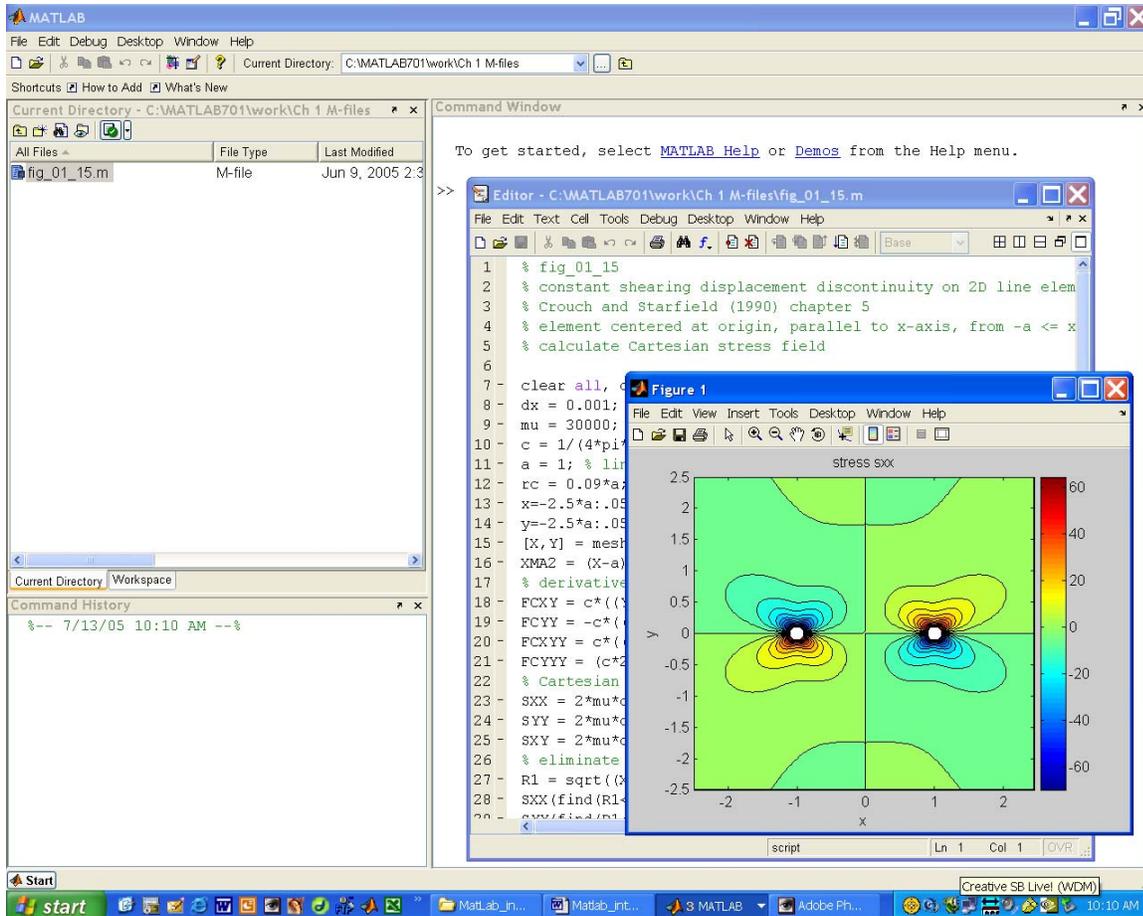


## Exercise: introduction to MATLAB

The following is a brief introduction to a few of the elementary features and functions of MATLAB designed to get a student started along the learning path. It is recommended that you purchase a book that offers a tutorial and reference guide to MATLAB such as:

Mastering MATLAB, 2012, Hanselman, D.C. and Littlefield, B., Pearson Education, Inc. (Prentice Hall), 864p.



### 1 What is it?

MATLAB stands for Matrix Laboratory. MATLAB is a programming tool that provides an interactive environment for numerical and symbolic computations along with a wide variety of data analysis and visualization tools.

