Massachvsetts Institvte of Technology

Department of Earth, Atmospheric and Planetary Sciences

Essentials of Geophysics
12.201/12.501

Introduction to MATLAB™

1 Getting started.

To start the program type MATLAB^{TM} at the UNIX^{TM} prompt. The system responds with:

```
Commands to get started: intro, demo, help help
Commands for more information: help, whatsnew, info, subscribe
>>
```

Looking for information on a function? Type `help function`.
Looking for something some function might be doing? Type `lookfor seomething`.

Our first command will make a record of the session, in a file named "session". [The `>>` is MATLAB^{TM}'s prompt, you don't need to type it]. Type:

```
>> diary session
```

Arithmetic uses some fairly standard notation. More than one command may be entered on a single line, if they are seperated by commas.

```
>> 2+3
>> 3*4, 4^2
```

Powers are performed before division and multiplication, which are done before subtraction and addition.

```
>> 2+3*4^2
```

The arrow keys allow "command-line editing," which cuts down on the amount of typing required, and allows easy error correction. Press the "up" arrow, and add "/2." What will this produce?

```
>> 2+3*4^2/2
```

Parentheses may be used to group terms, or to make them more readable.

```
>> (2 + 3*4^2)/2
```

The equality sign is used to assign values to variables.

```
>> x = 3
>> y = x^2
>> y/x
```

If no other name is given, an answer is saved in a variable named "ans."

```
>> ans, z=2*ans, ans
```

Here z was defined in terms of ans. The result was called z, so ans was unchanged.
To get a list of your variables, use one of

```
>> who, whos
```

In MATLAB^{TM}, like C or Fortran, variables must have a value [which might be numerical, or a string of characters, for example]. Complex numbers are automatically available [by default, both i and j are initially aliased to sqrt(-1)]. All arithmetic is done to double precision [about 16 decimal digits], even though results are normally displayed in a shorter form.

```
>> a=sqrt(2)
>> format long, b=sqrt(2)
>> a-b
>> format short
```

To save the value of the variable "x" to a plain text file named "x.value" use

```
>> save x.value x -ascii
```

To save all variables in a file named mysession.mat, in reloadable format, use

```
>> save mysession
```

To restore the session, use

```
>> load mysession
```

To find out about this kind of thing, consult the `help` system. There's even an HTML version! There's also a `lookfor`command, so that you don't have to guess the topic name precisely.

```
>> help
>> help general
>> doc
```

Finally, to stop MATLAB^{TM} and return to the operating system, use

```
>> quit
```

Then, to see the saved files from your session, type

```
% more session
% more x.value
```

2 Matrices.

A matrix is a rectangular array of numbers: for example,

```
[ 1 2 3 ]
[ 4 5 6 ]
```

defines a matrix with 2 rows, 3 columns, 6 elements. MATLAB^{TM} is designed to make matrix manipulation as simple as possible. Every MATLAB^{TM} variable refers to a matrix [a 1 row by 1 column matrix is a number]. Start MATLAB^{TM} again, and enter the following command.

```
>> a = [1,2,3; 4 5 6]
```

Note that the elements of a matrix being entered are enclosed by brackets; a matrix is entered in "row-major order" [ie all of the first row, then all of the second row, etc]; rows are seperated by a semicolon [or a newline], and the elements of the row may be seperated by either a comma or a space. [Caution: Watch out for extra spaces!]
The element in the i'th row and j'th column of a is referred to in the usual way:

```
>> a(1,2), a(2,3)
```

It's very easy to modify matrices:

```
>> a(2,3) = 10
```

The transpose of a matrix is the result of interchanging rows and columns. MATLAB^{TM} denotes the [conjugate] transpose by following the matrix with the single-quote [apostrophe].

```
>> a'
>> b=[1+i 2 + 2*i 3 - 3*i]'
```

New matrices may be formed out of old ones, in many ways.

```
>> c = [a; 7 8 9]
>> [a; a; a]
>> [a, a, a]
>> [a', b]
>> [ [a; a; a], [b; b] ]
```

There are many built-in matrix constructions. Here are a few:

```
>> rand(1,3), rand(2)
>> zeros(3)
>> ones(3,2)
>> eye(3), eye(2,3)
>> magic(3)
>> hilb(5)
```

This last command creates the 5 by 5 "Hilbert matrix," a favorite example in numerical analysis courses. Use a semicolon to suppress output:

```
>> s = zeros(20,25);
```

