Basic Statistics in MatLab

- The MatLab Statistics Toolbox provides functions to analyze and model data as well as perform machine learning on this data. In particular, you can perform regression and prediction, generate random numbers for Monte Carlo simulations, use statistical plots to explore data, and and perform hypothesis testing with this toolbox.

- Since CS2035 is **not** a statistics course and students are not assumed to have a statistical background, we do not cover the Statistics toolbox. Rather, we look at the simple kind of statistical analysis available in standard MatLab without this toolbox.

Random Numbers

- We'll illustrate many basic statistics computations using random numbers.
- MatLab provides commands to generate "pseudo" random numbers. These are usually numbers in a long sequence of numbers satisfying "randomness" properties.
- `randn('seed',1)` sets the normal random number generator to use a specific seed. (This is good for testing purposes as if everyone uses this command in their programs, they will get the same random number sequences in each of their MatLab sessions. In this case, code comparisons are possible.)

- Below, we give the code for a (very) early random number generator `rand()` used in C. It is no longer in use these days as there are now much better generators. This random number generator generates a very long sequence of numbers $(> 1,000,000)$ before it begins to repeat.

```
/*******************************************************/
/* Random function to generate numbers in the interval */
/* 0.0 to 1.0                                          */
/*******************************************************/
#include <stdio.h>

/* Change "next" to get different random number sequences */
unsigned long int next = 1;

float rand(void)
{
next = next * 1103515245 + 12345;
return((float) (next /65536 % 32768)/32767.0);
}
```

- Note that `next` acts like a "seed" whose value is the starting point for generation of the next number.

Normal/Gaussian Random Number Generation

- The most common distribution of numbers in nature follows the **Normal** or **Gaussian** distribution. For example, the distribution of grades on an exam taken by a large class of university students is usually normally distributed. [This is where the idea of “belling” grades comes from.]

- A 1D normal or Gaussian distribution of numbers follows the equation:

$$G(x,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-2(x-\mu)^2}{2\sigma^2}},$$

  where μ is the mean and σ is the standard deviation of the distribution of numbers. σ^2 is called the variance. The area under the curve $G(x,\sigma)$ for x in various ranges is given in the table below using **L18area_1D_gaussian.m**:

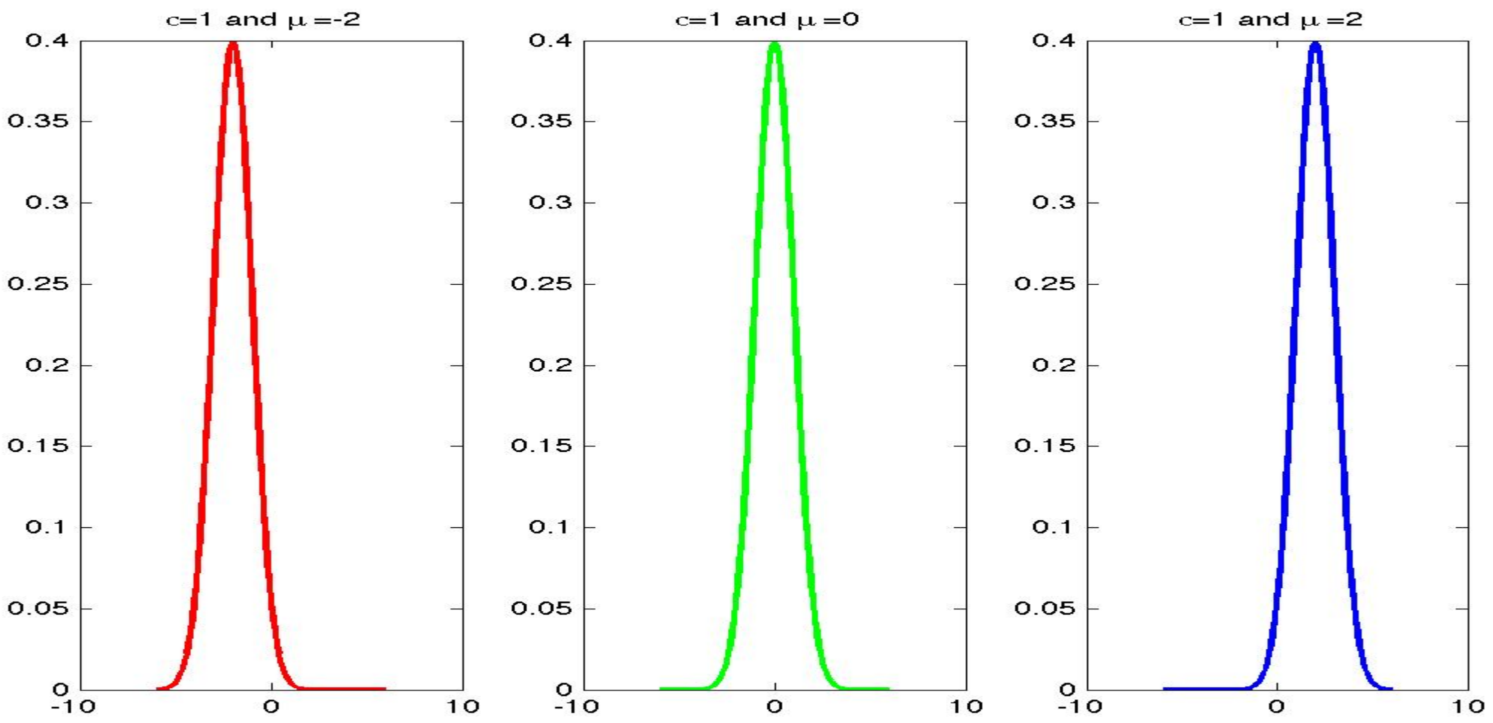
Range	Area
-1 to +1	0.682689
-2 to +2	0.954500
-3 to +3	0.997300
-4 to +4	0.999937
-5 to +5	0.999999
-∞ to +∞	1.0

The area under the 1D Gaussian for x in the above ranges with $\sigma = 1.0$ and $\mu = 0.0$.

