner of the interface.

Example: Using Basic Fitting GUI

The example in this section shows you how to use the Basic Fitting GUI.

- "Loading and Plotting Data" on page 3-12
- "Fitting Data" on page 3-13
- "Viewing and Saving Fit Parameters" on page 3-17
- "Interpolating and Extrapolating Values" on page 3-18
- "Generating an M-file" on page 3-21

Loading and Plotting Data

The file `census.mat` contains U.S. population data for the years 1790 through 1990.

To load and plot the data, type the following commands at the MATLAB prompt:

```
load census
plot(cdate,pop,'ro')
```

The `load` command adds the following two variables to the MATLAB workspace:

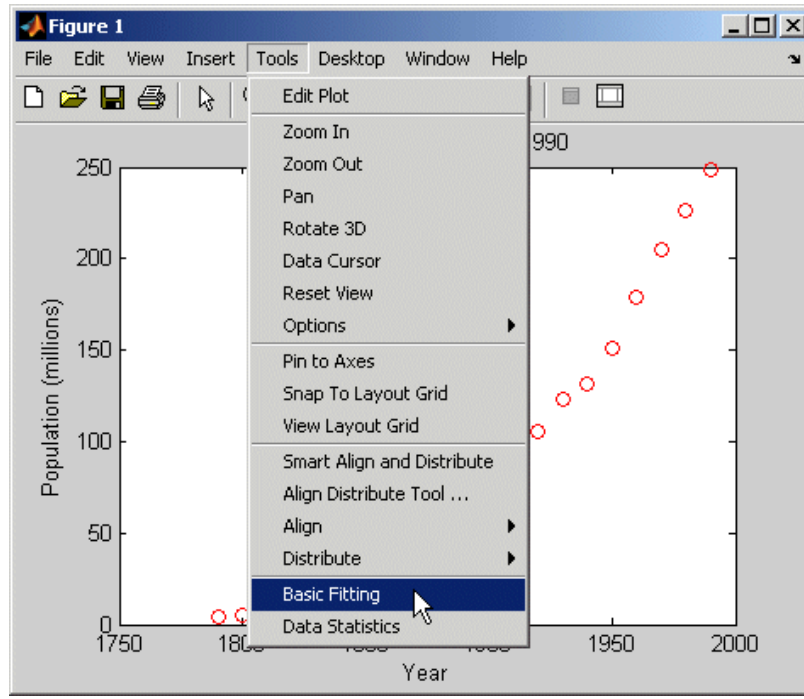
- `cdate` is a column vector containing the years from 1790 to 1990 in increments of 10. This is the predictor variable.
- `pop` is a column vector with U.S. population for each year in `cdate`. This is the response variable.

The data vectors are sorted in ascending order, by year. The plot shows the population as a function of year.

Now you are ready to fit the data.

Fitting Data

1 Open the Basic Fitting dialog box by selecting **Tools > Basic Fitting** in the Figure window.



2 In the **Plot fits** area of the Basic Fitting dialog box, select the **cubic** check box to fit a cubic polynomial to the data.

MATLAB displays the following warning:

```
Polynomial is badly conditioned. Removing
repeated data points or centering and scaling
may improve results.
```

The warning indicates that the computed coefficients for the model will be highly sensitive to random errors in the response (in this case, the measured population). To improve model accuracy, it is helpful to transform the predictors (in this case, the dates) by normalizing their center and scale. This is done by computing the *z-scores*:

$$z = \frac{x - \mu}{\sigma}$$

where x is the predictor data, μ is the mean of x, and σ is the standard deviation of x. This centers the data at 0, with a standard deviation of 1.

To perform this transformation on the predictor data, select the **Center and scale x data** check box.

After centering and scaling, model coefficients are computed for the y data as a function of z. These are different (and more robust) than the coefficients computed for y as a function of x. The form of the model, and the norm of the residuals, is unchanged. The Basic Fitting GUI automatically rescales the z-scores so that the fit is displayed on the same scale as the original x data.

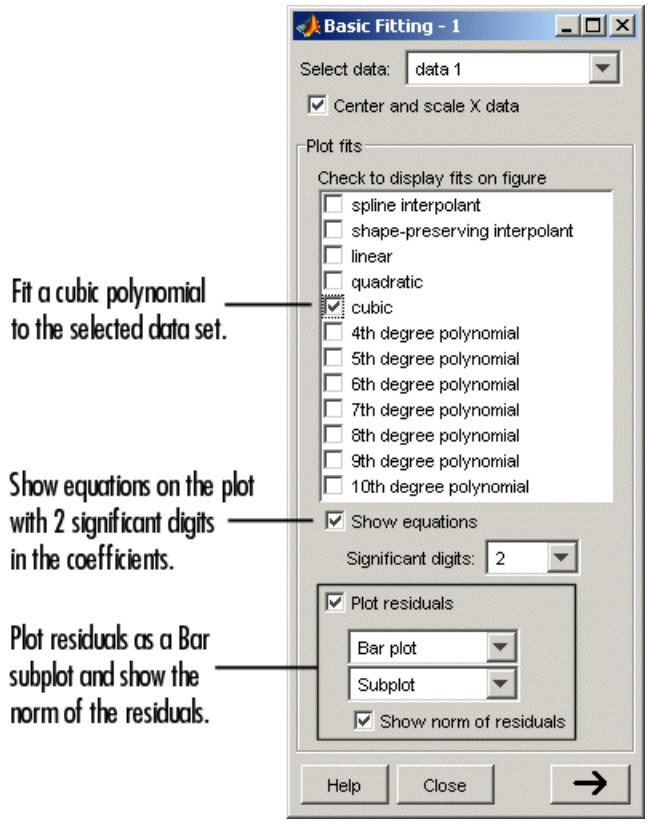
The Basic Fitting GUI calls the MATLAB functions `polyfit` and `polyval` to compute and display the fit. To understand the way in which the centered and scaled data is used as an intermediary to create the final plot, type the following at the MATLAB command prompt:

```
load census
x = cdate;
y = pop;
z = (x-mean(x))/std(x); % Compute z-scores of x data

plot(x,y,'ro') % Plot data
hold on

zfit = linspace(z(1),z(end),100);
pz = polyfit(z,y,3); % Compute conditioned fit
yfit = polyval(pz,zfit);

xfit = linspace(x(1),x(end),100);
```

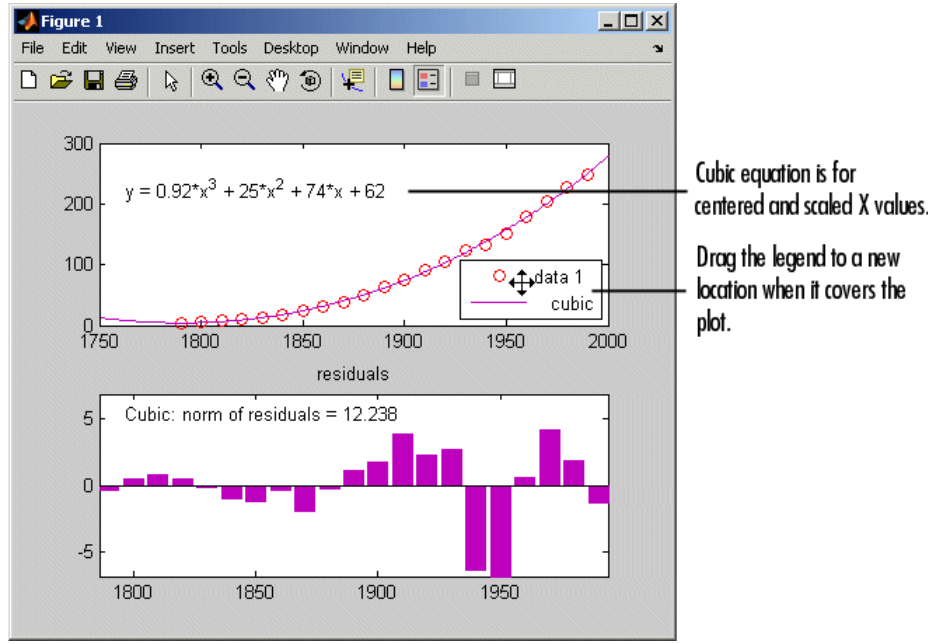
```
plot(xfit,yfit,'b-') % Plot conditioned fit vs. x data
```

3 Select the following options:

- Display the model equation in the plot
- Display the residuals as a subplot
- Display the norm of the residuals in the plot



The resulting display is shown in the following figure:



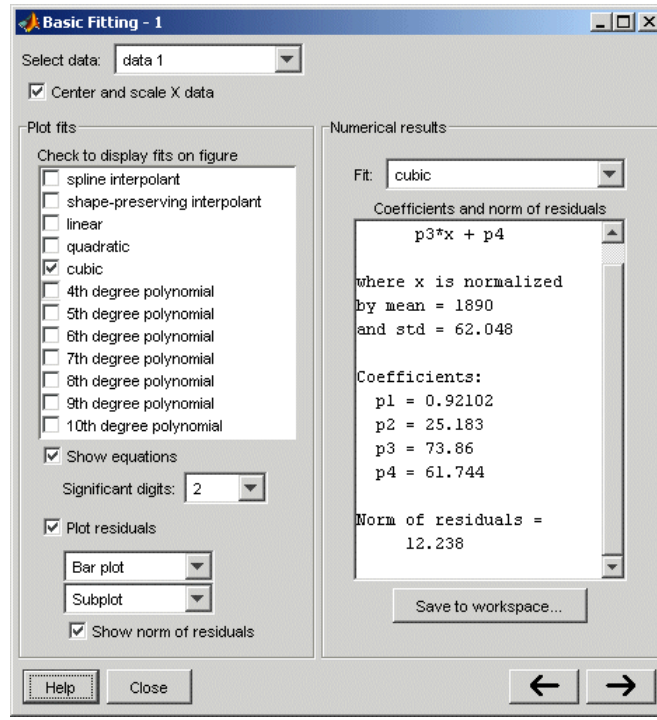
The cubic fit is a poor predictor before the year 1790, where it indicates a decreasing population. The model seems to approximate the data reasonably well after 1790, but a pattern in the residuals shows that the model does not meet the assumption of normal error, which is a basis for the least-squares fitting carried out by the Basic Fitting GUI.

For comparison, try fitting another equation to the census data by selecting it in the **Plot fits** area.

Tip You can change the default plot settings or rename data sets with the Property Editor.

Viewing and Saving Fit Parameters

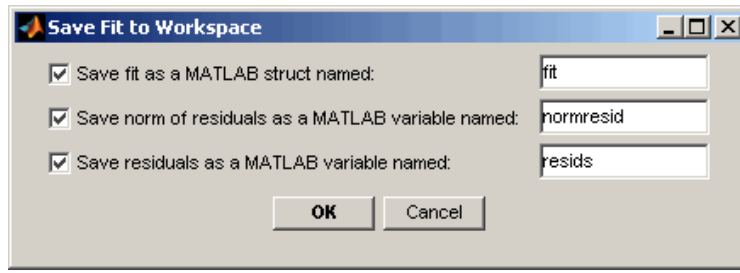
In the Basic Fitting dialog box, click the arrow button → to display the estimated coefficients and the norm of the residuals in the **Numerical results** panel.



To view a specific fit, select it from the **Fit** list. This displays the coefficients in the Basic Fitting dialog box, but does not plot the fit in the figure window.

Note If you also want to display a fit on the plot, you must select the corresponding **Plot fits** check box.

Save the fit data to the MATLAB workspace by clicking the **Save to workspace** button on the **Numerical results** panel. This opens the following dialog box:



Click **OK** to save the fit parameters as a MATLAB structure:

```
fit
fit =
     type: 'polynomial degree 3'
    coeff: [0.9210 25.1834 73.8598 61.7444]
```

You can now use the fit results in MATLAB programming, outside of the Basic Fitting GUI.

Interpolating and Extrapolating Values

Suppose you wish to use the cubic model to interpolate the U.S. population in 1965 (not in the original data).

In the Basic Fitting dialog box, click the → button to specify a vector of `x` values at which to evaluate the current fit.

1 In the **Enter value(s)...** field, type the following value:

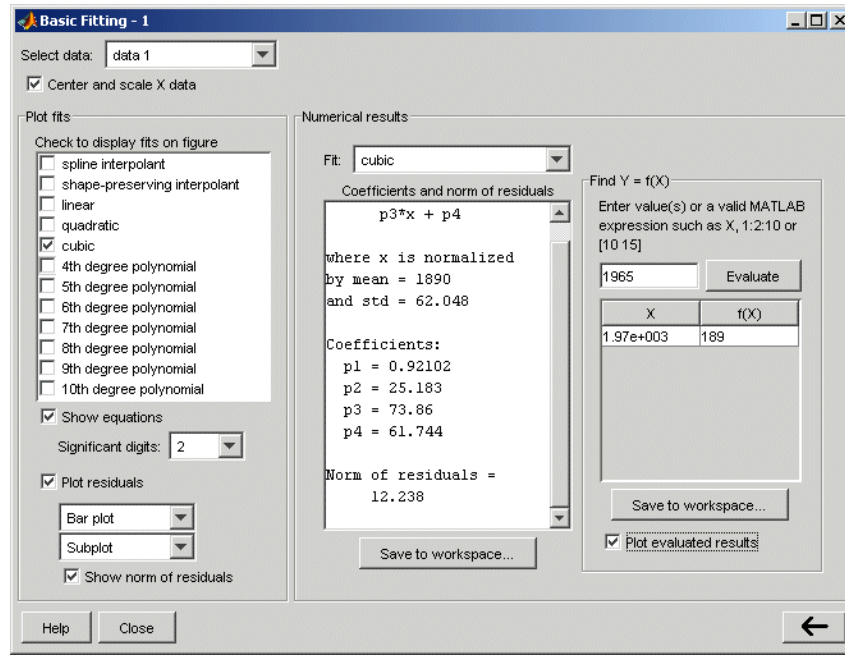
```
1965
```

Note Use unscaled and uncentered `x` values. You do not need to center and scale first, even though you selected to scale `x` values to obtain the coefficients in "Fitting Data" on page 3-13. Basic Fitting makes the necessary adjustments behind the scenes.

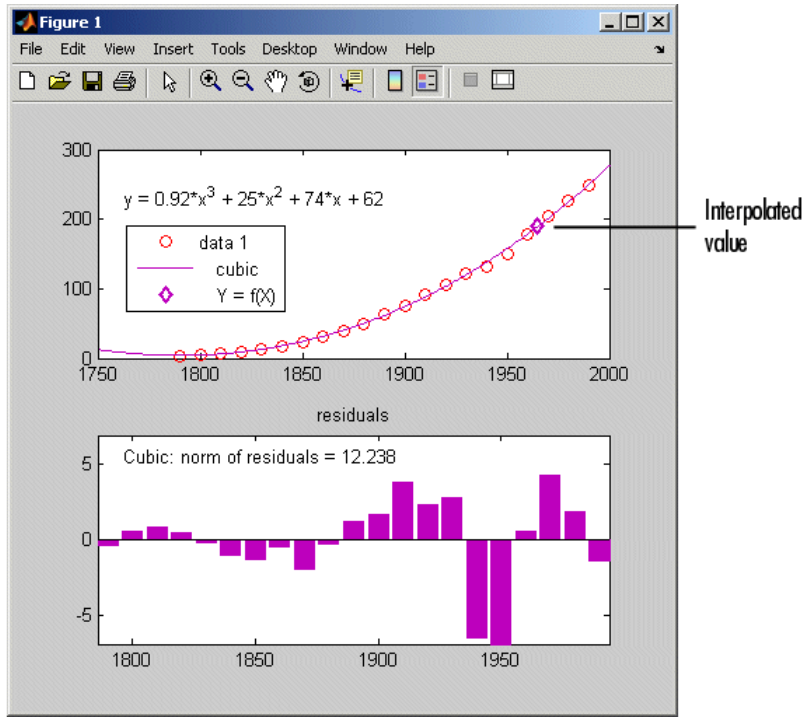
---

2 Click **Evaluate**.

The `x` values and the corresponding values for `f(x)` computed from the fit and displayed in a table, as shown below:

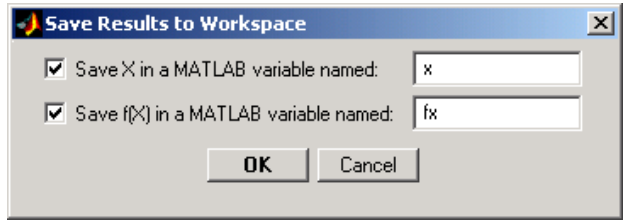


3 Select the **Plot evaluated results** check box to display the interpolated value:



4 Save the interpolated population in 1965 to the MATLAB workspace by clicking **Save to workspace**.

This opens the following dialog box, where you specify the variable names:



Generating an M-file

After completing a Basic Fitting session, you can generate an M-file that recomputes fits and reproduces plots with new data.

1. In the Figure window, select **File > Generate M-File**.

   This creates a function M-file and displays it in the MATLAB Editor. The code in the M-file shows you how to programmatically reproduce what you did interactively with the Basic Fitting dialog box.

2. Change the name of the function on the first line of the M-file from `createfigure` to something more specific, like `censusplot`. Save the file to your current folder with the file name `censusplot.m`

3. Generate some new, randomly perturbed census data:

   ```
   randpop = pop + 10*randn(size(pop));
   ```

4. Reproduce the plot with the new data and recompute the fit:

   ```
   censusplot(cdate,randpop,1965)
   ```

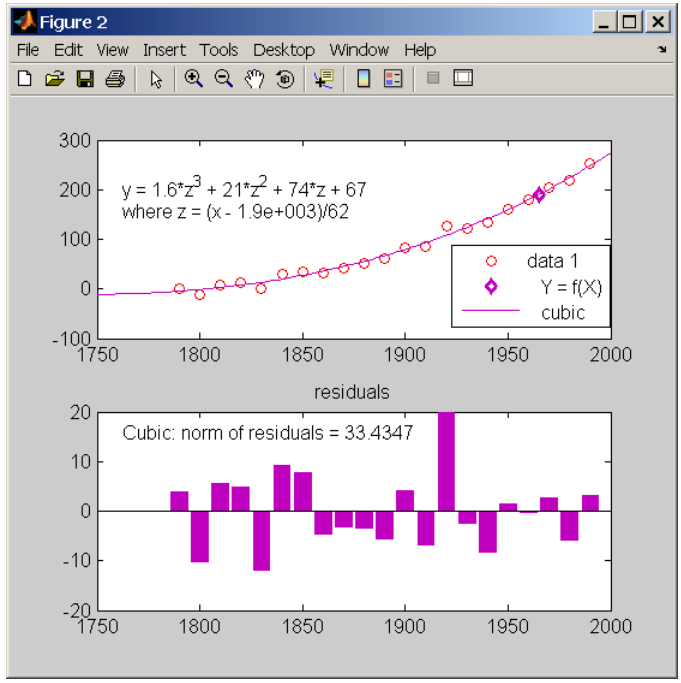


Figure 2
File Edit View Insert Tools Desktop Window Help
y = 1.6*z^3 + 21*z^2 + 74*z + 67
where z = (x - 1.9e+003)/62
data 1
Y = f(X)
cubic
300
200
100
0
-100
1750
1800
1850
1900
1950
2000
residuals
Cubic: norm of residuals = 33.4347
20
10
0
-10
-20

Programmatic Fitting

In this section...
"MATLAB Functions for Polynomial Models" on page 3-23
"Linear Model with Nonpolynomial Terms" on page 3-27
"Multiple Regression" on page 3-29
"Example: Programmatic Fitting" on page 3-30

MATLAB Functions for Polynomial Models

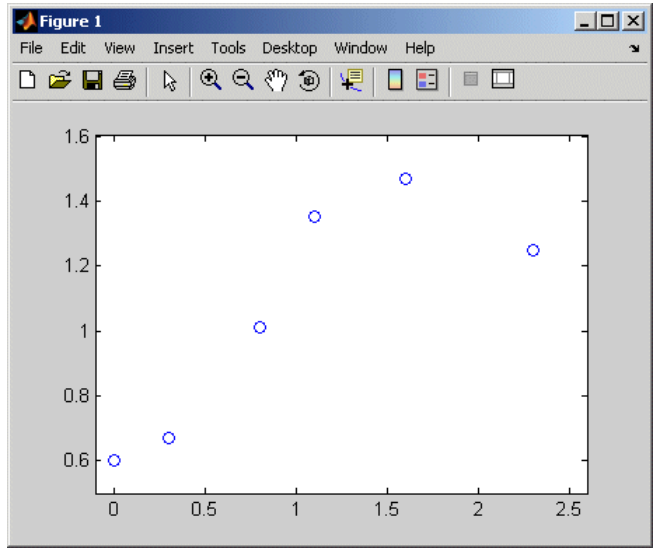
Two MATLAB functions can model your data with a polynomial.

Polynomial Fit Functions

Function	Description
`polyfit`	`polyfit(x,y,n)` finds the coefficients of a polynomial `p(x)` of degree `n` that fits the `y` data by minimizing the sum of the squares of the deviations of the data from the model (least-squares fit).
`polyval`	`polyval(p,x)` returns the value of a polynomial of degree `n` that was determined by `polyfit`, evaluated at `x`.

