CHƯƠNG 1: MATLAB CƠ BẢN

§1. KHỞI ĐỘNG MATLAB

1. Khởi động MATLAB: MATLAB (Matrix laboratory) là phần mềm dùng để giải một loạt các bài toán kĩ thuật, đặc biệt là các bài toán liên quan đến ma trận. MATLAB cung cấp các toolboxes, tức các hàm mở rộng môi trường MATLAB để giải quyết các vấn đề đặc biệt như xử lí tín hiệu số, hệ thống điều khiển, mạng neuron, fuzzy logic, mô phỏng v.v.

Để khởi động MATLAB ta nhấn đúp vào icon của nó trên màn hình.

2.Đánh lệnh trong cửa sổ lệnh : Khi ta đánh lệnh vào cửa sổ lệnh, nó sẽ được thi hành ngay và kết quả hiện lên màn hình. Nếu ta không muốn cho kết quả hiện lên màn hình thì sau lệnh ta đặt thêm dấu ";". Nếu lệnh quá dài, không vừa một dòng dòng có thể đánh lệnh trên nhiều dòng và cuối mỗi dòng đặt thêm dấu ... rồi xuống dòng. Khi soạn thảo lệnh ta có thể dùng các phím tắt :

↑	Ctrl-P	gọi lại lệnh trước đó
↓	Ctrl-N	gọi lệnh sau
←	Ctrl-B	lùi lại một kí tự
→	Ctrl-F	tiến lên một kí tự
Ctrl-→	Ctrl-R	sang phải một từ
Ctrl-←	Crtl-L	sang phải một từ
home	Ctrl-A	về đầu dòng
end	Ctrl-E	về cuối dòng
esc	Ctrl-U	xoá dòng
del	Ctrl-D	xoá kí tự tại chỗ con nháy đứng
backspace	Ctrl-H	xoá kí tự trước chỗ con nháy đứng

3. Set path: Khi chạy các chương trình MATLAB ở các thư mục khác thư mục hiện hiện hành ta phải đổi thư mục bằng lệnh ***File | Set Path...***

4. Help và Demo: Phần nay giúp chúng ta hiểu biết các hàm, các lệnh của MATLAB và chạy thử các chương trình demo

§2. CÁC MA TRẬN

1. Các toán tử: MATLAB không đòi hỏi phải khai báo biến trước khi dùng. MATLAB phân biệt chữ hoa và chữ thường.

Các phép toán :

+ , - , * , / , \ (chia trái) , ^ (mũ) , ‘ (chuyển vị hay số phức liên hiệp).

x = 2+3

a = 5

b = 2

a/b

a\b

Các toán tử quan hệ :

< , <= , > , >= , == , ~=

Các toán tử logic :

& , | (or) , ~ (not)

Các hằng :

pi	3.14159265
i	số ảo
j	tương tự i
eps	sai số 2^{-52}
realmin	số thực nhỏ nhất 2^{-1022}
realmax	số thực lớn nhất 2^{1023}
inf	vô cùng lớn
NaN	Not a number

2. **Các ma trận**:

a. Nhập ma trận: Ma trận là một mảng các số liệu có m hàng và n cột. Trường hợp ma trận chỉ có một phần tử(ma trận 1-1) ta có một số. Ma trận chỉ có một cột được gọi là một vectơ. Ta có thể nhập ma trận vào MATLAB bằng nhiều cách:

- nhập một danh sách các phần tử từ bàn phím
- nạp ma trận từ file số liệu
- tạo ma trận nhờ các hàm có sẵn trong MATLAB
- tạo ma trận nhờ hàm tự tạo

Khi nhập ma trận từ bàn phím ta phải tuân theo các quy định sau :

- ngăn cách các phần tử của ma trận bằng dấu “,” hay dấu trống
- dùng dấu “;” để kết thúc một hàng
- bao các phần tử của ma trận bằng cặp dấu ngoặc vuông [ ]

Ví dụ: Ta nhập một ma trận

A = [16 3 2 13 ; 5 10 11 8 ; 9 6 7 12 ; 4 15 14 1]

Bây giờ ta đánh lệnh:

sum(A)

ans =

34 34 34 34

nghĩa là nó đã lấy tổng các cột vì MATLAB được viết để là việc với các cột. Khi ta không chỉ biến chứa kết quả thì MATLAB dùng biến mặc định là ans, viết tắt của answer.

Muốn lấy tổng của các hàng ta cần chuyển vị ma trận bằng cách đánh vào lệnh:

A'

ans =

16	*5*	*9*	*4*
3	*10*	*6*	*15*
2	*11*	*7*	*14*
13	*8*	*12*	*1*

và đây là chuyển vị của ma trận A.

Ma trận a = [] là ma trận rỗng

b. Chỉ số: Phần tử ở hàng i cột j của ma trận có kí hiệu là A(i,j). Tuy nhiên ta cũng có thể tham chiếu tới phần tử của mảng nhờ một chỉ số, ví dụ A(k). Cách này thường dùng để tham chiếu vec tơ hàng hay cột. Trong trường hợp ma trận đầy đủ thì nó được xem là ma trận một cột dài tạo từ các cột của ma trận ban đầu. Như vậy viết A(8) có nghĩa là tham chiếu phần tử A(4, 2).

c. Toán tử ":" : Toán tử ":" là một toán tử quan trọng của MATLAB. Nó xuất hiện ở nhiều dạng khác nhau. Biểu thức

