

BEGINNING MATLAB

R.K. Beatson

Mathematics Department
University of Canterbury

Contents

1	Getting started	1
2	Matlab as a simple matrix calculator	2
3	Repeated commands	4
4	Subscripting, rows, columns and blocks	5
5	Edit, test, edit cycle	7
6	Functions and scripts	7
7	Input and output	9
8	Conditional branching	13
9	Finishing touches	15

1 Getting started

Matlab was originally a package for matrix algebra. It has evolved to include strong graphics abilities and an extensive programming language. It is available, in various versions, for various types of hardware: PCs, Macintoshes, SUN workstations, Vax's etc. On most of these systems Matlab will be started by entering the command

```
matlab
```

at the command prompt. This can however differ, depending on the whims of your system administrator. The command to exit Matlab is

`exit`

You can interrupt, or abort, execution of Matlab commands by entering a control `C`. To do this hold down the control key and, before releasing it, press the `C` key.

2 Matlab as a simple matrix calculator

The basic object in Matlab is a rectangular matrix with real or complex entries. Thus even a constant is viewed by Matlab as a 1×1 matrix. In entering a matrix, separate the elements in a row by spaces or commas, separate the rows by semi-colons, or by end of line characters. Thus the 2×2 matrix

$$a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

could be entered with the command

```
a = [ 1 2 ; 3 4 ]
```

or with the command

```
a = [ 1 2  
      3 4 ]
```

Similarly the 2×1 vector

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

could be entered with the command

```
x = [ 1 ; 1 ]
```

or the command

```
x = [ 1 1 ]'
```

where the prime (`'`) directs matlab to compute the transpose.

The three basic matrix arithmetic operations $+$, $-$, and \times are represented naturally by `+`, `-`, and `*`. The function `inv()` calculates the matrix inverse. For square matrices backslash, or left matrix division, is related

to left multiplication by an inverse, but is computed using Gaussian elimination. Thus $A \setminus b$ is roughly equivalent to `inv(A)*b`. Exponentiation is represented by `^`.

Matlab includes many other functions. For a listing of these simply enter the command `help`. For help on a particular command specify its name after the word `help`. For example `help size` will tell you how to determine the shape of matrix.

To compute operations on an element by element basis use the `.` operator. Thus for example `C = X.*Y` computes the element by element product of X and Y putting the result in the corresponding elements of C . Similarly `A.^3` stands for the element by element power, rather than the matrix power and `A./B` for element by element division.

The `format` or *number of significant figures* which Matlab uses to display its answers controlled by the `format` command. The options are

Format	Example
<code>format short</code>	1.5000
<code>format short e</code>	1.5000E+000
<code>format long</code>	1.5000000000000000
<code>format long e</code>	1.5000000000000000E+000

Exercises 2

- (1) Enter the arrays

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

into Matlab. Calculate $A * B$ and $B * A$. Why are these different? Calculate $B' * B$. Hence, or otherwise, say what type of matrix B is.

- (2) With A as above issue the commands `A^2` and `A.^2`. Why are the answers different?

- (3) Enter the matrix

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

into Matlab. Read Matlab's on line help on the `eig` function and hence determine numerically the eigenvalues and eigenvectors of A .

3 Repeated commands

The up-arrow key can be used to recall previous Matlab commands. Thus a surprisingly effective method for performing a short series of similar operations is the following. Firstly recall the appropriate previous command using the up and down arrow keys. Then edit it using the left and right arrow keys, and the delete, insert and backspace keys. Finally issue the newly modified command with the enter key.

The colon operator `:` is a simple method of generating a vector of equally spaced values. The syntax of the command is

```
variable = start[:increment]:end
```

where the square brackets indicate that the increment term may be omitted. In that case the increment defaults to 1. Thus

```
x = 1:4
```

generates the row vector

```
x = [ 1, 2, 3, 4 ]
```

The same vector could be generated, less efficiently, with a for loop

```
for k = 1:4
    x(k) = k;
end
```

The semi-colon in the statement `x(k) = k;` above, has been included to suppress printing of the result of the statement.

A typical use of the for loop would be generation of data for graphing as in the following code

```
h = .04
for k = 1:51
    x(k) = (k-1)*h*pi;
    y(k) = sin(x(k));
end
plot(x,y)
```

which generates a plot of $\sin(x)$. Matlab's execution can be speeded up by factors of up to twenty five by **vectorization**. The following **vectorized** code is faster than the previous for loop.

```
x = 0:0.04*pi:2*pi ;
y = sin(x);
plot(x,y)
```

In the above, the single statement $y = \sin(x)$, takes the sine of all 51 elements of x and puts the result in the corresponding elements of the vector y . In Matlab functions may be applied in this element by element manner to arrays.

