

# An introduction to Matlab

Ken Nyholm

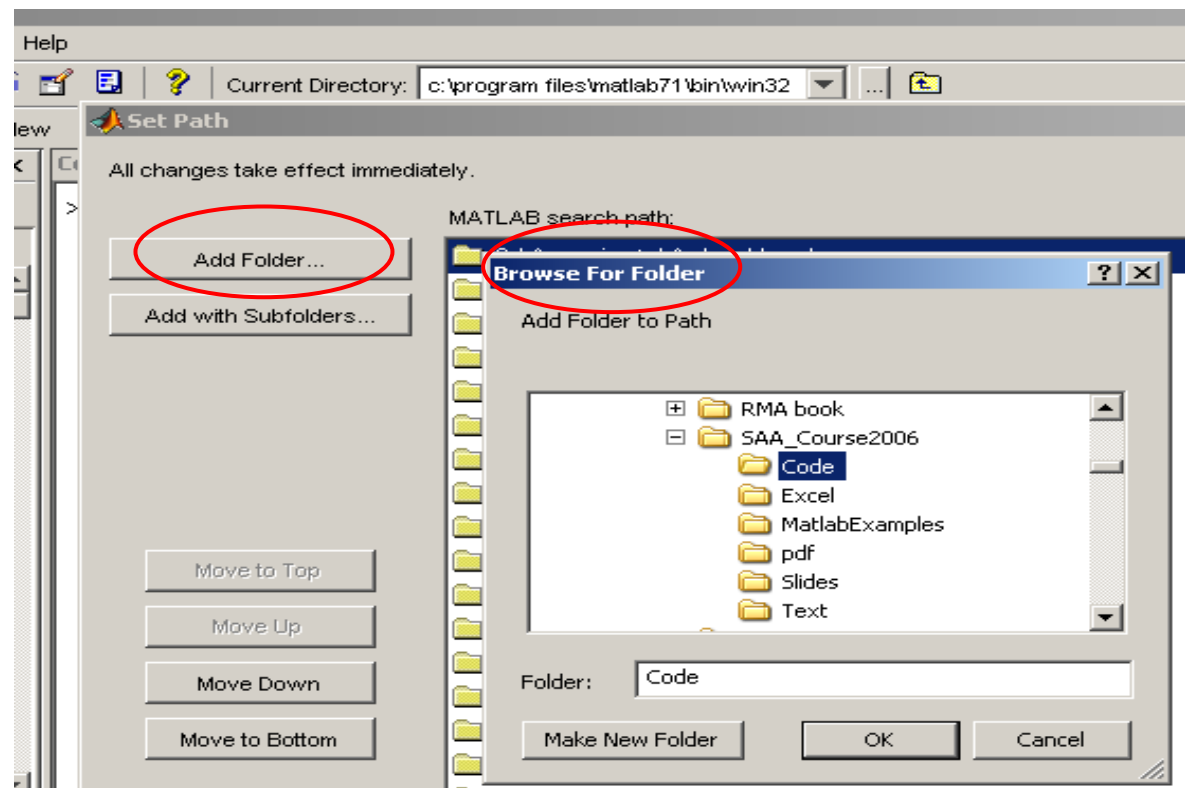
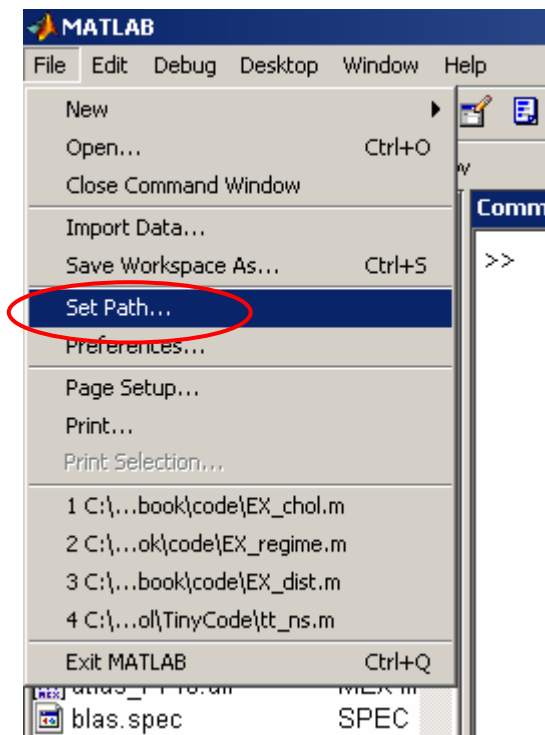
# Outline

---

- Getting Matlab up and running
  - Setting the path
  - Excel link
  - Command line
  - The Matlab work space
  - Writing and executing a script
- Functionalities
  - Help
  - Graphing
  - Variables
  - Branching and Looping
  - Functions
  - Fmincon

