NM262 - Lecture 01

Introduction to MATLAB

Prof. JAC Weideman

What is MATLAB?

From Wikipedia:

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, and Fortran.

In 2004, MATLAB had around one million users across industry and academia. MATLAB users come from various backgrounds of engineering, science, and economics. MATLAB is widely used in academic and research institutions as well as industrial enterprises.

Where is MATLAB used?

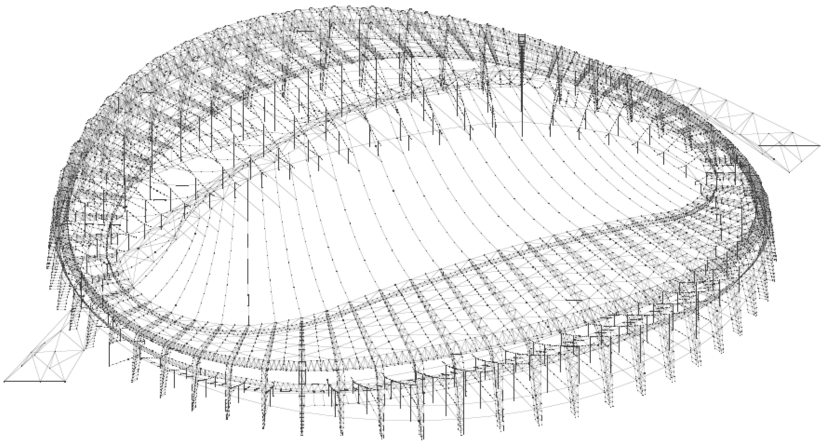
MATLAB is one of the major computing environments for Computational Science and Engineering.

From Wikipedia:

Computational science and engineering (CSE) is a relatively new discipline that deals with the development and application of computational models and simulations, often coupled with high-performance computing, to solve complex physical problems arising in engineering analysis and design (computational engineering) as well as natural phenomena (computational science). CSE has been described as the “third mode of discovery” (next to theory and experimentation).

Can you give an example, please?

Structural analysis: the determination of the effects of loads on physical structures and their components.



Introduction to NM262 and MATLAB

Numerical Methods are the tools of CSE

We shall cover:

Rootfinding (solving equations)	Ch 2 (2nd ed), Ch 3 (3rd ed)
Matrix computations	Ch 3 (2nd ed), Ch 4 (3rd ed)
Fitting curves to data	Ch 4 (2nd ed), Ch 6 (3rd ed)
Approximate differentiation	Ch 5 (2nd ed), Ch 8 (3rd ed)
Approximate integration	Ch 6 (2nd ed), Ch 9 (3rd ed)
Ordinary DEs (initial value)	Ch 7 (2nd ed), Ch 10 (3rd ed)
Ordinary DEs (boundary value)	Ch 8 (2nd ed), Ch 11 (3rd ed)
Solving partial differential equations	Chapter 7, Zill

We shall solve mostly toy problems, but the principles are the same for real-life problems

Introduction to MATLAB

Beginning at the Beginning

MATLAB can be used in one of two ways:

- Interactive mode (like a command-line calculator)
- M-file mode (by writing script or function files)

Unless it's a really quick calculation, M-file mode is recommended

M-file mode will be described in A.5 Script Files

For now we use interactive mode, in the command window

Protip: Use the up-arrow for line retrieval in interactive mode

Introduction to MATLAB

A.2 Starting with MATLAB

```
>> 2+3

ans =     5

>> ans*7

ans =     35

>> 7*12;  %<- semi-colon suppresses output, but still evaluates expression
>> ans

ans =  84

>> x = 2^6                          >>x = x+2

x =   64                             x = 66
```

Sometimes good to use words as variable names

```
>> radius = 10;
>> Area = pi*radius^2

Area =  314.1593
```

Note π is a built-in constant, as is the complex unit *i* (or *j*)

```
>> i^2

ans = -1
```

```
>> sqrt(-4)

ans = 0.0000 + 2.0000i
```

