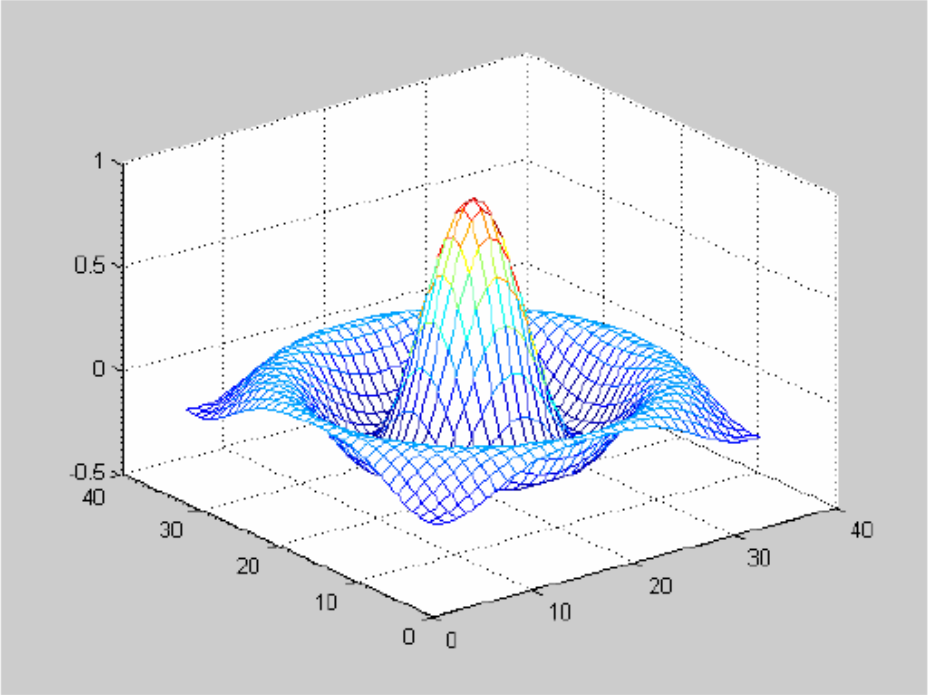
ĐẠI HỌC ĐÀ NẴNG
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA CÔNG NGHỆ THÔNG TIN

GIÁO TRÌNH

MATLAB



Dành cho sinh viên các ngành kỹ thuật

Tác giả : PHAN - THANH - TAO

Đà Nẵng - 2004

GIỚI THIỆU

Matlab *là một phần mềm toán học của hãng* **Mathworks** *để tính toán trên các số và có tính trực quan rất cao.*

Matlab *đã qua nhiều phiên bản, giáo trình này giới thiệu phiên bản 7.0 (release 14).*

Matlab *là viết tắt của* Matrix Laboratory. **Matlab** *làm việc chủ yếu với các ma trận. Ma trận cỡ* mxn *là bảng số chữ nhật gồm* mxn *số được sắp xếp thành* m *hàng và* n *cột. Trường hợp* m=1 *hoặc* n=1 *thì ma trận trở thành vectơ dòng hoặc cột; trường hợp* m=n=1 *thì ma trận trở thành một đại lượng vô hướng. Nói chung,* **Matlab** *có thể làm việc với nhiều kiểu dữ liệu khác nhau. Với xâu chữ (chuỗi ký tự)* **Matlab** *cũng xem là một dãy các ký tự hay là dãy mã số của các ký tự.*

Matlab *dùng để giải quyết các bài toán về giải tích số, xử lý tín hiệu số, xử lý đồ họa, ... mà không phải lập trình cổ điển.*

Hiện nay, **Matlab** *có đến hàng ngàn lệnh và hàm tiện ích. Ngoài các hàm cài sẵn trong chính ngôn ngữ,* ***Matlab*** *còn có các lệnh và hàm ứng dụng chuyên biệt trong các* Toolbox*, để mở rộng môi trường* **Matlab** *nhằm giải quyết các bài toán thuộc các phạm trù riêng. Các* Toolbox *khá quan trọng và tiện ích cho người dùng như* toán sơ cấp, xử lý tín hiệu số, xử lý ảnh, xử lý âm thanh, ma trận thưa, logic mờ,…

Người dùng cũng có thể tạo nên các hàm phục vụ cho chuyên môn của mình, lưu vào tệp M-file *để dùng về sau.*

Cần tính toán bằng công thức thì có thể dùng Toolbox SYMBOLIC. *Để có được* f='cos(x)' *bằng cách lấy đạo hàm của* g='sin(x)' *thì dùng lệnh* f=diff('sin(x)'). *Ngược lại để có* g *là tích phân bất định của* f *thì dùng lệnh* g=int(f).

Matlab *còn có giao diện đồ họa khá đẹp mắt và dể sử dụng. Người dùng có thể tính toán và tạo nên các hình ảnh đồ họa 2, 3 chiều cho trình ứng dụng của mình. Với các hình ảnh, nếu không chỉ định vè canh trục, phối màu thì* **Matlab** *thực hiện tự động một cách khá phù hợp.*

Vì tính mạnh mẽ để trợ giúp giải nhanh các bài toán kỹ thuật, chúng tôi cố gắng biên soạn tài liệu này để phục vụ một ít kiến thức cơ bản cho bạn đọc. Tuy nhiên, trên cơ sở đó bạn đọc có thể tự khai thác thêm các thành phần dùng riêng cho minh trong các Toolbox *và* Simulink.

Lần đầu xuất bản nên không thể tránh khỏi thiếu sót. Rất mong ý kiến đóng góp quý báu của bạn đọc.

Đà Nẵng, ngày 20/02/2004

Tác giả

Phan Thanh Tao

Chương 1. CÁC KHÁI NIỆM CƠ BẢN

MATLAB chỉ làm việc chủ yếu với các loại đối tượng là ma trận số có thể là số phức. Trong trường hợp đặc biệt, có thể là ma trận cấp **1** là các vô hướng, và các *ma trận dòng* hoặc *ma trận cột* là các vectơ.

Hãy bắt đầu với cách nhập ma trận cho **MATLAB.**

1.1. Nhập ma trận đơn giản

Ma trận có thể nhập cho **MATLAB** bằng nhiều cách:

- Nhập danh sách rõ ràng các phần tử.
- Phát sinh bằng các lệnh và hàm gắn liền.
- Tạo ra từ siêu tệp (M-file).
- Nạp từ các tệp dữ liệu bên ngoài.

Ngôn ngữ **MATLAB** không chứa các lệnh khai báo kích thước hoặc khai báo kiểu. Việc lưu trữ là tự động.

Cách dễ nhất của việc nhập ma trận là nhập danh sách rõ ràng các phần tử. Danh sách các phần tử cách nhau ký tự trống hoặc dấu phẩy, đặt trong cặp ngoặc vuông, **[** và **],** và dùng dấu chấm phẩy(**;**) để biểu hiện kết thúc dòng. Ví dụ, nhập lệnh

```
A = [ 1 2 3; 4 5 6; 7 8 9 ]
```

kết quả xuất là

```
A =
    1   2   3
    4   5   6
    7   8   9
```

Ma trận **A** được lưu để sử dụng về sau .

Ma trận lớn có thể được tách ra thành nhiều dòng, sang dòng thay cho dấu chấm phẩy. Mặc dù ít cần ma trận kích thước này, nhưng ma trận trên cũng có thể tách ra thành **3** dòng nhập như sau

```
A = [ 1   2   3
      4   5   6
      7   8   9 ]
```

Các ma trận có thể nhập từ tệp với tên mở rộng là "**.m**" . Nếu tệp có tên là **gena.m** chứa ba dòng văn bản

```
A = [ 1   2   3
      4   5   6
      7   8   9 ]
```

thì lệnh **gena** đọc tệp và phát sinh ra ma trận **A**.

Lệnh **load** có thể đọc các ma trận phát sinh từ các phần khác trước đó của **MATLAB** hoặc các ma trận ở dạng **ASCII** xuất từ các chương trình khác. Sẽ biết thêm sau này.

1.2. Các phần tử của ma trận

Các phần tử của ma trận có thể là biểu thức **MATLAB** bất kỳ; ví dụ, lệnh

```
x = [ -1.3  sqrt(3)  (1+2+3)*4/5 ]
```

kết quả là

```
x =
   -1.3000  1.7321  4.8000
```

Các phần tử riêng biệt của ma trận có thể được tham chiếu với các chỉ số bên trong cặp ngoặc đơn, **(** và **).** Tiếp ví dụ trên, lệnh

```
x(5) = abs(x(1))
```

cho ra

```
x =
   -1.3000  1.7321  4.8000  0.0000 1.3000
```

Lưu ý rằng kích thước của **x** được tự động tăng để phù hợp với các phần tử mới, và các phần tử trong khoảng không xác định được đặt giá trị *không*.

Ma trận lớn có thể được xây dựng bằng cách dùng các ma trận nhỏ như các phần tử. Ví dụ, có thể đưa thêm một dòng khác vào ma trận **A** với lệnh

```
r = [ 10  11  12 ];
A = [ A ; r ]
```

kết quả là

```
A =
   1   2   3
   4   5   6
   7   8   9
   10  11  12
```

Các ma trận nhỏ có thể được trích ra từ các ma trận lớn bằng cách dùng dấu hai chấm, **:** . Ví dụ, lệnh

A = A(1:3,:);

lấy ba dòng đầu và tất cả các cột của ma trận **A** hiện thời để đưa ma trận **A** về giá trị ban đầu. Sẽ biết thêm về dấu hai chấm sau này.

1.3. Câu lệnh và biến

MATLAB là **ngôn ngữ biểu thức**. Các biểu thức được đánh vào bởi người dùng, được thông dịch và ước lượng bởi hệ **MATLAB.** Các lệnh **MATLAB** thường có dạng:

variable = expression

hoặc đơn giản

expression

variable: tên biến,
expression: biểu thức.

Các biểu thức được cấu thành từ các toán tử và các ký tự đặc biệt khác, từ các hàm, và từ các tên biến. Việc ước lượng các biểu thức cho ra một ma trận, sau đó hiển thị trên màn hình và gán vào biến để sử dụng về sau. Nếu tên biến và dấu = bị bỏ qua thì một biến có tên là **ans,** viết tắt chữ "**answer**" (trả lời), được tự động tạo ra. Ví dụ, đánh vào

1900/81

cho ra **ans =**

23.4568

Một câu lệnh được kết thúc bình thường với ký tự sang dòng hay phím <**Enter**>. Tuy nhiên, nếu ký tự cuối cùng của câu lệnh là dấu chấm phẩy thì việc in ra kết quả được hủy, nhưng lệnh vẫn được thực hiện. Điều này là hữu ích trong các siêu tệp M-file (biết thêm sau này) và trong trường hợp kết quả đủ lớn không cần quan tâm từng số. Ví dụ, lệnh

p = conv(r,r);

tích chập các số trong **r** với chính chúng nhưng không hiển thị kết quả.