# Getting Matlab up and running

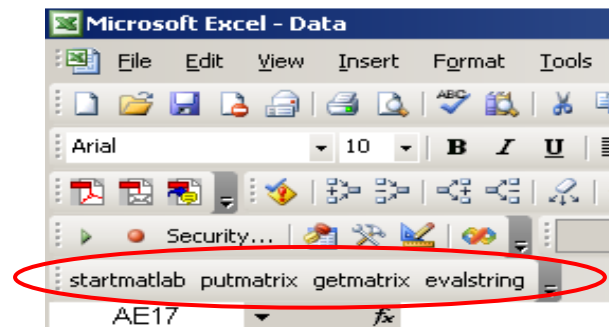
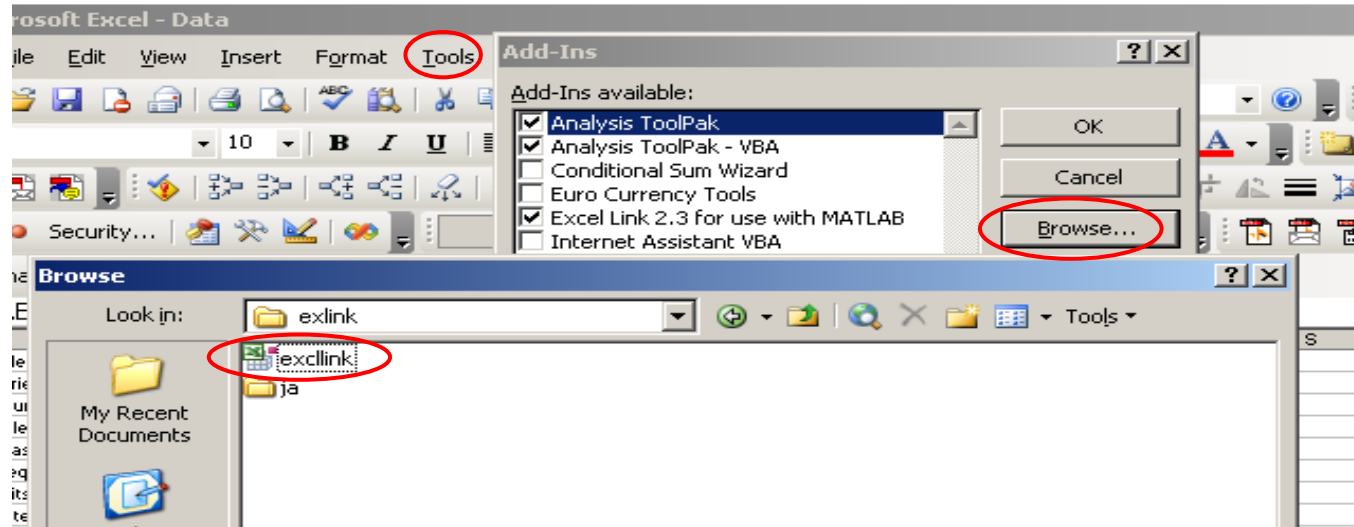
- The *path tool* tells matlab where your programs and scripts are stored



# Getting Matlab up and running

- The Excel link:
  - Makes it easier to transfer data between Matlab and Excel
  - Allows for execution of Matlab scripts from Excel, either directly or from VBA
- To install:
  - In Excel go to: Tools, add-ins, browse
  - And find the folder: Matlab\toolbox\excel link
  - see next slide

# Getting Matlab up and running



# Getting Matlab up and running

- Basically two types of input formats can be used with Matlab:
  - Command line entry for smaller tasks or when executing a program
  - Scripts/programs and functions for larger and re-occurring tasks

# Getting Matlab up and running

- Examples of command line entry:

The left screenshot shows the MATLAB Command Window with the command `a = randn(5,5)` entered and executed. The output is a 5x5 matrix of random numbers:

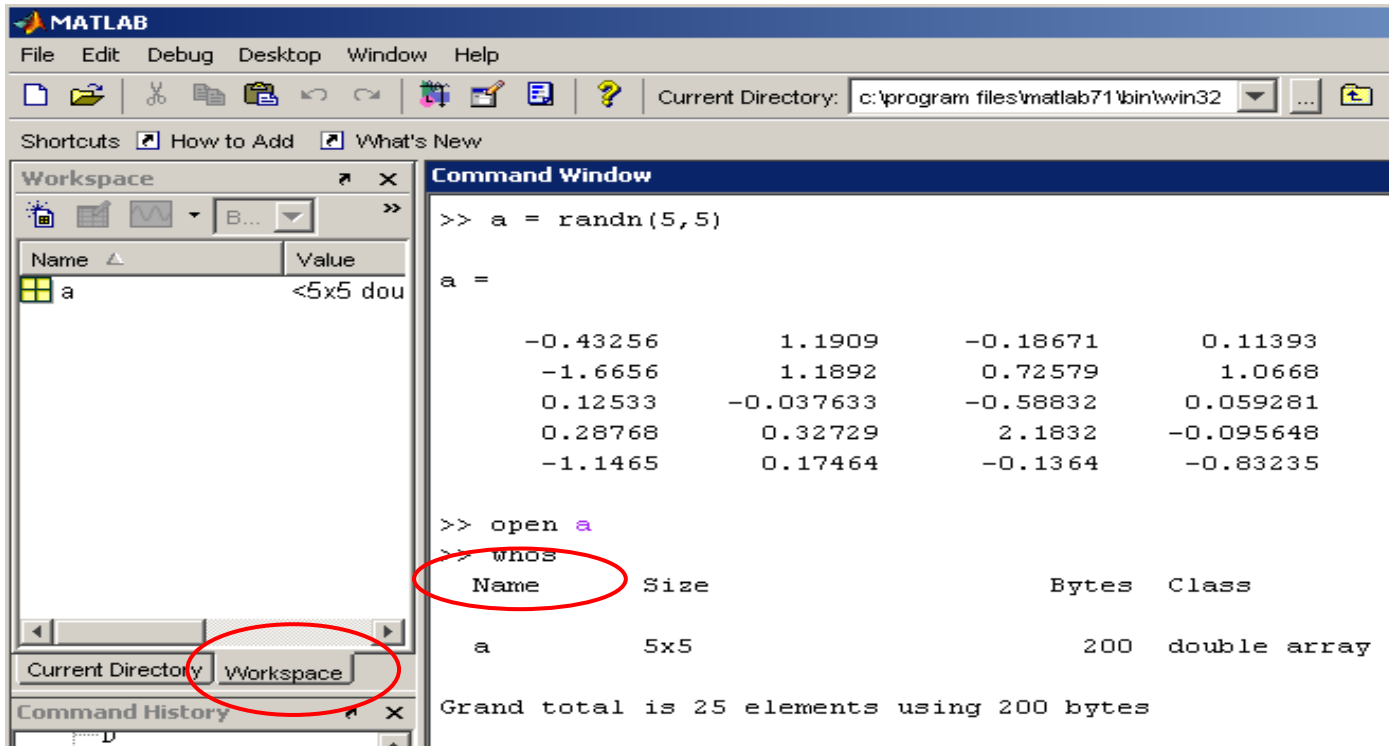
```
a =  
-0.43256    1.190  
-1.6656    1.189  
0.12533   -0.03763  
0.28768    0.3272  
-1.1465    0.1746  
>>
```

The right screenshot shows the MATLAB Array Editor displaying the matrix `a` and the Command Window with the command `open a` entered and executed. The output is the same 5x5 matrix of random numbers:

```
a =  
-0.43256    1.1909    -0.18671    0.11393    0.29441  
-1.6656    1.1892    0.72579    1.0668    -1.3362  
0.12533   -0.037633   -0.58832    0.059281    0.71432  
0.28768    0.32729    2.1832    -0.095648    1.6236  
-1.1465    0.17464    -0.1364    -0.83235    -0.69178  
>> open a  
>>
```

# Getting Matlab up and running

- The Matlab workspace



The screenshot shows the MATLAB environment with the following components:

- Workspace:** A table with two columns: Name and Value. It contains one entry: 'a' with a value of '<5x5 dou'.
- Command Window:** Contains the following text:

```
>> a = randn(5,5)
a =
    -0.43256    1.1909   -0.18671    0.11393
   -1.6656    1.1892    0.72579    1.0668
    0.12533   -0.037633  -0.58832    0.059281
    0.28768    0.32729    2.1832   -0.095648
   -1.1465    0.17464   -0.1364   -0.83235

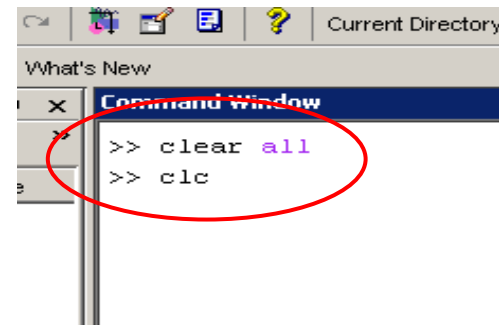
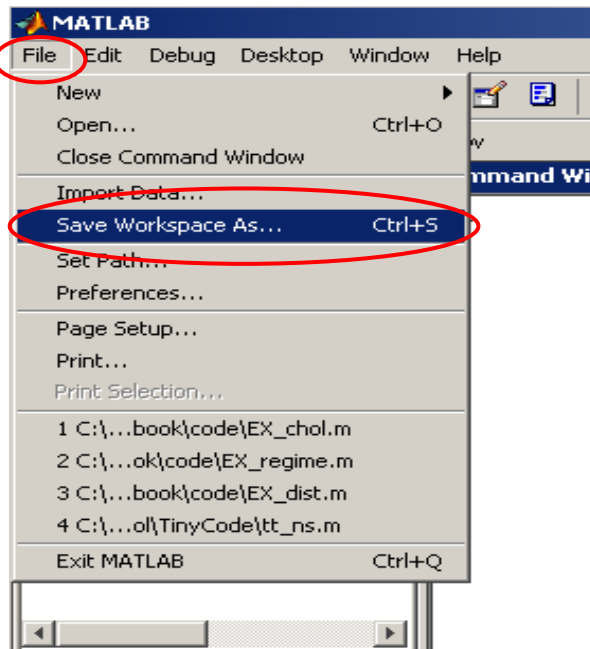
>> open a
>> whos
Name      Size      Bytes  Class
a         5x5       200    double array

Grand total is 25 elements using 200 bytes
```

Red circles highlight the 'whos' command in the Command Window and the 'Workspace' tab in the bottom-left pane.

