Laboratory 2: Using Matlab for Geologic Applications

George Hilley

1 Introduction

Matlab is a program that has been developed for the general solution of mathematical equations, data processing, and data display. It is an excellent tool for developing numerical models of problems, performing mathematical operations on large data sets, displaying data in one, two, three, or four dimensions, and solving simultaneous equations. In general, Matlab has been developed as a numerical tool and so best works on real numbers, rather than symbolic operations. However, some versions of Matlab (for example, the Solaris version) are capable of symbolic solutions of equations, symbolic integration and differentiation, and reduction of symbolic equations. The IRIX version is not capable of many symbolic operations at this time.

In short, if you can write an equation that you think represents a relationship between two or more variables and need to graph your results, Matlab is the tool to use. First, we will go through a short introduction on Matlab concepts, entry formats, graphics, M-files, and functions. In order to demonstrate the format of Matlab and how to make programs in Matlab, we will proceed through two examples. The first is how to set up a simple linear regression in Matlab and the second is how to set up a model of a bending plate in Matlab and then use data to find the best fitting solution to the controls on how the plate will bend. These examples are intended to provide an idea of the use of Matlab for geologic applications as well as provide some working code for you to understand and refer back to when developing your own programs.

2 Matlab Concepts and Basic Data Entry

First and foremost, it is important to know how to start Matlab. To start up the program, log into Alai or Darkwing (both machines have about the same version of the software). At the prompt, type "matlab". This will start the program. If you are telneting to the machine, graphics will be disabled; however, if you are on the console or are using an XDMCP connection to one of the machines, you will be able to display graphics. The standard Matlab prompt is ">>", so once you get this prompt, Matlab is ready to be used.

2.1 Variable Assignments in Matlab

title between curly braces Matlab is a vector-based program and works most efficiently if you can pose your problem in the context of vector, matrices, and tensors. In general, all mathematical operations are performed on matrices (a vector is a one column/row matrix, a single number is simply a one element matrix). In order to assign values to a matrix, you need a variable name for the value (which should have some significance to the variable in the problem you are addressing). Here is how to make a variable assignment:

```
>>myVariable = 6
```

You will notice that Matlab will echo your output by saying:

```
myVariable = 6
```

If you do not wish to see a confirmation of your assignment, include a semicolon after the operation. This rule holds for all assignments or operations– if you do not want to see the result of the operation, but simply wish to store it in another variable, use a semicolon:

```
>>myVariable = 6;
```

The assignment that you have just made consists of a one element vector. If you wish to assign an entire vector to a variable, enclose the vector's values in brackets. Columns are designated by spaces, while rows are designated by semicolons. For example

```
>>myVariable = [1 2 3 4 5 6 7 8 9 10];
```

creates a ten element row vector on which operations can be performed. To create a column vector, instead of spaces, use semicolons:

```
>>myVariable = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10;]
```

This creates a ten element row vector with the assigned values for each of the elements. A row vector is the transpose of a column vector, therefore, you can use the transpose command to convert from a row vector to a column vector and vice versa. The transpose command can be called in a couple of different ways. The first is to use the `transpose()` command which will return the transpose of the vector that you give the command:

```
>>newVariable = transpose(myVariable);
```

will return the transpose of `myVariable` and store the result in `newVariable`. Alternately, you could specify that the result of the transpose could be stored back into `myVariable`, and thus you are directly converting myVariable from a column vector to a row vector or vice versa:

```
>>myVariable = transpose(myVariable);
```

The other way to transpose a vector is to use the ' shorthand:

```
>>myVariable = myVariable';
```

Another way that you can make a vector assignment is to assign each element one after the other by using array notation. In this notation, the j^{th} element of the vector `myVariable` is represented by `myVariable(`j`)`. Therefore, you could assign a three element vector with the following commands:

```
myVariable(1) = 1;
myVariable(2) = 2;
myVariable(3) = 3;
```

These commands are equivalent to the following command:

```
myVariable = [1 2 3];
```

The final transition is from vectors to matrices. A vector is a one row or column matrix; therefore, a set of i row vectors of length j forms the $i \times j$ matrix (a matrix with i rows and j columns). Matrix assignments can be made as follows:

```
myMatrix = [1 2 3;4 5 6;7 8 9];
```

This will create the 3×3 matrix:

$$myMatrix = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

An alternative way to assign values to a matrix is to use the element array assignment method. This method is the same as assigning elements in a vector; however, elements of a matrix are denoted as (i, j) elements. For example, to set the element in the i^{th} row and j^{th} column of the matrix `myMatrix` to 2, use the `myMatrix(`i, j`) = 2` command at the Matlab prompt. To set up the matrix `myMatrix` shown above, you can issue the following commands:

```
myMatrix(1,1) = 1;
myMatrix(1,2) = 2;
myMatrix(1,3) = 3;
myMatrix(2,1) = 4;
myMatrix(2,2) = 5;
myMatrix(2,3) = 6;
myMatrix(3,1) = 7;
myMatrix(3,2) = 8;
myMatrix(3,3) = 9;
```

Those are the basics of variable assignment which is the first thing that you should learn with Matlab. Now that we have assigned variables, we can start to manipulate the variables in order to perform mathematical operations.