1:10

là một vec tơ hàng chứa 10 số nguyên từ 1 đến 10

ans =

1 2 3 4 5 6 7 8 9 10

100:-7:50

tạo một dãy số từ 100 đến 51, giảm 7 mỗi lần

ans =

100 93 86 79 72 65 58 51

0: pi/4: pi

tạo một dãy số từ 0 đến pi, cách đều nhau pi/4

ans =

0 0.7854 1.5708 2.3562 3.1416

Các biểu thức chỉ số tham chiếu tới một phần của ma trận. Viết A(1:k,j) là

tham chiếu đến k phần tử đầu tiên của cột j.
Ngoài ra toán tử ":" tham chiếu tới tất cả các phần tử của một hàng hay một cột.

```
A(:,3)
ans =
    2
   11
    7
   14
```

và

```
A(3, :)
ans =
    9    6    7   12
```

Viết *B = A(:, [1 3 2 4])*
ta tạo được ma trận B từ ma trận A bằng cách đổi thứ tự các cột từ [1 2 3 4] thành [1 3 2 4]

```
B =
   16    2    3   13
    5   11   10    8
    9    7    6   12
    4   14   15    1
```

d. Tạo ma trận bằng hàm có sẵn: MATLAB cung cấp một số hàm để tạo các ma trận cơ bản:

zeros tạo ra ma trận mà các phần tử đều là zeros

```
z = zeros(2, 4)
z =
     0    0    0    0
     0    0    0    0
```

ones tạo ra ma trận mà các phần tử đều là 1

```
x = ones(2, 3)
x =
    1    1    1
    1    1    1

y = 5*ones(2, 2)
y =
```

5 5
5 5

rand tạo ra ma trận mà các phần tử ngẫu nhiên phân bố đều

d = rand(4, 4)

d =

0.9501 0.8913 0.8214 0.9218
0.2311 0.7621 0.4447 0.7382
0.6068 0.4565 0.6154 0.1763
0.4860 0.0185 0.7919 0.4057

randn tạo ra ma trận mà các phần tử ngẫu nhiên phân bố trực giao

e = randn(3, 3)

e =

- 0.4326 0.2877 1.1892
- 1.6656 -1.1465 -0.0376
0.1253 1.1909 0.3273

magic(n) tạo ra ma trận cấp n gồm các số nguyên từ 1 đến n^2 với tổng các hàng bằng tổng các cột.n phải lớn hơn hay bằng 3.

pascal(n) tạo ra ma trận xác định dương mà các phần tử lấy từ tam giác Pascal.

pascal(4)

ans =

1 1 1 1
1 2 3 4
1 3 6 10
1 4 10 20

eye(n) tạo ma trận đơn vị

eye(3)

ans =

1 0 0
0 1 0
0 0 1

eye(m,n) tạo ma trận đơn vị mở rông

eye(3,4)

ans =

1 0 0 0
0 1 0 0
0 0 1 0

e. Lệnh load: Lệnh load dùng để đọc một file dữ liệu. Vì vậy ta có thể tạo một file chứa ma trận và nạp vào. Ví dụ có file mtran.dat chứa một ma trận thì ta nạp ma trận này như sau:

load mtran.dat

Khi dùng một trình soạn thảo văn bản để tạo ma trận cần chú ý :

- file chứa ma trận là một bảng hình chữ nhật
- mỗi hàng viết trên một dòng
- số phần tử ở các hàng phải bằng nhau
- các phần tử phải cách nhau bằng dấu trống

f. M-file: M-file là một file text chứa các mã của MATLAB. Để tạo một ma trận ta viết một m-file và cho MATLAB đọc file này. Ví dụ ta tạo file ***ct1_1.m*** như sau

A = [
1 2 3
2 3 4
3 *4 5*]

và nạp vào MATLAB bằng cách đánh lệnh:

ct1_1

***g. Lắp ghép*: Ta** có thể lắp ghép (concatenation) các ma trận có sẵn thành một ma trận mới. Ví dụ:

a = ones(*3, 3*)
a =
1 1 1
1 1 1
1 1 1
*b = 5*ones*(*3, 3*)
b =
5 5 5
5 5 5
5 5 5
c = [*a+2*; *b*]
c =
3 3 3
3 3 3

3 3 3
5 5 5
5 5 5
5 5 5

h. Xoá hàng và cột: Ta có thể xoá hàng và cột từ ma trận bằng dùng dấu [].

Ví dụ:

b =

5 5 5
5 5 5
5 5 5

Để xoá cột thứ 2 ta viết:

b(:, *2*) = []

b =

5 5
5 5
5 5

Viết *x*(*1:2:5*) = [] nghĩa là ta xoá các phần tử bắt đầu từ đến phần tử thứ 5 và cách 2 rồi sắp xếp lại ma trận.

3. Các lệnh xử lí ma trận:

Cộng	: X= A + B
Trừ	: X= A - B
Nhân	: X= A * B
	: X.*A nhân các phần tử tương ứng với nhau
Chia	: X = A/B lúc đó X*B = A
	: X = A\B lúc đó A*X = B
	: X=A./B chia các phần tử tương ứng với nhau
Luỹ thừa	: X = A^2
	: X = A.^2
Nghịch đảo	: X = inv(A)
Định thức	: d = det(A)

§3. LẬP TRÌNH TRONG MATLAB

1. Các phát biểu điều kiện if, else, elseif:

Cú pháp của if:

if <biểu thức điều kiện>

<phát biểu>
end

Nếu <biểu thức điều kiện> cho kết quả đúng thì phần lệnh trong thân của if được thực hiện.

Các phát biểu else và leseif cũng tương tự.

Ví dụ: Ta xét chương trình ***ct1_2. m*** để đoán tuổi như sau:

```
disp('Xin chao! Han hanh duoc lam quen');
x = fix(30*rand);
disp('Tuoi toi trong khoang 0 - 30');
gu = input('Xin nhap tuoi cua ban: ');
if gu < x
    disp('Ban tre hon toi');
elseif gu > x
    disp('Ban lon hon toi');
else
    disp('Ban bang tuoi toi');
end
```

2. switch: Cú pháp của switch như sau :

switch <biểu thức>
<lệnh 1>
case n1 : <lệnh 1>
case n2 : <lệnh 2>
. . . . . .
case nn : <lệnh n>
otherwise : <lệnh n+1>
end

3. While: vòng lặp while dùng khi không biết trước số lần lặp. Cú pháp của nó như sau :

while <biểu thức>
<phát biểu>
end

Ví dụ: Xét chương trình in ra chuoi "Xin chao" lên mà hình với số lần nhập từ bàn phím (***ct1_3.m***) như sau:

```
disp('xin chao');
  gu = input('Nhap so lan in: ');
  i = 0;
```

```
while i~=gu
  disp(['Xin chao' i]);
  i = i+1
end
```

