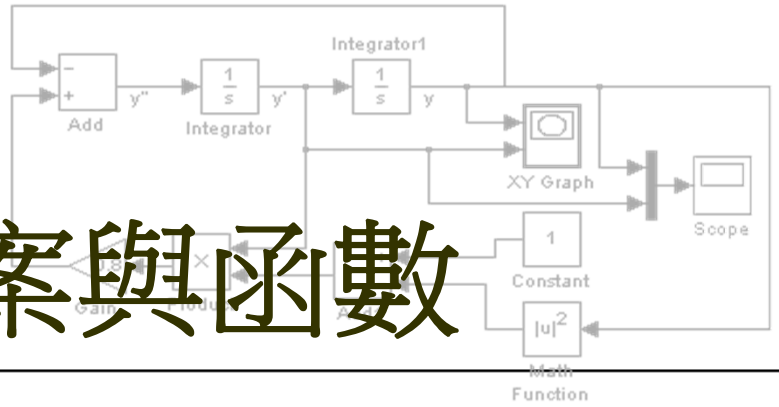


使用M檔案與函數

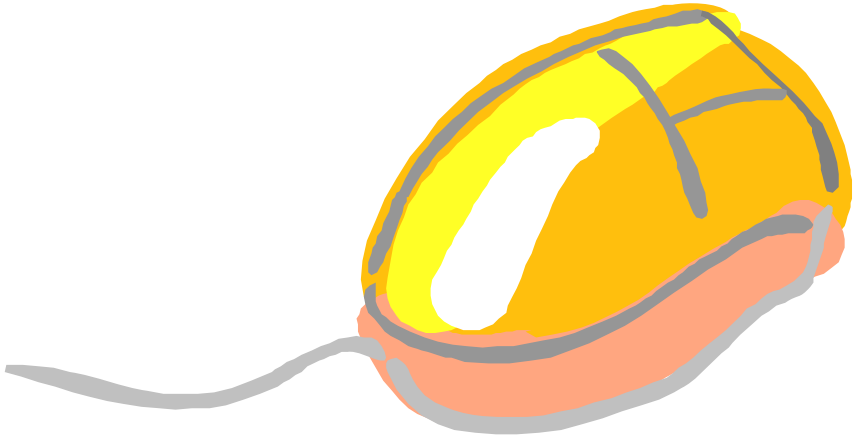
認識M檔案

撰寫底稿與函數

偵錯的技巧

使用全域變數

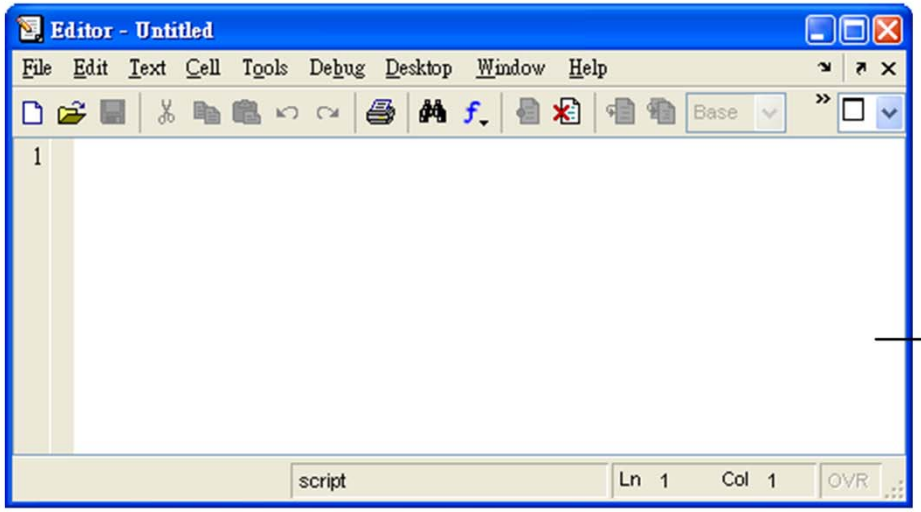
Matlab搜尋M檔案的方式



撰寫底稿

- 底稿（script）是由一系列Matlab的敘述所組成
- 底稿可方便編輯、除錯與執行程式碼
- 要開啟M檔案編輯器，可在指令視窗裡鍵入

```
>> edit
```



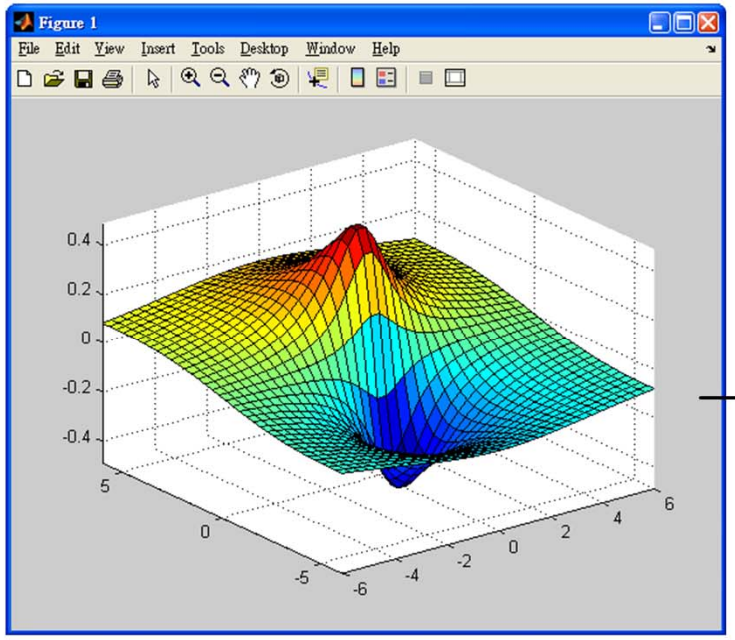
- 下面的程式碼鍵是簡單的底稿：

```
% script7_1.m, 底稿練習-繪出三維函數圖
clear
x=linspace(-6,6,36);
y=linspace(-6,6,36);
[xx yy]=meshgrid(x,y);
zz=yy./(xx.^2+yy.^2+1);
surf(xx,yy,zz); axis tight
```

