Introduction to MATLAB Tutorial

MATLAB, an acronym for **MAT**rix **LAB**oratory, is a high-performance matrix/array language that has been developed for technical computing and is used widely in mathematics, science, and engineering. Its power lies in its integration of computation, visualization, and programming. The MATLAB environment consists of tools for numerical modeling, mathematical analysis of large data sets, and data display. MATLAB graphics include 2-D and 3-D data visualization, image processing, animation, presentation graphics, and Graphical User Interfaces (GUIs).

When you open MATLAB, the primary workspace is the ***Command*** window. Command lines appear as follows:

```
>>
```

When the Command window is active, a blinking cursor appears to the right of this prompt, signaling that MATLAB is waiting to perform an operation. Other possible windows include Figure and Text Editor windows. Commands can be entered in a text file, called a script file or M-file, and saved in this format to be reopened and used over and over again as needed. Script file names must end with the extension '.m', as in 'GSAseacliff.m', an M-file that we will use in the Tectonics and Topography workshop.

MATLAB Concepts and Simple Math

The command window can used to do simple math in MATLAB, and unless specified otherwise, the answers are given as 'ans'.

```
»18+22

ans =

   40

»79-21

ans =

   58

»44*22

ans =

  968

»45/22
```

```
ans =

   2.0455

»22\45

ans =

   2.0455
```

All computations in MATLAB are done in double precision, but the FORMAT command can be used to switch among different output display formats, some of which are as follows:

FORMAT	Default. Same as SHORT.
FORMAT SHORT	Scaled fixed point format with 5 digits.
FORMAT LONG	Scaled fixed point format with 15 digits.
FORMAT SHORT E	Floating point format with 5 digits.
FORMAT LONG E	Floating point format with 15 digits.

As an example, change the format to floating point with 5 digits (short e):

```
»format short e

»45/22

ans =

  2.0455e+00
```

To return to the default format, type either

```
»format
```

or

```
»format short
```

A listing of all display formats can be found by typing the following at the command prompt:

»help format [Note: Use 'help xxx' to find out about xxx topics; e.g., help plot.]

The abbreviation for answer, 'ans', is actually a variable that is stored in the MATLAB memory after a given computation. The answer itself can then be renamed and/or used in other computations.

```
»90*pi/180

ans =
   1.5708
»sin(ans)
ans =
    1
»sin_90=ans
sin_90 =
    1
»asin(sin_90)
ans =
   1.5708
```

[Note: MATLAB only works in radians.]

Here are a few more examples of how to name variables:

```
»normal_faults=10
normal_faults =
   10
»strikeslip_faults=20
strikeslip_faults =
   20
»total_faults=normal_faults+strikeslip_faults
total_faults =
   30
```

Note that variable names are case sensitive, and have a limit of 32 characters.

Parentheses are used in the typical way so that the order of operations is clear:

```
»2*(sin(90)+sqrt(16))/7
ans =
   1.3983
```

You can edit expressions by pressing the Up arrow to recall the last line, and you can recall the string of recent commands by repeatedly pressing the Up and Down arrow keys. A handy shortcut is to type the first letter or number of a recent command and then the Up arrow key, and the command will be recopied to the new prompt line.

If you place a semicolon at the end of a command, it suppresses printing of the computed results.

```
»total_faults=normal_faults+strikeslip_faults;
```

```
»strikeslip_faults=20;
```

One other form of punctuation is important. The percent sign (%) indicates a comment statement. For example,

```
%total faults is the sum of normal and strike-slip faults
```

Vectors and Matrices

So far, all the computations have involved single numbers, or scalars. A single number is a one element vector. The power of MATLAB lies in its ability to work with more than one number at a time, using larger data arrays (i.e., vectors and matrices). A row or column of numbers is a row vector or a column vector, respectively. Brackets are used to identify arrays, as in this example for fault slip measurements.

```
»fault_slip=[6.2 5.5 5.8]

fault_slip =

   6.2000    5.5000    5.8000
```

Single spaces between each of the three numbers indicated separate values in the row vector (one row and 3 columns). Alternatively, you can use commas to separate values.

```
»fault_slip=[6.2,5.5,5.8]

fault_slip =

   6.2000    5.5000    5.8000
```

To make a column vector, use semi-colons rather than spaces or commas.

```
»fault_slip=[6.2;5.5;5.8]

fault_slip =

   6.2000
   5.5000
   5.8000
```

Row vectors are the transpose of column vectors (and vice versa), and can be converted with the transpose command or the transpose operator ('):

```
»Newfault_slip=transpose(fault_slip)

Newfault_slip =

   6.2000    5.5000    5.8000
```

OR

```
»fault_slip=Newfault_slip'

fault_slip =

   6.2000
   5.5000
   5.8000
```

A set of *i* row vectors of length *j* forms the *i x j* matrix (a matrix with *i* rows and *j* columns). Matrix assignments can be made as in this example of a *3 x 4* matrix:

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

myMatrix =

    1     2     3     4
    4     5     6     7
    1     2     3     4
```

The value in a given position, as in the 2nd row and 3rd column, can be referenced by its indices:

```
»myMatrix(2,3)

ans =

    6
```

To reference an entire column or row, you can use a wildcard symbol (the colon):

```
»myMatrix(:,2)

ans =

    2

    5

    2

»myMatrix(3,:)
```

```
ans =

    1    2    3    4
```

MATLAB has some simple built-in functions to make incremental vectors. One way is to use the colon sequence **start:increment:end**.