- Plots of the 1D Gaussian for $\mu = -2$, $\mu = 0$ and $\mu = 2$ can be made by the following MatLab code (**L18make_1D_gaussian.m**):

```
sigma=1.0;
x=-6.0:0.1:6.0;
for mu=-2:2:2
g=1/(sqrt(2*pi)*sigma)*exp(-(x-mu).*(x-mu)/(2*sigma*sigma));
figure
plot(x,g,'r-','linewidth',2.0);
title(['1D Gaussian with sigma=' num2str(sigma) ' and mu=' num2str(mu)]);
print(['1Dgauss-sigma=' num2str(sigma) '-mu=' num2str(mu) '.jpg'],'-djpeg');
end
```

- This program produces the following plots:



1D Gaussian plots $(\sigma = 1.0)$ for means (a) $\mu = -2$, (b) $\mu = 0$ and (c) $\mu = 2$.

- A 2D normal or Gaussian distribution of numbers follows the equation:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{\frac{-((x-\mu_x)^2+(y-\mu_y)^2}{2\sigma^2}},$$

where μ_x and μ_y are the 2D means in the x and y dimensions and σ is the standard deviation of the distribution of numbers (this could be different in the x and y dimensions as well). σ^2 is again called the variance. The area under the surface $G(x, y, \sigma)$ for x and y in various ranges is given in the table below using **L18area_2D_gaussian.m**:

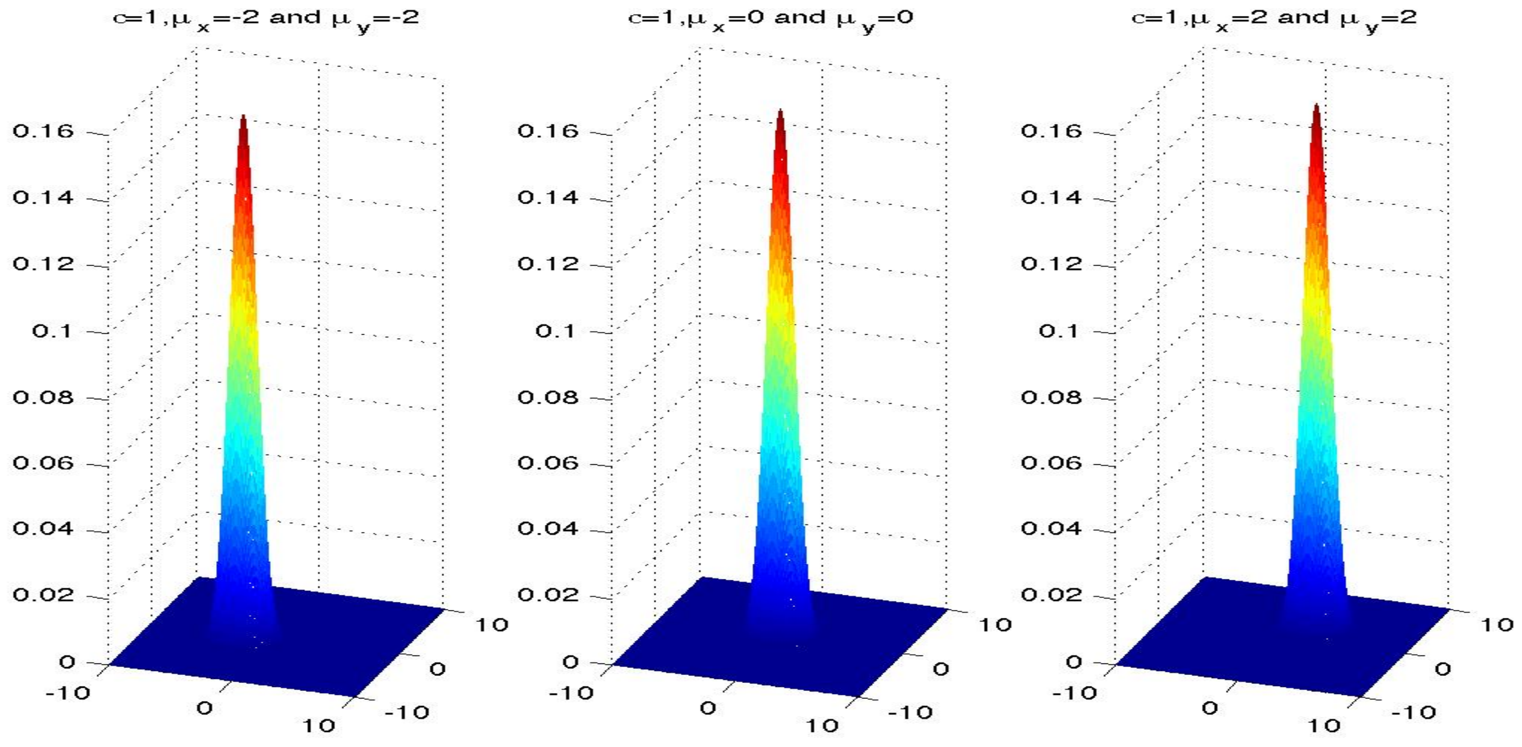
Range	Area
-1 to +1	0.466065
-2 to +2	0.911070
-3 to +3	0.994608
-4 to +4	0.999873
-5 to +5	0.999999
-∞ to +∞	1.0

The area under the 2D Gaussian for x and y in the above ranges with $\sigma = 1.0$ and $\mu_x = \mu_y = 0.0$.