This is valuable, when working with large matrices. If you forget it, and start printing screenfuls of unwanted data, Control-C is MATLAB^{TM}'s "break" key. To get more information on these, look at the help pages for elementary and special matrices.

```
>> help elmat
>> help specmat
```

A central part of MATLAB^{TM} syntax is the "colon operator," which produces a list.

```
>> -3:3
```

The default increment is by 1, but that can be changed.

```
>> x = -3 : .3 : 3
```

This can be read: "x is the name of the list, which begins at -3, and whose entries increase by .3, until 3 is surpassed." You may think of x as a list, a vector, or a matrix, whichever you like.
You may wish use this construction to extract "subvectors," as follows.

```
>> x(2:12)
>> x(9:-2:1)

>> x=10:100;
>> x(40:5:60)
```

The colon notation can also be combined with the earlier method of constructing matrices.

```
>> a = [1:6 ; 2:7 ; 4:9]
```

A very common use of the colon notation is to extract rows, or columns, as a sort of "wild-card" operator which produces a default list. The following command produces the matrix a, followed by its first row [with all of its columns], and then its second column [with all of its rows].

```
>> a, a(1,:), a(:,2)
>> s = rand(10,5);  s(6:7, 2:4)
```

Matrices may also be constructed by programming. Here is an example, creating a "program loop."

```
>> for i=1:10,
>>      for j=1:10,
>>              t(i,j) = i/j;
>>      end
>> end
```

There are actually two loops here, with one nested inside the other; they define t(1,1), t(1,2), t(1,3) ... t(1,10), t(2,1), t(2,2) ... , t(2,10), ... t(10,10) [in that order].

```
>> t
```

3 Matrix arithmetic.

If necessary, re-enter the matrices

```
>> a = [1 2 3 ; 4 5 6 ; 7 8 10], b = [1 1 1]'
```

Scalars multiply matrices as expected, and matrices may be added in the usual way; both are done "element by element."

```
>> 2*a, a/4
>> a + [b,b,b]
```

Scalars added to matrices produce a "strange" result, but one that is sometimes useful; the scalar is added to every element.

```
>> a+1, b+2
```

Matrix multiplication requires that the sizes match. If they don't, an error message is generated.

```
>> a*b, b*a
>> b'*a
>> a*a', a'*a
>> b'*b, b*b'
```

To perform an operation on a matrix element-by-element, precede it by a period.

```
>> a^2, a.^2
>> a.*a, b.*b
>> 1 ./ a
>> 1./a.^2
```

One of the main uses of matrices is in representing systems of linear equations. If a is a matrix containing the coefficients of a system of linear equations, x is a column vector containing the "unknowns," and b is the column vector of "right-hand sides," the constant terms, then the matrix equations

$$\mathbf{ax} = \mathbf{b} \tag{1}$$

represents the system of equations. MATLAB^{TM} provides a very efficient mechanism for solving linear equations:

```
>> x = a \ b
```

This can be read "x equals a-inverse times b." To verify this assertion, look at

```
>> a*x, a*x - b
```

Change b, and do the problem again.

```
>> b = [1 1 0]'
>> x = a\b
>> a*x, a*x - b
```

If there is no solution, a "least-squares" solution is provided [a*x - b is as small as possible]. Enter

```
>> a(3,3) = 9
```

[which makes the matrix singular] and do those again. [Use the up-arrow, to recall the commands without retyping them].
There is a related problem, to solve

$$\mathbf{xa} = \mathbf{b} \tag{2}$$

(given a and b), which is done with

```
>> x = b / a
```

This can be read "B times A-inverse." Again, if there is no solution, a least-squares solution is found.

4 Matrix functions.

There are a number of built-in matrix functions, for example the determinant, rank, nullspace, and condition number.

```
>> det(a)
>> rank(a)
>> norm(a)
>> null(a)
```

Enter

```
>> a(3,3) = 10
```

[which makes the matrix nonsingular] and do those again.
Other valuable functions find the inverse, eigenvalues and eigenvectors of a matrix.

```
>> h=hilb(5)
>> cond(a)
>> inv(h)
>> eig(h)
```

The "eig" function has two forms of output. The last command produced a vector of eigenvalues. The next command produces two matrices, the first containing the eigenvectors as its columns, and the second containing the eigenvalues, along its diagonal.

```
>> [v,d]=eig(h)
```

The matrix, h, times the first eigenvector, v(:,1), should equal the first eigenvalue, d(1,1), times that same eigenvector.

```
>> h*v(:,1)
>> d(1,1)*v(:,1)
>> v*d*inv(v), inv(v)*h*v
```

"Round-off error" is a primary concern in numerical computing. MATLAB^{TM} does numerical computation, which is to say, it works with limited precision; all decimal expansions are truncated at the sixteenth place [roughly speaking]. Even if this is acceptable for any single calculation, its effects may accumulate with unacceptable consequences. The machine's round-off, the smallest distinguishable difference between two numbers as represented in MATLAB^{TM}, is denoted "eps".

```
>> help eps
>> eps
```

We can check the assertion just made about eigenvectors and eigenvalues, as follows.

```
>> h*v(:,1) - d(1,1)*v(:,1)
```

This is "the zero vector, modulo round-off error."

