Data processing in MATLAB

Gianluca Bianchin
gbian001@ucr.edu

Graduate Quantitative Methods Center
University of California, Riverside

LFSC1425 - February 22, 2018

About this workshop

This workshop follows “Introduction to MATLAB“, where we discussed:

- Using the Command Window
- Scripts & storing data
- Processing arrays
- Saving and loading

Data processing in MATLAB: Outline

Advanced array operations

Array addressing and indexing

Load `examgrades.mat` (sample data set built-in in MATLAB) representing the exam grades on a scale $0 - 100$ for 120 students in 5 exam attempts

How do we visualize:

- Grade of student 3 on attempt 4
- Grades of all students on attempt 2
- Grades of student 119 on all attempts
- Grades of students $1, 3, 8$ on all attempts
- Grades of students $1, 3, 8$ on attempts 1 and 3
- Grades the first 10 students listed
- Grades of the last 10 students listed
- Grades of even students on attempts 4 and 5

Array addressing and indexing

Load `examgrades.mat` (sample data set built-in in MATLAB) representing the exam grades on a scale $0 - 100$ for 120 students in 5 exam attempts

How do we visualize:

- Grade of student 3 on attempt 4
- Grades of all students on attempt 2
- Grades of student 119 on all attempts
- Grades of students $1, 3, 8$ on all attempts
- Grades of students $1, 3, 8$ on attempts 1 and 3
- Grades the first 10 students listed
- Grades of the last 10 students listed
- Grades of even students on attempts 4 and 5

Array addressing and indexing (solution)

Load `examgrades.mat` (sample data set built-in in MATLAB) representing the exam grades on a scale $0 - 100$ for 120 students in 5 exam attempts

How do we extrapolate:

- Grade of student 3 on attempt 4 — `grades(3,4)`
- Grades of all students on attempt 2 — `grades(:,2)`
- Grades of student 119 on all attempts — `grades(119,:)`
- Grades of students $1, 3, 8$ on all attempts — `grades([1, 3, 8],:)`
- Grades of students $1, 3, 8$ on attempts 1, 3 — `grades([1, 3, 8],[1,3])`
- Grades the first 10 students listed — `grades(1:10,:)`
- Grades of the last 10 students listed — `grades(end-9:end,:)`
- Grades of even students on attempts 4, 5 — `grades(2:2:end,[4, 5])`

Array addressing and indexing (summary)

The colon symbol `(:)` indicates a sequence of indexes

The notation [1:2:10] denotes an array of indexes that starts at 1, with increments of 2 and ends at 10

Examples

- All indexes `:`
- Indexes from a to b `a:b`
- Indexes from a to b spaced by c `a:c:b`
- Indexes from a to the end of the array spaced by c `a:c:end`

Array construction

How do we create arrays

- With integer entries from 1 to 10
- With entries from -1 to 1 spaced by 0.1
- With 11 entries from 0 to π
- With integer entries from -8 to 5, excluding -3 and -2

The notation [-1:.1:1] creates an array that starts at -1, with increments of 0.1 and ends at 1. The MATLAB function `linspace` arguments are described by

`linspace(first_value,last_value,number_of_values)`

Array construction

How do we create arrays

- With integer entries from 1 to 10
- With entries from -1 to 1 spaced by 0.1
- With 11 entries from 0 to π
- With integer entries from -8 to 5, excluding -3 and -2

The notation [-1:.1:1] creates an array that starts at -1, with increments of 0.1 and ends at 1. The MATLAB function `linspace` arguments are described by

`linspace(first_value,last_value,number_of_values)`

Array construction (solution)

How do we create arrays

- With integer entries from 1 to 10 — `a=[1:10]`
- With entries from -1 to 1 spaced by 0.1 — `a=[-1:.1:1]`
- With 11 entries from 0 to π — `a=linspace(0,pi,11)`
- With integer entries from -8 to 5, excluding -3 and -2 — `b=[-8:-4],c=[-1:5], a=[b c]`

The notation [-1:.1:1] creates an array that starts at -1, with increments of 0.1 and ends at 1. The MATLAB function `linspace` arguments are described by

```
linspace(first_value,last_value,number_of_values)
```