For example, suppose you measure a quantity `y` at several values of time `t`:

```
t = [0 0.3 0.8 1.1 1.6 2.3];
y = [0.6 0.67 1.01 1.35 1.47 1.25];
plot(t,y,'o')
```



Plot of y Versus t

You can try modeling this data using a second-degree polynomial function:

$$y = a_2 t^2 + a_1 t + a_0$$

The unknown coefficients a_0, a_1, and a_2 are computed by minimizing the sum of the squares of the deviations of the data from the model (least-squares fit).

To find the polynomial coefficients, type the following at the MATLAB prompt:

```
p=polyfit(t,y,2)
```

MATLAB calculates the polynomial coefficients in descending powers:

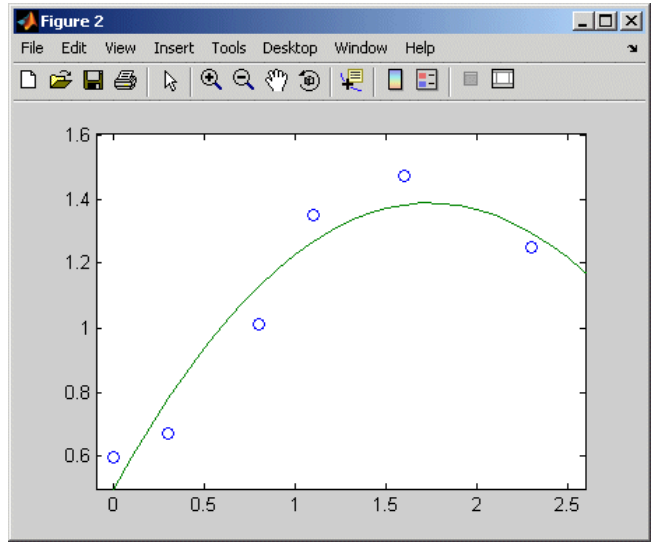
```
p =
   -0.2942    1.0231    0.4981
```

The second-degree polynomial model of the data is given by the following equation:

$$y = -0.2942 t^2 + 1.0231 t + 0.4981$$

To plot the model with the data, evaluate the polynomial at uniformly spaced times `t2` and overlay the original data on a plot:

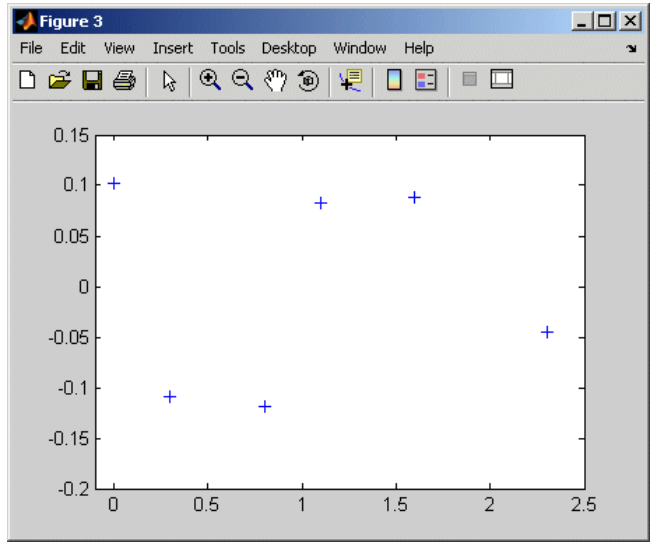
```
t2 = 0:0.1:2.8;     % Define a uniformly spaced time vector
y2=polyval(p,t2);   % Evaluate the polynomial at t2
figure
plot(t,y,'o',t2,y2) % Plot the fit on top of the data
                    % in a new Figure window
```



Plot of Data (Points) and Model (Line)

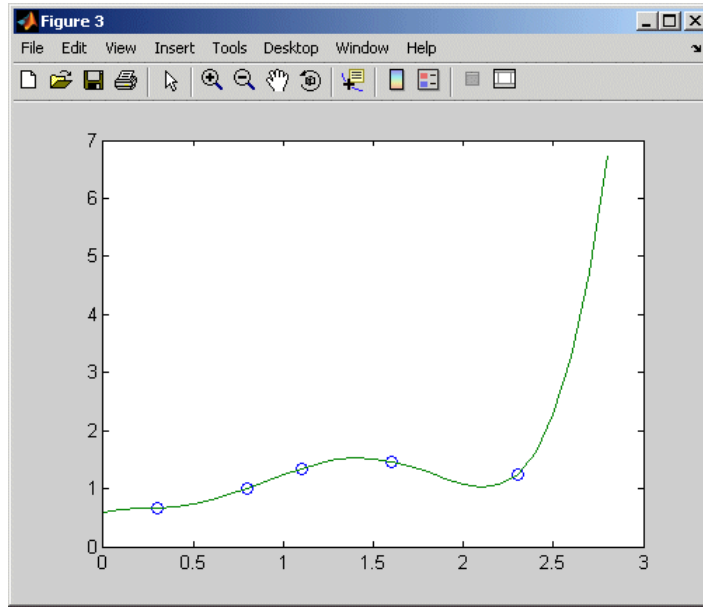
Use the following syntax to calculate the residuals:

```
y2=polyval(p,t); % Evaluate model at the data time vector
res=y-y2; % Calculate the residuals by subtracting
figure, plot(t,res,'+') % Plot the residuals
```



Plot of the Residuals

Notice that the second-degree fit roughly follows the basic shape of the data, but does not capture the smooth curve on which the data seems to lie. There appears to be a pattern in the residuals, which indicates that a different model might be necessary. A fifth-degree polynomial (shown next) does a better job of following the fluctuations in the data.



Fifth-Degree Polynomial Fit

Note If you are trying to model a physical situation, it is always important to consider whether a model of a specific order is meaningful in your situation.

Linear Model with Nonpolynomial Terms

When a polynomial function does not produce a satisfactory model of your data, you can try using a linear model with nonpolynomial terms. For example, consider the following function that is linear in the parameters a_0, a_1, and a_2, but nonlinear in the t data:

$$y = a_0 + a_1 e^{-t} + a_2 t e^{-t}$$

You can compute the unknown coefficients a_0, a_1, and a_2 by constructing and solving a set of simultaneous equations and solving for the parameters. The following syntax accomplishes this by forming a *design matrix*, where each column represents a variable used to predict the response (a term in the model) and each row corresponds to one observation of those variables:

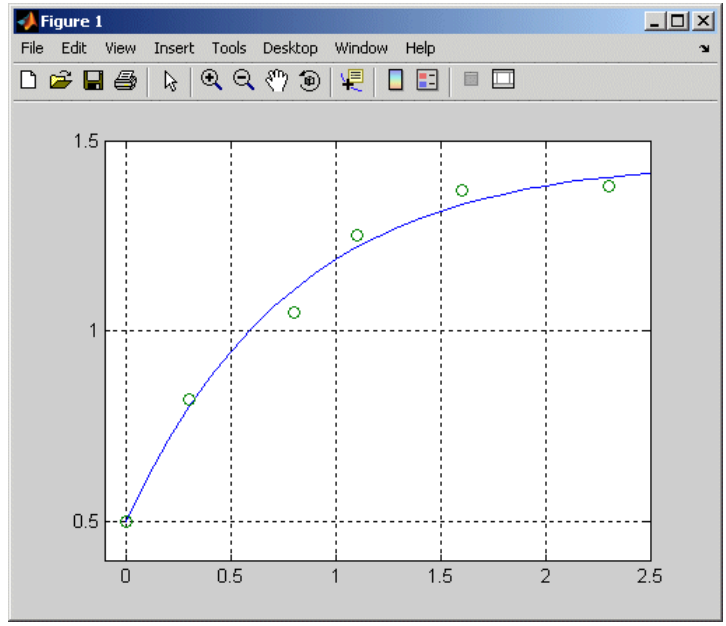
```
% Enter t and y as columnwise vectors
t = [0 0.3 0.8 1.1 1.6 2.3]';
y = [0.6 0.67 1.01 1.35 1.47 1.25]';

% Form the design matrix
X = [ones(size(t))  exp(-t)  t.*exp(-t)];

% Calculate model coefficients
a = X\y

a =
    1.3983
  - 0.8860
    0.3085
```

Therefore, the model of the data is given by

$$y = 1.3983 - 0.8860e^{-t} + 0.3085te^{-t}$$

Now evaluate the model at regularly spaced points and plot the model with the original data, as follows:

```
T = (0:0.1:2.5)';
Y = [ones(size(T))  exp(-T)  T.*exp(-T)]*a;
plot(T,Y,'-',t,y,'o'), grid on
```



Linear Fit with Nonpolynomial Terms

Multiple Regression

When y is a function of more than one predictor variable, the matrix equations that express the relationships among the variables must be expanded to accommodate the additional data. This is called *multiple regression*.

Suppose you measure a quantity y for several values of x_1 and x_2. Enter these variables in the MATLAB Command Window, as follows:

```
x1 = [.2 .5 .6 .8 1.0 1.1]';
x2 = [.1 .3 .4 .9 1.1 1.4]';
y  = [.17 .26 .28 .23 .27 .24]';
```

A model of this data is of the form

$$y = a_0 + a_1 x_1 + a_2 x_2$$

Multiple regression solves for unknown coefficients a_0, a_1, and a_2 by minimizing the sum of the squares of the deviations of the data from the model (least-squares fit).

Construct and solve the set of simultaneous equations by forming a design matrix, `X`, and solving for the parameters by using the backslash operator:

```
X = [ones(size(x1))  x1  x2];
a = X\y

a =
    0.1018
    0.4844
   -0.2847
```

The least-squares fit model of the data is

$$y = 0.1018 + 0.4844x_1 - 0.2847x_2$$

To validate the model, find the maximum of the absolute value of the deviation of the data from the model:

```
Y = X*a;
MaxErr = max(abs(Y - y))

MaxErr =
     0.0038
```

This value is much smaller than any of the data values, indicating that this model accurately follows the data.

Example: Programmatic Fitting

In this example, you use MATLAB functions to accomplish the following:

- "Calculating Correlation Coefficients" on page 3-32
- "Fitting a Polynomial to the Data" on page 3-32
- "Plot and Calculate Confidence Bounds" on page 3-34

This example uses the data in `census.mat`, which contains U.S. population data for the years 1790 to 1990.

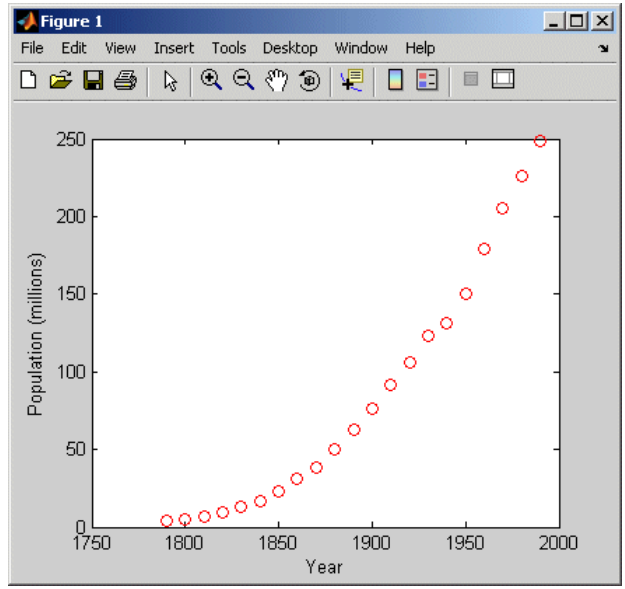
To load and plot the data, type the following commands at the MATLAB prompt:

```
load census
plot(cdate,pop,'ro')
```

This adds the following two variables to the MATLAB workspace:

- `cdate` is a column vector containing the years 1790 to 1990 in increments of 10.
- `pop` is a column vector with the U.S. population numbers corresponding to each year in `cdate`.

The following plot of the data shows a strong pattern, which indicates a high correlation between the variables.



U.S. Population from 1790 to 1990

Calculating Correlation Coefficients

In this portion of the example, you determine the statistical correlation between the variables `cdate` and `pop` to justify modeling the data. For more information about correlation coefficients, see “Linear Correlation” on page 3-2.

Type the following syntax at the MATLAB prompt:

```
corrcoef(cdate,pop)
```

MATLAB calculates the following correlation-coefficient matrix:

```
ans =

    1.0000    0.9597
    0.9597    1.0000
```

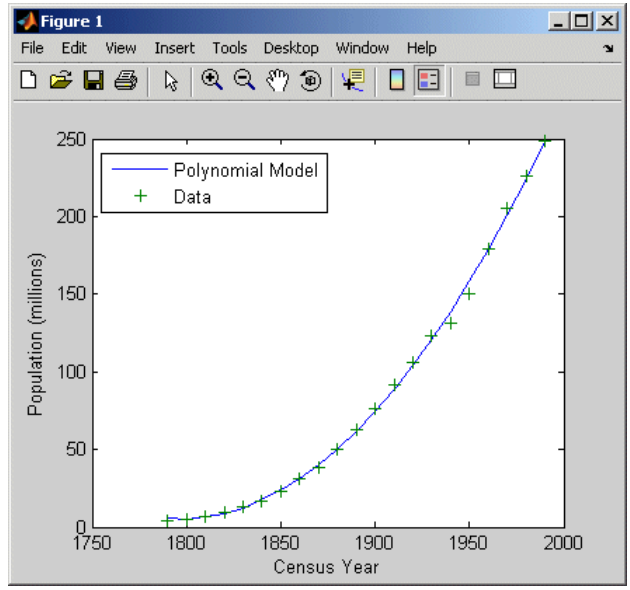
The diagonal matrix elements represent the perfect correlation of each variable with itself and are equal to 1. The off-diagonal elements are very close to 1, indicating that there is a strong statistical correlation between the variables `cdate` and `pop`.

Fitting a Polynomial to the Data

This portion of the example applies the `polyfit` and `polyval` MATLAB functions to model the data:

```
% Calculate fit parameters
[p,ErrorEst] = polyfit(cdate,pop,2);
% Evaluate the fit
pop_fit = polyval(p,cdate,ErrorEst);
% Plot the data and the fit
plot(cdate,pop_fit,'-',cdate,pop,'+');
% Annotate the plot
legend('Polynomial Model','Data');
xlabel('Census Year');
ylabel('Population (millions)');
```

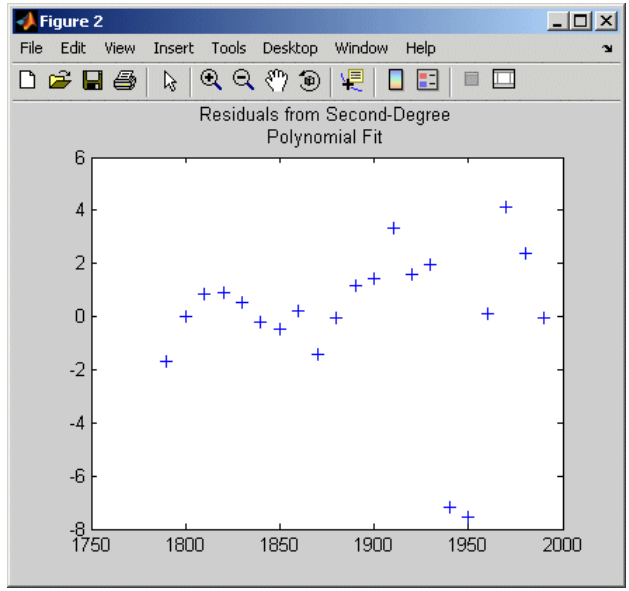
The following figure shows that the quadratic-polynomial fit provides a good approximation to the data:



Quadratic Polynomial Fit to the Census Data

To calculate the residuals for this fit, type the following syntax at the MATLAB prompt:

```
res = pop - pop_fit;
figure, plot(cdate,res,'+')
```



Residuals for the Quadratic Polynomial Model

Notice that the plot of the residuals exhibits a pattern, which indicates that a second-degree polynomial might not be appropriate for modeling this data.

Plot and Calculate Confidence Bounds

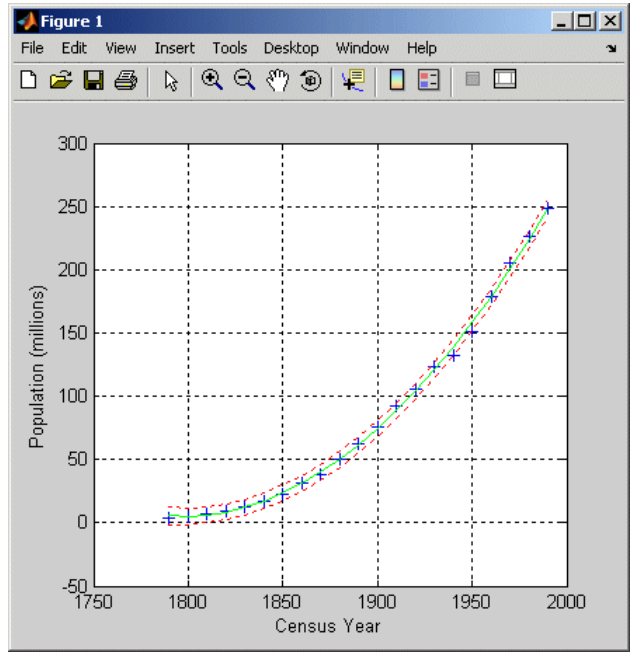
Confidence bounds are confidence intervals for a predicted response. The width of the interval indicates the degree of certainty of the fit.

This example applies `polyfit` and `polyval` to the `census` sample data to produce confidence bounds for a second-order polynomial model.

The following syntax uses an interval of $\pm 2\Delta$, which corresponds to a 95% confidence interval for large samples:

```
% Evaluate the fit and the prediction error estimate (delta)
[pop_fit,delta] = polyval(p,cdate,ErrorEst);
% Plot the data, the fit, and the confidence bounds
plot(cdate,pop,'+',...
     cdate,pop_fit,'g-',...
     cdate,pop_fit+2*delta,'r:',...
     cdate,pop_fit-2*delta,'r:');
% Annotate the plot
xlabel('Census Year');
ylabel('Population (millions)');
grid on
```

The 95% interval indicates that you have a 95% chance that a new observation will fall within the bounds.



Quadratic Polynomial Fit with Confidence Bounds

4

Time Series Analysis

- "Introduction" on page 4-2
- "Time Series Objects" on page 4-3
- "Time Series Tools" on page 4-39

Introduction

Time series are data vectors sampled over time, in order, often at regular intervals. They are distinguished from randomly sampled data, which form the basis of many other data analyses. Time series represent the time-evolution of a dynamic population or *process*. The linear ordering of time series gives them a distinctive place in data analysis, with a specialized set of techniques.

Time series analysis is concerned with:

- Identifying patterns
- Modeling patterns
- Forecasting values

Several dedicated MATLAB functions perform time series analysis. This section introduces objects and interactive tools for time series analysis.

Time Series Objects

In this section...

"Introduction" on page 4-3
"Time Series Data Sample" on page 4-4
"Example: Time Series Objects and Methods" on page 4-6
"Time Series Constructor" on page 4-21
"Time Series Methods" on page 4-29
"Time Series Collection Constructor" on page 4-33
"Time Series Collection Methods" on page 4-37

Introduction

MATLAB time series objects are of two types:

- `timeseries` — Stores data and time values, as well as the metadata information that includes units, events, data quality, and interpolation method
- `tscollection` — Stores a collection of `timeseries` objects that share a common time vector, convenient for performing operations on synchronized time series with different units

This section discusses the following topics:

- Using time series constructors to instantiate time series classes
- Modifying object properties using `set` methods or dot notation
- Calling time series functions and methods

To get a quick overview of programming with `timeseries` and `tscollection` objects, follow the steps in "Example: Time Series Objects and Methods" on page 4-6.

If you prefer to work with a graphical user interface (GUI), use MATLAB Time Series Tools to work with time series data. For more information about Time Series Tools, see "Example: Time Series Tools" on page 4-77.

Note If you are new to programming with `timeseries` and `tscollection` objects, you might want to start by working with Time Series Tools and enabling the **Record M-Code** feature. This generates reusable M-code based on the operations you perform in the GUI. For more information, see "Generating Reusable M-Code" on page 4-43.

Time Series Data Sample

To properly understand the description of `timeseries` object properties and methods in this documentation, it is important to clarify some terms related to storing data in a `timeseries` object—the difference between a *data value* and a *data sample*.

A *data value* is a single, scalar value recorded at a specific time. A *data sample* consists of one or more values associated with a specific time in the `timeseries` object. The number of data samples in a time series is the same as the length of the time vector.

For example, consider data that consists of three sensor signals: two signals represent the position of an object in meters, and the third represents its velocity in meters/second.

