A Concise Introduction to

MATLAB

by

Ted Scheick

General facts and Tips

Command screen

In the command screen, you enter commands and view the results of your calculations. This screen scrolls, so you can look at your previous work. It is all saved till you quit. It can be printed. However, it is best to select and copy from this screen and paste the results into a word processor. There you can add comments and produce a nice looking document.

All variable names are case sensitive.

All variables are arrays. Internal precision is 16 digits.

All indices in arrays start with 1. This cannot be changed.

Several commands may be placed on one line, if commas or semicolons separate them.

;	after a command suppresses printing of the result. Often you will be glad you did this.
pi	= π
i,j	= $\sqrt{-1}$. e.g. 2+3i, 5+pi*i
inf	infinity
Nan	not a number
y=x	assigns x to y (stores x in y)
disp(x)	displays the variable x. Useful for making tables.
ans	returned if the result was not assigned to another name. Can be used in the next calculation.
clc	clears the command screen
↑,↓	Matlab stores recent keyboard input. These keys scroll through them VERY handy.
←,→	These keys move through the command on the current command line. Handy for changing the current command line (or one that was recalled by the ↑, ↓ keys). Very handy.

The default numerical format has 4 digits after the decimal point. Change in the Options menu, or by

format short	4 digits after the decimal place
format long	all 16 digits
format short e	scientific notation with 5 significant digits
format long e	scientific notation with all 16 digits.

Workspace

As you work, all you variables are saved. You can examine them, clear them and save or load them.

who	shows the variables currently saved
whos	shows the variables currently saved, their size and memory used.
clear	clears all variables
clear x A	clears x, A
save *name*	saves all variables to *name*.mat in the current directory
load *name*	loads the variables stored in *name*.mat in the current directory

Directories and Files

what	lists all m-files in the current directory
dir (or ls)	lists all files and subdirectories in the current directory
cd (or chdir)	show the current directory
cd ..	move up one directory
cd *dir*	change the current directory to the subdirectory *dir*
cd *path*	change the current directory to the on specified in *path*, e.g. *path* = c:\matlab\bin

Help Useful, but sometimes frustrating.

You can use the help menu to search for topics or to look at a table of contents. There are many interesting things (and m-files) you can examine through the table of contents.

From the keyboard,

help *name*	(*name* = operator, command, topic) will display information on the entity.
e.g. help \	displays information on \ and other operators and special characters
help relop	information about the relational and logical operators

Creating matrices, Fundamental operations

Matlab carries 16 digits but shows 4 after the decimal place in the default mode. Matlab does not display the braces around matrices. The expression pi stands for π, and i or j = $\sqrt{-1}$. A complex number is entered as 2+3i, for example. Names are case sensitive.

Entries of matrices can be expressions or functions of known variables.

If the matrix is to created but not displayed, follow the entry with a ; before hitting return.