- Plots of the 2D Gaussian for $\mu_x = \mu_y = -2$, $\mu_x = \mu_y = 0$ and $\mu_x = \mu_y = 2$ can be made by the following MatLab code (**L18make_2D_gaussian.m**):

```
sigma=1.0;
x=-10:0.1:10;
y=-10:0.1:10;
[X,Y]=meshgrid(x,y);
mu_base=-2.0;
mu_inc=2.0;
for i=1:3
mu_x=mu_base+rem(i-1,3)*mu_inc;
mu_y=mu_x;
Z=1.0/(2*pi*sigma*sigma)*exp(-((X-mu_x).^2+(Y-mu_y).^2)...
     /(2*sigma*sigma));
subplot(1,3,i)
mesh(X,Y,Z)
view([20 10]);
title(['\sigma=' num2str(sigma) ',\mu_x=' num2str(mu_x) ...
       ' and \mu_y=' num2str(mu_y)]);
end % i

print(['Plots_2Dgauss-sigma-' num2str(sigma) '.jpeg'],'-djpeg');
```

- This program produces the following plots:



2D Gaussian plots ($\sigma = 1.0$) for means (left) $\mu_x = \mu_y = -2$, (middle) $\mu_x = \mu_y = 0$ and (right) $\mu_x = \mu_y = 2$.

Generating Normal (Gaussian) Distributions of Numbers in MatLab

- Consider the MatLab code:

```
% Initialize the random number generator
randn('seed',1)
x=randn(1000,1);
y=randn(1000,1);
```

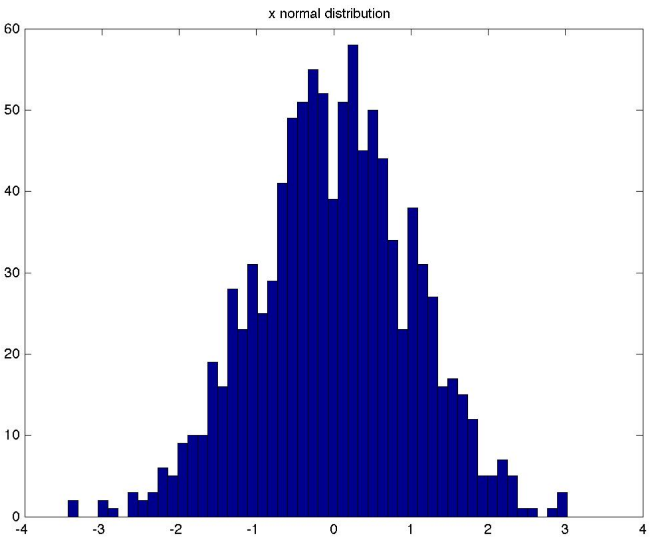
  generates 1000 component random numbers satisfying the normal (Gaussian) distribution.

- The figure below shows these x and y numbers plotted as histograms. Note how their distribution looks like the 1D Gaussian plot above.

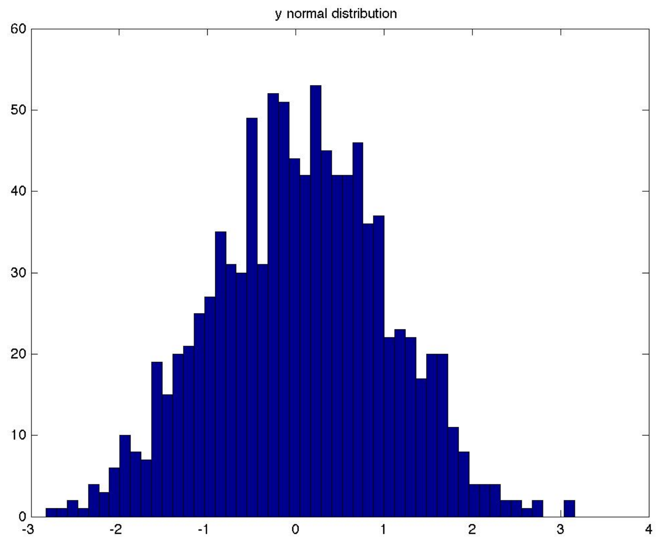
- We can see that x and y are normally distributed by plotting their histograms using:

```
randn('seed',1);
x=randn(1000,1);
y=randn(1000,1);
figure
hist(x,50)
title('x normal distribution');
print x_normal_histogram.jpg -djpeg
figure
hist(y,50)
title('y normal distribution');
print y_normal_histogram.jpg -djpeg
```

The figure below shows the 2 histograms produced:



(a)



(b)

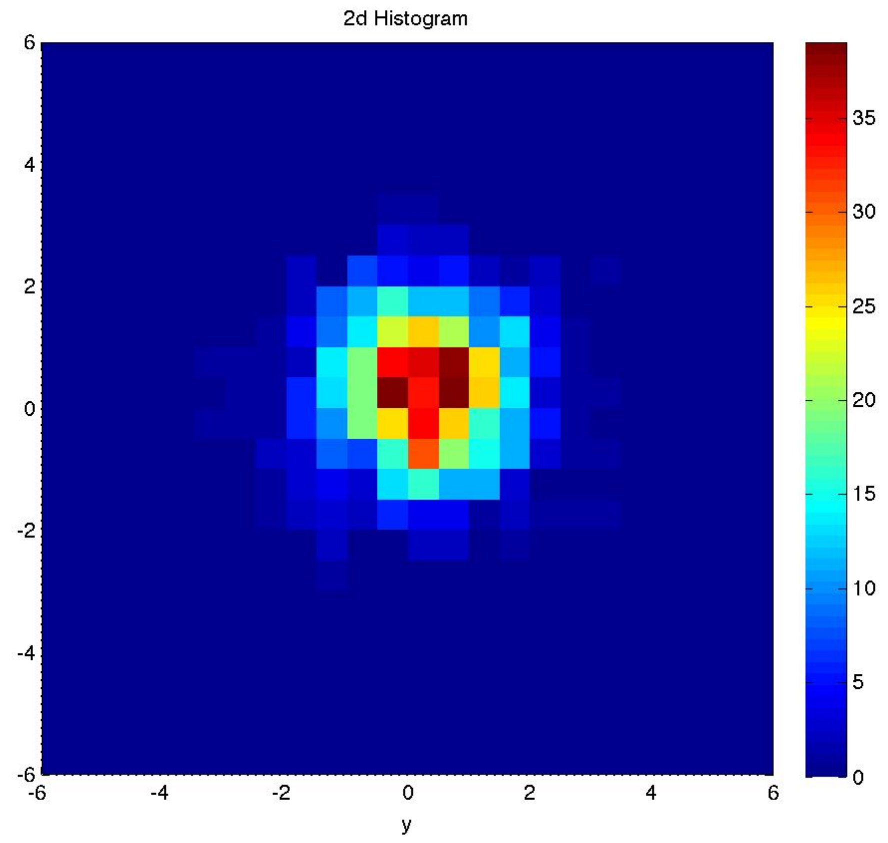
The normal distributions of variables x and y.

