MATLAB 簡介

MATLAB (MATrix LABoratory) 是一極為有用的數值計算工具，它除了提供簡單的加減乘除運算外，亦可繪圖，同時具有強大的程式功能以及內建程式庫，所以 MATLAB 可取代一般如 Fortran 或 C 之程式語言進行程式設計及數值計算，也可用圖形表示計算結果。本系四樓電腦教室中電腦皆已安裝 MATLAB ，同學們可前往使用；網路上有相當多有關 MATLAB 的介紹及使用說明[1]，以下為與本課程相關之 MATLAB 簡介[2] ：

1. 四則計算

MATLAB 可如計算機般的進行加減乘除運算，如：

```
»  ( 1 + 3 ) * 4 / 5
ans = 3.2000
» 3.4^12
ans =
  2.3864e+006
```

四則運算中，+ 及 - 分別為加減、 * 及/為乘除、^ 為次方。 至於取餘數(C 語言用%)在 Matlab 是用 **rem**，如

```
>> a = 9
a =
     9
>> b = 2
b =
     2
>> rem(a,b)
ans =
     1
```

[Example]

```
»  format long
»  sqrt(-1), 1/0
ans =
                  0 + 1.00000000000000i
ans =
   Inf
```

(Infinity, ∞)

```
» 0*log(0)
ans =
```

---

1 http://scv.bu.edu/SCV/Tutorials/MATLAB/
http://www.math.siu.edu/matlab/tutorials.html

2 本講義資料取自 http://www.mit.edu:8001/people/abbe/matlab/main.html 以及黃世宏教授的講義。

```
   NaN        ( Not-a-Number )
» pi
ans =
   3.14159265358979
```

在 MATLAB 中 ， 可定義 **a** 為一變數，如：

```
»  a = 3
a = 3
```

其中字母的大小寫代表不同的變數，例如 **a** 及 **A** 不同。在 MATLAB 中所有的變數皆內定為倍精準(就是 C 語言中的 double 型態)。當然，除了倍精準外，Matlab 的變數型態也可設為單精準、整數、字串等型態。例如：

```
» a = single(5)
a = 5
```

是把 a 設為單精準的浮點值。此外，如果安裝 symbolic 模組後，MATLAB 也可把數字或計算以任何位數之精準度表示之，如：

```
» a = vpa(sqrt(2),100)
a =
1.414213562373095048801688724209698078569671875376948073176679737990732478462107038850387534327641573
» b = vpa(pi,50)
b =
3.1415926535897932384626433832795028841971693993751
```

其中 **vpa** 為 variable precision arithmetic 之簡寫，括號內 100 即為位數。 不過，MATLAB 必須先安裝 symbolic toolbox (符號運算工具箱) 才能使用 vpa 這個指令。本校計網中心 MATLAB 的版權已包含該模組。

前已談及，在 MATLAB 中，任何變數的內定值事實上都是倍精準之矩陣(或稱陣列； matrix , array)；如果變數看起來像 scalar ，其實為一個 1×1 matrix (e.g., **a = 3**)，若變數看起來像 vector ，其實為一個 $N \times 1$ or $1 \times N$ matrix 。以下幾個例子說明如何在 MATLAB 中建立矩陣變數：

```
»  v = [ 1 3 6 8 9 ]
v =     1     3     6     8     9
```

如上述之例子，矩陣中同列(row)之元素間用空隔(space)區分，不同列則可用分號『；』或分行輸入，如：

```
»  c = [1 3 6; 2 7 9; 4 3 1]
c =
     1     3     6
     2     7     9
     4     3     1
```

或

```
» c = [  1  3  6
         2  7  9
         4  3  1  ]
c =
     1     3     6
     2     7     9
     4     3     1

» c = [ -1.3 sqrt(3) (1+2+3)*4/5 ]

c =

 -1.3000e+000  1.7321e+000  4.8000e+000
```

若在輸入行中最後輸入分號『；』，則在輸出螢幕上就不會顯示結果；如

```
» a = 2;
» b = 3;
» d = a + b;                    ( 不 顯 示 結 果 )
» d = a + b                     ( 顯 示 結 果 )
d =
     5
```

注意在 Matlab 及 C 語言中，『；』的用法不同。若要輸入一組數值為等間距的向量或矩陣時，可使用

```
» b = 1 : 2 : 15
b =
     1     3     5     7     9    11    13    15
```

這時就可得一 **b = [1 3 5 7 9 11 13 15]**的向量，其中每相隔的值相差為 2。其中間隔的值為正數或負數皆可。若省略間隔之值，則 MATLAB 內定其間隔值為 1，如 **b = 1 : 5** 得 **b = [1 2 3 4 5]**。一矩陣 **v** 的轉置矩陣 (transpose matrix ，行與列互換的矩陣) 可用 **v'**得之，如

```
» v = [ 1 3 6 8 9 ]
v =
     1     3     6     8     9
» v'
ans =
     1
     3
     6
     8
     9
```

Other vector generation functions include **linspace**, which allows the number of points rather than the increment, to be specified, e.g.,

```
» k = linspace(0,1,5)
k =
```

```
    0   2.5000e-001   5.0000e-001   7.5000e-001   1.0000e+000
```

and **logspace**, which generates logarithmically evenly spaced vectors:

```
»  k = logspace(0,2,5)
k =
  1.0000e+000   3.1623e+000   1.0000e+001   3.1623e+001   1.0000e+002
```

or k = [10^0, $10^{0.5}$, 10^1, $10^{1.5}$, 10^2].

A collection of functions generates special matrices that arise in linear algebra and signal processing: company, diag, gallery, hadamard, hankel, hilb, invhilb, magic, pascal, toeplitz, vander. 例如我們可利用 magic 指令創造一 4×4 的魔術方陣：

```
»  magic(4)
ans =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

上述的方陣中，各行與各列以及對角線的和均相同。

[練習] 啟動 Matlab 後在視窗中鍵入 **x = -1 : 0.1 : 1** ，然後輸入下列指令：

sin(x) ; **sqrt(x)** ; **x'** ; **2*x** ; **x/5**

2. Dealing with Matrices

Once you have a matrix, e.g.,

```
»  c = [1 3 6; 2 7 9; 4 3 1]

c =
     1     3     6
     2     7     9
     4     3     1
```

you can refer to specific elements in it. MATLAB indexes matrices by row and column. **c(3,1)** is the element in the third row, first column, which is 4. **c(2:3,1:2)** gives you the elements in rows 2~3, and columns 1~2, so you get

```
»  c(2:3,1:2)

ans =
     2     7
     4     3
```

as a result. **c(1:3,2)** gives you the elements in rows 1~3, and the second column, that is, the entire second column. You can shortcut this to:

```
»  c(:,2)
```

literally telling Matlab to use all the rows in the second column, i.e., the 2nd column:

```
ans =
     3
```

```
     7
     3
```

You can get a whole row of a matrix with

```
»  c(1,:)

ans =

     1     3     6
```

This literally tells Matlab to take the first row, all columns.

You can also refer to any matrix with only one index. It will use that index to count down the columns (先往下數，再往右；先行再列), for example,

```
»  c(6)

ans =

     3
```

Big matrices can be constructed using little matrices as elements. For example, we could attach another row to our matrix c with

```
» r = [10 11 12];

» c = [c;r]

c =
     1     3     6
     2     7     9
     4     3     1
    10    11    12
```

When you have a matrix or vector (anything with more than one element) you need to do a few things to make sure all of your math does what you want it to. You can add a constant or multiply by a constant normally (const+c, const*c, etc.) If you have data in two matrices that correspond (for example, a time vector and an x position vector that has x values for each point in time), you can add and subtract those normally (it will map each element properly.)

To multiply, divide, or raise to a power when you have a matrix or vector that is acting as a set of data points, you need to use

.* ./ .^

so that Matlab will multiply *each element* in the matrix instead of trying to do matrix multiplication or division.

[Example]