Note `sqrt` is a built-in function for $\sqrt{\ }$, as is `exp` for e^x and many others:

```
>> e =  exp(1)

e = 2.7183
```

```
>>log(1000)

ans = 6.9078
```

```
>> log10(1000)

ans = 3
```

```
>> cos(pi/3)

ans = 0.5000
```

Numbers in exponential form: For example, the speed of light $c = 3 \times 10^8$

```
>> c = 3e8      %(yes)
>> c = 3*10^8   %(no)
```

MATLAB's precision is about 16 decimal digits

```
>> a = 4/3;
>> b = a-1;
>> c = 3*b;
>> e = 1-c

e =  2.2204e-16
```

The latter number equals 2^{-52} and is called machine epsilon.
Reading assignment: There is a link on the class web page to the article "Floating Point Numbers" by Cleve Moler (creator of MATLAB). Pay attention to the built-in constants `eps`, `realmin`, `realmax`.

MATLAB makes provision for infinity and undefined numbers

```
>> pi/0
ans =
   Inf
```

```
>> 0/0
ans =
   NaN
```

```
>> Inf-Inf
ans =
   NaN
```

MATLAB follows the usual order of operations for arithmetical expressions

- first exponents and roots
- then multiplication and division
- and finally addition and subtraction

When in doubt, use brackets

```
>> 6/2*4
ans =
    12
```

```
>> 6/(2*4)
ans =
    0.7500
```

```
>> 2^3*4
ans =
    32
```

```
>> 2^(3*4)
ans =
    4096
```

Displayed output is controlled by the `format` command

```
>> pi

ans =  3.1416
```

```
>> format long
>> pi

ans = 3.141592653589793
```

```
>> format short
>> pi

ans = 3.1416
```

`format` has no effect on the working precision, it remains ≈ 16 decimal digits

For exponential notation, use `format long e` or `format short e`

```
>> format short e
>> exp(pi^2)

ans = 1.9334e+04
```

```
>> format long e
>> exp(pi^2)

ans = 1.933368907436514e+04
```

Row and column vectors are entered as follows

```
>> rv = [3 1 4 -1]
rv =

     3     1     4    -1
```

```
>> cv = [3; 1; 4; -1]
cv =
     3
     1
     4
    -1
```

Matrices as follows

```
>> M = [2 3 -1 0; 3 4 -9 1; -1 2 6 4]  % <-- semi-colon separates rows

M =
     2     3    -1     0
     3     4    -9     1
    -1     2     6     4
```

Consider again the column vector and the matrix

```
>> cv = [3; 1; 4; -1]
cv =
     3
     1
     4
    -1
```

```
>> M = [2 3 -1 0; 3 4 -9 1; -1 2 6 4]
M =
     2     3    -1     0
     3     4    -9     1
    -1     2     6     4
```

Entries can be extracted as follows

```
>> cv(2)
ans =
     1
```

```
>> cv(2:4)
ans =
     1
     4
    -1
```

```
>> M(2,3)
ans =
    -9
```

```
>> M(2:3,4)
ans =
     1
     4
```

```
>> M(:,3)
ans =
    -1
    -9
     6
```

Revision: Row and column vectors are entered as follows

```
>> rv = [3 1 4 -1]
rv =

     3     1     4    -1
```

```
>> cv = [3; 1; 4; -1]
cv =
     3
     1
     4
    -1
```

Matrices as follows

```
>> M = [2 3 -1 0; 3 4 -9 1; -1 2 6 4]  % <-- semi-colon separates rows

M =
     2     3    -1     0
     3     4    -9     1
    -1     2     6     4
```