2.2 Making Incremental Variables in Matlab

If you need to make variables that increment regularly, Matlab has some simple built-in functions to do this. For example, if you need to create a vector that starts at 1 and ends at 100 with a spacing of 1:

$$incrementVector = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & \cdots & 100 \end{bmatrix}$$

just use the command:

```
x=1:1:100;
```

Notice that the first number in the assignment is the starting point, the second number is the increment, and the third is the ending point. Both the starting and ending points are inclusive; therefore, the above command creates a 100 element vector.

This command is very useful when setting up a domain for the independent variable(s) in an analysis. In addition, graphics in Matlab are made easier with is feature. Below, we will see how the incremental vector is used in both analysis and graphics.

2.3 Mathematical Operations in Matlab

Once you have created one or more vectors, you can perform mathematical operations on these vectors. Adding and subtracting matricies, vectors, and elements uses the same syntax for each operation and is independent of matrix dimension. For example, if you would like to add vector a to vector b and store the result in vector c, you would type the command:

```
a + b = c
```

Likewise, if you would like to subtract vector b from vector a and store the result in vector c, use the command:

```
a - b = c
```

As with any matrix addition or subtraction, the operation is elementwise, thus both vectors a and b must be the same dimensions and the resultant vector c will have the dimensions of vectors a and b.

As in linear algebra, multiplication of matricies must be more specific due to the fact that there are two ways to multiply a matrix (i.e. dot product of two matricies or cross product of two matricies). If you wish to take the dot product of the matricies A and B (elementwise multiplication of matrix A and matrix B), and store the result in matrix C use the following command:

```
C = A .* B
```

Notice that when you take the dot product of two matricies, they must have the same dimensions and the resulting matrix will also have those dimensions.

If you need to take the cross product of a two matricies A and B and store the result in matrix C, use the following command:

```
C = A * B
```

When taking the cross product of two matricies, the dimensions of matrix A must be (i, j) and the dimensions of matrix B must by (j, i). The resulting matrix C will have dimensions of (i, j).

To raise an element to a power, use the power operator in Matlab, which is a `^`. For example, if you would like to raise a matrix or vector to a power of 2 (using cross products), type:

```
C = A^2
```

Notice that when you do this, the matrix A must be square. If you would like to perform a power operation elementwise on a matrix or vector, use the `.^` operator:

```
C = A.^2
```

Square roots are easily computed with fractional powers.

If you would like to find the inverse of a matrix, you can use the `inv()` command in Matlab. For example, if you would like to find the inverse of the matrix A and store the result in matrix C, use the following command:

```
C = inv(A)
```

Also, you could achieve the same result with the Matlab shorthand:

```
C = \A
```

These are the basic arithmetic operators that you will use in Matlab. Remember to be careful to specify the method of operation on multiplication, division, and matricies raised to powers.

2.4 Displaying Data in Matlab

A nice feature of Matlab is its easy to use graphics interface. It can create 1, 2, 3, and 4 dimensional plots of your data. We will stick to 2 and 3 dimensions but there is more information about making data animations in the Matlab Manual.

There are all kinds of plots that Matlab can make. Explanations of all of these plots and how to use them are given in the Matlab Graphics manual.