To enter the data matrix, type the following at the MATLAB prompt:

```
x = [-0.2 -0.3 13;
     -0.1 -0.4 15;
      NaN  2.8 17;
      0.5 0.3 NaN;
     -0.3 -0.1 15]
```

The NaN value represents a missing data value. MATLAB displays the following 5-by-3 matrix:

```
x=
   -0.2000   -0.3000   13.0000
   -0.1000   -0.4000   15.0000
    NaN       2.8000   17.0000
    0.5000    0.3000       NaN
   -0.3000   -0.1000   15.0000
```

The first two columns of x contain quantities with the same units and you can create a multivariate timeseries object to store these two time series. For more information about creating timeseries objects, see "Time Series Constructor Syntax" on page 4-23. The following command creates a timeseries object ts_pos to store the position values:

```
ts_pos = timeseries(x(:,1:2), 1:5, 'name', 'Position')
```

MATLAB responds by displaying the following properties of ts_pos:

```
Time Series Object: Position

Time vector characteristics

      Length                5
      Start time            1 seconds
      End time              5 seconds

Data characteristics

      Interpolation method  linear
      Size                  [5  2]
      Data type             double
```

The `Length` of the time vector, which is `5` in this example, equals the number of data samples in the `timeseries` object. Find the size of the data sample in `ts_pos` by typing the following at the MATLAB prompt:

```
getdatasamplesize(ts_pos)

ans =

     1     2
```

Similarly, you can create a second `timeseries` object to store the velocity data:

```
ts_vel = timeseries(x(:,3), 1:5, 'name', 'Velocity');
```

Find the size of each data sample in `ts_vel` by typing the following:

```
getdatasamplesize(ts_vel)

ans =

     1     1
```

Notice that `ts_vel` has one data value in each data sample and `ts_pos` has two data values in each data sample.

Note In general, when the time series data is an *M*-by-*N*-by-*P*-by-... multidimensional array with *M* samples, the size of each data sample is *N*-by-*P*-by-... .

If you want to perform operations on the `ts_pos` and `ts_vel` `timeseries` objects while keeping them synchronized, group them in a time series collection. For more information, see "Time Series Collection Constructor Syntax" on page 4-34.

Example: Time Series Objects and Methods

- "Creating Time Series Objects" on page 4-7
- "Viewing Time Series Objects" on page 4-9

- "Modifying Time Series Units and Interpolation Method" on page 4-12
- "Defining Events" on page 4-13
- "Creating Time Series Collection Objects" on page 4-14
- "Resampling a Time Series Collection Object" on page 4-15
- "Adding a Data Sample to a Time Series Collection Object" on page 4-16
- "Removing and Interpolating Missing Data" on page 4-17
- "Removing a Time Series from a Time Series Collection" on page 4-19
- "Changing a Numerical Time Vector to Date Strings" on page 4-19
- "Plotting Time Series Collection Members" on page 4-20

Creating Time Series Objects

This portion of the example illustrates how to create several `timeseries` objects from an array. For more information about the `timeseries` object, see "Time Series Constructor" on page 4-21.

The sample data provided with this example consists of a 24-by-3 matrix of double values, where each column represents the hourly traffic counts at three town intersections.

This adds the variable `count` to the MATLAB workspace:

```
%% Import the sample data
load count.dat
```

To view the `count` matrix, type

```
count
```

MATLAB displays the following 24-by-3 matrix:

```
    11    11     9
     7    13    11
    14    17    20
    11    13     9
    43    51    69
    38    46    76
    61   132   186
    75   135   180
    38    88   115
    28    36    55
    12    12    14
    18    27    30
    18    19    29
    17    15    18
    19    36    48
    32    47    10
    42    65    92
    57    66   151
    44    55    90
   114   145   257
    35    58    68
    11    12    15
    13     9    15
    10     9     7
```

Create three `timeseries` objects to store the data collected at each intersection:

```
count1 = timeseries(count(:,1), 1:24,'name', 'intersection1');
count2 = timeseries(count(:,2), 1:24,'name', 'intersection2');
count3 = timeseries(count(:,3), 1:24,'name', 'intersection3');
```

Note In the above construction, `timeseries` objects have both a variable name (e.g., `count1`) and an internal object name (e.g., `intersection1`). The variable name is used with MATLAB functions. The object name is a property of the object, accessed with object methods. For more information on `timeseries` object properties and methods, see "Time Series Properties" on page 4-24 and "Time Series Methods" on page 4-29.

Each time series has a time vector in units of seconds, starting at 1 second and increasing up to 24 seconds in 1-second increments. The software assumes this increment when you do not explicitly specify one. You will change the time units to hours in "Modifying Time Series Units and Interpolation Method" on page 4-12.

Note If you want to create a `timeseries` object that groups the three data columns in `count`, use the following syntax:

```
count_ts = timeseries(count, 1:24,'name','traffic_counts')
```

This is useful when all time series have the same units and you want to keep them synchronized during calculations.

Viewing Time Series Objects

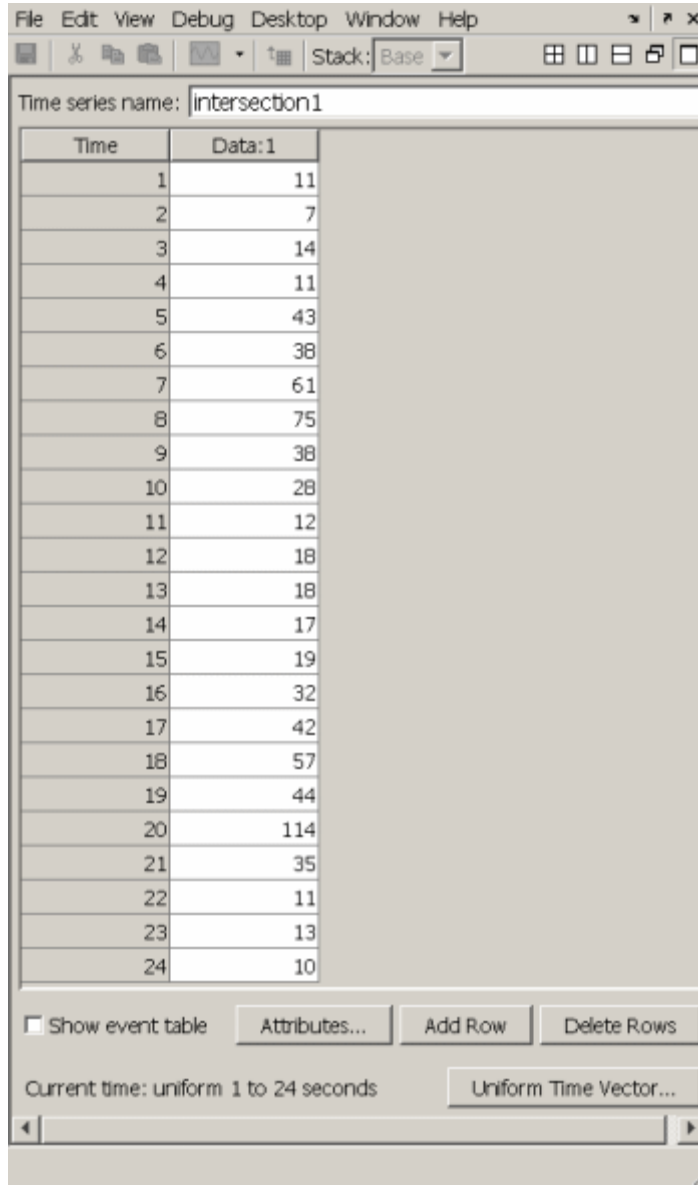
After creating a `timeseries` object, as described in "Creating Time Series Objects" on page 4-7, you can view it in either the Variable Editor or "Time Series Tools" on page 4-39.

To view a `timeseries` object like `count1` in the Variable Editor, use any one of several methods:

- Type `open('count1')` at the command prompt.
- Select `count1` in the Workspace Browser and click the **Open selection** button .
- Double-click `count1` in the Workspace Browser.
- Right-click `count1` in the Workspace Browser and select **Open selection** from the context menu.

To view `count1` in Time Series Tools, right-click `count1` in the Workspace Browser and choose **Open in Time Series Tools** from the context menu.

When a `timeseries` object is opened in either the Variable Editor or Time Series Tools, it is displayed with the Time Series Editor:



For information on using the Time Series Editor, see “Editing Data and Time” on page 4-69.

Modifying Time Series Units and Interpolation Method

After creating a `timeseries` object, as described in "Creating Time Series Objects" on page 4-7, you can modify its units and interpolation method using dot notation.

To view the current properties of `count1`, type

```
get(count1)
```

MATLAB responds by displaying the current property values of the `count1` `timeseries` object:

```
Events: []
Name: 'intersection1'
Data: [24x1 double]
DataInfo: [1x1 tsdata.datametadata]
Time: [24x1 double]
TimeInfo: [1x1 tsdata.timemetadata]
Quality: []
QualityInfo: [1x1 tsdata.qualmetadata]
IsTimeFirst: true
TreatNaNasMissing: true
```

To view the current `DataInfo` properties, use dot notation:

```
count1.DataInfo
```

Change the data units and the default interpolation method for `count1`, as follows:

```
count1.DataInfo.Units = 'cars';
        % Specify new data units
count1.DataInfo.Interpolation = tsdata.interpolation('zoh');
        % Set the interpolation method to zero-order hold
```

To verify that the `DataInfo` properties have been modified, type

```
count1.DataInfo
```

MATLAB confirms the change by displaying

```
Time Series Data Meta Data Object
      Unit                cars
      Interpolation Method  zoh
```

Modify the time units to be `'hours'` for the three time series:

```
count1.TimeInfo.Units = 'hours';
count2.TimeInfo.Units = 'hours';
count3.TimeInfo.Units = 'hours';
```

Defining Events

This portion of the example illustrates how to define events for a `timeseries` object by using the `tsdata.event` auxiliary object. Events mark the data at specific times. When you plot the data, event markers are displayed on the plot. Events also provide a convenient way to synchronize multiple time series.

Use the following syntax to add two events to the data that mark the times of the AM commute and PM commute:

```
%% Construct and add the first event to all time series
e1 = tsdata.event('AMCommute',8);
                        % Construct the first event at 8 AM
e1.Units = 'hours';     % Specify the time units of the time
count1 = addevent(count1,e1);  % Add the event to count1
count2 = addevent(count2,e1);  % Add the event to count2
count3 = addevent(count3,e1);  % Add the event to count3
%% Construct and add the second event to all time series
e2 = tsdata.event('PMCommute',18);
                        % Construct the first event at 6 PM
e2.Units = 'hours';     % Specify the time units of the time
count1 = addevent(count1,e2);  % Add the event to count1
count2 = addevent(count2,e2);  % Add the event to count2
count3 = addevent(count3,e2);  % Add the event to count3
```

Creating Time Series Collection Objects

This portion of the example illustrates how to create a `tscollection` object. Each individual time series in a collection is called a *member*. For more information about the `tscollection` object, see "Time Series Collection Constructor" on page 4-33.

Note Typically, you use the `tscollection` object to group synchronized time series that have different units. In this simple example, all time series have the same units and the `tscollection` object does not provide an advantage over grouping the three time series in a single `timeseries` object. For an example of how to group several time series in one `timeseries` object, see "Creating Time Series Objects" on page 4-7.

Use the following syntax to create a `tscollection` object named `count_coll` and use the constructor syntax to immediately add two of the three time series currently in the MATLAB workspace (you will add the third time series later):

```
tsc = tscollection({count1 count2},'name', 'count_coll')
```

MATLAB responds with

```
Time Series Collection Object: count_coll
Time vector characteristics
Start time            1 hours
End time              24 hours
Member Time Series Objects:
      intersection1
      intersection2
```

Note The time vectors of the `timeseries` objects you are adding to the `tscollection` must match.

Notice that the `Name` property of the `timeseries` objects is used to name the collection members as `intersection1` and `intersection2`.

Add the third `timeseries` object in the workspace to the `tscollection` by using the following syntax:

```
tsc = addts(tsc, count3)
```

All three members in the collection are listed:

```
Time Series Collection Object: count_coll
Time vector characteristics
Start time            1 hours
End time              24 hours
Member Time Series Objects:
      intersection1
      intersection2
      intersection3
```

Resampling a Time Series Collection Object

This portion of the example illustrates how to resample each member in a `tscollection` using a new time vector. The resampling operation is used to either select existing data at specific time values, or to interpolate data at finer intervals. If the new time vector contains time values that did not exist in the previous time vector, the new data values are calculated using the default interpolation method you associated with the time series.

To resample the time series to include data values every 2 hours instead of every hour and save it as a new `tscollection` object, enter the following syntax:

```
tsc1 = resample(tsc,1:2:24)
```

In some cases you might need a finer sampling of information than you currently have and it is reasonable to obtain it by interpolating data values. For example, the following syntax interpolates values at each half-hour mark:

```
tsc1 = resample(tsc,1:0.5:24)
```

To add values at each half-hour mark, the default interpolation method of a time series is used. For example, the new data points in `intersection1` are calculated by using the zero-order hold interpolation method, which

holds the value of the previous sample constant. You set the interpolation method for `intersection1` as described in "Modifying Time Series Units and Interpolation Method" on page 4-12.

The new data points in `intersection2` and `intersection3` are calculated using linear interpolation, which is the default method.

Adding a Data Sample to a Time Series Collection Object

This portion of the example illustrates how to add a data sample to a `tscollection`.

You can use the following syntax to add a data sample to the `intersection1` collection member at 3.25 hours (i.e., 15 minutes after the hour):

```
tsc1 = addsampletocollection(tsc1,'time',3.25,...
       'intersection1',5)
```

There are three members in the `tsc1` collection, and adding a data sample to one member adds a data sample to the other two members at 3.25 hours. However, because you did not specify the data values for `intersection2` and `intersection3` in the new sample, the missing values are represented by `NaN`s for these members. To learn how to remove or interpolate missing data values, see "Removing Missing Data" on page 4-17 and "Interpolating Missing Data" on page 4-18.

`tsc1` Data from 2.0 to 3.5 Hours

Hours	Intersection 1	Intersection 2	Intersection 3
2.0	7	13	11
2.5	7	15	15.5
3.0	14	17	20
3.25	5	NaN	NaN
3.5	14	15	14.5

To view all `intersection1` data (including the new sample at 3.25 hours), type

```
tsc1.intersection1
```

Similarly, to view all `intersection2` data (including the new sample at 3.25 hours containing a `NaN` value), type

```
tsc1.intersection2
```

Removing and Interpolating Missing Data

Missing data in a time series are represented by `NaN`s. This portion of the example illustrates how to either remove the missing data or interpolate it by using the interpolation method you specified for that time series. In "Adding a Data Sample to a Time Series Collection Object" on page 4-16, you added a new data sample to the `tsc1` collection at 3.25 hours.

There are three members in the `tsc1` collection, and adding a data sample to one member adds a data sample to the other two members at 3.25 hours. However, because you did not specify the data values for the `intersection2` and `intersection3` members at 3.25 hours, they currently contain missing values that are represented by `NaN`s.

Removing Missing Data. You can use the following syntax to find and remove the data samples containing `NaN` values in the `tsc1` collection:

```
tsc1 = delsamplefromcollection(tsc1,'index',...
       find(isnan(tsc1.intersection2.Data)));
```

This command searches one `tscollection` member at a time—in this case, `intersection2`. When a missing value is located in `intersection2`, the data at that time is removed from *all* members of the `tscollection`.

Note You can use the following dot-notation syntax to access the `Data` property of the `intersection2` member in the `tsc1` collection:

```
tsc1.intersection2.Data
```

For a complete list of `timeseries` properties, see "Time Series Properties" on page 4-24.

Interpolating Missing Data. For the sake of this example, you must reintroduce `NaN` values in `intersection2` and `intersection3` (which you removed):

```
tsc1 = addsampletocollection(tsc1,'time',3.25,...
       'intersection1',5);
```

To interpolate the missing values in `tsc1` using the current time vector (`tsc1.Time`), type the following syntax:

```
tsc1 = resample(tsc1,tsc1.Time)
```

This replaces the `NaN` values in `intersection2` and `intersection3` by using linear interpolation—the default interpolation method for these time series.

Note Dot notation `tsc1.Time` is used to access the `Time` property of the `tsc1` collection. For a complete list of `tscollection` properties, see "Time Series Collection Properties" on page 4-35.

To view `intersection2` data after interpolation, for example, type

```
tsc1.intersection2
```

New tsc1 Data from 2.0 to 3.5 Hours

Hours	Intersection 1	Intersection 2	Intersection 3
2.0	7	13	11
2.5	7	15	15.5

New tsc1 Data from 2.0 to 3.5 Hours (Continued)

Hours	Intersection 1	Intersection 2	Intersection 3
3.0	14	17	20
3.25	5	16	17.3
3.5	14	15	14.5

Removing a Time Series from a Time Series Collection

To remove the `intersection3` time series from the `tscollection` object `tsc1`, type:

```
tsc1 = removets(tsc1,'intersection3')
```

Two time series as members in the collection are now listed:

```
Time Series Collection Object: count_coll
Time vector characteristics
Start time            1 hours
End time              24 hours
Member Time Series Objects:
      intersection1
      intersection2
```

Changing a Numerical Time Vector to Date Strings

This portion of the example illustrates how to convert the display format of a numerical time vector to MATLAB date strings. For a complete list of the MATLAB date-string formats supported for `timeseries` and `tscollection` objects, see "Time Vector Format" on page 4-22.

To convert a numerical time vector to date strings, you must set the `StartDate` field of the `TimeInfo` property. All values in the time vector are converted to date strings using `StartDate` as a reference date.

For example, suppose the reference date occurs on December 25, 2004:

```
tsc1.TimeInfo.StartDate = 'DEC-25-2004 00:00:00';
```

To verify that the time vector now uses date strings, type the following command to look at the sixth element of the `intersection2` member:

```
tsc1.intersection2(6)
```

MATLAB responds with

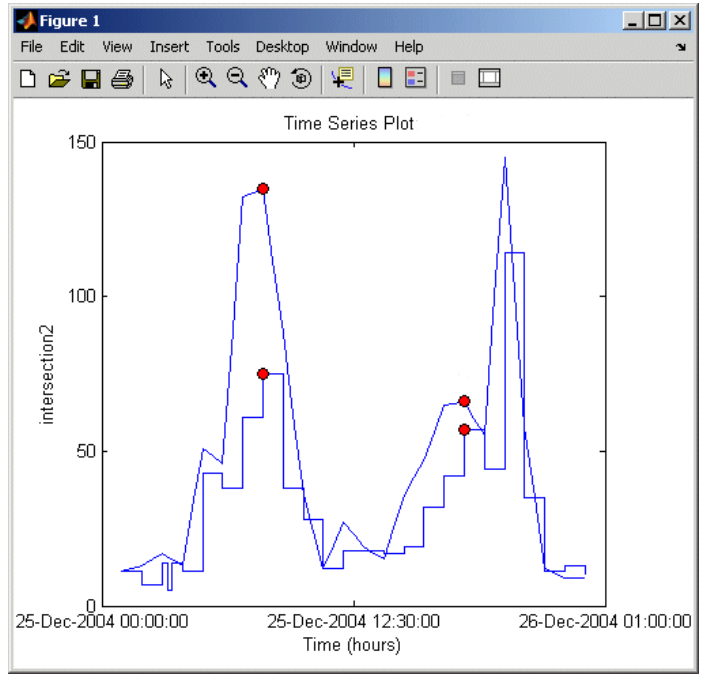
```
Time Series Object: unnamed
Time vector characteristics
      Length                1
      Start date            25-Dec-2004 03:15:00
      End date              25-Dec-2004 03:15:00
Data characteristics
      Interpolation method  linear
      Size                  [1  1]
      Data type             double
Time                              Data                  Quality
------------------------------------------------------------
25-Dec-2004 03:15:00              16
```

This result shows that the sixth element of `intersection2` has an interpolated data value of 16 cars at 3.25 hours (or 3:15:00).

Plotting Time Series Collection Members

You can plot the two remaining members in the `tsc1` collection by using the following command sequence:

```
plot(tsc1.intersection1); hold on;
plot(tsc1.intersection2)
```



Time Plot of Two Time Series in a Collection

This plot shows the two time series in the collection: `intersection1` and `intesection2`. `intersection1` uses the zero-order hold interpolation method and therefore has a jagged curve. In contrast, `intersection2` uses a linear interpolation method. The vertical axis is labeled as `intersection2` because this was the last time series plotted.

The filled circles on the plot indicate events, as specified in “Defining Events” on page 4-73.

Time Series Constructor

- “Time Vector Format” on page 4-22
- “Time Series Constructor Syntax” on page 4-23
- “Time Series Properties” on page 4-24

Time Vector Format

You can specify the time vector of the `timeseries` object either as numerical (double) values or as valid MATLAB date strings.

When the `timeseries` `TimeInfo.StartDate` property is empty, the numerical `Time` values are measured relative to 0 (or another numerical value) in specified units. In this case, the time vector is described as *relative* (that is, it contains time values that are not associated with a specific start date).

When `TimeInfo.StartDate` is nonempty, the time values are date strings measured relative to `StartDate` in specified units. In this case, the time vector is described as *absolute* (that is, it contains time values that are associated with a specific calendar date). For more information, see "Time Series Properties" on page 4-24.

MATLAB supports the following date-string formats for time series applications.

Date-String Format	Usage Example
`dd-mmm-yyyy HH:MM:SS`	`01-Mar-2000 15:45:17`
`dd-mmm-yyyy`	`01-Mar-2000`
`mm/dd/yy`	`03/01/00`
`mm/dd`	`03/01`
`HH:MM:SS`	`15:45:17`
`HH:MM:SS PM`	`3:45:17 PM`
`HH:MM`	`15:45`
`HH:MM PM`	`3:45 PM`
`mmm.dd,yyyy HH:MM:SS`	`Mar.01,2000 15:45:17`
`mmm.dd,yyyy`	`Mar.01,2000`
`mm/dd/yyyy`	`03/01/2000`

Time Series Constructor Syntax

Before implementing the various MATLAB functions and methods specifically designed to handle time series data, you must create a `timeseries` object to store the data.