# Getting Matlab up and running

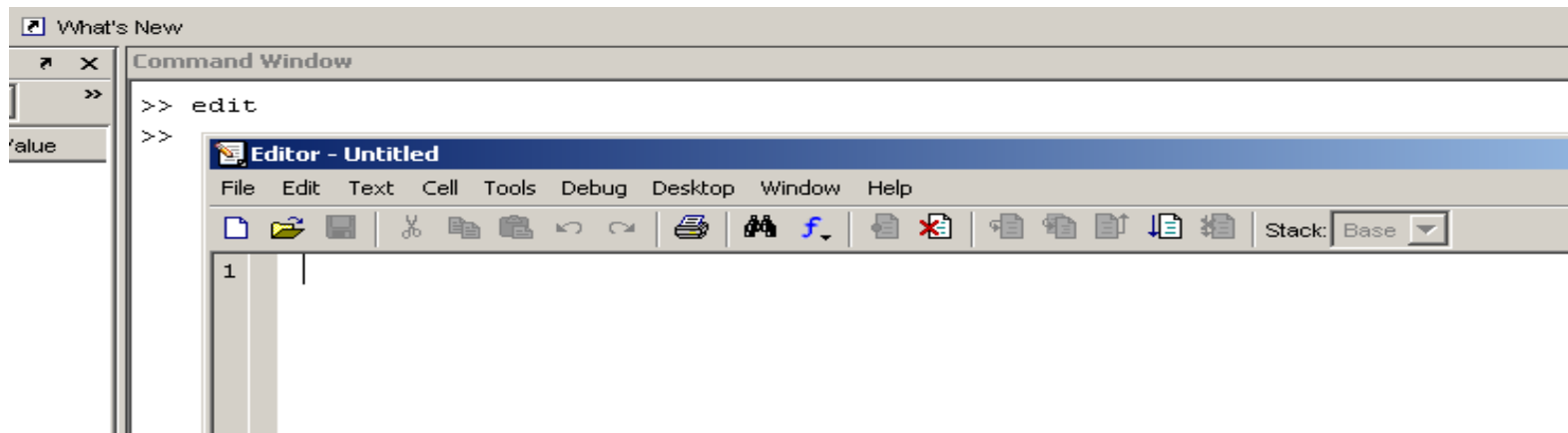
- The workspace can be saved so that all variables are preserved for later use and/or shared with colleagues



- The workspace can be cleared by using the command *clear all* and the screen is cleared by the command *clc*