Two Dimensional Plots

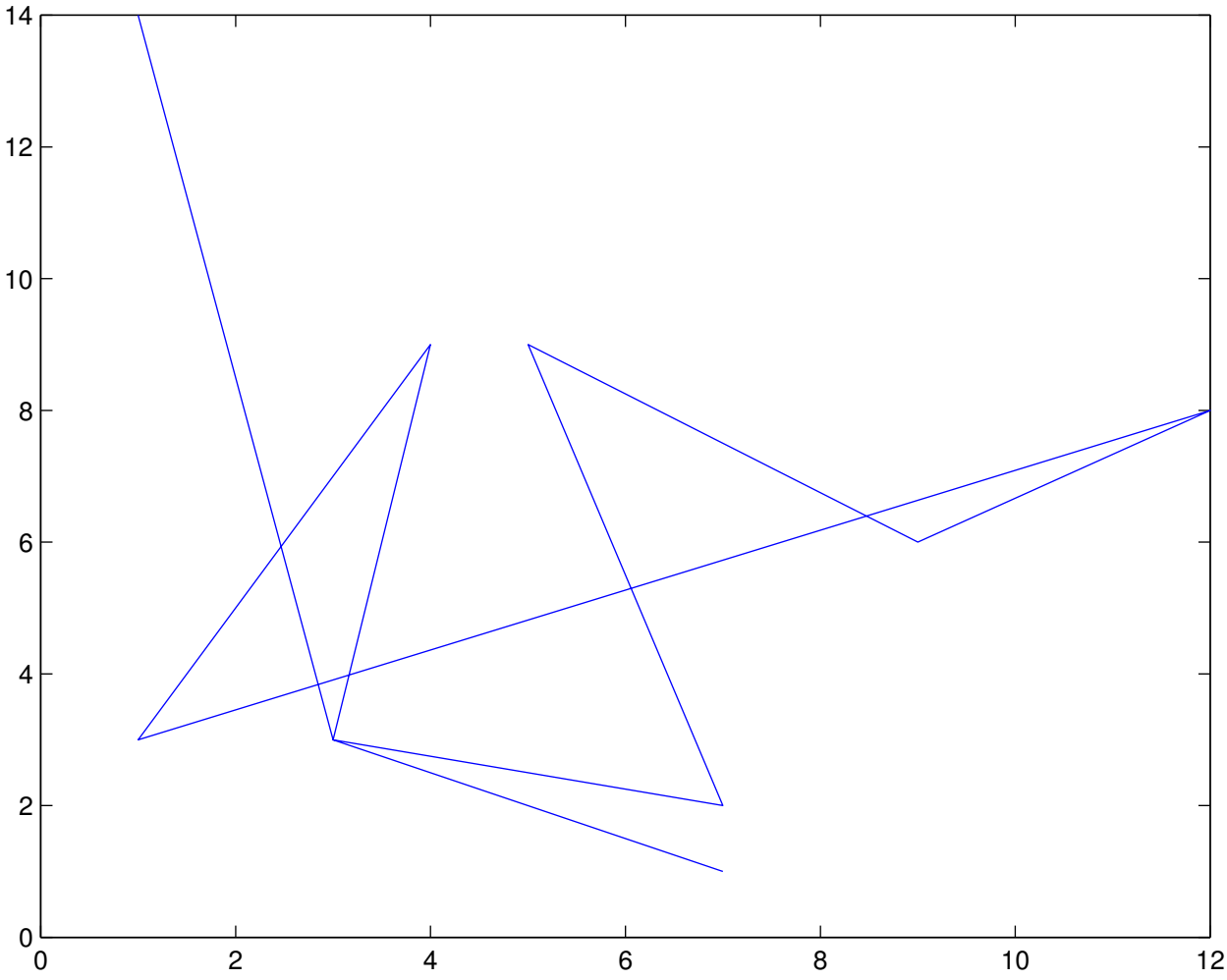
The basic two-dimensional plot can be created with the `plot()` command. The `plot(m,n)` command takes two vectors, m and n and plots m as the abscissa and n as the ordinate in two dimensional space. For example, if you have a vector x which represents all of the x-values of the points you wish to plot and a vector y with all of the y-values of the points that you wish to plot, make the plot by typing: `plot(x,y)`

This will create a two-dimensional plot whose points are the $x(i)$, $y(i)$ pairs given to the plot function. The plot command by default connects the (x, y) pairs as a line; however, below I discuss how to change this option.

The following is an example of how to create a two-dimensional plot:

```
>>x = [1 3 7 5 9 12 1 4 3 7];
>>y = [14 3 2 9 6 8 3 9 3 1];
>>plot(x,y)
```