```
» c = [1 3 6; 2 7 9; 4 3 1]

c =
     1     3     6
     2     7     9
```

```
     4     3     1

» c^2

ans =
    31    42    39
    52    82    84
    14    36    52

» c.^2

ans =
     1     9    36
     4    49    81
    16     9     1

» 1./c

ans =

    1.0000    0.3333    0.1667
    0.5000    0.1429    0.1111
    0.2500    0.3333    1.0000

» c^-1

ans =
    0.4000   -0.3000    0.3000
   -0.6800    0.4600   -0.0600
    0.4400   -0.1800   -0.0200
```

Note that the last expression, **c^-1**, yields the inversion of the matrix c, which can also be evaluated by **inv(c)**. Of course, it can treat matrices as actual matrices, so you can solve something like [A]x = b where A is a matrix of coefficients, x is a column vector of the x values you want to find, and b is also a column vector just by doing

x = A\b

after you define A and b. The \ represents "left division" (since to solve that equation you would have to divide both sides by A on the *left* side, since order is significant when dealing with matrices.).

[Example]

```
» 1/2
ans =
    0.5000
» 1\2
ans =
     2
```

若要解下列的聯立方程式：

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 3 \\ 1 & 1 & -1 \end{bmatrix} \mathbf{X} = \begin{bmatrix} 8 \\ 19 \\ 0 \end{bmatrix}$$

可用

```
» A = [ 1  2  1
        2  4  3
        1  1 -1 ]

A =
     1     2     1
     2     4     3
     1     1    -1

» B = [ 8 ; 19 ; 0 ]

B =
     8
    19
     0

» X = A\B

X =
     1
     2
     3
```

[練習一]　執行下列敘述並觀察結果：

```
x = [ 1 3 5 7 9 ]
y = -1 : 0.5 : 1
x.^y
x.*y
x./y
```

[練習二]　請建立下述的矩陣：

$$\mathbf{A}=\begin{bmatrix}1 & 3 & 9\\ 5 & 7 & 10\\ 3 & 5 & 8\end{bmatrix}$$

並請檢查 `A(1,1), A(2,1), A(4); A(1:2,:), A(:,1) A(:,2:3)` 是什麼？

[練習三]　利用 Matlab 解下列聯立方程式：

$$\begin{cases}2x+3y+4z+7w=12\\ 3x-4y+8z-w=2\\ 5x+6y-3z+2w=9\\ x+y-6z+4w=10\end{cases}$$

3. Useful Functions

Matlab has a lot of built-in functions (內建函數). To use a function, just type functionname(arguments) with the appropriate name and arguments. If you create a time vector **`t = 1:.25:6`**; you can get standard functions like

x = sin(t)

which returns a vector the same size as t, with values for the sin of t at all points.

```
»  t=1:.25:6

t =
  Columns 1 through 7
    1.0000    1.2500    1.5000    1.7500    2.0000    2.2500    2.5000
  Columns 8 through 14
    2.7500    3.0000    3.2500    3.5000    3.7500    4.0000    4.2500
  Columns 15 through 21
    4.5000    4.7500    5.0000    5.2500    5.5000    5.7500    6.0000

»  x = sin(t)

x =
  Columns 1 through 7
    0.8415    0.9490    0.9975    0.9840    0.9093    0.7781    0.5985
  Columns 8 through 14
    0.3817    0.1411   -0.1082   -0.3508   -0.5716   -0.7568   -0.8950
  Columns 15 through 21
   -0.9775   -0.9993   -0.9589   -0.8589   -0.7055   -0.5083   -0.2794
```

You can also do things like

s = sum(c)

which sums all the columns in a matrix c and returns a row vector of the sums.

```
»  sum(c)

ans =

     7    13    16
```

The function d = det(c) takes a matrix c and returns the determinant of it.

```
»  d = det(c)

d =

   -50
```

The Matlab booklet has a list of many of these useful functions. Here is a summary of relevant functions, followed by some examples:

```
Function      Meaning              Example
================================================
sin           sine                 sin(pi)   = 0.0
cos           cosine               cos(pi)   = 1.0
tan           tangent              tan(pi/4) = 1.0
asin          arcsine              asin(pi/2)= 1.0
acos          arccosine            acos(pi/2)= 0.0
atan          arctangent           atan(pi/4)= 1.0
exp           exponential          exp(1.0)  = 2.7183
log           natural logarithm    log(2.7183)  = 1.0
log10         logarithm base 10    log10(100.0) = 2.0
```

Note that the arguments to trigonometric functions are given in radians.

Some Special Matrices

Matlab also provides a number of useful built-in matrices:

```
Function      Meaning                    Example
=========================================================
ones(m,n)     m x n matrix of 1's      P = ones(2,3)
zeros(m,n)    m x n matrix of 0's      Z = zeros(2,3)
eye(n)        n x n identity matrix    I = eye(3)
diag(d)       diagonal matrix with     d = [ -3  4  2 ]
              diagonal entries d       D = diag(d)
diag(A)       extracts the diagonal    A = [ 0 1 8 7 ;  3 -2 -4 2 ;
              entries from A                 4 2 1 1 ]
                                       a = diag(A)
rand(m,n)     m x n matrix of random   Y = rand(2,3),  y = rand
              numbers
magic(n)      an n x n magic square    M = magic(4)
```

[練習一]　下面的敘述 Matlab 會給什麼結果?

```
x = 1 : 1 : 3
z = rand(3)
y = [ z ; x ]
c = rand(2)
e = eye(2)
f = [ c e ones(2) ]
d = sqrt(c)
a = inv(c)
b = c*c
g = c.*c
```

[練習二]　下面的敘述 Matlab 會給什麼結果?

```
A = [ 1 3 5 7 9 ];
B = diag(A, -1)
C = diag(A, 1)
```

[練習三]　下面的敘述 Matlab 會給什麼結果?

```
A = ones(2,5)*6
B = rand(3,5)
C = [ A ; B ]
C = [ A B ]   % 有沒有錯誤訊息?
A = eye(3)*6
C = [ A B ]
```

4. Help and Other Tools

Places to get help:

- Typing "**`help`**" at the Matlab prompt gives you a list of all the possible directories matlab can find commands in (which also tells you its "search path", or a list of the directories it is looking in for commands.)

- Typing "**`help commandname`**" gives you help on a specific command. For example,

```
» help roots

 ROOTS  Find polynomial roots.
    ROOTS(C) computes the roots of the polynomial whose coefficients
    are the elements of the vector C. If C has N+1 components,
    the polynomial is C(1)*X^N + ... + C(N)*X + C(N+1).

    See also POLY, RESIDUE, FZERO.
```

- Typing "**`lookfor keyword`**" gives you a list of commands that use that keyword. ie, "**`lookfor integral`**" lists commands that deal with integrals. It's pretty slow, choose the word wisely. You can use control-c to stop searching when you think you've found what you need. For example,

```
» lookfor integral
```

```
ELLIPKE Complete elliptic integral.
EXPINT Exponential integral function.
DBLQUAD Numerically evaluate double integral.
INNERLP Used with DBLQUAD to evaluate inner loop of integral.
QUAD   Numerically evaluate integral, low order method.
QUAD8  Numerically evaluate integral, higher order method.
COSINT Cosine integral function.
SININT Sine integral function.
COSINT Cosine integral function.
FOURIER Fourier integral transform.
IFOURIER Inverse Fourier integral transform.
SININT Sine integral function.