The following table summarizes the syntax when using the `timeseries` constructor. For an example of using the constructor, see "Creating Time Series Objects" on page 4-7.

Time Series Syntax Descriptions

Syntax	Description
`ts = timeseries`	Creates an empty `timeseries` object. The size of this object is 0-by-1.
`ts = timeseries(Data)`	Creates a `timeseries` object with the specified `Data`. `ts` has a default time vector ranging from `0` to `N-1` with 1-second increments, where `N` is the number of samples. The default name of the `timeseries` object is `'unnamed'`.
`ts = timeseries('Name')`	Creates an empty `timeseries` object with the name specified by a string `Name`. This name can differ from the `timeseries` variable name.
`ts = timeseries(Data,Time)`	Creates a `timeseries` object with the specified `Data` array and `Time`. When time values are date strings, you must specify `Time` as a cell array of date strings.

Time Series Syntax Descriptions (Continued)

Syntax	Description
`ts = timeseries(Data,Time,Quality)`	The `Quality` attribute is an integer vector containing values `-128` to `127` that specifies the quality in terms of codes defined by `QualityInfo.Code`. For more information about `QualityInfo`, see "Time Series Properties" on page 4-24.
`ts = timeseries(Data,..., 'Parameter',Value,...)`	Optionally enter the following parameter-value pairs after the `Data`, `Time`, and `Quality` arguments. You can specify the following parameters: • `Name` • `IsTimeFirst` `Name` and `IsTimeFirst` are described in "Time Series Properties" on page 4-24.

Time Series Properties

The following table lists the properties of the `timeseries` object. You can specify the `Data`, `IsTimeFirst`, `Name`, `Quality`, and `Time` properties as input arguments in the constructor. To assign other properties, use the `set` function or dot notation.

Note To get property information from the command line, type `help timeseries/tsprops` at the MATLAB prompt.

For an example of editing `timeseries` object properties, see "Modifying Time Series Units and Interpolation Method" on page 4-12.

Time Series Property Descriptions

Property	Description
`Data`	Time series data, where each data sample corresponds to a specific time. The data can be a scalar, a vector, or a multidimensional array. Either the first or last dimension of the data must align with `Time`. By default, `NaN`s represent missing or unspecified data. Set the `TreatNaNasMissing` property to determine how missing data is treated in calculations.
`DataInfo`	Contains fields for storing contextual information about `Data`: • `Unit` — String that specifies data units. • `Interpolation` — A `tsdata.interpolation` object that specifies the interpolation method for this time series. Fields in the `tsdata.interpolation` object include: - `Fhandle`: Function handle to a user-defined interpolation function. - `Name`: String that specifies the name of the interpolation method. Predefined interpolation methods include `'linear'` and `'zoh'` (zero-order hold). `'linear'` is the default. • `UserData` — Any user-defined information entered as a string.

Time Series Property Descriptions (Continued)

Property	Description
`Events`	An array of `tsdata.event` objects that stores event information for this `timeseries` object. You add events using the `addevent` method. Fields in the `tsdata.event` object include the following: • `EventData` — Any user-defined information about the event • `Name` — String that specifies the name of the event • `Time` — Time value when this event occurs, specified as a real number or a date string relative to `StartDate` • `Units` — Time units • `StartDate` — A reference date specified in MATLAB date string format. `StartDate` is empty when you have a numerical time vector.
`IsTimeFirst`	Logical value (`true` or `false`) that specifies whether the first or last dimension of the `Data` array aligns with the time vector. You can set this property when the `Data` array is square and it is ambiguous which dimension aligns with time. By default, the first `Data` dimension that matches the length of the time vector is aligned with `Time`. When you set this property to • `true`, the first dimension of the data array is aligned with the time vector. • `false`, the last dimension of the data array is aligned with the time vector.

Time Series Property Descriptions (Continued)

Property	Description
	After a time series is created, this property is read-only.
`Name`	`timeseries` object name entered as a string. This name can differ from the name of the `timeseries` variable in the MATLAB workspace.
`Quality`	An integer vector or array containing values `-128` to `127` that specifies the quality in terms of codes defined by the `QualityInfo.Code` field. When `Quality` is a vector, it must have the same length as the time vector. In this case, each `Quality` value applies to the corresponding data sample. When `Quality` is an array, it must have the same size as the data array. In this case, each `Quality` value applies to the corresponding value of the data array.
`QualityInfo`	Provides a lookup table that converts numerical `Quality` codes to readable descriptions. `QualityInfo` fields include the following: • `Code` — Integer vector containing values `-128` to `127` that defines the "dictionary" of quality codes, which you can assign to each `Data` value by using the `Quality` property • `Description` — Cell vector of strings, where each element provides a readable description of the associated quality `Code` • `UserData` — Stores any additional user-defined information The length of `Code` and `Description` must match.

Time Series Property Descriptions (Continued)

Property	Description
`Time`	Vector of time values. When `TimeInfo.StartDate` is empty, the numerical `Time` values are measured relative to `0` in specified units. When `TimeInfo.StartDate` is defined, the time values are date strings measured relative to `StartDate` in specified units. The length of `Time` must match either the first or the last dimension of `Data`.
`TimeInfo`	Uses the following fields to store contextual information about `Time`: • `Units` — Time units with the following values: `'weeks'`, `'days'`, `'hours'`, `'minutes'`, `'seconds'`, `'milliseconds'`, `'microseconds'`, and `'nanoseconds'` • `Start` — Start time • `End` — End time (read-only) • `Increment` — Interval between two subsequent time values • `Length` — Length of the time vector (read-only) • `Format` — String defining the date string display format. See the MATLAB `datestr` function reference page for more information. • `StartDate` — Date string defining the reference date. See the MATLAB `setabstime (timeseries)` function reference page for more information. • `UserData` — Stores any additional user-defined information
`TreatNaNasMissing`	Logical value that specifies how to treat `NaN` values in `Data`:

Time Series Property Descriptions (Continued)

Property	Description
	• `true` — (Default) Treat all `NaN` values as missing data except during statistical calculations. • `false` — Include `NaN` values in statistical calculations, in which case `NaN` values are propagated to the result.

Time Series Methods

- "General Methods" on page 4-29
- "Data and Time Manipulation Methods" on page 4-30
- "Event Methods" on page 4-31
- "Arithmetic Operation Methods" on page 4-32
- "Statistical Methods" on page 4-33

General Methods

Use the following methods to query and set object properties, and plot the data.

Methods for Querying Properties

Method	Description
`get (timeseries)`	Query `timeseries` object property values.
`getdatasamplesize`	Return the size of each data sample in a `timeseries` object.
`getqualitydesc`	Return data quality descriptions based on the `Quality` property values assigned to a `timeseries` object.

Methods for Querying Properties (Continued)

Method	Description
`isempty (timeseries)`	Evaluate to `true` for an empty `timeseries` object.
`length (timeseries)`	Return the length of the time vector.
`plot (timeseries)`	Plot the `timeseries` object.
`set (timeseries)`	Set `timeseries` property values.
`size (timeseries)`	Return the `size` property of a `timeseries` object.
`tstool`	Open the Time Series Tools GUI.

Data and Time Manipulation Methods

Use the following methods to add or delete data samples, and manipulate the `timeseries` object.

Methods for Manipulating Data and Time

Method	Description
`addsample`	Add a data sample to a `timeseries` object.
`ctranspose (timeseries)`	Transpose a `timeseries` object.
`delsample`	Delete a sample from a `timeseries` object.
`detrend (timeseries)`	Subtract the mean or best-fit line and remove all `NaN`s from time series data.
`filter (timeseries)`	Shape frequency content of time series data using a 1-D digital filter.
`getabstime (timeseries)`	Extract a date-string time vector from a `timeseries` object into a cell array.
`getinterpmethod`	Get the interpolation method for a `timeseries` object.

Methods for Manipulating Data and Time (Continued)

Method	Description
`getsampleusingtime (timeseries)`	Extract specified data samples from an existing `timeseries` object into a new `timeseries` object.
`idealfilter (timeseries)`	Apply an ideal pass or notch (noncausal) filter to a `timeseries` object.
`resample (timeseries)`	Select or interpolate data in a `timeseries` object using a new time vector.
`setabstime (timeseries)`	Set the time values in the time vector as date strings.
`setinterpmethod`	Set interpolation method for a `timeseries` object.
`synchronize`	Synchronize and resample two `timeseries` objects using a common time vector.
`transpose (timeseries)`	Transpose a `timeseries` object.
`vertcat (timeseries)`	Vertical concatenation for `timeseries` objects.

Event Methods

To construct an event object, use the constructor `tsdata.event`. For an example of defining events for a time series, see "Defining Events" on page 4-73.

Methods That Define and Use Events

Method	Description
`addevent`	Add one or more events to a `timeseries` object.
`delevent`	Delete one or more events from a `timeseries` object.

Methods That Define and Use Events (Continued)

Method	Description
`gettsafteratevent`	Create a new `timeseries` object by extracting the samples from an existing time series that occur after or at a specified event.
`gettsafterevent`	Create a new `timeseries` object by extracting the samples that occur after a specified event from an existing time series.
`gettsatevent`	Create a new `timeseries` object by extracting the samples that occur at the same time as a specified event from an existing time series.
`gettsbeforeatevent`	Create a new `timeseries` object by extracting the samples that occur before or at a specified event from an existing time series.
`gettsbeforeevent`	Create a new `timeseries` object by extracting the samples that occur before a specified event from an existing time series.
`gettsbetweenevents`	Create a new `timeseries` object by extracting the samples that occur between two specified events from an existing time series.

Arithmetic Operation Methods

Use the following operators to arithmetically combine `timeseries` objects.

Methods to Arithmetically Combine Time Series

Operation	Description
`+`	Add the corresponding data values of `timeseries` objects.
`-`	Subtract the corresponding data values of `timeseries` objects.
`.*`	Element-by-element multiplication of `timeseries` data.

Methods to Arithmetically Combine Time Series (Continued)

Operation	Description
*	Matrix-multiply timeseries data.
./	Right element-by-element division of timeseries data.
/	Right matrix division of timeseries data.
.\	Element-by-element left-array divide of timeseries data.
\	Left matrix division of timeseries data.

Statistical Methods

Use the following methods to calculate descriptive statistics for a timeseries object.

Methods for Calculating Descriptive Statistics

Method	Description
iqr (timeseries)	Return the interquartile range of timeseries data.
max (timeseries)	Return the maximum value of timeseries data.
mean (timeseries)	Return the mean of timeseries data.
median (timeseries)	Return the median of timeseries data.
min (timeseries)	Return the minimum of timeseries data.
std (timeseries)	Return the standard deviation of timeseries data.
sum (timeseries)	Return the sum of timeseries data.
var (timeseries)	Return the variance of timeseries data.

Time Series Collection Constructor

- "Introduction" on page 4-34

- "Time Series Collection Constructor Syntax" on page 4-34
- "Time Series Collection Properties" on page 4-35

Introduction

The MATLAB object, called `tscollection`, is a MATLAB variable that groups several time series with a common time vector. The `timeseries` objects that you include in the `tscollection` object are called *members* of this collection, and possess several methods for convenient analysis and manipulation of `timeseries`.

Time Series Collection Constructor Syntax

Before you implement the MATLAB methods specifically designed to operate on a collection of `timeseries` objects, you must create a `tscollection` object to store the data.

The following table summarizes the syntax for using the `tscollection` constructor. For an example of using this constructor, see "Creating Time Series Collection Objects" on page 4-14.

Time Series Collection Syntax Descriptions

Syntax	Description
`tsc = tscollection(ts)`	Creates a `tscollection` object `tsc` that includes one or more `timeseries` objects. The `ts` argument can be one of the following: • Single `timeseries` object in the MATLAB workspace • Cell array of `timeseries` objects in the MATLAB workspace The `timeseries` objects share the same time vector in the `tscollection`.

Time Series Collection Syntax Descriptions (Continued)

Syntax	Description
`tsc = tscollection(Time)`	Creates an empty `tscollection` object with the time vector `Time`. When time values are date strings, you must specify `Time` as a cell array of date strings.
`tsc = tscollection(Time, TimeSeries, 'Parameter', Value, ...)`	Optionally enter the following parameter-value pairs after the `Time` and `TimeSeries` arguments: • `Name` (see "Time Series Collection Properties" on page 4-35)

Time Series Collection Properties

This table lists the properties of the `tscollection` object. You can specify the `Name`, `Time`, and `TimeInfo` properties as input arguments in the `tscollection` constructor.

Time Series Collection Property Descriptions

Property	Description
`Name`	`tscollection` object name entered as a string. This name can differ from the name of the `tscollection` variable in the MATLAB workspace.

Time Series Collection Property Descriptions (Continued)

Property	Description
`Time`	A vector of time values. When `TimeInfo.StartDate` is empty, the numerical `Time` values are measured relative to 0 in specified units. When `TimeInfo.StartDate` is defined, the time values represent date strings measured relative to `StartDate` in specified units. The length of `Time` must match either the first or the last dimension of the `Data` property of each `tscollection` member.
`TimeInfo`	Uses the following fields to store contextual information about `Time`: • `Units` — Time units with the following values: `'weeks'`, `'days'`, `'hours'`, `'minutes'`, `'seconds'`, `'milliseconds'`, `'microseconds'`, and `'nanoseconds'` • `Start` — Start time • `End` — End time (read-only) • `Increment` — Interval between two subsequent time values. The increment is `NaN` when times are not uniformly sampled. • `Length` — Length of the time vector (read-only) • `Format` — String defining the date string display format. See the MATLAB `datestr` function reference page for more information. • `StartDate` — Date string defining the reference date. See the MATLAB `setabstime (timeseries)` function reference page for more information. • `UserData` — Stores any additional user-defined information

Time Series Collection Methods

- "General Time Series Collection Methods" on page 4-37
- "Data and Time Manipulation Methods" on page 4-37

General Time Series Collection Methods

Use the following methods to query and set object properties, and plot the data.

Methods for Querying Properties

Method	Description
`get (tscollection)`	Query `tscollection` object property values.
`isempty (tscollection)`	Evaluate to `true` for an empty `tscollection` object.
`length (tscollection)`	Return the length of the time vector.
`plot (timeseries)`	Plot the time series in a collection.
`set (tscollection)`	Set `tscollection` property values.
`size (tscollection)`	Return the size of a `tscollection` object.
`tstool`	Open the Time Series Tools GUI.

Data and Time Manipulation Methods

Use the following methods to add or delete data samples, and manipulate the `tscollection` object.

Methods for Manipulating Data and Time

Method	Description
`addts`	Add a `timeseries` object to a `tscollection` object.
`addsampletocollection`	Add data samples to a `tscollection` object.

Methods for Manipulating Data and Time (Continued)

Method	Description
`delsamplefromcollection`	Delete one or more data samples from a `tscollection` object.
`getabstime (tscollection)`	Extract a date-string time vector from a `tscollection` object into a cell array.
`getsampleusingtime (tscollection)`	Extract data samples from an existing `tscollection`object into a new `tscollection` object.
`gettimeseriesnames`	Return a cell array of time series names in a `tscollection` object.
`horzcat (tscollection)`	Horizontal concatenation of `tscollection` objects. Combines several `timeseries` objects with the same time vector into one time series collection.
`removets`	Remove one or more `timeseries` objects from a `tscollection` object.
`resample (tscollection)`	Select or interpolate data in a `tscollection` object using a new time vector.
`setabstime (tscollection)`	Set the time values in the time vector of a `tscollection` object as date strings.
`settimeseriesnames`	Change the name of the selected `timeseries` object in a `tscollection` object.
`vertcat (tscollection)`	Vertical concatenation of `tscollection` objects. Joins several `tscollection` objects along the time dimension.

Time Series Tools

In this section...

"Introduction" on page 4-39
"Importing and Exporting Data" on page 4-44
"Plotting Time Series" on page 4-50
"Selecting Data for Analysis" on page 4-64
"Editing Data, Time, Attributes, and Events" on page 4-67
"Processing and Manipulating Time Series" on page 4-76
"Example: Time Series Tools" on page 4-77

Introduction

- "Opening Time Series Tools" on page 4-39
- "Getting Help" on page 4-40
- "Time Series Tools Window" on page 4-41
- "Time Series Tools Workflow" on page 4-42
- "Generating Reusable M-Code" on page 4-43

Opening Time Series Tools

To open Time Series Tools, type the following at the MATLAB prompt:

```
tstool
```

You can also open Time Series Tools using the MATLAB **Start** button by selecting **Start > MATLAB > Time Series Tools**.

For a description of the Time Series Tools GUI, see "Time Series Tools Window" on page 4-41.

To learn how to import data into Time Series Tools, see "Importing and Exporting Data" on page 4-44.

You can also start Time Series Tools and simultaneously import the following kinds of objects from the MATLAB workspace:

- `timeseries`
- `tscollection`
- Simulink® logged signals

Note You cannot import Simulink logged signals that contain a `/` in their `Name` property at any point in the signal hierarchy.

Syntax for Loading Data from the MATLAB Workspace

MATLAB Object	Syntax	Description
`timeseries`	`tstool(tsname)`	`tsname` is the name of a `timeseries` object.
`tscollection`	`tstool(tscname)`	`tscname` is the name of a `tscollection` object.
Simulink logged signal data	`tstool(sldata)`	`sldata` is the name of a signal logged in a Simulink model.

Getting Help

Time Series Tools provides extensive context-sensitive help directly from the GUI.

In the Time Series Tools window, the context-sensitive help pane is available on the right to assist you with the primary tasks. To toggle between displaying or hiding the help pane, click the (**Help**) button in the toolbar. You can resize the help pane by dragging the vertical divider to the left or to the right.

Context-sensitive help is also available via the **Help** button in Time Series Tools dialog boxes.

Time Series Tools Window

The Time Series Tools window consists of the following three areas:

- Time Series Session tree

  Organizes time series data and plots (or **Views**).

  The **Simulink Time Series** node is shown only when you have installed Simulink software.

- Options and Settings pane

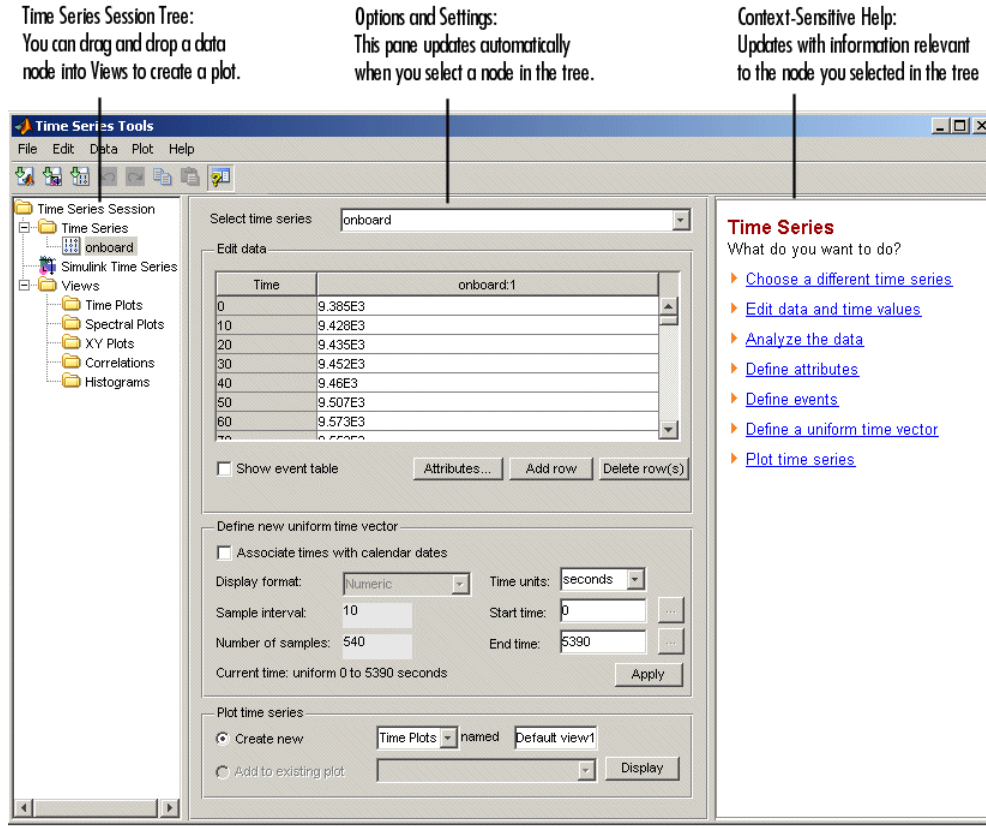
  After you select a node in the tree, this pane displays options and settings pertaining to the node you selected in the tree.

- Context-Sensitive Help pane

  Provides information and instructions about entering the options and settings currently shown in Time Series Tools. You can toggle between displaying or hiding this help by clicking the button in the toolbar. You can change the width of the help pane by dragging the vertical divider to the left or to the right.

  To learn about other help available in Time Series Tools, see "Getting Help" on page 4-40.

The following figure shows the three main areas of the Time Series Tools GUI:



Time Series Tools Workflow

When you analyze data using Time Series Tools, your workflow might include the following tasks:

1. Import data from an Microsoft Excel workbook, MAT-file, or MATLAB workspace.

   For more information, see “Importing and Exporting Data” on page 4-44.

