

MATLAB[®]

The Language of Technical Computing

- Computation
- Visualization
- Programming

Data Analysis

Version 7



How to Contact The MathWorks:



www.mathworks.com
comp.soft-sys.matlab
www.mathworks.com/contact_TS.html

Web
Newsgroup
Technical Support



suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
service@mathworks.com
info@mathworks.com

Product enhancement suggestions
Bug reports
Documentation error reports
Order status, license renewals, passcodes
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 2005-2006 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, Simulink, Stateflow, Handle Graphics, Real-Time Workshop, and xPC TargetBox are registered trademarks of The MathWorks, Inc.

Other product or brand names are 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)

Preparing Data for Analysis

1

Using MATLAB for Data Analysis	1-2
Calculations on Vectors and Matrices	1-3
MATLAB GUIs for Data Analysis	1-3
Related Toolboxes	1-4
Importing and Exporting Data	1-6
Plotting Data	1-7
Example — Loading and Plotting Data	1-7
Removing and Interpolating Missing Values	1-9
Representing Missing Data Values	1-9
Calculating with NaNs	1-9
Removing NaNs from the Data	1-10
Interpolating Missing Data	1-11
Removing Outliers	1-12
Filtering Data	1-14
Filter Function	1-14
Example 1 — Moving-Average Filter	1-15
Example 2 — Discrete Filter	1-16
Detrending Data	1-20
Example — Removing Linear Trends from the Data	1-20
Finite Differences	1-24
Descriptive Statistics	1-25
Functions for Calculating Descriptive Statistics	1-25
Example — Using MATLAB Data Statistics	1-28

Data Fitting Using Linear Regression

2

Introduction	2-2
Residuals and Goodness of Fit	2-2
When to Use the Curve Fitting Toolbox	2-3
Linear Correlation Analysis	2-4
Covariance	2-4
Correlation Coefficients	2-6
Using MATLAB Basic Fitting	2-8
What Is MATLAB Basic Fitting?	2-8
Sorting Data to Improve Performance	2-8
Opening MATLAB Basic Fitting	2-9
Example — Using MATLAB Basic Fitting	2-10
Data Fitting Using MATLAB Functions	2-20
MATLAB Functions for Polynomial Models	2-20
Linear Model with Nonpolynomial Terms	2-24
Multiple Regression	2-26
Example — Data Fitting Using MATLAB Functions ...	2-28
Calculating Correlation Coefficients	2-29
Fitting a Polynomial to the Data	2-30
Plot and Calculate Confidence Bounds	2-32

Fourier Analysis

3

Introduction	3-2
Function Summary	3-3
Calculating Fourier Transforms	3-4
Example — Calculating the FFT of a Column Vector	3-5

Example — Using FFT to Calculate Sunspot Periodicity	3-7
Magnitude and Phase of Transformed Data	3-11
FFT Length Versus Performance	3-13

Using Time-Series Objects and Methods

4

Introduction	4-2
Time-Series Data Sample	4-3
Example — Using Time-Series Objects and Methods ..	4-6
Creating timeseries Objects	4-6
Modifying timeseries Units and Interpolation Method	4-8
Defining Events	4-9
Creating tscollection Objects	4-10
Resampling a tscollection Object	4-11
Adding a Data Sample to a tscollection Object	4-12
Removing and Interpolating Missing Data	4-13
Removing a timeseries from a tscollection	4-15
Changing a Numerical Time Vector to Date Strings	4-16
Plotting tscollection Members	4-17
timeseries Constructor	4-18
Time Vector Format	4-18
timeseries Constructor Syntax	4-19
timeseries Properties	4-21
timeseries Methods	4-28
General Methods	4-28
Data and Time Manipulation Methods	4-29
Event Methods	4-30
Arithmetic Operation Methods	4-31
Statistical Methods	4-32

tscollection Constructor	4-33
tscollection Constructor Syntax	4-33
tscollection Properties	4-34
tscollection Methods	4-36
General tscollection Methods	4-36
Data and Time Manipulation Methods	4-36

Using Time Series Tools

5

Introduction	5-2
Opening Time Series Tools	5-2
Getting Help	5-3
Time Series Tools Window	5-3
Time Series Tools Workflow	5-5
Generating Reusable M-Code	5-6
Importing and Exporting Data	5-8
Types of Data You Can Import	5-8
How to Import Data	5-8
Changes to Data Representation During Import	5-10
Importing Multivariate Data	5-11
Importing Data with Missing Values	5-12
Exporting Data from Time Series Tools	5-13
Plotting Time Series	5-14
Types of Plots in Time Series Tools	5-14
Creating a Plot	5-15
Customizing Line and Marker Styles	5-16
Editing Plot Appearance	5-16
Time Plots	5-18
Spectral Plots	5-19
Histograms	5-21
Correlation Plots	5-22
XY Plots	5-27
Selecting Data for Analysis	5-29
Selecting Data Using Rules	5-29

Selecting Data Graphically	5-30
Excluding Data from Analysis	5-31
Editing Data, Time, Attributes, and Events	5-33
Displaying the Data Table	5-33
Editing Data and Time	5-34
Defining Data Attributes	5-36
Assigning Quality Codes to Data	5-38
Defining Events	5-39
Processing and Manipulating Time Series	5-43
Example — Using MATLAB Time Series Tools	5-44
Loading Data into the MATLAB Workspace	5-44
Starting Time Series Tools	5-44
Enabling M-Code Generation	5-44
Importing Data into Time Series Tools	5-45
Creating a Time Plot	5-47
Resampling Time Series	5-53
Comparing Data on an XY Plot	5-55
Viewing Generated M Code	5-57
Exporting Time Series to the Workspace	5-59

Index



Preparing Data for Analysis

The following sections summarize MATLAB® data-analysis capabilities, and provide information about preparing your data for analysis.

Using MATLAB for Data Analysis (p. 1-2)	Provides an overview of data analysis using MATLAB
Importing and Exporting Data (p. 1-6)	Explains where to get information about importing and exporting data
Plotting Data (p. 1-7)	Provides information about MATLAB plots, and includes an example of loading data from a text file and creating a time plot
Removing and Interpolating Missing Values (p. 1-9)	Describes using NaNs to represent missing data, as well as removing or interpolating these values
Removing Outliers (p. 1-12)	Describes how to identify and remove values that seem inconsistent with the majority of the data
Filtering Data (p. 1-14)	Describes how to smooth and shape data using filters
Detrending Data (p. 1-20)	Describes how to remove the mean or a best-fit line from the data
Finite Differences (p. 1-24)	Summarizes MATLAB functions for computing finite differences
Descriptive Statistics (p. 1-25)	Summarizes MATLAB functions for calculating descriptive statistics and provides an example of using the Data Statistics dialog box

Using MATLAB for Data Analysis

MATLAB® provides functions and GUIs to perform a variety of common data-analysis tasks, such as plotting data, computing descriptive statistics, and performing linear correlation analysis, data fitting, and Fourier analysis.

Typically, the first step to any data analysis is to plot the data. After examining the plot, you can determine which portions of the data to include in the analysis. You can also use the plot to evaluate if your data contains any features that might distort or confuse the analysis results, and then process your data to work only with the regions of interest.

This chapter describes the common techniques you can use to ready your data for analysis. When you work with empirical data, it is often necessary to treat it by doing the following:

- Removing or interpolating missing values. For more information, see “Removing and Interpolating Missing Values” on page 1-9.
- Removing outliers. For more information, see “Removing Outliers” on page 1-12.
- Smoothing the data using a first-order filter, a transfer function, or an ideal filter. For more information, see “Filtering Data” on page 1-14.
- Removing the mean or a linear trend (*detrending*). For more information, see “Detrending Data” on page 1-20.
- Differencing the data. For more information, see “Finite Differences” on page 1-24.

After isolating the data of interest, you can proceed with the core data-analysis tasks, which might include basic data fitting (see Chapter 2, “Data Fitting Using Linear Regression”) and Fourier analysis (see Chapter 3, “Fourier Analysis”). If your data analysis requires more advanced or specialized functionality, see “Related Toolboxes” on page 1-4 to learn about the toolboxes available from The MathWorks.

If you are working with time-series data, MATLAB provides the `timeseries` and `tscollection` objects and methods that enable you to efficiently represent and manipulate time-series data. For more information about creating and working with these objects, see Chapter 4, “Using Time-Series

Objects and Methods”. Alternatively, you can use the MATLAB Time Series Tools graphical user interface (GUI) to import, plot, and analyze time series. For more information, see Chapter 5, “Using Time Series Tools”.

Calculations on Vectors and Matrices

Whereas some MATLAB functions support only vector inputs, others accept matrices.

When your data is a vector, the result is the same whether the vector has a rowwise or columnwise orientation.

However, when your data is a matrix, MATLAB performs calculations independently for each column. This means that when you pass a matrix as an argument to the function `max`, for example, the result is a row vector containing maximum data values for each column in the matrix.

Note When your data is a matrix where each row contains a data set, you must transpose the matrix before proceeding with the data-analysis tasks to make the data sets have a columnwise orientation. For example, to transpose a real matrix A , use the syntax A' .

MATLAB GUIs for Data Analysis

In addition to the various MATLAB functions for performing data analysis, MATLAB provides four graphical user interfaces (GUIs) that facilitate common data-analysis tasks. The following table lists these GUIs and tells you how to get more information about each one.

MATLAB GUIs for Data Analysis

GUI	Description	More Information
MATLAB Figure window	For plotting variables in the MATLAB workspace and editing plot properties	MATLAB Graphics documentation

MATLAB GUIs for Data Analysis (Continued)

GUI	Description	More Information
Data Statistics dialog box	For calculating and plotting descriptive statistics	“Example — Using MATLAB Data Statistics” on page 1-28
Basic Fitting dialog box	For basic data fitting using polynomial and spline models, as well as plotting fitted data and residuals	“Using MATLAB Basic Fitting” on page 2-8
Time Series Tools	For plotting and manipulating time-series data	Chapter 5, “Using Time Series Tools”

Related Toolboxes

The following table summarizes the toolboxes that extend MATLAB data-analysis capabilities. For the latest information about these and other MathWorks products, point your Web browser to

www.mathworks.com

Toolboxes That Extend MATLAB Data Analysis

Toolbox	Description
	Read, analyze, and visualize genomic, proteomic, and microarray data.
Curve Fitting Toolbox	Perform model fitting and analysis.
Financial Time Series Toolbox	Analyze and manage financial time-series data.
Financial Toolbox	Analyze financial data and develop financial algorithms.
Image Processing Toolbox	Perform image processing, analysis, and algorithm development.
	Calibrate complex powertrain systems.

Toolboxes That Extend MATLAB Data Analysis (Continued)

Toolbox	Description
Neural Network Toolbox	Design and simulate neural networks.
“”	Perform signal processing, analysis, and algorithm development.
Spline Toolbox	Create and manipulate spline approximation models of data.
	Apply statistical algorithms and probability models.
System Identification Toolbox	Create linear dynamic models from measured input-output data.
Wavelet Toolbox	Analyze and synthesize signals and images using wavelet techniques.

Importing and Exporting Data

The first step in analyzing data is to import it into MATLAB. The MATLAB Programming documentation provides detailed information about supported data formats and the functions for bringing data into MATLAB.

The easiest way to import data into MATLAB is to use the MATLAB Import Wizard, as described in the MATLAB Programming documentation. With the Import Wizard, you can import the following types of data sources:

- Text files, such as `.txt` and `.dat`
- MAT-files
- Spreadsheet files, such as `.xls`
- 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.

After you finish analyzing your data, you might have created new variables. You can export these variables to a variety of file formats. For more information about exporting data from the MATLAB workspace, see the MATLAB Programming documentation.

When working with time-series data, it is easiest to use the Time Series Tools GUI to import the data and create `timeseries` objects. The Import Wizard in Time Series Tools also makes it easy to import or define a time vector for your data. For more information, see “Importing and Exporting Data” on page 5-8.

Plotting Data

After you import data into MATLAB, 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 MATLAB Graphics documentation fully describes the MATLAB Figure window, which displays the plot. It also discusses the various plot tools that are available in MATLAB to help you annotate and edit plot properties.

If you are working with time-series data, see Chapter 5, “Using Time Series Tools”, 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 1-7
- “Plotting the Data” on page 1-8

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 MATLAB 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.

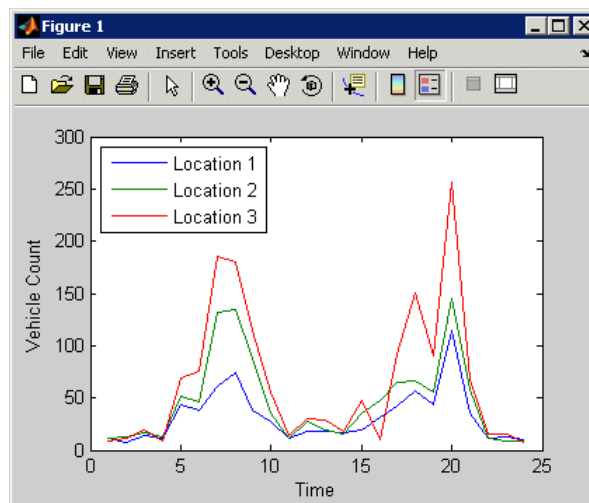
Plotting the Data

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

Removing and Interpolating Missing Values

The correct handling of missing data is a difficult problem in data analysis and often depends on your specific situation. Based on the context of your data, you must decide whether it is appropriate to exclude missing data from analysis or to replace it using a method such as interpolation.

This section contains the following topics:

- “Representing Missing Data Values” on page 1-9
- “Calculating with NaNs” on page 1-9
- “Removing NaNs from the Data” on page 1-10
- “Interpolating Missing Data” on page 1-11

Representing Missing Data Values

In MATLAB, missing or unavailable data values are represented 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 MATLAB variable that contains NaNs, 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 the Data” on page 1-10.

Removing NaNs from the Data

You can use the MATLAB 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
<code>i = find(~isnan(x)); x = x(i)</code>	Find the indices of elements in a vector <code>x</code> that are not NaNs. Keep only the non-NaN elements.
<code>x = x(~isnan(x));</code>	Remove NaNs from a vector <code>x</code> .
<code>x(isnan(x)) = [];</code>	Remove NaNs from a vector <code>x</code> (alternative method).
<code>X(any(isnan(X),2),:) = [];</code>	Remove any rows containing NaNs from a matrix <code>X</code> .

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 Analysis” on page 2-4.

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 MATLAB 5, which does not use `'extrapolate'` and uses `'spline'` when X is not equally spaced

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

```
help interp1
```

Removing Outliers

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)
```

MATLAB 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)
```

MATLAB 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

MATLAB provides functions for working 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.

To learn more about filter applications, see the Signal Processing Toolbox documentation.

This section contains the following topics:

- “Filter Function” on page 1-14
- “Example 1 — Moving-Average Filter” on page 1-15
- “Example 2 — Discrete Filter” on page 1-16

Filter Function

The function

$$y = \text{filter}(b,a,x)$$

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

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

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, $x(n) x(n - 1) \dots$, and previous outputs, $y(n - 1) y(n - 2) \dots$.

Example 1 – 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];
```

Enter the following syntax to load the sample data:

```
load count.dat
```

This adds the matrix `count` to the workspace.

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

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

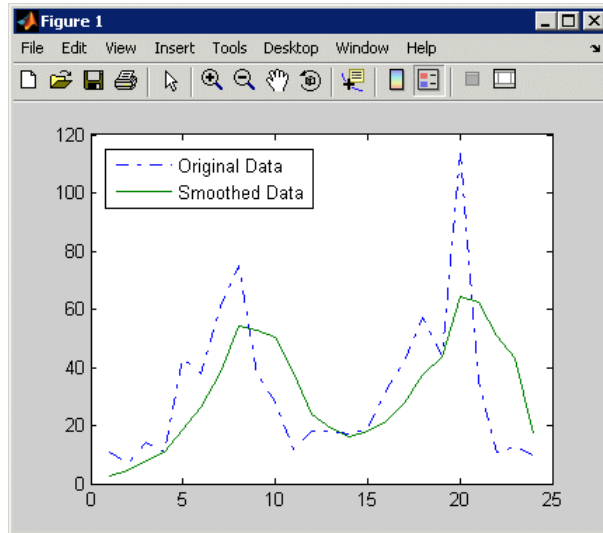
The 4-hour moving average of the data is calculated by

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

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

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

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 2 – 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

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

results in the following transfer function:

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

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 $2 + 2z^{-1}$:

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

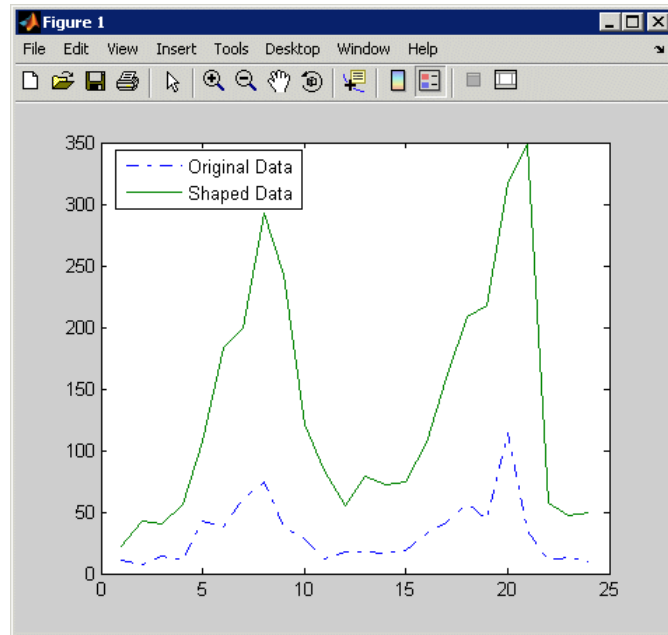
- 5** Call the filter function:

```
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

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, MATLAB detrends each data column 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 the 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 `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 1-20
- “Detrending Data and Plotting Results” on page 1-22

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.

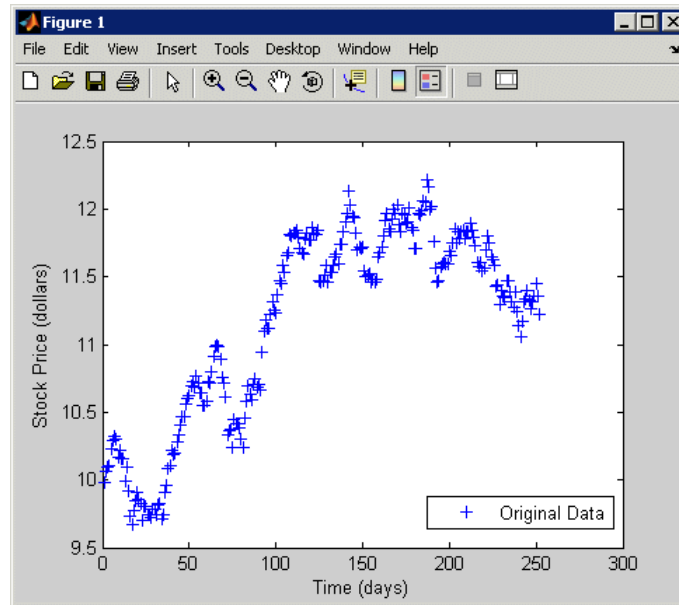
3 Identify and remove the NaN value from `sdata`:

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

For more information about removing NaNs, see “Removing NaNs from the Data” on page 1-10.

