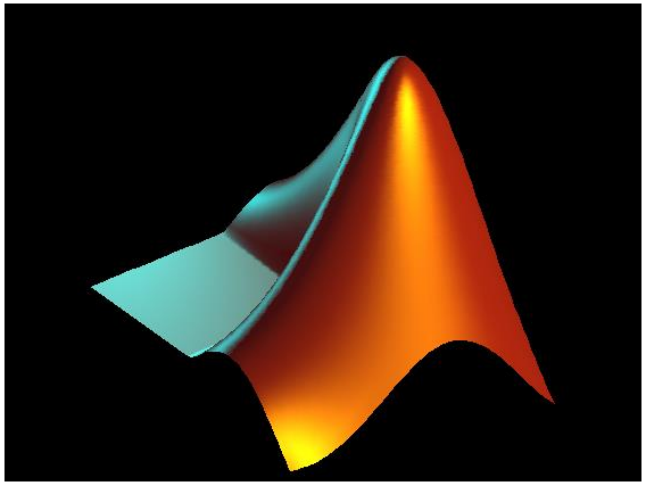
Computational Mathematics with MATLAB
Topic 3



Array indexing and manipulation
Introduction to script writing

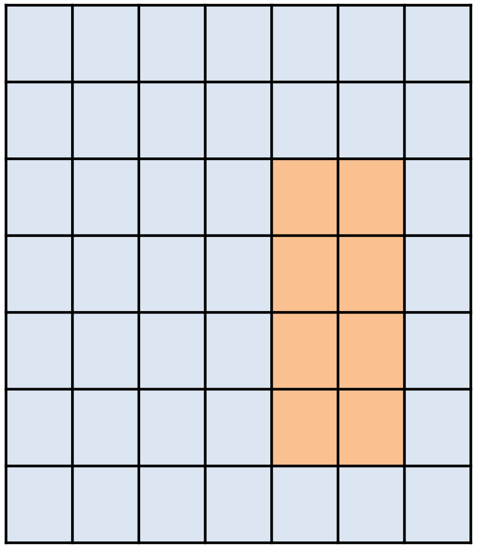
Outline

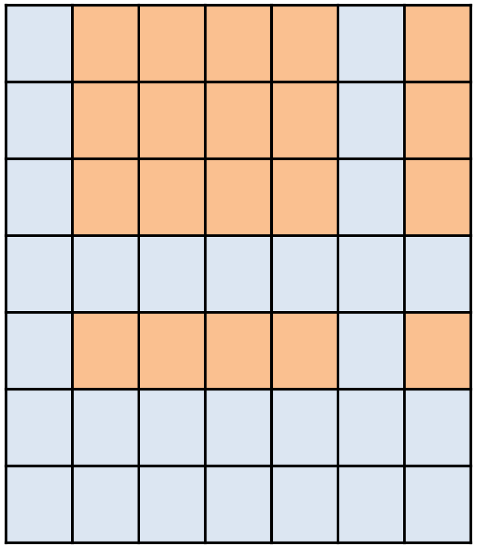
- indexing arrays and subarrays
- the colon operator (:) and the **end** keyword
- linear indexing of matrices
- rearranging array elements
- columnwise operating functions

  (examples: basic statistical functions e.g. **min, sum, sort**)
- displaying variables using the **disp** function
- writing and running script files
- using the **input** function
- creating scripts for plotting

Subarrays

- A **subarray** of a rectangular array is created by selecting elements in specific rows and/or columns only.





```
Rows selected: 3-6
Columns selected:5-6
```

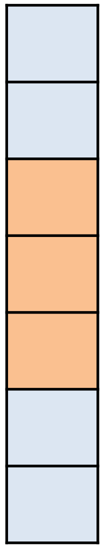
```
Rows selected: 1-3 and 5
Columns selected:2-5 and 7
```

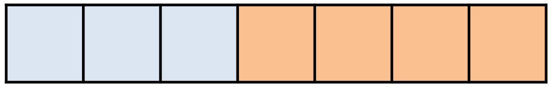
Referencing Subarrays – Examples

Complete rows or columns can be selected by using the colon operator (:) as shown below

