A Short Octave Tutorial

Jenn-Hong Tang
Department of Economics
National Tsing-Hua University

October 1, 2013

Octave 語法 I

Jenn-Hong Tang
Department of Economics
National Tsing-Hua University

September 19, 2011

安裝

- Download GNU Octave: http://wiki.octave.org/Octave_for_Windows

線上查詢

- `help`: 指令查詢。例如:

```
help gamma
```

- `lookfor`: 關鍵字查詢。例如:

```
lookfor gamma
```

- 線上求助的內容太長時會分頁顯示, 按 space 可以翻頁, 閱讀完畢按 q 可以回到原畫面。
- 如果不想分頁顯示功能, 可以指令 more off 關閉, 若要恢復可下指令 more on。

基本運算

本章的目的在於介紹 Octave 的基本語法, 較深入的語法將留待未來以邊做邊學的方式陸續介紹。 Octave 的基本運算指令如表 1 所示:

Table : 基本運算指令

加	減	乘	(右)除	左除	冪次
+	-	*	/	\	** 或 ^

左除是 MATLAB 比較特別的發明, $a\backslash b$ 等於 $a^{-1}b$。

```
octave:1> 1/2
ans =  0.50000
octave:2> 1\2
ans =  2
```

- 小數點是以英文句點 . 表示, 小數點後末位的 0 可省略, 故 2 與 2. 及 2.0 相同。
- 小括弧 () 表示優先計算的部份, 如果有多層小括弧, 則優先順序由裡而外, 由左而右, 先取冪次, 再先乘除後加減。
- 用來表示優先計算順序的只有小括弧, 沒有中括弧或大括弧 (這兩者另有重用)。

```
octave:6> 2 * (2 + (3-1))
ans =  8
```

答案等於 2 * (2 + 2), 再等於 2 * 4。

```
octave:7> 3*2^3
ans =  24
octave:8> 8/2*3
ans =  12
octave:9> 8/2/2-1
ans =  1
```

- 運算元前後留多少空白並不影響結果, 所以下列兩式的結果相等:

```
octave:10> 2 + 3 * 4
ans =  14
octave:11> 2+3*4
ans =  14
```

- 若不想顯示運算結果, 只需在敘述句末加上分號 (;) 即可。

```
octave:12> 3 + 2
ans =  5
octave:13> 3 + 2;
```

- 敘述若太長, 可以三句點 (...) 斷行, 再由下一行續寫。例如:

```
octave:14> 1 + 2 ...
> + 3
ans =  6
```

注意這裡在 2 與 ... 之間要留空白。如果少了空白, 程式就被解讀成 2. 之後再加二句點 .., 這樣就錯誤了。上式的效果等於 1 + 2 + 3.

使用變數

在計算時, 可以用自行命名變數來儲存數值。例如:

```
octave:1> a = 5;
octave:2> x = 10;
octave:3> y = (x - a)^2;
```

前兩行指令分別將數值 5 和 10 以變數 a 和 x 儲存起來, 這種敘述稱為「 指派 (assignment) 」。(雖然看似相同, 但「 指派 」並非數學上的「 等於 」, 請見以下的邏輯運算元。) 而第 3 行則計算 $(x-a)^2$, 並將計算值以變數 y 儲存。變數的命名有些限制:

1. 變數名稱的字首必須是英文字母或是特殊字元 _ (Matlab 只接受英文字母為字首)。
2. 變數名稱的長度不拘, 但 Octave 只辨識前 ? 個字元 (Matlab 則辨識前 31 個字元)。
3. 變數名稱中的英文字母是大小寫有別 (case-sensitive)。

另外有一些系統使用的變數名稱, 常見者如表 2 所列。在自行命名變時時, 應避免使用。

Table : 內建常數

`i, j, I, J`	單位虛根 ($\sqrt{-1}$)
`e`	歐拉 (Euler) 常數 ($e = 2.7182818\cdots$)
`pi`	圓周率
`eps`	機器精確值 (Machine epsilon), 約為 $2.2 \cdot 10^{-16}$ (若一數 x 小於 eps, 則電腦無法區分 $1 + x$ 與 1 孰大孰小)。
`inf`	無窮大 (∞); 例如 $1/0$
`nan`	非數 (Not a number); 例如 $0/0$

變數所儲存的數值可以更新, 只要將變數指派給定新數值即可。例如:

```
octave:4> a = 0
a = 0
octave:5> a = 1
a =  1
octave:6> a = a + 1
a =  2
```

第 1 行指派 $a = 0$, 第 2 行指令將 a 重新指派為 1, 而最後一行則先計算右手邊的 $a + 1$, 得出值為 2, 再將 a 更新。所以 2 就是 a 的最新值。

- 接前例,

```
octave:4> a = 0
a = 0
octave:5> a = 1
a =  1
octave:6> a = a + 1
a =  2
octave:7> b = a
b =  2
octave:8> b = 3
b =  3
octave:9> clear b
octave:10> clear
octave:11> who
```

第 4 行有新變數 b, 並將其指派為 a。第 5 行將 b 重新指派為 3。注意此處將 b 重新指派為 3 並不會對 a 造成影響, a 仍是 2。
接著一行是將 b 自工作空間去除。clear 是將全部變數均自工作空間清除。who 顯示目前工作空間有那些變數。

- 多個敘述也可以寫在同行, 但是要以逗號 (,) 或分號 (;) 加以隔開。例如, 前例也可寫成:

```
a = 0;   a = 1;   a = a + 1
```

應用例—

Solow 模型:

$$k_{t+1} = sy_t + (1-\delta)k_t, \quad y_t = Ak_t^{\alpha}. \tag{1}$$

合併兩式成為

$$k_{t+1} = sAk_t^{\alpha} + (1-\delta)k_t. \tag{2}$$

在長期均衡時,

$$k^* = sAk^{*\alpha} + (1-\delta)k^*. \tag{3}$$

可以解出

$$k^* = (sA/\delta)^{1/(1-\alpha)}. \tag{4}$$

當理論遇到實際—實證估計(年資料):

$$\alpha = 0.3, \quad \delta = 0.1, \quad s = 0.25.$$

再標準化 $A = 1$.

```
octave:14> alpha = 0.3;
octave:15> delta = 0.1;
octave:16> s     = 0.25;
octave:17> A     = 1;
octave:18> kss   = (s*A/delta)^(1/(1-alpha))
kss =  3.7024
```

陣列

- 陣列 (array) 是一種資料型態, 用以儲存一群資料。
- 陣列可以有許多維度 (dimension), 維度往往用以表示資料特性或是變數個數。例如: 時間序列可視為 2 維 (t, R); 追蹤資料 (panel data) 可視為 3 維 (t, i, R)。
- 一維陣列的建立很容易, 只要將資料置入中括號 [ ] 內, 中間用空白或逗號隔開即可。例如, 將 1, 2, 3, 4 四筆資料儲存於名為 a 的陣列:

```
octave:1> a = [1, 2, 3, 4]
a =

   1   2   3   4

octave:2> a = [1 2 3 4]
a =

   1   2   3   4
```

- 若中括號內不放任何元素, 則為空陣列。

向量

- 矩陣 (matrix) 是二維陣列, 具有行 (row) 以及列 (column) 兩維度。
- 向量 (vector) 則被視為一種特殊的矩陣, 其行或列的數目是 1。
- 向量亦可視為純量的並置 (concatenation), 水平並置以逗號或空白表示, 而垂直並置則以分號表示。

```
octave:3> A = [1 2]
A =

   1   2

octave:4> B = [1; 2]
B =

   1
   2
```

- 在 MATLAB 語言, 向量和純量都視為一種特殊的矩陣, 所以都是二維陣列。

冒號運算元 (:) 可用來建立行向量。尤其常用來建立等差數列, 語法是

$$a : s : b$$

這會建立一個 $a, a+s, a+2s, \ldots, a+\lfloor \frac{b-a}{s} \rfloor s$ 的等差數列。

- a, s, 和 b 可以是任意實數。
- 若 $s = 1$, 可以省略。
- 若 $s > 0$, 則應 $a < b$, 否則會產生空矩陣。同理可知 $s < 0$ 的情形。

```
octave:5> 1:5
ans =

   1   2   3   4   5

octave:6> 5:-1:1
ans =

   5   4   3   2   1
```

```
octave:7> -1:2:5
ans =

  -1   1   3   5

octave:8> -1:2:6
ans =

  -1   1   3   5

octave:9> -1:0.5:1
ans =

   -1.0000   -0.5000    0.0000    0.5000    1.0000

octave:10> -1:-2
ans = [](1x0)
```