- In theory we could plot a 2D histogram, but the front bars would hide the bars in the back. It is much better to print a 2D image of the 2D histogram. We can use a program called `L18histo2D` to plot a 2D histogram as an image with a colorbar. This function (histo2D.m) is available on the mathworks file exchange website.[1] The MatLab code (**L18make_2D_histogram.m**):

```
randn('seed',1)
x=randn(1000,1);
y=randn(1000,1);
min(x(:))
max(x(:))
D=[x,y];
H = L18histo2D(D,[-6 6],25,[-6 6],25, 'x','y','2d Histogram') ;
print 2Dhistogram.jpg -djpeg
```

produces the 2D image:

[1] http://www.mathworks.com/matlabcentral/fileexchange/8422-2-dimensional-histogram



The 2D histogram of the two 1D random normal arrays x and y binned and then plotted against each other. x and y range from -6 to +6 and there are 25 bins in the two dimensions.

Computing the Mean and Standard Deviation

- Consider the MatLab code (**L14mean_std_var.m**):

```
randn('seed',1)
x=randn(1000,1);
fprintf('The mean of x is %f\n',mean(x));
fprintf('The standard deviation of x is %f\n',std(x));
fprintf(['The variance of x is %f which is the\n'...
         'standard deviation squared %f\n'],...
          var(x),std(x)*std(x));
```

  produces the output:

```
The mean of x is 0.001962
The standard deviation of x is 1.009603
The variance of x is 1.019298 which is the
standard deviation squared 1.019298
```

- The mean and standard deviation of a sequence of n numbers can be computed as:

$$\mu = \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}, \quad \sigma_0 = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \text{ and } \sigma_1 = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

- Sometimes n is used instead of $n-1$ ($n-1$ is used because there is one less "degree of freedom"). `std(x)` or `std(x,0)` return σ_0 while `std(x,1)` returns σ_1,
- The standard deviation σ or the variance σ^2 tells you how much the data varies from its mean. Low σ values mean the data is (nearly) uniform (all data are roughly the same as the mean). High σ values mean there is large variation of the data about the mean.
- For the above data, the mean is about 0.0 and the standard deviation is about 1.0 (`randn` returns random numbers with mean 0.0 and standard deviation 1.0), unless your set some default arguments to other values.

Skewness and Kurtosis

- **Skewness** is a measure of symmetry, or more precisely, the lack of symmetry. A distribution, or data set, is symmetric if it looks the same to the left and right of the center point.

- **Kurtosis** is a measure of whether the data are peaked or flat relative to a normal distribution. Datasets with high kurtosis tend to have a distinct peak near the mean, decline rather rapidly, and have heavy tails. Datasets with low kurtosis tend to have a flat top near the mean rather than a sharp peak. A uniform distribution is an extreme case of low kurtosis.

- For a list of numbers y equal to $(y_1, y_2, y_3, y_4, ...y_n)$ skewness can be computed as:

$$s = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^3}{(n-1)\sigma^3},$$

  where $\bar{y}$ is the mean, σ is the standard deviation and n is the number of data points.

- The skewness for a normal distribution is zero and any symmetric data should have a skewness near zero. Negative values for the skewness indicate data that are skewed left and positive values for the skewness indicate data that are skewed right. By skewed left, we mean that the left tail is long relative to the right tail. Similarly, by skewed right, we mean that the right tail is long relative to the left tail.

- For a list of numbers y equal to $(y_1, y_2, y_3, y_4, ...y_n)$ kurtosis can be computed as:

$$k = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^4}{(n-1)\sigma^4},$$

where $\bar{y}$, σ and n are as before. For a perfect normal distribution, k is 3.

- Sometimes a **excess kurtosis** definition is used when 3 is subtracted from k, i.e. $k_e = k - 3$. Now k_e for a perfect normal distribution is 0 and positive k_e indicates a ”peaked” distribution while negative k_e indicates a ”flat” distribution.

- We can compute skewness and kurtosis in MatLab as in the **L18skewness_kurtosis_example.m** function:

```
% http://www.mathworks.com/help/stats/random.html
randn('seed', 1);
x = randn(1000,1);
fprintf('skewness of 1000 randon normal distributed numbers: %f\n',...
         skewness(x));
fprintf('kurtosis of 1000 randon normal distributed numbers: %f\n',...
         kurtosis(x));
```

which has output:

```
skewness of 1000 randon normal distributed numbers: -0.078637
kurtosis of 1000 randon normal distributed numbers: 3.025833
```

Linear Regression

- Linear Regression can be used to examine the relationship between 2 datasets. One way to do this is a **least squares** fit using the MatLab function `regress`. We have already seem the use of `polyfit/polyval` to do this in the previous lecture.

