MFE204TC
ARTIFICIAL INTELLIGENCE AND DATA ANALYSIS

LECTURE 4

LONG HUANG

Xi'an Jiaotong-Liverpool University
西交利物浦大學

CONTENTS

- Regression Analysis
  - Interactive Fitting
  - Programmatic Fitting
- Time Series Analysis

Certain contents of this presentation are adopted from training material at MathWorks

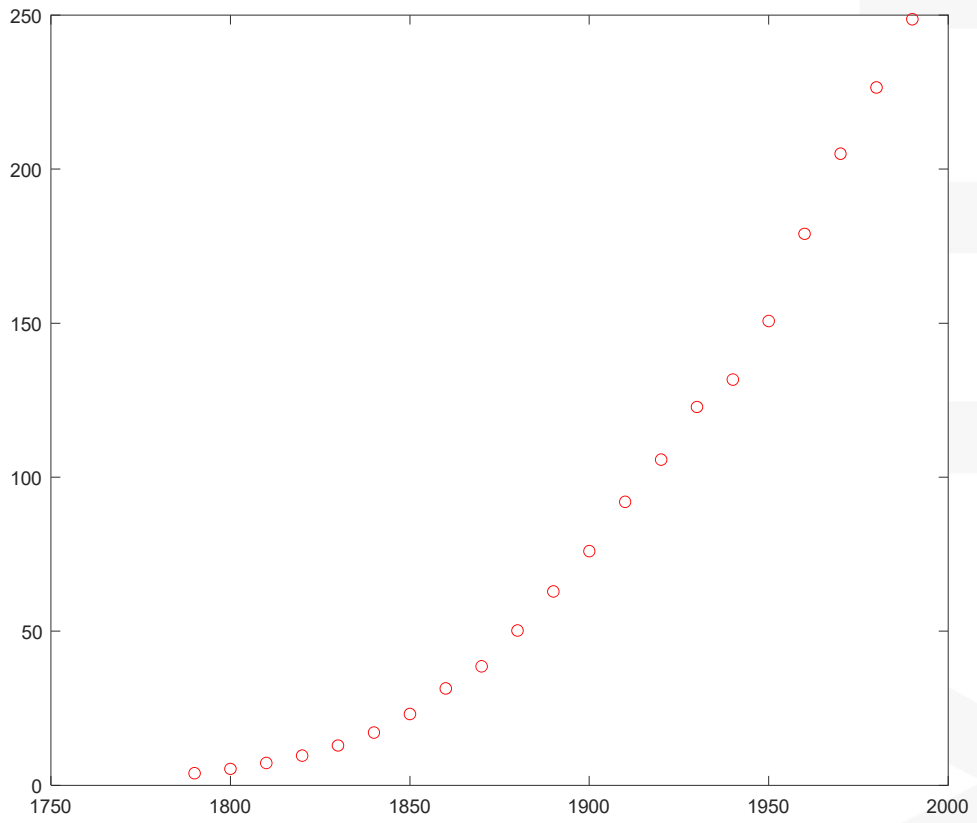
MATLAB INTERACTIVE FITTING

- Model data using a spline interpolant, a shape-preserving interpolant, or a polynomial up to the tenth degree
- Plot one or more fits together with data
- Plot the residuals of the fits
- Compute model coefficients
- Compute the norm of the residuals (a statistic you can use to analyze how well a model fits your data)
- Use the model to interpolate or extrapolate outside of the data
- Save coefficients and computed values to the MATLAB workspace for use outside of the dialog box
- Generate MATLAB code to recompute fits and reproduce plots with new data

MATLAB INTERACTIVE FITTING - EXAMPLE

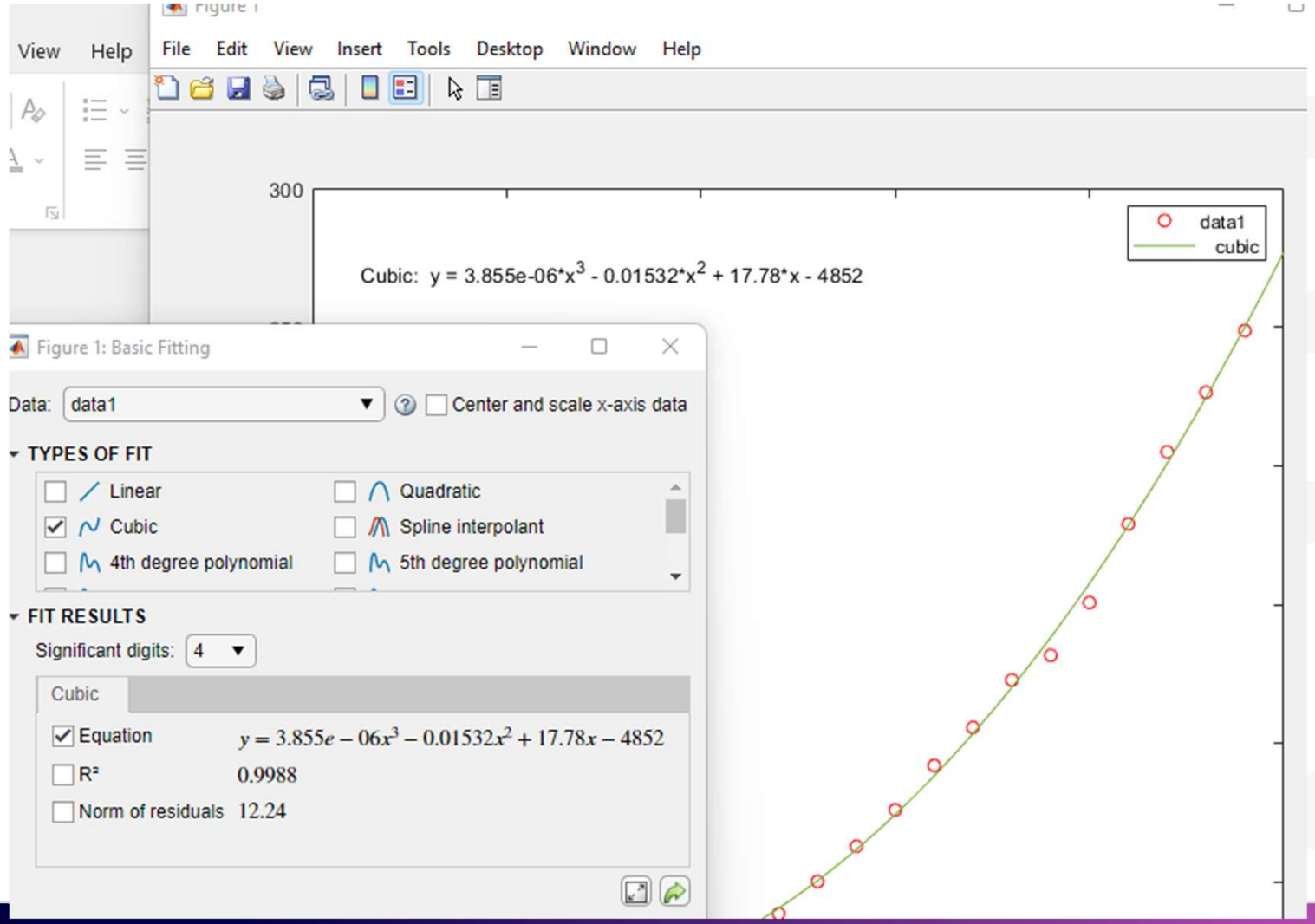
- The file census.mat contains U.S. population data for the years 1790 through 1990 at 10 year intervals.