### 2 History

MATLAB was conceived in the 1970s by Cleve Moler, who wanted to make several FORTRAN matrix libraries more accessible to his students at UNM. Thus he wrote MATLAB. In 1983 Moler joined forces with John Little (Stanford University) and others to write a professional version

with graphics functionality. In 1984 The MathWorks Inc. was founded and MATLAB has been evolving and improving rapidly since then.

Academic discounts for faculty are available for individual, group, concurrent, and classroom licenses from The MathWorks Inc. at:

<http://www.mathworks.com/>

Discounts also are available for students.

### **3 MATLAB structure**

#### *3.1 MATLAB core libraries*

These are the main programs, functions, and commands used in MATLAB.

#### *3.2 Toolboxes*

These are special purpose libraries of programs which are sold and licensed separately by The MathWorks Inc. and include:

Statistics Toolbox

Symbolic Math Toolbox

Partial Differential Equation Toolbox

Signal Processing Toolbox

Image Processing Toolbox

and many more.

#### *3.3 The User*

There are two ways the user can interact with MATLAB:

1. Type into the MATLAB command window
2. Write your own programs using script and function m-files.

### **4 Using MATLAB**

Open MATLAB on your computer and study the workspace that is provided.

#### *4.1 At first glance*

The default appearance is three windows in a graphical user interface (GUI):

Command Window

Workspace

Command History

You can arrange these however you want and you only need to have the Command Window open. It is convenient to have the Current Directory window open. If you intend to call variables or data files stored in a certain directory, that directory must be open as the Current Directory.

Note the Start button in the lower left corner of the MATLAB window. This is used to start Demos, Help, Desktop Tools, MATLAB toolboxes, and other primary functions of MATLAB.

The symbol `>>` is the standard prompt in the MATLAB Command Window. A good way to begin learning MATLAB is to start typing to the right of the prompt and read the response. There is an extensive help function that provides information about commands. For example you can learn about the general commands and functions in MATLAB by typing:

```
>> help general
```

The response has been suppressed for brevity here. Or you can learn about the exponential function by typing:

```
>> help exp
```

EXP Exponential.

EXP(X) is the exponential of the elements of X, e to the X.

For complex  $Z=X+i*Y$ ,  $EXP(Z) = EXP(X)*(COS(Y)+i*SIN(Y))$ .

See also expm1, log, log10, expm, expint.

Overloaded functions or methods (ones with the same name in other directories)

```
help sym/exp.m
```

Reference page in Help browser

```
doc exp
```

```
>>
```

To get the scoop on the help function itself you would type:

```
>> help help
```

The response has been suppressed for brevity here. Selecting MATLAB Help under Help in the menu bar at the top of the GUI Window provides an abundance of helpful tips. Also available on the Help menu are:

- Using the Desktop

- Using the Command Window

- Demos

After finishing this introduction have a look at these features to extend your knowledge of MATLAB.

## 4.2 Constants

MATLAB recognizes many commonly used constants, such as:

- pi  $\pi$ , the ratio circumference/diameter of a circle

- nan Not-A-Number, the ratio 0/0

- eps the smallest number such that  $1+eps > 1$   
the double precision accuracy of your computer  
usually smaller than  $2^{-16}$

The values of pi and eps are displayed with 4 digits after the decimal point. To increase the number of digits displayed to 14 type:

```
>> format long
```

Look again at pi and eps. To return to the standard precision type:

```
>> format short
```

The command format does not affect how MATLAB computations are done. Computations on floating point variables are done in the appropriate floating point precision, no matter how those variables are displayed.

### 4.3 Variables

Variables are broadly classified in MATLAB as scalars (with a single numerical value), vectors (with more than one value organized in one dimension) and arrays (with multiple numerical values organized in more than one dimension).

Scalars are defined as:

```
>> a=6
```

```
a =
```

```
6
```

Or, one could define the scalar b as:

```
>> b=32;
```

Note that ending a command line with the semi-colon suppresses the response that gives the value of the scalar. Of course scalars can be thought of as vectors of length one.

Row and column vectors are defined as:

```
>> r=[45,33,27,48]
```

```
r =
```

```
45 33 27 48
```

```
>> c=[45;33;27;48]
```

```
c =
```

```
45  
33  
27  
48
```

Note the use of the comma to separate elements in the definition of the row vector and the semi-colon to separate elements in the definition of the column vector. The row vector can be defined using a space between each element instead of the comma.

