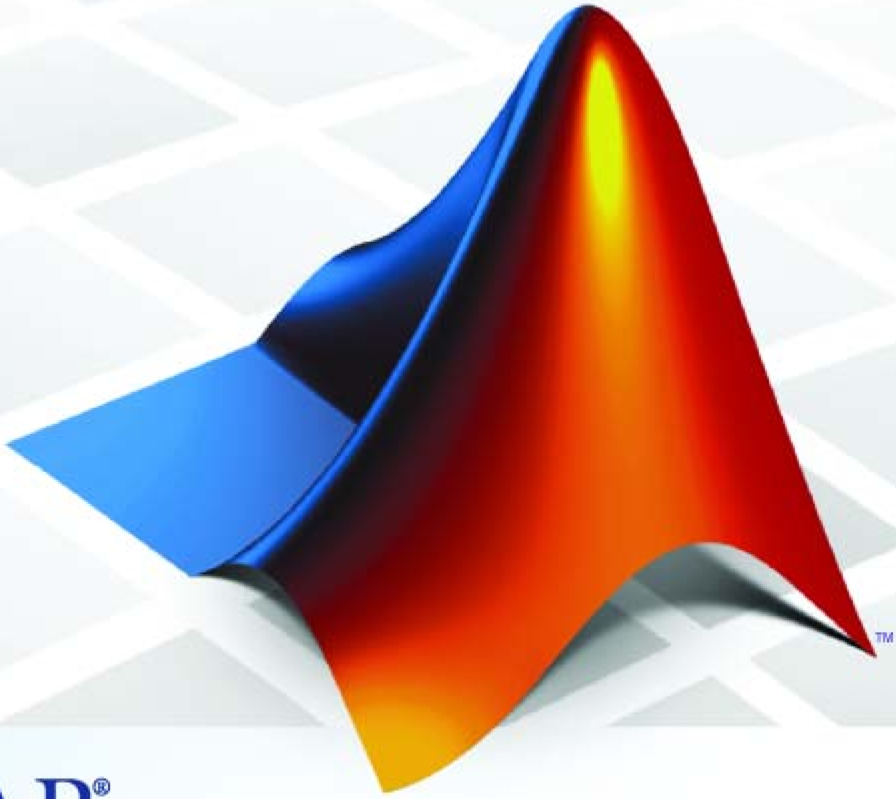
Curve Fitting Toolbox™ 1

User's Guide



MATLAB®

The MathWorks™

Accelerating the pace of engineering and science

How to Contact The MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	Technical Support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 (Phone)

508-647-7001 (Fax)

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

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

Curve Fitting Toolbox™ User's Guide

Trademarks

Patents

Revision History

July 2001	First printing	New for Version 1 (Release 12.1)
July 2002	Second printing	Revised for Version 1.1 (Release 13)
June 2004	Online only	Revised for Version 1.1.1 (Release 14)
October 2004	Online only	Revised for Version 1.1.2 (Release 14SP1)
March 2005	Online only	Revised for Version 1.1.3 (Release 14SP2)
June 2005	Third printing	Minor revision
September 2005	Online only	Revised for Version 1.1.4 (Release 14SP3)
March 2006	Online only	Revised for Version 1.1.5 (Release 2006a)
September 2006	Online only	Revised for Version 1.1.6 (Release 2006b)
November 2006	Fourth printing	Minor revision
March 2007	Online only	Revised for Version 1.1.7 (Release 2007a)
September 2007	Online only	Revised for Version 1.2 (Release 2007b)
March 2008	Online only	Revised for Version 1.2.1 (Release 2008a)
October 2008	Online only	Revised for Version 1.2.2 (Release 2008b)

Contents

1 Getting Started

2 Interactive Curve Fitting

3 Programmatic Curve Fitting

4 Curve Fitting Techniques

Getting Started

Product Overview

In this section...

Major Features

Curve Fitting Toolbox™ software is a collection of graphical user interfaces (GUIs) and M-file functions that operate in the MATLAB® technical computing environment. The toolbox supplements MATLAB features with:

- Data preprocessing capabilities, such as sectioning, excluding data, and smoothing
- Data fitting using parametric and nonparametric models:
  - The toolbox includes a library of parametric models, with polynomials, exponentials, rationals, sums of Gaussians, Fourier polynomials, and many others.
  - You can also define custom models to precisely reflect the goals of your data analysis.
  - Nonparametric models are available through a variety of smoothers and interpolants.
- Fitting methods for linear least squares, nonlinear least squares, weighted least squares, constrained least squares, and robust fitting are available
- Data and fit statistics to assist you in analyzing your models
- Postprocessing capabilities that allow you to interpolate, extrapolate, differentiate, and integrate the fit
- The ability to save your work in various formats, including workspace variables, binary files, and automatically generated MATLAB code

Interactive and Programmatic Environments

Curve Fitting Toolbox software allows you to work in two different environments:

- An interactive environment, Curve Fitting Tool, which is composed of multiple graphical user interfaces
- A programmatic environment that allows you to write object-oriented MATLAB code using curve fitting methods

To open Curve Fitting Tool, type

```
cftool
```

To list the Curve Fitting Toolbox functions for use in MATLAB programming, type

```
help curvefit
```

The code for any function can be opened in the MATLAB Editor by typing

```
edit function_name
```

Brief, command line help for any function is available by typing

```
help function_name
```

Complete documentation for any function is available by typing

```
doc function_name
```

You can change the way any toolbox function works by copying and renaming its M-file, examining your copy in the editor, and then modifying it.

You can also extend the toolbox by adding your own M-files, or by using your code in combination with functions from other toolboxes, such as Statistics Toolbox™ or Optimization Toolbox™ software.

Interactive Curve Fitting

In this section...

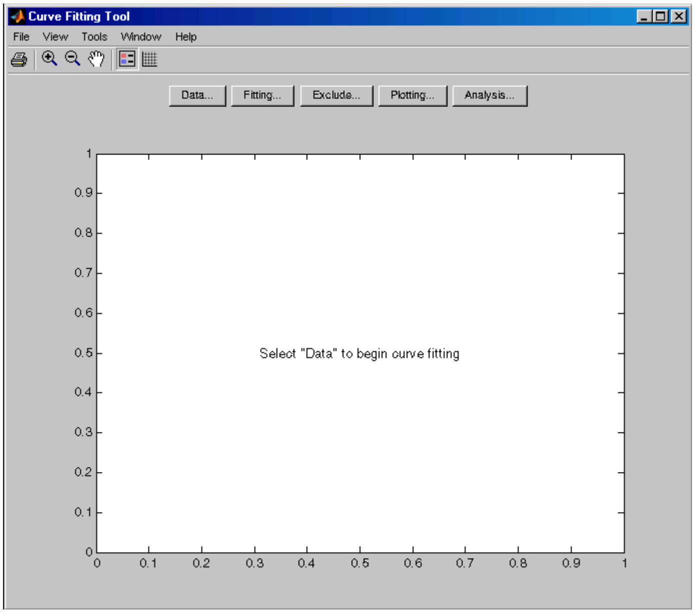
Opening Curve Fitting Tool

The Curve Fitting Tool is a graphical user interface (GUI) that allows you to

- Visually explore one or more data sets and fits as scatter plots.
- Graphically evaluate the goodness of fit using residuals and prediction bounds.
- Access additional interfaces for
  - Importing, viewing, and smoothing data
  - Fitting data, and comparing fits and data sets
  - Marking data points to be excluded from a fit
  - Selecting which fits and data sets are displayed in the tool
  - Interpolating, extrapolating, differentiating, or integrating fits

You open Curve Fitting Tool with the `cftool` command.

```
cftool
```



Importing Data

Before you can import data into Curve Fitting Tool, the data variables must exist in the MATLAB workspace. For this example, the data is stored in the MATLAB file census.mat.

```
load census
```

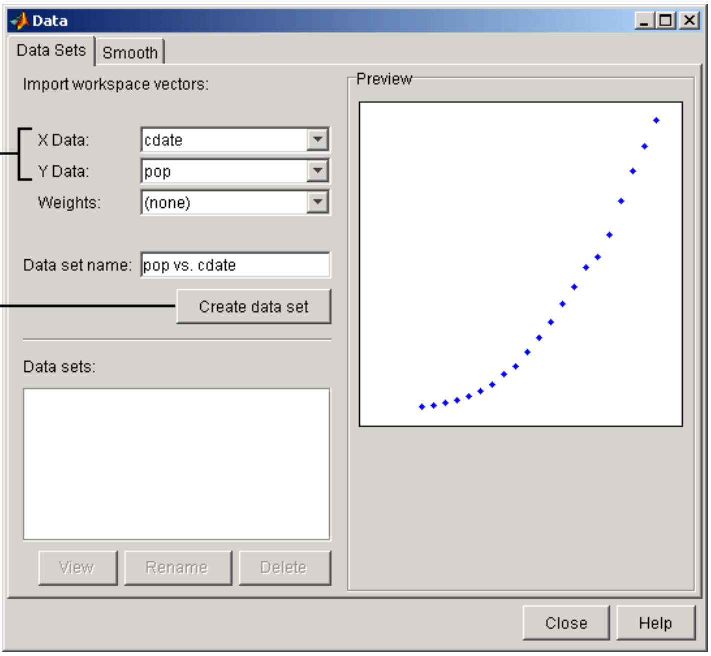
The workspace now contains two new variables, cdate and pop:

- cdate is a column vector containing the years 1790 to 1990 in 10-year increments.
- pop is a column vector with the US population figures that correspond to the years in cdate.

You can import data into Curve Fitting Tool with the Data GUI. You open this GUI by clicking the **Data** button on Curve Fitting Tool. As shown below, the Data GUI consists of two panes: Data sets and Smooth. The Data Sets pane allows you to

- Import predictor (X) data, response (Y) data, and weights. If you do not import weights, then they are assumed to be 1 for all data points.
- Specify the name of the data set.
- Preview the data.

To load `cdate` and `pop` into Curve Fitting Tool, select the appropriate variable names from the **X Data** and **Y Data** lists. The data is then displayed in the **Preview** window. Click the **Create data set** button to complete the data import process.



Import workspace vectors:

X Data: cdate

Y Data: pop

Weights: (none)

Data set name: pop vs. cdate

Select the data variable names.

Click Create data set to import the data.

The Smooth pane is described in "Preprocessing Data" on page 2-2.

Interactive Fitting

You fit data with the Fitting GUI. You open this GUI by clicking the **Fitting** button on Curve Fitting Tool. The Fitting GUI consists of two parts: the **Fit Editor** and the **Table of Fits**. The **Fit Editor** allows you to

- Specify the fit name, the current data set, and the exclusion rule.
- Explore various fits to the current data set using a library or custom equation, a smoothing spline, or an interpolant.
- Override the default fit options such as the coefficient starting values.
- Compare fit results including the fitted coefficients and goodness of fit statistics.

The **Table of Fits** allows you to

- Keep track of all the fits and their data sets for the current session.
- Display a summary of the fit results.
- Save or delete the fit results.

The Data Fitting Procedure

For this example, begin by fitting the census data with a second degree polynomial. Then continue fitting the data using polynomial equations up to sixth degree, and a single-term exponential equation.

The data fitting procedure follows these general steps:

1. From the **Fit Editor**, click **New Fit**.

   Note that this action always defaults to a linear polynomial fit type. You use **New Fit** at the beginning of your curve fitting session, and when you are exploring different fit types for a given data set.

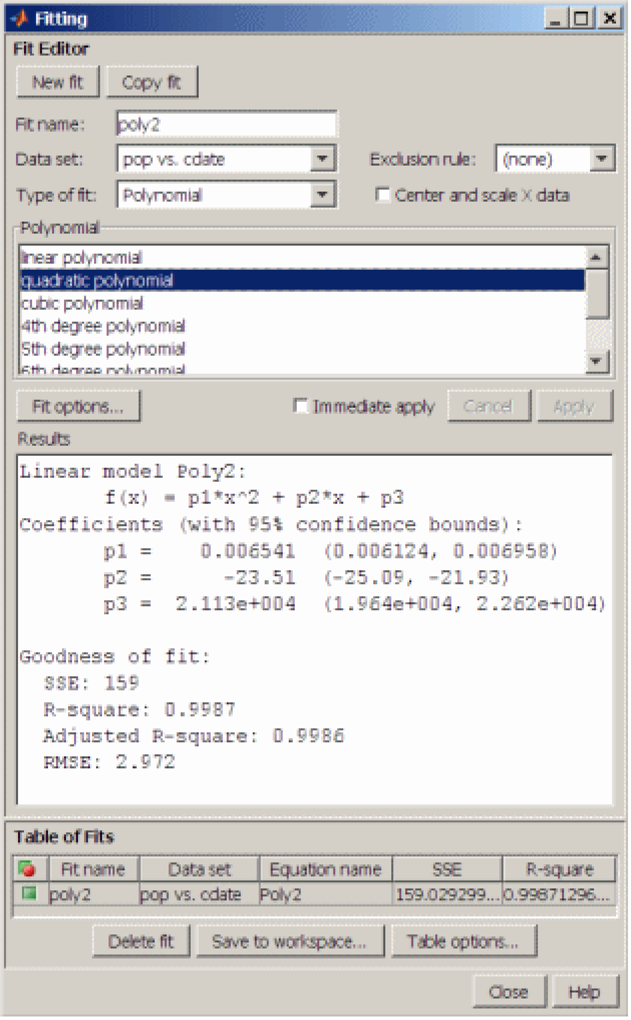
2. Because the initial fit uses a second degree polynomial, select **quadratic polynomial** from the **Polynomial** list. Name the fit `poly2`.

3. Click the **Apply** button or select the **Immediate apply** check box. The library model, fitted coefficients, and goodness of fit statistics are displayed in the **Results** area.

4. Fit the additional library equations.

> For fits of a given type (for example, polynomials), you should use **Copy Fit** instead of **New Fit** because copying a fit retains the current fit type state thereby requiring fewer steps than creating a new fit each time.

The Fitting GUI is shown below with the results of fitting the census data with a quadratic polynomial.



Fit name: poly2

Data set: pop vs. cdate

Exclusion rule: (none)

Type of fit: Polynomial

□ Center and scale X data

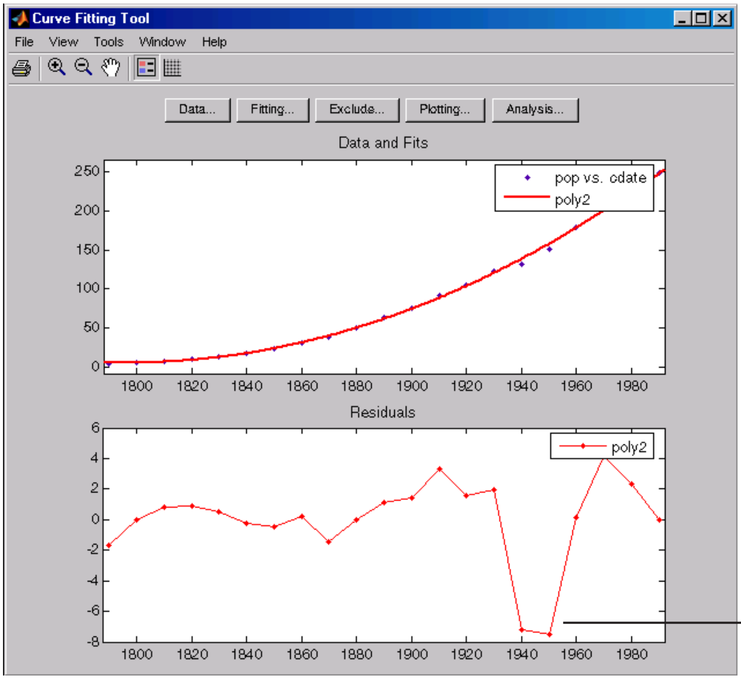
linear polynomial
quadratic polynomial
cubic polynomial
4th degree polynomial
5th degree polynomial
6th degree polynomial

□ Immediate apply

```
Linear model Poly2:
       f(x) = p1*x^2 + p2*x + p3
Coefficients (with 95% confidence bounds):
       p1 =    0.006541  (0.006124, 0.006958)
       p2 =      -23.51  (-25.09, -21.93)
       p3 =  2.113e+004  (1.964e+004, 2.262e+004)

Goodness of fit:
  SSE: 159
  R-square: 0.9987
  Adjusted R-square: 0.9986
  RMSE: 2.972
```

	Fit name	Data set	Equation name	SSE	R-square
🟩	poly2	pop vs. cdate	Poly2	159.029299...	0.99871296...

The data, fit, and residuals are shown below. You display the residuals as a line plot by selecting the menu item **View > Residuals > Line plot** from Curve Fitting Tool.



These residuals indicate that a better fit may be possible.

The residuals indicate that a better fit may be possible. Therefore, you should continue fitting the census data following the procedure outlined in the beginning of this section.

The residuals from a good fit should look random with no apparent pattern. A pattern, such as a tendency for consecutive residuals to have the same sign, can be an indication that a better model exists.

When you fit higher degree polynomials, the **Results** area displays this warning:

```
Equation is badly conditioned. Remove repeated data points
```

```
   or try centering and scaling.
```

The warning arises because the fitting procedure uses the `cdate` values as the basis for a matrix with very large values. The spread of the `cdate` values results in scaling problems. To address this problem, you can normalize the `cdate` data. Normalization is a process of scaling the predictor data to improve the accuracy of the subsequent numeric computations. A way to normalize `cdate` is to center it at zero mean and scale it to unit standard deviation.

```
(cdate - mean(cdate))./std(cdate)
```

To normalize data with Curve Fitting Tool, select the **Center and scale X data** check box.

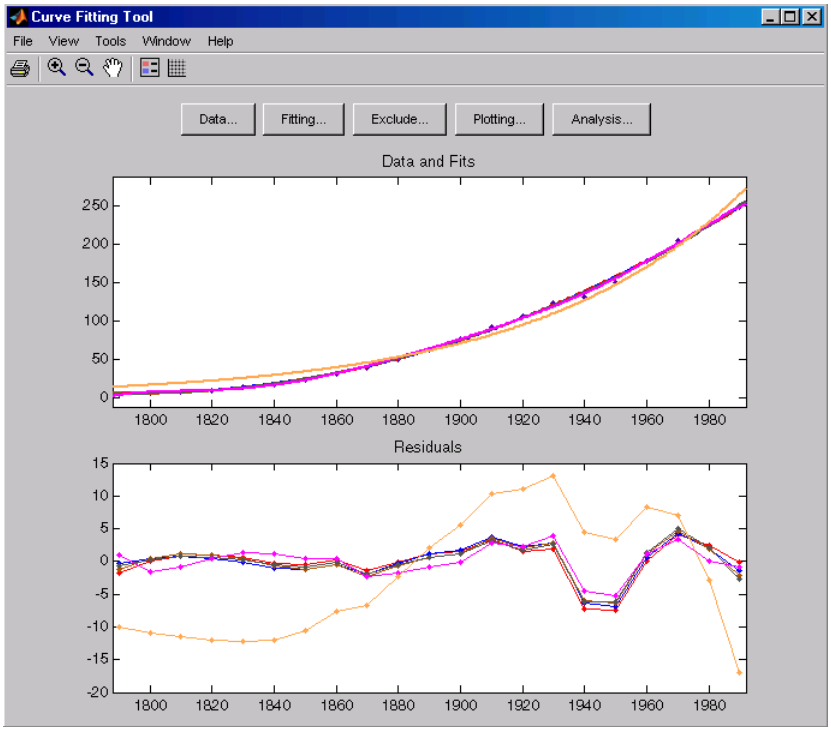
Note Because the predictor data changes after normalizing, the values of the fitted coefficients also change when compared to the original data. However, the functional form of the data and the resulting goodness of fit statistics do not change. Additionally, the data is displayed in Curve Fitting Tool using the original scale.

Determining the Best Fit

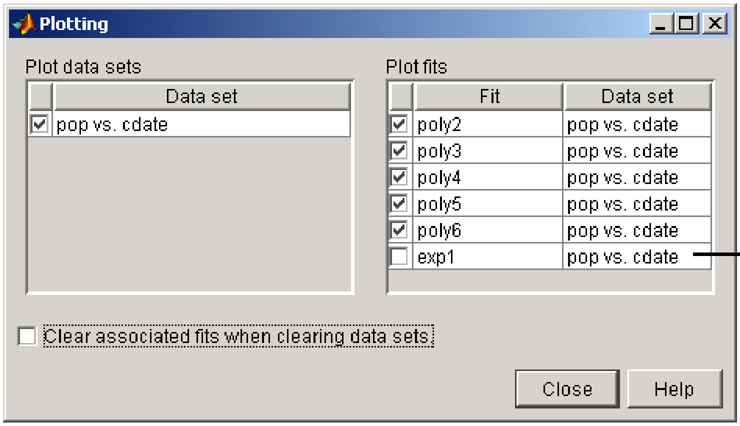
To determine the best fit, you should examine both the graphical and numerical fit results.

Examining the Graphical Fit Results. Your initial approach in determining the best fit should be a graphical examination of the fits and residuals. The graphical fit results shown below indicate that

- The fits and residuals for the polynomial equations are all similar, making it difficult to choose the best one.
- The fit and residuals for the single-term exponential equation indicate it is a poor fit overall. Therefore, it is a poor choice for extrapolation.



Use the Plotting GUI to remove `exp1` from the scatter plot display.



Plot data sets

	Data set
☑	pop vs. cdate

Plot fits

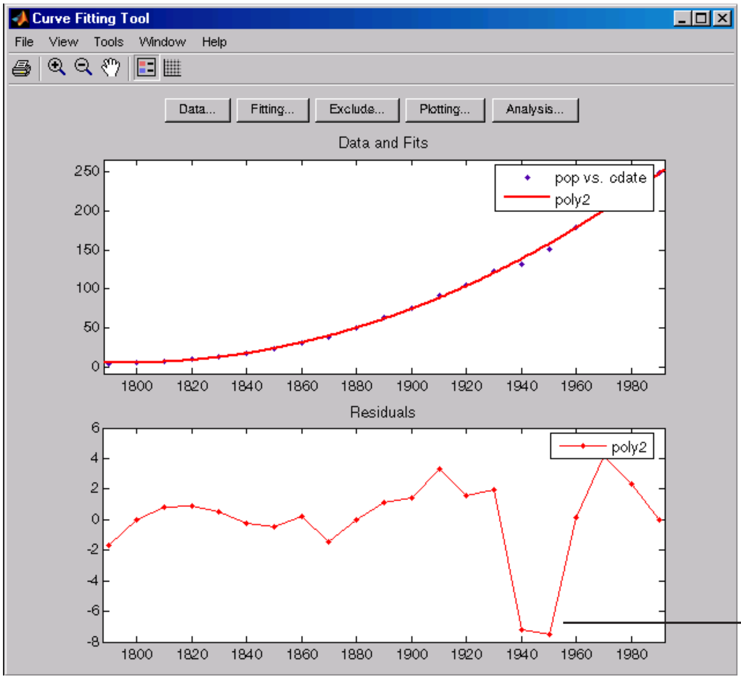
	Fit	Data set
☑	poly2	pop vs. cdate
☑	poly3	pop vs. cdate
☑	poly4	pop vs. cdate
☑	poly5	pop vs. cdate
☑	poly6	pop vs. cdate
☐	exp1	pop vs. cdate

☐ Clear associated fits when clearing data sets

Remove this fit from the scatter plot.

Because the goal of fitting the census data is to extrapolate the best fit to predict future population values, you should explore the behavior of the fits up to the year 2050. You can change the axes limits of Curve Fitting Tool by selecting the menu item **Tools > Axes Limit Control**.

The census data and fits are shown below for an upper abscissa limit of 2050. The behavior of the sixth degree polynomial fit beyond the data range makes it a poor choice for extrapolation.



These residuals indicate that a better fit may be possible.

As you can see, you should exercise caution when extrapolating with polynomial fits because they can diverge wildly outside the data range.

Examining the Numerical Fit Results. Because you can no longer eliminate fits by examining them graphically, you should examine the numerical fit results. There are two types of numerical fit results displayed in the Fitting GUI: goodness of fit statistics and confidence intervals on the fitted coefficients. The goodness of fit statistics help you determine how well the curve fits the data. The confidence intervals on the coefficients determine their accuracy.

Some goodness of fit statistics are displayed in the **Results** area of the **Fit Editor** for a single fit. All goodness of fit statistics are displayed in the **Table of Fits** for all fits, which allows for easy comparison.

In this example, the sum of squares due to error (SSE) and the adjusted R-square statistics are used to help determine the best fit. The SSE statistic is the least-squares error of the fit, with a value closer to zero indicating a better fit. The adjusted R-square statistic is generally the best indicator of the fit quality when you add additional coefficients to your model.

You can modify the information displayed in the **Table of Fits** with the Table Options GUI. You open this GUI by clicking the **Table options** button on the Fitting GUI. As shown below, select the adjusted R-square statistic and clear the R-square statistic.

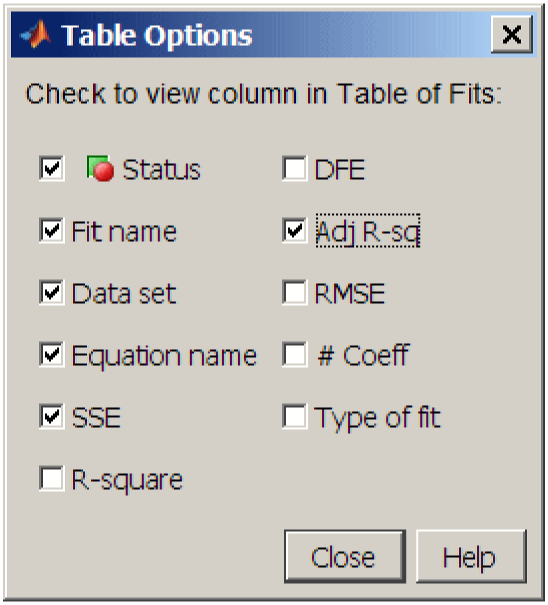


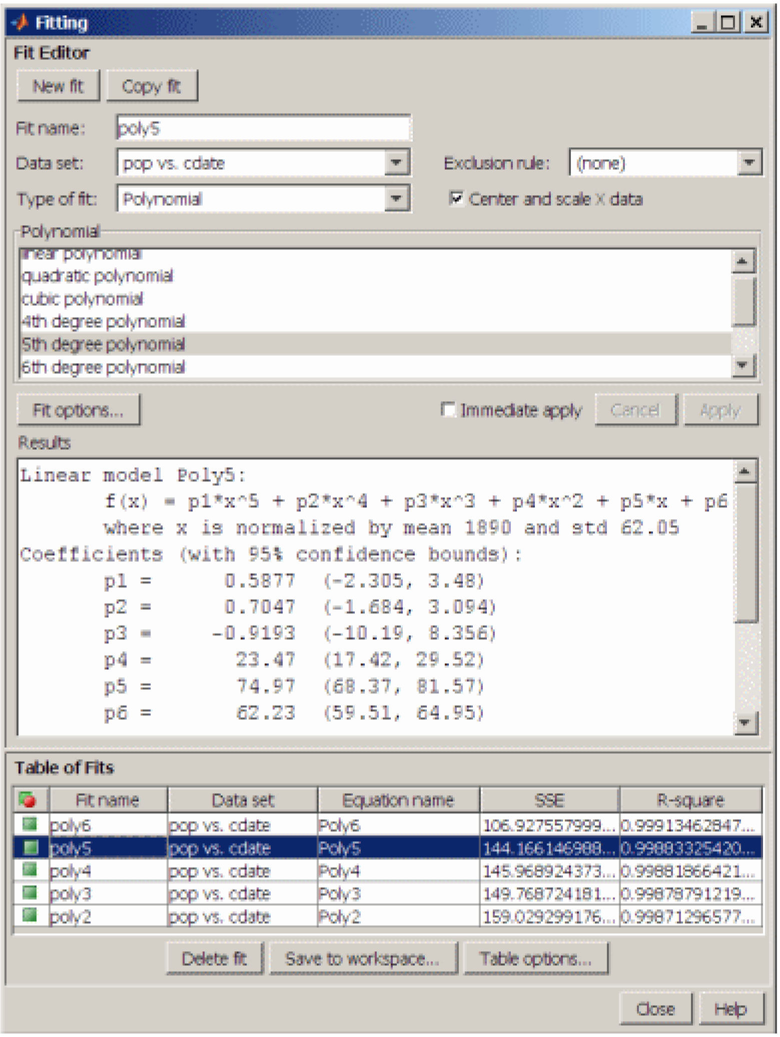
Table Options

Check to view column in Table of Fits:

☑ Status
☑ Fit name
☑ Data set
☑ Equation name
☑ SSE
□ R-square
□ DFE
☑ Adj R-sq
□ RMSE
□ # Coeff
□ Type of fit

The numerical fit results are shown below. You can click the **Table of Fits** column headings to sort by statistics results.

The SSE for `exp1` indicates it is a poor fit, which was already determined by examining the fit and residuals. The lowest SSE value is associated with `poly6`. However, the behavior of this fit beyond the data range makes it a poor choice for extrapolation. The next best SSE value is associated with the fifth degree polynomial fit, `poly5`, suggesting it may be the best fit. However, the SSE and adjusted *R*-square values for the remaining polynomial fits are all very close to each other. Which one should you choose?



Fit name: poly5

Data set: pop vs. cdate

Exclusion rule: (none)

Type of fit: Polynomial

☑ Center and scale X data

□ Immediate apply

```
Linear model Poly5:
       f(x) = p1*x^5 + p2*x^4 + p3*x^3 + p4*x^2 + p5*x + p6
       where x is normalized by mean 1890 and std 62.05
Coefficients (with 95% confidence bounds):
       p1 =      0.5877  (-2.305, 3.48)
       p2 =      0.7047  (-1.684, 3.094)
       p3 =     -0.9193  (-10.19, 8.356)
       p4 =       23.47  (17.42, 29.52)
       p5 =       74.97  (68.37, 81.57)
       p6 =       62.23  (59.51, 64.95)
```

Table of Fits

	Fit name	Data set	Equation name	SSE	R-square
🟩	poly6	pop vs. cdate	Poly6	106.927557999...	0.99913462847...
🟩	poly5	pop vs. cdate	Poly5	144.166146988...	0.99883325420...
🟩	poly4	pop vs. cdate	Poly4	145.968924373...	0.99881866421...
🟩	poly3	pop vs. cdate	Poly3	149.768724181...	0.99878791219...
🟩	poly2	pop vs. cdate	Poly2	159.029299176...	0.99871296577...

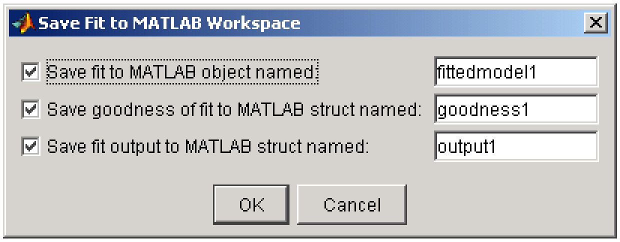
To resolve this issue, examine the confidence bounds for the remaining fits. By default, 95% confidence bounds are calculated. You can change this level by selecting the menu item **View > Confidence Level** from Curve Fitting Tool.

The p1, p2, and p3 coefficients for the fifth degree polynomial suggest that it overfits the census data. However, the confidence bounds for the quadratic fit, poly2, indicate that the fitted coefficients are known fairly accurately. Therefore, after examining both the graphical and numerical fit results, it appears that you should use poly2 to extrapolate the census data.

Note The fitted coefficients associated with the constant, linear, and quadratic terms are nearly identical for each polynomial equation. However, as the polynomial degree increases, the coefficient bounds associated with the higher degree terms increase, which suggests overfitting.

Saving the Fit Results

By clicking the **Save to workspace** button, you can save the selected fit and the associated fit results to the MATLAB workspace. The fit is saved as a MATLAB object and the associated fit results are saved as structures. This example saves all the fit results for the best fit, poly2.



Save Fit to MATLAB Workspace

☑ Save fit to MATLAB object named: fittedmodel1

☑ Save goodness of fit to MATLAB struct named: goodness1

☑ Save fit output to MATLAB struct named: output1

fittedmodel1 is saved as a Curve Fitting Toolbox cfit object.

```
whos fittedmodel1

  Name                  Size                    Bytes  Class
  fittedmodel1          1x1                      6178  cfit object

Grand total is 386 elements using 6178 bytes
```

The cfit object display includes the model, the fitted coefficients, and the confidence bounds for the fitted coefficients.

```
fittedmodel1
```

```
fittedmodel1 =
     Linear model Poly2:
       fittedmodel1(x) = p1*x^2 + p2*x + p3
     Coefficients (with 95% confidence bounds):
       p1 =    0.006541  (0.006124, 0.006958)
       p2 =      -23.51  (-25.09, -21.93)
       p3 =  2.113e+004  (1.964e+004, 2.262e+004)
```

The goodness1 structure contains goodness of fit results.

```
goodness1

goodness1 =
           sse: 159.0293
       rsquare: 0.9987
           dfe: 18
    adjrsquare: 0.9986
          rmse: 2.9724
```

The output1 structure contains additional information associated with the fit.

```
output1

output1 =
       numobs: 21
     numparam: 3
    residuals: [21x1 double]
     Jacobian: [21x3 double]
     exitflag: 1
    algorithm: 'QR factorization and solve'
```

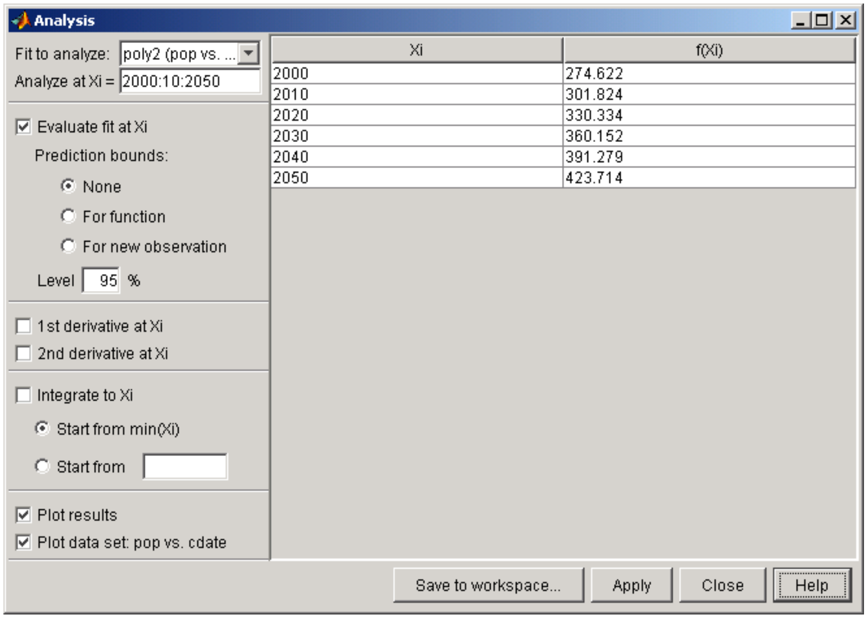
Analyzing the Fit

You can evaluate (interpolate or extrapolate), differentiate, or integrate a fit over a specified data range with the Analysis GUI. You open this GUI by clicking the **Analysis** button on Curve Fitting Tool.

For this example, you will extrapolate the quadratic polynomial fit to predict the US population from the year 2000 to the year 2050 in 10 year increments, and then plot both the analysis results and the data. To do this:

- Enter the appropriate MATLAB vector in the **Analyze at Xi** field.
- Select the **Evaluate fit at Xi** check box.
- Select the **Plot results** and **Plot data set** check boxes.
- Click the **Apply** button.

The numerical extrapolation results are shown below.



Specify the fit and data to analyze.

Fit to analyze: poly2 (pop vs. ...
Analyze at Xi = 2000:10:2050

Select this check box to extrapolate.

☑ Evaluate fit at Xi
Prediction bounds:
● None
○ For function
○ For new observation
Level 95 %

□ 1st derivative at Xi
□ 2nd derivative at Xi

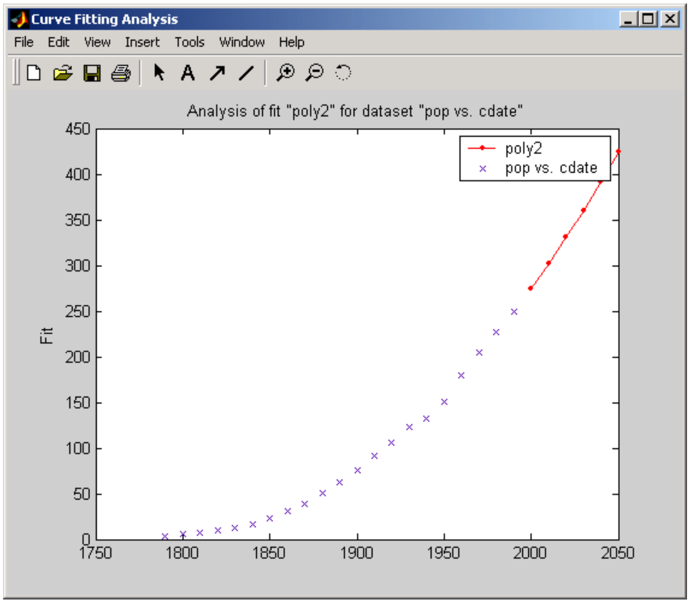
□ Integrate to Xi
● Start from min(Xi)
○ Start from

Plot both the analysis results and the data.

☑ Plot results
☑ Plot data set: pop vs. cdate

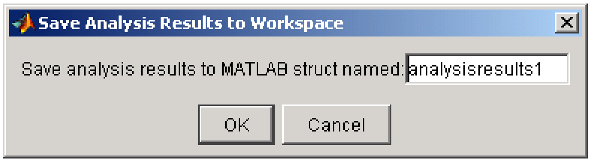
Xi	f(Xi)
2000	274.622
2010	301.824
2020	330.334
2030	360.152
2040	391.279
2050	423.714

The extrapolated values and the census data set are displayed together in a new figure window.



Saving the Analysis Results

By clicking the **Save to workspace** button, you can save the extrapolated values as a structure to the MATLAB workspace.



Save analysis results to MATLAB struct named: analysisresults1

The resulting structure is shown below.

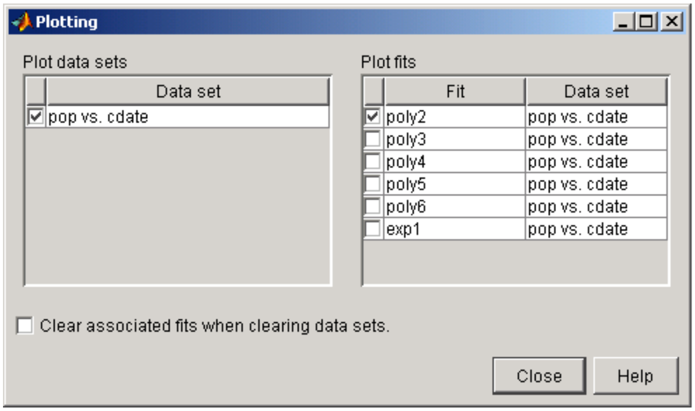
```
analysisresults1

analysisresults1 =
       xi: [6x1 double]
     yfit: [6x1 double]
```

Saving Your Work

Curve Fitting Toolbox software provides you with several options for saving your work. You can save one or more fits and the associated fit results as variables to the MATLAB workspace. You can then use this saved information for documentation purposes, or to extend your data exploration and analysis. In addition to saving your work to MATLAB workspace variables, you can

- “Save the Session” on page 1-20
- “Generate an M-File” on page 1-21

Before performing any of these tasks, you may want to remove unwanted data sets and fits from Curve Fitting Tool display. An easy way to do this is with the Plotting GUI. The Plotting GUI shown below is configured to display only the census data and the best fit, `poly2`.



Plot data sets

	Data set
☑	pop vs. cdate

Plot fits

	Fit	Data set
☑	poly2	pop vs. cdate
☐	poly3	pop vs. cdate
☐	poly4	pop vs. cdate
☐	poly5	pop vs. cdate
☐	poly6	pop vs. cdate
☐	exp1	pop vs. cdate

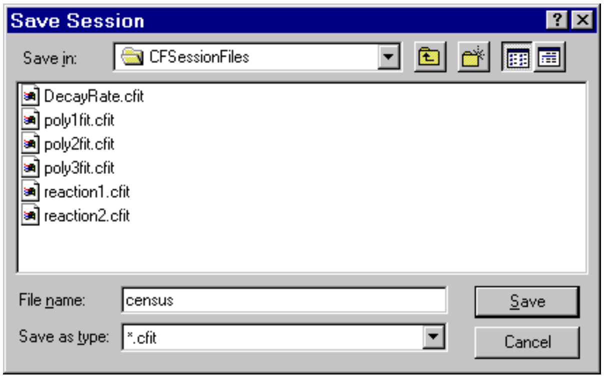
☐ Clear associated fits when clearing data sets.

Clear the remaining fits associated with the census data except the best fit.

Save the Session

The curve fitting session is defined as the current collection of fits for all data sets. You may want to save your session so that you can continue data exploration and analysis at a later time using Curve Fitting Tool without losing any current work.

Save the current curve fitting session by selecting the menu item **File > Save Session** from Curve Fitting Tool. The Save Session dialog is shown below.



Save in: CFSessionFiles

DecayRate.cfit
poly1fit.cfit
poly2fit.cfit
poly3fit.cfit
reaction1.cfit
reaction2.cfit

File name: census

Save as type: *.cfit

The session is stored in binary form in a `cfit` file, and contains this information:

- All data sets and associated fits
- The state of the Fitting GUI, including **Table of Fits** entries and exclusion rules
- The state of the Plotting GUI

To avoid saving unwanted data sets, you should delete them from Curve Fitting Tool. You delete data sets using the Data Sets pane of the Data GUI. If there are fits associated with the unwanted data sets, they are deleted as well.

You can load a saved session by selecting the menu item **File > Load Session** from Curve Fitting Tool. When the session is loaded, the saved state of Curve Fitting Tool display is reproduced, and may display the data, fits, residuals, and so on. If you open the Fitting GUI, then the loaded fits are displayed in the **Table of Fits**. Select a fit from this table to continue your curve fitting session.

Generate an M-File

You may want to generate an M-file that captures your work, so that you can continue your analysis outside of Curve Fitting Tool. The M-file can be used without modification, or it can be edited as needed.

To generate an M-file from a session in Curve Fitting Tool, select the menu item **File > Generate M-file**.

The M-file captures the following information from Curve Fitting Tool:

- Names of variables, fits, and residuals
- Fit options, such as whether the data should be normalized, initial values for the coefficients, and the fitting method
- Curve fitting objects and methods used to create the fit

You can recreate your Curve Fitting Tool session by calling the M-file from the command line with your original data as input arguments. You can also call the M-file with new data, and automate the process of fitting multiple data sets.

For more information on working with a generated M-file, see “Interactive Code Generation” on page 1-26.

Programmatic Curve Fitting

In this section...

Curve Fitting Objects and Methods

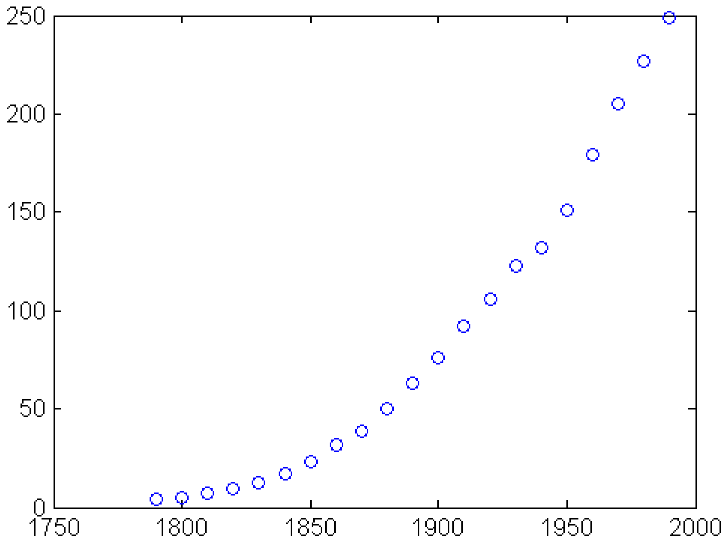
The Curve Fitting Tool is a graphical user interface that allows convenient, interactive use of Curve Fitting Toolbox functions, without programming. You can, however, access Curve Fitting Toolbox functions directly, and write programs that combine curve fitting functions with MATLAB functions and functions from other toolboxes. This allows you to create a curve fitting environment that is precisely suited to your needs.

Models and fits in Curve Fitting Tool are managed internally as curve fitting *objects*. Objects are manipulated through a variety of functions called *methods*. You can create curve fitting objects, and apply curve fitting methods, outside of Curve Fitting Tool.

For example, the following code, using Curve Fitting Toolbox methods, reproduces an analysis of the census data that was carried out interactively in Curve Fitting Tool in "Interactive Curve Fitting" on page 1-4.

Load and plot the data in `census.mat`:

```
load census
plot(cdate,pop,'o')
hold on
```



Create a fit options structure and a `fittype` object for the custom nonlinear model $y = a(x–b)^n$, where a and b are coefficients and n is a problem-dependent parameter:

```
s = fitoptions('Method','NonlinearLeastSquares',...
               'Lower',[0,0],...
               'Upper',[Inf,max(cdate)],...
               'Startpoint',[1 1]);
f = fittype('a*(x-b)^n','problem','n','options',s);
```

Fit the data using the fit options and a value of $n = 2$:

```
[c2,gof2] = fit(cdate,pop,f,'problem',2)
c2 =
     General model:
       c2(x) = a*(x-b)^n
     Coefficients (with 95% confidence bounds):
       a =    0.006092  (0.005743, 0.006441)
       b =        1789  (1784, 1793)
     Problem parameters:
       n =           2
gof2 =
           sse: 246.1543
```

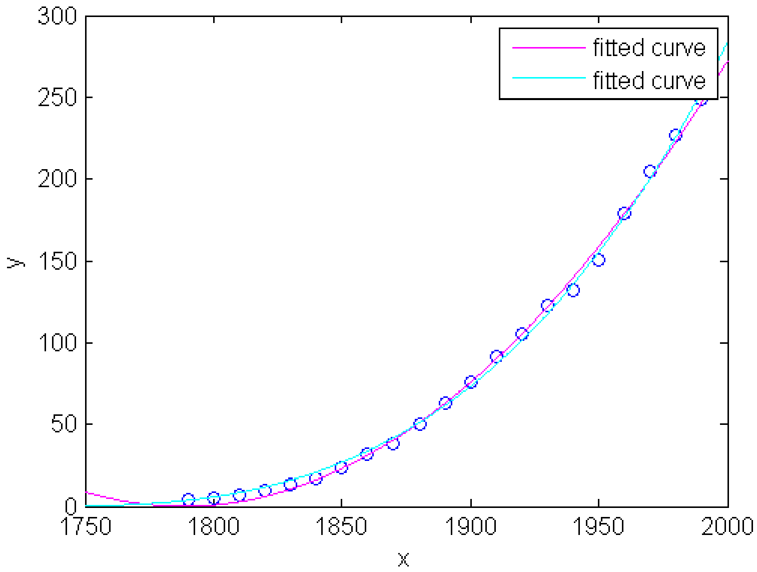
```
        rsquare: 0.9980
            dfe: 19
     adjrsquare: 0.9979
           rmse: 3.5994
```

Fit the data using the fit options and a value of $n = 3$:

```
[c3,gof3] = fit(cdate,pop,f,'problem',3)
c3 =
     General model:
       c3(x) = a*(x-b)^n
     Coefficients (with 95% confidence bounds):
       a =  1.359e-005  (1.245e-005, 1.474e-005)
       b =        1725  (1718, 1731)
     Problem parameters:
       n =           3
gof3 =
           sse: 232.0058
       rsquare: 0.9981
           dfe: 19
    adjrsquare: 0.9980
          rmse: 3.4944
```

Plot the fit results with the data:

```
plot(c2,'m')
plot(c3,'c')
```



Interactive Code Generation

Curve fitting code can be assembled into an M-file by hand, as shown in "Curve Fitting Objects and Methods" on page 1-23, or it can be generated automatically from an interactive session in Curve Fitting Tool, as described in "Generate an M-File" on page 1-21. In practice, automatically generated code, giving the broad outlines of an analysis, can be combined with hand-coded refinements. This allows you to write functions that are customized to your data and your analysis, without having to write all of the basic programming structures.

For example, the following M-file was generated from a session in Curve Fitting Tool that imported the data from `census.mat` and fit a custom nonlinear model of the form $y = a(x–b)^3$:

```
function myfit(cdate,pop)
%MYFIT    Create plot of datasets and fits
%   MYFIT(CDATE,POP)
%   Creates a plot, similar to the plot in the main curve fitting
%   window, using the data that you provide as input.  You can
%   apply this function to the same data you used with cftool
%   or with different data.  You may want to edit the function to
%   customize the code and this help message.
%
%   Number of datasets:  1
%   Number of fits:  1


% Data from dataset "pop vs. cdate":
%    X = cdate:
%    Y = pop:
%    Unweighted
%
% This function was automatically generated on 11-Sep-2007 01:07:11

% Set up figure to receive datasets and fits
f_ = clf;
figure(f_);
set(f_,'Units','Pixels','Position',[439.6 193.6 814.4 576.8]);
legh_ = []; legt_ = {};   % handles and text for legend
xlim_ = [Inf -Inf];       % limits of x axis
ax_ = axes;
set(ax_,'Units','normalized','OuterPosition',[0 0 1 1]);
set(ax_,'Box','on');
axes(ax_); hold on;


% --- Plot data originally in dataset "pop vs. cdate"
cdate = cdate(:);
pop = pop(:);
h_ = line(cdate,pop,'Parent',ax_,'Color',[0.333333 0 0.666667],...
     'LineStyle','none', 'LineWidth',1,...
     'Marker','.', 'MarkerSize',12);
xlim_(1) = min(xlim_(1),min(cdate));
xlim_(2) = max(xlim_(2),max(cdate));
legh_(end+1) = h_;
legt_{end+1} = 'pop vs. cdate';

% Nudge axis limits beyond data limits
if all(isfinite(xlim_))
   xlim_ = xlim_ + [-1 1] * 0.01 * diff(xlim_);
   set(ax_,'XLim',xlim_)
else
    set(ax_, 'XLim',[1788, 1992]);
end


% --- Create fit "fit 1"
ok_ = isfinite(cdate) & isfinite(pop);
if ~all( ok_ )
    warning( 'GenerateMFile:IgnoringNansAndInfs', ...
        'Ignoring NaNs and Infs in data' );
end
st_ = [0.51510504095942344 0.35210694524343056 ];
ft_ = fittype('a*(x-b)^3',...
     'dependent',{'y'},'independent',{'x'},...
     'coefficients',{'a', 'b'});

% Fit this model using new data
cf_ = fit(cdate(ok_),pop(ok_),ft_,'Startpoint',st_);

% Or use coefficients from the original fit:
if 0
   cv_ = { 1.3594203554767276e-005, 1724.6959436137356};
   cf_ = cfit(ft_,cv_{:});
end

% Plot this fit
h_ = plot(cf_,'fit',0.95);
legend off;  % turn off legend from plot method call
set(h_(1),'Color',[1 0 0],...
     'LineStyle','-', 'LineWidth',2,...
     'Marker','none', 'MarkerSize',6);
legh_(end+1) = h_(1);
legt_{end+1} = 'fit 1';

% Done plotting data and fits.  Now finish up loose ends.
hold off;
leginfo_ = {'Orientation', 'vertical', 'Location', 'NorthEast'};
h_ = legend(ax_,legh_,legt_,leginfo_{:});  % create legend
set(h_,'Interpreter','none');
```

```
xlabel(ax_,'');                 % remove x label
ylabel(ax_,'');                 % remove y label
```

A quick look through the code shows that it has automatically assembled for you many of the Curve Fitting Toolbox curve fitting methods, such as `fitoptions`, `fittype`, `fit`, and `plot`.

A natural modification of the M-file would be to edit the function declaration at the top of the file to return the `cfit` object created by the fit, as follows:

```
function cf_ = myfit(cdate,pop)
```

You might also modify the code to produce a variety of different plots, or to return goodness-of-fit statistics.

Coding with curve fitting objects and methods is given complete treatment in Chapter 3, “Programmatic Curve Fitting”.

2

Interactive Curve Fitting

Preprocessing Data

In this section...

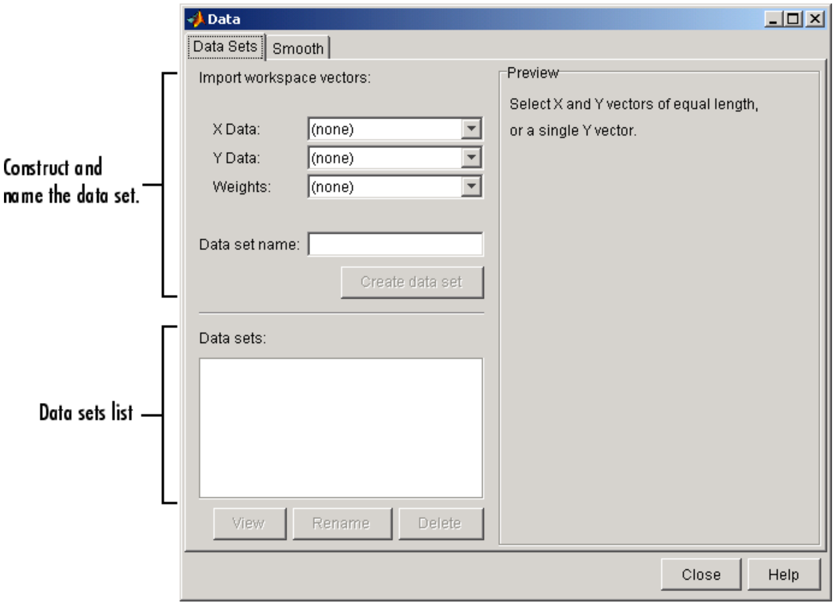
Importing Data

Introduction

You import data sets into Curve Fitting Tool with the Data Sets pane of the Data GUI. Using this pane, you can

- Select workspace variables that compose a data set
- Display a list of all imported data sets
- View, delete, or rename one or more data sets

The Data Sets pane is shown below followed by a description of its features.