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



MATLAB INTERACTIVE FITTING - EXAMPLE

- Tools -> Basic Fitting



PROGRAMMATIC FITTING

- Two MATLAB functions can model your data with a polynomial.

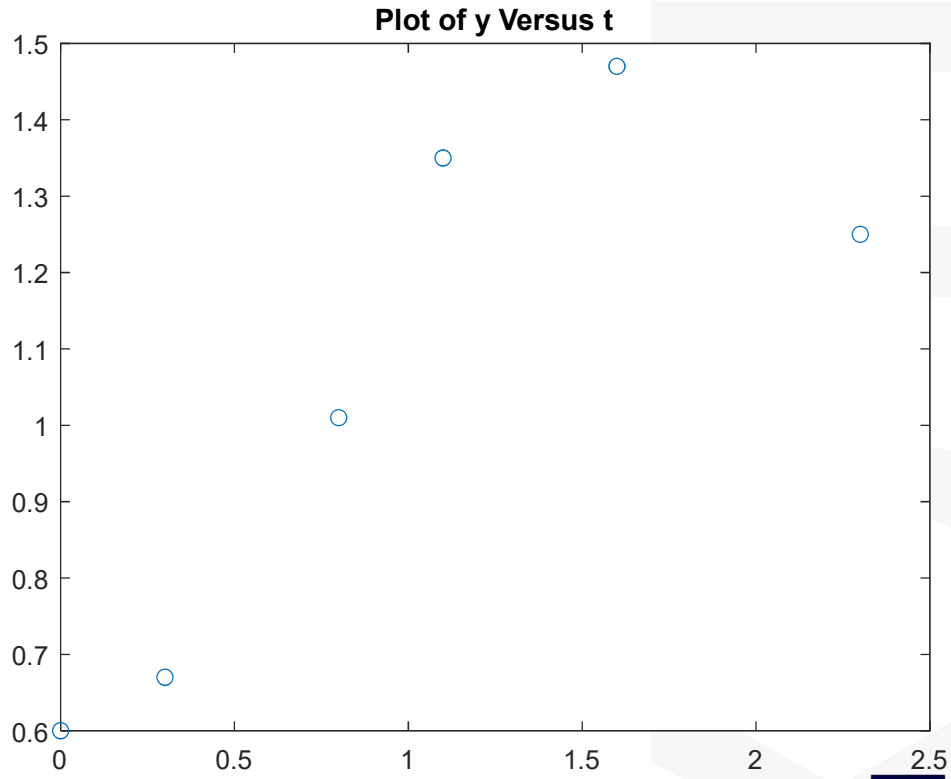
- *polyfit(x,y,n)* finds the coefficients of a polynomial p(x) of degree n that fits the y data by minimizing the sum of the squares of the deviations of the data from the model (least-squares fit).

- *polyval(p,x)* returns the value of a polynomial of degree n that was determined by polyfit, evaluated at x.

POLYNOMIAL MODELS - EXAMPLE

- Measure a quantity y at several values of time t.

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



POLYNOMIAL MODELS - EXAMPLE

- Try modeling this data using a second-degree polynomial function

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

```
t = [0 0.3 0.8 1.1 1.6 2.3];
y = [0.6 0.67 1.01 1.35 1.47 1.25];
plot(t,y,'o')
title('Plot of y Versus t')
p = polyfit(t,y,2)
```

```
p =

   -0.2942    1.0231    0.4981
```

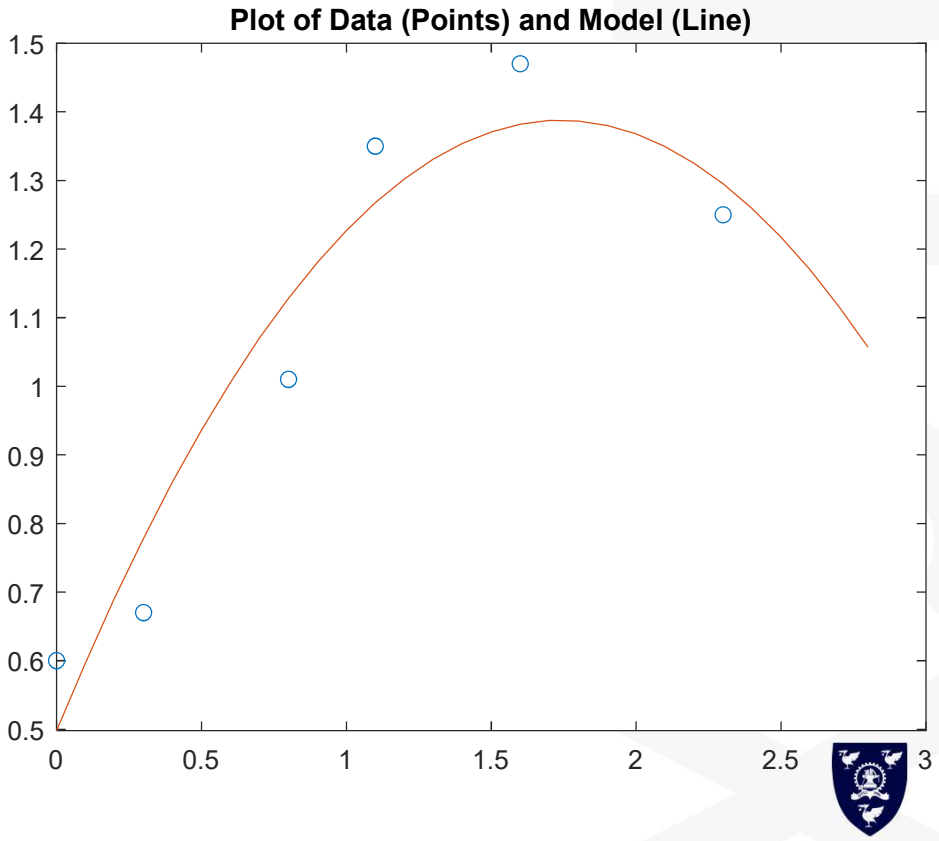
- The second-degree polynomial model of the data is given by the equation

