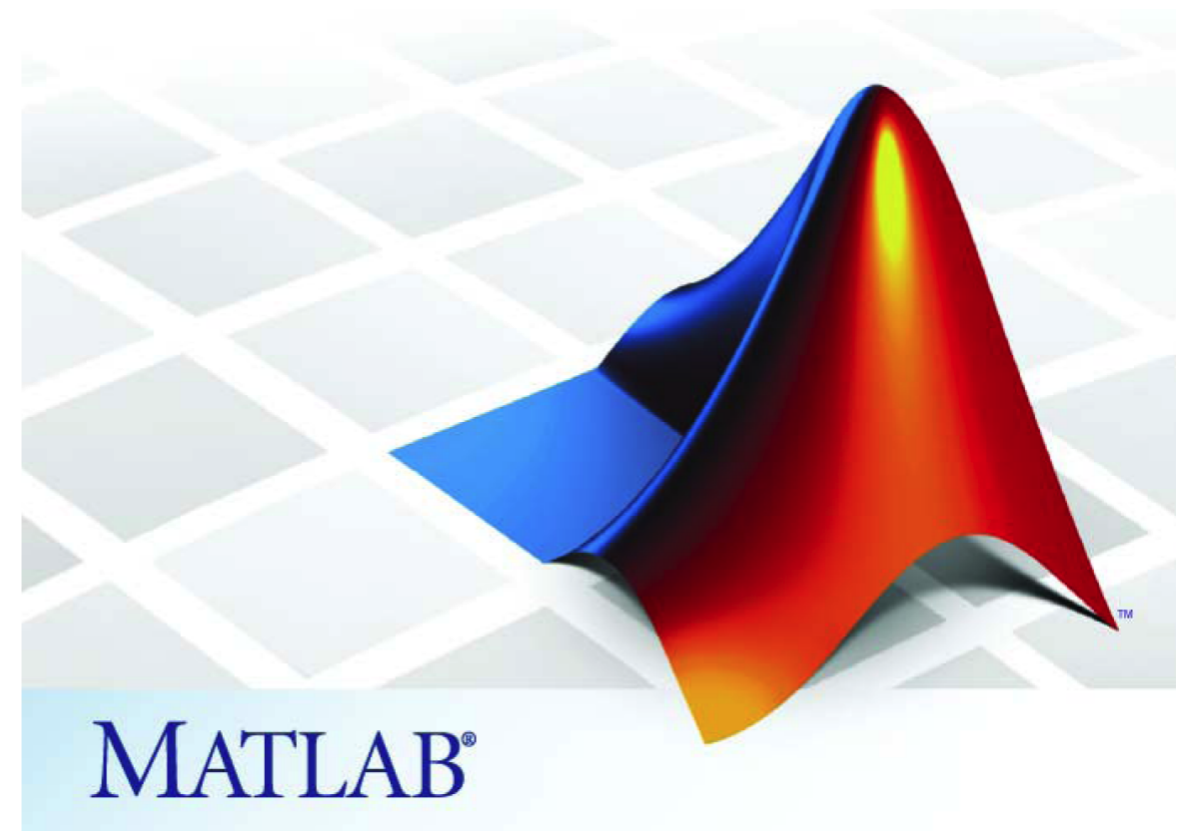


Introduction to Scientific Programming in Matlab

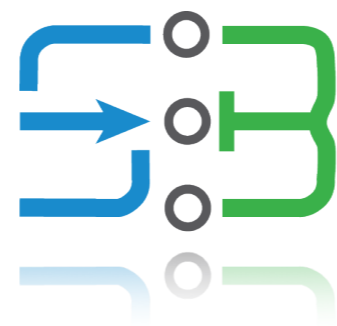
Prof Guy-Bart Stan

g.stan@imperial.ac.uk

www.bg.ic.ac.uk/research/g.stan



Imperial College
London

 Imperial College
Centre for Synthetic Biology
CENTRE FOR SYNTHETIC BIOLOGY

Content

Part A

1. Overview of Matlab
2. Getting started
3. Documentation and help
4. Variables
5. Matrix operations
6. Built-in functions
7. Controlling work flow

Part B

8. Basic input/output
9. Scripts and functions
10. Reading and writing data
11. Fitting a model to data
12. Solving differential equations
13. Plotting in 2d and 3d

www. http://www.stanford.edu/~wfsharpe/mia/mat/mia_mat3.htm

http://www.mathworks.com/academia/student_center/tutorials/launchpad.html

The labs are interactive, computer-based tutorials that offer us the opportunity to go over your exercises, as well as look into some related mathematics. Another good reference is the primer by Kermit Sigmon ([pdf](#)) as well as the [official Matlab documentation](#).

1. Overview of Matlab

Intuitive, easy-to-learn, high performance language integrating: computation, visualization, and programming.

- Math and computation
- Algorithm development
- Data acquisition
- Modeling, simulations, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, incl. graphical user interfaces

MATLAB stands for *matrix laboratory*. Its basic variables are arrays, i.e. vectors and matrices. Matlab also has many build-in functions (LAPACK and BLAST libs), as well as specialised add-on tool boxes. Features allow fast implementation of programs to solve computational problems.