Nếu biểu thức quá phức tạp để câu lệnh không thể đặt gọn trên một dòng thì có thể dùng dấu tĩnh lược (**...**) tiếp theo là ký tự sang dòng để biểu hiện câu lệnh được tiếp tục trên dòng tiếp theo. Ví dụ

```
s = 1 - 1/2 + 1/3 - 1/4 + 1/5 - 1/6 + 1/7 ...
    - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```

tính tổng riêng của chuỗi điều hòa, gán tổng vào biến **s**, nhưng không in ra gì cả. Các ký tự trống quanh các dấu +, - là tùy chọn nhưng được đưa vào đây để dễ đọc.

Các **tên biến**, **tên hàm** được định dạng bằng một chữ viết, tiếp theo là số bất kỳ các chữ viết và chữ số (hoặc dấu nối). Chỉ có **19** ký tự đầu được nhớ.

MATLAB là ngôn ngữ nhạy cảm; nó thường phân biệt chữ hoa/chữ thường, bởi vậy **a** và **A** không phải là tên của cùng một biến. Tất cả các tên hàm phải là chữ thường; lệnh **inv(A)** sẽ lấy ngịch đảo của ma trận **A**, nhưng lệnh **INV(A)** tham chiếu đến một hàm không được định nghĩa: Tuy nhiên, lệnh **casesen** làm cho **MATLAB** không phân biệt chữ hoa/chữ thường. Trong chế độ này **INV(a)** là lấy ma trận đảo của nó.

1.4. Cách lấy thông tin vùng làm việc

Các lệnh trong các ví dụ cho đến bây giờ tạo ra các biến được lưu trong vùng làm việc của **MATLAB.** Thực hiện lệnh

```
who
```

liệt kê các biến trong vùng làm việc:

```
your variables are:

A   ans   p   r   s   x

leaving 291636 bytes of memory free
```

ở đây trình bày **6** biến phát sinh bởi các ví dụ, kể cả biến **ans.** Để biết thêm chi tiết về kích thước của mỗi biến hiện thời, dùng lệnh **whos**, cũng với ví dụ, cho ra

```
Name      size          total         Complex

  A       3 by 3        9             No
 ans      1 by 1        1             No
  p       1 by 5        5             No
  r       1 by 3        3             No
  s       1 by 1        1             No
  x       1 by 5        5             No
```

Grand total is (24*8) = 192 bytes,

leaving 291636 bytes of memory free.

Mỗi phần tử của ma trận thực đòi hỏi **8** byte bộ nhớ, bởi vậy ma trận **A** cấp **3** dùng **72** byte và tất cả các biến dùng tổng cọng **192** byte. Tổng số không gian bộ nhớ tự do còn lại phụ thuộc vào từng loại máy khác nhau.

Biến **ans** cùng với một biến không liệt kê **eps** có ý nghĩa đặc biệt với **MATLAB.** Chúng là các biến cố định không thể xóa.

Biến **eps** (epsilon) dùng để xác định những giá trị gần kỳ dị (suy biến) và hạng ma trận. Giá trị khởi tạo của nó là khoảng cách từ **1.0** đến số thập phân lớn nhất tiếp theo. Đối với kỹ thuật số học **IEEE** (**I**nstitute of **E**lectrical and **E**lectronic **E**ngineers) dùng trên các máy cá nhân và các máy trạm, thì

$$\mathbf{eps = 2^{-52}}$$

khoảng $\mathbf{2.22 \times 10^{-16}}$. **eps** có thể được đặt lại với giá trị khác, kể cả giá trị **0**.

1.5. Số và biểu thức số

Các số dùng ký pháp thập phân qui ước với dấu chấm và dấu trừ đứng trước là tùy chọn. Có thể đưa vào cuối dạng khoa học (lũy thừa 10). Sau đây là vài ví dụ về các số hợp pháp:

3	**-99**	**0.0001**
9.6397238	**1.6040E-10**	**6.022252e23**

Trên các máy dùng kỹ thuật số học chấm động **IEEE** thì độ chính xác tương đối của các số là **eps,** khoảng **16** chữ số có nghĩa. Miền giá trị khoảng $\mathbf{10^{-308}}$ đến $\mathbf{10^{308}}$.

Các biểu thức có thể được tạo ra bằng cách dùng các phép toán số học thông thường và các qui tắc ưu tiên:

+	cộng
-	trừ
*	nhân
/	chia phải
\	chia trái
^	lũy thừa

Các phép toán trên ma trận để cho tiện có hai ký hiệu cho phép chia. Các biểu thức vô hướng **1/4** và **4\1** có cùng giá trị số, chính là **0.25**. Các cặp ngoặc đơn được dùng theo cách thông thường để xen vào việc ưu tiên của các phép toán số học.

Hầu hết các hàm toán sơ cấp thông thường trên các tính toán khoa học là các *hàm cài sẵn* của **MATLAB,** như **abs, sqrt, log,** và **sin, ...** Có thể thêm vào các hàm một cách dễ dàng với các siêu tệp M-file. Phần sau có một danh sách khá đầy đủ các hàm.

Một số các *hàm cài sẵn* đơn giản trả về các giá trị đặc biệt thường dùng. Hàm **pi** trả về số **π,** chương trình tính trước, đó là **4*atan(1)**. Một cách gọi khác để phát sinh số **π** là

imag(log(-1))

Hàm **inf**, viết tắt chữ **infinity** (vô định), được thấy trên rất ít hệ tính toán hoặc ngôn ngữ lập trình. Trên một số máy, nó được tạo ra bởi kỹ thuật số học **IEEE** cài trong bộ đồng xử lý toán học (*coprocessor*). Trên các máy khác, phần mềm chấm động được đưa vào để mô phỏng đồng xử lý toán học. Một cách để phát sinh giá trị trả về bởi hàm **inf** là

s = 1/0

kết quả là **s =**

∞

Warning: Divide by zero.

Trên các máy với kỹ thuật số học **IEEE**, việc chia cho số không không dẫn đến điều kiện lỗi hoặc kết thúc hoạt động. Cho ra một thông báo khuyến cáo và một giá trị đặc biệt có thể xử lý trong việc tính toán sau đó.

Biến **NaN** là một số **IEEE** quan hệ với hàm **inf**, nhưng có các đặc tính khác. Nó là viết tắt chữ **"Not a Number"** (không phải là một số) và được cho ra bởi các việc tính toán như **inf/inf** hoặc **0/0**.

1.6. Số phức và ma trận phức

Số phức được dùng trong tất cả các phép toán và các hàm của **MATLAB.** Số phức được nhập bằng các hàm đặc biệt là **i** và **j**. Vài người có thể dùng

z = 3 + 4*i

trong khi người khác lại thích dùng

z = 3 + 4*j

Một ví dụ khác là

w = r*exp(i*theta)

Có ít nhất hai cách thuận tiện để nhập ma trận phức. Chúng được minh họa bởi các lệnh

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

và

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

cho ra cùng kết quả. Khi các số phức được nhập như các phần tử của ma trận bên trong cặp ngoặc vuông, thì điều quan trọng là tránh mọi khoảng trống, vì một biểu thức như **1 + 5*i** với ký tự trống quanh dấu + biểu hiện hai số riêng biệt. (Giống như thế cho số thực; một ký tự trống trước phần mũ trong **1.23 e-4** gây ra lỗi).

Tên *hàm cài sẵn* có thể dùng như tên biến; trong trường hợp này hàm gốc trở nên không dùng được bên trong vùng làm việc hiện thời (hoặc hàm M-file cục bộ) cho đến khi biến bị xóa. Nếu dùng **i** và **j** là tên các biến, và đè lên các giá trị này, thì một đơn vị phức mới được phát sinh và sử dụng theo cách thông thường:

ii = sqrt(-1)

z = 3 + 4*ii

1.7. Dạng thức xuất

Kết quả của mọi lệnh gán của **MATLAB** được hiển thị trên màn hình, gán cho biến chỉ định hoặc cho **ans** nếu không cho biến. Dạng thức hiển thị số có thể điều khiển bằng lệnh **format**. Lệnh **format** chỉ ảnh hưởng đến cách hiển thị ma trận chứ không ảnh hưởng đến việc tính toán và lưu chúng (**MATLAB** thực hiện tất cả các tính toán theo độ chính xác kép "**double**").

Nếu tất cả các phần tử của ma trận đúng là số nguyên thì ma trận được hiển thị theo dạng không có phần thập phân. Ví dụ,

x = [-1 0 1]

kết quả luôn là **x =**

-1 0 1

Nếu ít nhất một phần tử của ma trận không là số nguyên thì có một số cách có thể hiển thị. Dạng ngầm định, gọi là dạng **short**, trình bày khoảng **5** chữ số có nghĩa. Các dạng khác trình bày nhiều chữ số hơn hoặc dùng dạng khoa học. Ví dụ, giả sử

x = [4/3 1.2345e-6]

Các dạng thức, và kết quả xuất cho vectơ này, là:

Dạng thức short
1.3333 0.0000

Dạng thức short e
1.3333E+000 1.2345E-006

Dạng thức long
1.333333333333338 0.000001234500000

Dạng thức long e
1.333333333333338E+000 1.234500000000003E-006

Dạng thức hex
3FF555555555555 3EB4B6231AFBD271

Dạng thức +
+ +

Đối với các dạng **long** thì chữ số cuối cùng có thể xuất hiện không đúng, nhưng việc xuất ra đúng là một biểu hiện độ chính xác của số nhị phân lưu trong máy.

Với các dạng **short** và **long**, nếu phần tử lớn nhất của ma trận lớn hơn **1000** hoặc nhỏ hơn **0.001** thì một thừa số chung được áp dụng cho toàn bộ ma trận khi hiển thị nó. Ví dụ, lệnh

x = 1.e20*x

nhân **x** cho $\mathbf{10^{20}}$ và kết quả hiển thị

x =

1.0E+020 *

1.3333 0.0000

Dạng thức + là cách cô đọng để hiển thị các ma trận lớn. Các ký hiệu +, - và ký tự trống được hiển thị cho các phần tử dương, âm và bằng không.

Lệnh cuối cùng, **format compact**, bỏ nhiều ký tự sang dòng xuất hiện giữa các hiển thị về ma trận và cho phép nhiều thông tin hiện trên màn hình.

1.8. Công cụ trợ giúp

Công cụ trợ giúp cung cấp thông tin trực tiếp về hầu hết các vấn đề của **MATLAB.** Để xem danh sách các vấn đề trợ giúp, đánh vào lệnh

help

Để lấy về một vấn đề chỉ định, đánh vào **help** *topic*.(*topic* là vấn đề cần trợ giúp). Ví dụ, lệnh

help eig

cung cấp thông tin về cách sử dụng hàm giá trị riêng,

help [

trình bày cách dùng các dấu ngoặc vuông để nhập ma trận, và

help help

là tham khảo chính nó, nhưng làm việc tốt đẹp.

1.9. Thoát và lưu vùng làm việc

Để thoát **MATLAB,** đánh vào lệnh **quit** hoặc **exit**. Việc kết thúc quá trình làm việc của **MATLAB** làm cho các biến trong vùng làm việc bị mất. Trước khi thoát, vùng làm việc có thể được lưu lại để dùng về sau bằng cách đánh vào lệnh

save

Lệnh này lưu tất cả các biến vào tệp có tên là **matlab.mat**. Khi gọi **MATLAB** lần sau, vùng làm việc có thể được phục hồi từ tệp **matlab.mat** bằng lệnh

load

Các lệnh **save** và **load** có thể dùng với các tên tệp khác, hoặc chỉ lưu các biến đã chọn. Lệnh **save temp** lưu các biến hiện thời vào tệp có tên là **temp.mat**. Lệnh

save temp X

chỉ lưu biến **X**, trong khi lệnh

save temp X Y Z

lưu **X, Y,** và **Z**.

Lệnh **load temp** lấy lại tất cả các biến từ tệp **temp.mat**. Các lệnh **load** và **save** cũng có thể dùng cho việc nhập và xuất các tệp dữ liệu dạng **ASCII**, xem phần tham khảo để biết thêm chi tiết.

1.10. Các hàm

Phần lớn tính năng của **MATLAB** nhận được từ tập hợp mở rộng của nó về các hàm. **MATLAB** có một số lớn các hàm, cho đến nay trên 500 hàm. Một số hàm là hàm nội tại hay hàm cài sẵn với chính trình xử lý **MATLAB.** Các hàm khác có thể ở thư viện các siêu tệp M-file bên ngoài cùng gói hàng của **MATLAB**(**MATLAB TOOLBOX**). Và một số được thêm vào bởi người dùng cho các trình ứng dụng đặc biệt.

Rõ ràng với người dùng thì một hàm có thể có hay không có trong trình **MATLAB** hoặc ở siêu tệp M-file. Đây là một mặt quan trọng của **MATLAB**; người dùng có thể tạo ra các hàm của riêng mình, và chúng hoạt động đúng như các hàm nội tại cài sẵn của **MATLAB** . Sẽ biết thêm về siêu tệp M-file trong phần sau.

Các phạm trù chung của các hàm toán học có thể dùng trong **MATLAB** gồm:

Toán sơ cấp
Các hàm đặc biệt
Ma trận sơ cấp
Ma trận đặc biệt
Tách và đặt thừa số ma trận
Phân tích dữ liệu
Đa thức
Giải phương trình vi phân
Phương trình phi tuyến và tối ưu phi tuyến
Tích phân số
Xử lý tín hiệu

Các phần sau sẽ giới thiệu các phạm trù khác nhau này về các hàm giải tích. Trong giáo trình này chúng tôi không đi vào chi tiết trên từng hàm; điều này được thực hiện bởi công cụ trợ giúp và trong phần tham khảo.

Cho đến bây giờ, chúng ta chỉ biết các hàm với một đối số nhập và một đối số xuất. Các hàm của **MATLAB** cũng có thể dùng với nhiều đối số. Ví dụ, lệnh

x = sqrt(log(z))

trình bày cách dùng tổ hợp hai hàm đơn giản. Có các hàm của **MATLAB** dùng hai hoặc nhiều đối số nhập. Ví dụ,

theta = atan2(y, x)

Tất nhiên, mỗi đối số có thể là một biểu thức.

Một số hàm trả về hai hoặc nhiều đối số xuất. Các giá trị xuất được bọc quanh bởi cặp ngoặc vuông, **[** và **]**, và cách nhau dấu phẩy:

[V,D] = eig(A)

[y,i] = max(X)

Hàm thứ nhất trả về hai ma trận, **V** và **D**, gồm vectơ riêng và các giá trị riêng tương ứng của ma trận **A**. Ví dụ thứ hai, dùng hàm **max**, trả về giá trị lớn nhất **y** và chỉ số **i** của giá trị lớn nhất trong vectơ **X**.

Các hàm cho phép nhiều đối số xuất có thể trả về ít đối số xuất hơn. Ví dụ, hàm **max** với một đối số xuất,

max(X)

trả về đúng giá trị lớn nhất.

Các đối số nhập hay đối số ở bên phải của một hàm không bao giờ được thay đổi. Các giá trị xuất, nếu có, của một hàm luôn trả về ở các đối số bên trái.

****

Chương 2. CÁC PHÉP TOÁN TRÊN MA TRẬN

Các phép toán trên ma trận là điều cơ bản của **MATLAB**; bất kỳ đâu có thể được, chúng biểu hiện như xuất hiện trên giấy, chỉ phụ thuộc vào dung lượng bộ nhớ của máy.

2.1. Chuyển vị ma trận

Ký tự đặc biệt là dấu nháy (') biểu hiện phép chuyển vị một ma trận. Các lệnh