4. For: vòng lặp for dùng khi biết trước số lần lặp. Cú pháp như sau :

for <chỉ số> = <giá trị đầu> : <mức tăng> : <giá trị cuối>

Ví dụ: Xây dựng chương trình đoán số (***ct1_4.m***)

```
x = fix(100*rand);
n = 7;
t = 1;
for k = 1:7
  num = int2str(n);
  disp(['Ban co quyen du doan ',num,' lan']);
  disp('So can doan nam trong khoang 0 - 100');
  gu = input('Nhap so ma ban doan: ');
  if gu < x
    disp('Ban doan nho hon');
  elseif gu>x
    disp('So ban doan lon hon');
  else
    disp('Ban da doan dung.Xin chuc mung');
    t = 0;
    break;
  end
  n = n-1;
end
if t > 0
  disp('Ban khong doan ra roi');
  numx = int2str(x);
  disp(['Do la so: ',numx]);
end
```

5. Break: phát biểu break để kết thúc vòng lặp for hay while mà không quan tâm đến điều kiện kết thúc vòng lặp đã thoả mãn hay chưa.

§4. CÁC FILE VÀ HÀM

1. Script file: Kịch bản là M-file đơn giản nhất, không có đối số. Nó rất có ích khi thi hành một loạt lệnh MATLAB theo một trình tự nhất định. Ta xét ví dụ hàm fibno để tạo ra các số Fibonnaci.

```
f = [1  1];
i = 1;
while(f(i)+f(i+1))<1000
        f(i + 2 )= f(i)  +f(i  +1);
        i = i + 1;
end
plot(f)
```

Ta lưu đoạn mã lệnh này vào một file tên là ***ct1_5.m***. Đây chính là một script file. Để thực hiện các mã chứa trong file ***ct1_5.m*** từ cửa sổ lệnh ta nhập ct1_5 và nhấn enter.

2. File hàm: Hàm là M-file có chứa các đối số. Ta có một ví dụ về hàm :

```
function y = tb(x)
%Tinh tri trung binh cua cac phan tu
    [m,n ] = size(x);
    if m = = 1
            m = n;
    end
    y = sum(x)/m;
```

Từ ví dụ trên ta thấy một hàm M-file gồm các phần cơ bản sau :

- Một dòng định nghĩa hàm gồm: function y = tb(x) gồm từ khoá function, đối số trả về y, tên hàm tb và đối số vào x.
- Một dòng h1 là dòng trợ giúp đầu tiên. Vì đây là dòng văn bản nên nó phải đặt sau %. Nó xuất hiện ta nhập lệnh lookfor <tên hàm>
- Phần văn bản trợ giúp để giúp người dùng hiểu tác dụng của hàm.
- Thân hàm chứa mã MATLAB
- Các lời giải thích dùng để cho chương trình sáng rõ. Nó được đặt sau dấu %.

Cần chú ý là tên hàm phải bắt đầu bằng kí tự và cùng tên với file chứa hàm.
Từ cửa sổ MATLAB ta đánh lệnh:

```
z = 1:99;
tb(z)
```

Ghi chú: tên hàm là tb thì tên file cũng là ***tb.m***

Các biến khai báo trong một hàm của MATLAB là biến địa phương. Các hàm khác không nhìn thấy và sử dụng được biến này. Muốn các hàm khác dùng được biến nào đó của hàm ta cần khai báo nó là global. Ví dụ ta cần giải hệ phương trình :

$$\dot{y}_1 = y_1 - \alpha y_1 y_2$$

$$\dot{y}_2 = -y_2 + \beta y_1 y_2$$

Ta tạo ra M-file tên là ***ct1_6.m***

```
function yp = lotka(t,y)
global alpha beta
yp = [y(1) - alpha*y(1)*y(2);-y(2) + beta*y(1)*y(2)];
```

và sau đó từ dòng lệnh ta nhập các lệnh sau :

```
global alpha beta
alpha = 0.01;
beta = 0.02;
[t,y] = ode23('ct1_6',[0 10],[1 1]);
plot(t,y)
```

Để tiện dụng ta có thể lưu đoạn lệnh trên vào M-file ***ct1_7.m***.

Một biến có thể định nghĩa là persistent để giá trị của nó không thay đổi từ lần gọi này sang lần gọi khác. Các biến persistent chỉ có thể khai báo trong hàm. Chúng tồn tại trong bộ nhớ cho đến khi hàm bị xoá hay thay đổi.

3. Điều khiển vào và ra: Các lệnh sau dùng để số liệu đưa vào và ra

disp(a) hiển thị nội dung của mảng a hay văn bản

```
a = [1  2  3];
disp(a)
t ='Xin chao';
disp(t)
```

format điều khiển khuôn dạng số

Lệnh	Kết quả	Ví dụ
format	Default. Same as short.	
format short	5 digit scaled fixed point	3.1416
format long	15 digit scaled fixed point	3.14159265358979
format short e	5 digit floating point	3.1416e+00
format long e	15 digit floating point	3.141592653589793e+00
format short g	Best of 5 digit fixed or floating	3.1416

format long g	Best of 15 digit fixed or floating	3.14159265358979
format hex	Hexadecimal	400921fb54442d18
format bank	Fixed dollars and cents	3.14
format rat	Ratio of small integers	355/113
format +	+,-, blank	+
format compact	Suppresses excess line feeds	
format loose	Adds line feeds	

input nhập dữ liệu

x = input('Cho tri cua bien x :')

Cho tri cua bien x :4

x =

4

4. Các hàm toán học cơ bản:

exp(x) hàm e^x

sqrt(x) căn bậc hai của x

log(x) logarit tự nhiên

log10(x) logarit cơ số 10

abs(x) modun của số phức x

angle(x) argument của số phức a

conj(x) số phức liên hợp của x

imag(x) phần ảo của x

real(x) phần thực của x

sign(x) dấu của x

cos(x)

sin(x)

tan(x)

acos(x)

asin(x)

atan(x)

cosh(x)

coth(x)

sinh(x)

tanh(x)

acosh(x)
acoth(x)
asinh(x)
atanh(x)

5. Các phép toán trên hàm toán học:

a. Biểu diễn hàm toán học: MATLAB biểu diễn các hàm toán học bằng cách dùng các biểu thức đặt trong M-file. Ví dụ để khảo sát hàm :

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

ta tạo ra một file, đặt tên là ***humps.m*** có nội dung :

```
function y = humps(x)
y = 1./((x - 0.3).^2 + 0.01) + 1./((x - 0.9).^2 + 0.04) - 6 ;
```

Cách thứ hai để biểu diễn một hàm toán học trên dòng lệnh là tạo ra một đối tượng inline từ một biểu thức chuỗi. Ví dụ ta có thể nhập từ dòng lệnh hàm như sau :

```
f = inline('1./((x - 0.3).^2 + 0.01) + 1./((x - 0.9).^2 + 0.04 ) - 6');
```

ta có thể tính trị của hàm tại x = 2 như sau: f(2) và được kết quả là -4.8552

b. Vẽ đồ thị của hàm: Hàm fplot vẽ đồ thị hàm toán học giữa các giá trị đã cho.