The Matlab system consists of 5 main parts:

1. Desktop tools and development environment

Mainly graphical user interfaces, editor, debugger, and workspace

2. Mathematical function library

Basic math functions such as sums, cosine, complex numbers

Advanced math functions such as matrix inversion, matrix eigenvalues, differential equations

3. The language

High-level language based on arrays, functions, input/output, and flow statements (*for*, *if*, *while*)

4. Graphics

Data plotting in 2d and 3d, as well as image analysis and animation tools

5. External interfaces

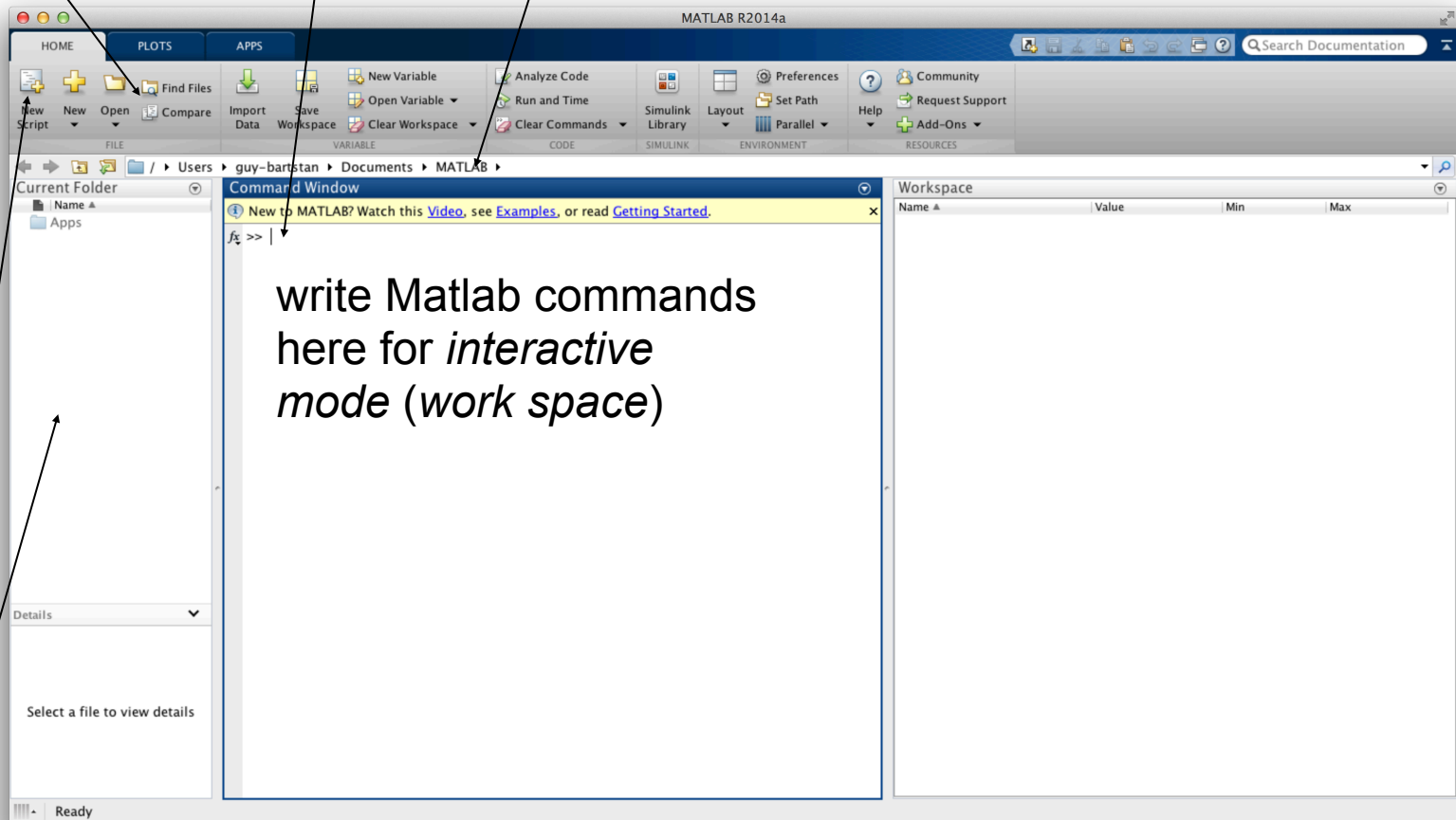
Interaction between C and Fortran programs with Matlab, either for linking convenient routines from Matlab in C/Fortran, or for Matlab to call fast C/Fortran programs

2. Getting started

Menus change, depending on the tool you are using

Enter MATLAB statements at the prompt

View or change the current directory



Go here to open new or existing Matlab files (**M-file**) and editor

Opening new and existing M-files (scripts or M-file functions):

new

existing

The image displays the MATLAB R2014a interface. The top window shows the 'HOME' tab with various toolbars. The 'FILE' toolbar includes 'New Script' (labeled 'new') and 'Open' (labeled 'existing'). The 'Current Folder' pane shows a list of files, including 'data_fitting.m' and 'fitcurvedemo.m'. The 'Command Window' shows the prompt 'fx >>'. The 'Workspace' pane is empty. The bottom window is the 'Editor' for 'data_fitting.m', showing the following code:

```
1 - clear all; close all;
2 -
3 - xdata = (0:1:10)';
4 - ydata = 40 * exp(-.5 * xdata) + randn(size(xdata));
5 -
6 - [estimates,model] = fitcurvedemo(xdata,ydata)
7 -
8 - plot(xdata,ydata,'*')
9 - hold on
10 - [sse,FittedCurve] = model(estimates);
11 - plot(xdata,FittedCurve,'r');
12 - xlabel('xdata')
13 - ylabel('f(estimates,xdata)')
14 - title(['Fitting to function ',func2str(model)])
15 - legend('data',['fit using ',func2str(model)])
16 - hold off
```

Ln 1 Col 1

Different ways to use Matlab:

(1) **Interactive mode:** just type commands and define variables, empty work space with command *clear*

(2) **Simple scripts:**

M-file (*name.m*) with list of commands

Operate on existing data in work space, or create new data to work on

Variables remain in workspace (until emptied)