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

POLYNOMIAL MODELS - EXAMPLE

- Evaluate the polynomial at uniformly spaced times, t2. Then, plot the original data and the model on the same plot.

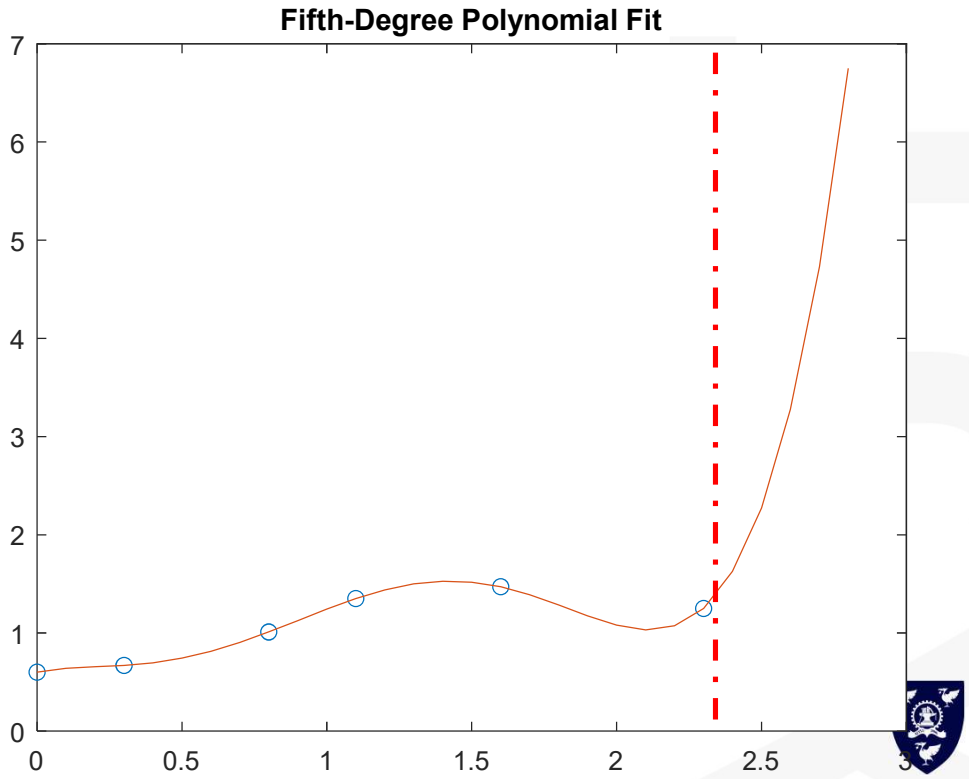
```
t = [0 0.3 0.8 1.1 1.6 2.3];
y = [0.6 0.67 1.01 1.35 1.47 1.25];
plot(t,y,'o')
title('Plot of y Versus t')
p = polyfit(t,y,2)
t2 = 0:0.1:2.8;
y2 = polyval(p,t2);
figure
plot(t,y,'o',t2,y2)
title('Plot of Data (Points) and Model (Line)')
```



POLYNOMIAL MODELS - EXAMPLE

- Second-degree fit roughly follows the basic shape of the data, but does not capture the smooth curve.
- A fifth-degree polynomial does a better job of following the fluctuations in the data.

```
t = [0 0.3 0.8 1.1 1.6 2.3];
y = [0.6 0.67 1.01 1.35 1.47 1.25];
plot(t,y,'o')
title('Plot of y Versus t')
p = polyfit(t,y,2)
t2 = 0:0.1:2.8;
y2 = polyval(p,t2);
figure
plot(t,y,'o',t2,y2)
title('Plot of Data (Points) and Model (Line)')
p5 = polyfit(t,y,5)
y3 = polyval(p5,t2);
figure
plot(t,y,'o',t2,y3)
title('Fifth-Degree Polynomial Fit')
```



LINEAR MODEL WITH NONPOLYNOMIAL TERMS

- Alternative when polynomial function does not produce a satisfactory model of your data
- Consider the following function that is linear in the parameters a0, a1, and a2, but nonlinear in the t data

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

- We can accomplish this by forming a design matrix, where each column represents a variable used to predict the response (a term in the model) and each row corresponds to one observation of those variables.

```
t = [0 0.3 0.8 1.1 1.6 2.3]';
y = [0.6 0.67 1.01 1.35 1.47 1.25]';
X = [ones(size(t)) exp(-t) t.*exp(-t)];
a = X\y
```