» lookfor integration

CUMTRAPZ Cumulative trapezoidal numerical integration.
TRAPZ  Trapezoidal numerical integration.
LOTKADEMO Demonstrate numerical integration of differential equations.
ADAMS Simulink 1.x ADAMS integration algorithm.
EULER Simulink 1.x EULER integration algorithm.
GEAR Simulink 1.x GEAR integration algorithm.
LINSIM Simulink 1.x LINSIM integration algorithm.
RK23 Simulink 1.x rk23 integration algorithm.
RK45 Simulink 1.x RK45 integration algorithm.
SFUNMEM A one integration-step memory block S-function.
INT    Alternative entry to the symbolic integration function.
```

- Typing "**doc**" starts up a web browser with the Matlab home page. This includes the entire reference manual for Matlab , a whole lot of other information on using matlab, and a pointer to the Matlab Primer, a good introduction to using Matlab.

Some Useful Tools:

- If you accidentally reassign a function name to a variable (i.e., you try saying sum = 3 and then you get errors when you try to use the sum function because it doesn't know it's a function anymore), you can restore it to its normal state using "**clear functionname**". You can also use clear to get rid of all variable values with "**clear**" or "**clear all**".

- **who**

  will tell you all the variables you have currently defined.

```
»  who

Your variables are:
a         b         d         v
ans       c         t         x
```

- **whos**

  will tell you the variables, their sizes, and some other info.

```
»  whos

  Name      Size         Bytes  Class
  a         1x1              8  double array
  ans       1x3             24  double array
  b         1x19           152  double array
```

```
  c         3x3             72  double array
  d         1x1              8  double array
  t         1x21           168  double array
  v         1x5             40  double array
  x         1x21           168  double array
```

- **pi**

  is a function of that returns the value of pi.

```
» pi
ans =
    3.1416
```

- **eps**

  is a function that returns the distance from 1.0 to the next largest floating point number.

```
»  eps
ans =
  2.2204e-016
```

- **Inf**

  The function Inf, which stands for infinity, is found in very few calculator systems or computer languages. On some computers, it is made possible by the IEEE arithmetic implemented in a math coprocessor. On other computers, floating point software is used to simulate a coprocessor. One way to generate the value returned Inf is

```
»  s = 1/0
```

  which results in

```
Warning: Divide by zero.
s =
   Inf
```

- **NaN**

  On machines with IEEE arithmetic, division by zero does not lead to an error condition or termination of execution. It does produce a warning message and a special value that can behave in a sensible manner in subsequent computation.

  The variable NaN is an IEEE number related to Inf but has different properties. It stands for "Not a Number" and is produced by calculations such as Inf/Inf or 0/0.

- **format long / format short**

switch between the long and short display format of numbers. Either way Matlab uses the same number of digits for its calculations, but normally (**`format short`**) it will only display the first four digits after the decimal point.

```
»  format long
»  pi
ans =
   3.14159265358979
»  format short
»  pi
ans =
    3.1416
```

- Typing

  `type functionname` for any function in Matlab's search path lets you see how that function is written.

```
» type bessel
function [w,ierr] = bessel(nu,z)
%BESSEL Bessel functions of various kinds.
%   Bessel functions are solutions to Bessel's differential
%   equation of order NU:
%            2                    2    2
%          x * y'' +  x * y' + (x - nu ) * y = 0
%
%   There are several functions available to produce solutions to
%   Bessel's equations.  These are:
%
%       BESSELJ(NU,Z)    Bessel function of the first kind
%       BESSELY(NU,Z)    Bessel function of the second kind
%       BESSELI(NU,Z)    Modified Bessel function of the first kind
%       BESSELK(NU,Z)    Modified Bessel function of the second kind
%       BESSELH(NU,K,Z)  Hankel function
%       AIRY(K,Z)        Airy function
%
%   See the help for each function for more details.

%   Copyright 1984-2002 The MathWorks, Inc.
%   $Revision: 5.12 $  $Date: 2002/04/09 00:29:44 $

[w,ierr] = besselj(nu,z);
```

5. Complex Numbers and Matrices

Complex numbers are allowed in all operations and functions in MATLAB. There are at least two convenient ways to enter complex matrices. They are illustrated by the statements

```
» A = [1 2; 3 4] + i*[5 6; 7 8]

A =
  1.0000e+000 +5.0000e+000i  2.0000e+000 +6.0000e+000i
  3.0000e+000 +7.0000e+000i  4.0000e+000 +8.0000e+000i
```

and

```
» A = [1+5*i  2+6*i;  3+7*i  4+8*i]
```

which produce the same result. When complex numbers are entered as matrix elements within brackets, it is important to avoid any blank spaces, because an expression like 1 + 5*i, with blanks surrounding the + sign, represents two separated numbers. (The same is true of real numbers; a blank before the exponent part in 1.23 e-4 causes an error.)

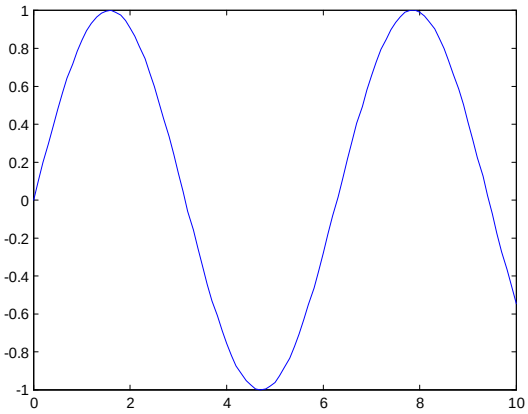
6. 2-D Plotting

The basic syntax to get a plot in Matlab is

```
plot(x1,y1)
```

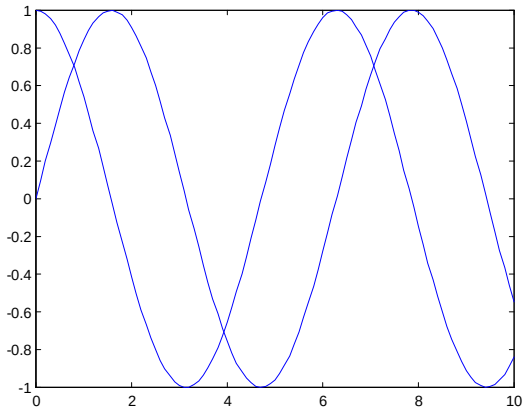
(The x values always come before the y values, x1 and y1 represent variables that your data is stored in.)

```
clear all
» x = 0:0.1:10;
» y = sin(x);
» plot(x,y)
```



If you type a second plot command later, it will clear your first plot. If you type "**hold on**" it will hold the current plot so you can add plots on top of one another (until you reset it by typing "**hold off**".). For example,

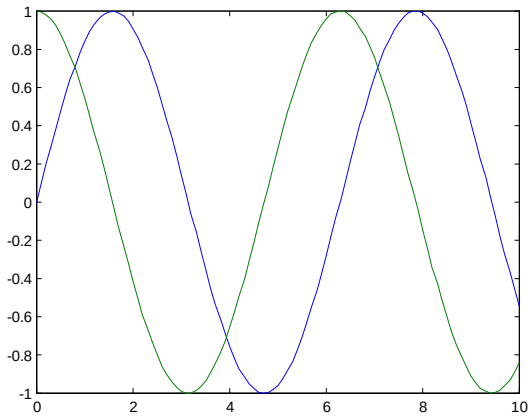
```
» y1 = cos(x);
» hold on
» plot(x,y1)
```



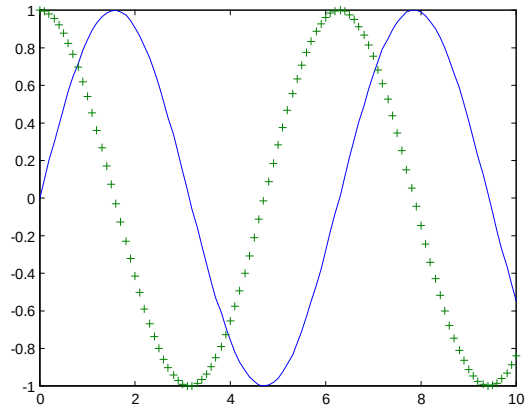
You can plot multiple values with **plot(x1,y1,x2,y2)** and you can specify the color and linetype of a plot as something like **plot(x1,y1,'w*')** to get white *'s for each data point.