Enter		Result (no brackets or , shown on screen)	
Row vectors			
r = [1/2 2.7 3]	a space between entries	[0.5 2.7 3]	is stored in r
r = [sin(1), 2^3-1, 2,pi]	commas separate entries	[0.8414 7 2 3.1416]	
some special rows:			
r = 2:6	(in general, r = m:n)	[2 3 4 5 6]	
r = a:h:b		[a, a+h, a+2h, …. b],	b=a+Nh (h<0 is o.k.)
e.g. r = 0:0.1:1		[0, 0.1, 0.2, … 0.9, 1]	
r = linspace(a,b,n)		n equispaced points from a to b, inclusive	
e.g. r = linspace(1,2,5)		[1, 1.25, 1.5, 1.75, 2]	
Column vectors c = r' where r is a row. ' means "transpose": make a row into a column and vice versa.			
c = (4:6)'		$\begin{bmatrix}4\\5\\6\end{bmatrix}$	is stored in c
Another way: use ; to start a new row c = [exp(1); sin(pi/2); 2/3]		$\begin{bmatrix}2.718\\1\\0.6667\end{bmatrix}$	
Matrices	; or ENTER starts a new row		
a = [1 3 1/2; 4 -1 2+sqrt(-1)]		$\begin{bmatrix}1 & 3 & 0.5\\4 & -1 & 2+i\end{bmatrix}$	is stored in a
a = [1 2 3 4 5 (hit ENTER) 11 12 13 14 15 (") 9 8 7 6 5] spaces were added to line up the entries		$\begin{bmatrix}1 & 2 & 3 & 4 & 5\\11 & 12 & 13 & 14 & 15\\9 & 8 & 7 & 6 & 5\end{bmatrix}$	
First define rows r1, r2, r3 of the same length. A = [r1; r2; r3]		$\begin{bmatrix}r1\\r2\\r3\end{bmatrix}$	i.e. A is a matrix with rows r1, r2, r3
First define columns c1, c2, c3 of the same length. a = [c1 c2 c3]		[c1 c2 c3] i.e. a is a matrix with columns c1, c2, c3.	
B = [A x]		Appends the column x to the right of A to form B	
C = [A B]	(a space or , separates columns)	Places the columns of B to the right of those of A and stores the result as C	
C = [A;B]	(a ; separates rows)	Places the rows of B below those of A to form C	

Using a loop:

```
for k = 1:m,
  for j = 1:n,
    A(k,j) = 1/(k+j+1);   (use  ;  to suppress printing)
  end
end
```

Creates A (mxn) with the given entries in row k, column j. m and n must be specified in the loop or predefined.

Other tricks and examples.

A = [ ]	defines A to be the empty matrix (one way to clear an old matrix)
B = [A [1 2 3]']	adds the column [1,2,3]' to A on the right. (A must have 3 rows).
B = [A; [2,3,5]]	adds the row [2,3,5] to A at the bottom.
c = A(:,3)	stores the 3rd column of A in c
r = B(5,:)	stores the 5th row of B in r
a(2,4) (return)	displays the entry of a in the 2nd row and 3rd column.
A(4,5) = 2	redefines A_{45} to be 2.
	If A were 3x3, A is redefined to be 4x5, and the undefined entries are set = 0.
A(:,5) = c	replaces column 5 of A by c
	If A had 3 columns, column 5 is set = c, and the 4th column of A is set = 0.
B(10,:) = [ ]	deletes row 10 of B
d(:,4) = [ ]	deletes column 4 of d

Suppose x is a row or column vector with entries x_1, x_2, … x_n

x(2:5)	is the same kind of vector but with entries x_2,x_3,x_4,x_5

Suppose A is a matrix with entries a_{ij}

A(2:5,:)	is the submatrix formed from rows 2,3,4,5
A(:,5,7,9)	is the submatrix formed from columns 5,7,9.
A(:,[$i_1,i_2,\ldots,i_s$])	is the submatrix composed of columns $i_1,\ldots,i_s$ of A.
A(1:3,4:6)	is the submatrix formed from the entries a_{ij} in rows 1,2,3 and columns 4,5,6.

Special Matrices

zero(m,n)	the zero matrix of size mxn
ones(m,n)	the m×n matrix full of ones
diag(v)	the diagonal matrix with diagonal entries $d_{ii} = v_i$ (v is a given vector).
eye(n)	the n×n identity matrix
rand(m,n)	an m×n matrix full of random numbers in (0,1)

Operations and matrix functions: A, B can be vectors or matrices (in most of these)