Ví dụ :

```
fplot('humps',[-5  5 ])
grid on
```

c. Tìm cực tiểu của hàm: Cho một hàm toán học một biến, ta có thể dùng hàm fminbnd của MATLAB để tìm cực tiểu địa phương của hàm trong khoảng đã cho.

Ví dụ :

```
f = inline('1./((x-0.3).^2+0.01)+1./((x-0.9).^2+0.04)-6 ');
x = fminbnd(f,0.3,1)
x =
        0.6370
```

Hàm fminsearch tương tự hàm fminbnd dùng để tìm cực tiểu địa phương của hàm nhiều biến.

Ví dụ: Ta có hàm ***three_var.m***:

```
function b = three_var(v)
    x = v(1);
    y = v(2);
```

```
z = v(3);
b = x.^2 + 2.5*sin(y) - z^2*x^2*y^2;
```

và bây giờ tìm cực tiểu đối với hàm này bắt đầu từ x = -0.6 , y = -1.2 và z = 0.135

```
v = [-0.6 -1.2  0.135];
a = fminsearch('three_var',v)
a =
    0.0000   -1.5708    0.1803
```

d. Tìm điểm zero: Hàm fzero dùng để tìm điểm zero của hàm một biến. Ví dụ để tìm giá trị không của hàm lân cận giá trị -0.2 ta viết :

```
f = inline('1./((x-0.3).^2+0.01)+1./((x-0.9).^2+0.04)-6 ');
a = fzero(f,-0.2)
Zero found in the interval: [-0.10949, -0.264].
a =
   -0.1316
```

§5. ĐỒ HOẠ

1. Các lệnh vẽ: MATLAB cung cấp một loạt hàm để vẽ biểu diễn các vec tơ số liệu cũng như giải thích và in các đường cong này.

plot	đồ họa 2-D với số liệu 2 trục vô hướng và tuyến tính
plot3	đồ họa 3-D với số liệu 2 trục vô hướng và tuyến tính
loglog	đồ hoạ với các trục logarit
semilogx	đồ hoạ với trục x logarit và trục y tuyến tính
semilogy	đồ hoạ với trục y logarit và trục x tuyến tính
plotyy	đồ hoạ với trục y có nhãn ở bên trái và bên phải

2. Tạo hình vẽ: Hàm plot có các dạng khác nhau phụ thuộc vào các đối số đưa vào. Ví dụ nếu y là một vec tơ thì plot(y) tạo ra một đường thẳng quan hệ giữa các giá trị của y và chỉ số của nó. Nếu ta có 2 vec tơ x và y thì plot(x,y) tạo ra đồ thị quan hệ giữa x và y.

Ví dụ:

```
t = [0:pi/100:2*pi]
y = sin(t);
plot(t,y)
grid on
```

3. Đặc tả kiểu đường vẽ: Ta có thể dùng các kiểu đường vẽ khác nhau khi vẽ hình. Muốn thế ta chuyển kiểu đường vẽ cho hàm plot.

```
t = [0:pi/100:2*pi];
y = sin(t);
plot(t,y,'. ') % vẽ bằng đường chấm chấm
grid on
```

(lưu trong file ***ct1_8.m***)

4. Đặc tả màu và kích thước đường vẽ: Để đặc tả màu và kích thước đường vẽ ta dùng các tham số sau:

LineWidth	độ rộng đường thẳng, tính bằng số điểm
MarkerEdgeColor	màu của các cạnh của khối đánh dấu
MarkerFaceColor	màu của khối đánh dấu
MarkerSize	kích thước của khối đánh dấu

Màu được xác định bằng các tham số:

Mã	Màu	Mã	Màu
r	red	m	magenta
g	green	y	yellow
b	blue	k	black
c	cyan	w	white

Các dạng đường thẳng xác định bằng:

Mã	Kiểu đường
-	đường liền
--	đường đứt nét
:	đường chấm chấm
-.	đường chấm gạch

Các dạng điểm đánh dấu xác định bằng:

Mã	Kiểu đánh dấu	Mã	Kiểu đánh dấu
+	dấu cộng	.	điểm
o	vòng tròn	x	chữ thập
*	dấu sao	s	hình vuông
d	hạt kim cương	v	điểm tam giác hướng xuống
^	điểm tam giác hướng lên	<	tam giác sang trái
>	tam giác sang phải	h	lục giác
p	ngũ giác		

Ví dụ (lưu trong ***ct1_9.m*)**:

```
x = -pi : pi/10 : pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'--rs','LineWidth',2,'MarkerEdgeColor','k',...
    'MarkerFaceColor','g','MarkerSize',10)
```

sẽ vẽ đường cong y = f(x) có các đặc tả sau :

- đường vẽ là đường đứt nét(--)
- khối đánh dấu hình vuông (s), đường vẽ màu đỏ(r)
- đường vẽ rộng 2 point
- các cạnh của khối đánh màu đen
- khối đánh dấu màu green
- kích thước khối đánh dấu 10 point

5. Thêm đường vẽ vào đồ thị đã có: Để làm điều này ta dùng lệnh *hold*. Khi ta đánh lệnh *hold on* thì MATLAB không xoá đồ thị đang có. Nó thêm số liệu vào đồ thị mới này. Nếu phạm vi giá trị của đồ thị mới vượt quá các giá trị của trục toạ độ cũ thì nó sẽ định lại tỉ lệ xích.