Useful built-in functions for vectors and matrices are `size`, `ones`, `zeros`, `eye`

```
>> size(rv)
ans =
        1     4
```

```
>> size(cv)
ans =
        4     1
```

```
>> size(M)
ans =
        3     4
```

```
>> ones(3)
ans =
     1     1     1
     1     1     1
     1     1     1
```

```
>> ones(size(M)) % <-- zeros works the same way
ans =
     1     1     1     1
     1     1     1     1
     1     1     1     1
```

```
>> eye(3)    % <-- Eye-dentity matrix, get it?
ans =
     1     0     0
     0     1     0
     0     0     1
```

Multiplication of scalars, vectors, matrices obeys rules of linear algebra

Scalar-vector multiplication

```
>> rv = [3 1 4 -1]

rv =
     3     1     4    -1
```

```
>> s = 2
s  =  2
```

```
>> s*rv
ans =
     6     2     8    -2
```

Scalar-matrix multiplication

```
>> M = [2 3 -1 0; 3 4 -9 1; -1 2 6 4]
M =
     2     3    -1     0
     3     4    -9     1
    -1     2     6     4
```

```
>> (-3)*M
ans =
    -6    -9     3     0
    -9   -12    27    -3
     3    -6   -18   -12
```

Matrix-vector multiplication

```
>> M = [2 3 -1 0; 3 4 -9 1; -1 2 6 4]
M =
     2     3    -1     0
     3     4    -9     1
    -1     2     6     4
```

```
>> cv = [3; 1; 4; -1]
cv =
     3
     1
     4
    -1
```

```
>> M*cv
ans =
     5
   -24
    19
```

Dimensions must agree

```
>> rv = [3 1 4 -1]

rv =

     3     1     4    -1
```

```
>> M*r
Error using  *
Inner matrix dimensions must agree.
```

Matrix-matrix multiplication

```
>> N = [1 0 -1; 2 3 0; 4 1 -2]
N =
     1     0    -1
     2     3     0
     4     1    -2
```

```
>> M = [2 3 -1 0; 3 4 -9 1; -1 2 6 4]
M =
     2     3    -1     0
     3     4    -9     1
    -1     2     6     4
```

Dimensions must agree

```
>> N*M
ans =

      3     1    -7    -4
     13    18   -29     3
     13    12   -25    -7
```

```
>> M*N
Error using  *
Inner matrix dimensions must agree.
```

Transpose of a matrix or vector:

```
>> M = [2 3 -1 0; 3 4 -9 1; -1 2 6 4]
M =
     2     3    -1     0
     3     4    -9     1
    -1     2     6     4
```

```
>> M'
ans =
     2     3    -1
     3     4     2
    -1    -9     6
     0     1     4
```

Two ways of computing the dot product (inner product) of two column vectors:

```
>> v1 =
     3
     1
     4
    -1
```

```
>> v2 =
     2
     4
    -5
     1
```

```
>> dot(v1,v2)
ans =
     -11
```

```
>> v1'*v2
ans =
     -11
```

Protip: Careful with transposes if your vectors or matrices have complex entries

Introduction to MATLAB

A.4 Mathematical Operations with Arrays

In addition to operations that obey the rules of linear algebra, MATLAB created new rules such as dot-multiply, dot-divide, dot-exponentiation

```
>> v1 =

     3
     1
     4
    -1
```

```
>> v2 =

     2
     4
    -5
     1
```

```
>> v1.*v2
ans =
     6
     4
   -20
    -1
```

```
>> v1./v2
ans =
    1.5000
    0.2500
   -0.8000
   -1.0000
```

```
>> v2.^2
ans =
     4
    16
    25
     1
```

Example of usage: Compute $y = 4 + 2x - x^2 - 3x^3$ at $x = 1, 2, 3$

```
>> x = [1; 2; 3]
x =
     1
     2
     3
```

```
>> y = 4+2*x-x.^2-3*x.^3   %<-- Q: Why not 2.*x?
y =
     2
   -20
   -80
```

The dot-operations work for matrices in the same way.