Re-useable

(3) **Versatile M-file functions:**

M-file

May return values

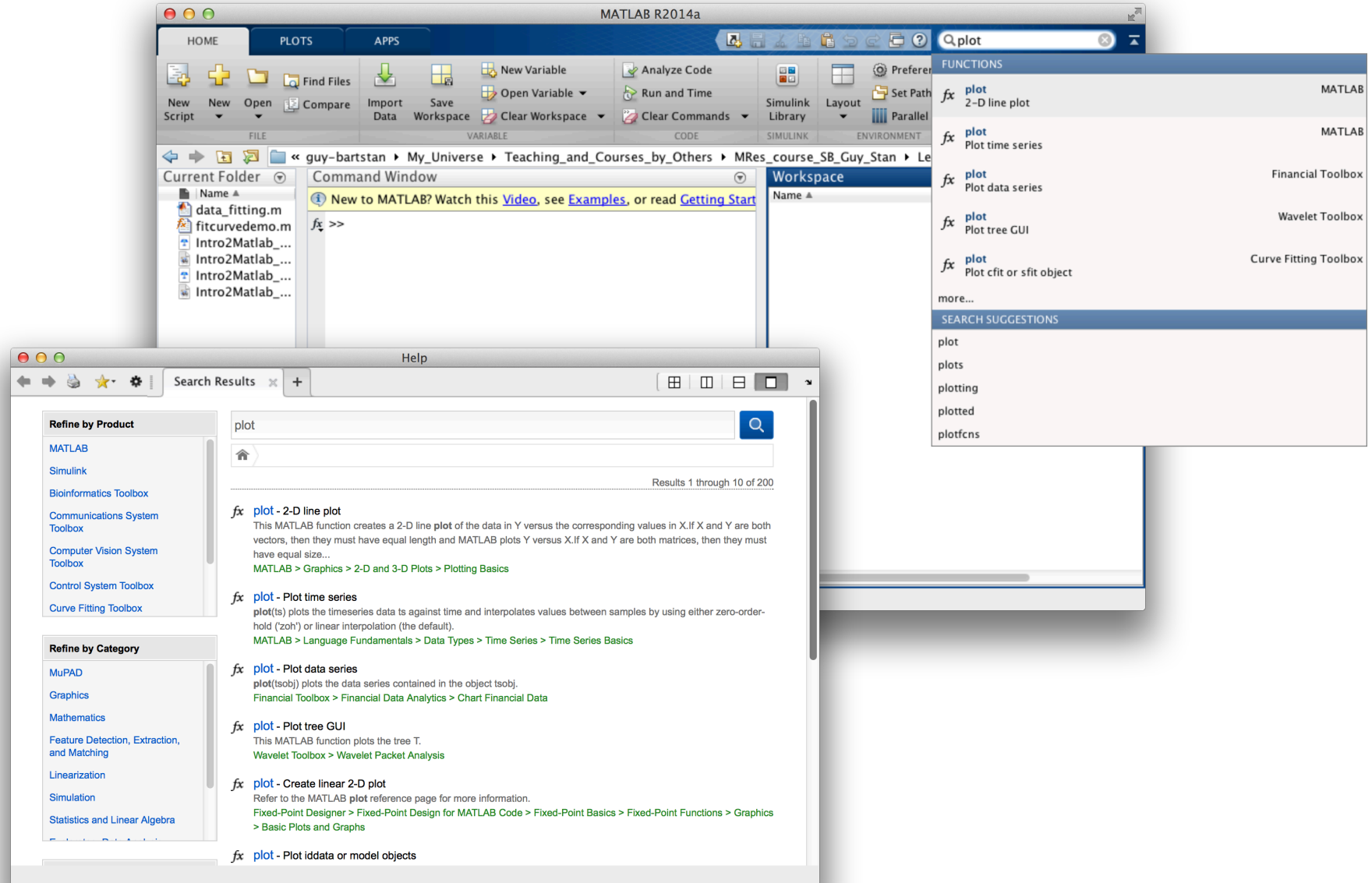
Re-usable

Easy to call from other functions

(make sure file is in Matlab search path, set by > File > Set Path)

3. Documentation and help

Matlab provides large amounts of documentation and tutorials:



4. Variables are represented as matrices

Matrix variables don't need to be declared. They are just assigned to values and know about their dimension.

Matrix assignment:

$C = A + B$ assigns matrix C as the sum of matrices A and B

If A and B are matrices of same dimension, e.g. $[3 \times 4]$ with 3 rows and 4 columns, C is $[3 \times 4]$ matrix with element-wise addition.

Example with $[2 \times 2]$ matrices:

$$A = \begin{pmatrix} 2 & 4 \\ 3 & 7 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 1 \\ 6 & 2 \end{pmatrix} \quad \longrightarrow \quad C = \begin{pmatrix} 7 & 5 \\ 9 & 9 \end{pmatrix}$$

If C existed before e.g. was a scalar $C = [1]$ (i.e. a $[1 \times 1]$ matrix), then C is overwritten by this new assignment.

If dimensions of A and B don't match, you will get an error:

??? Error using ==> +

Matrix dimensions must agree

Showing values:

To see content of a variable, just type its name:

```
C = A + B           provides  C =  
                        7      5  
                        9      9
```

To assign a variable without showing its content, use semicolon:

```
C = A + B;          (nothing)
```



Initializing matrices:

Provide initial values, e.g.

```
a=3;                (scalar)                typing d gives  
b=[ 1 2 3 ];        ([1x3] row vector)  
c=[ 4 ; 5 ; 6 ];    ([3x1] column vector)  
d=[ 1 2 3 ; 4 5 6 ]; ([2x3] matrix)
```

d =	1	2	3
	4	5	6

Values separated by spaces are put in the same row, e.g., $b=[1 \ 2 \ 3]$

Semicolon (or carriage return) separates rows, e.g., $c=[4 ; 5 ; 6]$



Making matrices from matrices:

```
a=[ 1 2 3];  
b=[ 4 5 6];    gives    c =  
c=[ a b];      1 2 3 4 5 6
```

while

```
a=[ 1 2 3];  
b=[ 4 5 6];    gives    c =  
c=[ a ; b];    1 2 3  
                4 5 6
```

Using portions of matrices:

```
d=[ 1 2 3 ; 4 5 6];    typing    d(1,2) returns 2  
         d(2,1) returns 4
```

and

```
d =  
    1    2    3  
    4    5    6  
d(1, :) returns    1    2    3  
d(:, 2) returns    2  
                   5
```

Using more portions of matrices:

$d = [1\ 2\ 3 ; 4\ 5\ 6]$; typing $d(2,[2\ 3])$ returns 5 6
 $d(2,[3\ 2])$ returns 6 5

Variables may also be used as indices of matrices

if you type
 $z = [2\ 3]$
then you will see that
 $d(2, z)$ returns 5 6

Use **colon** to produce string of consecutive integers

$x = 3 : 5$ produces vector $x = 3\ 4\ 5$

and

$d(1, 1:2)$ returns 1 2

Text strings:

A variable in Matlab is either *numeric* or a *string*.