X Data: (none)

Y Data: (none)

Weights: (none)

Data set name:

Select X and Y vectors of equal length, or a single Y vector.

Creating a Data Set

- **Import workspace vectors** — All selected variables must be the same length. You can import only vectors, not matrices or scalars. `Inf`s and `NaN`s are ignored because you cannot fit data containing these values, and only the real part of a complex number is used. To perform any curve-fitting task, you must select at least one vector of data:
  - **X data** — Select the predictor data.
  - **Y data** — Select the response data.
  - **Weights** — Select the weights associated with the response data. If weights are not imported, they are assumed to be 1 for all data points.
- **Preview** — The selected workspace vectors are displayed graphically in the preview window. Weights are not displayed.
- **Data set name** — The name of the imported data set. The toolbox automatically creates a unique name for each imported data set. You can change the name by editing this field. Click the **Create data set** button to complete the data import process.

Working with Data Sets

- **Data sets** — Lists all data sets added to Curve Fitting Tool. The data sets can be created from workspace variables, or from smoothing an existing imported data set. When you select a data set, you can perform these actions:
  - Click **View** to open the View Data Set GUI. Using this GUI, you can view a single data set both graphically and numerically. Additionally, you can display data points to be excluded in a fit by selecting an exclusion rule.
  - Click **Rename** to change the name of a single data set.
  - Click **Delete** to delete one or more data sets. To select multiple data sets, you can use the **Ctrl** key and the mouse to select data sets one by one, or you can use the **Shift** key and the mouse to select a range of data sets.

Example: Importing Data

This example imports the ENSO data set into the Curve Fitting Tool using the Data Sets pane of the Data GUI. The first step is to load the data from the file `enso.mat` into the MATLAB workspace.

```
load enso
```

The workspace contains two new variables, `pressure` and `month`:

- `pressure` is the monthly averaged atmospheric pressure differences between Easter Island and Darwin, Australia. This difference drives the trade winds in the southern hemisphere.
- `month` is the relative time in months.

Alternatively, you can import data by specifying the variable names as arguments to the `cftool` function.

```
cftool(month,pressure)
```

In this case, the Data GUI is not opened.

The data import process is described below:

1 Select workspace variables.

The predictor and response data are displayed graphically in the **Preview** window. Weights and data points containing `Infs` or `NaNs` are not displayed.

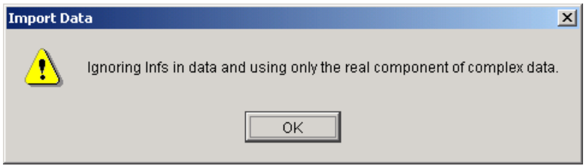
2 Specify the data set name.

You should specify a meaningful name when you import multiple data sets. If you do not specify a name, the default name, which is constructed from the selected variable names, is used.

3 Click the **Create data set** button.

The **Data sets** list box displays all the data sets added to the toolbox. Note that you can construct data sets from workspace variables, or by smoothing an existing data set.

If your data contains `Infs` or complex values, a warning message like this appears.

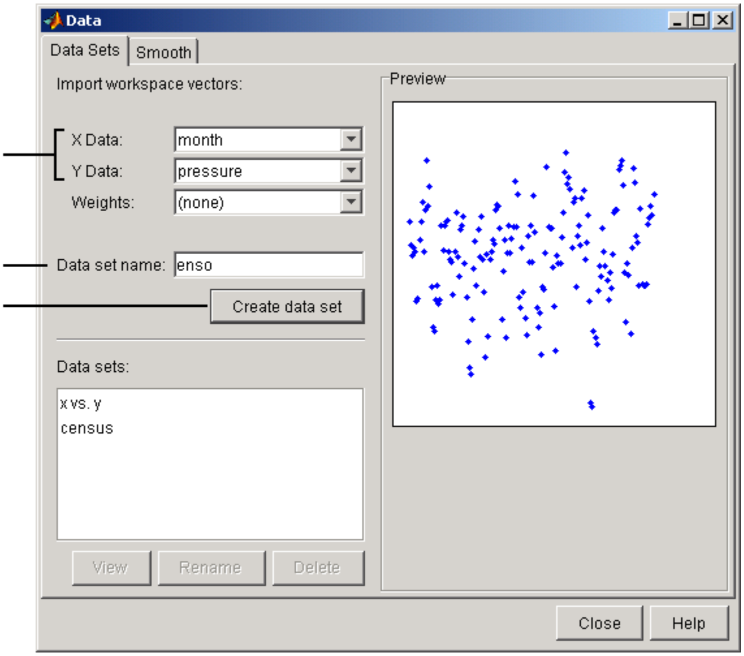


Import Data

Ignoring Infs in data and using only the real component of complex data.

After you click the **Create data set** window.

The Data Sets pane shown below displays the imported ENSO data in the **Preview** button, the data set `enso` is added to the **Data sets** list box. You can then view, rename, or delete `enso` by selecting it in the list box and clicking the appropriate button.



X Data: month

Y Data: pressure

Weights: (none)

Data set name: enso

Data sets:
x vs. y
census

Select the workspace variable names.

Specify the data set name.

Click **Create data set** to import the data.

Viewing Data

- "Viewing Data Graphically" on page 2-6
- "Viewing Data Numerically" on page 2-8

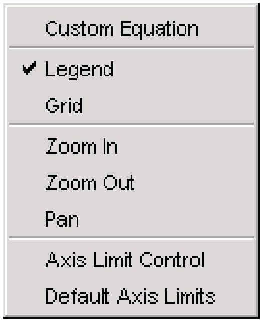
Viewing Data Graphically

After you import a data set, it is automatically displayed as a scatter plot in Curve Fitting Tool. The response data is plotted on the vertical axis and the predictor data is plotted on the horizontal axis.

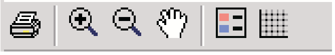
The scatter plot is a powerful tool because it allows you to view the entire data set at once, and it can easily display a wide range of relationships between the two variables. You should examine the data carefully to determine whether preprocessing is required, or to deduce a reasonable fitting approach. For example, it's typically very easy to identify outliers in a scatter plot, and to determine whether you should fit the data with a straight line, a periodic function, a sum of Gaussians, and so on.

Enhancing the Graphical Display. Curve Fitting Toolbox software provides several tools for enhancing the graphical display of a data set. These tools are available through the **Tools** menu, the GUI toolbar, and right-click menus.

You can zoom in or out, turn on or off the grid, and so on using the **Tools** menu and the GUI toolbar shown below.

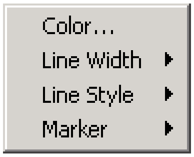


Tools Menu



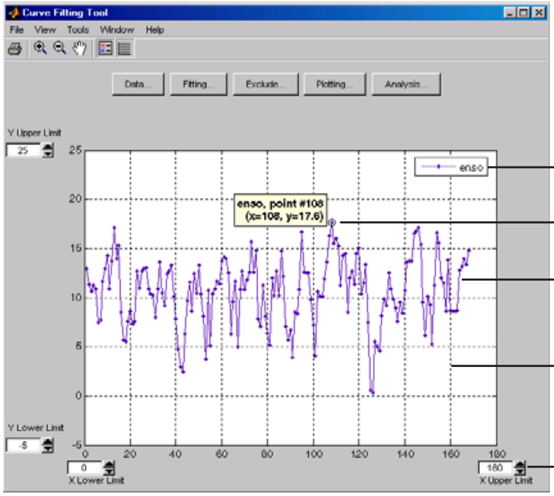
GUI Toolbar

You can change the color, line width, line style, and marker type of the displayed data points using the right-click menu shown below. You activate this menu by placing your mouse over a data point and right-clicking. Note that a similar menu is available for fitted curves.



Right-click menu

The ENSO data is shown below after the display has been enhanced using several of these tools.



Display the legend for the ENSO data set.

Display data tips using MATLAB's click functionality.

Change the color, marker type and line style for the data.

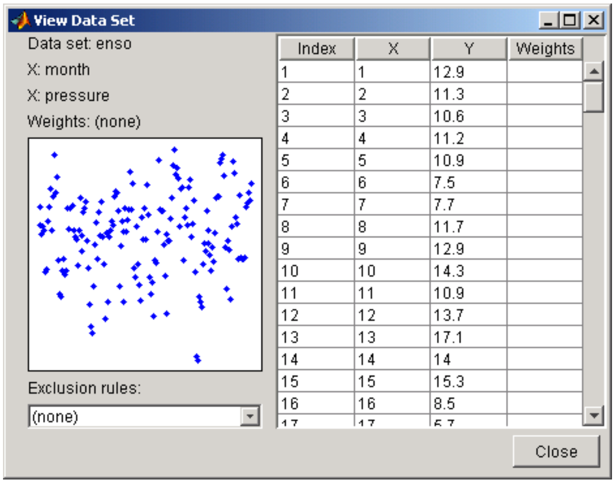
Display the grid.

Change the axis limits.

Viewing Data Numerically

You can view the numerical values of a data set, as well as data points to be excluded from subsequent fits, with the View Data Set GUI. You open this GUI by selecting a name in the **Data sets** list box of the Data GUI and clicking the **View** button.

The View Data Set GUI for the ENSO data set is shown below, followed by a description of its features.



Data set: enso

X: month

X: pressure

Weights: (none)

Exclusion rules: (none)

Index	X	Y	Weights
1	1	12.9	
2	2	11.3	
3	3	10.6	
4	4	11.2	
5	5	10.9	
6	6	7.5	
7	7	7.7	
8	8	11.7	
9	9	12.9	
10	10	14.3	
11	11	10.9	
12	12	13.7	
13	13	17.1	
14	14	14	
15	15	15.3	
16	16	8.5	
17	17	5.7	

- **Data set** — Lists the names of the viewed data set and the associated variables. The data is displayed graphically below this list.

  The index, predictor data (**X**), response data (**Y**), and weights (if imported) are displayed numerically in the table. If the data contains `Infs` or `NaNs`, those values are labeled "ignored." If the data contains complex numbers, only the real part is displayed.

- **Exclusion rules** — Lists all the exclusion rules that are compatible with the viewed data set. When you select an exclusion rule, the data points marked for exclusion are grayed in the table, and are identified with an "x" in the graphical display. To exclude the data points while fitting, you must create the exclusion rule in the Exclude GUI and select the exclusion rule in the Fitting GUI.

  An exclusion rule is compatible with the viewed data set if their lengths are the same, or if it is created by sectioning only.

Smoothing Data

- "Introduction" on page 2-10
- "Creating a Smoothed Data Set" on page 2-12

- "Smoothing Method" on page 2-12
- "Working with Smoothed Data Sets" on page 2-13
- "Example: Smoothing Data" on page 2-13

Introduction

If your data is noisy, you might need to apply a smoothing algorithm to expose its features, and to provide a reasonable starting approach for parametric fitting. The two basic assumptions that underlie smoothing are

- The relationship between the response data and the predictor data is smooth.
- The smoothing process results in a smoothed value that is a better estimate of the original value because the noise has been reduced.

  The smoothing process attempts to estimate the average of the distribution of each response value. The estimation is based on a specified number of neighboring response values.

You can think of smoothing as a local fit because a new response value is created for each original response value. Therefore, smoothing is similar to some of the nonparametric fit types supported by the toolbox, such as smoothing spline and cubic interpolation. However, this type of fitting is not the same as parametric fitting, which results in a global parameterization of the data.

Note You should not fit data with a parametric model after smoothing, because the act of smoothing invalidates the assumption that the errors are normally distributed. Instead, you should consider smoothing to be a data exploration technique.

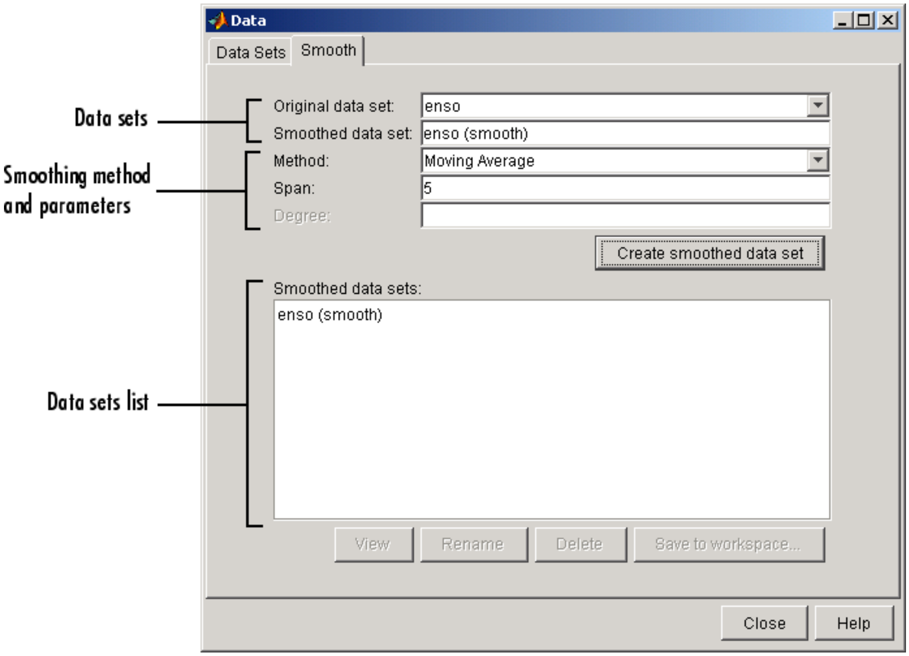
There are two common types of smoothing methods: filtering (averaging) and local regression. Each smoothing method requires a *span*. The span defines a window of neighboring points to include in the smoothing calculation for each data point. This window moves across the data set as the smoothed response value is calculated for each predictor value. A large span increases the smoothness but decreases the resolution of the smoothed data set, while a small span decreases the smoothness but increases the resolution of the smoothed data set. The optimal span value depends on your data set and the smoothing method, and usually requires some experimentation to find.

Curve Fitting Toolbox software supports these smoothing methods:

- Moving average filtering — Lowpass filter that takes the average of neighboring data points.
- Lowess and loess — Locally weighted scatter plot smooth. These methods use linear least-squares fitting, and a first-degree polynomial (lowess) or a second-degree polynomial (loess). Robust lowess and loess methods that are resistant to outliers are also available.
- Savitzky-Golay filtering — A generalized moving average where you derive the filter coefficients by performing an unweighted linear least-squares fit using a polynomial of the specified degree.

Note that you can also smooth data using a smoothing spline. Refer to "Nonparametric Fitting" on page 2-75 for more information.

You smooth data with the Smooth pane of the Data GUI. The pane is shown below followed by a description of its features.



Original data set: enso

Smoothed data set: enso (smooth)

Method: Moving Average

Span: 5

Smoothed data sets:

enso (smooth)

Creating a Smoothed Data Set

- **Original data set** — Select the data set you want to smooth.
- **Smoothed data set** — Specify the name of the smoothed data set. Note that the process of smoothing the original data set always produces a new data set containing smoothed response values.

Smoothing Method

- **Method** — Select the smoothing method. Each response value is replaced with a smoothed value that is calculated by the specified smoothing method.
  - **Moving average** — Filter the data by calculating an average.
  - **Lowess** — Locally weighted scatter plot smooth using linear least-squares fitting and a first-degree polynomial.

  - **Loess** — Locally weighted scatter plot smooth using linear least-squares fitting and a second-degree polynomial.
  - **Savitzky-Golay** — Filter the data with an unweighted linear least-squares fit using a polynomial of the specified degree.
  - **Robust Lowess** — Lowess method that is resistant to outliers.
  - **Robust Loess** — Loess method that is resistant to outliers.
- **Span** — The number of data points used to compute each smoothed value.

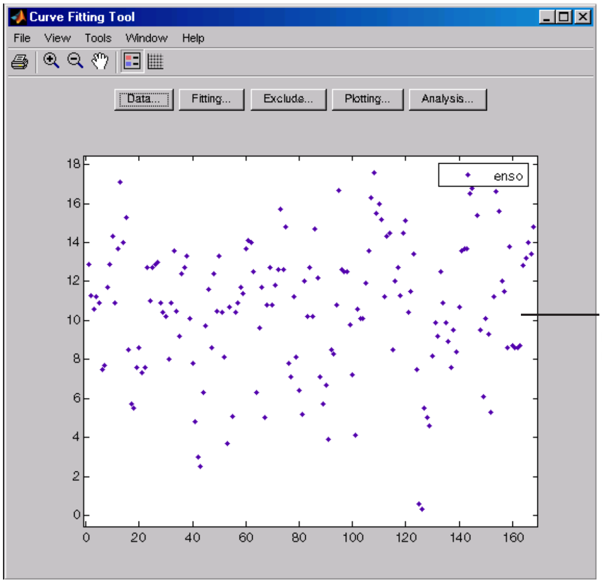
  For the moving average and Savitzky-Golay methods, the span must be odd. For all locally weighted smoothing methods, if the span is less than 1, it is interpreted as the percentage of the total number of data points.
- **Degree** — The degree of the polynomial used in the Savitzky-Golay method. The degree must be smaller than the span.

Working with Smoothed Data Sets

- **Smoothed data sets** — Lists all the smoothed data sets. You add a smoothed data set to the list by clicking the **Create smoothed data set** button. When you select a data set from the list, you can perform these actions:
  - Click **View** to open the View Data Set GUI. Using this GUI, you can view a single data set both graphically and numerically. Additionally, you can display data points to be excluded in a fit by selecting an exclusion rule.
  - Click **Rename** to change the name of a single data set.
  - Click **Delete** to delete one or more data sets. To select multiple data sets, you can use the **Ctrl** key and the mouse to select data sets one by one, or you can use the **Shift** key and the mouse to select a range of data sets.
  - Click **Save to workspace** to save a single data set to a structure.

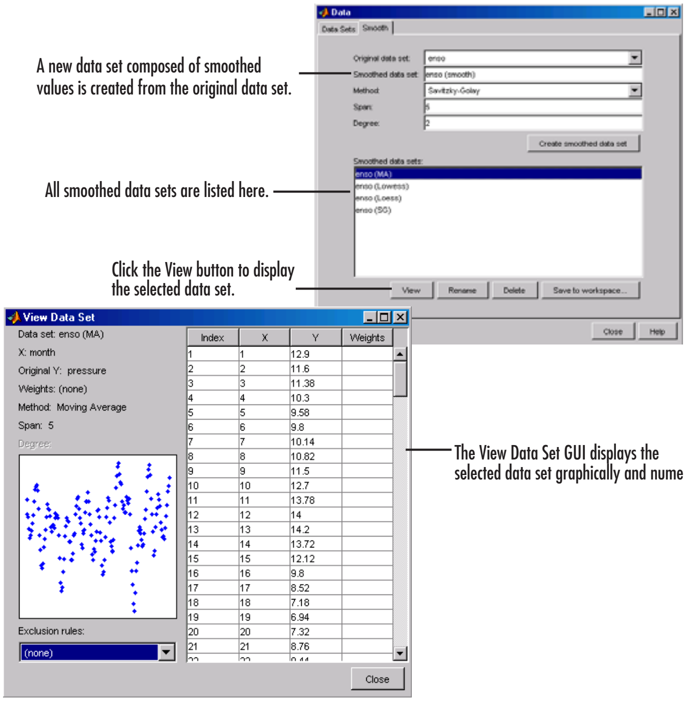
Example: Smoothing Data

This example smooths the ENSO data set using the moving average, lowess, loess, and Savitzky-Golay methods with the default span. As shown below, the data appears noisy. Smoothing might help you visualize patterns in the data, and provide insight toward a reasonable approach for parametric fitting.



Because the data appears noisy, smoothing might help uncover its structure.

The Smooth pane shown below displays all the new data sets generated by smoothing the original ENSO data set. Whenever you smooth a data set, a new data set of smoothed values is created. The smoothed data sets are automatically displayed in Curve Fitting Tool. You can also display a single data set graphically and numerically by clicking the **View** button.



Original data set: enso

Smoothed data set: enso (smooth)

Method: Savitzky-Golay

Span: 5

Degree: 2

Smoothed data sets:

enso (MA)
enso (Lowess)
enso (Loess)
enso (SG)

A new data set composed of smoothed values is created from the original data set.

All smoothed data sets are listed here.

Click the View button to display the selected data set.

Data set: enso (MA)

X: month

Original Y: pressure

Weights: (none)

Method: Moving Average

Span: 5

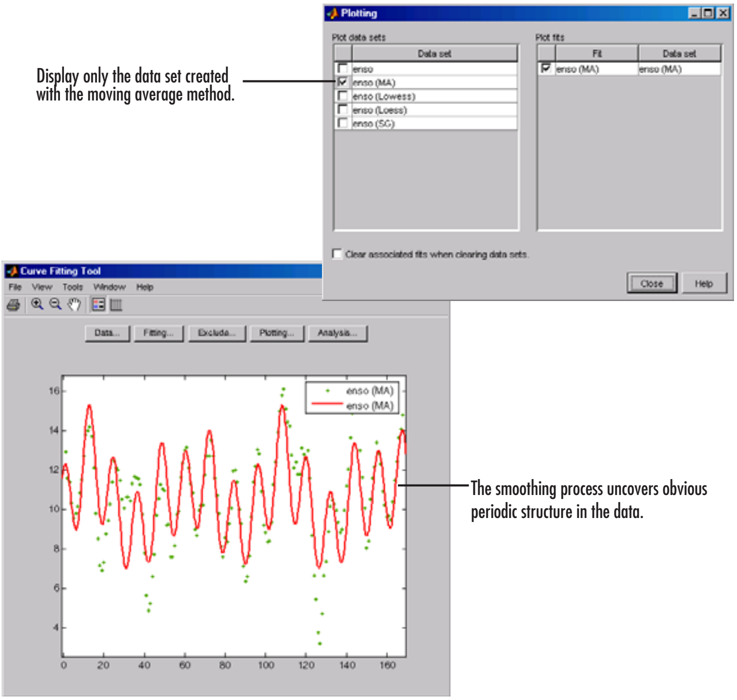
Degree:

Exclusion rules: (none)

Index	X	Y	Weights
1	1	12.9	
2	2	11.6	
3	3	11.38	
4	4	10.3	
5	5	9.58	
6	6	9.8	
7	7	10.14	
8	8	10.82	
9	9	11.5	
10	10	12.7	
11	11	13.78	
12	12	14	
13	13	14.2	
14	14	13.72	
15	15	12.12	
16	16	9.8	
17	17	8.52	
18	18	7.18	
19	19	6.94	
20	20	7.32	
21	21	8.76	

The View Data Set GUI displays the selected data set graphically and numerically.

Use the Plotting GUI to display only the data sets of interest. As shown below, the periodic structure of the ENSO data set becomes apparent when it is smoothed using a moving average filter with the default span. Not surprisingly, the uncovered structure is periodic, which suggests that a reasonable parametric model should include trigonometric functions.



Plotting

Plot data sets

Data set
□ enso
☑ enso (MA)
□ enso (Lowess)
□ enso (Loess)
□ enso (SG)

Plot fits

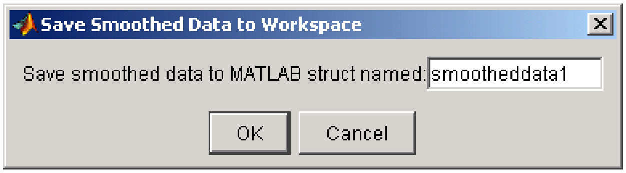
Fit	Data set
☑ enso (MA)	enso (MA)

□ Clear associated fits when clearing data sets.

Display only the data set created with the moving average method.

The smoothing process uncovers obvious periodic structure in the data.

Saving the Results. By clicking the **Save to workspace** button, you can save a smoothed data set as a structure to the MATLAB workspace. This example saves the moving average results contained in the `enso (ma)` data set.



Save Smoothed Data to Workspace

Save smoothed data to MATLAB struct named: smootheddata1

The saved structure contains the original predictor data `x` and the smoothed data `y`.

```
smootheddata1

smootheddata1 =
    x: [168x1 double]
    y: [168x1 double]
```

Excluding and Sectioning Data

Introduction

If there is justification, you might want to exclude part of a data set from a fit. Typically, you exclude data so that subsequent fits are not adversely affected. For example, if you are fitting a parametric model to measured data that has been corrupted by a faulty sensor, the resulting fit coefficients will be inaccurate.

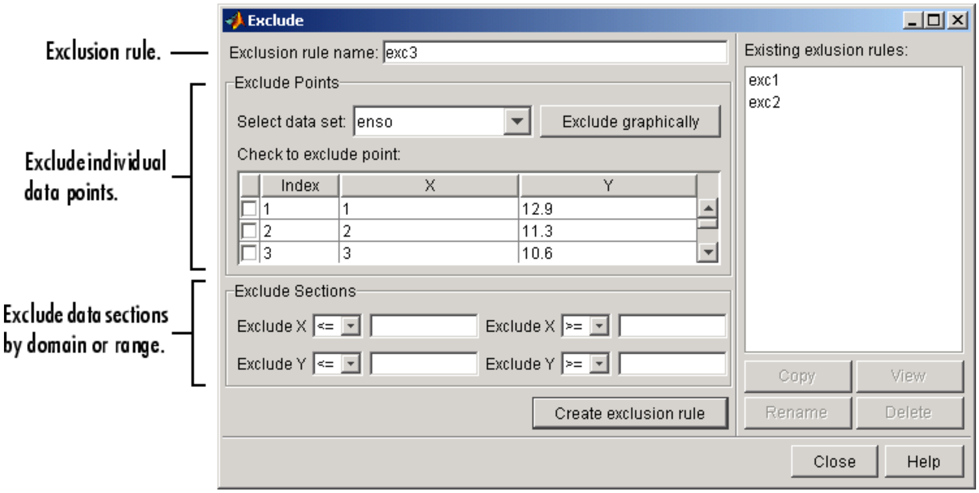
Curve Fitting Toolbox software provides two methods to exclude data:

- Marking Outliers — Outliers are defined as individual data points that you exclude because they are inconsistent with the statistical nature of the bulk of the data.
- Sectioning — Sectioning excludes a window of response or predictor data. For example, if many data points in a data set are corrupted by large systematic errors, you might want to section them out of the fit.

For each of these methods, you must create an *exclusion rule*, which captures the range, domain, or index of the data points to be excluded.

To exclude data while fitting, you use the Fitting GUI to associate the appropriate exclusion rule with the data set to be fit. Refer to "Example: Robust Fitting" on page 2-69 for more information about fitting a data set using an exclusion rule.

You mark data to be excluded from a fit with the Exclude GUI, which you open from Curve Fitting Tool. The GUI is shown below followed by a description of its features.



Exclusion rule name: exc3

Select data set: enso

	Index	X	Y
□	1	1	12.9
□	2	2	11.3
□	3	3	10.6

Existing exlusion rules:
exc1
exc2

Exclusion Rules

- **Exclusion rule name** — Specify the name of the exclusion rule that identifies the data points to be excluded from subsequent fits.
- **Existing exclusion rules** — Lists the names of all exclusion rules created during the current session. When you select an existing exclusion rule, you can perform these actions:
  - Click **Copy** to copy the exclusion rule. The exclusions associated with the original exclusion rule are recreated in the GUI. You can modify

  these exclusions and then click **Create exclusion rule** to save them to the copied rule.

  - Click **Rename** to change the name of the exclusion rule.
  - Click **Delete** to delete the exclusion rule. To select multiple exclusion rules, you can use the **Ctrl** key and the mouse to select exclusion rules one by one, or you can use the **Shift** key and the mouse to select a range of exclusion rules.
  - Click **View** to display the exclusion rule graphically. If a data set is associated with the exclusion rule, the data is also displayed.

Excluding Individual Data Points

- **Select data set** — Select the data set from which data points will be marked as excluded. You must select a data set to exclude individual data points.
- **Exclude graphically** — Open a GUI that allows you to exclude individual data points graphically.

  Individually excluded data points are marked by an "x" in the GUI, and are automatically identified in the **Check to exclude point** table.
- **Check to exclude point** — Select individual data points to exclude. You can sort this table by clicking on any of the column headings.

Excluding Data Sections in the Domain or Range

- **Section** — Specify data to be excluded. You do not need to select a data set to create an exclusion rule by sectioning.
  - **Exclude X** — Specify beginning and ending intervals in the predictor data to be excluded.
  - **Exclude Y** — Specify beginning and ending intervals in the response data to be excluded.

Marking Outliers

Outliers are defined as individual data points that you exclude from a fit because they are inconsistent with the statistical nature of the bulk of the data, and will adversely affect the fit results. Outliers are often readily identified by a scatter plot of response data versus predictor data.

Marking outliers with Curve Fitting Tool follows these rules:

- You must specify a data set before creating an exclusion rule.

  In general, you should use the exclusion rule only with the specific data set it was based on. However, the toolbox does not prevent you from using the exclusion rule with another data set provided the size is the same.

- Using the Exclude GUI, you can exclude outliers either graphically or numerically.

As described in “Parametric Fitting” on page 2-30, one of the basic assumptions underlying curve fitting is that the data is statistical in nature and is described by a particular distribution, which is often assumed to be Gaussian. The statistical nature of the data implies that it contains random variations along with a deterministic component.

data = *deterministic component* + *random component*

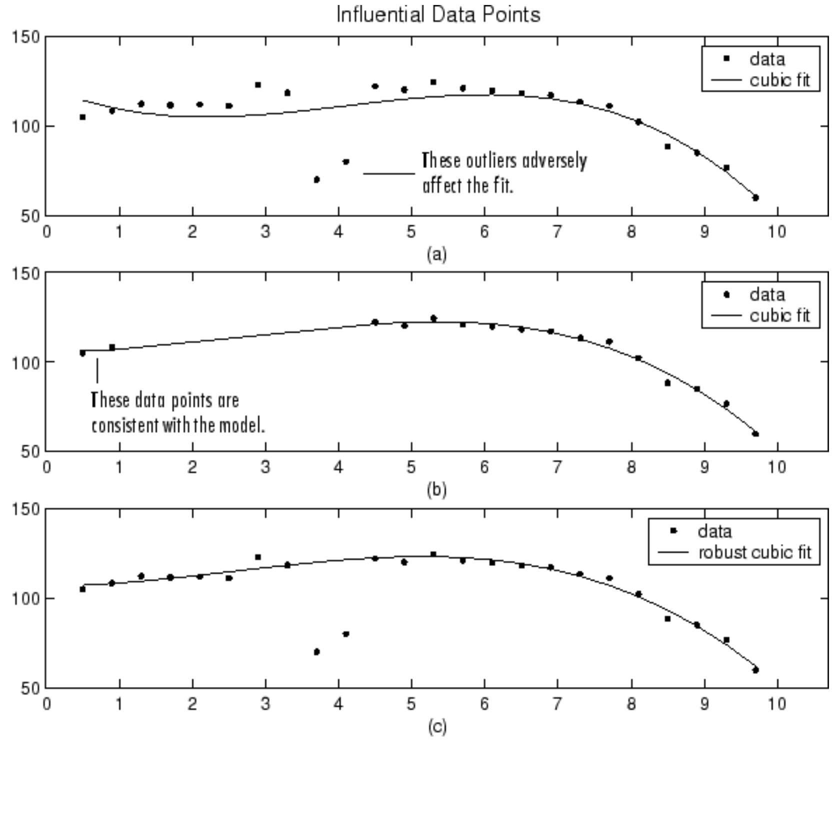
However, your data set might contain one or more data points that are non-statistical in nature, or are described by a different statistical distribution. These data points might be easy to identify, or they might be buried in the data and difficult to identify.

A non-statistical process can involve the measurement of a physical variable such as temperature or voltage in which the random variation is negligible compared to the systematic errors. For example, if your sensor calibration is inaccurate, the data measured with that sensor will be systematically inaccurate. In some cases, you might be able to quantify this non-statistical data component and correct the data accordingly. However, if the scatter plot reveals that a handful of response values are far removed from neighboring response values, these data points are considered outliers and should be excluded from the fit. Outliers are usually difficult to explain away. For example, it might be that your sensor experienced a power surge or someone wrote down the wrong number in a log book.

If you decide there is justification, you should mark outliers to be excluded from subsequent fits—particularly parametric fits. Removing these data points can have a dramatic effect on the fit results because the fitting process minimizes the square of the residuals. If you do not exclude outliers, the resulting fit will be poor for a large portion of your data. Conversely, if you do exclude the outliers and choose the appropriate model, the fit results should be reasonable.

Because outliers can have a significant effect on a fit, they are considered *influential data.* However, not all influential data points are outliers. For example, your data set can contain valid data points that are far removed from the rest of the data. The data is valid because it is well described by the model used in the fit. The data is influential because its exclusion will dramatically affect the fit results.

Two types of influential data points are shown below for generated data. Also shown are cubic polynomial fits and a robust fit that is resistant to outliers.



Plot (a) shows that the two influential data points are outliers and adversely affect the fit. Plot (b) shows that the two influential data points are consistent with the model and do not adversely affect the fit. Plot (c) shows that a robust fitting procedure is an acceptable alternative to marking outliers for exclusion.

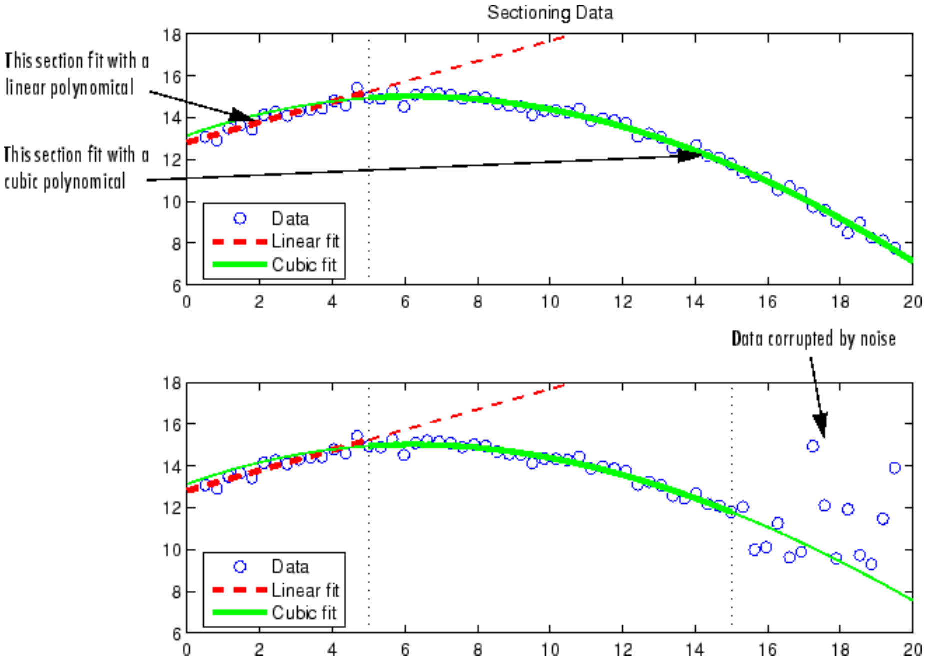
Sectioning

Sectioning involves specifying response or predictor data to exclude. You might want to section a data set because different parts of the data set are described by different models or are corrupted by noise, large systematic errors, and so on.

Sectioning data with Curve Fitting Tool follows these rules:

- If you are only sectioning data and not excluding individual data points, then you can create an exclusion rule without specifying a data set name.
- You can associate an exclusion rule with any data set provided that the exclusion rule overlaps with the data. This is useful if you have multiple data sets from which you want to exclude data points using the same rule.
- Use the Exclude GUI to create the exclusion rule.
- You can exclude vertical strips at the edges of the data, horizontal strips at the edges of the data, or a border around the data. Refer to "Example: Excluding and Sectioning Data" on page 2-24 for an example.
- To exclude multiple sections of data, you can use the `excludedata` function from the MATLAB command line.

Two examples of sectioning by domain are shown below for generated data.



The upper shows the data set sectioned by fit type. The section to the left of 4 is fit with a linear polynomial, as shown by the bold, dashed line. The section to the right of 4 is fit with a cubic polynomial, as shown by the bold, solid line.

The lower plot shows the data set sectioned by fit type and by valid data. Here, the right-most section is not part of any fit because the data is corrupted by noise.

Note For illustrative purposes, the preceding figures have been enhanced to show portions of the curves with bold markers. Curve Fitting Toolbox software does not use bold markers in plots.

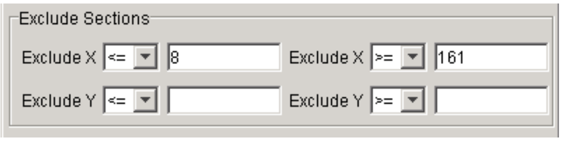
Example: Excluding and Sectioning Data

This example modifies the ENSO data set to illustrate excluding and sectioning data. First, copy the ENSO response data to a new variable and add two outliers that are far removed from the bulk of the data.

```
yy = pressure;
yy(ceil(length(month)*rand(1))) = mean(pressure)*2.5;
yy(ceil(length(month)*rand(1))) = mean(pressure)*3.0;
```

Import the variables `month` and `yy` as the new data set `enso1`, and open the Exclude GUI.

Assume that the first and last eight months of the data set are unreliable, and should be excluded from subsequent fits. The simplest way to exclude these data points is to section the predictor data. To do this, specify the data you want to exclude in the **Exclude Sections** field of the Exclude GUI.

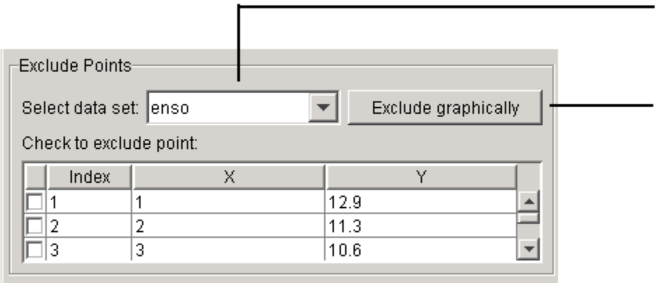


Exclude Sections

Exclude X <= 8 Exclude X >= 161

Exclude Y <= Exclude Y >=

There are two ways to exclude individual data points: using the **Check to exclude point** table or graphically. For this example, the simplest way to exclude the outliers is graphically. To do this, select the data set name and click the **Exclude graphically** button, which opens the Select Points for Exclusion Rule GUI.



Exclude Points

Select data set: enso

Check to exclude point:

	Index	X	Y
□	1	1	12.9
□	2	2	11.3
□	3	3	10.6

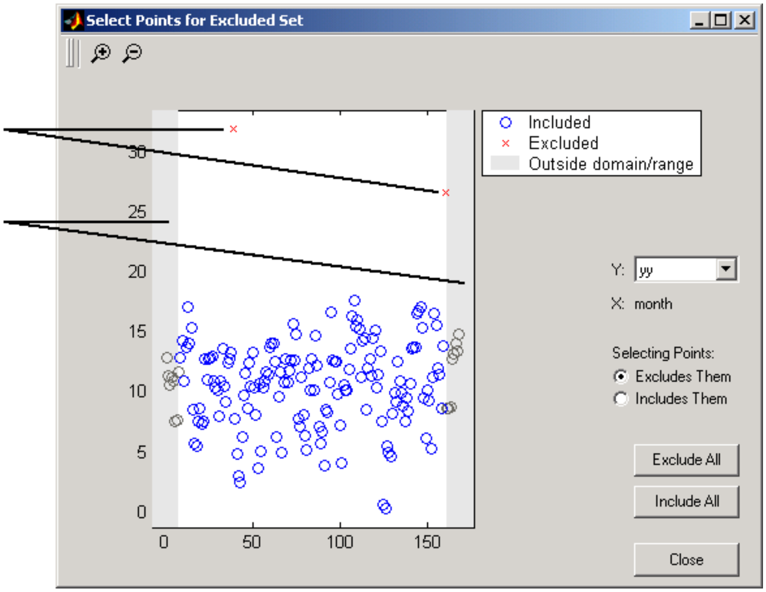
Select the data set.

Open the GUI to exclude data points graphically.

To mark data points for exclusion in the GUI, place the mouse cursor over the data point and left-click. The excluded data point is marked with a red

x. To include an excluded data point, right-click the data point or select the **Includes Them** radio button and left-click. Included data points are marked with a blue circle. To select multiple data points, click the left mouse button and drag the selection rubber band so that the rubber band box encompasses the desired data points. Note that the GUI identifies sectioned data with gray strips. You cannot graphically include sectioned data.

As shown below, the first and last eight months of data are excluded from the data set by sectioning, and the two outliers are excluded graphically. Note that the graphically excluded data points are identified in the **Check to exclude point** table. If you decide to include an excluded data point using the table, the graph is automatically updated.



The x's indicate data points excluded manually.

The vertical gray strips indicate data points sectioned by domain.

Select Points for Excluded Set

Y: yy

X: month

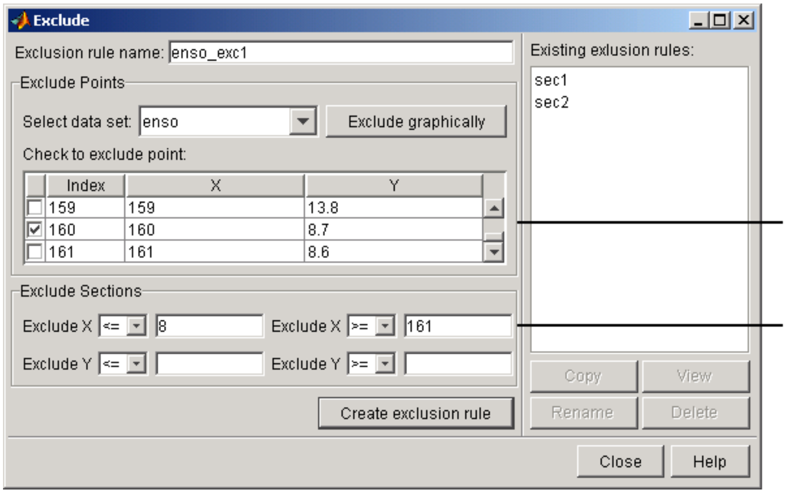
Selecting Points:

● Excludes Them

○ Includes Them

If there are fits associated with the data, you can exclude data points based on the residuals of the fit by selecting the residual data in the **Y** list.

The Exclude GUI for this example is shown below.



Exclusion rule name: enso_exc1

Existing exlusion rules:
sec1
sec2

Select data set: enso

Check to exclude point:

	Index	X	Y
□	159	159	13.8
☑	160	160	8.7
□	161	161	8.6

Exclude X <= 8 Exclude X >= 161

Exclude Y <= Exclude Y >=

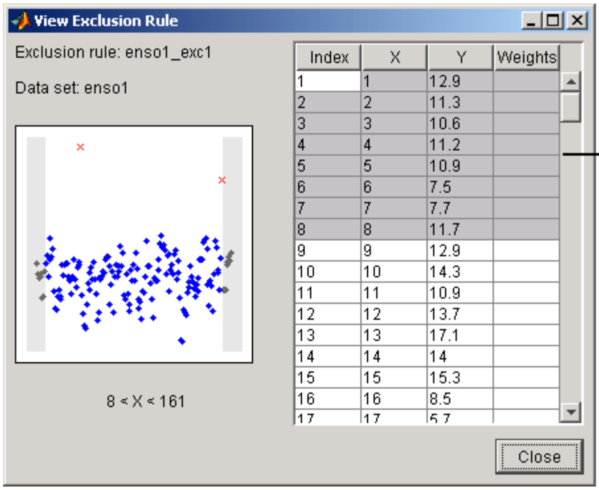
Individual data points marked for exclusion.

Data points outside the specified domain are marked for exclusion.

To save the exclusion rule, click the **Create exclusion rule** button. To exclude the data from a fit, you must select the exclusion rule from the Fitting GUI. Because the exclusion rule created in this example uses individually excluded data points, you can use it only with data sets that are the same size as the ENSO data set.

Viewing the Exclusion Rule. To view the exclusion rule, select an existing exclusion rule name and click the **View** button.

The View Exclusion Rule GUI shown below displays the modified ENSO data set and the excluded data points, which are grayed in the table.



Exclusion rule: enso1_exc1

Data set: enso1

8 < X < 161

Index	X	Y	Weights
1	1	12.9	
2	2	11.3	
3	3	10.6	
4	4	11.2	
5	5	10.9	
6	6	7.5	
7	7	7.7	
8	8	11.7	
9	9	12.9	
10	10	14.3	
11	11	10.9	
12	12	13.7	
13	13	17.1	
14	14	14	
15	15	15.3	
16	16	8.5	
17	17	5.7	

The excluded data points are grayed in the table.

Missing Values and Outliers

Although Curve Fitting Toolbox software ignores `Inf`s and `NaN`s when fitting data, and you can exclude outliers during the fitting process, you might still want to remove this data from your data set. To do so, you modify the associated data set variables from the MATLAB command line.

For example, when using toolbox functions such as `fit` from the command line, you must supply predictor and response vectors that contain finite numbers. To remove `Inf`s, you can use the `isinf` function.

```
ind = find(isinf(xx));
xx(ind) = [];
yy(ind) = [];
```

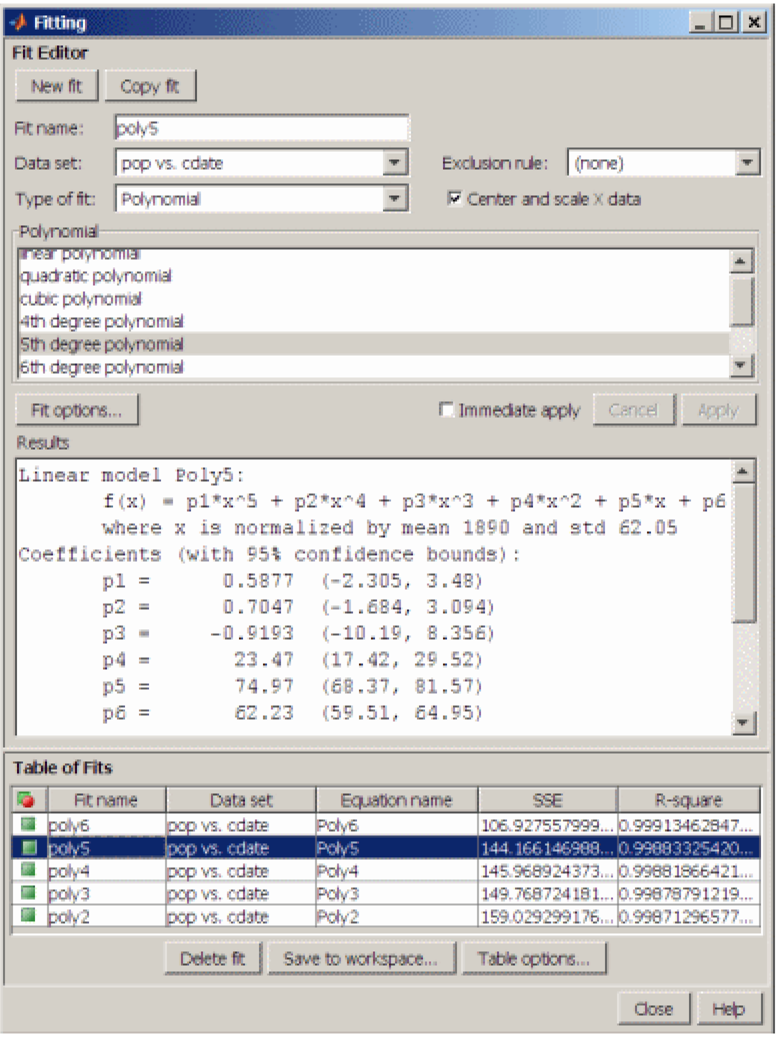
To remove `NaN`s, you can use the `isnan` function. For examples that remove `NaN`s and outliers from a data set, refer to "Missing Data" in the MATLAB documentation.

Fitting Data

The Fitting Process

You fit data using the Fitting GUI. To open the Fitting GUI, click the **Fitting** button from Curve Fitting Tool.

The Fitting GUI is shown below for the census data described in Chapter 1, "Getting Started", followed by the general steps you use when fitting any data set.



Fit name: poly5

Data set: pop vs. cdate

Exclusion rule: (none)

Type of fit: Polynomial

☑ Center and scale X data

Polynomial

- linear polynomial
- quadratic polynomial
- cubic polynomial
- 4th degree polynomial
- 5th degree polynomial
- 6th degree polynomial

□ Immediate apply

Results

```
Linear model Poly5:
       f(x) = p1*x^5 + p2*x^4 + p3*x^3 + p4*x^2 + p5*x + p6
       where x is normalized by mean 1890 and std 62.05
Coefficients (with 95% confidence bounds):
       p1 =      0.5877  (-2.305, 3.48)
       p2 =      0.7047  (-1.684, 3.094)
       p3 =     -0.9193  (-10.19, 8.356)
       p4 =       23.47  (17.42, 29.52)
       p5 =       74.97  (68.37, 81.57)
       p6 =       62.23  (59.51, 64.95)
```

Table of Fits

🟥	Fit name	Data set	Equation name	SSE	R-square
🟩	poly6	pop vs. cdate	Poly6	106.927557999...	0.99913462847...
🟩	poly5	pop vs. cdate	Poly5	144.166146988...	0.99883325420...
🟩	poly4	pop vs. cdate	Poly4	145.968924373...	0.99881866421...
🟩	poly3	pop vs. cdate	Poly3	149.768724181...	0.99878791219...
🟩	poly2	pop vs. cdate	Poly2	159.029299176...	0.99871296577...

1 Select a data set and fit name.

- Select the name of the current fit. When you click **New fit** or **Copy fit**, a default fit name is automatically created in the **Fit name** field. You can specify a new fit name by editing this field.

- Select the name of the current data set from the **Data set** list. All imported and smoothed data sets are listed.

2 Select an exclusion rule.

If you want to exclude data from a fit, select an exclusion rule from the **Exclusion rule** list. The list contains only exclusion rules that are compatible with the current data set. An exclusion rule is compatible with the current data set if their lengths are identical, or if it is created by sectioning only.

3 Select a fit type and fit options, fit the data, and evaluate the goodness of fit.

- The fit type can be a library or custom parametric model, a smoothing spline, or an interpolant.
- Select fit options such as the fitting algorithm, and coefficient starting points and constraints. Depending on your data and model, accepting the default fit options often produces an excellent fit.
- Fit the data by clicking the **Apply** button or by selecting the **Immediate apply** check box.
- Examine the fitted curve, residuals, goodness of fit statistics, confidence bounds, and prediction bounds for the current fit.

4 Compare fits.

- Compare the current fit and data set to previous fits and data sets by examining the goodness of fit statistics.
- Use the Table Options GUI to modify which goodness of fit statistics are displayed in the **Table of Fits**. You can sort the table by clicking on any column heading.

5 Save the fit results.

If the fit is good, save the results as a structure to the MATLAB workspace. Otherwise, modify the fit options or select another model.

Parametric Fitting

- "Introduction" on page 2-31

- “Library Models” on page 2-32
- “Custom Models” on page 2-37
- “Specifying Fit Options” on page 2-59
- “Example: Rational Fit” on page 2-63
- “Example: Robust Fitting” on page 2-69

Introduction

Parametric fitting involves finding coefficients (parameters) for one or more models that you fit to data. The data is assumed to be statistical in nature and is divided into two components: a deterministic component and a random component.

data = *deterministic component* + *random component*

The deterministic component is given by a parametric model and the random component is often described as error associated with the data.

data = *model* + *error*

The model is a function of the independent (predictor) variable and one or more coefficients. The error represents random variations in the data that follow a specific probability distribution (usually Gaussian). The variations can come from many different sources, but are always present at some level when you are dealing with measured data. Systematic variations can also exist, but they can lead to a fitted model that does not represent the data well.

The model coefficients often have physical significance. For example, suppose you have collected data that corresponds to a single decay mode of a radioactive nuclide, and you want to estimate the half-life ($T_{1/2}$) of the decay. The law of radioactive decay states that the activity of a radioactive substance decays exponentially in time. Therefore, the model to use in the fit is given by

$$y = y_0 e^{-\lambda t}$$

where y_0 is the number of nuclei at time $t = 0$, and λ is the decay constant. The data can be described by

$$\text{data} = y_0 e^{-\lambda t} + \text{error}$$

Both y_0 and λ are coefficients that are estimated by the fit. Because $T_{1/2} = \ln(2)/\lambda$, the fitted value of the decay constant yields the fitted half-life. However, because the data contains some error, the deterministic component of the equation cannot be determined exactly from the data. Therefore, the coefficients and half-life calculation will have some uncertainty associated with them. If the uncertainty is acceptable, then you are done fitting the data. If the uncertainty is not acceptable, then you might have to take steps to reduce it either by collecting more data or by reducing measurement error and collecting new data and repeating the model fit.

In other situations where there is no theory to dictate a model, you might also modify the model by adding or removing terms, or substitute an entirely different model.

Library Models

Curve Fitting Toolbox parametric library models are described below.

- “Exponentials” on page 2-32
- “Fourier Series” on page 2-33
- “Gaussian” on page 2-33
- “Polynomials” on page 2-34
- “Power Series” on page 2-35
- “Rationals” on page 2-35
- “Sum of Sines” on page 2-36
- “Weibull Distribution” on page 2-36

Exponentials. The toolbox provides a one-term and a two-term exponential model.

$$y = ae^{bx}$$

$$y = ae^{bx} + ce^{dx}$$

Exponentials are often used when the rate of change of a quantity is proportional to the initial amount of the quantity. If the coefficient associated with *e* is negative, *y* represents exponential decay. If the coefficient is positive, *y* represents exponential growth.