4 Plot the data:

```
plot(t, 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.

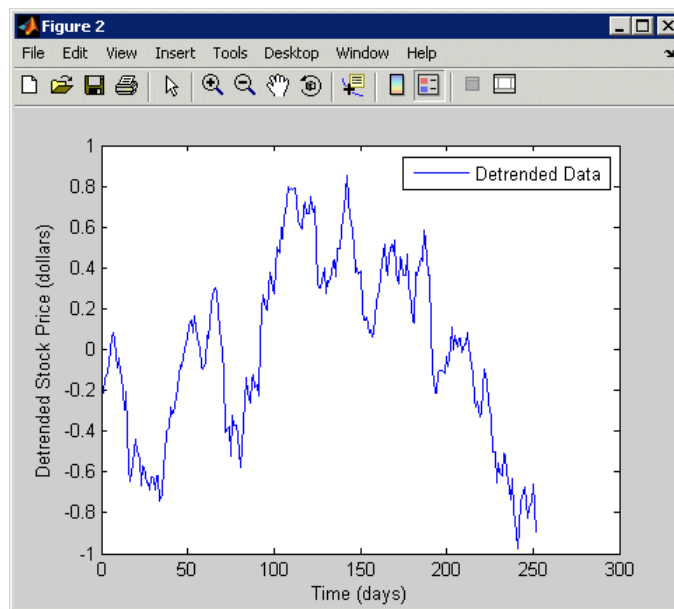
Detrending Data and Plotting Results

- 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.

Finite Differences

MATLAB provides three functions for 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 $[X(2) - X(1) \quad X(3) - X(2) \quad \dots \quad X(n) - X(n-1)]$. 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
<code>any(diff(x)==0)</code>	Tests whether there are any repeated elements in <code>X</code>
<code>all(diff(x)>0)</code>	Tests whether the values are monotonically increasing
<code>all(diff(diff(x))==0)</code>	Tests for equally spaced vector elements

Descriptive Statistics

MATLAB provides a number of functions for calculating descriptive statistics for your data. You can also use the MATLAB Data Statistics dialog box to calculate statistics and plot them with the data in a MATLAB Figure window.

This section contains the following topics:

- “Functions for Calculating Descriptive Statistics” on page 1-25
- “Example — Using MATLAB Data Statistics” on page 1-28

If you need more advanced statistics functionality, you might want to use the Statistics Toolbox. 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, MATLAB calculates descriptive statistics for each column independently.

Statistics Function Summary

Function	Description
max	Maximum value
mean	Average or mean value
median	Median value
min	Smallest value
mode	Most frequent value

Statistics Function Summary (Continued)

Function	Description
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 1-26
- “Example 2 — Subtracting the Mean” on page 1-28

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)
```

MATLAB responds with

```
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)
```

MATLAB responds with this result:

```
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 1-20.

Example – Using MATLAB Data Statistics

MATLAB provides the Data Statistics dialog box to help 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:

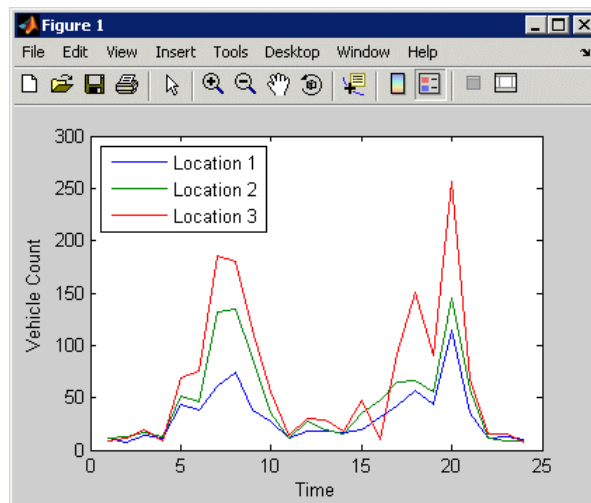
- “Calculating and Plotting Descriptive Statistics” on page 1-29
- “Saving Statistics to the MATLAB Workspace” on page 1-34
- “Formatting Data Statistics on Plots” on page 1-32

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

Calculating and Plotting Descriptive Statistics

1 Load and plot the data:

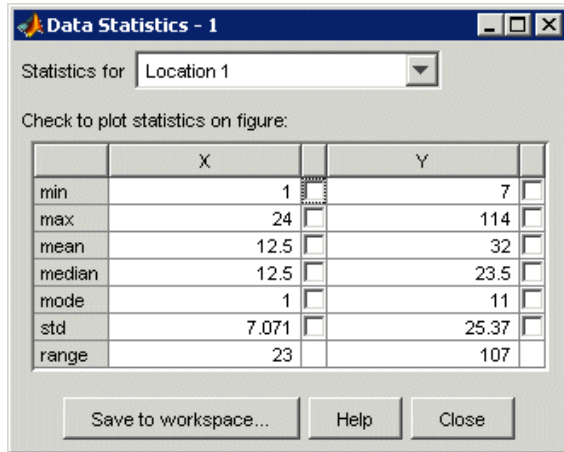
```
load count.dat
[n,p] = size(count);
% Define the x-values
t = 1:n;
% Plot the data and annotate the graph
plot(t,count)
legend('Location 1','Location 2','Location 3',2)
xlabel('Time'), ylabel('Vehicle Count')
```



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, MATLAB assigns them default names: data 1, data 2, and so on.

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

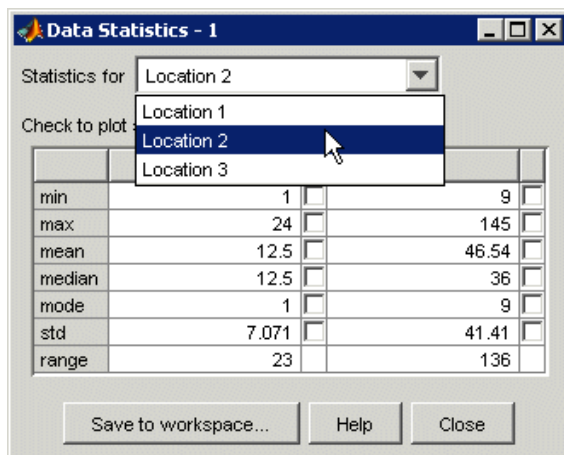
This opens the Data Statistics dialog box, which displays descriptive statistics for the X- and Y-data of the Location 1 data set.



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.

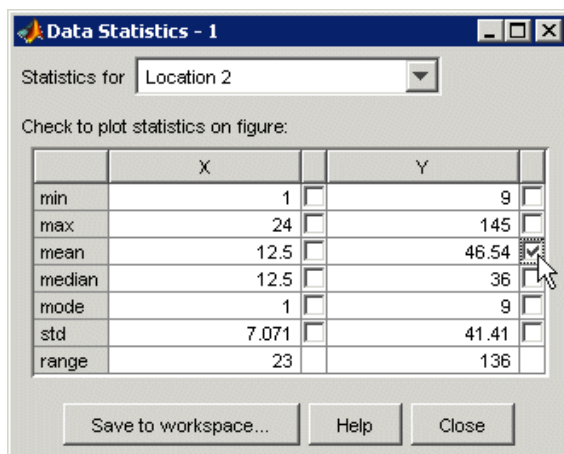
- 3** Select a different data set in the **Statistics for** list: Location 2.

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

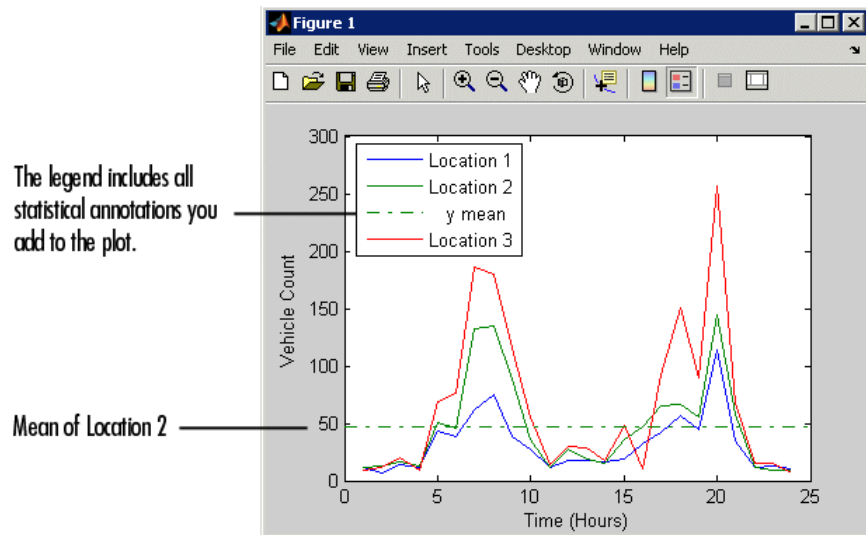


- 4** Select the check box for each statistic you want to display on the plot.

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.

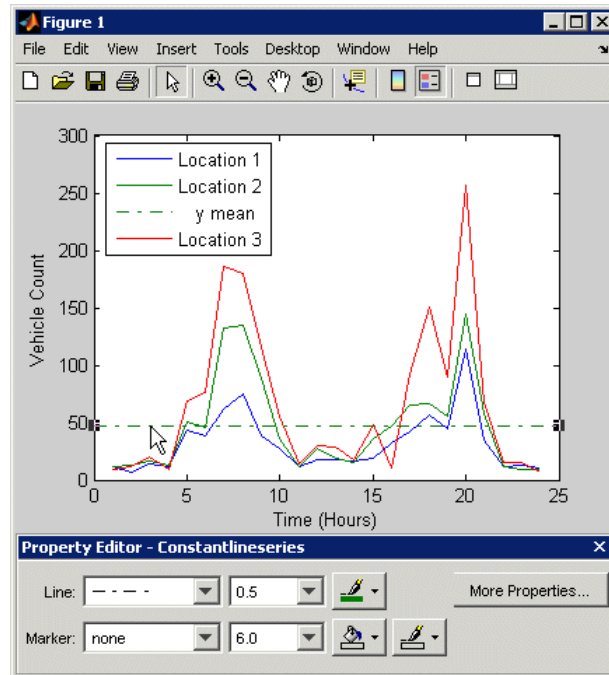
To modify the display of data statistics on a plot:

- 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.

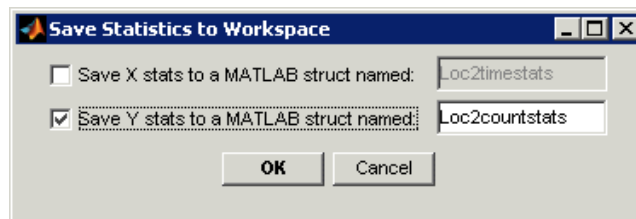
Saving Statistics to the MATLAB Workspace

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
```


Data Fitting Using Linear Regression

The following sections describe how to use correlation analysis in MATLAB to determine if quantities are related, and to model the relationship between pairs of quantities using linear regression.

Introduction (p. 2-2)	Summarizes MATLAB data-fitting capabilities
Linear Correlation Analysis (p. 2-4)	Describes how to calculate correlation coefficients and covariance
Using MATLAB Basic Fitting (p. 2-8)	Describes MATLAB Basic Fitting for fitting polynomial and spline models, generating plots of fitted data and residuals, and saving fit information to the MATLAB workspace
Data Fitting Using MATLAB Functions (p. 2-20)	Describes how to fit polynomials, general linear models, and multiple-regression models using MATLAB functions
Example — Data Fitting Using MATLAB Functions (p. 2-28)	Shows how to use MATLAB functions to fit polynomials, generate plots of fitted data and residuals, and calculate confidence bounds

Introduction

MATLAB allows you to model your data using linear regression. A *model* is a relationship between independent and dependent variables. Linear regression produces a 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.

Before you model the relationship between pairs of quantities, it is a good idea to perform correlation analysis to establish if a relationship exists between these quantities. For more information, see “Linear Correlation Analysis” on page 2-4.

MATLAB provides the Basic Fitting GUI for fitting 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 MATLAB Basic Fitting” on page 2-10. 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 — Data Fitting Using MATLAB Functions” on page 2-28.

If you need to fit nonlinear data using MATLAB, you can try transforming the variables in your model to make the model linear, use the nonlinear algorithm `fminsearch`, or use the Curve Fitting Toolbox (see the Curve Fitting Toolbox documentation).

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 dependent 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, MATLAB minimizes the sum of the squares of the residuals 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, 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. In this case, understanding what your data represents and how it was measured is just as important as evaluating the goodness of fit.

When to Use the Curve Fitting Toolbox

The Curve Fitting Toolbox 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.

Linear Correlation Analysis

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 is a method for establishing the degree of probability that a linear relationship exists between two measured quantities. 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.

MATLAB provides the following three functions for computing correlation coefficients and covariance. In typical data analysis applications, where you are mostly interested in the degree of relationship between variables, you need only to calculate correlation coefficients. That is, it is not necessary to calculate the covariance independently.

Function	Description
corrcoef	Correlation coefficient matrix
cov	Covariance matrix
xcorr (in Signal Processing Toolbox)	Crosscorrelation sequence of a random process (includes autocorrelation)

Covariance

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

In typical data analysis applications, where you are mostly interested in the degree of relationship between variables, you can calculate the correlation coefficients directly without calculating the covariance first.

The covariance matrix has the following properties:

- $\text{cov}(X)$ is symmetrical.
- $\text{diag}(\text{cov}(X))$ is a vector of variances for each data column, which represent a measure of the spread or dispersion of data in the corresponding column.
- $\text{sqrt}(\text{diag}(\text{cov}(X)))$ 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:

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

$$\sigma_{ij}^2 = \sigma_{ji}^2$$

Here, σ_{ij}^2 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.

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 2-4.

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.

Using MATLAB Basic Fitting

MATLAB provides the Basic Fitting GUI to help you fit polynomial and spline models to your data, plot the model on top of your data, and extract model coefficients.

This section contains the following topics:

- “What Is MATLAB Basic Fitting?” on page 2-8
- “Sorting Data to Improve Performance” on page 2-8
- “Opening MATLAB Basic Fitting” on page 2-9
- “Example — Using MATLAB Basic Fitting” on page 2-10

What Is MATLAB Basic Fitting?

MATLAB Basic Fitting provides a graphical user interface (GUI) to help you perform the following data-fitting tasks:

- Model the data using a spline interpolant, a shape-preserving interpolant, or a polynomial (up to the tenth degree)
- Plot one or more fits overlaying the data and the residuals
- Get model coefficients and the norm of the residuals (a goodness-of-fit indicator)
- Interpolate or extrapolate data by using the data model
- Save the fit and evaluated results to the MATLAB workspace

Note MATLAB Basic Fitting is available only for 2-D plots.

Sorting Data to Improve Performance

If your data set is large and the values are not sorted in ascending order, it takes longer to fit your data. This can occur because Basic Fitting sorts the values first.

In some cases, you can speed up the fitting process by presorting the data before you open the Basic Fitting GUI.

For example, use the following syntax to create sorted vectors, `x_sorted` and `y_sorted`, from the original data vectors `x` and `y`:

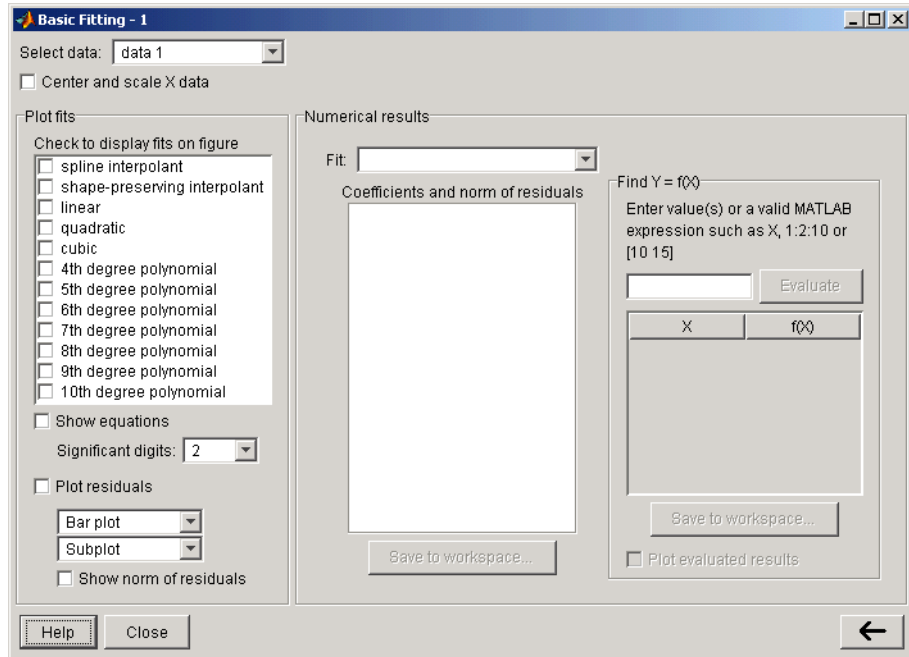
```
[x_sorted, i] = sort(x);  
y_sorted = y(i);
```

Opening MATLAB Basic Fitting

You open the Basic Fitting dialog box from the MATLAB Figure window where you plotted data by selecting **Tools > Basic Fitting**. To get detailed instructions about working with Basic Fitting, click **Help**.

MATLAB Basic Fitting enables you to do the following:

- Select the data you want to fit from a list of data sets currently in the MATLAB Figure window.
- Plot a model on top of the data, display the model equations, and plot the residuals.
- Display model coefficients.
- Interpolate and extrapolate ordinate values using the selected model.



Example – Using MATLAB Basic Fitting

In this example, you use MATLAB Basic Fitting to perform the following tasks:

- “Loading and Plotting the Data” on page 2-10
- “Fitting the Data” on page 2-12
- “Viewing and Saving Fit Parameters” on page 2-16
- “Interpolating Using the Model” on page 2-17

Loading and Plotting the Data

You will use data in the `census.mat` file for this example, which 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')
```

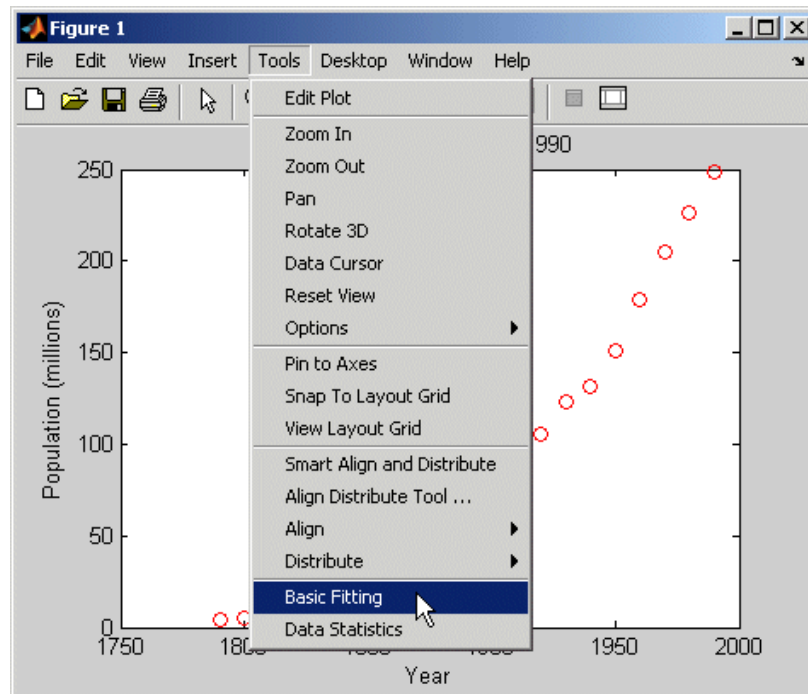
This adds the following two variables to the MATLAB workspace and plots them:

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

Now you are ready to fit the data.

Fitting the 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.
```

To improve model accuracy, select the **Center and scale X data** check box. For more information about this option, click the **Help** button.