However, the elements of a string matrix are represented by ASCII numbers, e.g. *space* is number 32, and captial *A* is 65 etc.

Strings are enclosed in single quotation marks (apostrophes)

`s = 'This is a string';` (a row vector of numbers)

Can create any matrix as long as rows have same length

`x = ['ab' ; 'cd']` produces

```
x =  
  ab  
  cd
```

`x = ['ab' 'cd']` produces

```
x =  
  ab  cd
```

5. Matrix operations

Matrix transposition is obtained by adding a prime (apostrophe)

if

$$x = \begin{matrix} 1 & 2 & 3 \end{matrix} \quad (\text{row vector})$$

then

$$x' = \begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \quad (\text{column vector})$$

Matrix addition is obtained by + sign, and
Matrix subtraction is obtained by - sign

If A is a $[3 \times 4]$ matrix and B is a $[4 \times 3]$ matrix, then

$C = A + B$ produces

while $C = A + B'$ works fine

??? Error using ==> +

Matrix dimensions must agree

Matrix multiplication is obtained by * symbol

$$C = A * B$$

Note that inner dimensions of the two operands must be the same, e.g. $A=[3 \times 4]$ and $B=[4 \times 2]$ works.

Element by element operations are given by

$$C = A .* B \quad (\text{multiplication})$$

$$C = A ./ B \quad (\text{division})$$

$$C = A .^ 2 \quad (\text{exponentiation})$$

but for first two, matrix dimensions have to agree!

Exceptions: For addition and subtraction, as well as element-by-element multiplication and division, matrix dimensions can be different if one of the operand is a scalar. In this case, the scalar is applied to each element in the matrix.

6. Built-in functions

value=sum(arg_1,arg_2)

Some provide one, others more than one answer.

Examples: *sum*, *max*, and *plot* functions

```
if x =  
    1  
    2  
    3  
  
then statement  
    y = sum(x) + 10  
  
produces  
    y =  
    16
```

```
if x =  
    1 4 3  
  
then statement  
    z = 10 + max(x)  
  
produces  
    z =  
    14
```

```
if x =  
    1 4 3  
  
then statement  
    [y n]= max(x)  
  
produces  
    y =  
    4  
    n =  
    2
```

position where found

multiple assignments possible as well

For more complicated cases, functions often have natural interpretation

```
if x =  
    1 2 3  
    4 5 6  
then  
    sum(x)  
  
    = 5 7 9 (column-wise addition)
```

Plotting function *plot*

plot(x, y)

produces a plot of y against x

but *plot(x)*

is also allowed and plots x against $1, 2, 3, \dots$

Example: sorting function *sort*

```
if x =  
    1 5  
    3 2  
    2 8  
then  
    y = sort( x )
```

produces

```
    y =  
        1 2  
        2 5  
        3 8
```

i.e., each column is
sorted separately

To obtain a record of the
rows from which the
sorted elements came:

```
[y r] = sort( x )
```

produces y as before and

```
    r =  
        1 2  
        3 1  
        2 3
```

Other built-in functions are: *mean*, *cov*, *min*, *max*, *ones*, *zeros*, *size*, *rand*, *randn*

M-file functions: provided in `\matlab\toolbox` or written by yourself
filename.m

What do built-in or M-file functions do?

To obtain description:

`help mean`

To see code:

`type mean`

Relational and logical operations:

Matlab knows *six relational operations*

< less than
<= less than or equal to
> greater than
>= greater than or equal to
== equal
~= not equal

Note:

$A=B$ assigns to A the values of B

$(A==B)$ tests whether A and B are equal

and the following *logical operators*

& and
| or
~ not

Whenever Matlab encounters a **relational operator**, it produces a 1 if the expression is true and a 0 if the expression is false:

$x = (1 < 3)$ produces $x=1$, while
 $x = (1 > 3)$ produces $x=0$

Relational operators can also be applied to matrices as long as they have the same dimension (as relational operators then work on an element-by-element basis):

```
if A =
    1 2
    3 4
and B =
    3 1
    2 2
then
    C = (A > B)

produces
    C =
    0 1
    1 1
```

```
if A =
    1 2
    3 4
then
    C = (A > 2)
produces
    C =
    0 0
    1 1
```

scalar

7. Controlling work flow

To change to a non-sequential order, use **for** and **while** loops, as well as **if** statements

for loops:

```
for j = 1 : n
    .....
end
```

while loops:

```
while (x > 0.5)
    .....
end
```

Note: avoid infinite loops by including termination condition

For clarity, introduce TRUE and FALSE variables

```
true = (1==1);
false = (1==0);
.....
done = false;
while not done
    .....
end
```

if statement:

```
if (x > 0.5)
    .....
end
or
if (x > 0.5)
    .....
else
    ....
end
```

Nesting:

```
for j = 1 : n
    for k = 1 : n
        if (x(j, k) > 0.5)
            x(j, k) = 1.5;
        end
    end
end
end
```

(indentations are for clarity only)

Nesting should be avoided for matrix operations, since very slow:

```
instead of
port_val = 0;
for j = 1 : n
    port_val = port_val + ( holdings(j) * prices(j) );
end
use port_val = holdings * prices ;
```