- 鍵入並儲存好了之後，鍵入底稿的名稱

```
>> script7_1
```

即可執行這個底稿：

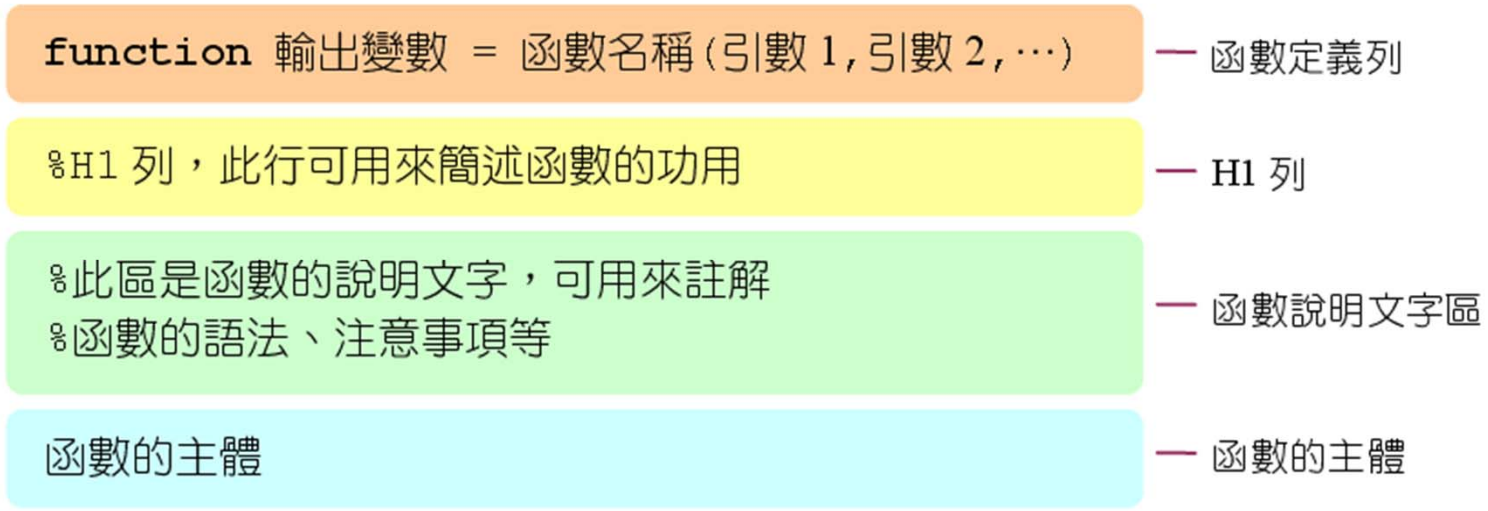


執行底稿 script7_1，即可繪出 $f(x,y)=y/(x^2+y^2+1)$ 的三維函數圖

設計函數

- 函數（function）也是M檔案的一種
- 函數與底稿不同之處：
  1. 函數可傳入引數，也可以把運算結果傳回工作區，而底稿不行
  2. 在函數內使用的變數是區域變數，底稿是全域

7.2.1 函數的基本架構


function 輸出變數 = 函數名稱(引數 1,引數 2,…)
函數定義列
%H1 列，此行可用來簡述函數的功用
H1 列
%此區是函數的說明文字，可用來註解
%函數的語法、注意事項等
函數說明文字區
函數的主體
函數的主體

```
>> type linspace.m
```

```
function y = linspace(d1, d2, n)        ——— 函數定義列
%LINSPACE Linearly spaced vector.       ——— H1 列
%   LINSPACE(X1, X2) generates a row vector of 100 linearly
%   equally spaced points between X1 and X2.
%
%   LINSPACE(X1, X2, N) generates N points between X1 and X2.
%   For N < 2, LINSPACE returns X2.
%
%   Class support for inputs X1,X2:
%      float: double, single
%
%   See also LOGSPACE, :.
```

函數說明文字區

```
%   Copyright 1984-2004 The MathWorks, Inc.
%   $Revision: 5.12.4.1 $  $Date: 2004/07/05 17:01:20 $

if nargin == 2
    n = 100;
end

n = double(n);
y = [d1+(0:n-2)*(d2-d1)/(floor(n)-1) d2];
```

函數的主體

簡單的範例

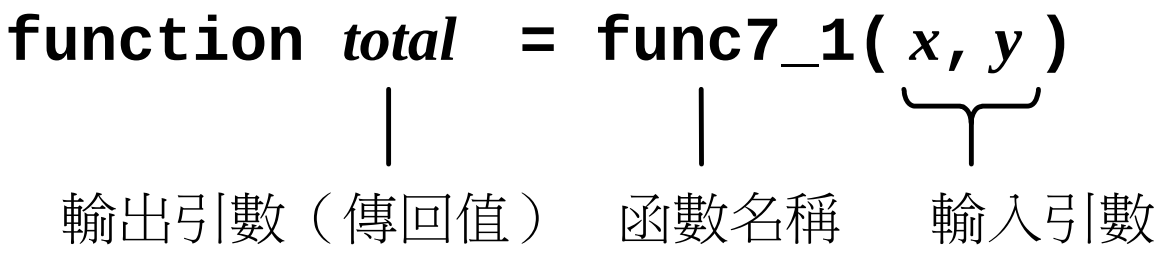
- 函數func7_1可接收兩個引數，並傳回其加總：

```
function total=func7_1(x,y)
%FUNC7_1 sum of two numbers or vectors.
%FUNC7_1(X,Y) computes X+Y and returns the result.
%X and Y can be scalars or vectors.
%function's body starts here
total=x+y;
```

```
>> func7_1(3,5)
ans =
     8
>> help func7_1
  FUNC7_1 sum of two numbers or vectors.
  FUNC7_1(X,Y) computes X+Y and returns the result.
  X and Y can be scalars or vectors.
```