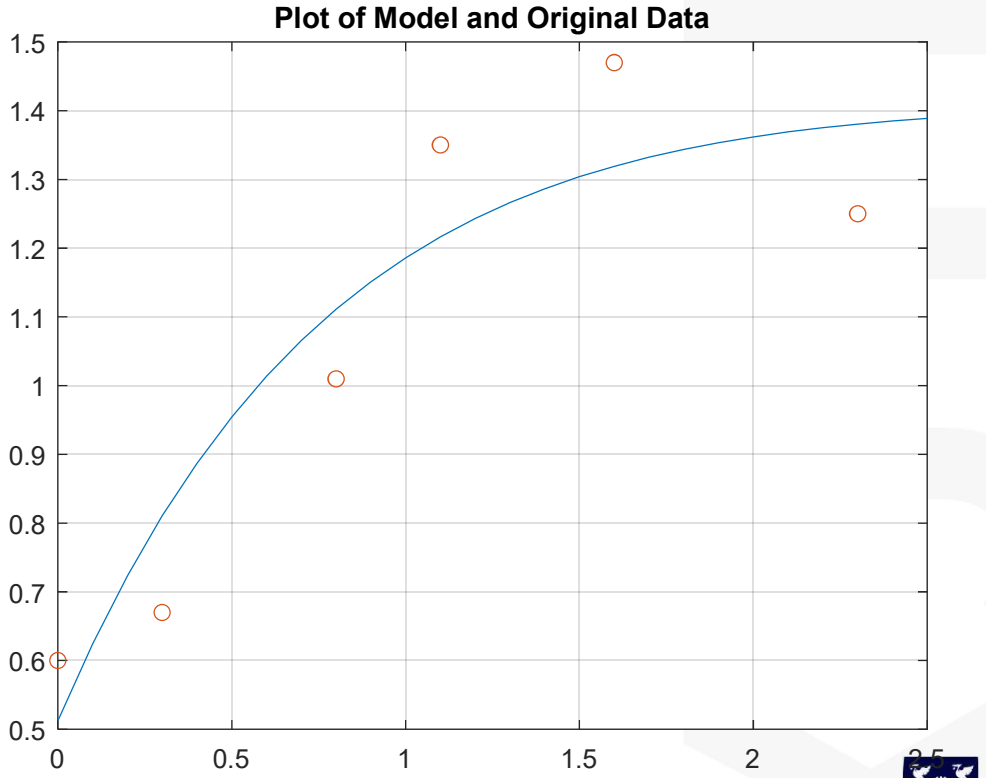
```
a =

    1.3983
   -0.8860
    0.3085
```

LINEAR MODEL WITH NONPOLYNOMIAL TERMS

- Now evaluate the model at evenly spaced points and plot the model with the original data.

```
t = [0 0.3 0.8 1.1 1.6 2.3]';
y = [0.6 0.67 1.01 1.35 1.47 1.25]';
X = [ones(size(t)) exp(-t) t.*exp(-t)];
a = X\y
T = (0:0.1:2.5)';
Y = [ones(size(T)) exp(-T) T.*exp(-T)]*a;
plot(T,Y,'-',t,y,'o'), grid on
title('Plot of Model and Original Data')
```



MULTIPLE REGRESSION

- Multiple regression: when y is a function of more than one predictor variable, the matrix equations that express the relationships among the variables must be expanded to accommodate the additional data.
- E.g.:

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

- Multiple regression solves for unknown coefficients a_0, a_1, and a_2 by minimizing the sum of the squares of the deviations of the data from the model (least-squares fit).
- Construct and solve the set of simultaneous equations by forming a design matrix, X.

MULTIPLE REGRESSION

- Measure a quantity y for several values of x_1 and x_2. Store these values in vectors x_1, x_2, and y,
- Construct and solve the set of simultaneous equations by forming a design matrix, X.

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

```
X = [ones(size(x1)) x1 x2];
a = X\y
Y = X*a;
MaxErr = max(abs(Y - y))
```

```
a =

    0.1018
    0.4844
   -0.2847

MaxErr =

    0.0038
```

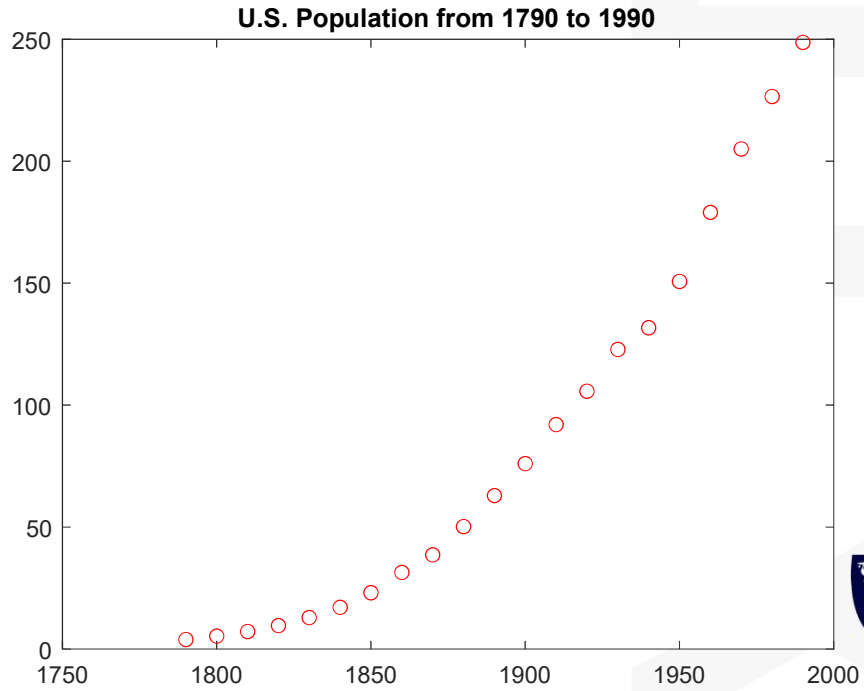
$$y = 0.1018 + 0.4844x_1 - 0.2847x_2.$$

- Q: how to plot this?

