Matlab tools for ME 646

Goals of this exercise

- Make sure you are prepared to perform the data analysis tasks for the labs in this course. We give example Matlab code that you are likely to use in subsequent labs. You should use this presentation as a reference.
- The code is saved as a bmp so you cannot just copy and paste. My hope is that having to type it in will reinforce the concepts and burn in the pattern.
- The primary focus is processing data and plotting methods. There are a number of examples for how you can improve the quality of your plots.
- While the primary focus is Matlab, we also require you to do some things in Excel. This will probably be most useful for the engine lab.
- You will be at a disadvantage if you have not taken a Matlab course. We will work closely with you if you ask to help you over the bumps. The link https://swcarpentry.github.io/matlab-novice-inflammation/ has an intro to Matlab for novices. There is probably some useful info for non-novices, too. ***Please contact us if you are one of the people who has not had Matlab experience.***

Deliverables – You must create a single script file where the different sections are separated using cell mode programming (i.e. %% Part1) for the first three bullets. Use the Matlab Publish capability to publish the code and the plots to a pdf (you must set publishing options to pdf). (You must suppress console output for these files by making sure that all variable statements have ; at the end of the line. The = sign is highlighted in brown for those lines that do not have the ;.) Also, include the Excel Spreadsheet for the fourth bullet. These must be both uploaded to Canvas for the assignment

- Use Matlab to manually input the data for force vs. displacement of a spring from the table entitled Spring on the next page. Fit the data to a linear equation. Determine the spring constant using a first order fit. Create a plot with discrete symbols for the data and a line for the best fit line that does not extend past the range over which the fit was performed. On the plot, list the spring constant as "k = xxx.xx (N/mm)" on the plot using no more than four significant figures (k = num2str(variablename,4) will convert the number to a character string with four digits. The position of the decimal point in xx.xx could move but there should be no more than four significant digits.)
- Use Matlab to read in the Excel file entitled power2.xlsx using xlsread('power2.xlsx'). This is true stress-plastic strain data. Fit the data to the power law equation to determine the constants in the equation.
  - Use the fit command demonstrated for exponential fits but use 'power1' instead of 'exp1'. Using this feature is required.
  - Plot σ vs. ε using a solid line, show the fit line (using a dashed line), show the fit constants, with units, on the plot to 4 significant figures and the symbolic equation. $\sigma_{true} = K\varepsilon_{pl}^{\ n}$ Use the Greek symbols σ and ε in the axes labels. The strain is unitless and the stress is in MPa.
  - The fit will not be good in that there will be strain dependent residuals (difference between the data and the prediction). We will discuss this at a later date.
- Use Matlab to read in the voltage vs. time file, Long17psi.lvm, and process it to find the start time. In the program, use the function, peakfinder.m, to identify the peaks (it must be in the same directory as your data and program). Show plots indicating the location of the start time and then magnified plots showing the peak locations. Supply a table of array index of the peak locations, magnitude, and times. Use the code provided to create a table. The units are included in the header information and should be listed on the plots axes and in the tables.
- Use Excel to determine the coefficients, R_o and β, in the thermistor equation from the file, thermistor_lab1.xlsx. Provide a linear plot of R vs. T and log-linear plot of R vs. 1/T (log y axis, linear x axis). The second plot should show the exponential fit line (exp1) and the equation with units. Assume that the reference temperature, T_o, is 298.15 K. Use the relevant variables in the fit line equation (i.e. rename y to R, etc.). Label your axes appropriately. Submit the file as **yourlastname_lab1a.xlsx**

Spring constant data for Deliverable 1

Displacement (mm)	Force (N)
0.303	5.0
0.423	10.0
0.934	15.0
0.769	20.0
1.287	30.0
2.343	40.0
1.967	50.0

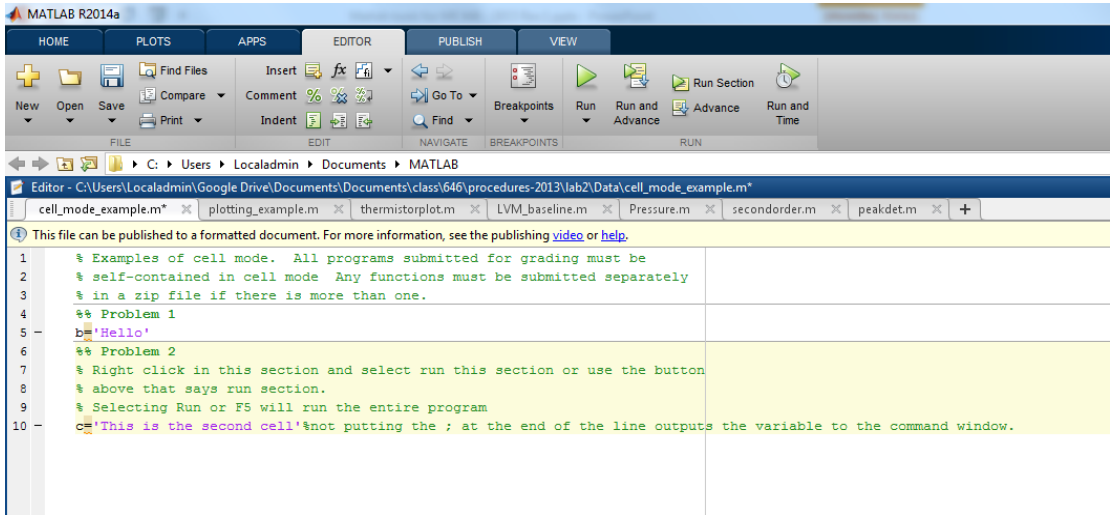
Purpose of the following slides

- The following slides contain detailed code examples to help accomplish tasks for the deliverables on the previous page. It is not the only way to accomplish these tasks. Most of these example codes can be slightly modified for use in subsequent assignments. This is one of the main goals of the assignment.
- While it may be annoying, I have saved these examples as pictures and you will not be able to copy and paste them in. My hope is that typing them will help you remember these code examples.
- In most cases, just copying the code and making minor changes will enable you to fulfill the assignment.
- Engineers are expected to precisely follow instructions however ridiculous they may seem. A significant portion of the grade in this assignment is focused on having you demonstrate that you can do this.
- Most measurements have no better than 0.01% uncertainty. This implies no more than four significant figures. The number of significant figures implies a certain level of uncertainty e.g. k =7.345619 implies an uncertainty of 0.0000001%. We will subtract points on all assignments that have unjustified significant figures.

Cell mode programming: Inserting two % signs creates a cell that can be independently evaluated from the other parts of the script file. We have asked you to provide your programs for three deliverables in in a single published code. The code below shows how it works and the screenshot shows how Matlab makes it visually apparent you are in a given cell by highlighting it in a sickly yellow.

You can run a section by clicking on the Run Section icon or by typing Cntrl-Enter when your cursor is in the cell.

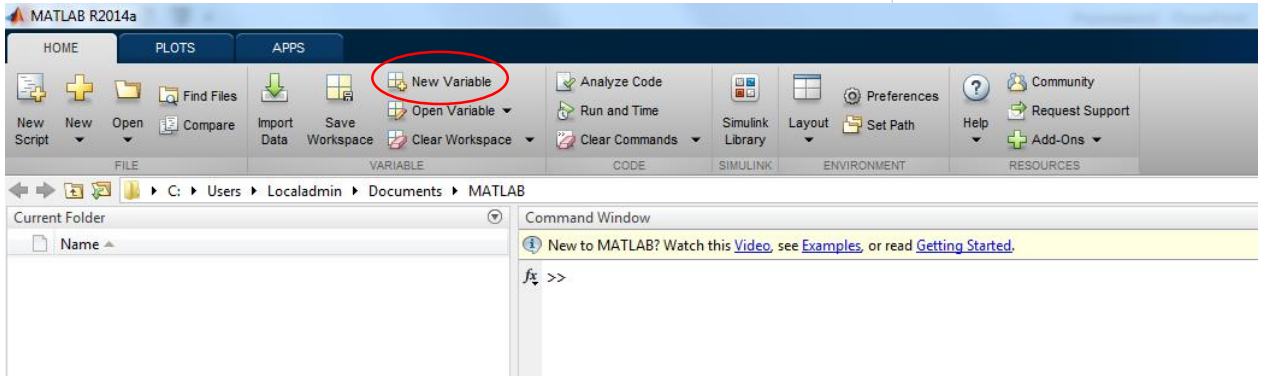
```
% Examples of cell mode.  All programs submitted for grading
must be
% self-contained in cell mode  Any functions must be submitted
separately
% in a zip file if there is more than one.
%% Problem 1
b='Hello'

%% Problem 2

% Right click in this section and select run this section or use
the button
% above that says run section.
% Selecting Run or F5 will run the entire program
c='This is the second cell'%not putting the ; at the end of the
line outputs the variable to the command window.
```