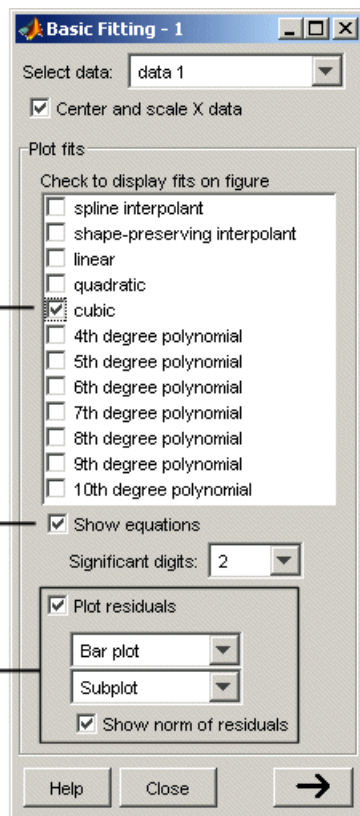
Note After centering and scaling the independent variable (X), the values of the fitted coefficients are changed from those obtained using the original data. This does not change the functional form of the model and the norm of the residuals. Furthermore, the plot still shows the original, unscaled X values.

- 3** Select the options that do the following:
- Display the model equation in the plot.
 - Plot the residuals as a subplot in the MATLAB Figure window that shows the data.
 - Display the norm of the residuals.

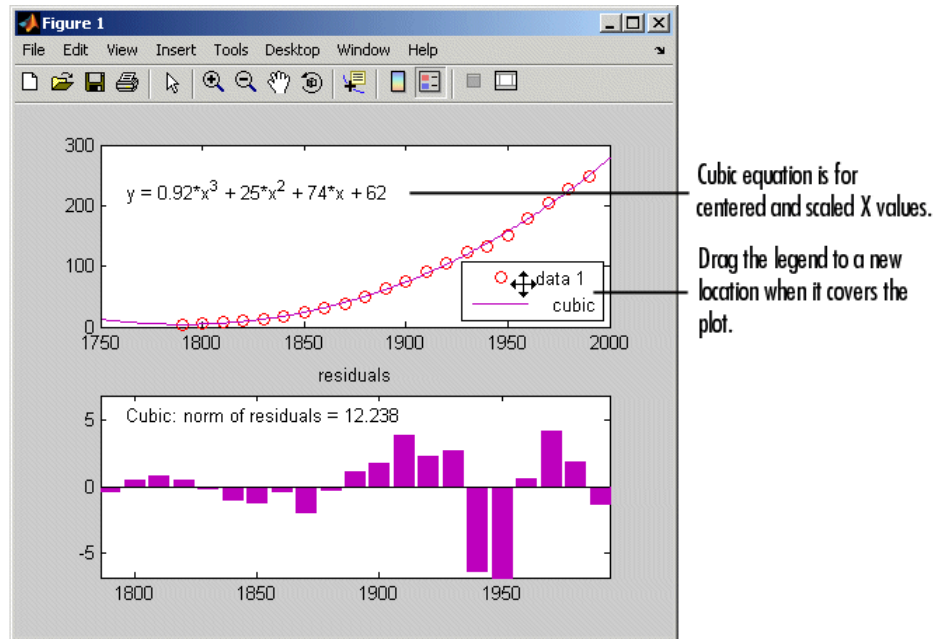
Fit a cubic polynomial
to the selected data set.

Show equations on the plot
with 2 significant digits
in the coefficients.

Plot residuals as a Bar
subplot and show the
norm of the residuals.



The resulting fit and residuals are shown in the following plot:




The cubic fit does not appear to be adequate before the year 1790, where it indicates a decreasing population. However, the model seems to approximate the data reasonably well after 1790. A pattern in the residuals indicates that this fit might not be appropriate for modeling this data. However, this decision must be made in the context of the data to determine if a model is useful.

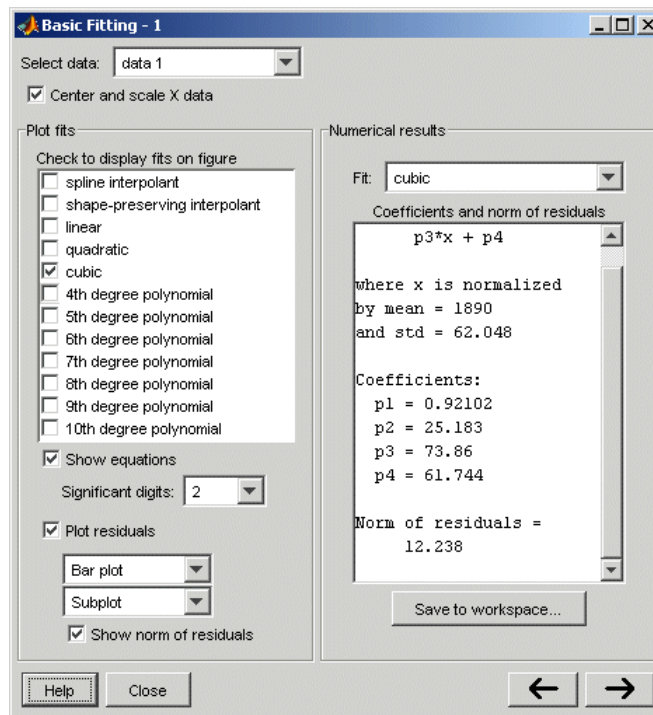
The legend contains the name of the data set and the fit equation. A *data set* refers to a column of data in each array you plot. Each data set is assigned a default name in the Basic Fitting GUI (for example, data 1 and data 2). In this example, you only have one data set — data 1.

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 by using the Property Editor. These changes are undone when you select to fit another model.

Viewing and Saving Fit Parameters

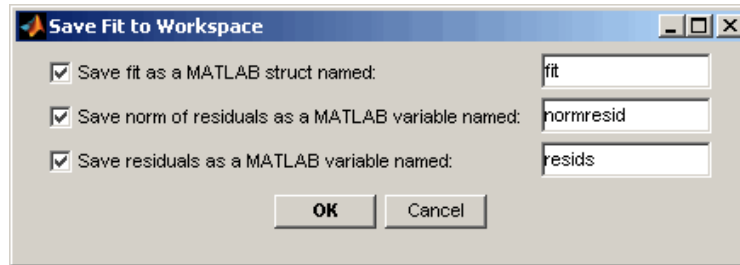
In the Basic Fitting dialog box, click the  button to display the estimated coefficients and the norm of the residuals (an indicator of the goodness of fit) in the **Numerical results** area.



To view the coefficients for a specific fit, select the type of fit from the **Fit** list. This displays the results in the Basic Fitting dialog box, but does not plot them in the Figure window.

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

You can save the fit data to the MATLAB workspace by clicking the **Save to workspace** button in the **Numerical results** area. 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]
```

Notice that the coefficients are based on the centered and scaled X values. For more information about centering and scaling for improved accuracy of the fit, see “Fitting the Data” on page 2-12.

Tip You can call this structure in subsequent analysis.

Interpolating Using the Model

In this portion of the example, you will interpolate the population in 1965 using the cubic model.

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

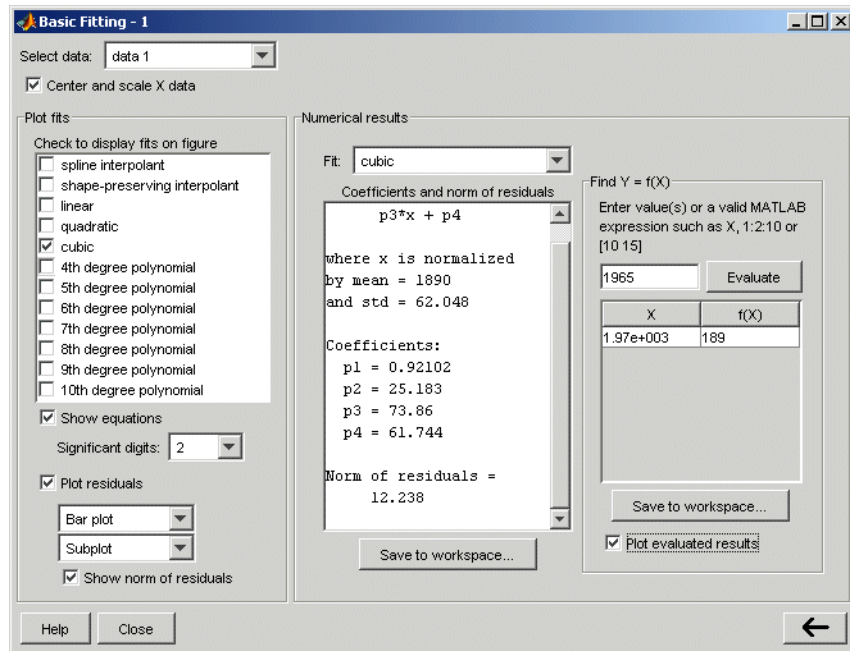
1 In the **Enter value(s)...** field, type the following value:

1965

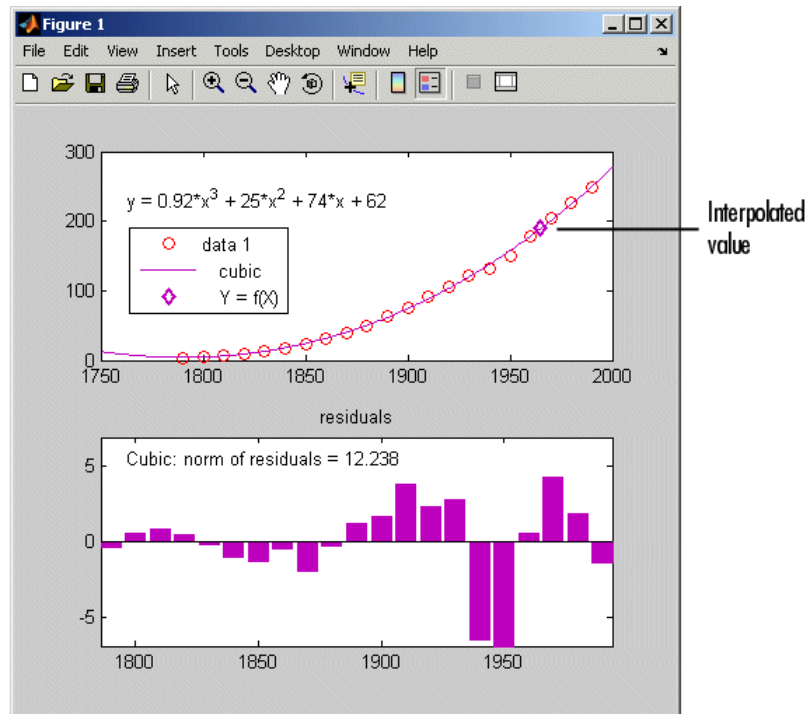
Note Use an unscaled and uncentered X value. You do not need to center and scale it first, even though you selected to scale X values to obtain the coefficients in “Fitting the Data” on page 2-12. Basic Fitting performs the necessary calculations behind the scenes.

2 Click **Evaluate**.

The X values and the corresponding values for $f(X)$ are evaluated from the fit and displayed in a table, as shown below:

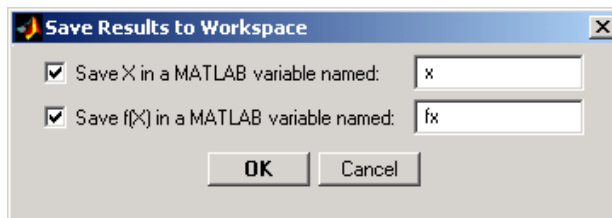


- 3 Select the **Plot evaluated results** check box to display the interpolated value:



- 4 Save the interpolated population (fx value) in 1965 (x value) to the MATLAB workspace by clicking **Save to workspace**.

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



Data Fitting Using MATLAB Functions

This section contains the following topics:

- “MATLAB Functions for Polynomial Models” on page 2-20
- “Linear Model with Nonpolynomial Terms” on page 2-24
- “Multiple Regression” on page 2-26

MATLAB Functions for Polynomial Models

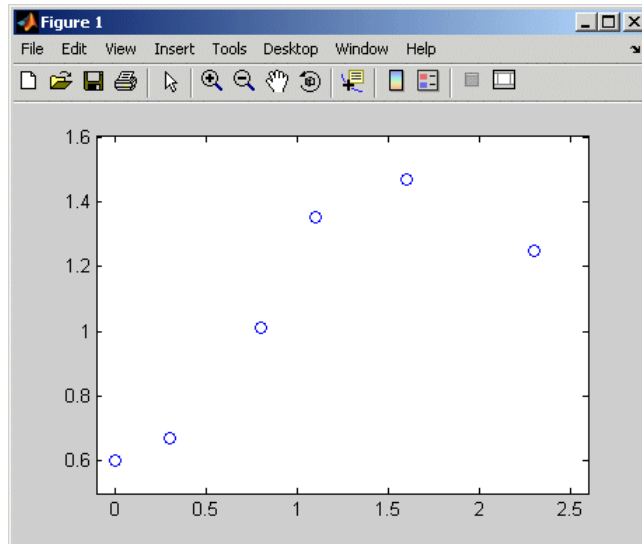
MATLAB provides two functions for modeling your data with a polynomial.

Polynomial Fit Functions

Function	Description
<code>polyfit</code>	<code>polyfit(x,y,n)</code> 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).
<code>polyval</code>	<code>polyval(p,x)</code> returns the value of a polynomial of degree n that was determined by <code>polyfit</code> , 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:

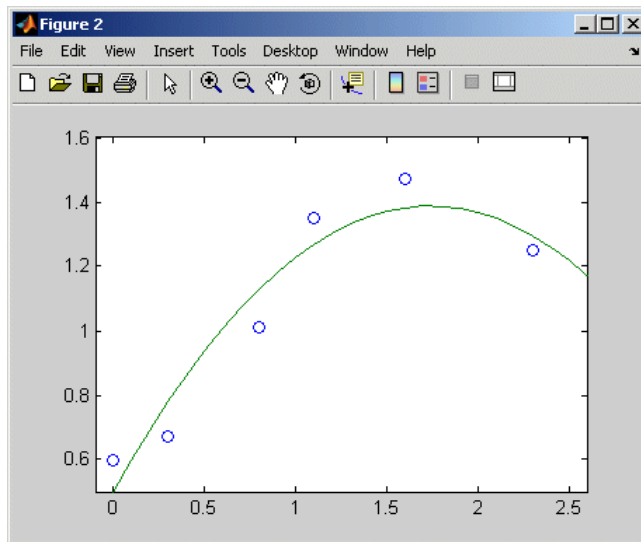
```
p =  
-0.2942    1.0231    0.4981
```

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

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

To plot the model with the data, evaluate the polynomial at uniformly spaced times t_2 and overlay the original data on a plot:

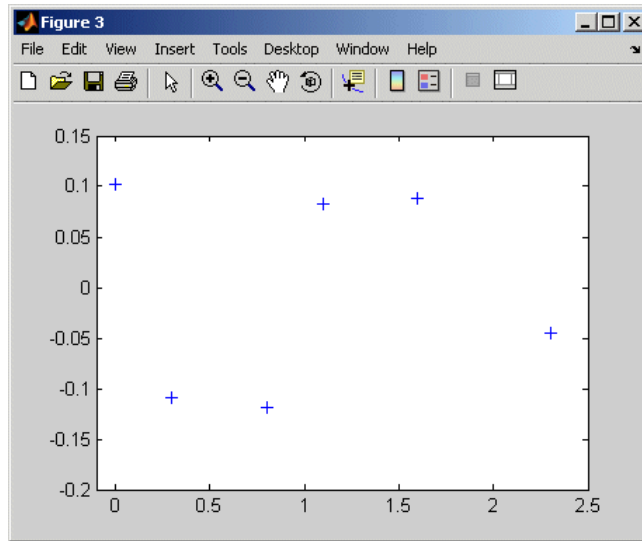
```
t2 = 0:0.1:2.8;    % Define a uniformly spaced time vector
y2=polyval(p,t2);  % Evaluate the polynomial at t2
figure
plot(t,y,'o',t2,y2) % Plot the fit on top of the data
                    % in a new Figure window
```



Plot of Data (Points) and Model (Line)

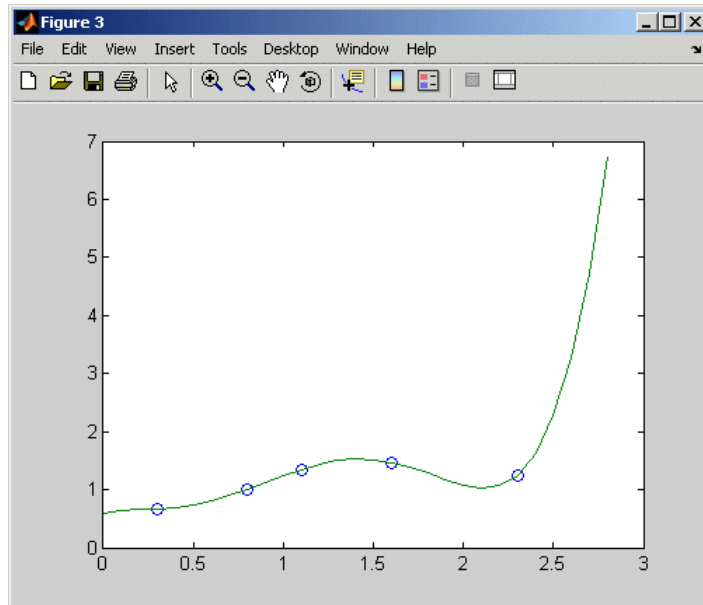
Use the following syntax to calculate the residuals:

```
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:

$$y = a_0 + a_1e^{-t} + a_2te^{-t}$$

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

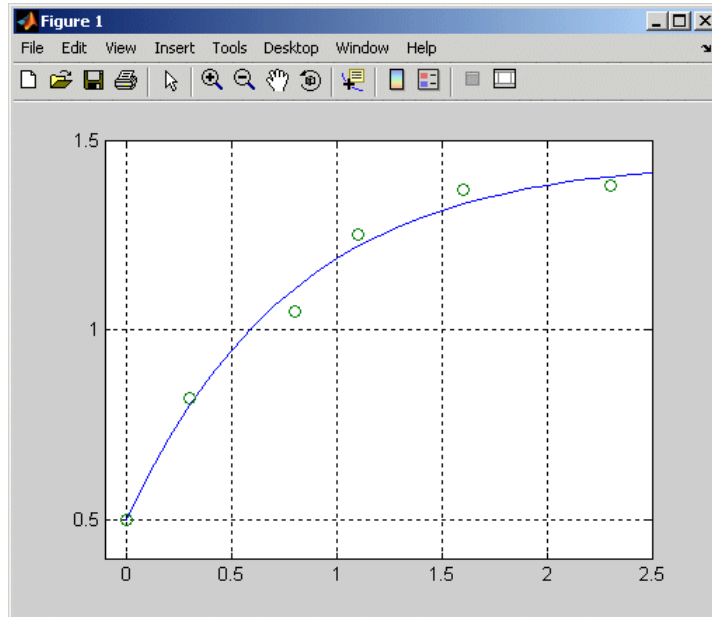
a =
    1.3983
   -0.8860
    0.3085
```

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 independent 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

$$y = a_0 + a_1x_1 + a_2x_2$$

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 – Data Fitting Using MATLAB Functions

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

- “Calculating Correlation Coefficients” on page 2-29
- “Fitting a Polynomial to the Data” on page 2-30
- “Plot and Calculate Confidence Bounds” on page 2-32

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

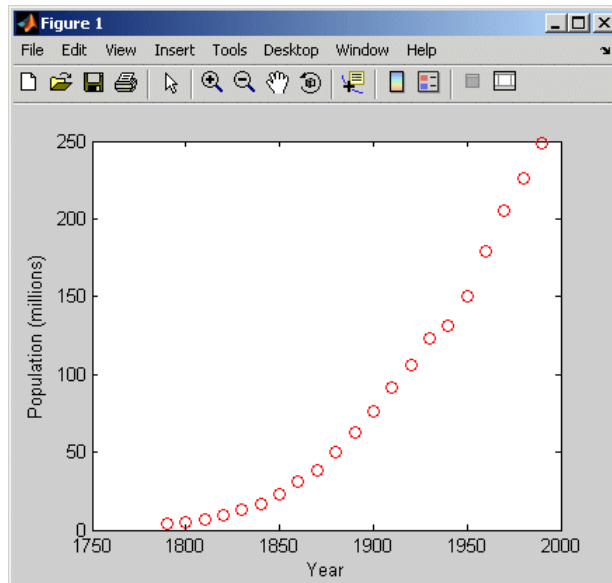
To load and plot the data, type the following commands at the MATLAB prompt:

```
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 Analysis” on page 2-4.