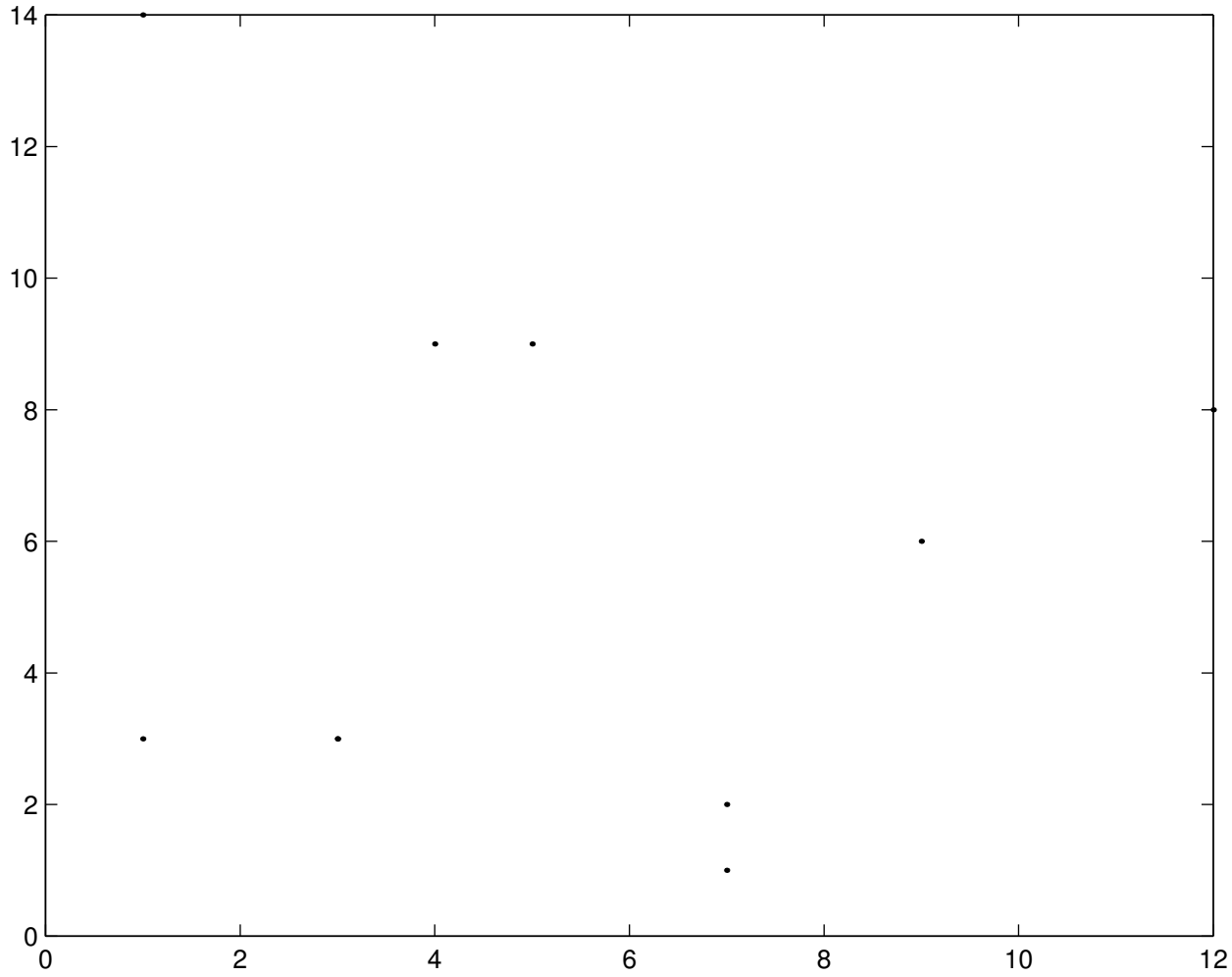
The following plot is printed by Matlab:



Notice that the line follows the path in order of the (x, y) coordinates. This is a bit messy, so we will want to change the plot output to plot as points, rather than lines:

```
>>plot(x,y,'k.')
```

This command plots the (x, y) points defined by vectors x and y.



You can alter the appearance of the way Matlab graphics plot by manipulating the third argument in the plot command. First, note that the third argument is always surrounded by quotes. Next, there are two parts to the third argument of the plot command– the first is a plot color, the second is a plot style. Characters such as 'r' (red), 'b' (blue), 'g' (green), may be used to alter the color of the plotted line or point. Likewise, characters such as '.' (point plot), '+' (points are plotted as crosses), 'v' (points are plotted as upside-down triangles), '–' (plot as dashed line), '-' (plot as solid line) may be used to alter the appearance of the line. For example, if you would like to plot the (x, y) pairs as a red, dashed line, use 'r–' as the third argument of the plot command.