- 另外一個常用的建立向量的指令是 linspace, 其語法是 linspace(*start,end,numbers*)

```
octave:11> linspace(0,1,5)
ans =

   0.00000   0.25000   0.50000   0.75000   1.00000
```

這會在 $[0,1]$ 間等距離取 5 點, 包括 0 與 1。

- 字元 end 可用來指向向量的最後一個元素。
- 如果要刪除某一元素, 可以將該元素重設為空集合。

```
octave:12> a = [1 2 3 4 5]
a =

   1   2   3   4   5

octave:13> a(end) = 9
a =

   1   2   3   4   9

octave:14> a(3) = []
a =

   1   2   4   9
```

矩陣

- 矩陣可以視為多個向量的並置。
- 向量的水平並置:

```
octave:15> A = [1 2]
A =

   1   2

octave:16> A = [A, [3 4]]
A =

   1   2   3   4
```

- 向量的垂直並置:

```
octave:17>  A = [1 2]
A =

   1   2

octave:18> A = [A; [3 4]]
A =

   1   2
   3   4
```

向量的水平或垂直並置應注意行數或列數的一致。

```
octave:19> A = [1 2 3; 4 5 6] % vertical concatenation
A =

   1   2   3
   4   5   6

octave:20> B = [[1;4], [2;5], [3;6]] % horizontal concatenation
B =

   1   2   3
   4   5   6
```

當水平和垂直並置同時出現時, 先水平, 後垂直。所以定義行向量的中括號有時可以省略, 例如:

```
octave:22> [[1 2 3]; [4 5 6]]
ans =
   1   2   3
   4   5   6
```

但以下的兩個敘述就不同了:

```
octave:23> [[1;4], [2;5]]
ans =
   1   2
   4   5
octave:24> [1;4, 2;5]
error: number of columns must match (2 != 1)
```

第 2 行錯誤因為 Octave 先處理 [4,2] 這個水平並置, 然後在處理垂直並置時便出現列數不合的錯誤。

函數 size() 可用來檢查矩陣的行數與列數, 例如:

```
octave:24> a = 1;
octave:25> b = [1, 2];
octave:26> c = [1 2; 3 4];
octave:27> size(a)
ans =

   1   1

octave:28> size(b)
ans =

   1   2

octave:29> size(c)
ans =

   2   2
```

矩陣的索引

假設 A 為一 $m \times n$ 矩陣, 索引有兩種方式。

- 以 $A(i,j)$ 的形式稱為二維索引, $A(i,j)$ 指向第 i 行-第 j 列的元素。
- 以 $A(h)$ 的形式則稱為一維索引, 其方式如同由第 1 列第 1 個元素開始往下計數, 該列的元素數完後換下一列, 然後反覆, 直到累計的元素個數到達 h 為止。
- 若 $h = i + (j-1)m$, 則 $A(h)$ 與 $A(i,j)$ 指向同一元素。

```
octave:30> A = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
A =
    1    2    3
    4    5    6
    7    8    9
   10   11   12
octave:31> A(3,2)
ans =  8
octave:32> A(7)
ans =  8
```

二維索引除了指向矩陣的某一元素, 也可以指向多個元素, 不過此時索引的行或列座標必需利用向量。

```
octave:33> A([2 4],[1 3])
ans =

    4    6
   10   12

octave:34> A([2 4],:)
ans =

    4    5    6
   10   11   12
```

第 33 行的二維索引利用向量 [2 4] 指向矩陣的第 2 和 4 兩行, 同時利用向量 [1 3] 指向矩陣的第 1 和 3 兩列。第 34 行的二維索引也是利用向量 [2 4] 指向矩陣的第 2 和 4 兩行, 但此時的列座標是利用冒號 (:) 指向全部各列。

```
octave:35> A(:,[1 3])
ans =

    1    3
    4    6
    7    9
   10   12
```

同理, 第 35 行的二維索引的行座標是以冒號指向矩陣全部各行, 所以此時應得矩陣的第 1 和 3 兩列的全部元素。
此外, 也可以 end 來指向最後一行或最後一列, 例如:

```
octave:36> A(3,end)
ans =  9
octave:37> A(end,2)
ans =  11
```

第 2 行指的是矩陣最後一列的第 3 行元素, 亦即 9; 而第 3 行指的是最後一行的第 2 列元素, 亦即 11。

一維索引若使用冒號, 會將全部元素組成一列。

```
octave:39> B = [1 2; 3 4]
B =

   1   2
   3   4

octave:40> B(:)
ans =

   1
   3
   2
   4
```

特殊矩陣

- `zeros(m,n)`: $m \times n$ 的矩陣, 全部元素均為 0.
- `ones(m,n)`: $m \times n$ 的矩陣, 全部元素均為 1.
- `eye(m)`: $m \times m$ 的 identity matrix.

```
octave:41> zeros(1,2)
ans =
   0   0
octave:42> ones(1,2)
ans =
   1   1
octave:43> eye(2)
ans =
Diagonal Matrix
   1   0
   0   1
```

矩陣的數學運算

- 矩陣與純量的四則運算均是作用於全部元素, 例如:

```
octave:44> a = [1 2 3];
octave:45> a + 1.5
ans =
   2.5000   3.5000   4.5000

octave:46> a / 1.5
ans =
   0.66667   1.33333   2.00000
```

同理適用於減法及乘法。

- 但乘冪則需多加一個句點, 例如, a 的每一元素均取平方:

```
octave:47> a.^2
ans =
   1   4   9
```

- 對全部元素左除也需加句點。例如, 對每一個元素均左除以 1:

```
octave:48> a.\1
ans =

   1.00000   0.50000   0.33333
```

- 矩陣的轉置分成普通轉置和厄米轉置 (Hermitian transpose) 兩種, 給定矩陣 A, 前者對應數學教科書裡的 A^T, 而後者對應 A^H。兩者都是 A 的轉置, 但 A^H 的元素是 A^T 的共軛複數。以實例來說:

```
octave:49> a = [1; 1+2i]
a =
   1 + 0i
   1 + 2i

octave:50> a.'
ans =
   1 + 0i   1 + 2i

octave:51> a'
ans =
   1 - 0i   1 - 2i
```

第 50 行是普通轉置, 而第 51 行是厄米轉置, 少一個句點。

- 相同大小的矩陣可以進行元素對元素 (element-by-element) 的四則運算，不過乘法和除法要多加一個句點，例如:

```
octave:52> a = [1, 2, 3];
octave:53> b = [2, 3, 5];
octave:54> a + b
ans =
   3   5   8

octave:55> a .* b
ans =
    2    6   15

octave:56> a ./ b
ans =
   0.50000   0.66667   0.60000

octave:57> a .\ b
ans =
   2.0000   1.5000   1.6667
```

Table : 矩陣運算

`C = A±B`	$c_{ij} = a_{ij} \pm b_{ij}$
`C = A*B`	$c_{ij} = \sum_k a_{ik} b_{kj}$
`C = A\B`	$C = A^{-1}B$
`C = A/B`	$C = AB^{-1}$, equivalent to `(B'\A')'`
`C = A.'`	$c_{ij} = a_{ji}$
`C = A.*B`	$c_{ij} = a_{ij} * b_{ij}$
`C = A./B`	$c_{ij} = a_{ij}/b_{ij}$
`C = A.\B`	$c_{ij} = (a_{ij})^{-1} b_{ij}$
`C = A.^2`	$c_{ij} = a_{ij}^2$

應用例二

Solow 模型:

$$k_{t+1} = sAk_t^{\alpha} + (1-\delta)k_t. \tag{5}$$

在長期均衡時,

$$k_{t+1} = k_t. \tag{6}$$

參數:

$$\alpha = 0.3, \quad \delta = 0.1, \quad s = 0.25, \quad A = 1.$$

繪製上述兩函數。

```
alpha = 0.3;
delta = 0.1;
s     = 0.25;
A     = 1;
kvec  = linspace(0,4.5,100);
f1    = s*A*kvec.^alpha + (1-delta)*kvec;
f2    = kvec;
figure(1);
plot(kvec,f1);
figure(2);
plot(kvec,f2);
figure(3);
plot(kvec,f1,kvec,f2)
```

