Introduction to MATLAB

This exercise[1] should be completed during your first few weeks before Modelling of Dynamic Systems commences. It should take you about 2 hours if have had no previous experience with MATAB. If you have any problems, please email

`j.f.whidborne@cranfield.ac.uk`

This exercise can be run using any Cranfield University machine. The lab is not assessed.

1 What is MATLAB?

MATLAB is a computer program for people doing numerical computation, especially linear algebra (matrices). It began as a "MATrix LABoratory" program, intended to provide interactive access to the libraries `Linpack` and `Eispack`. It has since grown well beyond these libraries, to become a powerful tool for visualization, programming, research, engineering, and communication. MATLAB 's strengths include cutting-edge algorithms, enormous data handling abilities, and powerful programming tools. MATLAB is not designed for symbolic computation, but does include a Maple kernel for such computations. The interface is mostly text-based, which may be disconcerting for some users.

MATLAB is packaged as a core program with several "toolboxes", sold separately. The university installation includes nearly all the toolboxes.

2 Getting started

MATLAB may take 15 to 20 seconds to load. Eventually, you should see several windows, menus and a "Command" window with the line

```
>>
```

which is the MATLAB prompt. The Command window is where you can interact with MATLAB directly. There are several other windows that you can view.

To check the version of MATLAB type (at the MATLAB prompt)

```
ver
```

To exit MATLAB , type

```
exit
```

or choose `File->Exit MATLAB`. To change to your Z: drive, type

```
cd z:
```

The Command Window

Typing the following:

```
3*12
```

should give

```
ans =
36
```

MATLAB inserts extra blank lines between practically everything. To turn off this feature, type

```
format compact
```

Previous input lines are recorded in the command history panel/window. Alternatively you can recall previous input lines by typing the "up" cursor button. If your type some text before hitting the "up" cursor button, MATLAB will recall lines matching the text you've type so far.

If you know the name of a MATLAB function you need help with, type

```
help functionname
```

to see the help text contained in the function definition itself. If you don't know the name of the function you need, try, for example,

```
lookfor cartesian
```

[1] Acknowledgements are due to the staff of the Indiana University Stat/Math Center and to Francisco Javier Sayas, Department of Mathematical Sciences, University of Delaware for borrowed material.

MATLAB then searches through the first line of help text of each function for the keyword. However, the search is very slow, and often deluges the user with many items. In addition, a search for a concept, such as "matrix" will almost never return what you need.

A more useful set of help files and documentation manuals is included. To access just type

```
doc
```

Exercise 1. Spend 5 minutes exploring the MATLAB section of the documentation.

How MATLAB Works

MATLAB works by executing the mathematical statements you enter in the command window. By default, any output is immediately printed to the window. You are also allowed to assign a name to an expression for your convenience. Keep in mind that the name you assign is only a name, and it does not represent a mathematical variable. Every name must have a value at all times. If you try to read the value of an unassigned name, you will get an error.

Nearly everything in MATLAB in a matrix, whether it looks like it or not. This takes some getting used to. We'll be introducing matrix-style operations along with their scalar counterparts so you can understand the patterns that arise in the syntax.

Basic Syntax

MATLAB was designed to use fairly standard notation. Arithmetic works as expected. Note that the result is given the name `ans` each time.

```
2 + 3
7-5
34*212
1234/5786
4^2
```

Note that `^` denotes "raise to the power of". You can choose your own names for things.

```
a = sqrt(2)
```

Putting a semi-colon at the end on the line suppresses the display of the answer. The result is still calculated though. Try

```
b=sin(.5);
b
```

You can use commas to put more than one command on a line. `pi`, `i`, `j` and `eps` are pre-defined variables. Variables `i` and `j` both represent the imaginary number, hence the expressions `3+2i`, `3+2*i`, `3+2j`, `3+2*j` and `3+2*sqrt(-1)` all have the same value. Try

```
c = a, pi, d=2 + 3i, conj(d)
```

Exercise 2. Check the well known identity

$$1 + e^{j\pi} = 0$$

by

```
1+exp(pi*j)
```

Why is the answer not exactly zero?