You may overlay plots by using the `hold on` command at the Matlab prompt. This causes overlaying of graphics. For example, let's say that you think that the above points obey some sort of relationship which is a straight line. You know that the slope of the line is 3 and its y-intercept is 2. In the equation:

$$y = ax + b$$

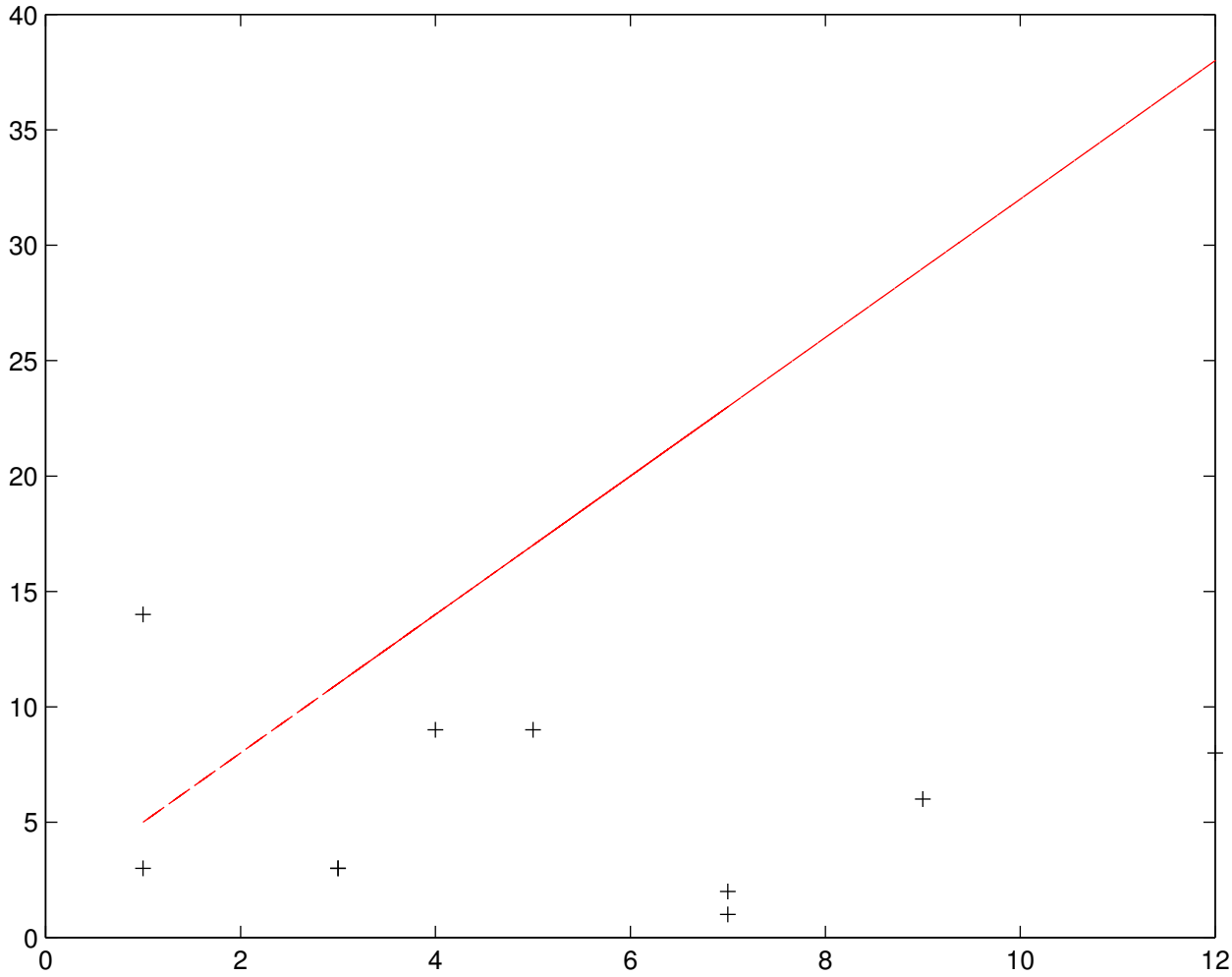
$a = 3$ and $b = 2$. In order to calculate where this line will fall in relationship to the data, we can type the Matlab commands:

```
>>a = 3;
>>b = 2;
>>ycalc = a.*x + b;
```

Therefore, we have two y vectors– one that contains the actual data (y), and one that contains the calculated (or modeled if you prefer) relationship (y_{calc}). Let's plot the actual data as black crosses and the calculated relationship as a red dashed line:

```
>>plot(x,y,'k+')
>>hold on
>>plot(x,ycalc,'r--')
```