應用例三

需求與供給模型:

$$D = 10 - P \tag{7}$$

$$S = 2 + 0.5P \tag{8}$$

均衡:

$$D = S = Q \tag{9}$$

解聯立方程式:

$$\begin{bmatrix} 1 & 1 \\ -0.5 & 1 \end{bmatrix} \begin{bmatrix} P \\ Q \end{bmatrix} = \begin{bmatrix} 10 \\ 2 \end{bmatrix} \tag{10}$$

令上式為 $Ax = b$, 均衡價格與數量為 $x = A^{-1}b$。

```
A = [1 1; -0.5 1];
b = [10; 2];
x = A\b;
```

常用數學函數

Table : 指數與對數

exp(x)	指數
log(x)	自然對數
log10(x)	以10為基底的對數
sqrt(x)	$\sqrt{x}$
nthroot(x,n)	$\sqrt[n]{x}$

Table : 複數的運算

abs(z)	長度
conj(z)	共軛複數
imag(z)	虛部
real(z)	實部

Table : 三角函數

sin(x)	正弦
cos(x)	餘弦

Table : 工具函數

fix(x)	在 x 小數點後無條件捨去得出的整數
round(x)	將 x 四捨五入後得出的整數
floor(x)	小於 x 的最大整數
ceil(x)	大於 x 的最小整數
rem(x,y)	$x - y * fix(x/y)$
mod(x,y)	$x - y * floor(x/y)$
min(x)	向量 x 的最小值
max(x)	向量 x 的最大值
mean(x)	平均值
median(x)	中位數

Table : 加乘

sum(x)	總和
prod(x)	總乘積
cumsum(x)	累加
cumprod(x)	累乘

邏輯運算元與布林變數

邏輯運算元 (logical operators) 有表示兩變數之大小關係的等號與不等號, 也有表示邏輯敘述的「非」,「且」與「或」, 詳如表 9 所示。

Table : 邏輯運算元

==	等於 (equal to)
>=	不小於 (not less than)
<=	不大於 (not greater than)
~= 或 <>	不等於 (not equal to)
<	小於 (less than)
>	大於 (greater than)
&	且 (and)
|	或 (or)
~ 或 !	非 (not)

布林值有 1 以及 0, 前者為真, 而後者為偽。例如:

```
octave:1> a = (4 <= 1)
a = 0
octave:2> b = (4 > 1)
b =  1
octave:3> a & b
ans = 0
octave:4> a | b
ans =  1
octave:5> c = 4*a
c = 0
octave:6> d = 4*b
d =  4
```

邏輯運算元可以作用在向量或矩陣的每一元素上, 例如:

```
octave:7> x = [1, -1, 2];
octave:8> a = (x > 0)
a =

   1   0   1

octave:9> all(a)
ans = 0
octave:10> any(a)
ans =  1
```

第 7 行建立行向量 x, 第 8 行則是對 x 的元素逐一檢查是否大於 0, 然後顯示布林值。 第 9 行指令是檢查向量 a 之各元素均為真 (或均非 0)。第 10 行則是檢查向量 a 是否有一元素為真 (非 0)。

字串變數

- 字串 (string) 是字元形成的數列, 以單引號 (quote) 來建立, 例如:

```
octave:1> x = 'apple'
x = apple
octave:2> y = 'apple '
y = apple
```

單引號內的字元是逐文引述 (verbatim), 空白也算一個字元, 因此, x 不等於 y 。

- 字串內容若需要使用單引號, 則需多輸入一個單引號, 例如:

```
octave:3> z = 'John''s apple'
z = John's apple
```

- 字串被視為字元形成的矩陣, 所以矩陣的索引方式也適用。例如, 將 x 的第 2 及 3 字元替代成 b:

```
octave:4> x(2:3) = 'bb'
x = abble
```

- 指令 `length` 計算字串的長度:

```
octave:5> length(x)
ans =  5
octave:6> length(y)
ans =  6
```

- 字串的水平並置和數值變數並無不同, 例如:

```
octave:7> x = 'apple'
x = apple
octave:8> y = [x, 's']
y = apples
```

- 字串的垂直並置特別簡易, 當字串長度不同時也可行:

```
octave:9> x = 'apples';
octave:10> y = 'apples ';
octave:11> z = 'John''s apples';
octave:12> a = [x; y; z]
a =

apples
apples
John's apples

octave:13> size(a)
ans =

    3   13
```

此處 x 的長度是 6, y 是 7, z 是 13, 長度不同, 但 Octave 會自動在 x 和 y 之後加空白, 使得 x, y, z 都是長度 13, 然後再垂直並置。

- 另一個垂直並置的方法是

```
octave:14> char(x, y, z)
ans =

apples
apples
John's apples
```

顯示字串

最常用的是 disp, 這會將字串直接顯示在螢幕上。

```
octave:15> disp('John is 100 years old')
John is 100 years old
```

另一種方式比較可以有較多的彈性:

```
octave:16> name = 'John';
octave:17> age = 100;
octave:18> str = sprintf('%s is %d years old', name, age);
octave:19> disp(str)
John is 100 years old
```

- sprintf 是類 C 語言的函數, 函數內的第一個變數是字串, 該字串中可以有數個以 % 開頭的「填空指示」, 所需填入的資料則以第 2, 3, ... 個變數表示。以 % 開頭的填空指示是一種 format, 表示填入資料的型態以及所欲顯示的 format。例如, 在本例中, 有兩個填空指示, 第一個是 %s, 這表示該填入一個字串, 而第二個是 %d, 這表示該填入一個整數。然後我們依序將 name 和 age 兩個資料填入, 完成了這個字串, 再將之指派給 str 這個變數。
- 最後一行則是以 disp 將 str 顯示出來。

Table : C format descriptors

Form	Use
%c	Single character
%s	Character string
%d	Signed decimal integer
%e	Floating-point number, e-notation
%f	Floating-point number, decimal notation
%g	The shorter one between %e and %f
%%	Print a percent sign

除了型態之外, 還可以限定顯示的長度, 常用的語法如:

```
%w.df
```

此處 w 是 minimal field width, d is digits after decimal point, 而 f 是以浮點型態顯示 (可換成其他型態)。比較以下兩例:

```
octave:20> disp(sprintf('Pi equals %6.4f', pi))
Pi equals 3.1416
octave:21> disp(sprintf('Pi equals %7.4f', pi))
Pi equals  3.1416
```

- 第 1 行限定 w=6, d=4, 所以顯示 π 時, 顯示出小數點後 4 位, 而且全長不得少於 6。由於 $\pi = 3.14159...$, 只顯示小數點後 4 位則應為 3.1416, 全長包含小數點恰為 6, 所以合乎 w=6 的要求。
- 第 2 行限定 w=7, d=4, 也是顯示出小數點後 4 位, 但全長不得少於 7。由前述已知, 顯示小數點後 4 位時全長僅為 6, 為滿足 w=7 的限制, 前方會加入 1 個空白。

程式檔

- 在互動模式 (interactive mode) 之外, 批次模式 (batch mode) 更有利於程式的修改與重覆利用。
- 在提示符號下鍵入 edit, 即會有 NotePad++ 的視窗彈出, 可以開始編寫程式檔。
- 程式檔撰寫完畢後, 須命名為附加檔名為 .m 的檔案, 並儲存於搜尋路徑上的目錄中。
- pwd: present working directory 顯示目前的工作目錄
- cd: change directory 可以改變工作目錄, 例如:

```
octave:1> cd /home/jenn-hong/octave/work
```

此處的路徑是 Linux 檔案系統, Windows 使用者請自行改成對應的檔案路徑, 如 `c:\octave\work` 之類。

- path: 顯示 Octave 的搜尋路徑

```
octave:2> path
```

- addpath: 將目錄加入搜尋路徑, 例如:

```
octave:3> addpath /home/jenn-hong/octave/work
```

程式儲存成檔可以修改並再利用。 例如, 將下列敘述存成 test0.m:

```
name = 'John';
age  = 100;
disp(sprintf('%s is %d years old', name, age))
```

將此檔儲存在搜尋路徑的目錄中, 然後在提示符號下直接呼叫主檔名即執行此程式:

```
octave:4> test0
John is 100 years old
```

