Lab 3:
Introduction to MATLAB

3.1 Introduction

During this lab, you will learn some of the basic skills required to load, process, and display numerical information in MATLAB. Specifically, you will take data from an actual engineering experiment and process it in such a way that you can determine whether the experimental apparatus behaves in a particular manner.

3.2 Resources

The additional resources required for this assignment include:

- Books: neither
- Pratt Pundit Pages: `MATLAB` and `MATLAB:Script`
- Lab Manual Appendices: C - "Making and Printing Plots in MATLAB."

3.3 Getting Started

1. Log into one of the PCs in the lab using your NET ID. Be sure it is set to log on to acpub.
2. Start X-Win 32 on the PC. Roll the mouse over the X to make sure it is set to Display 0. If not, quit all instances of X-Win 32 and re-open X-Win 32.
3. Start PuTTY on the PC. Load a session, make sure X11 Tunneling is set in the SSH category and connect.
4. Once connected to a machine, switch into your `EGR53` directory and create a `lab3` directory inside it:

   ```
   cd EGR53
   mkdir lab3
   ```

5. Switch to your ∼`/EGR53/lab3` directory:

   ```
   cd lab3
   ```

6. Copy all relevant files from Dr. G's public `lab3` directory:

   ```
   cp -i ∼mrg/public/EGR53/lab3/* .
   ```

   Do not forget the "." at the end.

7. Open MATLAB by typing `matlab &` at the prompt that appears in your terminal window. It will take MATLAB a few seconds to start up.

3.3.1 Preview of MATLAB

Note: this version is written assuming MATLAB R2007b. OIT will be updating to MATLAB R2009a sometime during the course of the semester; as such, the layout of the graphical user interface will change at that time.

1. There should be several components to the MATLAB window on your screen. What you are most interested in right now is the command window. This is where you type commands that allow MATLAB to run. The command prompt in MATLAB is the >> symbol. Go ahead and type `help load` in the command window at the command prompt and press `return`.

2. You have now activated the built-in help feature in MATLAB. Typing `help` followed by a command name will give you information on any command typed. Currently your command window should show the help entry for the *load* command. You should note that although the `help` feature puts all commands in CAPITAL LETTERS for clarity, all commands themselves are in lower case. Take a moment to look over the documentation for the `load` command. Then take a look at the documentation for the `polyfit` command by typing `help polyfit`. This shows you the syntax and options associated with this command to create a fit to polynomial data. In this laboratory we will be using both of these commands, and they will be explained later in more detail. For now, just get a general overview of what these commands do.

3. To get associated with the MATLAB command window and commands, type the following commands at the prompt.

   (a) `x=3`
   This command will set a variable named `x` equal to a 1x1 matrix with the value 3 in it. If `x` already existed, its previous contents are destroyed and replaced with the new matrix; otherwise, MATLAB will create a new variable called `x` and start from scratch. Regardless, `x` will be `[3]` when this command is finished.

   (b) `x=3;`
   Note that the semi-colon suppresses MATLAB's display response to any command. This is used when the user does not care to have MATLAB display a response to the command.

   (c) `x`
   A user is able to find the value of any variable simply by typing that variable's name into the command window.

   (d) `y=x+3`
   When assigning values to a variable, the user may implement already-defined variables in the calculation for that variable. Given the formula above, `y` should now be equal to 6. You can check this by typing `y` into the command window.

   It is important to note here that `y` is *not* created as a function of `x` - rather, it is created from calculations including `x`. Regardless of future changes to the `x` variable, the `y` variable will remain the same. To prove this, type `x=10` and then `y` at the command prompt; `y` is still 6.

   (e) `y+3`
   If no assignment operator (=) is used, but MATLAB performs a calculation or is asked to run a function that returns one or more variables, MATLAB will assign the evaluation of the expression to a variable called `ans`. You can check to make sure that the `ans` matrix contains 9 by typing `ans` at the command prompt.

   (f) `who`
   The user can see the names of the variables that have already been assigned by typing `who`.

   (g) `whos`
   By typing `whos` the user can find out of what type each variable assigned is and its size in addition to the names of the variables assigned.

3.3.2 `.m` Files

MATLAB is a command-line driven program, which means it does not save commands once they are typed. This is problematic when debugging programs because in the command window, if a command is mistyped, all previous commands may have to be retyped to get the program back to the correct state. Instead script files, or `.m`-files, are used in MATLAB to alleviate this problem. You can read more in Pratt Pundit about creating `.m`-files for scripts and functions - specifically in the pages `MATLAB:Script` and `MATLAB:User-defined Function` - though the basics needed for this laboratory are presented within this chapter.