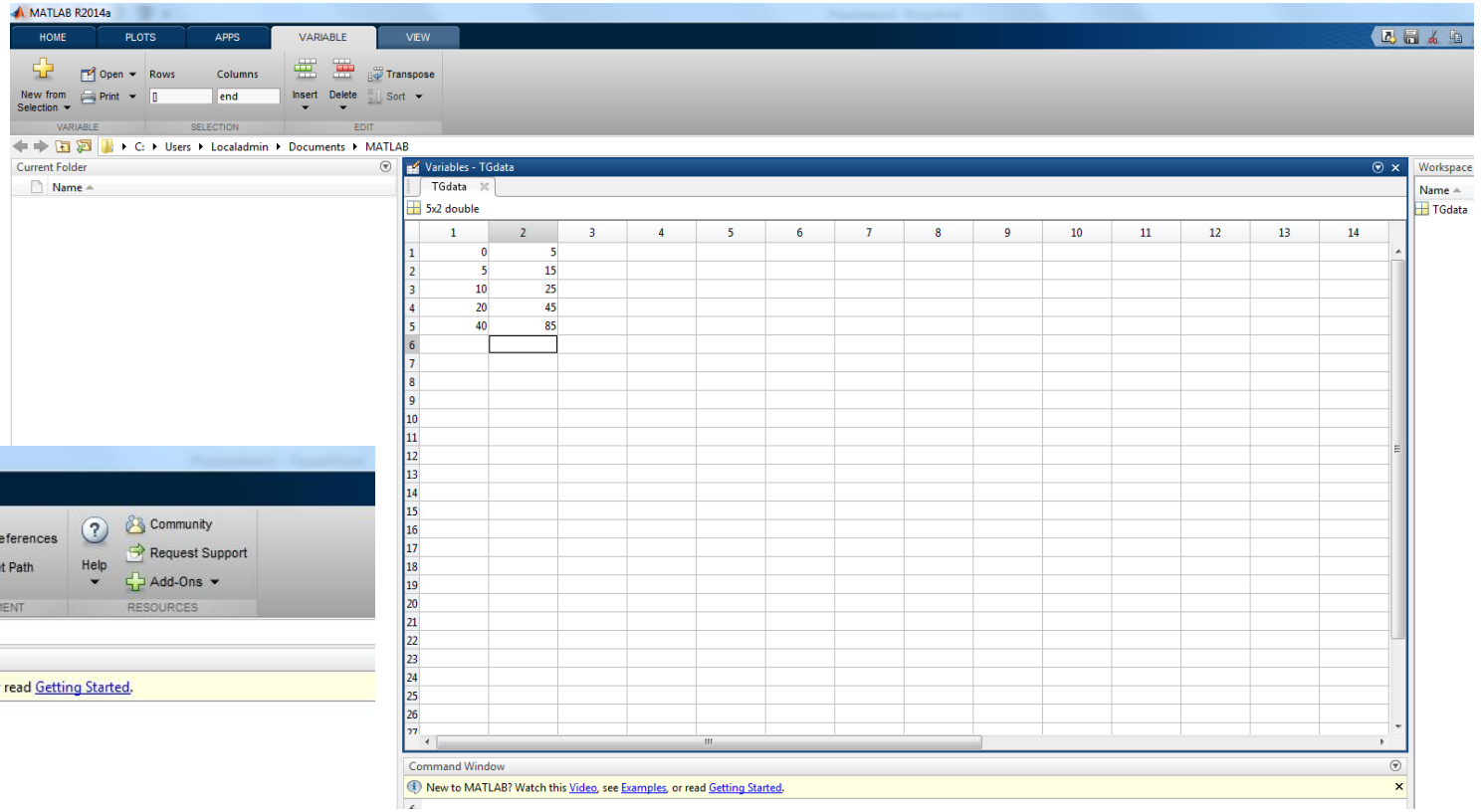


Entering, plotting, and fitting data

- We will use three ways to enter data
  - Manual entry in spreadsheet style format
  - Reading in from an Excel file using the xlsread command
  - Reading in structured data from an .lvm file (oscilloscope output) using importdata
- We will use linear plots, log-linear plots, and log-log plots
- We will fit data to a linear equation, an exponential equation, and a power law equation.

You can manually enter data by clicking on the New Variable icon (circled). When you click the New Variable icon, you will get a screen as shown below. Note that it automatically switches you from the Home to the Variable tab. Manually enter data from the next slide as shown below. You should rename the tab (I used TGdata) and it will rename the variable in the Workspace window. Save the data to a *.mat file by right clicking on the variable in the Workspace window. The *.mat file can be loaded at another time.





Arrays in Matlab

These descriptors will be useful in plot commands and restricting which rows you process.

- TGData is a two-dimensional array
- TGData(:,1) refers to all of the data in column 1
- TGData(:,2) refers to all of the data in column 2
- TGData(1,:) refers to the first row of data
- TGData(5,:) refers to the fifth row of data
- TGData(2:5,1) refers to rows 2 through 5 in column 1
- TGData(i:j,k) refers to rows i through j in column k. It is usually more convenient to use variables to specify rows and columns rather than numbers that might change when your criteria for selecting them change.

Add the data to the variable sheet and plot the data using the plot command just above the plot by entering in the editor window. (You can't copy and paste it because it is saved as an image.)

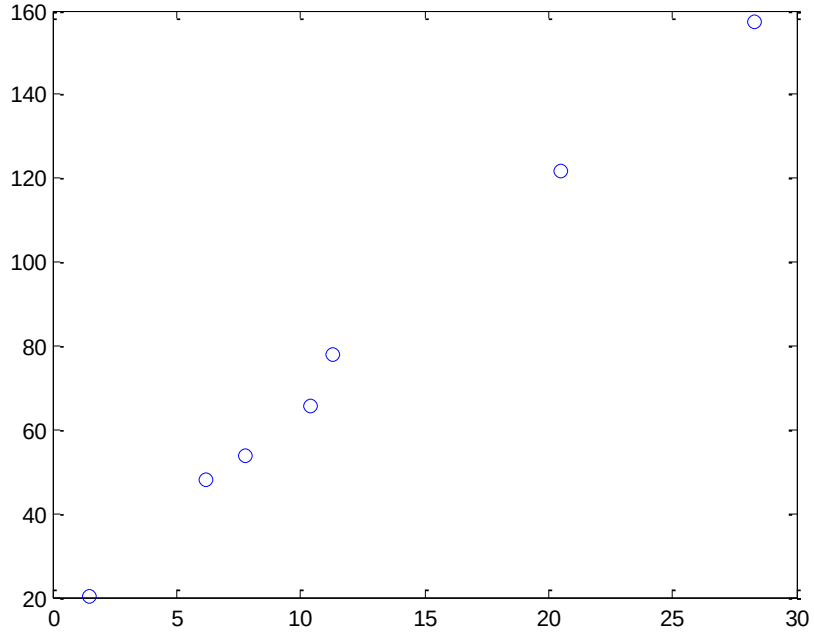
You should always plot discretely obtained data as discrete points when it was obtained individually. Use smaller markers, e.g. '.', when the data is denser. Do not use markers where the data is almost continuous. Se the last page of this PowerPoint to find a link with all the character codes for line style and color and plot symbols.

For subsequent reports in Word or presentations in PowerPoint, use Edit, Copy figure, and then paste it into Word or PowerPoint. **The low quality figures that will result from using screen shots, Paint, etc. will be graded as if you did not turn in a plot. There are some issues with plots from Apple computers – we will work with you on these.**

1.5	20.16
6.2	41.76
7.8	55.24
10.4	69.04
11.3	78.24
20.5	126.5
28.3	157.2

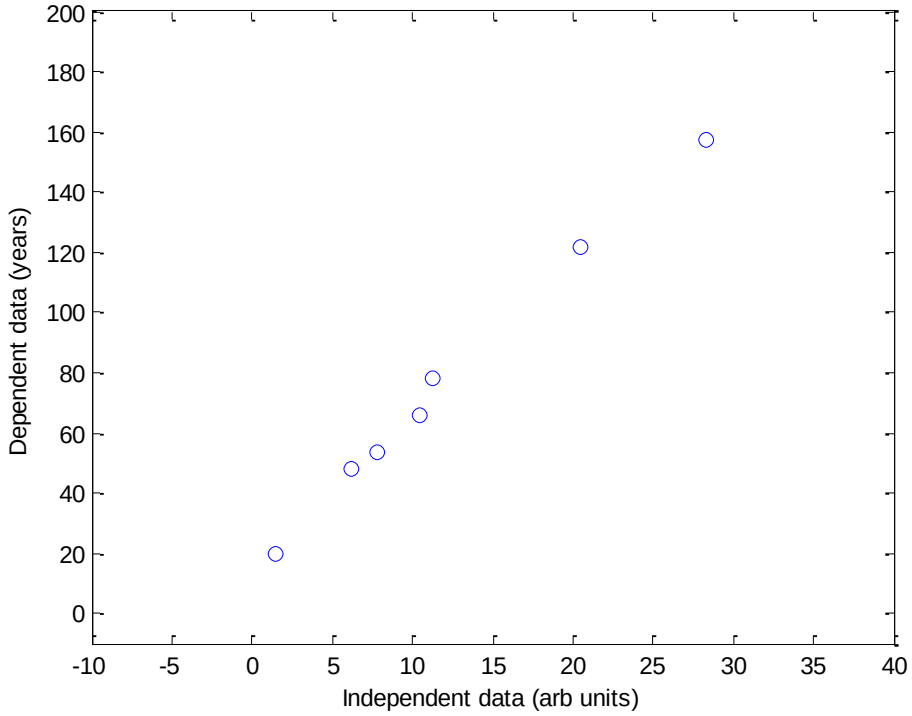
plot (TGdata(:,1),TGdata(:,2),'o')

This plots the data in column 1 in the x and column 2 in the y



You need to be able to label the axes and control the min and max of the axes. The text box below gives you the tools to do so. Enter them into the editor window and save it as a script. I did not choose the best values of axis limits in this plot. I recommend using variables, xmin, xmax, etc. in the axis command because you may be able to control the limits using programmed criteria.

```
plot (TGdata(:,1),TGdata(:,2),'o')
xlabel ('Independent data (arb units)')
ylabel ('Dependent data (years)')
xmin = -10;
xmax = 40;
ymin = -10;
ymax = 200;
axis ([xmin xmax ymin ymax])
```



Simple examples of using plot command for later

You will not be able to cut and paste these into your program because of the embedded formatting
There are many more options that you can access with guidance from Help Plot

- plot(xdata,ydata) – assuming both arrays are the same length, it will plot xdata on the x axis and ydata on the y axis and connect the points with a line with no symbols
- plot(xdata(i:j,k),ydata(i:j,m),'o') – plots rows i through j of column k in the xdata array and rows i through j in column m of the ydata array where the data points are plotted as circles and not connected by lines
- plot(x1(i:j),y1(i:j),'o',x2(m:n),y2(m:n),'+') – plots both x-y pairs of data on the same plot using circles for the first pair and + for the second pair. You can have multiple pairs of data as long as each pair has the same length.
- You can customize the colors, use different symbols, have symbols and lines of different types and a few more tweaks.
- The array names do not have to have x and y in them but the first array is the x coordinate and the second array is the y coordinate.