Array construction techniques

Array Construction Technique	Description
`x=[2 2*pi sqrt(2) 2-3j]`	Creates row vector x containing arbitrary elements
`x=first:last`	Creates row vector x starting with first, counting by 1, and ending at or before last (Note that x=[first:last] produces the same result, but takes longer, since MATLAB considers both bracket and colon array-creation forms.)
`x=first:increment:last`	Creates row vector x starting with first, counting by increment, and ending at or before last
`x=linspace(first,last,n)`	Creates linearly spaced row vector x starting with first, ending at last, having n elements
`x=logspace(first,last,n)`	Creates logarithmically spaced row vector x starting with 10^{first}, ending at 10^{last}, and having n elements

Other useful array construction functions

- `ones()`
- `zeros()`
- `eye()`
- `rand()`
- `randn()`
- `diag()`

Column array construction

Array orientation

In the preceding examples arrays contained one row and multiple columns (called row vectors). Column vectors can be created by (i) using semicolons, (ii) using the transpose operator (').

Array assignment and mathematics

How do we

- Penalize all grades by 2 points
- Increase all grades by 10%
- Update student 1 grading on attempt 5
- Penalize student 2 by 20%

- Sum two row vectors `a` and `b`
- Multiply each entry of vector `a` by the corresponding entry of vector `b`

Array assignment and mathematics (solutions)

How do we

- Penalize all grades by 2 points `grades=grades-2`
- Increase all grades by 10% `grades=grades*1.1`
- Update student 1 grading on attempt 5 `grades(1,5)=71`
- Penalize student 2 by 20% `grades(2,:)= grades(2,:)*0.8`
- Sum two row vectors `a` and `b` `c=a+b`
- Multiply each entry of vector `a` by the corresponding entry of vector `b` `c=a.*b`

Useful functions on arrays: sorting

`y = sort(x)` sorts in ascending order
`[y, ind] = sort(x)` returns sort index as well

We would like to:

- Sort and visualize grades based on attempt 2

```
load examgrades.mat

[xs, idx] = sort(grades(:,2));   % sort based on attempt 2

grades(idx,:)                    % display sorted grades
```

Useful functions on arrays: sorting

`y = sort(x)` sorts in ascending order
`[y, ind] = sort(x)` returns sort index as well

We would like to:

- Sort and visualize grades based on attempt 2

```
load examgrades.mat

[xs, idx] = sort(grades(:,2));   % sort based on attempt 2

grades(idx,:)                     % display sorted grades
```

Useful functions on arrays: searching

`[rows,cols] = find(X==a)` finds the row and column indices of values of array X that are equal to a.

We would like to:

- Find and visualize students and attempts whose grade is lower than 55

```
load examgrades.mat

[rows,cols] = find(grades<55);

rows                    % display students with
                        %(at least one) grade lower than

grades(rows,:)          % display student attempts
                        %(at least one) grade lower than
```

Useful functions on arrays: searching

`[rows,cols] = find(X==a)` finds the row and column indices of values of array X that are equal to a.

We would like to:

- Find and visualize students and attempts whose grade is lower than 55

```
load examgrades.mat

[rows,cols] = find(grades<55);

rows                    % display students with
                        %(at least one) grade lower than
grades(rows,:)          % display student attempts
                        %(at least one) grade lower than
```

Other useful functions on arrays

- Some useful functions for arrays:

  `max(), min()`

  `mean(), median(), cov(), var()`

  `sum(), diff(), cumsum()`
- Size: `length(v), size(M)`

How do we

- Find the max grade for student 3
- Find the min grade for attempt 5
- Compute the mean of attempt 5

Other useful functions on arrays (solutions)

- Some useful functions for arrays:
  `max(), min()`
  `mean(), median(), cov(), var()`
  `sum(), diff(), cumsum()`
- Size: `length(v), size(M)`

How do we

- Find the max grade for student 3 — `max(grades(3,:))`
- Find the min grade for attempt 5 — `max(grades(:,5))`
- Compute the mean of attempt 5 — `mean(grades(:,5))`

Customizing your plots

2D plotting

Two-dimensional line and points are created with the command `plot(xdata,ydata)`

We are interested in

- Plot attempt 4 and compare with attempt 5
- Plot student 1 and compare with student 2

```
load examgrades.mat

plot(1:120,grades(:,4))      % plot attempt 4
hold on
plot(1:120,grades(:,5))      % compare with attempt 5
title('Attempt 4 vs attempt 5')

figure
plot(1:5,grades(1,:))        % Plot student 1
hold on
plot(1:5,grades(2,:))        % Compare with student 2
title('Student 1 vs student 2')
```