Note: Squaring a matrix and dot-squaring it is different

```
>> A =                          >> A^2 =                          >> A.^2 =
     2     3    -1                        3    10     7                       4     9     1
     0     1     4                        4    -3    16                       0     1    16
     1    -1     3                        5    -1     4                       1     1     9
```

Question: Why is there no dot-add nor dot-subtract?

The colon operator generates vectors with uniformly spaced entries:

```
>> [1:4]
ans =
1     2     3     4
```

```
>> [3:2:9]
ans =
 3     5    7     9
```

```
>> [3:-1:1]
ans =
3     2     1
```

The built-in function linspace does something similar

```
>> linspace(0,2,5) % <-- In [0,2], create 5 evenly spaced points
ans =
         0    0.5000    1.0000    1.5000    2.0000
```

Example of usage: Compute $y = 4 + 2x - x^2 - 3x^3$ at a 1000 points in $[0, 4]$

```
>> x = linspace(0,4,1000);
>> y = 4+2*x-x.^2-3*x.^3;  %<-- The graph can be plotted by >>plot(x,y)
```

The built-in mathematical functions can be applied to vectors and matrices. The results are computed componentwise.

```
>> x = [0 pi/6 pi/3 pi/2]
x =    0     0.5236     1.0472     1.5708
>> sin(x)
ans = 0     0.5000     0.8660     1.0000
```

Together with linspace and the dot-operations, this feature is useful for plotting graphs (see A.6). For example, consider $y = \sin x + \cos^2 x$ on $[0, 2\pi]$

```
>> x = linspace(0, 2*pi, 1000);
>> y = sin(x) + cos(x).^2;
>> plot(x, y)
```

Introduction to MATLAB

Built-in functions

There are many built-in functions for creating and manipulating data

The built-in help function can give information about which functions exist and how they are used

- `>> help elmat` lists the elementary functions for the assembly and manipulation of matrices
- `>> help elfun` lists the elementary math functions, including trig, exponential, complex, and rounding functions

Useful examples:

```
zeros(), ones(), rand(), randn(), diag(), eye(), meshgrid()

sin(), cos(), tan(), asin(), acos(), atan(), exp(), log(), log10()

abs(), sqrt(z), round(), ceil(), floor(), sign()
```

MATLAB has powerful built-in graphics functions:

The main ones are

- `plot` (line plot in 2D)
- `plot3` (line plot in 3D)
- `mesh`, `surf` (mesh/surface plot in 3D)

MATLAB allows you to label and annotate your graphs using the commands

- `title`
- `xlabel`
- `ylabel`
- `legend`

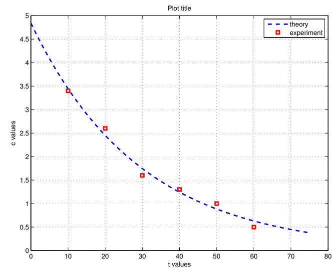
Here are data values for a chemical concentration (c) versus time (t)

t (min)	10	20	30	40	50	60
c (ppm)	3.4	2.6	1.6	1.3	1.0	0.5

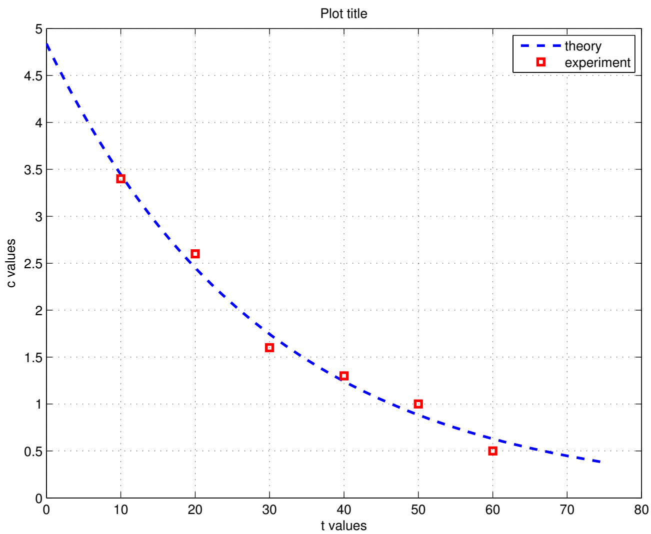
In theory this relationship can be described by $c = 4.84e^{-0.034t}$