For example, a single radioactive decay mode of a nuclide is described by a one-term exponential. *a* is interpreted as the initial number of nuclei, *b* is the decay constant, *x* is time, and *y* is the number of remaining nuclei after a specific amount of time passes. If two decay modes exist, then you must use the two-term exponential model. For each additional decay mode, you add another exponential term to the model.

Examples of exponential growth include contagious diseases for which a cure is unavailable, and biological populations whose growth is uninhibited by predation, environmental factors, and so on.

Fourier Series. The Fourier series is a sum of sine and cosine functions that is used to describe a periodic signal. It is represented in either the trigonometric form or the exponential form. The toolbox provides the trigonometric Fourier series form shown below,

$$y = a_0 + \sum_{i=1}^{n} a_i \cos(nwx) + b_i \sin(nwx)$$

where a_0 models a constant (intercept) term in the data and is associated with the $i = 0$ cosine term, w is the fundamental frequency of the signal, n is the number of terms (harmonics) in the series, and $1 \le n \le 8$.

For more information about the Fourier series, refer to “Fourier Transforms” in the MATLAB documentation.

Gaussian. The Gaussian model is used for fitting peaks, and is given by the equation

$$y = \sum_{i=1}^{n} a_i e^{\left[-\left(\frac{x-b_i}{c_i}\right)^2\right]}$$

where *a* is the amplitude, *b* is the centroid (location), *c* is related to the peak width, *n* is the number of peaks to fit, and $1 \le n \le 8$.

Gaussian peaks are encountered in many areas of science and engineering. For example, line emission spectra and chemical concentration assays can be described by Gaussian peaks.

Polynomials. Polynomial models are given by

$$y = \sum_{i=1}^{n+1} p_i x^{n+1-i}$$

where *n* + 1 is the *order* of the polynomial, *n* is the *degree* of the polynomial, and $1 \le n \le 9$. The order gives the number of coefficients to be fit, and the degree gives the highest power of the predictor variable.

In this guide, polynomials are described in terms of their degree. For example, a third-degree (cubic) polynomial is given by

$$y = p_1 x^3 + p_2 x^2 + p_3 x + p_4$$

Polynomials are often used when a simple empirical model is required. The model can be used for interpolation or extrapolation, or it can be used to characterize data using a global fit. For example, the temperature-to-voltage conversion for a Type J thermocouple in the 0° to 760° temperature range is described by a seventh-degree polynomial.

Note If you do not require a global parametric fit and want to maximize the flexibility of the fit, piecewise polynomials might provide the best approach. Refer to “Nonparametric Fitting” on page 2-75 for more information.

The main advantages of polynomial fits include reasonable flexibility for data that is not too complicated, and they are linear, which means the fitting process is simple. The main disadvantage is that high-degree fits can become unstable. Additionally, polynomials of any degree can provide a good fit within the data range, but can diverge wildly outside that range. Therefore, you should exercise caution when extrapolating with polynomials. Refer to “Determining the Best Fit” on page 1-10 for examples of good and poor polynomial fits to census data.

Note that when you fit with high-degree polynomials, the fitting procedure uses the predictor values as the basis for a matrix with very large values, which can result in scaling problems. To deal with this, you should normalize the data by centering it at zero mean and scaling it to unit standard deviation. You normalize data by selecting the **Center and scale X data** check box on the Fitting GUI.

Power Series. The toolbox provides a one-term and a two-term power series model.

$$y = ax^b$$

$$y = a + bx^c$$

Power series models are used to describe a variety of data. For example, the rate at which reactants are consumed in a chemical reaction is generally proportional to the concentration of the reactant raised to some power.

Rationals. Rational models are defined as ratios of polynomials and are given by

$$y = \frac{\sum_{i=1}^{n+1} p_i x^{n+1-i}}{x^m + \sum_{i=1}^{m} q_i x^{m-i}}$$

where n is the degree of the numerator polynomial and $0 \le n \le 5$, while *m* is the degree of the denominator polynomial and $1 \le m \le 5$. Note that the coefficient associated with x^m is always 1. This makes the numerator and denominator unique when the polynomial degrees are the same.

In this guide, rationals are described in terms of the degree of the numerator/the degree of the denominator. For example, a quadratic/cubic rational equation is given by

$$y = \frac{p_1x^2 + p_2x + p_3}{x^3 + q_1x^2 + q_2x + q_3}$$

Like polynomials, rationals are often used when a simple empirical model is required. The main advantage of rationals is their flexibility with data that has complicated structure. The main disadvantage is that they become unstable when the denominator is around zero. For an example that uses rational polynomials of various degrees, refer to "Example: Rational Fit" on page 2-63.

Sum of Sines. The sum of sines model is used for fitting periodic functions, and is given by the equation

$$y = \sum_{i=1}^{n} a_i \sin(b_i x + c_i)$$

where *a* is the amplitude, *b* is the frequency, and *c* is the phase constant for each sine wave term. *n* is the number of terms in the series and $1 \le n \le 8$. This equation is closely related to the Fourier series described previously. The main difference is that the sum of sines equation includes the phase constant, and does not include a constant (intercept) term.

Weibull Distribution. The Weibull distribution is widely used in reliability and life (failure rate) data analysis. The toolbox provides the two-parameter Weibull distribution

$$y = abx^{b-1}e^{-ax^b}$$

where *a* is the scale parameter and *b* is the shape parameter. Note that there is also a three-parameter Weibull distribution with *x* replaced by $x - c$ where *c* is the location parameter. Additionally, there is a one-parameter Weibull distribution where the shape parameter is fixed and only the scale parameter is fitted. To use these distributions, you must create a custom equation.

Curve Fitting Toolbox software does not fit Weibull probability distributions to a sample of data. Instead, it fits curves to response and predictor data such that the curve has the same shape as a Weibull distribution.

Custom Models

- "Custom Models vs. Library Models" on page 2-37
- "Creating Custom Models" on page 2-37
- "Editing and Saving Custom Models" on page 2-41
- "Example: Legendre Polynomial" on page 2-43
- "Example: Fourier Series" on page 2-49
- "Example: Gaussian with Exponential Background" on page 2-55

Custom Models vs. Library Models. If the toolbox library does not contain a desired parametric equation, you can create your own custom equation. Library models, however, offer the best chance for rapid convergence. This is because:

- For most library models, optimal default coefficient starting points are calculated. For custom models, the default starting points are chosen at random on the interval [0,1].
- Library models use an analytic Jacobian; custom models use finite differencing.
- When using the Analysis GUI, library models use analytic derivatives and integrals if the integral can be expressed in closed form; custom models use numerical approximations.

Creating Custom Models. Create custom equations with the New Custom Equation GUI. Open the GUI in one of two ways:

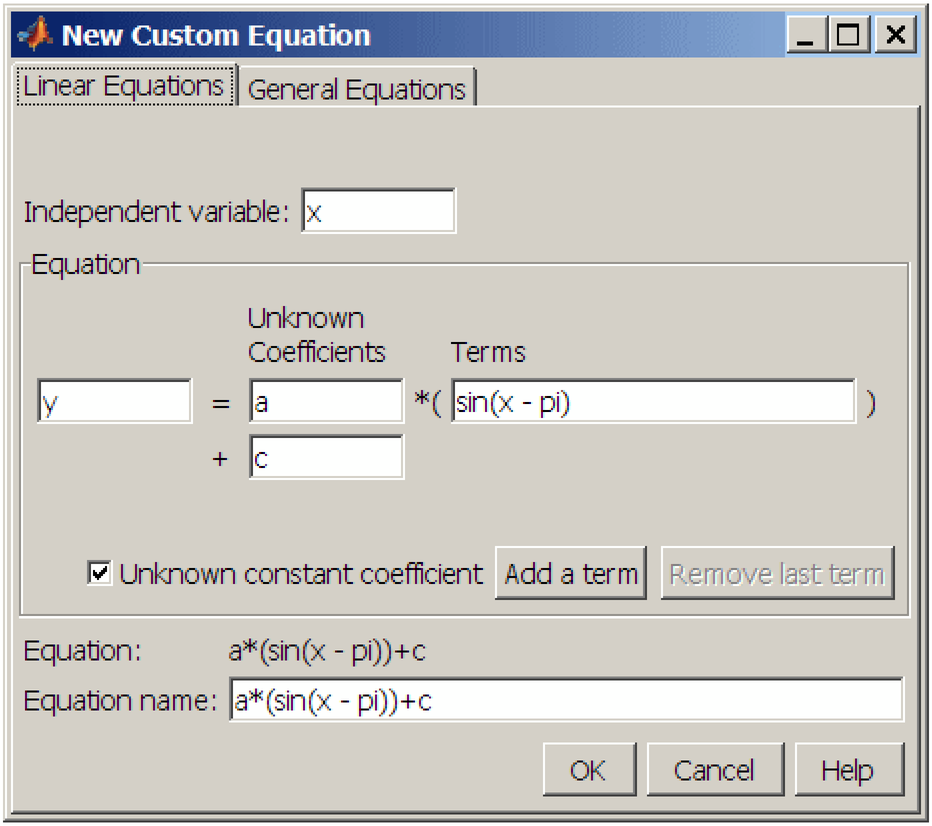
- From Curve Fitting Tool, select **Tools > Custom Equation**.

- From the Fitting GUI, select **Custom Equations** from the **Type of fit** list, then click the **New** button.

The GUI contains two panes: one for creating linear custom equations and one for creating general (nonlinear) custom equations.

Linear Equations

Linear models are linear combinations of (perhaps nonlinear) terms. They are defined by equations that are linear in the parameters. Use the Linear Equations pane on the New Custom Equation GUI to create custom linear equations. Interface controls are described below.



Independent variable: x

y = a *(sin(x - pi))

+ c

☑ Unknown constant coefficient

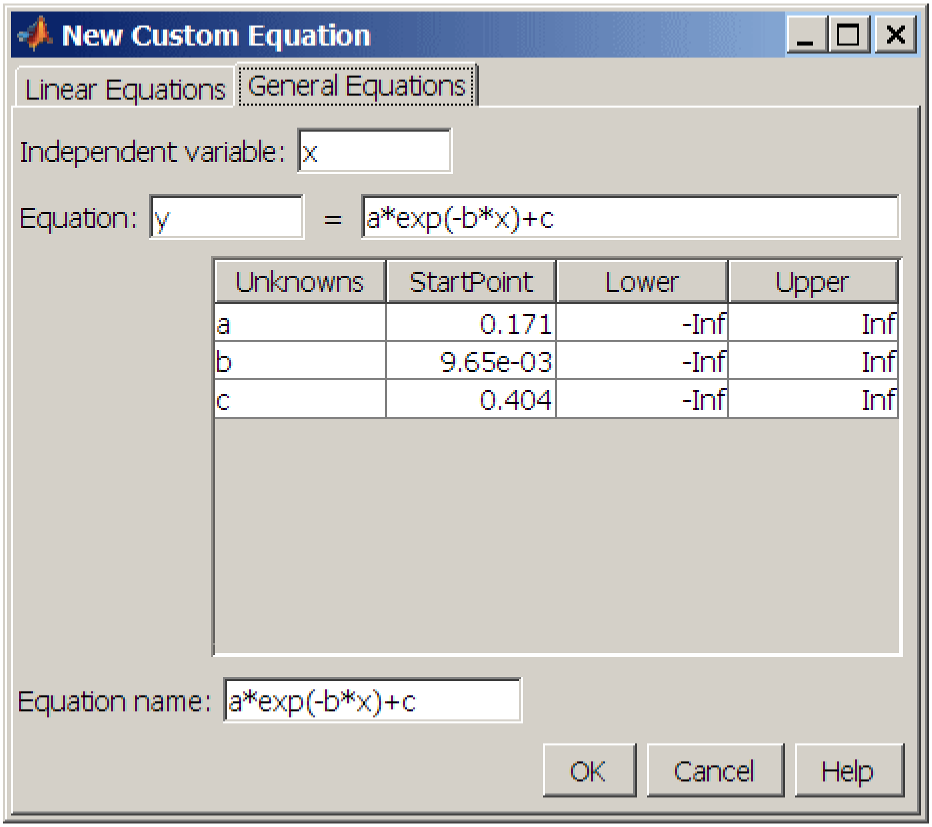
Equation: a*(sin(x - pi))+c

Equation name: a*(sin(x - pi))+c

- **Independent variable** — Symbol representing the independent (predictor) variable. The default symbol is `x`.
- **Equation** — Symbol representing the dependent (response) variable, followed by the linear equation. The default symbol is `y`.
  - **Unknown Coefficients** — The unknown coefficients to be determined by the fit. The default symbols are `a`, `b`, `c`, and so on.
  - **Terms** — Functions of the independent variable. These may be nonlinear. Terms may not contain a coefficient to be fitted.
  - **Unknown constant coefficient** — If selected, a constant term (*y*-intercept) is included in the equation. Otherwise, a constant term is not included.
  - **Add a term** — Add a term to the equation. An unknown coefficient is automatically added for each new term.
  - **Remove last term** — Remove the last term added to the equation.
- **Equation name** — The name of the equation. By default, the name is automatically updated to be identical to the custom equation given by **Equation**. If you override the default, the name is no longer automatically updated.

General Equations

General models are, in general, nonlinear combinations of (perhaps nonlinear) terms. They are defined by equations that may be nonlinear in the parameters. Use the General Equations pane on the New Custom Equation GUI to create custom general equations. Interface controls are described below.



Independent variable: x

Equation: y = a*exp(-b*x)+c

Unknowns	StartPoint	Lower	Upper
a	0.171	-Inf	Inf
b	9.65e-03	-Inf	Inf
c	0.404	-Inf	Inf

Equation name: a*exp(-b*x)+c

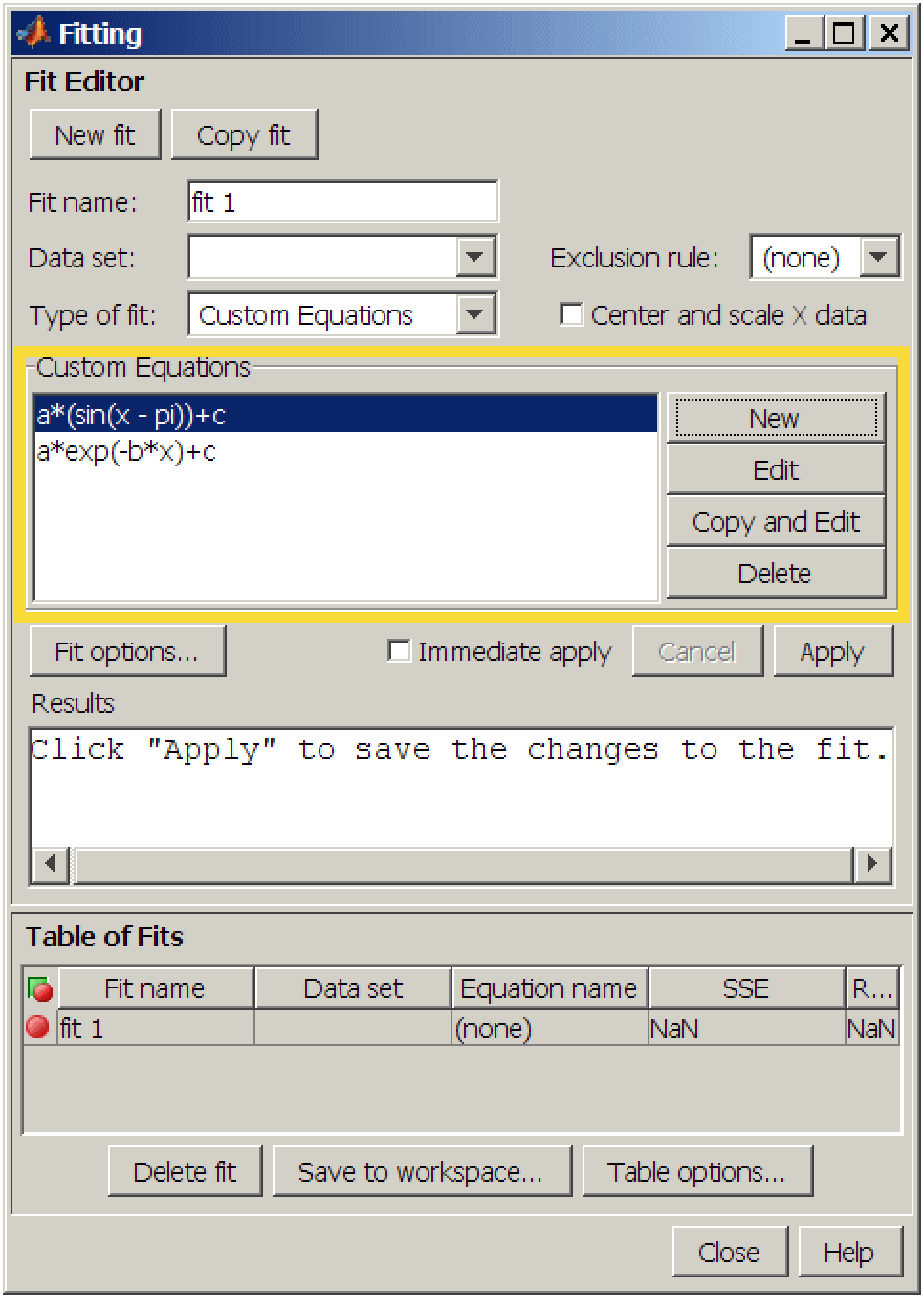
- **Independent variable** — Symbol representing the independent (predictor) variable. The default symbol is x.
- **Equation** — Symbol representing the dependent (response) variable, followed by the general equation. The default symbol is y. As you type in the terms of the equation, the unknown coefficients, associated starting values, and constraints automatically populate the table. By default, the starting values are randomly selected on the interval [0,1] and are unconstrained.

  You can immediately change the default starting values and constraints in this table, or you can change them later using the Fit Options GUI.
- **Equation name** — The name of the equation. By default, the name is automatically updated to be identical to the custom equation given by

Equation. If you override the default, the name is no longer automatically updated.

Note If you use the General Equations pane to define a linear equation, a nonlinear fitting procedure is used. While this is allowed, it is inefficient, and can result in less than optimal fitted coefficients. Use the **Linear Equations** pane to define custom linear equations.

Editing and Saving Custom Models. When you click **OK** on the New Custom Equation GUI, the displayed **Equation name** is saved for the current session in the **Custom Equations** list on the Fitting GUI. The list is highlighted in the picture of the Fitting GUI below.



Fit name: fit 1

Data set:

Exclusion rule: (none)

Type of fit: Custom Equations

☐ Center and scale X data

Custom Equations

- a*(sin(x - pi))+c
- a*exp(-b*x)+c

☐ Immediate apply

```
Click "Apply" to save the changes to the fit.
```

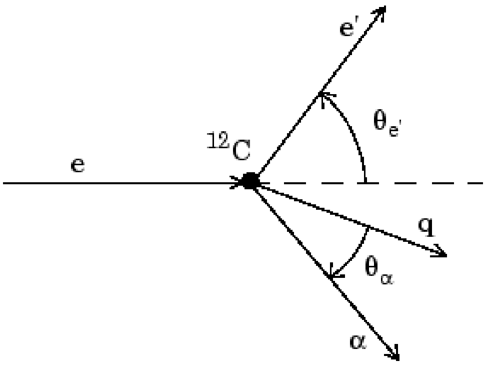
	Fit name	Data set	Equation name	SSE	R...
🔴	fit 1		(none)	NaN	NaN

To edit a custom equation, select the equation in the **Custom Equations** list and click the **Edit** button. The Edit Custom Equation GUI appears. The Edit Custom Equation GUI is identical to the New Custom Equation GUI, but is pre-populated with the selected equation. After editing an equation in the Edit Custom Equation GUI, click **OK** to save it back to the **Custom Equations** list for further use in the current session. A button to **Copy and Edit** is also available, if you want to save both the original and edited equations for the current session.

To save custom equations for future sessions, select the **File > Save Session** menu item in Curve Fitting Tool.

Example: Legendre Polynomial. This example fits data using several custom linear equations. The data is generated, and is based on the nuclear reaction ^{12}C(e,e'α)^{8}Be. The equations use sums of Legendre polynomial terms.

Consider an experiment in which 124 MeV electrons are scattered from ^{12}C nuclei. In the subsequent reaction, alpha particles are emitted and produce the residual nuclei ^{8}Be. By analyzing the number of alpha particles emitted as a function of angle, you can deduce certain information regarding the nuclear dynamics of ^{12}C. The reaction kinematics are shown below.



e is the incident electron.
^{12}C is the carbon target.
q is the momentum transferred to ^{8}Be.
e' is the scattered electron.
α is the emitted alpha particle.
$\theta_{e'}$ is the electron scattering angle.
θ_α is the alpha scattering angle.

The data is collected by placing solid state detectors at values of Θ_α ranging from 10° to 240° in 10° increments.

It is sometimes useful to describe a variable expressed as a function of angle in terms of Legendre polynomials

$$y(x) = \sum_{n=0}^{\infty} a_n P_n(x)$$

where $P_n(x)$ is a Legendre polynomial of degree n, x is $\cos(\Theta_\alpha)$, and a_n are the coefficients of the fit. Refer to the `legendre` function for information about generating Legendre polynomials.

For the alpha-emission data, you can directly associate the coefficients with the nuclear dynamics by invoking a theoretical model. Additionally, the theoretical model introduces constraints for the infinite sum shown above. In particular, by considering the angular momentum of the reaction, a fourth-degree Legendre polynomial using only even terms should describe the data effectively.

You can generate Legendre polynomials with Rodrigues' formula:

$$P_n(x) = \frac{1}{2^n n!}\left(\frac{d}{dx}\right)^n (x^2-1)^n$$

The Legendre polynomials up to fourth degree are given below.

Legendre Polynomials up to Fourth Degree

n	$P_n(x)$
0	1
1	x
2	$(1/2)(3x^2 - 1)$
3	$(1/2)(5x^3 - 3x)$
4	$(1/8)(35x^4 - 30x^2 + 3)$

The first step is to load the ^{12}C alpha-emission data from the file `carbon12alpha.mat`, which is provided with the toolbox.

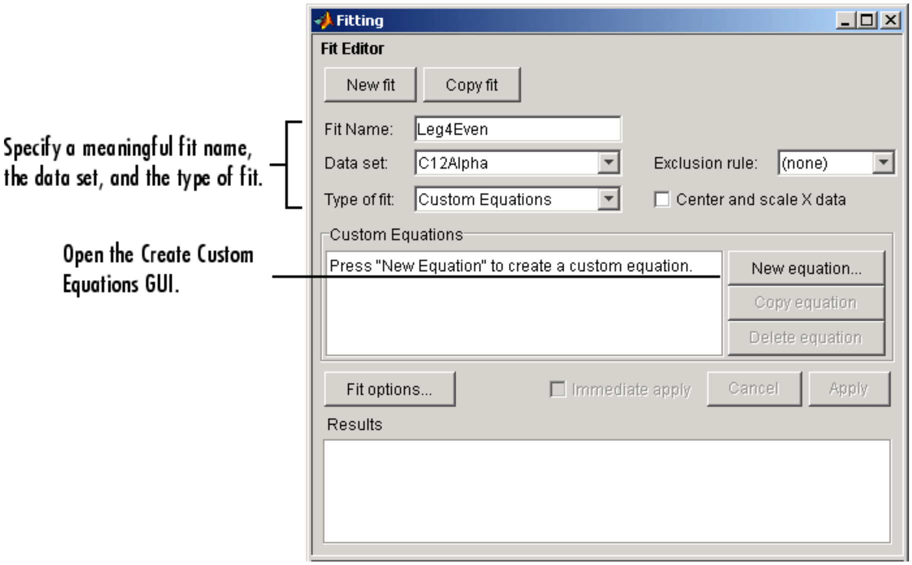
```
load carbon12alpha
```

The workspace now contains two new variables, `angle` and `counts`:

- `angle` is a vector of angles (in radians) ranging from 10º to 240º in 10º increments.
- `counts` is a vector of raw alpha particle counts that correspond to the emission angles in `angle`.

Import these two variables into Curve Fitting Tool and name the data set `C12Alpha`.

The **Fit Editor** for a custom equation fit type is shown below.



Specify a meaningful fit name, the data set, and the type of fit.

Open the Create Custom Equations GUI.

Fitting

Fit Editor

New fit | Copy fit

Fit Name: Leg4Even

Data set: C12Alpha

Exclusion rule: (none)

Type of fit: Custom Equations

□ Center and scale X data

Custom Equations

Press "New Equation" to create a custom equation.

New equation... | Copy equation | Delete equation

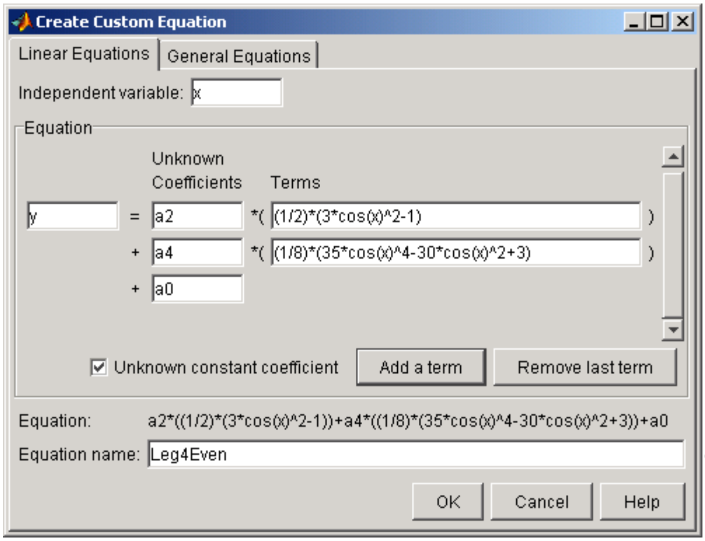
Fit options... | □ Immediate apply | Cancel | Apply

Results

Fit the data using a fourth-degree Legendre polynomial with only even terms:

$$y_1(x) = a_0 + a_2\left(\frac{1}{2}\right)(3x^2 - 1) + a_4\left(\frac{1}{8}\right)(35x^4 - 30x^2 + 3)$$

Because the Legendre polynomials depend only on the predictor variable and constants, you use the Linear Equations pane on the Create Custom Equation GUI. This pane is shown below for the model given by $y_1(x)$. Note that because `angle` is given in radians, the argument of the Legendre terms is given by cos(Θ_a).



Independent variable: x

y = a2 *((1/2)*(3*cos(x)^2-1))
+ a4 *((1/8)*(35*cos(x)^4-30*cos(x)^2+3))
+ a0

☑ Unknown constant coefficient

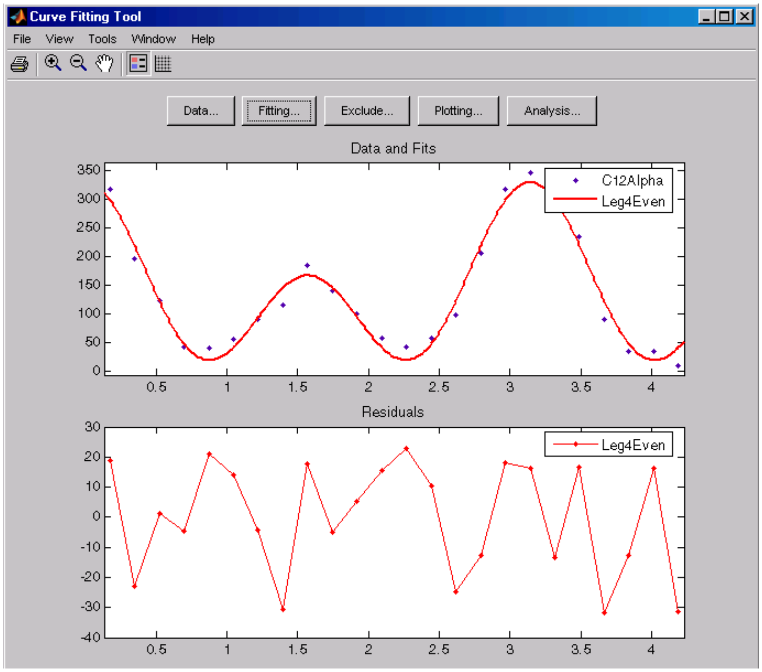
Equation: a2*((1/2)*(3*cos(x)^2-1))+a4*((1/8)*(35*cos(x)^4-30*cos(x)^2+3))+a0

Equation name: Leg4Even

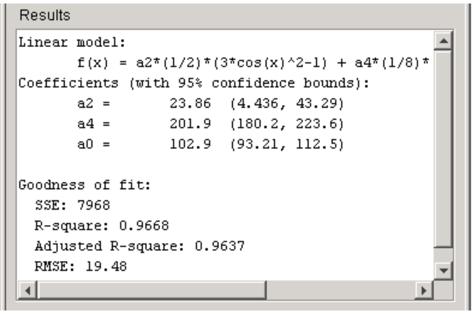
Create a custom linear equation using even Legendre terms up to fourth degree.

Specify a meaningful equation name.

The fit and residuals are shown below. The fit appears to follow the trend of the data well, while the residuals appear to be randomly distributed and do not exhibit any systematic behavior.



The numerical fit results are shown below. The 95% confidence bounds indicate that the coefficients associated with $P_0(x)$ and $P_4(x)$ are known fairly accurately, but that the $P_2(x)$ coefficient has a relatively large uncertainty.



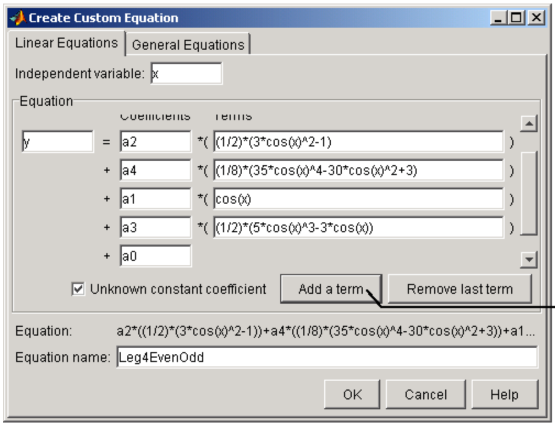
```
Linear model:
       f(x) = a2*(1/2)*(3*cos(x)^2-1) + a4*(1/8)*
Coefficients (with 95% confidence bounds):
       a2 =       23.86  (4.436, 43.29)
       a4 =       201.9  (180.2, 223.6)
       a0 =       102.9  (93.21, 112.5)

Goodness of fit:
  SSE: 7968
  R-square: 0.9668
  Adjusted R-square: 0.9637
  RMSE: 19.48
```

The coefficients associated with $P_0(x)$ and $P_4(x)$ are known accurately, but the $P_2(x)$ coefficient has a larger uncertainty.

To confirm the theoretical argument that the alpha-emission data is best described by a fourth-degree Legendre polynomial with only even terms, fit the data using both even and odd terms:

$$y_2(x) = y_1(x) + a_1x + a_3\left(\frac{1}{2}\right)(5x^3 - 3x)$$

The Linear Equations pane of the Create Custom Equation GUI is shown below for the model given by $y_2(x)$.



Independent variable: x

Coefficients Terms

y = a2 *((1/2)*(3*cos(x)^2-1))
+ a4 *((1/8)*(35*cos(x)^4-30*cos(x)^2+3))
+ a1 *(cos(x))
+ a3 *((1/2)*(5*cos(x)^3-3*cos(x)))
+ a0

☑ Unknown constant coefficient

Equation: a2*((1/2)*(3*cos(x)^2-1))+a4*((1/8)*(35*cos(x)^4-30*cos(x)^2+3))+a1...

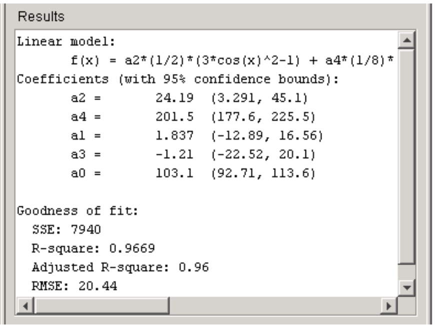
Equation name: Leg4EvenOdd

Create a custom linear equation using even and odd Legendre terms up to fourth degree.

Click Add a term to add the odd Legendre terms.

Specify a meaningful equation name.

The numerical results indicate that the odd Legendre terms do not contribute significantly to the fit, and the even Legendre terms are essentially unchanged from the previous fit. This confirms that the initial model choice is the best one.



```
Linear model:
       f(x) = a2*(1/2)*(3*cos(x)^2-1) + a4*(1/8)*
Coefficients (with 95% confidence bounds):
       a2 =       24.19  (3.291, 45.1)
       a4 =       201.5  (177.6, 225.5)
       a1 =       1.837  (-12.89, 16.56)
       a3 =       -1.21  (-22.52, 20.1)
       a0 =       103.1  (92.71, 113.6)

Goodness of fit:
  SSE: 7940
  R-square: 0.9669
  Adjusted R-square: 0.96
  RMSE: 20.44
```

The odd Legendre coefficients are likely candidates for removal to simplify the fit because their values are small and their confidence bounds contain zero.

Example: Fourier Series. This example fits the ENSO data using several custom nonlinear equations. The ENSO data consists of monthly averaged atmospheric pressure differences between Easter Island and Darwin, Australia. This difference drives the trade winds in the southern hemisphere.

As shown in “Example: Smoothing Data” on page 2-13, the ENSO data is clearly periodic, which suggests it can be described by a Fourier series

$$y(x) = a_0 + \sum_{i=1}^{\infty} a_i \cos\left(2\pi\frac{x}{c_i}\right) + b_i \sin\left(2\pi\frac{x}{c_i}\right)$$

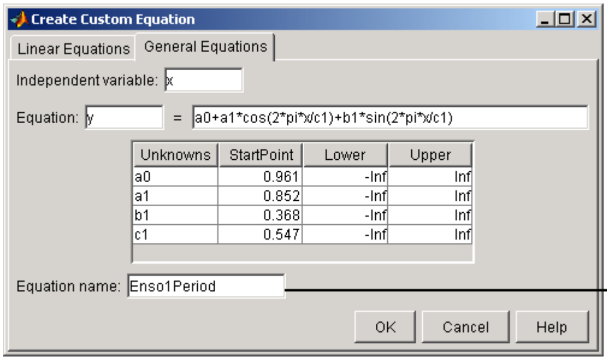
where a_i and b_i are the amplitudes, and c_i are the periods (cycles) of the data. The question to be answered in this example is how many cycles exist? As a first attempt, assume a single cycle and fit the data using one sine term and one cosine term.

$$y_1(x) = a_0 + a_1 \cos\left(2\pi\frac{x}{c_1}\right) + b_1 \sin\left(2\pi\frac{x}{c_1}\right)$$

If the fit does not describe the data well, add additional sine and cosine terms with unique period coefficients until a good fit is obtained.

Because there is an unknown coefficient c_1 included as part of the trigonometric function arguments, the equation is nonlinear. Therefore, you must specify the equation using the General Equations pane of the Create Custom Equation GUI.

This pane is shown below for the equation given by $y_1(x)$.



Independent variable: x

Equation: y = a0+a1*cos(2*pi*x/c1)+b1*sin(2*pi*x/c1)

Unknowns	StartPoint	Lower	Upper
a0	0.961	-Inf	Inf
a1	0.852	-Inf	Inf
b1	0.368	-Inf	Inf
c1	0.547	-Inf	Inf

Equation name: Enso1Period

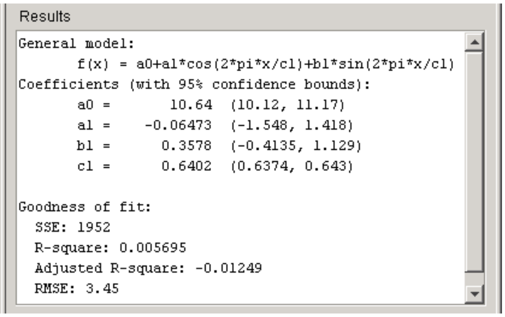
Begin with a single cycle.

By default, the coefficients are unbounded and have random starting values between 0 and 1.

Specify a meaningful equation name.

Note that the toolbox includes the Fourier series as a nonlinear library equation. However, the library equation does not meet the needs of this example because its terms are defined as fixed multiples of the fundamental frequency *w*. Refer to "Fourier Series" on page 2-33 for more information.

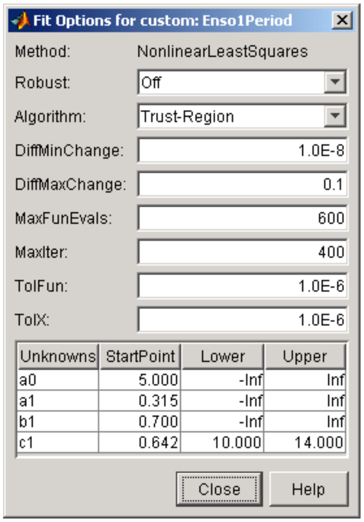
The numerical results shown below indicate that the fit does not describe the data well. In particular, the fitted value for `c1` is unreasonably small. Because the starting points are randomly selected, your initial fit results might differ from the results shown here.



```
General model:
       f(x) = a0+a1*cos(2*pi*x/c1)+b1*sin(2*pi*x/c1)
Coefficients (with 95% confidence bounds):
       a0 =       10.64  (10.12, 11.17)
       a1 =    -0.06473  (-1.548, 1.418)
       b1 =      0.3578  (-0.4135, 1.129)
       c1 =      0.6402  (0.6374, 0.643)

Goodness of fit:
  SSE: 1952
  R-square: 0.005695
  Adjusted R-square: -0.01249
  RMSE: 3.45
```

As you saw in "Example: Smoothing Data" on page 2-13, the data include a periodic component with a period of about 12 months. However, with `c1` unconstrained and with a random starting point, this fit failed to find that cycle. To assist the fitting procedure, constrain `c1` to a value between 10 and

14. To define constraints for unknown coefficients, use the Fit Options GUI, which you open by clicking the **Fit options** button in the Fitting GUI.



Fit Options for custom: Enso1Period

Method: NonlinearLeastSquares

Robust: Off

Algorithm: Trust-Region

DiffMinChange: 1.0E-8

DiffMaxChange: 0.1

MaxFunEvals: 600

MaxIter: 400

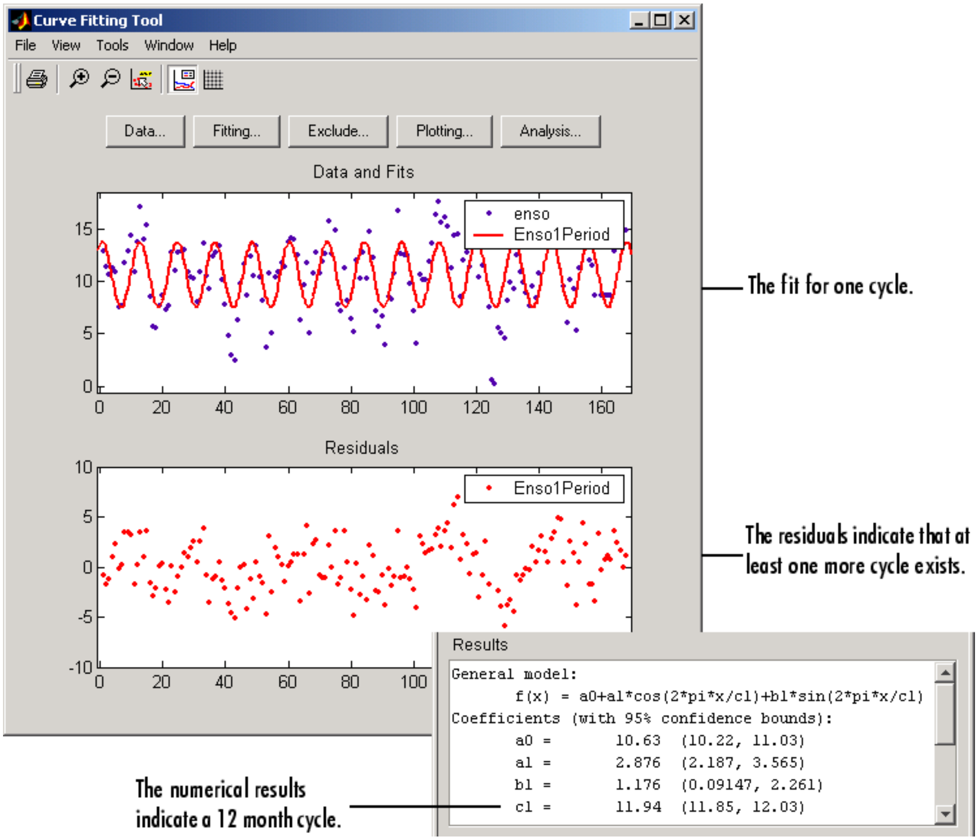
TolFun: 1.0E-6

TolX: 1.0E-6

Unknowns	StartPoint	Lower	Upper
a0	5.000	-Inf	Inf
a1	0.315	-Inf	Inf
b1	0.700	-Inf	Inf
c1	0.642	10.000	14.000

Constrain the cycle to be between 10 and 14 months.

The fit, residuals, and numerical results are shown below.



```
General model:
     f(x) = a0+a1*cos(2*pi*x/c1)+b1*sin(2*pi*x/c1)
Coefficients (with 95% confidence bounds):
     a0 =       10.63  (10.22, 11.03)
     a1 =       2.876  (2.187, 3.565)
     b1 =       1.176  (0.09147, 2.261)
     c1 =       11.94  (11.85, 12.03)
```

The fit for one cycle.

The residuals indicate that at least one more cycle exists.

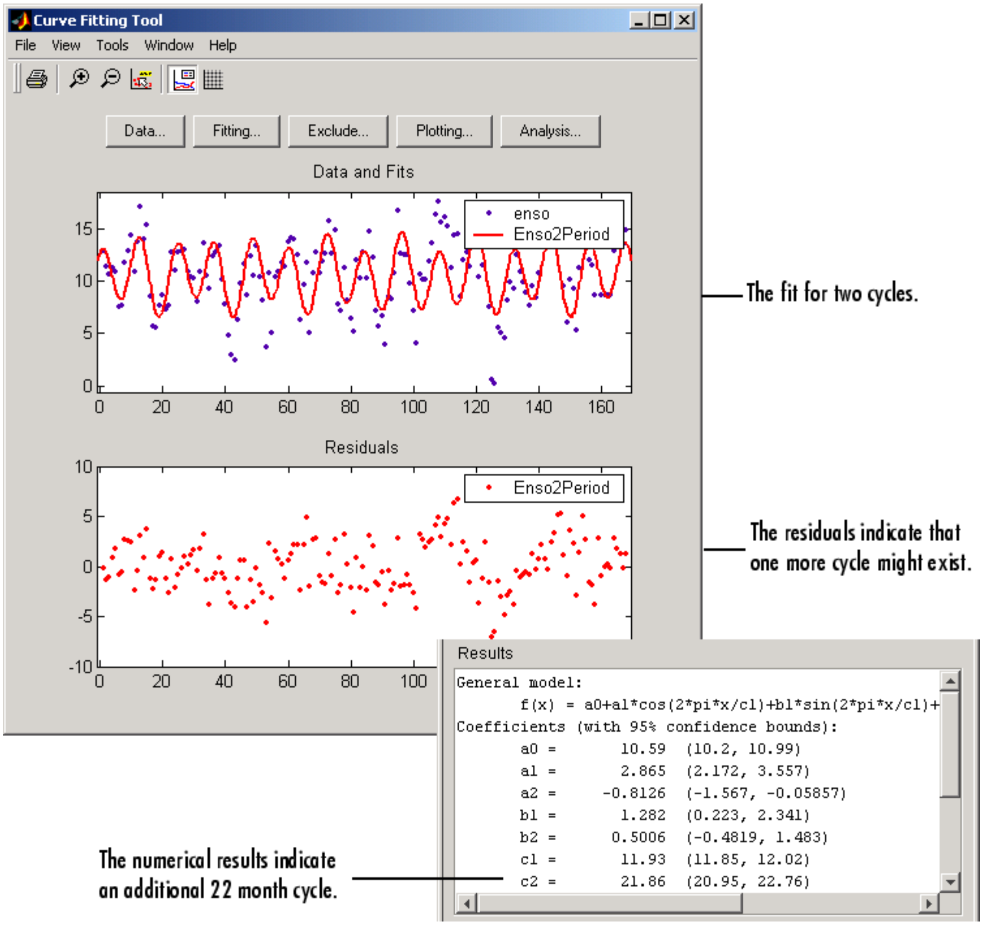
The numerical results indicate a 12 month cycle.

The fit appears to be reasonable for some of the data points but clearly does not describe the entire data set very well. As predicted, the numerical results indicate a cycle of approximately 12 months. However, the residuals show a systematic periodic distribution indicating that there are additional cycles that you should include in the fit equation. Therefore, as a second attempt, add an additional sine and cosine term to $y_1(x)$

$$y_2(x) = y_1(x) + a_2\cos\left(2\pi\frac{x}{c_2}\right) + b_2\sin\left(2\pi\frac{x}{c_2}\right)$$

and constrain the upper and lower bounds of c_2 to be roughly twice the bounds used for c_1.

The fit, residuals, and numerical results are shown below.



```
General model:
       f(x) = a0+a1*cos(2*pi*x/c1)+b1*sin(2*pi*x/c1)+
Coefficients (with 95% confidence bounds):
       a0 =       10.59  (10.2, 10.99)
       a1 =       2.865  (2.172, 3.557)
       a2 =     -0.8126  (-1.567, -0.05857)
       b1 =       1.282  (0.223, 2.341)
       b2 =      0.5006  (-0.4819, 1.483)
       c1 =       11.93  (11.85, 12.02)
       c2 =       21.86  (20.95, 22.76)
```

The fit for two cycles.

The residuals indicate that one more cycle might exist.

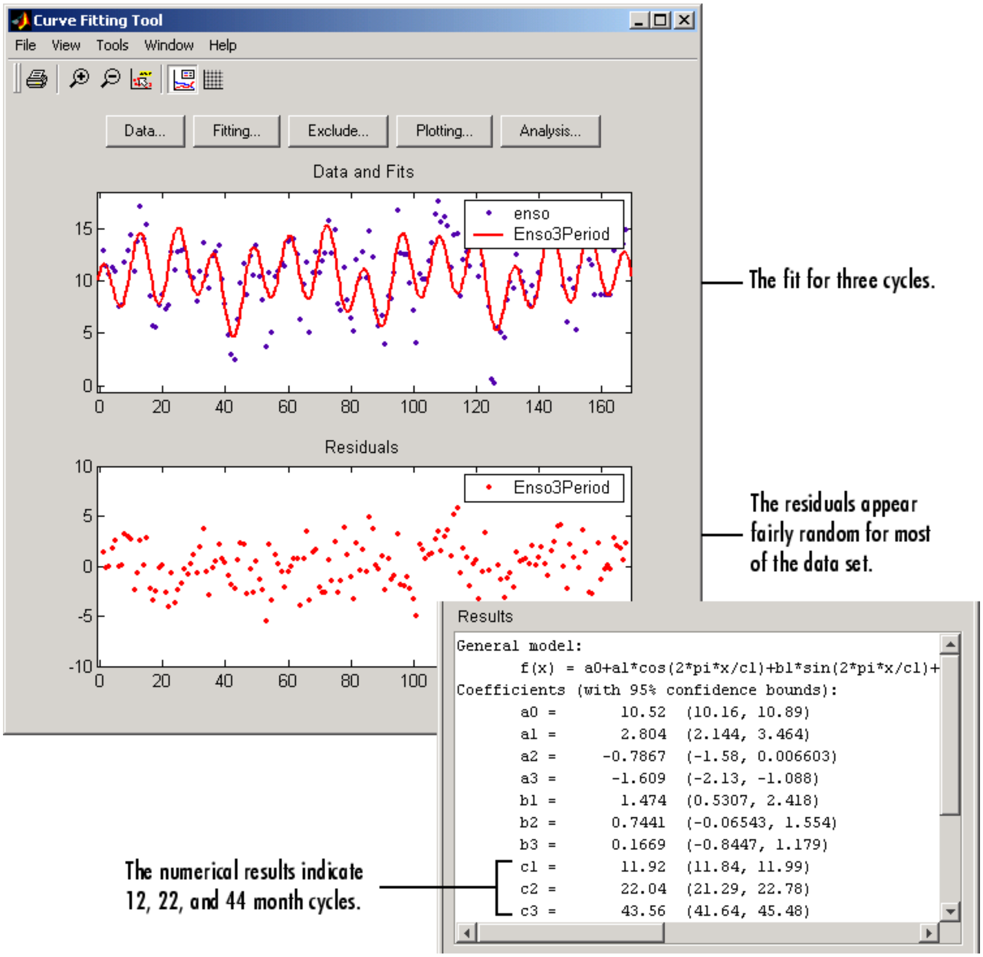
The numerical results indicate an additional 22 month cycle.

The fit appears to be reasonable for most of the data points. However, the residuals indicate that you should include another cycle to the fit equation. Therefore, as a third attempt, add an additional sine and cosine term to $y_2(x)$

$$y_3(x) = y_2(x) + a_3\cos\left(2\pi\frac{x}{c_3}\right) + b_3\sin\left(2\pi\frac{x}{c_3}\right)$$

and constrain the lower bound of c_3 to be roughly three times the value of c_1.

The fit, residuals, and numerical results are shown below.



```
General model:
       f(x) = a0+a1*cos(2*pi*x/c1)+b1*sin(2*pi*x/c1)+
Coefficients (with 95% confidence bounds):
       a0 =       10.52  (10.16, 10.89)
       a1 =       2.804  (2.144, 3.464)
       a2 =     -0.7867  (-1.58, 0.006603)
       a3 =      -1.609  (-2.13, -1.088)
       b1 =       1.474  (0.5307, 2.418)
       b2 =      0.7441  (-0.06543, 1.554)
       b3 =      0.1669  (-0.8447, 1.179)
       c1 =       11.92  (11.84, 11.99)
       c2 =       22.04  (21.29, 22.78)
       c3 =       43.56  (41.64, 45.48)
```

The fit for three cycles.

The residuals appear fairly random for most of the data set.

The numerical results indicate 12, 22, and 44 month cycles.

The fit is an improvement over the previous two fits, and appears to account for most of the cycles present in the ENSO data set. The residuals appear random for most of the data, although a pattern is still visible indicating that additional cycles may be present, or you can improve the fitted amplitudes.

In conclusion, Fourier analysis of the data reveals three significant cycles. The annual cycle is the strongest, but cycles with periods of approximately 44 and 22 months are also present. These cycles correspond to El Nino and the Southern Oscillation (ENSO).

Example: Gaussian with Exponential Background. This example fits two poorly resolved Gaussian peaks on a decaying exponential background using a general (nonlinear) custom model. To get started, load the data from the file `gauss3.mat`, which is provided with the toolbox.

```
load gauss3
```

The workspace now contains two new variables, `xpeak` and `ypeak`:

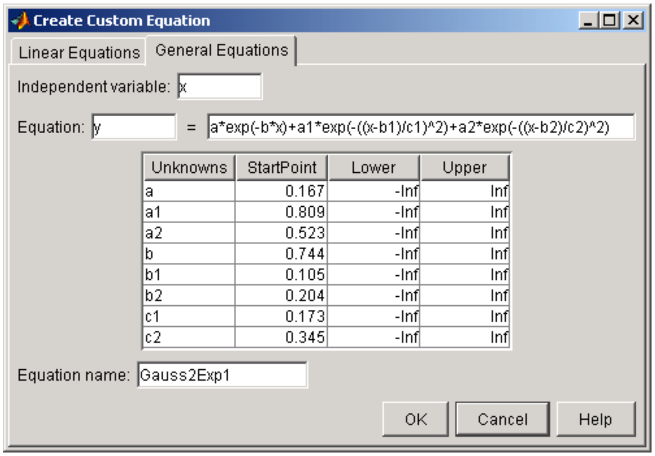
- `xpeak` is a vector of predictor values.
- `ypeak` is a vector of response values.

Import these two variables into Curve Fitting Tool and accept the default data set name `ypeak vs. xpeak`.

You will fit the data with the following equation

$$y(x) = ae^{-bx} + a_1 e^{-\left(\frac{x-b_1}{c_1}\right)^2} + a_2 e^{-\left(\frac{x-b_2}{c_2}\right)^2}$$

where a_i are the peak amplitudes, b_i are the peak centroids, and c_i are related to the peak widths. Because there are unknown coefficients included as part of the exponential function arguments, the equation is nonlinear. Therefore, you must specify the equation using the General Equations pane of the Create Custom Equation GUI. This pane is shown below for $y(x)$.