1. Go to the *File* menu and choose the *New* option, followed by the *Blank M-file* option. This will open a new editing window.

2. You can now type commands, just as if you were typing in the command window. The difference is, these commands will not execute until you save the script and type its name in the command window (or hit *F5* while in the editing window, which both saves *and* runs the active script). When the script is run, MATLAB will execute all the commands in the file. Type the lines below in your new script. Note that your file does not yet have a name.

```
x=3
y=4;
z=5
a=x+y;
b=y+z
c=x+z;
x+y+z;
```

3. Save and run this script as `myTest.m` by going to the *Debug* menu and choosing *Save and Run* or by using the *F5* shortcut key on the keyboard. Save the file in your current directory.

4. Now type `myTest` in the MATLAB command window. You will notice that everything in the script runs again, just as if you had typed each of the lines of code in the script at the command prompt. The semi-colon still suppresses output, and any expression without an assignment is still stored in the variable `ans`. You can see that the variables have all been stored the same way by typing `who` or `whos`.

3.4 Cantilever Beam Analysis

Now that you have learned a little bit about how to get around in MATLAB, we will put your new found skills to use by having you interpret a data file in MATLAB and present the data in several different forms. We will be using measurements from a real-life situation, namely the displacement of a cantilever beam. A cantilever beam is an object that is fixed at one end but bends at the other, much like a diving board. The question we will trying to answer is, "Does a cantilever beam act like a regular spring?" The scientific way of asking this question is, "Does Hooke's Law model a cantilever beam over some limited range of deflections?" Hooke's Law states that the deflection of a spring is directly proportional to the force applied to the spring:

$$F = k\ \Delta x$$

This particular laboratory relates to several of the NAE Grand Challenges for Engineering. First, as with all the labs, you will be learning ways to engineer the tools of scientific discovery - specifically by using computational tools to load data, perform calculations to convert the data to a usable form, then use higher-level functions in MATLAB to perform a least-squares fit analysis. You will also be presenting the data graphically, which is a fundamental part of working to enhance virtual reality. Finally, the specific item under consideration - a cantilever beam - is a widely-used structure in architecture and construction; knowledge of just how one functions is key in the effort to restore and improve urban infrastructure.

As this lab serves to provide an introduction to how MATLAB can be used, we will present the code to be used step by step, and you will build a script file that will produce data and graphs to help answer the above question. You then will be responsible for modifying the code such that it analyzes three more data sets to answer the same question on that data.

The data file contains information obtained from an experiment where objects of known mass were placed on the end of an instrumented cantilever beam. The deflection at the far end of the beam was then measured and stored in the data file. Because the amount of force applied is the independent variable and the displacement is the dependent variable, the equation we will be looking at has displacement as a function of force:

$$\Delta x = \frac{1}{k}F + (\Delta x)_0$$

where $1/k$ is the spring's *compliance* and $(\Delta x)_0$ is the initial displacement of the end of the spring.

In order to answer the above question of whether a cantilever beam acts the same as a spring, we will take data from the experiment, mathematically determine the compliance and initial displacement values that make the best predictions for all the data points, and then graphically determine whether it seems the equation properly predicts the data. Note that in a later lab, you will be able to quantitatively establish the goodness of fit.

3.5 Creating the Script

In this section you will be creating a script that can be used to solve much of what you will be asked to do in this lab. In the sections below, you will be shown various MATLAB commands to help process data and generate plots. When you are done, you should have evidence that will help you support or refute the notion that cantilever beams behave the same way that springs do. Once you complete the script, you can then use it to analyze other data sets.

To begin, open a new m-file in MATLAB and save it as `RunCan.m`. The finished contents of this section of the lab handout are copied in Section 3.7 on p. Lab 3 – 16, but you should follow line by line through the narrative below to understand how each line works.

3.5.1 Lab Manual Syntax

Code and results in the lab manual will be set off by certain cues to help you determine what code should go into your work and what code is for demonstration purposes only. Code that you will actually be adding to your script will be surrounded by a shadow box:

```
code you should include
```

while code that is shown for reference purposes only will by surrounded by a single frame:

```
interesting code NOT found in your script
```

MATLAB output will be bracketed by double lines:

```
ans =

My Output
```

3.5.2 Comments

For this lab, comments are both explicitly included in the text of the lab manual itself and are also included in the final code in Section 3.7 on p. Lab 3 – 16. Comments in MATLAB are set off by a single percent sign, %. If there is a % in a line of code, anything else on that line is ignored - this can be useful for writing a quick note about the purpose of a particular set of commands:

```
myVal = 2 + 2; % add two values
```

Also note in MATLAB's m-file editor that using two percent symbols to start a line will cause the editor to believe that starts a new "section" of code. These "sections" are not formally defined as far as the programming itself, but the m-file editor will graphically delineate different sections from each other. Notice in the code on p. Lab 3 – 16 how there are two types of comments - those starting with %% that seem to be "big concept" comments and those just starting with % that are more specific.