自定函數

- 自定函數與內建函數一樣, 有 input 及 output, 也允許無 output。
- 例如, 寫下列簡單的函數, 並存檔為 test1.m:

```
function test1(name,age)
% Display the age of a particular person

disp(sprintf('%s is %d years old', name, age))
```

- 第一行的內容一定是
  function *output = function_name(input)*
- 百分比 % 或井字號 # 之後的敘述為註解, 不會被執行。
- 使用自定函數如同使用內建函數, 例如, 可以在提示符號下直接呼叫上述函數 (使用程式檔名):

```
octave:5> test1('Bob', 99)
Bob is 99 years old
```

- 通常比較良好的習慣是函數名與檔名一致, 但有時可能會疏忽而發生不一致, 此時仍是呼叫檔名, 不會有任何影響。

- 函數內的變數除特別指定外, 均是局部變數 (local variables)。例如:

```
function test2(name,year)
str1  = sprintf('%s is %d years old', name, year)
month = 12*year;
str2  = sprintf('%s is %d months old', name, month)
disp(str1)
disp(str2)
```

- 此處 name, year, str1, str2, month 均是局部變數, 只有在函數 test2 做內部運算的時候才會出現, 一旦運算完畢, 就從工作空間消失了。
- 指令 who 可以查看目前的工作空間有那些全域變數 (global variables)。

```
octave:8> clear
octave:9> test2('Chris', 98)
Chris is 98 years old
Chris is 1176 months old
octave:10> who
```

for 迴圈

```
for i = vector
    execute commands
end
```

其中變數 i 的值會被依次設定為向量 vector 中的值, 而對應著每一個向量值, 運算式會被執行一次。例如:

```
octave:1> for i = 1:4
> disp('execute this line')
> end
execute this line
execute this line
execute this line
execute this line
```

變數 i 會依次等於 1, 2, 3, 4, 而對應著每一個值, 會在螢幕上顯示 execute this line, 所以結果是反覆顯示同樣的訊息 4 次。

通常 for 迴圈裡執行的運算式會與計數器 i 有關, 例如:

```
x = 0;
for i = 1:4
    x = x + i
end
```

第 1 行指派 $x = 0$, 第 2–4 行為迴圈, 其中 i 依序取值 1, 2, 3, 4。

當 i = 1, 執行 x = x + i, 所以 x = 0 + 1 = 1
當 i = 2, 執行 x = x + i, 所以 x = 1 + 2 = 3
當 i = 3, 執行 x = x + i, 所以 x = 3 + 3 = 6
當 i = 4, 執行 x = x + i, 所以 x = 6 + 4 = 10

```
octave:2> x = 0;
octave:3> for i = 1:4
> x = x + i
> end
x =  1
x =  3
x =  6
x =  10
```

另一個 for 迴圈的例子:

```
x = zeros(1,4);
for i = 1:4
    x(i) = 1/i
end
```

第 1 行指派 x = [0 0 0 0], 接著進入迴圈, i 依序取值 1, 2, 3, 4
當 i = 1, 執行 x(1) = 1/i = 1, 所以 x = [1 0 0 0]
當 i = 2, 執行 x(2) = 1/i = 1/2, 所以 x = [1 0.5 0 0]
當 i = 3, 執行 x(3) = 1/i = 1/3, 所以 x = [1 0.5 0.33 0]
當 i = 4, 執行 x(4) = 1/i = 1/4, 所以 x = [1 0.5 0.33 0.25]

```
octave:6> x = zeros(1,4)
x =
   0   0   0   0

octave:7> for i = 1:4
> x(i) = 1/i
> end
x =
   1   0   0   0

x =
   1.00000   0.50000   0.00000   0.00000

x =
   1.00000   0.50000   0.33333   0.00000

x =
   1.00000   0.50000   0.33333   0.25000
```

迴圈可以是多層或巢狀 (nested), 例如:

```
x = zeros(2,2);
for i = 1:2
    for j = 1:2
        x(i,j) = j;
    end
    x(i,:) = x(i,:)*i;
end
```

第 1 行指派 x = [0 0; 0 0], 接著進入迴圈, i 依序取值 1, 2,
當 i = 1, 執行內圈, j 依序取值 1, 2,
    當 j = 1, 執行 x(1,1) = j = 1, 所以 x = [1 0; 0 0]
    當 j = 2, 執行 x(1,2) = j = 2, 所以 x = [1 2; 0 0]
    然後執行外圈, x(1,:) = x(1,:)*1, 所以 x = [1 2; 0 0]
當 i = 2, 執行內圈, j 依序取值 1, 2,
    當 j = 1, 執行 x(2,1) = j = 1, 所以 x = [1 2; 1 0]
    當 j = 2, 執行 x(2,2) = j = 2, 所以 x = [1 2; 1 2]
    然後執行外圈, x(2,:) = x(2,:)*2, 所以 x = [1 2; 2 4]

```
octave:12> x = zeros(2,2)
x =
   0   0
   0   0
octave:13> for i = 1:2
> for j = 1:2
> x(i,j) = j
> end
> x(i,:) = x(i,:)*i
> end
x =
   1   0
   0   0
x =
   1   2
   0   0
x =
   1   2
   0   0
```

```
x =
   1   2
   1   0
x =
   1   2
   1   2
x =
   1   2
   2   4
```

while 迴圈

```
while  condition
    execute commands
end
```

此迴圈先檢查條件 condition 是否為真, 若是, 則執行運算式, 然後反覆此迴圈。例如:

```
x = 0;
i = 1;
while i < 5
    x = x + i;
    i = i + 1;
end
```

第 1 行指派 x = 0 以及 i = 1, 接著進入迴圈,
i=1, 條件 i<5 為真, 執行 x=x+i=1, i=i+1=2, 所以 x=1, i=2;
i=2, 條件 i<5 為真, 執行 x=x+i=3, i=i+1=3, 所以 x=3, i=3;
i=3, 條件 i<5 為真, 執行 x=x+i=6, i=i+1=4, 所以 x=6, i=4;
i=4, 條件 i<5 為真, 執行 x=x+i=10, i=i+1=5, 所以 x=10, i=5;
i=5, 條件 i<5 為偽, 迴圈停止。

```
octave:1> x = 0; i = 1;
octave:2> while i < 5
> x = x + i
> i = i + 1
> end
x =  1
i =  2
x =  3
i =  3
x =  6
i =  4
x =  10
i =  5
```

強迫迴圈停止按 Ctrl-C。例如, 下例是個無限循環:

```
x = 0;
i = 1;
while i < 5
    x = x + i;
end
```

由於計數器 i 始終沒有改變, while 的檢測條件永遠成立, 所以迴圈無法停止 (事實上還是會停, 不過可能要等很久), 這時可按 Ctrl-C 終止。

條件敘述

if *condition 1 is true*
    execute command 1
elseif *condition 2 is true*
    execute command 2
...
elseif *condition n is true*
    execute command n
else
    execute last command
end

```
x = 3;
if x > 0
    disp(sprintf('%d is a positive integer', x))
elseif x < 0
    disp(sprintf('%d is a negative integer', x))
else
    disp(sprintf('%d is zero', x))
end
```

另一個條件少的例子:

```
x = 3; y = 4;
if x > y
    disp('x > y')
else
    disp('x < = y')
end
```

應用例四

Solow 模型:

$$k_{t+1} = sAk_t^{\alpha} + (1-\delta)k_t. \tag{11}$$

若給定起始值 $k_1 > 0$, 利用上式可以反覆求得 k_t 數列 $k_2, k_3, \ldots$ 參數:

$$\alpha = 0.3, \quad \delta = 0.1, \quad s = 0.25, \quad A = 1.$$

繪製上述數列。

```
alpha = 0.3;
delta = 0.1;
s     = 0.25;
A     = 1;
nsim  = 20;
ksim  = zeros(nsim,1);
ksim(1) = 0.1;
for i = 2:nsim
    ksim(i) = s*A*ksim(i-1)^alpha + (1-delta)*ksim(i-1);
end
figure(1);
plot((1:nsim)',’o’);
xlabel(’t’)
ylabel(’k_t’)
```

記錄與退出

- diary on: 開始記錄螢幕上所出現的畫面
- diary off: 停止記錄, 會在 pwd 出現一個 diary 文字檔。
- exit: 退出 Octave