5 Graphics.

MATLAB^{TM} has outstanding graphics capabilities. Start with

```
>> x = -10:.1:10;
>> plot( x.^2 )
>> figure
>> plot( x, x.^2 )
>> figure
>> plot( x.^2, x )
```

Note that x must be assigned values, before the plot command is issued [although you could use

```
>> plot( (-10 : .1 : 10).^ 2 )
```

if you really really wanted to].

```
>> plot( x, x.*sin(x) )
>> plot( x.*cos(x), x.*sin(x) )
>> comet( x.*cos(x), x.*sin(x) )
>> plot3(x.*cos(x),x.*sin(x),x)
```

Functions of two variables may be plotted, as well, but some "setup" is required!

```
>> [x y] = meshgrid(-3:.1:3, -3:.1:3);
>> z = x.^2 - y.^2;
>> mesh(x,y,z)
>> plot3(x,y,z)
>> surf(x,y,z)
>> contour(z)
>> help slice
```

There's a very interesting example, in the help page for slice; use the mouse to cut and paste it to the MATLAB^{TM} prompt.
The following commands bring up lists of useful graphics commands [each has a help page of its own].

```
>> help plotxy
>> help plotxyz
>> help graphics
```

6 Scripts and functions.

MATLAB^{TM} statements can be prepared with any editor, and stored in a file for later use. The file is referred to as a script, or an "m-file" (since they must have names of the form foo.m). Writing m-files will make you much more productive.
Using your favorite editor, create the following file, named sketch.m:

```
[x y] = meshgrid(-3:.1:3, -3:.1:3);
z = x.^2 - y.^2;
mesh(x,y,z);
```

Then start MATLAB^{TM} from the directory containing this file, and enter

```
>> sketch
```

The result is the same as if you had entered the three lines of the file, at the prompt.
You can also enter data this way: if a file named mymatrix.m in the current working directory contains the lines

```
A = [2 3 4; 5 6 7; 8 9 0]
inv(A)
quit
```

then the command

```
>> mymatrix
```

reads that file, generates A and the inverse of A, and quits MATLAB^{TM} [quitting is optional]. You may prefer to do this, if you use the same data repeatedly, or have an editor that you like to use. You can use Control-Z to suspend MATLAB^{TM}, then edit the file, and then use "fg" to bring MATLAB^{TM} back to the foreground, to run it.
MATLAB^{TM} may be ran in "batch mode," in a very similar way. If a file named "test.in" contains the [non-graphics] commands you want processed, at the UNIX^{TM} prompt type:

```
% {\sc Matlab}$^{{\mrm \ssts TM}}$ < mymatrix.m > homework.out
```

This is read, "Run MATLAB^{TM}, with input from test.in, and output to test.out." The input file does not need to be named "something-dot-m," but it must end with "quit"!
Functions are like scripts, but are compiled the first time they are used in a given session, for speed. Create a file, named sqroot.m, containing the following lines.

```
function sqroot(x)
% Compute square root by Newton's method

% Initial guess
xstart = 1;

for i = 1:100
        xnew = ( xstart + x/xstart)/2;
        disp(xnew);
        if abs(xnew - xstart)/xnew < eps, break, end;
        xstart = xnew;
end
```

Save this file, start MATLAB^{TM}, and enter the commands

```
>> format long
>> sqroot(19)
```

A good exercise would be to create the STAT function described in the help file for function. Note that

```
>> stat(x)
```

and

```
>> [m,sd] = stat(x)
```

produce different results.
The "m-files" which came with MATLAB^{TM} provide lots of examples! To find their location, use

```
>> path
```

This will also lead you to some really nifty demos.