8. Basic input/output

Basic data input:

1. Type instructions in interactive mode or in script mode.

Examples: *radius* = [12.50 37.875 12.25]
 molecules = ['sugars' ; 'amino acids' ; 'proteins']
 data = [100 200
 300 400] (line breaks for increased clarity)

2. Read text file and put in matrix *test*: *load test.txt*

Basic data output:

1. Display data: *disp ('test');*
2. Dump stuff from display into file: *diary filename* to start and *diary off* to stop
3. Save data from a matrix, use *save newdata.txt test -ascii*
4. Save variables etc. from interactive Matlab session in *.mat* file, use
 save temp (saves complete session in *temp.mat* file)
 save temp radius molecules data (saves only certain variables in *temp.mat*)
 load temp (restores session later)

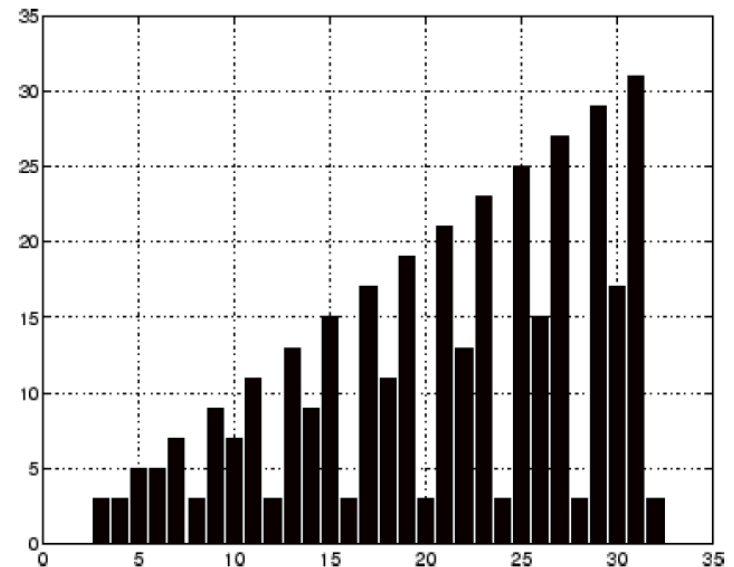
9. Scripts and functions

Example script:

An M-file called **magicrank.m** may contain following code

```
% Investigate the rank of magic squares
r = zeros(1,32);
for n = 3:32
    r(n) = rank(magic(n));
end
r
bar(r)
```

Typing *magicrank* executes script, and computes rank of first 30 magic squares and plots bar chart of results



Example function:

Functions have the advantage that they can be re-used in different programs. A function starts with a line declaring the function, its arguments and its outputs.

Examples: must be saved in port_val.m

```
function y = port_val(holdings, prices)
y = holdings * prices;
```

returns
one value

This function is called by

local variables

```
v = port_val( h, p );
```

variables can be named differently
in calling statement

```
function [total_val , avg_val] = port_val(holdings, prices)
total_val = holdings * prices;
avg_val = total_val/size( holdings , 2 );
```

returns
two values

This function is called by

```
[tval aval ] = port_val( h, p );
```

Functions:

Name of M-file and function should be the same. Variables only defined in function, not common workspace.

M-file *rank.m* is available in directory `toolbox/matlab/matfun`

To see file, write `type rank`, which produces

```
function r = rank(A,tol)
% RANK Matrix rank.
% RANK(A) provides an estimate of the number of linearly
% independent rows or columns of a matrix A.
% RANK(A,tol) is the number of singular values of A
% that are larger than tol.
% RANK(A) uses the default tol = max(size(A)) * norm(A) * eps.

s = svd(A);
if nargin==1
    tol = max(size(A)') * max(s) * eps;
end
r = sum(s > tol);
```

To get info, i.e., first lines of comments (starting with %), write `help rank`

Function can be called as

```
rank(A)
r = rank(A)
r = rank(A,1.e-6)
```

Primary and subfunctions:

Each M-file has a required primary function that **appears first in file**, which **can be invoked from outside the M-file**. Additionally, the **M-file can contain any number of subfunctions that follow it, which are only visible to the primary and other subfunctions**

Anonymous functions:

```
f = @(arglist)expression
```

Don't require an M-file. Are **defined in one line**.

```
sqr = @(x) x.^2;
```

Then, if you type:

```
a = sqr(5)
```

You get:

```
a =
```

```
25
```

Function handles:

Create a handle to any Matlab function and then use it to reference the function.
Often used to pass function as an argument list to other functions.

Create: `fhandle = @sin;`

Use: `fhandle(arg1, arg2, ...);`

Or create: `function x = plot_fhandle(fhandle, data)
plot(data, fhandle(data))`

And, use: `plot_fhandle(@sin, -pi:0.01:pi)`

Function of functions:

Functions, which operate on functions, e.g.,

Zero finding

Optimization

Quadrature (integration)

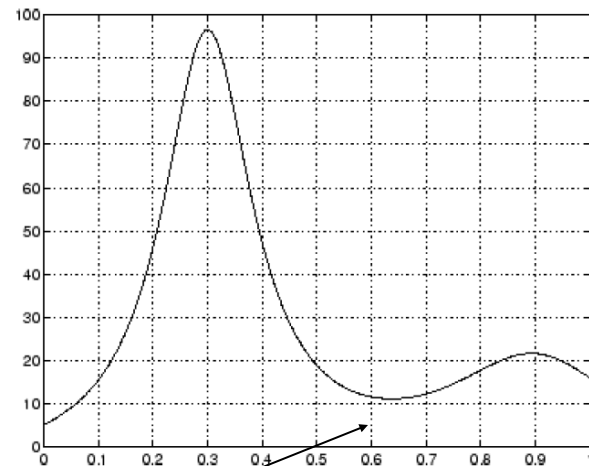
Ordinary differential equations

Example:

```
function y = humps(x)
y = 1./((x-.3).^2 + .01) + 1./((x-.9).^2 + .04) - 6;
```

```
evaluate    x = 0:.002:1;
            y = humps(x);
```