- Linear regression (basically a least squares fit with some additional statistics) examines the linear relationship between 2 datasets by fitting a line to the data. The data is assumed to have measurement error.

- MatLab function `regress` computes the linear regression coefficients and corresponding statistics for 2 datasets. Basically, for some `x` coordinates, what is the line that fits some `y` measurements for those `x` values?

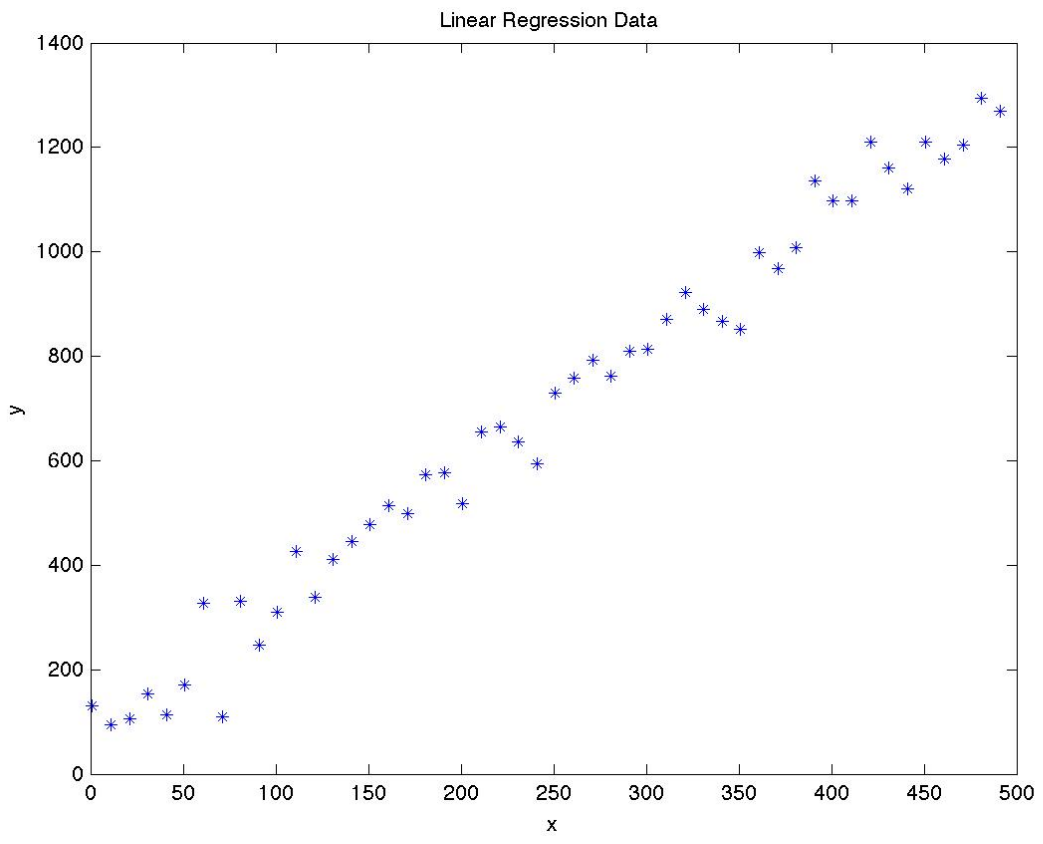
- Consider the modified MatLab linear regression example in **L18linear_regression.m** from:

```
www.agnld.uni-potsdam.de/~marwan/matlab-tutorials/html/statistics.html
----------------------------------------------------------------------
randn('seed',1);
% x is a column vector with values 1, 11, 21, ..., 491
% The next value 501 is bigger than 500
x = [1:10:500]';
% y is a 50 component column vector of a line
% with slope 2.5 and y intercept 80 perturbed
% with 50 component normal random noise vector
% computed as 50*randn(50,1)
y = 2.5*x+80+50*randn(50,1);
figure
plot(x,y,'*')
xlabel('x');
ylabel('y');
title('Linear Regression Data');
print linear_regression_data.jpg -djpeg

% perform linear regression
% ones(length(x),1) is a 50 component column vector of 1's
[b, bint] = regress(y,[ones(length(x),1),x],0.05)
% plot x versus y=slope*x+y_intercept
% b(1) - intersection of line with y axis
% b(2) - slope of line
hold on
```

```
plot(1:500,b(2)*[1:500]+b(1),'-r','linewidth',2)
xlabel('x');
ylabel('y');
title('Linear Regression Fit');
print linear_regresssion_line.jpg -djpeg

[b,bint,residuals]=regress(y,[ones(length(x),1),x],0.05);
figure
plot(residuals,'-g','linewidth',2)
xlabel('x');
ylabel('Residuals')
axis([0 50 -300 300]);
title('Linear Regression Residuals');
print linear_regression_residuals.jpg -djpeg
```

- The first part of this code generates a line $y = 2.5x + 80$, where x is a 50 component vector with values `1,11,...,481,491`. Scaled normal noise, `50*randn(50,1)`, is added to y to simulate measurement error. This noise has mean $\mu = 0.0$ and standard deviation $\sigma = 1$) and is scaled by 50. The scale value as determined by trial and error.