PROGRAMMATIC FITTING –EXAMPLE

- A typical data fitting process includes the following:
  - Calculate Correlation Coefficients
  - Fit a Polynomial to the Data
  - Plot and Calculate Confidence Bounds
- Example: U.S. population data from 1790 to 1990

```
load census
plot(cdate,pop,'ro')
title('U.S. Population from 1790 to 1990')
```



PROGRAMMATIC FITTING –EXAMPLE

- Calculate Correlation Coefficients

```
load census
plot(cdate,pop,'ro')
title('U.S. Population from 1790 to 1990')
corrcoef(cdate,pop)
```

```
ans =

    1.0000    0.9597
    0.9597    1.0000
```

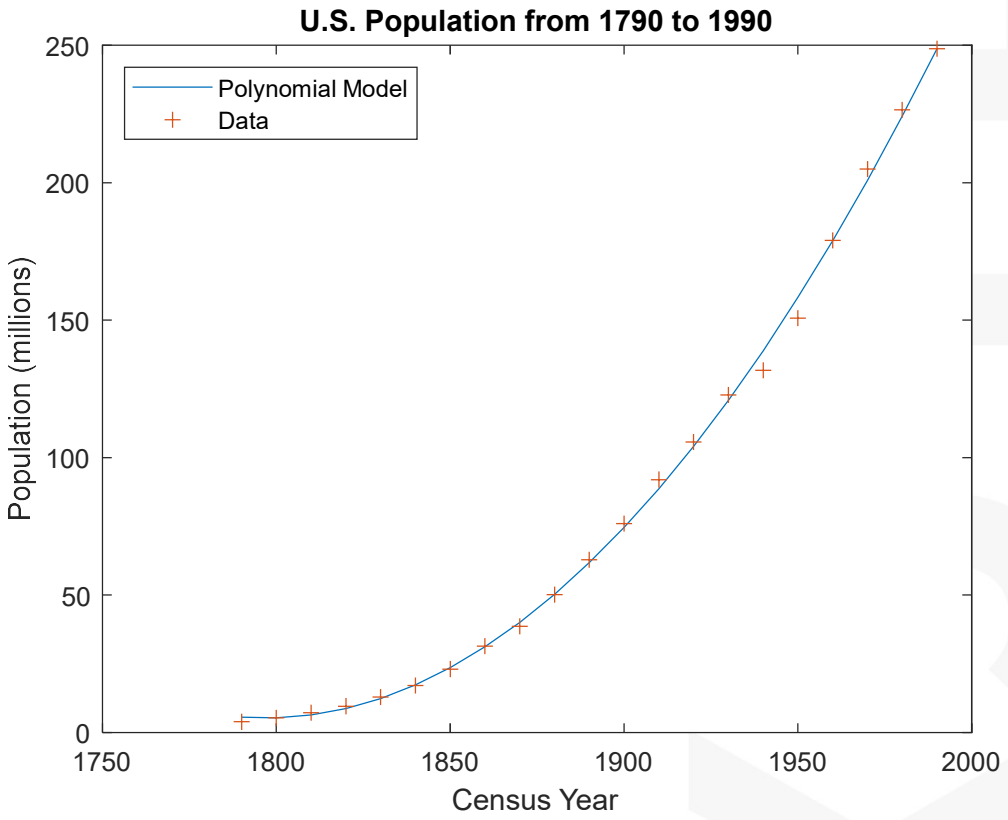
- The diagonal matrix elements represent the perfect correlation of each variable with itself and are equal to 1.

- The off-diagonal elements are very close to 1, indicating that there is a strong statistical correlation between the variables cdate and pop

PROGRAMMATIC FITTING –EXAMPLE

- Fit a Polynomial to the Data

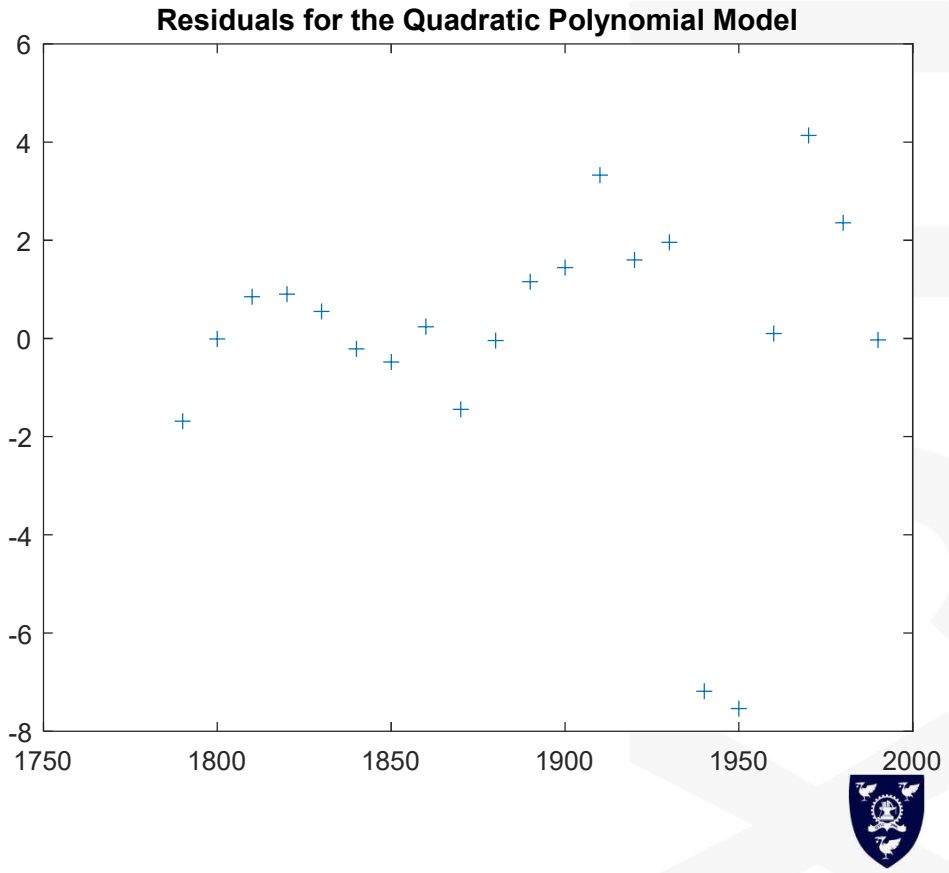
```
load census
plot(cdate,pop,'ro')
title('U.S. Population from 1790 to 1990')
[p,ErrorEst] = polyfit(cdate,pop,2);
pop_fit = polyval(p,cdate,ErrorEst);
plot(cdate,pop_fit,'-',cdate,pop,'+');
title('U.S. Population from 1790 to 1990')
legend('Polynomial
Model','Data','Location','NorthWest');
xlabel('Census Year');
ylabel('Population (millions)');
```



PROGRAMMATIC FITTING –EXAMPLE

- Calculate the residuals for this fit.