函數的引數與傳回值

- 從工作區接收的引數稱為輸入引數
- 輸出到工作區的引數稱為輸出引數，或稱為傳回值



- 下表列出了幾種情況下，函數定義列的寫法：

表 7.2.2　函數定義列的幾種範例

函數定義列的格式	說　明
`function [x,y]=myfun(a)`	有一個輸入引數 a，有兩個輸出引數 x 與 y
`function [x]=myfun(a)` `function x=myfun(a)`	有一個輸入引數 a，有一個輸出引數 x
`function [x,y]=myfun()` `function [x,y]=myfun`	沒有輸入引數，但有兩個輸出引數 x 與 y
`function []=myfun(a)` `function myfun(a)`	沒有輸出引數，但有一個輸入引數 a

- 有兩個傳回值的函數：

```
function [mn,mx]=func7_2(v)
mn=min(v);
mx=max(v);
```

```
>> [x,y]=func7_2([8 7 3 9 1])

x =
     1
y =
     9
```

- 不需傳入引數的函數

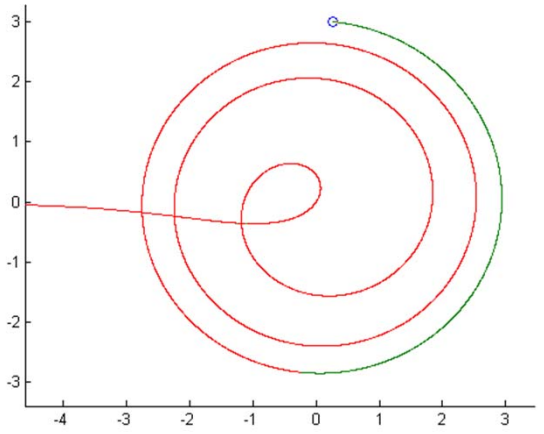
```
function num=func7_3()
num=length(primes(100));
```

```
>> func7_3()
ans =
    25

>> func7_3
ans =
    25
```

- 沒有傳回值的函數

```
function func7_4(n)
t=linspace(0.01,10*pi,n);
r=log(t);
comet(r.*cos(t),r.*sin(t));
```

```
>> func7_4(10000)
```



追蹤函數的執行與偵錯
在函數執行時列印訊息

- 要格式化的列印出某些訊息，可用fprintf函數：

表 7.3.1　格式化列印函數 fprintf 的語法

函數定義列的格式	說　明
`fprintf('str',e1,e2,…)`	依格式字串 *str* 所記載的格式碼，依序將運算式 e_1, e_2 填入 *str* 中列印出來。下面列出了格式字串裡常用的格式碼： %c：列印字元 %s：列印字串 %*m*d：　以 *m* 個欄位的寬度列印整數 %*m*.*n*f：以 *n* 個小數位數，總共 *m* 個欄位的寬度列印數值 %*m*.*n*e：同上，但以指數型式來列印數值

- fprintf函數所使用的特殊字元：

表 7.3.2　用於 fprintf 函數裡的特殊字元

特殊字元	說 明
\n	換行
\t	跳格
''	印出單引號
\\	印出反斜線
%%	印出百分比符號

```
function func7_5(n)
if mod(n,2)==0
    fprintf('%d is even\n',n);
else
    fprintf('%d is odd\n',n);
end
```

```
>> func7_5(14)
14 is even

>> func7_5(63)
63 is odd
```