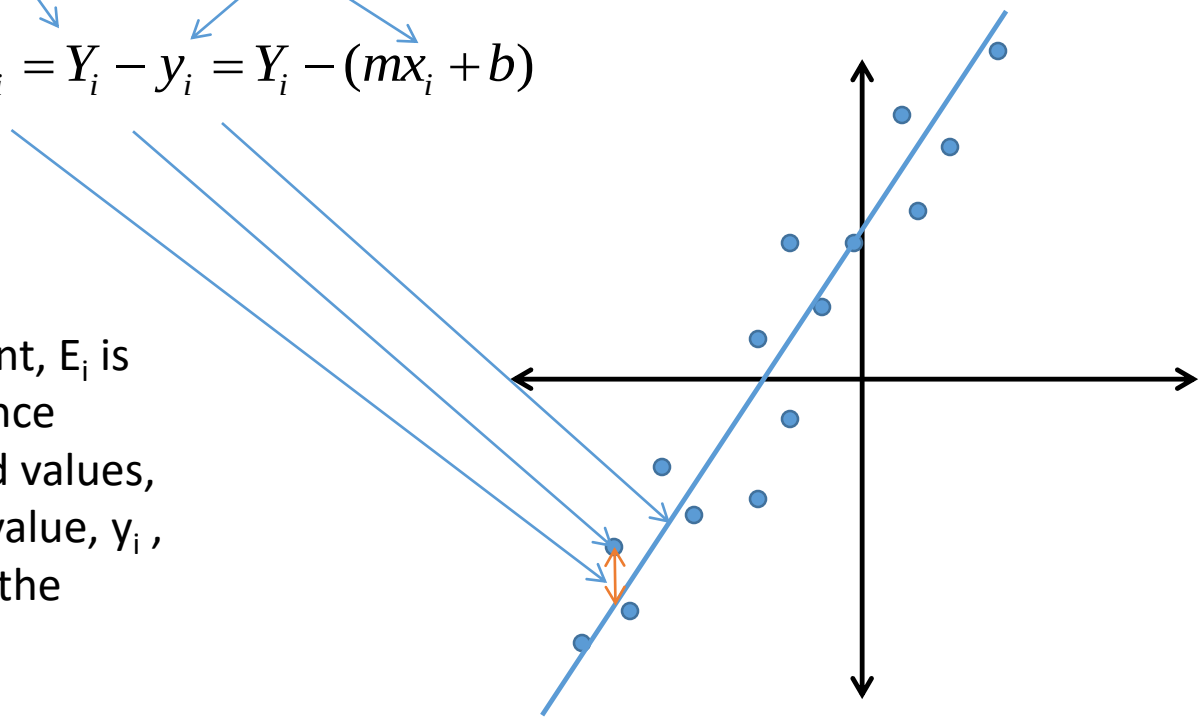
Plotting using hold on and hold off

- I personally don’t like programs that do this because I have the (possibly unsupported) opinion that it makes it harder to debug programs. The big problem is locating where the missing hold on or hold off is located.
- It does allow you to separate out the different inputs in the plot command.
- I will sigh and then get over it if I see hold on/off when I help you debug your programs.

Data fitting: Linear Regression

observed expected

$$E_i = Y_i - y_i = Y_i - (mx_i + b)$$



The error for each point, E_i is defined as the difference between the observed values, Y_i , and the expected value, y_i , which is described by the equation of the line.

We square the error and sum all of the individual squared errors to obtain the total squared error for N data points.

$$S = \sum_{i=1}^{N} \left(Y_i - mx_i - b \right)^2$$

$$S = \sum_{i=1}^{N} \left(Y^2{}_i - 2mx_i Y_i - 2bY_i + 2mbx_i + m^2 x_i^2 + b^2 \right)$$

$$S = \sum_{i=1}^{N} Y^2{}_i - 2m \sum_{i=1}^{N} x_i Y_i - 2b \sum_{i=1}^{N} Y_i + 2mb \sum_{i=1}^{N} x_i + m^2 \sum_{i=1}^{N} x_i^2 + Nb^2$$

If we take the derivative of the total squared error with respect to the coefficients of the best fit line, m and b, and set both derivatives to zero, we can find the values of m and b which give the minimum squared error. (When you take the derivatives, remember that the individual summations are just constants as far as the derivative is concerned.

$$\frac{dS}{db} = -2\sum_{i=1}^{N} Y_i + 2m\sum_{i=1}^{N} x_i + 2Nb = 0$$

$$\frac{dS}{dm} = -2\sum_{i=1}^{N} x_i Y_i + 2b\sum_{i=1}^{N} x_i + 2m\sum_{i=1}^{N} x_i^2 = 0$$

Since we have two equations and two unknowns, we can explicitly solve for m and b.

$$m = \frac{\sum_{i=1}^{N} x_i \sum_{i=1}^{N} Y_i - N\sum_{i=1}^{N} x_i Y_i}{\left(\left(\sum_{i=1}^{N} x_i\right)^2 - N\sum_{i=1}^{N}\left(x_i\right)^2\right)} \qquad b = \frac{\sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i Y_i - \sum_{i=1}^{N} Y_i \sum_{i=1}^{N}\left(x_i\right)^2}{\left(\left(\sum_{i=1}^{N} x_i\right)^2 - N\sum_{i=1}^{N}\left(x_i\right)^2\right)}$$

$$m = \frac{\sum_{i=1}^{N} x_i \sum_{i=1}^{N} Y_i - N \sum_{i=1}^{N} x_i Y_i}{\left(\left(\sum_{i=1}^{N} x_i \right)^2 - N \sum_{i=1}^{N} (x_i)^2 \right)} \qquad b = \frac{\sum_{i=1}^{N} x_i \sum_{i=1}^{N} x_i Y_i - \sum_{i=1}^{N} Y_i \sum_{i=1}^{N} (x_i)^2}{\left(\left(\sum_{i=1}^{N} x_i \right)^2 - N \sum_{i=1}^{N} (x_i)^2 \right)}$$

The above equations contain summations of observed x and y values. These equations uniquely define the best fit line. It is called the least squares line because it represents the line with the minimum squared error between the data and the line.

The method can be extended to solve for the best fit coefficients for a best fit curve that has linear coefficients for a set of basis functions, f_i.

$$f(x) = a_0 + a_1 f_1(x) + a_2 f_2(x) + a_3 f_3(x).....$$

The next slide shows the set of equations that are generated when we take the derivative of S with respect to each coefficient, a_i.

Here we have four equations and four unknowns so we can find the coefficients that best fit the data. The solution method is usually matrix inversion or Gauss-Jordan methods. The polyfit command in Matlab provides the coefficients for up to a ninth order polynomial.

$$S = \sum_{i=1}^{N} \left(F(x_i) - \left(a_0 + a_1 f_1(x_i) + a_2 f_2(x_i) + a_3 f_3(x_i) \right) \right)^2$$

$$\frac{dS}{da_0} = \sum_{i=1}^{N} -2 \left(F(x_i) - \left(a_0 + a_1 f_1(x_i) + a_2 f_2(x_i) + a_3 f_3(x_i) \right) \right) = 0$$

$$\frac{dS}{da_1} = \sum_{i=1}^{N} -2 f_1(x_i) \left(F(x_i) - \left(a_0 + a_1 f_1(x_i) + a_2 f_2(x_i) + a_3 f_3(x_i) \right) \right) = 0$$

$$\frac{dS}{da_2} = \sum_{i=1}^{N} -2 f_2(x_i) \left(F(x_i) - \left(a_0 + a_1 f_1(x_i) + a_2 f_2(x_i) + a_3 f_3(x_i) \right) \right) = 0$$

$$\frac{dS}{da_3} = \sum_{i=1}^{N} -2 f_3(x_i) \left(F(x_i) - \left(a_0 + a_1 f_1(x_i) + a_2 f_2(x_i) + a_3 f_3(x_i) \right) \right) = 0$$

polyfit command

The command polyfit is a least squares fit function for a polynomial. It has the following structure:

p=polyfit(xdata(rowstart:rowend,columnx),ydata(rowstart:rowend,columny),fitorder);

It returns the fit results into the vector, p, that has fitorder+1 rows. If the fitorder = 1, the first value, p(1) will be the slope and the second value, p(2) will be the y intercept.

If fitorder > 1, then it returns the fit results for a polynomial of order, fitorder. For example, if fitorder = 3, the polynomial is:

$$y = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

$$p(1) = a_3;\ p(2) = a_2;\ p(3) = a_1;\ p(4) = a_0$$

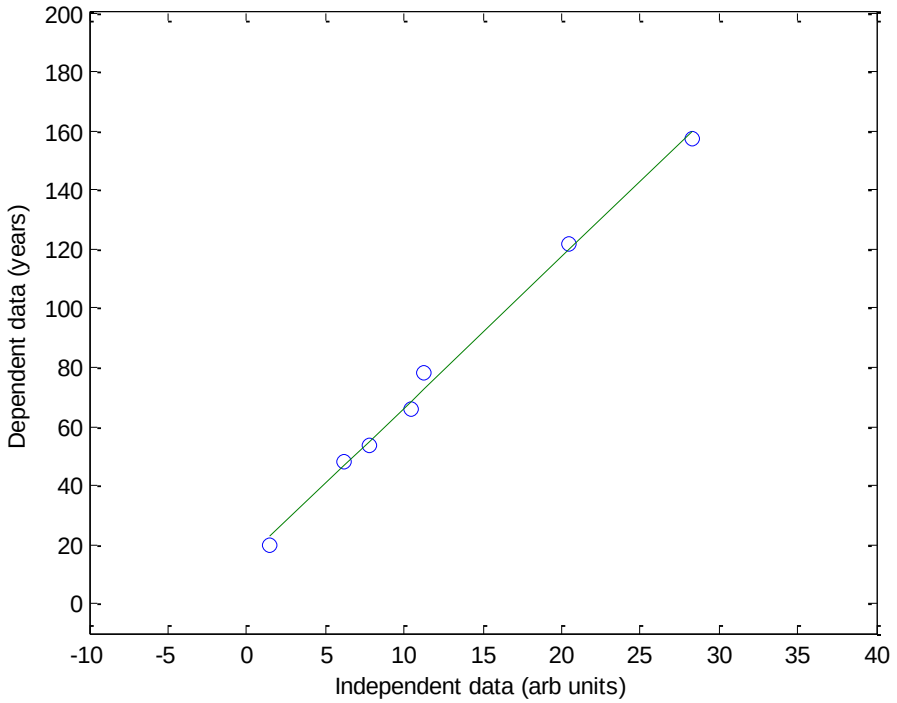
The command can fit up to a 9th order polynomial.

The number of rows must be the same for both vectors (e.g. rowendx-rowstartx=rowendy-rowstarty). If you do not specify the rows, it fits all of the data in the two vectors. You do not always have 2D vectors so the columnx and columny are not always necessary. You could be fitting two columns of the same vector if it is 2D. This is what we do in the following example.

The data that we provided exhibits a linear dependence. We want you to plot a best fit line using the polyfit command to generate the first order fit constants. Create a new vector that has the best fit values for each x value using the constants from polyfit and the x data from TGdata. Using the x data column restricts the line to the region you used for the best fit line. You can use the linespec option to change the color of the lines, the marker types and sizes, and the style of the lines.