Octave 語法 II

Jenn-Hong Tang
Department of Economics
National Tsing-Hua University

October 3, 2011

多維陣列

三維甚至更高維度的陣列並不常用, 多半只是用於儲存資料, 所以在此只簡略介紹三維陣列。三維陣列具有行、列、及「 頁 」(page) 三個維度。對於「 追蹤資料 」(panel data), 可以考慮使用。

	Alex		Ben	
year	income	consumption	income	consumption
2008	100	80	200	160
2009	150	120	240	200
2010	200	180	300	320

我們可以將兩人的資料分成兩頁, 然後用二維矩陣儲存所得及消費。

```
octave:1> A = [100 80; 150 120; 200 180];
octave:2> B = [200 160; 240 200; 300 320];
octave:3> data(:,:,1) = A;  %(row,column,page)
octave:4> data(:,:,2) = B;
octave:5> data
data =

ans(:,:,1) =

   100    80
   150   120
   200   180

ans(:,:,2) =

   200   160
   240   200
   300   320
```

索引的方法和矩陣相彷, 只不過多了頁坐標。

```
octave:6> data(1,2,1)
ans =  80
octave:7> data(3,2,2)
ans =  320
```

異質陣列

之前的陣列, 內容都是相同的資料型態。如果可以容納異質的資料, 或許可以帶來某些便利。異質陣列 (cell array) 可以將不同型態的資料儲存在同一數列, 其建構的方式類似普通陣列, 不過是用大括號而非小括號:

```
octave:1> A = {'Alex', [100 150 200], [80 120 180]}
A =
{
  [1,1] = Alex
  [1,2] =

     100   150   200

  [1,3] =

      80   120   180
}
```

索引異質陣列的內容用大括弧。

```
octave:2> A{1,1}
ans = Alex
octave:3> A{1,2}
ans =

   100   150   200

octave:4> A{1,3}
ans =

    80   120   180
```

另有小括號索引, 得出卻是保留異質陣列的資料型態:

```
octave:5> x = A{1,1}
x = apple
octave:6> y = A(1,1)
y =

{
  [1,1] = apple
}

octave:7> ischar(x) %check if x is a character string
ans =  1
octave:8> iscell(y) %check if y is a cell array
ans =  1
```

如果要建立一個類似「矩陣」的異質陣列, 可以利用小括號索引:

```
octave:8> data = cell(2,3);
octave:9> data(:,1) = {'Alex', 'Ben'};
octave:10> data(:,2) = {[100 150 200], [200 240 300]};
octave:11> data(:,3) = {[80 120 180], [160 200 320]};
```

如果要更改資料, 兩種索引均可:

```
octave:16> data{1,2} = [110 150 200];
octave:17> data(1,2) = {[110 150 200]};
```

由於異質陣列容納不同資料型態, 所以沒有任何數學運算。

異質陣列用來儲存字串變數會比矩陣帶來一些方便:

```
octave:1> L = ['alex';'benjamin';'chris']
L =
alex
benjamin
chris
octave:2> cL = {'alex','benjamin','chris'}
cL =
{
  [1,1] = alex
  [1,2] = benjamin
  [1,3] = chris
}
octave:3> disp(sprintf('%s is 100 years old', L(1,:)))
alex     is 100 years old
octave:4> disp(sprintf('%s is 100 years old', cL{1}))
alex is 100 years old
```

將字串垂直並置成矩陣時, 較短的字串後面會加入空白, 以使每個字串等長。所以前頁第 3 行列印時, alex 後方會有多餘空白, 形成過大的間距。但異質陣列將每個字串依原樣儲存, 沒有多餘空白的問題。所以在第 4 行列印時, 每個字的間距都是正確的。

指令 cellstr 可以將字串矩陣轉成異質陣列, 而 char 提供逆向轉換:

```
octave:5> cellstr(L)
ans =
{
  [1,1] = alex
  [2,1] = benjamin
  [3,1] = chris
}
octave:6> char(cL)
ans =
alex
benjamin
chris
```

結構陣列

結構陣列 (structure array) 是另一種儲存異質資料的「容器」(container), 語法是: *structure name.field name = data*

```
octave:1> worker.name = 'Alex';
octave:2> worker.income = [100 150 200];
octave:3> worker.consumption = [80 120 180];
octave:4> worker
worker =
{
  name = Alex
  income =

     100   150   200

  consumption =

     80   120   180
}
```

加入另一種勞工的資料:

```
octave:5> worker(2).name = 'Ben';
octave:6> worker(2).income = [200 240 300];
octave:7> worker(2).consumption = [160 200 320];
octave:8> worker(1)
ans =
{
  name = Alex
  income =

     100   150   200

  consumption =

     80   120   180

}
```

```
octave:9> worker(2)
ans =
{
  name = Ben
  income =

     200   240   300

  consumption =

     160   200   320

}
```

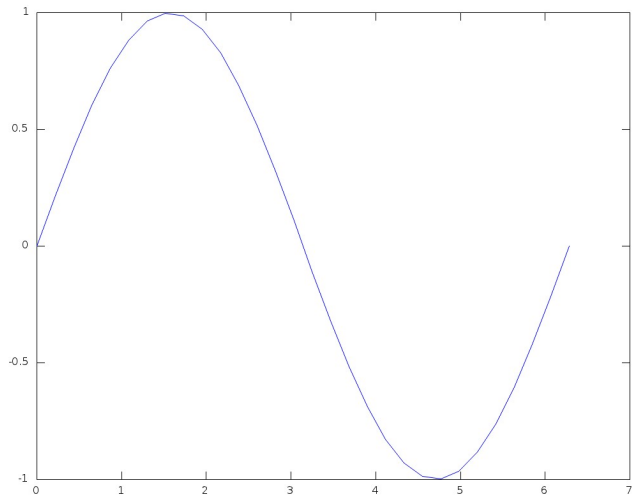
索引的方式與之前無異。例如, 索引第 2 位勞工的第 2 年的所得:

```
octave:11> worker(2).income(1,2)
ans =  240
```

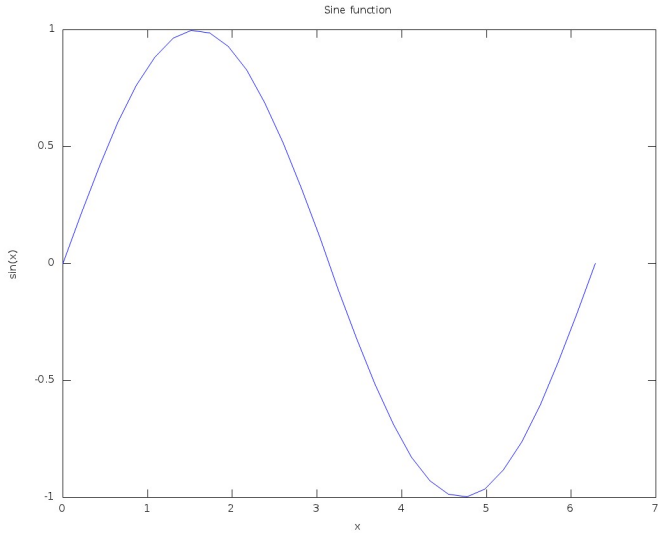
二維平面繪圖

```
octave:1> x = linspace(0,2*pi,30);
octave:2> y = sin(x);
octave:3> figure(1); clf;
octave:4> plot(x,y)
```