Matlab的M檔案偵錯環境

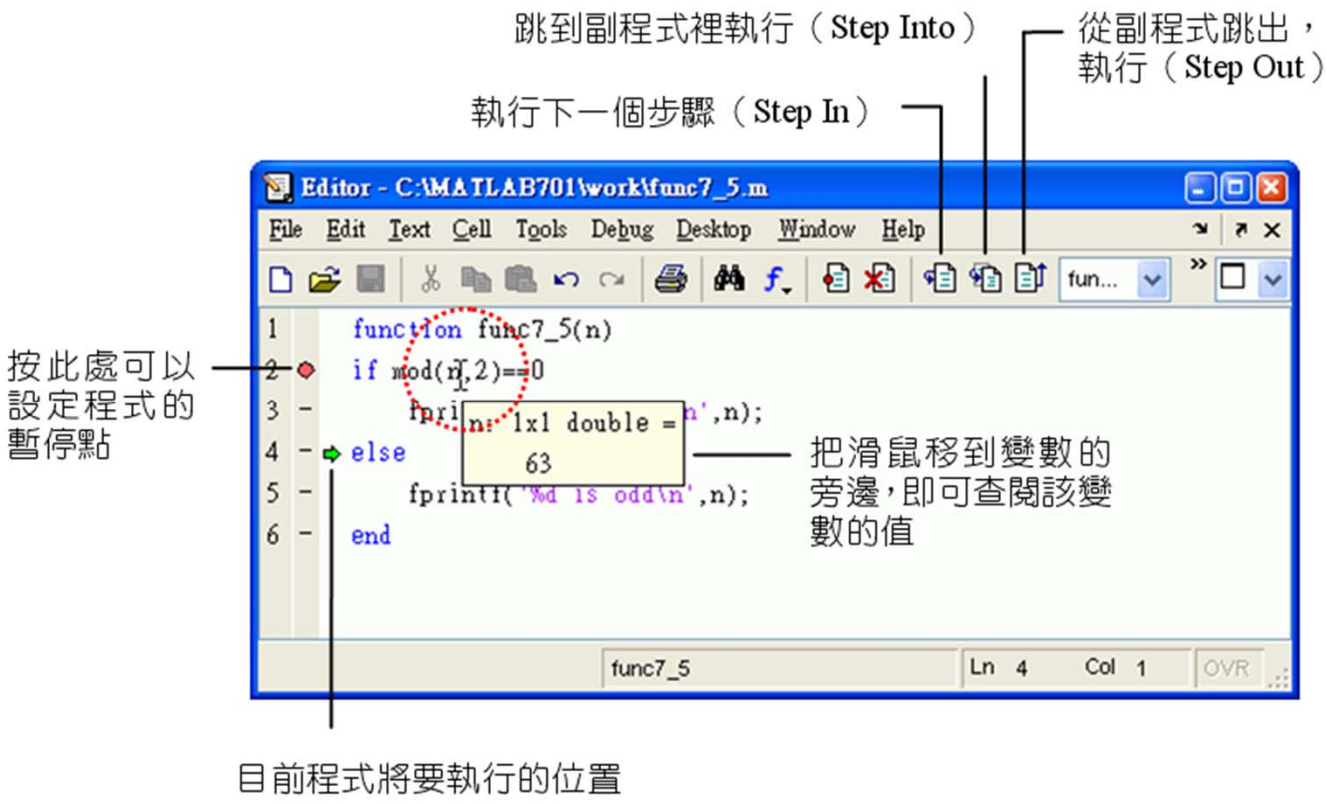


圖 7.3.1

使用 M 檔案編輯器的偵錯功能

函數的進階認識
指令類型的函數

- 如果函數my_func(a, b) 沒有任何的輸出引數：

  1. 可用一般呼叫函數的方式來呼叫：

     ```
     my_func(a,b);
     ```

  2. 也可以採用類似指令的方式來呼叫它：

     ```
     my_func a b ;
     ```

- 沒有輸出引數的函數也稱為 指令類型的函數

```
function func7_6(str)
fprintf('You input ''%s''.\n',str)
```

```
>> func7_6('sss')
You input 'sss'.

>> func7_6 'sss'
You input 'sss'.

>> func7_6 sss
You input 'sss'.
```

函數輸入引數與輸出的個數

- 函數的輸入與輸出引數的個數可以不同：

```
plot(y)                  % 只有一個輸入引數
plot(x,y)                % 有兩個輸入引數
plot(x1,y1,x2,y2)        % 有四個輸入引數
zz=peaks;                % 不需輸入引數
[xx,yy,zz]=peaks(n)      % 有一個輸入引數
```

- nargin與nargout二個變數，可查詢有幾個引數傳進來與傳出去：

表 7.4.1　nargin 與 nargout 變數

變數名稱	說 明
`nargin`	函數裡輸入引數的個數
`nargout`	函數裡傳回值的個數

```
function [x1,x2,x3]=func7_7(a1,a2)
fprintf('nargin = %d, ',nargin)
fprintf('nargout= %d\n',nargout)
x1=a1+a2;
x2=a1-a2;
x3=(a1+a2)/2;
```

```
>> [x,y,z]=func7_7(6,12)
nargin = 2, nargout= 3
x =
    18
y =
    -6
z =
     9

>> total=func7_7(6,12)
nargin = 2, nargout= 1
total =
    18
```

函數內變數的等級

- 在使用全域變數之前，必須利用global關鍵字宣告：

表 7.4.2　使用全域變數

語 法	說 明
`global` var_1 var_2 …	宣告全域變數 $var_1, var_2, \cdots$
`whos global`	查詢工作區內的全域變數
`clear global` var_1 var_2 …	刪除全域變數 $var_1, var_2, \cdots$

```
function func7_10(num)
global VAR;
VAR=VAR+num;
fprintf('在函數內，VAR=%g\n',VAR);
```

```
>> global VAR; VAR=10;
>> func7_10(5)
在函數內，VAR=15

>> VAR
VAR =
    15
```

子函數與私有化目錄

- 同一個M檔案裡可以撰寫多個函數
- 一個M檔案只能有一個主函數，但可以有多個子函數
- 主函數可以呼叫子函數，子函數之間也可以相互呼叫
- 撰寫在M檔案的子函數，只能被同一個檔案內的函數呼叫。

```
function func7_11(v)  % 主函數func7_11
subf(v);
fprintf('End of main function\n');
function subf(n)       % 子函數subf
fprintf('sum(n)=%g\n',sum(n))
fprintf('prod(n)=%g\n',prod(n))
```

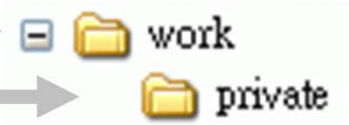
```
>> func7_11([1 2 3 4 5])
sum(n)=15
prod(n)=120
End of main function

>> subf([1 2 3 4 5])
??? Undefined command/function 'subf'.
```

- 私有化目錄：

  可讓上層函數呼叫存放在private子資料夾裡的子函數

```
function func7_12(v) %主函數 func7_12
subf(v);
fprintf('End of func7_12\n')
```

```
function subf(n)    % 子函數 subf
fprintf('sum(n)=%g\n',sum(n))
fprintf('prod(n)=%g\n',prod(n))
```