```
>> A=1:5:25
```

which yields

```
A =

    1    6   11   16   21
```

Matrix Algebra

When arrays have the same dimensions, they can be added, subtracted, multiplied, and divided on an element-by-element basis:

```
»myMatrix

myMatrix =

    1    2    3    4
    4    5    6    7
    1    2    3    4

»newMatrix =[3 3 3 3;5 5 5 5;2 2 2 2]

newMatrix =

    3    3    3    3

    5    5    5    5

    2    2    2    2

»add_matrices=myMatrix+newMatrix

add_matrices =

    4    5    6    7

    9   10   11   12

    3    4    5    6

»minus_matrices=myMatrix-newMatrix

minus_matrices =
   -2   -1    0    1

   -1    0    1    2

   -1    0    1    2
```

Element-by-element multiplication and division use the dot-product notation '.*' and '.\', respectively, as follows:

```
»multiply_matrices=myMatrix.*newMatrix

multiply_matrices =

     3     6     9    12

    20    25    30    35

     2     4     6     8

»divide_matrices=myMatrix.\newMatrix

divide_matrices =

    3.0000    1.5000    1.0000    0.7500

    1.2500    1.0000    0.8333    0.7143

    2.0000    1.0000    0.6667    0.5000
```

The dot product on two 2x2 matrices performs the following computations:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} \times b_{11} & a_{12} \times b_{12} \\ a_{21} \times b_{21} & a_{22} \times b_{22} \end{bmatrix}$$

Without the dot, the multiplication of matrices is a cross product, resulting in actual matrix multiplication and division.

```
»A=[1 2;3 4]

A =

     1     2

     3     4

»B=[5 6;10 2]

B =
     5     6
    10     2
```

```
»C=A*B
```

```
C =

    25    10
    55    26
```

At any time, you can see a list of variables stored in your workspace by typing the 'who' command:

```
»who
```

```
Your variables are:

A                   divide_matrices       normal_faults
B                   fault_slip            sin90
C                   minus_matrices        sin_90
Newfault_slip       multiply_matrices     strikeslip_faults
add_matrices        myMatrix              total_faults
ans                 newMatrix
```

MATLAB Graphics

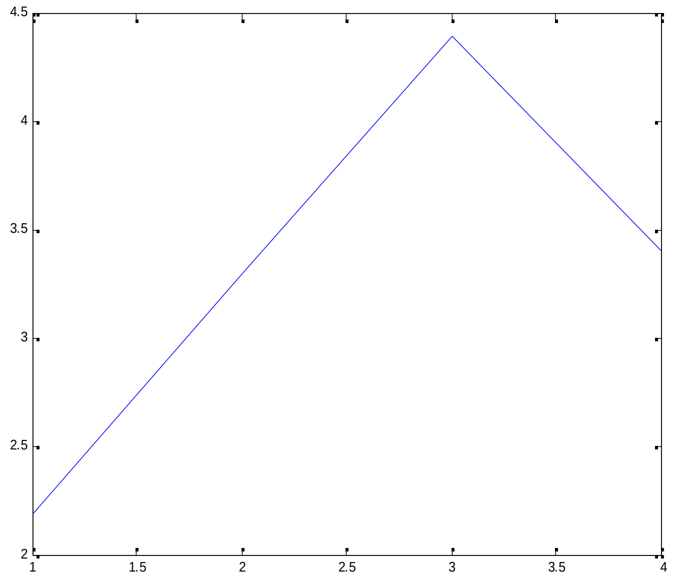
MATLAB has extensive facilities to display vectors and matrices as 1-, 2-, and 3-dimensional plots. Common plots used in the Earth Sciences include two-dimensional line and point plots, two-dimensional contour plots, and three-dimensional mesh plots.

Two-Dimensional Plots

The basic two-dimensional plot can be created with the `plot()` command. For example,

```
>>Fault_slip=[2.2 3.3 4.4 3.4];

>>plot(Fault_slip)
```

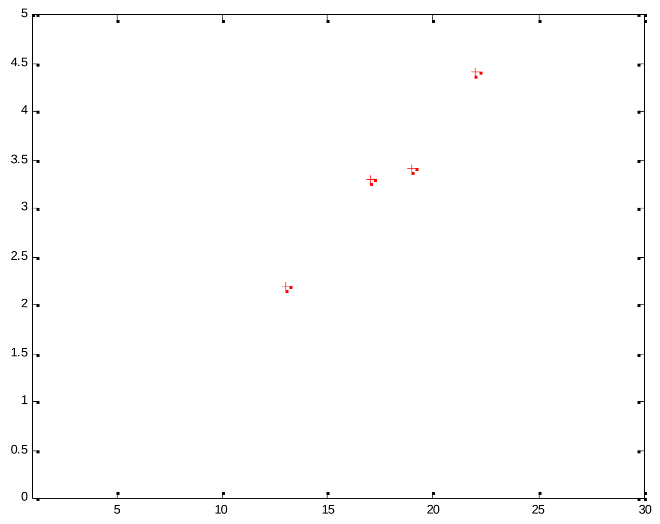
produces a piecewise linear graph of the elements of Fault_slip (a vector) versus the index of the elements of y (i.e.,1, 2, 3, 4). Data points are connected with lines by default.



PLOT(X,Y) plots vector Y versus vector X, as in this example:

```
>> Fault_length=[13 17 22 19];

>>plot(Fault_length,Fault_slip,'r+')

>> axis([1 30 0 5])
```

The final string of characters, ‘r+’ in this case, defines the data point color (e.g., red) and symbol (+), and the axis command line determines the XMin, XMax, YMin, and YMax, in that order.



Various line types, plot symbols and colors may be obtained with PLOT(X,Y,S) where S is a character string made from one element from any or all the following 3 colunms:

COLOR		SYMBOL		LINE	
y	yellow	.	point	-	solid
m	magenta	o	circle	:	dotted
c	cyan	x	x-mark	-.	dashdot
r	red	+	plus	--	dashed
g	green	*	star		
b	blue	s	square		
w	white	d	diamond		
k	black	v	triangle (down)		
		^	triangle (up)		
		<	triangle (left)		

For example, PLOT(X,Y,'c+:') plots a cyan dotted line with a plus at each data point; PLOT(X,Y,'bd') plots blue diamonds at each data point but does not draw any line.

Labels and a title can be added with the 'xlabel' , 'ylabel', and 'title' commands, as follows:

```
»xlabel(‘Length of normal fault, km’)
```

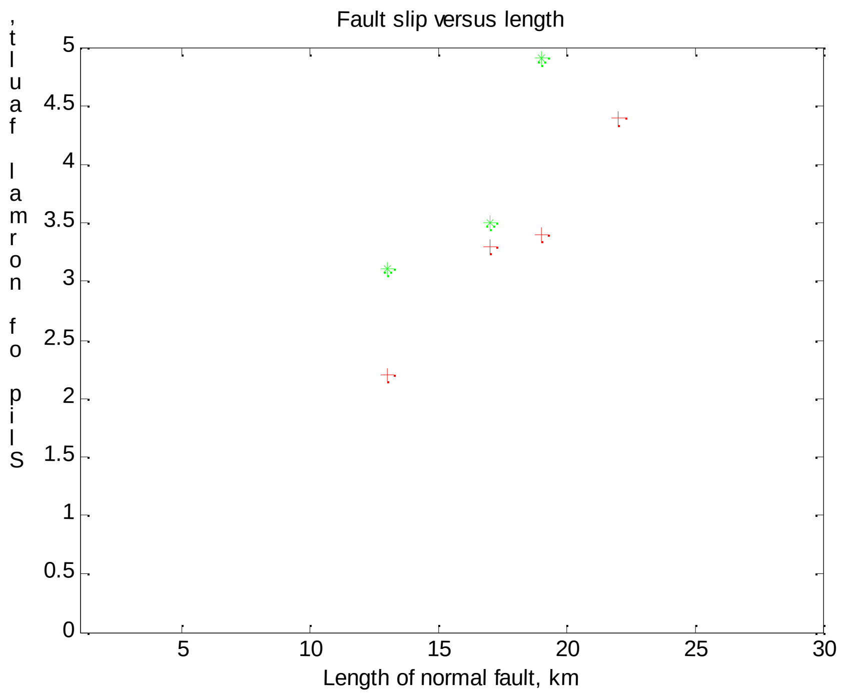
```
»ylabel('Slip of normal fault, m')
```

```
»title('Fault slip versus length’)
```

Note also that the axes can be squared relative to one another with the 'axis square' command.

You can overlay plots with the ‘hold on’ command at the Matlab prompt. The hold is undone by a ‘hold off’ command. For example,

```
>> hold on
>> NewFault_slip=[3.1 3.5 5.4 4.9];
>> plot(Fault_length,NewFault_slip,'g*')
```

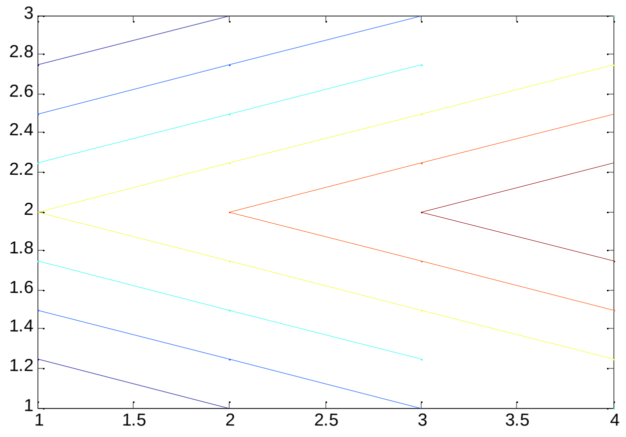


2-D Contour Plots

Before continuing with the MATLAB tutorial, type 'hold off' at the command line or 'clf' to clear the current figure image. CONTOUR(Z) is a contour plot of matrix Z treating the values in Z as heights above a plane. For example,

```
>> ContourMatrix=[1 2 3 4;5 6 7 8;1 2 3 4]