7 Command reference.

7.1 Some (case-sensitive!) basic commands.

`matlab`	loads the program MATLAB^{TM} into your workspace
`quit`	quits MATLAB^{TM}, returning you to the operating system
`exit`	same as quit
`who`	lists all of the variables in your MATLAB^{TM} workspace
`whos`	list the variables and describes their matrix size
`clear`	deletes all matrices from active workspace
`clear x`	deletes the matrix x from active workspace
`...`	the ellipsis defining a line continuation is three successive periods
`save`	saves all the matrices defined in the current session into the file, **matlab.mat**
`load`	loads contents of MATLAB^{TM}.mat into current workspace
`save filename`	saves the contents of workspace into filename.mat
`save filename x y z`	saves the matrices x, y and z into the file titled **filename.mat**
`load filename`	loads the contents of filename into current workspace; the file can be a binary (**.mat**) file or an ASCII file
`!`	the ! preceding any UNIX^{TMTM} command causes the UNIX^{TM} command to be executed from MATLAB^{TM}

7.2 Commands Useful in Plotting.

`who`	lists all of the variables in your MATLAB^{TM} workspace
`plot(x,y)`	creates an Cartesian plot of the vectors **x** and **y**
`plot(y)`	creates a plot of **y** vs. the numerical values of the elements in the y-vector
`semilogx(x,y)`	plots log(x) vs y
`semilogy(x,y)`	plots x vs log(y)
`loglog(x,y)`	plots log(x) vs log(y)
`grid`	creates a grid on the graphics plot
`title('text')`	places a title at top of graphics plot
`xlabel('text')`	writes 'text' beneath the x-axis of a plot
`ylabel('text')`	writes 'text' beside the y-axis of a plot

`text(x,y,'text')`	writes 'text' at the location (x,y)
`text(x,y,'text','sc')`	writes 'text' at point x,y assuming lower left corner is (0,0) and upper right corner is (1,1)
`gtext('text')`	writes text according to placement of mouse
`hold on`	maintains the current plot in the graphics window while executing subsequent plotting commands
`hold off`	turns OFF the 'hold on' option
`polar(theta,r)`	creates a polar plot of the vectors r and theta where theta is in radians
`bar(x)`	creates a bar graph of the vector x. (Note also the command `stairs(y)` .)
`bar(x,y)`	creates a bar-graph of the elements of the vector y, locating the bars according to the vector elements of 'x'. (Note also the command `stairs(x,y)` .)
`hist(x)`	creates a histogram. This differs from the bargraph in that frequency is plotted on the vertical axis
`mesh(z)`	creates a surface in xyz space where z is a matrix of the values of the function z(x,y). z can be interpreted to be the height of the surface above some xy reference plane
`surf(z)`	similar to mesh(z), only surface elements depict the surface rather than a mesh grid
`contour(z)`	draws a contour map in xy space of the function or surface z
`meshc(z)`	draws the surface z with a contour plot beneath it
`meshgrid`	`[X,Y]=meshgrid(x,y)` transforms the domain specified by vectors x and y into arrays X and Y that can be used in evaluating functions for 3D mesh/surf plots
`print`	sends the contents of graphics window to printer
`print filename -dps`	writes the contents of current graphics to `filename` in PostScript format

7.3 Equation Fitting.

`polyfit(x,y,n)`	returns the coefficients of the n-degree polynomial for the vectors x and y. n must be at least 1 larger than the length of the vectors x and y. If n+1 =length(x) the result is an interpolating polynomial. If n+1 ¿ length(x) the result is a least-squares polynomial fit. The coefficients are stored in order with that of the highest order term first and the lowest order last
`polyval(c,x)`	calculates the values of the polynomial whose coefficients are stored in c, calculating for every value of the vector x

7.4 Data Analysis Commands.

`max(x)`	returns the maximum value of the elements in a vector or if x is a matrix, returns a row vector, whose elements are the maximum values from each respective column of the matrix
`min (x)`	returns the minimum of x (see max(x) for details)
`mean(x)`	returns the mean value of the elements of a vector or if x is a matrix, returns a row vector whose elements are the mean value of the elements from each column of the matrix
`median(x)`	same as mean(x), only returns the median value
`sum(x)`	returns the sum of the elements of a vector or if x is a matrix, returns the sum of the elements from each respective column of the matrix
`prod(x)`	same as sum(x), only returns the product of elements
`std(x)`	returns the standard deviation of the elements of a vector or if x is a matrix, a row vector whose elements are the standard deviations of each column of the matrix
`sort(x)`	sorts the values in the vector x or the columns of a matrix and places them in ascending order. Note that this command will destroy any association that may exist between the elements in a row of matrix x
`hist(x)`	plots a histogram of the elements of vector, x. Ten bins are scaled based on the max and min values
`hist(x,n)`	plots a histogram with 'n' bins scaled between the max and min values of the elements