- 在一個M檔案裡呼叫其它的函數時，Matlab呼叫的次序依序為

1. 同一個M檔案內的子函數
2. 若子函數不存在，則呼叫私有化目錄內的子函數
3. 若私有化目錄內的子函數也不存在，則依搜尋路徑來找尋

保護程式碼—pcode

- pcode可保護程式碼，不讓外人查看。
- M檔案轉換成pcode之後，結果會是亂碼，但是還是可以執行，其執行方式與M檔案完全相同。

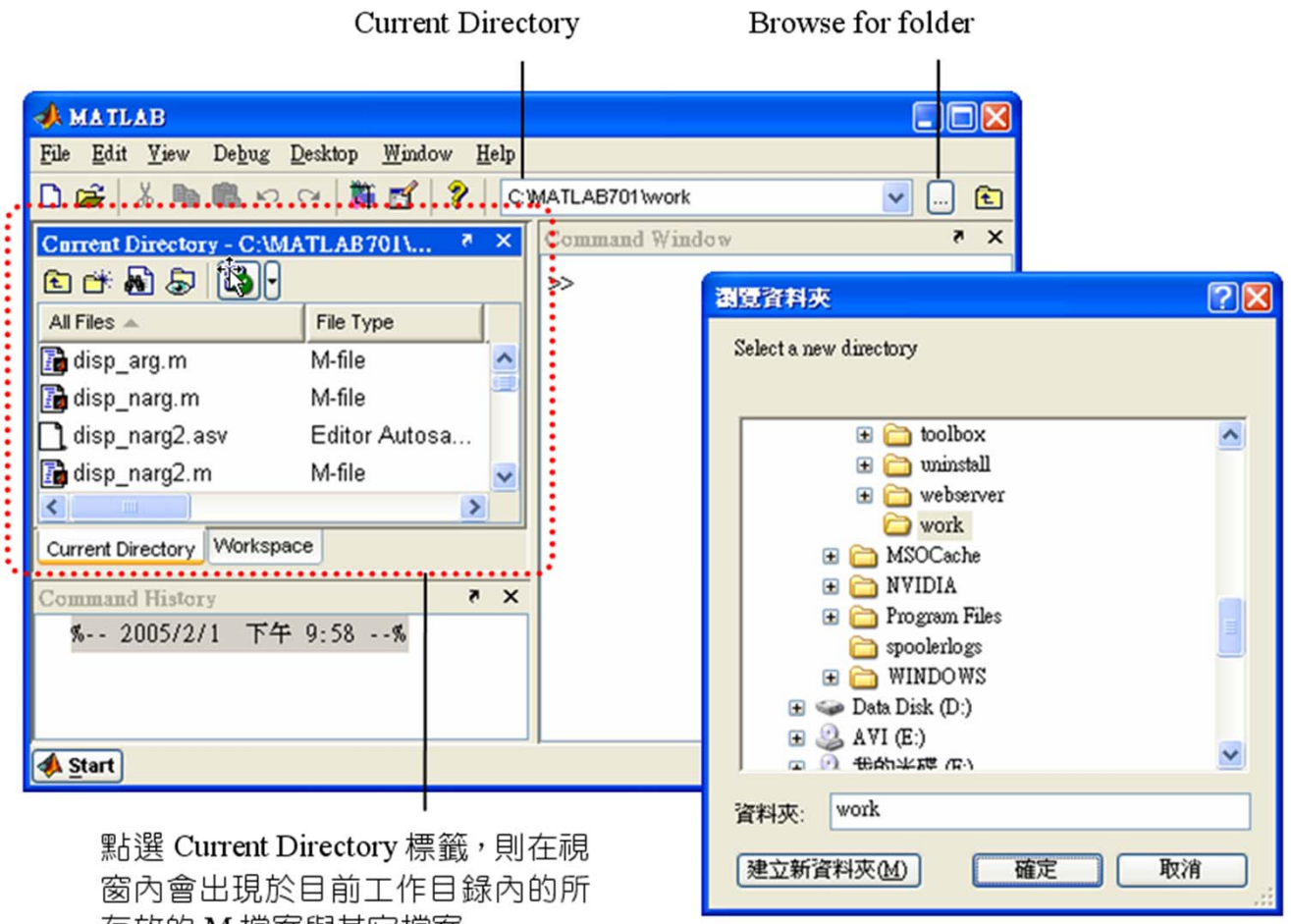
表 7.4.3　使用 pcode 函數

語 法	說 明
`pcode` *file_name.m*	將 M 檔案轉換成 pcode

```
>> pcode func7_4.m      % 將func7_4.m 轉換成pcode
```