A+B	add A and B
α*A	multiply A by the scalar α
A/α	= (1/α)*A (α is a scalar)
A*B	matrix product (if defined)
A'	conjugate transpose of A; the transpose if A is real
A+c	adds the scalar c to every entry of the matrix or vector A
inv(A)	A^{-1}, if A is nxn and it exists. You are warned if it is numerically close to non-existence.
A^n	A*A*…*A n times. Matlab does NOT compute n products to get this (too slow).
A.*B	entrywise product $[a_{ij}*b_{ij}]$
A.^p	entrywise powers $[a_{ij}^{p}]$
f(A)	entrywise computation: f is applied to every entry of A.
	f = abs, sign, sqrt, exp, sin, cos, tan, asin, acos, atan, log etc. (see help elfun.)
polyval(c,x)	The values of a polynomial at each entry of the vector x. If c has length k+1,
	poylval(c,x) = c(1)*x.^k + … c(k)*x + c(k+1)
x=size(A)	gives x=[x(1) x(2)], x(1) = #rows, x(2) = #columns of A
length(x)	length of the vector x
det(A)	determinant of A
rank(A)	rank of A (NOT computed by row reduction)
rref(A)	row-reduced echelon form of A
round(A)	rounds entries of A to the nearest integer
norm(x)	the norm of a vector x. (other norms are also available, use help norm).
x = A\b	If nonsingular, x = A^{-1}b = inv(A)*b, but computed by elimination. b can be a column or matrix. If A is not square, x is one of the "best approximate solutions". Can be quirky.

For HELP: type help *name* and hit return. E.g. help +, help rref, help round

sum(A)	If A is a vector, then sum(A) = sum of entries, otherwise sum(A) is a row whose entries are the sums of the columns of A.
min(A)	If A is a vector, min(A) is the smallest entry. Otherwise min(A) is a row whose entries are the least entry in each column.
max(A)	similar to min(A), but replace min by max
diag(A)	the column containing the diagonal entries of A
triu(A)	the upper triangular part of A (including the diagonal)
tril(A)	the lower triangular part of A (see help triu, tril for other uses).
[L U P] = lu(A)	PA = LU, P = permutation matrix, L lower triangular, U upper triangular.
eig(A)	the eigenvalues of A, as a column.
[P L] = eig(A)	P is a matrix whose columns are unit eigenvectors, L is a diagonal matrix of eigenvalues, with AP = PL. You get this even if A is defective.
c=poly(A)	c = [c_1 c_2 … c_{n+1}] consisting of the coefficients of the $(-1)^n$ times the characteristic polynomial of A: in backwards order: $p(\lambda) = c_1\lambda^n + \ldots + c_n\lambda + c_{n+1} = \|\lambda I - A\|$.
roots(c)	returns the zeros of the polynomial $p(\lambda) = c_1\lambda^n + \ldots + c_{n+1}$, where c=[$c_1$ c_2 … c_{n+1}].
poly(r)	returns the coefficients of the polynomial whose zeros form the vector r.
p=polyval(c,x)	x is a matrix or vector. c=[c_1 c_2 … c_{n+1}]. $p = c_1x^n + \ldots + c_nx + c_{n+1}$, computed entrywise.
eig(A,B)	the generalized eigenvalues λ: $Ae = \lambda Be$.
[P L]=eig(A,B)	P is the generalized eigenvector matrix, L the diagonal matrix of eigenvalues, AP = BPL
Q=orth(A)	the columns of Q are an orthonormal basis for R(A)
N=null(A)	the columns of N are an orthonormal basis for N(A)
[Q R] = qr(A)	A = QR, the Q-R decomposition. A need not be square, nor of full rank. See help qr.
[U T]=schur(A)	$A = UTU^H$, the Schur decomposition, U unitary, T upper triangular with $t_{ii} = \lambda_i$.
schur(A)	returns T, in the Schur decomposition.
svd(A)	the column of singular values of A.
[U S V] = svd(A)	the singular value decomposition A = USV. See help svd for the short svd.
pinv(A)	the pseudo inverse of A. See help pinv to see how to set a tolerance.
R=chol(A)	the cholesky decomposition: $A = R^TR$, if A is positive definite.
norm(A)	the L_2 norm of A = largest singular value. Other norms are available, use help.
cond(A)	L_2 condition number of A = largest singular value/smallest singular value
rcond(A)	an estimate of 1/cond(A)

Other items of interest

Matlab has a number of relational and boolean operators with which to compare matrices. To see them, execute: help relop. They are discussed on another page in a limited way.