Try

```
c = sin(pi)
eps
```

`eps` is the current limit of precision. Anything smaller than `eps` is probably zero. You can overwrite these variables (but this is not advised as good MATLAB programming practice)

```
i=5
```

To return to the default value, type

```
clear i
```

Note that MATLAB understands (and expects you to understand!) scientific notation. Vectors can be entered as follows

```
d = [1 2 3 4 5 6 7 8 9]
e = [1:9]
f = 1:9
```

Note the use of the ":" operator - it counts (by ones) from one number to the next. To count in other increments try

```
g = 0:2:10
h= 10:-2:0
```

The colon can also be used in other ways. You can use it to get slices of a vector (or matrix, or cube, etc), or get the whole thing.

```
f(3)
f(2:7)
f(:)
```

A single quote "' " computes the transpose of a matrix, or in this case, switches between row and column vectors.

```
h=[1 2 3]
h * h'
h .* h
h + h
```

"*" is matrix (or vector) multiplication, and so the dimensions must line up correctly. ".*" is entry-by-entry multiplication (otherwise known as the Hadamard product, the entrywise product or the Schur product). A matrix is entered by

```
g = [ 1 2 3; 4 5 6; 7 8 9]
```

Elements, rows and columns can be extracted:

```
g(2,3)
g(:,1)
g(3,:)
```

or assigned:

```
g(2,3) = 4
g(3,[1:2]) = [-2 -3]
```

How to Control Output

Two useful commands are format and more.

To control line spacing, use

```
format compact
```

To see all 15 digits that were used in calculation, use

```
format long
```

To see just 5 digits, use

```
format short
```

To suppress output completely, use a semi-colon at the end of the command. To see one page of output at a time use

```
more on
```

Note that MATLAB always uses double precision (about 15 digits) in its calculations. These commands merely adjust the display

3 Matrices

Try the following. If it is not obvious to you what the operation means, use the help facility.

```
X=ones(3,2)
Y=ones(2)
Z=eye(3)
```

New matrices may be formed out of old ones. Try

```
a = [1 2; 3 4]
[a, a, a]
[a; a; a]
[a, zeros(2); zeros(2), a']
```

Matrices may also be constructed (very inefficiently) by programming. Here is an example using two "for" loops.

```
for irow=1:10,
   for jcol=1:10,
      t(irow,jcol) = irow/jcol;
   end
end
```

Note that we use `irow` and `jcol` rather than `i` and `j` so as not to confuse with the MATLAB imaginary numbers `i` and `j`

MATLAB has all the common operations built-in, as well as most of the obscure ones. Try

```
k = [16  2  3;
     5  11 10;
     9   7  6]
size(k)
diag(k)
rank(k)
det(k)
inv(k)
```

The eigenvalues can be obtained. Try

```
[vec,val] = eig(k)
```

The columns of `vec` are the eigenvectors, and the diagonal entries of `val` are the eigenvalues.

There are many more functions like these. Type `help matfun` to see them all.

Solving Linear Equations

One of the main uses of matrices is in representing systems of linear equations. Consider the set of simultaneous linear equations represented by

$$Ax = b$$

where x is the vector of unknowns. This can be solved in MATLAB as follows

```
A = [1 2 3; 4 5 6; 7 8 10]; b = [1 1 1]';
x= inv(A)*b
```

or by

```
x = A \ b
```

This can be read "x equals A-inverse times b." To check the solution type

```
A*x, A*x - b, eps
```

Notice that `A*x - b` is very close to `eps`; which means that it is as close to zero as possible and a well-conditioned problem.

Saving and loading matrices

If you would like a record of your work, you can turn on "logging" by typing

```
diary session.txt
```

Notice that output will be saved alongside your input, so the file can't be used directly as a script. Stop recording with

```
diary off
```

Alternatively copy from the MATLAB Command Window and paste into a text file — this only saves the commands.