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
```

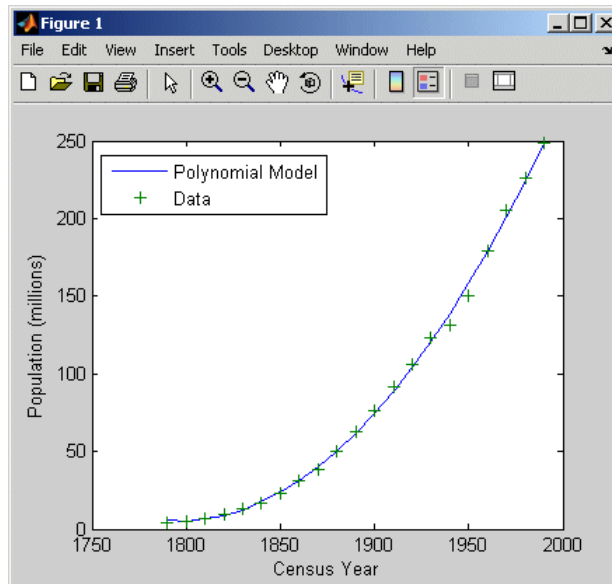
The diagonal matrix elements represent the perfect correlation of each variable with itself and are equal to 1. The off-diagonal elements are very close to 1, indicating that there is a strong statistical correlation between the variables `cdate` and `pop`.

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)');
```

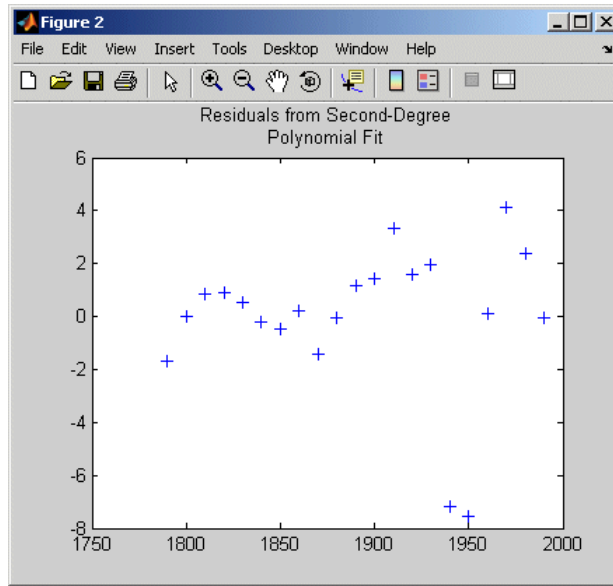
The following figure shows that the quadratic-polynomial fit provides a good approximation to the data:



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

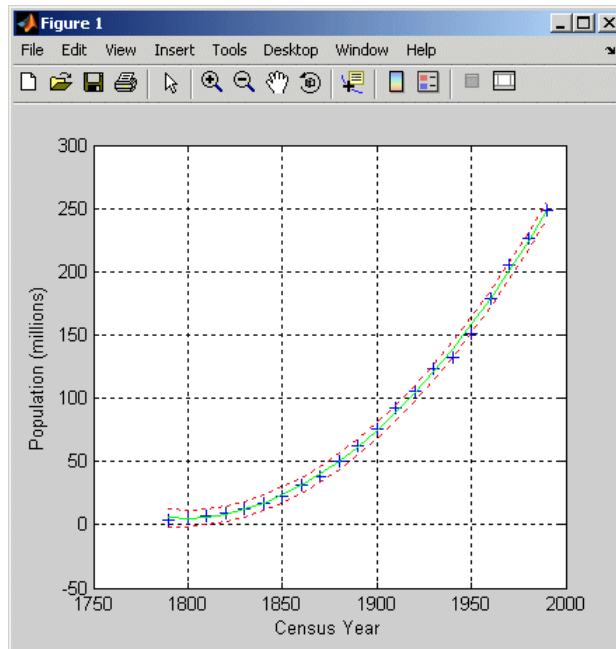
Confidence bounds are confidence intervals for a predicted response. The width of the interval indicates the degree of certainty of the fit.

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

Fourier Analysis

The following sections describe how to perform Fourier analysis in MATLAB for gaining insight into periodic signals.

Introduction (p. 3-2)	Provides an overview of MATLAB Fourier analysis capabilities
Function Summary (p. 3-3)	Summarizes functions for computing and manipulating Fourier transforms
Calculating Fourier Transforms (p. 3-4)	Describes how to calculate a Fourier transform and provides an example
Example — Using FFT to Calculate Sunspot Periodicity (p. 3-7)	Shows how to determine the periodicity in sunspot data
Magnitude and Phase of Transformed Data (p. 3-11)	Describes how to calculate the magnitude and phase of transformed data
FFT Length Versus Performance (p. 3-13)	Describes how to improve performance by changing the length of the Fourier transform

Introduction

Fourier analysis is particularly useful in areas such as signal and image processing, filtering, convolution, frequency analysis, and power spectrum estimation.

Fourier analysis provides insight into the periodicities in data by representing the data using a linear combination of sinusoidal components with different frequencies. The amplitude and phase of each sinusoidal component in the sum determines the relative contribution of that frequency component to the entire signal.

For discretely-sampled data, Fourier analysis is performed using the discrete Fourier transform (DFT). MATLAB calculates the DFT of a data sequence by applying the fast Fourier transform (FFT) algorithms; the FFT is an efficient computational method and not a different kind of transform.

To learn about more advanced power-spectrum methods, see the Signal Processing Toolbox documentation.

Function Summary

MATLAB provides the following functions for computing and working with Fourier transforms.

FFT Function Summary

Function	Description
abs	Absolute value and complex magnitude
angle	Phase angle
cplxpair	Sort numbers into complex conjugate pairs
fft	One-dimensional discrete Fourier transform, computed with a fast Fourier transform (FFT) algorithm
fft2	Two-dimensional discrete Fourier transform
fftn	N-dimensional discrete Fourier transform
fftshift	Shift DC component of the discrete Fourier transform to the center of spectrum
ifft	Inverse one-dimensional discrete Fourier transform
ifft2	Inverse two-dimensional discrete Fourier transform
ifftn	Inverse N-dimensional discrete Fourier transform
ifftshift	Inverse FFT shift
nextpow2	Next higher power of 2
unwrap	Unwrap phase angle in radians

Calculating Fourier Transforms

MATLAB performs Fourier analysis by computing the discrete Fourier transform (DFT) using the fast Fourier transform (FFT) algorithms, which improve computational performance.

Consider an input sequence $x(n)$ of length N . The DFT of this sequence is given by the vector $X(k)$, as follows:

$$X(k) = \sum_{n=1}^N x(n)e^{-j2\pi(k-1)\left(\frac{n-1}{N}\right)} \quad 1 \leq k \leq N$$

You use the `fft` function in MATLAB to compute the spectrum. The length of $X(k)$ is the same as the length of $x(n)$. Notice that the value of $X(1)$ equals the sum of the data values in $x(n)$.

Note Traditional Fourier equations have summations from 0 to $N - 1$. However, because the first element of a MATLAB vector has the index of 1, the summations in the above equations are from 1 to N and are equivalent to traditional equations.

For a discrete input sequence, there is an upper limit on the frequency at which you can get meaningful information about the periodicities in the data. The highest frequency that can be uniquely fit to the data, called the *Nyquist frequency*, equals one cycle every two successive measurements. This makes sense because you cannot get information about the variations in the data at frequencies higher than the sampling rate — the rate at which you measured successive data values. For example, suppose that your data consists of daily temperature measurements in your town; here, 1 cycle (the time between two successive measurements) equals 1 day and the Nyquist frequency is 0.5 cycle/day. In this situation, you can only determine variations in the temperature from one day to the next. If you want to study temperature fluctuations during the day, you must collect the data at more frequent intervals.

The lowest frequency that can be uniquely fit to the data, called the *fundamental frequency*, is one cycle for the entire length of the data vector.

The inverse DFT of a transformed sequence is given by:

$$x(n) = \frac{1}{N} \sum_{k=1}^N X(k) e^{j2\pi(k-1)\left(\frac{n-1}{N}\right)} \quad 1 \leq n \leq N$$

You use the `ifft` function in MATLAB to synthesize the signal from its spectrum.

When $x(n)$ is real, you can rewrite the synthesis equation as a sum of sine and cosine functions with real coefficients:

$$x(n) = \frac{1}{N} \sum_{k=1}^N a(k) \cos\left(\frac{2\pi(k-1)(n-1)}{N}\right) + b(k) \sin\left(\frac{2\pi(k-1)(n-1)}{N}\right)$$

where

$$\begin{aligned} a(k) &= \text{real}[X(k)] \\ b(k) &= -\text{imag}[X(k)] \\ 1 &\leq n \leq N \end{aligned}$$

Example – Calculating the FFT of a Column Vector

Consider the following column vector:

$$x = [4 \ 3 \ 7 \ -9 \ 1 \ 0 \ 0 \ 0]';$$

In this example, the length of the input sequence $N = 8$. The Nyquist frequency is 1 cycle every 2 observations, or 0.5. The index of the component k at the Nyquist frequency is determined by setting the frequency to the Nyquist frequency value:

$$f = \frac{\omega}{2\pi} = \frac{k-1}{N} = 0.5$$

Compute the FFT of x as follows:

```
y = fft(x)
```

MATLAB responds with the following FFT vector:

```
y =  
6.0000  
11.4853 - 2.7574i  
-2.0000 -12.0000i  
-5.4853 +11.2426i  
18.0000  
-5.4853 -11.2426i  
-2.0000 +12.0000i  
11.4853 + 2.7574i
```

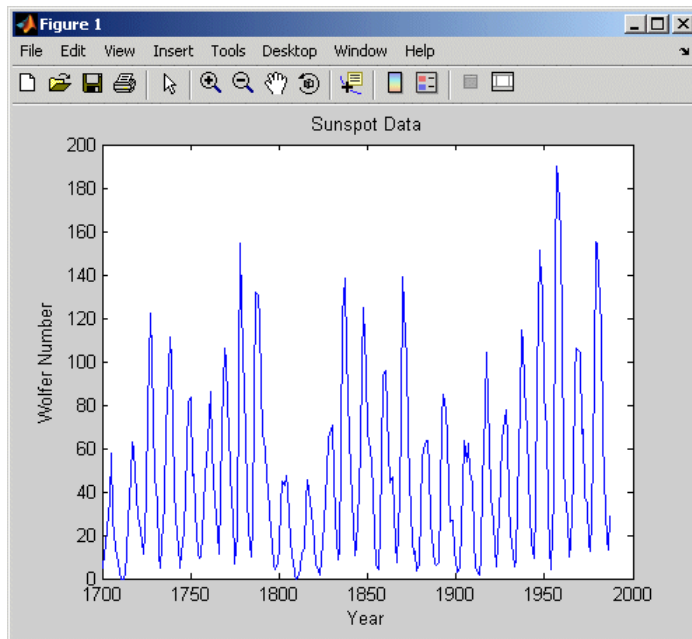
Notice that although the input sequence x is real, y is complex. The first element of y is the sum of the data values. The fifth element corresponds to the contribution at the Nyquist frequency. The last three values of y correspond to negative frequencies and, for the real sequence x , they are complex conjugates of three components in the first half of y .

Example — Using FFT to Calculate Sunspot Periodicity

In this example, you use the MATLAB `fft` function to analyze the variations in sunspot activity. You will use data collected by astronomers for almost 300 years of a quantity called the Wolfer number, which measures both the number and the size of sunspots.

Load and plot the sunspot data:

```
load sunspot.dat
year = sunspot(:,1);
wolfer = sunspot(:,2);
plot(year,wolfer)
title('Sunspot Data')
```



Sunspot Data

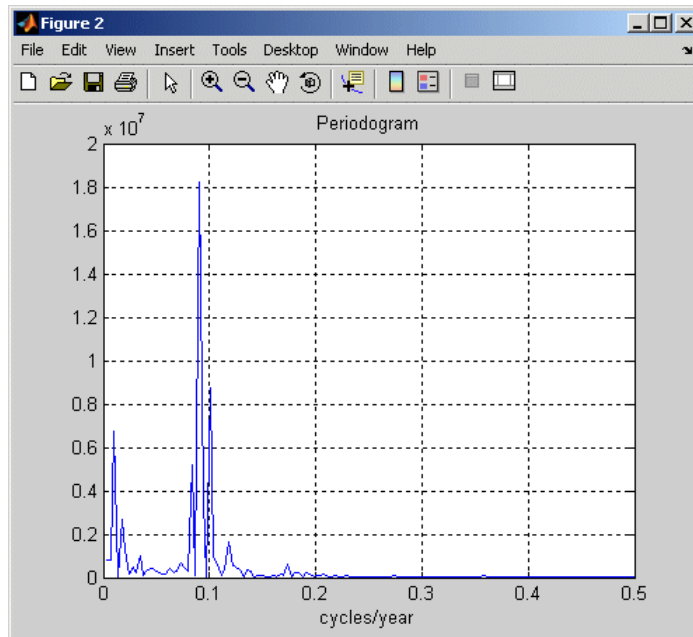
Take the FFT of the sunspot data:

```
Y = fft(wolfer);
```

The result of this transform is the complex vector Y . The magnitude of Y squared is called the estimated power spectrum. A plot of the estimated power spectrum versus frequency is called a *periodogram*.

Because the first component of Y , which is simply the sum of the data, has a large magnitude, the following syntax removes it before generating the periodogram:

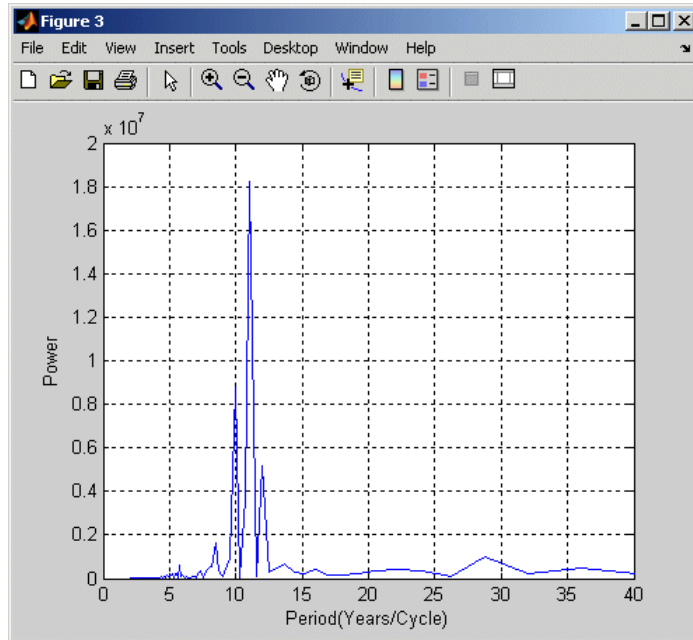
```
N = length(Y);  
Y(1) = [];  
power = abs(Y(1:N/2)).^2;  
nyquist = 1/2;  
freq = (1:N/2)/(N/2)*nyquist;  
plot(freq,power), grid on  
xlabel('cycles/year')  
title('Periodogram')
```



Periodogram of Sunspot Data

The frequency scale is in cycles/year, which is inconvenient because for estimating the period of one cycle in years. Therefore, plot the power versus period (where $\text{period} = 1./\text{freq}$) from 0 to 40 years/cycle:

```
period = 1./freq;
plot(period,power), axis([0 40 0 2e7]), grid on
ylabel('Power')
xlabel('Period(Years/Cycle)')
```



Power Spectrum Versus Period of Sunspot Data

In order to determine the cycle more precisely, use the following syntax:

```
[mp,index] = max(power);  
period(index)
```

```
ans =  
    11.0769
```

This plot confirms the cyclical nature of sunspot activity, which reaches a maximum about every 11 years.

Magnitude and Phase of Transformed Data

Important information about a transformed data sequence includes its magnitude and phase. The MATLAB functions `abs` and `angle` calculate this information.