Matrices are two dimensional arrays. For example, the matrix A is defined:

```
>> A=[11 12 13;21 22 23;31 32 33; 41 42 43]
```

A =

```
11 12 13
21 22 23
31 32 33
41 42 43
```

Note the elements of a given row are separated from those of the next row by a semi-colon. Also note the use of lower case names for scalars and vectors, and upper case names for matrices. This is not a necessary syntax, but some convention to help distinguish arrays of more than one dimension can be helpful. The general notion of arrays continues to three or more dimensions.

There are many useful functions for managing vectors and arrays. These can be reviewed by typing:

```
>> help elmat
```

For example, the function length returns the number of values in a vector:

```
>> e=length(r)
```

e =

```
4
```

The function ndims returns the number of dimensions of an array:

```
>> f=ndims(A)
```

f =

```
2
```

The function size returns the number of rows and columns of an array:

```
>> g=size(A)
```

g =

```
4 3
```

#### *4.4 Indexing*

Most of the data used in MATLAB consists of vectors and arrays and at times only a subset of numbers are of interest. This is where indexing becomes important. One of the most useful operators in this context is the colon. For example, to create a row vector containing the integers from 6 to 12 we write:

```
>> 6:12
```

```
ans =
```

```
6 7 8 9 10 11 12
```

An increment other than one can be specified as follows:

```
>> 6:2:12
```

```
ans =
```

```
6 8 10 12
```

For example, if only the first 4 values of a vector,  $x$ , of length 8 are needed in a calculation the subset,  $x_{\text{sub}}$ , may be defined using the colon operator:

```
>> x=[21 76 93 44 57 25 11 98];
```

```
>> xsub=x(1:4)
```

```
xsub =
```

```
21 76 93 44
```

Recall the matrix  $A$  and use the colon by itself to specify all of the elements in column 2:

```
>> A
```

```
A =
```

```
11 12 13  
21 22 23  
31 32 33  
41 42 43
```

```
>> A(:,2)
```

```
ans =
```

```
12  
22
```

32  
42

What would you expect as a result for A(:,:)?

If only the values in rows 2 through 3 and in columns 1 through 2 are needed from the matrix A that has 4 rows and 3 columns, the subset, ASUB, is defined:

```
>> ASUB=A(2:3,1:2)
```

ASUB =

```
21 22  
31 32
```

To select an interval through the last value in the rows and in the columns use, for example:

```
>> ASUB2=A(2:end,3:end)
```

ASUB2 =

```
23  
33  
43
```

The find() command evaluates any mathematical or logical expression, which is entered in the parentheses, and returns the indices of an array of numbers for which the expression is true. For example, to find the indices of values of x greater than 50 use:

```
>> xbig=find(x>50)
```

xbig =

```
2 3 5 8
```

Note that it is the indices returned and not the values of the elements with those indices. To find the values use:

```
>> x(xbig)
```

ans =

```
76 93 57 98
```

For loops repeat a calculation a certain number of times. For example, let's multiply 2 times the 8 elements of the vector x. The For loop begins by setting a counter, i, that varies over the

required range, here 1 to 8, and ends with the statement end. Between the For and the end you write the calculations with the particular value of x designated as x(i).

```
>> for i=1:length(x)
x(i)=2*x(i);
end
>> x
```

x =

42 152 186 88 114 50 22 196

#### 4.5 Mathematical functions

Elementary math functions may be reviewed by typing:

```
>> help elfun
```

MATLAB provides an extensive set of mathematical functions including all of the standard arithmetic, trigonometric, exponential, complex, and specialized functions. For example consider the trigonometric equation:

$$t = \sin(x)\cos(x) \quad (1)$$

In MATLAB this can be written for the scalar x as follows given an angle of 45 degrees:

```
>> x=45*pi/180;
>> t=sin(x)*cos(x)
```

t =

0.5000

Note the angle 45 (degrees) is converted to radians because trigonometric functions expect arguments as radians.