6. Chỉ vẽ các điểm số liệu: Để vẽ các điểm đánh dấu mà không nối chúng lại với nhau ta dùng đặc tả nói rằng không có các đường nối giữa các điểm ta gọi hàm *plot* chỉ với đặc tả màu và điểm đánh dấu.

Ví dụ:

```
x = -pi : pi/10 : pi;
y = tan(sin(x)) - sin(tan(x));
plot(x,y,'s','MarkerEdgeColor','k')
```

(lưu trong ***ct1_10.m***)

7. Vẽ các điểm và đường: Để vẽ cả các điểm đánh dấu và đường nối giữa chúng ta cần mô tả kiểu đường và kiểu điểm.

Ví dụ (lưu trong ***ct1_11.m***):

```
x = 0:pi/15:4*pi;
y = exp(2*sin(x));
plot(x,y,'-r',x,y,'ok')
```

vẽ đường cong y = f(x). Đường nối liền, màu đỏ. Điểm đánh dấu chữ o có màu đen.

8. Vẽ với hai trục y: Lệnh *plotyy* cho phép tạo một đồ thị có hai trục y. Ta cũng

có thể dùng *plotyy* để cho giá trị trên hai trục y có kiểu khác nhau nhằm tiện so sánh.

Ví dụ:

```
t = 0:900;
A = 1000;
b = 0.005;
a = 0.005;
z2 = sin(b*t);
z1 = A*exp(-a*t);
[haxes, hline1, hline2] = plotyy(t,z1,t,z2,'semilogy','plot');
```

(lưu trong ***ct1_12.m***)

9. Vẽ đường cong với số liệu 3-D: Nếu x,y,z là 3 vec tơ có cùng độ dài thì *plot3* sẽ vẽ đường cong 3D.

Ví dụ:

```
t = 0:pi/50:10*pi;
plot3(sin(t),cos(t),t)
axis square;
grid on
```

(lưu trong ***ct1_13.m***)

10. Đặt các thông số cho trục: Khi ta tạo một hình vẽ, MATLAB tự động chọn các giới hạn trên trục toạ độ và khoảng cách đánh dấu dựa trên số liệu dùng để vẽ. Tuy nhiên ta có thể mô tả lại phạm vi giá trị trên trục và khoảng cách đánh dấu theo ý riêng. Ta có thể dùng các lệnh sau:

axis	đặt lại các giá trị trên trục toạ độ
axes	tạo một trục toạ độ mới với các đặc tính được mô tả
get và set	cho phép xác định và đặt các thuộc tính của trục toạ độ đang có
gca	trở về trục toạ độ cũ

a. Giới hạn của trục và chia vạch trên trục: MATLAB chọn các giới hạn trên trục toạ độ và khoảng cách đánh dấu dựa trên số liệu dùng để vẽ. Dùng lệnh *axis* có thể đặt lại giới hạn này. Cú pháp của lệnh:

```
axis[ xmin , xmax , ymin , ymax]
```

Ví dụ:

```
x = 0:0.025:pi/2;
plot(x,tan(x),'-ro')
```

```
axis([0 pi/2 0 5])
```
(lưu trong ***ct1_14.m***)

MATLAB chia vạch trên trục dựa trên phạm vi dữ liệu và chia đều. Ta có thể mô tả cách chia nhờ thông số xtick và ytick bằng một vec tơ tăng dần.

Ví dụ:

```
x = -pi:.1:pi;
y = sin(x);
plot(x,y)
set(gca,'xtick',-pi:pi/2:p);
set(gca,'xticklabel',{'-pi','-pi/2','0','pi/2','pi'})
```
(lưu trong ***ct1_15.m***)

8. Ghi nhãn lên các trục toạ độ: MATLAB cung cấp các lệnh ghi nhãn lên đồ hoạ gồm :

title	thêm nhãn vào đồ hoạ
xlabel	thêm nhãn vào trục x
ylabel	thêm nhãn vào trục y
zlabel	thêm nhãn vào trục z
legend	thêm chú giải vào đồ thị
text	hiển thị chuỗi văn bản ở vị trí nhất định
gtext	đặt văn bản lên đồ hoạ nhờ chuột
\bf	bold font
\it	italics font
\sl	oblique font (chữ nghiêng)
\rm	normal font

Các kí tự đặc biệt xem trong String properties.

Ta dùng các lệnh xlabel , ylabel , zlabel để thêm nhãn vào các trục toạ độ.

Ví dụ:

```
x = -pi:.1:pi;
y = sin(x);
plot(x,y)
xlabel('t = 0 to 2\pi','Fontsize',16)
ylabel('sin(t)','Fontsize',16)
title('\it{Gia tri cua sin tu zero đến 2 pi}','Fontsize',16)
```
(lưu trong ***ct1_16.m***)

9. Thêm văn bản vào đồ hoạ : Ta có thể thêm văn bản vào bất kì chỗ nào trên

hình vẽ nhờ hàm text .

Ví dụ:

*text(3*pi/4,sin(3*pi/4),'\leftarrowsin(t)=0.707','FontSize',12)*

10. Định vị văn bản trên hình vẽ: Ta có thể sử dụng đối tượng văn bản để ghi chú các trục ở vị trí bất kì. MATLAB định vị văn bản theo đơn vị dữ liệu trên trục. Ví dụ để vẽ hàm $y = Ae^{\alpha t}$ với A = 0.25 , t = 0 đến 900 và $\alpha = 0.005$ ta viết :

Ví dụ (lưu trong ***ct1_17.m***) :

```
t = 0:900;
plot(t,0.25*exp(-0.005*t))
```

Để thêm ghi chú tại điểm t = 300 ta viết :

```
text(300,.25*exp(-.005*300),...
 '\bullet\leftarrow\fontname{times}0.25{\ite}^(-0.005{\itt}} at,...
{\itt}=300','FontSize',14)
```

Tham số HorizontalAlignment và VerticalAlignment định vị văn bản so với các toạ độ x, y, z đã cho.