路徑的設定

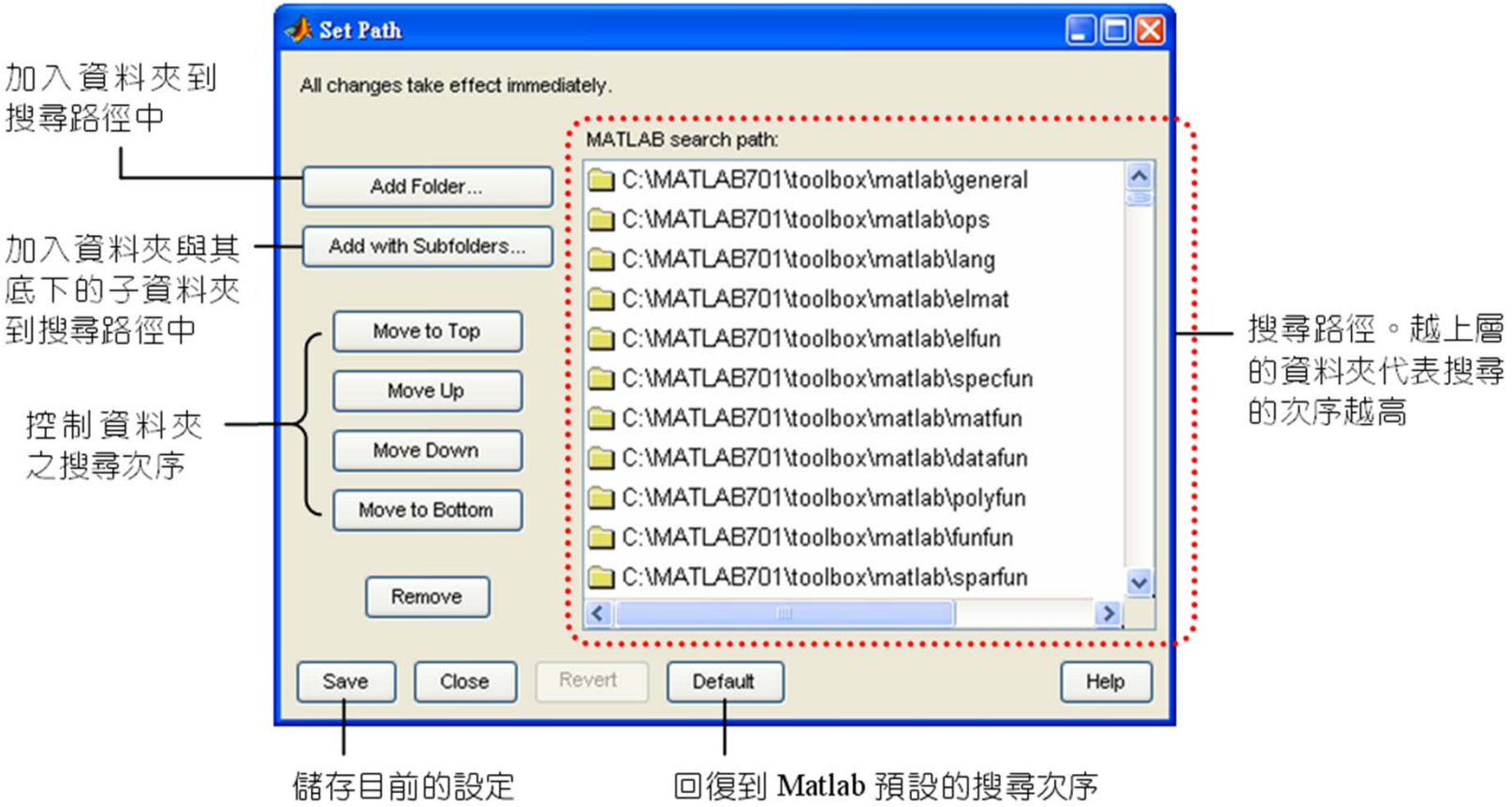
設定目前工作目錄



點選 Current Directory 標籤，則在視窗內會出現於目前工作目錄內的所存放的 M 檔案與其它檔案

設定Matlab搜尋的路徑

- Set Path對話方塊：



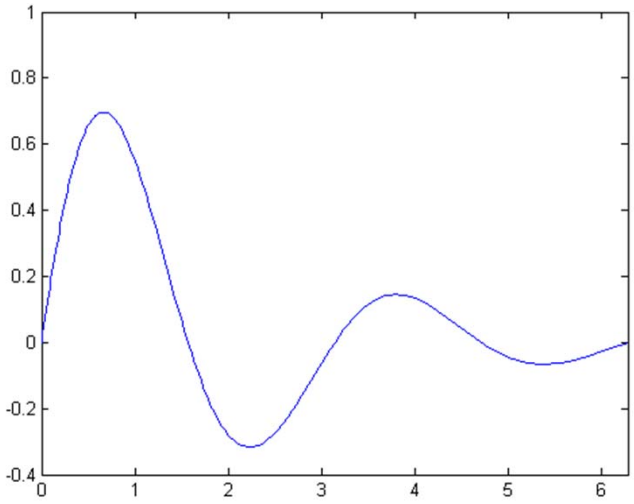
匿名函數

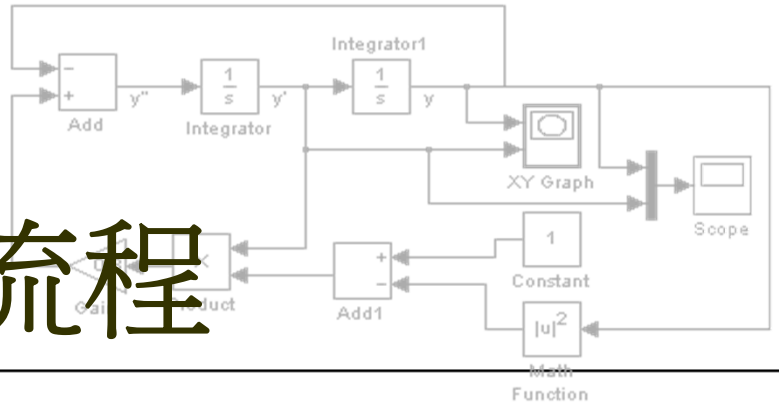
- 匿名函數（anonymous functions）可以在Matlab的指令視窗裡直接定義一個函數，而不用把函數寫在M檔案裡：

表 7.6.1　匿名函數的定義

指 令	說 明
fname=@(*arg_list*) *expr*	定義匿名函數，函數名稱為 *fname*，輸入引數為 *arg_list*，函數的內容則定義在 *expr* 的位置

```
>> f=@(x) sin(2*x).*exp(-x/2)
f =
    @(x) sin(2*x).*exp(-x/2)

>> fplot(f,[0,2*pi])

>> f(2.3)
ans =
    0.0748
```





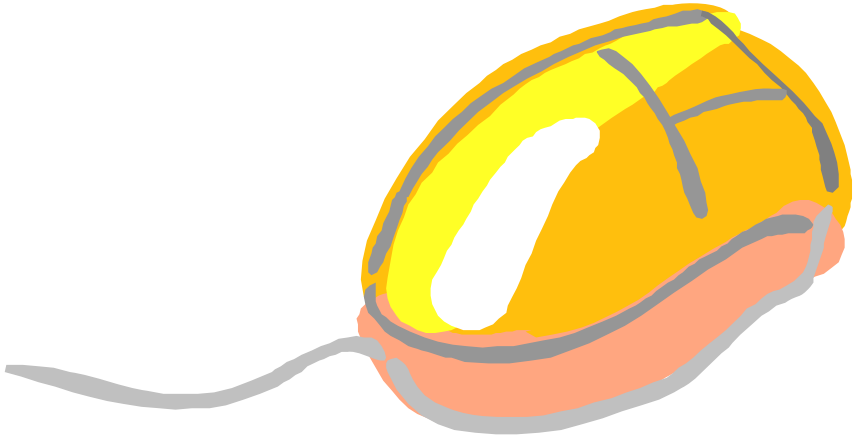
程式控制流程

關係運算子與邏輯運算子

選擇性敘述的用法

各種迴圈的用法

迴圈的效率問題



關係運算子與邏輯運算子

關係運算子

- 關係運算子（relational operators）可用來比較二個數字之間的大小：

表 8.1.1　關係運算子

關係運算子	說 明
<	小於
<=	小於或等於
>	大於
>=	大於或等於
==	等於
~=	不等於

- 簡單的範例：

```
>> 4<=12
ans =
     1

>> test=[1 3 4]<[2 2 1]
test =
     1     0     0

>> a=[-1 0 2 3 1 5]
a =
    -1     0     2     3     1     5

>> b=a<2
b =
     1     1     0     0     1     0
```