Now consider the following vector equation:

$$T_i = \sin(x_i)\cos(x_i), \quad i = 1, n \quad (2)$$

For the vector  $x_i$  with  $n = 4$  uniformly spaced values from 0 to pi, this function is evaluated:

```
>> X=linspace(0,pi,4);
>> T=sin(X).*cos(X)
```

T =

0 0.4330 -0.4330 -0.0000

Note the multiplication operator `.*` which is used to multiply each element of a vector by the corresponding element of a vector of the same length. This *times* operator also is used for element-by-element multiplication of matrices. If A is a matrix of order [m by n] and B is a matrix of order [m by n], then `A .* B` produces a matrix that also is of order [m by n] as follows:

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & \dots & \dots & A_{mn} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \dots & B_{1n} \\ B_{21} & B_{22} & \dots & B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & \dots & \dots & B_{mn} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} & A_{12}B_{12} & \dots & A_{1n}B_{1n} \\ A_{21}B_{21} & A_{22}B_{22} & \dots & A_{2n}B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1}B_{m1} & \dots & \dots & A_{mn}B_{mn} \end{bmatrix} \quad (3)$$

In contrast, the multiplication operator `*` is used to multiply a matrix by a vector, or a matrix by a matrix, using the rules of linear algebra, as we will see below.

#### 4.6 Linear algebra

Elementary matrix functions may be reviewed by typing:

```
>> help matfun
```

For example, consider matrix-vector multiplication of an [m by n] matrix, B, by an [n by 1] column vector, x, using the *mtimes* operator `*` that produces an [m by 1] column vector:

$$\begin{bmatrix} B_{11} & B_{12} & \dots & B_{1n} \\ B_{21} & B_{22} & \dots & B_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & \dots & \dots & B_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} B_{11}x_1 + B_{12}x_2 + \dots + B_{1n}x_n \\ B_{21}x_1 + B_{22}x_2 + \dots + B_{2n}x_n \\ \vdots \\ B_{m1}x_1 + B_{m2}x_2 + \dots + B_{mn}x_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad (4)$$

This produces a column vector y of length m with elements that are the sum of the products of the elements from successive rows of B with the corresponding elements from the column of x. This is accomplished in MATLAB as follows:

```
>> B=[11 12 13;21 22 23;31 32 33];
>> x=[1;2;3];
>> y=B*x
```

y =

```
74
134
194
```

The *mtimes* operator enables matrix-matrix multiplication, for example  $A*B$ , if the order of  $A$  is  $[m \text{ by } p]$  and the order of  $B$  is  $[p \text{ by } n]$ . The number of columns in  $A$  must equal the number of rows in  $B$ , and the order of the resulting product matrix is  $[m \text{ by } n]$ .

One of the many useful linear algebra and matrix functions is `eig` which operates on a square matrix  $X$  with the following syntax:

```
[V,D] = eig(X)
```

This function produces a diagonal matrix  $D$  of eigenvalues and a full matrix  $V$  whose columns are the corresponding eigenvectors. This function has important applications in structural geology where it is used to operate on the stress or strain tensor. These tensors are  $3 \times 3$  symmetric matrices: the eigenvalues are the so-called principal values, and the orientations of the principal stresses or strains are given by the eigenvectors.

Suppose you are given the six independent components of a stress tensor in Cartesian coordinates. The principal values and directions are found as follows:

```
>> sxx = 2; syy = 1; szz = .5; sxy = .75; syz = .5; szx = .25;  
>> S = [sxx sxy szx; sxy syy syz; szx syz szz]
```

S =

```
2.0000  0.7500  0.2500  
0.7500  1.0000  0.5000  
0.2500  0.5000  0.5000
```

```
>> [V,D] = eig(S)
```

V =

```
-0.1361  0.5256 -0.8398  
0.5965 -0.6333 -0.4930  
-0.7909 -0.5681 -0.2273
```

D =

```
0.1659    0    0  
0  0.8261    0  
0    0  2.5080
```

Another useful linear algebra function is `meshgrid` which has the following syntax:

```
[X,Y] = meshgrid(x,y)
```