To try this, create a time vector `t`, and use this vector to create a sequence `x` consisting of two sinusoids at different frequencies:

```
t = 0:1/100:10-1/100;  
x = sin(2*pi*15*t) + sin(2*pi*40*t);
```

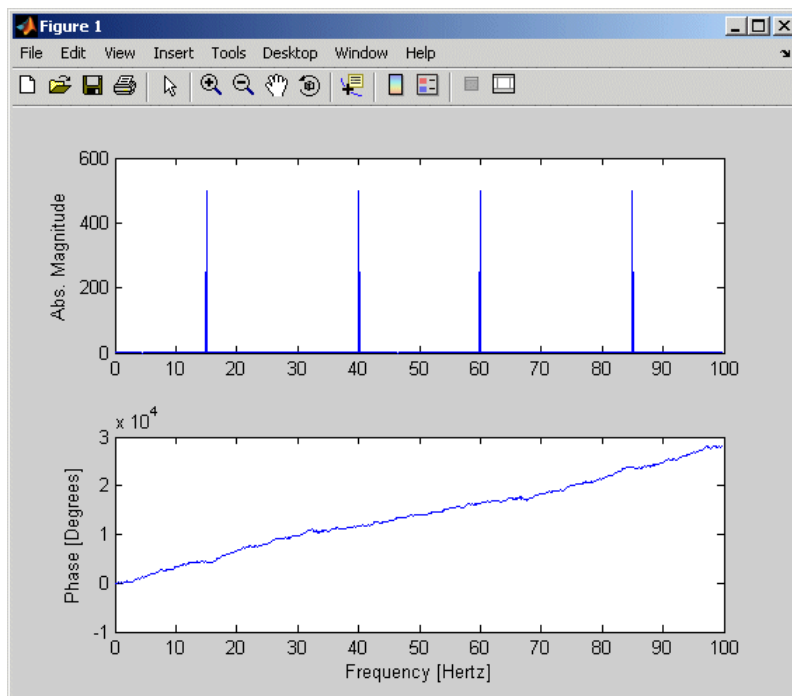
Now use the `fft` function to compute the DFT of the sequence. The following code calculates the magnitude and phase of the transformed sequence. It also uses the `abs` function to obtain the magnitude of the data, the `angle` function to obtain the phase information, and the `unwrap` function to remove phase jumps greater than π to their 2π complement:

```
y = fft(x);  
m = abs(y);  
p = unwrap(angle(y));
```

Now create a frequency vector for the x -axis and plot the magnitude and phase:

```
f = (0:length(y)-1)'*100/length(y);  
subplot(2,1,1), plot(f,m),  
ylabel('Abs. Magnitude'), grid on  
subplot(2,1,2), plot(f,p*180/pi)  
ylabel('Phase [Degrees]'), grid on  
xlabel('Frequency [Hertz]')
```

The magnitude plot is perfectly symmetrical about the Nyquist frequency of 50 Hz. The useful information in the signal is found in the range 0 to 50 Hz. For more information about the Nyquist frequency, see “Calculating Fourier Transforms” on page 3-4.



Magnitude and Phase Information in Transformed Data

FFT Length Versus Performance

The execution time for the `fft` depends on the length of the transform.

You can add a second argument to `fft` to specify a number of points `n` in the transform:

```
y = fft(x,n)
```

With this syntax, `fft` pads `x` with 0s if it is shorter than `n`, or truncates it if it is longer than `n`. If you do not specify `n`, `fft` defaults to the length of the input sequence. `fft` is fastest for powers of 2. It is almost as fast for lengths that have only small prime factors. It is typically several times slower for lengths that are prime or have large prime factors.

The inverse FFT function `ifft` also accepts a transform length argument.

Using Time-Series Objects and Methods

The following sections describe how to analyze time-series data using MATLAB objects and methods.

Introduction (p. 4-2)	Summarizes the MATLAB <code>timeseries</code> and <code>tscollection</code> objects
Time-Series Data Sample (p. 4-3)	Defines a <i>data sample</i> for the <code>timeseries</code> constructor
Example — Using Time-Series Objects and Methods (p. 4-6)	Provides an example of creating and performing basic operations on <code>timeseries</code> and <code>tscollection</code> objects
<code>timeseries</code> Constructor (p. 4-18)	Describes the <code>timeseries</code> constructor syntax and object properties
<code>timeseries</code> Methods (p. 4-28)	Summarizes commonly used <code>timeseries</code> methods
<code>tscollection</code> Constructor (p. 4-33)	Describes the <code>tscollection</code> constructor syntax and object properties
<code>tscollection</code> Methods (p. 4-36)	Summarizes commonly used <code>tscollection</code> methods

Introduction

MATLAB provides methods for analyzing time-series data. These methods operate on the following MATLAB objects:

- `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

In this chapter, you learn how to

- Use time-series constructors to instantiate time-series classes
- Modify object properties using set methods or dot notation
- Call time-series functions and methods

To get a quick overview of programming with `timeseries` and `tscollection` objects, follow the steps in “Example — Using 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 Chapter 5, “Using Time Series Tools”.

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 5-6.

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 “`timeseries` Constructor Syntax” on page 4-19. 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 “`tscollection` Constructor Syntax” on page 4-33.

Example – Using Time-Series Objects and Methods

Follow the steps in this example to learn how to work with the `timeseries` and `tscollection` functions and methods.

This example illustrates the following typical operations you perform on time-series data:

- “Creating `timeseries` Objects” on page 4-6
- “Modifying `timeseries` Units and Interpolation Method” on page 4-8
- “Defining Events” on page 4-9
- “Creating `tscollection` Objects” on page 4-10
- “Resampling a `tscollection` Object” on page 4-11
- “Adding a Data Sample to a `tscollection` Object” on page 4-12
- “Removing and Interpolating Missing Data” on page 4-13
- “Removing a `timeseries` from a `tscollection`” on page 4-15
- “Changing a Numerical Time Vector to Date Strings” on page 4-16
- “Plotting `tscollection` Members” on page 4-17

Creating `timeseries` Objects

This portion of the example illustrates how to create several `timeseries` objects from an array. For more information about the `timeseries` object, see “`timeseries` Constructor” on page 4-18.

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 responds by displaying 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');
```

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 timeseries Units and Interpolation Method” on page 4-8.

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.

Modifying timeseries Units and Interpolation Method

After creating the `timeseries` object, as described in “Creating timeseries Objects” on page 4-6, you can modify its units and interpolation method by 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.datametadate]
Time: [24x1 double]
TimeInfo: [1x1 tsdata.timemetadate]
Quality: []
QualityInfo: [1x1 tsdata.qualmetadate]
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 `tscollection` 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 “`tscollection` 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 `timeseries` Objects” on page 4-6.

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)
```

MATLAB now lists all three members in the collection:

```
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 `tscollection` 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 timeseries Units and Interpolation Method” on page 4-8.

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

Adding a Data Sample to a `tscollection` 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

NaNs for these members. To learn how to remove or interpolate missing data values, see “Removing and Interpolating Missing Data” on page 4-13.

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

MATLAB uses NaNs to represent missing data in a time series. 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 `tscollection` Object” on page 4-12, 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 NaNs.

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 “`timeseries` Properties” on page 4-21.

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 “`tscollection` Properties” on page 4-34.

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
3.0	14	17	20
3.25	5	16	17.3
3.5	14	15	14.5

Removing a timeseries from a `tscollection`

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

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

MATLAB now lists two time series as members in the collection:

```
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 4-18.

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

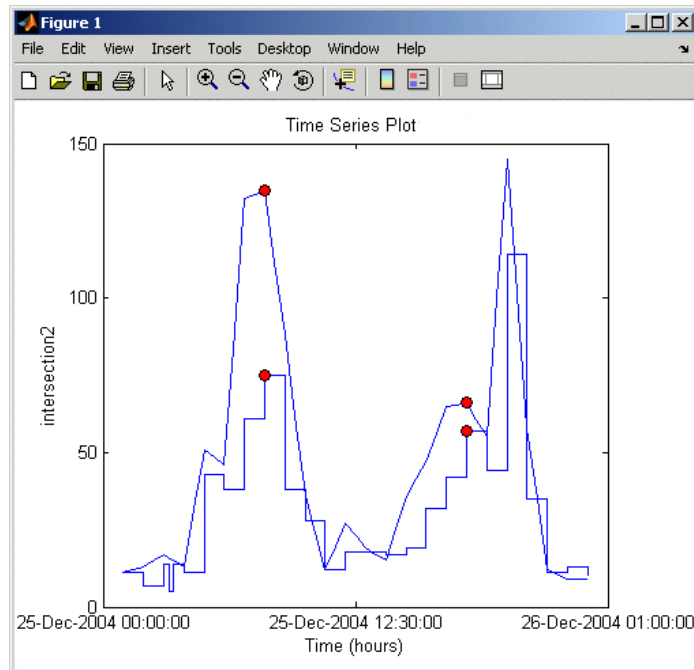
```
Time Series Object: unnamed
Time vector characteristics
    Length          1
    Start date      25-Dec-2004 03:15:00
    End date        25-Dec-2004 03:15:00
Data characteristics
    Interpolation method linear
    Size              [1 1]
    Data type         double
Time                Data                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 tscollection 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, `intesection2` 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 4-9.

timeseries Constructor

The MATLAB object called `timeseries` is a MATLAB variable that stores time-indexed data and data properties in a single structure. In addition to storing the data and time values, you can use the `timeseries` object to store events, descriptive information about data and time, data quality, and the interpolation method.

This section contains the following topics:

- “Time Vector Format” on page 4-18
- “timeseries Constructor Syntax” on page 4-19
- “timeseries Properties” on page 4-21

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 “timeseries Properties” on page 4-21.

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

Date-String Format	Usage Example
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

For an example of how to represent a numerical time vector relative to calendar dates, see “Changing a Numerical Time Vector to Date Strings” on page 4-16.

timeseries 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 timeseries Objects” on page 4-6.

timeseries Syntax Descriptions

Syntax	Description
<code>ts = timeseries</code>	Creates an empty <code>timeseries</code> object. The size of this object is 0-by-1.

timeseries Syntax Descriptions (Continued)

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

timeseries Syntax Descriptions (Continued)

Syntax	Description
<pre>ts = timeseries(Data,Time,Quality)</pre>	<p>The Quality attribute is an integer vector containing values -128 to 127 that specifies the quality in terms of codes defined by <code>QualityInfo.Code</code>.</p> <p>For more information about <code>QualityInfo</code>, see “timeseries Properties” on page 4-21.</p>
<pre>ts = timeseries(Data,...,'Parameter Value, ...)</pre>	<p>Optionally enter the following parameter-value pairs after the Data, Time, and Quality arguments. You can specify the following parameters:</p> <ul style="list-style-type: none"> • Name • IsTimeFirst • IsDatenum <p>Name and IsTimeFirst are described in “timeseries Properties” on page 4-21.</p> <p>When set to true, IsDatenum specifies that Time values are dates in the format of MATLAB serial dates.</p>

timeseries 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 `timeseries` Units and Interpolation Method” on page 4-8.

timeseries Property Descriptions

Property	Description
Data	<p>Time-series data, where each data sample corresponds to a specific time.</p> <p>The data can be a scalar, a vector, or a multidimensional array. Either the first or last dimension of the data must align with <code>Time</code>.</p> <p>By default, NaNs represent missing or unspecified data. Set the <code>TreatNaNasMissing</code> property to determine how missing data is treated in calculations.</p>

timeseries Property Descriptions (Continued)

Property	Description
DataInfo	<p>Contains fields for storing contextual information about Data:</p> <ul style="list-style-type: none">• Unit — String that specifies data units.• Interpolation — A <code>tsdata.interpolation</code> object that specifies the interpolation method for this time series. Fields in the <code>tsdata.interpolation</code> object include:<ul style="list-style-type: none">▪ 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.

timeseries Property Descriptions (Continued)

Property	Description
Events	<p data-bbox="675 373 1313 465">An array of <code>tsdata.event</code> objects that stores event information for this <code>timeseries</code> object. You add events using the <code>addevent</code> method.</p> <p data-bbox="675 482 1246 546">Fields in the <code>tsdata.event</code> object include the following:</p> <ul data-bbox="675 581 1320 986" style="list-style-type: none"><li data-bbox="675 581 1320 638">• <code>EventData</code> — Any user-defined information about the event<li data-bbox="675 656 1320 685">• <code>Name</code> — String that specifies the name of the event<li data-bbox="675 703 1320 795">• <code>Time</code> — Time value when this event occurs, specified as a real number or a date string relative to <code>StartDate</code><li data-bbox="675 812 957 841">• <code>Units</code> — Time units<li data-bbox="675 859 1320 986">• <code>StartDate</code> — A reference date specified in MATLAB date-string format. <code>StartDate</code> is empty when you have a numerical (nondate-string) time vector.

timeseries Property Descriptions (Continued)

Property	Description
IsTimeFirst	<p>Logical value (true or false) that specifies whether the first or last dimension of the Data array aligns with the time vector.</p> <p>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.</p> <p>When you set this property to</p> <ul style="list-style-type: none"> • 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 <p>After a time series is created, this property is read-only.</p>
Name	<p>timeseries object name entered as a string. This name can differ from the name of the timeseries variable in the MATLAB workspace.</p>
Quality	<p>An integer vector or array containing values - 128 to 127 that specifies the quality in terms of codes defined by the QualityInfo.Code field.</p> <p>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.</p> <p>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.</p>

timeseries Property Descriptions (Continued)

Property	Description
QualityInfo	<p>Provides a lookup table that converts numerical Quality codes to readable descriptions. QualityInfo fields include the following:</p> <ul style="list-style-type: none"> • 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 <p>The length of Code and Description must match.</p>
Time	<p>Vector of time values.</p> <p>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.</p> <p>The length of Time must match either the first or the last dimension of Data.</p>

timeseries Property Descriptions (Continued)

Property	Description
TimeInfo	<p>Uses the following fields to store contextual information about Time:</p> <ul style="list-style-type: none"> • 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 <code>datestr</code> function reference page for more information. • StartDate — Date string defining the reference date. See the MATLAB <code>setabstime</code> (timeseries) function reference page for more information. • UserData — Stores any additional user-defined information
TreatNaNasMissing	<p>Logical value that specifies how to treat NaN values in Data:</p> <ul style="list-style-type: none"> • 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.

timeseries Methods

The following method categories are available for working with `timeseries` objects:

- “General Methods” on page 4-28
- “Data and Time Manipulation Methods” on page 4-36
- “Event Methods” on page 4-30
- “Arithmetic Operation Methods” on page 4-31
- “Statistical Methods” on page 4-32

General Methods

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

Methods for Querying Properties

Method	Description
<code>get (timeseries)</code>	Query <code>timeseries</code> object property values.
<code>getdatasamplesize</code>	Return the size of each data sample in a <code>timeseries</code> object.
<code>getqualitydesc</code>	Return data quality descriptions based on the <code>Quality</code> property values assigned to a <code>timeseries</code> object.
<code>isempty (timeseries)</code>	Evaluate to <code>true</code> for an empty <code>timeseries</code> object.
<code>length (timeseries)</code>	Return the length of the time vector.
<code>plot (timeseries)</code>	Plot the <code>timeseries</code> object.
<code>set (timeseries)</code>	Set <code>timeseries</code> property values.

Methods for Querying Properties (Continued)

Method	Description
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 NaNs 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.
getsamplesingtime (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.

Methods for Manipulating Data and Time (Continued)

Method	Description
<code>resample (timeseries)</code>	Select or interpolate data in a <code>timeseries</code> object using a new time vector.
<code>setabstime (timeseries)</code>	Set the time values in the time vector as date strings.
<code>setinterpmethod</code>	Set interpolation method for a <code>timeseries</code> object.
<code>synchronize</code>	Synchronize and resample two <code>timeseries</code> objects using a common time vector.
<code>transpose (timeseries)</code>	Transpose a <code>timeseries</code> object.
<code>vertcat (timeseries)</code>	Vertical concatenation for <code>timeseries</code> 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 4-9.

Methods That Define and Use Events

Method	Description
<code>addevent</code>	Add one or more events to a <code>timeseries</code> object.
<code>delevent</code>	Delete one or more events from a <code>timeseries</code> object.
<code>gettsafteratevent</code>	Create a new <code>timeseries</code> object by extracting the samples from an existing time series that occur after or at a specified event.
<code>gettsafterevent</code>	Create a new <code>timeseries</code> object by extracting the samples that occur after a specified event from an existing time series.

Methods That Define and Use Events (Continued)

Method	Description
gettsatevent	Create a new <code>timeseries</code> object by extracting the samples that occur at the same time as a specified event from an existing time series.
gettsbeforeatevent	Create a new <code>timeseries</code> object by extracting the samples that occur before or at a specified event from an existing time series.
gettsbeforeevent	Create a new <code>timeseries</code> object by extracting the samples that occur before a specified event from an existing time series.
gettsbetweenevents	Create a new <code>timeseries</code> 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 `timeseries`

Operation	Description
+	Add the corresponding data values of <code>timeseries</code> objects.
-	Subtract the corresponding data values of <code>timeseries</code> objects.
.*	Element-by-element multiplication of <code>timeseries</code> data.
*	Matrix-multiply <code>timeseries</code> data.
./	Right element-by-element division of <code>timeseries</code> data.
/	Right matrix division of <code>timeseries</code> data.

Methods to Arithmetically Combine timeseries (Continued)

Operation	Description
<code>.\</code>	Element-by-element left-array divide of timeseries data.
<code>\</code>	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
<code>iqr (timeseries)</code>	Return the interquartile range of timeseries data.
<code>max (timeseries)</code>	Return the maximum value of timeseries data.
<code>mean (timeseries)</code>	Return the mean of timeseries data.
<code>median (timeseries)</code>	Return the median of timeseries data.
<code>min (timeseries)</code>	Return the minimum of timeseries data.
<code>std (timeseries)</code>	Return the standard deviation of timeseries data.
<code>sum (timeseries)</code>	Return the sum of timeseries data.
<code>var (timeseries)</code>	Return the variance of timeseries data.

tscollection Constructor

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.

MATLAB provides several methods for convenient analysis and manipulation of `timeseries` in a `tscollection` object.

tscollection 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 `tscollection` Objects” on page 4-10.

tscollection Syntax Descriptions

Syntax	Description
<code>tsc = tscollection(ts)</code>	<p>Creates a <code>tscollection</code> object <code>tsc</code> that includes one or more <code>timeseries</code> objects.</p> <p>The <code>ts</code> argument can be one of the following:</p> <ul style="list-style-type: none"> • Single <code>timeseries</code> object in the MATLAB workspace • Cell array of <code>timeseries</code> objects in the MATLAB workspace <p>The <code>timeseries</code> objects share the same time vector in the <code>tscollection</code>.</p>

tscollection Syntax Descriptions (Continued)

Syntax	Description
<code>tsc = tscollection(Time)</code>	Creates an empty <code>tscollection</code> object with the time vector <code>Time</code> . When time values are date strings, you must specify <code>Time</code> as a cell array of date strings.
<code>tsc = tscollection(Time, TimeSeries, 'Parameter', Value, ...)</code>	Optionally enter the following parameter-value pairs after the <code>Time</code> and <code>TimeSeries</code> arguments: <ul style="list-style-type: none"> • Name (see “<code>tscollection</code> Properties” on page 4-34) • <code>IsDatenum</code> When set to true, <code>IsDatenum</code> specifies that <code>Time</code> values are dates in the format of MATLAB serial dates.

tscollection 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.

tscollection Property Descriptions

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

tscollection Property Descriptions (Continued)

Property	Description
Time	<p>A vector of time values.</p> <p>When <code>TimeInfo.StartDate</code> is empty, the numerical Time values are measured relative to 0 in specified units. When <code>TimeInfo.StartDate</code> is defined, the time values represent date strings measured relative to <code>StartDate</code> in specified units.</p> <p>The length of <code>Time</code> must match either the first or the last dimension of the <code>Data</code> property of each <code>tscollection</code> member.</p>
TimeInfo	<p>Uses the following fields to store contextual information about <code>Time</code>:</p> <ul style="list-style-type: none"> • 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 <code>datestr</code> function reference page for more information. • StartDate — Date string defining the reference date. See the MATLAB <code>setabstime (timeseries)</code> function reference page for more information. • UserData — Stores any additional user-defined information

tscollection Methods

The following method categories are available for working with `tscollection` objects:

- “General `tscollection` Methods” on page 4-36
- “Data and Time Manipulation Methods” on page 4-36

General `tscollection` Methods

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

Methods for Querying Properties

Method	Description
<code>get (tscollection)</code>	Query <code>tscollection</code> object property values.
<code>isempty (tscollection)</code>	Evaluate to true for an empty <code>tscollection</code> object.
<code>length (tscollection)</code>	Return the length of the time vector.
<code>plot (timeseries)</code>	Plot the time series in a collection.
<code>set (tscollection)</code>	Set <code>tscollection</code> property values.
<code>size (tscollection)</code>	Return the size of a <code>tscollection</code> object.
<code>tstool</code>	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.
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.
getsamplingsingtime (tscollection)	Extract data samples from an existing tscollectionobject 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.

Using Time Series Tools

The following sections describe how to use the MATLAB Time Series Tools graphical user interface (GUI) for analyzing time-series data.

Introduction (p. 5-2)	Summarizes the Time Series Tools window and workflow
Importing and Exporting Data (p. 5-8)	Describes supported data sources and instructions for importing and exporting data in Time Series Tools
Plotting Time Series (p. 5-14)	Describes how to work with time plots, histograms, spectral plots, correlation plots, and XY plots
Selecting Data for Analysis (p. 5-29)	Provides instructions for selecting data on which to focus your analysis
Editing Data, Time, Attributes, and Events (p. 5-33)	Describes how to edit data, time, units, interpolation method, quality codes, and events for time series
Processing and Manipulating Time Series (p. 5-43)	Provides instructions for processing time series, including filtering, interpolating, resampling, and algebraically manipulating data
Example — Using MATLAB Time Series Tools (p. 5-44)	Provides an example of importing, plotting, and analyzing time series

Introduction

The Time Series Tools graphical user interface (GUI) extends the MATLAB environment for analyzing time-series data. For an example of using Time Series Tools, see “Example — Using MATLAB Time Series Tools” on page 5-44.

For more information about implementing MATLAB objects and methods, see Chapter 4, “Using Time-Series Objects and Methods”.

This section contains the following topics:

- “Opening Time Series Tools” on page 5-2
- “Getting Help” on page 5-3
- “Time Series Tools Window” on page 5-3
- “Time Series Tools Workflow” on page 5-5
- “Generating Reusable M-Code” on page 5-6

Opening Time Series Tools

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