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



local minimum near 0.6

find minimum near 0.5

```
p = fminsearch(@humps, .5)
```

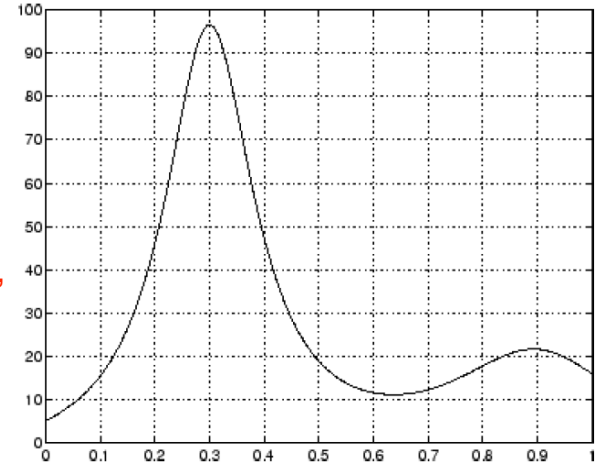
```
p =  
    0.6370
```

Using @humps
we pass **the function**
"humps" as an argument
of the function "fminsearch"

value at minimum

```
humps(p)
```

```
ans =  
    11.2528
```



integrate from 0 to 1

```
Q = quad1(@humps,0,1)
```

```
Q =  
    29.8583
```

find zero near 0.5

```
z = fzero(@humps, .5)
```

```
z =  
   -0.1316
```

10. Reading and writing data



File Formats

Readable **file** formats

Description

This table shows the **file formats** that the MATLAB® software is capable of reading.

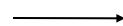
File Format	Extension	File Content	Read Command	Returns
Text	MAT	Saved MATLAB workspace	load	Variables in file
	CSV	Comma-separated numbers	csvread	Double array
	DAT	Formatted text	importdata	Double array
	DLM	Delimited text	dlmread	Double array
	TAB	Tab-separated text	dlmread	Double array
Scientific Data	CDF	Data in Common Data Format	cdfread	Cell array of CDF records
	FITS	Flexible Image Transport System data	fitsread	Primary or extension table data
	HDF4	Data in Hierarchical Data Format, version 4	hdfread	HDF 4 or HDF-EOS 2 data set
	HDF5	Data in Hierarchical Data Format, version 5	hdf5read	HDF5 or HDF-EOS 5 data set
Spreadsheet	XLS	Microsoft® Excel® worksheet	xlsread	Double or cell array
	WK1	Lotus 123 worksheet	wklread	Double or cell array
Image	BMP	BMP image	imread	True color or indexed image
	CUR	Cursor image	imread	Indexed image
	GIF	GIF image	imread	Indexed image
	HDF4	HDF4 image	imread	True color, grayscale, or indexed image(s)

ICO	Icon image	imread	Indexed image
JPEG	JPEG image	imread	True color or grayscale image
PBM	PBM image	imread	Grayscale image
PCX	PCX image	imread	Indexed image
PGM	PGM image	imread	Grayscale image
PNG	PNG image	imread	True color, grayscale, or indexed image
PPM	PPM image	imread	True color image
RAS	SUN raster image	imread	True color or indexed
TIFF	TIFF image	imread	True color, grayscale, or indexed image(s)
XWD	XWD image	imread	Indexed image

Reading an Excel file:

(1) Reading file 'testdata2.xls'
with numbers and text

1	6
2	7
3	8
4	9
5	text



```
A = xlsread('testdata2.xls')
A =
     1     6
     2     7
     3     8
     4     9
     5    NaN
```

(2) Reading rows 4 and 5

```
A = xlsread('testdata2.xls', 1, 'A4:B5')
```

```
A =
     4     9
     5    NaN
```

↑
spreadsheet 1 of file

(3) Reading numbers only

Time	Temp
12	98
13	99
14	97

```
ndata = xlsread('tempdata.xls', 'Temperatures')
```

```
ndata =
     12     98
     13     99
     14     97
```

↑
spreadsheet labelled
'Temperatures' of file

(4) Reading numbers and header text

```
[ndata, headertext] = xlsread('tempdata.xls', 'Temperatures')
```

```
ndata =  
    12    98  
    13    99  
    14    97
```

```
headertext =  
    'Time'    'Temp'
```

Convert row to column vectors and concatenate (cat) arrays along specified dimension (here dim "1", i.e., row dimension)

Writing to a text file:

```
data=cat(1,time',measurements');  
fprintf(fid, '%10.6f %10.6f \n', data);  
fclose(fid);
```

line break (return)

10 places, 6 precision floating-point variable

→ Produces file in table format with two numbers per line

11. Fitting a model to data (code avail. on BB)

In this example, we fit an exponential function of the form $Ae^{-\lambda t}$ to some data. The M-file is given by:

data needs to be provided

function handle


```
function [estimates, model] = fitcurvedemo(xdata, ydata)
% Call fminsearch with a random starting point.
start_point = rand(1, 2);
model = @expfun;
estimates = fminsearch(model, start_point);
% expfun accepts curve parameters as inputs, and outputs sse,
% the sum of squares error for A*exp(-lambda*xdata)-ydata,
% and the FittedCurve. FMINSEARCH only needs sse, but we want
% to plot the FittedCurve at the end.
function [sse, FittedCurve] = expfun(params)
    2 fitting parameters { A = params(1);
                          lambda = params(2);
                          FittedCurve = A .* exp(-lambda * xdata);
                          ErrorVector = FittedCurve - ydata;
                          sse = sum(ErrorVector .^ 2);
    end
end
```

fitting fcn is hard-wired

To use, create some random data first:

```
xdata = (0:.1:10)';  
ydata = 40 * exp(-.5 * xdata) + randn(size(xdata));
```

additive noise in the data:
normal distributed random
numbers between 0 and 1



and then call fitting function:

```
[estimates, model] = fitcurvedemo(xdata,ydata)
```

This returns the optimal parameters:

```
estimates =  
  
40.1334    0.5025
```

and a function handle *model* to the best model.