While you are generally not *required* to include extensive comments in your code (other than the comments with your name and the honor code statement at the start of each MATLAB file), comments are extremely useful for programmers and can make life much, much easier when troubleshooting codes.

3.5.3 Initializing the Workspace

The first part of the script for this lab will prepare MATLAB to do work. Specifically, the memory will be wiped clean, MATLAB will be told how to display numbers, and the script will pull up and clear a figure window. The code for this section all falls under the comment line:

```
%% Initialize the workspace
```

1. First, it is good practice to clear all variables before using a script. This is to make sure that previous commands within MATLAB will not change how the current program behaves. To do this, use the `clear` command:

   ```
   % Clear all variables
   clear
   ```

2. For this assignment, you will be reading data off the screen. If a matrix contains values that are of vastly different sizes, the default format (`format short`) may report values with varying precision or may lead you to believe values are zero when they are not. For example,

   ```
   [1.2345e2 1.2345e-4]
   ```

   in MATLAB displays as

   ```
   ans =

     123.4500    0.0001
   ```

   in MATLAB while

   ```
   [1.2345e2 1.2345e-5]
   ```

shows up as

```
ans =

  123.4500    0.0000
```

Using `format long` can help this some, but if the numbers are of wildly different scales, even `format long` has difficulty. For instance,

```
[1.2345e2 1.2345e-15]
```

shows up as

```
ans =

   1.0e+02 *
   1.23450000000000   0.00000000000000
```

with `format long`. Instead, one of the exponential formats should be used. These display the values in the matrix using scientific notation, so even the short version gives *each value* five significant figures:

```
format short e
[1.2345e2 1.2345e-15]
```

gives

```
ans =

   1.2345e+02   1.2345e-15
```

Given all that, add a line to your code to guarantee which format you are in:

```
% Change display to short exponential format
format short e
```

3. Next, you will tell MATLAB to start a new figure window (or at least make Figure 1 active). MATLAB can have multiple figures open at once, so it is good programming practice to specify which figure is being used if graphics are being created. To bring up or activate a particular figure, just use the `figure` command with an argument specifying which one:

```
% Bring up a figure window
figure(1)
```

4. In future labs, you will learn that there are a variety of different ways to either subdivide figure windows or overlay the results of several graphical commands onto a single window. In either case, it is generally a good idea, when running a new script, to make sure your figures are starting from scratch. The `clf` command does just that:

```
% Clear the figure window
clf
```

At this stage, you have completed the tasks in the "Initialize the workspace" section of the code. If you save and run the script, you should see a blank figure window called "Figure 1" pop up. If not, carefully check your code to make sure it matches the top of the `RunCan.m` code on p. Lab 3 – 16

3.5.4 Loading and Manipulating the Data

The next section of the code goes through the process of importing the data and manipulating the values such that they have sensible names and appropriate units. This section falls under the comment:

```
%% Load and manipulate the data
```

1. The data file for this particular lab consists of a text file containing eight rows with two columns each of data. MATLAB's `load` command is a simple yet powerful way to load rectangular matrices of data from a text file. To get the data into MATLAB, then, add the code:

   ```
   % Load data from Cantilever.dat
   load Cantilever.dat
   ```

   to your script. This will load the file `Cantilever.dat` from your current working directory and put the data into a matrix called `Cantilever`. Note that when MATLAB loads a text-based data file, it gives the matrix the same name as the file, excluding anything after the first dot in the file name.

   At this point, you should save and run your script by hitting the `F5` key.

2. Now you will want to check on the progress of your program. Click in MATLAB's command window and type `whos`. You will see that you have created a variable called `Cantilever`. If you type `Cantilever` you can see what values are in the matrix.

3. The first column of the data file contains mass data in kilograms. It would be convenient, therefore, if we could extract the data from the first column and put it into a matrix with a more meaningful name, like `Mass`.