2D plotting

Two-dimensional line and points are created with the command `plot(xdata,ydata)`

We are interested in

- Plot attempt 4 and compare with attempt 5
- Plot student 1 and compare with student 2

```
load examgrades.mat

plot(1:120,grades(:,4))      % plot attempt 4
hold on
plot(1:120,grades(:,5))      % compare with attempt 5
title('Attempt 4 vs attempt 5')

figure
plot(1:5,grades(1,:))        % Plot student 1
hold on
plot(1:5,grades(2,:))        % Compare with student 2
title('Student 1 vs student 2')
```

Note on plot function

When the `plot` function is called with only one argument (e.g `plot(grades)`), the command is interpreted as `plot(1:length(grades),grades)`; that is, `grades` is plotted versus an index of its values. When `grades` is a matrix, this interpretation is applied to each column of `grades`.

Linestyle, markers, and colors

We can specify our own colors, markers, and linestyles by giving `plot` a third argument

Symbol	Color	Symbol	Marker	Symbol	Linestyle
b	Blue	.	Point	-	Solid line
g	Green	o	Circle	:	Dotted line
r	Red	x	Cross	-.	Dash-dot line
c	Cyan	+	Plus sign	--	Dashed line
m	Magenta	*	Asterisk		
y	Yellow	s	Square		
k	Black	d	Diamond		
w	White	v	Triangle (down)		
		^	Triangle (up)		
		<	Triangle (left)		
		>	Triangle (right)		
		p	Pentagram		
		h	Hexagram		

```
plot(x,y,'b:p'), plot(x,y,'c-'), plot(x,y,'m+')
```

Customizing your plot (GUI)

Most of figure properties can be edited from a user-friendly window accessible at
Edit $\rightarrow$ Figure Properties

From there it is possible to

- (Line properties) select plot type, Line, Marker, ...
- (Axis properties) edit axis limits, fonts, title color, ..
- (Figure properties) choose background, colormap, ..

Customizing your plot (command)

However, if we are running several simulations, we may desire to systematically repeat these operations for each figure.

- We have seen how properties of lines can be customized through optional parameters in the command `plot`

? Axis properties

? Figure properties

Customizing axes

Common commands for axis customization:

- `axis([xmin xmax ymin ymax])` Sets axis limits on current plot
- `axis square` Makes the axis box square
- `axis off` Turns off all axis labeling, tick marks, and background

We can also add labels to our axis:

- `xlabel('text')` Sets label for x axis
- `ylabel('text')` Sets label for y axis
- `xlabel('$t_1$','interpreter','latex','FontSize',12)`

Customizing axes

Common commands for axis customization:

- `axis([xmin xmax ymin ymax])` Sets axis limits on current plot
- `axis square` Makes the axis box square
- `axis off` Turns off all axis labeling, tick marks, and background

We can also add labels to our axis:

- `xlabel('text')` Sets label for x axis
- `ylabel('text')` Sets label for y axis
- `xlabel('$t_1$','interpreter','latex','FontSize',12)`

Multiple figures

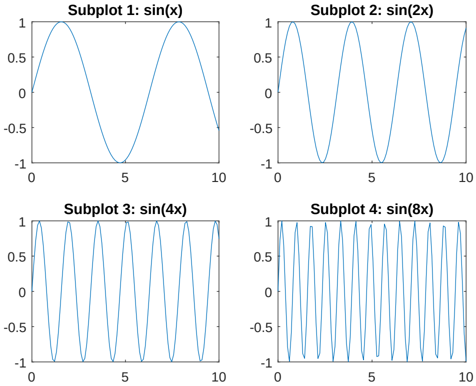
It is possible to create multiple *Figure* windows and plot different data in each one.