You may just want to save one or more matrices. The diary is a very clumsy way to do this. To save the value of the variable `x` to a plain text file named `x.txt` use

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

Under Windows you may open the appropriate file with Notepad. To save all variables in a file named `mysession.mat` in re-loadable format, use

```
save mysession
```

To restore the session, use

```
load mysession
```

4 Graphics

The graphics capabilities of MATLAB are very good. Try

```
x = -10:0.5:10;
y = x.^2;
```

Now plot the points. Each pair is plotted, so `<x(1), y(1)>` is a point, `<x(2), y(2)>` is a point, etc.

```
plot(x,y)
```

Two or more plots can be plotted on the same axes:

```
x = -10:0.5:10;
plot(x,sin(x))
hold on
plot(x,cos(x))
hold off
```

The Plots may be edited using the toolbar functions, and arrows, lines, and text comments can be added to your plot. Try this. Plots may also be saved in a variety of formats (`.emf` is best for Microsoft Word) including `.fig` which can be loaded by MATLAB .

Try the following to get a good idea of how plotting works

```
t = 0:0.1:2*pi;
x = cos(t);
y = sin(t);
t = 0:pi/5:2*pi;
u = cos(t);
v = sin(t);
plot(u,v)
plot(x,y, 'ro-')
plot(x,y, 'r-', u,v,'b*:')
figure(2)
subplot(1,2,1)
plot(x,y)
title('Fine Mesh')
subplot(1,2,2)
plot(u,v)
title('Coarse Mesh')
```

Log scale plotting is also possible with `semilogx`, `semilogy` and `loglog` commands.

Implicit functions can also be plotted. For example, the curve of the function

$$f(x,y) = x^2 \sin(x+y^2) + y^2 e^{x+y} + 5\cos(x^2+y) = 0$$

cannot be computed and plotted easily with the `plot` command. The following command should be used:

```
ezplot('x^2 *sin(x+y^2) + y^2 * exp(x+y) + 5*cos(x^2 +y)')
```

The plot range can be adjusted by

```
ezplot('x^2 *sin(x+y^2) + y^2 * exp(x+y) + 5*cos(x^2 +y)' ,[-10 10])
```

Three dimensional plotting is also possible. Plot a curve in 3-d space by

```
figure(3)
t = -4*pi:pi/16:4*pi;
x = cos(t);
y = sin(t);
z = t;
plot3(x,y,z)
```

This command is useful to show ground tracking

```
stem3(x,y,z)
```

Plot a surface in 3-d space by

```
figure(4)
[x, y] = meshgrid(-3:0.1:3,-3:0.1:3);
z = x.^2 - y.^2;
mesh(x,y,z)
surf(x,y,z)
plot3(x,y,z)
```

5 Polynomials and Curve Fitting

MATLAB has a specific Curve Fitting Toolbox, but here we will just explore the basic curve fitting facilities, that fit polynomials to data. The Curve Fitting Toolbox has more advanced facilities such as spline fitting. Note that MATLAB can import data from various different file formats, see the documentation for details.

To define a polynomial, for example,

$$p(x) = x^4 - 10x^3 + 35x^2 - 50x + 24$$

enter

```
p=[ 1   -10    35   -50    24]
```

Obtain the roots of the polynomial $p(x) = 0$ by

```
r=roots(p)
```

Now enter

```
poly([1 2 3 4])
```

and note the result.

Evaluate the polynomial $p(x)$ at $x = 2$ by

```
polyval(p,2)
```

We can plot the polynomial function $y = p(x)$ by, for example,

```
x=0:0.1:5;
y=polyval(p,x);
plot(x,y)
```

Type

```
grid
```

and check that the roots of the polynomial are correct.

Curve fitting

Define some test data, for example

```
x = [1 2 3 4 5];
y = [5.5 43.1 128 290.7 498.4];
```

A second degree (quadratic) polynomial that approximately fits the data is

```
p = polyfit(x,y,2)
```

Let us see the result of the curve fitting by

```
x2 = 1:.1:5;
y2 = polyval(p,x2);
plot(x,y,'o',x2,y2)
grid on
```

Exercise 3. Repeat the curve fitting for a straight-line fit and a 3rd degree polynomial fit. Calculate a measure of the curve fit accuracy and compare the results for the three curve fits.