```
A = [ 1  2  3; 4  5  6; 7  8  0 ]

B = A'
```

kết quả là

```
A =

    1    2    3

    4    5    6

    7    8    0

B =

    1    4    7

    2    5    8

    3    6    0
```

và lệnh

```
x = [ -1  0  2 ]'
```

cho ra

```
x =

   -1
    0
    2
```

Dấu nháy **'** chuyển vị ma trận theo ý nghĩa hình thức; nếu **Z** là ma trận phức thì **Z'** là chuyển vị liên hợp của nó. Điều này đôi khi dẫn đến kết quả không như ý

muốn nếu dùng dữ liệu phức một cách bất cẩn. Đối với một chuyển vị không liên hợp thì dùng biểu thức **Z.'** hoặc hàm **conj(Z')**.

2.2. Cộng và trừ ma trận

Cộng và trừ ma trận được biểu hiện bằng các ký hiệu + và - . Các phép toán được định nghĩa cho các ma trận cùng cỡ. Ví dụ, với các ma trận trên, **A+x** là không đúng, vì **A** là ma trận vuông cấp **3** và **x** là ma trận cỡ **3x1**. Tuy nhiên,

```
C = A + B
```

là chấp nhận được, và kết quả là

```
C =

     2     6    10

     6    10    14

    10    14     0
```

Các phép cộng và trừ cũng được định nghĩa nếu một trong các toán hạng là đại lượng vô hướng, đó là ma trận cấp một. Trong trường hợp này, đại lượng vô hướng được cộng hoặc trừ vào tất cả các phần tử của toán hạng kia. Ví dụ

```
y =  x – 1
```

cho ra

```
y =

    -2
    -1
     1
```

2.3. Nhân ma trận

Phép nhân ma trận được biểu hiện bởi ký hiệu * . Phép toán được định nghĩa cho các ma trận có kích thước bên trong bằng nhau, đó là **X*Y** cho phép nếu số cột của ma trận **X** bằng số hàng của ma trận **Y**. Ví dụ, cả hai ma trận **x** và **y** ở trên có cỡ **3x1**, vì vậy biểu thức **x*y** không được định nghĩa và kết quả là một thông báo lỗi. Tuy nhiên, vài phép nhân khác về vectơ được định nghĩa, và rất hữu ích. Thông dụng nhất là tích nội tại, cũng được gọi là tích điểm hay tích vô hướng. Đây là

```
x'*y
```

kết quả là

```
ans =

     4
```

Tất nhiên, **y'*x** cho cùng kết quả. Có hai tích ngoại lai, chúng là chuyển vị của nhau.

```
x*y' =

     2     1    -1

     0     0     0

    -4    -2     2

y*x' =

     2     0    -4

     1     0    -2

    -1     0     2
```

Phép nhân trên từng phần tử sẽ được mô tả trong phần sau. (**MATLAB** không cung cấp đặc biệt cho việc tính toán vectơ qua các phép nhân. Tuy nhiên, người nào cần thì dễ dàng viết một siêu tệp M-file để tính toán chúng.)

Các phép nhân ma trận với vectơ là các trường hợp đặc biệt của nhân tổng quát ma trận với ma trận. Với ví dụ ma trận **A** và vectơ **x** thì

```
b = A*x
```

là được phép và kết quả xuất là

```
b =

     5
     8
    -7
```

Một cách tự nhiên, một đại lượng vô hướng có thể nhân hoặc bị nhân với ma trận bất kỳ.

```
pi*x
ans =

   -3.1416

    0.0000

    6.2432
```

2.4. Chia ma trận

Trong **MATLAB** có hai ký hiệu "**chia ma trận**", \ và /. Nếu **A** là ma trận không suy biến, thì **A\B** và **B/A** tương ứng hình thức với nhân trái và nhân phải của **B** cho nghịch đảo của **A**, đó là **inv(A)*B** và **B*inv(A)**, nhưng kết quả nhận được trực tiếp chứ không tính toán qua phép nghịch đảo. Nói chung,

X = A\B là lời giải phương trình **A*X = B**

X = B/A là lời giải phương trình **X*A = B**

Phép chia trái, **A\B**, được định nghĩa với **B** có cùng số dòng với **A**.Nếu **A** là ma trận vuông, thì nó được phân tích bằng phép khử **Gauss**. Các nhân tử được dùng để giải các phương trình **A*X(:,j) = B(:,j)**, ở đây **B(:,j)** biểu hiện cột thứ ***j*** của **B**. Kết quả là ma trận **X** có cùng cỡ với **B**. Nếu **A** suy biến (tùy theo ước lượng điều kiện **LINPACK** là **RCOND**) thì một thông báo lỗi được hiển thị.

Nếu **A** là ma trận vuông thì nó được phân tích bằng phương pháp trực giao **House-holder** với việc định trục xoay về cột. các nhân tử được dùng để giải các phương trình xác định dưới hoặc trên theo phương pháp bình phương bé nhất. Kết quả là một ma trận **X** cỡ ***m*x*n***, ở đây ***m*** là số cột của **A** và ***n*** là số cột của **B**. Mỗi cột của **X** có nhiều nhất ***k*** thành phần khác không, với ***k*** là hạng thực thụ của **A**.

Phép chia phải, **B/A**, được định nghĩa theo dạng chia trái là **B/A = (A'\B')'**.

Ví dụ, khi vectơ **b** được tính là **A*x** thì lệnh

```
z = A\b
```

có kết quả

```
z =

    -1
     0
     2
```

Đôi lúc dùng \ và / để tính các lời giải hệ phương trình xác định dưới hoặc trên bằng phương pháp bình phương bé nhất đưa đến nhiều điều đáng ngạc nhiên. Đó là khả năng "**chia**" một vectơ cho vectơ khác. Ví dụ, với các vectơ **x** và **y** ở trên thì

```
s = x\y
```

cho ra

```
s =

    0.8000
```

Đây là vì **s = 0.8** là giá trị vô hướng giải được từ phương trình ***xs = y*** theo phương pháp bình phương bé nhất. Chúng tôi đề nghị bạn đọc giải thích tại sao

```
S = y/x
```

cho ra

```
S =

    0.0000    0.0000   -1.0000
    0.0000    0.0000   -0.5000
    0.0000    0.0000    0.5000
```

2.5. Lũy thừa ma trận

Biểu thức **A^p** nâng **A** lên lũy thừa bậc **p** và được định nghĩa nếu **A** là ma trận vuông và **p** là đại lượng vô hướng. Nếu **p** là số nguyên lớn hơn **1** thì phép lũy thừa được tính bằng cách nhân lặp. Đối với các giá trị khác của **p** thì việc tính toán gồm các giá trị riêng và các vectơ riêng, vì vậy nếu **[V,D] = eig(A)** thì