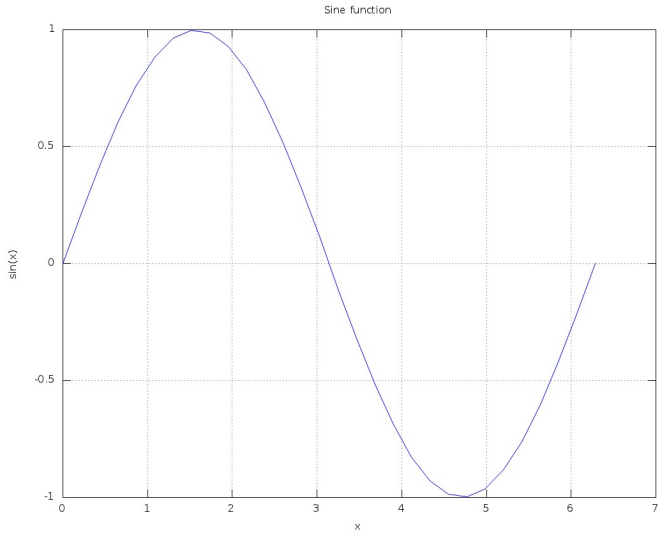
clf 清除視窗內的圖形。



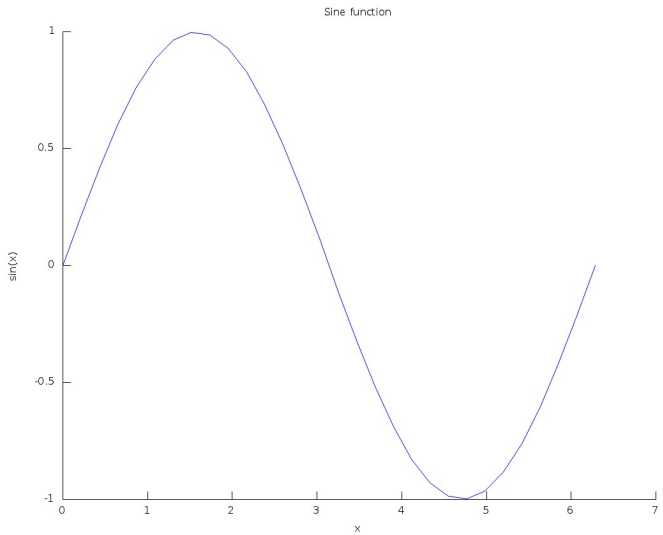

```
octave:5> xlabel('x')
octave:6> ylabel('sin(x)')
octave:7> title('Sine function')
```



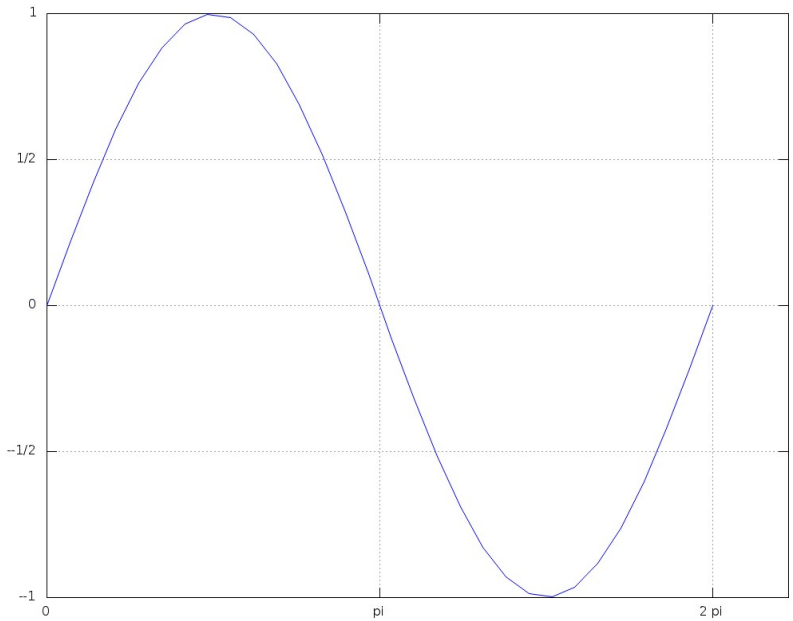
```
octave:8> grid on
```




```
octave:9> box off
```

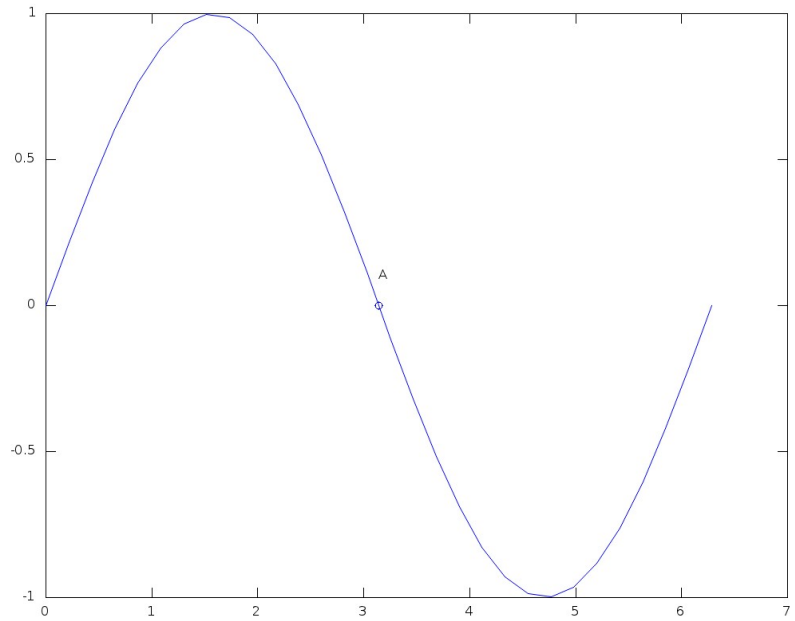


```
octave:10> set(gca,'xtick',[0,pi,2*pi])
octave:11> set(gca,'xticklabel',{'0','pi','2 pi'}) %cell arrays
octave:12> set(gca,'ytick',[-1,-1/2, 0, 1/2, 1])
octave:13> set(gca,'yticklabel',{'--1','--1/2', '0','1/2','1'})
```



- gca = get current axis
- xtick, ytick, xticklabel, yticklabel = properties of axis
- x(y)tick: (縱)軸的刻度; x(y)ticklabel: 刻度標記。

```
octave:14> plot(x,y)
octave:15> text(pi,0.1,'A')
octave:16> hold on
octave:17> plot(pi,0,'o',...
'markersize',10)
octave:18> hold off
```



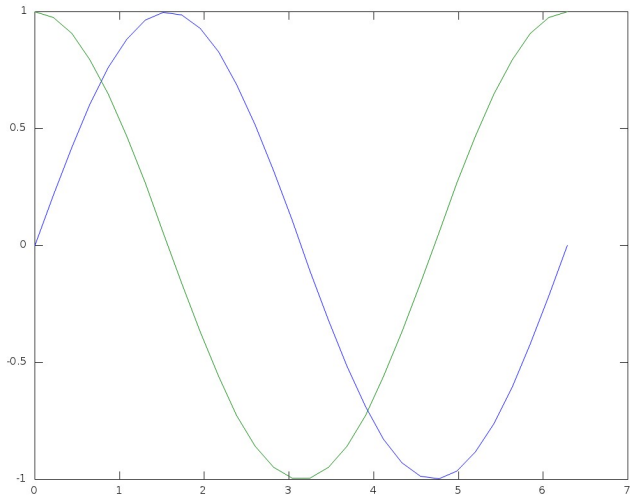
第 14 行繪圖, 接著加入文字, 第 16 行 `hold on` 將現有的圖形保留, 第 17 行在原圖形上的 $(\pi, 0)$ 位置加上一個 o, 最後一行將 `hold` 取消。若不用 `hold on`, 在利用如 17 行的 `plot` 指令加入新的線條時, 原圖形就會被取代。

Color	Symbol	Style
w white	. point	- solid
m magneta	o circle	-- dashed†
c cyan	x x-mark	: dotted†
r red	+ plus	-. dash-dotted†
g green	* star	
b blue		

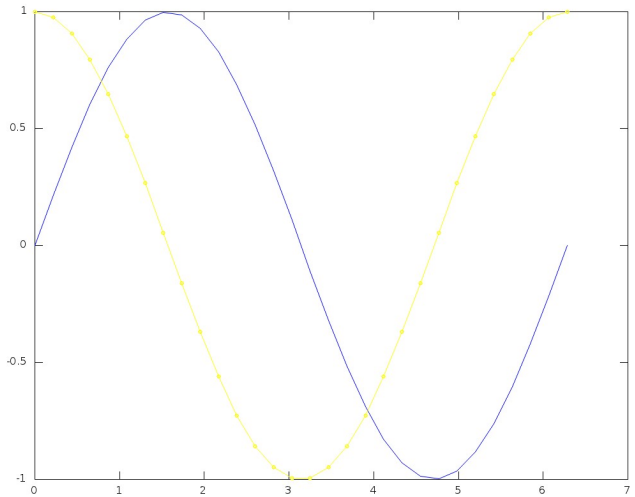
† 似乎只適用於 Version 3.4 之後。

Octave 是利用 GNUPLOT 做為繪圖工具, 目前為止, 繪圖能力及效果似乎還不及 Matlab 來得花俏。

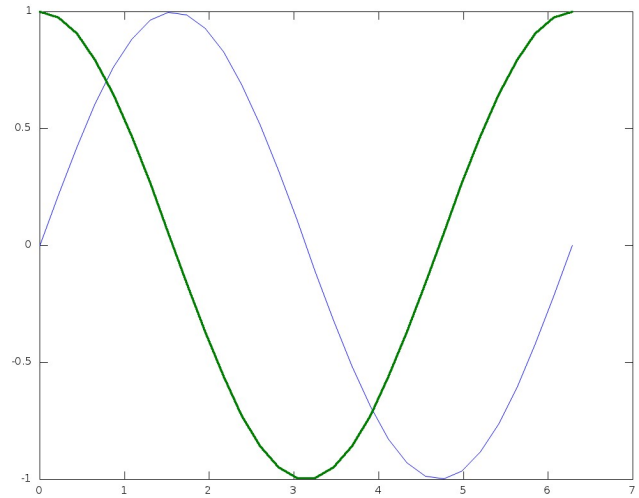
```
octave:8> z = cos(x);
octave:9> plot(x,y,x,z)
```