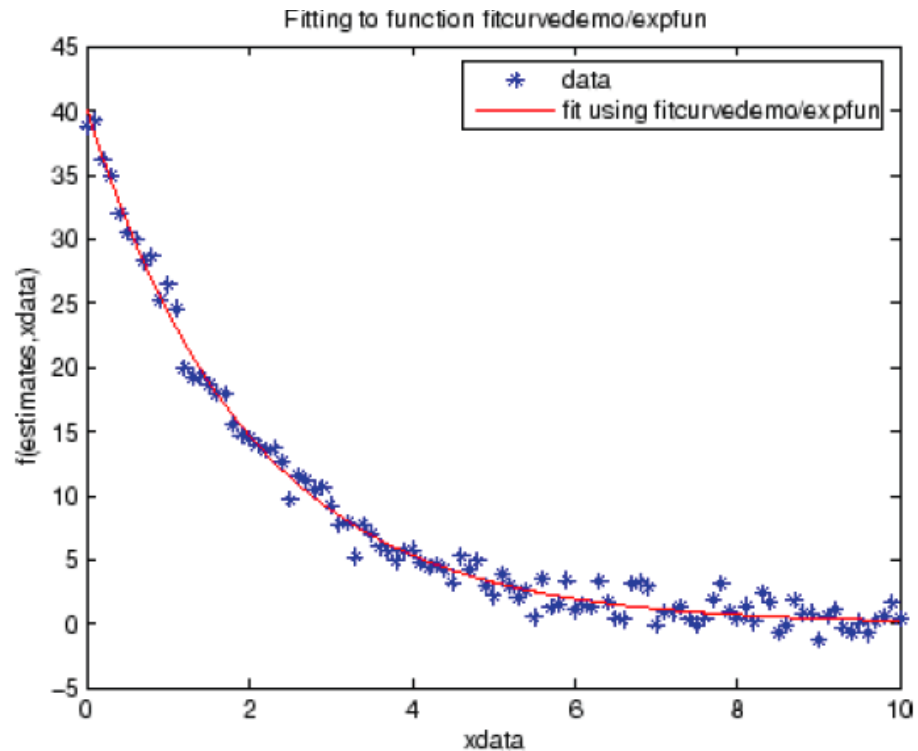
To plot data and fitted model, enter the following commands:

```
plot(xdata, ydata, '*')
hold on
[sse, FittedCurve] = model(estimates);
plot(xdata, FittedCurve, 'r')
```

labels
axis and
makes a
legend

```
xlabel('xdata')
ylabel('f(estimates,xdata)')
title(['Fitting to function ', func2str(model)]);
legend('data', ['fit using ', func2str(model)])
hold off
```

This produces plot:



12. Solving ordinary differential equations

Matlab ODE solvers only accept first-order differential equation $\dot{y} = f(t, y)$

Solvers provided are:

Solver	Solves These Kinds of Problems	Method
ode45	Nonstiff differential equations	Runge-Kutta
ode23	Nonstiff differential equations	Runge-Kutta
ode113	Nonstiff differential equations	Adams
ode15s	Stiff differential equations and DAEs	NDFs (BDFs)
ode23s	Stiff differential equations	Rosenbrock
ode23t	Moderately stiff differential equations and DAEs	Trapezoidal rule
ode23tb	Stiff differential equations	TR-BDF2
ode15i	Fully implicit differential equations	BDFs

To solve n -th order ODE $y^{(n)} = f(t, y, \dot{y}, \ddot{y}, \dots, y^{(n-1)})$

write it as a set of n coupled first-order ODEs:

For that: make substitutions $y_1 = y, y_2 = \dot{y}, y_3 = \ddot{y}, \dots, y_n = y^{(n-1)}$

and obtain

$$\dot{y}_1 = y_2$$

$$\dot{y}_2 = y_3$$

$$\vdots$$

$$\dot{y}_n = f(t, y_1, y_2, \dots, y_n)$$

Initial value problem: since there are many potential solutions for an ODE, you need to specify initial values:

$$\dot{y} = f(t, y)$$
$$y(t_0) = y_0$$

Example: Solve two coupled ODEs with solver **ode45**

```
function dydt = vdp1(t,y)
dydt = [y(2); (1-y(1)^2)*y(2)-y(1)];
```