The following lines of code display the data using square symbols and plot the graph using a dotted line for $t = 0$ to $t = 75$ min.

```
tdata = 10:10:60;
cdata = [3.4, 2.6, 1.6, 1.3, 1, 0.5];
t = linspace(0, 75, 100);
c = 4.84*exp(-0.034*t);
plot(t, c, 'b--', 'LineWidth', 2), hold on % <-- plot and hold
plot(tdata, cdata, 'rs', 'LineWidth', 2); hold off
title('Plot title'), xlabel('t values'), ylabel('c values')
legend('theory', 'experiment') % <-- add a legend
grid on % <-- show a grid
```



A.6 Plotting - example



MATLAB can display different colors, point styles (symbols), and line styles.

Examples:

- `'-b'` - blue line with no points (default)
- `'ro:'` - red dotted line with circles at the points
- `'gd'` - green diamonds at the points with no line
- `'m--'` - magenta dashed line with no point symbols

Other useful options:

- `'LineWidth'`, `'FontSize'`, `'MarkerSize'` - Change the line width, font of labels, size of symbols
- `hold on`, `hold off` - Don't overwrite the plot
- `axis equal` - Set the x and y axis to have the same scale
- `grid on` - Show a grid on the plot

Scripts are useful for running several lines of code. A script file can be opened in the editor from the toolstrip at the top, or from the command line as in

```
>> edit helloworld.m
```

`.m` is the extension MATLAB uses for its files – referred to as "M-files"

Very useful advice: begin each script with (something like)

```
%%HELLOWORLD.M  This script says 'hello' to the world.
% B Simpson - 23/07/2014.
```

These lines will be displayed when you type `>> help helloworld`

You can run a script from the command line, as in `>> helloworld`
Alternatively, by clicking on the Green arrow marked Run in the editor.

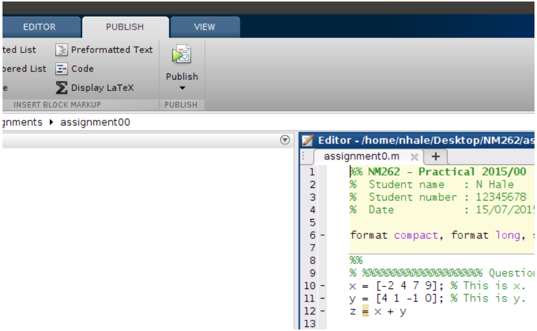
A.5 Scripts - example

```
%% NM262 - Practical 2015/01
%  Student name   : N Hale
%  Student number : 12345678
%  Date           : 15/07/2015

format compact, format long, set(0, 'DefaultLineLinewidth', 2)

%%
% %%%%%%%%%%%%%%%%%%%% Question 1: %%%%%%%%%%%%%%%%%%%%
x = [-2 4 7 9];
y = [4 1 -1 0];
z = x + y

%%
% You can add comments like this.
v = y - x

%%
% The double %% serves to break up the output into blocks.
a = x.*y

%%
% %%%%%%%%%%%%%%%%%%%% Question 2: %%%%%%%%%%%%%%%%%%%%
x = linspace(-5, 5);
y1 = x.^3 - 5*x.^2 + 7*x - 12;
y2 = sin(x.^2) - 5*x.^2 + 2;
plot(x, y1, x, y2), grid, shg
xlabel('x'), ylabel('y')

 ...
```

Introduction to MATLAB

A.5 Scripts - publishing

Scripts like the one on the previous page can be published using the Publish button 



For assignments you should print the resulting html file and submit hardcopy.
(Do not simply print the .m file! – You will get 0 marks.)