   MATLAB allows you to isolate parts of the matrix by specifying the range of columns and rows you would like to copy. The `:` character is a way of expressing *all*. For example, the command `Mass = Cantilever(:,1)` tells MATLAB to take all of the rows and the first column of the matrix `Cantilever` and store copies of them in the variable `Mass`. Similarly you can extract the data in the second column, which represents the displacements that were measured when the mass values in the first column were set on the edge of the beam, with the command `Displacement=Cantilever(:,2)`. You should therefore add the following lines of code to your m-file:

   ```
   % Copy data from each column into new variables
   Mass = Cantilever(:,1);
   Displacement = Cantilever(:,2);
   ```

   Remember, the semi-colon at the end means that MATLAB will do its job but will not display the result on the screen. Go ahead and save and run your script again, and fix any errors that may have developed.

4. If you go back into the command window, you will now notice by typing `whos` that you have created two more variables, `Mass` and `Displacement`. If you type their names in the command window and compare them to the original data in `Cantilever`, you will see they are the first and second columns, respectively.

5. Next you need to convert the data in the vectors into the variables needed to perform the analysis. You also want to make sure all values are in consistent units. Remember - the goal is to analyze how the data fits with the equation

   $$\Delta x = \frac{1}{k}F + (\Delta x)_0$$

   To begin, the `Mass` data needs to be converted to the appropriate values of force for this planet. One of the nice features in MATLAB is that it allows the user to perform operations on an entire vector at

once. To convert a mass in kilograms into a force in Newtons, on Earth, you must multiply the mass by the value of acceleration due to gravity on Earth $\left(\approx 9.81\frac{m}{s^2}\right)$. In your script, you can do this by adding the code:

```
% Convert Mass to a Force measurement
Force = Mass*9.81;
```

This will take all eight entries in the `Mass` matrix, multiply them by 9.81, and store the 8x1 matrix that was generated in a variable called `Force`.

In order to maintain consistent units, the values in the `Displacement` matrix need to be converted to meters from their current unit of inches. Knowing that 2.54 cm=1 in and that 100 cm=1 m, we can come up with the expression $Displacement = \frac{Displacement*2.54}{100}$. Add the MATLAB code equivalent of this command to the script with:

```
% Convert Displacement in inches to meters
Displacement = (Displacement*2.54)/100;
```

in your script. Go ahead and save and run your script again, fixing any errors.

6. Now if you go into the command window and type `whos`, you should see the variable `Force` has been created. If you check the values in the `Force` vector, you will see they are different from both the first column of `Cantilever` as well as the values in `Mass`. If you check the values of the `Displacement` vector, you will see that they have changed, but that the second column of the `Cantilever` matrix has *not*. This demonstrates that in MATLAB you can take a vector and replace it with a modified version of itself, as is happening here with the `Displacement` vector. Your script now has all the building blocks, in the right units, for performing the analysis.

3.5.5 Generating Plots

Before performing calculations, you may find it useful to take a quick "look" at the data. And rather than trying to figure out the relationship between the displacement and the force by staring at a table of numbers, that look could be a plot of the displacement data as a function of the force. For the moment, then, we are going to skip ahead to the

```
%% Generate and save plots
```

part of the script. MATLAB code does not need to be added to the script in the order it is to be executed - sometimes, jumping to the end for a bit can be helpful. For developing the solution to this problem, we will plot the data we have obtained. We can plot data on a figure using the `plot` command. The form of this command that we want to use requires data for the x and y-coordinates of each point. The independent data in each pair, which in this case would be the force since the force values came from the masses we elected to put at the end of the beam, is generally the x-coordinate. The dependent data, which will be the y-coordinate, will be the displacement since this is what we *measured*. The `plot` command also allows the user to add a third argument, which is put in single quotes, that allows the user to define the color, symbol, and line type of the plot. Table C.1 on page App C – 2 of Appendix C shows the different options available to the user.

During the lab, the instructor will have you create several different graphs in the command window using various methods of plotting data. For this script, we will represent this data set by using black circles, so once that part of the demonstration is over, you will be asked to add the following to your script:

```
% Plot Displacement as a function of Force
plot(Force, Displacement, 'ko')
```

3.5.6 Polynomials in MATLAB

At this point, you might be able to state with some certainty whether the data points follow a straight line. For rudimentary analyses, this may be all the code you need. If you just wanted to know for yourself what the experiment did, this code would suffice. For this lab, however, you are going to perform some calculations to determine the best possible straight line, then plot it, to more easily compare the data points and that line.