Matlab can do FFTs, but you have to know how the discrete fourier and inverse fourier transforms work to do problems.

Many of the common special functions are well computed in Matlab.

There are several numerical analysis procedures in Matlab, e.g. ode solvers, integrators and zero finders. They are in m-files you can look at.

Plotting in Matlab

2d Plotting

Autoscaling is in effect unless the plotting window is specifically set. On each plot command, the previous graph is erased and a new one is drawn, unless this is overridden. See below.

The basic commands: t, y, x, z are vectors, A is a matrix.

plot(y)	plots y_i versus i
plot(t,y)	plots y_i versus t_i. t and y are of the same length.
plot(t,y,x,z)	plots y_i versus t_i and z_i versus x_i. in the same window. Different line colors are used. t and x can be of different lengths.
plot(A)	plot each column of A versus i, Line colors rotate.
plot(t,A)	plot each column of A versus t_i

<u>Adding labels and titles</u>.

Note: int2tstr(n) converts the integer n to a string,
num2str(x) converts the number x to a string
[a b] concatenates the strings a and b.

xlabel('*name*')	labels the x axis with the string *name*
ylabel('*name*')	labels the y axis *name*
title('*caption*')	adds the title *caption* above the graph
text(x,y,'*name*')	starts the text string *name* at the location (x,y) on the plot
gtext(x,y,'*name*')	waits for a mouse click to position the text insertion point

<u>Using markers and specifying colors.</u>

Colors:	y, m (magenta), c (cyan), r, g, b, w, k(black).
Markers:	. (point), o (circle), x, +, *, - (solid line), : (dotted line), -. (dash-dot line), -- (dashed line)

Note: for black and white printing, use one graph color and change line styles.

plot(x,y,'b:')	uses a blue dotted line.
plot(x,y,'r',t,z,'g+)	y versus x is plotted with a red line, z versus t is plotted with green + signs

<u>Overplots and setting the plotting window</u>.

hold on	Allows overplots. Subsequent plots are placed on the same graph. Autoscaling is in effect unless turned off.
hold off	Turns off the overplotting.
axis([l r b t])	Sets the plotting window to [l, r]×[b t]
v=axis	Returns v = [l r b t], the widow currently used.
axis(axis)	Freezes the window at the current size (for overplots in a fixed window).
axis auto	Turns on autoscaling
clf	clear graph window and reset to autoscaling defaults
cla	clear graph window of all plots and text, keeps the same window
figure	start a new plotting window

There are other 2d plotting options available. They work the same as PLOT. They are

semilogx(…)	log base 10 is used for the x scale
semilogy(…)	log base 10 is used for the y scale
loglog(…)	both x and y scales are logarithmic
polar(t,r)	polar coordinate plot: t = vector of angles, r = vector of radii.

Matlab can put several plots in an array on a page. See help subplot.

3D surface plots

After the surface has been plotted, you can rotate the figure or zoom in or out. These options are under the Tools menu in the menu bar. You can also add comments, etc. To set the viewing angle by a command, see help view.

1. **Wireframe plots**

E.g. plot $z = f(x,y) = x \sin(x-y^2)$ over $[-1,1]\times[0,3]$ using 50 x points and 60 y points.

```
    x = linspace(-1,1,50)        sets xi
    y = linspace(0,3,60)         sets yj
    [X,Y] = meshgrid(x,y);       used to build Z to plot      (don't forget the  ; )
(*) Z = X.*sin(X-Y.^2);          create z                     (don't forget the  ; )
    mesh(x,y,Z)                  does the plot, with colors.  mesh(X,Y,Z) also works.
```

Line (*) can be replace with a loop (note the i,j order).

```
for i=1:length(x)
   for j=1:length(y)
      Z(j,i) = $x_i \sin(x_i - y_j^2)$
   end
end
```

2. **Patch plots**

The surface is made of patches, bounded by the wireframe lines.
To do the same example in this style, do the following.

```
x = linspace(-1,1,50)        sets xi
y = linspace(0,3,60)         sets yj
[X,Y] = meshgrid(x,y);       used to build Z to plot      (don't forget the  ; )
Z = X.*sin(X-Y.^2);          create z                     (don't forget the  ; )
surf(x,y,Z)                  does the plot, with colors.  surf(X,Y,Z) also works.
```

Axes can be labeled using xlabel(), ylabel(), and zlabel() as in the 2d plotting section.