`hist((x(:,2))`	plots a histogram of the elements of the 2nd column from the matrix x
`fliplr(x)`	reverses the order of a vector. If x is a matrix, this reverse the order of the columns in the matrix
`flipud(x)`	reverses the order of a matrix in the sense of exchanging or reversing the order of the matrix rows. This will not reverse a row vector!
`reshape(A,m,n)`	reshapes the matrix A into an $m \times n$ matrix from element (1,1) working column-wise

8 Special matrices.

`zeros(n)`	creates an nxn matrix whose elements are zero.
`zeros(m,n)`	creates a m-row, n-column matrix of zeros
`ones(n)`	creates a n x n square matrix whose elements are 1's
`ones(m,n)'`	creates a mxn matrix whose elements are 1's
`ones(A)`	creates an m x n matrix of 1's, where m and n are based on the size of an existing matrix, A
`zeros(A)`	creates an mxn matrix of 0's, where m and n are based on the size of the existing matrix, A
`eye(n)`	creates the nxn identity matrix with 1's on the diagonal

8.1 Miscellaneous Commands.

`length(x)`	returns the number elements in a vector
`size(x)`	returns the size m(rows) and n(columns) of matrix x
`rand`	returns a random number between 0 and 1
`randn`	returns a random number selected from a normal distribution with a mean of 0 and variance of 1
`rand(A)`	returns a matrix of size A of random numbers

9 Algebraic operations in MATLAB^{TM}.

9.1 Scalar calculations.

+	addition
-	subtraction
*	multiplication
/	right division (a/b means a divided by b)
\	left division (a\b means b divided by a)
^	exponentiation

The precedence or order of the calculations included in a single line of code follows the below order:

Precedence	Operation
1	parentheses
2	exponentiation, left to right
3	multiplication and division, left to right
4	addition and subtraction, left to right

10 Matrix algebra.

In matrix multiplication, the elements of the product, C, of two matrices A*B is calculated from

$$C_{ij} = \sum_{k}^{N} A_{ik} B_{kj} \tag{3}$$

To form this sum, the number of columns of the first or left matrix (A) must be equal to the number of rows in the second or right matrix (B). The resulting product, matrix C, has an order for which the number of rows equals the number of rows of the first (left) matrix (A) and the product (C) has a number of columns equal to the number of columns in the second (right) matrix (B). It is clear that A*B is not necessarily equal to B*A!
The product of a scalar and a matrix is a matrix in which every element of the matrix has been multiplied by the scalar.

11 Array products.

Sometimes it is desired to simply multiply or divide each element of an matrix by the corresponding element of another matrix. These are called "array operations" in MATLAB^{TM}. Array or element-by-element operations are executed when the operator is preceded by a '.' (period).
Thus

`a .* b`	multiplies each element of a by the respective element of b
`a ./ b`	divides each element of a by the respective element of b
`a . b`	divides each element of b by the respective element of a
`a .^b`	raise each element of a by the respective b element

12 Transpose of a matrix.

`x'`	The transpose of a matrix is obtained by interchanging the rows and columns. The MATLAB^{TM} operator that creates the transpose is the single quotation mark, '

13 Inner product of two vectors.

The inner product of two row vectors G1 and G2 is G1*G2'. The inner product of two column vectors H and J is H'*J.

14 Outer product of two vectors.

If two row vectors exist, G1 and G2, the outer product is simply

```
G1' * G2
```

(note G1' is nx1 and G2 is 1xn)
and the result is a square matrix in contrast to the scalar result for the inner product. Don't confuse the outer product with the vector product!! If the two vectors are column vectors, the outer product must be formed by the product of one vector times the transpose of the second!

15 Solution to simultaneous equations.

15.1 Using the Matrix Inverse.

`inv(a)` returns the inverse of the matrix a.
If $\mathbf{ax} = \mathbf{b}$ is a matrix equation and $\mathbf{a}$ is the coefficient matrix, the solution $\mathbf{x}$ is $\mathbf{x} = \mathbf{a}^{-1}\mathbf{b}$

15.2 Using Back Substitution.

`a\` returns a column vector solution for the matrix equation $\mathbf{ax} = \mathbf{b}$ where a is a coefficient matrix

`b/a` returns a row vector solution for the matrix equation $\mathbf{xa} = \mathbf{b}$ where a is a coefficient matrix