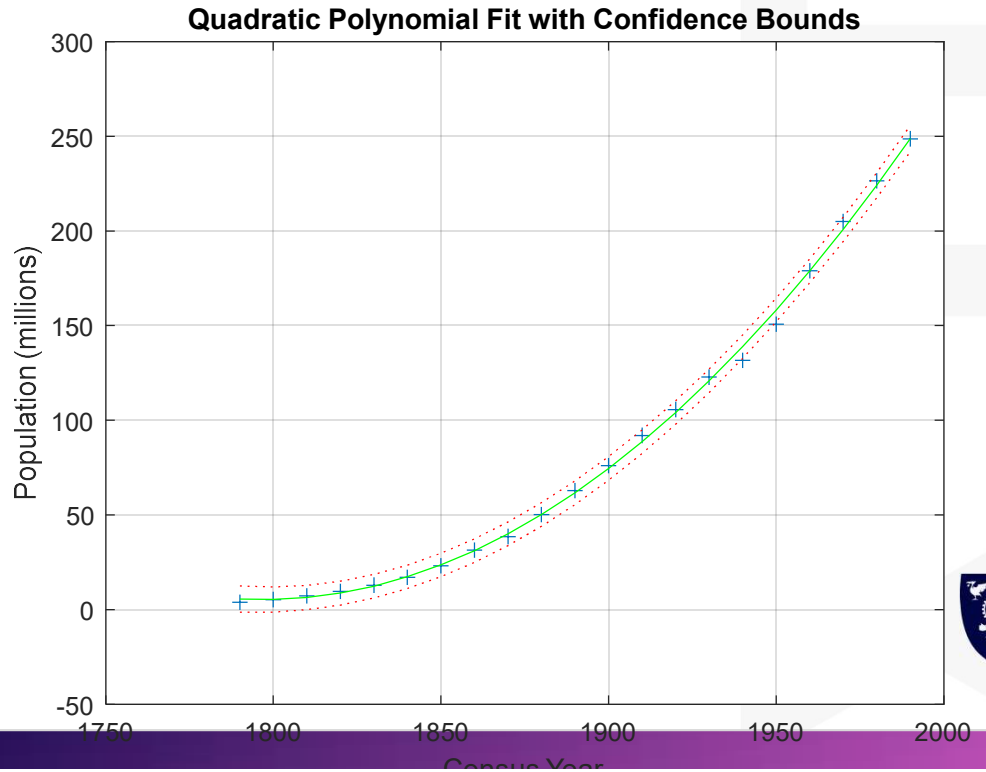
```
load census
plot(cdate,pop,'ro')
title('U.S. Population from 1790 to 1990')
[p,ErrorEst] = polyfit(cdate,pop,2);
pop_fit = polyval(p,cdate,ErrorEst);
plot(cdate,pop_fit,'-',cdate,pop,'+');
title('U.S. Population from 1790 to 1990')
legend('Polynomial
Model','Data','Location','NorthWest');
xlabel('Census Year');
ylabel('Population (millions)');
res = pop - pop_fit;
figure, plot(cdate,res,'+')
title('Residuals for the Quadratic Polynomial
Model')
```



PROGRAMMATIC FITTING –EXAMPLE

- Confidence bounds are confidence intervals for a predicted response. The width of the interval indicates the degree of certainty of the fit.
- The 95% interval indicates that you have a 95% chance that a new observation will fall within the bounds.

```
load census
plot(cdate,pop,'ro')
title('U.S. Population from 1790 to 1990')
[p,ErrorEst] = polyfit(cdate,pop,2);
pop_fit = polyval(p,cdate,ErrorEst);
plot(cdate,pop_fit,'-',cdate,pop,'+');
title('U.S. Population from 1790 to 1990')
legend('Polynomial Model','Data','Location','NorthWest');
xlabel('Census Year');
ylabel('Population (millions)');
res = pop - pop_fit;
figure, plot(cdate,res,'+')
title('Residuals for the Quadratic Polynomial Model')
[pop_fit,delta] = polyval(p,cdate,ErrorEst);
plot(cdate,pop,'+',...
cdate,pop_fit,'g-',...
cdate,pop_fit+2*delta,'r:',...
cdate,pop_fit-2*delta,'r:');
xlabel('Census Year');
ylabel('Population (millions)');
title('Quadratic Polynomial Fit with Confidence Bounds')
grid on
```



TIME SERIES ANALYSIS

- Time series are data vectors sampled over time, in order, often at regular intervals.
- Time series represent the time-evolution of a dynamic population or process.
- Time series analysis is concerned with:
  - Identifying patterns
  - Modeling patterns
  - Forecasting values

TYPES OF TIME SERIES AND THEIR USES

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

TIME SERIES DATA SAMPLE

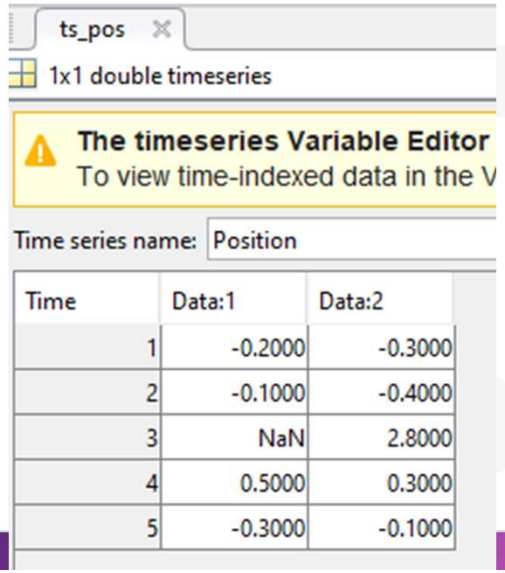
- The difference between a data value and a data sample
  - A data value is a single, scalar value recorded at a specific time.
  - A data sample consists of one or more values associated with a specific time in the timeseries object.
  - The number of data samples in a time series is the same as the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

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

TIME SERIES DATA SAMPLE

- Consider the following data matrix where NaN value represents a missing data value.
- The first two columns of x contain quantities with the same units and you can create a multivariate timeseries object to store these two time series.
- The Length of the time vector, which is 5 in this example, equals the number of data samples in the timeseries object.

```
x = [-0.2 -0.3 13;
-0.1 -0.4 15;
NaN 2.8 17;
0.5 0.3 NaN;
-0.3 -0.1 15]
ts_pos = timeseries(x(:,1:2), 1:5, 'name',
'Position')
ts_vel = timeseries(x(:,3), 1:5, 'name', 'Velocity');
```