- Use the `Figure` command to create a new window
- Every time a new figure window is created, a number identifying it (handle) is returned
- To reuse a figure window for a new plot, it has to be made active by: (i) clicking on the figure with your mouse, or (ii) the command `figure(h)`, where `h` is the handle

Useful commands for figure windows

- `hold on`, `hold off`, `close`, `close all`

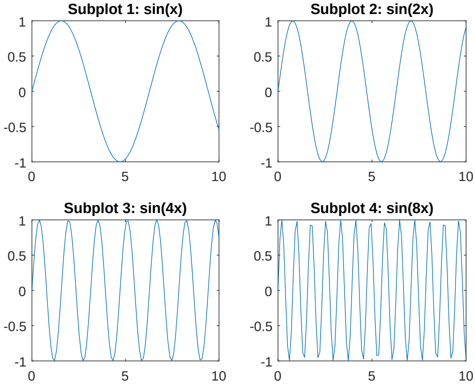
Subplots



One Figure window can hold more than one set of axes. The command `subplot(m,n,p)` subdivides the current figure into a m by n matrix of plotting areas, and choses the p-th area to be active.

- Subplot numbering is left to right along the top row, then along the second row, and so on

Subplots



One Figure window can hold more than one set of axes. The command `subplot(m,n,p)` subdivides the current figure into a m by n matrix of plotting areas, and choses the p-th area to be active.

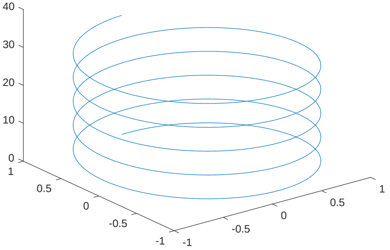
- Subplot numbering is left to right along the top row, then along the second row, and so on

Subplots example

```
x = linspace(0,10);         % x data

subplot(2,2,1)              % Plot data on area 1
y1 = sin(x);
plot(x,y1)
title('Subplot 1: sin(x)')

subplot(2,2,2)              % Plot data on area 2
y2 = sin(2*x);
plot(x,y2)
title('Subplot 2: sin(2x)')

subplot(2,2,3)              % Plot data on area 3
y3 = sin(4*x);
plot(x,y3)
title('Subplot 3: sin(4x)')

subplot(2,2,4)              % Plot data on area 4
y4 = sin(8*x);
plot(x,y4)
title('Subplot 4: sin(8x)')
```

3D Graphics: Line plots

Curves (3D-valued functions of 1 variable) can be plotted through `plot3(x1,y1,z1,...)`

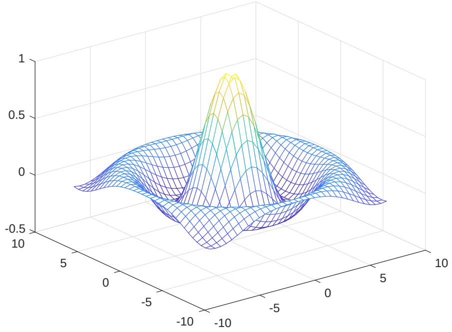



```
t = 0:pi/50:10*pi;   % z data
st = sin(t);         % x data
ct = cos(t);         % y data

plot3(st,ct,t)       % Plot the curve
```

3D Graphics: Scalar functions of 2 variables

Scalar functions of two variables help visualize how a certain quantity (z-axis) varies as a function of two other quantities (x-axis and y-axis)

$$z = f(x, y)$$



Note how the line colors are related to the height of the mesh

3D Graphics: mesh

- First of all we need to define a *grid* for the x and y axis
  `[X,Y] = meshgrid(-8:.1:8,-8:.1:8)`
- Then, we need to compute the value of z, $z = f(x, y)$
  `Z=(X+Y).^2` (vector entry-wise operations)
- Finally we can plot the mesh surface
  `mesh(X,Y,Z)`

The result looks like a fishing net with knots at the data points.

3D Graphics: mesh example

We would like to plot the function $z = f(x, y) = (x + y)^2$

```
x = -8:.5:8;                % Define grid points (x)
y = -8:.5:8;                % Define grid points (y)
[X,Y] = meshgrid(x,y);      % Create grid
Z = (X+Y).^2;               % Compute the function

figure
mesh(X,Y,Z)                 % Plot the mesh surface
```

3D Graphics: mesh example

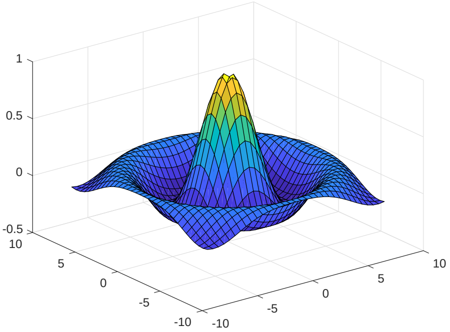
We would like to plot the function $z = f(x, y) = (x + y)^2$

```
x = -8:.5:8;               % Define grid points (x)
y = -8:.5:8;               % Define grid points (y)
[X,Y] = meshgrid(x,y);     % Create grid
Z = (X+Y).^2;              % Compute the function

figure
mesh(X,Y,Z)                % Plot the mesh surface
```

3D Graphics: SURFace plot

A surface plot looks like a mesh plot, except that the spaces between the lines, called patches, are filled in.