The Curve Fitting Toolbox includes a gui tool, `cftool`, that provides a user-friendly interface to the curve-fitting utilities.

6 Programming

Scripts

MATLAB 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 `filename.m`).

Select `File -> New ->Script` to open MATLAB 's built-in editor/debugger. Enter the following code.

```
% comments: my first m-file
[x y] = meshgrid(-3:.1:3, -3:.1:3);
z = x.^2 - y.^2;
mesh(x,y,z);
```

Save as `sketch.m`:

Next, in MATLAB , be sure the current folder is the same folder that you saved the file. Now type

```
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];
```

then the command `mymatrix` reads that file and generates `A`.

Functions

Functions are like any other m-file, but they accept arguments, and they are compiled the first time they are used in a given session (for speed). Use the editor to create a file named `sqroot.m`, containing the following lines.

```
function xroot=sqroot(x)
% SQROOT compute square roots by Newton's method

% Initial guess
xrootguess = 1;

for icount = 1:100
   xrootnew = (xrootguess + x/xrootguess)/2;
   disp(xrootnew);
   if abs(xrootnew - xrootguess)/xrootnew < eps, break, end;
   xrootguess = xrootnew;
end
xroot=xrootnew;
return
```

Type

```
help sqroot
```

and you will see that MATLAB now recognizes the function.

In MATLAB enter the commands

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

You should see the output from your function. Now compare the result with the results calculated by MATLAB using 4 different methods:

```
sqrt(19)
19^(0.5)
exp(log(19)/2)
10^(log10(19)/2)
```

Two caveats:

- A function has access to the variables in the workspace from which it was called, but the variables created within the function (`xstart` and `xnew`, in the preceeding example) are local, which means that they are not shared with the calling workspace.
- Note: if you edit a function during a session, use `clear function_name` to remove the compiled version, so that the new one will be read.

Exercise 4. Extend `sqroot.m` so that the function can return the square root of negative numbers. Type `help if` to learn how to do conditional statements in MATLAB .

Vectorizing functions

A scalar function of one variable is vectorized when if the input is an array, the output is an array of the same size. For scalar functions of several variables, vectorization assumes that in the input variables we give arrays of the same size and we expect as output an array of the given size.

For construction of vectorized functions, you can count on the many capabilities of MATLAB in this direction:

- The symbols for sum and subtraction are vectorized.
- Addition of scalars, change of sign and multiplication by scalars are vectorized.
- For element by element multiplication and division use the symbols
  `.*`
  and
  `./`
  respectively.
- The symbol
  `.^`
  is used to get the element by element power of a matrix by a number.
- Most common mathematical functions are vectorized. For instance: `abs`, `sin`, `cos`, `exp`, `log`, etc are vectorized.

For example

```
1 + [2 3 5]
[2 3;-1 2].^4
[3 5;1 2].*[2 1;-1 4] % element by element product
[3 5;1 2]*[2 1;-1 4] % matrix product
```

If you do not how to vectorize an expression, you can ask MATLAB to do it for you, using the command vectorize. E.g.

```
vectorize('x^2+sin(x)*y')
```

Anonymous functions

There is another way of defining functions in the MATLAB workspace known as anonymous functions. Try this vectorized function of one variable:

```
f =@(x) x.^2+2
whos f
f([1 2 3])
```

Or a vectorized function of two variables:

```
g=@(x,y) x.^2+y.^2
g([1 2 3],[4 5 6])
```

Or to return the function $\sqrt{a^2+b^2-2ab\cos\theta}$ by

```
c = @(a, b, theta) sqrt(a.^2+b.^2-2*a.*b.*cos(theta))
SideThree = c(2, 3, pi/6)
```

Note that anonymous functions can only have one expression and can only return a single variable (though that variable can be a matrix). However, you can construct complex expressions by combining multiple anonymous functions. Anonymous functions can also be passed to functions.

JF Whidborne
Centre for Aeronautics
Cranfield University
October 18, 2016