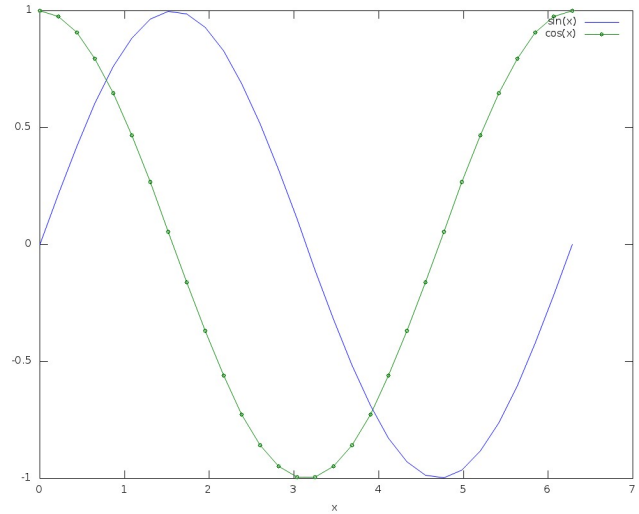

```
octave:10> z = cos(x);
octave:11> plot(x,y,x,z,'-og')
```



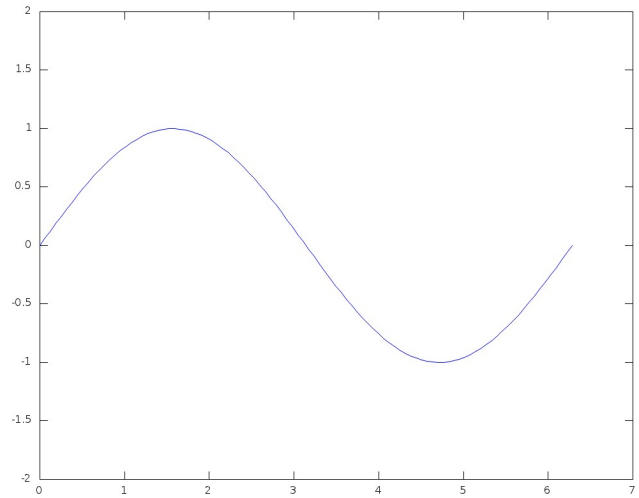
```
octave:12> plot(x,y,'LineWidth',1,...
> x,z,'LineWidth',3)
```




```
octave:13> plot(x,sin(x),x,cos(x),'-o')
octave:14> xlabel('x')
octave:15> legend('sin(x)','cos(x)')
```

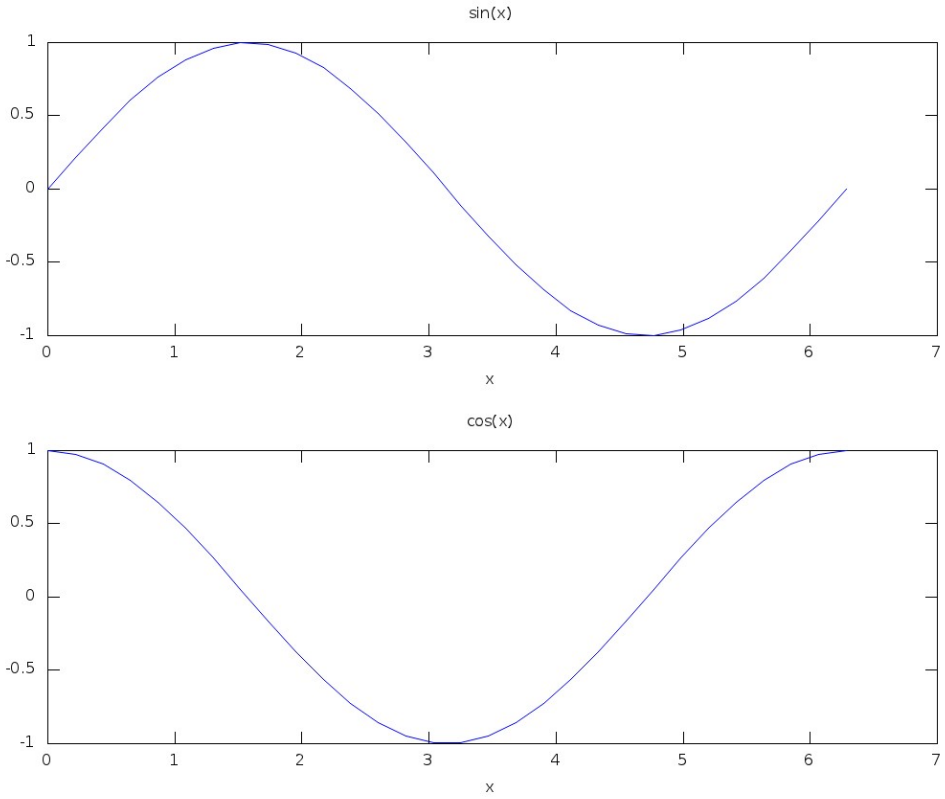


```
octave:14> x = linspace(0,2*pi,100);
octave:15> plot(x,sin(x))
octave:16> axis
ans =
   0   7  -1   1   0   1
octave:17> axis_old = axis;
octave:18> axis([axis_old(1:2) -2 2])
octave:19> axis
ans =
   0   7  -2   2   0   1
```