The equation for a straight line is a specific case of a *polynomial*. Often, engineers will want to see if a data set fits a particular order of polynomial, and if so, what the best coefficients of that polynomial might be to have the data points collectively as close as possible to the model. The general format for a polynomial can be written as:

$$y = a_0x^0 + a_1x^1 + ... + a_{N-1}x^{N-1} + a_Nx^N = \sum_{n=0}^{N} a_nx^n$$

MATLAB has a built-in command called `polyfit` to determine polynomial fits to data. It takes three arguments: an independent data set (in this case, the force), a dependent data set (displacement), and an integer N (the order fit desired). This command, however, returns the values of the polynomial coefficients in a slightly different order from that shown above. Specifically, if the variable `P` is where you have told MATLAB to store the polynomial coefficients, the equation that MATLAB will fit for an N^{th} order polynomial is:

$$y = P(1)x^N + P(2)x^{N-1} + ... + P(N)x^1 + P(N+1)x^0 = \sum_{i=1}^{N+1} P(i)\ x^{(N+1-i)}$$

This is because of the way MATLAB interprets arguments to functions that are supposed to represent polynomials. To clarify this further, some common polynomials are found in Table 3.1. Note in the

Order	N	Example	Polynomial Representation
First Order	N=1	$y = 3x + 4$	`P=[3 4]`
Second Order	N=2	$y = 3x^2 + 4x + 5$	`P=[3 4 5]`
Third Order	N=3	$y = 3x^3 + 4x^2 + 5x + 6$	`P=[3 4 5 6]`
Third Order	N=3	$y = 3x^3 + 5x$	`P=[3 0 5 0]`

Table 3.1: Common Polynomial Orders

last case that the far right entry in the `P` matrix *always* represents the coefficient of the zeroth power - if that coefficient happens to be 0, you cannot simply omit it. Similarly, any other "missing" coefficients must be represented by 0's in the representational vector for MATLAB to be able to understand the coefficients properly.

For this lab, you want to perform a first-order polynomial fit on the current data. This will come under a section of code with the heading

```
%% Perform calculations
```

which comes *after* manipulating the data but *before* plotting anything.

First, note that fits are generally performed where you solve an equation for the dependent data as a function of the independent data. This means that the equation to fit will be for displacement as a function of the force (i.e. $\Delta x = a_1F + a_0$ or, using MATLAB's terminology, Δx=P(1) F+P(2)). Add the following command to your script:

```
% Use polyfit to find first-order fit polynomials
P = polyfit(Force, Displacement, 1)
```

(note there is no semi-colon at the end) and save and run your script. Once you fix any errors that may exist, you will notice that the values for the first order fit's coefficients are stored in the variable `P`. Make sure the slope and intercept presented in the `P` matrix make sense given the data set.

3.5.7 Generate Predictions

Now that we have a polynomial equation representing the best linear fit of the data, we want to plot it on the same graph as the original data. To do that, you must produce a matrix that contains the numerical values of that equation over the domain of the independent data.

Remember that to plot a curve in MATLAB, you should provide the `plot` command with arrays containing the x and y coordinates as well as information about what color, symbol, and/or line style should be used. The process of creating these vectors will take place in a section of code headed with

```
%% Generate predictions
```

which comes between performing calculations and generating the plots.

(1) We first need to generate several points along the x-axis (`Force`) that can then be plugged into the polynomial equation. We can accomplish this by using a MATLAB command called `linspace`, which generates a row vector of equally spaced points. Add the following line to your script:

```
% Create 100 representational Force values
ForceModel = linspace(min(Force),max(Force),100);
```

This creates 100 equally spaced points between the minimum and maximum value in the `Force` vector. We will only be making predictions for how the cantilever beam behaves within the range of the experimental data.

(2) Now we can calculate the corresponding displacement values based on the polynomial equation, which again is represented by the `P` vector, at each of the 100 points in the `ForceModel` vector. To do this we will use the `polyval` command. `polyval` takes a `polyfit` vector and an independent data set as arguments. For example, if you were to want to calculate the results of $r = t^3 + 2t + 5$ for $t = [1\ 2\ 3\ 4\ 5]$, you could issue the commands:

```
myCoefs = [1 0 2 5]
t = [1 2 3 4 5]
r = polyval(myCoefs, t)
```

and MATLAB would calculate:

```
r =

     8    17    38    77   140
```

You could also perform all the work in one line:

```
r = polyval([1 0 2 5], [1 2 3 4 5]);
```

to get the results. Note the 0 in the second entry of `myCoefs` - recall that you must include coefficients for all integer powers of the independent value from the highest down to the zeroth power.

For the script, the coefficient vector `P` and independent model points `ForceModel` have already been calculated, so add the command

```
% Calculate Displacement predictions
DispModel = polyval(P, ForceModel);
```

3.5.8 Generating Plots (revisited)

At this stage, we now have another set of data to plot - specifically, we now want to plot the model line on the same graph as the experimental data points so they may be compared visually. We will use a line this time to represent the polynomial fit. The code for this section is back in the

```
%% Generate and save plots
```

section and will be added below the first plot command.