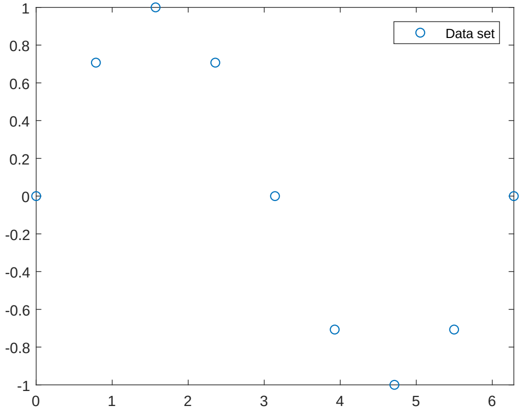
```
surf(X,Y,Z)
```



Data interpolation and regression

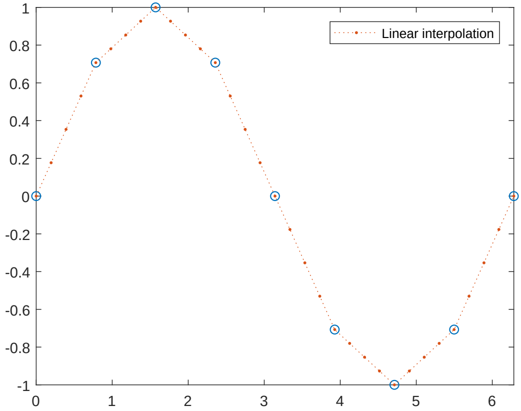
Interpolation: basics

Interpolation is a way of estimating the values of a function between the points given by some set of data points.



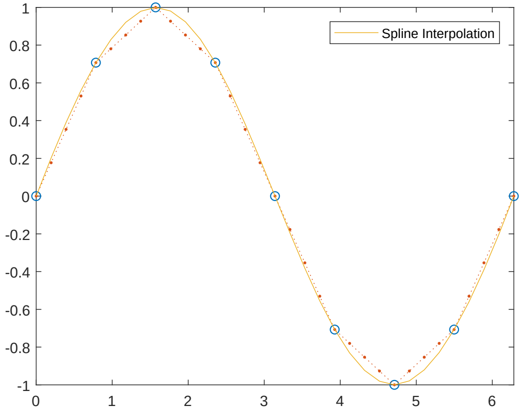
Interpolation: basics

Interpolation is a way of estimating the values of a function between the points given by some set of data points.



Interpolation: basics

Interpolation is a way of estimating the values of a function between the points given by some set of data points.



Interpolation in MATLAB

`Vq = interp1(X,V,Xq,method)` interpolates to find `Vq`, the values of the underlying function `V=F(X)` at the query points `Xq` using the specified method

Typical methods: `'linear'`, `'nearest'`, `'cubic'`, `'spline'`

```
x = 0: pi/4 :2*pi;          % length(x)-> 9
v = sin(x);                 % We know the function only at x

xq = 0: pi/16 :2*pi;        % Can we obtain the function inbetween
                            % the points of x?   (length(xq)-> 16)

vq1 = interp1(x,v,xq,'linear');       % Linear interpolation

plot(x,v,'o')               % Plot original data points (9 points)
hold on
b = plot(xq,vq1,':.');      % Plot interpolated data (16 points)
```

Interpolation in MATLAB

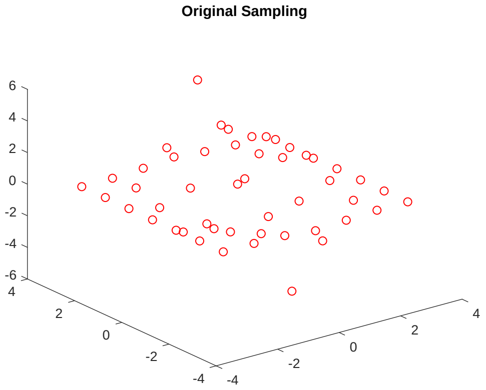
`Vq = interp1(X,V,Xq,method)` interpolates to find `Vq`, the values of the underlying function `V=F(X)` at the query points `Xq` using the specified method