Anonymous functions are used to create simple functions with a single scalar, vector or matrix as output:

```
>> f = @(x) exp(x.^2)./sqrt(x.^2 + 5);
```

The function is evaluated in the obvious way:

```
>> f(2)
ans =
   18.1994
```

Because we used dot operations we can evaluate a matrix input and receive a matrix output:

```
>> f([-1 0 ; 1 2])
ans =
    1.1097    0.4472
    1.1097   18.1994
```

Anonymous functions may have many inputs:

```
>> h = @(x,y) 2*x.^2 - 4*x.*y + y.^2;
>> h(2,3)
ans =
      -7
```

Anonymous functions can also define vector-valued or matrix-valued functions:

```
>> g = @(x,y) [2*x-y, 3-x-y]
>> g(2,4)
ans =
     0    -3
```

Plotting an anonymous function

```
>> f = @(x) sin(x)+cos(x).^2;
>> x = linspace(0, 2*pi, 1000);
>> plot(x, f(x))
```

It is usually more efficient for MATLAB to perform calculations on an entire array rather than processing it element by element:

Compare:

```
% For loop
k = 0;
for t = 0:0.02:50
    k = k + 1;
    y(k) = cos(t);
end
```

```
% Vectorisation
t = 0:0.02:50;
y = cos(t);
```

It is often good practice in MATLAB to avoid **`for`** loops.

Try and replace them with built-in functions or linear algebra operations for better efficiency.

Example:

Compute $S = \frac{1}{1^2} + \frac{1}{2^2} + \cdots + \frac{1}{10^2}$

```
% For loop
S = 0;
for k = 1:1:10
    S = S + 1/k^2;
end
```

```
% Vectorisation
n = 1:10;
S = sum(1./n.^2);
```

Here is a vectorized version of the coin toss simulation we saw earlier:

```
%% COIN TOSS SIMULATION.
>> p = rand(1000, 1);
>> [sum(p < 0.5), sum(p > 0.5), sum(p == 0.5)]
ans =
   493   507     0
```

Useful functions to help vectorize your code: **sum**, **cumsum**

```
>> rv = [3 1 4 -1]
rv =
     3     1     4    -1
```

```
>> sum(rv)
ans =
     7
```

```
>> cumsum(rv)
ans =
     3     4     8     7
```

prod, **cumprod**

```
>> n = 4
n =
     4
```

```
>> nfact = prod(1:n)
ans =
    24
```

```
>> nfacts = cumprod(1:n)
ans =
     1     2     6    24
```

sort

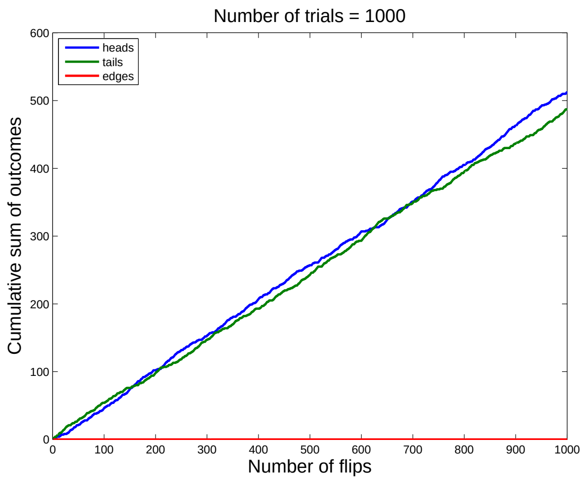
```
>> rv = [3 1 4 -1]
rv =
     3     1     4    -1
```

```
>> sort(rv)
ans =
    -1     1     3     4
```

Script file for plotting the cumulative sums of the outcomes of the coin flip examples:

```
nf = 1000;
p = rand(nf, 1);
flips = [cumsum(p < 0.5), cumsum(p > 0.5), cumsum(p == 0.5)];
nn = 1:nf;
plot(nn, flips(:,1), nn, flips(:,2), nn, flips(:,3), 'LineWidth', 2)
legend('heads', 'tails', 'edges', 'Location', 'NorthWest')
xlabel('Number of flips', 'FontSize',16)
ylabel('Cumulative sum of outcomes', 'FontSize', 16)
title(['Number of trials = ' num2str(nf)], 'FontSize', 16)
```

In the title command, note how a number can be converted to a string via **`num2str`**.