11. Đồ hoạ đặc biệt:

a. Khối và vùng: Đồ hoạ khối và vùng biểu diễn số liệu là vec tơ hay ma trận. MATLAB cung cấp các hàm đồ hoạ khối và vùng :

bar hiển thị các cột của ma trận m*n như là m nhóm, mỗi nhóm có n bar

barh hiển thị các cột của ma trận m*n như là m nhóm, mỗi nhóm có n bar nằm ngang

bar3 hiển thị các cột của ma trận m*n như là m nhóm, mỗi nhóm có n bar dạng 3D

bar3h hiển thị các cột của ma trận m*n như là m nhóm, mỗi nhóm có n bar dạng 3D nằm ngang

Mặc định, mỗi phần tử của ma trận được biểu diễn bằng một bar.

Ví dụ:

```
y =   [5  2  1
       6  7  3
       8  6  3
       5  5  5
       1  5  8];
bar(y)
```

(lưu trong ***ct_18.m***). Sau đó nhập lệnh bar3(y) ta có đồ thị 3D.

b. Mô tả dữ liệu trên trục: Ta dùng các hàm *xlabel* và *ylabel* để mô tả các dữ liệu trên trục.

Ví dụ:

```
nhdo = [29 23 27 25 20 23 23 27];
ngay = 0:5:35;
bar(ngay,nhdo)
xlabel('ngay')
ylabel('Nhiet do (^{o}C)')
```

(lưu trong ***ct1_19.m***)

Mặc định,phạm vi giá trị của trục y là từ 0 đến 30. Để xem nhiệt độ trong khoảng từ 15 đến 30 ta thay đổi phạm vi giá trị của trục y

```
set(gca,'YLim',[15 30],'Layer','top')
```

và trên đồ thị, phạm vi giá trị của trục y đã thay đổi.

c. Xếp chồng đồ thị: Ta có thể xếp chồng số liệu trên đồ thị thanh bằng cách tạo ra một trục khác trên cùng một vị trí và như vậy ta có một trục y độc lập với bộ số liệu khác.

Ví dụ: Khảo sát nhịp độ sinh học liên quan đến mật độ trichloetylene(TCE) cho số liệu:

```
TCE = [515 420 370 250 135 120 60 20];
nhdo = [29 23 27 25 20 23 23 27];
ngay = 0:5:35;
bar(ngay,nhdo)
xlabel('Ngay')
ylabel('Nhiet do (^{o}C)')
```

Để xếp chồng một số liệu lên một đồ thị thanh ở trên, có trục thứ 2 ở cùng vị trí như trục thứ nhất ta viết :

```
h1 = gca;
```

và tạo trục thứ 2 ở vị trí trục thứ nhất trước nhất vẽ bộ số liệu thứ 2

```
h2 = axes('Position',get(h1,'Position'));
plot(days,TCE,'LineWidth',3)
```

Để trục thứ 2 không gây trở ngại cho trục thứ nhất ta viết :

```
set(h2,'YAxisLocation','right','Color','none','XTickLabel',[])
set(h2,'XLim',get(h1,'XLim'),'Layer','top')
```

Để ghi chú lên đồ thị ta viết:

```
text(11,380,'Mat do','Rotation',--55,'FontSize',16)
ylabel('TCE Mat do (PPM)')
title('Xep chong do thi','FontSize',16)
```

(lưu trong ***ct1_20.m***)

d. ***Đồ hoạ vùng***: Hàm *area* hiển thị đường cong tạo từ một vec tơ hay từ một cột của ma trận. Nó vẽ các giá trị của một cột của ma trận thành một đường cong riêng và tô đầy vùng không gian giữa các đường cong và trục x.

Ví dụ *(lưu trong* **ct1_21.m***)*:

```
Y =  [5 1 2
      8 3 7
      9 6 8
      5 5 5
      4 2 3];
area(Y)
```

hiển thị đồ thị có 3 vùng, mỗi vùng một cột. Độ cao của mỗi đồ thị vùng là tổng các phần tử trong một hàng. Mỗi đường cong sau sử dụng đường cong trước làm cơ sở. Để hiển thị đường chia lưới ta dùng lệnh:

```
set(gca,'Layer','top')
set(gca,'XTick',1:5)
grid on
```

f. Đồ thị pie :Đồ thị pie hiển thị theo tỉ lệ phần trăm của một phần tử của một vec tơ hay một ma trận so với tổng các phần tử . *pie* và *pie3* tạo ra đồ thị 2D và 3D.

Ví dụ (lưu trong ***ct1_22.m***):

```
X =  [19.3   22.1   51.6;
      34.2   70.3   82.4;
      61.4   82.9   90.8;
      50.5   54.9   59.1;
      29.4   36.3   47.0];
x = sum(X);
explode = zeros(size(x));
[c,offset] = max(x);
explode(offset) = 1;
h = pie(x,explode)
```

Khi tổng các phần tử trong đối số thứ nhất bằng hay lớn hơn 1, pie và pie3 chuẩn hoá các giá trị. Như vậy cho vec tơ x, mỗi phần có diện tích $x_i / \text{sum}(x_i)$ với x_i là một phần tử của x. Giá trị được chuẩn hoá mô tả phần nguyên của mỗi vùng. Khi tổng các phần tử trong đối số thứ nhất nhỏ hơn 1, pie và pie3 không chuẩn hoá các phần tử của vec tơ x. Chúng vẽ một phần pie.

Ví dụ:

```
x = [.19 .22 .41];
pie(x)
```

g. Làm hình chuyển động: Ta có thể tạo ra hình chuyển động bằng 2 cách

- tạo và lưu nhiều hình khác nhau và lần lượt hiển thị chúng
- vẽ và xoá liên tục một đối tượng trên màn hình,mỗi lần vẽ lại có sự thay đổi.

Với cách thứ nhất ta thực hiện hình chuyển động qua 3 bước:

- dùng hàm *moviein* để dành bộ nhớ cho một ma trận đủ lớn nhằm lưu các khung hình.
- dùng hàm *getframes* để tạo các khung hình.
- dùng hàm *movie* để hiển thị các khung hình.