```
tstool
```

For a description of the Time Series Tools GUI, see “Time Series Tools Window” on page 5-3.

To learn how to import data into Time Series Tools, see “Importing and Exporting Data” on page 5-8.

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 signals	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.

- Options and Settings pane

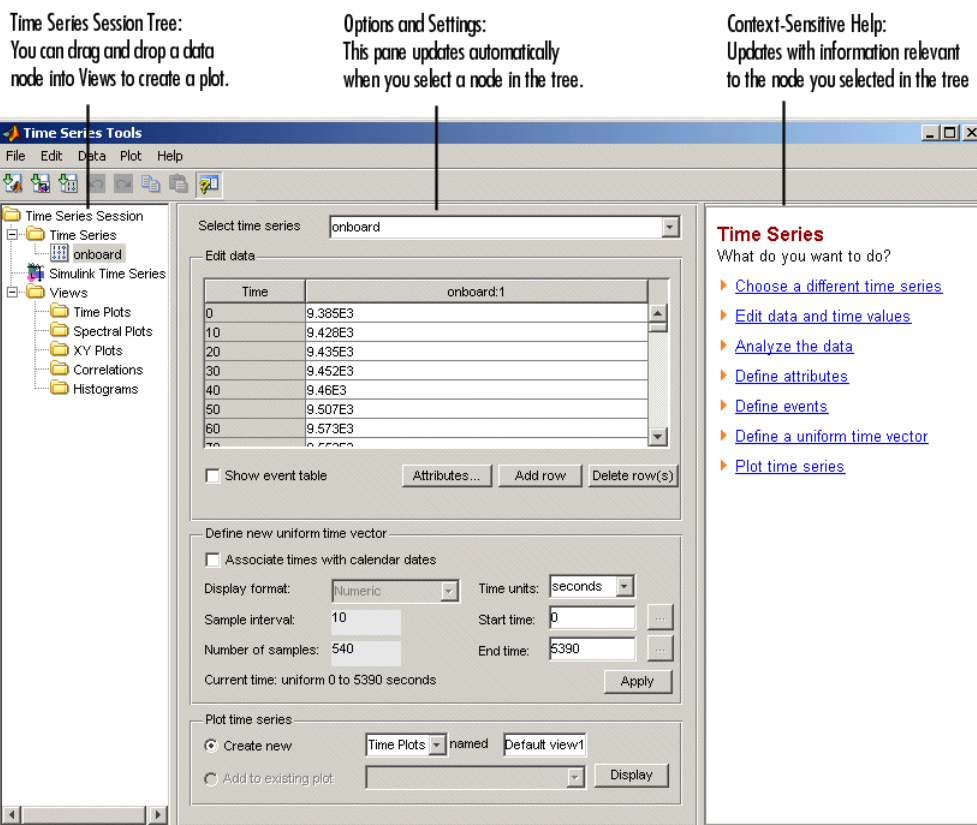
After you select a node in the tree, this pane displays options and settings pertaining to the node you selected in the tree.

- 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 5-3.

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 Excel workbook, MAT-file, or MATLAB workspace.

For more information, see “Importing and Exporting Data” on page 5-8.

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

For more information, see “Creating a Plot” on page 5-15.

3 Select data subset for analysis.

For more information, see “Selecting Data for Analysis” on page 5-29.

4 Edit the data by

- Identifying and removing outliers or “dead time” (see “Selecting Data Using Rules” on page 5-29).
- Manually correcting errors (see “Editing Data, Time, Attributes, and Events” on page 5-33).

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 5-43.

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

For more information, see “Plotting Time Series” on page 5-14.

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 5-13.

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 Chapter 4, “Using Time-Series Objects and Methods”.

For an example of automatically generating and viewing M-code, see “Example — Using MATLAB Time Series Tools” on page 5-44.

Importing and Exporting Data

After starting Time Series Tools, as described in “Opening Time Series Tools” on page 5-2, you can import data from a file or from the MATLAB workspace.

This section contains the following topics:

- “Types of Data You Can Import” on page 5-8
- “How to Import Data” on page 5-8
- “Changes to Data Representation During Import” on page 5-10
- “Importing Multivariate Data” on page 5-11
- “Importing Data with Missing Values” on page 5-12
- “Exporting Data from Time Series Tools” on page 5-13

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 Chapter 4, “Using Time-Series Objects and Methods”.

- Simulink 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:

- “Import Commands in Time Series Tools” on page 5-9
- “Import Wizard in Time Series Tools” on page 5-9

Import Commands in Time Series Tools

You use the following commands to import data into Time Series Tools. Each command opens a dialog box where 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.
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 .

Import Wizard in Time Series Tools

When you import data from a file or from the MATLAB workspace in Time Series Tools, you open the Import Wizard. The Import Wizard facilitates selecting what data to import when you want to analyze 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 Data into Time Series Tools” on page 5-45.

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 NaNs.
- 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 5-13.

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 5-16.

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 Chapter 4, “Using Time-Series Objects and Methods”, 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 `tscollection` Objects” on page 4-10.
- 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 NaNs. NaNs 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 in MATLAB.

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

This section contains the following topics:

- “Types of Plots in Time Series Tools” on page 5-14
- “Creating a Plot” on page 5-15
- “Customizing Line and Marker Styles” on page 5-16
- “Editing Plot Appearance” on page 5-16
- “Time Plots” on page 5-18
- “Spectral Plots” on page 5-19
- “Histograms” on page 5-21
- “Correlation Plots” on page 5-22
- “XY Plots” on page 5-27

Types of Plots in Time Series Tools

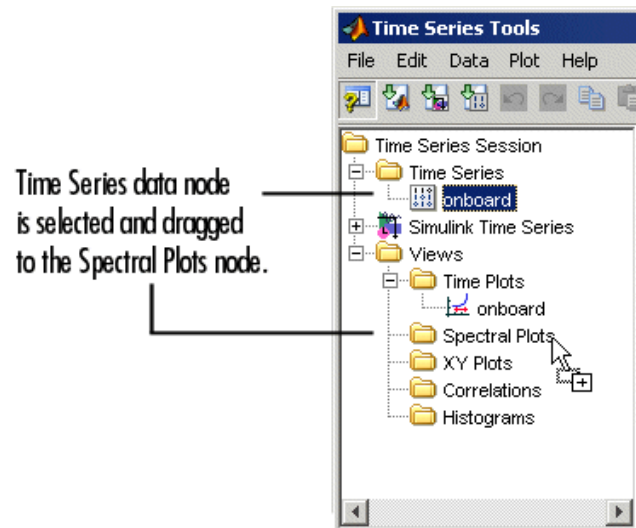
You can generate the following types of plots in Time Series Tools.

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 <i>bins</i> .
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 crosscorrelation 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 Time Series Tools by dragging and dropping a **Time Series** data node in the **Time Series Session** tree into a **Views** node.

The following figure shows an example of how to create a spectral plot by dragging the onboard time series into 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.

When you drag several time series into the same plot node, all time series are plotted in the same window but on different axes. When a time series contains several columns of data, all data columns are plotted on the same axes. For information about customizing the appearance of plot lines and markers, see “Customizing Line and Marker Styles” on page 5-16.

XY plots and crosscorrelation plots compare two time series. To create these plots, you must first drag one time series into a plot node, and then a second time series into 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 Time Plot” on page 5-47.

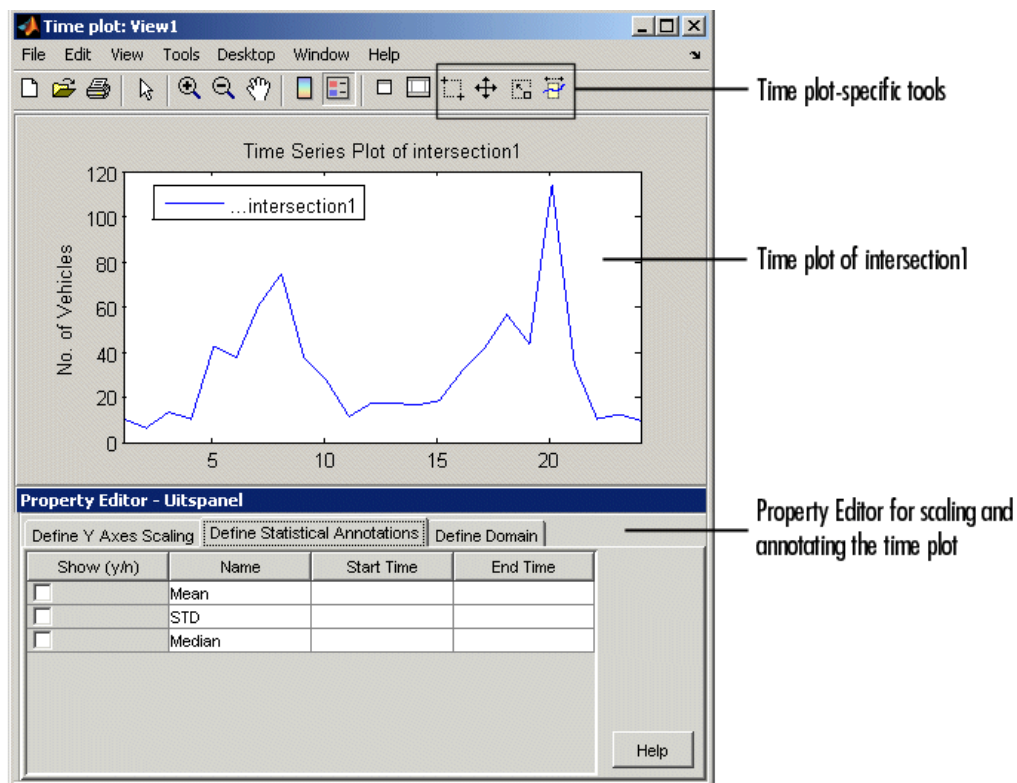
Editing Plot Appearance

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.

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 5-15.





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 (STD), 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 5-39.
	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 5-15.

The periodogram is the unbiased estimate of the power spectral density of a time series, calculated as the scaled absolute value of the $(\text{FFT})^2$ 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 5-15.

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 NaNs. For more information, see “Removing or Replacing Data with NaNs” on page 5-22. Then, you can interpolate all values tagged as NaNs using the selected interpolation method. For more information about specifying an interpolation method, see “Defining Data Attributes” on page 5-36.


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” on page 5-21, you can delete it or replace it with NaNs. If you want to interpolate specific data values, you must replace the selected data with NaNs first.

To delete data, right-click the selected region and select **Remove Selection** from the shortcut menu.

To replace data with NaNs, right-click the selected region and select **Replace with NaNs** from the shortcut menu.

Correlation Plots

You can create autocorrelation plots (correlograms) and crosscorrelation 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 (crosscorrelation).

This section includes the following topics:

- “Autocorrelation of a Time Series” on page 5-23
- “Crosscorrelation of Time Series” on page 5-24
- “Interpreting Correlation Plots” on page 5-26

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 5-43.

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 2-6.

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 crosscorrelations of the various data columns. For more information, see “Crosscorrelation of Time Series” on page 5-24.

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 5-43.

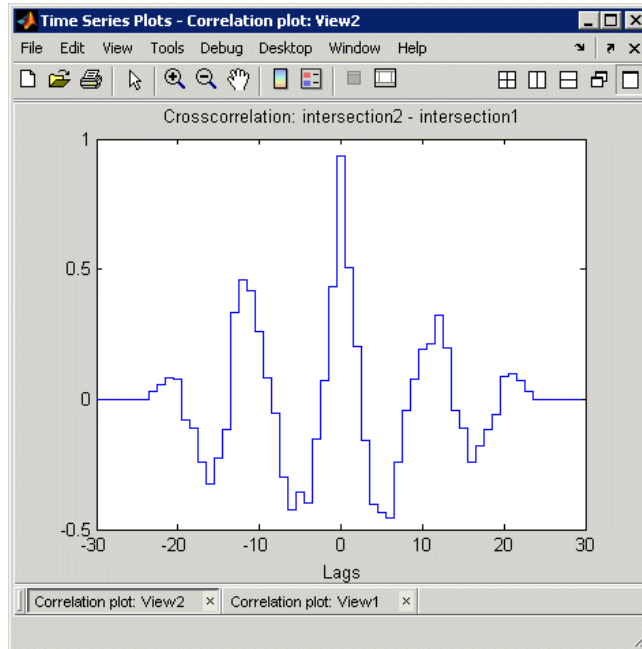
Crosscorrelation of Time Series

Crosscorrelation 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 crosscorrelation plot, make sure that both time series have the same uniform time vector.

To create a crosscorrelation 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.

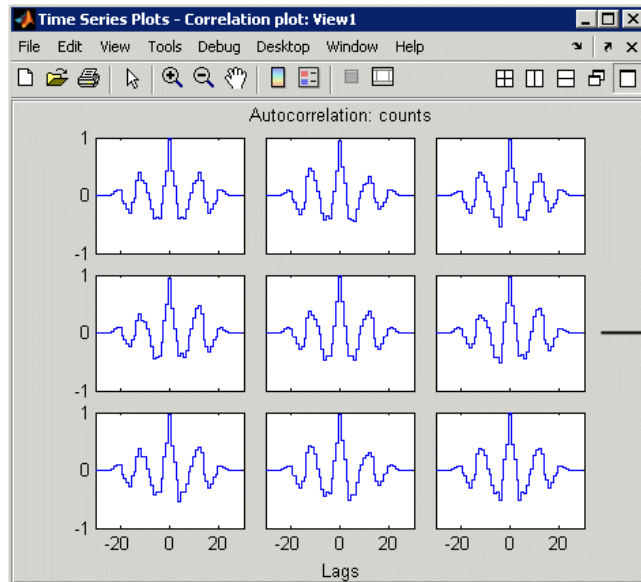
A crosscorrelation plot of two time series, where each contains a single column of data, shows the degree of linear relationship between the data values in the two time series at various lags. For example, the following figure shows a crosscorrelation plot of two time series, `intersection1` and `intersection2`. There is a high correlation when there is no lag in the data, as well as for lags of about -11 and 11.



Crosscorrelation of Two Time Series

A crosscorrelation 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 crosscorrelation 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.



Subplot indices correspond to the data columns correlated in time series with 3 data columns.