```
» clear all
```

```
» clf
» x = 0:0.1:10;
» y1 = sin(x);
» y2 = cos(x);
» plot(x,y1,x,y2)
```




```
» plot(x,y1,'--',x,y2,'+')
```



The available options are listed as follows:

Line Types		Point Types		Colors	
solid	-	point	.	red	r
dashed	--	plus	+	green	g
dotted	:	star	*	blue	b
dashed-dot	-.	oh's	o	white	w
		x's	x	invisible	i

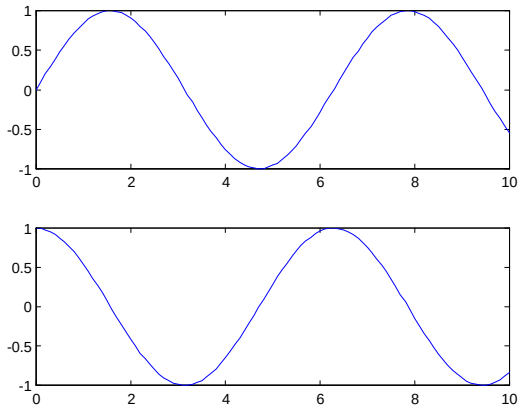
To split your plot into a bunch of smaller plots, you can use the **subplot** command to split it up into rows and columns:

```
subplot(r,c,n)
```

will split the plot window into r rows and c columns of plots and set the current plot to plot number n of those rows and columns. For example, **subplot(2,1,1)** splits the plot window into two rows in a single column and prepares to plot in the top plot. Then your plot command will plot in the top plot. Then you could switch to the

bottom plot with **subplot(2,1,2)** and use another plot command to plot in the bottom plot.

```
» subplot(2,1,1)
» plot(x,y1,'-')
» subplot(2,1,2)
» plot(x,y2,'-.')
```

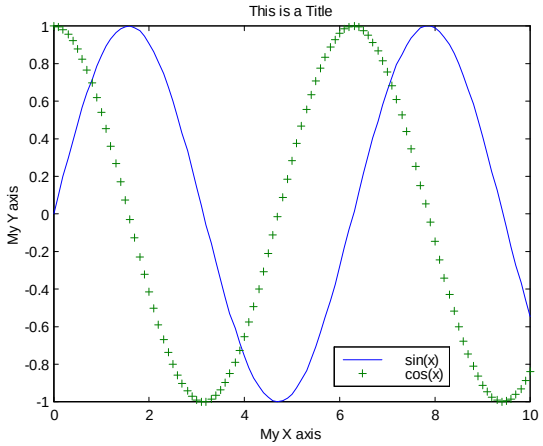


You can add titles, labels, and legends to plots.

```
title('This is a Title')
xlabel('My X axis')
ylabel('My Y axis')
legend('First Thing Plotted','Second Thing Plotted')
```

legend creates a legend box (movable with the mouse) that automatically uses the right symbols and colors and sticks the descriptions in the legend command after them.

```
» plot(x,y1,'--',x,y2,'+')
» title('This is a Title')
» xlabel('My X axis')
» ylabel('My Y axis')
» legend('sin(x)','cos(x)')
```



Other useful 2-D plotting facilities are

```
» semilogx(x,y)
» semilogy(x,y)
» loglog(x,y)
» polar(x,y)
```

[Exercise] 請利用 Matlab 在同一張圖上繪製 $y_1 = x^2 \cos x$ 和 $y_2 = x^2 \sin x$ 兩個函數； x 的範圍在 x = - 2 : 0.1 : 2 ，並請在圖上加入註解。

[Exercise] 同上題，但兩函數分上下兩部份分別繪圖。

7. 3D Graphs and Other Graphics Capabilities

(1) 3D plotting

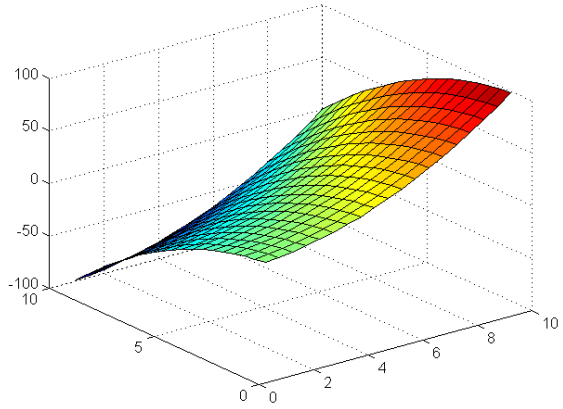
When you make a 3-dimensional plot, you usually have a z variable that is a function of both x and y. When you want x and y to vary over some range, you need a matrix (rather than a vector) for x and y to get a complete domain that covers all the different combinations of those x and y values over some range. A function called **meshgrid** will set up x and y matrices like this for you. The x matrix varies the x down rows and keeps it constant in columns, and the y matrix varies the y in columns and keeps it constant across rows, so you get all combinations of x and y if you use the two matrices.

To get a rectangular domain that goes from x = 1 to x = 10 in steps of .5, and from y=1 to y=10 in steps of .5 using **meshgrid**, you can use

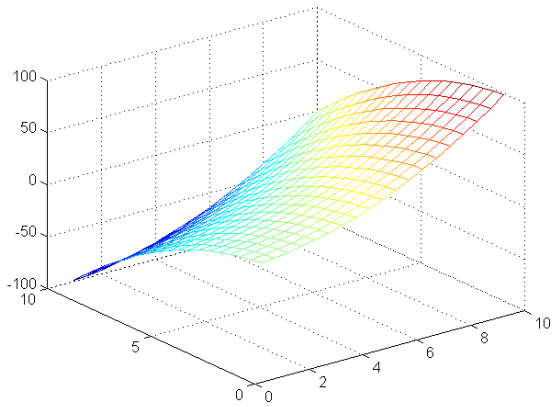
```
» [x, y] = meshgrid([1:.5:10],[1:.5:10]);
```

Then, you can do a 3d plot by calculating some z values on the x and y domains, and doing a surface, mesh, or contour plot.

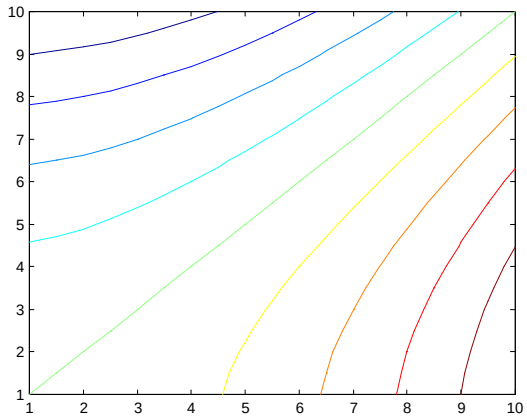
```
» z = x.^2 - y.^2;
» surf(x,y,z)
```




```
» mesh(x,y,z)
```




```
»  contour(x,y,z)
```

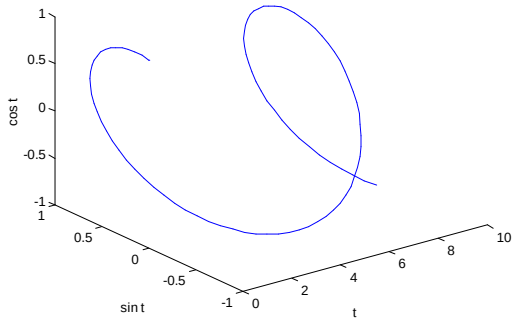


You could also use **surfc** or **meshc** to get surface or mesh plots with a contour plot drawn on the x-y plane. Surface plots and mesh plots color the plots according to the z value, by default.

A 3-D curve can be shown by the **plot3** command:

[Example]