```
A^p = V*D.^p/V
```

Nếu **P** là một ma trận ,và **a** là đại lượng vô hướng thì **a^P** nâng **a** lên lũy thừa **P** bằng cách dùng các giá trị riêng và các vectơ riêng. **X^P**, với **X** và **P** đều là ma trận, là một lỗi.

2.6. Các hàm sơ cấp về ma trận

Trong **MATLAB**, các biểu thức như **exp(A)** và **sqrt(A)** được xem như các phép toán về mảng, xác định trên từng phần tử của **A**. **MATLAB** cũng có thể tính các hàm siêu việt về ma trận, như hàm mũ và hàm logarit. Các hàm đặc biệt này

chỉ được định nghĩa cho các ma trận vuông, tính toán khá khó và tốn thời gian, và đôi lúc có các tính chất toán học khá hấp dẫn.

Một hàm toán học siêu việt được thông dịch là hàm về ma trận nếu có chữ "**m**" nối thêm vào cuối tên hàm, như **expm(A)** và **sqrtm(A)**. Theo trọn bộ của **MATLAB** thì ba hàm sau đây được định nghĩa:

Hàm siêu việt trên ma trận	
expm	hàm mũ
logm	hàm loga
sqrtm	hàm căn bậc hai

Tuy nhiên, danh sách có thể được mở rộng bằng cách thêm vào các siêu tệp M-file, hoặc dùng lệnh **funm**. Xem các siêu tệp M-file **sqrtm**, **logm**, và **funm** trong **MATLAB TOOLBOX**, và **expm** và **funm** trong phần tham khảo.

Các hàm sơ cấp khác về ma trận gồm

Hàm sơ cấp trên ma trận	
poly	tính đa thức đặc trưng
det	tính định thức
trace	tìm vết ma trận
kron	tích tenxơ Kronecker

Xem phần tham khảo để biết thêm chi tiết.

****

Chương 3. CÁC PHÉP TOÁN TRÊN MẢNG

Chúng tôi dùng từ *phép toán trên mảng* để nói về các phép toán số học trên từng phần tử, thay cho các phép toán đại số tuyến tính thông thường về ma trận biểu hiện bởi các ký hiệu *** / \ ^ '** . Đưa vào trước phép toán dấu chấm để biểu hiện phép toán trên mảng hay phép toán trên từng phần tử.

3.1. Cộng và trừ trên mảng

Đối với các phép cộng và trừ thì phép toán trên mảng và trên ma trận là giống nhau, vì vậy + và - có thể được xem là các phép toán hoặc là trên ma trận, hoặc là trên mảng.

3.2. Nhân và chia trên mảng

Phép nhân trên mảng hoặc nhân từng phần tử được biểu hiện bằng .* . Nếu A và B cùng kích thước thì A.* B biểu hiện mảng mà các phần tử của nó đơn giản là tích của từng cặp phần tử của A và B. Ví dụ, nếu

```
x = [ 1  2  3 ];  y = [ 4  5  6 ];
```

thì

```
z = x .* y
```

kết quả là

```
z =

    4    10    18
```

Các biểu thức A ./ B và A .\ B cho ra thương của từng cặp phần tử. Vì vậy,

```
z = x .\ y
```

kết quả là

```
z =

   4.0000    2.5000    2.0000
```

3.3. Lũy thừa trên mảng

Lũy thừa từng phần tử biểu hiện bởi **.^** . Sau đây là một số ví dụ, dùng các vectơ **x** và **y** ở trên. Đánh vào

```
z = x .^ y
```

kết quả là

```
z =

     1    32   729
```

Phần mũ có thể là một đại lượng vô hướng.

```
z = 2 .^ [x y]

z =

     2 4 8 16 32 64
```

Ví dụ cuối cùng minh họa cho một trong các đặc tính hấp dẫn về cú pháp của **MATLAB.** Mặc dù khó thấy, nhưng khoảng trống giữa chữ số **2** và dấu chấm là quan trọng. Nếu không có thì dấu chấm sẽ được thông dịch là một dấu chấm thập phân quan hệ với số **2**. Rồi **MATLAB** chỉ xem dấu mũ đứng riêng và tính lũy thừa ma trận, trong trường hợp này kết quả là một thông báo lỗi vì ma trận mũ không vuông. Xen vào cặp ngoặc đơn để thực hiện cấp độ ưu tiên toán tử.

3.4. Phép toán quan hệ

Có 6 phép toán quan hệ để so sánh hai ma trận cùng cỡ.

Phép toán quan hệ	
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
>	Lớn hơn
>=	Lớn hơn hoặc bằng
==	Bằng
~=	Khác

Phép so sánh được thực hiện giữa các cặp phần tử tương ứng; kết quả là một ma trận gồm các số **1** và **0**, với **1** biểu hiện cho giá trị *đúng* (**TRUE**) và **0** biểu hiện cho giá trị *sai* (**FALSE**).

Ví dụ

```
2+2 ~= 4
```

đơn giản là **0**.

Các phép quan hệ có thể trình bày mẫu của các phần tử của ma trận thỏa mãn các điều kiện khác nhau. Ví dụ, sau đây là ma phương cấp **6**.

```
A = magic(6)

A =

   35    1    6   26   19   24
    3   32    7   21   23   25
   31    9    2   22   27   20
    8   28   33   17   10   15
   30    5   34   12   14   16
    4   36   29   13   18   11
```

Ma phương cấp *n* là một ma trận cấp *n* được xây dựng từ các số nguyên từ **1** đến n^2 với tổng các dòng và các cột bằng nhau. Nếu nhìn vào ma trận đủ lớn thì có thể lưu ý thấy các phần tử bội **3** nằm trên đường chéo thứ ba. Để hiển thị điều kỳ lạ này, đánh vào

```
P = (rem(A,3) == 0)
```

Dấu = kép là toán tử kiểm tra bằng, **rem(A,3)** là ma trận gồm các số dư, **0** được mở rộng thành ma trận không, và **P** trở thành ma trận gồm các số **1** và **0**.

```
P =

     0     0     1     0     0     1
     1     0     0     1     0     0
     0     1     0     0     1     0
     0     0     1     0     0     1
     1     0     0     1     0     0
     0     1     0     0     1     0
```

Để thấy mẫu nhỏ hơn và rõ ràng, lệnh **format +** in ma trận ở dạng cô đọng, với dấu + cho số dương, dấu **-** cho số âm, và ký tự trống cho số **0**.

```
format +

P
      +                 +
  +                     +
    +                   +
      +                 +
  +                     +
    +                   +
```

Hàm **find** là hữu ích với các phép toán quan hệ, đó là tìm các số khác không trong ma trận **0-1**, và các phần tử dữ liệu thỏa mãn điều kiện quan hệ nào đó. Ví dụ, nếu **Y** là một vectơ thì hàm **find(Y < 3.0)** trả về một vectơ chứa các chỉ số của các phần tử trong **Y** nhỏ hơn **3.0**.

Các lệnh

```
i = find(Y >3.0);

Y(i) = 10*ones(i);
```

thay tất cả các phần tử trong **Y** lớn hơn **3.0** với giá trị **10.0**. Nó làm việc với ngay cả **Y** là một ma trận, vì ma trận có thể được tham chiếu như mảng các vectơ cột.

Biểu thức quan hệ **X ==NaN** luôn cho ra **NaN**, tùy theo các chỉ định số học **IEEE**, mọi phép toán trên **NaN** cho ra **NaN**. Nhưng đôi lúc cần kiểm tra các giá trị **NaN**. Vì vậy, hàm **isnan(X)** được cung cấp để trả về **1** cho các phần tử **NaN** của **X,** và **0** với các phần tử khác.

Cũng có ích là hàm **finite(x)** trả về **1** cho $-\infty < x < \infty$.

3.5. Phép toán logic

Có **3** phép toán logic làm việc với từng phần tử và thường dùng với các ma trận **0-1**.

Phép toán quan hệ	
&	Phép và
|	Phép hoặc
~	Phép phủ định

Các phép **&** và **|** so sánh hai đại lượng vô hướng, hoặc hai ma trận cùng cỡ. Đối với các ma trận thì chúng làm việc trên từng phần tử; nếu **A** và **B** là các ma trận **0-1** thì biểu thức **A & B** là một ma trận **0-1** khác biểu hiện logic **AND** của các phần tử tương ứng của **A** và **B**. Các phép toán logic xem mọi số khác **0** là đúng (**TRUE**). Chúng trả về **1** cho **TRUE** và **0** cho **FALSE**.

Phép **NOT**, hoặc bù logic, là toán tử đơn hạng. Biểu thức **~A** trả về các số **0** cho các phần tử khác **0** của **A** và **1** cho các phần tử **0**. Do đó hai biểu thức

```
P | (~P)

P & (~P)
```

trả về tất cả **1** và tất cả **0** tương ứng.

Các hàm **any** và **all** là hữu ích trong việc liên kết với các phép toán logic. Nếu **x** là vectơ **0-1** thì **any(x)** trả về **1** nếu phần tử bất kỳ của **x** khác không, và ngược lại trả về **0**.

Hàm **all(x)** trả về **1** chỉ nếu tất cả các phần tử của **x** khác không. Các hàm này đặc biệt hữu ích trong câu lệnh **if**,

if all(A < .5)

thực hiện các lệnh

end

Một lệnh if muốn trả lời cho một điều kiện đơn giản thì không phải là một vectơ có thể nhầm lẫn.

Đối với các đối số là ma trận thì các hàm **any** và **all** làm việc trên từng cột và trả về một vectơ dòng với kết quả của mỗi cột. áp dụng hàm hai lần, như **any(any(A))** luôn thu gọn ma trận về một điều kiện vô hướng.

Sau đây là bảng tóm tắt các hàm quan hệ và logic của **MATLAB:**

Hàm quan hệ và logic	
any	Điều kiện logic
all	Điều kiện logic
find	Tìm chỉ số của các điều kiện logic
exist	Kiểm tra nếu các biến tồn tại
isnan	Dò tìm các giá trị NaN
finite	Dò tìm các giá trị vô định
isempty	Dò tìm các ma trận rỗng
isstr	Dò tìm các biến xâu chữ
strcmp	So sánh các biến xâu chữ

3.6. Các hàm toán sơ cấp

Một tập hợp các hàm toán sơ cấp được áp dụng vào mảng trên cơ sở từng phần tử. Ví dụ,

A = [1 2 3; 4 5 6]

B = fix(pi*A)

C = cos(pi*B)

cho ra **A =**

1 2 3

4 5 6

B =

3 6 9

12 15 18

C =

-1 1 -1
1 -1 1

Các hàm có thể sử dụng gồm các hàm lượng giác và các hàm sơ cấp thông dụng:

Hàm lượng giác	
sin	Hàm sin
cos	Hàm cosin
tan	Hàm tang
asin	Hàm arcsin
acos	Hàm arccos
atan	Hàm arctang
atan2	Hàm arctang...
sinh	Hàm sin hyperbol
cosh	Hàm cosin hyperbol
tanh	Hàm tang hyperbol
asinh	Hàm arcsin hyperbol
acosh	Hàm arccos hyperbol
atanh	Hàm arctang hyperbol

Hàm toán sơ cấp	
abs	Trị tuyệt đối hoặc argument số phức
angle	Góc pha
sqrt	Căn bậc hai
real	Phần thực
imag	Phần ảo
conj	Liên hợp của số phức
round	Làm tròn về số nguyên gần nhất
fix	Làm tròn về phía số **0**
floor	Làm tròn về phía **-∞**
ceil	Làm tròn về phía **∞**
sign	Hàm dấu
rem	Phần dư hoặc môđun
exp	Mũ cơ số **e**
log	Logarit tự nhiên
log10	Logarit cơ số **10**

3.7. Các hàm toán học đặc biệt

Một số hàm đặc biệt cung cấp nhiều khả năng nâng cao:

Hàm đặc biệt	
bessel	Hàm **Bessel**
gamma	Hàm **gamma** và **gamma bù**
rat	Hàm xấp xỉ
erf	Hàm lỗi
invert	Hàm đảo lỗi
ellipk	Hàm **tích phân bù elliptic loại I**
ellipj	Hàm **elliptic Jacôbiên**

Giống như các hàm sơ cấp, chúng thực hiện trên từng phần tử khi nhập ma trận. Xem phần tham khảo để biết thêm thông tin.

****

Chương 4. THAO TÁC TRÊN VECTƠ VÀ MA TRẬN

Các công cụ về mô tả chỉ số của **MATLAB** cho phép thực hiện về dòng, về cột, về từng phần tử riêng biệt và từng phần của ma trận. Tâm điểm của việc mô tả chỉ số là vectơ, được phát sinh bằng cách dùng "Ký pháp Hai chấm". Vectơ và việc mô tả chỉ số là các thao tác hay dùng trong **MATLAB** và làm cho nó thực hiện các thao tác trên dữ liệu phức tạp khá hiệu lực.

4.1. Cách phát sinh vectơ

Dấu hai chấm, **:**, là ký tự quan trọng trong **MATLAB.** Lệnh

```
x = 1:5
```

phát sinh ra một vectơ dòng chứa các số từ **1** đến **5** theo chiều tăng đơn vị. Nó cho ra

```
x =

   1   2   3   4   5
```

Các cách tăng khác có thể dùng được.