In file
vdp1.m

```
[t,y] = ode45(@vdp1, [0 20], [2; 0]);
```

time interval initial values

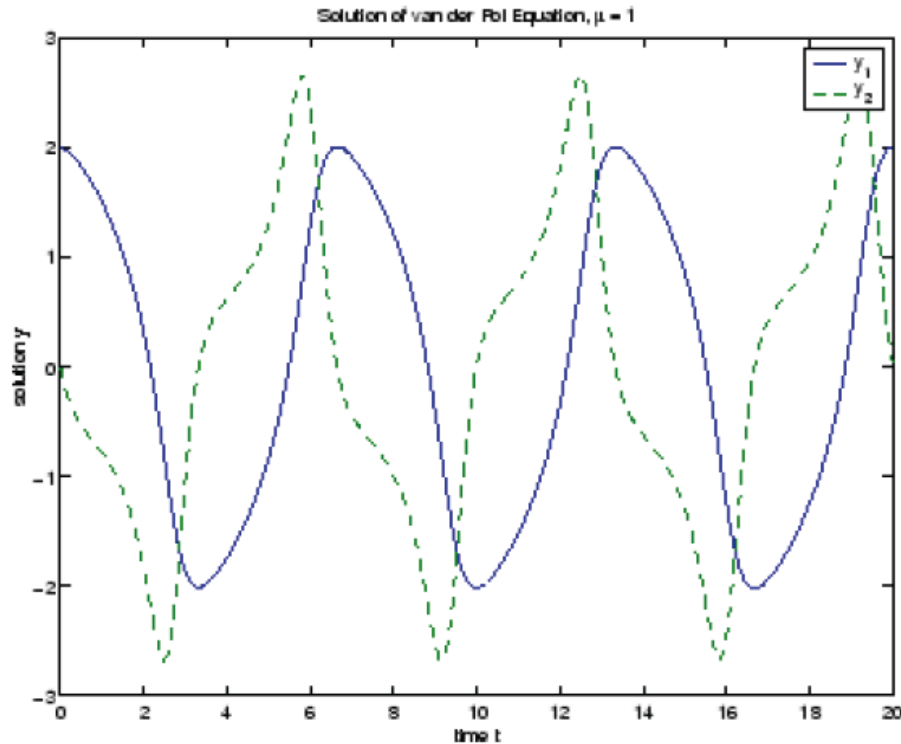
Script
myscript.m

Plot result:

```
plot(t,y(:,1),'-',t,y(:,2),'--')
title('Solution of van der Pol Equation, \mu = 1');
xlabel('time t');
ylabel('solution y');
legend('y_1','y_2')
```

LaTeX symbols

Script
myscript.m



13. Plotting in 2d & 3d

To plot x versus y (2d plot), use command `plot(x,y,'color_style_marker')`

a string, containing between 1 to four characters enclosed by '...', indicating color, line style, and marker type.

Type	Values	Meanings
Color	'c' 'm' 'y' 'r' 'g' 'b' 'w' 'k'	cyan magenta yellow red green blue white black
Line style	'-' '--' '.' '-.' no character	solid dashed dotted dash-dot no line
Marker type	'+' 'o' '*' 'x' 's' 'd' '^' 'v' '> '< 'p' 'h' no character or none	plus mark unfilled circle asterisk letter x filled square filled diamond filled upward triangle filled downward triangle filled right-pointing triangle filled left-pointing triangle filled pentagram filled hexagram no marker

Examples:

- (1) `plot(x,y,'ks')` for black squares at each point and no line
- (2) `plot(x,y,'r:+')` for red-dotted line and plus-sign markers at each data point
- (3) `plot(x,y,'r:+', 'LineWidth',2, 'MarkerSize',10)`

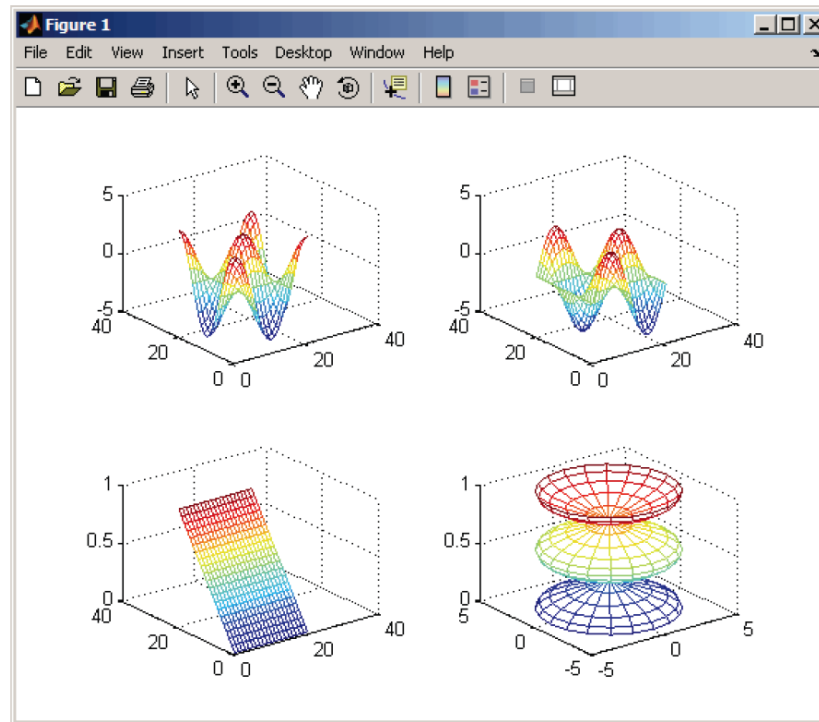
same as (2), but thicker line and larger markers

Multiple panels: To arrange plots in a $m \times n$ matrix use

`subplot(m,n,p)`

Example: four 3d plots

```
t = 0:pi/10:2*pi;
[X,Y,Z] = cylinder(4*cos(t));
subplot(2,2,1); mesh(X)
subplot(2,2,2); mesh(Y)
subplot(2,2,3); mesh(Z)
subplot(2,2,4); mesh(X,Y,Z)
```

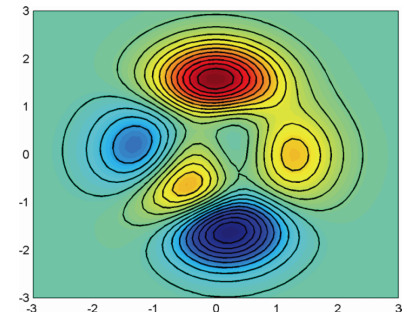


plots X versus $1:n$ and $1:m$ with $[m,n]=size(X)$

draws wireframe mesh with color determined by height Z as a function of X and Y

Additional 2d plots are: *loglog*, *semilogx*, and *semilogy*

Other 3d plots are: *plot3*, *contour*, and *surf*



- To download the files log on Blackboard:

<https://bb.imperial.ac.uk>

- The files are also on:

[http://www.bg.ic.ac.uk/research/g.stan/#Lecture Notes](http://www.bg.ic.ac.uk/research/g.stan/#Lecture_Notes)