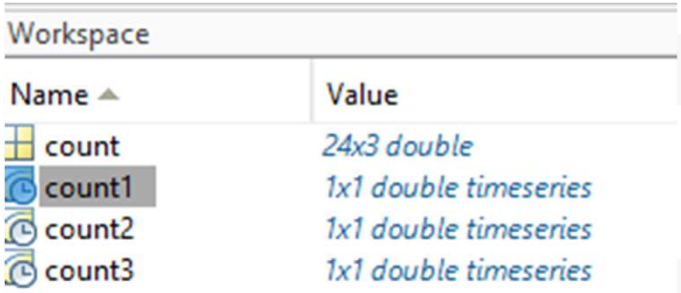


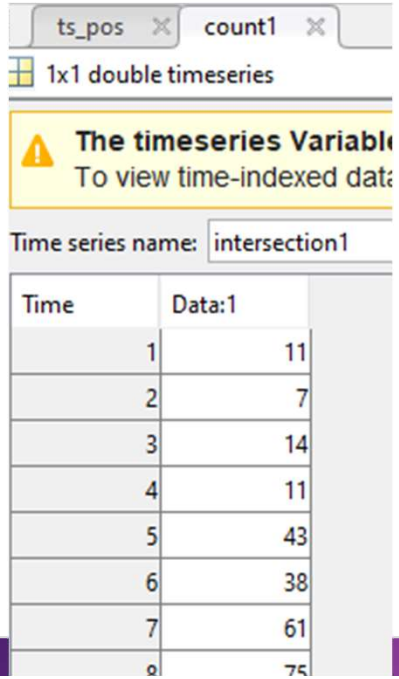
Time	Data:1	Data:2
1	-0.2000	-0.3000
2	-0.1000	-0.4000
3	NaN	2.8000
4	0.5000	0.3000
5	-0.3000	-0.1000

CREATING TIME SERIES OBJECTS

- For the sample data from count.dat, create three timeseries objects to store the data collected at each intersection.

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

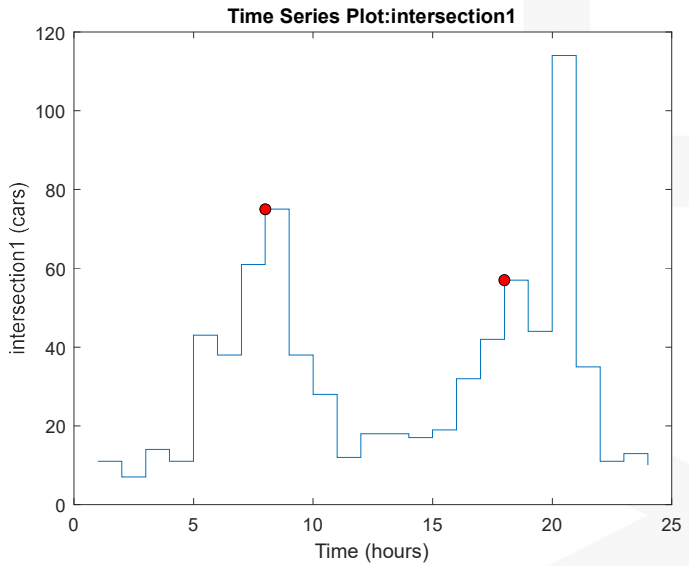




MODIFYING TIME SERIES UNITS

- Change the data units for count1 to 'cars’.
- Converting count1 the discrete-time values to a continuous-time by setting the interpolation method for count1 to zero-order hold. This holds each sample value for one sample interval.
- Modify the time units to be 'hours' for the three time series.

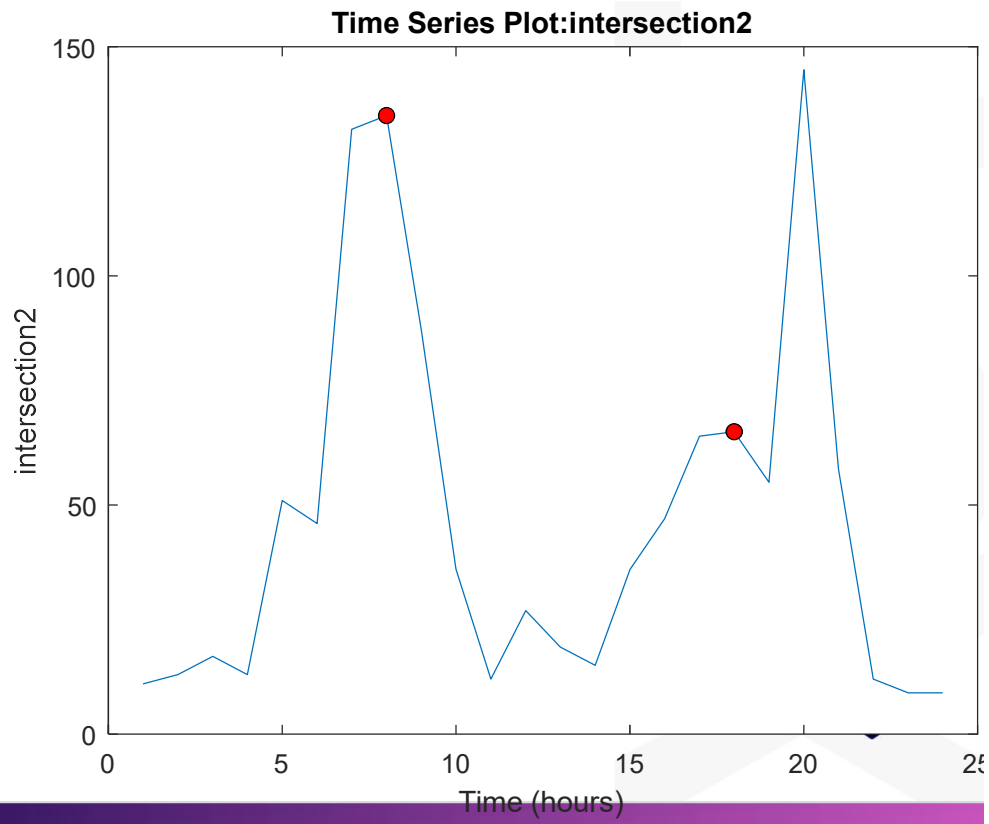
```
load count.dat
count1 = timeseries(count(:,1), 1:24,'name', 'intersection1');
count2 = timeseries(count(:,2), 1:24,'name', 'intersection2');
count3 = timeseries(count(:,3), 1:24,'name', 'intersection3');
count1.DataInfo.Units = 'cars';
count1.DataInfo.Interpolation = tsdata.interpolation('zoh');
count1.TimeInfo.Units = 'hours';
count2.TimeInfo.Units = 'hours';
count3.TimeInfo.Units = 'hours';
```