It is a matter of personal taste, but, using a single plot command to plot multiple data pairs is efficient in that you do not need to use hold on/hold off. The downside is that it creates long command lines. I find debugging programs with extensive hold on/hold off statements to be frustrating – I will get over it.

```
p=polyfit(TGdata(:,1),TGdata(:,2),1);
bestfit=p(1)*TGdata(:,1)+ p(2);
plot (TGdata(:,1),TGdata(:,2),'o',TGdata(:,1),bestfit)
xlabel ('Independent data (arb units)')
ylabel ('Dependent data (years)')
xmin = -10;
xmax = 40;
ymin = -10;
ymax = 200;
axis ([xmin xmax ymin ymax])
```



Linear regression for linear equation with zero intercept

$$y = a_o x$$

$$S = \sum_{i=1}^{N} \left(Y_i - a_o x_i \right)^2$$

$$S = \sum_{i=1}^{N} \left(Y^2{}_i - 2a_o x_i Y_i - a_o{}^2 x_i^2 \right)$$

$$S = \sum_{i=1}^{N} Y^2{}_i - 2a_o \sum_{i=1}^{N} x_i Y_i + a_o{}^2 \sum_{i=1}^{N} x_i^2$$

$$\frac{dS}{da_o} = -2\sum_{i=1}^{N} x_i Y_i + 2a_o \sum_{i=1}^{N} x_i^2 = 0$$

$$a_o = \frac{\sum_{i=1}^{N} x_i^2}{\sum_{i=1}^{N} x_i Y_i}$$

```
x=0:1:10;
noise=rand()/10;
for i=1:length(x)
    y(i)=5*x(i)*(1.1-rand()/10);
end
sumxy=0;
sumx2=0;
for i=1:length(x)
    sumxy=x(i)*y(i)+sumxy;
    sumx2=x(i)^2+sumx2;
end
% sumxy=sum(x.*y);
% sumx2=sum(x.^2);
co=sumxy/sumx2
yfit=co*x;
p=polyfit(x,y,1)
yfitpoly=p(1)*x+p(2);
plot(x,y,'o',x,yfit,x,yfitpoly,'--')
```

You must use this in Lab 2 when you fit ln(Γ) vs. t to determine τ for the equation:

$$\ln \Gamma = -t / \tau$$

Purpose of the following slides

- Show how to fit power law and exponential equations.
- One can linearize these equations by taking the log of both sides of the equation. This is shown in the following three slides.
- Matlab and Excel also have tools to fit exponentials and power laws. The fourth slide after this shows how to fit an exponential. I want you to modify this code to fit a power law equation for Deliverable 2.

Thermistor constant determination by data fitting

A thermistor is a resistance-based temperature sensor. The material in a thermistor is a semiconducting oxide and the number of free carriers (electrons or holes) is a strong function of temperature up to some limiting value when the thermal energy is greater than the band gap. The resistance of a thermistor is an exponential function of temperature over a certain temperature range. (The resistance of an RTD, which we will use later, is a linear function of temperature.)

The standard form of the equation that describes the thermistor resistance, R, is given on the left where T is the absolute temperature, R_o is the room temperature resistance, β is a material constant, and T_o is room temperature. The right hand equations show how to relate it to a simple exponential function.

$$R = R_o \exp\left[\beta\left(\frac{1}{T} - \frac{1}{T_o}\right)\right]$$

$$T_o = 298.15\,K$$

$$R = R_o \exp\left[\beta\left(\frac{1}{T} - \frac{1}{T_o}\right)\right] = R_o \exp\left[\frac{-\beta}{T_o}\right]\exp\left[\frac{\beta}{T}\right]$$

$$R = A\exp[\beta x]$$

$$where\ A = R_o \exp\left[\frac{-\beta}{T_o}\right]\ \ and\ \ x = \frac{1}{T}$$

Sometimes, you do not have the exponential fit option. You can use a linear fit if you transform the equation by taking the log or ln of both sides.

The program on the next page shows you how to input data from an Excel file, plot a semilog plot, and fit a transformed exponential function using polyfit.

The equation for the resistance of a thermistor as a function of temperature is given to the right. Taking the natural log of both sides of the equation gives a linear equation that you can fit with polyfit. You must create new variables for y and x that are ln(R) and 1/T, respectively. The first term from polyfit will be β. You must use the second term from polyfit to solve for R_o with knowledge of β and T_o.

$$R = R_o \exp\left[\beta\left(\frac{1}{T} - \frac{1}{T_o}\right)\right]$$

$$\ln R = \ln R_o + \beta\left(\frac{1}{T} - \frac{1}{T_o}\right)$$

$$\ln R = \left(\ln R_o - \frac{\beta}{T_o}\right) + \beta\left(\frac{1}{T}\right)$$

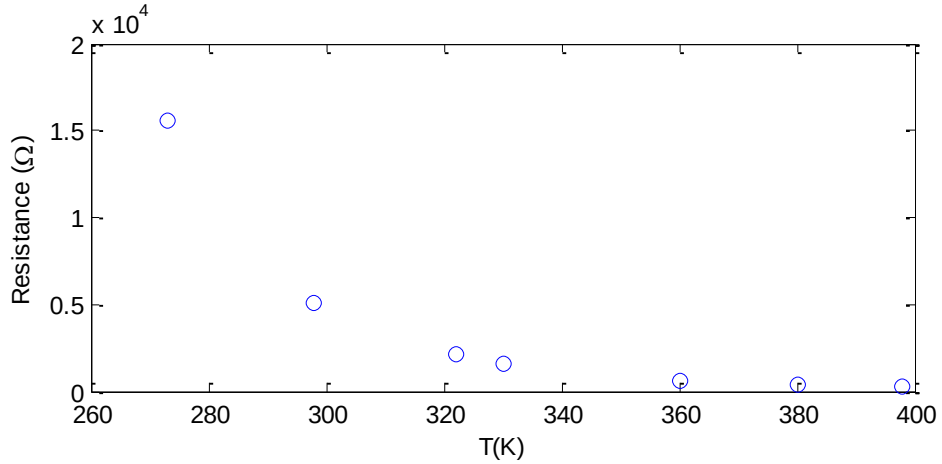
$$\equiv y' = a_o + a_1 x'$$

$$\text{where } \; y' = \ln R \; \text{ and } \; x' = \frac{1}{T}$$

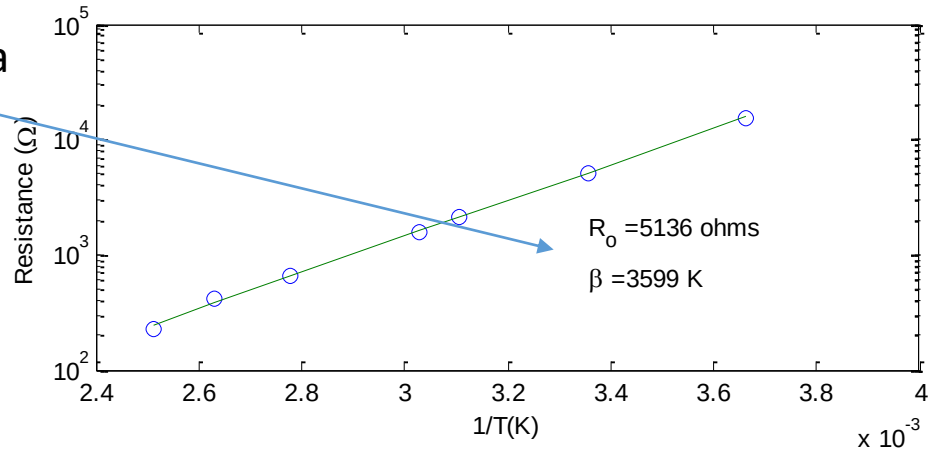
$$\text{and } \; a_o = \ln R_o - \frac{\beta}{T_o} \; \text{ and } \; a_1 = \beta$$

Read in thermistor.xlsx, plot R vs. T and R vs. 1/T on a semilog plot. Transform the data to a linear equation using the method on the previous page. Use polyfit to determine β and R_O on the transformed data. Then, create a fit line using the x data, plot the fit, and list the fit constants on the plot. The example shows how to use num2str to convert a number to a string variable and strcat to concatenate (combine) string variables. It also shows how to create a subscript and Greek letters. The _ gives a subscript and ^ gives a superscript. The subplot command allows two stacked graphs.

```
a=xlsread('Thermistor_Lab1.xlsx'); %Loads the spreadsheet into an multi-
% column array, first column T (K) second R (ohms)
subplot(2,1,1)
plot(a(:,1),a(:,2),'o')
xlabel ('T(K)')
ylabel ('Resistance (\Omega)')
% R = Ro*exp(beta*(1/T-1/To))
T=a(:,1);
R=a(:,2);
Tinv=1./T;% Inverse of absolute temperature
To=298.15; %room temp reference for thermistor
RL=log(R);%natural log of R
subplot (2,1,2)
p=polyfit(Tinv,RL,1);
beta=p(1);
Ro=exp(p(2)+beta/To);
Rfit=Ro*exp(beta*(1./T-1/To));
subplot(2,1,2)
semilogy(Tinv,R,'o',Tinv,Rfit)
xlabel ('1/T(K)')
ylabel ('Resistance (\Omega)')
B=num2str(beta,4); %converts number to string (text) with 4 significant
digits
B=strcat('\beta',' = ',B,' K'); %combines multiple string variables into
% single variable
RO=num2str(Ro,4);
RO=strcat('R_o = ',RO,' ohms');
text(1/T(2),R(5),B) %Puts text at x,y location on plot.
text(1/T(2),R(4),RO)
```



These are last year's results. This year's data has been changed.



Same data and plotting format but uses exp1 fit option instead of polyfit. Reads in data using slightly different strategy. The a:a reads in all of the data in column a. Loading it into a specific variable saves memory (which is not always a problem but it is good practice to conserve memory). You can specify the rows you want to load if desired.