This function transforms the domain specified by vectors  $x$  and  $y$  into arrays  $X$  and  $Y$  that can be used for the evaluation of functions of two variables and the construction of 3-D surface plots. The rows of the output array  $X$  are copies of the vector  $x$  and the columns of the output array  $Y$  are copies of the vector  $y$ . For example:

```
>> x=[0 1 2 3];  
>> y=[0 1 2 3];  
>> [X,Y]=meshgrid(x,y)
```

$X =$

```
0  1  2  3  
0  1  2  3  
0  1  2  3  
0  1  2  3
```

$Y =$

```
0  0  0  0  
1  1  1  1  
2  2  2  2  
3  3  3  3
```

Taking the corresponding elements in the first rows of  $X$  and  $Y$  as pairs, defines the coordinates of points along the  $X$ -axis with  $X$  values of 0, 1, 2, and 3 respectively. The second rows define coordinates with the same  $X$  values along the line  $Y = 1$ , and so forth.

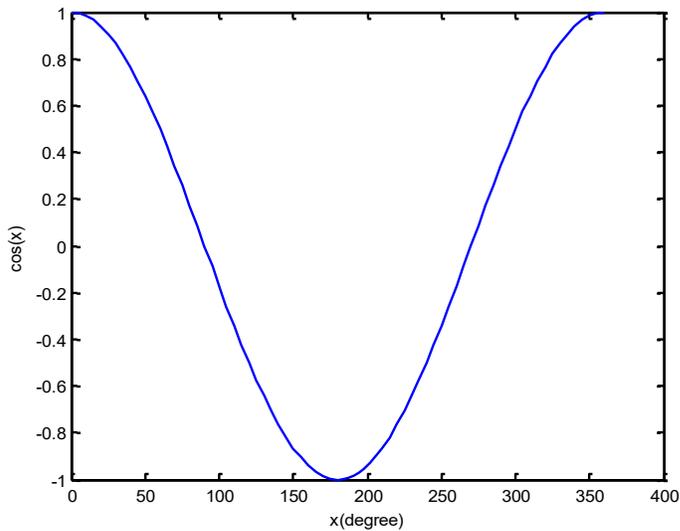
#### 4.7 Plotting in MATLAB

MATLAB has many built in plotting routines that provide outstanding visualizations of data.

For example, the cosine function can be plotted as follows:

```
>> x = [0:pi/36:2*pi];  
>> y = cos(x);  
>> plot(x*180/pi,y)  
>> xlabel('x(degree)'), ylabel('cos(x)')
```

Fundamentals of Structural Geology  
Exercise: introduction to MATLAB



The plot appears in a separate window called Figure 1. There are many ways to manipulate the plot to achieve a desired formatting.

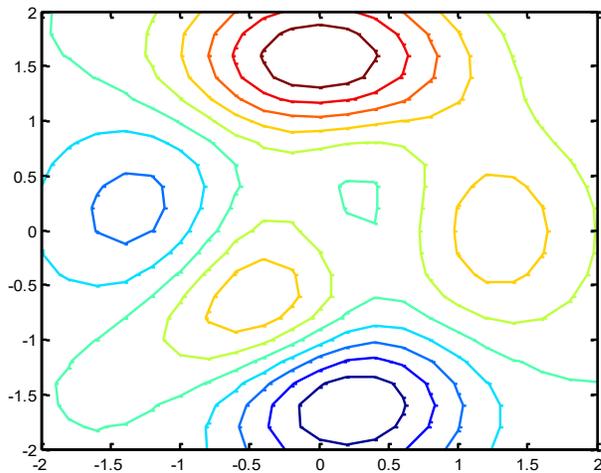
Three dimensional graphs are particularly useful and one can learn more about all the functionality for this kind of plotting in MATLAB by typing:

```
>> help graph3d
```