Sau đây là ví dụ sử dụng movie để quan sát hàm fft(eye(n)).Ta tạo hàm ***ct1_23.m*** như sau :

```
axis equal
M = moviein(16,gcf);
set(gca,'NextPlot','replacechildren')
h = uicontrol('style','slider','position',[100 10 500 20],'Min',1,'Max',16)
for j = 1:16
        plot(fft(eye(j + 16)))
        set(h,'Value',j)
M(:,j) = getframe(gcf);
end
clf;
axes('Position',[0 0 1 1]);
movie(M,30)
```

Bước đầu tiên để tạo hình ảnh chuyển động là khởi gán ma trận. Tuy nhiên trước khi gọi hàm *moviein*, ta cần tạo ra các trục toạ độ có cùng kích thước với kích thước mà ta muốn hiển thị hình. Do trong ví dụ này ta hiển thị các số liệu cách đều trên vòng tròn đơn vị nên ta dùng lệnh *axis equal* để xác định tỉ lệ các trục. Hàm *moviein* tạo ra ma trận đủ lớn để chứa 16 khung hình. Phát biểu :

```
set(gca,'NextPlot','replacechildren')
```

ngăn hàm *plot* đưa tỉ lệ các trục về *axis normal* mỗi khi nó được gọi. Hàm *getframe* không đối số trả lại các điểm ảnh của trục hiện hành ở hình hiện có. Mỗi khung hình gồm các số liệu trong một vec tơ cột. Hàm *getframe(gcf)* chụp toàn bộ phần trong của một cửa sổ hiện hành. Sau khi tạo ra hình ảnh ta có thể chạy chúng một số lần nhất định ví dụ 30 lần nhờ hàm *movie(M,30)* .

Một phương pháp nữa để tạo hình chuyển động là vẽ và xoá, nghĩa là vẽ một đối tượng đồ hoạ rồi thay đổi vị trí của nó bằng cách thay đổi toạ độ x,y và z một lượng nhỏ nhờ một vòng lặp. Ta có thể tạo ra các hiệu ứng khác nhau nhờ các cách xoá hình khác nhau. Chúng gồm:

- none MATLAB không xoá đối tượng khi nó di chuyển
- background MATLAB xoá đối tượng bằng cách vẽ nó có màu nền
- xor MATLAB chỉ xoá đối tượng

Ví dụ: Ta tạo ra M-file có tên là ***ct1_24.m*** như sau :

```
A = [ -8/3 0 0; 0 -10 10; 0 28 -1 ];
y = [35 -10 -7]';
h = 0.01;
p = plot3(y(1),y(2),y(3),'.', ...
'EraseMode','none','MarkerSize',5); % dat EraseMode ve none
axis([0 50 -25 25 -25 25])
hold on
for i = 1:4000
    A(1,3) = y(2);
    A(3,1) = -y(2);
    ydot = A*y;
    y = y + h*ydot;
    set(p,'XData',y(1),'YData',y(2),'ZData',y(3)) % thay doi toa do
    drawnow
    i = i + 1;
end
```

12. Đồ hoạ 3D:

a.Các lệnh cơ bản: Lệnh *mesh* và *surf* tạo ra mặt 3D từ ma trận số liệu. Gọi ma trận số liệu là z mà mỗi phần tử của nó z(i,j) xác định tung độ của mặt thì *mesh(z)* tạo ra một lưới có màu thể hiện mặt z còn *surf(z)* tạo ra một mặt có màu z.

b. Đồ thị các hàm hai biến: Bước thứ nhất để thể hiện hàm 2 biến z=f(x,y) là tạo ma trận x và y chứa các toạ độ trong miền xác định của hàm. Hàm meshgrid sẽ biến đổi vùng xác định bởi 2 vec tơ x và y thành ma trận x và y. Sau đó ta dùng ma trận này để đánh giá hàm.

Ví dụ: Ta khảo sát hàm sin(r)/r. Để tính hàm trong khoảng -8 và 8 theo x và y ta chỉ cần chuyển một vec tơ đối số cho *meshgrid* :

[x,y] = meshgrid(-8:.5:8);

r = sqrt(x.^2 + y.^2) + 0.005;

ma trận r chứa khoảng cách từ tâm của ma trận. Tiếp theo ta dùng hàm mesh để vẽ hàm.

z = sin(r)./r;

mesh(z)

(lưu trong ***ct1_25.m***)

c.Đồ thị đường đẳng mức: Các hàm contour tạo, hiển thị và ghi chú các đường đẳng mức của một hay nhiều ma trận. Chúng gồm:

clabel tạo các nhãn sử dụng ma trận contour và hiển thị nhãn

contour hiển thị các đường đẳng mức tạo bởi một giá trị cho trước của ma trận Z.

contour3 hiển thị các mặt đẳng mức tạo bởi một giá trị cho trước của ma trận Z.

contourf hiển thị đồ thị contour 2D và tô màu vùng giữa 2 các đường

contourc hàm cấp thấp để tính ma trận contour

Hàm meshc hiển thị contour và lưới và surfc hiển thị mặt contour.

Ví dụ:

[X,Y,Z] = peaks;

contour(X,Y,Z,20)

Mỗi contour có một giá trị gắn với nó. Hàm *clabel* dùng giá trị này để hiển thị nhãn đường đồng mức 2D. Ma trận contour chứa giá trị clabel dùng cho các đường contour 2D. Ma trận này được xác định bởi *contour,contour3* và *contourf*.

Ví dụ: Để hiển thị 10 đường đẳng mức của hàm peak ta viết :

Z = peaks;

[C,h] = contour(Z,10);

clabel(C,h)

title({'Cac contour co nhan','clabel(C,h)'})

(lưu trong ***ct1_26.m***)

Hàm *contourf* hiển thị đồ thị đường đẳng mức trên một mặt phẳng và tô màu vùng còn lại giữa các đường đẳng mức. Để kiểm soát màu tô ta dùng hàm *caxis*.