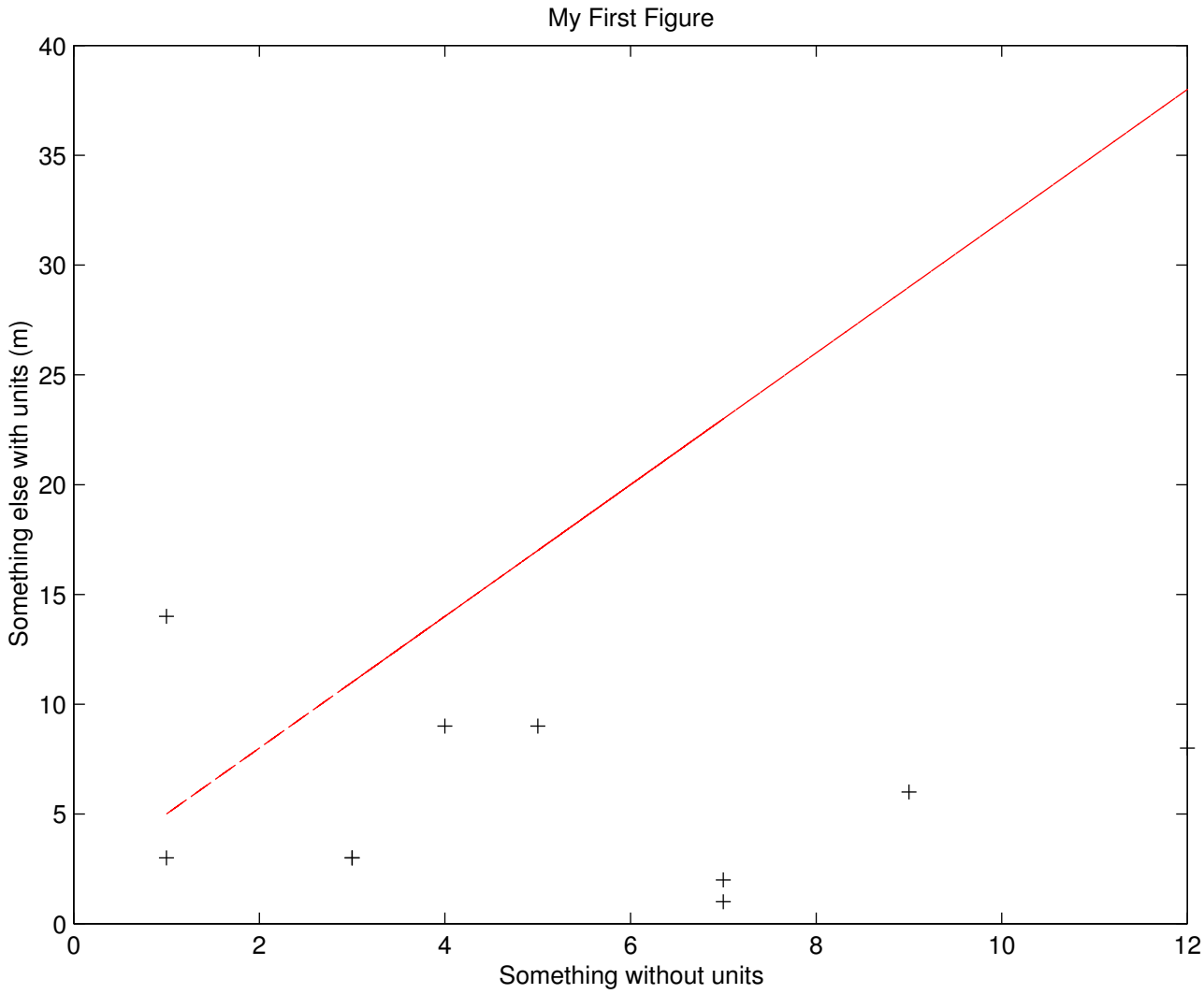
This creates the following output:



Now, you may wish to dress up your plot a little bit by adding a title and axes labels. Matlab will allow you to create a title using the `title()` command and make axis labels using the `xlabel()` and `ylabel()` commands. So, let's add a title and axes labels to the figure:

```
>>title('My First Figure')
>>xlabel('Something without units')
>>ylabel('Something else with units (m)')
```

So, here is your finished figure:



3 Writing Functions in Matlab

So far, you have used matlab commands that are provided with the program to do certain tasks. For example, to make a simple plot, you used the `plot(x,y)` command, or to take the square root of a number or matrix, you used the `sqrt(x)` command. However, it is possible to make your own functions to perform complex operations. For example, a standard operation that you may have to calculate is to take the square root of a certain number, add another number, and then raise this value to a power. In other words,

$$y = (\sqrt{x + a})^n \tag{1}$$

Instead of typing the appropriate values for a, x, and n every time you want to perform this calculation, you can make a "function" to do this. A function takes input arguments, performs a series of operations on them, and then may return an output argument.