# Getting Matlab up and running

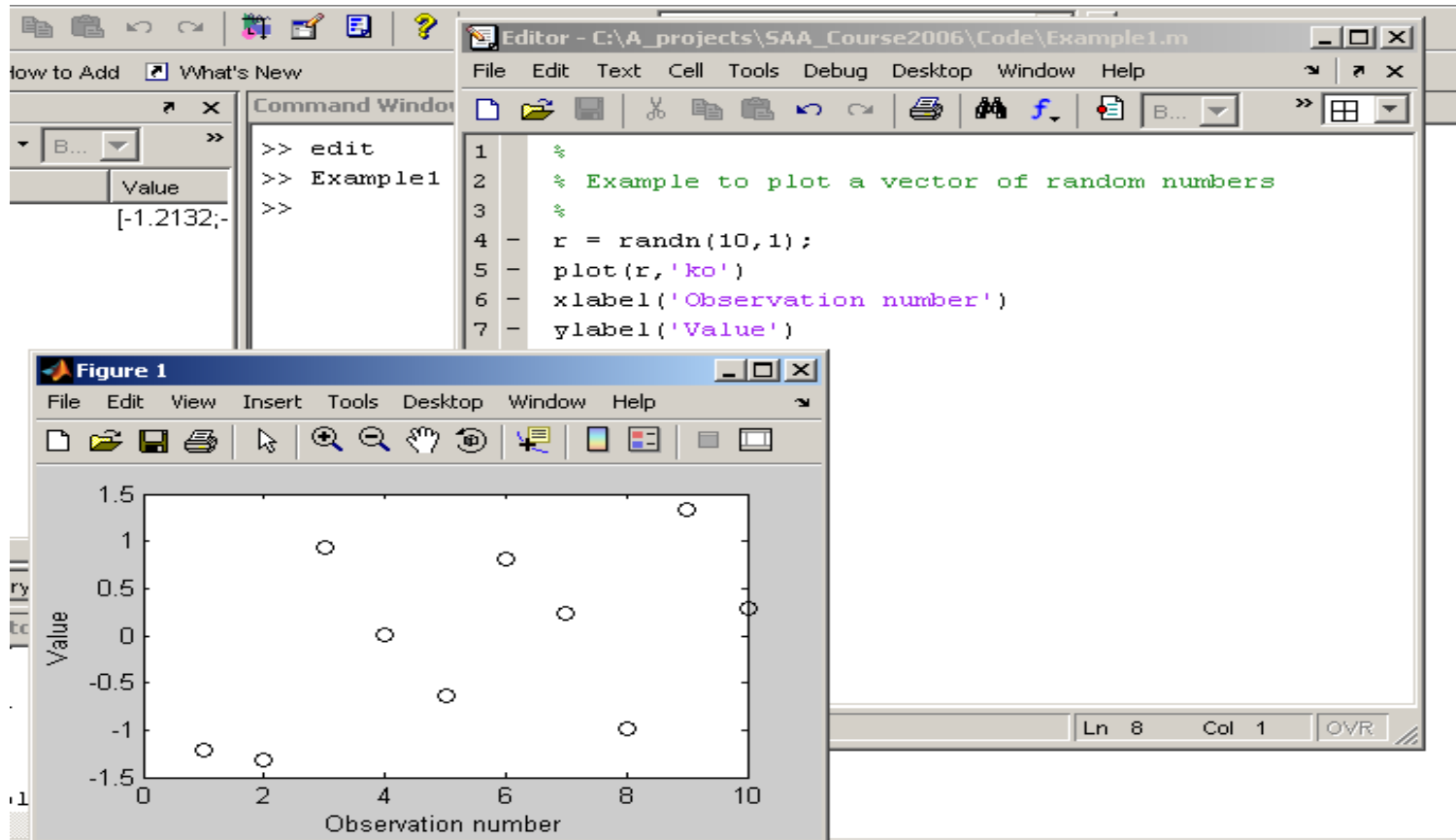
- The Matlab editor is used to write scripts and functions
- It can be started from the command line by writing *edit*
- The following editor window then appears:



# Getting Matlab up and running

- The script should be given a name and saved in the appropriate folder
- An example is given in the lecture notes page 5 (see next slide for the output produced)
- Notice the significance of the symbols:
  - Semicolon ;
  - Percentage %