ContourMatrix =
    1    2    3    4
    5    6    7    8
    1    2    3    4
>> contour(ContourMatrix)
```

produces the following plot.



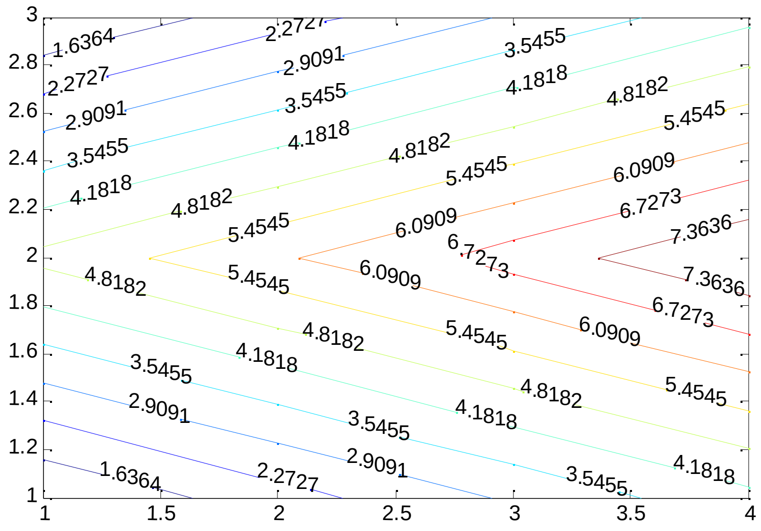
Handle graphics are used to refer to objects within a graph.

[C,H] = CONTOUR(...) returns contour matrix C as described in
CONTOURC and a column vector H of handles to LINE or PATCH
objects, one handle per line. Both of these can be used as
input to CLABEL to label contour values.

```
>>[c,h]=contour(ContourMatrix,10)
```
results in 10 contour lines.

The `clabel` command takes the contour plot's handles as arguments and will automatically label the lines:

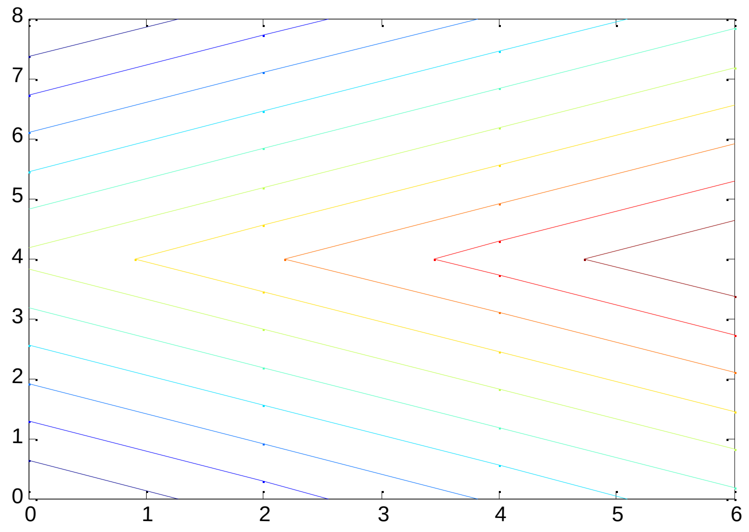
```
>>clabel(c,h);
```



MATLAB also can graph three variables as a contour plot. Two independent variables are plotted along the x- and y-axes with a resultant z-value. You can specify the size of each of the elements of a given matrix by defining a vector that contains the X-values for each column in the Z-matrix, and a vector that contains the Y-values for each row in the Z-matrix. For our example, let's assume that X-values are spaced 2 units apart, and Y-values 4 units apart. We'll use the incremental notation with colons to create the X and Y vectors.

```
>>X=0:2:6;

>>Y=0:4:8;

>>[c,h] = contour(X,Y,ContourMatrix,10)
```

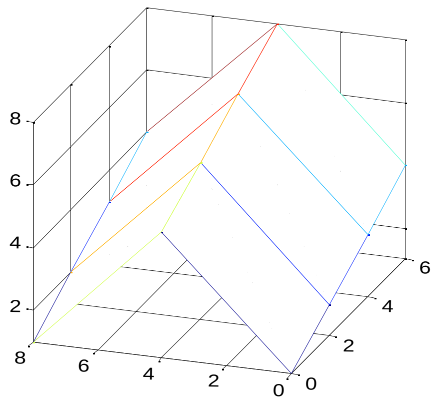
produces the following plot:



3-D Mesh and Surface plots

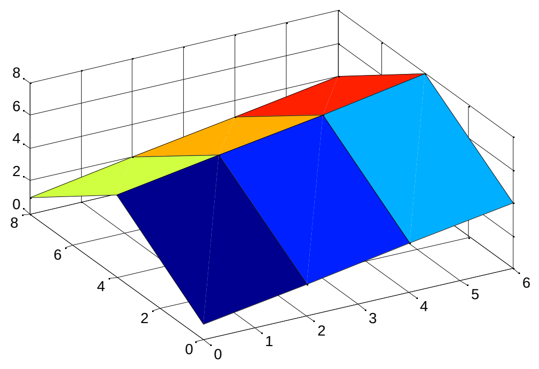
MATLAB can create other types of 3D plots, including mesh, surf, and surfl.