Independent variable: x

Equation: y = a*exp(-b*x)+a1*exp(-((x-b1)/c1)^2)+a2*exp(-((x-b2)/c2)^2)

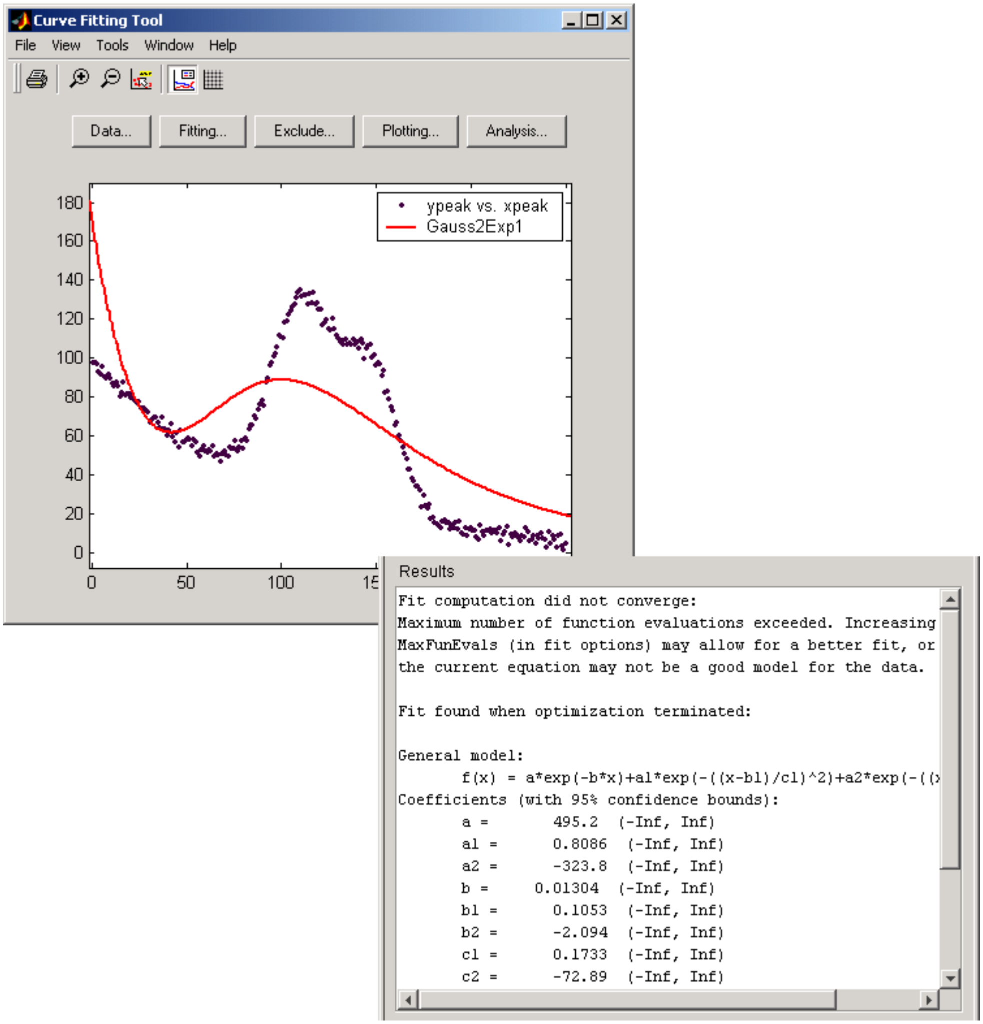
Unknowns	StartPoint	Lower	Upper
a	0.167	-Inf	Inf
a1	0.809	-Inf	Inf
a2	0.523	-Inf	Inf
b	0.744	-Inf	Inf
b1	0.105	-Inf	Inf
b2	0.204	-Inf	Inf
c1	0.173	-Inf	Inf
c2	0.345	-Inf	Inf

Equation name: Gauss2Exp1

Two Gaussian peaks on an exponential background.

By default, the coefficients are unbounded and have random starting values between 0 and 1.

The data, fit, and numerical fit results are shown below. Clearly, the fit is poor.



```
Fit computation did not converge:
Maximum number of function evaluations exceeded. Increasing
MaxFunEvals (in fit options) may allow for a better fit, or
the current equation may not be a good model for the data.

Fit found when optimization terminated:

General model:
       f(x) = a*exp(-b*x)+a1*exp(-((x-b1)/c1)^2)+a2*exp(-((x
Coefficients (with 95% confidence bounds):
       a =       495.2  (-Inf, Inf)
       a1 =      0.8086  (-Inf, Inf)
       a2 =      -323.8  (-Inf, Inf)
       b =     0.01304  (-Inf, Inf)
       b1 =      0.1053  (-Inf, Inf)
       b2 =      -2.094  (-Inf, Inf)
       c1 =      0.1733  (-Inf, Inf)
       c2 =      -72.89  (-Inf, Inf)
```

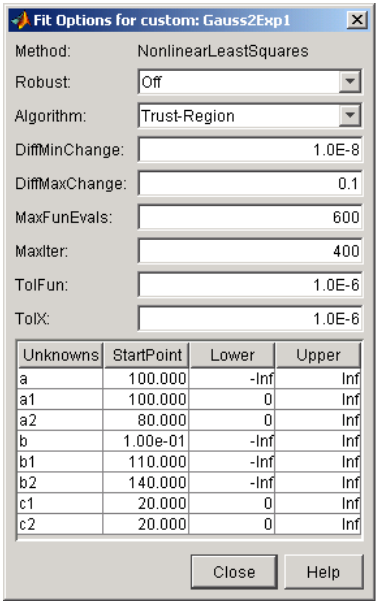
Because the starting points are randomly selected, your initial fit results might differ from the results shown here.

The results include this warning message.

```
Fit computation did not converge:
Maximum number of function evaluations exceeded. Increasing
MaxFunEvals (in fit options) may allow for a better fit, or
the current equation may not be a good model for the data.
```

To improve the fit for this example, specify reasonable starting points for the coefficients. Deducing the starting points is particularly easy for the current model because the Gaussian coefficients have a straightforward interpretation and the exponential background is well defined. Additionally, as the peak amplitudes and widths cannot be negative, constrain a_1, a_2, c_1, and c_2 to be greater then zero.

To define starting values and constraints for unknown coefficients, use the Fit Options GUI, which you open by clicking the **Fit options** button. The starting values and constraints are shown below.



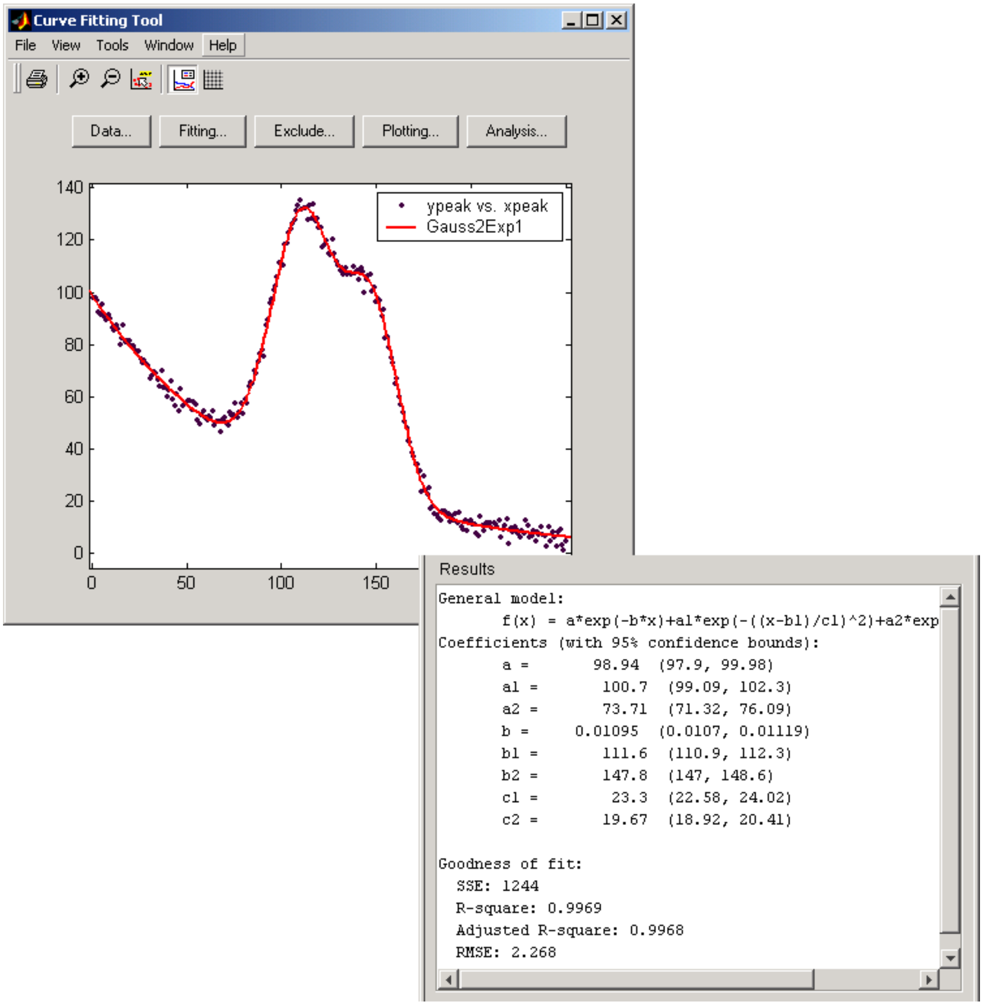
Fit Options for custom: Gauss2Exp1

Method: NonlinearLeastSquares
Robust: Off
Algorithm: Trust-Region
DiffMinChange: 1.0E-8
DiffMaxChange: 0.1
MaxFunEvals: 600
MaxIter: 400
TolFun: 1.0E-6
TolX: 1.0E-6

Unknowns	StartPoint	Lower	Upper
a	100.000	-Inf	Inf
a1	100.000	0	Inf
a2	80.000	0	Inf
b	1.00e-01	-Inf	Inf
b1	110.000	-Inf	Inf
b2	140.000	-Inf	Inf
c1	20.000	0	Inf
c2	20.000	0	Inf

Specify reasonable coefficient starting values and constraints.

The data, fit, residuals, and numerical results are shown below.



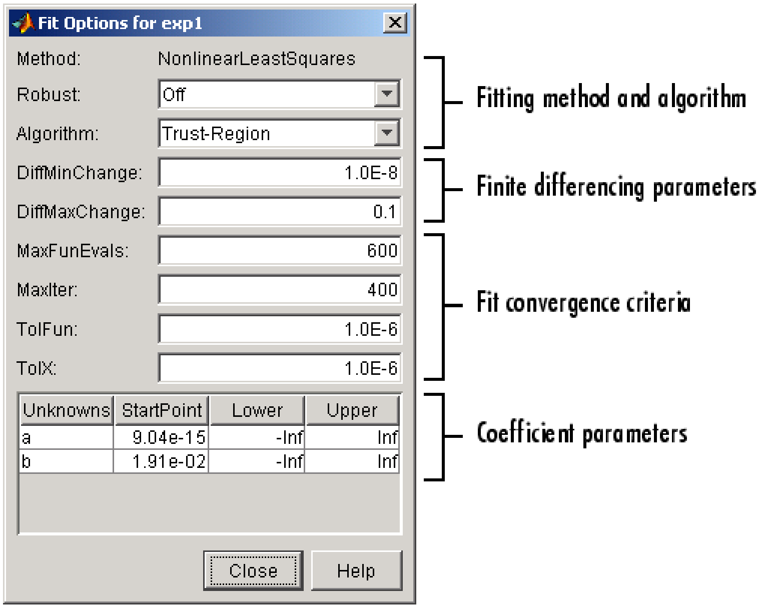
```
General model:
       f(x) = a*exp(-b*x)+a1*exp(-((x-b1)/c1)^2)+a2*exp
Coefficients (with 95% confidence bounds):
       a =       98.94  (97.9, 99.98)
       a1 =       100.7  (99.09, 102.3)
       a2 =       73.71  (71.32, 76.09)
       b =     0.01095  (0.0107, 0.01119)
       b1 =       111.6  (110.9, 112.3)
       b2 =       147.8  (147, 148.6)
       c1 =        23.3  (22.58, 24.02)
       c2 =       19.67  (18.92, 20.41)

Goodness of fit:
  SSE: 1244
  R-square: 0.9969
  Adjusted R-square: 0.9968
  RMSE: 2.268
```

Specifying Fit Options

- “Introduction” on page 2-60
- “Fitting Method and Algorithm” on page 2-60

- “Finite Differencing Parameters” on page 2-61
- “Fit Convergence Criteria” on page 2-62
- “Coefficient Parameters” on page 2-62

Introduction. You specify fit options with the Fit Options GUI. The fit options for the single-term exponential are shown below. The coefficient starting values and constraints are for the census data.



Fit Options for exp1

Method: NonlinearLeastSquares

Robust: Off

Algorithm: Trust-Region

DiffMinChange: 1.0E-8

DiffMaxChange: 0.1

MaxFunEvals: 600

MaxIter: 400

TolFun: 1.0E-6

TolX: 1.0E-6

Unknowns	StartPoint	Lower	Upper
a	9.04e-15	-Inf	Inf
b	1.91e-02	-Inf	Inf

Fitting method and algorithm

Finite differencing parameters

Fit convergence criteria

Coefficient parameters

The available GUI options depend on whether you are fitting your data using a linear model, a nonlinear model, or a nonparametric fit type. All the options described below are available for nonlinear models. **Method**, **Robust**, and coefficient constraints (**Lower** and **Upper**) are available for linear models. Interpolants and smoothing splines include **Method**, but no configurable options.

Fitting Method and Algorithm.

- **Method** — The fitting method.

The method is automatically selected based on the library or custom model you use. For linear models, the method is **LinearLeastSquares**. For nonlinear models, the method is **NonlinearLeastSquares**.

- **Robust** — Specify whether to use the robust least-squares fitting method. The values are
  - **Off** — Do not use robust fitting (default).
  - **On** — Fit with default robust method (bisquare weights).
  - **LAR** — Fit by minimizing the least absolute residuals (LAR).
  - **Bisquare** — Fit by minimizing the summed square of the residuals, and down-weight outliers using bisquare weights. In most cases, this is the best choice for robust fitting.
- **Algorithm** — Algorithm used for the fitting procedure:
  - **Trust-Region** — This is the default algorithm and must be used if you specify coefficient constraints.
  - **Levenberg-Marquardt** — If the trust-region algorithm does not produce a reasonable fit, and you do not have coefficient constraints, you should try the Levenberg-Marquardt algorithm.
  - **Gauss-Newton** — This algorithm is included for pedagogical reasons and should be the last choice for most models and data sets.

Finite Differencing Parameters.

- **DiffMinChange** — Minimum change in coefficients for finite difference Jacobians. The default value is 10^{-8}.
- **DiffMaxChange** — Maximum change in coefficients for finite difference Jacobians. The default value is 0.1.

Note that **DiffMinChange** and **DiffMaxChange** apply to

- Any nonlinear custom equation — that is, a nonlinear equation that you write.
- Some, but not all, of the nonlinear equations provided with Curve Fitting Toolbox software.

However, **DiffMinChange** and **DiffMaxChange** do not apply to any linear equations.

Fit Convergence Criteria.

- **MaxFunEvals** — Maximum number of function (model) evaluations allowed. The default value is 600.
- **MaxIter** — Maximum number of fit iterations allowed. The default value is 400.
- **TolFun** — Termination tolerance used on stopping conditions involving the function (model) value. The default value is 10^{-6}.
- **TolX** — Termination tolerance used on stopping conditions involving the coefficients. The default value is 10^{-6}.

Coefficient Parameters.

- **Unknowns** — Symbols for the unknown coefficients to be fitted.
- **StartPoint** — The coefficient starting values. The default values depend on the model. For rational, Weibull, and custom models, default values are randomly selected within the range [0,1]. For all other nonlinear library models, the starting values depend on the data set and are calculated heuristically.
- **Lower** — Lower bounds on the fitted coefficients. The bounds are used only with the trust region fitting algorithm. The default lower bounds for most library models are `-Inf`, which indicates that the coefficients are unconstrained. However, a few models have finite default lower bounds. For example, Gaussians have the width parameter constrained so that it cannot be less than 0.
- **Upper** — Upper bounds on the fitted coefficients. The bounds are used only with the trust region fitting algorithm. The default upper bounds for all library models are `Inf`, which indicates that the coefficients are unconstrained.

For more information about these fit options, refer to the Optimization Toolbox documentation.

The default coefficient starting points and constraints for library and custom models are given below. If the starting points are optimized, then they are calculated heuristically based on the current data set. Random starting points are defined on the interval [0,1] and linear models do not require starting points.

If a model does not have constraints, the coefficients have neither a lower bound nor an upper bound. You can override the default starting points and constraints by providing your own values using the Fit Options GUI.

Default Starting Points and Constraints

Model	Starting Points	Constraints
Custom linear	N/A	None
Custom nonlinear	Random	None
Exponentials	Optimized	None
Fourier series	Optimized	None
Gaussians	Optimized	$c_i > 0$
Polynomials	N/A	None
Power series	Optimized	None
Rationals	Random	None
Sum of sines	Optimized	$b_i > 0$
Weibull	Random	$a, b > 0$

Note that the sum of sines and Fourier series models are particularly sensitive to starting points, and the optimized values might be accurate for only a few terms in the associated equations.

Example: Rational Fit

This example fits measured data using a rational model. The data describes the coefficient of thermal expansion for copper as a function of temperature in degrees kelvin.

To get started, load the thermal expansion data from the file `hahn1.mat`, which is provided with the toolbox.

```
load hahn1
```

The workspace now contains two new variables, `temp` and `thermex`:

- `temp` is a vector of temperatures in degrees kelvin.
- `thermex` is a vector of thermal expansion coefficients for copper.

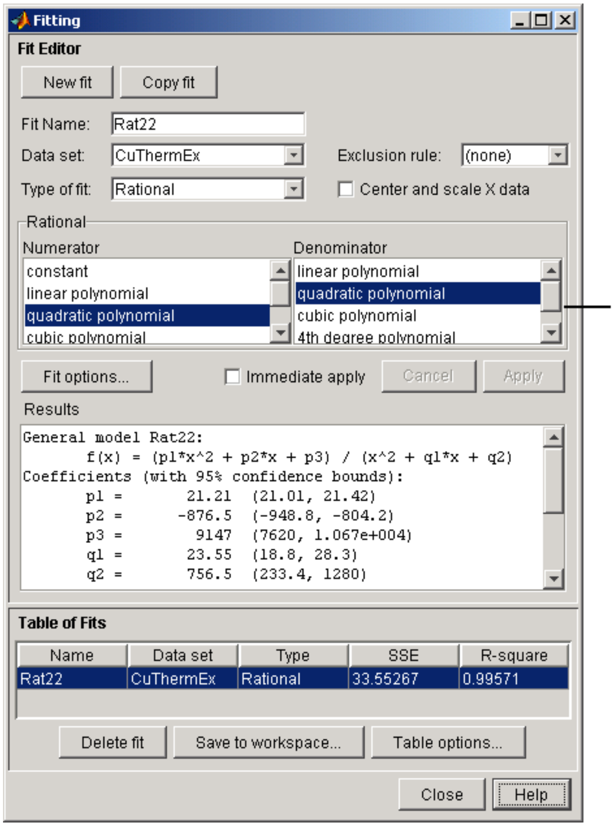
Import these two variables into Curve Fitting Tool and name the data set `CuThermEx`.

For this data set, you will find the rational equation that produces the best fit. As described in “Library Models” on page 2-32, rational models are defined as a ratio of polynomials

$$y = \frac{p_1 x^n + p_2 x^{n-1} + \ldots + p_{n+1}}{x^m + q_1 x^{m-1} + \ldots + q_m}$$

where n is the degree of the numerator polynomial and m is the degree of the denominator polynomial. Note that the rational equations are not associated with physical parameters of the data. Instead, they provide a simple and flexible empirical model that you can use for interpolation and extrapolation.

As you can see by examining the shape of the data, a reasonable initial choice for the rational model is quadratic/quadratic. The Fitting GUI configured for this equation is shown below.



Fit Name: Rat22

Data set: CuThermEx

Exclusion rule: (none)

Type of fit: Rational

□ Center and scale X data

Numerator: quadratic polynomial

Denominator: quadratic polynomial

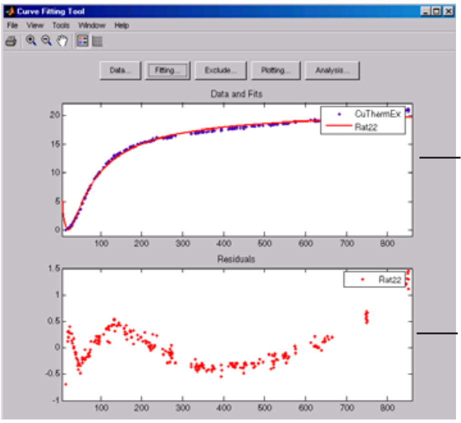
□ Immediate apply

```
General model Rat22:
       f(x) = (p1*x^2 + p2*x + p3) / (x^2 + q1*x + q2)
Coefficients (with 95% confidence bounds):
       p1 =       21.21  (21.01, 21.42)
       p2 =      -876.5  (-948.8, -804.2)
       p3 =        9147  (7620, 1.067e+004)
       q1 =       23.55  (18.8, 28.3)
       q2 =       756.5  (233.4, 1280)
```

Name	Data set	Type	SSE	R-square
Rat22	CuThermEx	Rational	33.55267	0.99571

Begin the fitting process with a quadratic/quadratic rational fit.

The data, fit, and residuals are shown below.

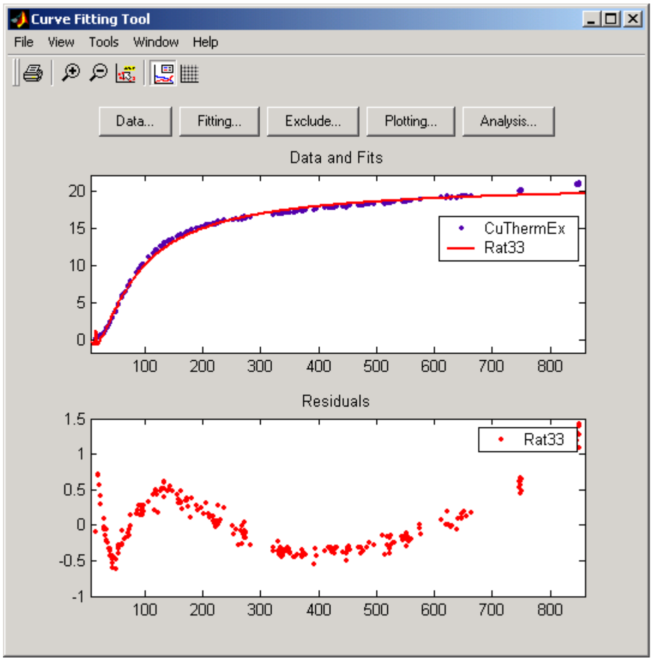


The fit clearly misses some of the data.

The residuals show a strong pattern indicating a better fit is possible.

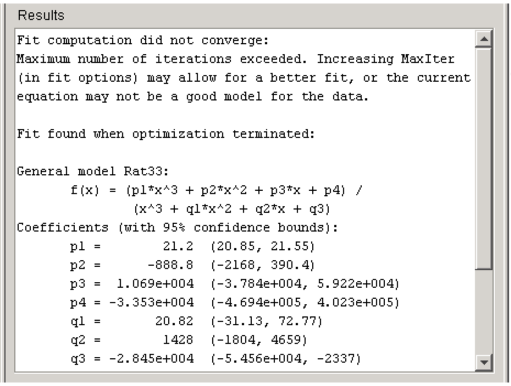
The fit clearly misses the data for the smallest and largest predictor values. Additionally, the residuals show a strong pattern throughout the entire data set indicating that a better fit is possible.

For the next fit, try a cubic/cubic equation. The data, fit, and residuals are shown below.



The fit exhibits several discontinuities around the zeros of the denominator.

The numerical results shown below indicate that the fit did not converge.

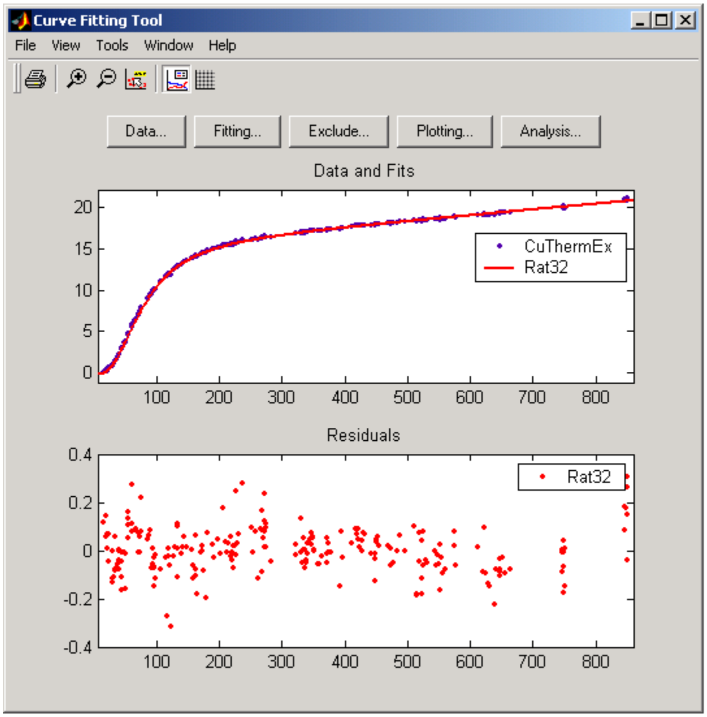


```
Fit computation did not converge:
Maximum number of iterations exceeded. Increasing MaxIter
(in fit options) may allow for a better fit, or the current
equation may not be a good model for the data.

Fit found when optimization terminated:

General model Rat33:
       f(x) = (p1*x^3 + p2*x^2 + p3*x + p4) /
               (x^3 + q1*x^2 + q2*x + q3)
Coefficients (with 95% confidence bounds):
       p1 =        21.2  (20.85, 21.55)
       p2 =      -888.8  (-2168, 390.4)
       p3 =  1.069e+004  (-3.784e+004, 5.922e+004)
       p4 = -3.353e+004  (-4.694e+005, 4.023e+005)
       q1 =       20.82  (-31.13, 72.77)
       q2 =        1428  (-1804, 4659)
       q3 = -2.845e+004  (-5.456e+004, -2337)
```

The fit did not converge, which indicates that the model might be a poor choice for the data.

Although the message in the **Results** window indicates that you might improve the fit if you increase the maximum number of iterations, a better choice at this stage of the fitting process is to use a different rational equation because the current fit contains several discontinuities. These discontinuities are due to the function blowing up at predictor values that correspond to the zeros of the denominator.

As the next try, fit the data using a cubic/quadratic equation. The data, fit, and residuals are shown below.



The fit is well behaved over the entire data range.

The residuals are randomly scattered about zero.

The fit is well behaved over the entire data range, and the residuals are randomly scattered about zero. Therefore, you can confidently use this fit for further analysis.

Example: Robust Fitting

This example fits data that is assumed to contain one outlier. The data consists of the 2000 United States presidential election results for the state of Florida. The fit model is a first degree polynomial and the fit method is robust linear least squares with bisquare weights.

In the 2000 presidential election, many residents of Palm Beach County, Florida, complained that the design of the election ballot was confusing, which they claim led them to vote for the Reform candidate Pat Buchanan instead of the Democratic candidate Al Gore. The so-called “butterfly ballot” was used only in Palm Beach County and only for the election-day ballots for the presidential race. As you will see, the number of Buchanan votes for Palm Beach is far removed from the bulk of data, which suggests that the data point should be treated as an outlier.

To get started, load the Florida election result data from the file `flvote2k.mat`, which is provided with the toolbox.

```
load flvote2k
```

The workspace now contains these three new variables:

- `buchanan` is a vector of votes for the Reform Party candidate Pat Buchanan.
- `bush` is a vector of votes for the Republican Party candidate George Bush.
- `gore` is a vector of votes for the Democratic Party candidate Al Gore.

Each variable contains 68 elements, which correspond to the 67 Florida counties plus the absentee ballots. The names of the counties are given in the variable `counties`. From these variables, create two data sets with the Buchanan votes as the response data: `buchanan vs. bush` and `buchanan vs. gore`.

For this example, assume that the relationship between the response and predictor data is linear with an offset of zero.

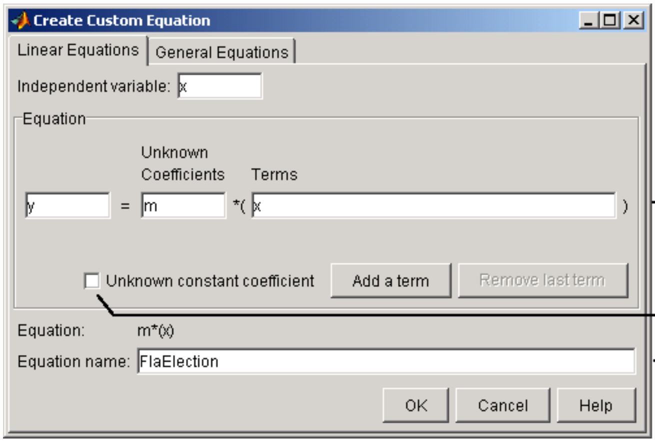
$\textit{buchanan votes} = (\textit{bush votes})(m_1)$

$\textit{buchanan votes} = (\textit{gore votes})(m_2)$

m_1 is the number of Bush votes expected for each Buchanan vote, and m_2 is the number of Gore votes expected for each Buchanan vote.

To create a first-degree polynomial equation with zero offset, you must create a custom linear equation. You create a custom equation using the Fitting GUI by selecting **Custom Equations** from the **Type of fit** list, and then clicking the **New Equation** button.

The Linear Equations pane of the Create Custom Equation GUI is shown below.



Independent variable: x

y = m *(x)

□ Unknown constant coefficient

Equation: m*(x)

Equation name: FlaElection

Create a first-degree polynomial with zero offset.

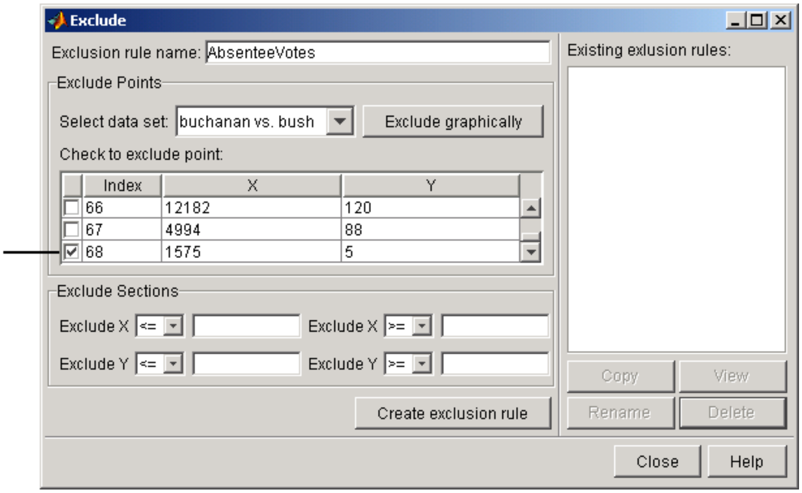
Clear this check box.

Assign a meaningful name to the equation.

Before fitting, you should exclude the data point associated with the absentee ballots from each data set because these voters did not use the butterfly ballot. As described in “Marking Outliers” on page 2-19, you can exclude individual data points from a fit either graphically or numerically using the Exclude GUI. For this example, you should exclude the data numerically. The index of the absentee ballot data is given by

```
ind = find(strcmp(counties,'Absentee Ballots'))
ind =
    68
```

The Exclude GUI is shown below.



Exclusion rule name: AbsenteeVotes

Existing exlusion rules:

Select data set: buchanan vs. bush

Check to exclude point:

	Index	X	Y
□	66	12182	120
□	67	4994	88
☑	68	1575	5

Exclude X <=

Exclude X >=

Exclude Y <=

Exclude Y >=

Mark the absentee votes to be excluded.

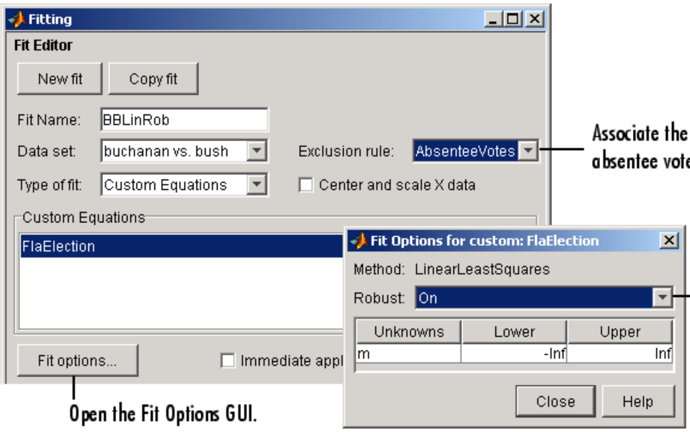
The exclusion rule is named `AbsenteeVotes`. You use the Fitting GUI to associate an exclusion rule with the data set to be fit.

For each data set, perform a robust fit with bisquare weights using the `FlaElection` equation defined above. For comparison purposes, also perform a regular linear least-squares fit.

You can identify the Palm Beach County data in the scatter plot by using the data tips feature, and knowing the index number of the data point.

```
ind = find(strcmp(counties,'Palm Beach'))
ind =
    50
```

The **Fit Editor** and the Fit Options GUI are shown below for a robust fit.



Fit Name: BBLinRob

Data set: buchanan vs. bush

Exclusion rule: AbsenteeVotes

Type of fit: Custom Equations

□ Center and scale X data

FlaElection

□ Immediate appl

Fit Options for custom: FlaElection

Method: LinearLeastSquares

Robust: On

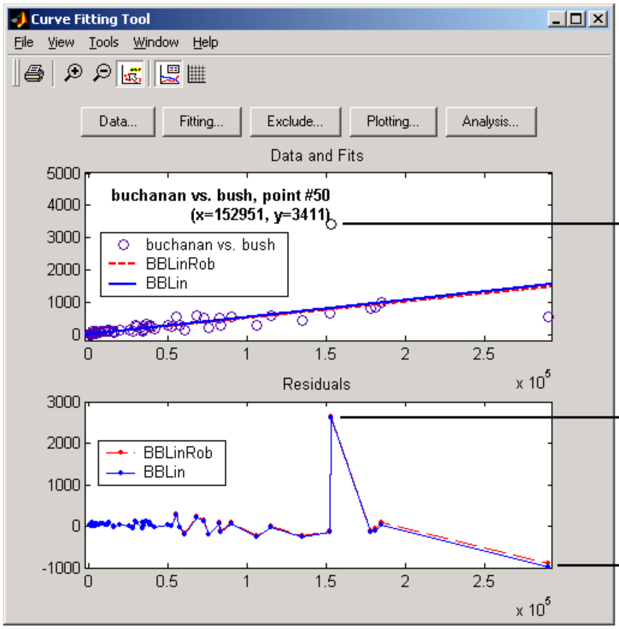
Unknowns	Lower	Upper
m	-Inf	Inf

Associate the excluded absentee votes with the fit.

Choose robust fitting with bisquare weights.

Open the Fit Options GUI.

The data, robust and regular least-squares fits, and residuals for the buchanan vs. bush data set are shown below.



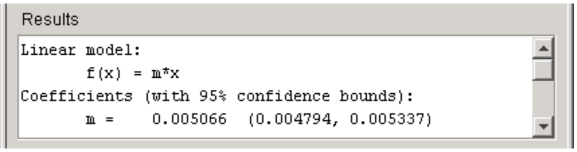
The data tip shows that Buchanan received 3411 votes in Palm Beach County.

The Palm Beach County residual is very large.

The Miami/Dade County residual is also very large.

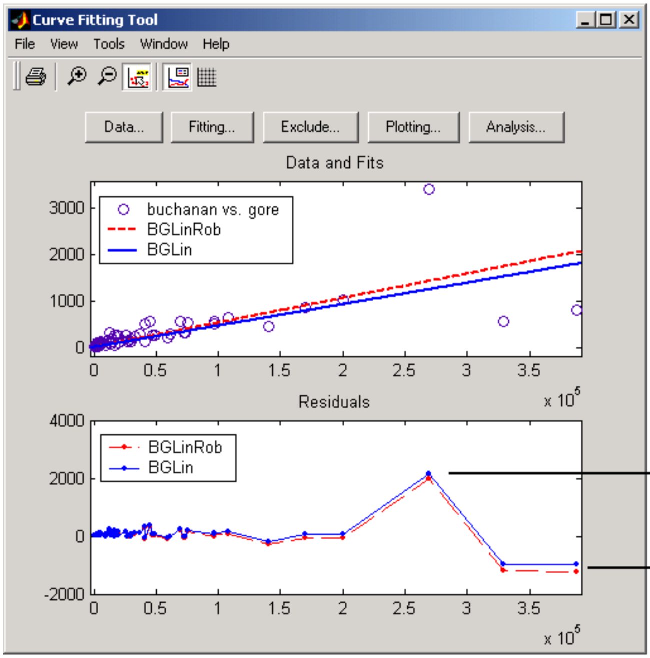
The graphical results show that the linear model is reasonable for the majority of data points, and the residuals appear to be randomly scattered around zero. However, two residuals stand out. The largest residual corresponds to Palm Beach County. The other residual is at the largest predictor value, and corresponds to Miami/Dade County.

The numerical results are shown below. The inverse slope of the robust fit indicates that Buchanan should receive one vote for every 197.4 Bush votes.



```
Linear model:
       f(x) = m*x
Coefficients (with 95% confidence bounds):
       m =    0.005066  (0.004794, 0.005337)
```

The data, robust and regular least-squares fits, and residuals for the `buchanan vs. gore` data set are shown below.

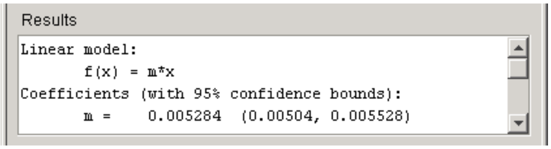


The Palm Beach County residual is very large.

The Miami/Dade and Broward County residuals are also very large.

Again, the graphical results show that the linear model is reasonable for the majority of data points, and the residuals appear to be randomly scattered around zero. However, three residuals stand out. The largest residual corresponds to Palm Beach County. The other residuals are at the two largest predictor values, and correspond to Miami/Dade County and Broward County.

The numerical results are shown below. The inverse slope of the robust fit indicates that Buchanan should receive one vote for every 189.3 Gore votes.



```
Linear model:
       f(x) = m*x
Coefficients (with 95% confidence bounds):
       m =    0.005284  (0.00504, 0.005528)
```

Using the fitted slope value, you can determine the expected number of votes that Buchanan should have received for each fit. For the Buchanan versus Bush data, you evaluate the fit at a predictor value of 152,951. For the Buchanan versus Gore data, you evaluate the fit at a predictor value of 269,732. These results are shown below for both data sets and both fits.

Expected Buchanan Votes in Palm Beach County

Data Set	Fit	Expected Buchanan Votes
Buchanan vs. Bush	Ordinary least squares	814
	Robust least squares	775
Buchanan vs. Gore	Ordinary least squares	1246
	Robust least squares	1425

The robust results for the Buchanan versus Bush data suggest that Buchanan received 3411 – 775 = 2636 excess votes, while robust results for the Buchanan versus Gore data suggest that Buchanan received 3411 – 1425 = 1986 excess votes.

The margin of victory for George Bush is given by

```
margin = sum(bush)-sum(gore)
margin =

   537
```

Therefore, the voter intention comes into play because in both cases, the margin of victory is less than the excess Buchanan votes.

In conclusion, the analysis of the 2000 United States presidential election results for the state of Florida suggests that the Reform Party candidate received an excess number of votes in Palm Beach County, and that this excess number was a crucial factor in determining the election outcome. However, additional analysis is required before a final conclusion can be made.

Nonparametric Fitting

Introduction

In some cases, you are not concerned about extracting or interpreting fitted parameters. Instead, you might simply want to draw a smooth curve through your data. Fitting of this type is called *nonparametric fitting*. The Curve Fitting Toolbox software supports these nonparametric fitting methods:

- Interpolants — Estimate values that lie between known data points.
- Smoothing spline — Create a smooth curve through the data. You adjust the level of smoothness by varying a parameter that changes the curve from a least-squares straight-line approximation to a cubic spline interpolant.

For more information about interpolation, refer to "Polynomials" and the `interp1` function in the MATLAB documentation.

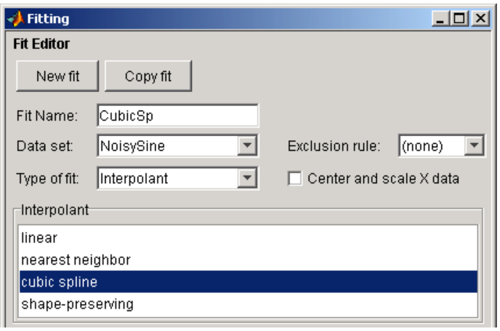
Example: Nonparametric Fitting

This example fits the following data using a cubic spline interpolant and several smoothing splines.

```
x = (4*pi)*[0 1 rand(1,25)];
```

```
y = sin(x) + .2*(rand(size(x))-.5);
```

As shown below, you can fit the data with a cubic spline interpolant by selecting **Interpolant** from the **Type of fit** list.



Fit Name: CubicSp

Data set: NoisySine

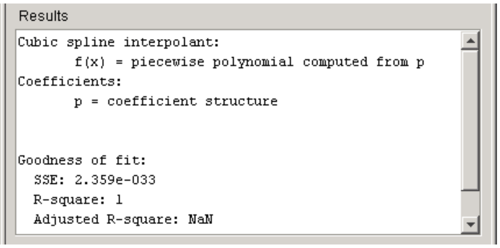
Exclusion rule: (none)

Type of fit: Interpolant

□ Center and scale X data

Interpolant: cubic spline

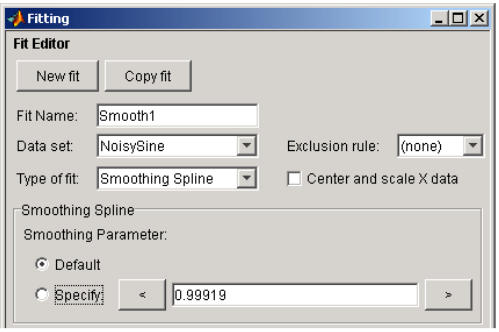
The results shown below indicate that goodness-of-fit statistics are not defined for interpolants.



```
Cubic spline interpolant:
       f(x) = piecewise polynomial computed from p
Coefficients:
       p = coefficient structure


Goodness of fit:
  SSE: 2.359e-033
  R-square: 1
  Adjusted R-square: NaN
```

A cubic spline interpolation is defined as a piecewise polynomial that results in a structure of coefficients. The number of “pieces” in the structure is one less than the number of fitted data points, and the number of coefficients for each piece is four because the polynomial degree is three. The toolbox does not allow you to access the structure of coefficients.

As shown below, you can fit the data with a smoothing spline by selecting **Smoothing Spline** in the **Type of fit** list.



Fit Name: Smooth1

Data set: NoisySine

Exclusion rule: (none)

Type of fit: Smoothing Spline

□ Center and scale X data

Smoothing Spline

Smoothing Parameter:

● Default

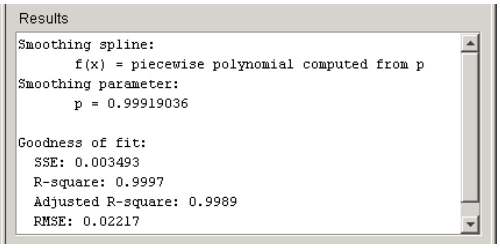
○ Specify 0.99919

The default smoothing parameter is based on the data set you fit.

The level of smoothness is given by the **Smoothing Parameter**. The default smoothing parameter value depends on the data set, and is automatically calculated by the toolbox after you click the **Apply** button.

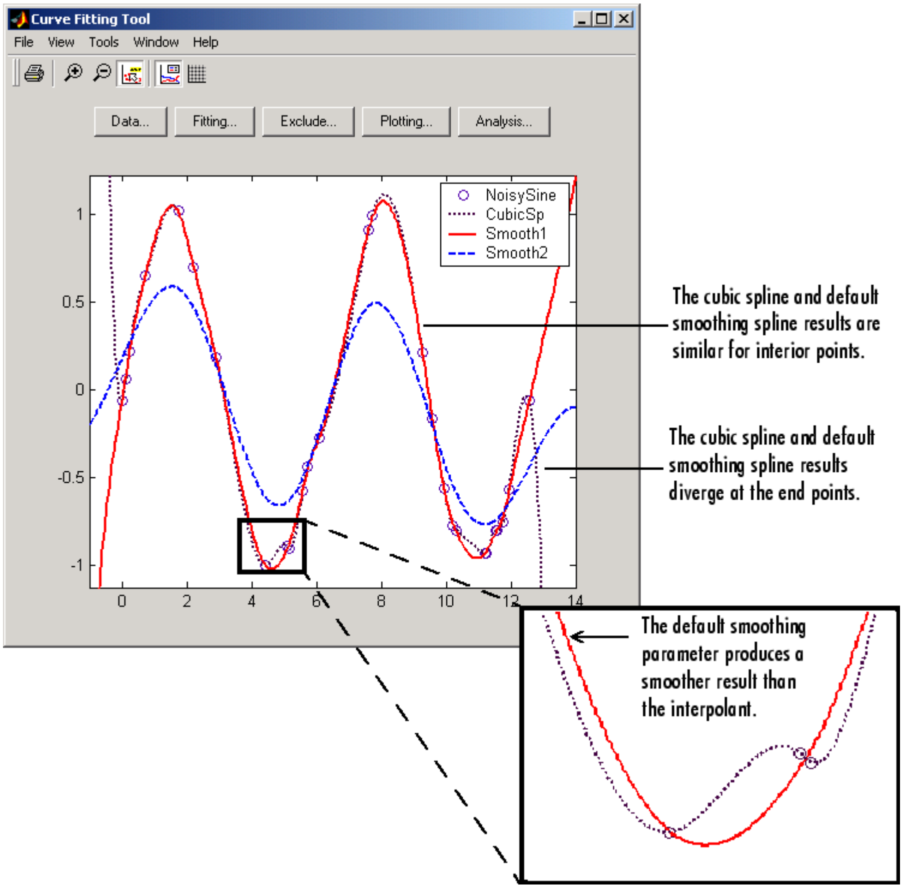
For this data set, the default smoothing parameter is close to 1, indicating that the smoothing spline is nearly cubic and comes very close to passing through each data point. Create a fit for the default smoothing parameter and name it `Smooth1`. If you do not like the level of smoothing produced by the default smoothing parameter, you can specify any value between 0 and 1. A value of 0 produces a linear polynomial fit, while a value of 1 produces a piecewise cubic polynomial fit that passes through all the data points. For comparison purposes, create another smoothing spline fit using a smoothing parameter of 0.5 and name the fit `Smooth2`.

The numerical results for the smoothing spline fit `Smooth1` are shown below.



```
Smoothing spline:
       f(x) = piecewise polynomial computed from p
Smoothing parameter:
       p = 0.99919036

Goodness of fit:
  SSE: 0.003493
  R-square: 0.9997
  Adjusted R-square: 0.9989
  RMSE: 0.02217
```

The data and fits are shown below. The default abscissa scale was increased to show the fit behavior beyond the data limits. You change the axes limits with **Tools > Axes Limit Control** menu item.



The cubic spline and default smoothing spline results are similar for interior points.

The cubic spline and default smoothing spline results diverge at the end points.

The default smoothing parameter produces a smoother result than the interpolant.

Note that the default smoothing parameter produces a curve that is smoother than the interpolant, but is a good fit to the data. In this case, decreasing the smoothing parameter from the default value produces a curve that is smoother still, but is not a good fit to the data. As the smoothing parameter increases beyond the default value, the associated curve approaches the cubic spline interpolant.

Programmatic Curve Fitting

Curve Fitting Objects and Methods

In this section...

This section describes how to use Curve Fitting Toolbox functions to write object-oriented programs for curve fitting applications.

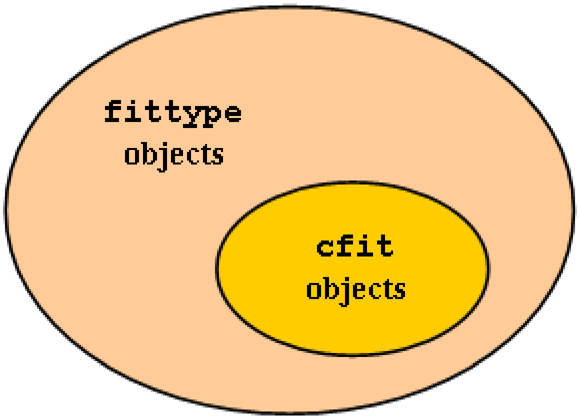
Overview

In MATLAB programming, all workspace variables are *objects* of a particular *class*. Familiar examples of MATLAB classes are `double`, `char`, and `function_handle`. You can also create custom MATLAB classes, using object-oriented programming.

Methods are functions that operate exclusively on objects of a particular class. *Data types* package together objects and methods so that the methods operate exclusively on objects of their own type, and not on objects of other types. A clearly defined encapsulation of objects and methods is the goal of object-oriented programming.

Curve Fitting Toolbox software provides you with two new MATLAB data types for performing curve fitting:

- `fittype` — Objects allow you to encapsulate information describing a parametric model for your data. Methods allow you to access and modify that information.
- `cfit` — A subtype of `fittype`. Objects capture information from a particular fit by assigning values to coefficients, confidence intervals, fit statistics, etc. Methods allow you to post-process the fit through plotting, extrapolation, integration, etc.



Because cfit is a subtype of fittype, cfit inherits all fittype methods. In other words, you can apply fittype methods to both fittype and cfit objects, but cfit methods are used exclusively with cfit objects.

As an example, the fittype method islinear, which determines if a model is linear or nonlinear, would apply equally well before or after a fit; that is, to both fittype and cfit objects. On the other hand, the cfit methods coeffvalues and confint, which, respectively, return fit coefficients and their confidence intervals, would make no sense if applied to a general fittype object which describes a parametric model with undetermined coefficients.

Curve Fitting Objects

Curve fitting objects have properties that depend on their type, and also on the particulars of the model or the fit that they encapsulate. For example, the following code uses the constructor methods for the two curve fitting types to create a fittype object f and a cfit object c:

```
f = fittype('a*x^2+b*exp(n*x)')
f =
     General model:
       f(a,b,n,x) = a*x^2+b*exp(n*x)
c = cfit(f,1,10.3,-1e2)
c =
     General model:
```

```
     c(x) = a*x^2+b*exp(n*x)
   Coefficients:
     a =          1
     b =       10.3
     n =       -100
```

Note that the display method for `fittype` objects returns only basic information, piecing together outputs from `formula` and `indepnames`.

`cfit` and `fittype` objects are evaluated at predictor values `x` using `feval`. You can call `feval` indirectly using the following functional syntax:

```
y = cfun(x) % cfit objects;
y = ffun(coef1,coef2,...,x) % fittype objects;
```

Curve Fitting Methods

Curve fitting methods allow you to create, access, and modify curve fitting objects. They also allow you, through methods like `plot` and `integrate`, to perform operations that uniformly process the entirety of information encapsulated in a curve fitting object.

The methods listed in the following table are available for all `fittype` objects, including `cfit` objects.

Fit Type Method	Description
`argnames`	Get input argument names
`category`	Get fit category
`coeffnames`	Get coefficient names
`dependnames`	Get dependent variable name
`feval`	Evaluate model at specified predictors
`fittype`	Construct `fittype` object
`formula`	Get formula string
`indepnames`	Get independent variable name
`islinear`	Determine if model is linear
`numargs`	Get number of input arguments

Fit Type Method	Description
`numcoeffs`	Get number of coefficients
`probnames`	Get problem-dependent parameter names
`type`	Get name of model

The methods listed in the following table are available exclusively for `cfit` objects.

Curve Fit Method	Description
`cfit`	Construct `cfit` object
`coeffvalues`	Get coefficient values
`confint`	Get confidence intervals for fit coefficients
`differentiate`	Differentiate fit
`integrate`	Integrate fit
`plot`	Plot fit
`predint`	Get prediction intervals
`probvalues`	Get problem-dependent parameter values

A complete list of methods for a curve fitting object can be obtained with the MATLAB `methods` command. For example,

```
f = fittype('a*x^2+b*exp(n*x)');
methods(f)

Methods for class fittype:

argnames        fitoptions      nonlinearcoeffs
cat             fittype         numargs
category        formula         numcoeffs
char            getcoeffmatrix  prettyname
clearhandles    horzcat         probnames
coeffnames      indepnames      saveobj
constants       integexpr       setoptions
dependnames     isempty         startpt
```

```
derivexpr        islinear         subsasgn
disp             linearexprs      subsref
display          linearterms      symvar
exist            loadobj          type
feval            nargin           vertcat
fevalexpr        nargout
```

Note that some of the methods listed by `methods` do not appear in the tables above, and do not have reference pages in the Curve Fitting Toolbox documentation. These additional methods are generally low-level operations used by Curve Fitting Tool, and not of general interest when writing curve fitting applications.