[1,1]	[1,2]	[1,3]
[2,1]	[2,2]	[2,3]
[3,1]	[3,2]	[3,3]

Crosscorrelation 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.

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.

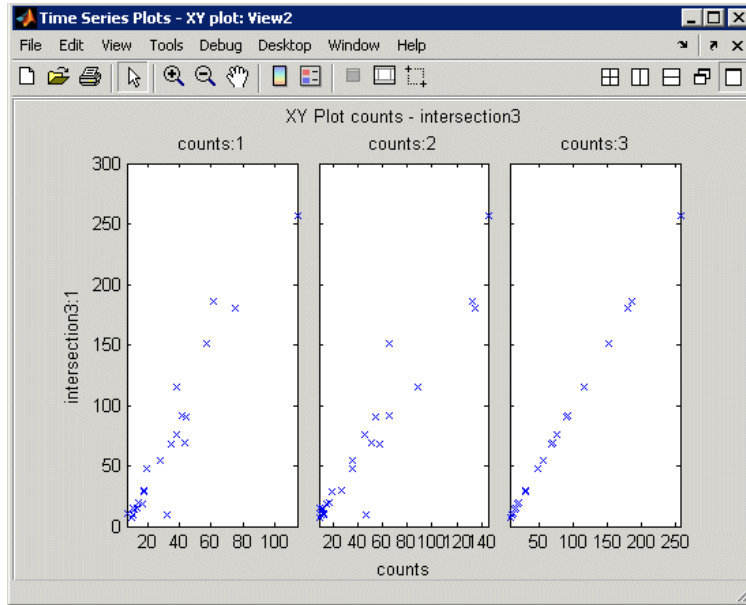
Note To generate an XY plot, both time series must have the same time vectors.

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 5-55.

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

Before beginning data analysis, you can select a subset of the data on which to focus your analysis. First create a time plot to see which portions are of interest and, in the time plot, select the data. For more information on creating a time plot, see “Creating a Plot” on page 5-15.

This section contains the following topics:

- “Selecting Data Using Rules” on page 5-29 — Describes how to access a dialog box where you create logical expressions to identify outliers and constant values.
- “Selecting Data Graphically” on page 5-30 — Describes how to use plot tools to select values on a plot.
- “Excluding Data from Analysis” on page 5-31 — Describes how to exclude regions of data from analysis.

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 5-15.

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 5-31.

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 5-15.

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 5-30.

- **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 5-31.

Tip To learn how you can select specific data values in a histogram plot, see “Selecting Data” on page 5-21.


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 5-31).

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 5-31).

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

After importing data into Time Series Tools, you can edit specific data and time values. In addition, you can specify descriptive information for your time series (*metadata*), including the interpolation method, data quality, units, and events.

This section contains the following topics:

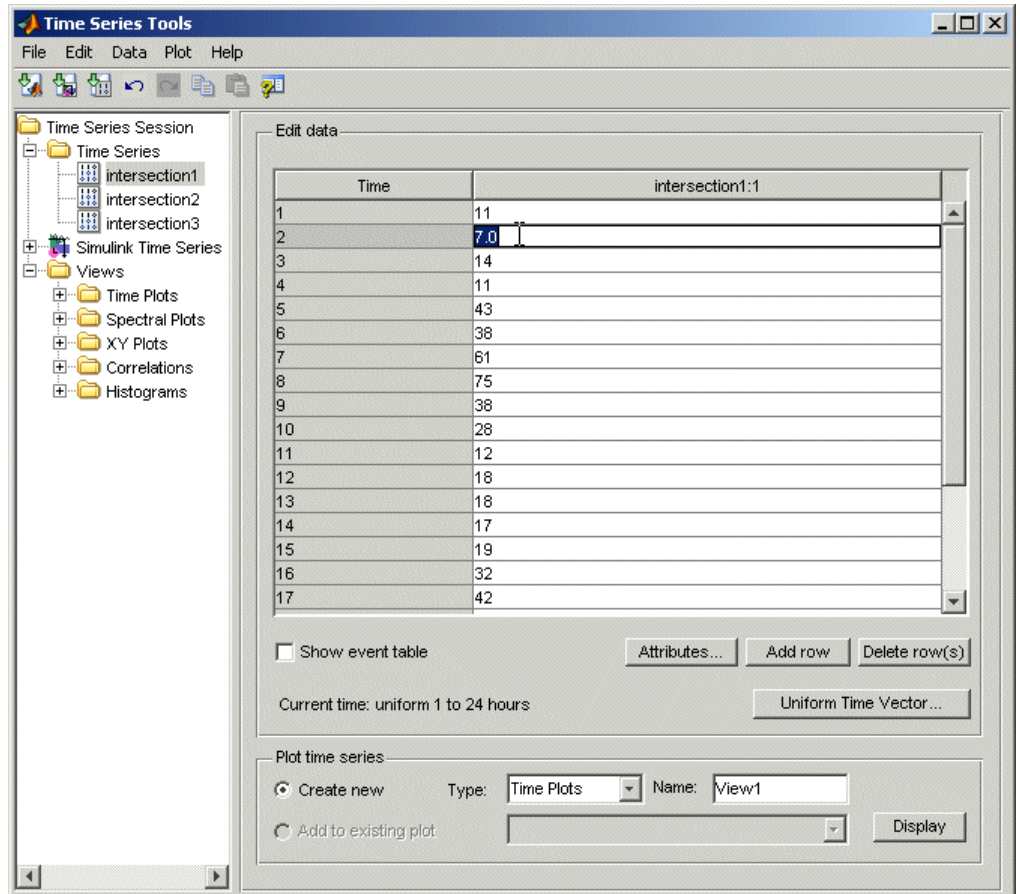
- “Displaying the Data Table” on page 5-33
- “Editing Data and Time” on page 5-34
- “Defining Data Attributes” on page 5-36
- “Assigning Quality Codes to Data” on page 5-38
- “Defining Events” on page 5-39


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 5-14.



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 5-33, 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 4-18.

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 5-33.

Units and Interpolation Method

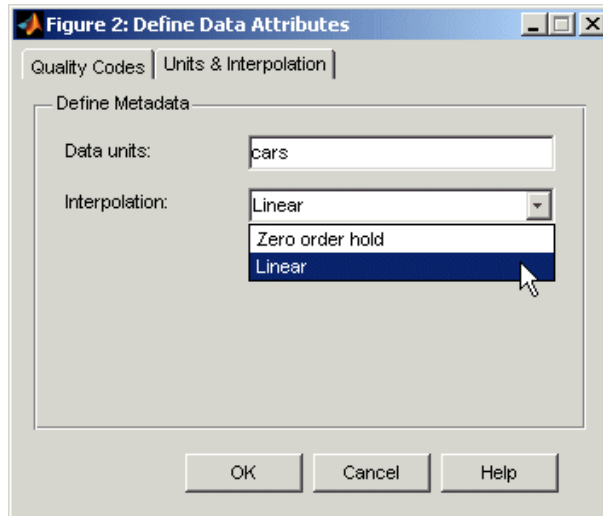
Data units are stored as metadata for the currently selected time series. If this time series contains multiple data columns, all data is assigned the same units.

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 method “holds” the last value constant until the next available measurement.



Quality Codes

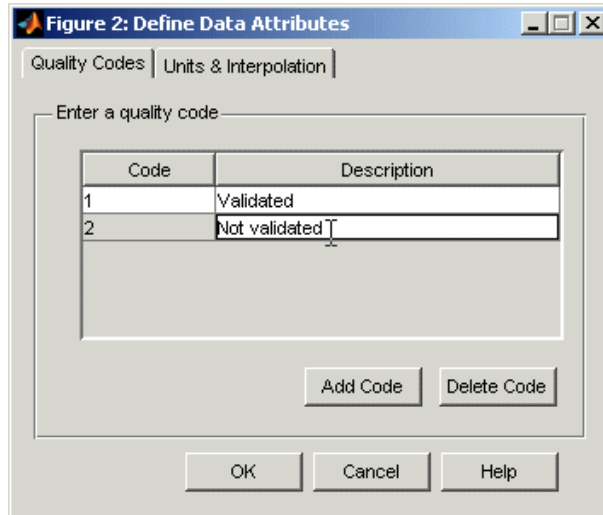
You can define quality codes to annotate the quality of each value in the data table. Each quality attribute consists of a numerical code and a brief description. For information about assigning quality codes to specific data values, see “Assigning Quality Codes to Data” on page 5-38.

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.

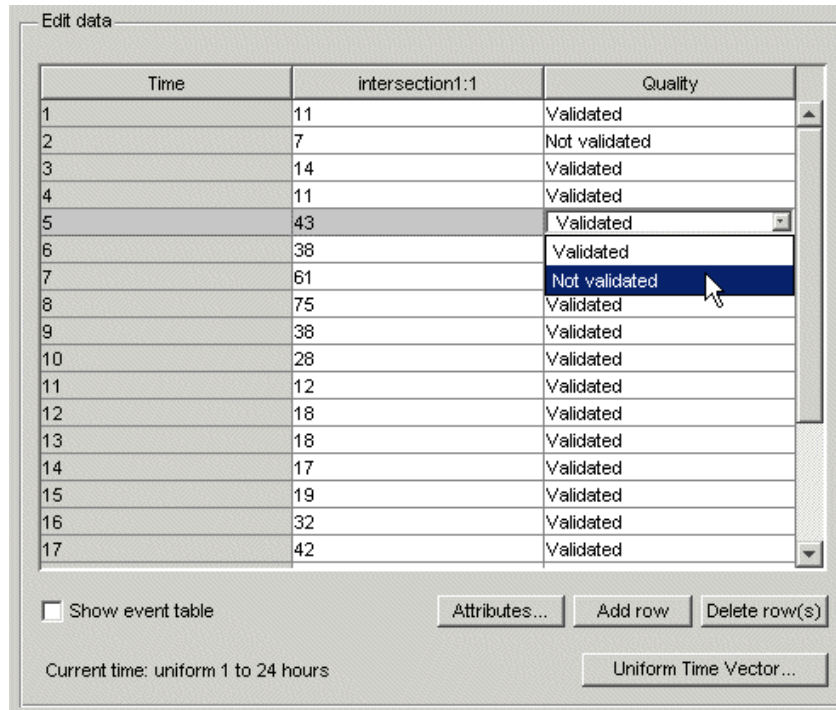


Note To delete a quality attribute, select it and click **Delete Code**.

Assigning Quality Codes to Data

After you define quality codes, as described in “Quality Codes” on page 5-37, 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 5-33.

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.



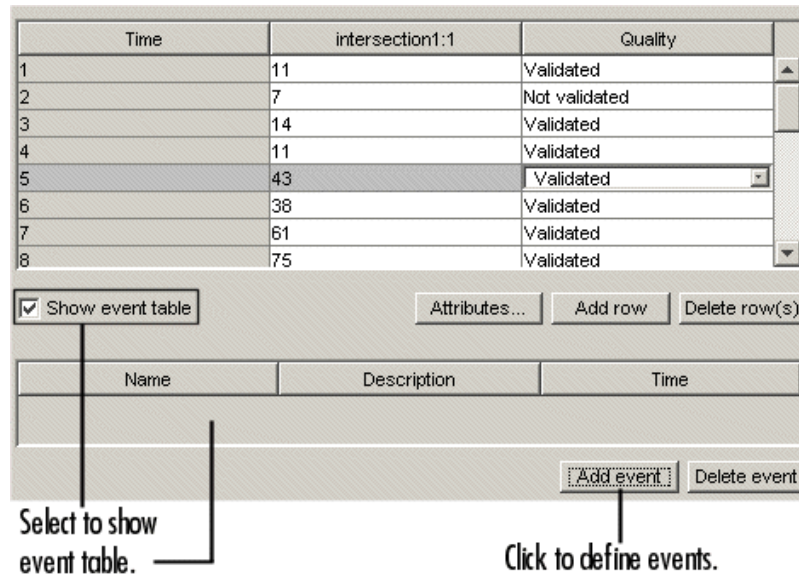
Defining Events

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 5-33.

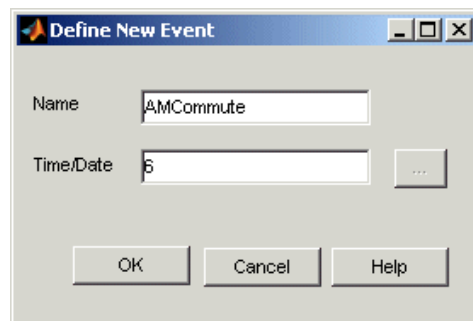
You can also use events as reference points when shifting time series in time. For more information about synchronizing time series, see “Processing and Manipulating Time Series” on page 5-43.

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 4-18.

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.

Edit data

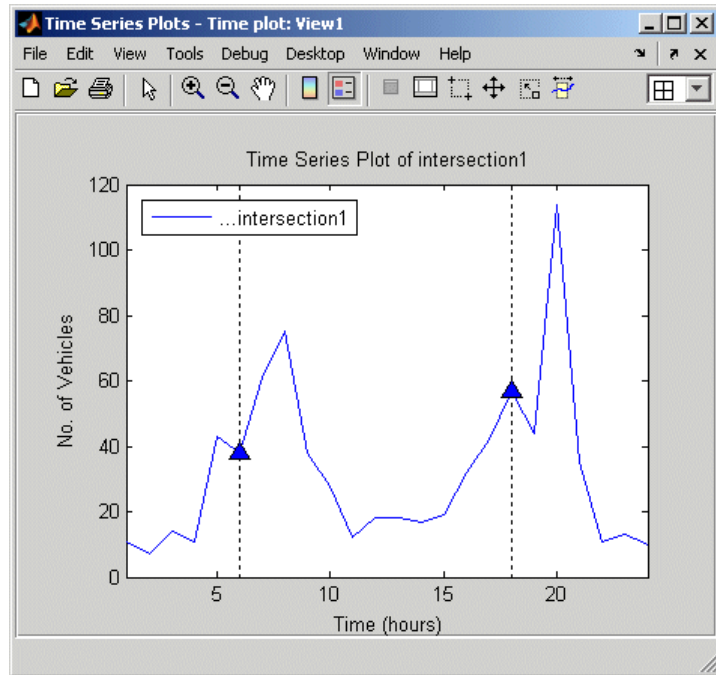
Time	intersection1:1
1	11
2	7
3	14
4	11
5	43
6	38
6.0	AMCommute
7	61

Show event table

Name	Description	Time
AMCommute		6.000
PMCommute		18.000

Current time: uniform 1 to 24 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 5-14.

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 – Using MATLAB Time Series Tools

This example illustrates how to perform the following tasks using Time Series Tools:

- “Loading Data into the MATLAB Workspace” on page 5-44
- “Starting Time Series Tools” on page 5-44
- “Enabling M-Code Generation” on page 5-44
- “Importing Data into Time Series Tools” on page 5-45
- “Creating a Time Plot” on page 5-47
- “Resampling Time Series” on page 5-53
- “Comparing Data on an XY Plot” on page 5-55
- “Viewing Generated M Code” on page 5-57
- “Exporting Time Series to the Workspace” on page 5-59

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

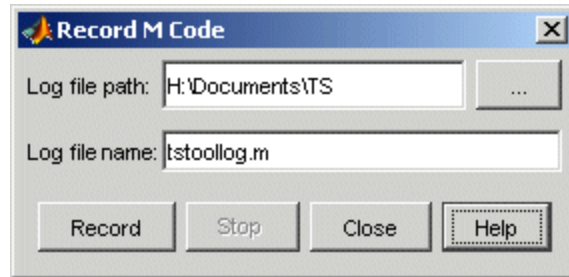
```
tstool
```

This opens the Time Series Tools window. For more information about this GUI, see “Time Series Tools Window” on page 5-3.

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 5-57.

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.

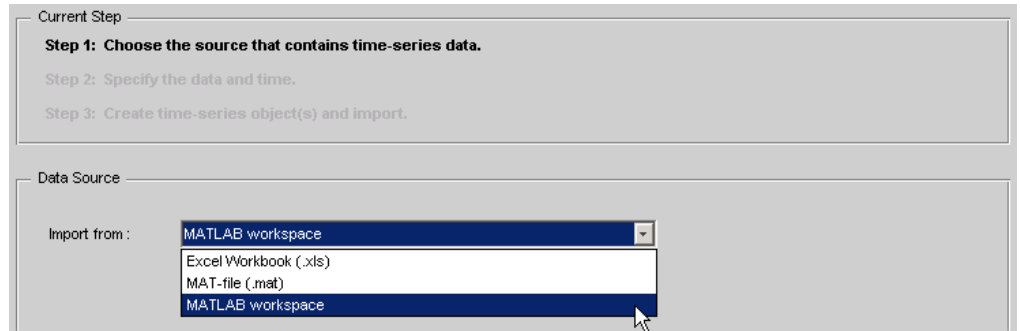
Importing Data into Time Series Tools

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.

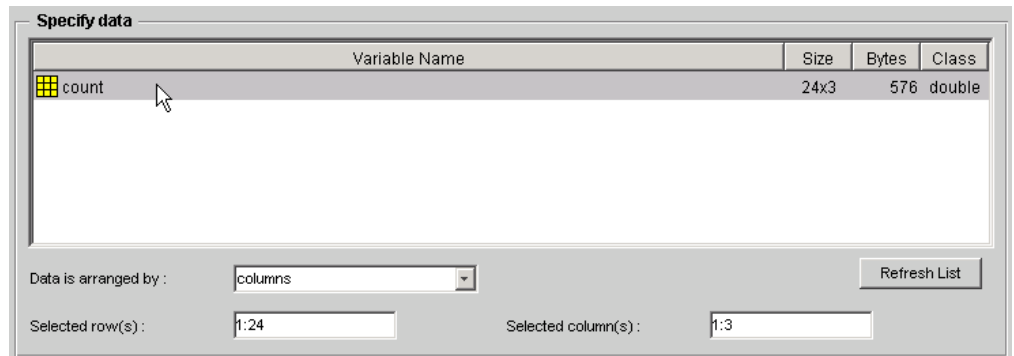
Note To get help on a specific field in the Import Wizard, right-click the field label and select **What’s This** from the shortcut menu.

- 1 In the Time Series Tools window, select **File > Import from Workspace > Array Data**. This opens the Import Wizard.

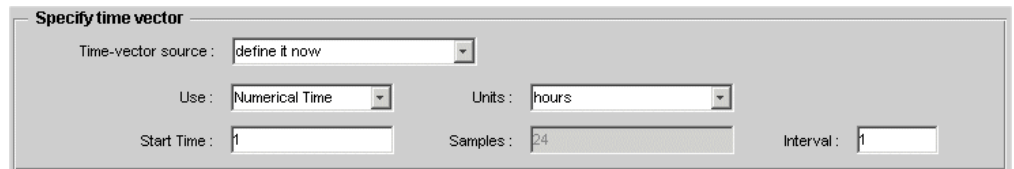
2 In the **Import from** list, select **MATLAB workspace** and click **Next**.



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.



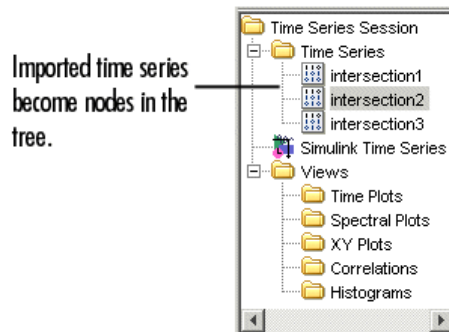
4 In the **Specify Time Vector** area, select hours from the **Units** list. In the **Start Time** field, type 1 to start the time vector at 1 hour. The Import Wizard has already filled in the remaining options to define a uniformly spaced time vector with a length of 24 and an interval of 1.



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.

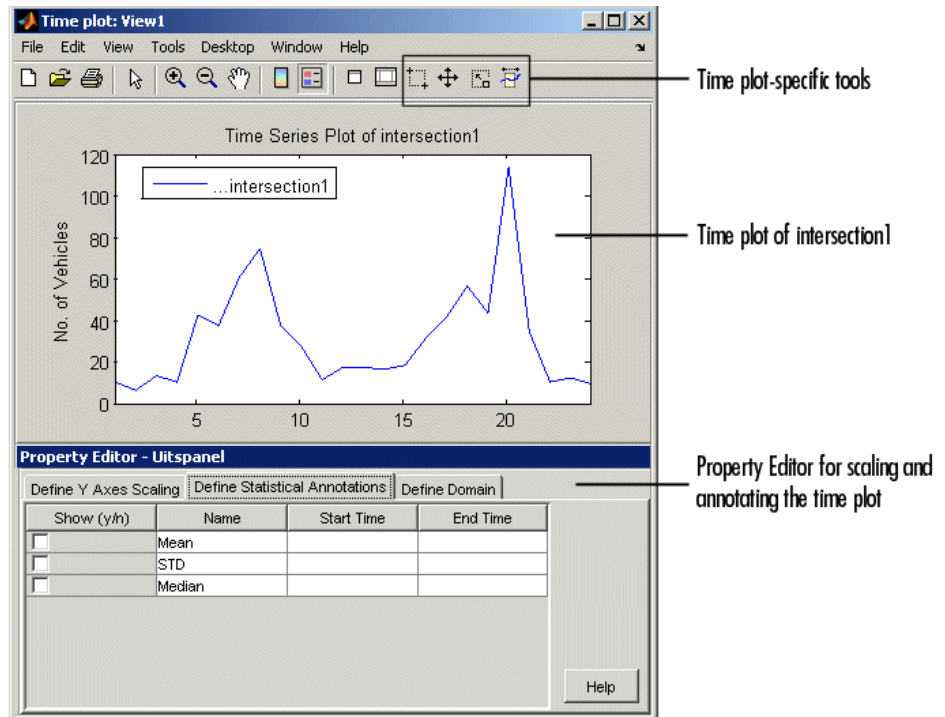
7 Click **Finish**. This adds three time series to the Time Series Session tree: intersection1, intersection2, and intersection3 (as shown below).



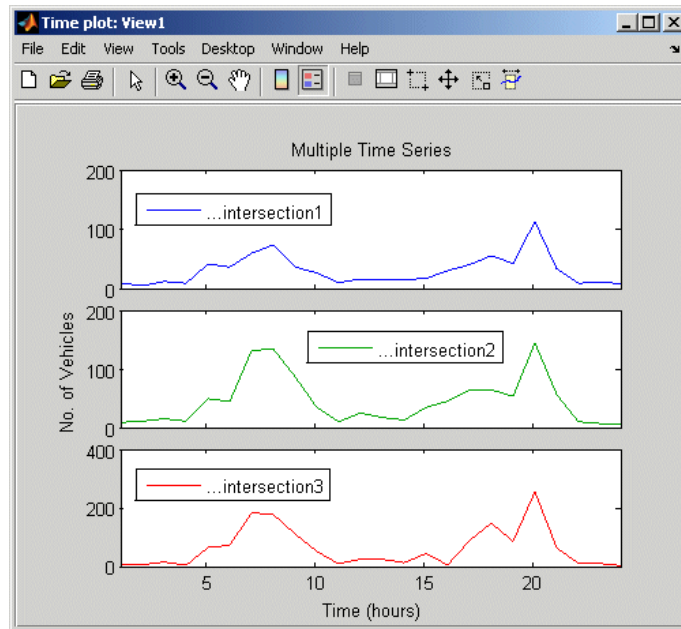
Creating a Time Plot

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.



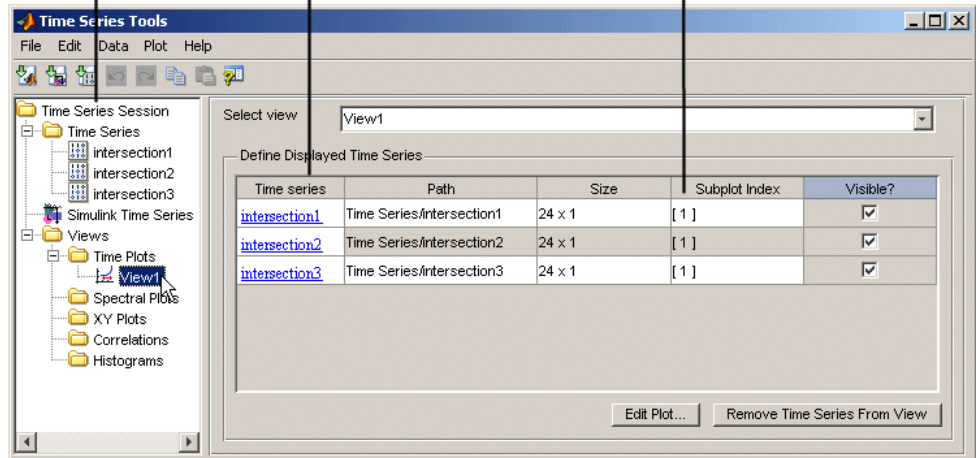
- 3** To change the appearance of the time series in the plot, select **Plot > Set Line Properties**. This opens the Line Styles dialog box.

- 4 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**.

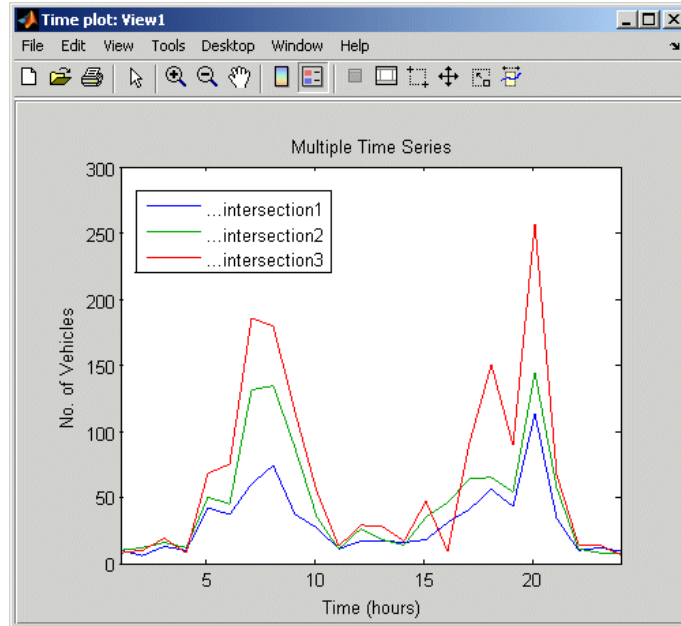
Select the plot in the tree to edit its display.

Click the hyperlink to display and edit the data.

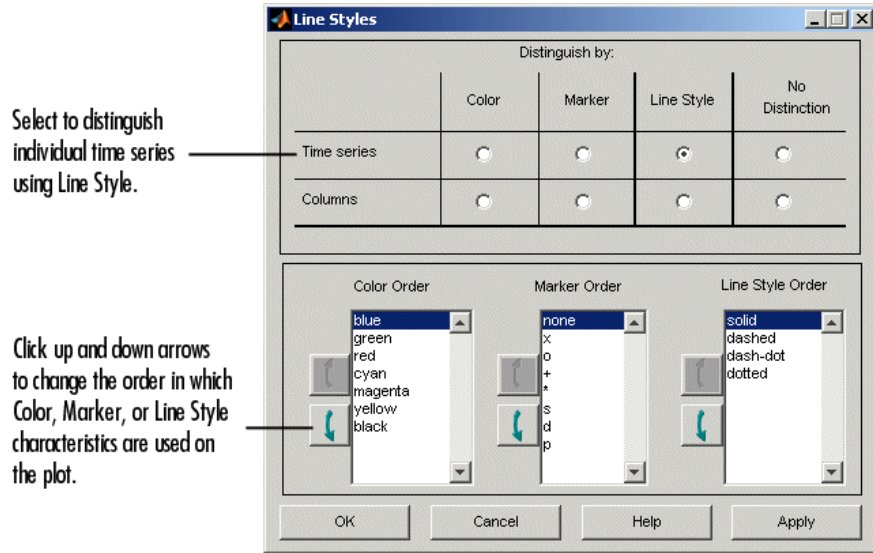
Change Subplot Index to [1] to display all time series on the same axes.



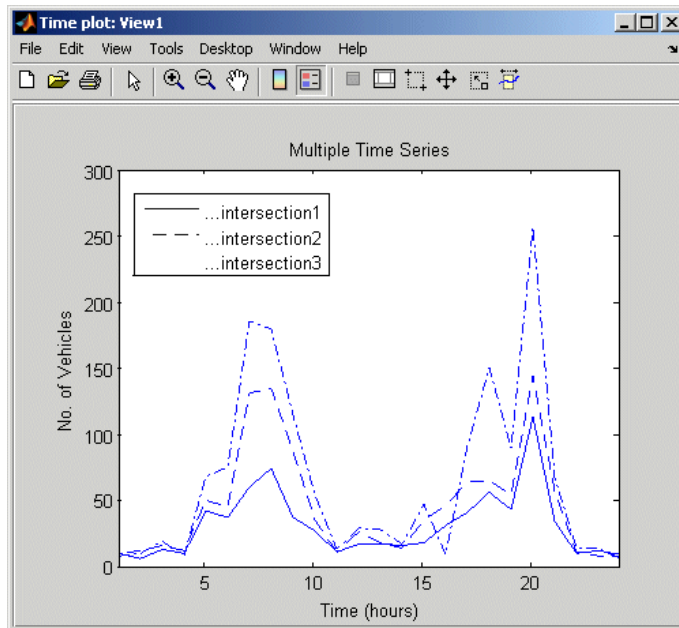
This displays all time series on the same axes, as follows:



- 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 5-36.

This portion of the example shows

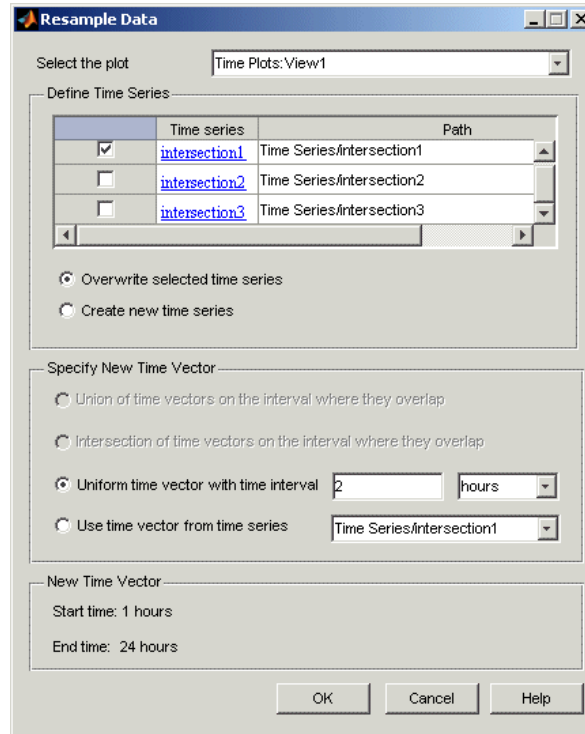
- “Resampling on a Uniform Time Vector” on page 5-53
- “Resampling by Finding a Common Time Vector” on page 5-55

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 Time Plot” on page 5-47 and select **Data > Resample** 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 Time Plot” on page 5-47 and select **Data > Resample** 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.

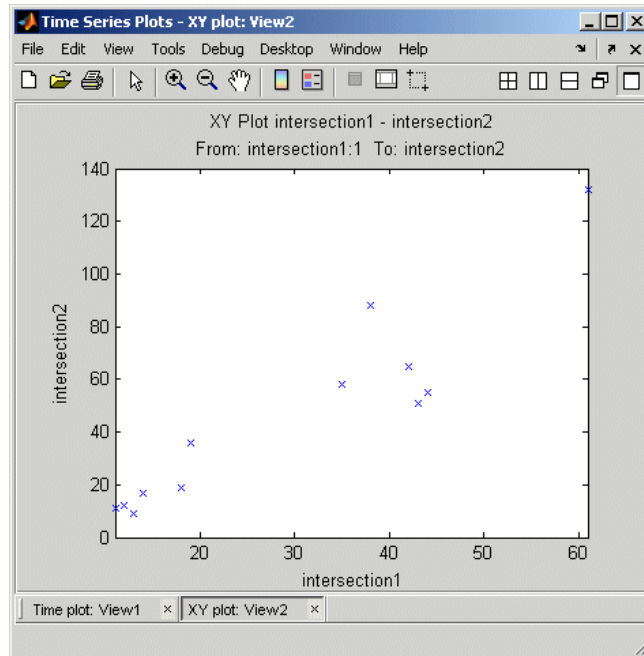
Comparing Data on an XY Plot

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.

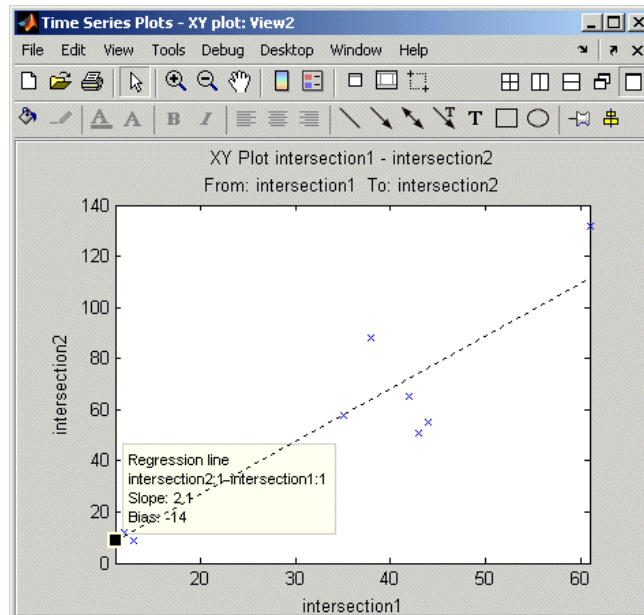
In this portion of the example, you examine the relationship between the corresponding data values of `intersection1` and `intersection2` by using an XY plot.

- 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.

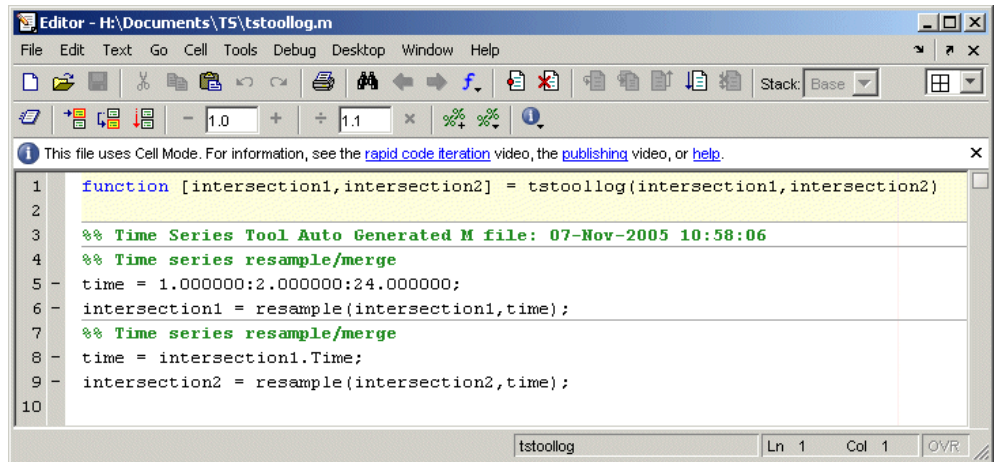


Viewing Generated M Code

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.