```
y = 0:pi/4:pi
```

kết quả là

```
y =

   0.0000   0.7854   1.5708   2.3562   3.1416
```

Có thể thay đổi theo đơn vị âm.

```
z = 6:-1:1
```

cho ra

```
z =

   6   5   4   3   2   1
```

Ký pháp hai chấm cho phép phát sinh các bảng một cách dễ dàng. Để lấy một bảng sắp xếp theo chiều đứng thì chuyển vị vectơ dòng nhận được từ ký pháp hai chấm, tính toán cột giá trị, rồi định dạng ma trân từ hai cột. Ví dụ

```
x = (0.0 : 0.2 : 3.0)';
```

```
y = exp(-x) .*sin(x);

[x y]
```

cho ra

```
ans =

    0.0000    0.0000

    0.2000    0.1627

    0.4000    0.2610

    0.6000    0.3099

    0.8000    0.3223

    1.0000    0.3096

    1.2000    0.2807

    1.4000    0.2430

    1.6000    0.2018

    1.8000    0.1610

    2.0000    0.1231

    2.2000    0.0896

    2.4000    0.0613

    2.6000    0.0383

    2.8000    0.0204

    3.0000    0.0070
```

Các hàm phát sinh vectơ khác gồm **linspace**, cho phép số thực tốt hơn là cách tăng như đã chỉ định,

```
k = linspace(-pi,pi,4)

k =

  -3.1416   -1.0472    1.0472    3.1416
```

và hàm **logspace** phát sinh **vectơ logarit** đồng đều.

4.2. Mô tả chỉ số

Các phần tử riêng biệt của ma trận có thể được tham chiếu bằng cách đưa vào chỉ số trong cặp ngoặc đơn. Một biểu thức dùng làm chỉ số được làm tròn thành số nguyên gần nhất. Ví dụ, cho ma trận **A** :

A =

1 2 3

4 5 6

7 8 9

thì lệnh

A(3,3) = A(1,3) + A(3,1)

kết quả là

A =

1 2 3

4 5 6

7 8 10

Một chỉ số có thể là một vectơ.

Nếu **X** và **V** là các vectơ thì **X(V)** là **[X(V(1), X(V(2),..., X(V(n))]**. Đối với các ma trận chỉ số vectơ cho phép truy cập đến các ma trận con liên tục và không liên tục. Ví dụ, giả sử **A** là ma trận cấp **10**. Thì

A(1:5,3)

là ma trận con cỡ **5x1**, hay là vectơ cột gồm **5** phần tử đầu trên cột thứ **3** của ma trận **A**. Tương tự,

A(1:5, 7:10)

là ma trận con cỡ **5x4** gồm các phần tử từ **5** dòng đầu và **4** cột cuối.

Dùng chính dấu hai chấm đặt tại vị trí mô tả chỉ số biểu hiện tất cả các dòng hoặc các cột tương ứng. Ví dụ,

A(:,3)

là cột thứ **3** và

A(1:5,:)

là **5** cột đầu.

Hiệu lực khá tinh vi là nhận được từ việc tham chiếu ma trận con ở cả hai phía của lệnh gán. Ví dụ,

A(:,[3 5 10]) = B(:,1:3)

thay các cột thứ **3**, **5** và **10** của **A** với **3** cột đầu của **B**.

Nói chung, nếu **v** và **w** là các vectơ với các thành phần nguyên thì

A(v,w)

là ma trận nhận được bằng cách lấy các phần tử của **A** với chỉ số dòng trong **v** và chỉ số cột trong **w**. Vì vậy

A(:,n:-1:1)

đảo lại các cột của **A** và

```
v = 2:2:n;

w = [3  1  4  1  6];

A(v,w)
```

là hợp pháp, nhưng có lẽ đáng ngờ.

Một đặc điểm nữa cũng khá hữu ích là **A(:)**. ở vế phải câu lệnh gán, **A(:)** biểu hiện tất cả các phần tử của **A** được gióng thành một vectơ cột. Ví dụ

```
A = [ 1  2; 3  4; 5  6 ]

b = A(:)
```

kết quả là

```
A =

    1    2
```

```
     3    4

     5    6

b =

     1
     2
     3
     4
     5
     6
```

Ở vế trái câu lệnh gán, **A(:)** có thể được dùng để đổi lại cỡ ma trận. Để làm điều này, trước hết **A** phải có. Sau đó **A(:)** biểu hiện ma trận cùng cỡ với **A**, nhưng với nội dung mới lấy bên vế phải. Ví dụ, **A** ở trên cỡ **3X2**, vì vậy

```
A(:) = 11:16
```

đổi **6** phần tử của vectơ dòng thành ma trận cỡ **3X2**,

```
A =

    11    14

    12    15

    13    16
```

4.3. Mô tả chỉ số bằng vectơ 0-1

Có thể dùng các vectơ **0-1**, thường được tạo ra bằng các phép toán quan hệ, để tham chiếu các ma trận con. Giả sử **A** là ma trận cỡ ***mxn*** và **L** là vectơ ***m*** chiều gồm các phần tử **0** và **1**. Thì **A(L,:)** chỉ định các dòng của **A** ứng với các phần tử **1** của **L**.

Sau đây là các cách trích ra, các phần tử lớn hơn **3** lần độ lệch chuẩn bị xóa trong vectơ:

```
x = x(x<=3*std(x));
```

Tương tự,

```
L = X(:,3)>100;

X = X(L,:);
```

thay **X** với các dòng có phần tử ở cột thứ **3** lớn hơn **100**.

4.4. Ma trận rỗng

Lệnh

x = [ ]

gán ma trận cỡ **0x0** cho **x**. Sau đó dùng ma trận này sẽ không dẫn đến điều kiện lỗi; mà truyền cho ma trận rỗng. Lệnh này khác với lệnh

clear x

Xóa **x** từ danh sách các biến hiện thời. Các ma trận rỗng có trong vùng làm việc; chúng có đúng cỡ **0x0**. Hàm **exist** có thể dùng để kiểm tra sự tồn tại của một ma trận (hoặc một tệp cho ma trận), trong khi đó hàm **isempty** kiểm tra ma trận rỗng.

Có thể phát sinh các vectơ rỗng. Nếu **n<1** thì **1:n** không chứa phần tử nào cả và do đó

x = 1:n

là một cách việc tạo một vectơ **x** rỗng.

Quan trọng hơn nữa là một cách có hiệu lực để xóa các dòng, các cột của một ma trận là gán chúng cho một ma trận rỗng. Ví dụ

A(:,[2,4]) = [ ]

xóa cột **2** và **4** của ma trận **X.**

Chắc chắn các hàm ma trận sẽ trả về các giá trị hợp lý về mặt toán học nếu ma trận rỗng. Chúng là các hàm **det, cond, prod, sum**, và một số hàm khác. Ví dụ, các hàm **prod, det**, và **sum** trả về **1, 1**, và **0** tương ứng khi ma trận đối là rỗng.

Hơn nữa chúng ta đã biết không có đại số về các ma trận rỗng. Chúng tôi không chắc rằng chúng tôi sẽ thực hiện điều đó một cách đúng đắn, nhưng chúng tôi đã tìm thấy nhiều điều hữu ích.

4.5. Ma trận đặc biệt

Tập hợp các hàm phát sinh các ma trận đặc biệt để đưa vào trong đại số tuyến tính và xử lý tín hiệu.

Các ma trận đặc biệt	
compan	Ma trận Liên hợp
diag	Ma trận Chéo
gallery	Ma trận riêng
hadamard	Ma trận **Hadamard**
hankel	Ma trận **Hankel**
hilb	Ma trận **Hilbert**
invhilb	Ma trận **Hilbert** đảo
magic	Ma phương
pascal	Tam giác **Pascal**
toeplitz	Ma trận **Toeplitz**
vander	Ma trận **Vandermonde**

Ví dụ, phát sinh một ma trận liên hợp với đa thức $\mathbf{x^3 - 7x + 6}$.

```
p = [ 1  0  -7  6 ]

a = compan(p)

a =

    0    7   -6

    1    0    0

    0    1    0
```

Các giá trị riêng của **a** là nghiệm của đa thức.

```
-3.0000
 2.0000
 1.0000
```

Một ma trận **Toeplitz** không đồng bộ về đường chéo là

```
c = [ 1   2   3   4   5 ];

r = [1.5  2.5  3.5  4.5  5.5];

t =toeplitz(c,r)

t =
```

1.000	**2.500**	**3.500**	**4.500**	**5.500**
2.000	**1.000**	**2.500**	**3.500**	**4.500**
3.000	**2.000**	**1.000**	**2.500**	**3.500**
4.000	**3.000**	**2.000**	**1.000**	**2.500**
5.000	**4.000**	**3.000**	**2.000**	**1.000**

Các hàm khác phát sinh các ma trận tiện ích ít quan tâm nhưng hữu ích hơn.

Các ma trận tiện ích	
zeros	Ma trận zero
ones	Ma trận một
rand	Ma trận phần tử ngẫu nhiên
eye	Ma trận đơn vị
linspace	Vectơ không gian tuyến tính
logspace	Vectơ không gian loga
meshdom	Phạm vi để vẽ lưới

Trong các hàm này có hàm **eye(A)** trả về ma trận đơn vị cùng cỡ với **A**. Nên dùng tên dễ nhớ vì **I** và **i** thường dùng như các chỉ số hay đơn vị ảo **sqrt(-1)**.

Các hàm **zeros** và **ones** phát sinh các ma trận hằng có kích cỡ khác nhau, và hàm **rand** để phát sinh các ma trận phân bố đồng bộ hoặc bình thường các phần tử ngẫu nhiên. Ví dụ, để phát sinh ma trận cỡ **4x3**

```
A = rand(4,3)