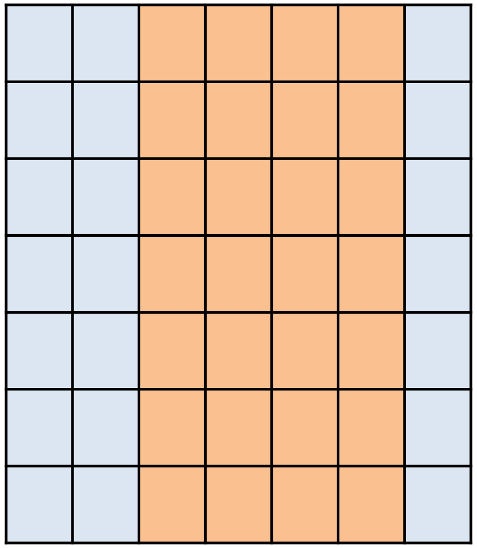
The end keyword can be used to represent the last index

v(3:5)

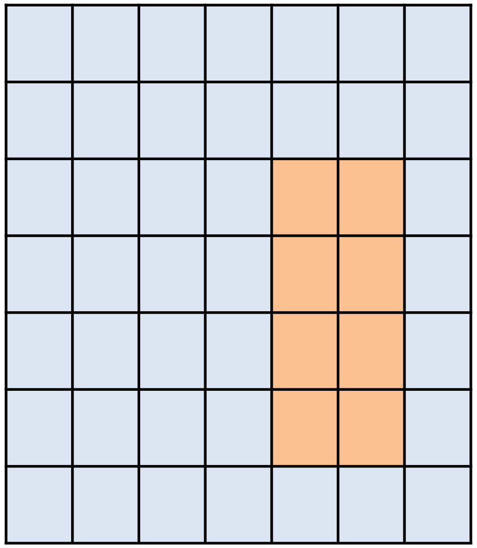




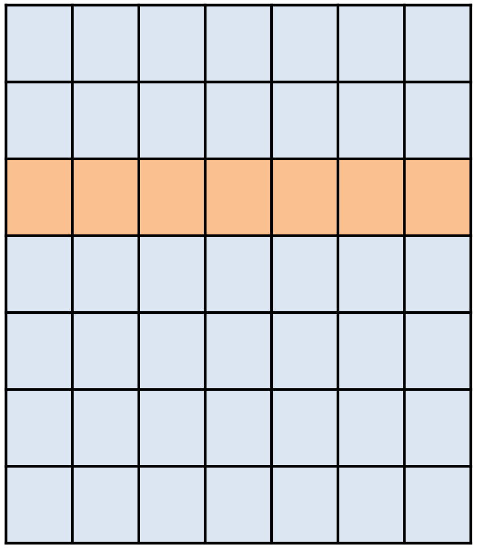
u(4:7) or u(4:end)



A(: , 3:6)