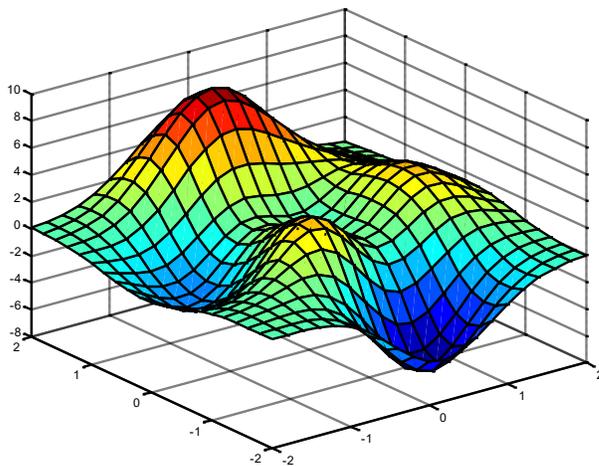
Plotting commands used in the exercises for the textbook describe, for example, stress fields or displacement fields. The following example illustrates how fields of data can be plotted as a contour map:

```
>> v = [-2:.2:2];  
>> [X,Y,Z] = peaks(v);  
>> contour(X,Y,Z,10)
```

The peaks function is a function of two variables, obtained by translating and scaling Gaussian distributions. It is useful for demonstrating MATLAB plotting routines. Type the lines of code given above and produce the following contour map of the peaks function for the range of parameter  $v$ .



Next try typing the function `surf(X,Y,Z)` in the Command Window to produce the following 3D visualization of the contour plot for the peaks function:



To see how MATLAB plots vectors try the following:

```
>> [U,V] = gradient(Z,0.2);  
>> hold on  
>> quiver(X,Y,U,V)
```

Further useful MATLAB graphical commands include:

<code>figure</code>	opens a new figure
<code>clf</code>	clears a figure
<code>hold on/off</code>	holds a plot, which admits several plotting commands in one figure
<code>colorbar</code>	adds a reference bar of colors used in a plot.

#### 4.8 Creating an m-file

It can become cumbersome to write a number of lines of code in the Command Window. Errors are not easily corrected and it is not easy to run the code multiple times with different choices of parameters. The alternative is to create an m-file.

In the File menu go to New and select M-File. This opens a new window that is the Editor. In this window you can write many lines of code and use the mouse to position the insertion point as in a text editor. In the Debug menu choose Save and Run (or simply Run) to execute the lines of code.

#### **4.9 Loading data**

MATLAB enables loading of many different types of data. A couple of the more useful methods that are used in the exercises are mentioned here.

Most data in the exercises will come in ASCII text files in which values are arranged in rows and columns as an array. MATLAB easily recognizes those, but is sensitive to missing entries which should be replaced with nan. Text files are loaded using:

```
>> DATA = load('data.txt');
```

The variable DATA will be an array containing all rows and columns of data.txt.

Another file type that is easily loaded and may be used in the exercises is an EXCEL file. MATLAB loads these using:

```
>> data = xlsread('data.xls');
```

### **5 Putting it all together**

All the calculations so far were done in the command window, but when you solve more complicated problems, you may get tired of retyping everything into the command window. There are two ways MATLAB allows you to write your own routines (programs).

Scripts operate on the data in the workspace. They do not accept input arguments or return output arguments. Input is taken from the workspace and output is written to the workspace. Advantages are a slightly easier syntax and access to all data in the workspace. The disadvantage is that the workspace might get cluttered with variables. Many scripts are provided on the textbook website that reproduce graphical figures from the textbook and solve exercise problems.

Functions accept input arguments and return output arguments. All variables defined or calculated within the function remain local to the function and are not saved to the workspace. The main advantage is that workspace does not get cluttered. The syntax is a little more complicated than that for scripts.

### **6 Exercise using an example script**

Below is an example script that can be used to reproduce Figure 1.15 in the text, a contour map of the normal stress component acting parallel to a model fault. The Matlab version you will

create is in color, whereas the text version is gray scale. The ranges for both  $x$  and  $y$  are from  $-2.5a$  to  $+2.5a$  where  $a$  is the half length of the model fault. This is referred to as the 'local' stress field in the vicinity of the model fault. The sign conventions associate a positive displacement discontinuity across the model fault with left-lateral slip. Also, tensile stress is positive and compressive stress is negative.