```
>> mesh(X,Y,ContourMatrix)
```

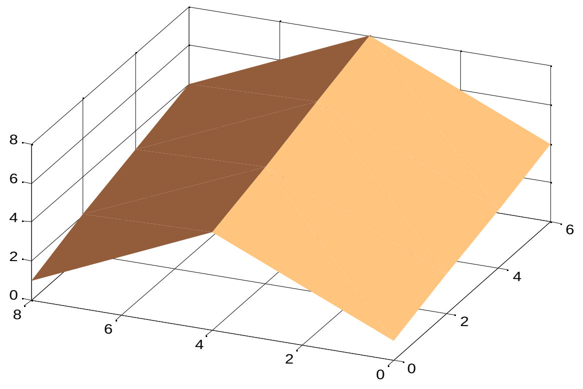


Note: The Rotate 3D option in the Tools menu of the figure window was used to spin this diagram to the current view. It also is possible to set the view azimuth and elevation angles with a command, as in >> view(45,45).

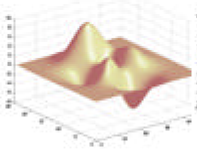
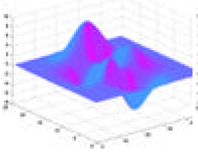
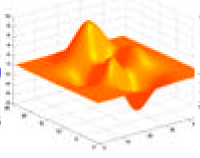
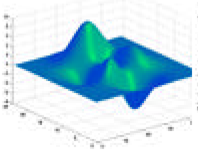
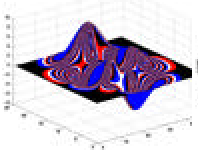
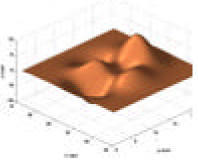
```
>> surf(X,Y,ContourMatrix)
```



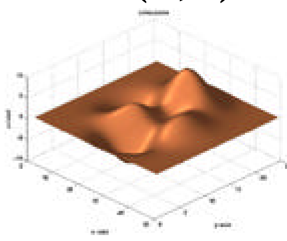
```
The commands
      >> surfl(X,Y,ContourMatrix)
      >> hold on
      >> shading interp
      >> colormap copper

produce the following:
```



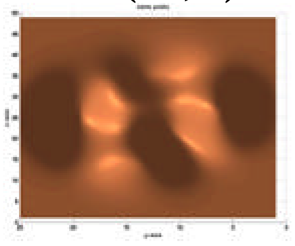
The colormap command represents color values (array of RGB values). Here are a number of built-in MATLAB colormaps, followed by 3 different view angles/elevations:

colormap pink	**colormap cool**	**colormap autumn**	**colormap winter**	**colormap flag**	**colormap copper**
					

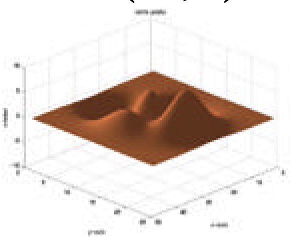
view(45,45)



view(270,90)



view(135,35)



Gridding

In many cases in the earth sciences, the data are irregularly spaced and must be gridded before they can be plotted as mesh or surface diagrams. The MathWorks Web site provides help on how to use MATLAB, and includes a list of questions/problems with answers. One of these is the question "How do I grid vector data?" This site (http://www.mathworks.com/support/tech-notes/1200/1212.shtml), which I will call the "Gridding Tutorial", provides instructions for different ways of gridding data. The text below is from the final example of the Gridding Tutorial. After reading through the text, you can try a real example with a data file included on the GSA WorkshopCD.

Data Stored in Sample.dat file

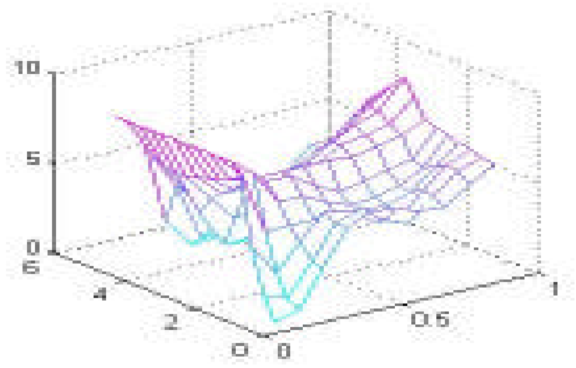
X	Y	Z		X	Y	Z
0.1	0.5	0		0.5	3	5
0.1	1	2		0.5	4.5	0
0.1	1.5	8		0.5	5	0
0.1	5	8		0.6	1	3
0.2	0.5	0.5		0.6	2.5	6
0.2	1	1		0.6	3.5	5
0.2	1.5	4		0.6	4	3
0.2	2.5	6		0.6	5	1
0.2	4	5		0.7	0.5	4
0.2	5	7		0.7	1.5	4
0.3	0.5	3		0.7	3	7
0.3	2	6		0.7	3.5	6
0.3	3.5	4		0.7	4.5	3
0.3	4	2		0.8	1	6
0.3	5	0		0.8	2	5
0.4	0.5	5		0.8	3	7
0.4	1	5		0.8	5	4
0.4	4	0		0.9	0.5	6
0.4	4.5	1		0.9	2.5	7
0.5	0.5	5		0.9	3	9
0.5	1.5	4		0.9	4	7
0.5	2	4		0.9	5	4

The following M-file demonstrates how to grid the data:

```
 % Load the data (file called sample.dat)and extract the X, Y,
% and Z information

 load sample.dat
 X = sample(:,1); %column 1
 Y = sample(:,2); %column 2
 Z = sample(:,3); %column 3
```

```
% Determine the minimum and maximum X and Y values
minX = min(X);minY = min(Y);
maxX = max(X); maxY = max(Y);

% Define the density of the grid
m = 15; %number of rows
n = 10; % number of columns

% Grid X and Y (make an X by Y grid, called [Xi,Yi]
x1 = linspace(minX,maxX,n);
y1 = linspace(minY,maxY,m);
[Xi,Yi] = meshgrid(x1,y1);

% Map Z on to the grid
Zi = griddata(X,Y,Z,Xi,Yi);

% Generate the mesh plot.  Contour can also be used.
mesh(x1,y1,Zi)
colormap(cool(8))
```

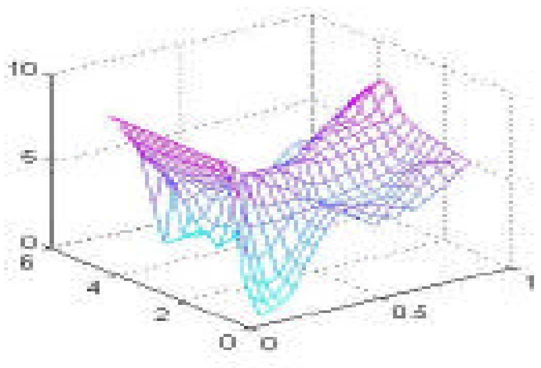
The mesh plot looks like this:



By mapping the original data on to a 15x10 grid, we produce a fairly smooth plot. To create a smoother surface plot, simply increase the density of the grid. For example, the following 20 by 20 grid is used to produce the next plot:

```
% Define the density of the grid
m = 20;
n = 20;

% Grid X and Y
x1 = linspace(minX,maxX,n);
y1 = linspace(minY,maxY,m);
[Xi,Yi] = meshgrid(x1,y1);

% Map Z on to the grid
Zi = griddata(X,Y,Z,Xi,Yi);