To declare a function, open a new M-file. The first line of a function declaration must look like this:

```
function [y] = myoperation(x,a,n)
```

In this first line, matlab is told that this is file will be a function, it will output the argument y, it will take as inputs x, a, and n, and it will be performed when you use the "myoperation" command at the matlab prompt. To perform the above operation, the entire function looks like this:

```
function [y] = myoperation(x,a,n)
y = (sqrt(x+a)).^n;
```

Now, save this file, and you can use it at the matlab command prompt by this command: `>> y = myoperation(2,1,12)` So, this command will take the square root of 3 (2+1) and raise it to the 12th power.

4 Lab Assignment

You have been hired as a consultant to help assess the seismic hazard in a tectonically active region. The hazard is apparently generated from a blind thrust fault adjacent to a major city. Microseismicity data indicate that the fault dips approximately 30 degrees and reaches a depth of 10 km; however, the proximity of this fault to the surface remains unknown. Above the fault, a 36 km-long fold (along strike) has formed. In addition, earthquakes apparently occur on this fault approximately every 200 years.

To better understand the hazard, you have mapped and collected topographic data from six fluvial terrace profiles along the length of the fault. Presumably, uplift and warping of the terraces records the effects of faulting in the area. For each of the six profiles, four terraces were surveyed. In addition, radiocarbon ages for each of the terraces in each drainage were determined. Each river profile runs perpendicular to the crest of the fold. The profiles are located 4, 12, 18, 24, 32, and 34 km from the termination of the folding. Importantly, the current slope of the channel is approximately constant along the surveyed section and is 2 degrees in the downstream direction. Presumably this defines the initial geometry of each of the terraces.

Your job is to use Matlab to analyze the topographic data in terms of the fault slip that produced the warping of the terraces. You will compute the surface displacements by summing the effects of two opposing, infinitely long edge dislocations starting at the upper (w_1 in Figure 1) and lower (w_2 in Figure 1a) edges of the fault, respectively. The equations for relating surface displacements to fault geometric parameters are [Du *et al.*, 1994]:

$$U_x = -\frac{1}{\pi}\left(b_x \tan^{-1}(d) + \frac{b_z - b_x d}{1+d^2}\right) \tag{2}$$

$$U_z = \frac{1}{\pi}\left(b_z \tan^{-1}(d) + \frac{b_x + b_z d}{1+d^2}\right) \tag{3}$$

where, U_x	is the horizontal displacement at the free surface (L),
U_z	is the vertical displacement at the free surface (L),
b_x	is the horizontal component of the Burgers vector (L),
b_z	is the vertical component of the Burgers vector (L),
and d	is the geometric parameter.