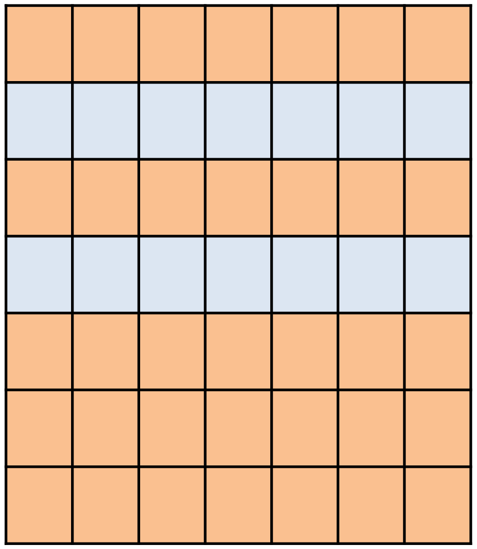


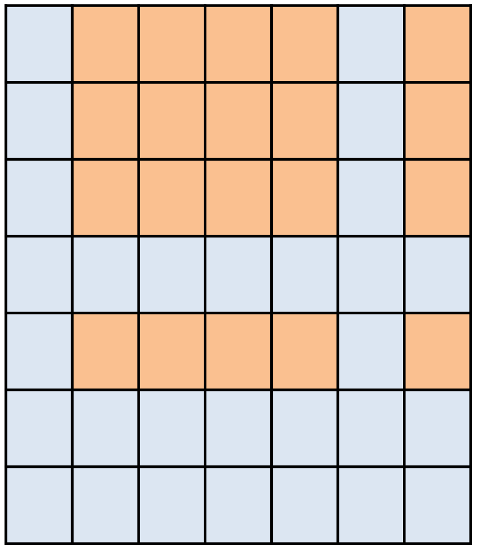
A(3:6 , 5:6)



A(3, :)

Referencing Subarrays – Examples





A([1, 3, 5:end], :)

A([1:3, 5], [2:5,7])

- Use the (:) to select ALL elements in a row (R) or column (C).
- If there are gaps between the selected rows/columns, list the selection in list/vector format, as shown.
- A([1:3, 5], [2:5,7]) means that the selection includes the elements in R1-3 and R5 from C2-5 and C7.

Array Manipulation Basics

Let **A =**

1	6	11	16	21	26
2	7	12	17	22	27
3	8	13	18	23	28
4	9	14	19	24	29
5	10	15	20	25	30

Could you recreate A without entering the elements 1-by-1?

The elements of A can be changed (overwritten) in the workspace editor or from the command line. Examples (try them!):

`>> A(2, 3) = 777`	changes element A(2,3) to 777
`>> A(3, [2:4,6]) = 777`	changes the elements in R3 from C2-4 and C6 to 777
`>> A(6, :)= 1:6`	the 6th row of A is defined as [1 2 3 4 5 6]
`>> A(1, : ) = []`	removes Row 1
`>> A( : , [3,5]) = []`	removes Column 3 and Column 5

Subscripts vs Linear Indices

A is a 5 x 6 matrix so it has 30 elements.

- A is a special matrix because each element of A is equal to its **linear index.**
- Linear indices are obtained by counting the elements along the columns, starting at the top left corner.

5 x 6 *Subscript Matrix*

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)

5 x 6 *Matrix of Linear Indices* (*A*)

1	6	11	16	21	26
2	7	12	17	22	27
3	8	13	18	23	28
4	9	14	19	24	29
5	10	15	20	25	30

A(3,5) is A(23) in a 5 x 6 matrix.

Index to Subscript Conversion (both ways)

Examples

```
>> k=sub2ind([7,8],3,4)

k =
    24

% element (3,4) in a 7 x 8
matrix has linear index 24


>> [i,j] = ind2sub([7,8],24)

i =
     3

j =
     4
```

```
k = sub2ind([n,m],i,j)
```

Finds the linear index of element (*i*, *j*) in an *n x m* matrix and stores the output in variable *k*

```
[i,j] = ind2sub([n,m],k)
```

Finds the row and column numbers of element *k* in an *n x m* matrix and stores the output in variables *i* and *j*.

It no output variables are given, then only the row number is stored in ans.

Rearranging array elements (using `reshape`)

`reshape(A,p,q)`

Rearranges the elements of a matrix A into a *p x q* matrix so that the linear indices remain unchanged. The total number of elements cannot change so this only works if ***p·q*** **`= numel(A)`**

```
>> C=reshape(A,3,10), D=reshape(A,6,5)

C =
     1     4     7    10    13    16    19    22    25    28
     2     5     8    11    14    17    20    23    26    29
     3     6     9    12    15    18    21    24    27    30

D =
     1     7    13    19    25
     2     8    14    20    26
     3     9    15    21    27
     4    10    16    22    28
     5    11    17    23    29
     6    12    18    24    30
```