```
1 function [intersection1,intersection2] = tstoolog(intersection1,intersection2)
2
3 %% Time Series Tool Auto Generated M file: 07-Nov-2005 10:58:06
4 %% Time series resample/merge
5 time = 1.000000:2.000000:24.000000;
6 intersection1 = resample(intersection1,time);
7 %% Time series resample/merge
8 time = intersection1.Time;
9 intersection2 = resample(intersection2,time);
10
```

Automatically Generated M Code

You can reuse this M code by calling the `tstoolog` 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 5-44.

Examine the code of the `tstoolog` 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 generated M-file contains only syntax for manipulating the data in Time Series Tools. It does not contain the commands for generating and editing the 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 5-53.

- 1 Click the **intersection1** 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.

A

attributes of time series 5-33
autocorrelation of time series 5-23

B

Basic Fitting dialog box 2-8
usage example 2-10

C

confidence bounds 2-32
correlation analysis 2-4
correlation coefficients 2-6
correlation plots 5-22
interpreting 5-26
covariance 2-4
crosscorrelation of time series 5-24
curve fitting, *see* data fitting
Curve Fitting Toolbox 2-3
customizing time-series plots 5-16

D

data analysis
MATLAB GUIs for 1-3
of matrix data 1-3
plotting data 1-7
preparing data for 1-1
related toolboxes 1-4
data filtering, *see* filtering
data fitting 2-1
confidence bounds 2-32
example using functions 2-28
functions 2-20
multiple regression 2-26
nonpolynomial 2-24
polynomial 2-20
residuals 2-2
data statistics, *see* statistics

Data Statistics dialog box 1-28
saving statistics 1-34
usage example 1-28
descriptive statistics 1-25
detrending data 1-20
in Time Series Tools 5-43
difference equations 1-14
discrete filter 1-16
discrete Fourier transform, *see* Fourier transforms

E

editing time series 5-33
events in time series 5-33
exporting data
from MATLAB 1-6
from Time Series Tools 5-8

F

fast Fourier transform, *see* Fourier transforms
filter function 1-14
filtering
detrending data 1-20
difference equations 1-14
discrete filter 1-16
filter function 1-14
in Time Series Tools 5-43
moving average 1-15
finite differences 1-24
Fourier analysis 3-1
calculating sunspot periodicity 3-7
calculating the FFT 3-4
calculation performance 3-13
phase and magnitude 3-11
functions
for data fitting 2-20
for data statistics 1-25
for Fourier analysis 3-3

G

goodness of fit 2-2

GUIs

for data fitting 1-3

for plotting 1-3

for statistics 1-3

for time-series analysis 1-3

H

histogram 5-21

used to select data 5-21

I

importing data

into MATLAB 1-6

into Time Series Tools 5-8

interpolating missing data 1-11

define method for time series 5-36

in Time Series Tools 5-43

isnan function 1-10

L

linear regression 2-1

load function 1-7

M

M-code from Time Series Tools 5-6

magnitude of Fourier transform 3-11

maximum 1-25

mean 1-25

median 1-25

methods

for timeseries object 4-28

for tscollection object 4-36

minimum 1-25

missing data

in calculations 1-9

in time series 5-12

interpolating 1-9

removing 1-9

removing in Time Series Tools 5-43

representing by NaNs 1-9

mode 1-25

moving-average filter 1-15

multiple regression 2-26

N

NaNs

in calculations 1-9

removing from data 1-10

nonpolynomial fit 2-24

O

objects for time-series analysis 4-2

outliers

removing 1-12

P

periodogram 5-19

filtering data from 5-20

phase of Fourier transform 3-11

plot function 1-8

plotting data

in MATLAB 1-7

in Time Series Tools 5-14

polyfit function 2-20

polynomial regression 2-20

polyval function 2-20

properties

of timeseries object 4-21

of tscollection object 4-34

Property Editor

in Time Series Tools 5-16

Q

quality codes for time-series data 5-37

R

range 1-25

regression 2-1

 multiple 2-26

 nonpolynomial 2-24

 polynomial 2-20

removing

 missing data 1-10

 NaNs 1-10

 outliers 1-12

resampling

 in Time Series Tools 5-43

 tscollection object 4-11

residuals 2-2

S

Simulink logged signals 5-8

spectral plot 5-19

 filtering data from 5-20

standard deviation 1-25

statistics

 formatting on plots 1-32

 functions 1-25

 in Time Series Tools 5-43

 MATLAB Data Statistics 1-28

 removing NaNs 1-10

 removing outliers 1-12

 showing on plots 1-29

sunspot periodicity

 calculating using Fourier transforms 3-7

T

time plot 5-18

Time Series Tools

 customizing plots 5-16

 define time-series units 5-33

 defining data quality 5-37

 defining events 5-39

 defining interpolation method 5-36

 detrending data 5-43

 editing data 5-33

 filtering data 5-43

 generating M-code 5-6

 getting help 5-3

 Import Wizard 5-9

 importing data 5-8

 interpolating data 5-43

 opening 5-2

 plot Property Editor 5-16

 plotting data 5-14

 removing missing data 5-43

 resampling data 5-43

 selecting data 5-29

 transforming data algebraically 5-43

 usage example 5-44

 viewing statistics 5-43

 window 5-3

 workflow 5-5

time vector

 format 4-18

 uniform 5-35

time-series analysis

 autocorrelation 5-23

 crosscorrelation 5-24

 example using methods 4-6

 example using Time Series Tools 5-44

 methods 4-1

 multivariate data 5-11

 using Time Series Tools 5-1

timeseries object

 constructor 4-19

 creating 4-18

 definition of data sample 4-3

 methods 4-28

- properties 4-21
- tools
 - MATLAB Basic Fitting 2-8
 - MATLAB Data Statistics 1-28
 - Time Series Tools 5-1
- transfer-function filter 1-16
- tscollection object
 - constructor 4-33
 - creating 4-33
 - methods 4-36
 - properties 4-34

U

- uniform time vector 5-35

V

- variance 1-25

W

- workflow
 - in Time Series Tools 5-5

X

- XY plot 5-27