```
»  t = 0:0.1:3.0*pi;
»  plot3(t,sin(t),cos(t))
»  xlabel('t'), ylabel('sin t'), zlabel('cos t')
```



(2) Colormaps and Visibility

The default colormap that Matlab uses is called '**hsv**'. This colormap cycles all the way around from red to red, so both lows and highs will look red. This is often not very useful in engineering applications. You can switch to a colormap that goes from blue to red with

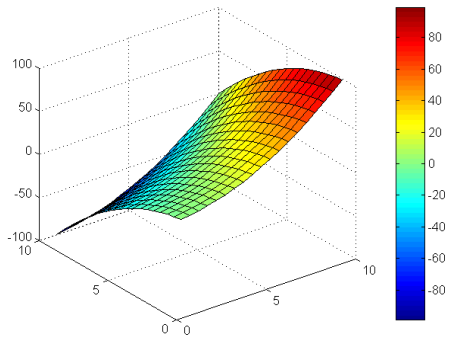
```
» colormap('jet')
```

If you type "**help hsv**", the see also list includes the other predefined colormaps. You can get a colorbar that indicates what colors correspond to what values with

```
» colorbar
```

For example:

```
» surf(x,y,z)
» colorbar
```



Sometimes, a 3d surface plot may cover up portions of its plot. You can take advantage of the fact that Matlab doesn't plot NaN's and Inf's to make "cutouts" in a plot.

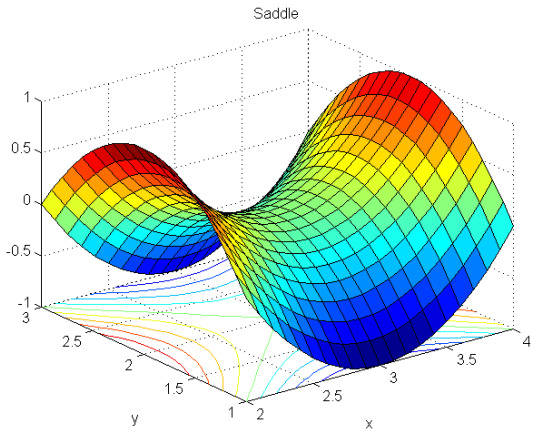
(3) More Examples

[Example] Plot the surface defined by the function

$$f(x,y) = (x-3)^2 - (y-2)^2$$

for $2 \le x \le 4$ and $1 \le y \le 3$.

```
» clear all
» [x,y] = meshgrid(2:0.1:4,1:0.1:3);
» z = ( x - 3 ).^2 - ( y - 2 ).^2;
» surfc(x,y,z)
» title('Saddle'),xlabel('x'),ylabel('y')
```

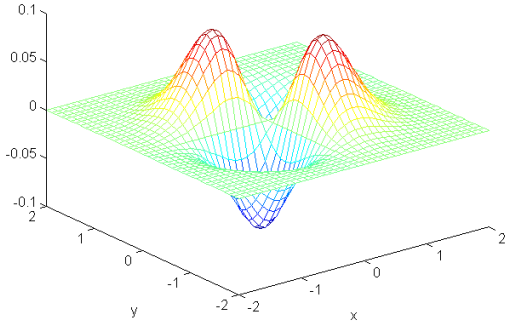


[Example] Plot the surface defined by the function

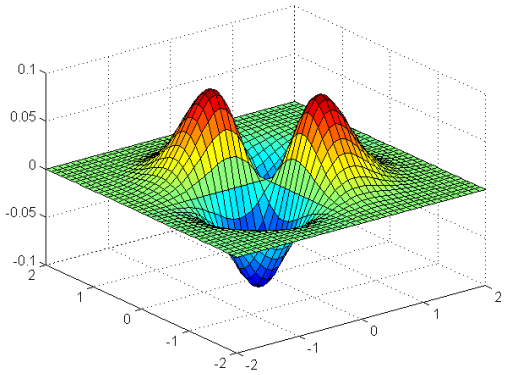
$$f = -x\,y\,e^{-2(x^2+y^2)}$$

on the domain for $-2 \le x \le 2$ and $-2 \le y \le 2$.

```
» clear all
» [x,y] = meshgrid(-2:0.1:2, -2:0.1:2);
» f = - x.*y.*exp(-2*(x.^2+y.^2));
» mesh(x,y,f), xlabel('x'), ylabel('y'), grid
```



```
surf(x,y,f)
```



8. Polynomials

Matlab can treat a vector as a polynomial. It will assume that the numbers represent the coefficients of the polynomial going from highest-order to lowest order. For example,

```
»  p = [ 1 2 2 4 1 ]

p =
     1     2     2     4     1
```

can represent the polynomial $x^4 + 2x^3 + 2x^2 + 4x + 1$. There are a number of functions that use this representation of polynomials:

```
»  roots(p)
```

gives you the roots of the polynomial represented by the vector p.

```
ans =

 -1.90696494950524
  0.09333575522422 + 1.36604798694679i
  0.09333575522422 - 1.36604798694679i
 -0.27970656094320
```

The value of polynomial can be evaluated by the function **polyval**, for example,

```
»  polyval(p,4)
```

gives you the value of the polynomial p when x = 4.

```
ans =

   433
```

Similarly,

```
>>polyval(p,[1:10])
```

gives you the value of the polynomial evaluated at each of the points in the vector. (It returns another vector the same size.)

```
ans =
  Columns 1 through 6
        10          49         166         433         946        1825
  Columns 7 through 10
      3214        5281        8218       12241
```

9. Symbolic Math (Optional)

裝了符號運算工具箱(Symbolic Toolbox)後，Matlab 可以進行符號相關的運算。Matlab is capable of doing fairly simple symbolic math analysis (i.e. giving you symbolic equations as results rather than matrices or vectors.). It is designed for numerical work, rather than symbolic work, and in order to do symbolic math, Matlab actually uses the engine for another math software product, Maple. If you find that what you're trying to do in Matlab 's symbolic math toolkit involves mostly calling the "**maple**" command (which simulates Maple), you probably just want to go ahead and use Maple. For basics, though, the symbolic math toolkit is useful. A symbolic equation in Matlab is represented as a string, such as 'x + 2 = 5'. The basic symbolic math operations all take string equations as arguments, and return string equations. Matlab decides which variable in that string is the symbolic variable (as opposed to a variable that actually has a numerical value in its workspace) by assuming that the one-letter variable closest to the end of the alphabet is the symbolic variable. Basic symbolic functions include

```
»  symadd('x^2 + 2*x + 3','3*x+5')

ans =

x^2+5*x+8
```

symsub, **symdiv**, **symmul**, and **sympow** all work as you would expect, too. Matlab also has symbolic functions such as **int** and **diff** to integrate and differentiate, **factor** to factor, **simplify** to simplify an expression, and **taylor** to generate a Taylor series expansion. For example,

```
»  x = sym('x');
»  factor(x^9-1)
ans =
      (x-1)*(x^2+x+1)*(x^6+x^3+1)
```

You can also use **solve** to solve algebraic equations, and **dsolve** to solve a differential equation. For example,

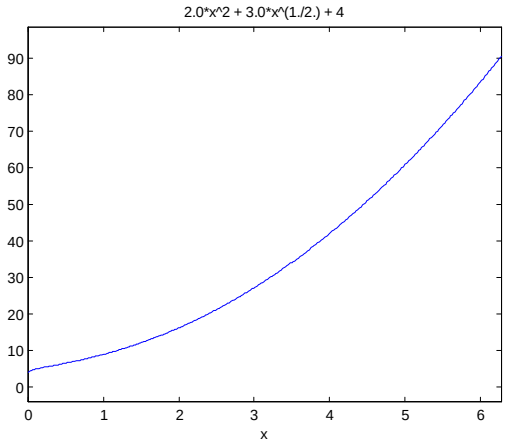
```
»  p = 'a*x^2 + 2*b*x + c'
p =
a*x^2 + 2*b*x + c
» solve(p)
ans =
-(b + (b^2 - a*c)^(1/2))/a
```

```
-(b - (b^2 - a*c)^(1/2))/a
```

The function **`numeric('symbolic expression')`** lets you convert a symbolic representation to numbers wherever possible, which also helps in simplification.

The function **`ezplot('symbolic function')`** takes a symbolic equation and plots it as a function of x. You can specify a range or let it use the default. To specify a range, use **`ezplot('symbolic function',[xmin xmax]')`**.