Examples of functions acting elementwise

```
>> B=A([1,3],:)/2    % create a test matrix

B =
    0.5000    3.0000    5.5000    8.0000   10.5000   13.0000
    1.5000    4.0000    6.5000    9.0000   11.5000   14.0000

>> C=round(B), D=log2(B), E=mod(B,2)

C = % rounding each element to the nearest integer

     1     3     6     8    11    13
     2     4     7     9    12    14

D =  % taking the base 2 log of each element

   -1.0000    1.5850    2.4594    3.0000    3.3923    3.7004
    0.5850    2.0000    2.7004    3.1699    3.5236    3.8074

E =  % the remainder after diving each element by 2

    0.5000    1.0000    1.5000         0    0.5000    1.0000
    1.5000         0    0.5000    1.0000    1.5000         0
```

Examples of functions acting columnwise

```
>> F=randi(100,4,7)          % create a test matrix

F =
    85    93    62    59    76     8    94
    26    35    48    55    76     6    13
    82    20    36    92    39    54    57
    25    26    84    29    57    78    47

>> X=sum(F), Y=max(F), Z=sort(F)

X =                                         % vector of
   218   174   230   235   248   146   211  column sums

Y =
    85    93    84    92    76    78    94   % vector of
                                             column maximums

Z =
    25    20    36    29    39     6    13   % each column is
    26    26    48    55    57     8    47   sorted separately
    82    35    62    59    76    54    57   (top to bottom)
    85    93    84    92    76    78    94
```

Examples of functions acting columnwise

- Many basic statistical functions act **columnwise** by default:

  `min, max, sort, sum, cumsum, mean, median, std,` etc

- This type of functions can also be made to **act along rows** using additional (optional) parameters.

Try the following

```
>> X=sum(F,2)
>> Y=max(F,[],2)
>> Z=sort(F,2)
```

To calculate the min or mean or sum of (the whole) matrix F, use

```
>> min(min(F))
>> max(max(F))
>> sum(sum(F))
```

Displaying Variables

Let variable x be defined as

```
>> x=10;
```

We can later display the value of x by simply entering its name

```
>> x
x =
    10
```

in which case the variable name x is also shown.

The **disp** function displays the value of a variable *without printing its name:*

```
>> disp(x)
    10
```

The disp function

The argument of **disp** can also be a string, e.g.

```
>> disp('hello')
hello!
```

Important: the **disp** function can only take one argument!

To list a sequence of variables, we must include them in a vector:

```
>> x1=12;  x2=23;  x3=52;  x4=67;  x5=81;
>> disp (x1, x2, x3, x4, x5)          ← this does NOT work:
??? Error using ==> disp
Too many input arguments.

>> disp ([x1, x2, x3, x4, x5])        ← this does
12    23    52    67    81
```

MATLAB Scripts

- A script is a sequence of MATLAB commands which is saved in .m (m-file) format
- To open MATLAB's integrated text editor, simply select the **New Script** from the top menu bar or press **CTRL+N.**