Typical methods: `'linear'`, `'nearest'`, `'cubic'`, `'spline'`

```
x = 0: pi/4 :2*pi;          % length(x)-> 9
v = sin(x);                 % We know the function only at x

xq = 0: pi/16 :2*pi;        % Can we obtain the function inbetween
                            % the points of x?   (length(xq)-> 16)

vq1 = interp1(x,v,xq,'linear');      % Linear interpolation

plot(x,v,'o')               % Plot original data points (9 points)
hold on
b = plot(xq,vq1,':.');      % Plot interpolated data (16 points)
```

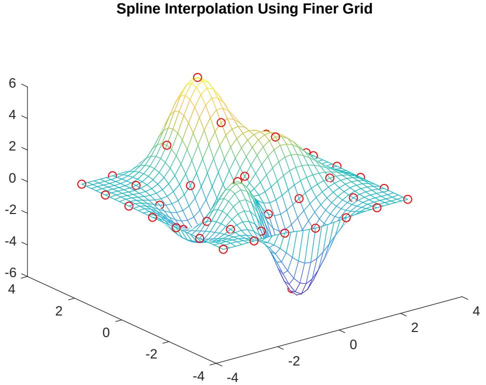
Two dimensional interpolation

Based on the same underlying ideas as 1D interpolation, `Vq = interp2(X,Y,V,Xq,Yq,method)` returns interpolated values `Vq` of a function of two variables `V=F(X,Y)` at query points `(Xq,Yq)` using the specified interpolation method



Two dimensional interpolation

Based on the same underlying ideas as 1D interpolation,
`Vq = interp2(X,Y,V,Xq,Yq,method)` returns interpolated values `Vq` of a function of two variables `V=F(X,Y)` at query points `(Xq,Yq)` using the specified interpolation method



Two dimensional interpolation: example

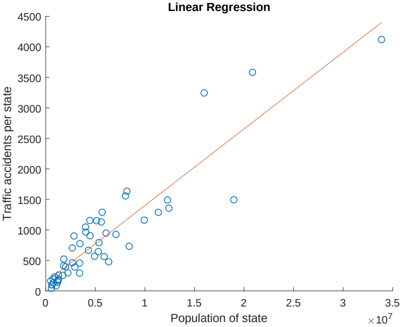
```
[X,Y] = meshgrid(-3:3);        % Original data has 49 points
V = peaks(X,Y);


figure
plot3(X,Y,V,'ro');             % Plot original data set

[Xq,Yq] = meshgrid(-3:0.1:3);             % We would like to find
Vq = interp2(X,Y,V,Xq,Yq,'spline');       % z-data at 3721 points

hold on
mesh(Xq,Yq,Vq);                % Plot mesh surface
title('2D interpolation');
```

Regression

Regression **models** the relation between a dependent, or response, variable and one or more independent, or predictor, variables

Simple linear regression:

$$y = \beta_0 + \beta_1 x + \epsilon$$



Regression: basics

Regression **models** the relation between a dependent, or response, variable and one or more independent, or predictor, variables

Simple linear regression:

$$y = \beta_0 + \beta_1 x + \epsilon$$

- $x \in \mathbb{R}^n$ is a set of observation of the predictor variable
- $y \in \mathbb{R}^n$ is a set of observation of the response variable
- $\epsilon \in \mathbb{R}^n$ is the (observation) error term
- Linear in the coefficients (may be nonlinear in x)

How do we “learn“ the model, that is, parameters β_0 and β_1?