2. Create a time plot to gain insight into the data features.

   For more information, see “Creating a Plot” on page 4-51.

3 Select data subset for analysis.

   For more information, see “Selecting Data Using Rules” on page 4-64.

4 Edit the data by

   - Identifying and removing outliers or “dead time” (see “Selecting Data Using Rules” on page 4-64.
   - Manually correcting errors (see “Editing Data and Time” on page 4-69).

5 Process the data by

   - Interpolating or removing missing values.
   - Detrending data by subtracting a mean value or a linear trend.
   - Filtering to smooth and shape the data.
   - Algebraically manipulating existing time series to create a new time series.
   - Resampling data using a specified time vector by selecting or interpolating values.

   For more information, see “Processing and Manipulating Time Series” on page 4-76.

6 Generating correlation plots, spectral plots, histograms, and *XY* plots.

   For more information, see “Plotting Time Series” on page 4-50.

7 Exporting data from Time Series Tools to the MATLAB workspace or to a file.

   For more information, see “Exporting Data from Time Series Tools” on page 4-49.

Generating Reusable M-Code

You can enable automatic generation of reusable M-code while you perform operations that modify data in Time Series Tools. To do this, select **File > Record M-Code** in the Time Series Tools window.

If you are new to programming with MATLAB `timeseries` methods, you can use the generated M-code to get syntax examples. For more information about programming with MATLAB `timeseries` objects, see "Time Series Objects" on page 4-3.

For an example of automatically generating and viewing M-code, see "Example: Time Series Tools" on page 4-77.

Note The scope of the **Record M-Code** feature is restricted to recording actions on the time series data itself. It does not generate code to import data or reproduce time series plots.

Importing and Exporting Data

- "Types of Data You Can Import" on page 4-44
- "How to Import Data" on page 4-45
- "Changes to Data Representation During Import" on page 4-46
- "Importing Multivariate Data" on page 4-47
- "Importing Data with Missing Values" on page 4-49
- "Exporting Data from Time Series Tools" on page 4-49

Types of Data You Can Import

You can import data into Time Series Tools from

- A Microsoft Excel workbook, a text file, or a MAT-file.
- An array in the MATLAB workspace.
- A `timeseries` or `tscollection` object in the MATLAB workspace.

  For more information about creating these objects, see "Time Series Objects" on page 4-3.
- Logged signal data from a Simulink model.

Note You cannot import a `timeseries` or `tscollection` object from a MAT-file.

How to Import Data

This section includes the following topics:

- "Importing Time Series and Time Series Collection Objects" on page 4-45
- "Importing Data from External Files" on page 4-45
- "Using the Import Wizard" on page 4-46

Importing Time Series and Time Series Collection Objects. If you have already encapsulated time series data in a `timeseries` or `tscollection` object in the MATLAB workspace, you can open Time Series Tools and import the data in a single operation. Simply right-click the object name in the Workspace Browser and choose **Open in Time Series Tools** from the context menu.

Importing Data from External Files. Once you have opened Time Series Tools, use the following commands to import data from external files. Each command opens a dialog box. You can get detailed information about options by clicking **Help**.

Data Source	Import Command
Microsoft Excel worksheet (`.xls`)	Select **File > Create Time Series from File** to open the Import Wizard.
Text file (`.csv`, `.txt`, `.dat`)	Select **File > Create Time Series from File** to open the Import Wizard.
MAT-file array (`.mat`)	Select **File > Create Time Series from File** to open the Import Wizard.
MATLAB workspace array	Select **File > Import from Workspace > Array Data** to open the Import Wizard.

Data Source	Import Command
`timeseries` or `tscollection` object in the MATLAB workspace	Select **File > Import from Workspace > Time Series Objects or Collections**.
Simulink logged signal	Select **File > Import from Workspace > Simulink Data Logs**.

Using the Import Wizard. When in Time Series Tools, you import data from the MATLAB workspace or an external file using the Import Wizard. The Import Wizard lets you select the data to import when analyzing a portion of an Excel worksheet or specific columns or rows in a MATLAB array.

After you select the data, you can specify to import time values from a file or define a uniformly spaced time vector in the Import Wizard. For an example of importing data from an Excel worksheet, see "Importing and Exporting Data" on page 4-44.

Each time series you import is added as a data node to the **Time Series Session** tree.

Note The Import Wizard in Time Series Tools imports data as `timeseries` objects. This is different from the Import Wizard you access from the MATLAB Command Window, which imports data as MATLAB vectors and matrices.

For instructions about working with the Import Wizard, click **Help** in the Import Wizard window. You can also get help on specific fields in the Wizard as follows:

1. Right-click the text label of a field for which you want to get help.
2. Select **What's This** from the shortcut menu.

Changes to Data Representation During Import

When you import data into Time Series Tools, a copy of the data is imported without affecting the original data source.

The data copy is changed during import, as follows:

- Rowwise data is transposed to become columnwise with the time vector in the first column.
- Data with more than two dimensions is reshaped to two dimensions such that dimensions three and higher become additional columns. For example, a 2-by-3-by-5 data array becomes a 2-by-15 data array.
- Non-`double` data, such as `int`, `logical`, and `fixed-point`, is converted to `double`.
- Missing data values are replaced by `NaN`s.
- A sparse matrix is converted to a full matrix.

Caution When you export data from Time Series Tools to a file or to the MATLAB workspace, please note that its representation might differ from what you imported into Time Series Tools. For more information about exporting data, see "Exporting Data from Time Series Tools" on page 4-49.

Importing Multivariate Data

When your data consists of several related variables measured at the same time, you might want to group this data so that you can plot variables together or perform calculations on all variables simultaneously.

There are two ways to represent multivariate data in Time Series Tools:

- Create a time series collection with a common time vector, where each time series is a member of the collection.
- Import a data array into a single `timeseries` object, where each time series is stored as a column.

Choosing How to Represent Multivariate Data. How you choose to represent your data depends on whether the variables have the same or different units.

When your data contains different measurements of the same quantity (same units), you can store all measurements as separate columns in a single time

series. Plotting such a time series displays all columns on the same axes and distinguishes the data sets by line and marker styles. For more information, see "Customizing Line and Marker Styles" on page 4-52.

When your data contains different quantities, measured in different units, you might want to distinguish these quantities on plots and during analysis. In this case, we recommend that you store each quantity as a separate time series and then group them into a time series collection. For example, if you are working with stock-price data in a portfolio, you might represent each stock as a separate time series and group them in a collection. When you plot this collection, each member is plotted on separate axes. However, when you perform data-analysis operations on the collection, such as filtering or interpolation, these operations are applied to all time series in the collection simultaneously.

Creating a Time Series Collection. You can create a time series collection in the MATLAB Command Window, as described in "Time Series Objects" on page 4-3, and then import the collection into Time Series Tools. Alternatively, you can use the Import Wizard to facilitate creating the `timeseries` objects and then group them into a collection in the MATLAB Command Window.

The following procedure describes one way to create a time series collection using data from a file.

Note At each step, you can click the **Help** button in the GUI to access context-sensitive help.

---

1. To import each variable in the Microsoft Excel worksheet or MATLAB array as a separate time series in Time Series Tools, select **File > Import from Workspace > Array Data**. This opens the Import Wizard.

2. After importing the data, select the **Time Series** node in the tree and export these time series to the MATLAB workspace.

3. In the MATLAB Command Window, combine individual time series into a time series collection object. For an example of creating a time series collection, see "Creating Time Series Collection Objects" on page 4-14.

4 In Time Series Tools, select **File > Import from Workspace > Time Series Objects or Collections** and import the collection from the MATLAB workspace.

Importing Data with Missing Values

When you import data from a Microsoft Excel worksheet into Time Series Tools that contains missing values, the missing data is automatically replaced with `NaN`s. `NaN`s are ignored in Time Series Tools calculations.

To remove or interpolate missing values:

1 Select a time series or a collection in the **Time Series Session** tree containing missing values.

2 Select **Data > Interpolate** or **Data > Remove Missing Data**, depending on the operation you want to perform. This opens the Process Data dialog box.

3 Click **Help** to access context-sensitive help on specific options in the dialog box.

Exporting Data from Time Series Tools

Importing data into Time Series Tools creates a copy of the original data. After you finish analyzing the data in Time Series Tools, you must export it to a file or to the MATLAB workspace to make it available for other processing.

To export a time series or a collection, select the desired node in the **Time Series Session** tree. Then, do one of the following:

- Export to a file (Microsoft Excel worksheet or MAT-file):

  Select **File > Export > To File**.

  When you export a time series collection, the individual time series are extracted into separate Microsoft Excel worksheets.

- Export to the MATLAB workspace:

  Select **File > Export > To Workspace**.

Plotting Time Series

- "Types of Plots in Time Series Tools" on page 4-50
- "Creating a Plot" on page 4-51
- "Customizing Line and Marker Styles" on page 4-52
- "Editing Plot Appearance" on page 4-52
- "Time Plots" on page 4-54
- "Spectral Plots" on page 4-55
- "Histograms" on page 4-57
- "Correlation Plots" on page 4-58
- "*XY* Plots" on page 4-62

Types of Plots in Time Series Tools

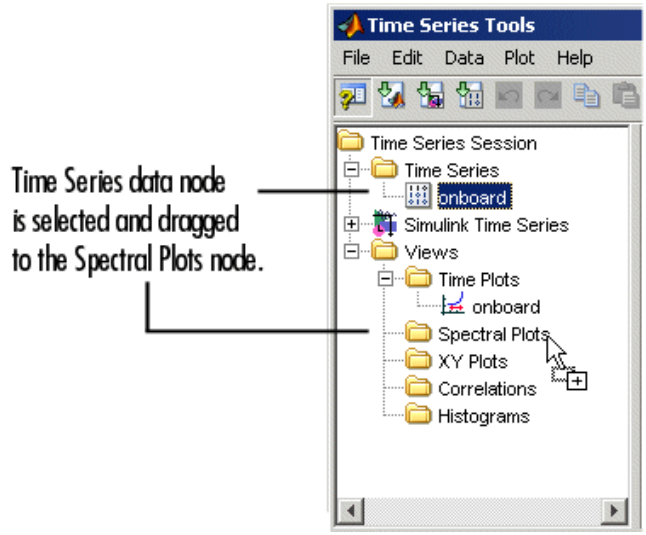
You can generate the following types of plots in Time Series Tools.

Plot Type	Description
Time Plot	Plots data as a function of time to help you see important features, such as outliers, discontinuities, trends, and periodicities.
Histogram	Plots the number of data values that occur in specified data ranges, called *bins*.
Spectral Plot	Shows data periodicities by plotting the estimated power spectral density as a function of frequency.
Correlation Plot	Shows the autocorrelation of a time series or cross-correlation between two time series.
XY Plot	Shows the relationship between two time series by plotting the data values of one on the x-axis and the data values of the other on the y-axis.

Creating a Plot

You can create a plot in the Time Series Tools GUI by dragging a **Time Series** data node from the **Time Series Session** tree and dropping it onto a **Views** folder node.

The following figure shows an example of how to create a spectral plot by dragging the `onboard` time series onto the **Spectral Plots** node:



This opens the spectral plot in the Time Series Plots window and adds a tree node under **Spectral Plots**. The Time Series Plots window is similar to the MATLAB Figure window but includes additional commands in the toolbar and the **Tools** menu.

Tip To change the default plot name, right-click the plot node and select **Rename** and enter the new name.

Subplots. To create subplots in a single figure window, drag several time series onto the same plot node. If a time series contains several columns of data, all data columns are plotted on the same axes. See "Editing Plot

Appearance" on page 4-52 for information on interactively modifying the appearance of subplots.

***XY* and cross-correlation plots.** These plots require two time series. To create these plots, drag one time series onto a plot node and then drag a second time series onto the same plot node.

Customizing Line and Marker Styles

When you plot several time series on the same axes, or a single `timeseries` object that contains multiple columns of data, you can specify how to visually distinguish between the different sets of data in the plot.

To distinguish data by color, type of marker, or line style, select **Plot > Set Line Properties** in the Time Series Tools window. This opens the Line Styles dialog box. Click **Help** to learn how to work with this dialog box.

Note Your changes are applied to all open plots.

For an example of setting line styles, see "Creating a Plot" on page 4-51.

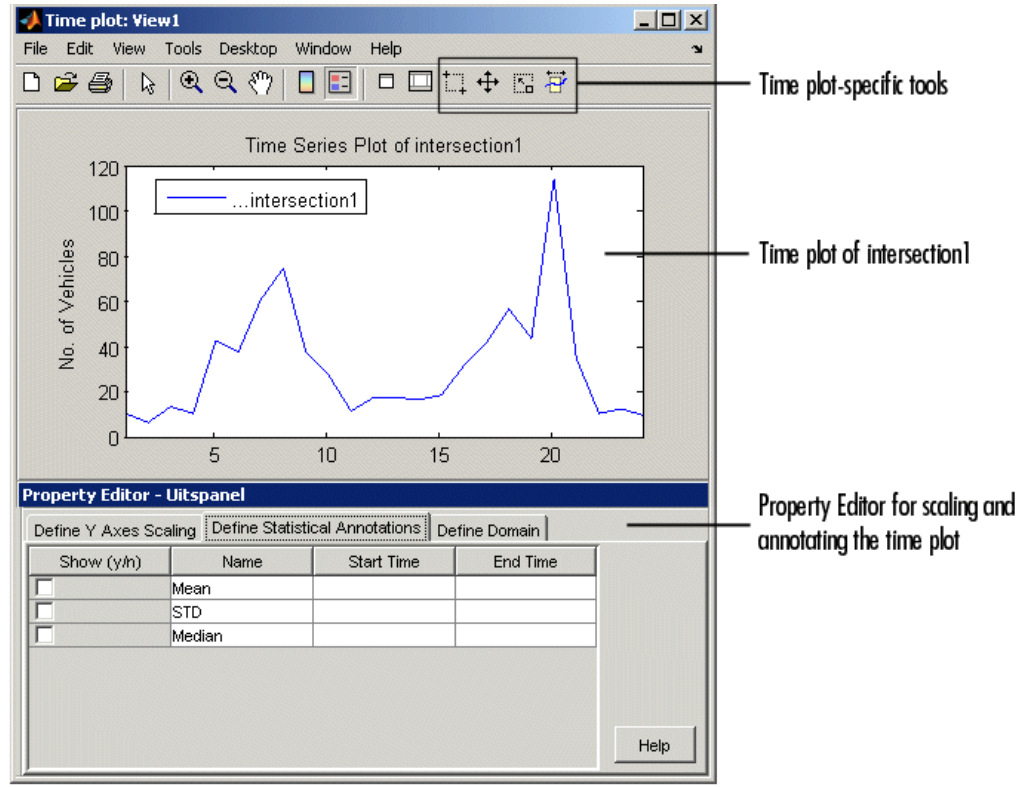
Editing Plot Appearance

After you create a plot, you can modify the plot appearance using the Property Editor as follows:

- Change the range of the horizontal and vertical axes.
- Show statistical annotations on the plot, such as the mean and standard deviation.

  The kinds of statistical quantities you can display vary depending on the type of plot.

The following figure shows the location of the Property Editor relative to the plot window:



To display the Property Editor for any Time Series Tools plot:

1. Select the plot in the **Time Series Session** tree.
2. In Time Series Tools, click the **Edit Plot** button. This displays the plot window on top with the Property Editor below the plot.
3. In the Property Editor, click **Help** to get information about options and settings.

Note The Property Editor options change depending on the type of plot and the plot item you select, such as lines or plot legends.

Subplots. You can change subplot indices interactively. To do so, click on a plotted line in a time series view and drag and drop it from one subplot to another. To create a new subplot, drag and drop the plotted line below the bottom axes.

Time Plots

By plotting data as a function of time, you can quickly gain insight into the following data features:

- Outliers, or values that do not appear to be consistent with the rest of the data
- Discontinuities
- Trends
- Periodicities
- Time intervals containing the data of interest

These features, when considered in the context of the data, enable you to plan your analysis strategy. For more information about creating a time plot, see "Creating a Plot" on page 4-51.

After you create the plot, you can use the Property Editor to

- Define Y-axis scale.
- Display statistical annotations on the plot, such as mean, standard deviation, and median.
- Define X-axis scale (or domain).

In the Property Editor, click **Help** to get information about options and settings.

The Time Plot window contains the following toolbar commands specific to working with time series data.

Time Plot Commands

Button	Description
	Select Data — Enables you to click and drag a rectangular region on the time plot to select the data inside the region.
	Move Time Series — Enables you to click and drag a time series to translate a time series on the plot and recalculate the data and time values. When you translate a time series in time, its time vector is shifted by a constant offset. If you had associated any events with this time series, the events are not shifted with the time series. For more information about editing event times, see "Defining Events" on page 4-13.
	Rescale Time Series — Rescales both axes of the time plot to the original view.
	Select Interval — Enables you to click and drag to select data corresponding to one or more time intervals. You can select multiple disconnected intervals.

Spectral Plots

You use a spectral plot (or periodogram) to determine the frequencies of the periodic variations in the data and to filter the data. For more information about creating a periodogram, see "Creating a Plot" on page 4-51.

The periodogram is the unbiased estimate of the power spectral density of a time series, calculated as the scaled absolute value of the $(FFT)^2$ of the time series. The corresponding frequency vector is computed in cycles per unit time and has the same length as the power vector. The periodogram is scaled so that the variance equals the mean of the periodogram.

The periodogram is useful for picking out periodic components in the presence of noise; a peak in the periodogram indicates an important contribution to variance frequencies near the value that corresponds to the peak.

After you create the plot, you can use the Property Editor to

- Define Y-axis scale.
- Display the variance for a selected frequency range on the plot.

  The periodogram is scaled so that the variance equals the mean of the periodogram.
- Define frequency scale.

In the Property Editor, click **Help** to get information about options and settings.

Filtering the Data. You can use the spectral plot to apply an ideal pass or stop filter to the data.

You use the *ideal notch (stop) filter* when you want to attenuate the variations in the data for a specific frequency range. Alternatively, you use the *ideal pass filter* to allow only the variations in a specific frequency range. These filters are "ideal" in the sense that they are not realizable; an ideal filter is noncausal and the ends of the filter amplitude are perfectly flat in the frequency domain.

To apply an ideal filter:

1. In the Spectral Plot window, click the **Select Frequency Interval(s)** button in the toolbar.
2. Click and drag on the plot to select a frequency interval. The selected interval appears in a different color.
3. Decide if you want to select another frequency interval.
   - If yes, repeat step 2. The previously selected remains selected.
   - If no, go to step 4.
4. Right-click a selected region on the plot and select one of the following from the shortcut menu:
   - To allow only the variations in the selected frequency range, select **Pass**.
   - To remove the variations in the selected frequency range, select **Notch**.

Histograms

The histogram plot shows the distribution of data by counting the number of data values within a specific range of values and displaying each range as a rectangular bin. The heights of the bins represent the numbers of values that fall within each range. For more information about creating a histogram, see "Creating a Plot" on page 4-51.

You can use a histogram plot to select data values that fall in a specific range to exclude or include them in your analysis. If you want to interpolate specific data values, you can select them in a histogram plot first, and then replace them with `NaN`s. For more information, see "Removing and Interpolating Missing Data" on page 4-17. Then, you can interpolate all values tagged as `NaN`s using the selected interpolation method. For more information about specifying an interpolation method, see "Defining Data Attributes" on page 4-70.

Note Time Series Tools generates a histogram plot of a time series by applying the MATLAB `hist` function.

After you create the plot, you can use the Property Editor to

- Define Y-axis scale.
- Display statistical annotations on the plot, including the mean and the median.
- Define data bins.

In the Property Editor, click **Help** to get information about options and settings.

Selecting Data.

1. In the Histogram window, click the **Select Y Range Interval** button in the toolbar.
2. Click and drag a rectangular region on the plot to select a data interval. The selected interval appears in a different color.

3 Decide if you want to select another data range.

- If yes, repeat step 2. The previously selected remains selected.
- If no, you are done.

Removing or Replacing Data with NaNs. After you select the data, as described in "Selecting Data for Analysis" on page 4-64, you can delete it or replace it with `NaN`s. If you want to interpolate specific data values, you must replace the selected data with `NaN`s first.

To delete data, right-click the selected region and select **Remove Selection** from the shortcut menu.

To replace data with `NaN`s, right-click the selected region and select **Replace with NaNs** from the shortcut menu.

Correlation Plots

You can create autocorrelation plots (*correlograms*) and cross-correlation plots in Time Series Tools. A correlation plot shows correlation coefficients on the vertical axis, and lag values on the horizontal axis.

A *lag* is defined as the number of time steps by which a time series is shifted relative to itself (when autocorrelated), or relative to the corresponding time values of another time series (when crosscorrelated). Notice that a lag is not a time shift (in specified time units). However, you can interpret a lag as a time shift when the time series is uniformly sampled (autocorrelation), or when both time series are uniformly sampled with the same time interval (cross-correlation).

This section includes the following topics:

- "Autocorrelation of a Time Series" on page 4-59
- "Cross-Correlation of Time Series" on page 4-59
- "Interpreting Correlation Plots" on page 4-61
- "Cross-Correlation Algorithm" on page 4-62

Note If your data is sampled at irregular time intervals, resample it on a uniform time vector before creating correlation plots. This is because correlation analysis only considers the number of time steps between data values, and not the actual time elapsed between successive measurements. For more information about resampling time series, see "Processing and Manipulating Time Series" on page 4-76.