A =

   0.2113   0.8096   0.4832

   0.0824   0.8474   0.6135

   0.7599   0.4524   0.2749

   0.0087   0.8075   0.8807
```

4.6. Cách tạo ra ma trận lớn

Các ma trận lớn có thể được tạo ra từ các ma trận nhỏ bằng cách dùng cặp ngoặc vuông bao quanh các ma trận nhỏ. Ví dụ,

C = [A eye(4); ones(A) A^2]

tạo ra ma trận lớn với giả thiết **A** có **4** dòng. Các ma trận nhỏ hơn trong kiểu này của cách xây dựng phải cùng cỡ hoặc kết quả là một thông báo lỗi.

4.7. Thực hiện trên ma trận

Các hàm sẽ quay, đổi hàng-cột, thay đổi kích thước, hoặc trích ra các phần của ma trận.

Thao tác trên ma trận	
rot90	Quay ma trận
fliplr	Đổi cột ma trận
flipud	Đổi cột ma trận
diag	Trích hoặc tạo ra đường chéo
tril	Phần tam giác dưới
triu	Phần tam giác trên
reshape	Đặt lại kích thước
.'	Chuyển vị
:	Sắp xếp tổng quát

Ví dụ, để dặt lại kích thước ma trận cỡ **3x4** thành ma trận **2x6**:

a =

1 4 7 10

2 5 8 11

3 6 9 12

b = reshape(a,2,3)

b =

1 3 5 7 9 11

2 4 6 8 10 12

Ba hàm **diag, triu**, và **tril** cung cấp truy cập đến đường chéo, tam giác trên, và tam giác dưới của ma trận. Ví dụ,

tril(rand(4,3))

cho ra **ans =**

0.2113	**0**	**0**
0.0824	**0.8474**	**0**
0.7599	**0.4524**	**0.2749**
0.0087	**0.8075**	**0.8807**

Cũng rất hữu ích là các hàm **size** và **length**. Hàm **size** trả về vectơ **2** chiều chứa số dòng và số cột của một ma trận. Nếu biến là một vectơ thì **length** trả về số chiều của vectơ, hoặc **max(size(V))**.

****

Chương 5. THAO TÁC TRÊN VECTƠ VÀ MA TRẬN

Phần này giới thiệu về cách phân tích dữ liệu bằng cách dùng **MATLAB** và mô tả vài công cụ thống kê cơ bản. Kỹ thuật mạnh hơn là dùng đại số tuyến tính và các hàm xử lý tín hiệu bàn đến trong phần sau.

5.1. Phân tích theo hướng cột

Tất nhiên các ma trận dùng để giữ tất cả dữ liệu, nhưng điều này dẫn đến một lựa chọn của hướng dữ liệu khác nhau. Theo quy ước thì các biến khác nhau trong tập hợp dữ liệu được đặt theo các cột, cho phép quan sát qua các dòng. một tập hợp dữ liệu gồm **50** mẫu, **13** biến được lưu trong một ma trận cỡ **50x13**.

Bắt đầu bằng ví dụ, dữ liệu kinh tế **Longley** đã có gồm các biến

1) Lạm phát quốc dân

2) Thu nhập quốc dân

3) Thất nghiệp

4) Lực lượng quân đội

5) Dân số

6) Năm

7) Lực lượng lao động

Nói chung có nhiều cách đưa dữ liệu vào **MATLAB**; điều này được khám phá trong phần sau. Giả sử dữ liệu chưa có trong dạng máy đọc được thì cách nhập dữ liệu dễ nhất là dùng một trình soạn thảo văn bản hoặc trình xử lý từ. Nếu tạo ra một tệp tên là **longley.m** chứa các lệnh gán

ldata = [

83.0 234.289 235.6 159.0 107.608 1947 60.232

88.5 259.426 232.5 145.6 108.623 1948 61.122

88.2 258.054 368.2 161.6 109.773 1949 60.171

89.5 284.599 335.1 165.0 110.929 1950 61187

96.2 328.975 209.9 309.9 112.075 1951 63.221

98.1 346.999 193.2 359.4 113.270 1952 63.639

99.0 365.385 187.0 354.7 115.094 1953 64.989

100.0 363.112 357.8 335.0 116.219 1954 63.761

101.2 397.469 290.4 304.8 117.388 1955 66.019

104.6 419.180 282.2 285.7 118.734 1956 67.857

108.4 442.769 293.6 279.8 120.445 1957 68.169

110.8 444.546 468.1 263.7 121.950 1958 66.513

112.6 482.704 381.3 255.2 123.366 1959 68.655

114.2 502.601 393.1 251.4 125.368 1960 69.564

115.7 518.173 480.6 257.2 127.852 1961 69.331

116.9 554.894 400.7 282.7 130.081 1962 70.551]

thì có thể thực hiện lệnh **longley**. Lệnh này truy cập tệp **longley.m** và tạo ra ma trận ldata (hoặc tên bất kỳ khác nếu muốn) trong vùng làm việc.Thử nhập ma trận này trong chế độ tương tác, nhưng chỉ được sửa đổi ở lần đầu. Nếu nhập sai thì không có cách sửa đổi.

Nếu quan sát nhiều hơn là có trên màn hình, các dòng có thể tiếp tục trên dòng tiếp theo bằng cách dùng dấu tĩnh lược gồm **3** dấu chấm. Ma trận cũng có thể nhập trong các khối cột và nối toàn bộ lại ở cuối dòng.

Với dữ liệu **longley** có **16** mẫu xét gồm **7** biến. Điều này biểu lộ bởi

```
[n,p] = size(ldata)

n =

    16

p =

    7
```

Đối với dữ liệu nhập theo từng cột này thì một nhóm các hàm cung cấp các công cụ phân tích dữ liệu cơ bản:

Phân tích dữ liệu theo từng cột	
max	giá trị cực đại
min	giá trị cực tiểu
mean	giá trị trung bình
median	giá trị trung gian
std	độ lệch chuẩn
sort	sắp xếp
sum	tổng các phần tử
prod	tích các phần tử
cumsum	tổng tích lũy các phần tử
cumprod	tích tích lũy các phần tử
diff	đạo hàm xấp xỉ
hist	biểu đồ tần số
corrcoef	hệ số tương quan
cov	ma trận hiệp phương sai
cplxpair	Sắp lại thành cặp số phức

Đối với các đối là vectơ thì các hàm này không xét đến các vectơ có được định hướng theo dòng hay theo cột. Đối với các đối là mảng thì các hàm thực hiện theo cách định hướng cột trên dữ liệu trên mảng. Điều này có nghĩa là, chẳng hạn nếu hàm **max** áp dụng cho mảng thì kết quả là một vectơ dòng chứa các giá trị lớn nhất trên mỗi cột.

Do đó, nếu

```
A =

   9   8   4

   1   6   5

   3   2   7
```

thì

```
m = max(A)

mv = mean(A)

s = sort(A)
```

kết quả là

```
m =
```

```
    9   8   7

mv =

    4.3333   5.3333   5.333

s =

   1   2   4

   3   6   5

   9   8   7
```

Hoặc với dữ liệu **longley**

```
m = median(ldata)

m =

  1.0E+003*

  0.1012   0.3975   0.3351   0.2798   0.1174   1.9550   0.0660
```

Có thể trừ giá trị trung bình mỗi cột của **ldata** bằng cách dùng phép nhân bên ngoài

```
lmean = ldata - ones(n,1)*m;
```

Có thể thêm vào danh sách này bằng cách dùng các siêu tệp M-file, nhưng khi dùng hãy cẩn thận để xử lý trường hợp vectơ dòng. Nếu viết các tệp M-file theo hướng cột riêng thì hãy xem cách hoàn thành điều này trong các tệp M-file khác, ví dụ **mean.m** và **diff.m.**

5.2. Các giá trị bỏ qua

Giá trị đặc biệt, **NaN**, viết tắt chữ **Not-a-Number** trong **MATLAB**. Thông thường cho ra bởi các biểu thức không xác định như **0/0**, nguyên nhân của một thông báo lỗi, tùy theo quy ước thiết lập bởi chuẩn **IEEE**. Đối với công dụng thống kê thì các giá trị **NaN** có thể dùng để biểu hiện các giá trị bỏ qua hoặc dữ liệu không dùng được, **NA**.

Cách "sửa đổi" các giá trị **NA** là một điều khó khăn và thường khác nhau tùy theo từng trường hợp cụ thể. Tuy nhiên, **MATLAB** đồng dạng và nghiêm ngặt trong cách xem xét của nó về các giá trị **NaN**; chúng truyền một cách tự nhiên cho kết quả cuối cùng của mọi tính toán. Do đó nếu một giá trị **NaN** được dùng trong

mọi lần tính toán trung gian thì kết quả cuối cùng sẽ là một **NaN**, trừ khi kết quả cuối cùng không phụ thuộc vào giá trị **NaN**.

Về mặt thực hành, điều này có nghĩa là nên xóa các **NaN** trong dữ liệu trước khi thực hiện việc tính toán thống kê. Các **NaN** trong vectơ **x** được tìm ở:

```
i = find(isnan(x));
```

vì vậy

```
x = x(find(~isnan(x)))
```

trả về dữ liệu trong **x** với các **NaN** đã xóa. Có **2** cách khác nhau để thực hiện việc này là

```
x = x(~isnan(x));

x(isnan(x)) = [ ];
```

có lẽ cách thứ hai là rõ nhất. Phải dùng hàm đặc biệt **isnan** để tìm các **NaN** vì không thể dùng lệnh

```
x(x==NaN) = [ ];
```

Các **NaN** trả về **NaN** cho mọi phép toán, kể cả các phép toán quan hệ.

Nếu thay vì vectơ, dữ liệu ở trên các cột của ma trận, và xóa mọi dòng của ma trận có **NaN** thì dùng

```
X(any(isnan(X)'),:) = [ ];
```

đây là một lệnh khá thô, nhưng có hiệu lực. Nếu cho rằng khó nhớ thì hoàn toàn biện hộ. Nếu thường cần xóa các **NaN** thì cách giải quyết là viết một tệp M-file, ví dụ

```
function  X = excise(X)

X(any(isnan(X)'),:) = [ ];
```

Bây giờ đánh vào

```
X = excise(X);
```

là hoàn thành cùng công việc. Biết thêm về các tệp M-file sau này.

5.3. Cách xóa các giá trị quá hạn

Cách xóa các giá trị quá hạn trong dữ liệu giống như cách xóa các **NaN**. Với dữ liệu **Longley**, giá trị trung bình và các độ lệch chuẩn của mỗi cột dữ liệu là:

```
mv = mean(ldata)

sigma = std(ldata)

mv =

   1.0E+003*

   0.101   0.387   0.319   0.260   0.117   1.954   0.065

sigma =

      10.448   96.238   90.479   67.382   6.735   4.609   3.400
```

Số dòng có giá trị chênh lệch lớn hơn **3** lần độ lệch chuẩn là:

```
[n,p] = size(ldata);

e = ones(n,1);

dist = asb(ldata-e*mv);

outliers = dist > 3*e*sigma;

nout = sum(any(outliers'))

nout =
      0
```

Không có. Nếu có thì chúng bị xóa với lệnh

```
X(any(outliers'),:) = [ ];
```

5.4. Hồi quy và đường cong thực nghiệm

Trước khi đưa đường cong thực nghiệm vào dữ liệu thì phải chuẩn hóa dữ liệu. Việc chuẩn hóa có thể cải tiến độ chính xác của kết quả cuối cùng. Vẫn làm việc với dữ liệu Longley, một cách chuẩn hóa là xóa giá trị trung bình

```
X = X - e* mean(X);
```

và để chuẩn hóa thành đơn vị độ lệch chuẩn

```
X = X ./ (e*std(X);
```

Có thể tính hồi quy thất nghiệp (cột cuối cùng) theo các cột trước đó, dùng trong dữ liệu thô trong trường hợp này,

```
y = ladat(:,7);

A = [ldata(:,1:6)  ones(y)];

coef = A\y
```

kết quả là

```
coef =

  1.0E+0003

  0.00001506187227

 -0.00003581917929

 -0.00002020229804

 -0.00001033226867

 -0.00005110410565

  0.00182915146461

 -3.48225863459802
```

Dữ liệu **Longley** có tương quan cao, xem qua các hệ số tương quan.