Regression: basics (2)

Simple linear regression:

$$y = \beta_0 + \beta_1 x + \epsilon$$

- Start with a set of observed values (x_1, y_1), (x_2, y_2), $\ldots$, (x_n, y_n)
- Represent these equations in matrix form as

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}}_{b}$$

`b = regress(y,X)` returns the vector `b` of regression coefficients in the linear model `y = X*b`

Regression: example

We are interested in modeling the (linear) relation between the number of traffic accidents and the population in available data from several states

```
load accidents
x = hwydata(:,14);   %Population of states
y = hwydata(:,4);    %Accidents per state

X = [ones(length(x),1) x];  % y = b0 + b1*x
b = regress(y, X)           % Regression

scatter(x,y)         % Plot initial data set
hold on
yCalc1 = X*b;        % Generate and plot samples from the model
plot(x,yCalc1)
xlabel('Population of state')
ylabel('Traffic accidents per state')
title('Linear Regression')
```

Regression: exercise

The sample data set `carsmall` contains data representing vehicles weight, horsepower, and MPG. It is reasonable to model the MPG in relation to weight and horse power according to:

$$MPG = \beta_0 + \beta_1\text{*weight} + \beta_2\text{*horsepower} + \beta_3\text{*(weight*horsepower)}$$

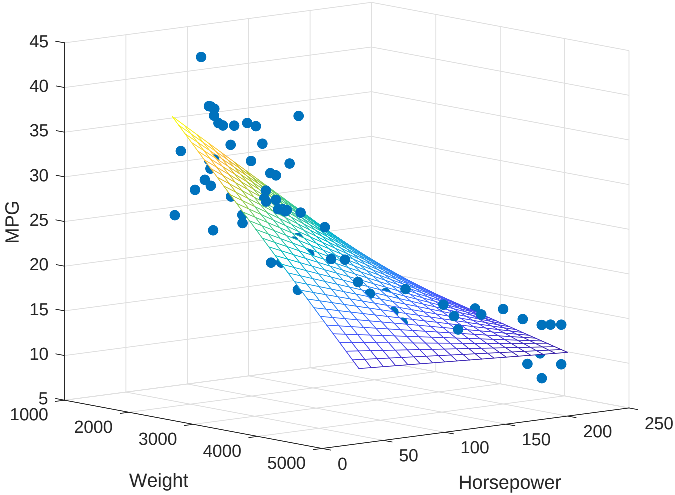
Find β_1, β_2, and β_3

```
load carsmall
x1 = Weight;
x2 = Horsepower;
y = MPG;            % y = b0 + b1*x1 + b2*x2 + b3*(x1*x2)
```

Regression: exercise (2)

```
load carsmall
x1 = Weight;
x2 = Horsepower;
y = MPG;            % y = b0 + b1*x1 + b2*x2 + b3*(x1*x2)

X = [ones(size(x1)) x1 x2 x1.*x2];  % Define matrix X
b = regress(y,X)                    % Regression



scatter3(x1,x2,y,'filled')  % Plot sample data
hold on
% Generate and plot samples from the model
x1fit = min(x1):100:max(x1);
x2fit = min(x2):10:max(x2);
[X1FIT,X2FIT] = meshgrid(x1fit,x2fit);
YFIT = b(1) + b(2)*X1FIT + b(3)*X2FIT + b(4)*X1FIT.*X2FIT;
mesh(X1FIT,X2FIT,YFIT)
```

Regression: exercise (3)



Cell arrays and structures

Cell arrays

Cell arrays are MATLAB arrays whose elements are cells. Each cell can any MATLAB data type

- Cell arrays can be useful when we wave a series of heterogeneous data types in for (while) loops (e.g. sequence of strings)
- Cell arrays are created through assignment statements:
  `C(i,j)= {data}`

Cell arrays: creation

```
A(1,1) = {[1 2 3; 4 5 6; 7 8 9]}
A(1,2) = {'This is a string'}
A(2,1) = {2}
A(2,2) = {-12:2:0}
```

```
>> A

A =

  2×2 cell array

    {3×3 double}    {'This is a string'}
    {[       2]}    {1×7 double        }
```

Structures

Structures are similar to cell arrays, but instead of addressing elements by number, their elements are addressed by field

```
circle.radius = 2.5;
circle.center = [0 1];
circle.linestyle = '--';
circle.color = 'red';
```

```
>> circle

circle =

  struct with fields:

       radius: 2.500000000000000
       center: [0 1]
    linestyle: '--'
        color: 'red'
```

GradQuat at UCR

GradQuant:

- Website: http://gradquant.ucr.edu
- Hours: Monday Thursday, 9 am - 3 pm
- Location: Life Sciences Building, Room #1425

If you seek help with MATLAB:

- Drop-in hours (Gianluca): Thursday 12pm-2pm
- Schedule a consultation (Gianluca)
- Email: GQstaff1@ucr.edu

MATLAB resources:
http://gradquant.ucr.edu/gq-calendar/workshop-resources/