% Generate the mesh plot
mesh (x1,y1,Zi)
```

```
colormap(cool(8))
```



MATLAB GRIDDING EXERCISE

Background to Geologic Problem:
Deformation at the Northern Tip of the San Andreas Fault

The Quaternary geology of the Point Delgada area (photo above) at the northern termination of the San Andreas fault (SAF) was mapped previously by *McLaughlin et al*. (1983). With new survey data and a recently established onshore location of the SAF, however, we interpret the ages of wave-cut platforms and the deformation history of this area differently than previous workers. In particular, we conclude that alluvial deposits overlying the wave-cut platform at Point Delgada are parts of alluvial fans that have been abandoned and offset progressively greater distances from their source for at least 80 to 100 ky. These conclusions indicate that the SAF is not only onshore at Point Delgada, but also that the onshore trace has been active for tens to perhaps hundreds of thousands of years.

Emergent wave-cut platforms are valuable datum planes for estimating amounts and rates of surface uplift. At the time of platform formation, the altitude of the inner edge of the platform--also known as the shoreline angle--coincides closely with that of mean sea level. The trace of the inner edge, or strandline, is approximately horizontal relative to the Earth's geoid. This inner-edge trace provides a datum for estimating subsequent uplift and tilt [c.f., *Ota,* 1986; *Lajoie*, 1986; *Merritts and Bull*, 1989; and *Merritts.*, 1996].

Surveying and Mapping Emergent Marine Platforms

In June-July of 1995, we surveyed a prominent late-Pleistocene platform(s) from its location several hundred meters west of the SAF at Shelter Cove, westward to Point Delgada and then northward for a distance of 2 km, where the coastline is 1.4 km west of the fault. Our survey included 1) multiple points on the Pleistocene platform(s) and overlying marine, and fluvial deposits; 2) points at each radiocarbon sample site; and 3) frequent points at the best-estimate of sea level within the wave run-up zone. In addition, we surveyed the surface of a nearly continuous late-Holocene platform from its first appearance about 7 km north of Point Delgada to a distance 6 km farther to the northwest. This platform emerges above sea level 3 km due west of Saddle Mountain (notch in upper left of above photo), where we are no longer able to find clear evidence of a possible trace of the SAF.

Over the combined areas, about 13 km of coast were surveyed, with high concentrations of survey points in areas of exposed platform remnants. Equipment used was a Lietz total geodetic station (angular accuracy ±3" at one standard deviation; linear accuracy ±(5mm + 3ppm) at one standard deviation). To minimize random survey errors and close each survey loop, at least three temporary benchmarks were surveyed and resurveyed prior to and after each time the total station was moved. At each station set up, horizontal and vertical errors were less than ±4 and ±5 cm, respectively. After returning from the field, we further minimized horizontal error at each station set-up through angular rotation of benchmark data that were resurveyed between two different set-ups. The value of angular rotation was that necessary to minimize the sum-of-squares distances between the two sets of benchmark data. We then applied the appropriate corrections to all other (non-benchmark) survey data. We checked the horizontal accuracy of our survey results by plotting the data at the same scale as digitized coastlines from USGS 1:24,000 topographic maps. To minimize vertical error between two different station set-ups, we first determined the difference (positive or negative) in average elevation of all temporary benchmarks between each set-up. We then added this amount to all points surveyed after the new instrument station set-up.

Tidal charts were used to correlate actual sea levels to surveyed sea-level altitudes estimated from wave run-up. Points were surveyed frequently in the wave zone, and the exact time of day was recorded in order to calibrate the survey data to tidal chart levels. Regression analysis of tidal chart altitudes versus surveyed altitudes for all estimated sea-level positions was used to obtain a best-fit correction factor, and survey altitudes were converted to absolute altitudes by subtracting this value from all numbers.

Rotation of the Late-Pleistocene Wave-Cut Platform Into the San Andreas Fault Plane

The geometry of the wave-cut platform(s) at Point Delgada indicates that it does not dip gently seaward (westward in this case), as do all wave-cut platforms at the time of formation; instead, it has been rotated eastward, so that it tilts into the plane of the SAF. We surveyed about 330 points along the fringe of bedrock platform rimming the point (see photo above), then plotted the data as a mesh grid and performed a first-order trend surface analysis on it . The polynomial function for the trend surface analysis is

$$Z = -20.17486 - 0.00137*(\text{East}) + 0.00352*(\text{North})$$

with Z, East, and North representing survey data for elevation and horizontal coordinates. The direction of maximum slope for the first-order surface is S22E. The tilt of the platform is 3.8 m/km to the southeast along a S22E direction.

Rotation of the wave-cut platform from a west-dipping, or seaward, direction to an east-dipping direction is consistent with multiple lines of evidence of up-to-the-west motion for the SAF at Point Delgada (c.f., field observations of Matthes as described in *Lawson*, 1908; also, *Brown*, 1995; and *Prentice et al*., 1999). In essence, the wave-cut surface reveals a large-amplitude drag fold, similar to smaller-amplitude drag folds that we examined in the rock fabric immediately west of the fault plane. In addition, we consider the direction of tilt to be similar to a passive marker, in that its counterclockwise orientation (N22W-S22E) with respect to the trend of the SAF (N11W-S11E) is consistent with dextral shear.

REFERENCES

Lajoie, K. R., Coastal Tectonics, *in* Wallace, R., panel chairman, Active Tectonics: Studies in Geophysics Series, Geophysics Research Forum, National Academy Press, 95-124, 1986.

Lawson, A. C., The California earthquake of April 18, 1906, Report of the State Earthquake Investigation Commission, Carnegie Institute of Washington, 1908.

McLaughlin, R. J., Lajoie, K. R., Sorg, D. H., Morrison, S. D., and Wolfe, J. A., Tectonic uplift of a middle Wisconsin marine platform near the Mendocino triple junction, California, Geology, 11, 35-39, 1983.

Merritts, D. J., and W. B. Bull, Interpreting Quaternary uplift rates at the Mendocino triple junction, northern California, from uplifted marine terraces, Geology, 17, 1020-1024, 1989.

Merritts, D. J., The Mendocino triple junction: Active faults, episodic coastal emergence, and rapid uplift, J. Geophys. Res.,101, 6051-6070, 1996.

Ota, Yoko, Marine terraces as reference surfaces in late Quaternary tectonics studies: Examples from the Pacific Rim, Royal Society of New Zealand Bulletin, 24, 357-375, 1986.

Prentice, C.P., Merritts, D. J., Beutner, E. C., Bodin, P., Schill, A., and Muller, J.R., Northern San Andreas fault near Shelter Cove, California, 111, 4, 512-523, 1999.

Analyzing the Wave-Cut Platform Survey Data with MATLAB

To try the Point Delgada gridding/trend surface analysis yourself, go to MATLAB and open the trend.m script file (Open is an option under the File pull-down menu). This file is slightly modified from an m-file (i.e., script file) presented in Middleton (2000). The script determines the function b = trend(X,m,lab,ngrid), with

X=name of matrix;
m=polynomial order;
lab=label contours;
ngrid = number of grid nodes for x and y directions

First, set the MATLAB working directory to

GSA_MatlabFiles\MtlbGridExercs\m-files.

Second, load the survey data using import wizard—the data file name is AllPlatsTab.dat.
(Import Data is an option under the File pull-down menu—follow the simple steps of the wizard)

The survey data matrix has 325 rows and 3 columns (325 x 3 matrix). Columns 1, 2, and 3 equal East, North, and Elevation, respectively.

To look at a map view of the survey data, make a 2-D plot as follows.