Exercises 3

- (1) Generate a plot of $\cos(x)$ for $x \in [-\pi, \pi]$ by modifying the code above. Read the online help on Matlab's `title` command, and then put title on the plot.
- (2) Read the online help on the `surf` function. Note in particular that it can be called with arguments x , y and Z , being two vectors and an array respectively. These variables specifying the coordinates of a rectangular mesh and the values of a function at the grid points of that mesh. Hence get Matlab to plot a graph of the function $e^{-(x^2+y^2)}$ on the domain $[-2, 2] \times [-2, 2]$.

4 Subscripting, rows, columns and blocks

Matlab's subscripts begin at 1. Thus if x is a row vector with 5 elements these are numbered $x(1), \dots, x(5)$, rather than starting with $x(0)$. Arrays are subscripted in the usual manner, with `A(3,3)` for example standing for a_{33} . Powerful vector subscripts are also allowed so that `A(3:4,3:4)` specifies the 2×2 submatrix of A with top left element a_{33} . A colon (`:`) by itself represents all of a row or column. Thus elementary row or column operations may be performed easily in Matlab. For example the following command would subtract 3 times the second row of matrix A from the first and store the result back in the first row of A .

```
A(1,:) = A(1,:) - 3*A(2,:)
```

Exercises 4

- (1) Using Matlab as a calculator perform the forward elimination part of Gaussian elimination without partial pivoting on the tridiagonal matrix

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{bmatrix}$$

Initialize l as a 4×4 identity using Matlab's `eye` command and calculate and store the multipliers in l as you go. For example to reduce $a(2,1)$ to zero use the commands

$$\begin{aligned} l(2,1) &= a(2,1)/a(1,1) \\ a(2,:) &= a(2,:) - l(2,1)*a(1,:) \end{aligned}$$

Check your result by multiplying l ($=L$) by the final matrix a ($=U$).

- (2) The process of question (1) decomposes A into the form LU where U is the upper triangular matrix obtained from A by the Gaussian elimination, and L is the unit lower triangular matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ m_{21} & 1 & 0 & 0 \\ m_{31} & m_{32} & 1 & 0 \\ m_{41} & m_{42} & m_{43} & 1 \end{bmatrix}$$

whose entries are the multipliers. Noting that solution of $Ax = b$ can be performed by first solving $Ly = b$ for y , and then $Ux = y$ for x , determine the 4,4 element of A^{-1} . (*Hint: A times the fourth column of A^{-1} equals So the fourth column of A^{-1} is the solution of ...*)

- (3*) Consider an $n \times n$ tridiagonal matrix A with constant diagonals. 4 on the main diagonal, 1 on the super-diagonal, and 1 on the sub-diagonal. Considering the result of problem (2) above write down a recurrence for the n,n element of A^{-1} , denoted by $A_{n,n}^{-1}$. Hence calculate $\lim_{n \rightarrow \infty} A_{n,n}^{-1}$.

5 Edit, test, edit cycle

In developing programming code the programmer is inevitably involved in an edit, test, edit cycle. In Matlab the edit, test cycle is most conveniently done using the shell escape `!`. The command

```
! program_name parameters
```

runs the non-Matlab program *program_name* while leaving the Matlab session intact. For example, if your editor was `xedit` then the command

```
! xedit f.m
```

would invoke the `xedit` editor on the file “f.m”. When `xedit` was exited the Matlab session would be re-established ready to test the new version of “f.m”.

6 Functions and scripts

A script is a file with file type “.m” containing a list of Matlab commands. Invoking a script file is the same as issuing these commands, one by one, from the keyboard.

A function file differs from a script in that it has specific input and output *parameters*. Variables changed within a function are *local*, and a function changes the workspace only by the assignment of its *output* to a variable, or variables. As is usual for a high level language, the *actual parameters* (with which the function is called) and the *formal parameters* (named in the statement of the function) may differ.

The first example is a simple function to evaluate the $g(x) = x^2 - 2x$. The following lines would be edited into the file “g.m”.

```
function [y]=g(x)
% g(x)= x*x-2*x
y = x*x-2*x;
```

The function could then be invoked by issuing the command

```
u = g(1.5)
```

from within matlab. (Reminder: It is very important not to put extra spaces in Matlab expressions such as `x*x-2*x` as Matlab interprets *space* as a separator between elements in a row.)