- 如果希望判斷結果為陣列的索引值，可用find函數：

表 8.1.2　find 函數的使用

函 數	說 明
ind=find(*array*)	找出陣列 *array* 中，元素值不是 0 之元素的一維索引值，並把此索引值設定給變數 *ind* 存放
[*r*,*c*]=find(*array*)	同上，但回應陣列的二維索引值，並把列索引值設定給變數 *r* 存放，把行索引值設定給變數 *c* 存放

```
>> find([0 2 0;0 1 0;0 0 0])
ans =
     4
     5
```

邏輯運算子

- 邏輯運算子是用來運算2個邏輯變數之間的關係：

表 8.1.3　邏輯運算子

邏輯運算子	說 明
&	and 運算。兩個運算元必須同為 true，其結果才為 true
|	or 運算。兩個運算元中，只要有一個為 true，其結果便為 true
~	not 運算，也就是把 true 變為 false，或把 false 變為 true

```
>> 4&0
ans =
     0

>> [1 0 3] | [2 0 0]
ans =
     1     0     1
```

- all與any可用來判斷陣列裡是否有非零的元素：

表 8.1.4　any 與 all 函數

函 數	說 明
`all(v)`	當向量 *v* 裡所有的元素皆為 true（非零）時，則傳回 1，否則傳回 0
`any(v)`	只要向量 *v* 裡有一個元素為 true（非零）時，則傳回 1，否則傳回 0

```
>> any([1 2 0 0 0])
ans =
     1

>> all([1 2 0 0 0])
ans =
     0
```

性質測試函數

- 性質測試函數可測試其引數是否符合某些性質：

表 8.1.5　性質測試函數

函 數	說 明
`ischar(a)`	測試引數 *a* 是否為一個字元陣列
`isempty(a)`	測試引數 *a* 是否為一個空陣列
`isequal(a,b)`	測試引數 *a* 與 *b* 裡的元素個數與數值是否均相等
`isfloat(a)`	測試引數 *a* 是否為一個浮點數陣列（包含虛數）
`isinteger(a)`	測試引數 *a* 是否為一個 n-bit 整數陣列
`islogical(a)`	測試引數 *a* 是否為一個邏輯型態的陣列
`isnan(a)`	測試引數 *a* 是否為一個 NaN（not a number）的陣列
`isnumeric(a)`	測試引數 *a* 是否為數值（包含 n-bit 整數、實數與虛數）
`isprime(a)`	測試引數 *a* 是否為質數
`isreal(a)`	測試引數 *a* 是否為實數（含邏輯型態，但不包含 n-bit 整數）

表 8.1.5　性質測試函數

函 數	說 明
`isscalar(a)`	測試引數 *a* 是否為純量（scalars）
`issorted(a)`	測試引數 *a* 是否為已經排序好
`isspace(a)`	測試字元陣列 *a* 裡是否有包含空白字元
`isvector(a)`	測試引數 *a* 是否為一個向量

```
>> isempty([])
ans =
     1
>> isequal([1 2 3 4],[1 2 3 0])
ans =
     0
>> isfloat(5)
ans =
     1
```

選擇性敘述
使用if-elseif-else指令

- 當程式中有分歧的判斷敘述時，便可使用if-elseif-else指令來處理：

表 8.2.1　if 與 if-else 指令

指令	說 明
`if` 判斷條件 　　敘述主體 `end`	若判斷條件為 true，則執行敘述主體

`if` 判斷條件
　　敘述主體 1
`else`
　　敘述主體 2
`end`
	若判斷條件為 true，則執行敘述主體 1，否則執行敘述主體 2

```
%script8_1.m
if 5>3
    disp('5大於3')
end
```

```
>> script8_1
5大於3
```

```
function func8_1(num)
if mod(num,2)==0
    fprintf('%g是偶數\n',num)
else
    fprintf('%g是奇數\n',num)
end
```

```
>> func8_1(12)
12是偶數
```

- 如果判斷敘述裡需要有多個一連串的判斷，可使用if-elseif-else指令：

表 8.2.2　if-elseif-else 敘述

指 令	說 明

`if` 判斷條件 1
　　敘述主體 1
`elseif` 判斷條件 2
　　敘述主體 2
`elseif` 判斷條件 3
　　敘述主體 3
　　`...`
`else`
　　敘述主體 *n*
`end`
	if-elseif-else 敘述。若所有的判斷條件皆不成立，則執行主體 *n*

```
function func8_3(num)
if isinteger(num)
   disp('傳入的引數是n-bit整數')
elseif islogical(num)
   disp('傳入的引數是邏輯型態')
elseif isfloat(num)
   disp('傳入的引數是浮點數')
else
   disp('傳入的引數是其它型態')
end
```

```
>> func8_3(logical(1))
傳入的引數是邏輯型態

>> func8_3(uint8(1))
傳入的引數是n-bit整數
```