```
corr(X)

ans =

1.0000   0.9916   0.6206   0.4647   0.9792   0.9911   0.9709
0.9916   1.0000   0.6043   0.4464   0.9911   0.9953   0.9836
0.6206   0.6043   1.0000  -0.1774   0.6866   0.6683   0.5025
0.4647   0.4464  -0.1774   1.0000   0.3644   0.4172   0.4573
0.9792   0.9911   0.6866   0.3644   1.0000   0.9940   0.9604
0.9911   0.9953   0.6683   0.4172   0.9940   1.0000   0.9713
0.9709   0.9836   0.5025   0.4573   0.9604   0.9713   1.0000
```

Thường đưa đa thức vào dữ liệu là có ích. Nói chung, một đa thức đưa vào dữ liệu theo các vectơ **x** và **y** là một hàm, ***p***, có dạng:

$$p(x) = c_1x^d + c_2x^{d-1} + ...+ c_n$$

Cấp là ***d*** và số hệ số là ***n = d + 1***. Các hệ số $c_1, c_2, ..., c_n$ được xác định bằng cách giải hệ phương trình tuyến tính:

Ac = y

Các cột của **A** là lũy thừa thoái của vectơ **x**. Sau đây là một cách tạo ra **A**

```
for j=1:n
        A(:,j) = x.^(n-j);
end
```

Lời giải của hệ phương trình tuyến tính **Ac = y** nhận được với phép chia ma trận của **MATLAB**:

```
c = A\y
```

Hàm **polyfit.m** trong **MATLAB TOOLBOX** tự động làm thủ tục này.

Trong bài toán hồi quy, các hàm khác, thường là hàm nhiều biến các cột của ma trận dữ liệu, được đưa vào dữ liệu bằng cách tìm dạng của ma trận **A** tương ứng. Ví dụ, dùng dữ liệu **longley**,

```
A = [ldata(:,1)  ldata(:,2).^2  sin(ldata(:,3))  ones(n,1)];
coef = A\y;
```

tìm các hệ số hồi quy cho một hàm phức tạp hơn.

****

Chương 6. HÀM MA TRẬN

Nhiều khả năng toán học của **MATLAB** nhận được từ các hàm ma trận của nó. Một số hàm gắn liền với trình xử lý **MATLAB**. Các hàm khác, trong thư viện các tệp M-file, phân phối cùng với **MATLAB**. Và một số được thêm vào bởi từng người dùng, hoặc nhóm người cho các trình ứng dụng đặc biệt. ở đây chúng tôi không đi sâu vào chi tiết từng hàm; điều đó được thực hiện trong công cụ trợ giúp và phần tham khảo. Thông tin thêm nữa cũng có thể tìm trong hướng dẫn sử dụng phần mềm **LINPACK** và **EISPACH**, cung cấp cơ bản về thuật toán cho **MATLAB**. Trong phần này, chúng tôi cho xem qua các hàm được nhóm theo các hàm thừa số ma trận và hàm phân tích ma trận.

Gồm **4** nhóm:

- Thừa số tam giác
- Thừa số trực giao
- Tách giá trị riêng
- Tách giá trị kỳ dị

6.1. Thừa số tam giác

Cách tách thừa số cơ bản nhất là tách ma trận vuông bất kỳ thành tích **2** ma trận tam giác, một ma trận là hoán vị của một ma trận tam giác dưới và ma trận kia là ma trận tam giác trên. Việc tách thừa số thường gọi là "thừa số **LU**" hoặc đôi khi gọi là "thừa số **LR**". Hầu hết các thuật toán để tính là các phép biến đổi theo phương pháp khử **Gauss**.

Chính các thừa số lấy được từ hàm **lu**. Các thừa số được dùng để nhận nghịch đảo ma trận với hàm **inv** và lấy định thức với hàm **det**. Đó cũng là cơ sở cho việc giải hệ phương trình tuyến tính hay "chia ma trận vuông" với các toán tử \ và /.

Ví dụ, bắt đầu với

A =

1 2 3

4 5 6

7 8 0

Để xem phân tích **LU**, dùng câu lệnh gán kép của **MATLAB**.

[L,U] = lu(A)

cho ra

L =

0.1429	**1.0000**	**0**
0.5714	**0.5000**	**1.0000**
1.0000	**0**	**0**

U =

7.0000	**8.0000**	**0.0000**
0	**0.8571**	**3.0000**
0	**0**	**4.5000**

Lưu ý rằng **L** là hoán vị của ma trận tam giác dưới có các số **1** trên đường chéo, và **U** là tam giác trên. Để kiểm tra thừa số có thể tính tích

L*U

cho ra giá trị **A** gốc. Đó là,

ans =

1	**2**	**3**
4	**5**	**6**
7	**8**	**0**

Ma trận đảo của ma trận ví dụ nhận được với

X = inv(A)

Ma trận đảo được tính toán thực sự qua các nghịch đảo của các thừa số tam giác

X = inv(U)*inv(L)

Định thức của ma trận ví dụ nhận được với

d = det(A)

cho ra

d =

27

Được tính từ định thức của các thừa số tam giác

d = det(L)*det(U)

cho ra

d =

27.0000

Tại sao **2** lần in ra **d** ở dạng khác nhau? Khi **MATLAB** được yêu cầu tính **det(A)**, nó nhận thấy tất cả các các phần tử của **A** là nguyên, vì vậy nó cho định thức là số nguyên. Nhưng khi tính lần hai, các phần tử của **U** không nguyên, vì vậy **MATLAB** không cho kết quả là số nguyên.

Như một ví dụ về hệ phương trình tuyến tính, lấy

b =

1
3
5

Lời giải phương trình **Ax = b** nhận được với phép chia ma trận của **MATLAB**

x = A\b

cho ra

x =

0.3333
0.3333
0.0000

Lời giải được tính toán thực sự bằng cách giải **2** hệ tam giác,

y = L\b, x = U\y

Lời giải trung gian là

y =

5.0000
0.2857
0.0000

Thừa số tam giác cũng được dùng bởi một hàm đặc biệt là **rcond**. Đây là sản phẩm của một số chương trình con của **LINPACK** để ước lượng số điều kiện tính nghịch đảo của một ma trận vuông.

Hai hàm khác, **chol** và **rref**, có thể được đưa vào nhóm này vì thuật toán cơ bản quan hệ gần gủi với thừa số **LU**. Hàm **chol** cho thừa số **Cholesky** của một ma trận đối xứng xác định dương. Dạng xếp bậc thu gọn dòng của ma trận chữ nhật, **rref**, có một ít đáng quan tâm trong lý thuyết đại số tuyến tính, mặc dù có giá trị tính toán không lớn. Nó được đưa vào **MATLAB** vì tính sư phạm.

6.2. Thừa số trực giao

Thừa số "**QR**" là hữu ích cho cả ma trận vuông lẫn ma trận chữ nhật. Nó tách một ma trận thành tích của một ma trận trực chuẩn và một ma trận tam giác trên. Ví dụ, lấy

A =

1 2 3

4 5 6

7 8 9

10 11 12

là ma trận khuyết hạng; cột giữa là trung bình cộng của hai cột kia. Tính khuyết hạng có thể được biểu lộ bằng thừa số.

[Q,R] = qr(A)

cho ra

Q =

-0.0776 -0.8331 0.5444 0.0605

-0.3105 -0.4512 -0.7709 0.3251

```
   -0.5433    -0.0694    -0.0913    -0.8317
   -0.7762     0.3124     0.3178     0.4461

R =

  -12.8841   -14.5916   -16.2992
         0    -1.0413    -2.0826
         0          0     0.0000
         0          0          0
```

Có thể kiểm tra rằng tích **Q*R** cho ra **A** gốc, nhưng đừng băn khoăn điều đó. Cấu trúc tam giác của **R** cho nó các số **0** dưới đường chéo; số **0** trên đường chéo ở **R(3,3)** cho thấy rằng **R,** và do đó **A,** không phải là ma trận đủ hạng.

Phân tích thừa số **QR** được dùng để giải các hệ phương trình tuyến tính với số phương trình nhiều hơn số ẩn. Ví dụ

```
b =

   1
   3
   5
   7
```

Hệ phương trình **Ax=b** là hệ **4** phương trình chỉ có **3** ẩn. Lời giải tốt nhất theo phương pháp bình phương bé nhất được tính bởi

```
x = A\b
```

cho ra

```
Warning: Rank deficient, rank = 2     tol = 1.4594E-014

x                                                    =
   0.5000
   0.0000
   0.1667
```

Báo trước về sự khuyết hạng. Giá trị **tol** dùng để xác định rằng một phần tử trên đường chéo của **R** là không đáng kể. Lời giải **x** được tính bằng cách tách thừa số và qua **2** bước

```
y = Q'*b;
x = R\y
```

Nếu kiểm tra lời giải đã tính theo công thức **A*x** thì thấy rằng nó bằng **b** với sai số làm tròn. Điều này nói rằng mặc dù hệ phương trình **Ax=b** là vô định và khuyết hạng nhưng chúng vẫn thích hợp. Có nhiều lời giải vô định vectơ **x**; thừa số **QR** tìm ra một lời giải trong chúng.

Việc phân tích thừa số này cũng là cơ sở cho các hàm **null** và **orth**, chúng phát sinh các cơ sở trực giao cho không gian **0** và phạm vi của một ma trận đã cho.

6.3. Tách giá trị kỳ dị

Chúng tôi không có ý giải thích cách tách giá trị kỳ dị ở đây; chúng ta phải chấp nhận với ý kiến cho rằng nó là công cụ mạnh mẽ cho việc giải các bài toán về ma trận. Xem sách hướng dẫn sử dụng **LINPACK** hoặc sách viết bởi **Golub** và **VanLoan** đối với vấn đề này. Trong **MATLAB**, lệnh gán ba

```
[U,S,V] = svd(A)
```

cho ra **3** thừa số trong việc tách giá trị kỳ dị,

```
A = U*S*V'
```

U và **V** là ma trận trực giao và **S** là ma trận chéo. Bằng chính nó, hàm **svd(A)** trả về đúng các phần tử trên đường chéo của **S**, đó là các giá trị kỳ dị của **A**.

Việc tách giá trị kỳ dị được sử dụng cho một số hàm khác, kể cả hàm giả đảo, **pinv(A)**; tính hạng, **rank(A)**; chuẩn ma trận **Ơ-clit**, **norm(A,2)**; và số điều kiện, **cond(A)**.

6.4. Giá trị riêng

Nếu **A** là ma trận vuông cấp ***n*** thì ***n*** số **λ** thỏa mãn **Ax = λx** gọi là các giá trị riêng của **A**. Chúng được tính bằng

```
eig(A)
```

trả về các giá trị riêng trong một vectơ cột. Nếu **A** là ma trận thực và đối xứng thì các giá trị riêng là thực. Nhưng nếu **A** không đối xứng thì các giá trị riêng luôn là số phức. Ví dụ, với