Contour plots	(Many options. See help contour)
contour(Z)	contour plot of the matrix Z, z_{ij} = height above the z=0 plane. The row index runs vertically, the column index horizontally.
contour(Z,'k')	plots all the contours black on a white background
contour(Z,n)	the same, but with n contour lines (overrides default)
contour(Z,v)	the same, but with contour lines at v_i, $v = [v_1\ v_2 \ldots v_n]$.

To do contour plots with specific x and y ranges, define x,y and Z as in the examples for surface plots.

contour(x,y,Z)	plots with the default number of colored contours. contour(X,Y,Z) does the same.
contour(x,y,Z,n)	plots with n contours
contour(x,y,Z,v)	plots with the vector of specified contours.

Matlab can label contours too. See help clabel.

Note: The plots from Matlab can be copied to the clipboard and pasted into MS Word. There they can be resized without loss of detail. It is better to start with a plot a bit too large and shrink it. To do this, after the plot window appears, in the Edit menu, select copy figure. Then paste into Word. You can put several plots on one page, and add typing or handwriting. You can also select lines from the command window and copy and paste them into Word.

M-files in Matlab

M-files serve as programs, subroutines or function procedures in Matlab. There are two types: script m-files and function m-files. Both are text files.

A script m-file can consist of exactly the commands you enter at the keyboard to perform a task, or it can be a program written in Matlab's simple language. For a long involved task, it is better to make an m-file, so than one mistake does not necessitate redoing many calculations.

A function m-file accepts inputs and returns outputs (see the more complete description below).

All m-files must be saved in the form *filename*.m. One m-file can call another while it is being executed.

Scope of variables

In a script m-file, all variables are global; in a function m-file they are local.

Comments The % sign allows you to make comments in the m-file. All text after the % sign is ignored.

It is good practice to put the name of your m-file and description of what it does at the start of the file. Begin each such line with %. It is a good idea to make liberal comments in the m-file. If you need to come back and change the file later, the comments will help you remember what you were doing.

Creating an m-file

Start Matlab. Pull down the File menu, choose New, M-file. (the m-file editor should appear).

Type in your commands and comments. Save your m-file: pull down the file menu, chose save.

Modifying an m-file (after starting Matlab).

If the Editor is open: just use the File>open>… sequence as usual.

If the Editor is not open, use File>Open M-file in Matlab.

Make your changes, and save the file with Save or Save As.

Running an m-file. The m-file must be in a path recognized by Matlab, or in the current directory.

You can set the current directory in the File menu by selecting Set Path…, and using Browse. To put your floppy in Matlab's path, execute: path(path,'a').

To run a script m-file from the command window, type the name of your m-file (without the .m) and hit enter. E.g., type pnfit (and hit enter).

Script m-files can be invoked by putting their name in a line of another m-file

Function m-files.

The first line of a simple function m-file which returns one output must look like

function y = *name*(*variablelist*)

name is your name for the function, *variablelist* is a list of variables, separated by commas. This line could be preceded by %*comments*.

e.g. y = quad(x)
y = apiv(piv,m,n)

If you wish to have two outputs y1, y2 returned, use

function [y1,y2] = *name*(*variablelist)*.

The rest of the function m-file is a sequence of Matlab commands or programming constructs, as usual.

All variables inside the function m-file are local. That is, if they have the same name as a variable in your workspace, the variable in the workspace will not be overwritten, and they are deleted after the execution of the function m-file. A function m-file is executed, for example, by

A = apiv(v,k,l)

The result of the function is stored in A. The variables v, k, l must have been defined before, or they can be entered in the list of variables directly. They must be of the same type as piv, m, n in the m-file called. You can execute the function m-file from the keyboard or from within another m-file.