DEFINING EVENTS

- Events mark the data at specific times.
- Add two events to the data that mark the times of the 8 AM commute and 6 PM commute.

```
load count.dat
count1 = timeseries(count(:,1), 1:24,'name', 'intersection1');
count2 = timeseries(count(:,2), 1:24,'name', 'intersection2');
count3 = timeseries(count(:,3), 1:24,'name', 'intersection3');
count1.DataInfo.Units = 'cars';
count1.DataInfo.Interpolation = tsdata.interpolation('zoh');
count1.TimeInfo.Units = 'hours';
count2.TimeInfo.Units = 'hours';
count3.TimeInfo.Units = 'hours';
e1 = tsdata.event('AMCommute',8);
e1.Units = 'hours'; % Specify the units for time
count1 = addevent(count1,e1); % Add the event to count1
count2 = addevent(count2,e1); % Add the event to count2
count3 = addevent(count3,e1); % Add the event to count3
e2 = tsdata.event('PMCommute',18);
e2.Units = 'hours'; % Specify the units for time
count1 = addevent(count1,e2); % Add the event to count1
count2 = addevent(count2,e2); % Add the event to count2
count3 = addevent(count3,e2); % Add the event to count3
figure
plot(count2)
```



CREATING TIME SERIES COLLECTION OBJECTS

- Each individual time series in a collection is called a member.

```
tsc = tscollection({count1 count2},'name', 'count_coll')
tsc = addts(tsc, count3)
```

Time vector characteristics

Start time	1 hours
End time	24 hours

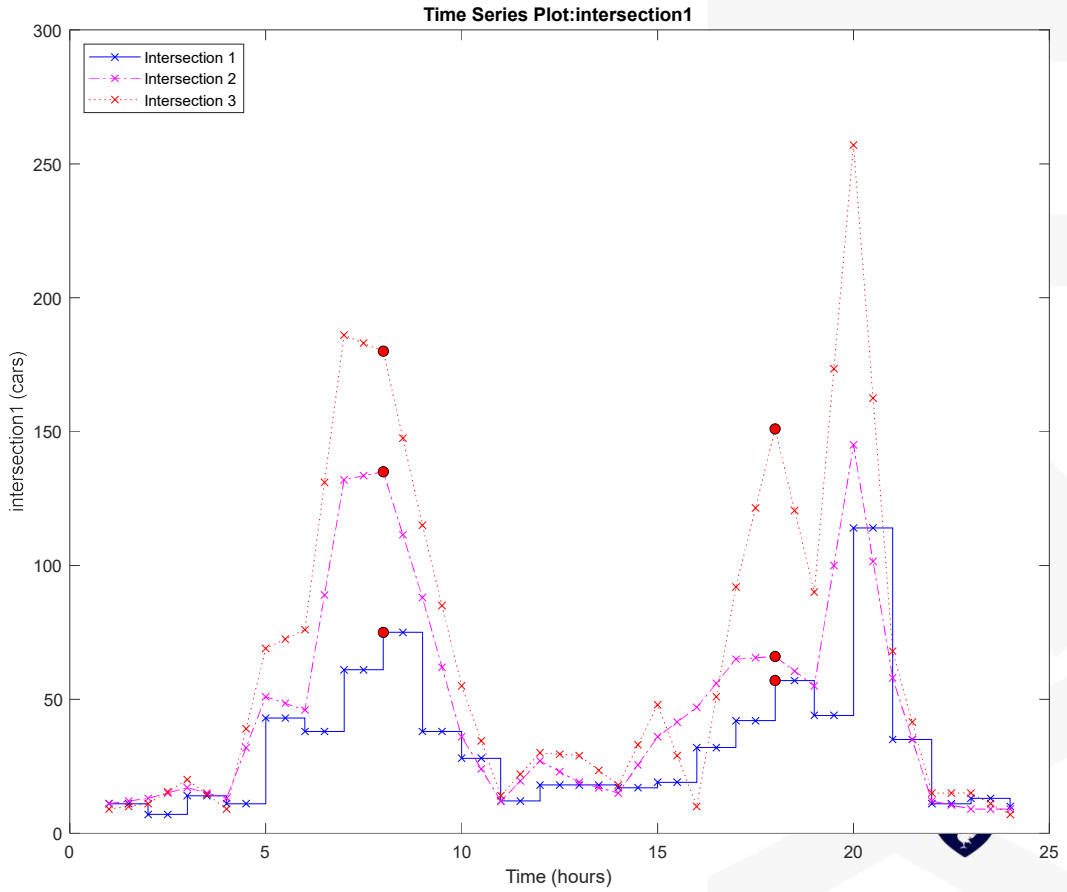
Member Time Series Objects:

intersection1
intersection2
intersection3

RESAMPLING A TIME SERIES COLLECTION OBJECT

- Interpolate values at each half-hour mark.
- The default interpolation method of a time series is used.

```
tsc1 = resample(tsc,1:0.5:24)
hold off % Allow axes to clear before
plotting
plot(tsc1.intersection1,'-
xb','Displayname','Intersection 1')
hold on
plot(tsc1.intersection2,'-
.xm','Displayname','Intersection 2')
plot(tsc1.intersection3,':xr','Displayna
me','Intersection 3')
legend('show','Location','NorthWest')
```



ADDING/REMOVING A DATA SAMPLE

- Adding a data sample to ‘intersection1’
  - Q: what are the values at 3.25hours for ‘intersection2’ and ‘intersection3’?

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

- Removing missing data
  - Q: does the data in ‘intersection1’ stay?

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

- Interpolating missing data

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