```
»  p = '2.0*x^2 + 3.0*x^(1./2.) + 4'
p =
2.0*x^2 + 3.0*x^(1./2.) + 4
»  ezplot(p)
```



For more details on the symbolic toolkit, try "**`help sym`**", or try reading the manual.

10 Function Functions (會呼叫函數的函數)

A class of functions in MATLAB works not with numerical matrices, but with mathematical functions. These function functions include:

Numerical integration
Nonlinear equations and optimization
Differential equation solution
. . .

Mathematical functions are represented in MATLAB by function m-files (有關 m-file 詳細建立的方法會在後面詳加說明). For example, the function

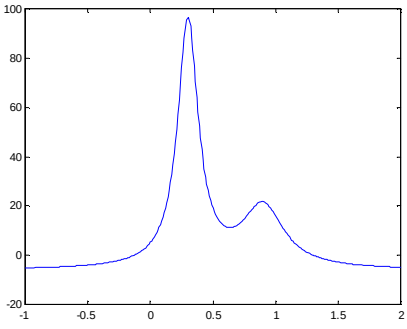
$$humps(x) = \frac{1}{(x-0.3)^2+0.01} + \frac{1}{(x-0.9)^2+0.04} - 6$$

is made available to MATLAB by creating an M-file called humps.m:

```
function result = humps ( x )
%  HUMPS computes a function that has three roots, and some humps.
%
result = 1.0 ./ ( ( x - 0.3 ).^2 + 0.01 ) ...
       + 1.0 ./ ( ( x - 0.9 ).^2 + 0.04 ) - 6.0;
```

如果敘述句太長無法在一句結束，則可用在句末加上 `...` 表示上、下行是接續的。例如上式所示。 A graph of the function is:

```
x = -1 :.01:2;
plot(x,humps(x))
```



(1) Numerical Integration

The area beneath humps(x) can be determined by numerically integrating humps(x), a process referred to as 'quadrature'. To integrate humps from 0 to 1:

```
» q = quad('humps',0,1)
q =
  2.9858e+001
```

The two MATLAB functions for quadrature are: quad, quad8. Notice that the first argument to quad is a quoted string containing the name of a function.

This is why we call quad a function function – it is a function that operates on other functions.

(2) Nonlinear Equations and Optimization

The function functions for nonlinear equations and optimization include: fminbnd, fminsearch, fsolve, fzero. Continuing our example, the location of the minimum of humps(x) in the region from 0.5 to 1 is computed with fminbnd:

```
» xm = fminbnd('humps', 0.5, 1 )
xm =
  6.3701e-001
```

Its value at the minimum, is:

```
» y = humps(xm)
y =
  1.1253e+001
```

From looking at the graph, it is apparent that the function humps(x) has two zeros. The location of the zero near x = 0 is:

```
» xz1 = fzero('humps', 0)
xz1 =
 -1.3162e-001
```

The zero near x = 1 is at:

```
» xz2 = fzero('humps', 1)
xz2 =
  1.2995e+000
```

(3) 微分方程式 Differential Equation Solution (optional)

MATLAB's functions for solving ordinary differential equations are: ode23, ode45. Consider the second order differential equation known as the Van der Pol equation

$$\ddot{x} + (x^2 - 1)\dot{x} + x = 0$$

We can rewrite this as a system of coupled first order differential equations

$$\dot{x}_1 = x_1(1 - x_2^2) - x_2$$
$$\dot{x}_2 = x_1$$

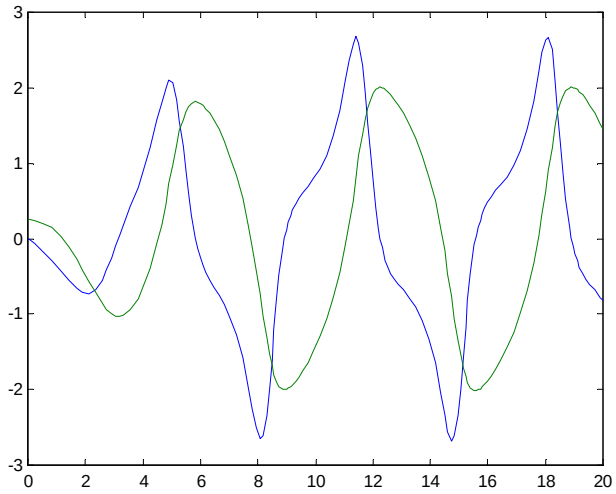
The first step toward simulating this system is to create a function M-file containing these differential equations. We can call it vdpol.m:

```
function xdot = vdpol(t,x)
xdot = zeros(2,1);
```

```
xdot(1) = x(1) .* (1 - x(2).^2) - x(2);
xdot(2) = x(1);
```

To simulate the differential equation defined in vdpol over the interval $0 \le t \le 20$ invoke ode23

```
t0 = 0; tf = 20;
x0 = [ 0 0.25]'; % Initial conditions
[t,x] = ode23('vdpol',[t0 tf],x0);
plot(t,x)
```



11. Programming in Matlab

Matlab 是由 C 語言演化來的，許多語法與 C 極為類似。在這裡我們僅介紹 Matlab 最基本的語法與用法。

(1) Writing Functions and Scripts

MATLAB 程式是以 .m 為副檔名(即所謂 m-files)，而 m-files 依照功能可分為 Script 與 Function 兩類：

Scripts (指令批次檔)

Script 是一堆指令、函式呼叫、程式敘述的集合，它不需要輸入與輸出參數。執行 script 的效果等同於在 command window 中逐行執行 script 中的每行敘述。Script 中宣告的變數也是 workspace 的一部份。Script 通常使用在經常使用、固定不變的工作上。Script 無特定的格式。將執行檔存成 xx.m 檔後，僅需鍵入 xx 就可執行該指令批次檔。

[Example]

將下列敘述利用 Matlab 內建的文書處理器鍵入後，存成 myscript.m 檔：

```
% A Sample Script File
disp('Calculating the Volume of an Ideal Gas.')
R = 8314;     % Gas constant
t = input('Vector of temperatures (K) =');
p = input('Pressure (bar) = ') * 1.0e5;
v = R*t/p ;   % Ideal gas law
% Plotting the results
plot(t,v)
xlabel('T (K) ')
ylabel('V (m^3/kmol)')
title('Ideal Gas Volume vs. Temperature')
```

在指令中若出現 % 符號，那麼該行出現在 % 符號後的部份不會執行 (就像 C 語言中的 /* comments */ 或 //)。將上述字句存成 myscript.m 檔後，在 Matlab 視窗中鍵入：

```
» myscript
```

就可以執行剛剛輸入的批次執行檔；輸入所需的資料後(例如溫度 200:5:400 ；壓力 2.0)，看看 matlab 為你做了哪些事！

Functions

Function 就是所謂的副程式 (sub-program)。Function 中使用輸入及回傳參數與其他 function 或是 command window 進行溝通；Function 之間也可以互相呼叫。Function 中宣告的變數和 C 語言類似，屬於區域

變數，有別於 workspace 與其他 function 中同名的變數。將常用的工作包裝寫成 function 有助於程式的結構化。注意：一個 m-file 只能包含一個 function ，檔名與 function 名稱必須相同。函式的第一句必定是

```
function [list of outputs]  = functionname(list of inputs)
```

這樣 Matlab 才知道這個 m-file 檔案是函數。 Each function needs to have its own file, and the file has to have the same name as the function. If the first line of the function is