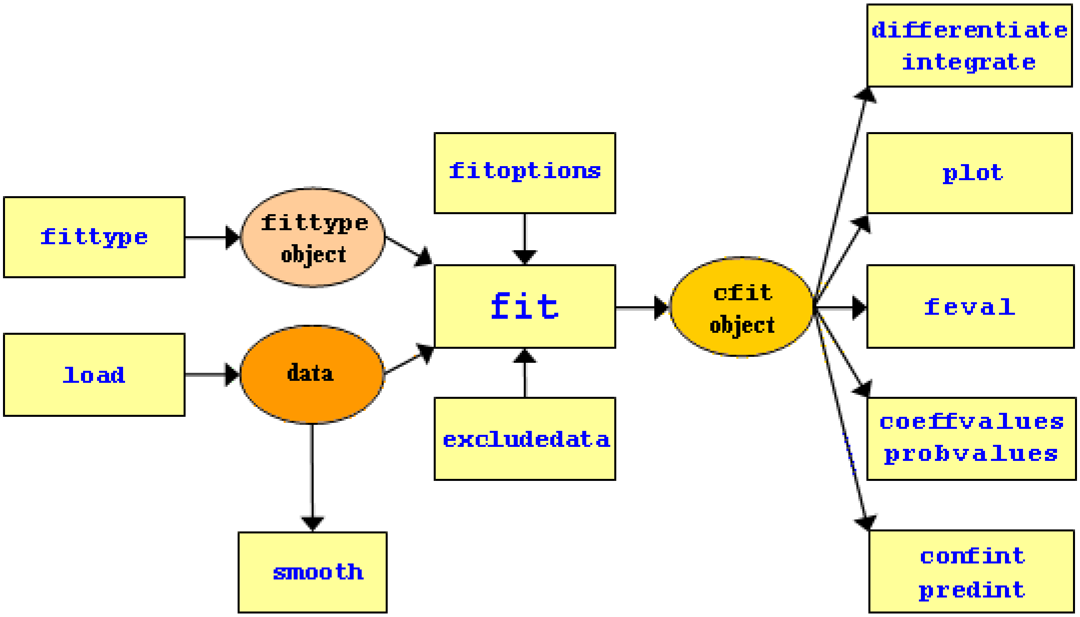
There are no global accessor methods, comparable to `getfield` and `setfield`, available for `fittype` objects. Access is limited to the methods listed above. This is because many of the properties of `fittype` objects are derived from other properties, for which you do have access. For example,

```
f = fittype('a*cos( b*x-c )')
f =
     General model:
       f(a,b,c,x) = a*cos( b*x-c )

formula(f)
ans =
a*cos( b*x-c )

argnames(f)
ans =
    'a'
    'b'
    'c'
    'x'
```

You construct the `fittype` object `f` by giving the formula, so you do have write access to that basic property of the object. You have read access to that property through the `formula` method. You also have read access to the argument names of the object, through the `argnames` method. You don't, however, have direct write access to the argument names, which are derived from the formula. If you want to set the argument names, set the formula.

Workflow for Object-Oriented Fitting

Curve Fitting Toolbox software provides a variety of methods for data analysis and modeling. In application, these methods are applied in a systematic manner, which can be represented in a standard workflow diagram such as the one below.



A typical analysis using curve fitting methods proceeds as follows:

1. Import your data into the MATLAB workspace using the `load` command (if your data has previously been stored in MATLAB variables) or any of the more specialized MATLAB functions for reading data from particular file types.

2. If your data is noisy, you might want to smooth it using the `smooth` function. Smoothing is used to identify major trends in the data that can assist you in choosing an appropriate family of parametric models. If a parametric model is not evident or appropriate, smoothing can be an end in itself, providing a nonparametric fit of the data.

Note Smoothing estimates the center of the distribution of the response at each predictor. It invalidates the assumption that errors in the data are independent, and so also invalidates the methods used to compute confidence and prediction intervals. Accordingly, once a parametric model is identified through smoothing, the *original* data should be passed to the `fit` function.

3. A parametric model for the data—either a Curve Fitting Toolbox library model or a custom model that you define—is specified as a `fittype` object using the `fittype` function. Library models are displayed with the `cflibhelp` function.

4. A fit options structure can be created for the fit using the `fitoptions` function. Fit options specify things like weights for the data, fitting methods, and low-level options for the fitting algorithm.

5. An exclusion rule can be created for the fit using the `excludedata` function. Exclusion rules indicate which data values will be treated as outliers and excluded from the fit.

6. Data, a `fittype` object, and (optionally) a fit options structure and an exclusion rule are all passed to the `fit` function to perform the fit. The `fit` function returns a `cfit` object that encapsulates the computed coefficients and the fit statistics.

7. `cfit` objects returned by the `fit` function can then be passed to a variety of postprocessing functions, such as `feval`, `differentiate`, `integrate`, `plot`, `coeffvalues`, `probvalues`, `confint`, and `predint`.

Examples

The following examples illustrate the standard workflow outlined in "Workflow for Object-Oriented Fitting" on page 3-7. Further examples of programmatic fitting can be found in the reference pages for individual curve fitting methods.

- "Example: Smoothing Data I" on page 3-9
- "Example: Smoothing Data II" on page 3-10

- "Example: Excluding Data" on page 3-11
- "Example: Specifying Fit Options" on page 3-14
- "Example: Robust Fitting" on page 3-15
- "Example: Differentiating and Integrating a Fit" on page 3-17
- "Example: Prediction Intervals" on page 3-21

Example: Smoothing Data I

Load the data in `count.dat`:

```
load count.dat
```

The 24-by-3 array `count` contains traffic counts at three intersections for each hour of the day.

First, use a moving average filter with a 5-hour span to smooth all of the data at once (by linear index) :

```
c = smooth(count(:));
C1 = reshape(c,24,3);
```

Plot the original data and the smoothed data:

```
subplot(3,1,1)
plot(count,':');
hold on
plot(C1,'-');
title('Smooth C1 (All Data)')
```

Second, use the same filter to smooth each column of the data separately:

```
C2 = zeros(24,3);
for I = 1:3,
    C2(:,I) = smooth(count(:,I));
end
```

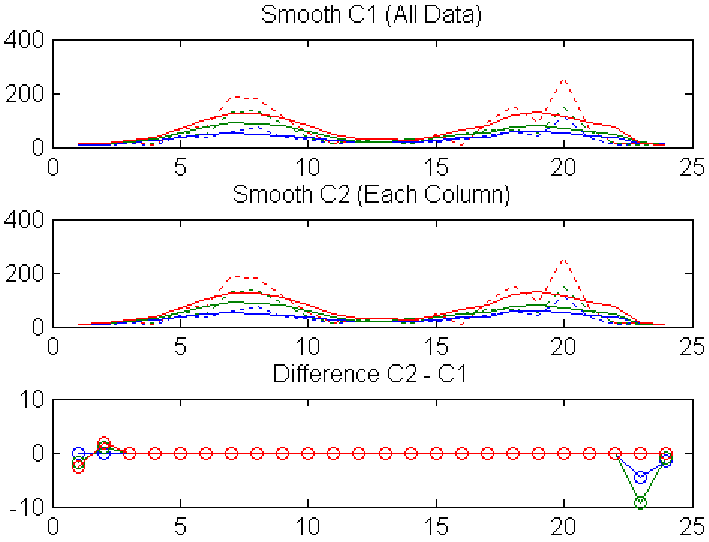
Again, plot the original data and the smoothed data:

```
subplot(3,1,2)
plot(count,':');
```

```
hold on
plot(C2,'-');
title('Smooth C2 (Each Column)')
```

Plot the difference between the two smoothed data sets:

```
subplot(3,1,3)
plot(C2 - C1,'o-')
title('Difference C2 - C1')
```



Note the additional end effects from the 3-column smooth.

Example: Smoothing Data II

Create noisy data with outliers:

```
x = 15*rand(150,1);
y = sin(x) + 0.5*(rand(size(x))-0.5);
y(ceil(length(x)*rand(2,1))) = 3;
```

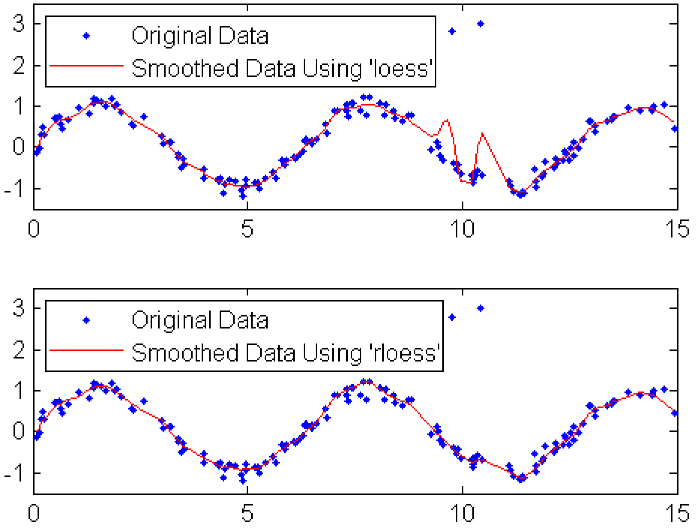
Smooth the data using the `loess` and `rloess` methods with a span of 10%:

```
yy1 = smooth(x,y,0.1,'loess');
```

```
yy2 = smooth(x,y,0.1,'rloess');
```

Plot original data and the smoothed data.

```
[xx,ind] = sort(x);
subplot(2,1,1)
plot(xx,y(ind),'b.',xx,yy1(ind),'r-')
set(gca,'YLim',[-1.5 3.5])
legend('Original Data','Smoothed Data Using ''loess''',...
       'Location','NW')
subplot(2,1,2)
plot(xx,y(ind),'b.',xx,yy2(ind),'r-')
set(gca,'YLim',[-1.5 3.5])
legend('Original Data','Smoothed Data Using ''rloess''',...
       'Location','NW')
```



Note that the outliers have less influence on the robust method.

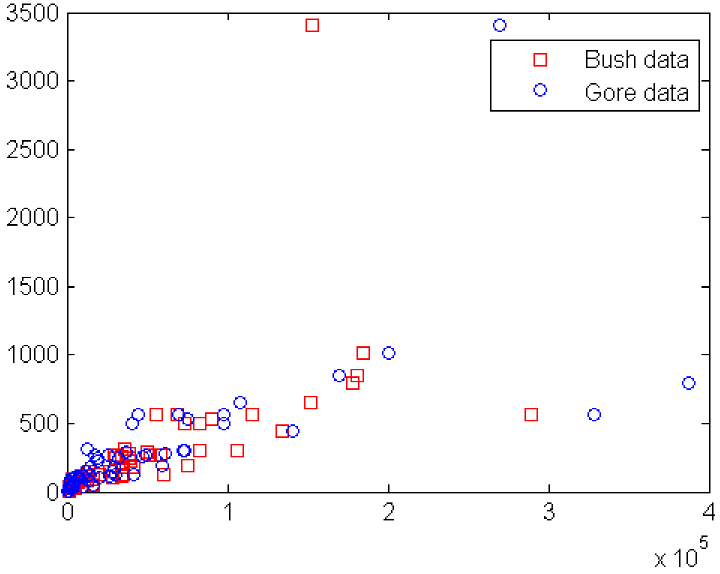
Example: Excluding Data

Load the vote counts and county names for the state of Florida from the 2000 U.S. presidential election:

```
load flvote2k
```

Use the vote counts for the two major party candidates, Bush and Gore, as predictors for the vote counts for third-party candidate Buchanan, and plot the scatters:

```
plot(bush,buchanan,'rs')
hold on
plot(gore,buchanan,'bo')
legend('Bush data','Gore data')
```



Assume a model where a fixed proportion of Bush or Gore voters choose to vote for Buchanan:

```
f = fittype({'x'})
f =
     Linear model:
       f(a,x) = a*x
```

Exclude the data from absentee voters, who did not use the controversial “butterfly” ballot:

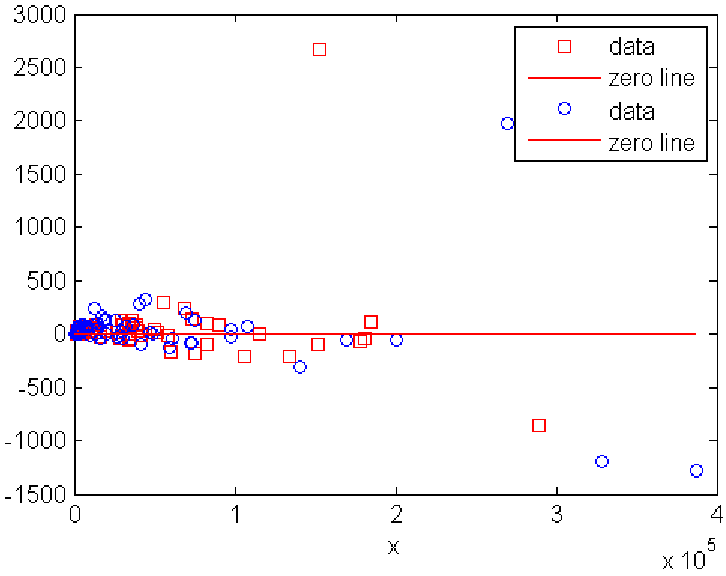
```
absentee = find(strcmp(counties,'Absentee Ballots'));
nobutterfly = excludedata(bush,buchanan,'indices',absentee);
```

Perform a bisquare weights robust fit of the model to the two data sets, excluding absentee voters:

```
bushfit = fit(bush,buchanan,f,...
              'Exclude',nobutterfly,'Robust','on');
gorefit = fit(gore,buchanan,f,...
              'Exclude',nobutterfly,'Robust','on');
```

Robust fits give outliers a low weight, so large residuals from a robust fit can be used to identify the outliers:

```
figure
plot(bushfit,bush,buchanan,'rs','residuals')
hold on
plot(gorefit,gore,buchanan,'bo','residuals')
```



The residuals in the plot above can be computed as follows:

```
bushres = buchanan - feval(bushfit,bush);
```

```
goreres = buchanan - feval(gorefit,gore);
```

Large residuals can be identified as those outside the range [-500 500]:

```
bushoutliers = excludedata(bush,bushres,'range',[-500 500]);
goreoutliers = excludedata(gore,goreres,'range',[-500 500]);
```

The outliers for the two data sets correspond to the following counties:

```
counties(bushoutliers)
ans =
    'Miami-Dade'
    'Palm Beach'

counties(goreoutliers)
ans =
    'Broward'
    'Miami-Dade'
    'Palm Beach'
```

Miami-Dade and Broward counties correspond to the largest predictor values. Palm Beach county, the only county in the state to use the "butterfly" ballot, corresponds to the largest residual values.

Example: Specifying Fit Options

Create the default fit options structure and set the option to center and scale the data before fitting:

```
options = fitoptions;
options.Normal = 'on';
options
options =
    Normalize: 'on'
    Exclude: [1x0 double]
    Weights: [1x0 double]
    Method: 'None'
```

Modifying the default fit options structure is useful when you want to set the `Normalize`, `Exclude`, or `Weights` fields, and then fit your data using the same options with different fitting methods. For example:

```
load census
f1 = fit(cdate,pop,'poly3',options);
f2 = fit(cdate,pop,'exp1',options);
f3 = fit(cdate,pop,'cubicsp',options);
```

Data-dependent fit options are returned in the third output argument of the `fit` function. For example:

```
[f,gof,out] = fit(cdate,pop,'smooth');
smoothparam = out.p
smoothparam =
    0.0089
```

The default smoothing parameter can be modified for a new fit:

```
options = fitoptions('Method','Smooth','SmoothingParam',0.0098);
[f,gof,out] = fit(cdate,pop,'smooth',options);
```

Example: Robust Fitting

Create a baseline sinusoidal signal:

```
xdata = (0:0.1:2*pi)';
y0 = sin(xdata);
```

Add noise to the signal with non-constant variance:

```
% Response-dependent Gaussian noise
gnoise = y0.*randn(size(y0));

% Salt-and-pepper noise
spnoise = zeros(size(y0));
p = randperm(length(y0));
sppoints = p(1:round(length(p)/5));
spnoise(sppoints) = 5*sign(y0(sppoints));

ydata = y0 + gnoise + spnoise;
```

Fit the noisy data with a baseline sinusoidal model:

```
f = fittype('a*sin(b*x)');
fit1 = fit(xdata,ydata,f,'StartPoint',[1 1]);
```

Identify “outliers” as points at a distance greater than 1.5 standard deviations from the baseline model, and refit the data with the outliers excluded:

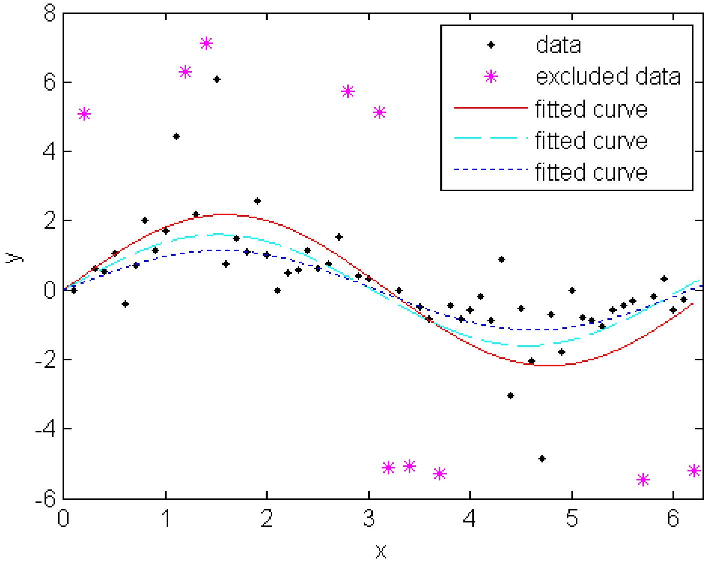
```
fdata = feval(fit1,xdata);
I = abs(fdata - ydata) > 1.5*std(ydata);
outliers = excludedata(xdata,ydata,'indices',I);

fit2 = fit(xdata,ydata,f,'StartPoint',[1 1],'Exclude',outliers);
```

Compare the effect of excluding the outliers with the effect of giving them lower bisquare weight in a robust fit:

```
fit3 = fit(xdata,ydata,f,'StartPoint',[1 1],'Robust','on');
```

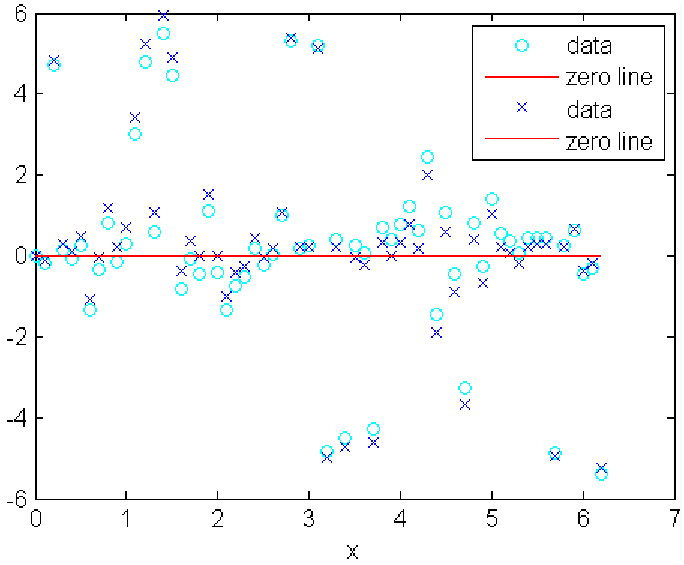
Plot the data, the outliers, and the results of the fits:

```
plot(fit1,'r-',xdata,ydata,'k.',outliers,'m*')
hold on
plot(fit2,'c--')
plot(fit3,'b:')
xlim([0 2*pi])
```



Plot the residuals for the two fits considering outliers:

```
figure
plot(fit2,xdata,ydata,'co','residuals')
hold on
plot(fit3,xdata,ydata,'bx','residuals')
```



Example: Differentiating and Integrating a Fit

Create a baseline sinusoidal signal:

```
xdata = (0:.1:2*pi)';
y0 = sin(xdata);
```

Add noise to the signal:

```
noise = 2*y0.*randn(size(y0)); % Response-dependent noise
ydata = y0 + noise;
```

Fit the noisy data with a custom sinusoidal model:

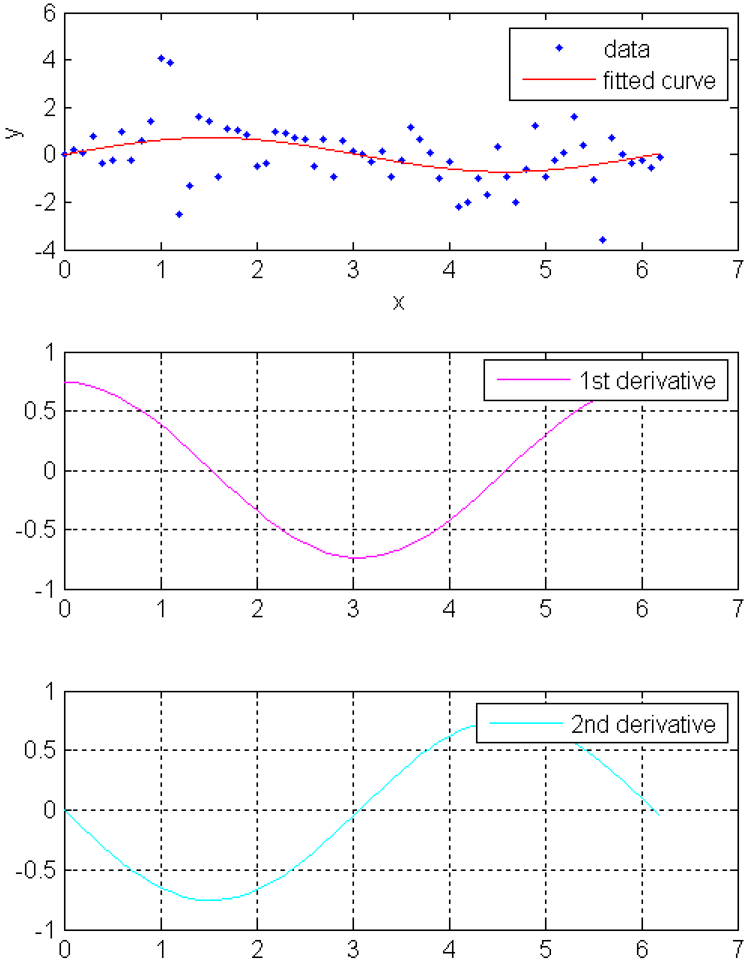
```
f = fittype('a*sin(b*x)');
fit1 = fit(xdata,ydata,f,'StartPoint',[1 1]);
```

Find the derivatives of the fit at the predictors:

```
[d1,d2] = differentiate(fit1,xdata);
```

Plot the data, the fit, and the derivatives:

```
subplot(3,1,1)
plot(fit1,xdata,ydata) % cfit plot method
subplot(3,1,2)
plot(xdata,d1,'m') % double plot method
grid on
legend('1st derivative')
subplot(3,1,3)
plot(xdata,d2,'c') % double plot method
grid on
legend('2nd derivative')
```



Note that derivatives can also be computed and plotted directly with the `cfit` `plot` method, as follows:

```
plot(fit1,xdata,ydata,{'fit','deriv1','deriv2'})
```

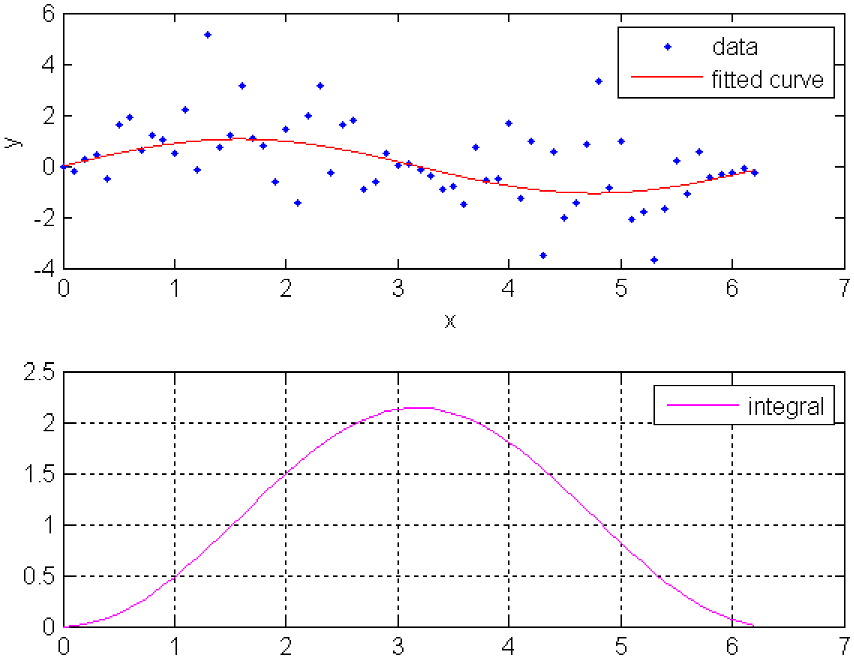
The `plot` method, however, does not return data on the derivatives.

Find the integral of the fit at the predictors:

```
int = integrate(fit1,xdata,0);
```

Plot the data, the fit, and the integral:

```
subplot(2,1,1)
plot(fit1,xdata,ydata) % cfit plot method
subplot(2,1,2)
plot(xdata,int,'m') % double plot method
grid on
legend('integral')
```



Note that integrals can also be computed and plotted directly with the `cfit` `plot` method, as follows:

```
plot(fit1,xdata,ydata,{'fit','integral'})
```

The `plot` method, however, does not return data on the integral.

Example: Prediction Intervals

Generate data with an exponential trend:

```
x = (0:0.2:5)';
y = 2*exp(-0.2*x) + 0.5*randn(size(x));
```

Fit the data using a single-term exponential:

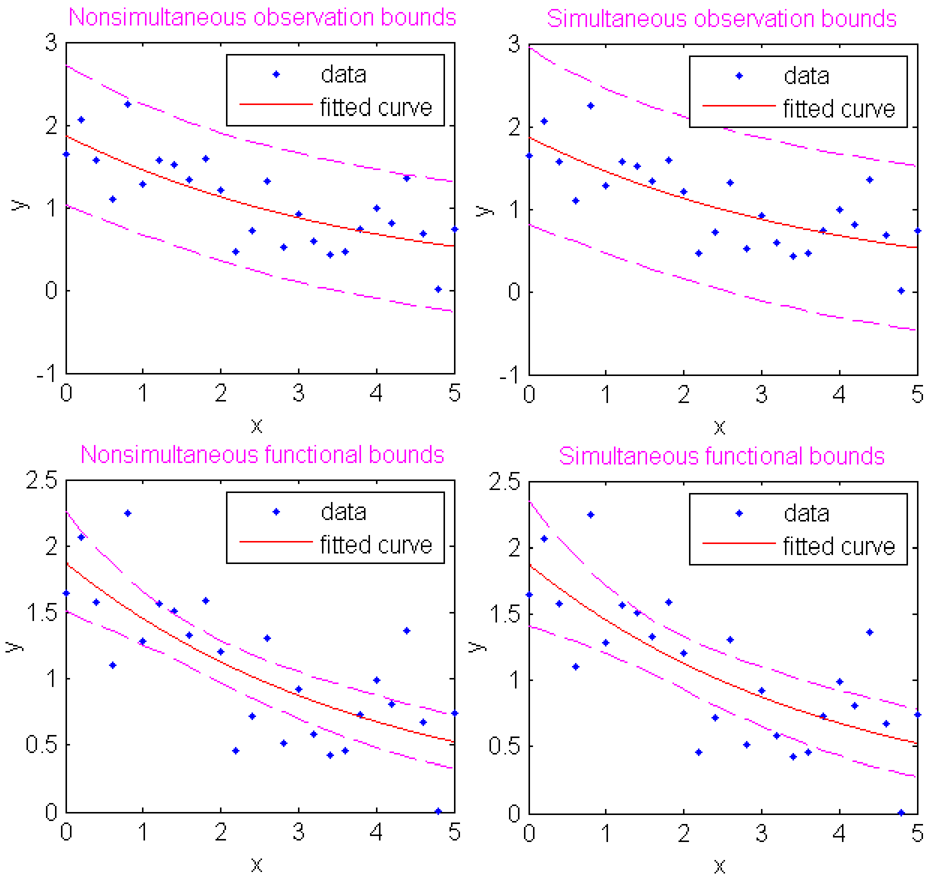
```
fitresult = fit(x,y,'exp1');
```

Compute prediction intervals:

```
p11 = predint(fitresult,x,0.95,'observation','off');
p12 = predint(fitresult,x,0.95,'observation','on');
p21 = predint(fitresult,x,0.95,'functional','off');
p22 = predint(fitresult,x,0.95,'functional','on');
```

Plot the data, fit, and prediction intervals:

```
subplot(2,2,1)
plot(fitresult,x,y), hold on, plot(x,p11,'m--'), xlim([0 5])
title('Nonsimultaneous observation bounds','Color','m')
subplot(2,2,2)
plot(fitresult,x,y), hold on, plot(x,p12,'m--'), xlim([0 5])
title('Simultaneous observation bounds','Color','m')
subplot(2,2,3)
plot(fitresult,x,y), hold on, plot(x,p21,'m--'), xlim([0 5])
title('Nonsimultaneous functional bounds','Color','m')
subplot(2,2,4)
plot(fitresult,x,y), hold on, plot(x,p22,'m--'), xlim([0 5])
title('Simultaneous functional bounds','Color','m')
```


Nonsimultaneous observation bounds
Simultaneous observation bounds
Nonsimultaneous functional bounds
Simultaneous functional bounds
data
fitted curve
x
y
-1
0
1
2
3
0.5
1.5
2.5
4
5

Interactive Code Generation

In this section...

This section describes how to generate and use MATLAB code from an interactive session in Curve Fitting Tool.

Overview

One way to quickly assemble curve fitting objects and methods into useful programs is to generate an M-file from a session in Curve Fitting Tool. In this way, interactive analysis of a single data set is transformed into a reusable function for batch processing of multiple data sets. The generated M-file can be used without modification, or it can be edited and customized as needed.

To generate an M-file from a session in Curve Fitting Tool, select the menu item **File > Generate M-file**.

The M-file captures the following information from Curve Fitting Tool:

- Names of variables, fits, and residuals
- Fit options, such as whether the data should be normalized, initial values for the coefficients, and the fitting method
- Curve fitting objects and methods used to create the fit

You can recreate your Curve Fitting Tool session by calling the M-file from the command line with your original data as input arguments. You can also call the M-file with new data, applying the assembled curve fitting methods to re-compute curve fitting objects.

The Generated M-file

M-files generated from Curve Fitting Tool are constructed from building-block components of code, which you can analyze, modify, and reuse in your own M-files. The components of the generated M-file provide good examples of how to assemble curve fitting objects and methods to perform basic tasks. The larger M-file shows you how to assemble those tasks into a complete analysis of your data.

For example, the following M-file was generated from a session in Curve Fitting Tool that imported the data from `census.mat` and fit a custom nonlinear model of the form $y = a(x–b)^3$:

```
function myfit(cdate,pop)
%MYFIT    Create plot of datasets and fits
%   MYFIT(CDATE,POP)
%   Creates a plot, similar to the plot in the main curve fitting
%   window, using the data that you provide as input.  You can
%   apply this function to the same data you used with cftool
%   or with different data.  You may want to edit the function to
%   customize the code and this help message.
%
%   Number of datasets:  1
%   Number of fits:  1


% Data from dataset "pop vs. cdate":
%    X = cdate:
%    Y = pop:
%    Unweighted
%
% This function was automatically generated on 11-Sep-2007 01:07:11

% Set up figure to receive datasets and fits
f_ = clf;
figure(f_);
set(f_,'Units','Pixels','Position',[439.6 193.6 814.4 576.8]);
legh_ = []; legt_ = {};   % handles and text for legend
xlim_ = [Inf -Inf];       % limits of x axis
ax_ = axes;
set(ax_,'Units','normalized','OuterPosition',[0 0 1 1]);
set(ax_,'Box','on');
axes(ax_); hold on;


% --- Plot data originally in dataset "pop vs. cdate"
cdate = cdate(:);
pop = pop(:);
h_ = line(cdate,pop,'Parent',ax_,'Color',[0.333333 0 0.666667],...
     'LineStyle','none', 'LineWidth',1,...
     'Marker','.', 'MarkerSize',12);
xlim_(1) = min(xlim_(1),min(cdate));
xlim_(2) = max(xlim_(2),max(cdate));
legh_(end+1) = h_;
legt_{end+1} = 'pop vs. cdate';

% Nudge axis limits beyond data limits
if all(isfinite(xlim_))
   xlim_ = xlim_ + [-1 1] * 0.01 * diff(xlim_);
   set(ax_,'XLim',xlim_)
else
    set(ax_, 'XLim',[1788, 1992]);
end


% --- Create fit "fit 1"
ok_ = isfinite(cdate) & isfinite(pop);
if ~all( ok_ )
    warning( 'GenerateMFile:IgnoringNansAndInfs', ...
        'Ignoring NaNs and Infs in data' );
end
st_ = [0.51510504095942344 0.35210694524343056 ];
ft_ = fittype('a*(x-b)^3',...
     'dependent',{'y'},'independent',{'x'},...
     'coefficients',{'a', 'b'});

% Fit this model using new data
cf_ = fit(cdate(ok_),pop(ok_),ft_,'Startpoint',st_);

% Or use coefficients from the original fit:
if 0
```

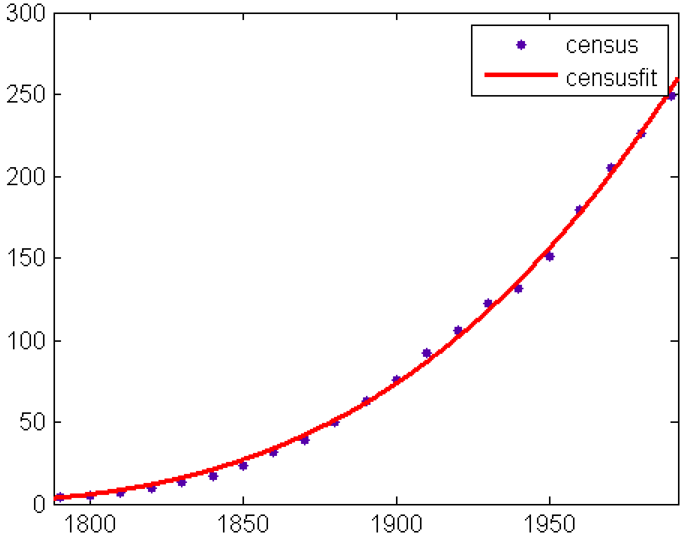
```
  cv_ = { 1.3594203554767276e-005, 1724.6959436137356};
  cf_ = cfit(ft_,cv_{:});
end

% Plot this fit
h_ = plot(cf_,'fit',0.95);
legend off;  % turn off legend from plot method call
set(h_(1),'Color',[1 0 0],...
   'LineStyle','-', 'LineWidth',2,...
   'Marker','none', 'MarkerSize',6);
legh_(end+1) = h_(1);
legt_{end+1} = 'fit 1';

% Done plotting data and fits.  Now finish up loose ends.
hold off;
leginfo_ = {'Orientation', 'vertical', 'Location', 'NorthEast'};
h_ = legend(ax_,legh_,legt_,leginfo_{:});  % create legend
set(h_,'Interpreter','none');
xlabel(ax_,'');               % remove x label
ylabel(ax_,'');               % remove y label
```

A quick look through the code shows that it has automatically assembled for you many of the Curve Fitting Toolbox curve fitting methods, such as `fitoptions`, `fittype`, `fit`, and `plot`.

Running the Generated M-file

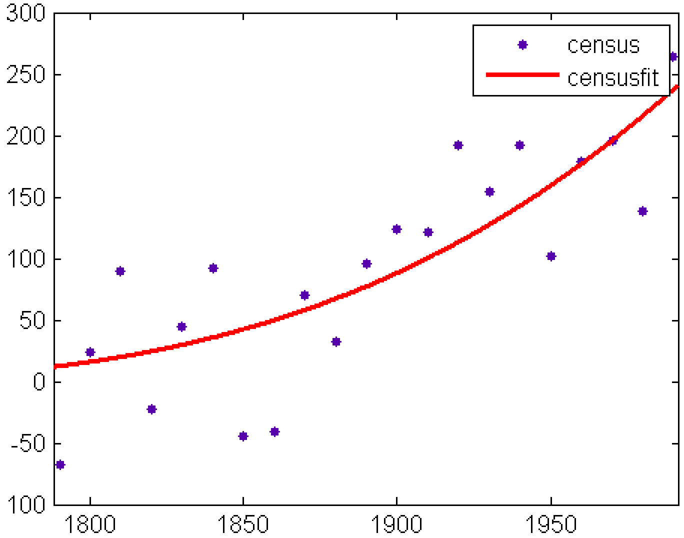
To run the generated M-file without modification, and reproduce your original Curve Fitting Tool session, type:

```
load census
myfit(cdate,pop)
```



To run the M-file without modification on new data, pass the new data to the function as input arguments:

```
newpop = pop + 50*randn(size(pop));
myfit(cdate,newpop)
```



The M-file recomputes the cfit object for the fit and displays the new data with the new fit.

Components of the Generated M-File

It is useful to take a closer look at the components of the generated M-file, to understand the role that each component plays in the overall visualization and analysis of the data. This allows you to change the M-file, and customize it to your needs.

The M-file begins with a function declaration:

```
function myfit(cdate,pop)
```

The function accepts predictor and response data for a predefined fit type. The inputs are called cdate and pop because those were the predictor and response variables used in Curve Fitting Tool session that produced the file. If you like, you can find and replace the input names here and elsewhere in the file to indicate a more generic application of the fit.

Note that the file, as generated, returns no outputs. It simply applies the fit to the input data and displays the results.

The next component of the M-file, after the help information, is the following:

```
% Set up figure to receive datasets and fits
f_ = clf;
figure(f_);
set(f_,'Units','Pixels','Position',[439.6 193.6 814.4 576.8]);
legh_ = []; legt_ = {};   % handles and text for legend
xlim_ = [Inf -Inf];       % limits of x axis
ax_ = axes;
set(ax_,'Units','normalized','OuterPosition',[0 0 1 1]);
set(ax_,'Box','on');
axes(ax_); hold on;
```

These are Handle Graphics® methods, applied to Handle Graphics objects that encapsulate information on the display of the figure window, the legend, and the axes. This component of the M-file creates a figure for plotting that mimics the Plotting GUI in Curve Fitting Tool. Note that at the end of this component hold is toggled on. This allows the input data and the fit to be plotted together on the axes.

The next component of the M-file plots the input data, using Handle Graphics methods to set properties of the line object, the axes, and the legend that mimic the plot in Curve Fitting Tool:

```
% --- Plot data originally in dataset "pop vs. cdate"
cdate = cdate(:);
pop = pop(:);
h_ = line(cdate,pop,'Parent',ax_,'Color',[0.333333 0 0.666667],...
     'LineStyle','none', 'LineWidth',1,...
     'Marker','.', 'MarkerSize',12);
xlim_(1) = min(xlim_(1),min(cdate));
xlim_(2) = max(xlim_(2),max(cdate));
legh_(end+1) = h_;
legt_{end+1} = 'pop vs. cdate';
```

The next component “nudges” the *x*-axis limits, leaving a space of 1% of the *x* data range between the data and the vertical axes. This gives a tight plot, while preventing data from being plotted directly onto the vertical axes, where it would be difficult to see.

```
% Nudge axis limits beyond data limits
if all(isfinite(xlim_))
   xlim_ = xlim_ + [-1 1] * 0.01 * diff(xlim_);
   set(ax_,'XLim',xlim_)
else
    set(ax_, 'XLim',[1788, 1992]);
end
```

After all of the preliminaries, the M-file gets down to the business of fitting the data. The next component of the M-file uses fitoptions and fittype to create a fit options structure fo_ and a fittype object ft_ that encapsulate, respectively, information on the fitting method and the model. The inputs to fitoptions and fittype are read from the Fitting GUI in Curve Fitting Tool.

```
% --- Create fit "fit 1"
ok_ = isfinite(cdate) & isfinite(pop);
if ~all( ok_ )
    warning( 'GenerateMFile:IgnoringNansAndInfs', ...
        'Ignoring NaNs and Infs in data' );
end
st_ = [0.51510504095942344 0.35210694524343056 ];
ft_ = fittype('a*(x-b)^3',...
     'dependent',{'y'},'independent',{'x'},...
     'coefficients',{'a', 'b'});
```

The fit method is then called to fit the predefined fit type to the input data. Note that NaNs are removed from the data before the fit, using the logical vector ok_ defined in the previous component.

```
% Fit this model using new data
cf_ = fit(cdate(ok_),pop(ok_),ft_,'Startpoint',st_);
```

The next component of the M-file is a little obscure, since it uses a conditional with a guard condition that is always false (0). This code is generated intentionally, to give you the option of plotting the new input data against a fit based on the old data (the data that was originally imported into Curve Fitting Tool). To do so, simply change the 0 to true. The modified M-file then uses the cfit method to set the coefficients of the cfit object cf_ to the stored values computed with the old data. If you do not wish to do this, leave this component of the M-file alone, or delete it.

```
% Or use coefficients from the original fit:
if 0
   cv_ = { 1.3594203554767276e-005, 1724.6959436137356};
   cf_ = cfit(ft_,cv_{:});
end
```

With the fitting complete, the M-file calls the plot method to plot the cfit object cf_. Note that plot is called with the default plot type 'fit' (data and fit), but is also passed a confidence level of 0.95. To use this confidence level to plot prediction bounds for the fit or for new observations, change 'fit' to 'predfunc' or 'predobs', respectively. The rest of the code in this component of the M-file is more Handle Graphics, along the lines of previous components, setting Handle Graphics object properties that mimic the plot of the fit in Curve Fitting Tool.

```
% Plot this fit
h_ = plot(cf_,'fit',0.95);
legend off;  % turn off legend from plot method call
```

```
set(h_(1),'Color',[1 0 0],...
     'LineStyle','-', 'LineWidth',2,...
     'Marker','none', 'MarkerSize',6);
legh_(end+1) = h_(1);
legt_{end+1} = 'fit 1';
```

Finally, the M-file takes care of “loose ends”: `hold` is toggled `off` to its default behavior, the legend is positioned, and the *x* and *y* labels (`'x'` and `'y'` by default) are removed.

```
% Done plotting data and fits.  Now finish up loose ends.
hold off;
leginfo_ = {'Orientation', 'vertical', 'Location', 'NorthEast'};
h_ = legend(ax_,legh_,legt_,leginfo_{:});  % create legend
set(h_,'Interpreter','none');
xlabel(ax_,'');              % remove x label
ylabel(ax_,'');              % remove y label
```

Modifying the Code

With an understanding of the components of the generated M-file, it is easy to modify the code to produce a customized curve fit and display. The basic structure of the M-file is already in place for you, and you can concentrate on the details that interest you most.

A natural modification of the M-file would be to edit the function declaration at the top of the file to return the `cfit` object `cf_` created by the fit, as follows:

```
function cf_ = myfit2(cdate,pop)
```

Note the change in the function name from `myfit` to `myfit2`. The modified M-file should then be saved to a file named `myfit2.m`.

You might also want to return goodness-of-fit statistics, which the M-file, by default, does not compute. You would have to modify both the call to `fit`:

```
[cf_,gof] = fit(cdate(ok_),pop(ok_),ft_,fo_);
```

and the function declaration:

```
function [cf_,gof] = myfit2(cdate,pop)
```

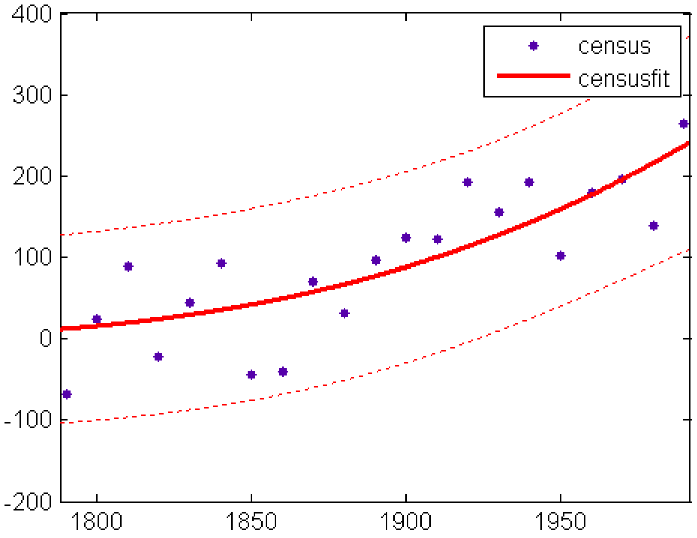
You might also want to alter the call to `plot`, say to show prediction intervals for new observations:

```
h_ = plot(cf_,'predobs',0.95);
```

Running the M-file with the above modifications on the new data from "Running the Generated M-file" on page 3-26 produces the following output to the command line:

```
[c,g] = myfit2(cdate,newpop)
c =
     General model:
       c(x) = a*(x-b)^3
     Coefficients (with 95% confidence bounds):
       a =  7.211e-006  (-2.389e-006, 1.681e-005)
       b =        1670  (1548, 1792)
g =
           sse: 5.5691e+004
       rsquare: 0.6561
           dfe: 19
    adjrsquare: 0.6380
          rmse: 54.1398
```

and the following display:


400
300
200
100
0
-100
-200
1800
1850
1900
1950
census
censusfit

4

Curve Fitting Techniques

Data Transformations

Changing variables through data transformations may lead to a simplified relationship between the transformed predictor variable and the transformed response. As a result, model descriptions and predictions may be simplified.

Common transformations include the logarithm ln(y), and power functions such as $y^{1/2}$, y^{-1}, and so on. Using these transformations, you can linearize a nonlinear model, contract response data that spans one or more orders of magnitude, or simplify a model so that it involves fewer coefficients.

Note You must transform variables at the MATLAB command line, and then import those variables into Curve Fitting Tool. You cannot transform variables using any of the graphical user interfaces.

For example, suppose you want to use the following model to fit your data.

$$y = \frac{1}{ax^2 + bx + c}$$

If you decide to use the power transform y^{-1}, then the transformed model is given by

$$y^{-1} = ax^2 + bx + c$$

As another example, the equation

$$y = ae^{bx}$$

becomes linear if you take the log transform of both sides.

$$\ln(y) = \ln(a) + bx$$

You can now use linear least-squares fitting procedures.

There are several disadvantages associated with performing transformations:

- For the log transformation, negative response values cannot be processed.
- For all transformations, the basic assumption that the residual variance is constant is violated. To avoid this problem, you could plot the residuals on the transformed scale. For the power transformation shown above, the transformed scale is given by the residuals

  $$r_i = y_i^{-1} - \hat{y}_i^{-1}$$

  Note that the residual plot associated with Curve Fitting Tool does not support transformed scales.

Deciding on a particular transformation is not always obvious. However, a scatter plot will often reveal the best form to use. In practice you can experiment with various transforms and then plot the residuals from the command line using the transformed scale. If the errors are reasonable (they appear random with minimal scatter, and don't exhibit any systematic behavior), the transform is a good candidate.

Filtering and Smoothing

In this section...

Moving Average Filtering

A moving average filter smooths data by replacing each data point with the average of the neighboring data points defined within the span. This process is equivalent to lowpass filtering with the response of the smoothing given by the difference equation

$$y_s(i) = \frac{1}{2N+1}(y(i+N) + y(i+N-1) + \ldots + y(i-N))$$

where $y_s(i)$ is the smoothed value for the `i`th data point, N is the number of neighboring data points on either side of $y_s(i)$, and $2N$+1 is the span.

The moving average smoothing method used by Curve Fitting Toolbox software follows these rules:

- The span must be odd.
- The data point to be smoothed must be at the center of the span.
- The span is adjusted for data points that cannot accommodate the specified number of neighbors on either side.
- The end points are not smoothed because a span cannot be defined.

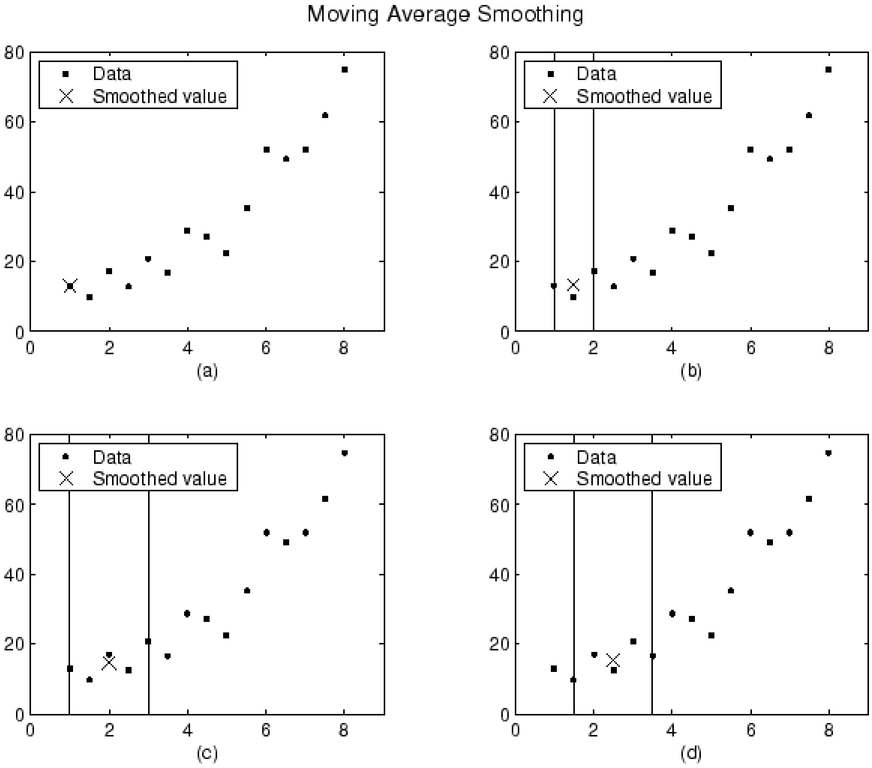
Note that you can use `filter` function to implement difference equations such as the one shown above. However, because of the way that the end points are treated, the toolbox moving average result will differ from the result returned by `filter`. Refer to Difference Equations and Filtering in the MATLAB documentation for more information.

For example, suppose you smooth data using a moving average filter with a span of 5. Using the rules described above, the first four elements of y_s are given by

$$
\begin{aligned}
y_s(1) &= y(1)\\
y_s(2) &= (y(1)+y(2)+y(3))/3\\
y_s(3) &= (y(1)+y(2)+y(3)+y(4)+y(5))/5\\
y_s(4) &= (y(2)+y(3)+y(4)+y(5)+y(6))/5
\end{aligned}
$$

Note that $y_s(1)$, $y_s(2)$, ... ,$y_s(\mathrm{end})$ refer to the order of the data after sorting, and not necessarily the original order.

The smoothed values and spans for the first four data points of a generated data set are shown below.



Plot (a) indicates that the first data point is not smoothed because a span cannot be constructed. Plot (b) indicates that the second data point is smoothed using a span of three. Plots (c) and (d) indicate that a span of five is used to calculate the smoothed value.

Savitzky-Golay Filtering

Savitzky-Golay filtering can be thought of as a generalized moving average. You derive the filter coefficients by performing an unweighted linear least-squares fit using a polynomial of a given degree. For this reason, a Savitzky-Golay filter is also called a digital smoothing polynomial filter or a least-squares smoothing filter. Note that a higher degree polynomial makes it possible to achieve a high level of smoothing without attenuation of data features.

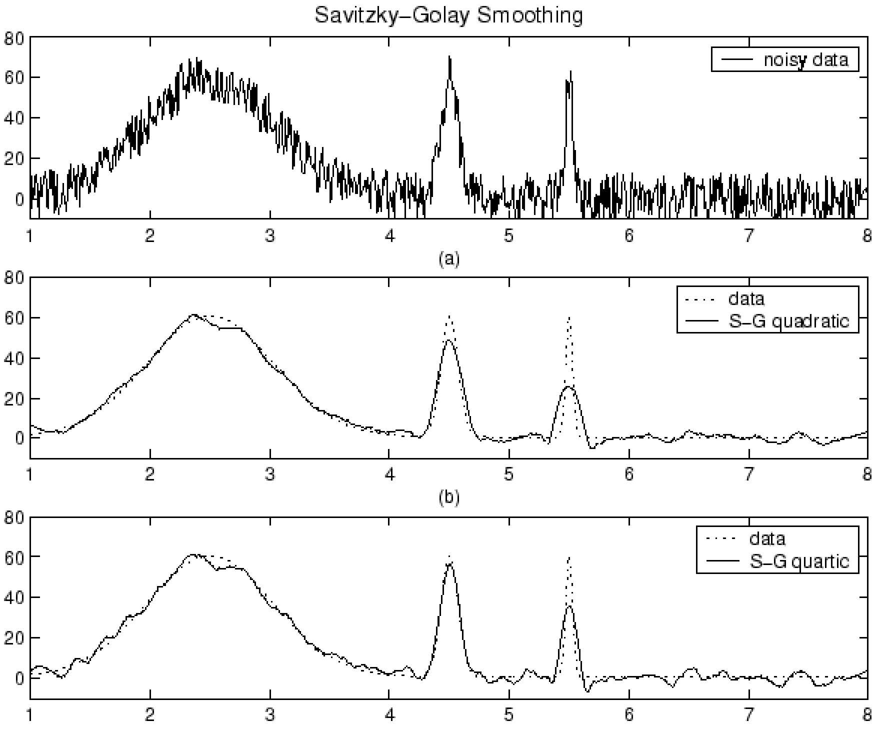
The Savitzky-Golay filtering method is often used with frequency data or with spectroscopic (peak) data. For frequency data, the method is effective at preserving the high-frequency components of the signal. For spectroscopic data, the method is effective at preserving higher moments of the peak such as the line width. By comparison, the moving average filter tends to filter out a significant portion of the signal's high-frequency content, and it can only preserve the lower moments of a peak such as the centroid. However, Savitzky-Golay filtering can be less successful than a moving average filter at rejecting noise.

The Savitzky-Golay smoothing method used by Curve Fitting Toolbox software follows these rules:

- The span must be odd.
- The polynomial degree must be less than the span.
- The data points are not required to have uniform spacing.

  Normally, Savitzky-Golay filtering requires uniform spacing of the predictor data. However, the Curve Fitting Toolbox algorithm supports nonuniform spacing. Therefore, you are not required to perform an additional filtering step to create data with uniform spacing.

The plot shown below displays generated Gaussian data and several attempts at smoothing using the Savitzky-Golay method. The data is very noisy and the peak widths vary from broad to narrow. The span is equal to 5% of the number of data points.