Autocorrelation of a Time Series. The *autocorrelation* function is an important diagnostic tool for analyzing time series in the time domain. You use the autocorrelation plot, or *correlogram*, to better understand the evolution of a process through time by the probability of relationship between data values separated by a specific number of time steps.

The correlogram plots correlation coefficients on the vertical axis, and lag values on the horizontal axis. To learn more about correlation coefficients, see "Correlation Coefficients" on page 3-5.

To create a correlogram, drag and drop a time series into a **Correlations** node. Then explore the plot by editing the lag range in the Property Editor.

If a time series contains multiple data columns, your plot contains cross-correlations of the various data columns. For more information, see "Cross-Correlation of Time Series" on page 4-59.

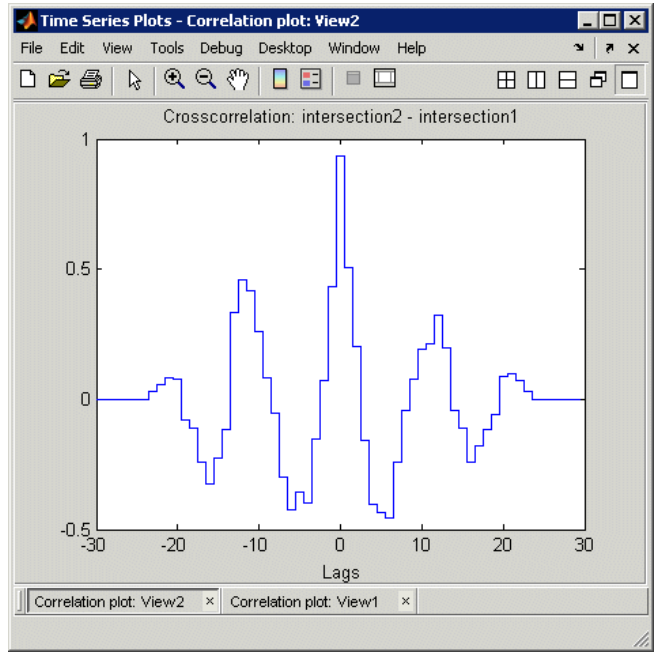
Note A correlogram is not useful when the data contains a trend; data at all lags will appear to be correlated because a data value on one side of the mean tends to be followed by a large number of values on the same side of the mean. You must remove any trend in the data before you create a correlogram. For more information about accessing detrending functionality, see "Processing and Manipulating Time Series" on page 4-76.

Cross-Correlation of Time Series. *Cross-correlation* is a measure of the degree of the linear relationship between two time series. A high correlation between time series at a specific lag might indicate a time delay in the system.

Note Before creating a cross-correlation plot, make sure that both time series have the same uniform time vector.

To create a cross-correlation plot, successively drag and drop the first time series and the second time series into the same **Correlations** node in the **Time Series Session** tree. Then explore the plot by varying the lag range in the Property Editor.

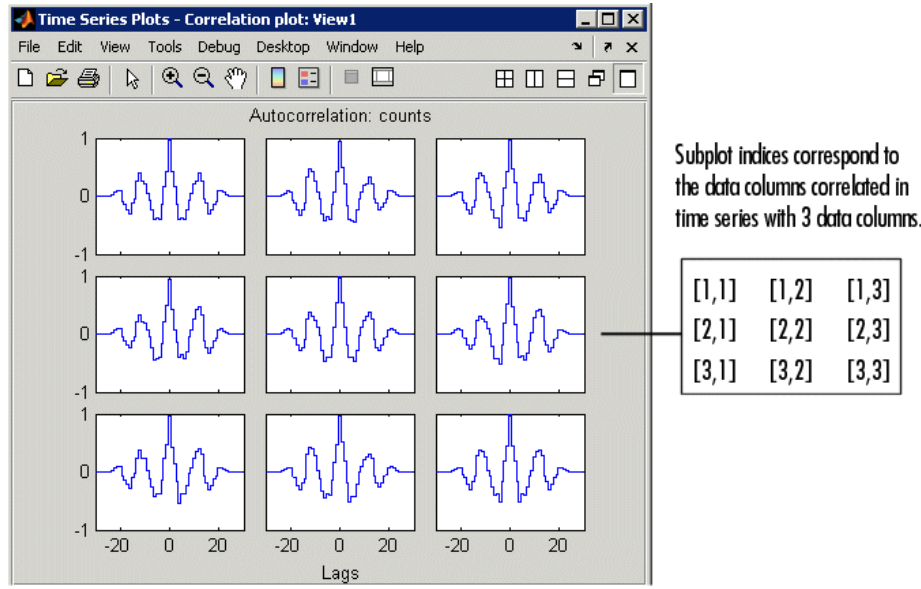
A cross-correlation plot of two time series, where each contains a single column of data, shows the degree of linear relationship between the data values in the two time series at various lags. For example, the following figure shows a cross-correlation plot of two time series, `intersection1` and `intersection2`. There is a high correlation when there is no lag in the data, as well as for lags of about `-11` and `11`.



Cross-Correlation of Two Time Series

A cross-correlation plot of two time series, where each contains multiple data columns, is displayed as a grid of subplots. The number of subplots equals the number of columns of data in the first time series multiplied by the number of columns of data in the second time series.

When you autocorrelate a time series with multiple data columns, the resulting plot also contains subplots. The diagonal of the subplot is the autocorrelation of a specific data column. The off-diagonal subplots are cross-correlation plots of the various columns. The subplot indices correspond to the indices of the data columns being correlated. For example, the figure below shows a correlation plot of the time series `counts` with three data columns.



Cross-Correlation of Multiple Data Columns in a Time Series

Interpreting Correlation Plots. The following table describes the degree of relationship between the data values at a given lag for various correlation values.

Correlation Value	Meaning
Close to 1	There is a relationship between data values at a specific lag: an increase in one corresponds to an increase in the other.
0	The variations in the data show no relationships at this lag.
Close to -1	There is an anticorrelation between the data values at a specific lag: a decrease in one data value corresponds to an increase in the other data value.

Cross-Correlation Algorithm. When computing the cross-correlation of two vector-valued time series `x` and `y`, Time Series Tools uses an algorithm that is functionally equivalent to calling the Signal Processing Toolbox `xcorr` function from with the `'biased'` option, after the time series means have been removed. Unlike `xcorr`, however, the cross-correlation estimate in Time Series Tools also works for matrix-valued time series `X` and `Y`, where it computes the cross-correlation of `X(:,i)` against `Y(:,j)` for all combinations of columns `i` and `j`. Note that Time Series Tools do not actually use the `xcorr` code, but rather a simplified version which works under these restricted assumptions.

XY Plots

An *XY* plot plots the data values of one time series against the data values of another time series at corresponding times. Any relationship between the two time series is evident from a pattern on the plot. For example, when the points on the *XY* plot form a straight line, there is a linear relationship between the data values of the two time series plotted. The *XY* plot does not show any time information.

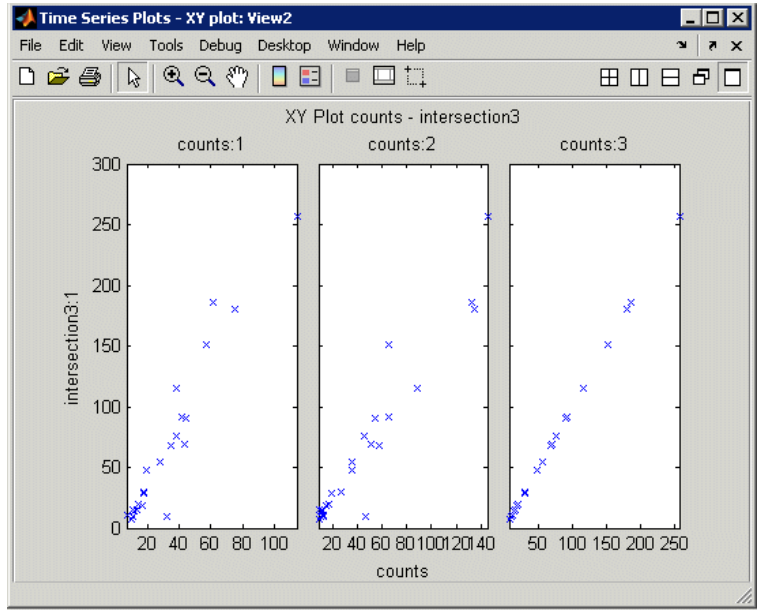
Note To generate an *XY* plot, both time series must have the same time vectors.

To create an *XY* plot, successively drag and drop the first time series and the second time series into the same **XY Plots** node in the **Time Series Session** tree.

When you are plotting two time series where each contains a single column of data, the *XY* plot includes a single set of axes. The pairs of data values from the same position in the column of data; that is, the third data point from one column is plotted against the third data point from the other column. For an example of generating such an *XY* plot, see "Comparing Data on an *XY* Plot" on page 4-89.

An *XY* plot of two time series, where each contains one or multiple data columns, is displayed as a grid of subplots. The number of subplots equals the number of columns of data in the first time series multiplied by the number of columns of data in the second time series. The subplot indices correspond to the indices of the data columns.

The following figure shows an *XY* plot, where the data values in time series `count` are plotted on the *X*-axis against the corresponding data values of `intersection1` on the *Y*-axis. Because `count` contains three data columns and `intersection1` contains one data column, the *XY* plot window shows three subplots.



***XY* Plot Where One Time Series Contains Three Data Columns**

Selecting Data for Analysis

- "Selecting Data Using Rules" on page 4-64
- "Selecting Data Graphically" on page 4-65
- "Excluding Data from Analysis" on page 4-66

Selecting Data Using Rules

You can select data using logical expressions in the Select Data Using Rules dialog box, which you access from a time plot. For more information about creating a time plot, see "Creating a Plot" on page 4-51.

To open the Select Data Using Rules dialog box, right-click inside the time plot and choose **Select Data** from the shortcut menu. Click **Help** in the dialog box to get information about specific options.

You can define up to four kinds of data-selection conditions:

- Bounds — Upper and lower bounds for time and data values
- Outliers — Condition for detecting outliers, or data values that are outside a specified confidence level
- MATLAB expression — A logical MATLAB expression that selects specific data values
- Flatlines – Condition for detecting a specified number of successive data points with a constant value

---

Tip To learn how to exclude data from analysis based on your selection, see "Excluding Data from Analysis" on page 4-66.

Selecting Data Graphically

This section describes how to select data in a time plot by using the mouse. For more information on creating a time plot, see "Creating a Plot" on page 4-51.

You can select data using two modes:

- Data mode — Enables you to select data values in a rectangular region on the time plot.

  For more information, see "Selecting Data in a Rectangular Region" on page 4-66.

- Time mode — Enables you to select data values in one or more time intervals on the time plot.

  For more information, see "Selecting Data in a Time Interval" on page 4-66.

Tip To learn how you can select specific data values in a histogram plot, see "Selecting Data for Analysis" on page 4-64.

Selecting Data in a Rectangular Region.

1 In the Time Plot window, click the **Select Data** button in the toolbar.

2 Click and drag a rectangular region on the plot that encloses the data you want to select.

The data values are selected when you release the mouse button.

3 Decide if you want to select another region.

- If yes, repeat step 2. This does not clear the previous selection.
- If no, you can continue by excluding data from analysis (see "Excluding Data from Analysis" on page 4-66).

Selecting Data in a Time Interval.

1 In the Time Plot window, click the **Select Time Interval(s)** button in the toolbar.

2 Click the start of a region that encloses the time interval where you want to select data, and then drag it. The selected time interval appears in a different color.

3 Decide if you want to select another time interval.

- If yes, repeat step 2. This does not clear the previous selection.
- If no, you can continue by excluding data from analysis (see "Excluding Data from Analysis" on page 4-66).

Excluding Data from Analysis

After you select the data, you can either exclude or keep the selected values. The following table summarizes how to do this.

Task	Operation
Exclude selected data from analysis	Right-click the selected data in the time plot and select **Remove Observations** from the shortcut menu. When there are multiple data columns in a single time series, this removes the entire data sample at that time.
Exclude unselected data from analysis	Right-click the selected data in the time plot and select **Keep Observations** from the shortcut menu.

Editing Data, Time, Attributes, and Events

- "Displaying the Data Table" on page 4-67
- "Editing Data and Time" on page 4-69
- "Defining Data Attributes" on page 4-70
- "Assigning Quality Codes to Data" on page 4-72
- "Defining Events" on page 4-73

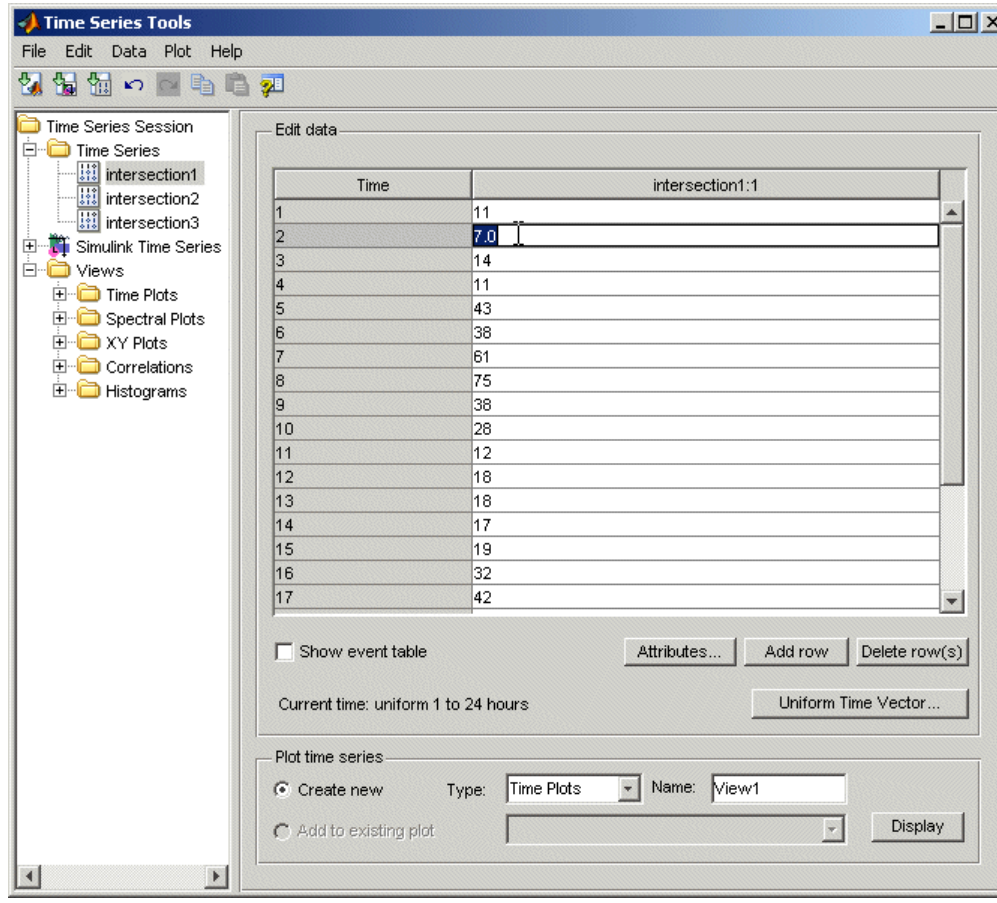
Displaying the Data Table

To display the time series in an editable table, select the time series node in the Time Series Session tree.

In the following figure, the time series `intersection1` is selected in the tree and its data table is shown on the right. The **Time** column contains time values and the **intersection1:1** column contains the corresponding data values in the first column and only data column of `intersection1`.

If `intersection1` had multiple data columns, they would appear in the table and numbered as **intersection1:2**, **intersection1:3**, and so on. The data column headers are also used as plot labels to distinguish time series in

plots. For more information about creating plots, see “Plotting Time Series” on page 4-50.



Note To toggle between displaying and hiding the help pane in Time Series Tools, click the button in the toolbar.

Editing Data and Time

After you display the time series data, as described in "Displaying the Data Table" on page 4-67, you can edit specific data and time values, define a uniform time vector, and add or remove data samples.

Edit Time or Data Values. To edit a specific time or data value, double-click that cell in the table and enter the new value. Press **Enter**.

Note When entering time values, you must use the current display format of your time vector. For more information, see "Time Vector Format" on page 4-22.

Define a Uniform Time Vector. To define a uniformly-increasing time vector, click **Uniform Time Vector** below the data table. This opens the Define Uniform Time Vector dialog box.

Here, you specify the start and end time of the time vector, the time units, and the display format. The time interval is calculated automatically by dividing the total time range by the number of data samples. You can get more instructions by clicking **Help** in the Define Uniform Time Vector dialog box.

When you are done specifying the time vector, the new time values replace the previous time values in the data table.

Add Data Samples. To insert a row in the data table, click any cell in a row and click the **Add Row** button. Enter the time and the corresponding data values.

Delete Data Samples. To delete a row in the data table, select one or more rows with the mouse and click the **Delete Row(s)** button.

Defining Data Attributes

The following attributes are defined for time series:

- Units — Stored as metadata for each time series.
- Interpolation method — Default method used to fill in missing data or to resample data on a new time vector.
- Quality codes — Used to annotate the quality of each value in the data table.

Click the **Attributes** button below the data table to open the Define Data Attributes dialog box. For information about displaying the data table, see "Displaying the Data Table" on page 4-67.

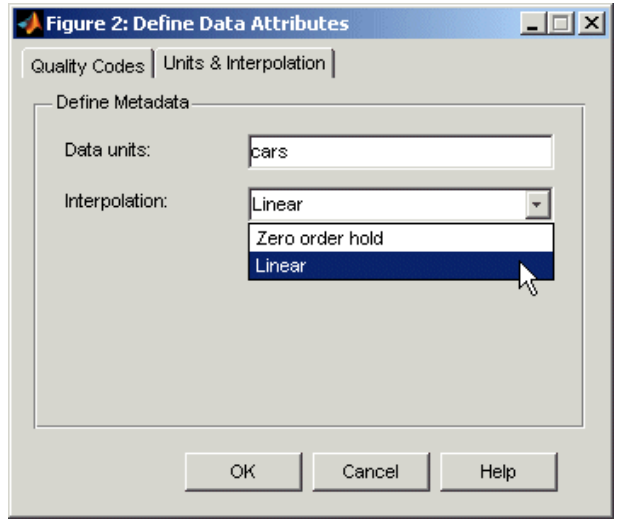
Units and Interpolation Method. Data units are stored as metadata for the currently selected time series. If this time series contains multiple data columns, all data is assigned the same units.

In the **Units & Interpolation** tab, enter a string in the **Data units** field. For example, enter `N/m^2`.

The interpolation method you select here is used by default for this time series to fill in missing data or to resample the data on a new time vector.

In the **Units & Interpolation** tab, select one of the following **Interpolation** methods:

- `Linear` — A 1-D interpolation method that implements the MATLAB function `interp1` to fit a straight line between a pair of existing data points to calculate the missing value.
- `Zero-order hold` — Calculates the missing value by setting it equal to the last available data value. In other words, this methods "holds" the last value constant until the next available measurement.



Quality Codes. You can define quality codes to annotate the quality of each value in the data table. Each quality attribute consists of a numerical code and a brief description. For information about assigning quality codes to specific data values, see "Assigning Quality Codes to Data" on page 4-72.

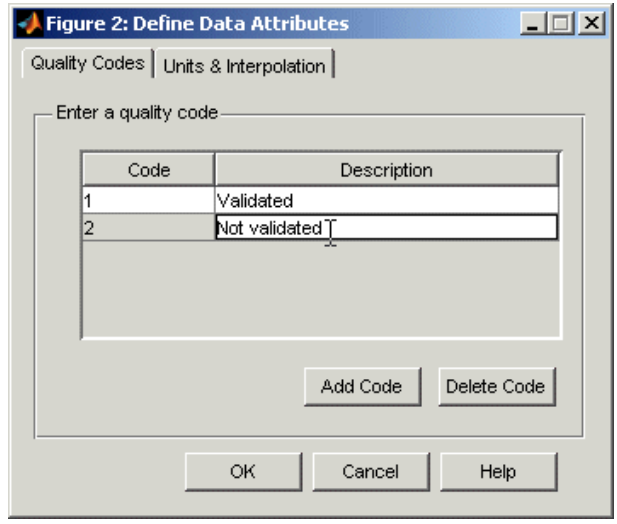
Tip To save time, first define the quality attribute that applies to most of your data values. It is automatically assigned to all data values. Then, define the attributes that occur less frequently and set them manually in the **Quality** column of the data table.

---

1. In the Define Data Attributes dialog box, click the **Quality Codes** tab.
2. Click the **Add Code** button. This adds an empty row in the Quality Codes table.
3. Click the empty cell in the **Code** column and type an integer from 0 to 127.
4. Press the **Tab** key. This highlights the cell in the **Description**. Type one or two words that briefly describe the numerical code, such as `Validated`.

5 To add another quality code, repeat steps 2 to 4. Or click **OK** to close the dialog box. This also assigns the first quality code you defined to all data values in the table.

The following figure shows two quality codes: `Validated` and `Not validated`.