```
>>plot(AllPlatsTab(:,1),AllPlatsTab(:,2),'r.')
```

At the Matlab command prompt, create a variable for the matrix of 3 columns of coordinate data as follows:

```
>> X=[AllPlatsTab(:,1),AllPlatsTab(:,2),AllPlatsTab(:,3)];
```

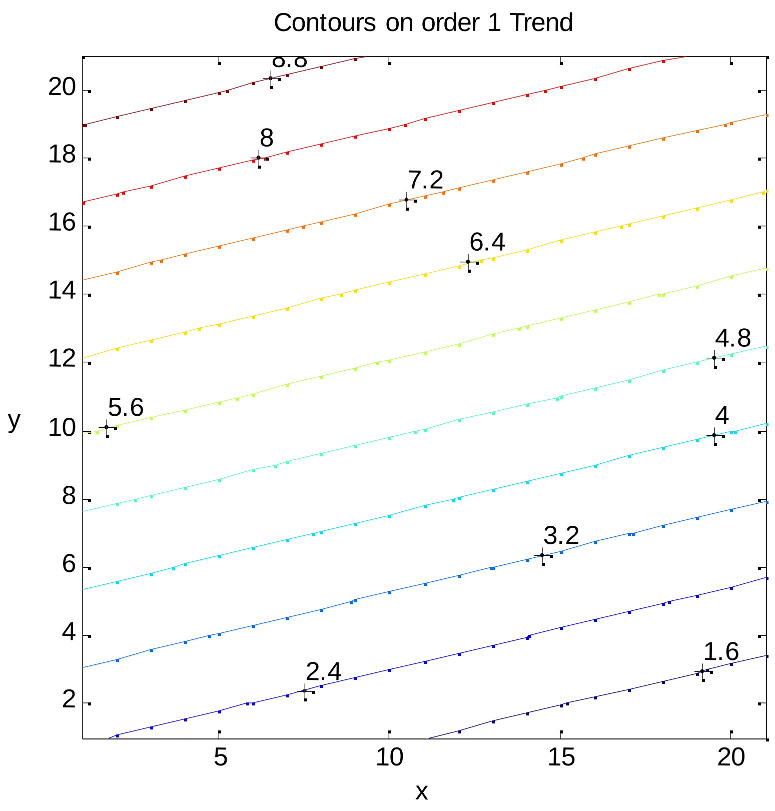
Do not include the semi-colon at the end of the command line if you want to see the matrix displayed.

Third, to use the trend surface analysis, type the following command:

```
>> b=trend(X,1,1,20)
```

(for a first-order polynomial surface with labeled contours and 20 nodes).

Keep pressing the Enter key as the script runs through the analysis and prompts you. When the script is done, you should have 3 figures: a contour map of the data; a contoured fitted (polynomial) surface; and a map of the residuals from the fitted surface. Here is the fitted surface, with labeled contours:

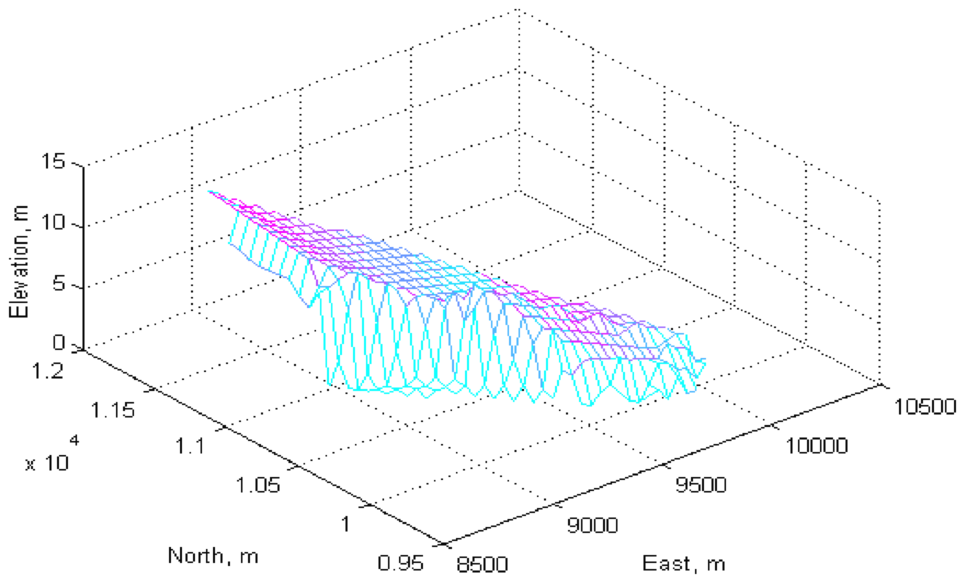


Note that the slope of the fitted surface is to the southeast (S22E).

Finally, to make a grid surface of the survey data, do the following.

```
% Define the density of the grid
>> m=30;n=30;

%Name variables for each vector
>> E=AllPlatsTab(:,1);
>> N=AllPlatsTab(:,2);
>> Z=AllPlatsTab(:,3);

%Determine min and max values
>>minE=min(E);maxE=max(E);
>>minN=min(N);maxN=max(N);

% Grid X and Y (i.e.,East and North)
>>x1 = linspace(minE,maxE,n);
>>y1 = linspace(minN,maxN,m);
>>[Xi,Yi] = meshgrid(x1,y1);

 % Map Z on to the grid
>>Zi = griddata(E,N,Z,Xi,Yi);
```

```
% Generate the mesh plot
>>mesh (x1,y1,Zi)
```



The above view has been rotated to give a similar orientation to the photo of Point Delgada above. To include the raw data with the grid, do the following:

```
>>hold on
>>plot3(E,N,Z,’ko’);
```