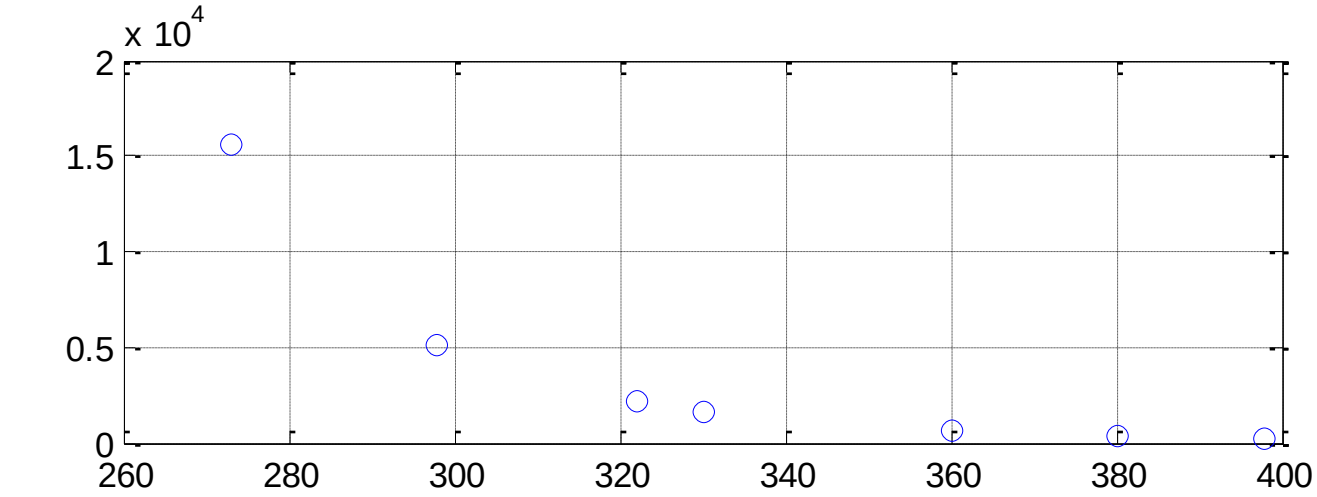
```
close all;clear all;
T=xlsread('Thermistor_Lab1.xlsx','a:a'); %reads in all numeric values in
second column
% It will ignore text headers
R=xlsread('Thermistor_Lab1.xlsx','b:b'); %reads in all numeric values in
first column
subplot(2,1,1) %plot two rows of plots, one column, top graph
plot (T,R,'o')
grid on
subplot(2,1,2)%plot two rows of plots, one column, bottom graph
To=298.15; %constant for thermistor
Tinv=1./T ;%inverse of absolute temperature
p=fit(Tinv,R,'exp1'); %Fits data to exponential fn, inputs fit data into
array p
p2=coeffvalues(p); %extracts values from p into p2
% p2(2)is beta
% p2(1) is A
Rfit=p2(1)*exp(p2(2)*Tinv);%create vector of fit values for each x value
beta=p2(2);
Ro=p2(1)*exp(beta/To);
semilogy(Tinv,R,'o',Tinv,Rfit)
xlabel ('1/T(K)')
ylabel ('Resistance (ohms)')
B=num2str(beta,4); %num2str converts a number to a string value
%number after comma specifies number of significant figures
B=strcat('\beta',' = ',B,' K'); %strcat concatenates string values
RO=num2str(Ro,4);
RO=strcat('R_o = ',RO,' ohms');%underscore gives subscript
text(1/T(2),R(5),B) %text (xlocation, ylocation, textdata)
text(1/T(2),R(4),RO)
```

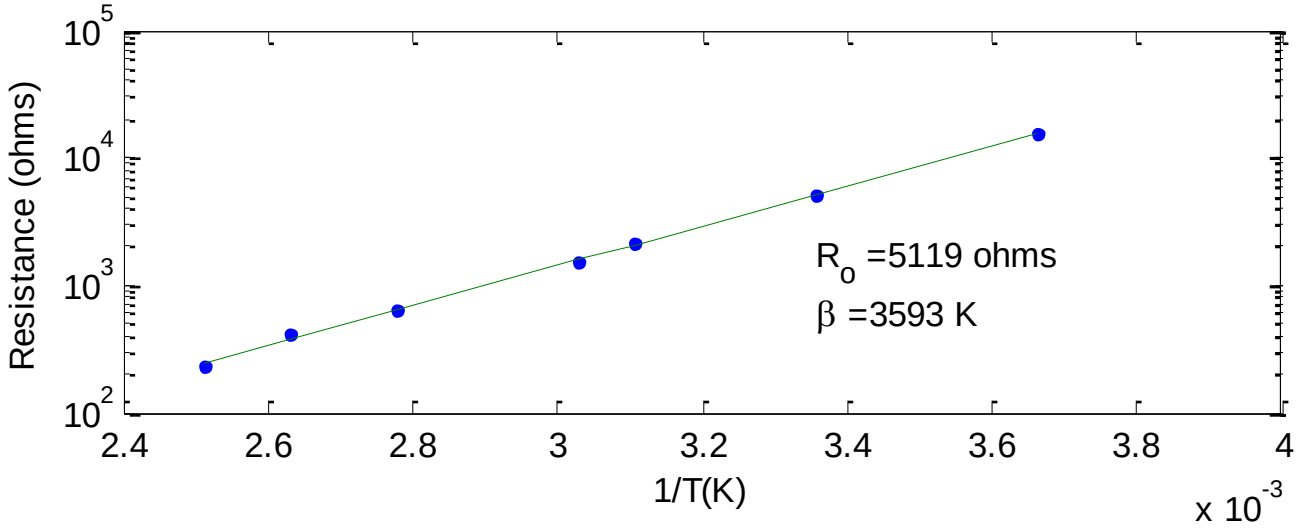
While this option to the xlsread is convenient, the data loads much more slowly. I do not recommend this for large data files (like the ones in Lab 2a).

Using the command

```
grid on
```

after the plot command will plot grid lines as shown in the upper plot. Some people like them, some don't.





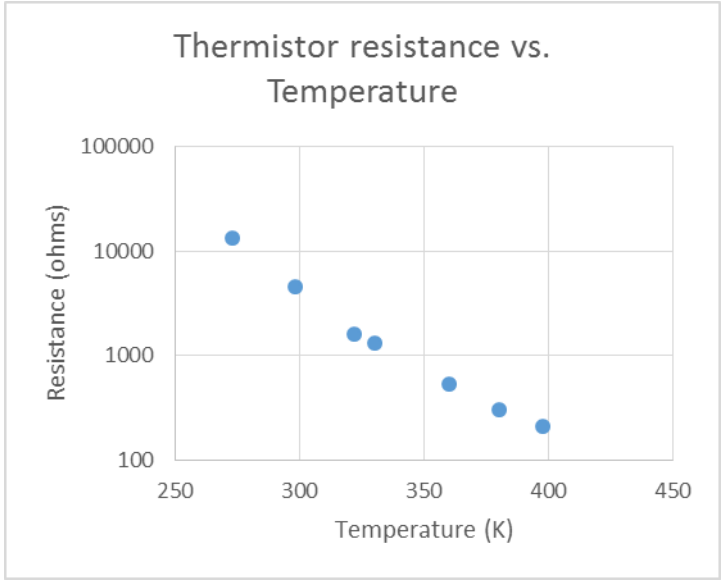
Deliverable 4

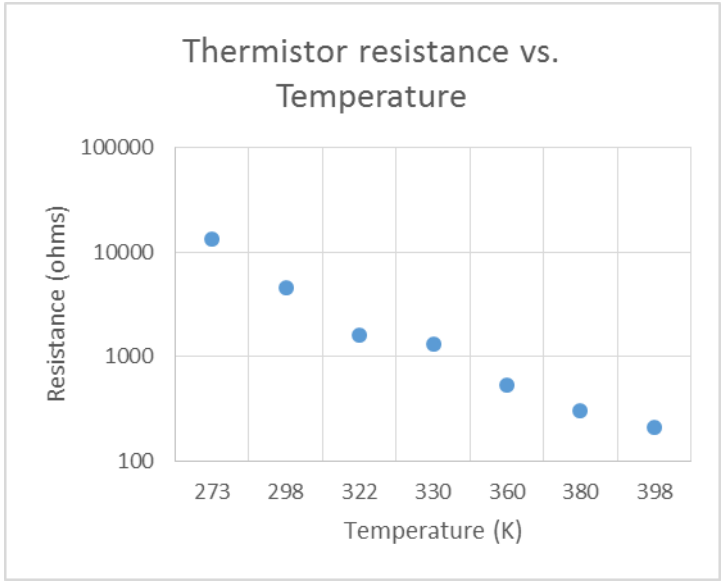
- The following slides provide some guidance for how to complete Deliverable 4 using Excel.

Excel is a modern and heavily used tool. You may not have access to Matlab or it may be too inconvenient, so it is a good idea to learn how to plot things in Excel. The first thing you need to know is that you must use a Scatter Plot and **NOT** the Line Plot. The Line Plot will plot the data evenly spaced in x regardless of the value of x.

The top plot is a Scatter Plot and is the correct plot. The bottom plot is a Line Plot using the same input data and it should be clear that the x spacing is not the same (and not correct). If your data is evenly spaced, you will not be able to tell the difference.

PLEASE DO NOT MAKE THIS MISTAKE IN LAB OR ANYWHERE ELSE.



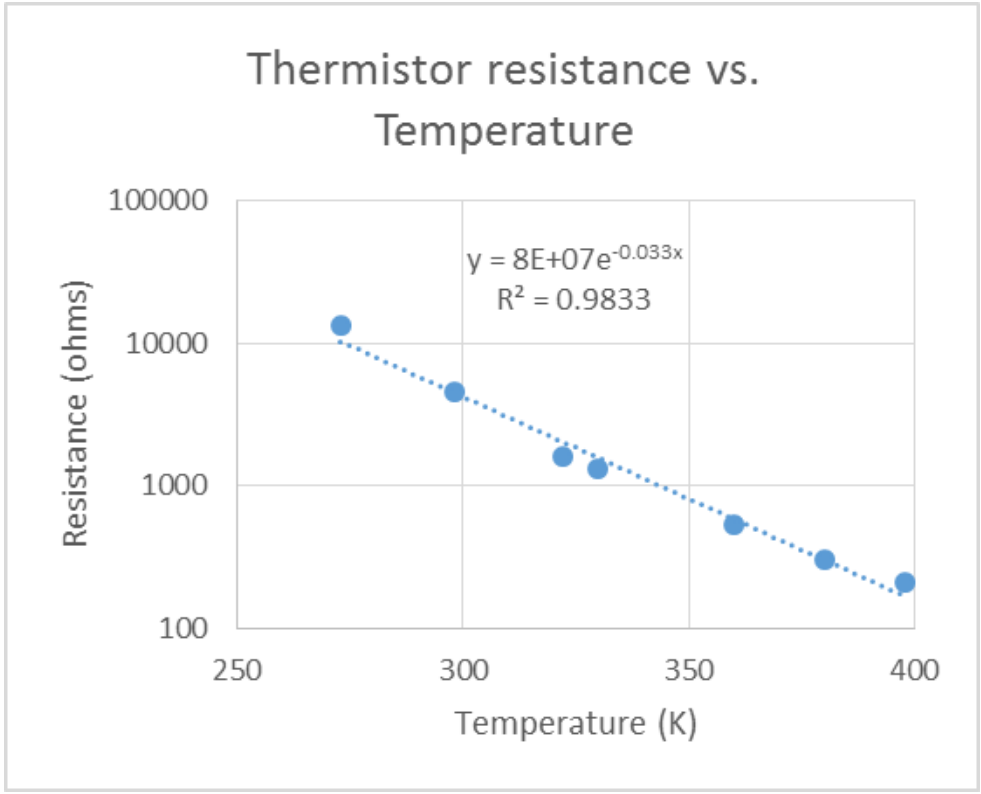


Excel has fairly powerful fitting options. If you right click on the data and choose trendline, the right hand side of the screen gives you choices for different fit types. I chose Exponential (which is incorrect for R vs. T, but it gives a reasonable looking fit). Make sure you choose the Display Equation on Chart.