Whenever you want to get more than one data set onto a graph, one way to proceed is to tell MATLAB to hold on to the graph that is already in place and keep adding data sets to it until you are done. You can tell MATLAB this by typing `hold on`. While this command is in place, future `plot` commands will all use the same graph (though the axes will expand if they need to in order to accommodate the new data range). When you are ready to start over, issue the `hold off` command. It is important to note that the `hold on` command needs to be given *after* the first plot command, so that the first plot command clears out any previous plots. In this particular case, we are only adding one plot to the figure; if you want three or more plots on one figure you would put the second and subsequent plot commands between the `hold on` and `hold off` lines.

(1) At this point in your script, you will want to add the following lines of code:

```
% Turn hold on, plot the model values, and turn hold off
hold on
plot (ForceModel, DispModel, 'k-')
hold off
```

(2) Many times, it will be useful to turn a grid on to more easily determine where particular data points are located on a graph. To add a grid to this figure window, add

```
% Turn the grid on
grid on
```

to your code.

(3) Now, to create a *proper* graph[1], we need to add labels. The commands `xlabel` will add a label on the x-axis, `ylabel` will add a label to the y-axis, and `title` will generate a title. For this particular graph, add the following commands:

```
% Label and title the graph
xlabel('Force (Newtons)')
ylabel('Displacement (meters)')
title('Displacement vs. Force for Cantilever.dat (NET ID)')
```

where NET ID is your NET ID. Every graph you produce for this class should have your NET ID in the title as further proof that it is your work. Now you should save and run your script to produce a graph with the original data as black circles and the model line as a straight line.

(4) This is the final plot we will use to analyze the data. You can estimate the "goodness of fit" by looking at the overall spacing between the model line and each of the data points. If each point is relatively close to the line, this is likely a good fit; if several or all of the points are far away, it is not as good. Later in the course, we will learn quantitative methods that determine this goodness of fit. Since we want to print this plot out, add the command

```
% Save the graph to PostScript
print -deps RunCanPlot
```

[1] Another way of saying this is, "Now, to create a graph for which you can receive more than zero credit..."

to the end of your script. This will tell MATLAB to generate a PostScript file with the current plot it in called `RunCanPlot.eps` - it will *not* actually send the graph to the ePrint queue. Note: if the file name you tell MATLAB to print to has no dots in the title, MATLAB will automatically add the `.eps` to the end.

As an aside, to view this graph without including it in a LaTeX file, go to the *terminal* window (*not* the MATLAB command window) and type

```
kghostview RunCanPlot.eps &
```

For this class, you will generally import the graphs as part of a lab report and print them out that way. If you have some need to print out a MATLAB graph on its own, however, you can send the file to the printer from `kghostview`. Another way to print a file is, in Unix, type `lpr FILENAME` and the file will be sent to the ePrint queue.

Congratulations, you have completed the walk through portion of this lab! If you have followed the instructions correctly, your plot should look like Figure 3.1. Now move on to the assignment and documentation portions of the lab.