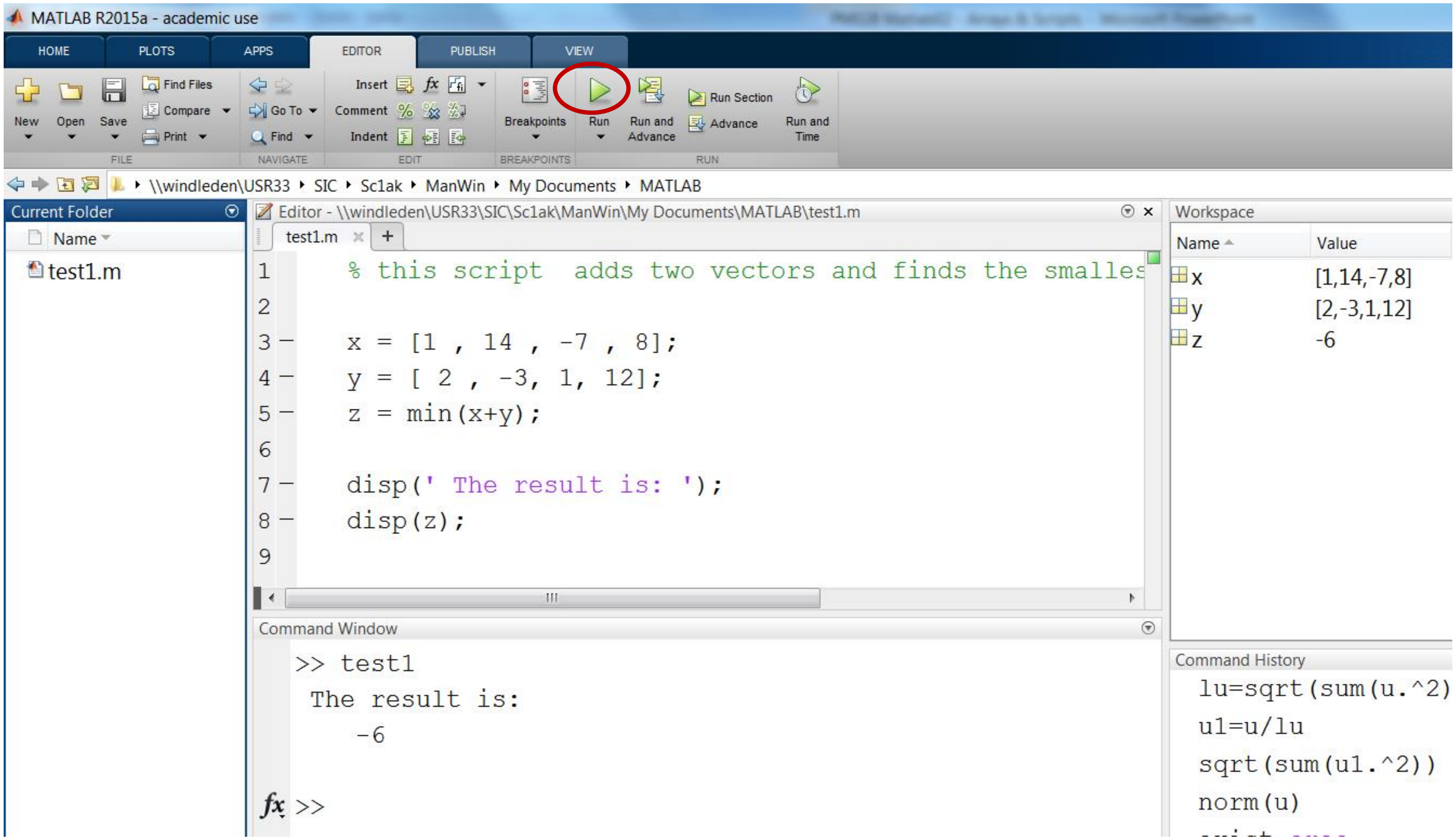
Copy the following code into the Editor and press the Run button or F5 (to save and run the script)

```
% this script adds two vectors and finds the smallest element
of the new vector

x = [1 , 14 , -7 , 8];
y = [ 2 , -3, 1, 12];
z = min(x+y);

disp(' The result is: ');
disp(z);
```

Basic Script Example



```
% this script  adds two vectors and finds the smalles

x = [1 , 14 , -7 , 8];
y = [ 2 , -3, 1, 12];
z = min(x+y);

disp(' The result is: ');
disp(z);
```

```
>> test1
 The result is:
    -6

>>
```

Name	Value
x	[1,14,-7,8]
y	[2,-3,1,12]
z	-6

```
lu=sqrt(sum(u.^2)
u1=u/lu
sqrt(sum(u1.^2))
norm(u)
```

The naming rules are the same for files and variables.

Writing Scripts

- Running a script will give the same result as inputting each line of the script in the MATLAB command window and pressing Enter.
- Variables created in a script will appear in the Workspace and will remain there until the MATLAB session is over.
- The output of any statement without a closing ; will appear in the Command Window
- It is also possible to run a saved script by entering its name in the Command Window (without the .m extension). This will work if the file is in the Current Directory or if it is saved on the MATLAB Path.
- To check/change your MATLAB Path, go to

Home -> Environment -> Set Path

Scripts with User Input

- Our first program was not very useful because variables x and y were fixed inside the script. We can only change the input if we rewrite the script.
- It is a better solution to use the **input** function to prompt (ask) for values from within the program.