Copy an electronic version of this script from the textbook website under Chapters -> Chapter 1 -> Chapter Scripts, paste this into the MATLAB Editor, and run the script to confirm that it does reproduce the figure. Read the script line by line and try to understand how the script is organized. Use the help command to get detailed information about the MATLAB constants and functions. Focus your attention on the script itself and not on the mechanical concepts related to faulting or the derivation of the equations for the stress component. Those concepts and derivations will be considered in a later chapter.

```
% fig_01_15
% constant shearing displacement discontinuity on 2D line element
% Crouch and Starfield (1990) chapter 5
% element centered at origin, parallel to x-axis, from -a <= x <= +a
% calculate Cartesian normal stress component in x.

clear all, clf reset; % initialize memory and figure
dx = 0.001; % displacement discontinuity tangential to element
mu = 30000; pr = 0.25; % elastic shear modulus and Poisson's ratio
c = 1/(4*pi*(1-pr)); % constant multiplier for stress components
a = 1; % line element half length
rc = 0.09*a; % radius of dislocation core at element tips
x=-2.5*a:.05*a+eps:2.5*a; % x-coords. of grid
y=-2.5*a:.05*a:2.5*a; % y-coords. of grid
[X,Y] = meshgrid(x,y); % Cartesian grid
XMA2 = (X-a).^2; XPA2 = (X+a).^2; Y2 = Y.^2; % common terms
% derivatives of stress function
FCXY = c*( (Y./(XMA2+Y2)) - (Y./(XPA2+Y2)) );
FCXY2 = c*( ((XMA2-Y2)./(XMA2+Y2).^2) - ((XPA2-Y2)./(XPA2+Y2).^2) );
SXX = 2*mu*dx*(2*FCXY + Y.*FCXY2); % Cartesian stress component
% eliminate values within dislocation core at tips of element
R1 = sqrt((X-a).^2+Y.^2); R2 = sqrt((X+a).^2+Y.^2);
SXX(find(R1<(rc))) = nan; SXX(find(R2<(rc))) = nan;
contourf(X,Y,SXX,25) % plot contour map of stress component sxx
title('stress sxx'), xlabel('x'), ylabel('y'), colorbar
```

1) Run the script and print the contour map of the local stress field. Describe the stress distribution, paying particular attention to the symmetry and the juxtaposition of stress with opposite sign across the model fault.

2) Describe what is accomplished in each of the following lines of script:

```
clear all, clf reset;

x=-2.5*a:.05*a+eps:2.5*a;

[X,Y] = meshgrid(x,y);
```

Fundamentals of Structural Geology  
Exercise: introduction to MATLAB

```
R1 = sqrt((X-a).^2+Y.^2); SXX(find(R1<(rc))) = nan;  
contourf(X,Y,SXX,25),title('stress sxx'), xlabel('x'), ylabel('y'), colorbar
```

3) Write an alternative piece of code not using the ':' operator to accomplish the same thing as the following line.

```
y=-2.5*a:.05*a:2.5*a;
```

4) Run the script to evaluate and plot the normal stress distribution in the region immediately around the right-hand model fault tip (this is the so-called 'near-tip' field). For example, compute out to distances that are only twice the radius of the dislocation core,  $rc$ . Print this contour map. Describe the stress distribution, contrasting this with the 'local' stress distribution. How are they similar? How are they different?

5) Run the script to evaluate and plot the normal stress distribution in the region that includes points far from the model fault relative to its half length (the so-called 'remote' field). For example compute out to distances of  $25a$ . Print this contour map. Describe the stress distribution, contrasting this with the 'local' and 'near-tip' stress distributions. How are they similar? How are they different?

6) Plot the colored parametric surface in 3D for the local stress distribution. Use this plot to illustrate and describe where this component of stress appears to be discontinuous.

7) What outcrop-scale geologic structures might be associated with the stress concentrations that have different signs on either side of the model fault?

## 7 Summary

This has been a very cursory introduction to a few of the myriad functions and operations of MATLAB. It is meant to simply whet your appetite for using this wonderful tool. This would be a good time to get a book with tutorial coverage of all the basic functions and operations and continue your learning experience.

We have found that MATLAB is an invaluable tool for structural geologists. The approach we advocate in the textbook is to adopt this tool and use m-scripts to analyze structural data and build models of tectonic processes. It is our opinion that becoming an accomplished user of MATLAB puts students in a better position to succeed in this profession than learning to use many special purpose programs for individual tasks such as construction of a stereographic projection or solving a particular boundary value problem.