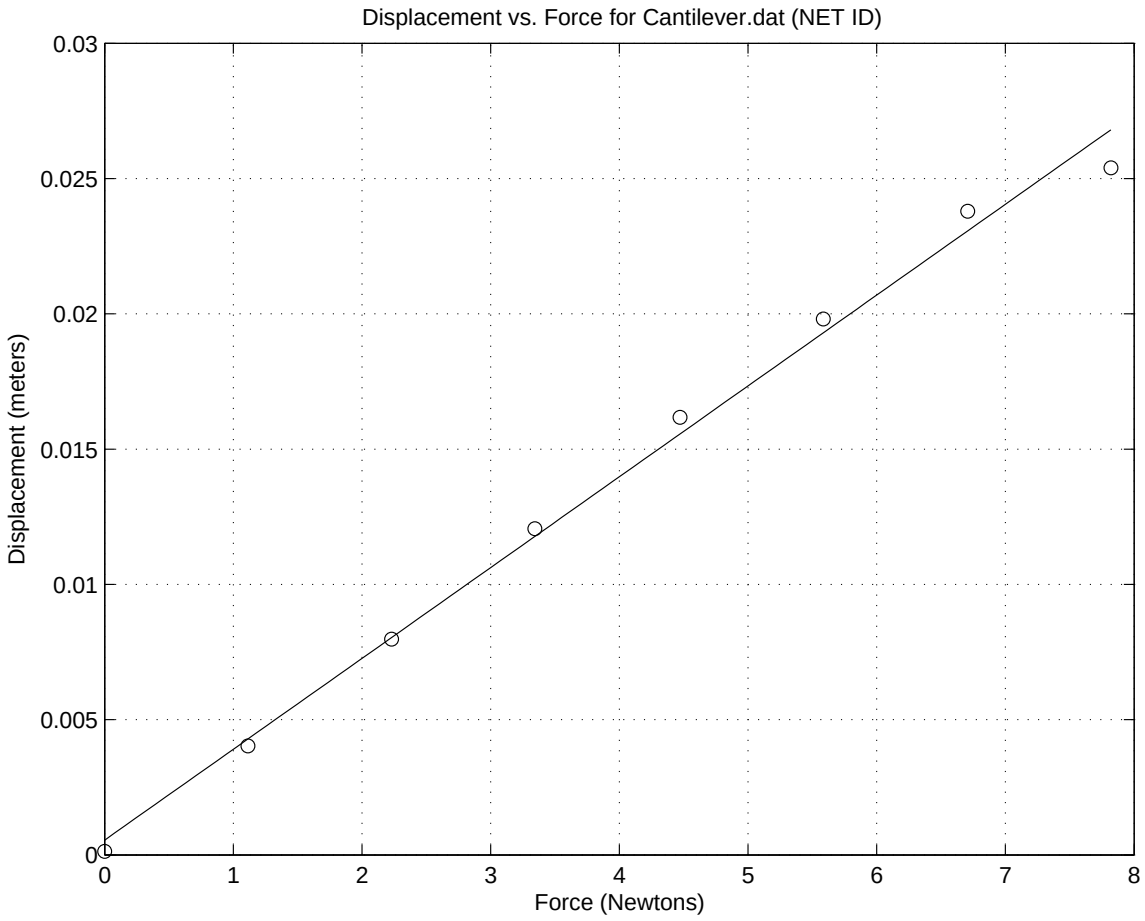

Figure 3.1: Linear Fit of Cantilever Beam Data

3.6 The Assignment

Since you have now acquired some MATLAB tools in your engineer's toolbox, you should be able to write a program in MATLAB that will be of use in analyzing data. You are going to write programs that will load displacement and mass data for three different data sets and then determine the parameters for a first order fit. We are interested in the parameters of that fit in the equation:

$$\text{Displacement} = \text{Compliance} * \text{Force} + \text{Initial Displacement}$$

where Force is your independent variable, Displacement is your dependent variable, and Compliance and Initial Displacement are the parameters you will find given your fit (that is, `P(1)` and `P(2)`, respectively). We also want to know, *qualitatively*, how good the fit is. Later in the course, we will cover methods for determining *quantitatively* how good a fit is.

3.6.1 What the `.m`-files Should Do

Basically the `.m` file is going to repeat line for line the code that we have already typed, only for different data files. Please complete the following steps for each of the three additional data files (`Beam1.dat`, `Beam2.dat`, and `Beam3.dat`) you copied from the public directory for this lab. You should write three scripts - one for each data file - but can copy and paste your work as appropriate. Make sure that the title of each plot includes not only the words `Displacement vs. Force` but also an indication of which of the three data files is being used as well as your NET ID. The instructions below specifically refer to the `Beam1.dat`.

1. Load a data set. For example if you are loading Beam1.dat, use `load Beam1.dat`:
2. Separate the data into a mass vector and a displacement vector. For example if you have loaded the `Beam1.dat` file and wish to extract the `Mass` data use the command `Mass=Beam1(:,1);`
3. Convert the mass in kilograms to a force in Newtons and convert the units of displacement from inches to meters.
4. Plot a displacement versus force graph, remembering that the independent values (force) go on the x-axis. Use black circles without lines to represent the data.
5. Determine a first-order fit for displacement (the dependent variable) as a function of the force using `polyfit`.
6. Generate a model of this fit using at least 100 points and plot it, using black lines, on the same graph as above, using the results of the `polyfit` command as well as the `linspace`, `polyval`, and `plot` commands.
7. Be sure to label the graph properly using the `xlabel`, `ylabel`, and `title` commands.
8. Add a grid to the graph using the `grid on` command.
9. Save this figure to a PostScript file `BeamN_graph.eps` using the `print -deps` command. For example, for `Beam1.dat`, the file would be called `Beam1_graph.eps`.

While writing your code, pay special attention to which lines of code you must change from script to script and which code remains the same. Later in the course, we will work on making code more flexible and interactive so that more data files can be analyzed using fewer changes.