The second example is a system of three function files for performing one step of a simple Newton iteration to find a zero of a function of two variables.

```
function [v]=f(x)
% Evaluate the vector valued function f and return
% the result.
%
% Syntax [v]=f(x)
v = [ x(1)*x(1)+x(2)*x(2)-2
      exp(x(1)-1)+x(2)^3 -2];

function [a]=jac(x)
% Evaluates the Jacobian of the function f
% at a point.
%
% Syntax [a]=jac(x)
a = [ 2*x(1)      2*x(2)
      exp(x(1)-1) 3*x(2)*x(2) ];

function [v]=nr(x)
% Makes a single step of Newton's method for finding
% a zero of a function of several variables.
% Calls functions f and jac.
%
% Syntax [v]=nr(x)
v = x-jac(x)\f(x);
```

These files would be created with an editor and named “f.m”, “jac.m” and “nr.m” respectively. Then issuing the commands

```
x = [2 2]
x = nr(x)
```

would perform one step of Newton’s method starting from $x = (2, 2)$.

The percentage sign (%) in the above examples starts a comment. Typing `help function_name`, as well as giving help on Matlab’s built in commands, will print any initial block of comments in a user defined function.

Exercises 6

- (1) The function

$$\psi(x, c) = \frac{1}{2} \left(\sqrt{(x+1)^2 + c^2} - 2 * \sqrt{x^2 + c^2} + \sqrt{(x-1)^2 + c^2} \right)$$

with $c \geq 0$, is of interest for modelling data. Write a Matlab function to evaluate this mathematical function. Plot a graph of this function when $c = 1$, on the domain $[-5, 5]$. Using Matlab also calculate an approximation to the infinite sum $\sum_{\ell=-\infty}^{\infty} \psi(x - \ell, c)$ at a few points in $[-1, 1]$. Can you guess what function the infinite sum converges to?

7 Input and output

If the file *filename* contains a rectangular array of figures with blanks as a separator within rows, then the command

```
load filename
```

will load the array into Matlab's workspace giving it a name formed by stripping the *filetype* (the stuff from the "." on) off the file name. Conversely, the command

```
save variable_name filename /ascii
```

will create a file *filename* containing the variable *variable_name* in ascii readable form.

Another way of importing data is via script files. For example, if the file *def_x.m* contained the lines

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

then the command *def_x* would cause this script to execute, thus defining *x*.

A simple means for displaying data and text is the `disp` command. For example the command

```
disp('This program solves band systems');
```

will display the message on a line. `disp` can also be used to display data. For example if x is the 1×3 vector `[1, 2, 3]` and the current *format option* is *short* then

```
disp(x);
```

will result in the output

```
1.0000 2.0000 3.0000
```

Note that `disp` does not echo the name of the variable as the command `x` would.

For greater control of the output format of numeric quantities use the *C-like* command `fprintf`. The syntax of `fprintf` is

```
fprintf([filename,] format [,x,y,z])
```

where the square brackets indicate that the *filename*, and the *numeric arguments* x , y , z , may be omitted.

If the *filename* is omitted output is sent to the screen. If the *filename* option is used and file *filename* does not exist then it is created. If it does exist, then the output is appended to it.

The format string specifies how the numbers are to be printed. Legal format specifiers are `%e`, `%f` and `%g`. Optionally a string of the form `n.m` can be placed between the `%` and the conversion character. The number to the right of the decimal specifies minimum field width and the number to the left, the number of decimal places. Finally to insert a newline character in the output string use the C format specifier `\n`.

The following is an example of a *script file* which creates a row vector with six elements and then prints these out using two `fprintf` statements.

```
a = [1 2 3 4 5 6];
fprintf('%e %e %e',a(1),a(2),a(3));
fprintf('%e %e %e \n',a(4),a(5),a(6));
```

The next example prints out a table of values k^2 against k .

```
for k=0:5
    fprintf('k = %3.0f k squared = %6.0f \n',k,k*k);
end
```

An alternative to `fprintf`-ing to a file is to use the `diary` command. An initial command

```
diary filename
```

erases the file *filename* and causes subsequent input and non-graphics output to be echoed to this file. Echoing of input and output to the file can be turned off and on mid-session with the commands

```
diary off
diary on
```

At the end of the Matlab session the diary file may be edited or sent to the printer.