```
function answer = myfun(arg1,arg2)
answer = (arg1+arg2)./arg1
```

then the file must be named myfun.m. The function has arg1 and arg2 to work with inside the function (plus anything else you want to define inside it, and possibly some global variables as well), and by the end of the function, anything that is supposed to be returned should have a value assigned to it. This particular function is just one line long, and it returns answer, which is defined in terms of the two arguments arg1 and arg2.

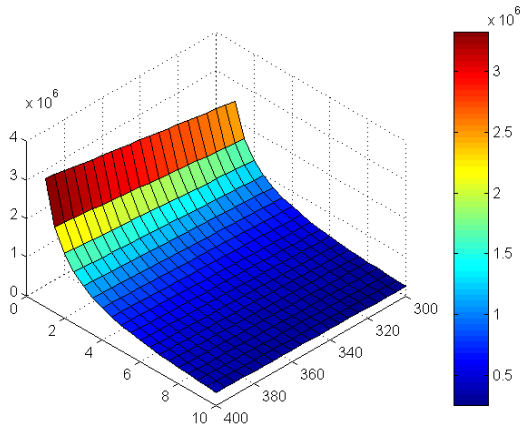
[Example]

Let us write a function to do the ideal gas volume calculations that we have already done in a script:

```
function v = myfunction(t,p)
% Function 'myfunction.m'
% This function calculates the specific volume
% of an ideal gas
R = 8314;               % Gas constant
for k = 1:length(p)
    v(k,:) = R*t/p(k);  % Ideal gas law
end
```

This function must be saved as 'myfunction.m'. Now we can call this function in Matlab:

```
»  p = 1:0.5:10; t = 300:5:400;
»  vol = myfunction(t,p);
»  surf(t,p,vol)
»  colorbar
»  view(135,45)
```



Some useful tools for functions and scripts

- **nargin**

  This variable used within a function tells you how many arguments the function was called with. (nargin stands for number of input arguments)

  You can write functions that can accept different numbers of arguments and decide what to do based on whether it gets one argument or two arguments. For example, the following function calculates the surface area and volume of a sphere or a cylinder depending on the number of input arguments:

```
      function [area, volume] = AV( radius, height )
%
%     AV calculates the surface area and volume of a cylinder
%     or a sphere.  If height is provided in input argument, AV
%     calculates the surface area and volume of a cylinder.
%     Otherwise, AV calculates the surface area and volume of
%     a sphere.
%
      if nargin == 1
         volume = 4.0/3.0*pi*radius^3;
         area   = 4.0*pi*radius^2;
      else
         volume = pi*radius^2*height;
         area   = 2.0*pi*radius*height + 2.0*pi*radius^2;
      end
%
%
```

  Now in Matlab,

```
» [area,volume] = av(4.0)
```

```
area =
    2.010619298297468e+002
volume =
    2.680825731063290e+002
```

where area and volume are the surface area and volume of a sphere of radius R = 4, while

```
» [area,volume] = av(4.0, 1.0)
area =
    1.256637061435917e+002
volume =
  50.26548245743669
```

yield the surface area and volume of a cylinder of radius R = 4.0 and height H = 1.0, respectively.

- **feval**

  The function evaluates a function for a given set of arguments. For example, **feval('sin',[0:pi/4:2*pi])** is the same thing as saying **sin([0:pi/4:2*pi])**. If you're dealing with a situation where you might want to specify which function to use as an argument to another function, you might use **feval**.

(2) Global Variables (全域變數)

When you define a variable at the Matlab prompt, it is defined inside of Matlab's "workspace." Running a script does not affect this, since a script is just a collection of commands, and they're actually run from the same workspace. If you define a variable in a script, it will stay defined in the workspace.

Functions, on the other hand, do not share the same workspace. A function won't know what a variable is unless it gets the variable as an argument, or unless the variable is defined as a variable that is shared by the function and the Matlab workspace, or a global variable. To use a global variable, every place (function, script, or at the Matlab prompt) that needs to share that variable must have a line near the top identifying it as a global variable, i.e.:

```
global phi;
```

Then when the variable is assigned a value in one of those places, it will have a value in all the places that begin with the global statement.

(3) Loops and Control

Sometimes, you do need to use some kind of loop to do what you need, rather than just operating on an entire matrix or vector at once.

while ... end

The syntax for a **while** loop is

```
while (some logical expression)
        do something;
        do something else;
end
```

For Example

```
x = 0;
while x <= 1
     x, y = sin(x)
     x = x + 0.1;
end
```

注意在 Matlab 中需用 end 來代表迴圈的區間。

for ... end

An example for the syntax for a **for** loop is:

```
k = 0;
for x = 0:0.2:1
    k = k + 1
    y(k) = exp(-x)
end
```

You should try to avoid using i and j as counters, since you will wind up redefining i (which is initially defined as the imaginary i.). The loops can be nested and are usually indented for readability, e.g.,

```
m = 5;
n = 5;
for i = 1:m
   for j = 1:n
      A(i,j) = 1/(i+j-1);
   end
end
A
```

Yields

```
A =

  1.0000e+000  5.0000e-001  3.3333e-001  2.5000e-001  2.0000e-001
  5.0000e-001  3.3333e-001  2.5000e-001  2.0000e-001  1.6667e-001
  3.3333e-001  2.5000e-001  2.0000e-001  1.6667e-001  1.4286e-001
  2.5000e-001  2.0000e-001  1.6667e-001  1.4286e-001  1.2500e-001
  2.0000e-001  1.6667e-001  1.4286e-001  1.2500e-001  1.1111e-001
```

if ... (else ...) ... end

The syntax for an **if** statement is

```
if (logical expression)
        matlab command
elseif (other logical expression)
        another matlab command
else
        a matlab command
end
```

(You don't need an **elseif** or an **else**, but you do need an **end**.)

switch case ... otherwise end

Syntax of the switch-case construction is

```
switch expression (scalar or string)
      case value1 (executes if expression evaluates to value1)
         commands
      case value2 (executes if expression evaluates to value2)
         commands
      .
      .
      .
      otherwise
         statements
end
```

In the following example a random integer number x from the set {1, 2, … , 10} is generated. If x = 1 or x = 2, then the message Probability = 20% is displayed to the screen. If x = 3 or 4 or 5, then the message Probability = 30% is displayed, otherwise the message Probability = 50% is generated. The script file fswitch.m utilizes a switch as a tool for handling all cases mentioned above.

```
% Script file fswitch.
% Generate a random integer in {1, 2, ... , 10}
x = ceil(10*rand);
switch x
   case {1,2}
      disp('Probability = 20%');
   case {3,4,5}
      disp('Probability = 30%');
   otherwise
```

```
        disp('Probability = 50%');
    end
```

The MATLAB functions that are used in file fswitch are

rand – uniformly distributed random numbers in the interval (0, 1)
ceil – round towards plus infinity (see Section 2.5 for more details)
disp – display string/array to the screen

Let us test this code ten times:

```
for k = 1:10

fswitch

end

Probability = 50%
Probability = 30%
Probability = 50%
Probability = 30%
Probability = 50%
Probability = 50%
Probability = 30%
Probability = 20%
Probability = 50%
Probability = 30%
```

break and **pause**

You can use **break** command to jump out of a loop before it is completed. The **pause** command will cause the program to wait for a key to be pressed before continuing. For example,

```
k = 0;
for x = 0:0.2:1
    if k > 3
       break
    end
    k = k + 1
    y(k) = exp(-x)
    pause
end
```

In the process of a loop, if you define an element of a vector that doesn't exist yet (the vector is smaller than the element you're trying to assign), Matlab will increase the size of the vector or matrix to allow for the new element to go where you've specified. However, if you know that you're going to be assigning elements until the matrix grows to be some specific size, it's better to "preallocate" the matrix by defining it to be all zeros initially.