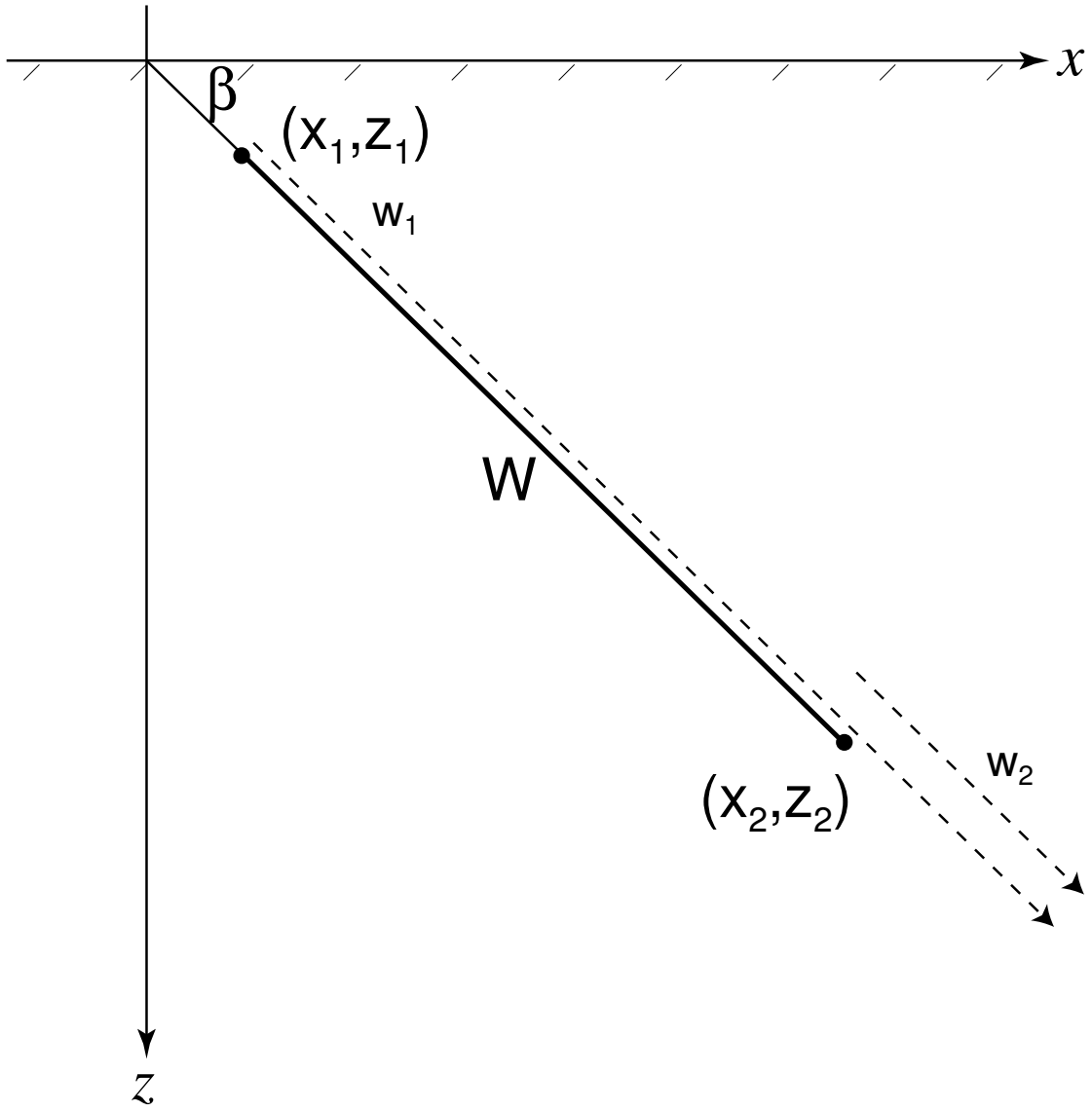


Figure 1: Model fault geometry and controlling parameters. The surface displacements are computed by summing the effect of slip along two infinite dislocations extending downward (w_1, w_2), whose ends start at (x_1, y_1) and (x_2, y_2), respectively. The hachured line is the free surface and the solid line is the idealized fault.

The geometric parameter locates the tip of the dislocation relative to the point of interest along the free surface:

$$d = \frac{(x - x_n)}{z_n} \tag{4}$$

where x is the horizontal coordinate of the surface point of interest (L),
x_n is the x coordinate of the tip of the n^{th} dislocation (L),
and z_n is the vertical coordinate of the tip of the n^{th} dislocation (L).

Finally, the dip of the dislocation is defined by the magnitudes of the components of the Burgers vectors. The components of the Burgers vector are related to the fault dip as, $b_x = b\cos\beta$ and $b_z = b\sin\beta$.

Surface displacements are determined by summing the displacements U_x and U_z due to the two oppositely oriented dislocations defining the slip patch. By repeating this calculation for many points along the surface, we can determine the surface displacement field due to slip along the simulated fault. Although these equations relate fault slip to surface displacement, they may also be used to compute surface displacement rates by differentiating Equations 2 and 3 with respect to time.

So, here is what to do:

1)Load each of the profiles from the `ProfileData` file. Use the matlab `load ProfileData`. Next, plot each of the terraces in the profile on the same graph.

2) Make a function in Matlab that computes the displacements at the surface in terms of the fault dip (30 degrees in this case) the total fault displacement (unknown), the location of the upper tip of the fault under the surface (unknown), the location of the fault relative to the center of the profile ($x = 0$), and the location of each of the surveyed points.

3) For each terrace at each profile, find the closest match between the observed topographic profile and that which would be produced from a fault with a given slip and burial depth. Plot best-fit modeled profiles with observed topographic profiles. Assume that the best-fit depth does not change between either terrace surfaces in a single drainage or between all drainages.

4) Use the ages of the terraces to infer a slip rate along the blind fault at each of the six profile locations. Plot this slip rate as a function of distance along the fold axis.

Writeup:

In your report, make a detailed assessment of the slip history along the fault (rates, how it may change along the fold), using the graphs prepared in the exercise. Assuming the fault ruptures every 200 years, what is the average slip along the fault? What is the peak slip? Do you think this poses a significant seismic hazard. Please write the report in the form of Introduction, Methods (your data analysis methods), Results (what you found), Interpretations (what are the interesting aspects of your data and how do they relate to seismic hazard), and Conclusions (what do you think about the hazard in the area?).

Terrace Ages

	Profile Number 1	Profile Number 2	Profile Number 3	Profile Number 4	Profile Number 5	Profile Number 6
1	4	6	2.5	8	4	7
2	10	9	8	14	20	17
3	40	25	12	20	35	27
4	50	60	40	70	55	50