(a)

(b)

(c)

Plot (a) shows the noisy data. To more easily compare the smoothed results, plots (b) and (c) show the data without the added noise.

Plot (b) shows the result of smoothing with a quadratic polynomial. Notice that the method performs poorly for the narrow peaks. Plot (c) shows the result of smoothing with a quartic polynomial. In general, higher degree polynomials can more accurately capture the heights and widths of narrow peaks, but can do poorly at smoothing wider peaks.

Local Regression Smoothing

- “Lowess and Loess” on page 4-8

- "The Local Regression Method" on page 4-8
- "Robust Local Regression" on page 4-11

Lowess and Loess

The names "lowess" and "loess" are derived from the term "locally weighted scatter plot smooth," as both methods use locally weighted linear regression to smooth data.

The smoothing process is considered local because, like the moving average method, each smoothed value is determined by neighboring data points defined within the span. The process is weighted because a regression weight function is defined for the data points contained within the span. In addition to the regression weight function, you can use a robust weight function, which makes the process resistant to outliers. Finally, the methods are differentiated by the model used in the regression: lowess uses a linear polynomial, while loess uses a quadratic polynomial.

The local regression smoothing methods used by Curve Fitting Toolbox software follow these rules:

- The span can be even or odd.
- You can specify the span as a percentage of the total number of data points in the data set. For example, a span of 0.1 uses 10% of the data points.

The Local Regression Method

The local regression smoothing process follows these steps for each data point:

1. Compute the *regression weights* for each data point in the span. The weights are given by the tricube function shown below.

$$w_i = \left(1 - \left|\frac{x - x_i}{d(x)}\right|^3\right)^3$$

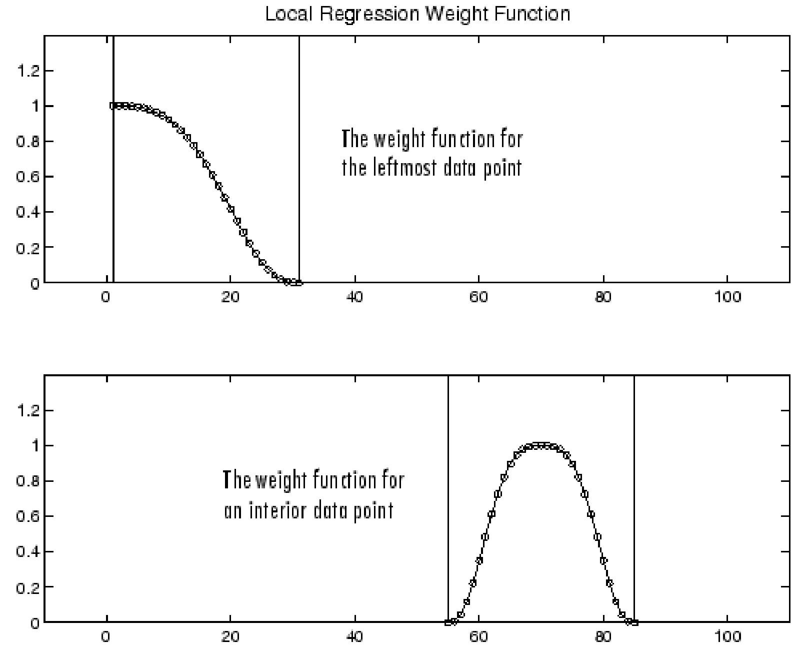
   x is the predictor value associated with the response value to be smoothed, x_i are the nearest neighbors of x as defined by the span, and $d(x)$ is the distance along the abscissa from x to the most distant predictor value within the span. The weights have these characteristics:

- The data point to be smoothed has the largest weight and the most influence on the fit.
- Data points outside the span have zero weight and no influence on the fit.

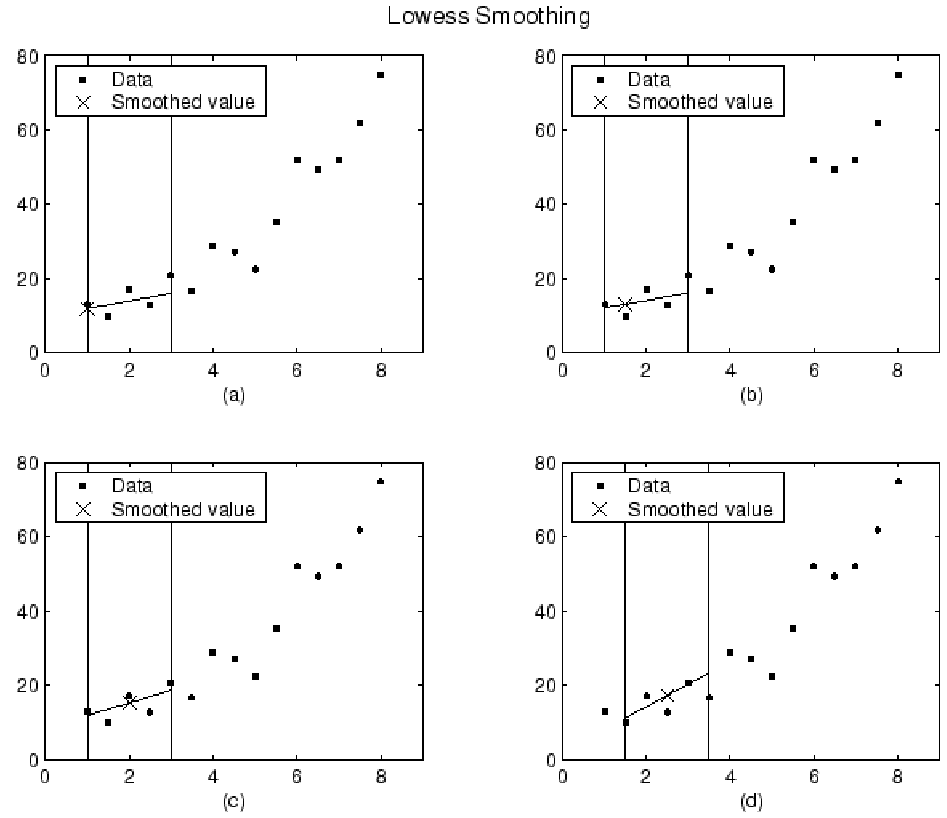
2. A weighted linear least-squares regression is performed. For lowess, the regression uses a first degree polynomial. For loess, the regression uses a second degree polynomial.
3. The smoothed value is given by the weighted regression at the predictor value of interest.

If the smooth calculation involves the same number of neighboring data points on either side of the smoothed data point, the weight function is symmetric. However, if the number of neighboring points is not symmetric about the smoothed data point, then the weight function is not symmetric. Note that unlike the moving average smoothing process, the span never changes. For example, when you smooth the data point with the smallest predictor value, the shape of the weight function is truncated by one half, the leftmost data point in the span has the largest weight, and all the neighboring points are to the right of the smoothed value.

The weight function for an end point and for an interior point is shown below for a span of 31 data points.



Using the lowess method with a span of five, the smoothed values and associated regressions for the first four data points of a generated data set are shown below.



Notice that the span does not change as the smoothing process progresses from data point to data point. However, depending on the number of nearest neighbors, the regression weight function might not be symmetric about the data point to be smoothed. In particular, plots `(a)` and `(b)` use an asymmetric weight function, while plots `(c)` and `(d)` use a symmetric weight function.

For the loess method, the graphs would look the same except the smoothed value would be generated by a second-degree polynomial.

Robust Local Regression

If your data contains outliers, the smoothed values can become distorted, and not reflect the behavior of the bulk of the neighboring data points. To overcome this problem, you can smooth the data using a robust procedure that is not influenced by a small fraction of outliers. For a description of outliers, refer to “Marking Outliers” on page 2-19.

Curve Fitting Toolbox software provides a robust version for both the lowess and loess smoothing methods. These robust methods include an additional calculation of robust weights, which is resistant to outliers. The robust smoothing procedure follows these steps:

1. Calculate the residuals from the smoothing procedure described in the previous section.

2. Compute the *robust weights* for each data point in the span. The weights are given by the bisquare function shown below.

$$w_i = \begin{cases} (1 - (r_i/6MAD)^2)^2 & |r_i| < 6MAD \\ 0 & |r_i| \geq 6MAD \end{cases}$$

   r_i is the residual of the ith data point produced by the regression smoothing procedure, and *MAD* is the median absolute deviation of the residuals:

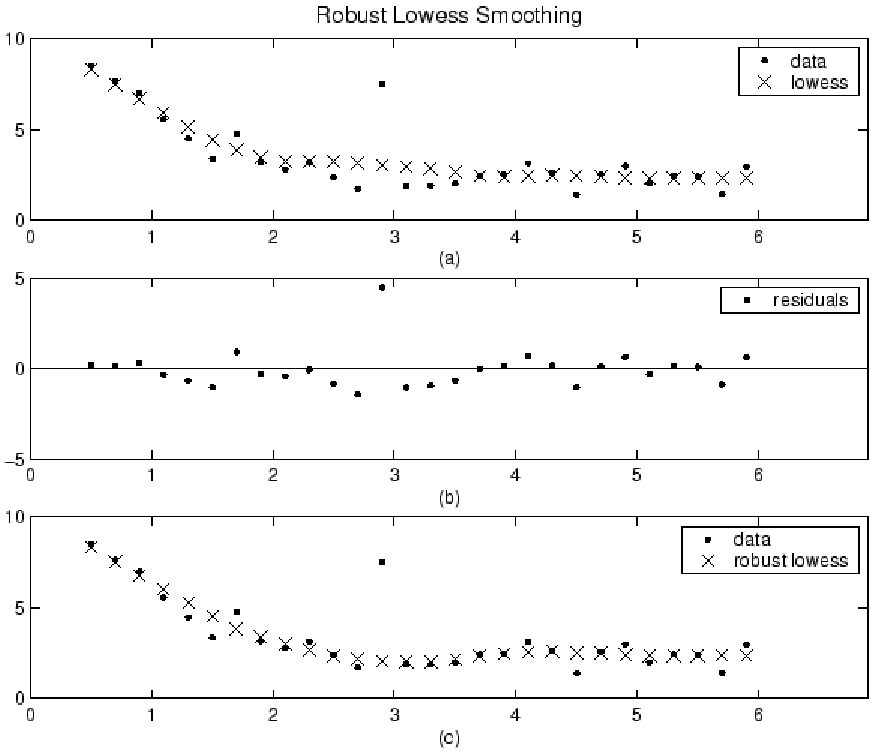
$$MAD = \text{median}(|r|)$$

   The median absolute deviation is a measure of how spread out the residuals are. If r_i is small compared to 6*MAD*, then the robust weight is close to 1. If r_i is greater than 6*MAD*, the robust weight is 0 and the associated data point is excluded from the smooth calculation.

3. Smooth the data again using the robust weights. The final smoothed value is calculated using both the local regression weight and the robust weight.

4. Repeat the previous two steps for a total of five iterations.

The smoothing results of the lowess procedure are compared below to the results of the robust lowess procedure for a generated data set that contains a single outlier. The span for both procedures is 11 data points.



(a)

(b)

(c)

Plot (a) shows that the outlier influences the smoothed value for several nearest neighbors. Plot (b) suggests that the residual of the outlier is greater than six median absolute deviations. Therefore, the robust weight is zero for this data point. Plot (c) shows that the smoothed values neighboring the outlier reflect the bulk of the data.

Smoothing Splines

If your data is noisy, you might want to fit it using a smoothing spline. Alternatively, you can use one of the smoothing methods described in "Smoothing Data" on page 2-9.

The smoothing spline s is constructed for the specified *smoothing parameter p* and the specified weights w_i. The smoothing spline minimizes

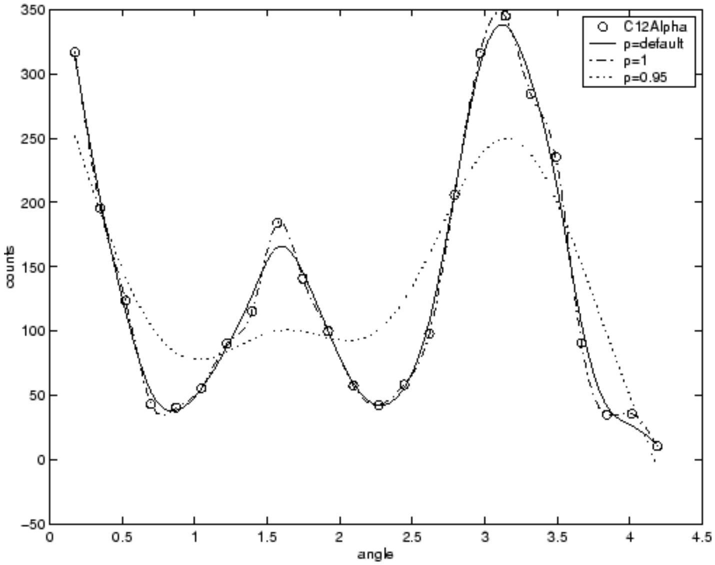
$$p\sum_i w_i(y_i - s(x_i))^2 + (1-p)\int\left(\frac{d^2 s}{dx^2}\right)^2 dx$$

If the weights are not specified, they are assumed to be 1 for all data points.

p is defined between 0 and 1. $p = 0$ produces a least-squares straight-line fit to the data, while $p = 1$ produces a cubic spline interpolant. If you do not specify the smoothing parameter, it is automatically selected in the "interesting range." The interesting range of *p* is often near $1/(1+h^3/6)$ where *h* is the average spacing of the data points, and it is typically much smaller than the allowed range of the parameter. Because smoothing splines have an associated smoothing parameter, you might consider these fits to be parametric in that sense. However, smoothing splines are also piecewise polynomials like cubic spline or shape-preserving interpolants and are considered a nonparametric fit type in this guide.

Note The Curve Fitting Toolbox smoothing spline algorithm is based on the Spline Toolbox™ `csaps` function.

The nuclear reaction data from the file `carbon12alpha.mat` is shown below with three smoothing spline fits. The default smoothing parameter ($p = 0.99$) produces the smoothest curve. The cubic spline curve ($p = 1$) goes through all the data points, but is not quite as smooth. The third curve ($p = 0.95$) misses the data by wide margin and illustrates how small the "interesting range" of *p* can be.


C12Alpha
p=default
p=1
p=0.95
counts
angle
350
300
250
200
150
100
50
0
−50
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5

Least-Squares Fitting

In this section...

Introduction

Curve Fitting Toolbox software uses the method of least squares when fitting data. Fitting requires a parametric model that relates the response data to the predictor data with one or more coefficients. The result of the fitting process is an estimate of the model coefficients.

To obtain the coefficient estimates, the least-squares method minimizes the summed square of residuals. The residual for the *i*th data point r_i is defined as the difference between the observed response value y_i and the fitted response value $\hat{y}_i$, and is identified as the error associated with the data.

$$r_i = y_i - \hat{y}_i$$

$$\text{residual} = \text{data} - \text{fit}$$

The summed square of residuals is given by

$$S = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where *n* is the number of data points included in the fit and *S* is the sum of squares error estimate. The supported types of least-squares fitting include:

- Linear least squares

- Weighted linear least squares
- Robust least squares
- Nonlinear least squares

Error Distributions

When fitting data that contains random variations, there are two important assumptions that are usually made about the error:

- The error exists only in the response data, and not in the predictor data.
- The errors are random and follow a normal (Gaussian) distribution with zero mean and constant variance, σ^2.

The second assumption is often expressed as

$$\mathrm{error} \sim N(0, \sigma^2)$$

The errors are assumed to be normally distributed because the normal distribution often provides an adequate approximation to the distribution of many measured quantities. Although the least-squares fitting method does not assume normally distributed errors when calculating parameter estimates, the method works best for data that does not contain a large number of random errors with extreme values. The normal distribution is one of the probability distributions in which extreme random errors are uncommon. However, statistical results such as confidence and prediction bounds do require normally distributed errors for their validity.

If the mean of the errors is zero, then the errors are purely random. If the mean is not zero, then it might be that the model is not the right choice for your data, or the errors are not purely random and contain systematic errors.

A constant variance in the data implies that the “spread” of errors is constant. Data that has the same variance is sometimes said to be of *equal quality*.

The assumption that the random errors have constant variance is not implicit to weighted least-squares regression. Instead, it is assumed that the weights provided in the fitting procedure correctly indicate the differing levels of quality present in the data. The weights are then used to adjust the amount of influence each data point has on the estimates of the fitted coefficients to an appropriate level.

Linear Least Squares

Curve Fitting Toolbox software uses the linear least-squares method to fit a linear model to data. A linear model is defined as an equation that is linear in the coefficients. For example, polynomials are linear but Gaussians are not. To illustrate the linear least-squares fitting process, suppose you have *n* data points that can be modeled by a first-degree polynomial.

$$y = p_1 x + p_2$$

To solve this equation for the unknown coefficients p_1 *and* p_2, you write S as a system of *n* simultaneous linear equations in two unknowns. If *n* is greater than the number of unknowns, then the system of equations is *overdetermined.*

$$S = \sum_{i=1}^{n} (y_i - (p_1 x_i + p_2))^2$$

Because the least-squares fitting process minimizes the summed square of the residuals, the coefficients are determined by differentiating S with respect to each parameter, and setting the result equal to zero.

$$\frac{\partial S}{\partial p_1} = -2 \sum_{i=1}^{n} x_i (y_i - (p_1 x_i + p_2)) = 0$$

$$\frac{\partial S}{\partial p_2} = -2 \sum_{i=1}^{n} (y_i - (p_1 x_i + p_2)) = 0$$

The estimates of the true parameters are usually represented by *b*. Substituting b_1 and b_2 for p_1 and p_2, the previous equations become

$$\sum x_i(y_i - (b_1 x_i + b_2)) = 0$$
$$\sum (y_i - (b_1 x_i + b_2)) = 0$$

where the summations run from $i = 1$ to n. The *normal equations* are defined as

$$b_1 \sum x_i^2 + b_2 \sum x_i = \sum x_i y_i$$
$$b_1 \sum x_i + n b_2 = \sum y_i$$

Solving for b_1

$$b_1 = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - (\sum x_i)^2}$$

Solving for b_2 using the b_1 value

$$b_2 = \frac{1}{n}(\sum y_i - b_1 \sum x_i)$$

As you can see, estimating the coefficients p_1 and p_2 requires only a few simple calculations. Extending this example to a higher degree polynomial is straightforward although a bit tedious. All that is required is an additional normal equation for each linear term added to the model.

In matrix form, linear models are given by the formula

$$y = X\beta + \varepsilon$$

where

- y is an n-by-*1* vector of responses.
- ß is a m-by-*1* vector of coefficients.
- X is the n-by-m design matrix for the model.
- ε is an n-by-*1* vector of errors.

For the first-degree polynomial, the n equations in two unknowns are expressed in terms of y, X, and ß as

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ . \\ . \\ . \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ . & \\ . & \\ . & \\ x_n & 1 \end{bmatrix} \times \begin{bmatrix} p_1 \\ p_2 \end{bmatrix}$$

The least-squares solution to the problem is a vector b, which estimates the unknown vector of coefficients ß. The normal equations are given by

$$(X^T X)b = X^T y$$

where X^T is the transpose of the design matrix X. Solving for b,

$$b = (X^T X)^{-1} X^T y$$

Use the MATLAB backslash operator (`mldivide`) to solve a system of simultaneous linear equations for unknown coefficients. Because inverting $X^T X$ can lead to unacceptable rounding errors, the backslash operator uses QR decomposition with pivoting, which is a very stable algorithm numerically. Refer to Arithmetic Operators in the MATLAB documentation for more information about the backslash operator and QR decomposition.

You can plug b back into the model formula to get the predicted response values, $\hat{y}$.

$$\hat{y} = Xb = Hy$$

$$H = X(X^T X)^{-1} X^T$$

A hat (circumflex) over a letter denotes an estimate of a parameter or a prediction from a model. The projection matrix *H* is called the hat matrix, because it puts the hat on *y*.

The residuals are given by

$$r = y - \hat{y} = (1 - H)y$$

Weighted Least Squares

It is usually assumed that the response data is of equal quality and, therefore, has constant variance. If this assumption is violated, your fit might be unduly influenced by data of poor quality. To improve the fit, you can use weighted least-squares regression where an additional scale factor (the weight) is included in the fitting process. Weighted least-squares regression minimizes the error estimate

$$S = \sum_{i=1}^{n} w_i (y_i - \hat{y}_i)^2$$

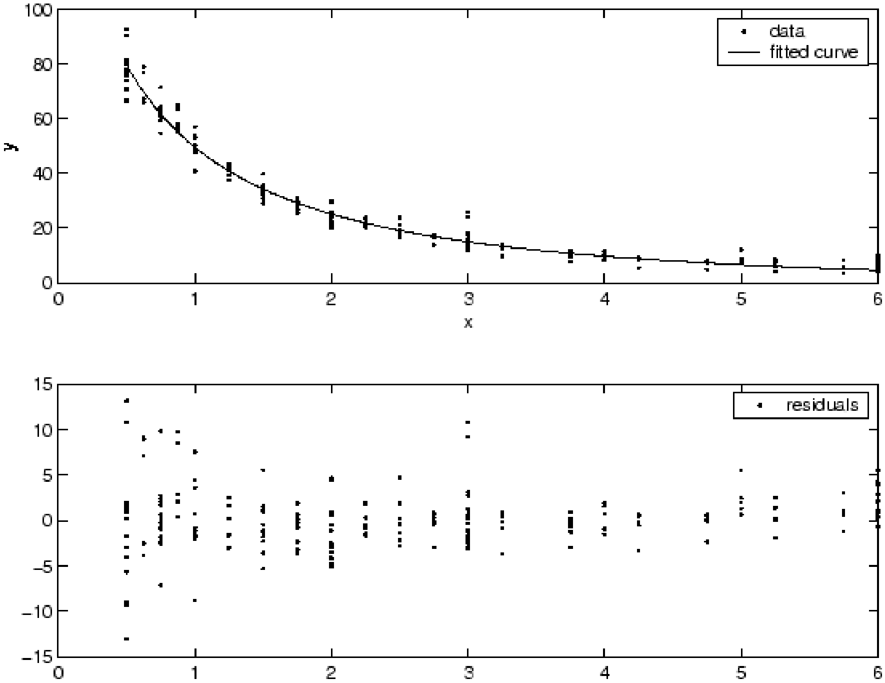
where w_i are the weights. The weights determine how much each response value influences the final parameter estimates. A high-quality data point influences the fit more than a low-quality data point. Weighting your data is recommended if the weights are known, or if there is justification that they follow a particular form.

The weights modify the expression for the parameter estimates *b* in the following way,

$$b = \hat{\beta} = (X^T W X)^{-1} X^T W y$$

where *W* is given by the diagonal elements of the weight matrix *w*.

You can often determine whether the variances are not constant by fitting the data and plotting the residuals. In the plot shown below, the data contains replicate data of various quality and the fit is assumed to be correct. The poor quality data is revealed in the plot of residuals, which has a “funnel” shape where small predictor values yield a bigger scatter in the response values than large predictor values.



The weights you supply should transform the response variances to a constant value. If you know the variances of the measurement errors in your data, then the weights are given by

$$w_i = 1/\sigma_i^2$$

Or, if you only have estimates of the error variable for each data point, it usually suffices to use those estimates in place of the true variance. If you do not know the variances, it suffices to specify weights on a relative scale. Note that an overall variance term is estimated even when weights have been specified. In this instance, the weights define the relative weight to each point in the fit, but are not taken to specify the exact variance of each point.

For example, if each data point is the mean of several independent measurements, it might make sense to use those numbers of measurements as weights.

Robust Least Squares

It is usually assumed that the response errors follow a normal distribution, and that extreme values are rare. Still, extreme values called *outliers* do occur.

The main disadvantage of least-squares fitting is its sensitivity to outliers. Outliers have a large influence on the fit because squaring the residuals magnifies the effects of these extreme data points. To minimize the influence of outliers, you can fit your data using robust least-squares regression. The toolbox provides these two robust regression methods:

- Least absolute residuals (LAR) — The LAR method finds a curve that minimizes the absolute difference of the residuals, rather than the squared differences. Therefore, extreme values have a lesser influence on the fit.
- Bisquare weights — This method minimizes a weighted sum of squares, where the weight given to each data point depends on how far the point is from the fitted line. Points near the line get full weight. Points farther from the line get reduced weight. Points that are farther from the line than would be expected by random chance get zero weight.

  For most cases, the bisquare weight method is preferred over LAR because it simultaneously seeks to find a curve that fits the bulk of the data using the usual least-squares approach, and it minimizes the effect of outliers.

Robust fitting with bisquare weights uses an iteratively reweighted least-squares algorithm, and follows this procedure:

1. Fit the model by weighted least squares.

2. Compute the *adjusted residuals* and standardize them. The adjusted residuals are given by

   $$r_{adj} = \frac{r_i}{\sqrt{1-h_i}}$$

r_i are the usual least-squares residuals and h_i are *leverages* that adjust the residuals by down-weighting high-leverage data points, which have a large effect on the least-squares fit. The standardized adjusted residuals are given by

$$u = \frac{r_{adj}}{Ks}$$

K is a tuning constant equal to 4.685, and s is the robust variance given by *MAD*/0.6745 where *MAD* is the median absolute deviation of the residuals.

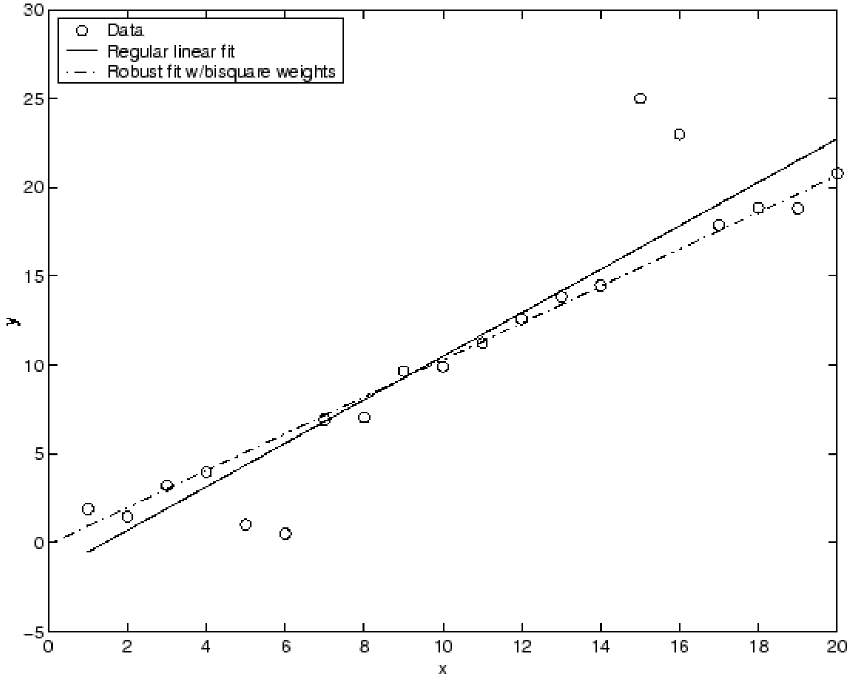
3 Compute the robust weights as a function of u. The bisquare weights are given by

$$w_i = \begin{cases} (1 - (u_i)^2)^2 & |u_i| < 1 \\ 0 & |u_i| \geq 1 \end{cases}$$

Note that if you supply your own regression weight vector, the final weight is the product of the robust weight and the regression weight.

4 If the fit converges, then you are done. Otherwise, perform the next iteration of the fitting procedure by returning to the first step.

The plot shown below compares a regular linear fit with a robust fit using bisquare weights. Notice that the robust fit follows the bulk of the data and is not strongly influenced by the outliers.



Instead of minimizing the effects of outliers by using robust regression, you can mark data points to be excluded from the fit. Refer to "Excluding and Sectioning Data" on page 2-17 for more information.

Nonlinear Least Squares

Curve Fitting Toolbox software uses the nonlinear least-squares formulation to fit a nonlinear model to data. A nonlinear model is defined as an equation that is nonlinear in the coefficients, or a combination of linear and nonlinear in the coefficients. For example, Gaussians, ratios of polynomials, and power functions are all nonlinear.

In matrix form, nonlinear models are given by the formula

$$y = f(X, \beta) + \varepsilon$$

where

- y is an n-by-1 vector of responses.
- f is a function of β and X.
- β is a m-by-1 vector of coefficients.
- X is the n-by-m design matrix for the model.
- ε is an n-by-1 vector of errors.

Nonlinear models are more difficult to fit than linear models because the coefficients cannot be estimated using simple matrix techniques. Instead, an iterative approach is required that follows these steps:

1. Start with an initial estimate for each coefficient. For some nonlinear models, a heuristic approach is provided that produces reasonable starting values. For other models, random values on the interval [0,1] are provided.

2. Produce the fitted curve for the current set of coefficients. The fitted response value $\hat{y}$ is given by

$$\hat{y} = f(X, b)$$

   and involves the calculation of the *Jacobian* of $f(X,b)$, which is defined as a matrix of partial derivatives taken with respect to the coefficients.

3. Adjust the coefficients and determine whether the fit improves. The direction and magnitude of the adjustment depend on the fitting algorithm. The toolbox provides these algorithms:

   - Trust-region — This is the default algorithm and must be used if you specify coefficient constraints. It can solve difficult nonlinear problems more efficiently than the other algorithms and it represents an improvement over the popular Levenberg-Marquardt algorithm.
   - Levenberg-Marquardt — This algorithm has been used for many years and has proved to work most of the time for a wide range of nonlinear models and starting values. If the trust-region algorithm does not produce a reasonable fit, and you do not have coefficient constraints, you should try the Levenberg-Marquardt algorithm.
   - Gauss-Newton — This algorithm is potentially faster than the other algorithms, but it assumes that the residuals are close to zero. It's

included with the toolbox for pedagogical reasons and should be the last choice for most models and data sets.

4 Iterate the process by returning to step 2 until the fit reaches the specified convergence criteria.

You can use weights and robust fitting for nonlinear models, and the fitting process is modified accordingly.

Because of the nature of the approximation process, no algorithm is foolproof for all nonlinear models, data sets, and starting points. Therefore, if you do not achieve a reasonable fit using the default starting points, algorithm, and convergence criteria, you should experiment with different options. Refer to "Specifying Fit Options" on page 2-59 for a description of how to modify the default options. Because nonlinear models can be particularly sensitive to the starting points, this should be the first fit option you modify.

Residual Analysis

In this section...

Introduction

After fitting data with one or more models, you should evaluate the goodness of fit. A visual examination of the fitted curve displayed in Curve Fitting Tool should be your first step. Beyond that, the toolbox provides these methods to assess goodness of fit for both linear and nonlinear parametric fits:

- Residual analysis
- Goodness of fit statistics
- Confidence and prediction bounds

As is common in statistical literature, the term *goodness of fit* is used here in several senses: A "good fit" might be a model

- that your data could reasonably have come from, given the assumptions of least-squares fitting
- in which the model coefficients can be estimated with little uncertainty
- that explains a high proportion of the variability in your data, and is able to predict new observations with high certainty

A particular application might dictate still other aspects of model fitting that are important to achieving a good fit, such as a simple model that is easy to interpret. The methods described here can help you determine goodness of fit in all these senses.

These methods group into two types: graphical and numerical. Plotting residuals and prediction bounds are graphical methods that aid visual interpretation, while computing goodness-of-fit statistics and coefficient confidence bounds yield numerical measures that aid statistical reasoning.

Generally speaking, graphical measures are more beneficial than numerical measures because they allow you to view the entire data set at once, and they can easily display a wide range of relationships between the model and the data. The numerical measures are more narrowly focused on a particular aspect of the data and often try to compress that information into a single number. In practice, depending on your data and analysis requirements, you might need to use both types to determine the best fit.

Note that it is possible that none of your fits can be considered suitable for your data, based on these methods. In this case, it might be that you need to select a different model. It is also possible that all the goodness-of-fit measures indicate that a particular fit is suitable. However, if your goal is to extract fitted coefficients that have physical meaning, but your model does not reflect the physics of the data, the resulting coefficients are useless. In this case, understanding what your data represents and how it was measured is just as important as evaluating the goodness of fit.

Computing Residuals

The residuals from a fitted model are defined as the differences between the response data and the fit to the response data at each predictor value.

residual = *data* – *fit*

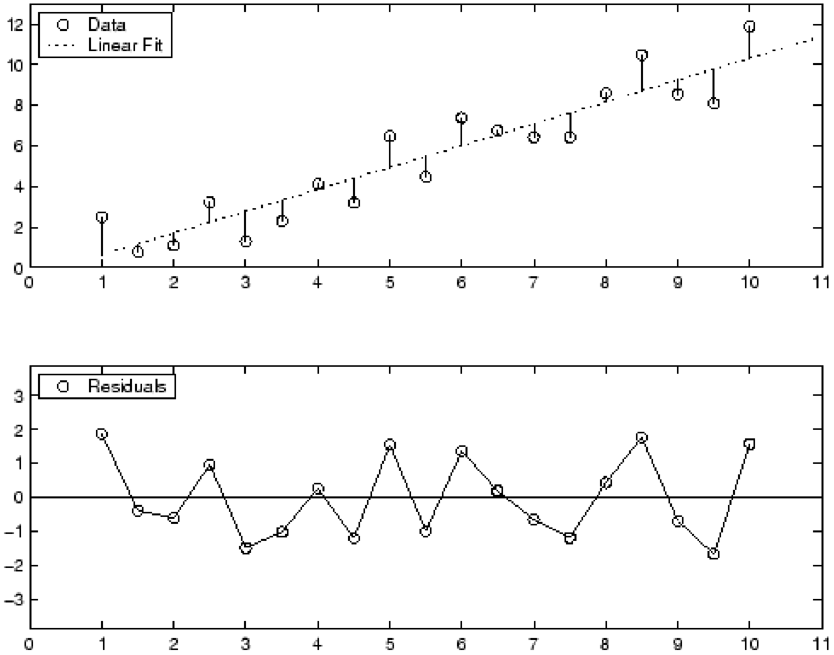
You display the residuals in Curve Fitting Tool by selecting the menu item **View > Residuals**.

Mathematically, the residual for a specific predictor value is the difference between the response value *y* and the predicted response value $\hat{y}$.

$$r = y - \hat{y}$$

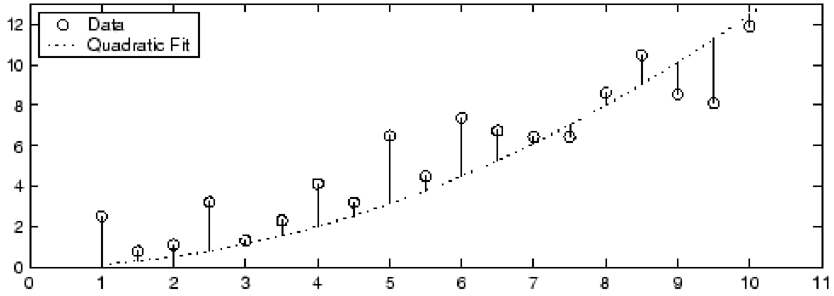
Assuming the model you fit to the data is correct, the residuals approximate the random errors. Therefore, if the residuals appear to behave randomly, it suggests that the model fits the data well. However, if the residuals display a systematic pattern, it is a clear sign that the model fits the data poorly. Always bear in mind that many results of model fitting, such as confidence bounds, will be invalid should the model be grossly inappropriate for the data.

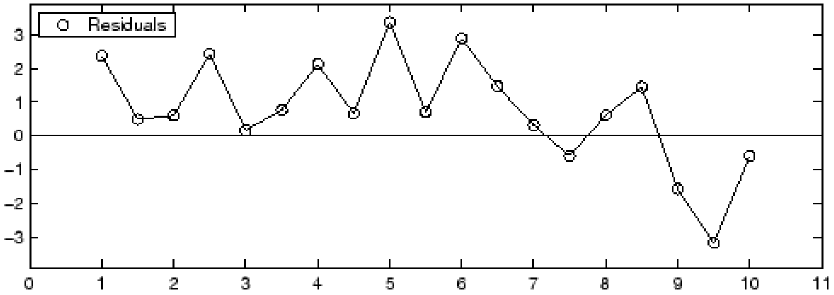
A graphical display of the residuals for a first degree polynomial fit is shown below. The top plot shows that the residuals are calculated as the vertical distance from the data point to the fitted curve. The bottom plot displays the residuals relative to the fit, which is the zero line.



The residuals appear randomly scattered around zero indicating that the model describes the data well.

A graphical display of the residuals for a second-degree polynomial fit is shown below. The model includes only the quadratic term, and does not include a linear or constant term.





The residuals are systematically positive for much of the data range indicating that this model is a poor fit for the data.

Goodness-of-Fit Statistics

After using graphical methods to evaluate the goodness of fit, you should examine the goodness-of-fit statistics. Curve Fitting Toolbox software supports these goodness-of-fit statistics for parametric models:

- The sum of squares due to error (SSE)
- R-square
- Adjusted R-square
- Root mean squared error (RMSE)

For the current fit, these statistics are displayed in the **Results** list box in the **Fit Editor**. For all fits in the current curve-fitting session, you can compare the goodness-of-fit statistics in the **Table of fits**.

Sum of Squares Due to Error

This statistic measures the total deviation of the response values from the fit to the response values. It is also called the summed square of residuals and is usually labeled as *SSE*.

$$SSE = \sum_{i=1}^{n} w_i(y_i - \hat{y}_i)^2$$

A value closer to 0 indicates that the model has a smaller random error component, and that the fit will be more useful for prediction.

R-Square

This statistic measures how successful the fit is in explaining the variation of the data. Put another way, R-square is the square of the correlation between the response values and the predicted response values. It is also called the square of the multiple correlation coefficient and the coefficient of multiple determination.

R-square is defined as the ratio of the sum of squares of the regression (*SSR*) and the total sum of squares (*SST*). *SSR* is defined as

$$SSR = \sum_{i=1}^{n} w_i(\hat{y}_i - \bar{y})^2$$

SST is also called the sum of squares about the mean, and is defined as

$$SST = \sum_{i=1}^{n} w_i(y_i - \bar{y})^2$$

where *SST* = *SSR* + *SSE*. Given these definitions, R-square is expressed as

$$\text{R-square} = \frac{SSR}{SST} = 1 - \frac{SSE}{SST}$$

R-square can take on any value between 0 and 1, with a value closer to 1 indicating that a greater proportion of variance is accounted for by the model. For example, an R-square value of 0.8234 means that the fit explains 82.34% of the total variation in the data about the average.

If you increase the number of fitted coefficients in your model, R-square will increase although the fit may not improve in a practical sense. To avoid this situation, you should use the degrees of freedom adjusted R-square statistic described below.

Note that it is possible to get a negative R-square for equations that do not contain a constant term. Because R-square is defined as the proportion of variance explained by the fit, if the fit is actually worse than just fitting a horizontal line then R-square is negative. In this case, R-square cannot be interpreted as the square of a correlation. Such situations indicate that a constant term should be added to the model.

Degrees of Freedom Adjusted R-Square

This statistic uses the R-square statistic defined above, and adjusts it based on the residual degrees of freedom. The residual degrees of freedom is defined as the number of response values *n* minus the number of fitted coefficients *m* estimated from the response values.

$$v = n - m$$

v indicates the number of independent pieces of information involving the *n* data points that are required to calculate the sum of squares. Note that if parameters are bounded and one or more of the estimates are at their bounds, then those estimates are regarded as fixed. The degrees of freedom is increased by the number of such parameters.

The adjusted R-square statistic is generally the best indicator of the fit quality when you compare two models that are *nested* — that is, a series of models each of which adds additional coefficients to the previous model.

$$\text{adjusted R-square} = 1 - \frac{SSE(n-1)}{SST(v)}$$

The adjusted R-square statistic can take on any value less than or equal to 1, with a value closer to 1 indicating a better fit. Negative values can occur when the model contains terms that do not help to predict the response.

Root Mean Squared Error

This statistic is also known as the fit standard error and the standard error of the regression. It is an estimate of the standard deviation of the random component in the data, and is defined as

$$RMSE = s = \sqrt{MSE}$$

where *MSE* is the mean square error or the residual mean square

$$MSE = \frac{SSE}{v}$$

Just as with *SSE*, an *MSE* value closer to 0 indicates a fit that is more useful for prediction.

Confidence and Prediction Bounds

Curve Fitting Toolbox software lets you calculate confidence bounds for the fitted coefficients, and prediction bounds for new observations or for the fitted function. Additionally, for prediction bounds, you can calculate simultaneous bounds, which take into account all predictor values, or you can calculate nonsimultaneous bounds, which take into account only individual predictor values. The coefficient confidence bounds are presented numerically, while the prediction bounds are displayed graphically and are also available numerically.

The available confidence and prediction bounds are summarized below.

Types of Confidence and Prediction Bounds

Interval Type	Description
Fitted coefficients	Confidence bounds for the fitted coefficients
New observation	Prediction bounds for a new observation (response value)
New function	Prediction bounds for a new function value

Note Prediction bounds are also often described as confidence bounds because you are calculating a confidence interval for a predicted response.

Confidence and prediction bounds define the lower and upper values of the associated interval, and define the width of the interval. The width of the interval indicates how uncertain you are about the fitted coefficients, the predicted observation, or the predicted fit. For example, a very wide interval for the fitted coefficients can indicate that you should use more data when fitting before you can say anything very definite about the coefficients.

The bounds are defined with a level of certainty that you specify. The level of certainty is often 95%, but it can be any value such as 90%, 99%, 99.9%, and so on. For example, you might want to take a 5% chance of being incorrect about predicting a new observation. Therefore, you would calculate a 95% prediction interval. This interval indicates that you have a 95% chance that the new observation is actually contained within the lower and upper prediction bounds.

Calculating and Displaying Confidence Bounds

The confidence bounds for fitted coefficients are given by

$$C = b \pm t\sqrt{S}$$

where b are the coefficients produced by the fit, t depends on the confidence level, and is computed using the inverse of Student's t cumulative distribution function, and S is a vector of the diagonal elements from the estimated covariance matrix of the coefficient estimates, $(X^{T}X)^{-1}s^{2}$. In a linear fit, X is the design matrix, while for a nonlinear fit X is the Jacobian of the fitted values with respect to the coefficients. X^T is the transpose of X, and s^2 is the mean squared error.

The confidence bounds are displayed in the **Results** list box in the **Fit Editor** using the following format.

```
p1 = 1.275  (1.113, 1.437)
```

The fitted value for the coefficient `p1` is 1.275, the lower bound is 1.113, the upper bound is 1.437, and the interval width is 0.324. By default, the confidence level for the bounds is 95%. You can change this level to any value with the **View > Confidence Level** menu item in Curve Fitting Tool.

You can calculate confidence intervals at the command line with the `confint` function.

Calculating and Displaying Prediction Bounds

As mentioned previously, you can calculate prediction bounds for a new observation or for the fitted curve. In both cases, the prediction is based on an existing fit to the data. Additionally, the bounds can be simultaneous and measure the confidence for all predictor values, or they can be nonsimultaneous and measure the confidence only for a single predetermined predictor value. If you are predicting a new observation, nonsimultaneous bounds measure the confidence that the new observation lies within the interval given a single predictor value. Simultaneous bounds measure the confidence that a new observation lies within the interval regardless of the predictor value.

The nonsimultaneous prediction bounds for a new observation at the predictor value x are given by

$$P_{n,o} = \hat{y} \pm t\sqrt{s^2 + xSx'}$$

where s^2 is the mean squared error, t depends on the confidence level, and is computed using the inverse of Student's t cumulative distribution function, and S is the covariance matrix of the coefficient estimates, $(X^TX)^{-1}s^2$. Note that *x* is defined as a row vector of the design matrix or Jacobian evaluated at a specified predictor value.

The simultaneous prediction bounds for a new observation and for all predictor values are given by

$$P_{s,o} = \hat{y} \pm f\sqrt{s^2 + xSx'}$$

where *f* depends on the confidence level, and is computed using the inverse of the *F* cumulative distribution function.

The nonsimultaneous prediction bounds for the function at a single predictor value *x* are given by

$$P_{n,f} = \hat{y} \pm t\sqrt{xSx'}$$

The simultaneous prediction bounds for the function and for all predictor values are given by

$$P_{s,f} = \hat{y} \pm f\sqrt{xSx'}$$

You can graphically display prediction bounds two ways: using Curve Fitting Tool or using the Analysis GUI. With Curve Fitting Tool, you can display nonsimultaneous prediction bounds for new observations with **View > Prediction Bounds**. By default, the confidence level for the bounds is 95%. You can change this level to any value with **View > Confidence Level**. With the Analysis GUI, you can display nonsimultaneous prediction bounds for the function or for new observations. Additionally, you can view prediction bounds in the **Results** box of the Analysis GUI.

You can display numerical prediction bounds of any type at the command line with the `predint` function.

To understand the quantities associated with each type of prediction interval, recall that the data, fit, and residuals are related through the formula

data = fit + residuals

where the fit and residuals terms are estimates of terms in the formula

data = *model* + *random error*

Suppose you plan to take a new observation at the predictor value x_{n+1}. Call the new observation $y_{n+1}(x_{n+1})$ and the associated error ε_{n+1}. Then

$$y_{n+1}(x_{n+1}) = f(x_{n+1}) + \varepsilon_{n+1}$$

where $f(x_{n+1})$ is the true but unknown function you want to estimate at x_{n+1}. The likely values for the new observation or for the estimated function are provided by the nonsimultaneous prediction bounds.

If instead you want the likely value of the new observation to be associated with any predictor value, the previous equation becomes

$$y_{n+1}(x) = f(x) + \varepsilon$$

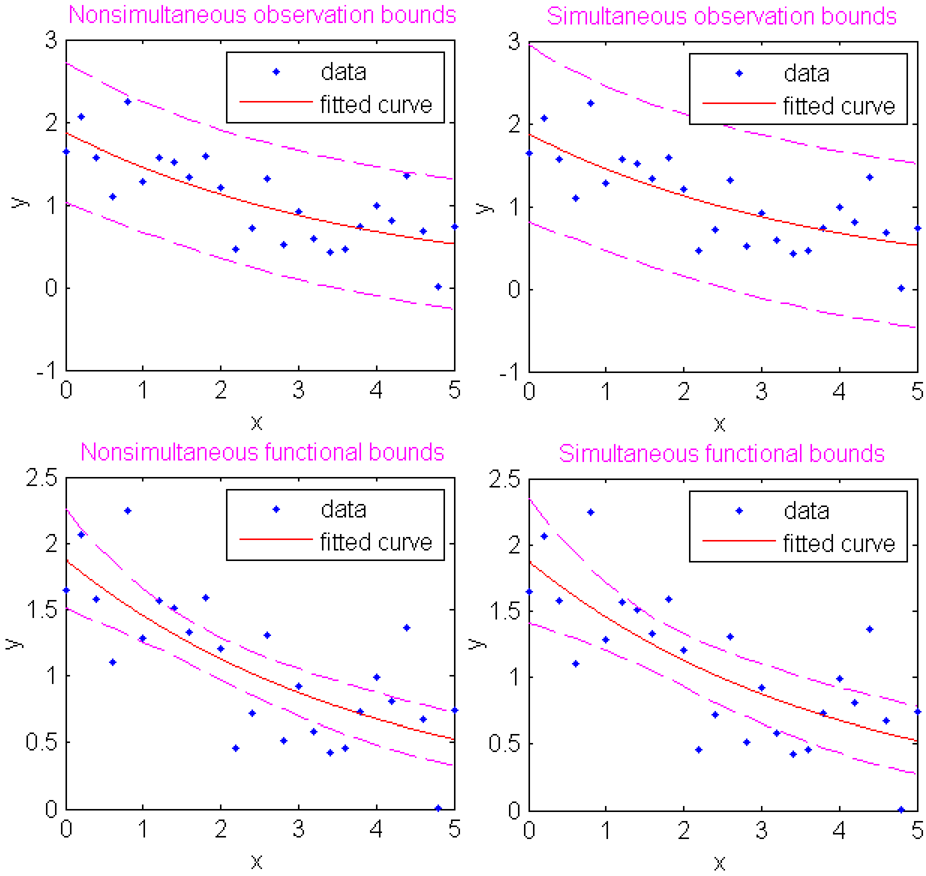
The likely values for this new observation or for the estimated function are provided by the simultaneous prediction bounds.

The types of prediction bounds are summarized below.

Types of Prediction Bounds

Type of Bound	Simultaneous or Non-simultaneous	Associated Equation
Observation	Non-simultaneous	$y_{n+1}(x_{n+1})$
	Simultaneous	$y_{n+1}(x)$, for all x
Function	Non-simultaneous	$f(x_{n+1})$
	Simultaneous	$f(x)$, for all x

The nonsimultaneous and simultaneous prediction bounds for a new observation and the fitted function are shown below. Each graph contains three curves: the fit, the lower confidence bounds, and the upper confidence bounds. The fit is a single-term exponential to generated data and the bounds reflect a 95% confidence level. Note that the intervals associated with a new observation are wider than the fitted function intervals because of the additional uncertainty in predicting a new response value (the curve plus random errors).



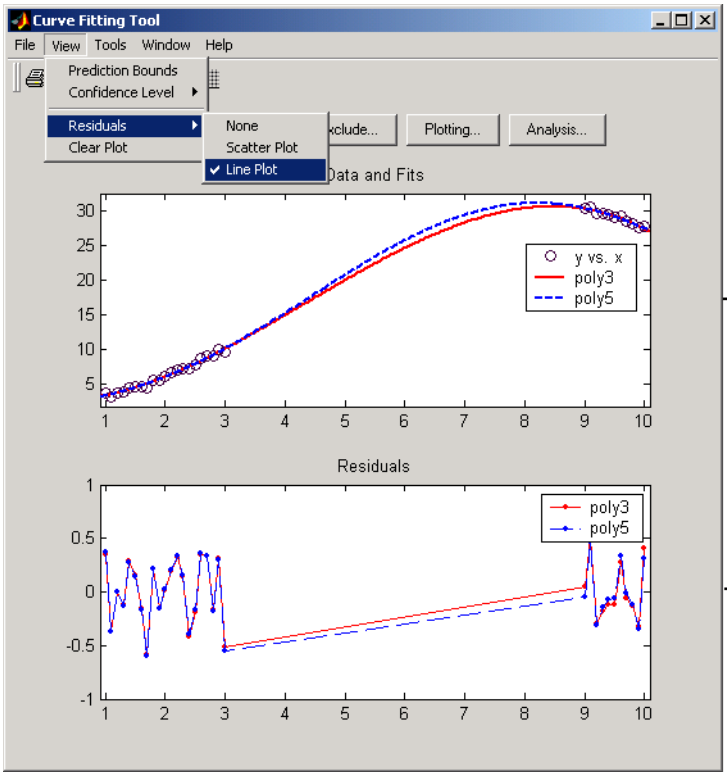
Example: Residual Analysis

This example fits several polynomial models to generated data and evaluates how well those models fit the data and how precisely they can predict. The data is generated from a cubic curve, and there is a large gap in the range of the *x* variable where no data exist.

```
x = [1:0.1:3 9:0.1:10]';
c = [2.5 -0.5 1.3 -0.1];
```

```
y = c(1) + c(2)*x + c(3)*x.^2 + c(4)*x.^3 + (rand(size(x))-0.5);
```

After you import the data, fit it using a cubic polynomial and a fifth degree polynomial. The data, fits, and residuals are shown below. You display the residuals in Curve Fitting Tool with the **View > Residuals** menu item.

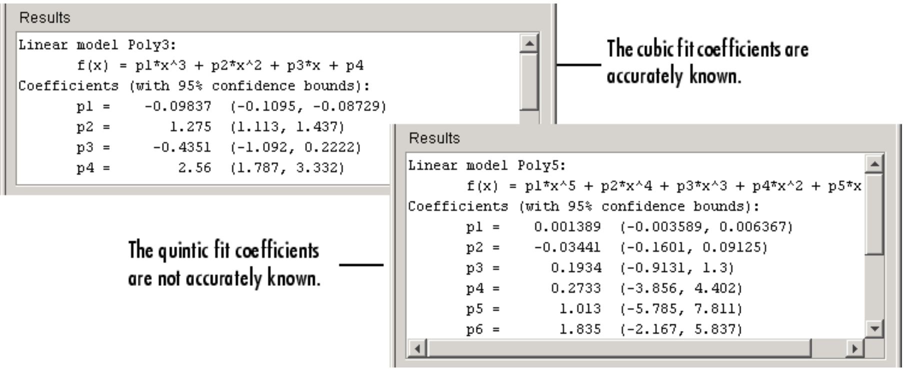


Both fits appear to model the data well.

The residuals for both fits appear to be randomly distributed.

Both models appear to fit the data well, and the residuals appear to be randomly distributed around zero. Therefore, a graphical evaluation of the fits does not reveal any obvious differences between the two equations.

The numerical fit results are shown below.



Results

```
Linear model Poly3:
       f(x) = p1*x^3 + p2*x^2 + p3*x + p4
Coefficients (with 95% confidence bounds):
       p1 =    -0.09837  (-0.1095, -0.08729)
       p2 =       1.275  (1.113, 1.437)
       p3 =     -0.4351  (-1.092, 0.2222)
       p4 =        2.56  (1.787, 3.332)
```

The cubic fit coefficients are accurately known.

Results

```
Linear model Poly5:
       f(x) = p1*x^5 + p2*x^4 + p3*x^3 + p4*x^2 + p5*x
Coefficients (with 95% confidence bounds):
       p1 =    0.001389  (-0.003589, 0.006367)
       p2 =    -0.03441  (-0.1601, 0.09125)
       p3 =      0.1934  (-0.9131, 1.3)
       p4 =      0.2733  (-3.856, 4.402)
       p5 =       1.013  (-5.785, 7.811)
       p6 =       1.835  (-2.167, 5.837)
```

The quintic fit coefficients are not accurately known.