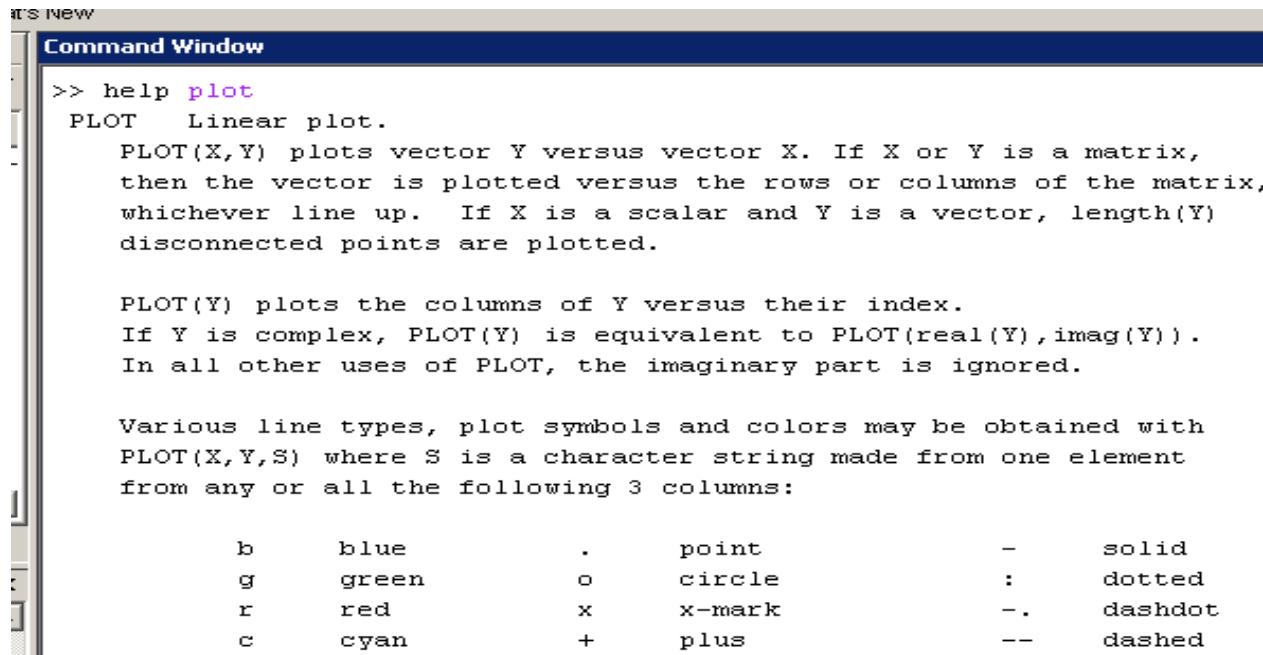
# Getting Matlab up and running



# Functionalities

---

- There are different ways to obtain help in Matlab
- a) Help on a particular function:
  - From command line: `>> help function name`
  - For example:



```
IT'S NEW
Command Window
>> help plot
PLOT Linear plot.
PLOT(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
then the vector is plotted versus the rows or columns of the matrix,
whichever line up. If X is a scalar and Y is a vector, length(Y)
disconnected points are plotted.

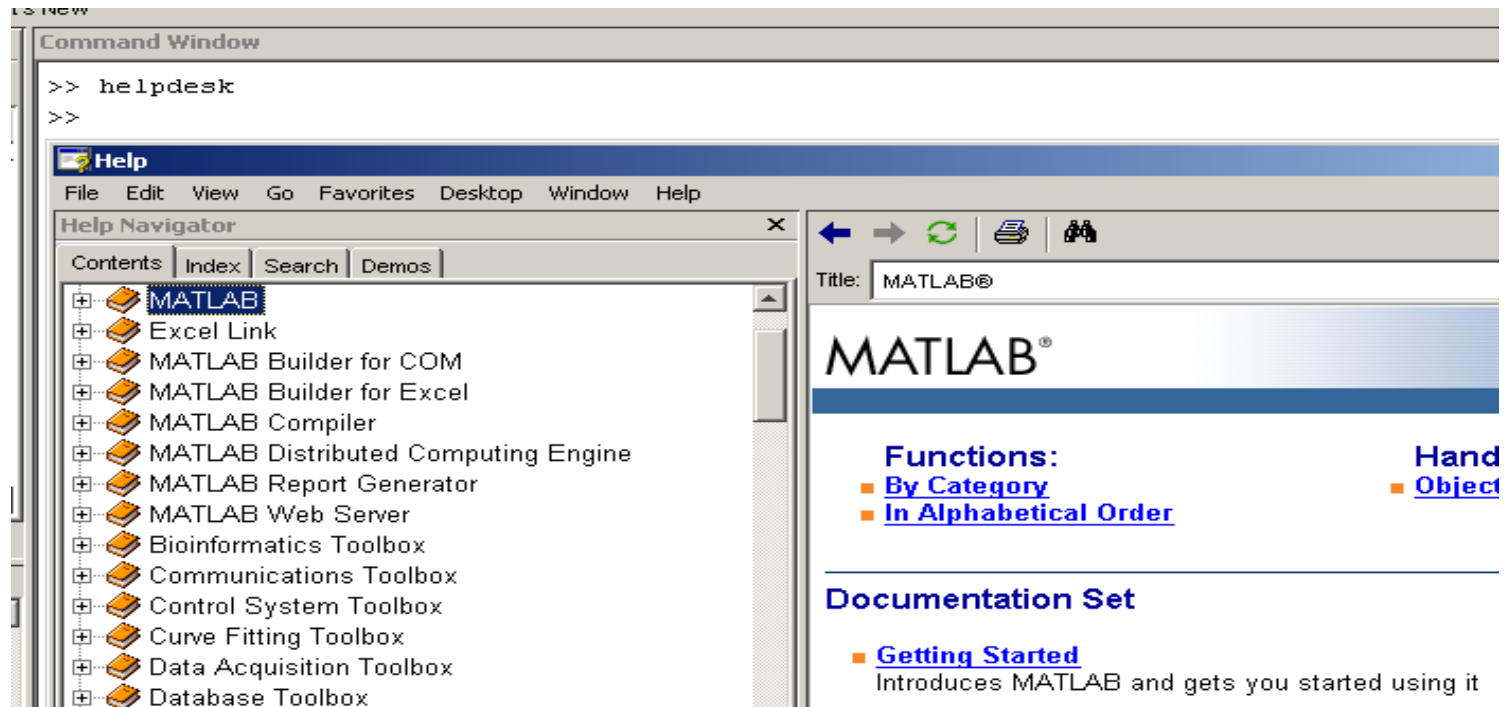
PLOT(Y) plots the columns of Y versus their index.
If Y is complex, PLOT(Y) is equivalent to PLOT(real(Y),imag(Y)).
In all other uses of PLOT, the imaginary part is ignored.

Various line types, plot symbols and colors may be obtained with
PLOT(X,Y,S) where S is a character string made from one element
from any or all the following 3 columns:

      b      blue      .      point      -      solid
      g      green     o      circle     :      dotted
      r      red       x      x-mark    -.     dashdot
      c      cyan      +      plus      --     dashed
```

# Functionalities

b) From the Matlab built-in help function:



# Functionalities

---

- Graphing exercises in Matlab
- Exercise 1: re-do the following example from the lecture notes

```
.....  
[1] clear all    % deletes all variables in the workspace  
[2] clc         % clears the view on the workspace  
[3] r1 = rand(7,1); % generates uniform random variables  
[4] r2 = [5;4;3;0;-1;-1;-10]; % column vector of the entered values  
[5] plot(r1)  
[6] hold on  
[7] plot(r2)  
.....
```

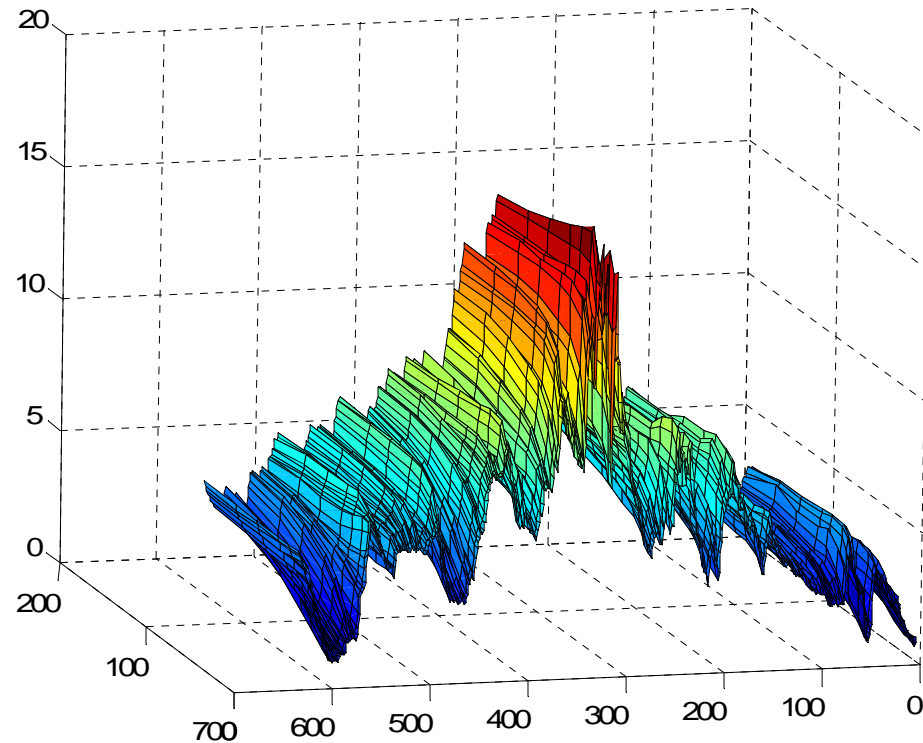
Delete the plot, as usual in Windows(tm), by clicking in the right hand corner of the plot-window. Then proceed with the following:

```
.....  
[1] plot(r1)  
[2] figure  
[3] plot(r2)  
.....
```

# Functionalities

---

- Exercise 2: Do a 3-D plot of the yield curve data contained in Data.xls, sheet:Rates, range:A1-I626



# Functionalities

---

- Variables serve as a storage for our data
  - A normal variable is a 2 or 3 dimensional array
  - A structured variable is a variable of variables
- Arrays:

```
Command Window
>> A = randn(10,2);
>> B = randn(10,2);
>> C(:,:,1) = A;
>> C(:,:,2) = B;
>> whos
  Name          Size          Bytes  Class
  ---          -
  A             10x2             160   double array
  B             10x2             160   double array
  C             10x2x2           320   double array

Grand total is 80 elements using 640 bytes
```

# Functionalities

---

- An example of structured variables:

```
.....  
[1] Q = [1 2 3; 4 5 6; 7 8 9];  
[2] r1 = Q(2, :);  
[3] r2 = Q(3, 2);  
[4] r3 = Q(1:2, 3);  
[5] Sum_data.Q = Q;  
[6] Sum_data.r1 = r1;  
[7] Sum_data.r2 = r2;  
[8] Sum_data.r3 = r3;  
.....
```

We can now investigate the structure variable *Sum\_data* by writing:

```
.....  
[1] Sum_data  
.....
```

will show the variables contained by the structure named *Sum\_data*. We will make more use of structures later in this chapter and also in following chapters once we embark on writing our own Matlab(tm) functions.