```
matrix = zeros(rows, columns);
```

will do that.

(4) Relational and Logical Constructs

The relational operators in MATLAB are

Operator	Description
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
==	equal
~=	not equal.

Note that '=' is used in an assignment statement while '==' is used in a relation. Relations may be connected or quantified by the logical operators:

Operator	Description
&	and
|	or
~	not.

(注意在 C 語言中， and, or 及 not 分別是 &&, || 及！。) When applied to scalars, a relation is actually the scalar 1 or 0 depending on whether the relation is true or false (indeed, throughout this section you should think of 1 as true and 0 as false). For example

```
» 3 < 5
ans =
     1
» a = 3 == 8
a =
     0
```

注意在上式中， 3==8 先運算(答案是非，也就是 0)，然後 a 再等於此值。 When logical operands are applied to matrices of the same size, a relation is a matrix of 0's and 1's giving the value of the relation between corresponding entries. For example:

```
» A = [ 1 2; 3 4 ];
» B = [ 6 7; 8 9 ];
» A == B
ans =
     0     0
     0     0
» A < B
ans =
     1     1
     1     1
```

To see how the other logical operators work, you should also try

```
» ~A
» A&B
» A & ~B
» A | B
» A | ~A
```

(5) [Example: Temperature Conversion Program]

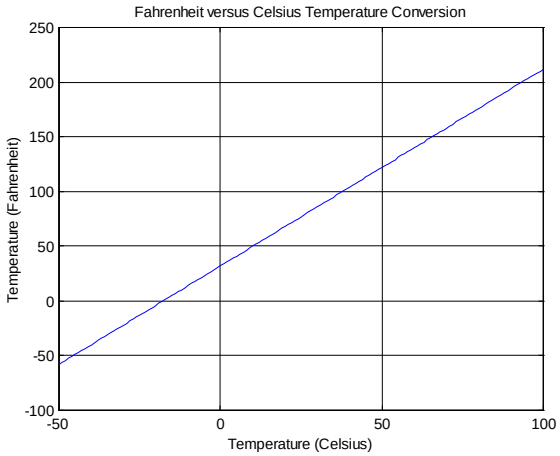
The relationship between temperature measured in Fahrenheit (°F) and temperature Celsius (°C) is given by the equation:

$$T(°F) = 9 * T(°C) / 5 + 32$$

We can now write an m-file that computes and plots the temperature conversion relationship over the range – 50 through 100 °C.

```
% ================================================================
% Temperature Conversion Program for Celsius to Fahrenheit
% covering the range 0 through 100 degrees Celsius.
% Temp_Conv.m
% ================================================================

Num_Points = 151;

% Initialize Temp_C
% Temp_C = zeros(1,Num_Points);

for i = 1 : Num_Points
    Temp_C(i) = i - 51;
    Temp_F(i) = 9*Temp_C(i)/5 + 32;
end

plot(Temp_C, Temp_F);
grid;
xlabel('Temperature (Celsius)');
ylabel('Temperature (Fahrenheit)');
title('Fahrenheit versus Celsius Temperature Conversion');
```

Run the program and we will get the following plot:



[Exercise] 請將上一段 Matlab 批次檔改寫為 C 語言；繪圖部份除外，但需要包含螢幕輸出攝氏與華式溫度部份。

[Exercise] 請將下列 Matlab script 改寫為一個完整的 C 語言程式：

```
%     Calculates exp(x) by Taylor Series Expansion

disp( ' This program calculates exp(x) for various x. ')

   x = input( ' Please input x: ');

   TrueValue = exp(x) ;
   Sum = 1.0;
   Term = 1.0;
   for i = 1: 30000

       Term = Term*x/i;
       Sum1 = Sum + Term;
       if(Sum1 == Sum)
          break
       end
       Sum = Sum1;

   end

RelativeError = ( Sum - TrueValue )/TrueValue;

disp('    i          Sum         TrueValue       RelativeError')
fprintf('%5i %16.7e %16.7e %16.7e \n',i, Sum, ...
        TrueValue, RelativeError )
```

[Exercise] 請將下列 Matlab 程式片斷改寫為 C 語言。

```
%
      for k = 1:n-1
%
%
        ifLag = k;
        amaxi = abs(a(k,k));
        for L = k:n
           if abs(a(L,k)) > amaxi
              amaxi = abs(a(L,k));
              ifLag = L;
           end
        end
%
%
        for i = k+1:n
          xmuLt = a(i,k)/a(k,k);
          for j = k+1:n;
            a(i,j) = a(i,j) - xmuLt*a(k,j);
          end
          b(i) = b(i) - xmuLt*b(k);
        end
      end
%
```

(6) Recursion vs. Iteration– Example of Fibonacci Numbers

下列兩個 Matlab 函數都是計算 Fibonacci Numbers，第一個 **fibonacci(n)** 是一般利用迴圈的方式產生，第二 **fibnum(n)** 則是利用遞迴函數的方法產生。

```
function  f  =  fibonacci(n)
%  FIBONACCI    Fibonacci  sequence
```

```
% f = FIBONACCI(n) generates the first n Fibonacci
numbers.
f = zeros(n,1);
f(1) = 1;
f(2) = 2;
for k = 3 : n
f(k) = f(k-1) + f(k-2);
end


function f = fibnum(n)
% FIBNUM Fibonacci number.
% FIBNUM(n) generates the nth Fibonacci number.
if n <= 1
f = 1;
else
f = fibnum(n-1) + fibnum(n-2);
end
```

先在 Matlab 中建立這兩個檔案後(分別是 **fibonacci.m** 及 **fibnum.m**)，然後鍵入

```
>> fibonacci(12)
ans =
     1
     2
     3
     5
     8
    13
    21
    34
    55
    89
   144
   233
>> fibnum(12)
ans =
   233
```

兩者都得到 233。另外，請鍵入

```
>> tic, fibonacci(24), toc
ans =
          1
          2
          3
          5
          8
         13
         21
         34
         55
         89
        144
        233
        377
        610
        987
       1597
       2584
       4181
       6765
      10946
      17711
      28657
```

```
        46368
        75025
Elapsed time is 0.000148 seconds.
>> tic,  fibnum(24),  toc
ans =
        75025
Elapsed time is 1.002141 seconds.
```

來比較兩者計算的速度，其中 `tic, ... toc` 就是 Matlab 中用來計算程式運算速度的時鐘。看看哪種寫法比較有效率？(Causion: 切勿鍵入 `tic, fibnum(100), toc`)

Description	MATLAB	C
Equal to	==	==
Not equal to	~=	!=
Less than	<	<
Less or equal	<=	<=
Greater than	>	>
Greater or equal	>=	>=
Logical NOT	~	!
Logical AND	&	&&
Logical inclusive OR	!	||
Logical exclusive OR	xor	
Logical equivalent	==	==
Logical not equivalent	~=	!=

Loop	MATLAB	C++

Indexed loop	`for index=matrix`
	`  statements`
	`  end`
		`for (init;test;inc)`
		`  {`
		`  statements`
		`  }`

Pre-test loop	`while test`
	`  statements`
	`  end`
		`while (test) {`
		`  statements`
		`  }`

Post-test loop		`do {`
		`  statements`
		`  } while (test)`

MATLAB	C++

`if l_expression`
`  true group`
`end`
	`if (l_expression)`
	`  {`
	`  true group;`
	`  }`

MATLAB	C++

`if l_expression`
`  true group A`
`else`
`  false group B`
`end`
	`if (l_expression)`
	`  {`
	`  true group A`
	`  }`
	`else`
	`  {`
	`  false group B`
	`  }`

C

`switch (expression)`
`  {`
`  case value 1 :`
`  group 1`
`  break;`
`  case value 2 :`
`  group 2`
`  break;`
`  ⋮`
`  case value n :`
`  group n`
`  break;`
`  default:`
`  default group`
`  break;`
`  }`