Note that this a log-linear plot. Right click on the axis and select format axis. You can choose logarithmic axis at this point. The control of the labeling, limits, and tick spacing is limited.

You add labels when you are editing the plot by selecting Design, Add Chart Element.

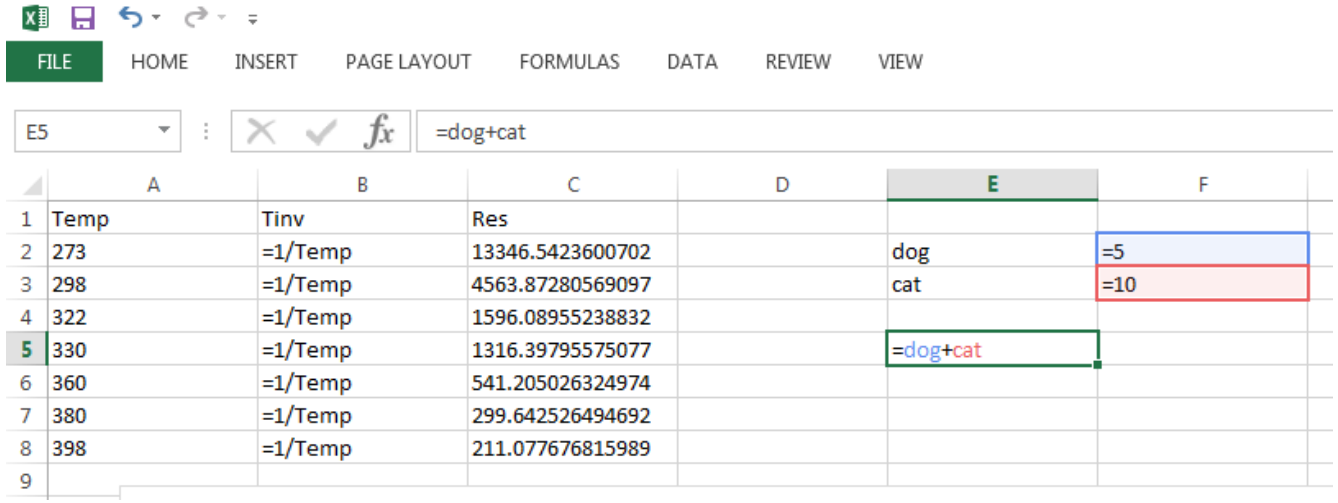
When you have long data columns and make a plot, it often puts it somewhere in the column that is not easy to find. For these plots, it makes sense to move the chart to its own page. Right click on the plot, select Move Chart, and the put it in a new sheet.



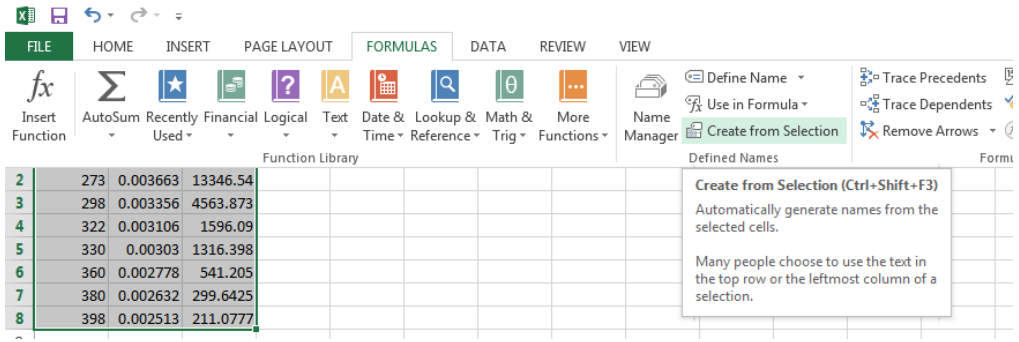
Using Formula in Excel makes debugging the equations much easier.

Create a column or group of columns and give the top row a name that you might use easily in a formula. Highlight the columns, Choose the Formula tab and then Create from Selection. (See figure on upper right). It will ask you where the label is. It is also useful to have a table of constants (like dog and cat). Again, highlight the data and the label and choose Create from Selection. The figure on the lower right shows the dialog box that will pop up.

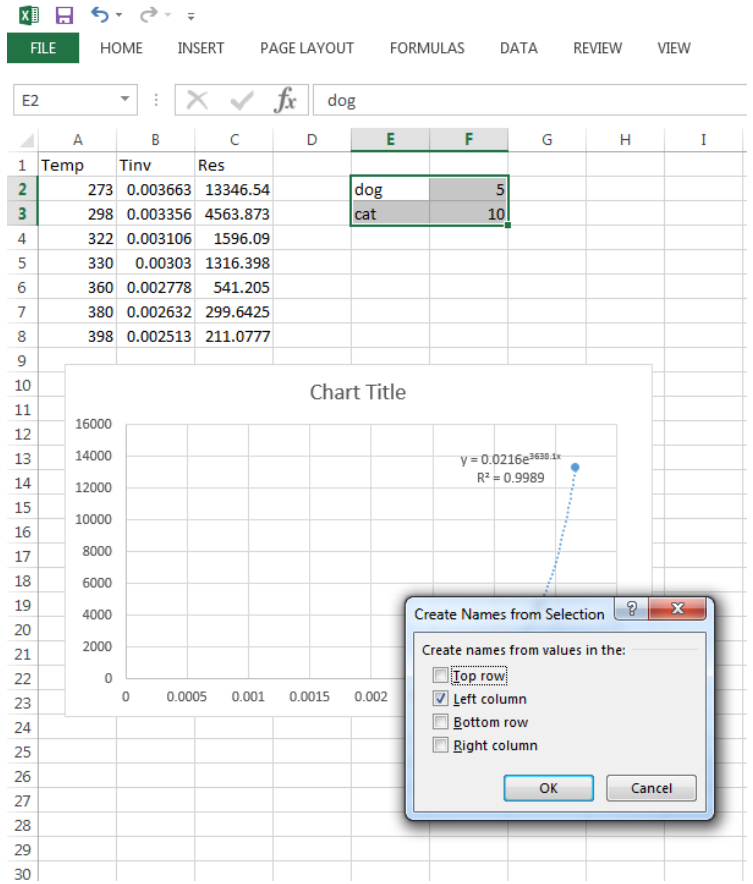
If you choose Formulas and then Show Formulas, your equations will be visible as shown below.



	A	B	C	D	E	F
1	Temp	Tinv	Res			
2	273	=1/Temp	13346.5423600702		dog	=5
3	298	=1/Temp	4563.87280569097		cat	=10
4	322	=1/Temp	1596.08955238832			
5	330	=1/Temp	1316.39795575077		=dog+cat	
6	360	=1/Temp	541.205026324974			
7	380	=1/Temp	299.642526494692			
8	398	=1/Temp	211.077676815989			
9						



2	273	0.003663	13346.54
3	298	0.003356	4563.873
4	322	0.003106	1596.09
5	330	0.00303	1316.398
6	360	0.002778	541.205
7	380	0.002632	299.6425
8	398	0.002513	211.0777



	A	B	C	D	E	F
1	Temp	Tinv	Res			
2	273	0.003663	13346.54		dog	5
3	298	0.003356	4563.873		cat	10
4	322	0.003106	1596.09			
5	330	0.00303	1316.398			
6	360	0.002778	541.205			
7	380	0.002632	299.6425			
8	398	0.002513	211.0777			

Working with time series data – Deliverable 3

- The following slides will give you tools for Deliverable 3.
- We expect you to use these tools in Lab 2a and subsequent labs.
- For regularly spaced data, time and array index are equivalent. The array index is an integer that can be used in for/if loops. We want you to learn to use the array index for selecting which rows to include in data processing of time series data.

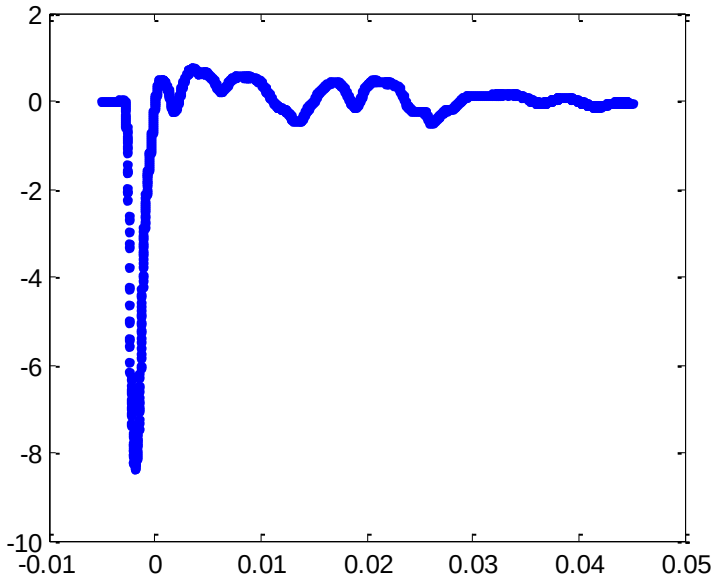
Working with time series data

- You will often record data from multiple inputs at specified time intervals. This is stored in a multi column array where the data in a given row is taken at the same time. The first column is often time.
- Many of the situations you deal with will record data before the time you would like to associate as t = 0. This requires you to identify the row that corresponds to the beginning of the event you want to record output as a function of time.
- Once you identify the row corresponding to t = 0, it is more convenient to use the array index as time in your “for” loops instead of the actual time. This is because “for” loops require you to use integer values. Fortunately, time and array index in time series data obtained by recording devices are usually (but not always) linearly proportional to one another.