- The following data points are plotted:



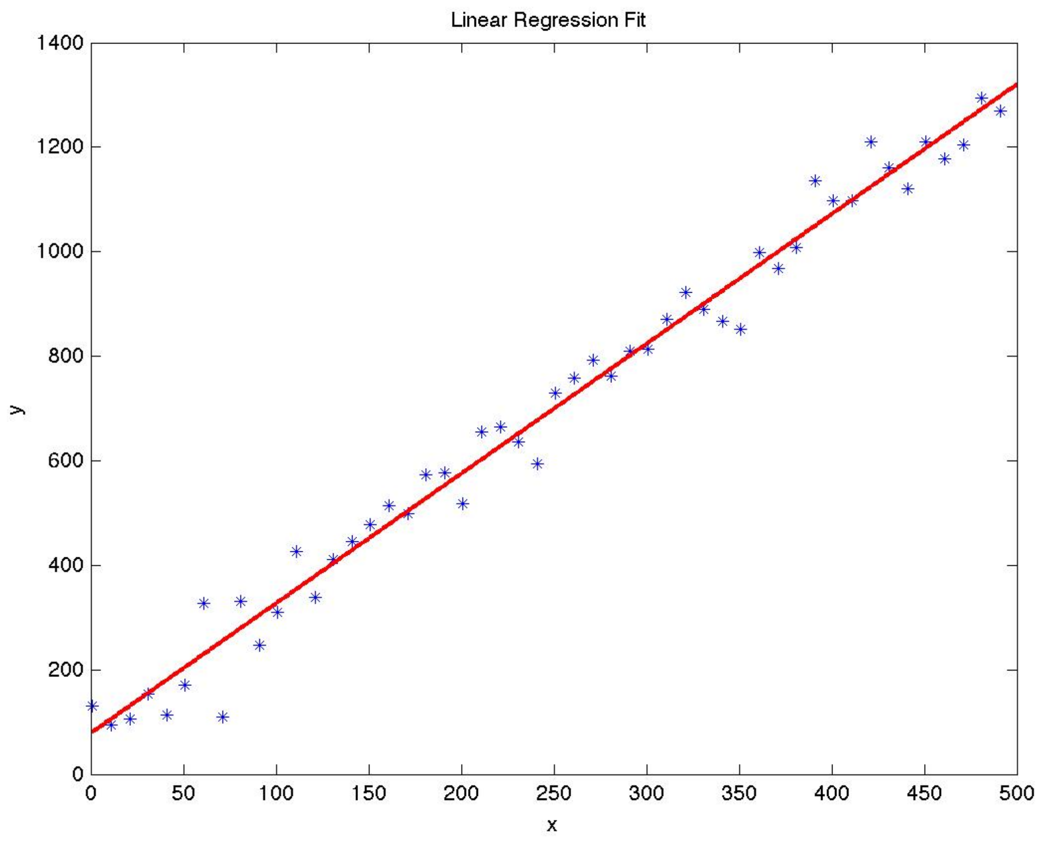
Linear regression data points.

- The second part of this code fits a line via Least Squares to the data. The column vectors `y` and `[1 x]` are passed to `regress` and the y intercept and slope are computed and returned in `b(2)` and `b(1)`. The argument value 0.05 (the significance level) indicates a 95% confidence interval. The code prints:

```
b = 78.1034
     2.4832
bint = 50.3461  105.8608
        2.3859    2.5805
```

  Thus the slope is 2.4832 (versus 2.5 for the error-free data) and y intercept is 78.1034 (versus 80 for the error-free data). 95% confidence level means that 95% of the time for random error, the value of the slope will be in `[2.3859 2.5805]` and the value of the y intercept will be in `[50.3461 105.8608]`.

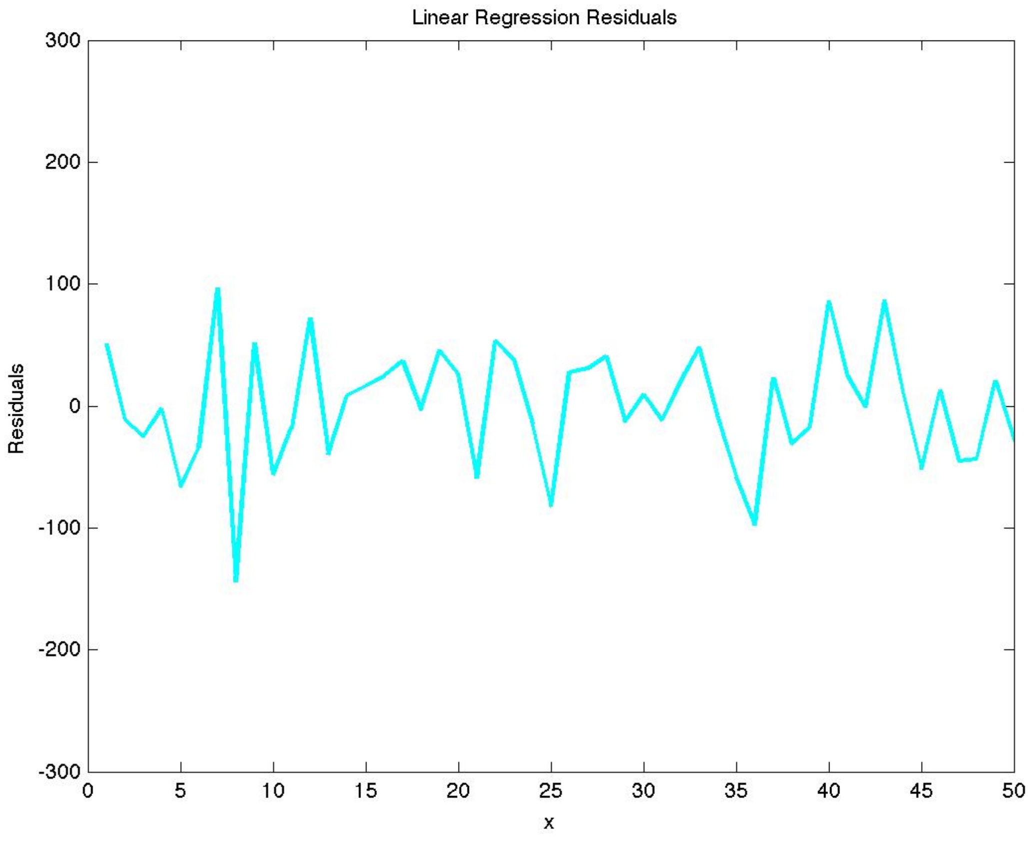
- The line fit to the data is show in the figure below:



Linear regression line fit to the data points.