Ví dụ (lưu trong ***ct1_27.m***):

Z = peaks;

[C,h] = contourf(Z,10);

caxis([-20 20])

title({'Contour co to mau','contourf(Z,10)'})

Các hàm contour(z,n) và contour(z,v) cho phép ta chỉ rõ số lượng mức contour hay một mức contour cần vẽ nào đó với z là ma trận số liệu, n là số đường contour và v là vec tơ các mức contour. MATLAB không phân biệt giữa đại lượng vec tơ một phần tử hay đại lượng vô hướng. Như vậy nếu v là vec tơ một phần tử mô tả một contour đơn ở một mức hàm *contour* sẽ coi nó là số lượng đường contour chứ không phải là mức contour. Như vậy,*contour(z,v)* cũng như *contour(z,n)*. Để hiển thị một đường đẳng mức ta cần cho v là một vec tơ có 2 phần tử với cả hai phần tử bằng mức mong muốn.V í dụ để tạo ra một đường đẳng mức 3D của hàm peaks

Ví dụ (lưu trong ***ct1_28.m***):

```
xrange = -3:.125:3;
yrange = xrange;
[X,Y] = meshgrid(xrange,yrange);
Z = peaks(X,Y);
contour3(X,Y,Z)
```

Để hiển thị một mức ở Z = 1, ta cho v là [1 1]

```
v = [1 1]
contour3(X,Y,Z,v)
```

Hàm *ginput* cho phép ta dùng chuột hay các phím mũi tên để chọn các điểm vẽ. Nó trả về toạ độ của vị trí con trỏ. Ví dụ sau sẽ minh hoạ các dùng hàm *ginput* và hàm spline để tạo ra đường cong nội suy hai biến.

Ví dụ: Ta tạo một M-file có tên ***ct1_29.m*** như sau :

```
disp('Chuot phai tro cac diem tren duong ve')
disp('Chuot trai tro diem cuoi cua duong ve')
axis([0 10 0 10])
hold on
x = [];
y  =[];
n = 0;
but = 1;
while but = =1
        [xi,yi,but] = ginput(1);
        plot(xi,yi,'go')
        n = n  +1;
        x(n,1) = xi;
        y(n,1) = yi;
end
```

```
t = 1:n;
ts = 1:0.1:n;
xs = spline(t,x,ts);
ys = spline(t,y,ts);
plot(xs,ys,'c-');
hold off
```

13. Vẽ các vectơ: Có nhiều hàm MATLAB dùng hiển thị các vec tơ có hướng và vec tơ vận tốc. Ta định nghĩa một vec tơ bàng cách dùng một hay 2 đối số. Các đối số mô tả thành phần x và thành phần y của vec tơ. Nếu ta dùng 2 đối số thì đối số thứ nhất sẽ mô tả thành phần x và đối số thứ ha mô tả thành phần y. Nếu ta chỉ dùng một đối số thì MATLAB xử lí nó như một số phức, phần thực là thành phần x và phần ảo là thành phần y.

Các hàm vẽ vec tơ gồm:

compass	vẽ các véc tơ bắt đầu từ gốc toạ độ của hệ toạ độ cực
feather	vẽ các vec tơ bắt đầu từ một đường thẳng
quiver	vẽ các vec tơ 2D có các thành phần (u, v)
quiver3	vẽ các vec tơ 3D có các thành phần (u, v, w)

a. Hàm compass: Ta xét ví dụ vẽ hướng và tốc độ gió. Các vec tơ xác định hướng (góc tính bằng độ) và tốc độ gió (km/h) là:

```
hg = [45 90 90 45 360 335 360 270 335 270 335 335];
td = [6 6 8 6 3 9 6 8 9 10 14 12];
```

Ta biến đổi hướng gió thành radian trước khi biến đổi nó thành toạ độ vuông góc.

```
hg1 = hg * pi/180;
[x,y] = pol2cart(hg1,td);
compass(x,y)
```

và tạo ra ghi chú trên đồ thị:

```
gc = {'Huong gio và suc gio tai san bay Da Nang')
text(–28,15,gc)
```

(lưu trong ***ct1_30.m***)

b. Hàm feather: Hàm feather hiển thị các vec từ bắt đầu từ một đường thẳng song song với trục x. Ví dụ để tạo ra các vec tơ có góc từ 90^0 đến 0^0 và cùng độ dài ta có:

```
theta = 90:–10:0;
r = ones(size(theta));
```

trước khi vẽ, chuyển các số liệu sang toạ độ vuông góc và tăng độ lớn thành r để dễ nhìn(lưu trong ***c1_31.m***):

```
[u,v] = pol2cart(theta*pi/180,r*10);
feather(u,v)
axis equal
```

Nếu đối số là số phức z thì feather coi phần thực là x và phần ảo là y (lưu trong ***ct1_32.m***):

```
t = 0:0.3:10;
s = 0.05 + i;
Z = exp(–s*t);
feather(Z)
```

c. Hàm quiver: Hàm quiver hiển thị các vec tơ ở các điểm đã cho trong mặt phẳng. Các vec tơ này được xác định bằng các thành phần x và y. Ví dụ để tạo ra 10 contour của hàm peaks ta viết(lưu trong ***ct1_33.m***):

```
n = –2.0:.2:2.0;
[X,Y,Z] = peaks(n);
contour(X,Y,Z,10)
```

Bây giờ dùng hàm gradient để tạo các thành phần của vec tơ dùng làm đối số cho quiver:

```
[U,V] = gradient(Z,.2);
```

Đặt hold on để thêm đường contour:

```
hold on
quiver(X,Y,U,V)
hold off
```

d. Hàm quiver3: Hàm quiver3 hiển thị các vec tơ có các thành phần (u,v,w) tại điểm (x, y, z). Ví dụ ta biểu diễn quỹ đạo của một vật được ném đi theo t. Phương trình của chuyển động là:

$$z(t) = v_0 t + \frac{at^2}{2}$$

Trước hết ta gán vận tốc ban đầu và gia tốc a:

```
v0 = 10; % Van toc ban dau
a = –32; % gia toc
```

Tiếp theo tính z tại các thời điểm:

```
t = 0:.1:1;
z = vz*t + 1/2*a*t.^2;
```

Tính vị trí theo hướng x và y:

```
vx = 2;
```

```
x = vx*t;
vy = 3;
y = vy*t;
```

Tính các thành phần của vec tơ vận tốc và hiển thị bằng các dùng quiver3:

```
u = gradient(x);
v = gradient(y);
w = gradient(z);
scale = 0;
quiver3(x,y,z,u,v,w,scale)
axis square
```

(lưu trong ***ct1_34.m***)