As expected, the fit results for `poly3` are reasonable because the generated data follows a cubic curve. The 95% confidence bounds on the fitted coefficients indicate that they are acceptably precise. However, the 95% confidence bounds for `poly5` indicate that the fitted coefficients are not known precisely.

The goodness-of-fit statistics are shown in the **Table of Fits**. By default, the adjusted R-square and RMSE statistics are not displayed in the table. To display these statistics, click the **Table options** button and select **Adj R-sq** and **RMSE**, as shown below.

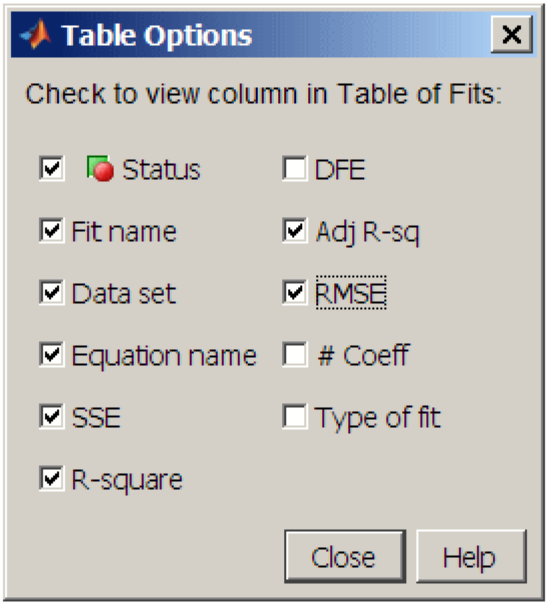


Table Options

Check to view column in Table of Fits:

☑ Status

☑ Fit name

☑ Data set

☑ Equation name

☑ SSE

☑ R-square

☐ DFE

☑ Adj R-sq

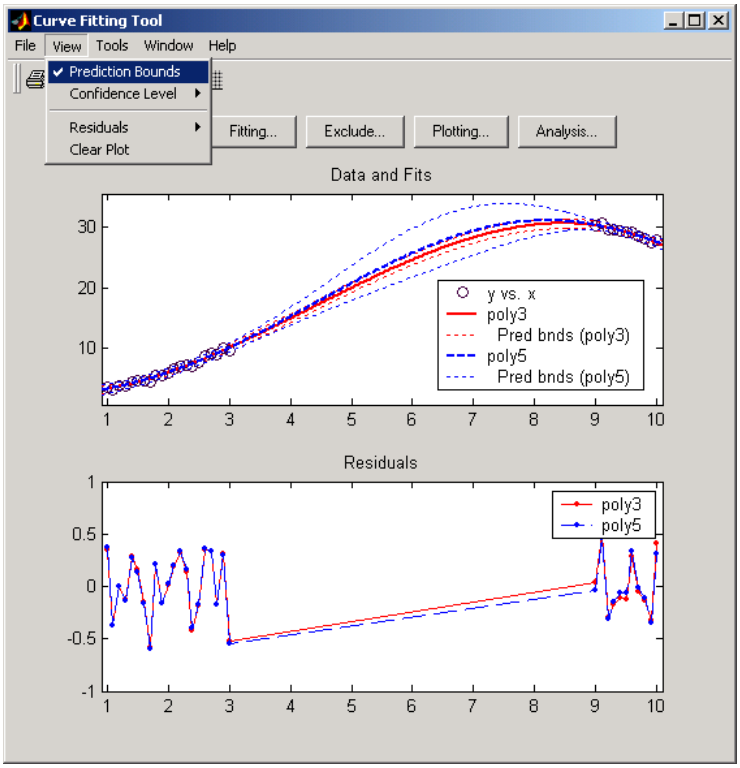
☑ RMSE

☐ # Coeff

☐ Type of fit

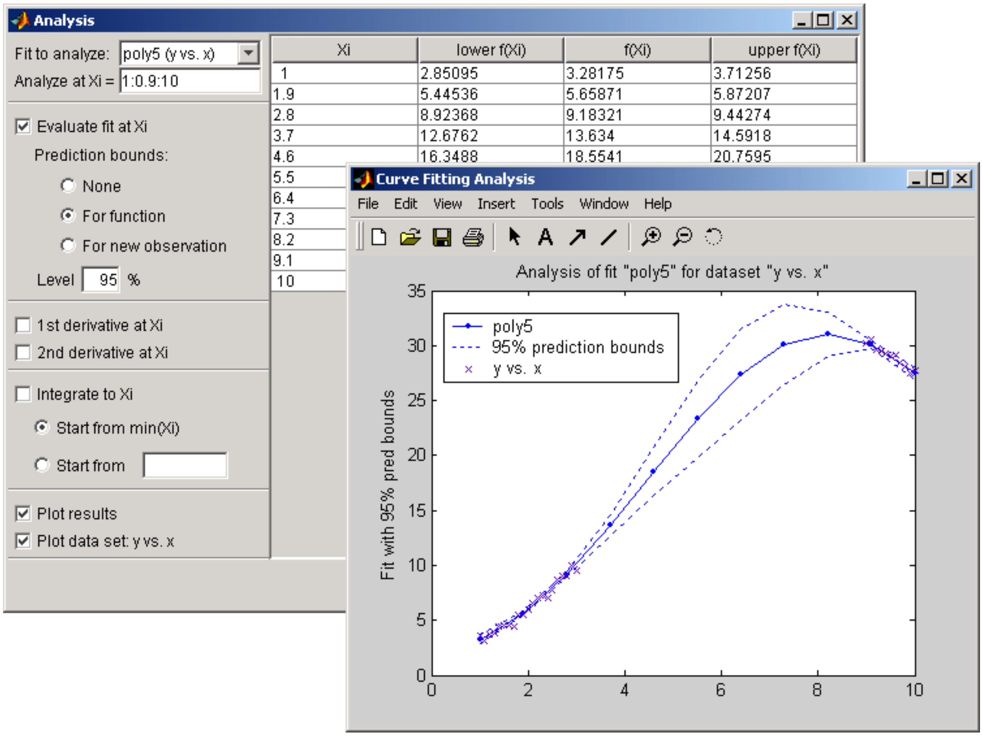
The statistics do not reveal a substantial difference between the two equations.

The 95% nonsimultaneous prediction bounds for new observations are shown below. To display prediction bounds in Curve Fitting Tool, select the **View > Prediction Bounds** menu item. Alternatively, you can view prediction bounds for the function or for new observations using the Analysis GUI.



The prediction bounds for `poly3` indicate that new observations can be predicted with a small uncertainty throughout the entire data range. This is not the case for `poly5`. It has wider prediction bounds in the area where no data exist, apparently because the data does not contain enough information to estimate the higher degree polynomial terms accurately. In other words, a fifth-degree polynomial overfits the data. You can confirm this by using the Analysis GUI to compute bounds for the functions themselves.

The 95% prediction bounds for the fitted function using `poly5` are shown below. As you can see, the uncertainty in predicting the function is large in the center of the data. Therefore, you would conclude that more data must be collected before you can make precise predictions using a fifth-degree polynomial.



Fit to analyze: poly5 (y vs. x)

Analyze at Xi = 1:0.9:10

☑ Evaluate fit at Xi

Prediction bounds:

○ None

● For function

○ For new observation

Level 95 %

□ 1st derivative at Xi

□ 2nd derivative at Xi

□ Integrate to Xi

● Start from min(Xi)

○ Start from

☑ Plot results

☑ Plot data set: y vs. x

Xi	lower f(Xi)	f(Xi)	upper f(Xi)
1	2.85095	3.28175	3.71256
1.9	5.44536	5.65871	5.87207
2.8	8.92368	9.18321	9.44274
3.7	12.6762	13.634	14.5918
4.6	16.3488	18.5541	20.7595
5.5			
6.4			
7.3			
8.2			
9.1			
10			

In conclusion, you should examine all available goodness-of-fit measures before deciding on the fit that is best for your purposes. A graphical examination of the fit and residuals should always be your initial approach. However, some fit characteristics are revealed only through numerical fit results, statistics, and prediction bounds.

Interpolants

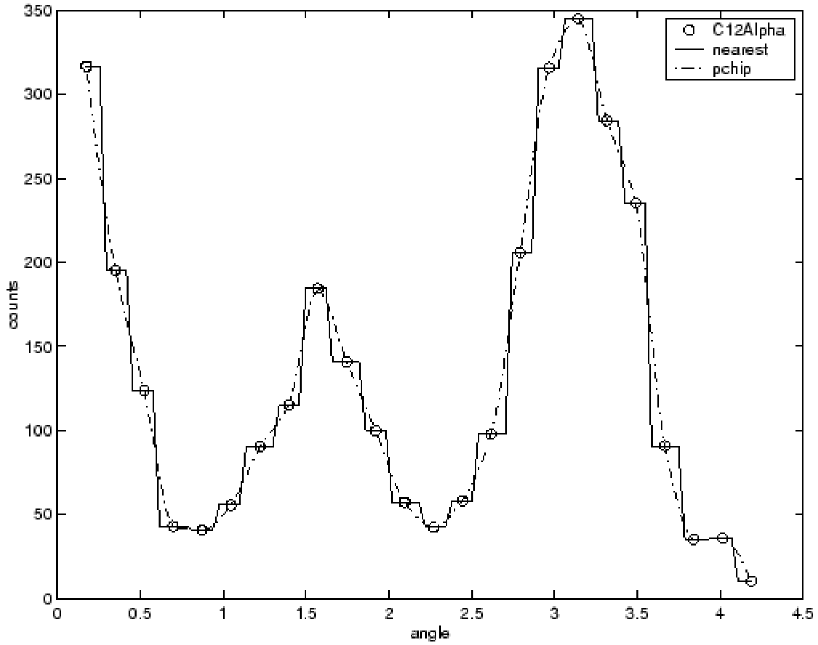
Interpolation is a process for estimating values that lie between known data points. The supported interpolant methods are shown below.

Interpolant Methods

Method	Description
Linear	Linear interpolation. This method fits a different linear polynomial between each pair of data points.
Nearest neighbor	Nearest neighbor interpolation. This method sets the value of an interpolated point to the value of the nearest data point. Therefore, this method does not generate any new data points.
Cubic spline	Cubic spline interpolation. This method fits a different cubic polynomial between each pair of data points.
Shape-preserving	Piecewise cubic Hermite interpolation (PCHIP). This method preserves monotonicity and the shape of the data.

The type of interpolant you should use depends on the characteristics of the data being fit, the required smoothness of the curve, speed considerations, post-fit analysis requirements, and so on. The linear and nearest neighbor methods are fast, but the resulting curves are not very smooth. The cubic spline and shape-preserving methods are slower, but the resulting curves are often very smooth.

For example, the nuclear reaction data from the file `carbon12alpha.mat` is shown below with a nearest neighbor interpolant fit and a shape-preserving (PCHIP) interpolant fit. Clearly, the nearest neighbor interpolant does not follow the data as well as the shape-preserving interpolant. The difference between these two fits can be important if you are interpolating. However, if you want to integrate the data to get a sense of the total strength of the reaction, then both fits provide nearly identical answers for reasonable integration bin widths.



Note Goodness-of-fit statistics, prediction bounds, and weights are not defined for interpolants. Additionally, the fit residuals are always zero (within computer precision) because interpolants pass through the data points.

Interpolants are defined as *piecewise polynomials* because the fitted curve is constructed from many "pieces." For cubic spline and PCHIP interpolation, each piece is described by four coefficients, which are calculated using a cubic (third-degree) polynomial. Refer to the `spline` function for more information about cubic spline interpolation. Refer to the `pchip` function for more information about shape-preserving interpolation, and for a comparison of the two methods.

It is possible to fit a single "global" polynomial interpolant to data, with a degree one less than the number of data points. However, such a fit can have wildly erratic behavior between data points. In contrast, the piecewise polynomials described here always produce a well-behaved fit, and thus they are more flexible than parametric polynomials and can be effectively used for a wider range of data sets.

5

Function Reference

Data Preprocessing

`cftool`	Open Curve Fitting Tool
`excludedata`	Exclude data from fit
`smooth`	Smooth response data

Data Fitting

`cftool`	Open Curve Fitting Tool
`fit`	Fit model to data
`fitoptions`	Create or modify fit options structure
`fittype`	Constructor for `fittype` object
`get`	Get fit options structure field names and values
`set`	Assign values in fit options structure

Fit Type Methods

`argnames`	Input argument names of `cfit` or `fittype` object
`category`	Category of fit of `cfit` or `fittype` object
`coeffnames`	Coefficient names of `cfit` or `fittype` object
`dependnames`	Dependent variable of `cfit` or `fittype` object
`feval`	Evaluate `cfit` or `fittype` object
`fittype`	Constructor for `fittype` object

`formula`	Formula of `cfit` or `fittype` object
`indepnames`	Independent variable of `cfit` or `fittype` object
`islinear`	Determine if `cfit` or `fittype` object is linear
`numargs`	Number of input arguments of `cfit` or `fittype` object
`numcoeffs`	Number of coefficients of `cfit` or `fittype` object
`probnames`	Problem-dependent parameter names of `cfit` or `fittype` object
`type`	Name of `cfit` or `fittype` object

Curve Fit Methods

`argnames`	Input argument names of `cfit` or `fittype` object
`category`	Category of fit of `cfit` or `fittype` object
`cfit`	Constructor for `cfit` object
`coeffnames`	Coefficient names of `cfit` or `fittype` object
`coeffvalues`	Coefficient values of `cfit` object
`confint`	Confidence intervals for fit coefficients of `cfit` object
`dependnames`	Dependent variable of `cfit` or `fittype` object
`differentiate`	Differentiate `cfit` object
`feval`	Evaluate `cfit` or `fittype` object
`formula`	Formula of `cfit` or `fittype` object

`indepnames`	Independent variable of `cfit` or `fittype` object
`integrate`	Integrate `cfit` object
`islinear`	Determine if `cfit` or `fittype` object is linear
`numargs`	Number of input arguments of `cfit` or `fittype` object
`numcoeffs`	Number of coefficients of `cfit` or `fittype` object
`plot`	Plot `cfit` object
`predint`	Prediction intervals for `cfit` object
`probnames`	Problem-dependent parameter names of `cfit` or `fittype` object
`probvalues`	Problem-dependent parameter values of `cfit` object
`type`	Name of `cfit` or `fittype` object

Fit Postprocessing

`cftool`	Open Curve Fitting Tool
`coeffvalues`	Coefficient values of `cfit` object
`confint`	Confidence intervals for fit coefficients of `cfit` object
`differentiate`	Differentiate `cfit` object
`feval`	Evaluate `cfit` or `fittype` object
`integrate`	Integrate `cfit` object
`plot`	Plot `cfit` object

`predint`	Prediction intervals for `cfit` object
`probvalues`	Problem-dependent parameter values of `cfit` object

Information and Help

`cflibhelp`	Information on library models
`datastats`	Data statistics

6

Functions — Alphabetical List

argnames

Purpose

Input argument names of `cfit` or `fittype` object

Syntax

`args = argnames(fun)`

Description

`args = argnames(fun)` returns the input argument (variable and coefficient) names of the `cfit` or `fittype` object `fun` as an `n`-by-1 cell array of strings `args`, where `n = numargs(fun)`.

Example

```
f = fittype('a*x^2+b*exp(n*x)');
nargs = numargs(f)
nargs =
     4
args = argnames(f)
args =
    'a'
    'b'
    'n'
    'x'
```

See Also

`fittype`, `formula`, `numargs`

Purpose

Category of fit of cfit or fittype object

Syntax

```
cname = category(fun)
```

Description

cname = category(fun) returns the fit category cname of the cfit or fittype object fun, where cname is one of 'custom', 'interpolant', 'library', or 'spline'.

Example

```
f1 = fittype('a*x^2+b*exp(n*x)');
category(f1)
ans =
custom

f2 = fittype('pchipinterp');
category(f2)
ans =
interpolant

f3 = fittype('fourier4');
category(f3)
ans =
library

f4 = fittype('smoothingspline');
category(f4)
ans =
spline
```

See Also

fittype, type, cflibhelp

cfit

Purpose

Constructor for cfit object

Syntax

```
cfun = cfit(ffun,coeff1,coeff2,...)
```

Description

cfun = cfit(ffun,coeff1,coeff2,...) constructs the cfit object cfun using the model type specified by the fittype object ffun and the coefficient values coeff1, coeff2, etc.

Note cfit is called by the fit function when fitting fittype objects to data. To create a cfit object that is the result of a regression, use fit.

You should only call cfit directly if you want to assign values to coefficients and problem parameters of a fittype object *without* performing a fit.

Example

```
f = fittype('a*x^2+b*exp(n*x)')
f =
     General model:
       f(a,b,n,x) = a*x^2+b*exp(n*x)
c = cfit(f,1,10.3,-1e2)
c =
     General model:
       c(x) = a*x^2+b*exp(n*x)
     Coefficients:
       a =           1
       b =        10.3
       n =        -100
```

See Also

fit, fittype, feval

Purpose

Information on library models

Syntax

```
cflibhelp
cflibhelp libtype
```

Description

`cflibhelp` displays the names, equations, and descriptions of all models in the curve-fitting library. Library names are used as input arguments in the `fit` and `fittype` functions.

`cflibhelp libtype` restricts the display of names, equations, and descriptions to the subcategory of library models `libtype`. Recognized library types are listed in the table below.

`libtype`	Description
`distribution`	Probability distribution models
`exponential`	One-term and two-term exponential models
`fourier`	Fourier polynomial models
`gaussian`	Sums of Gaussian models, up to eight terms
`interpolant`	Interpolating models, including linear, nearest neighbor, cubic spline, and shape-preserving cubic spline
`polynomial`	Polynomial models, up to ninth degree
`power`	One-term and two-term power models
`rational`	Ratios of polynomial models, up to degree 5 in both the numerator and the denominator
`sin`	Sums of sinusoidal models, up to eight terms
`spline`	Cubic spline and smoothing spline models

For more information on library models, refer to the “Library Models” on page 2-32 section of the User’s Guide.

Example

```
cflibhelp polynomial

  POLYNOMIAL MODELS

        MODELNAME               EQUATION

          poly1                   Y = p1*x+p2
          poly2                   Y = p1*x^2+p2*x+p3
          poly3                   Y = p1*x^3+p2*x^2+...+p4
          ...
          poly9                   Y = p1*x^9+p2*x^8+...+p10
```

See Also

`fit`, `fittype`

Purpose

Open Curve Fitting Tool

Syntax

```
cftool
cftool(xdata,ydata)
cftool(xdata,ydata,w)
```

Description

`cftool` opens Curve Fitting Tool, an interactive environment for fitting curves to one-dimensional data.

`cftool(xdata,ydata)` opens Curve Fitting Tool with predictor data `xdata` and response data `ydata`. `xdata` and `ydata` must be vectors of the same size. `Infs`, `NaNs`, and imaginary parts of complex numbers are ignored in the data.

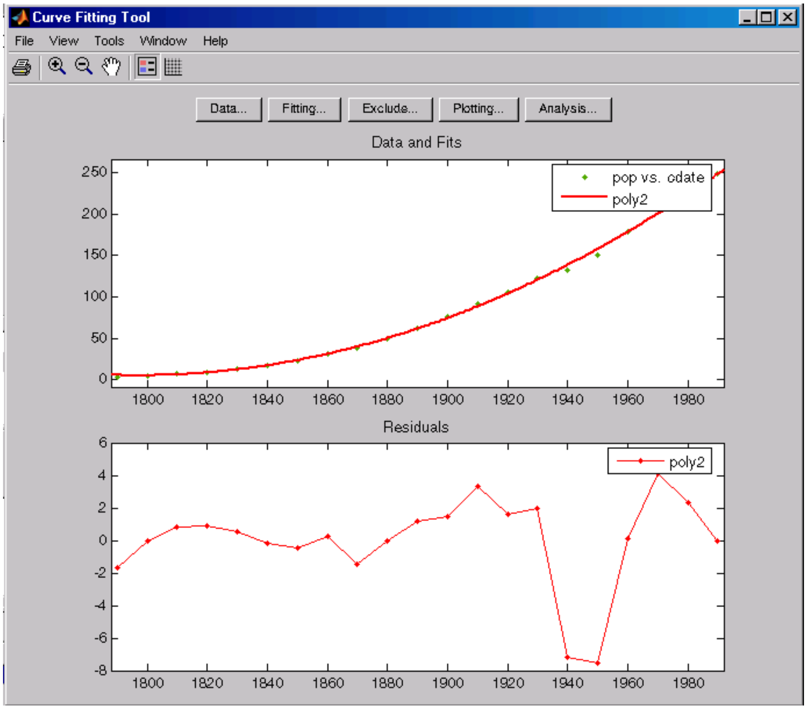
`cftool(xdata,ydata,w)` also imports the weight vector `w` into Curve Fitting Tool for weighting data in subsequent fits. `w` must be the same length as `xdata` and `ydata`.

Remarks

The Curve Fitting Tool is an interactive environment presented in the form of a graphical user interface. It allows you to

- Import data from the MATLAB workspace
- Explore the data graphically
- Preprocess the data for fitting using exclusion rules and smoothing
- Fit a variety of library or custom models to the data
- Generate relevant regression statistics
- Post-process the fit through interpolation, extrapolation, differentiation, and integration
- Export results back to the MATLAB workspace for further analysis and visualization

The main Curve Fitting Tool interface is shown below.



Clicking the **Data**, **Fitting**, **Exclude**, **Plotting**, or **Analysis** buttons opens associated GUIs, described below.

In the figure above, data was imported from the MAT-file `census` using the Data GUI and fit with a quadratic polynomial using the Fitting GUI. Residuals were displayed in the subplot by selecting **View** > **Residuals** > **Line Plot**.

For a complete example that uses many of these GUIs, refer to Chapter 1, "Getting Started".

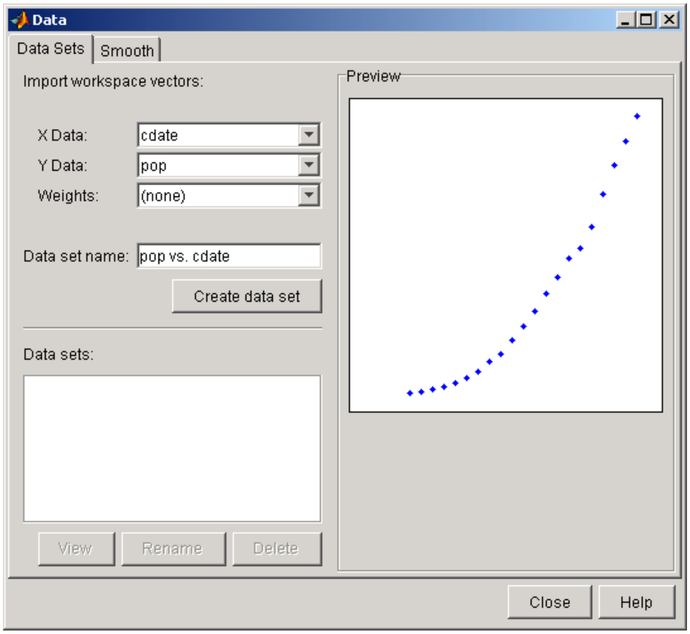
The Data GUI

The Data GUI allows you to

- Import, name, preview, and delete data sets

- Smooth data using a variety of methods

The Data GUI is shown below with the census data loaded.



Import workspace vectors:

X Data: cdate

Y Data: pop

Weights: (none)

Data set name: pop vs. cdate

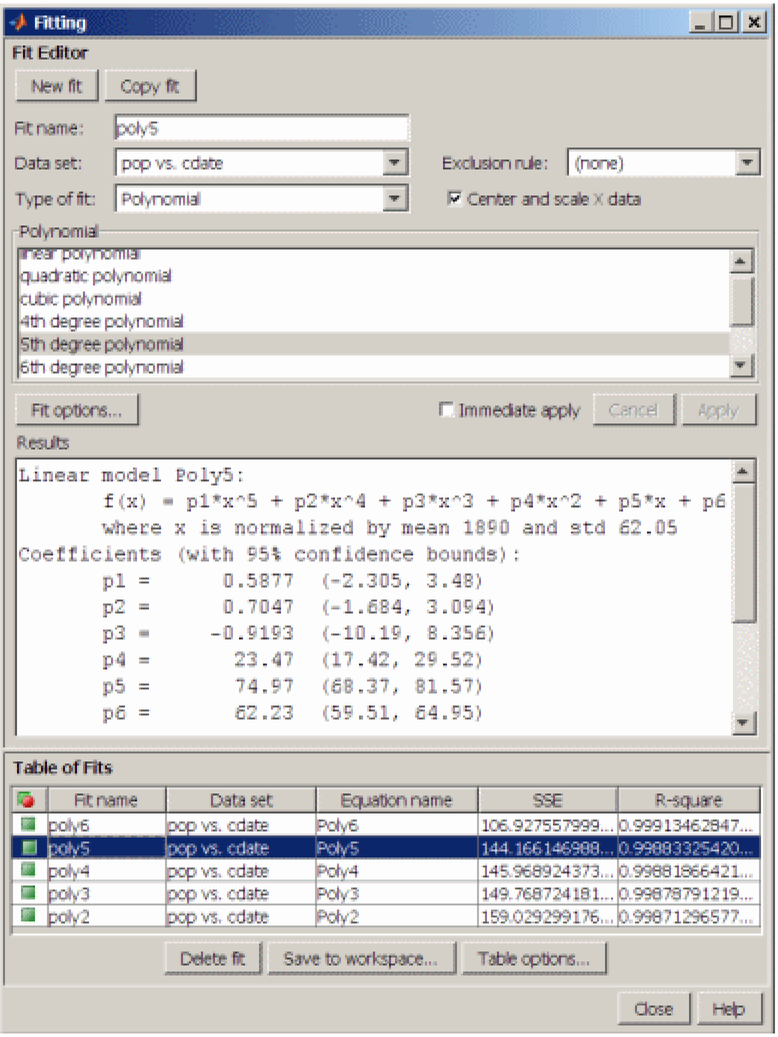
Refer to “Preprocessing Data” on page 2-2 for more information about the Data GUI.

The Fitting GUI

The Fitting GUI allows you to

- Fit your data using parametric or nonparametric models
- Set algorithm options for nonlinear fits
- Compare coefficients and goodness of fit statistics from different models
- Keep track of all data sets and fits in the current session

The Fitting GUI is shown below with the results of fitting the census data.



Fit name: poly5

Data set: pop vs. cdate

Exclusion rule: (none)

Type of fit: Polynomial

☑ Center and scale X data

☐ Immediate apply

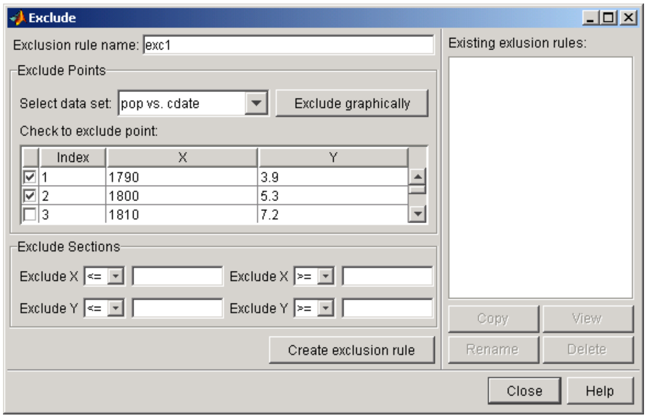
```
Linear model Poly5:
       f(x) = p1*x^5 + p2*x^4 + p3*x^3 + p4*x^2 + p5*x + p6
       where x is normalized by mean 1890 and std 62.05
Coefficients (with 95% confidence bounds):
       p1 =      0.5877  (-2.305, 3.48)
       p2 =      0.7047  (-1.684, 3.094)
       p3 =     -0.9193  (-10.19, 8.356)
       p4 =       23.47  (17.42, 29.52)
       p5 =       74.97  (68.37, 81.57)
       p6 =       62.23  (59.51, 64.95)
```

●	Fit name	Data set	Equation name	SSE	R-square
🟩	poly6	pop vs. cdate	Poly6	106.927557999...	0.99913462847...
🟩	poly5	pop vs. cdate	Poly5	144.166146988...	0.99883325420...
🟩	poly4	pop vs. cdate	Poly4	145.968924373...	0.99881866421...
🟩	poly3	pop vs. cdate	Poly3	149.768724181...	0.99878791219...
🟩	poly2	pop vs. cdate	Poly2	159.029299176...	0.99871296577...

The Exclude GUI

The Exclude GUI allows you to create exclusion rules for a data set. An exclusion rule identifies data to be excluded while fitting. The excluded data can be individual data points, or a section of predictor or response data.

The Exclude GUI is shown below with the first two points of the `census` data marked for exclusion.



Exclusion rule name: exc1

Select data set: pop vs. cdate

Check to exclude point:

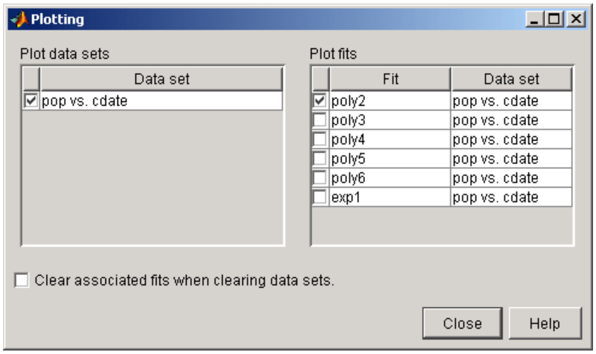
	Index	X	Y
☑	1	1790	3.9
☑	2	1800	5.3
□	3	1810	7.2

Exclude X <=
Exclude X >=
Exclude Y <=
Exclude Y >=

The Plotting GUI

The Plotting GUI allows you to determine the data sets and fits displayed by Curve Fitting Tool.

The Plotting GUI is shown below with the `census` data and the fit `poly2` checked for display.



Plot data sets

	Data set
☑	pop vs. cdate

Plot fits

	Fit	Data set
☑	poly2	pop vs. cdate
☐	poly3	pop vs. cdate
☐	poly4	pop vs. cdate
☐	poly5	pop vs. cdate
☐	poly6	pop vs. cdate
☐	exp1	pop vs. cdate

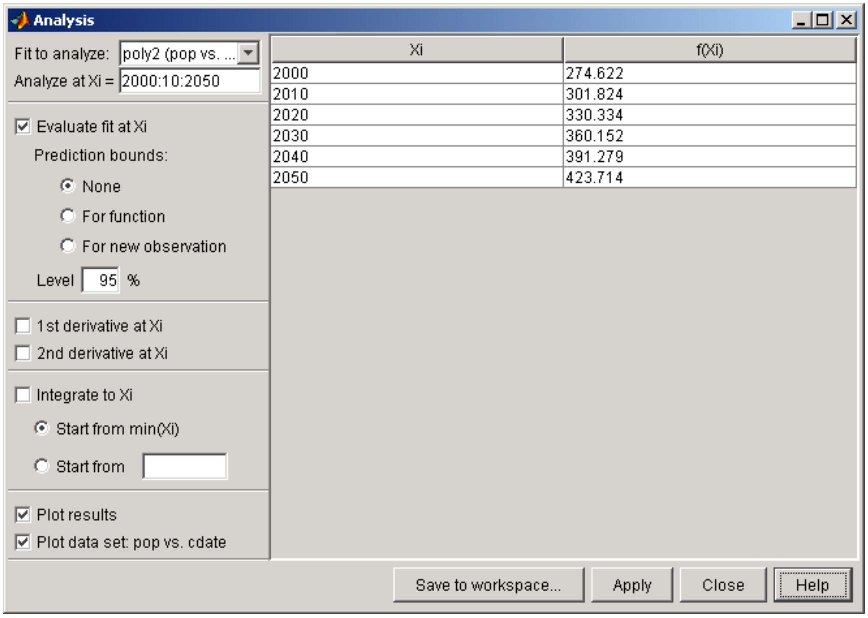
☐ Clear associated fits when clearing data sets.

The Analysis GUI

The Analysis GUI allows you to

- Interpolate, extrapolate, differentiate, or integrate a fit
- Display the results of your analysis numerically or in a plot

The Analysis GUI is shown below with a numerical display of the results of extrapolating the `census` data from the year 2000 to the year 2050 in 10-year increments.



Fit to analyze: poly2 (pop vs. ...

Analyze at Xi = 2000:10:2050

☑ Evaluate fit at Xi

Prediction bounds:

● None

○ For function

○ For new observation

Level 95 %

□ 1st derivative at Xi

□ 2nd derivative at Xi

□ Integrate to Xi

● Start from min(Xi)

○ Start from

☑ Plot results

☑ Plot data set: pop vs. cdate

Xi	f(Xi)
2000	274.622
2010	301.824
2020	330.334
2030	360.152
2040	391.279
2050	423.714

Refer to “Analyzing the Fit” on page 1-17 for an example that uses the Analysis GUI.

coeffnames

Purpose

Coefficient names of `cfit` or `fittype` object

Syntax

```
coeffs = coeffnames(fun)
```

Description

`coeffs = coeffnames(fun)` returns the coefficient (parameter) names of the `cfit` or `fittype` object `fun` as an `n`-by-1 cell array of strings `coeffs`, where `n = numcoeffs(fun)`.

Example

```
f = fittype('a*x^2+b*exp(n*x)');
ncoeffs = numcoeffs(f)
ncoeffs =
     3
coeffs = coeffnames(f)
coeffs =
    'a'
    'b'
    'n'
```

See Also

`fittype`, `formula`, `numcoeffs`, `probnames`, `coeffvalues`

Purpose

Coefficient values of `cfit` object

Syntax

```
coeffvals = coeffvalues(fun)
```

Description

`coeffvals = coeffvalues(fun)` returns the values of the coefficients (parameters) of the `cfit` object `fun` as a 1-by-`n` vector `coeffvals`, where `n = numcoeffs(fun)`.

Example

```
load census

f = fittype('poly2');
coeffnames(f)
ans =
    'p1'
    'p2'
    'p3'
formula(f)
ans =
p1*x^2 + p2*x + p3

c = fit(cdate,pop,f);
coeffvalues(c)
ans =
  1.0e+004 *
    0.0000   -0.0024    2.1130
```

See Also

`coeffnames`,`confint`, `predint`, `probvalues`

Purpose

Confidence intervals for fit coefficients of `cfit` object

Syntax

```
ci = confint(fitresult)
ci = confint(fitresult,level)
```

Description

`ci = confint(fitresult)` returns 95% confidence bounds `ci` on the coefficients associated with the `cfit` object `fitresult`. `fitresult` must be an output from the `fit` function to contain the necessary information for `ci`. `ci` is a 2-by-`n` array where `n = numcoeffs(fitresult)`. The top row of `ci` contains the lower bound for each coefficient; the bottom row contains the upper bound.

`ci = confint(fitresult,level)` returns confidence bounds at the confidence level specified by `level`. `level` must be between `0` and `1`. The default value of `level` is `0.95`.

Remarks

To calculate confidence bounds, `confint` uses R^{-1} (the inverse R factor from QR decomposition of the Jacobian), the degrees of freedom for error, and the root mean squared error. This information is automatically returned by the `fit` function and contained within `fitresult`.

If coefficients are bounded and one or more of the estimates are at their bounds, those estimates are regarded as fixed and do not have confidence bounds.

Note that you cannot calculate confidence bounds if `category(fitresult)` is `'spline'` or `'interpolant'`.

Example

```
load census

fitresult = fit(cdate,pop,'poly2')
fitresult =
     Linear model Poly2:
       fitresult(x) = p1*x^2 + p2*x + p3
     Coefficients (with 95% confidence bounds):
       p1 =    0.006541  (0.006124, 0.006958)
       p2 =      -23.51  (-25.09, -21.93)
       p3 =  2.113e+004  (1.964e+004, 2.262e+004)

ci = confint(fitresult,0.95)
ci =

    0.0061242      -25.086         19641
    0.0069581      -21.934         22618
```

Note that `fit` and `confint` display the confidence bounds in slightly different formats.

See Also

`fit`, `predint`

Purpose

Data statistics

Syntax

```
xds = datastats(xdata)
[xds,yds] = datastats(xdata,ydata)
```

Description

`xds = datastats(xdata)` returns statistics for the column vector `xdata` to the structure `xds`. Fields in `xds` are listed in the table below.

Field	Description
`num`	The number of data values
`max`	The maximum data value
`min`	The minimum data value
`mean`	The mean value of the data
`median`	The median value of the data
`range`	The range of the data
`std`	The standard deviation of the data

`[xds,yds] = datastats(xdata,ydata)` returns statistics for the column vectors `xdata` and `ydata` to the structures `xds` and `yds`, respectively. `xds` and `yds` contain the fields listed in the table above. `xdata` and `ydata` must be of the same size.

Remarks

If `xdata` or `ydata` contains complex values, only the real parts are used in computing the statistics. Data containing `Inf` or `NaN` are processed using the usual MATLAB rules.

Example

Compute statistics for the census data in `census.mat`:

```
load census
[xds,yds] = datastats(cdate,pop)
xds =
       num: 21
       max: 1990
```

```
           min: 1790
          mean: 1890
        median: 1890
         range: 200
           std: 62.048
    yds =
           num: 21
           max: 248.7
           min: 3.9
          mean: 85.729
        median: 62.9
         range: 244.8
           std: 78.601
```

See Also

`excludedata`, `smooth`

dependnames

Purpose

Dependent variable of cfit or fittype object

Syntax

```
dep = dependnames(fun)
```

Description

dep = dependnames(fun) returns the (single) dependent variable name of the cfit or fittype object fun as a 1-by-1 cell array of strings dep.

Example

```
f1 = fittype('a*x^2+b*exp(n*x)');
dep1 = dependnames(f1)
dep1 =
     'y'

f2 = fittype('a*x^2+b*exp(n*x)','dependent','power');
dep2 = dependnames(f2)
dep2 =
     'power'
```

See Also

indepnames, fittype, formula

Purpose

Differentiate `cfit` object

Syntax

```
d1 = differentiate(fun,x)
[d1,d2] = differentiate(...)
```

Description

`d1 = differentiate(fun,x)` differentiates the `cfit` object `fun` at the points specified by the vector `x` and returns the result in `d1`. `d1` is a column vector the same length as `x`.

`[d1,d2] = differentiate(...)` also returns the second derivative in `d2`. `d2` is a column vector the same length as `x`.

Remarks

For library models with closed forms, derivatives are calculated analytically. For all other models, the first derivative is calculated using the centered difference quotient

$$y' = \frac{y_{x+h} - y_{x-h}}{2h}$$

where x is the value at which the derivative is calculated, h is a small number (on the order of the cube root of `eps`), y_{x+h} is `fun` evaluated at $x+h$, and y_{x-h} is `fun` evaluated at $x - h$. The second derivative is calculated using the expression

$$y'' = \frac{y_{x+h} + y_{x-h} - 2y_x}{h^2}$$

Example

Create a baseline sinusoidal signal:

```
xdata = (0:.1:2*pi)';
y0 = sin(xdata);
```

Add noise to the signal:

```
noise = 2*y0.*randn(size(y0));  % Response-dependent
                   % Gaussian noise
ydata = y0 + noise;
```

Fit the noisy data with a custom sinusoidal model:

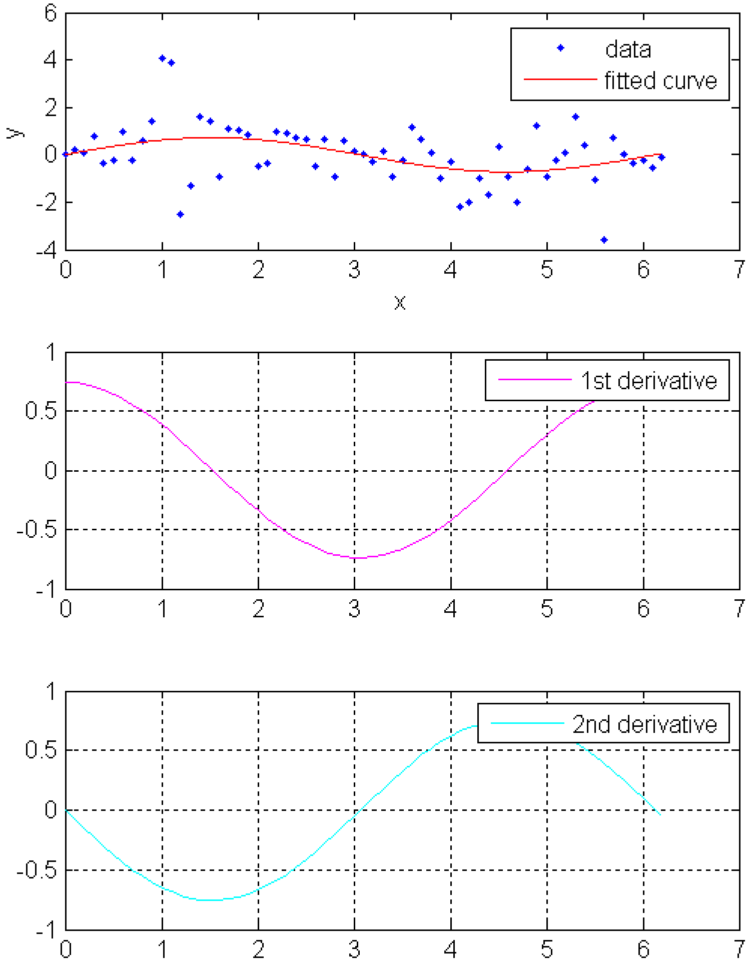
```
f = fittype('a*sin(b*x)');
fit1 = fit(xdata,ydata,f,'StartPoint',[1 1]);
```

Find the derivatives of the fit at the predictors:

```
[d1,d2] = differentiate(fit1,xdata);
```

Plot the data, the fit, and the derivatives:

```
subplot(3,1,1)
plot(fit1,xdata,ydata) % cfit plot method
subplot(3,1,2)
plot(xdata,d1,'m') % double plot method
grid on
legend('1st derivative')
subplot(3,1,3)
plot(xdata,d2,'c') % double plot method
grid on
legend('2nd derivative')
```



Note that derivatives can also be computed and plotted directly with the `cfit` `plot` method, as follows:

```
plot(fit1,xdata,ydata,{'fit','deriv1','deriv2'})
```

The `plot` method, however, does not return data on the derivatives.

See Also

`fit`, `plot`, `integrate`

Purpose

Exclude data from fit

Syntax

```
outliers = excludedata(xdata,ydata,MethodName,MethodValue)
```

Description

`outliers = excludedata(xdata,ydata,`*`MethodName`*`,MethodValue)` identifies data to be excluded from a fit using the specified *`MethodName`* and `MethodValue`. `outliers` is a logical vector, with `1` marking predictors (`xdata`) to exclude and `0` marking predictors to include. Supported *`MethodName`* and `MethodValue` pairs are given in the table below.

`MethodName`	**`MethodValue`**
`'box'`	A four-element vector specifying the edges of a closed box in the *xy*-plane, outside of which data is to be excluded from a fit. The vector has the form `[xmin xmax ymin ymax]`.
`'domain'`	A two-element vector specifying the endpoints of a closed interval on the *x*-axis, outside of which data is to be excluded from a fit. The vector has the form `[xmin xmax]`.
`'indices'`	A vector of indices specifying the data points to be excluded.
`'range'`	A two-element vector specifying the endpoints of a closed interval on the *y*-axis, outside of which data is to be excluded from a fit. The vector has the form `[ymin ymax]`.

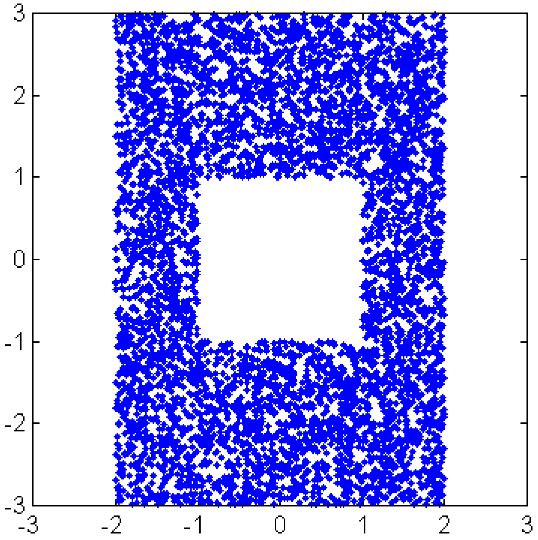
Remarks

You can combine data exclusion rules using logical operators. For example, to exclude data *inside* the box `[-1 1 -1 1]` or *outside* the domain `[-2 2]`, use:

```
outliers1 = excludedata(xdata,ydata,'box',[-1 1 -1 1]);
outliers2 = excludedata(xdata,ydata,'domain',[-2 2]);
outliers = ~outliers1|outliers2;
```

You can visualize the combined exclusion rule using random data:

```
xdata = -3 + 6*rand(1,1e4);
ydata = -3 + 6*rand(1,1e4);
plot(xdata(~outliers),ydata(~outliers),'.')
axis ([-3 3 -3 3])
axis square
```



Example

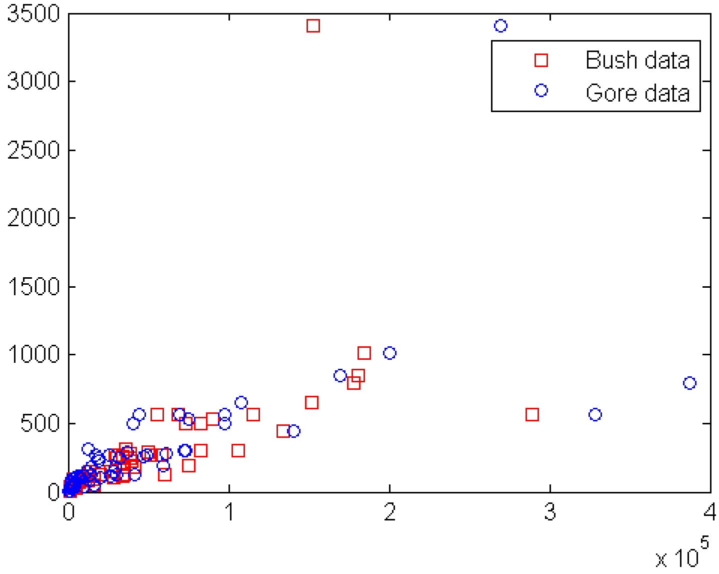
Load the vote counts and county names for the state of Florida from the 2000 U.S. presidential election:

```
load flvote2k
```

Use the vote counts for the two major party candidates, Bush and Gore, as predictors for the vote counts for third-party candidate Buchanan, and plot the scatters:

```
plot(bush,buchanan,'rs')
hold on
plot(gore,buchanan,'bo')
```

```
legend('Bush data','Gore data')
```



Assume a model where a fixed proportion of Bush or Gore voters choose to vote for Buchanan:

```
f = fittype({'x'})
f =
     Linear model:
       f(a,x) = a*x
```

Exclude the data from absentee voters, who did not use the controversial "butterfly" ballot:

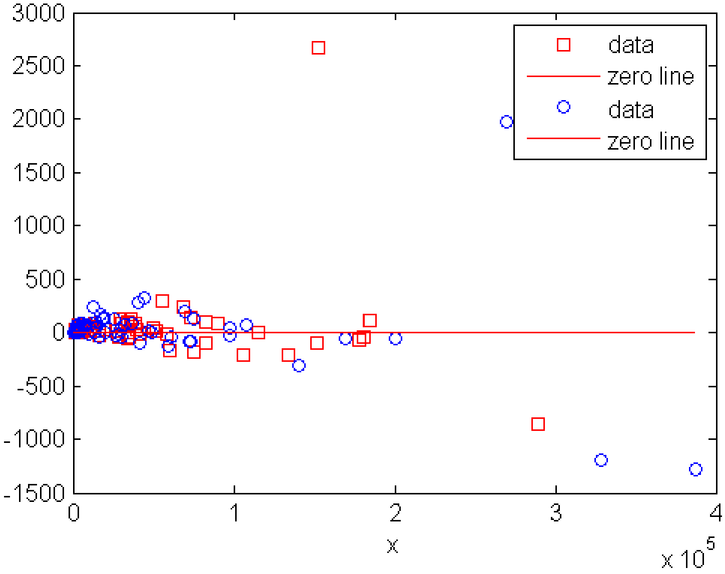
```
absentee = find(strcmp(counties,'Absentee Ballots'));
nobutterfly = excludedata(bush,buchanan,...
                          'indices',absentee);
```

Perform a bisquare weights robust fit of the model to the two data sets, excluding absentee voters:

```
bushfit = fit(bush,buchanan,f,...
                  'Exclude',nobutterfly,'Robust','on');
gorefit = fit(gore,buchanan,f,...
                  'Exclude',nobutterfly,'Robust','on');
```

Robust fits give outliers a low weight, so large residuals from a robust fit can be used to identify the outliers:

```
figure
plot(bushfit,bush,buchanan,'rs','residuals')
hold on
plot(gorefit,gore,buchanan,'bo','residuals')
```



The residuals in the plot above can be computed as follows:

```
bushres = buchanan - feval(bushfit,bush);
goreres = buchanan - feval(gorefit,gore);
```

Large residuals can be identified as those outside the range `[-500 500]`:

```
bushoutliers = excludedata(bush,bushres,...
                           'range',[-500 500]);
goreoutliers = excludedata(gore,goreres,...
                           'range',[-500 500]);
```

The outliers for the two data sets correspond to the following counties:

```
counties(bushoutliers)
ans =
    'Miami-Dade'
    'Palm Beach'

counties(goreoutliers)
ans =
    'Broward'
    'Miami-Dade'
    'Palm Beach'
```

Miami-Dade and Broward counties correspond to the largest predictor values. Palm Beach county, the only county in the state to use the “butterfly” ballot, corresponds to the largest residual values.

See Also

`fit`, `fitoptions`

Purpose

Evaluate cfit or fittype object

Syntax

```
y = feval(cfun,x)
y = feval(ffun,coeff1,coeff2,...,x)
```

Description

y = feval(cfun,x) evaluates the cfit object cfun at the predictor values in the column vector x and returns the response values in the column vector y.

y = feval(ffun,coeff1,coeff2,...,x) assigns the coefficients coeff1, coeff2, etc. to the fittype object ffun, evaluates it at the predictor values in the column vector x, and returns the response values in the column vector y. ffun cannot be a cfit object in this syntax. To evaluate cfit objects, use the first syntax.

Remark

cfit or fittype objects can call feval indirectly using the following functional syntax:

```
y = cfun(x) % cfit objects;
y = ffun(coef1,coef2,...,x) % fittype objects;
```

Example

```
f = fittype('a*x^2+b*exp(n*x)');
c = cfit(f,1,10.3,-1e2);
X = rand(2)
X =
    0.0579    0.8132
    0.3529    0.0099

y1 = feval(f,1,10.3,-1e2,X)
y1 =
    0.0349    0.6612
    0.1245    3.8422
y1 = f(1,10.3,-1e2,X)
y1 =
    0.0349    0.6612
    0.1245    3.8422
```

```
y2 = feval(c,X)
y2 =
    0.0349
    0.1245
    0.6612
    3.8422
y2 = c(X)
y2 =
    0.0349
    0.1245
    0.6612
    3.8422
```

See Also

`fit`, `fittype`, `cfit`

Purpose

Fit model to data

Syntax

```
cfun = fit(xdata,ydata,libname)
cfun = fit(...,PropName,PropVal,...)
cfun = fit(xdata,ydata,libname,options)
cfun = fit(xdata,ydata,ffun,...)
cfun = fit(...,'problem',vals)
[cfun,gof] = fit(...)
[cfun,gof,output] = fit(...)
```

Description

`cfun = fit(xdata,ydata,`*`libname`*`)` fits the data in the column vectors `xdata` and `ydata` with the library model specified by *`libname`*. `xdata` and `ydata` cannot contain `Inf` or `NaN`. Only the real parts of complex data are used in the fit. You can display library model names with the `cflibhelp` function. The fit result is returned as a `cfit` object `cfun`.

`cfun = fit(...,`*`PropName`*`,PropVal,...)` fits the data using specified property name/value pairs. You can display the supported property names and values for specific library models with the `fitoptions` function.

`cfun = fit(xdata,ydata,`*`libname`*`,options)` fits the data using the options specified by the fit options structure `options`. Fit options structures are created with the `fitoptions` function.

`cfun = fit(xdata,ydata,ffun,...)` fits the data with the `fittype` object `ffun`. `fittype` objects are created with the `fittype` function.

`cfun = fit(...,'problem',vals)` assigns `vals` to the problem-dependent parameters of the model before fitting. `vals` is a scalar or a cell array with one element per parameter.

`[cfun,gof] = fit(...)` returns goodness-of-fit statistics to the structure `gof`. The `gof` structure has the fields shown in the table below.

Field	Value
`sse`	Sum of squares due to error
`rsquare`	Coefficient of determination

Field	Value
dfe	Degrees of freedom
adjrsquare	Degree-of-freedom adjusted coefficient of determination
rmse	Root mean squared error (standard error)

`[cfun,gof,output] = fit(...)` returns the structure `output`, which contains information associated with the fitting algorithm. Fields depend on the algorithm. For example, the `output` structure for nonlinear least-squares algorithms has the fields shown in the table below.