- The third part of the code shows that the `residuals` vector can also be captured from `regress`. This just measures the distance of each fitted point with the measurement point, i.e. $r = y - mx - b$, where r is the column residual vector.

- The next figure shows the residual values plotted against x. Note that we explicitly control the y axis limits to be in `[-300 300]`. These residuals are actually quite small relative to the data points.



Residuals of the linear regression fit.

Nonlinear Regression

- What about if the 2 datasets do **NOT** have a linear relationship. For nonlinear relationships, `polyfit` is more appropriate.
- Consider the modified MatLab nonlinear regression example in **L18nonlinear_regression.m** from:

  www.agnld.uni-potsdam.de/ marwan/matlab-tutorials/html/statistics.html:
- The code is as follows:

```
randn('seed',1);
x=[1:10:500]';
y=sin(x/50)./x+0.002*randn(50,1);
figure
plot(x,y,'*')
print nonlinear_regression_data.jpg -djpeg

% order 5 polynomial fir to the data
p=polyfit(x,y,5);
```

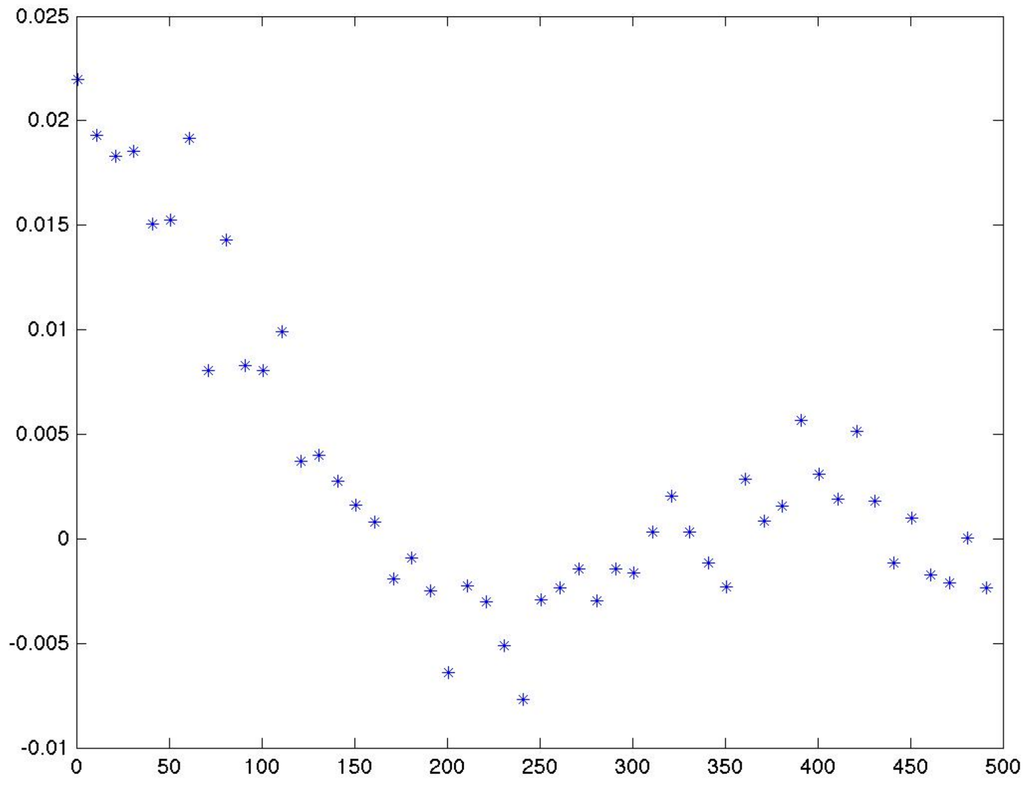
```
hold on
plot(x,polyval(p,x))
print nonlinear_regression_fit.jpg -djpeg

[p s]=polyfit(x,y,5);
[y_fit delta]=polyval(p,x,s);
% plot on the existing polynomial plot
hold on
plot(x,y_fit+2*delta,'r:','linewidth',2);
hold on
plot(x,y_fit-2*delta,'g:','linewidth',2);
print nonlinear_regression_confidence_95.jpg -djpeg
```

- The first part of this code generates data points from a sinc-like function $\frac{\sin(x)}{x}$ with some scaled normal random error added to it. The figure below shows this data. Notice how curvey it is (a line would not be a good fit!!!).

- One thing: each time you run `polyfit` with a high order polynomial (5 here) a warning message may be printed by MatLab:

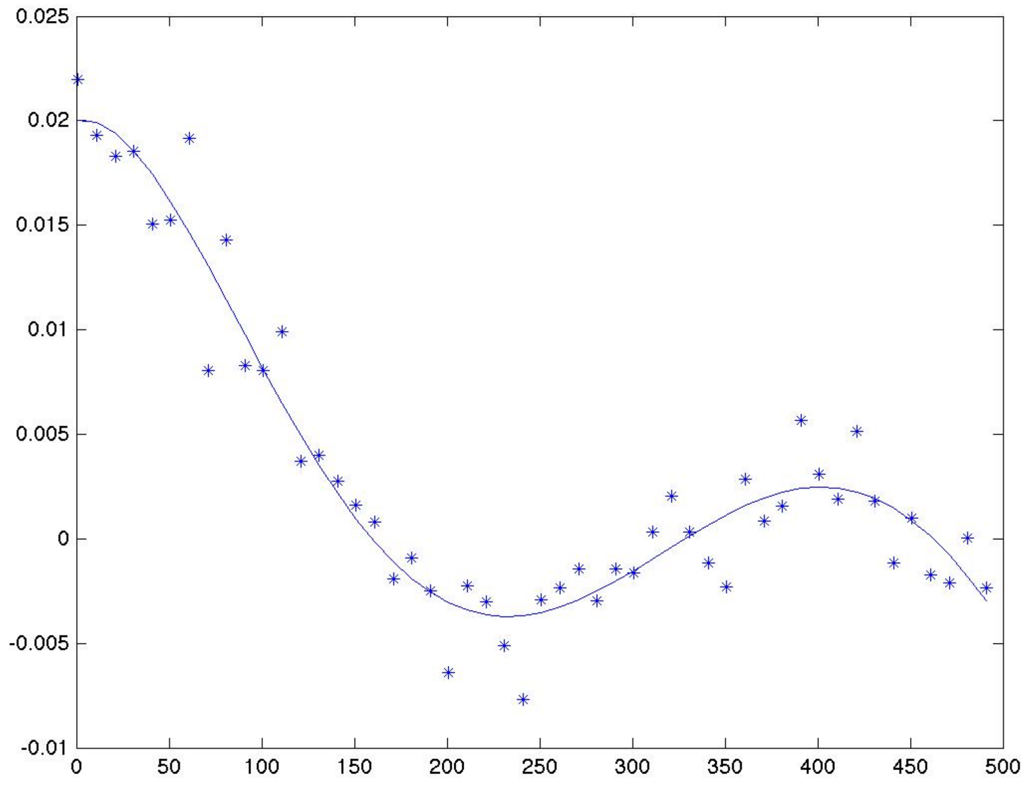
```
Warning: Polynomial is badly conditioned. Add points with
distinct X values, reduce the degree of the polynomial, or
try centering and scaling as described in HELP POLYFIT.
```

  The condition number of the Vandermonde matrix used by `polyfit` is quite high meaning the system of equation is not as stable as we might like.



Nonlinear regression data from a sinc-like function.

- The second part of this code calls `polyfit` to compute the 5th order polynomial best fitting the data. This curve (in blue) is plotted on top of the data points. The figure below shows this fit.

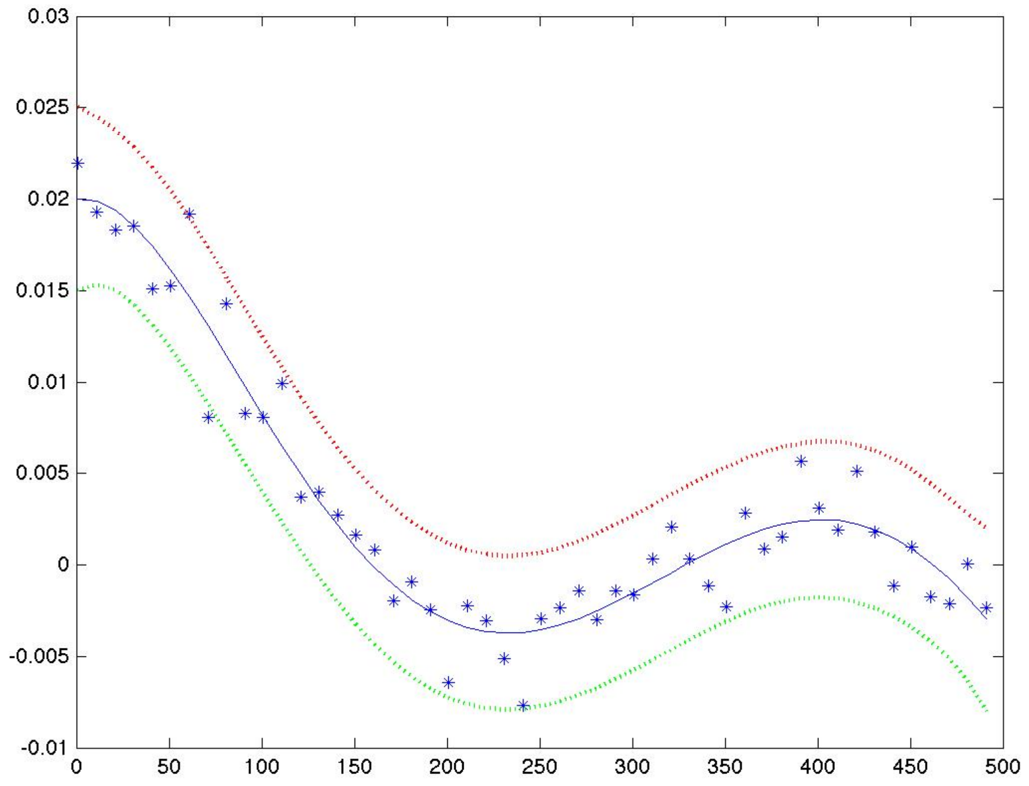


Nonlinear Regression 5th order polynomial fit to the data points.

- The third part of this code call `polyfit` again and captures a 2^{nd} output argument `s`. According to online MatLab documentation:

> `s` is a structure for use with polyval to obtain error estimates or predictions. Structure `s` contains fields `R`, `df`, and `normr`, for the triangular factor from a QR decomposition of the Vandermonde matrix of `x`, the degrees of freedom, and the norm of the residuals, respectively. If the data `y` are random, an estimate of the covariance matrix of `p` is `(Rinv*Rinv')*normr^2/df`, where `Rinv` is the inverse of `R`. If the errors in the data `y` are independent of each other and normal with constant variance, `polyval` produces error bounds that contain at least 95% of the predicted data points.

- `s` allows `polyval` to estimate `delta`, We plot `y_fit` $\pm$ `2*delta` as the 95% confidence interval for the fit. We various sets of different normal random noise we expect 95% of the data to lie between these curves. The figure below shows this plot:



95% confidence curves, red for +2*delta and green for -2 delta.