A *function* or *script* can prompt for input using the `input` command. For example the line

```
n = input('Enter the value of n : ')
```

will issue the prompt *Enter the ...* and wait for the user to type in a number. When the user presses *Enter* the number entered will be assigned to the variable `n`. There is also a string version of the `input` command whose syntax is

```
variable = input(prompt string, 's')
```

For example the following code shows how to execute a loop a user specified number of times.

```
ans = 'y';
while ~strcmp(ans, 'n')
% while not string_compare (ans, 'n')
% So anything but 'n' continues! Even the
% empty string.
...
...
...
ans = input('Continue y/n [y] ? : ', 's');
end % while
```

Another useful command is `pause`, which causes processing to stop until a key is pressed, or until a specified number of seconds has elapsed. In the form


```

end
x_n = b_n/d_n
for k = n - 1 downto 1 do begin
    x_k = (b_k - c_k*x_{k+1})/d_k
end

```

Implement this algorithm as two Matlab functions, `felim` and `triback`. The function declarations should read

```

function [a, d, c] = felim(a,d,c,n)    and
function [x] = triback(a,d,c,b,n)

```

respectively. Test your routine on the system

$$\begin{bmatrix} 4 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 6 \\ 5 \end{bmatrix}$$

(Note that if the output of `felim` is assigned to the three vectors `[e, f, c]` and `triback` is called with the parameters `e, f, c, b, n` then `a` and `b` will not actually be changed by the `felim` routine.)

8 Conditional branching

Matlab has the following relational operators: `<`, `<=`, `>`, `>=`, `==` and `~=`, which can be used in comparing matrices of equal dimension. The `==` and `~=` operators test for equality and non-equality respectively. The output of a comparison is a matrix of zeros and ones, of the same size (often 1×1) as the input matrices. The value one standing for TRUE and zero for FALSE.

The relational operators may be used in conjunction with the `if` and `while` statements to control the flow of a program.

For example consider evaluating the function

$$f(x) = \begin{cases} 0, & \text{if } x = 0, \\ x^2 * \log(|x|), & \text{if } x \neq 0. \end{cases}$$

This is a perfectly well defined, and smooth, function with $f(0) = f'(0) = 0$. However, trying numerically to evaluate $\log(x)$, with x very small or zero, would cause an error. After all $\log(0)$ is undefined! This problem can be avoided by using an `if` statement as in the following function file.

```

function u=f(x)
% Evaluate a thin plate spline in one dimension
u = abs(x);
% eps is a Matlab reserved variable containing
% the machine epsilon.
if u < eps
% u is very close to zero so avoid attempting to
% evaluate log by using instead a Taylor series
% approximation to f.
    u = 0;
else
    u = u*u*log(u);
end

```

The `if` statement has an optional `elseif` clause and so can operate like the *case* statement of many computer languages.

Exercises 8

- (1) Consider the Newton-Raphson function of section 6. Recode this routine so that it takes as input a precision *precis*, maximum number of iterations *itcount*, and print frequency *prnfreq*. The routine is to terminate whenever the function value $y = f(x)$ and the step $s = -J(x)\backslash f(x)$ simultaneously have Euclidean norm less than *precis*, or the number of iterations exceeds *itcount*. Display the current value of the iteration counter, x , $\text{norm}(x)$, $f(x)$, $\text{norm}(f(x))$, s and $\text{norm}(s)$ every *prnfreq* steps including the first, and also on exit. Also display the reason the iteration was terminated. Try your code on the given function with a starting point of (2,0.7). (Hints: Use Matlab's `norm` function, and the AND operator `&`. Also modular programming is best: So write a function whose only job is to perform the printout of a single iterations result.)
- (2) As for (1) above but use *damped Newton*. Display the number of dampings within the current group in your printout, and count each damping towards the iteration count.

9 Finishing touches

The Matlab function `feval` enables one to pass *function names* as parameters. The syntax of the command is

```
feval('function',x1,...,xn)
```

which evaluates

```
function(x1, ... ,xn)
```

For example the function *myplot* below will plot a user specified function on the domain `[-4, 4]`

```
function myplot()
    fnm =input('Enter a function name e.g. sin ','s');
    x = -4:.1:4;
    y = feval(fnm,x);
    plot(x,y);
```

As with some other computer languages it is not necessary to supply *all the formal input or output parameters* when calling a function. The reserved variable `nargin` records the number of input variables a Matlab function was called with, while `nargout` records the number of output variables. By testing `nargin` your Matlab functions can supply reasonable defaults for omitted parameters, or print helpful messages. For example, the function *plus* below prints a helpful message if it is called with insufficient parameters

```
function u = plus(x,y)
    if nargin < 2
        disp('Insufficient parameters supplied !');
        disp('The desired syntax is ');
        disp('    u = plus(x,y)    ');
        return
    end % if
    u = x+y;
```