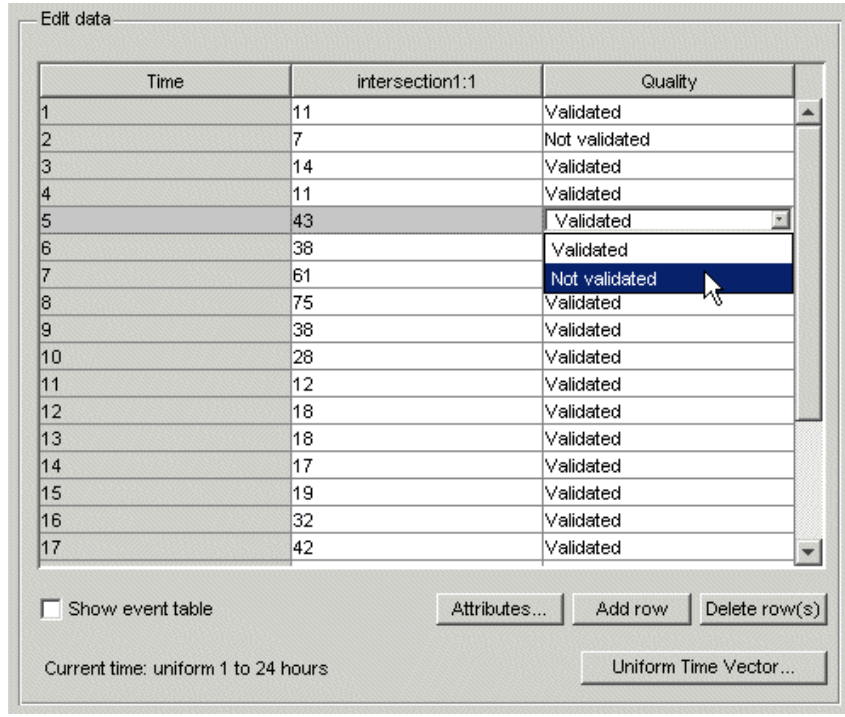


Note To delete a quality attribute, select it and click **Delete Code**.

Assigning Quality Codes to Data

After you define quality codes, as described in "Quality Codes" on page 4-71, the quality code you defined first is automatically assigned to all data values in the data table. For information about displaying the data table, see "Displaying the Data Table" on page 4-67.

To assign a different quality code to a specific data value, click the corresponding cell in the **Quality** column and select a different value from the drop-down list.



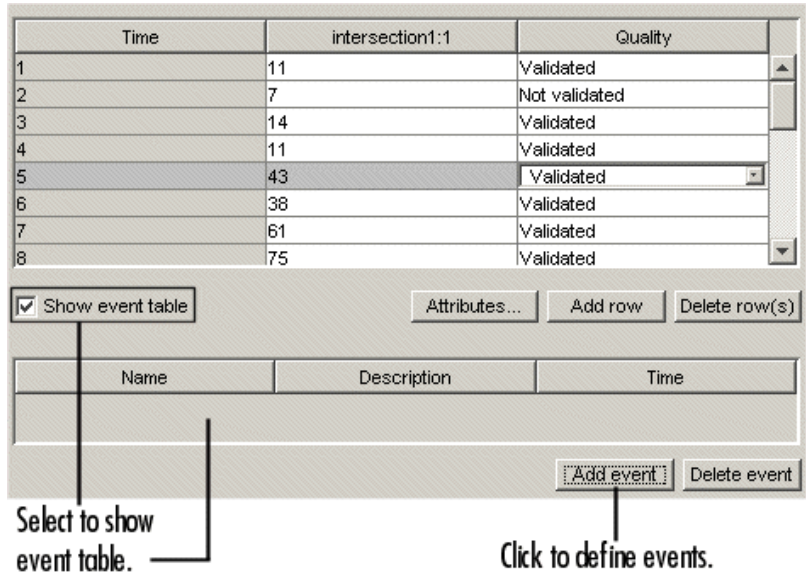
Defining Events

Events are stored as metadata for each time series. Time series events mark the data at a specific time in the data table and on a plot. For information about displaying the data table, see “Displaying the Data Table” on page 4-67.

You can also use events as reference points when shifting time series in time. For more information about synchronizing time series, see “Processing and Manipulating Time Series” on page 4-76.

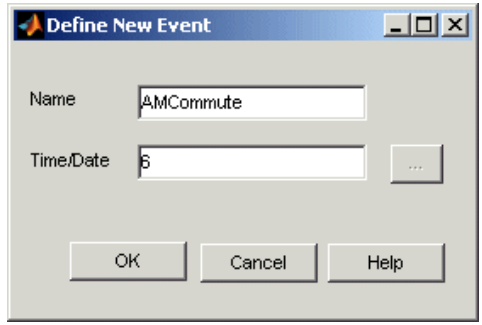
To define events for the selected time series:

1. Make sure that the **Show event table** check box is selected. This check box is located below the data table:



2 Click the **Add event** button below the event table. This opens the Define New Event dialog box.

3 In the **Name** field, enter the name of the event, such as `AMCommute`.

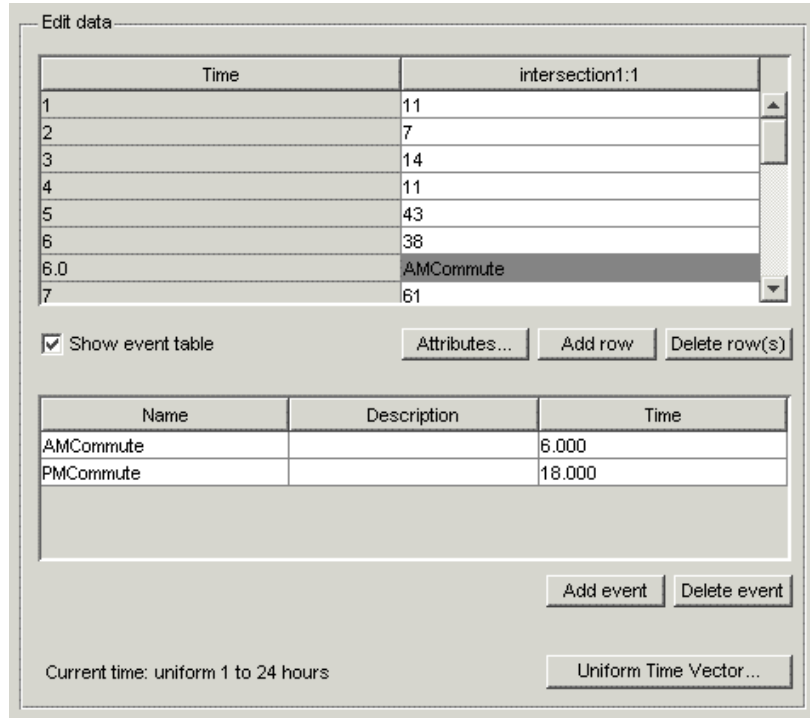


4 In the **Time/Date** field, enter or edit the time of the event in the appropriate display format. For information about time-vector formats, see "Time Vector Format" on page 4-22.

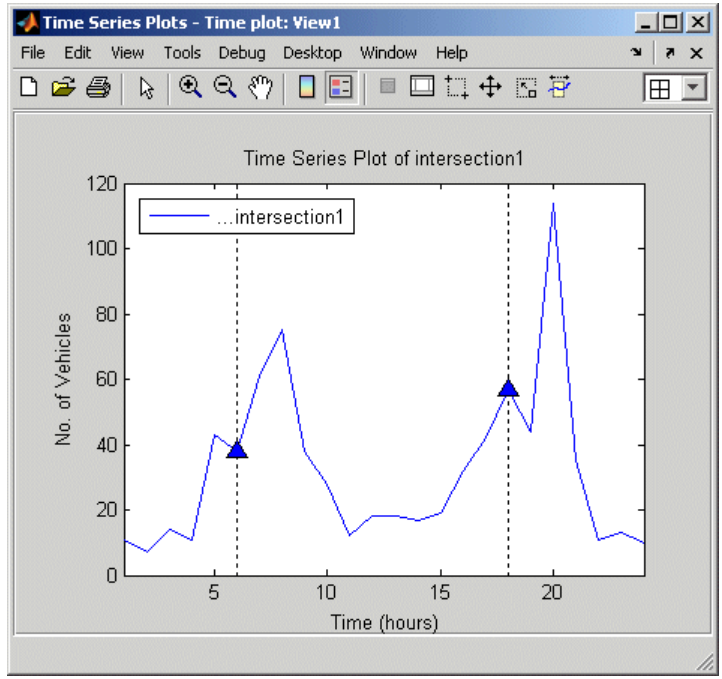
Tip To facilitate entering a date string, click the ... (**Browse**) button to open the Specify Date/Time dialog box. Select the month, year, and day. Then enter the **Time** in HH:MM:SS format.

5 Click **OK**.

The following figure shows two events in the event table: `AMCommute` and `PMCommute`. The data table also contains both events and `AMCommute` is shown at 6.0 hours.



Events are displayed as markers on time series plots. The following figure shows the `AMCommute` marker (at 6.0 hours) and `PMCommute` marker (at 18.0 hours) on a time plot.



Time Plot with Event Markers

Processing and Manipulating Time Series

The following table summarizes the operations you can perform on individual time series or time series collection. These commands are available from the **Data** menu in Time Series Tools after you select a time series or collection node in the Time Series Session tree.

Note If you are viewing a time plot, these operations are available by right-clicking inside the time plot and selecting a command from the shortcut menu. For more information about plotting data, see "Plotting Time Series" on page 4-50.

Each command opens a dialog box where you can get detailed instructions by clicking the **Help** button.

Data Analysis Commands

Command	Description
Data > Remove Missing Data	Delete the times that contain missing data.
Data > Detrend	Subtract a constant or a linear trend from the data.
Data > Filter	Smooth and shape the time series data.
Data > Interpolate	Interpolate missing values.
Data > Resample	Select or interpolate data values using a specified time vector.
Data > Transform Algebraically	Create a new time series by algebraically manipulating existing time series. This command is available only when you select an individual time series in the tree.
Data > Descriptive Statistics	Get summary statistics for each time series.

Example: Time Series Tools

- "Loading Data into the MATLAB Workspace" on page 4-78
- "Starting Time Series Tools" on page 4-78
- "Enabling M-Code Generation" on page 4-78
- "Importing Data into Time Series Tools" on page 4-79
- "Creating a Time Plot" on page 4-81
- "Resampling Time Series" on page 4-87
- "Comparing Data on an *XY* Plot" on page 4-89
- "Viewing Generated M-Code" on page 4-91
- "Exporting Time Series to the Workspace" on page 4-93

Loading Data into the MATLAB Workspace

Type the following command at the MATLAB prompt to load the hourly traffic counts at three road intersections, collected over a 24-hour period:

```
load count.dat
```

This adds the variable `count` to the MATLAB workspace.

Starting Time Series Tools

To start Time Series Tools, type

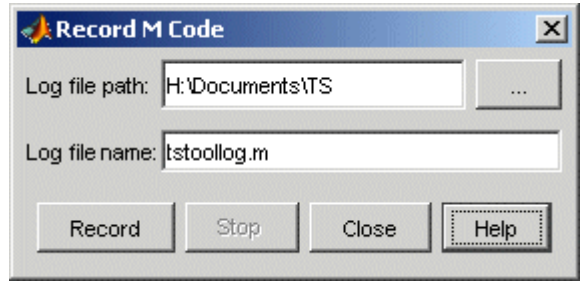
```
tstool
```

This opens the Time Series Tools window. For more information about this GUI, see "Time Series Tools" on page 4-39.

Enabling M-Code Generation

In this portion of the example, you will enable automatic M-code generation in Time Series Tools to capture reusable M-code as a MATLAB function.

1. In the Time Series Tools window, select **File > Record M-Code**. This opens the Record M-Code dialog box.



2. Click the ... button and select the folder where you want to store the M-file.

3. In the **Log file name** field, either select the name of a recently used file, or type a new name. The file name creates the function name you call in your M-code to reuse this function.

4 To begin capturing M-code, click **Record**. The M-code is recorded until you stop recording, as described in "Viewing Generated M-Code" on page 4-91.

Tip You can close this dialog box without interrupting the recording operation by clicking **Close**. To reopen the dialog box, select **File > Record M-Code** in the Time Series Tools window.

Note The scope of the **Record M-Code** feature is restricted to recording actions on the time series data itself. It does not generate code to import data or reproduce time series plots.

Importing Data into Time Series Tools

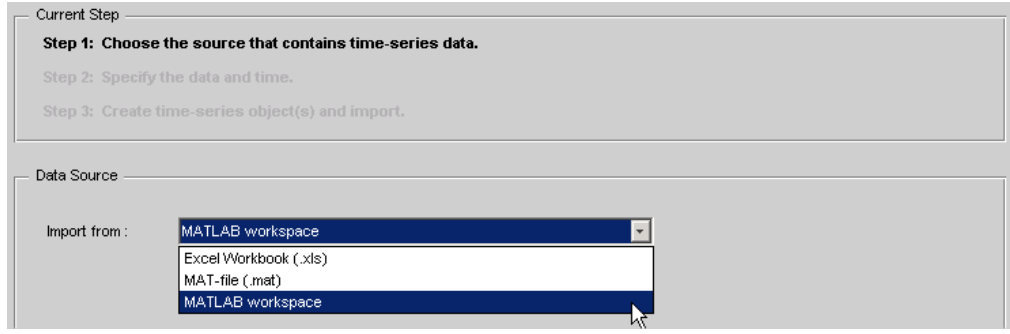
This portion of the example shows how to create three time series from the 24-by-3 count array you loaded into the MATLAB workspace.

Note To get help on a specific field in the Import Wizard, right-click the field label and select **What's This** from the shortcut menu.

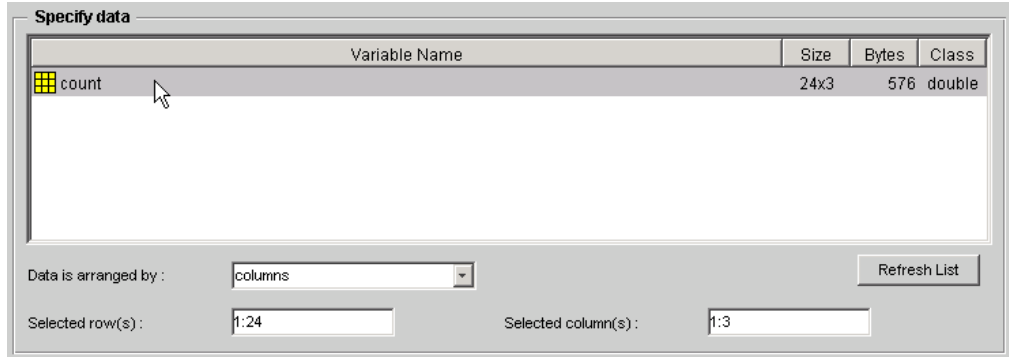
---

1 In the Time Series Tools window, select **File > Import from Workspace > Array Data**. This opens the Import Wizard.

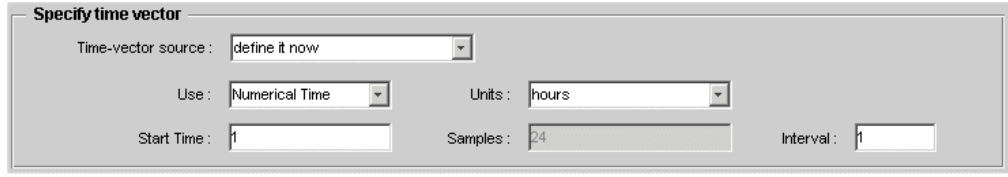
2 In the **Import from** list, select **MATLAB workspace** and click **Next**.



3 In **Step 2** of the Import Wizard, select the `count` variable. The Import Wizard infers from the data that it is arranged in columns.

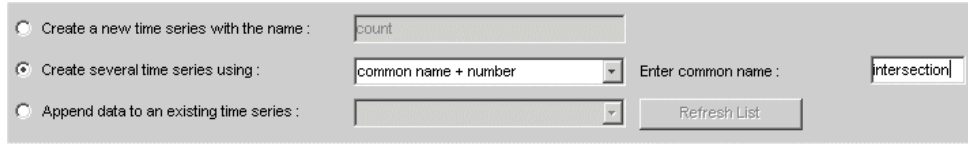


4 In the **Specify Time Vector** area, select `hours` from the **Units** list. In the **Start Time** field, type `1` to start the time vector at 1 hour. The Import Wizard has already filled in the remaining options to define a uniformly spaced time vector with a length of `24` and an interval of `1`.

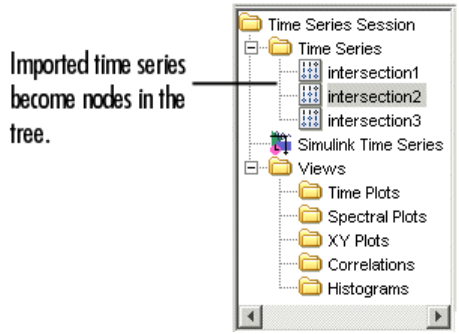


5 Click **Next**.

6 In **Step 3** of the Import Wizard, select **Create several time series using:** `common name+number`. In the **Enter common name** field, type `intersection`.



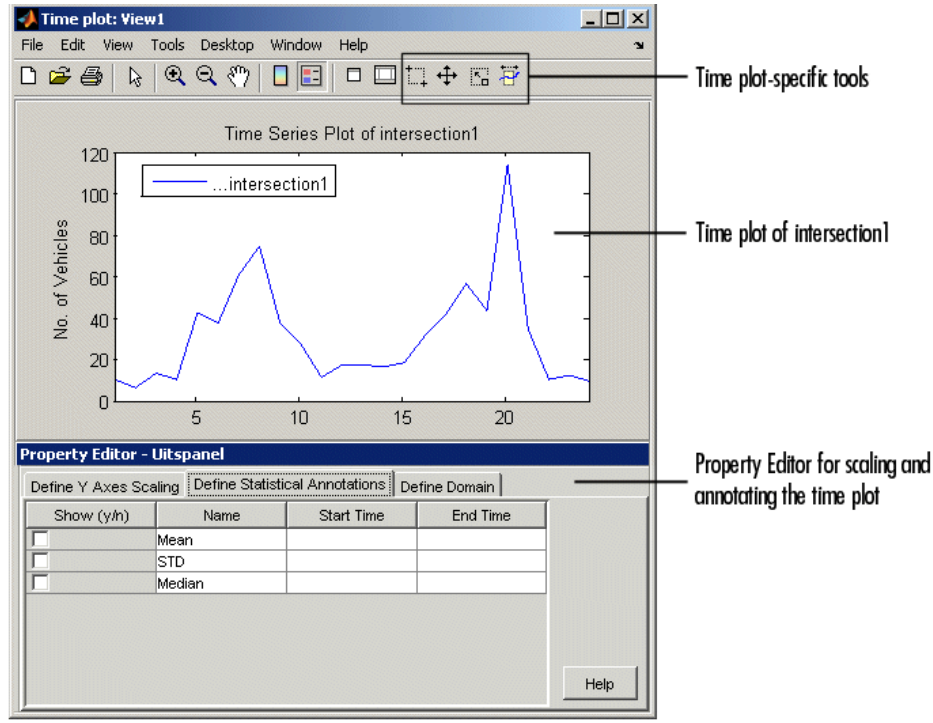
7 Click **Finish**. This adds three time series to the Time Series Session tree: `intersection1`, `intersection2`, and `intersection3` (as shown below).



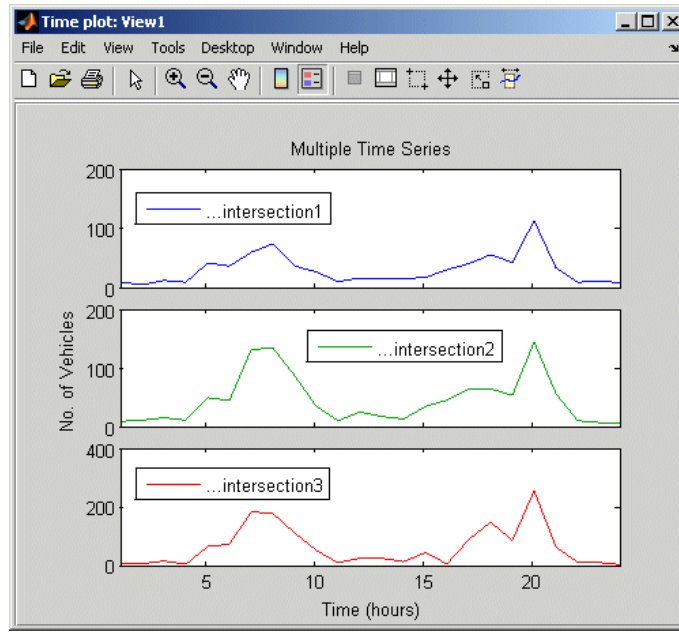
Creating a Time Plot

To explore the data, you can create a time plot of the three time series in the Time Series Tools window.