Number of trials = 1000
heads
tails
edges
Cumulative sum of outcomes
Number of flips
0
100
200
300
400
500
600
700
800
900
1000

Introduction to MATLAB

disp, display, and fprintf

`display()` displays the array or string

```
>> a = 1
a =
     1
```

```
>> display(a)
a =
     1
```

```
>> disp(a)
     1
```

```
>> disp('hello world')
hello world
```

Text can be written to the command window using **`fprintf`**`()`.

```
fprintf('format', x, y, ...)}
```

or

```
fprintf('text as string \%5.2f more text', var-name)
```

```
>> fprintf('%10.8f\n', pi)
3.14159265
>> fprintf('pi to eight places %10.8f\n', pi)
pi to eight places 3.14159265
```

Format codes are the same as in C.

- `%f` - decimal format
- `%d` - integer format
- `%e` - scientific format with lowercase e
- `%E` - scientific format with uppercase E
- `%g` - the more compact of %e or %f

A few special codes

- `\n` - start a new line   Important
- `\t` - tab
- `\\` - print the character \

Introduction to MATLAB

disp, display, and fprintf

Example:

```
>> % An example from Mathworks:
>> name = 'Alice';  age = 12;
>> fprintf('My name is %s. I am %d years old!\n', name, age)
My name is Alice. I am 12 years old!
```

You can also use **`fprintf`**`()` to print to a file:

```
>> fid = fopen('foo.txt', 'w+'); % <-- open with write permission
>> fprintf(fid, 'My name is %s. I am %d years old!\n', name, age);
>> fclose(fid); % <-- close the open file
>> type foo.txt % <-- show the contents of foo.txt
My name is Alice. I am 12 years old!
```

Note the **`type`** instruction. It can be used to look at the contents of any script or function file, even some of MATLAB's own. Try `>>type linspace`

MATLAB allows you to **`save`** and **`load`** your variables: .

- `save filename var1 var2 ... varn`
  Saves the listed variables into a file named `filename.mat`.
  If no variable is listed, all variables in the workspace are saved.
- `load filename var1 var2 ... varn`
  Loads the listed variables from a file named `filename.mat`.
  If no variable is listed, all variables in the file are loaded.

Note - these files are not text files! You cannot view a `.mat` file with the editor, and do not try to print them!

You can view the variables in the current workspace using **`who`** or **`whos`**.

Introduction to MATLAB

Exporting figures

Suppose we have a nice plot we'd like to save and/or print:

```
>> x = linspace(0, 10, 1000);
>> y = sin(x.^2) + sin(x).^2;
>> plot(x, y, 'm', 'LineWidth', 3); axis equal, grid on
```

It can be printed directly by clicking on **`print`** in the figure window.

If you would like to save it as a MATLAB figure, click on **`save`** in the figure window and choose the `.fig` option. You will then be able to open this figure at a future time without having to execute the script that generated it.

If you would like to save it in a different format, perhaps for incorporating it into an essay or presentation, click on **`save`** but choose one of the options `.png`, `.pdf`, `.jpg` or .**`eps`**.

`print` also works from the command line. To see how, try `>>help print`

To show two or more plots together, use **subplot**

```
>> subplot(1,2,1) % <-- 1x2 table of axes, select 1st
>> pie(rand(9,1)) % <-- a pie chart
>> title('This is a pie chart');
>> subplot(1,2,2) % <-- 1x2 table of axes, select 2nd
>> bar(rand(3,4), 1.5, 'grouped') % <-- a bar chart
>> title('This is a bar chart');
```