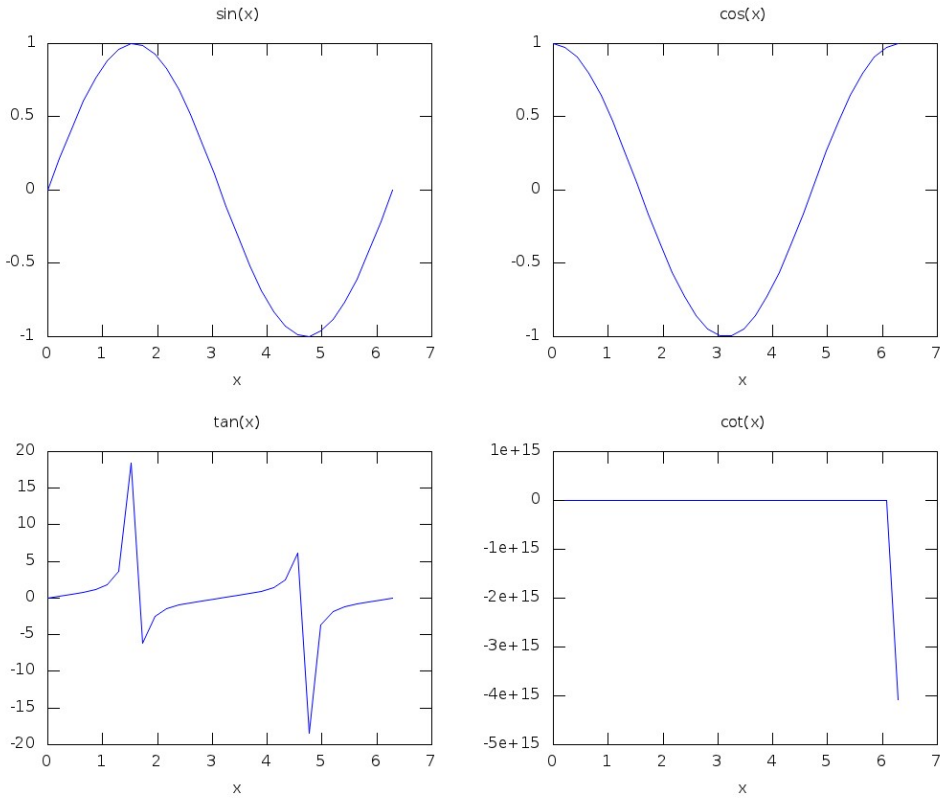


第 16 行顯示目前 gca 的坐標, 是個 1×6 的向量 ([xmin, xmax, ymin, ymax, zmin, zmax])。第 17 行將目前的坐標以 axis_old 先存起來, 接著 18 行設定新坐標, 將 (ymin, ymax) 設成 $(-2, 2)$。第 19 行顯示更新後的坐標。

```
octave:1> x = linspace(0,2*pi,30);
octave:2> subplot(2,1,1),plot(x,sin(x)),xlabel('x'),title('sin(x)')
octave:3> subplot(2,1,2),plot(x,cos(x)),xlabel('x'),title('cos(x)')
```



```
octave:1> x = linspace(0,2*pi,30);
octave:2> subplot(2,2,1),plot(x,sin(x)),xlabel('x'),title('sin(x)')
octave:3> subplot(2,2,2),plot(x,cos(x)),xlabel('x'),title('cos(x)')
octave:4> subplot(2,2,3),plot(x,tan(x)),xlabel('x'),title('tan(x)')
octave:5> subplot(2,2,4),plot(x,cot(x)),xlabel('x'),title('cot(x)')
```



利用 `print` 函數可以將圖形輸出儲存:

```
octave:1> x = linspace(0,2*pi,30);
octave:2> figure(1); clf;
octave:3> plot(x,sin(x))
octave:4> print('~/octave/work/graph1.jpg','-djpg')
```

Octave 提供多種檔案型態, 常見的有:

jpg
pdf
eps
png

在 Windows 系統下, 改成
`print('c:\mywork\...\graph1.jpg','-djpg')`。

檔案讀寫

最簡單的讀寫是利用 save 和 load 兩個指令。

```
octave:1> A = rand(5,2);
octave:2> save ~/octave/work/data1.mat A
```

再把剛儲存的資料讀入

```
octave:3> clear
octave:4> load ~/octave/work/data1.mat
octave:5> who
Variables in the current scope:

A
```

以上是最基本的, 以二進制 (binary) 儲存。如果用 Windows, 將指令改成

```
save c:\mywork\...\data1.mat A
load c:\mywork\...\data1.mat
```

也可以用 ASCII 形式:

```
octave:7> save ~/octave/work/data1.txt A -ascii
octave:8> clear
octave:9> load ~/octave/work/data1.txt
octave:10> who
Variables in the current scope:

data1

octave:11> data1
data1 =
   0.452914   0.808764
   0.503164   0.644444
   0.710218   0.087674
   0.140414   0.348918
   0.941483   0.021266
```

以 ASCII 儲存時, 沒有保存變數名稱, 所以再載入時, 沒有原始的變數名稱, 只有以檔案名稱做為變數名稱。

有 format 控制的讀寫:

```
octave:7> Celsius = 35:40;
octave:8> Fahrenheit = Celsius*9/5 + 32;
octave:9> fid = fopen('~/octave/work/temperature.txt','w');
octave:10> fprintf(fid,'%6.2f %6.2f\n',[Celsius; Fahrenheit]);
octave:11> fclose(fid);
```

第 10 行裡, fprintf 是逐列讀取 (read by columns), 但逐行書寫 (write by rows)。所以我們讓 fprintf 的第 3 個變數是 Celsius 以及 Fahrenheit 兩個行向量的垂直並置, 是個 2x6 的矩陣。儲存在 temperature.txt 的內容就會是 6x2 的矩陣:

```
35.00   95.00
36.00   96.80
37.00   98.60
38.00  100.40
39.00  102.20
40.00  104.00
```

```
octave:13> fid = fopen('~/octave/work/temperature.txt','r');
octave:14> dd = fscanf(fid,'%f%f',[2,inf])
dd =
    35.000    36.000    37.000    38.000    39.000    40.000
    95.000    96.800    98.600   100.400   102.200   104.000
octave:15> fclose(fid);
```

fscanf 是逐行讀取 (read by rows), 但會逐列書寫 (write by columns)。由於 temperature.txt 的矩陣列數為 2, fscanf 的結果就必有行數為 2, 所以 fscanf 的第 3 個變數設成 [2,inf], 限定行數為 2 但不限定列數。如果不加入 [2,inf] 的設定, 結果會是一個列向量, 有如 dd(:)。

常見錯誤

- 使用未定義之變數

```
octave:1> a = 1;
octave:2> x = 2;
octave:3> y = (x - b)^2
error: ‘b’ undefined near line 3 column 10
```

- 大小寫混淆或打字錯誤

```
octave:3> y = (X-a)^2
error: ‘X’ undefined near line 3 column 6
```

- 括號不對稱

```
octave:4> 1+(2*(3+4);
parse error:
  syntax error
>>> 1+(2*(3+4);
               ^
```

- 使用未定義之函數或誤植函數名

```
octave:5> eyes(3)
error: 'eyes' undefined near line 7 column 1
```

- 矩陣維度不合

```
octave:6> A = eye(5);
octave:7> B = [1:5, A]
error: number of rows must match (5 != 1) near line 4, column 11
```

- 索引與矩陣維度不合

```
octave:8> x = zeros(1,5);
octave:9> x(1) = 0;
octave:10> shocks = randn(1,5); % random sample from N(0,1)
octave:11> for i = 1:5
> x(i+1) = 0.95*x(i) + shocks(i+1);
> end
error: A(I): Index exceeds matrix dimension.
```

- 邏輯錯誤: 語法無誤, 但卻得出錯誤結果。例如, 正確的程式是:

```
octave:1> a = 2;
octave:2> x = 1/(2*a);
```

卻誤植為

```
octave:3> x = 1/(2-a);
warning: division by zero
```

- 有時錯誤更加難以發現

```
octave:4> x = 1/(2+a);
```

- 想得到 1×4 的矩陣, 卻得到 1×5

```
octave:5> x = zeros(1,4); x(1) = 10;
octave:6> for i = 1:4
> x(i+1) = 0.9*x(i);
> end
octave:7> x
x =
   10.0000    9.0000    8.1000    7.2900    6.5610
```

要減少錯誤, 平時就要培養好習慣:

- 將工作分成小段, 每一小段完成後檢查結果是否合理。
- 程式力求清晰及可讀性, 以方便閱讀及除錯。
- 平時練習具有解析解的問題, 或是期刊論文上的問題, 以方便檢驗練習的結果是否正確。
- 儘量將程式函數化或模組化, 儘量使用內建函數, 以減少程式撰寫量, 並減少錯誤。
- 多觀摩, 多練習, 以累積程式除錯的經驗, 並學習程式撰寫的技巧。

Octave: Symbolic Computation

Jenn-Hong Tang
Department of Economics
National Tsing-Hua University

September 26, 2011

- Down load the package `symbolic-1.0.9.tar.gz` from sourceforge.net
- Move the tar.gz file to `c:\octave\3.2.4_gcc-4.4.0\bin`
- Under the Octave prompt, execute `pkg install xxx.tar.gz`
- Octave 的符號運算套件功能不多, 而且語法與 Matlab 也有很大出入, 不過所幸本課程使用符號運算的機會不多。

```
octave:1> symbols
octave:2> x = sym('x');
octave:3> f = Log(x)
f =

log(x)
octave:4> g = Exp(x)
g =

exp(x)
octave:5> y = sym('y');
octave:6> g = subs(g,x,y)
g =

exp(y)
```

```
octave:7> df = differentiate(f,x)
df =

x^(-1)
octave:8> Hf = differentiate(f,x,2)
Hf =

-x^(-2)
octave:9> differentiate(df,x)
ans =

-x^(-2)
```

```
octave:10> xval = 0.5;
octave:11> dfval = subs(df,x,xval)
dfval =

2.0
octave:12> is_ex(dfval)
ans =  1
octave:13> sqrt(dfval)
error: octave_base_value::sqrt (): wrong type argument ‘ex’
octave:14> dfval = to_double(dfval)
dfval =  2
octave:15> sqrt(dfval)
ans =  1.4142
```