`x=input('text ')` **displays the chosen text in the Command Window and requests user input. Whatever the user enters will be stored as variable x.**

```
>> x = input('Input array x: ');
Input array x: [4 5 ; 6 7]
>> disp(x)
     4     5
     6     7
```

For clarity, always include a space before the closing quotation mark.

Basic Script Example with User Input

It is good practice to add comments to describe your code.

Do you find it difficult to write comments? Maybe it is so because you are not sure about what you are doing!

Try the following

```
% this script adds two arrays and finds the minimum of the new array

clc % to clear the command window

% instructions to the user
disp('Please use arrays of the same size!')
% user input
x = input('Input array x: ');
y = input('Input array y: ');
% calculating the output
z = min(x+y);

% we cannot combine string and numerical variables inside disp
disp(' The result is: ');
disp(z);
```

Practice Problem 1

Write a script to calculate the angle between 2 vectors. The vectors should be chosen by the user. Remind the user that the input vectors must have the same dimensions!

It is always a good idea to create a **structure plan** (see example below).

Step 1: Display a message asking the user to input vectors of the same size
Step 2: Take in the 1st input vector and name it u
Step 3: Take in the 2nd input vector and name it w
Step 4: Call the angle variable ang. Write down the all calculations you need to do to find ang (you don't have calculate the angle in 1 line). It is ok to introduce additional variables – just note them in your structure plan.
Step 5: Display the output (ang)
Step 6: Display a short text explaining the output
(optional – but if you do this then display the text before displaying ang)

(See also 2.3.8 in the textbook – Structure Plan section)

Practice Problem 2

Write a script that does exactly what the ind2sub function does (coverts linear indices into subscripts).

Input variables (3):

number of rows (`n`), number of columns (`m`), linear index (`k`)

Output variables (2):

row number (`i`), column number (`j`)

* Start with a structure plan

* Avoid using variable names `i` and `j` if you are working with complex numbers

Practice Problem 3

Write a script which solves the following problem.

Let A, B and C be 3 points in 3D space.

Find the distance between Point C and the line which passes through Points A and B.

* Start with a structure plan

* Remember the distances handout (Week 1 - Maths Workshop)

Plot Script Example – Cosine Graphs

```
close all % open a new figure window

% define the independent variable
x=linspace(0,2*pi,500);

% define the 3 dependent variables
f1=cos(x); f2=cos(3*x); f3=cos(8*x);

hold on % plot the graphs on top of each other
plot(x,f1,'k','LineWidth' ,2)
plot(x,f2,'g','LineWidth' ,2)
plot(x,f3,'m','LineWidth' ,2)

% use the same horizontal and vertical scale
axis equal

% add title, legends and axis labels
title('The graphs of cos(x), cos(3x) and cos(8x)')
legend('cos(x)',' cos(3x)', 'cos(8x)',0)
xlabel('x')
% to make the y-label horizontal, set the angle of rotation to 0
ylabel('f(x)', 'rot', 0)
```

Plot Script Example – Line Graphs

```
% Plot two sets of 3 parallel lines

close all
x=-5:0.01:5;

m=3*rand; % choose the gradient randomly.
% rand is short for rand(1)
% m will be between 0 and +3

% define the 6 dependent variables (lines)
y1=m*x;    y2=-m*x;  y3=m*x-2;
y4=-m*x-2; y5=m*x+2; y6=-m*x+2;

hold on
plot(x,y1,'m',x,y2,'c',x,y3,'b','LineWidth' ,2) % plot 3 lines at once
plot(x,y4,'g',x,y5,'r',x,y6,'y','LineWidth' ,4) % plot 3 thicker lines

axis equal
title('Sets of Parallel lines')
xlabel('x');
```

The gradient is recalculated each time the script is run.

Keep pressing the Run button (or F5) and see how the figure changes.