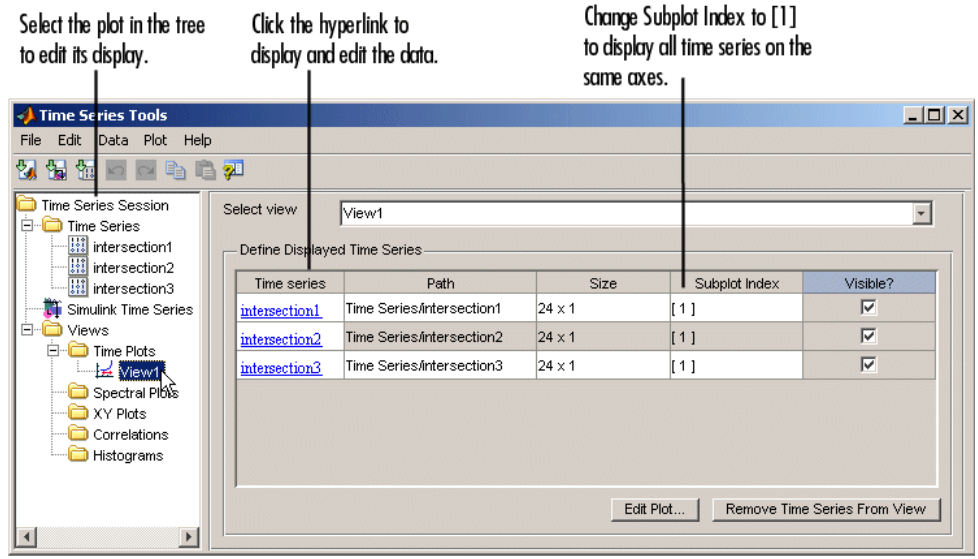
1 In the **Time Series Session** tree, drag and drop the `intersection1` time series into the **Time Plots** node. This creates a time plot in a new window with the default name **View1**.



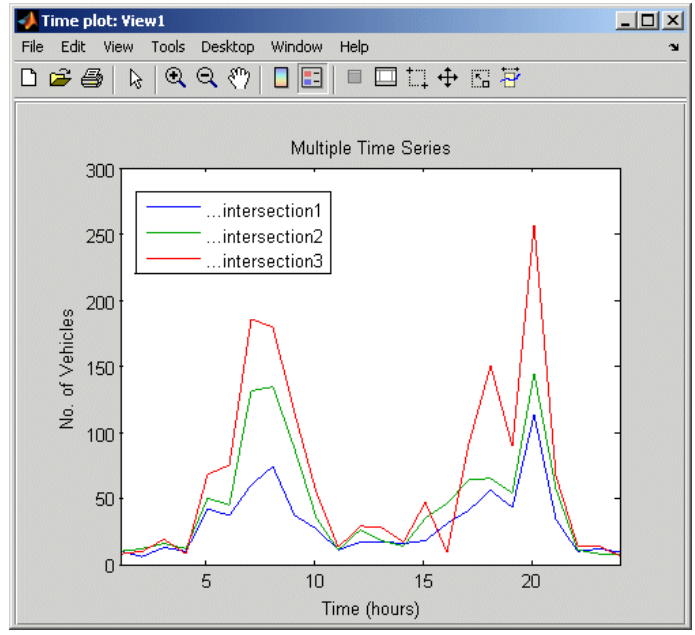
2 In the **Time Series Session** tree, drag and drop the `intersection2` and `intersection3` time series into **View1** to add them to the plot.



3 To display all three time series on the same axes, click the **View1** node in the Time Series Tools window. Change the subplot indices for `intersection2` and `intersection3` to `[1]` and press **Enter**.

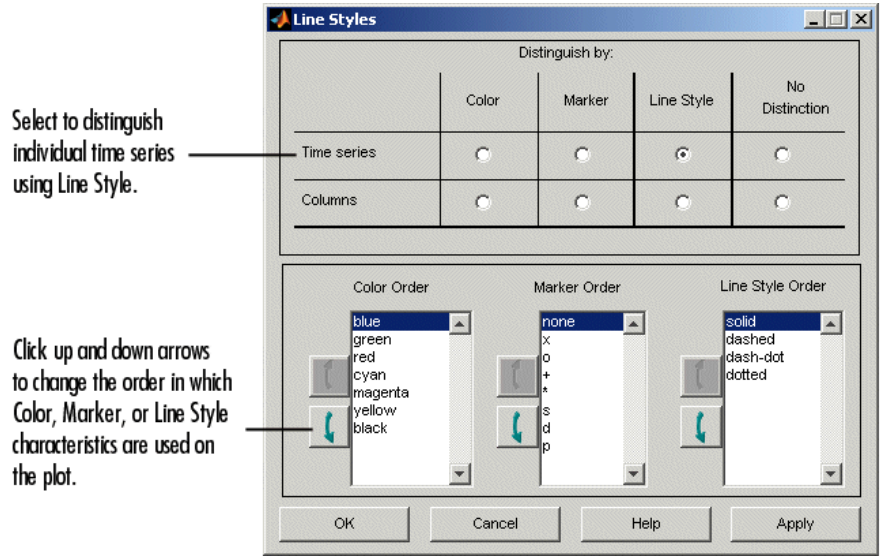


This displays all time series on the same axes, as follows:

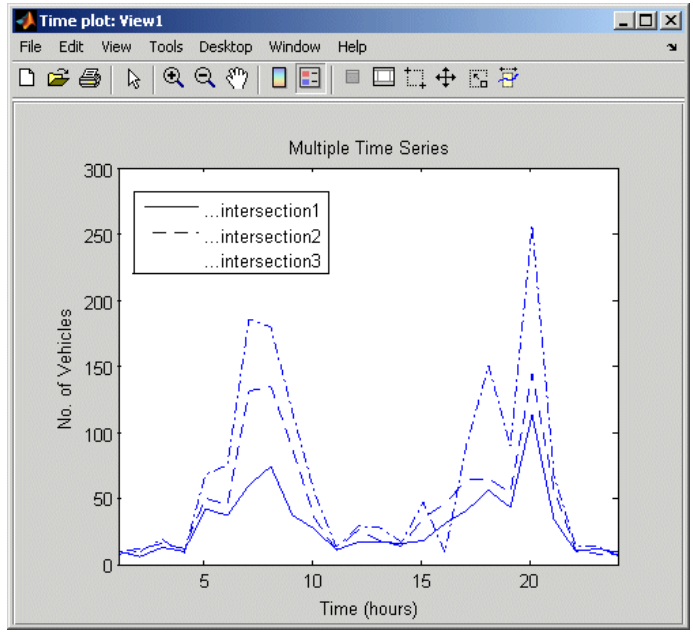


4. To change the appearance of the time series in the plot, go to the main Time Series Tools window and select **Plot > Set Line Properties**. This opens the Line Styles dialog box.

5. In the Line Styles dialog box, click **Line Style** to distinguish the time series, shown as follows.



The plot now looks like this.



Resampling Time Series

You can select or interpolate time series data using a specified time vector. When the new time vector contains time values that are not present in the original time vector, the intermediate data values are calculated using the interpolation method you associated with this time series. Linear interpolation is used by default. For more information about specifying the interpolation method, see "Defining Data Attributes" on page 4-70.

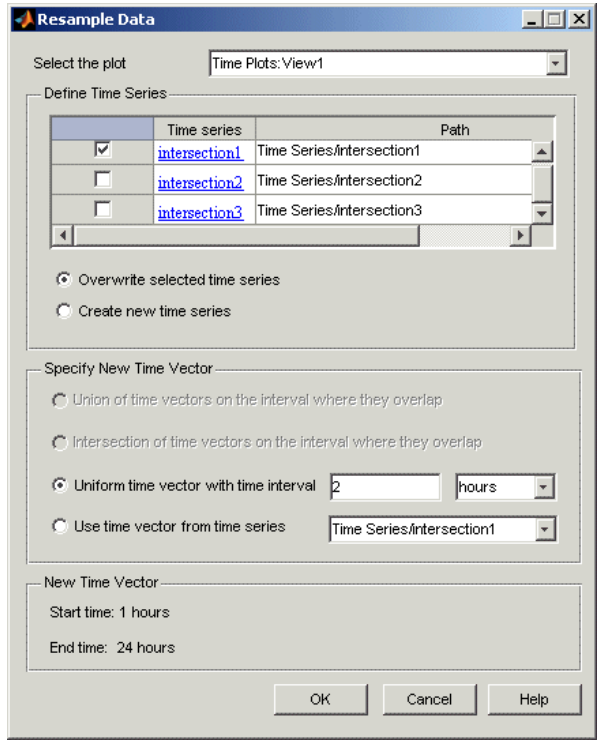
This portion of the example shows

- "Resampling on a Uniform Time Vector" on page 4-87
- "Resampling by Finding a Common Time Vector" on page 4-89

Note You can only resample one time series at a time.

Resampling on a Uniform Time Vector. First, you resample the time series `intersection1` to include values every 2 hours.

1 Right-click inside the time plot you created in "Creating a Plot" on page 4-51 and select **Resample Data** from the shortcut menu. This opens the Resample Data dialog box.



2 In the **Define Time Series** area, select only `intersection1` and clear the rest.

3 In the **Specify New Time Vector** area, click **Uniform time vector with time interval** and specify the time interval as 2 hours. Click **OK**.

Tip To verify that `intersection1` is resampled, select it in the **Time Series Session** tree and examine the data table. It should have a time vector that starts at 1 hour and increases in increments of 2 hours.

Resampling by Finding a Common Time Vector. In some cases, you might want one time series to have the same time vector as another time series on the overlapping region of time values. This is especially useful when you want a specific time series to inherit a nonuniformly spaced time vector.

In this example, you resample `intersection2` on the same time vector as `intersection1`.

1. Right-click inside the time plot you created in "Creating a Plot" on page 4-51 and select **Resample Data** from the shortcut menu. This opens the Resample Data dialog box.
2. In the **Define Time Series** area, select only `intersection2` and clear the rest.
3. In the **Specify New Time Vector** area, click **Use time vector from time series** and select `intersection1` from the list. Click **OK**.

To verify that `intersection2` is resampled, select it in the **Time Series Session** tree and examine the data table. It should have a time vector that starts at 1 hour and increases in increments of 2 hours.

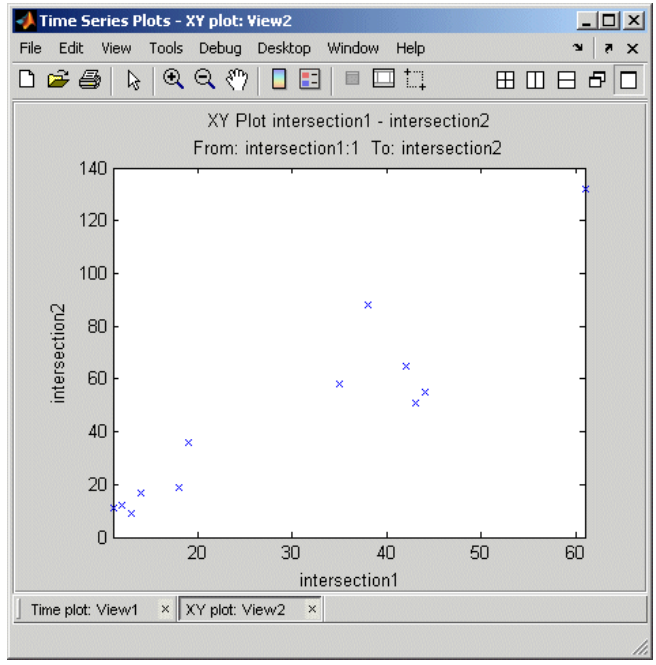
Comparing Data on an *XY* Plot

The *XY* plot is useful for visually determining a relationship between the data values of time series at corresponding times. For example, when the points on an *XY* plot form a straight line, there is a linear relationship between the two time series.

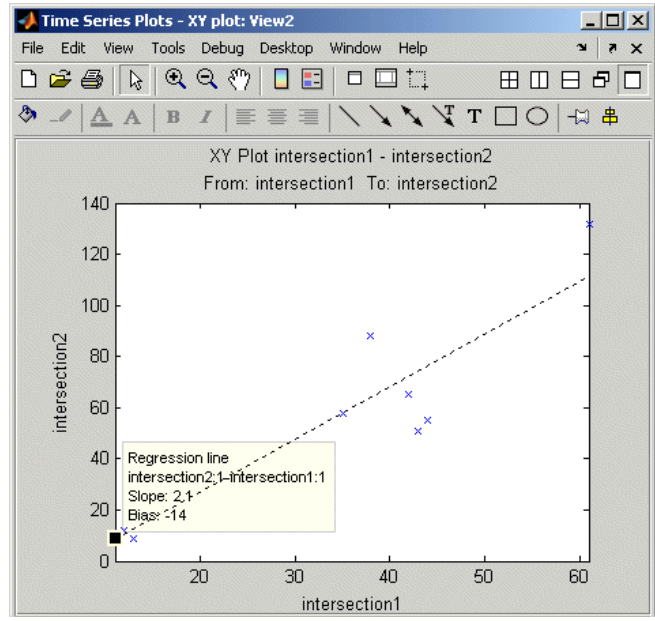
In this portion of the example, you examine the relationship between the corresponding data values of `intersection1` and `intersection2` by using an *XY* plot.

1. In the Time Series Session tree, drag and drop the **intersection1** time series into the **XY Plots** node. This creates a new plot node with the default name **View2**.

2 Drag and drop the **intersection2** time series into the **View2** node. This creates the following *XY* plot.



3 To show the best-fit line on the *XY* plot, click the **Define Statistical Annotations** tab in the Property Editor and select the **Best fit line** check box. Then, click the line to display the line equation on the plot.



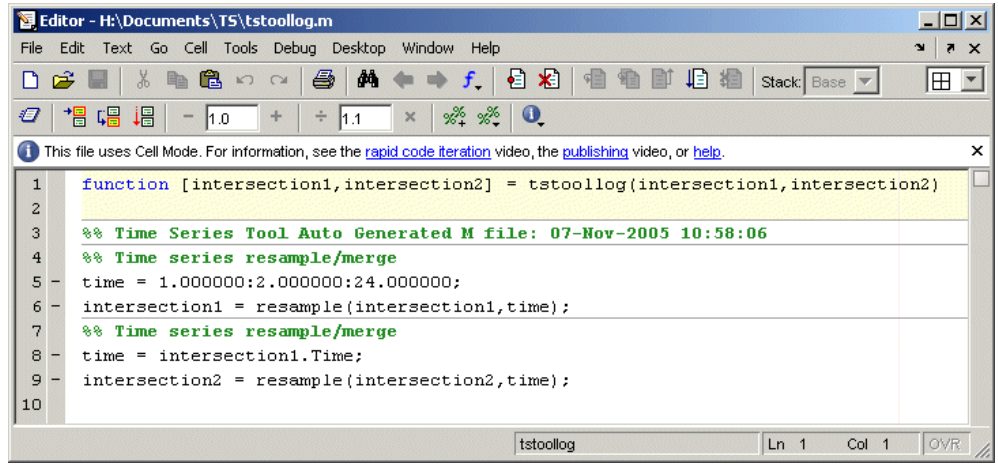
Viewing Generated M-Code

You can now view the M-code that Time Series Tools generated while you performed the previous steps in this example.

To view the M-file:

1 In the Time Series Tools window, select **File > Record M-Code** to open the Record M-Code dialog box.

2 Click **Stop** to open the M-file with the generated M-code in the MATLAB Editor.



Automatically Generated M-Code

You can reuse this M-code by calling the `tstoollog` function, which has the same name as this M-file. You specified the file name when you enabled M-code generation in this example, as described in "Enabling M-Code Generation" on page 4-78.

Examine the code of the `tstoollog` function to confirm that it takes two time series as input arguments and resamples them using a uniform time vector with the range 1 to 24 and intervals of 2.

Note The scope of the **Record M-Code** feature is restricted to recording actions on the time series data itself. It does not generate code to import data or reproduce time series plots.

Exporting Time Series to the Workspace

You can export individual time series, as well as time series collections, from Time Series Tools to the MATLAB workspace. You can also export time series to a Microsoft Excel worksheet or a MAT-file.

In this portion of the example, you will export the time series `intersection1` as a variable to the MATLAB workspace. This time series differs from the original data you imported into Time Series Tools because it has been resampled, as described in "Resampling Time Series" on page 4-87.

1. Click the **interesection1** node in the Time Series Session tree to select it.
2. Select **File > Export > To Workspace**. The variable `intersection1` is now listed in the MATLAB workspace.

Note If the MATLAB workspace is hidden, select **Desktop > Workspace** from the MATLAB window to display it.

Index

A

attributes of time series 4-67
autocorrelation of time series 4-59

B

Basic Fitting 3-10
Basic Fitting dialog box
    usage example 3-12

C

condition
    data 3-13
confidence bounds 3-34
correlation analysis 3-2
correlation coefficients 3-5
correlation plots 4-58
    interpreting 4-61
covariance 3-2
cross-correlation of time series 4-59 4-62
curve fitting. *See* data fitting
Curve Fitting Toolbox
    for regression analysis 3-8
customizing time series plots 4-52

D

Data
    badly conditioned 3-13
    center and scale 3-13
data analysis
    plotting data 1-4
data brushing
    3-D plots 2-9
    defined 2-4
    multiple plots 2-9
    techniques for 2-6
data cursor mode
    update function example 2-25
data filtering. *See* filtering
data fitting 3-1
    confidence bounds 3-34
    example using functions 3-30
    functions 3-23
    multiple regression 3-29
    nonpolynomial 3-27
    polynomial 3-23
    residuals 3-8
data linking
    broken links 2-23
    controls for 2-21
    defined 2-13
    reasons for using 2-14
data statistics. *See* statistics
Data Statistics dialog box 1-26
    generating an M-file 1-33 3-21
    saving statistics 1-32
    usage example 1-26
datatips
    example of customizing 2-25
descriptive statistics 1-23
detrending data 1-18
    in Time Series Tools 4-76
difference equations 1-12
discrete filter 1-14

E

editing time series 4-67
events in time series 4-67
exploratory data analysis 2-2
exporting data
    from MATLAB 1-2
    from Time Series Tools 4-44

F

`filter` function 1-12
filtering

detrending data 1-18
difference equations 1-12
discrete filter 1-14
`filter` function 1-12
in Time Series Tools 4-76
moving average 1-13
finite differences 1-22
functions
for data fitting 3-23
for data statistics 1-23

G

goodness of fit 3-8

H

histogram 4-57
used to select data 4-57

I

importing data
into MATLAB 1-2
into Time Series Tools 4-44
interactive data exploration 2-2
interpolating missing data 1-9
define method for time series 4-70
in Time Series Tools 4-76
`isnan` function 1-8

L

linear regression 3-1
linked plots
behavior of 2-16
information bar 2-14
working with 2-13
linking versus refreshing graphs 2-19
`load` function 1-4

M

M-code from Time Series Tools 4-43
maximum 1-23
mean 1-23
median 1-23
methods
for `timeseries` object 4-29
for `tscollection` object 4-37
minimum 1-23
missing data
in calculations 1-7
in time series 4-49
interpolating 1-7
removing 1-7
removing in Time Series Tools 4-76
representing by `NaNs` 1-7
mode 1-23
moving-average filter 1-13
multiple regression 3-29

N

`NaNs`
in calculations 1-7
removing from data 1-8
nonpolynomial fit 3-27

O

objects for time series analysis 4-3
outliers
removing 1-10

P

periodogram 4-55
filtering data from 4-56
`plot` function 1-5
plotting data
in MATLAB 1-4

in Time Series Tools 4-50
polyfit function 3-23
polynomial regression 3-23
polyval function 3-23
properties
of timeseries object 4-24
of tscollection object 4-35
Property Editor
in Time Series Tools 4-52

Q

quality codes for time series data 4-71

R

range 1-23
refreshing versus linking graphs 2-19
regression 3-1
multiple 3-29
nonpolynomial 3-27
polynomial 3-23
removing
missing data 1-8
NaNs 1-8
outliers 1-10
resampling
in Time Series Tools 4-76
tscollection object 4-15
residuals 3-8

S

Simulink logged signals 4-44
spectral plot 4-55
filtering data from 4-56
standard deviation 1-23
statistics
formatting on plots 1-30
functions 1-23
in Time Series Tools 4-76
MATLAB Data Statistics 1-26
removing NaNs 1-8
removing outliers 1-10
showing on plots 1-27

T

time plot 4-54
time series 4-2
time series analysis
autocorrelation 4-59
cross-correlation 4-59 4-62
example using methods 4-6
example using Time Series Tools 4-77
methods 4-3
multivariate data 4-47
using Time Series Tools 4-39
Time Series Tools
customizing plots 4-52
define time series units 4-67
defining data quality 4-71
defining events 4-73
defining interpolation method 4-70
detrending data 4-76
editing data 4-67
filtering data 4-76
generating M-code 4-43
getting help 4-40
Import Wizard 4-46
importing data 4-44
interpolating data 4-76
opening 4-39
plot Property Editor 4-52
plotting data 4-50
removing missing data 4-76
resampling data 4-76
selecting data 4-64
transforming data algebraically 4-76
usage example 4-77
viewing statistics 4-76

window 4-41
workflow 4-42
time vector
format 4-22
uniform 4-69
`timeseries` object
constructor 4-23
creating 4-21
definition of data sample 4-4
methods 4-29
properties 4-24
tools
MATLAB Basic Fitting 3-10
MATLAB Data Statistics 1-26
Time Series Tools 4-39
transfer-function filter 1-14
`tscollection` object
constructor 4-34
creating 4-33
methods 4-37
properties 4-35

U

uniform time vector 4-69

V

variance 1-23
visual data analysis 2-2

W

workflow
in Time Series Tools 4-42

X

XY plot 4-62