Importdata command for .lvm files, baseline extraction, determination of event start (trigger).

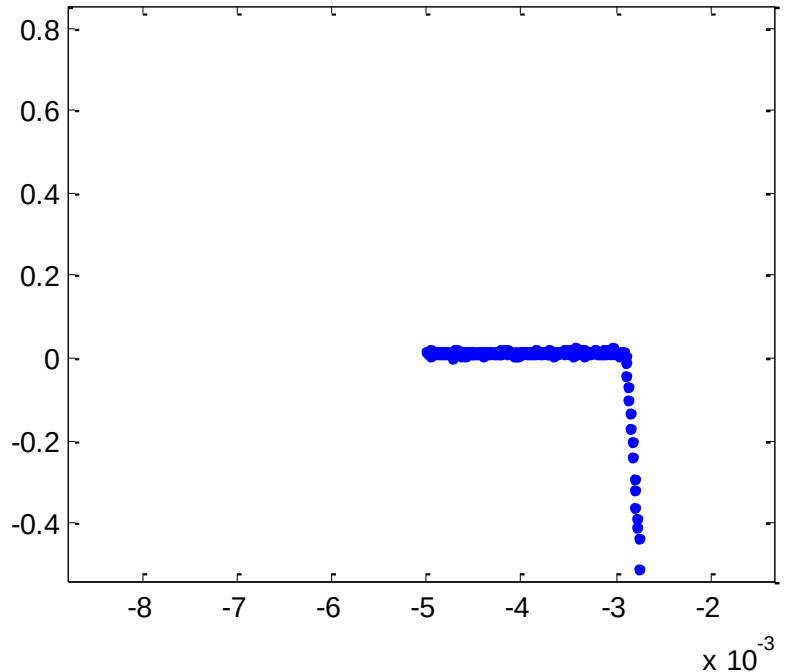
The NI workstations produce .lvm files that have a number of text lines at the beginning that cannot be ignored. The importdata command allows you to import all of the data in the file into a structured variable that has text components and data components. One must independently determine the number of text lines prior to importing the data. We supply a file hammer.lvm that has 29 header lines. Type in the code below to obtain the plot to the right.

```
close all; clear all;
nheaderlines=29;
hammer = importdata('hammer.lvm','\t',nheaderlines);
%nheaderlines is the number of text lines in the file before you get to the
%data.  The \t specification indicates that it is a tab delimited file
%The importdata function will set variable as a Matlab structure with two
%subsequent handles which yield the headerlines and a matrix of data
%denoted by hammer.textdata and hammer.data respectively.
%Access the by variablename.text or variablenmae.data
%You must inspect the file with Excel to determine the number of
%header lines

time = hammer.data(:,1);
voltage = hammer.data(:,2);
plot(time,voltage,'.')
```

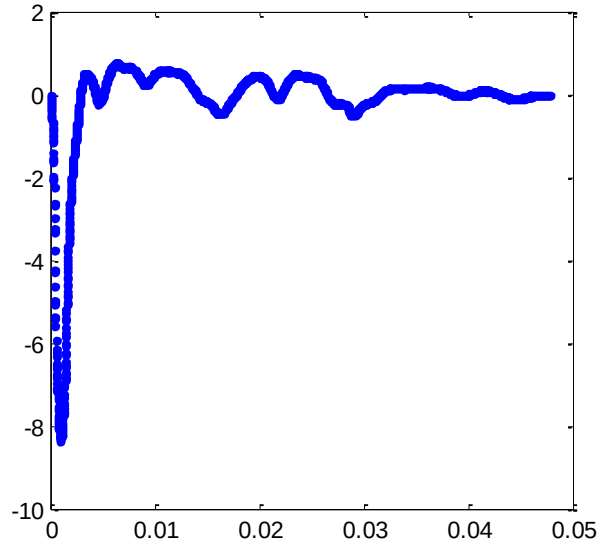


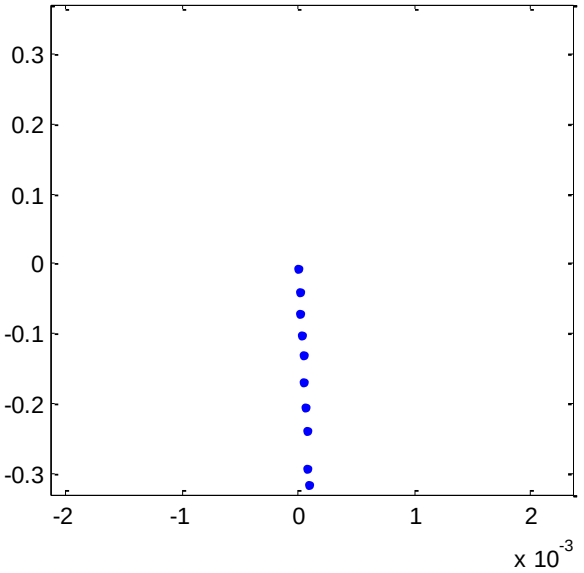
Note that the event does not start at t = 0. We identify the initial time of the event by analyzing the baseline data to determine the average value and the standard deviation. One must decide how far to analyze the baseline by inspection. Then, step through the data until the signal is different from the average value by some multiple of the standard deviations. Five standard deviations is usually enough.

Use the zoom button on the plot menu to zoom in on the baseline. On this plot, we can see that the event starts at about -0.003 seconds. To be safe, determine the average and standard deviation of the data for times less than -0.0035 seconds. Then, probe the voltage array to find the row (array index) where the signal is outside of the five standard deviation range.



```
close all; clear all;
nheaderlines=29;
hammer = importdata('hammer.lvm','\t',nheaderlines);

time = hammer.data(:,1);
voltage = hammer.data(:,2);

%find the row corresponding to the first time the time is >-0.0035
for i=1:length(time)
    if time(i)>-0.0035
        basetime=i;
        break
    end
end
baseline=mean(voltage(1:basetime)); %average of data in baseline region
basedev=std(voltage(1:basetime)); %standard deviation of baseline region
threshold=5*basedev; %threshold to define start of event.

for i=1:length(time)
    if (abs(voltage(i)-baseline)>threshold)
        starttime=i; %row corresponding to start of event
        break
    end
end
%create new variables that start from t = 0 and only contain
%event data
newtime=time(starttime:length(time))-time(starttime);
newvolt=voltage(starttime:length(time));

plot(newtime,newvolt,'.')
```

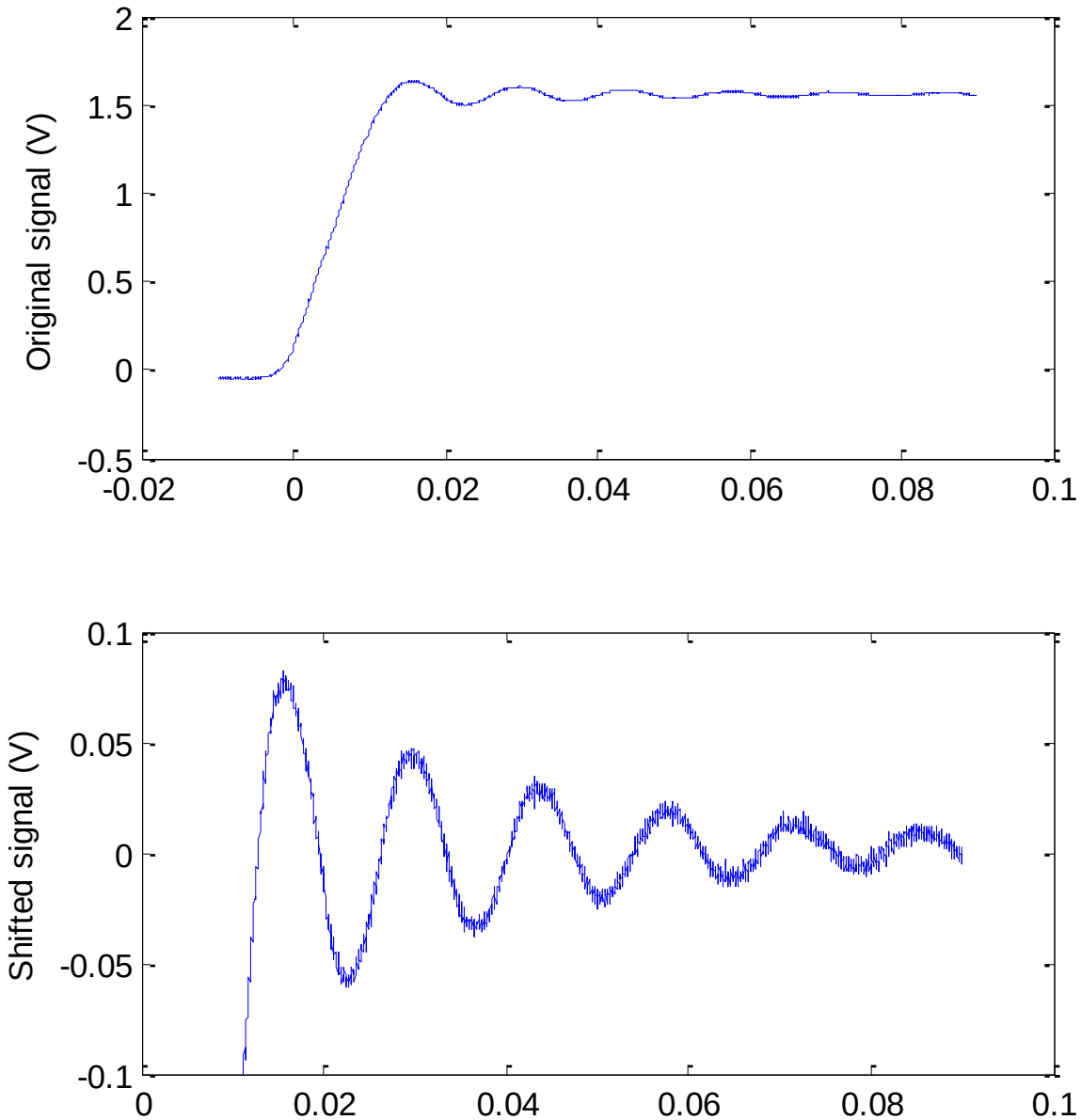




This example creates new variables, newtime, and newvolt. It may be just as easy to use the time variable and specify the relevant rows. It makes the program a bit cleaner.