# Functionalities

---

- Branching and Looping, i.e. conditional execution of code, requires relation operators:

Operator	Explanation
<, <=	Less than, Less than or equal to
>, >=	Greater than, Greater than or equal to
==	Equal to (not to be confused with "=" that assigns a value)
~=	Not equal to
~	Not

# Functionalities

---

- Branching:

Branching is needed when a given set of instructions are to be performed only if certain criteria are fulfilled. Functions and programs/scripts can be branched by the use of "IF" and "Case" statements. "Case" is useful in the event that a given know number of possible branches exist, while "IF" is a more universal branching structure. The generic forms of these statements can be seen in Matlab(tm) by typing:

```
.....  
[1] help if
```

```
.....  
and,
```

```
.....  
[1] help case  
.....
```

# Functionalities

---

- The if and case statements:

The "if" statement has the following structure:

```
if expression
  statements
elseif
  statements
else
  statements
end
```

and the "case" statement has this structure:

```
switch expression
case expression_A
  statements
case {expression_B, expression_C}
  statements
otherwise
  statements
end
```

# Functionalities

---

- Looping:

Looping can be done in Matlab(tm) by the use of "for" and "while" statements. The former statement is often used when a set of statements are to be executed a known number of times, and the "while" statement is used to loop until a given set of criteria are fulfilled. The generic structure of these statements are given below.

"for" structure	"while" structure
for (variable=1:step_size:N)	while (variable evaluated against expression)
statements	statements
end	end

In the "for" structure the "variable=1:step\_size:N" denotes the counting part of the loop, and should be read: for variable equal one to N in steps of step\_size.

# Functionalities

---

- An function in Matlab typically has the following components:
  - Header that specifies the name, input and output variables, and the keyword function that tells Matlab that the following should be interpreted as a function
  - An introduction section that tell the user how to use the function
  - The text that performs the calculations of the function
- Functions are a useful structure for tasks that are repeated many time and that can be formulated in somewhat general terms

# Functionalities

- An example of a function:

```
bond - WordPad
File Edit View Insert Format Help
[Icons]
function [out] = bond(in)
*
* Chapter: Risk and Return
*
* Calculates the price, modified duration and convexity of a coupon bond
*     having annual payments
*
* Usage:
*     [out] = bond(in)
*
* in (structure):
*     .Y    - yield of the bond 1-by-1 e.g. 0.053 = 5.3%
*     .C    - coupon of the bond 1-by-1 e.g. 5 (=5% bond)
*     .N    - years to maturity 1-by-1 e.g. 2.5 (=2 years and 6 months)
* out (structure):
*     .P    - Price of the bond
*     .MD   - Modified duration of the bond
*     .Conv - Convexity of the bond
*
* date: November 2006
* report bugs to: email@kennyholm.com
*
y = in.Y; c = in.C; n = in.N;
p = c/y*( 1-1/(1+y)^n ) + 100/(1+y)^n;
md = 1/p*( c/y^2*(1-1/(1+y)^n) - n/(1+y)^(n+1)*(100-c/y) );
conv = (1/p)*(1/(1+y)^2*( (n+1)*n/(1+y)^n*(100-c/y) - 2*c/(1+y)^(n-2)*(1/y^3 + n/((1+y)*y^2)) ) + 2*c/y^3);
out.P=p; out.MD=md; out.Conv=conv;
```

# Functionalities

---

- One of the most useful function for our purposes is the Matlab built-in function *fmincon*
- It can be use to find the arguments that minimise a particular function
- We will use this function to:
  - Solve likelihood optimisation
  - Find optimal regression coefficients
  - Find optimal portfolio weights
  - Find efficient frontier portfolios
  - Find the yield (internal rate of return) on a stream of bond payments

# Functionalities

## Command Window

```
>> help fmincon
FMINCON finds a constrained minimum of a function of several variables.
  FMINCON attempts to solve problems of the form:
      min F(X)  subject to:  A*X <= B, Aeq*X = Beq (linear constraints)
      X          C(X) <= 0, Ceq(X) = 0   (nonlinear constraints)
                        LB <= X <= UB

X=FMINCON(FUN,XO,A,B) starts at XO and finds a minimum X to the function
FUN, subject to the linear inequalities A*X <= B. FUN accepts input X and
returns a scalar function value F evaluated at X. XO may be a scalar,
vector, or matrix.

X=FMINCON(FUN,XO,A,B,Aeq,Beq) minimizes FUN subject to the linear equalities
Aeq*X = Beq as well as A*X <= B. (Set A=[] and B=[] if no inequalities exist.)

X=FMINCON(FUN,XO,A,B,Aeq,Beq,LB,UB) defines a set of lower and upper
bounds on the design variables, X, so that a solution is found in
the range LB <= X <= UB. Use empty matrices for LB and UB
if no bounds exist. Set LB(i) = -Inf if X(i) is unbounded below;
set UB(i) = Inf if X(i) is unbounded above.

X=FMINCON(FUN,XO,A,B,Aeq,Beq,LB,UB,NONLCON) subjects the minimization to the
constraints defined in NONLCON. The function NONLCON accepts X and returns
the vectors C and Ceq, representing the nonlinear inequalities and equalities
respectively. FMINCON minimizes FUN such that C(X)<=0 and Ceq(X)=0.
(Set LB=[] and/or UB=[] if no bounds exist.)
```

## Command Window

```
[X,FVAL,EXITFLAG,OUTPUT,LAMBDA,GRAD,HESSIAN]=FMINCON(FUN,XO,...) returns the
value of the HESSIAN of FUN at the solution X.
```