Field	Value
numobs	Number of observations (response values)
numparam	Number of unknown parameters (coefficients) to fit
residuals	Vector of residuals
Jacobian	Jacobian matrix
exitflag	Describes the exit condition of the algorithm. Positive flags indicate convergence, within tolerances. Zero flags indicate that the maximum number of function evaluations or iterations was exceeded. Negative flags indicate that the algorithm did not converge to a solution.
iterations	Number of iterations
funcCount	Number of function evaluations
firstorderopt	Measure of first-order optimality (absolute maximum of gradient components)
algorithm	Fitting algorithm employed

Remarks

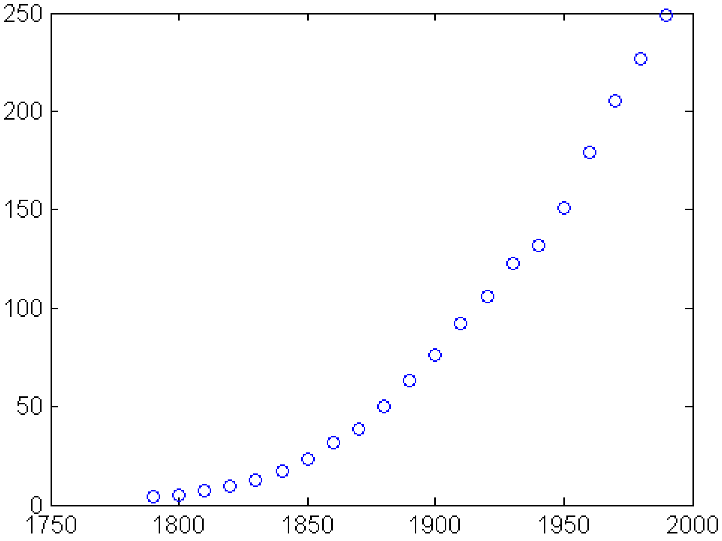
For some nonlinear library models (rational and Weibull), and all custom nonlinear models, default initial values for coefficients are selected uniformly at random from the interval (0,1). As a result, multiple fits using the same data and model may lead to different fitted coefficients. To avoid this, initial values for coefficients can be specified through a `fitoptions` structure or a vector value for the `StartPoint` property. Alternatively, the initial state of the random number generator `rand` can be set before fitting.

All other nonlinear library models automatically compute reasonable initial values. These initial values depend on the data, and are based on model-specific heuristics.

Example

Load and plot the data in `census.mat`:

```
load census
plot(cdate,pop,'o')
hold on
```



Create a fit options structure and a `fittype` object for the custom nonlinear model $y = a(x–b)^n$, where a and b are coefficients and n is a problem-dependent parameter:

```
s = fitoptions('Method','NonlinearLeastSquares',...
               'Lower',[0,0],...
               'Upper',[Inf,max(cdate)],...
               'Startpoint',[1 1]);
f = fittype('a*(x-b)^n','problem','n','options',s);
```

Fit the data using the fit options and a value of $n = 2$:

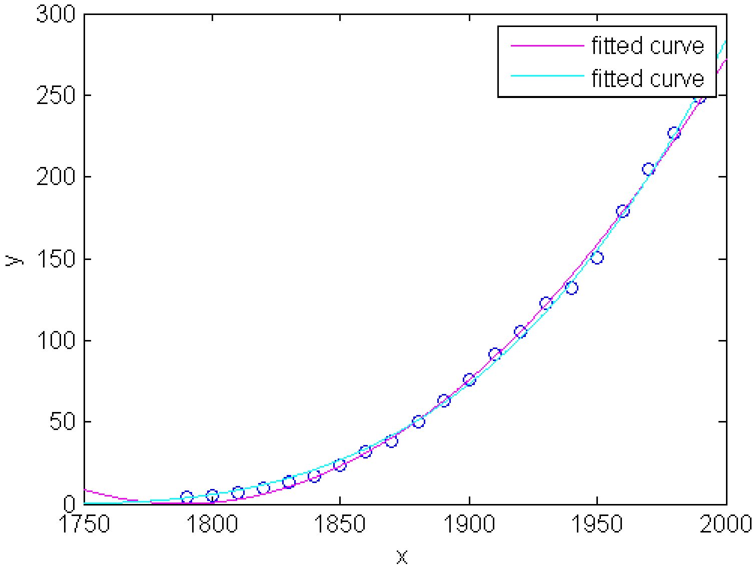
```
[c2,gof2] = fit(cdate,pop,f,'problem',2)
c2 =
     General model:
       c2(x) = a*(x-b)^n
     Coefficients (with 95% confidence bounds):
       a =    0.006092  (0.005743, 0.006441)
       b =        1789  (1784, 1793)
     Problem parameters:
       n =           2
gof2 =
           sse: 246.1543
       rsquare: 0.9980
           dfe: 19
    adjrsquare: 0.9979
          rmse: 3.5994
```

Fit the data using the fit options and a value of $n = 3$:

```
[c3,gof3] = fit(cdate,pop,f,'problem',3)
c3 =
     General model:
       c3(x) = a*(x-b)^n
     Coefficients (with 95% confidence bounds):
       a =  1.359e-005  (1.245e-005, 1.474e-005)
       b =        1725  (1718, 1731)
     Problem parameters:
           n =            3
gof3 =
             sse: 232.0058
         rsquare: 0.9981
             dfe: 19
      adjrsquare: 0.9980
            rmse: 3.4944
```

Plot the fit results with the data:

```
plot(c2,'m')
plot(c3,'c')
```



See Also

cflibhelp, fitoptions, fittype, feval, plot

Purpose

Create or modify fit options structure

Syntax

```
options = fitoptions
options = fitoptions(model)
options = fitoptions(model,fld1,val1,fld2,val2,...)
options = fitoptions('Method',method)
options =
fitoptions('Method',method,fld1,val1,fld2,val2,...)
newoptions = fitoptions(options,fld1,val1,fld2,val2,...)
newoptions = fitoptions(options1,options2)
```

Description

`options = fitoptions` creates the default fit options structure `options`. Fields in the options structure, listed in the table below with their default values, are supported by all fitting methods.

Field Name	Values
`Normalize`	Specifies whether the data is centered and scaled. Values are `'off'` or `'on'`. The default is `'off'`.
`Exclude`	A logical vector indicating data points to exclude from the fit. The `excludedata` function can be used to create this vector. The default is empty.
`Weights`	A vector of weights the same size as the response data. The default is empty.
`Method`	The fitting method. A complete list of supported fitting methods is given below. The default is `'None'`.

`options = fitoptions(model)` creates the default fit options structure for the library or custom model specified by the string `model`. You can display library model names with the `cflibhelp` function.

`options = fitoptions(model,`*`fld1`*`,val1,`*`fld2`*`,val2,...)` creates a fit options structure for the specified `model` with the fields specified by the strings *`fld1`*, *`fld2`*, ... set to the values `val1`, `val2`, ..., respectively.

`options = fitoptions('Method',`*`method`*`)` creates the default fit options structure for the fitting method specified by the string *`method`*. Supported fitting methods are listed in the table below.

method	Description
`'NearestInterpolant'`	Nearest neighbor interpolation
`'LinearInterpolant'`	Linear interpolation
`'PchipInterpolant'`	Piecewise cubic Hermite interpolation
`'CubicSplineInterpolant'`	Cubic spline interpolation
`'SmoothingSpline'`	Smoothing spline
`'LinearLeastSquares'`	Linear least squares
`'NonlinearLeastSquares'`	Nonlinear least squares

`options = fitoptions('Method',`*`method`*`,`*`fld1`*`,val1,`*`fld2`*`,val2,...)` creates the default fit options structure for the fitting method specified by the string *`method`* with the fields specified by the strings *`fld1`*, *`fld2`*, ... set to the values `val1`, `val2`, ..., respectively.

`newoptions = fitoptions(options,`*`fld1`*`,val1,`*`fld2`*`,val2,...)` modifies the existing fit options structure `options` by setting the fields specified by the strings *`fld1`*, *`fld2`*, ... set to the values `val1`, `val2`, ..., respectively. The new options structure is returned in `newoptions`.

`newoptions = fitoptions(options1,options2)` combines the input fit options structures `options1` and `options2` to create the output fit options structure `newoptions`. If the input structures have `Method` fields set to the same value, the nonempty values for the fields in `options2` override the corresponding values in `options1` in the output structure. If the input structures have `Method` fields set to different values, the output structure will have the same `Method` as `options1`, and only the values of the `Normalize`, `Exclude`, and `Weights` fields of `options2` will override the corresponding values in `options1`.

Remarks

Field values in a fit options structure can be referenced with the get method and assigned with the set method. For example:

```
options = fitoptions('fourier1');
get(options,'Method')
ans =
NonlinearLeastSquares
get(options,'MaxIter')
ans =
   400
set(options,'Maxiter',1e3);
get(options,'MaxIter')
ans =
        1000
```

Field values can also be referenced and assigned using the dot notation. For example:

```
options.MaxIter
ans =
        1000
options.MaxIter = 500;
options.MaxIter
ans =
   500
```

Additional Fit Options

Additional fields in the fit options structure, beyond the default fields Normalize, Exclude, Weights, and Method, are available according to the fitting method.

If the Method field has the value 'NearestInterpolant', 'LinearInterpolant', 'PchipInterpolant', or 'CubicSplineInterpolant', there are no additional fields in the fit options structure.

If the Method field has the value SmoothingSpline, the SmoothingParam field is available to configure the smoothing parameter. Its value must be between 0 and 1. The default value depends on the data set.

If the Method field has the value LinearLeastSquares, the additional fields available in the fit options structure are listed in the table below.

Field	Description
Robust	Specifies the robust linear least-squares fitting method to be used. Values are 'on', 'off', 'LAR', or 'Bisquare'. The default is 'off'. 'LAR' specifies the least absolute residual method and 'Bisquare' specifies the bisquare weights method. 'on' is equivalent to 'Bisquare', the default method.
Lower	A vector of lower bounds on the coefficients to be fitted. The default value is an empty vector, indicating that the fit is unconstrained by lower bounds. If bounds are specified, the vector length must equal the number of coefficients. Individual unconstrained lower bounds can be specified by -Inf.
Upper	A vector of upper bounds on the coefficients to be fitted. The default value is an empty vector, indicating that the fit is unconstrained by upper bounds. If bounds are specified, the vector length must equal the number of coefficients. Individual unconstrained upper bounds can be specified by Inf.

If the Method field has the value NonlinearLeastSquares, the additional fields available in the fit options structure are listed in the table below.

Property	Description
`Robust`	Specifies the robust linear least-squares fitting method to be used. Values are `'on'`, `'off'`, `'LAR'`, or `'Bisquare'`. The default is `'off'`. `'LAR'` specifies the least absolute residual method and `'Bisquare'` specifies the bisquare weights method. `'on'` is equivalent to `'Bisquare'`, the default method.
`Lower`	A vector of lower bounds on the coefficients to be fitted. The default value is an empty vector, indicating that the fit is unconstrained by lower bounds. If bounds are specified, the vector length must equal the number of coefficients. Individual unconstrained lower bounds can be specified by `-Inf`.
`Upper`	A vector of upper bounds on the coefficients to be fitted. The default value is an empty vector, indicating that the fit is unconstrained by upper bounds. If bounds are specified, the vector length must equal the number of coefficients. Individual unconstrained upper bounds can be specified by `Inf`.
`StartPoint`	A vector of initial values for the coefficients. The default value of `StartPoint` is an empty vector. If the default value is passed to the `fit` function, starting points for some library models are determined heuristically. For other models, the values are selected uniformly at random on the interval (0,1).
`Algorithm`	The algorithm used for the fitting procedure. Values are `'Levenberg-Marquardt'`, `'Gauss-Newton'`, or `'Trust-Region'`. The default is `'Trust-Region'`.

Property	Description
`DiffMaxChange`	The maximum change in coefficients for finite difference gradients. The default is 0.1.
`DiffMinChange`	The minimum change in coefficients for finite difference gradients. The default is 10^{-8}.
`Display`	Controls the display in the command window. `'notify'`, the default, displays output only if the fit does not converge. `'final'` displays only the final output. `'iter'` displays output at each iteration. `'off'` displays no output.
`MaxFunEvals`	The maximum number of evaluations of the model allowed. The default is 600.
`MaxIter`	The maximum number of iterations allowed for the fit. The default is 400.
`TolFun`	The termination tolerance on the model value. The default is 10^{-6}.
`TolX`	The termination tolerance on the coefficient values. The default is 10^{-6}.

Note For the fields `Upper`, `Lower`, and `StartPoint`, the order of the entries in the vector value is the order of the coefficients returned by the `coeffnames` method. For example, if

```
f = fittype('b*x^2+c*x+a');
coeffnames(f)
ans =
    'a'
    'b'
    'c'
```

then setting

```
options.StartPoint = [1 3 5];
```

assigns initial values to the coefficients as follows: `a = 1`, `b = 3`, `c = 5`. Note that this is not the order of the coefficients in the expression used to create `f` with `fittype`.

Example

Create the default fit options structure and set the option to center and scale the data before fitting:

```
options = fitoptions;
options.Normal = 'on';
options
options =
    Normalize: 'on'
    Exclude: [1x0 double]
    Weights: [1x0 double]
    Method: 'None'
```

Modifying the default fit options structure is useful when you want to set the `Normalize`, `Exclude`, or `Weights` fields, and then fit your data using the same options with different fitting methods. For example:

```
load census
```

```
f1 = fit(cdate,pop,'poly3',options);
f2 = fit(cdate,pop,'exp1',options);
f3 = fit(cdate,pop,'cubicsp',options);
```

Data-dependent fit options are returned in the third output argument of the `fit` function. For example:

```
[f,gof,out] = fit(cdate,pop,'smooth');
smoothparam = out.p
smoothparam =
    0.0089
```

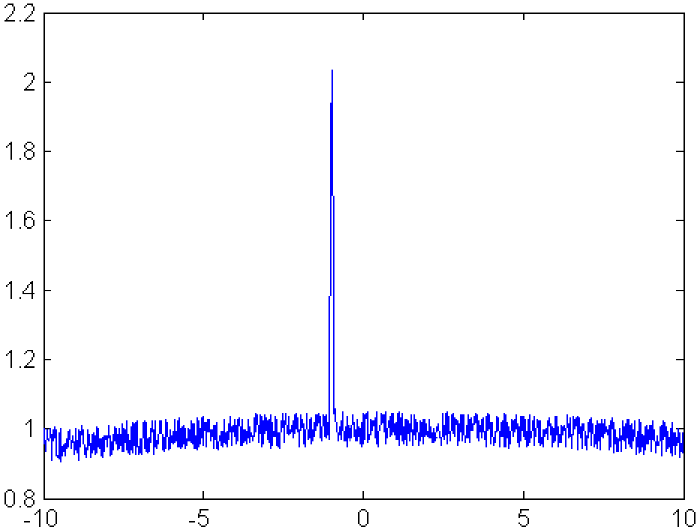
The default smoothing parameter can be modified for a new fit:

```
options = fitoptions('Method','Smooth',...
                     'SmoothingParam',0.0098);
[f,gof,out] = fit(cdate,pop,'smooth',options);
```

Example

Create a noisy sum of two Gaussian peaks—one with a small width, and one with a large width:

```
a1 = 1; b1 = -1; c1 = 0.05;
a2 = 1; b2 = 1; c2 = 50;
x = (-10:0.02:10)';
gdata = a1*exp(-((x-b1)/c1).^2) + ...
        a2*exp(-((x-b2)/c2).^2) + ...
        0.1*(rand(size(x))-.5);
plot(x,gdata)
```



Fit the data using the two-term Gaussian library model:

```
f = fittype('gauss2');
gfit = fit(x,gdata,f)
gfit =
     General model Gauss2:
       gfit(x) =  a1*exp(-((x-b1)/c1)^2) +
                  a2*exp(-((x-b2)/c2)^2)
     Coefficients (with 95% confidence bounds):
       a1 =    -0.05388  (-0.136, 0.02826)
       b1 =      -2.651  (-2.718, -2.584)
       c1 =     0.05373  (-0.04106, 0.1485)
       a2 =       1.012  (1.006, 1.018)
       b2 =      0.6703  (0.06681, 1.274)
       c2 =        41.2  (36.54, 45.85)
```

The algorithm is having difficulty, as indicated by the wide confidence intervals for some of the coefficients. To help the algorithm, we could specify lower bounds for the nonnegative amplitudes `a1`, `a2` and widths `c1`, `c2`:

```
options = fitoptions('gauss2');
options.Lower = [0 -Inf 0 0 -Inf 0];
```

Recompute the fit with the bound constraints on the coefficients:

```
gfit = fit(x,gdata,ftype,options)
gfit =
     General model Gauss2:
       gfit(x) =  a1*exp(-((x-b1)/c1)^2) +
                  a2*exp(-((x-b2)/c2)^2)
     Coefficients (with 95% confidence bounds):
       a1 =       1.003  (0.9641, 1.042)
       b1 =          -1  (-1.002, -0.9987)
       c1 =     0.04972  (0.04748, 0.05197)
       a2 =       1.002  (0.999, 1.004)
       b2 =       1.136  (0.725, 1.547)
       c2 =       48.89  (45.32, 52.47)
```

This is a much better fit. The fit can be further improved by assigning reasonable values to other fields in the fit options structure.

See Also

cflibhelp, fit, get, set

Purpose

Constructor for `fittype` object

Syntax

```
ffun = fittype(libname)
ffun = fittype(expr)
ffun = fittype({expr1,...,exprn})
ffun = fittype(expr,PropName,PropVal,...)
ffun = fittype({expr1,...,exprn},PropName,PropVal,...)
```

Description

`ffun = fittype(libname)` constructs the `fittype` object `ffun` for the library model specified by `libname`. You can display library model names with the `cflibhelp` function.

`ffun = fittype(expr)` constructs the `fittype` object `ffun` for the custom nonlinear model specified by the expression in the string `expr`. By default, the independent variable is assumed to be `x` and the dependent variable is assumed to be `y`. All other variables are assumed to be coefficients. All coefficients must be scalars.

Note The following coefficient names are not allowed in the expression string `expr`: `i`, `j`, `pi`, `inf`, `nan`, `eps`.

`ffun = fittype({expr1,...,exprn})` constructs the `fittype` object `ffun` for the custom linear model with terms specified by the expressions in the strings `expr1`, `expr2`, ... , `exprn`. Coefficients are not included in the expressions for the terms. If there is a constant term, use `'1'` as the corresponding expression in the cell array.

Note `islinear` assumes that all models specified with the syntax `ffun = fittype(expr)` are nonlinear models. To create a linear model with `fittype` that will be recognized as linear by `islinear` (and, importantly, by the algorithms of `fit`), use the syntax `ffun = fittype({expr1,...,exprn})`.

`ffun = fittype(expr,`*`PropName`*`,PropVal,...)` or `ffun = fittype({expr1,...,exprn},`*`PropName`*`,PropVal,...)` constructs the `fittype` object `ffun` using specified property name/value pairs. Supported property names and values are given in the table below.

PropName	**PropVal**
`'coefficients'`	The coefficient names. Use a cell array if there are multiple names. The following names are not allowed: `i`, `j`, `pi`, `inf`, `nan`, `eps`.
`'dependent'`	The dependent (response) variable name
`'independent'`	The independent (predictor) variable name
`'options'`	The default fit options for the object
`'problem'`	The problem-dependent (fixed) parameter names. Use a cell array if there are multiple names. The default is none.

Example

Construct a `fittype` object for the `rat33` library model:

```
f = fittype('rat33')
f =
   General model Rat33:
   f(p1,p2,p3,p4,q1,q2,q3,x) =
         (p1*x^3 + p2*x^2 + p3*x + p4)/
                (x^3 + q1*x^2 + q2*x + q3)
```

Construct a `fittype` object for a custom nonlinear model, designating `n` as a problem-dependent parameter and `u` as the independent variable:

```
g = fittype('a*u+b*exp(n*u)',...
            'problem','n',...
            'independent','u')
g =
     General model:
       g(a,b,n,u) = a*u+b*exp(n*u)
```

Construct a `fittype` object for a custom linear model, specifying the names of the coefficients:

```
h = fittype({'cos(x)','1'},'coefficients',{'a1','a2'})
h =
     Linear model:
       h(a1,a2,x) = a1*cos(x) + a2
```

See Also

`fit`, `cfit`

formula

Purpose

Formula of `cfit` or `fittype` object

Syntax

```
formula(fun)
```

Description

`formula(fun)` returns the formula of the `cfit` or `fittype` object `fun` as a character array.

Example

```
f = fittype('weibull');
formula(f)
ans =
a*b*x^(b-1)*exp(-a*x^b)

g = fittype('cubicspline');
formula(g)
ans =
piecewise polynomial
```

See Also

`fittype`, `coeffnames`, `numcoeffs`, `probnames`, `coeffvalues`

Purpose

Get fit options structure field names and values

Syntax

```
get(options)
s = get(options)
value = get(options,fld)
```

Description

`get(options)` displays all field names and values of the fit options structure `options`.

`s = get(options)` returns a copy of the fit options structure `options` as the structure `s`.

`value = get(options,*fld*)` returns the value of the field `*fld*` of the fit options structure `options`. `*fld*` can be a cell array of strings, in which case `value` is also a cell array.

Example

```
options = fitoptions('fourier1');
get(options,'Method')
ans =
NonlinearLeastSquares
get(options,'MaxIter')
ans =
   400
set(options,'Maxiter',1e3);
get(options,'MaxIter')
ans =
        1000
```

Field values can also be referenced and assigned using the dot notation. For example:

```
options.MaxIter
ans =
        1000
options.MaxIter = 500;
options.MaxIter
ans =
   500
```

See Also

fitoptions, set

Purpose

Independent variable of cfit or fittype object

Syntax

```
indep = indepnames(fun)
```

Description

indep = indepnames(fun) returns the (single) independent variable name of the cfit or fittype object fun as a 1-by-1 cell array of strings indep.

Example

```
f1 = fittype('a*x^2+b*exp(n*x)');
indep1 = indepnames(f1)
indep1 =
    'x'

f2 = fittype('a*x^2+b*exp(n*x)','independent','n');
indep2 = indepnames(f2)
indep2 =
    'n'
```

See Also

dependnames, fittype, formula

integrate

Purpose

Integrate `cfit` object

Syntax

```
int = integrate(fun,x,x0)
```

Description

`int = integrate(fun,x,x0)` integrates the `cfit` object `fun` at the points specified by the vector `x`, starting from `x0`, and returns the result in `int`. `int` is a vector the same size as `x`. `x0` is a scalar.

Example

Create a baseline sinusoidal signal:

```
xdata = (0:.1:2*pi)';
y0 = sin(xdata);
```

Add noise to the signal:

```
noise = 2*y0.*randn(size(y0)); % Response-dependent
                               % Gaussian noise
ydata = y0 + noise;
```

Fit the noisy data with a custom sinusoidal model:

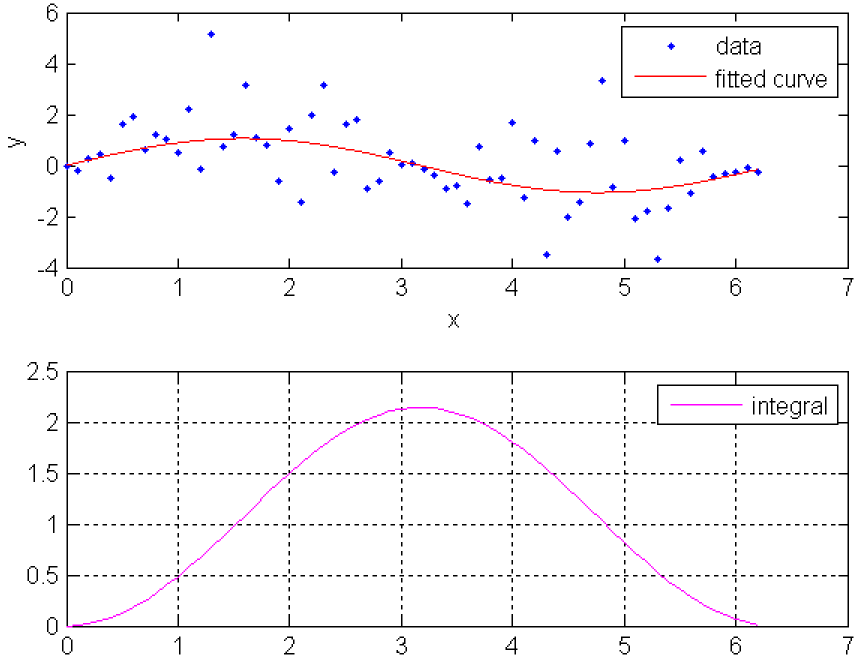
```
f = fittype('a*sin(b*x)');
fit1 = fit(xdata,ydata,f,'StartPoint',[1 1]);
```

Find the integral of the fit at the predictors:

```
int = integrate(fit1,xdata,0);
```

Plot the data, the fit, and the integral:

```
subplot(2,1,1)
plot(fit1,xdata,ydata) % cfit plot method
subplot(2,1,2)
plot(xdata,int,'m') % double plot method
grid on
legend('integral')
```



Note that integrals can also be computed and plotted directly with the `cfit` `plot` method, as follows:

```
plot(fit1,xdata,ydata,{'fit','integral'})
```

The `plot` method, however, does not return data on the integral.

See Also

`fit`, `plot`, `differentiate`

islinear

Purpose

Determine if `cfit` or `fittype` object is linear

Syntax

```
flag = islinear(fun)
```

Description

`flag = islinear(fun)` returns a `flag` of `1` if the `cfit` or `fittype` object `fun` represents a linear model, and a `flag` of `0` if it does not.

Note `islinear` assumes that all custom models specified by the `fittype` function using the syntax `ftype = fittype('expr')` are nonlinear models. To create a linear model with `fittype` that will be recognized as linear by `islinear` (and, importantly, by the algorithms of `fit`), use the syntax `ftype = fittype({'expr1','expr2',...,'exprn'})`.

Example

```
f = fittype('a*x+b')
f =
     General model:
       f(a,b,x) = a*x+b

g = fittype({'x','1'})
g =
     Linear model:
       g(a,b,x) = a*x + b

h = fittype('poly1')
h =
     Linear model Poly1:
       h(p1,p2,x) = p1*x + p2

islinear(f)
ans =
     0
islinear(g)
ans =
      1
islinear(h)
ans =
      1
```

See Also

`fittype`

numargs

Purpose

Number of input arguments of `cfit` or `fittype` object

Syntax

```
nargs = numargs(fun)
```

Description

`nargs = numargs(fun)` returns the number of input arguments `nargs` of the `cfit` or `fittype` object `fun`.

Example

```
f = fittype('a*x^2+b*exp(n*x)');
nargs = numargs(f)
nargs =
     4
args = argnames(f)
args =
    'a'
    'b'
    'n'
    'x'
```

See Also

`fittype`, `formula`, `argnames`

Purpose

Number of coefficients of `cfit` or `fittype` object

Syntax

```
ncoeffs = numcoeffs(fun)
```

Description

`ncoeffs = numcoeffs(fun)` returns the number of coefficients `ncoeffs` of the `cfit` or `fittype` object `fun`.

Example

```
f = fittype('a*x^2+b*exp(n*x)');
ncoeffs = numcoeffs(f)
ncoeffs =
     3
coeffs = coeffnames(f)
coeffs =
    'a'
    'b'
    'n'
```

See Also

`fittype`, `formula`, `coeffnames`

Purpose

Plot `cfit` object

Syntax

```
plot(fun)
plot(fun,xdata,ydata)
plot(fun,xdata,ydata,DataLineSpec)
plot(fun,FitLineSpec,xdata,ydata,DataLineSpec)
plot(fun,xdata,ydata,outliers)
plot(fun,xdata,ydata,outliers,OutlierLineSpec)
plot(...,ptype,...)
plot(...,ptype,level)
h = plot(...)
```

Description

`plot(fun)` plots the `cfit` object `fun` over the domain of the current axes, if any. If there are no current axes, and `fun` is an output from the `fit` function, the plot is over the domain of the fitted data.

`plot(fun,xdata,ydata)` plots `fun` together with the predictor data `xdata` and the response data `ydata`.

`plot(fun,xdata,ydata,DataLineSpec)` plots the predictor and response data using the color, marker symbol, and line style specified by the `DataLineSpec` formatting string. `DataLineSpec` strings take the same values as `LineSpec` strings used by the MATLAB `plot` function.

`plot(fun,FitLineSpec,xdata,ydata,DataLineSpec)` plots `fun` using the color, marker symbol, and line style specified by the `FitLineSpec` formatting string, and plots `xdata` and `ydata` using the color, marker symbol, and line style specified by the `DataLineSpec` formatting string. `FitLineSpec` and `DataLineSpec` strings take the same values as `LineSpec` strings used by the MATLAB `plot` function.

`plot(fun,xdata,ydata,outliers)` plots data indicated by `outliers` in a different color. `outliers` is a logical array the same size as `xdata` and `ydata`. `outliers` can be computed with the `excludedata` function.

`plot(fun,xdata,ydata,outliers,OutlierLineSpec)` plots `outliers` using the color, marker symbol, and line style specified by the `OutlierLineSpec`. `OutlierLineSpec` strings take the same values as `LineSpec` strings used by the MATLAB `plot` function.

plot(...,ptype,...) uses the plot type specified by ptype. Supported plot types are:

- 'fit' — Data and fit (default)
- 'predfunc' — Data and fit with prediction bounds for the fit
- 'predobs' — Data and fit with prediction bounds for new observations
- 'residuals' — Residuals
- 'stresiduals' — Standardized residuals (residuals divided by their standard deviation).
- 'deriv1' — First derivative of the fit
- 'deriv2' — Second derivative of the fit
- 'integral' — Integral of the fit

Plot types can be single or multiple, with multiple plot types specified as a cell array of strings. With a single plot type, plot draws to the current axes and can be used with commands like hold and subplot. With multiple plot types, plot creates subplots for each plot type.

plot(...,ptype,level) plots prediction intervals with a confidence level specified by level. level must be between 0 and 1. The default value of level is 0.95.

h = plot(...) returns a vector of handles to the plotted objects.

Example

Create a baseline sinusoidal signal:

```
xdata = (0:0.1:2*pi)';
y0 = sin(xdata);
```

Add noise to the signal with non-constant variance:

```
% Response-dependent Gaussian noise
gnoise = y0.*randn(size(y0));
```

```
% Salt-and-pepper noise
spnoise = zeros(size(y0));
p = randperm(length(y0));
sppoints = p(1:round(length(p)/5));
spnoise(sppoints) = 5*sign(y0(sppoints));

ydata = y0 + gnoise + spnoise;
```

Fit the noisy data with a baseline sinusoidal model:

```
f = fittype('a*sin(b*x)');
fit1 = fit(xdata,ydata,f,'StartPoint',[1 1]);
```

Identify “outliers” as points at a distance greater than 1.5 standard deviations from the baseline model, and refit the data with the outliers excluded:

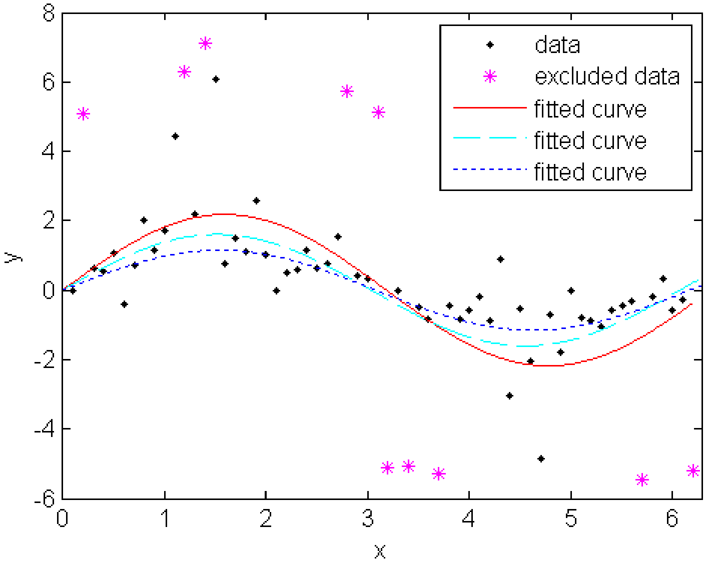
```
fdata = feval(fit1,xdata);
I = abs(fdata - ydata) > 1.5*std(ydata);
outliers = excludedata(xdata,ydata,'indices',I);

fit2 = fit(xdata,ydata,f,'StartPoint',[1 1],...
           'Exclude',outliers);
```

Compare the effect of excluding the outliers with the effect of giving them lower bisquare weight in a robust fit:

```
fit3 = fit(xdata,ydata,f,'StartPoint',[1 1],'Robust','on');
```

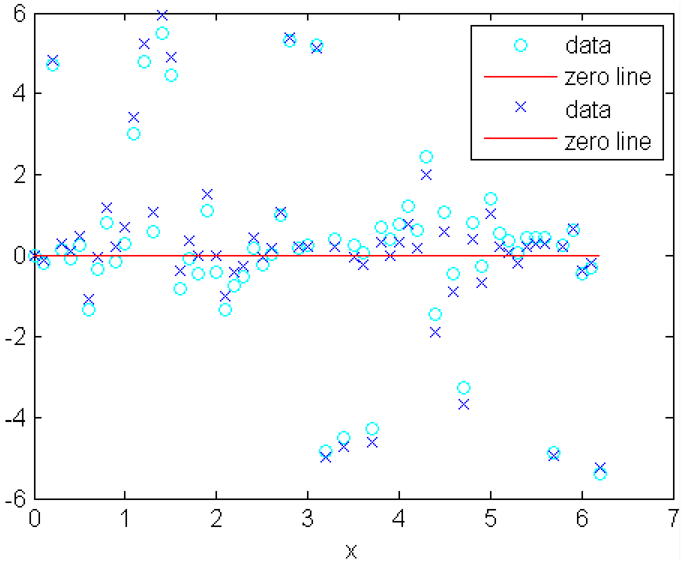
Plot the data, the outliers, and the results of the fits:

```
plot(fit1,'r-',xdata,ydata,'k.',outliers,'m*')
hold on
plot(fit2,'c--')
plot(fit3,'b:')
xlim([0 2*pi])
```



Plot the residuals for the two fits considering outliers:

```
figure
plot(fit2,xdata,ydata,'co','residuals')
hold on
plot(fit3,xdata,ydata,'bx','residuals')
```



See Also `cftool`, `excludedata`, `fit`, `differentiate`, `integrate`

Purpose

Prediction intervals for cfit object

Syntax

```
ci = predint(fitresult,x)
ci = predint(fitresult,x,level)
ci = predint(fitresult,x,level,intopt,simopt)
[ci,y] = predint(...)
```

Description

`ci = predint(fitresult,x)` returns upper and lower 95% prediction bounds for response values associated with the `cfit` object `fitresult` at the new predictor values specified by the vector `x`. `fitresult` must be an output from the `fit` function to contain the necessary information for `ci`. `ci` is an `n`-by-2 array where `n = length(x)`. The left column of `ci` contains the lower bound for each coefficient; the right column contains the upper bound.

`ci = predint(fitresult,x,level)` returns prediction bounds with a confidence level specified by `level`. `level` must be between `0` and `1`. The default value of `level` is `0.95`.

`ci = predint(fitresult,x,level,`*`intopt`*`,`*`simopt`*`)` specifies the type of bounds to compute.

`intopt` is one of

- `'observation'` — Bounds for a new observation (default)
- `'functional'` — Bounds for the fitted curve

`simopt` is one of

- `'off'` — Non-simultaneous bounds (default)
- `'on'` — Simultaneous bounds

Observation bounds are wider than functional bounds because they measure the uncertainty of predicting the fitted curve plus the random variation in the new observation. Non-simultaneous bounds are for individual elements of `x`; simultaneous bounds are for all elements of `x`.

`[ci,y] = predint(...)` returns the response values `y` predicted by `fitresult` at the predictors in `x`.

Example

Generate data with an exponential trend:

```
x = (0:0.2:5)';
y = 2*exp(-0.2*x) + 0.5*randn(size(x));
```

Fit the data using a single-term exponential:

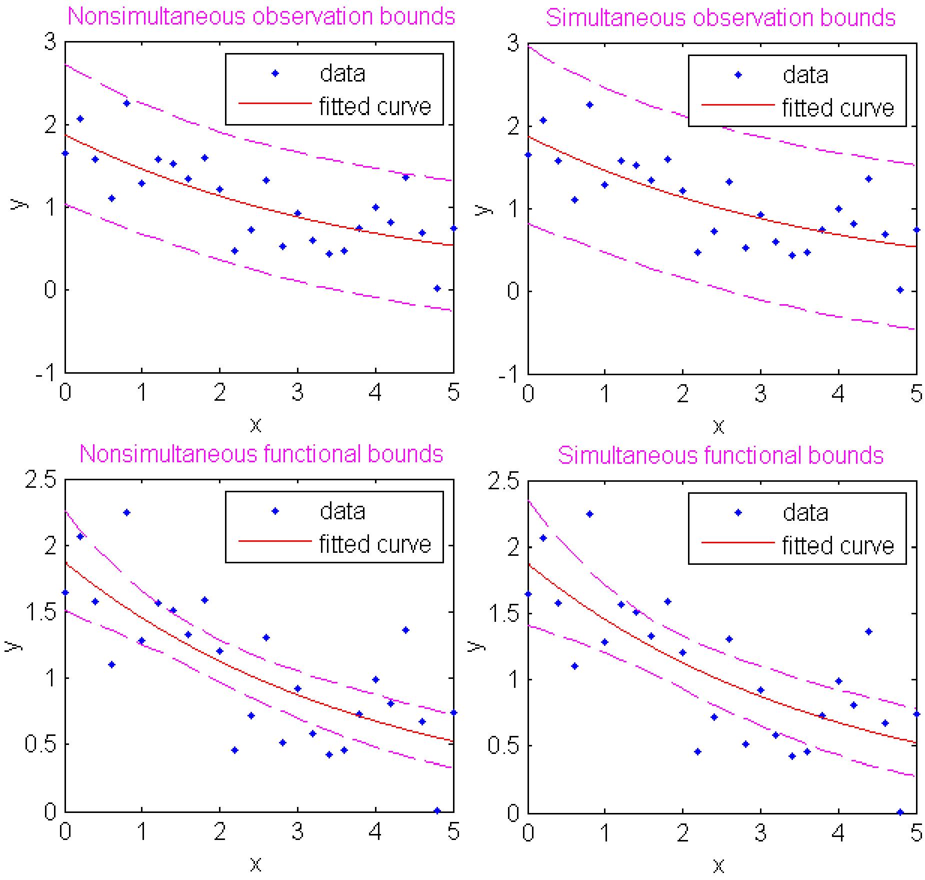
```
fitresult = fit(x,y,'exp1');
```

Compute prediction intervals:

```
p11 = predint(fitresult,x,0.95,'observation','off');
p12 = predint(fitresult,x,0.95,'observation','on');
p21 = predint(fitresult,x,0.95,'functional','off');
p22 = predint(fitresult,x,0.95,'functional','on');
```

Plot the data, fit, and prediction intervals:

```
subplot(2,2,1)
plot(fitresult,x,y),hold on,plot(x,p11,'m--'),xlim([0 5])
title('Nonsimultaneous observation bounds','Color','m')
subplot(2,2,2)
plot(fitresult,x,y),hold on,plot(x,p12,'m--'),xlim([0 5])
title('Simultaneous observation bounds','Color','m')
subplot(2,2,3)
plot(fitresult,x,y),hold on,plot(x,p21,'m--'),xlim([0 5])
title('Nonsimultaneous functional bounds','Color','m')
subplot(2,2,4)
plot(fitresult,x,y),hold on,plot(x,p22,'m--'),xlim([0 5])
title('Simultaneous functional bounds','Color','m')
```



See Also

`confint`, `fit`, `plot`

probnames

Purpose

Problem-dependent parameter names of `cfit` or `fittype` object

Syntax

```
pnames = probnames(fun)
```

Description

`pnames = probnames(fun)` returns the names of the problem-dependent (fixed) parameters of the `cfit` or `fittype` object `fun` as a cell array of strings.

Example

```
f = fittype('(x-a)^n + b','problem',{'a','b'});
coeffnames(f)
ans =
    'n'
probnames(f)
ans =
    'a'
    'b'

load census

c = fit(cdate,pop,f,'problem',{cdate(1),pop(1)},...
       'StartPoint',2);
coeffvalues(c)
ans =
    0.9877
probvalues(c)
ans =
  1.0e+003 *
    1.7900    0.0039
```

See Also

`fittype`, `coeffnames`, `probvalues`

Purpose

Problem-dependent parameter values of cfit object

Syntax

```
pvals = probvalues(fun)
```

Description

pvals = probvalues(fun) returns the values of the problem-dependent (fixed) parameters of the cfit object fun as a row vector.

Example

```
f = fittype('(x-a)^n + b','problem',{'a','b'});
coeffnames(f)
ans =
    'n'
probnames(f)
ans =
    'a'
    'b'

load census

c = fit(cdate,pop,f,'problem',{cdate(1),pop(1)},...
        'StartPoint',2);
coeffvalues(c)
ans =
    0.9877
probvalues(c)
ans =
  1.0e+003 *
    1.7900    0.0039
```

See Also

fit, fittype, probnames

set

Purpose

Assign values in fit options structure

Syntax

```
set(options)
s = set(options)
set(options,fld1,val1,fld2,val2,...)
set(options,flds,vals)
```

Description

set(options) displays all field names of the fit options structure options. If a field has a finite list of possible string values, these values are also displayed.

s = set(options) returns a structure s with the same field names as options. If a field has a finite list of possible string values, the value of the field in s is a cell array containing the possible string values. If a field does not have a finite list of possible string values, the value of the field in s is an empty cell array.

set(options,*fld1*,val1,*fld2*,val2,...) sets the fields specified by the strings *fld1*, *fld2*, ... to the values val1, val2, ..., respectively.

set(options,*flds*,vals) sets the fields specified by the cell array of strings *flds* to the corresponding values in the cell array vals.

Example

Create a custom nonlinear model, and create a default fit options structure for the model:

```
f = fittype('a*x^2+b*exp(n*c*x)','problem','n');
options = fitoptions(f);
```

Set the Robust and Normalize fields of the fit options structure using field name/value pairs:

```
set(options,'Robust','LAR','Normalize','On')
```

Set the Display, Lower, and Algorithm fields of the fit options structure using cell arrays of field names/values:

```
set(opts,{'Disp','Low','Alg'},...
         {'Final',[0 0 0],'Levenberg'})
```

See Also

`fitoptions`, `get`

Purpose

Smooth response data

Syntax

```
yy = smooth(y)
yy = smooth(y,span)
yy = smooth(y,method)
yy = smooth(y,span,method)
yy = smooth(y,'sgolay',degree)
yy = smooth(y,span,'sgolay',degree)
yy = smooth(x,y,...)
```

Description

`yy = smooth(y)` smooths the data in the column vector `y` using a moving average filter. Results are returned in the column vector `yy`. The default span for the moving average is `5`.

The first few elements of `yy` are given by

```
yy(1) = y(1)
yy(2) = (y(1) + y(2) + y(3))/3
yy(3) = (y(1) + y(2) + y(3) + y(4) + y(5))/5
yy(4) = (y(2) + y(3) + y(4) + y(5) + y(6))/5
...
```

Because of the way endpoints are handled, the result differs from the result returned by the `filter` function.

`yy = smooth(y,span)` sets the span of the moving average to `span`. `span` must be odd.

`yy = smooth(y,`*`method`*`)` smooths the data in `y` using the method *`method`* and the default span. Supported values for *`method`* are listed in the table below.

`method`	Description
`'moving'`	Moving average (default). A lowpass filter with filter coefficients equal to the reciprocal of the span.

method	Description
'lowess'	Local regression using weighted linear least squares and a 1st degree polynomial model
'loess'	Local regression using weighted linear least squares and a 2nd degree polynomial model
'sgolay'	Savitzky-Golay filter. A generalized moving average with filter coefficients determined by an unweighted linear least-squares regression and a polynomial model of specified degree (default is 2). The method can accept nonuniform predictor data.
'rlowess'	A robust version of 'lowess' that assigns lower weight to outliers in the regression. The method assigns zero weight to data outside six mean absolute deviations.
'rloess'	A robust version of 'loess' that assigns lower weight to outliers in the regression. The method assigns zero weight to data outside six mean absolute deviations.

yy = smooth(y,span,*method*) sets the span of *method* to span. For the loess and lowess methods, span is a percentage of the total number of data points, less than or equal to 1. For the moving average and Savitzky-Golay methods, span must be odd (an even span is automatically reduced by 1).

yy = smooth(y,'sgolay',degree) uses the Savitzky-Golay method with polynomial degree specified by degree.

yy = smooth(y,span,'sgolay',degree) uses the number of data points specified by span in the Savitzky-Golay calculation. span must be odd and degree must be less than span.

yy = smooth(x,y,...) additionally specifies x data. If x is not provided, methods that require x data assume x = 1:length(y). You should specify x data when it is not uniformly spaced or sorted.

If x is not uniform and you do not specify method, lowess is used. If the smoothing method requires x to be sorted, the sorting occurs automatically.

Remarks

Another way to generate smoothed data is to fit it with a smoothing spline. Refer to the fit function for more information.

Example

Load the data in count.dat:

```
load count.dat
```

The 24-by-3 array count contains traffic counts at three intersections for each hour of the day.

First, use a moving average filter with a 5-hour span to smooth all of the data at once (by linear index) :

```
c = smooth(count(:));
C1 = reshape(c,24,3);
```

Plot the original data and the smoothed data:

```
subplot(3,1,1)
plot(count,':');
hold on
plot(C1,'-');
title('Smooth C1 (All Data)')
```

Second, use the same filter to smooth each column of the data separately:

```
C2 = zeros(24,3);
for I = 1:3,
    C2(:,I) = smooth(count(:,I));
end
```

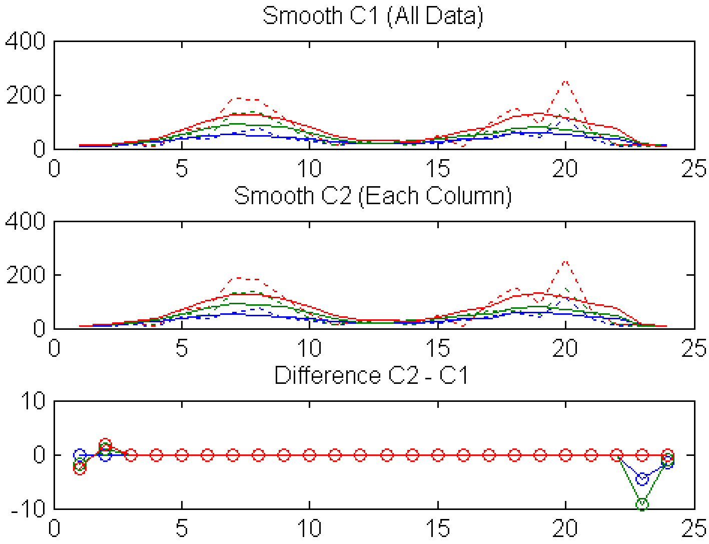
Again, plot the original data and the smoothed data:

```
subplot(3,1,2)
```

```
plot(count,':');
hold on
plot(C2,'-');
title('Smooth C2 (Each Column)')
```

Plot the difference between the two smoothed data sets:

```
subplot(3,1,3)
plot(C2 - C1,'o-')
title('Difference C2 - C1')
```



Note the additional end effects from the 3-column smooth.

Example

Create noisy data with outliers:

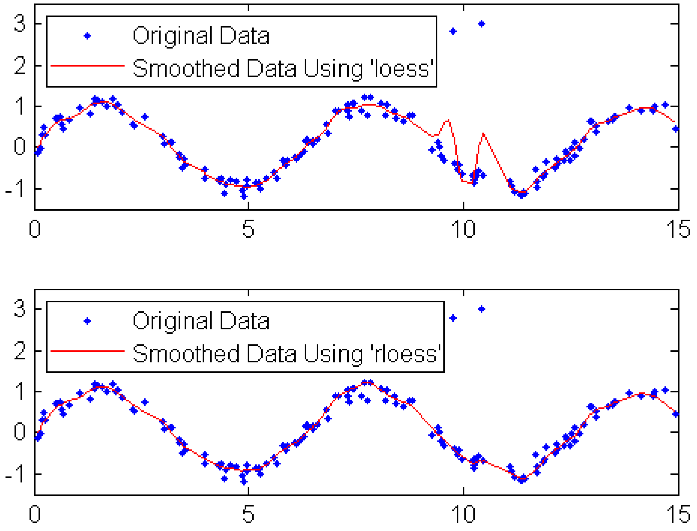
```
x = 15*rand(150,1);
y = sin(x) + 0.5*(rand(size(x))-0.5);
y(ceil(length(x)*rand(2,1))) = 3;
```

Smooth the data using the `loess` and `rloess` methods with a span of 10%:

```
yy1 = smooth(x,y,0.1,'loess');
yy2 = smooth(x,y,0.1,'rloess');
```

Plot original data and the smoothed data.

```
[xx,ind] = sort(x);
subplot(2,1,1)
plot(xx,y(ind),'b.',xx,yy1(ind),'r-')
set(gca,'YLim',[-1.5 3.5])
legend('Original Data','Smoothed Data Using ''loess''',...
       'Location','NW')
subplot(2,1,2)
plot(xx,y(ind),'b.',xx,yy2(ind),'r-')
set(gca,'YLim',[-1.5 3.5])
legend('Original Data','Smoothed Data Using ''rloess''',...
       'Location','NW')
```



Note that the outliers have less influence on the robust method.

See Also `fit`, `sort`

Purpose

Name of `cfit` or `fittype` object

Syntax

```
name = type(fun)
```

Description

`name = type(fun)` returns the custom or library name `name` of the `cfit` or `fittype` object `fun` as a character array.

Example

```
f = fittype('a*x^2+b*exp(n*x)');
category(f)
ans =
custom
type(f)
ans =
customnonlinear

g = fittype('fourier4');
category(g)
ans =
library
type(g)
ans =
fourier4
```

See Also

`fittype`, `category`, `cflibhelp`

Bibliography

[1] Barber, C. B., D.P. Dobkin, and H.T. Huhdanpaa. "The Quickhull Algorithm for Convex Hulls." *ACM Transactions on Mathematical Software*. Vol. 22, No. 4, 1996, pp. 469–483.

[2] Bevington, P.R., and D.K. Robinson. *Data Reduction and Error Analysis for the Physical Sciences*, 2nd Ed. Boston: WCB/McGraw-Hill, 1992.

[3] Branch, M.A., T.F. Coleman, and Y. Li. "A Subspace, Interior, and Conjugate Gradient Method for Large-Scale Bound-Constrained Minimization Problems." *SIAM Journal on Scientific Computing*. Vol. 21, Number 1, 1999, pp. 1–23.

[4] Carroll, R.J., and Ruppert, D. *Transformations and Weighting in Regression*. London: Chapman & Hall, 1988.

[5] Chambers, J., W.S. Cleveland, B. Kleiner, and P. Tukey. *Graphical Methods for Data Analysis*. Belmont, CA: Wadsworth International Group, 1983.

[6] Cleveland, W.S. "Robust Locally Weighted Regression and Smoothing Scatterplots." *Journal of the American Statistical Association*. Vol. 74, 1979, pp. 829–836.

[7] Cleveland, W.S., and S.J. Devlin. "Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting." *Journal of the American Statistical Association*. Vol. 83, 1988, pp. 596–610.

[8] Daniel, C., and F.S. Wood. *Fitting Equations to Data*. New York: John Wiley & Sons, 1980.

[9] DeAngelis, D.J., J.R. Calarco, J.E. Wise, H.J. Emrich, R. Neuhausen, and H. Weyand. “Multipole Strength in ^{12}C from the (e,e’α) Reaction for Momentum Transfers up to 0.61 fm^{-1}.” *Physical Review C*. Vol. 52, No. 1, 1995, pp. 61–75.

[10] de Boor, C. *A Practical Guide to Splines*. Berlin: Springer-Verlag, 1978.

[11] Draper, N.R., and H. Smith. *Applied Regression Analysis*, 3rd Ed. New York: John Wiley & Sons, 1998.

[12] DuMouchel, W., and F. O’Brien. “Integrating a Robust Option into a Multiple Regression Computing Environment.” *Computing Science and Statistics: Proceedings of the 21st Symposium on the Interface*. (K. Berk and L. Malone, eds.) Alexandria, VA: American Statistical Association, 1989, pp. 297–301.

[13] Goodall, C. “A Survey of Smoothing Techniques.” *Modern Methods of Data Analysis*. (J. Fox and J.S. Long, eds.) Newbury Park, CA: Sage Publications, 1990, pp. 126–176.

[14] Holland, P.W., and R.E. Welsch. “Robust Regression Using Iteratively Reweighted Least-Squares.” *Communications in Statistics—Theory and Methods*. Vol. A6, 1977, pp. 813–827.

[15] Huber, P.J. *Robust Statistics*. New York: John Wiley & Sons, 1981.

[16] Hutcheson, M.C. “Trimmed Resistant Weighted Scatterplot Smooth.” Master’s Thesis, Cornell University, Ithaca, NY, 1995.

[17] Levenberg, K. “A Method for the Solution of Certain Problems in Least Squares.” *Quarterly of Applied Mathematics*. Vol. 2, 1944, pp. 164–168.

[18] Marquardt, D. “An Algorithm for Least-Squares Estimation of Nonlinear Parameters.” *SIAM Journal on Applied Mathematics*, Vol. 11. 1963, pp. 431–441.

[19] Orfanidis, S.J. *Introduction to Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1996.

[20] Press, W.H., S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C, The Art of Scientific Computing*. Cambridge, England: Cambridge University Press, 1993.

[21] Street, J.O., R.J. Carroll, and D. Ruppert. "A Note on Computing Robust Regression Estimates Via Iteratively Reweighted Least Squares." *The American Statistician*. Vol. 42, 1988, pp. 152–154.

[22] Watson, David E. *Contouring: A Guide to the Analysis and Display of Spatial Data.* Tarrytown, NY: Pergamon, 1992.

Index