Programming, flow control

Matlab has a simple straightforward programming language. However it is very powerful because all numerical variables are arrays of complex numbers.

There are three common structures: loops, while and block if statements. Simple examples of these can be found in Matlab help.

Loops: (can be nested) (help for)

```
    for i=1:n
        {executable statements}
    end
```

Block if: (elseif and else are optional) (help if)

```
    if test1
      {statements}
    elseif test2
      {statements}
    else
      {statements}
    end
```

test1 and *test2* are conditional statements involving the relational operators. (help relop)

== (equal), ~= (not equal), <, > ,<=, >=

and the boolean operators

&, | (or), ~ (not), xor (exclusive or).

When *test1* is true, its value is 1, and when false its value is 0.

While loop: (help while)

```
    while test
        {statements}
    end
```

test is a conditional statement involving the relational operators. Entering the loop, *test* is evaluated. If false, the loop is skipped. If true, the statements are executed and then *test* is evaluated again, and the cycle is repeated until *test* evaluates as false. Beware of infinite loops.

Additional statements

x = input('*promptstring*')	allows prompted inputs.
e.g.	
x = input('enter x= ')	inputs a scalar
x = input('x = [a b c] ')	type [1 2 3] (and hit return) to assign x = [1 2 3].
pause	pauses execution, waits for a keypress
break	terminates the execution of a for or while loop; only the inner (or current) loop is exited.
return	causes a return to the invoking function
disp(A)	displays the array A. Use string arrays for words. Useful for making tables.
e.g.	
disp('x y')	labels the columns; play with the spacing
disp([x y])	shows the columns x and y beside each other

Structured programming, a simple example.

The main program can be in a script m-file. Other script m-files can be used as subroutines (the variables are all global). They are invoked simply by placing the name of the called m-file in the main program as is done at the keyboard. Function m-files are invoked by inserting a statement like

A = apiv(piv,m,n)

where apiv(…) is the name of a function m-file, in the calling m-file. The variables in a function m-file are local, and will not overwrite the variables in the calling program, nor are they available to the calling program.

Sample M-files

The first example is a file which fits a polynomial of degree n to a function, or to data. To use it to fit data, delete (or comment out) the line yd = sin(exp(xd)) and remove the % sign in front of the yd = […] line.

```
%pnfit
%polynomial fits to a function or data
xd=[0 .2 .4 .6 .8 1 1.2 1.4 1.6 1.8 2]';                   %data xi
yd=sin(exp(xd));                                            %yi values by a function
%yd=[1 1.05 1.15 1.35 1.6 1.8 2 1.9 1.55 .9 .24]';      % yi by data list
n=input('degree=')
one=ones(size(xd));
a=[one];                            %build normal eq.
for i=1:n
  a=[a xd.^i];
end
c=(a'*a)\(a'*yd )                   %solve the normal equations
%       plot the fit (using 51 points in [0 2]), and the data with o's
x=linspace(0,2,51)';
onex=ones(size(x));
ax=[onex];
for i=1:n
 ax=[ax x.^i];
end
px=ax*c;                            %vector of polynomial values
deg=int2str(n);                     %integer to string for title
plot(x,px,'k',xd,yd,'ko');          %k=black, o=small circles
title(['degree=' deg])
xlabel('x-axis')
ylabel('y-axis')
```

The second example is of a function m-file which calls other function m-files.

```
function y=arref(R,m,a,b)
%Generates m x n matrix with integer entries a,a+1,...,b
%which has a specified r x n rref = R of rank r.  R, a, b are inputs.
dim=size(R);
r=dim(1);                     % number of rows of R
n=dim(2);                     % number of columns of R
P=prand(r,a,b);               %prand and arand are other function m-files
Q=arand(m-r,r,a,b);
y=[P;Q]*R;
```