3.6.2 The Lab Report

The lab report will contain the following information:

- A brief description of what your program does and what questions you are expected to answer based on the data and graphs you obtain from it.
- The original data for each of your three data sets. Put each set in a tabular environment - for example:

Cantilever.dat	
Mass (kg)	**Disp.** (in)
0.0000	0.0052
0.7971	1.0000

The code for the table above is simply:

```
\begin{center}
\begin{tabular}{|c|c|}\hline
\multicolumn{2}{|c|}{Cantilever.dat}\\ \hline
{\bf Mass} & {\bf Disp.}\\
(kg) & (in)\\ \hline
    0.0000&    0.0052\\
    0.7971&    1.0000\\ \hline
\end{tabular}
\end{center}
```

You can use either the `format short` format for numbers, shown above, or the `format short e` version MATLAB produces.

- Produce an estimate of the Compliance (`P(1)`) and Initial Displacement (`P(2)`) for each of the data files based on your polynomial fit. Put all estimates for the three beams in one tabular environment. Clearly label which constant your are giving and for which data file. For example:

Data File	Compliance (UNITS)	Displacement (UNITS)
`Beam1.dat`	α	β
`Beam2.dat`	γ	δ
`Beam3.dat`	ϵ	ζ

You must determine and include the appropriate units, and you will replace the Greek letters above with the appropriate values. You can use the `format short e` version of the numbers here.

- The required figures will be placed in an appendix, though you can use the `ref` and `pageref` commands in LaTeX to refer to them in the body of the report. The skeleton for the lab report shows sample code for how to accomplish this, as well as code to automatically produce a list of figures. More detailed information on importing figures is given in the Appendix B.
- Write a paragraph explaining the results you obtained. Also state, based on the graphical evidence you have generated, which beams act like a spring and which do not (i.e. how well does the equation model the force-displacement relationship for each data set?).

- **Appendices**
  - A copy of each of the three files you used. Each script will go in its own appendix, and code is provided in the skeleton for how to accomplish this.
  - One figure for each data set, including the data and the model line. It is critical that the title of the plot include a reference the specific data file you used for that graph as well as your Net ID.

3.6.3 Processing the Lab Report

Because the lab report contains `label`, `ref`, and `pageref` commands, as well as a table of contents and a list of figures, you may need to run LaTeX on it three times before it is properly compiled! The first time through, LaTeX will figure out to what and where the labels refer. The second time through, LaTeX will replace all the `ref` and `pageref` commands with the appropriate information. The third time through, LaTeX will correct any changes in the document caused by the replacement characters. In UNIX, the easiest way to repeat the previous command is simply to hit the up-arrow and hit return.

Sadly, the most common mistakes on lab reports involve spelling, grammar, and carelessness. There is a Spell Checking option in `emacs` under the `Tools` menu. Be sure to use it. Also make sure to re-read your document one last time before submitting it; you may be surprised at the typographical and grammatical errors you find in the "final" run-through. Almost as surprised at how many typographical and grammatical errors you find in this document, despite years of "final" runs-through.

3.7 Code From Lab

The following is a copy of the code written while following the directions included in this lab, including extra comments and line numbers:

```
1  %% Initialize the workspace
2  % Clear all variables
3  clear
4  % Change display to short exponential format
5  format short e
6  % Bring up a figure window
7  figure(1)
8  % Clear the figure window
9  clf
10
11 %% Load and manipulate the data
12 % Load data from Cantilever.dat
13 load Cantilever.dat
14 % Copy data from each column into new variables
15 Mass = Cantilever(:,1);
16 Displacement = Cantilever(:,2);
17 % Convert Mass to a Force measurement
18 Force = Mass*9.81;
19 % Convert Displacement in inches to meters
20 Displacement = (Displacement*2.54)/100;
21
22 %% Perform calculations
23 % Use polyfit to find first-order fit polynomials
24 P = polyfit(Force, Displacement, 1)
25
26 %% Generate predictions
27 % Create 100 representational Force values
28 ForceModel = linspace(min(Force),max(Force),100);
29 % Calculate Displacement predictions
30 DispModel = polyval(P, ForceModel);
31
32 %% Generate and save plots
33 % Plot Displacement as a function of Force
34 plot(Force, Displacement, 'ko')
35 % Turn hold on, plot the model values, and turn hold off
36 hold on
37 plot (ForceModel, DispModel, 'k-')
38 hold off
39 % Turn the grid on
40 grid on
41 % Label and title the graph
42 xlabel('Force (Newtons)')
43 ylabel('Displacement (meters)')
44 title('Displacement vs. Force for Cantilever.dat (NET ID)')
45 % Save the graph to PostScript
46 print -deps RunCanPlot
```