The next task we will address is data smoothing and peak detection. We will want you to determine the magnitude and time corresponding to multiple peaks in the data obtained in Lab 3 and Lab 6. If the data were perfectly smooth, we could use the max function and a suitable moving window to identify the peaks. However, experimental data always has some noise and one must smooth the data to identify peaks and valleys with some reliability.

The plot to the right is the output of a pressure sensor subjected to a step change in pressure. The upper plot shows the full original signal. The lower plot shows a zoomed in area near the peaks we want to identify where I subtracted off the final value so that the data would oscillate around zero. The noise in the data is apparent.



We can use the convolution function, conv, to smooth the data. It is basically a moving average. The 3 minute video at http://blogs.mathworks.com/videos/2012/04/17/using-convolution-to-smooth-data-with-a-moving-average-in-matlab/ explains how this works. The code sample I used can be found at http://www.mathworks.com/help/matlab/learn_matlab/data-analysis.html#zmw57dd0e4321 .

We empirically determine how many points we need to average to be able to detect the signal peaks instead of the noise. sig is the original vector and sig2 is the smoothed data. I experimented with the data and found a span = 41 worked well for this data. If the span is too large, it will decrease the amplitude. There are several other ways to do this (e.g. regression, low-pass). One result of this tool is that the beginning and ending of the file will be truncated. This means you must adjust loop ranges that probe this data. For a span of 41, the first 20 data points and the last 20 data points will not be useful because they are based on incomplete smoothing.

```
span = 41; % Size of the averaging window
window = ones(span,1)/span;
sig2 = conv(sig,window,'same');
```

Peak detection

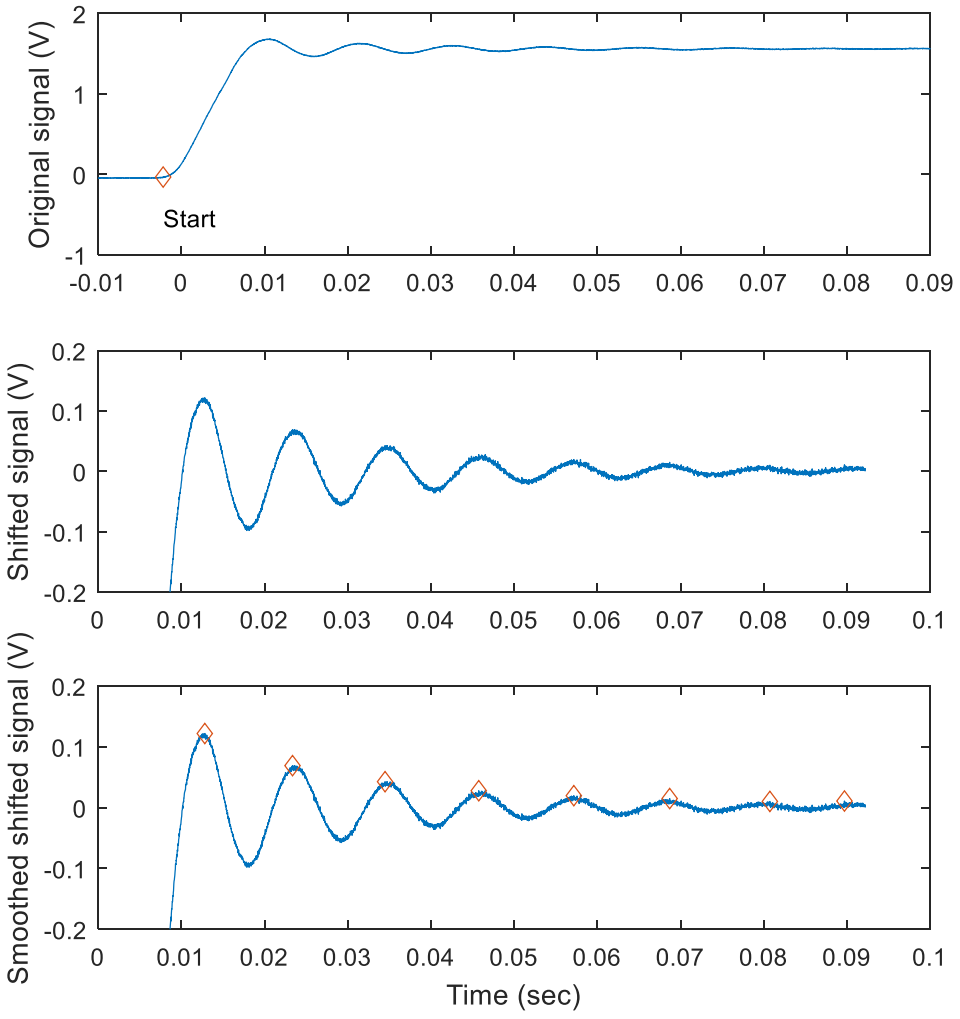
We used to use peakdet function to find the peaks and valleys. I leave last year's description of peakdet below. This year, I am suggesting but not requiring you to use **peakfinder**. It seems to be more robust. https://www.mathworks.com/matlabcentral/fileexchange/25500-peakfinder-x0--sel--thresh--extrema--includeendpoints--interpolate- The description at the beginning of the file is quite good and you should be able to run the code without having to perform the smoothing or shifting it to oscillate around zero. However, you will have to strip off the final value when you use this to determine the damping ratio in Lab 3 and Lab 6.

Peakdet description.
I subtracted the end value off of all of the data so that it oscillated around zero. The function must be in the directory of the main program and was downloaded from http://www.mathworks.com/matlabcentral/fileexchange/47264-peakdet . It has a typo in the last argument where they misspell threshold as theshold. I left it in to preserve the original function. The function call is:

```
[pks,dep,pidx,didx]=peakdet(sig2,th,'theshold');
```

The magnitude of the peaks and depressions are stored in the vectors pks and dep. The vectors pidx and pidy contain the indices of the location of the peaks and valleys in the sig2 array. You pass the signal you want to process, sig2, the fraction of the signal that defines a peak, th, and the detection type (threshold or zero crossing). One must experiment with the magnitude of th and the span to detect only the peaks and valleys in the relevant signal and not the noise. Please look at the function and understand what it does.

The following script file produces the plot below (with the exception you must adjust the th variable and uncomment the axis commands. The peaks indicated by symbols. (The last slide has a link with a list of the symbol codes.)




```
close all; clear all;
nheaderlines=30;
pressure = importdata('S1_1.lvm','\t',nheaderlines);
time = pressure.data(:,1);
P = pressure.data(:,2);
%find the row corresponding to the first time the time is >-0.0035
for i=1:length(time)
    if time(i)>-0.0035
        basetime=i;
        break
    end
end
baseline=mean(P(1:basetime)); %average of data in baseline region
basedev=std(P(1:basetime)); %standard deviation of baseline region
threshold=5*basedev; % 5*sigma threshold to define start of event.
for i=1:length(time)
    if (abs(P(i)-baseline)>threshold)
        starttime=i; %row corresponding to start of event
        break
    end
end
%create new variables that start from t = 0 and only contain event data
%It is not necessary to do this but it sometimes makes it easier.
newtime=time(starttime:end)-time(starttime);
Pnew=P(starttime:end)-P(end);
th=0.15; %this is something you need to adjust, the value here is incorrect
[pidx,pks]=peakfinder(Pnew,th);
subplot(3,1,1)
plot(time,P,time(starttime),P(starttime),'d')
ylabel ('Original signal (V)')
text(time(starttime),P(starttime)-.5,'Start')
subplot (3,1,2)
plot(newtime,Pnew)
% axis([0 .1 -.2 .2])
ylabel ('Shifted signal (V)')
subplot (3,1,3)
plot(newtime,Pnew,newtime(pidx),pks,'d')
% axis([0 .1 -.2 .2])
ylabel ('Smoothed shifted signal (V)')
xlabel ('Time (sec)')
```

Modified code using the peakfinder function

```
close all; clear all;
nheaderlines=30;
pressure = importdata('S1_1.lvm','\t',nheaderlines);
time = pressure.data(:,1);
P = pressure.data(:,2);
%find the row corresponding to the first time the time is >-0.0035
for i=1:length(time)
    if time(i)>-0.0035
        basetime=i;
        break
    end
end
baseline=mean(P(1:basetime)); %average of data in baseline region
basedev=std(P(1:basetime)); %standard deviation of baseline region
threshold=5*basedev; % 5*sigma threshold to define start of event.
for i=1:length(time)
    if (abs(P(i)-baseline)>threshold)
        starttime=i; %row corresponding to start of event
        break
    end
end
%create new variables that start from t = 0 and only contain event data
%It is not necessary to do this but it sometimes makes it easier.
newtime=time(starttime:end)-time(starttime);
Pnew=P(starttime:end)-P(end);
th=0.015; %this is something you need to adjust, the value here is incorrect
[pidx,pks]=peakfinder(Pnew,th);
subplot(3,1,1)
plot(time,P,time(starttime),P(starttime),'d')
ylabel ('Original signal (V)')
text(time(starttime),P(starttime)-.5,'Start')
subplot (3,1,2)
plot(newtime,Pnew)
axis([0 .1 -.2 .2])
ylabel ('Shifted signal (V)')
subplot (3,1,3)
plot(newtime,Pnew,newtime(pidx),pks,'d')
axis([0 .1 -.2 .2])
ylabel ('Smoothed shifted signal (V)')
xlabel ('Time (sec)')
```

Creating tables in Matlab

The following code will create adequate tables in Matlab.

```
tablehelper = newtime(pidx); %creates array for time values at array locations defined by pidx
f = figure('Position',[200 200 400 150]); %positions figure, could be better optimized
%pks(i) contains the actual values of the pks, pidx contains the array
%index of the peak location
dat = [tablehelper(1) pks(1) ; tablehelper(2) pks(2) ; tablehelper(3) pks(3); tablehelper(4)
pks(4); tablehelper(5) pks(5); tablehelper(6) pks(6)];
cnames = {'Time(s)','Amplitude(V)'}; %column names
rnames = {'First Peak','Second Peak','Third Peak','Fourth Peak','Fifth Peak','Sixth Peak'}; %Row
names
t = uitable('Parent',f,'Data',dat,'ColumnName',cnames,'RowName',rnames,'Position',[20 5 275 135]);
```

Useful links

- https://www.mathworks.com/help/matlab/creating_plots/greek-letters-and-special-characters-in-graph-text.html - character codes for Greek symbols at end of link.
- https://www.mathworks.com/help/matlab/ref/linespec.html - character codes for plot symbols, line style codes, and color codes.