使用switch-case-otherwise指令

- switch可依據某個運算式的值，來決定是哪個敘述主體會被執行：

表 8.2.3　switch-case-otherwise 敘述

指 令	說 明

`switch` 運算式
　`case` 選擇值 1
　　敘述主體 1
　`case` 選擇值 2
　　敘述主體 2
　　`...`
　`otherwise`
　　敘述主體 *n*
`end`
	若接在 case 後面的選擇值不只一個時，可用大括號將它們括起來，如
	{選擇值 1, 選擇值 2, ..., 選擇值 *n*}

```
function func8_4(method)
switch method
   case {'linear','bilinear'}
      disp('linear/bilinear method')
   case 'cubic'
      disp('Cubic method')
   otherwise
      disp('Unknown method')
end
```

```
>> func8_4('bilinear')
linear/bilinear method

>> func8_4('newton')
Unknown method
```

迴圈
使用for迴圈

- For 迴圈的語法如下所示：

表 8.3.1　for 迴圈敘述

指 令	說 明
`for` 迴圈變數=向量 　敘述主體 `end`	將變數依序設定成向量裡的每一個元素值，然後執行敘述主體
`for` 迴圈變數=矩陣 　敘述主體 `end`	將變數依序設定成矩陣裡的每一個直行，然後執行敘述主體

- 下列的範例是利用for迴圈計算2~100之間，所有質數的總和：

```
%script8_2.m
total=0;
for num=2:100
    if isprime(num)
        total=total+num;
    end
End
fprintf('sum=%d\n',total)
```

```
>> script8_2
sum=1060
```

- 設定

  for 變數 i = 矩陣A

  則變數 I 會依序被設定成矩陣A裡的每一行元素，然後進到迴圈裡執行：

```
%script8_4
for i=[1 2;3 4]
    i    %於指令視窗內印出變數i的值
end
```

```
>> script8_4
i =
     1
     3
i =
     2
     4
```

使用while迴圈

- 無法事先知道迴圈該執行多少次時，可以考慮使用while迴圈：

表 8.3.2　while 迴圈敘述

指 令	說 明
`while` 判斷條件 　敘述主體 `end`	當判斷條件為 true 時，則執行敘述主體，直到判斷條件為 false 為止

- 下面的範例是利用while迴圈，找出所有小於100之質數的總和：

```
%script8_5.m
total=0;
num=2;
while num<=100
    if isprime(num)
        total=total+num;
    end
    num=num+1;
end
fprintf('sum=%d\n',total)
```

```
>> script8_5
sum=1060
```

使用break 與continue指令

- 要中斷迴圈的執行，可利用break與continue指令：

表 8.3.3　break 與 continue 指令（以 for 迴圈為例）

指 令	說 明
for 迴圈變數=向量 　敘述主體 1 　break 　敘述主體 2 end for 迴圈之後的敘述	立即離開迴圈，繼續執行迴圈外的下一個敘述
for 迴圈變數=向量 　敘述主體 1 　continue 　敘述主體 2 end for 迴圈之後的敘述	立即停止執行剩餘的迴圈主體，回到迴圈的開始處繼續執行

- 下面的範例是利用while配合break指令來找出大於1000的最小質數：

```
%script8_7.m
num=1000;
while 1
    if isprime(num)
        break
    else
        num=num+1;
    end
end
fprintf('大於1000的最小質數為%3d\n',num)
```

```
>> script8_7
大於1000的最小質數為1009
```

- 下面的範例是利用continue敘述，印出小於20，但不是質數的正整數：

```
%script8_8.m
for num=1:20
    if isprime(num)
        continue
    end
    fprintf('%3d',num)
end
fprintf('\n')
```

```
>> script8_8
 1  4  6  8  9 10 12 14 15 16 18 20
```

迴圈執行效率的探討
計時指令

- Matlab以tic和toc做為程式執行時間的計時指令：

表 8.4.1　tic 與 toc 指令

指 令	說 明
tic 　程式敘述 toc	tic 可啟動計時器，toc 則是停止計時器的執行，並顯示執行"程式敘述" 所需的時間

```
>> tic,length(primes(2^24)),toc
ans =
     1077871
Elapsed time is 1.656000 seconds.
```

將迴圈向量化

- 將迴圈向量化，可以讓程式碼更為簡潔，也能提昇程式執行的效能。

迴圈型式的M檔案：

$$\sum_{n=1}^{100} \frac{1}{n^2+1}$$

```
%script8_9.m
total=0;
for  n=1:100
    total=total+1/(n^2+1);
end
total   % 顯示執行的結果
```

向量化：

```
>> n=1:100; sum(1./(n.^2+1))
```

- 下面是分別利用迴圈與向量化的程式碼，來分析

$$\sum_i \frac{\sin(i)}{(\log_2 i)^{\log(i)}}$$ 的執行效率：

利用迴圈來計算：

```
%script8_10.m
tic
total=0;
for i=linspace(1,2*pi,10^6)
    total=total+sin(i)/log2(i)^log(i);
end
toc
```

```
>> script8_10
Elapsed time is 4.063000 seconds.
```

將迴圈向量化：

```
%script8_11.m
tic
i=linspace(1,2*pi,10^6);
sum(sin(i)./log2(i).^log(i));
toc
```

```
>> script8_11
ans =
  1.9158e+005
Elapsed time is 1.765000 seconds.
```

** 與迴圈指令相比，速度提昇了2.3倍 **

預先配置記憶空間給陣列

- 預先配置記憶空間給陣列，可有效的提升執行速度

沒有預先配置記憶空間：

```
%script8_13.m
tic
clear a;
for i=1:50000
    a(i)=sin(i)+cos(i);
end
toc
```

```
>> script8_13
Elapsed time is 5.026017 seconds.
```

預先配置記憶空間：

```
%script8_14.m
tic
a=zeros(1,50000); %預先配置記憶空間給陣列a
for i=1:50000
    a(i)=sin(i)+cos(i);
end
toc
```

```
>> script8_14
Elapsed time is 0.023829 seconds.
```

** 得到0.023829秒。與前例相比，可發現執行的效能有顯著的提昇 **