```
A =

  0   1
```

-1 0

Lệnh **eig(A)** cho ra

ans =

0.0000 + 1.0000 i

0.0000 - 1.0000 i

Các vectơ riêng và các giá trị riêng nhận được với lệnh gán kép,

[X,D] = eig(A)

Trong trường hợp này các phần tử trên đường chéo của **D** là các giá trị riêng và các cột của **X** là các vectơ riêng tương ứng mà **A*X = X*D**.

Hai kết quả trung gian dùng trong việc tính các giá trị riêng là chuẩn **Hessenberg**, **hess(A)**, và chuẩn **Schur**, **Schur(A)**. Chuẩn **Schur** dùng để tính các hàm ma trận siêu việt, như **sqrtm(A)** và **logm(A)**.

Nếu **A** và **B** là các ma trận vuông thì hàm **eig(A,B)** trả về một vectơ chứa các giá trị riêng suy rộng từ lời giải phương trình

$$\mathbf{Ax = \lambda Bx}$$

Lệnh gán kép dùng để nhận các vectơ riêng

[X,D] = eig(A,B)

cho ra ma trận chéo **D** gồm các giá trị riêng suy rộng và ma trận **X** đầy đủ có các cột là các vectơ riêng tương ứng mà **A*X = B*X*D**. Các kết quả trung gian trong lời giải của bài toán giá trị riêng suy rộng này có thể dùng từ hàm **qz(A,B)**.

6.5. Hạng và điều kiện

Các hàm của **MATLAB** liên quan đến hạng và điều kiện gồm:

Điều kiện về ma trận	
Cond	Số điều kiện trong chuẩn **2**
norm	chuẩn **1**, chuẩn **2**, chuẩn F, chuẩn ∞
rank	hạng ma trận
rcond	ước lượng điều kiện

Có nhiều nơi trong **MATLAB** có tính hạng ma trận: trong **rref(A)**, trong **A\B** với **A** không vuông, trong **orth(A)** và **null(A)**, và trong giả đảo **pinv(A)**. Ba thuật toán khác nhau với ba tiêu chuẩn khác nhau không đáng kể và vì vậy ba giá trị khác nhau đó có thể cho ra ma trận giống nhau.

Với **rref(A)** thì hạng của **A** là số dòng khác không. Thuật toán khử của **rref** là nhanh nhất trong **3** thuật toán xác định hạng ma trận, nhưng ít tinh vi và ít tin cậy nhất.

Với **A\B**, **orth(A)**, và **null(A)**, cách phân tích thừa số **QR** đươc dùng như mô tả trong chương **9** sách hướng dẫn **LINPACK** .

Với **pinv(A)**, thuật toán dựa vào cách phân tích giá trị kỳ dị và được mô tả trong chương **11** sách hướng dẫn **LINPACK** . thuật toán **pinv** tốn thời gian nhất, nhưng đáng tin cậy nhất và do đó cũng được dùng để tính hạng ma trận, **rank(A)**.

****

Chương 7. ĐA THỨC VÀ XỬ LÝ TÍN HIỆU

7.1. Đa thức

Các đa thức biểu hiện trong **MATLAB** như các vectơ dòng chứa các hệ số theo lũy thừa thoái. Ví dụ, Phương trình đặc trưng của ma trận

```
A =

   1   2   3

   4   5   6

   7   8   9
```

được tính với

```
p = poly(A)

p =

   1   -6   -72   -27
```

Đây là biểu hiện **MATLAB** của ma trận $s^3 - 6s^2 - 72s - 27$.

Các nghiệm của phương trình này là:

```
r = roots(p)

r =

   12.1229
   -5.7345
   -0.3884
```

Tất nhiên giống các giá trị riêng của ma trận **A**. Có thể ráp ngược về đa thức gốc với hàm **poly**,

```
p2 = poly(r)

p2 =

   1   -6   -72   -27
```

Xét các đa thức $\mathbf{a}(s) = s^2 + 2s + 3$ và $\mathbf{b}(s) = 4s^2 + 5s + 6$. Tích các đa thức là tích chập các hệ số:

```
a = [1  2  3]; b = [4  5  6];

c = conv(a,b)

c =

   4    13    28    27    18
```

Dùng hàm tách tích chập để chia ngược lại,

```
[q,r] = deconv(c,a)

q =

   4    5    6

r =

   0    0    0    0    0
```

Danh sách đầy đủ các hàm về đa thức gồm:

Đa thức	
poly	Đa thức đặc trưng
roots	Nghiệm đa thức-phương pháp ma trận liên hợp
roots1	Nghiệm đa thức-phương pháp **Laguerre**
polyval	Ước lượng đa thức
polyvalm	Ước lượng đa thức ma trận
conv	Nhân đa thức
deconv	Chia đa thức
residue	Khai triển thừa số từng phần
polyfit	Vẽ đường cong đa thức

7.2. Xử lý tín hiệu

Các vectơ dùng để giữ các tín hiệu dữ liệu mẫu, hoặc chuỗi, cho việc xử lý tín hiệu. Đối với hệ thống nhiều dữ kiện nhập, mỗi dòng ứng với một điểm mẫu, với việc quan sát bảng qua các cột của ma trận. Một vài hàm xử lý tín hiệu được đưa vào hệ thống chính của **MATLAB**:

Xử lý tín hiệu	
abs	Chuẩn của số phức
angle	Góc pha
conv	Tích chập

cov	Hiệp phương sai
deconv	Tách tích chập
fit	Biến đổi Fourier nhanh
ifft	Nghịch đảo biển đổi Fourier nhanh
fftshift	Hoán đổi dạng toàn phương ma trận

Vài hàm có bản sao **2** chiều, trong trường hợp đó "tín hiệu" đúng là một ma trận:

Xử lý tín hiệu 2 chiều	
fft2	**FFT 2** chiều
ifft2	**FFT 2** chiều ngược
fftshift	Sắp xếp lại các kết quả **FFT**
conv2	Tích chập **2** chiều

Có nhiều hàm xử lý tín hiệu nữa có thể sử dụng trong **SIGNAL PROCESSING TOOLBOX**. Phần này có ý giới thiệu sơ bộ về khả năng xử lý tín hiệu của **MATLAB**; để biết thêm thông tin xem riêng sách hướng dẫn sử dụng **SIGNAL PROCESSING TOOLBOX**.

7.3. Lọc dữ liệu

Trong **SIGNAL PROCESSING TOOLBOX**, hàm

y =filter(b,a,x)

lọc dữ liệu trong vectơ **x** với bộ lọc mô tả bởi các vectơ **a** và **b**, tạo ra dữ liệu **y** đã lọc. Cấu trúc lọc là bộ lọc dãy tổng quát mô tả bởi phương trình vi phân:

y(n) = b(1)x(n) + b(2)x(n-1) + ... + b(nb)x(n-nb+1)

-a(2)y(n-1) - ... -a(na)y(n-na+1)

hoặc tương đương phép biến đổi **Z**

$$\mathbf{H(z)} = \frac{\mathrm{Y(z)}}{\mathrm{X(z)}} = \frac{b(1)+b(2)z^{-1}+...+b(nb)z^{-(nb-1)}}{1+a(2)z^{-1}+...+a(na)z^{-(na-1)}}$$

Ví dụ, để tìm và vẽ đơn vị ***n*** điểm đáp ứng bộ lọc:

x =[1 zeros(1,n-1);

y = filter(b,a,x);

plot(y,'o')

Hàm **freqz** trả về kết quả phức của bộ lọc số. Kết quả là hàm **H(z)** ước lượng quanh đơn vị tròn trong mặt phẳng phức, **x =e$^{j\omega}$** . Để dùng **freqz** để tìm và vẽ đường cong thực nghiệm ***n*** điểm:

[h,w] = frqz(b,a,n);

mag = abs(h);

phase = angle(h);

semilogy(w,mag), plot(w,phase)

Các hàm có thể sử dụng trong **SIGNAL PROCESSING TOOLBOX** dùng để thiết kế bộ lọc số. Chúng tôi đưa nội dung vào đây để yêu cầu một số kiến thức về kỹ thuật thiết kế bộ lọc, có thể dùng nhiều phương pháp. Ví dụ, các hàm số học về số phức cho phép về kỹ thuật giống như biến đổi song tuyến tính và vẽ cực **0** để đổi sang các nguyên mẫu **s-phạm trù** sang **z-phạm trù**. Cũng như thế, các bộ lọc **FIR** được thiết kế một cách dễ dàng bằng kỹ thuật về cửa sổ.

7.4. FFT(Fast Fourier Transform-Biến đổi Fourier nhanh)

Phải nói rằng thuật toán **FFT** chủ yếu dùng cho việc tính toán phép biến đổi **Fourier** của chuỗi là thích hợp của việc xử lý tín hiệu số. Miền giá trị sử dụng của nó từ việc lọc dữ liệu, tích chập, tính toán các yêu cầu thường xuyên đến các trình ứng dụng trong việc ước lượng quang năng.

Hàm **fft(x)** là phép biến đổi **Fourier** của vectơ **x**, tính toán biến đổi **Fourier** cơ số **2** nhanh nếu độ dài của **x** là bội lũy thừa của **2**, và với thuật toán chuyển cơ số nếu độ dài của **x** không phải là bội lũy thừa của **2**. Nếu **X** là ma trận thì **fft(X)** là biến đổi **Fourier** nhanh của mỗi cột của **X**.

Hàm **fft(x,n)** là **FFT n-điểm**. Nếu độ dài của **x** nhỏ hơn ***n*** thì **x** được thêm với đuôi các số **0** thành độ dài ***n***. Nếu độ dài của **x** lớn hơn ***n*** thì **x** bị cắt phần đuôi. Khi **X** là ma trận thì độ dài các cột của **X** được chỉnh lý theo cùng cách này.

Hàm **ifft(x)** là phép biến đổi **Fourier** ngược của vectơ **x**, hàm **ifft(x,n)** là **FFT** ngược **n-điểm**.

Cặp hai hàm cài đặt phép biến đổi và biến đổi ngược cho bởi:

$$\mathbf{X}(k+1) = \sum_{n=0}^{N-1} x(n+1) W_N^{kn}$$

$$\mathbf{x}(n+1) = 1/N\sum_{k=0}^{N-1} X(k+1)W_N^{-kn}$$

ở đây $W_N = e^{-j\,(2\pi/N)}$ và N = ***length(x)***. Lưu ý rằng chỉ số được viết theo cách không chính tắc chạy đến ***n+1*** và ***k+1*** thay vì đến ***n*** và ***k*** bình thường, vì các vectơ của **MATLAB** chạy từ **1** đến ***N*** thay vì từ **0** đến ***N-1***.

Giả sử một dãy độ dài chẵn gồm ***N*** điểm có cùng tần số mẫu của f_S . Sau đó chuyển sang tần số **Nyquist**, hoặc điểm ***n =N/2+1***, thì quan hệ giữa số nhị phân và tần số thực là:

$$f = (bin\_number - 1) * f_S / N$$

FFT của vectơ cột **x**

```
x = [ 4  3  7  -9  1  0  0  0]';
```

tìm thấy với

```
y = fft(x)
```

kết quả là

```
y =

   6.0000

  11.4853 -   2.7574 i

  -2.0000  -  12.0000 i

  -5.4853 +  11.2426 i

  18.0000

  -5.4853 -  11.2426 i

  -2.0000 +  12.0000 i

  11.4853 +   2.7574 i
```

Lưu ý rằng mặc dù dãy **x** là thực, nhưng **y** lại là phức. Thành phần thứ nhất của dữ liệu biến đổi là đóng góp **DC** và phần tử thứ năm tương ứng với tần số **Nyquist**. Ba giá trị cuối cùng của **y** tương ứng các tần số âm và, đối với dãy số thực **x**, chúng là các liên hợp phức của **y(4)**, **y(3)** và **y(2)**.

Để biết thêm thông tin thì xem phần tham khảo. Nếu làm nhiều với việc xử lý tín hiệu thì xem sách hướng dẫn sử dụng **SIGNAL PROCESSING TOOLBOX**.

****

Chương 8. HÀM CÓ ĐỐI SỐ LÀ HÀM

Một lớp các hàm trong **MATLAB** không làm việc với các ma trận số mà với các hàm toán học. Các *hàm có đối số là hàm* này gồm:

Tích phân số

Phương trình và tối ưu phi tuyến

Giải phương trình vi phân

Các hàm toán học được biểu hiện trong **MATLAB** bởi các tệp M-file về hàm. Ví dụ, hàm

$$\textbf{\textit{humps}(x)} = \frac{1}{(x-.3)^2+.01} + \frac{1}{(x-.9)^2+.04} - 6$$

được tạo ra có thể dùng trong **MATLAB** bằng cách tạo ra tệp M-file có tên là **humps.m**:

```
function y = humps(x)
y = 1 ./ ((x-.3).^2+.01) + 1 ./ ((x-.9).^2 +.04) - 6;
```

Đồ thị của hàm là:

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

Kết quả là hình 8.1.

8.1. Tích phân số

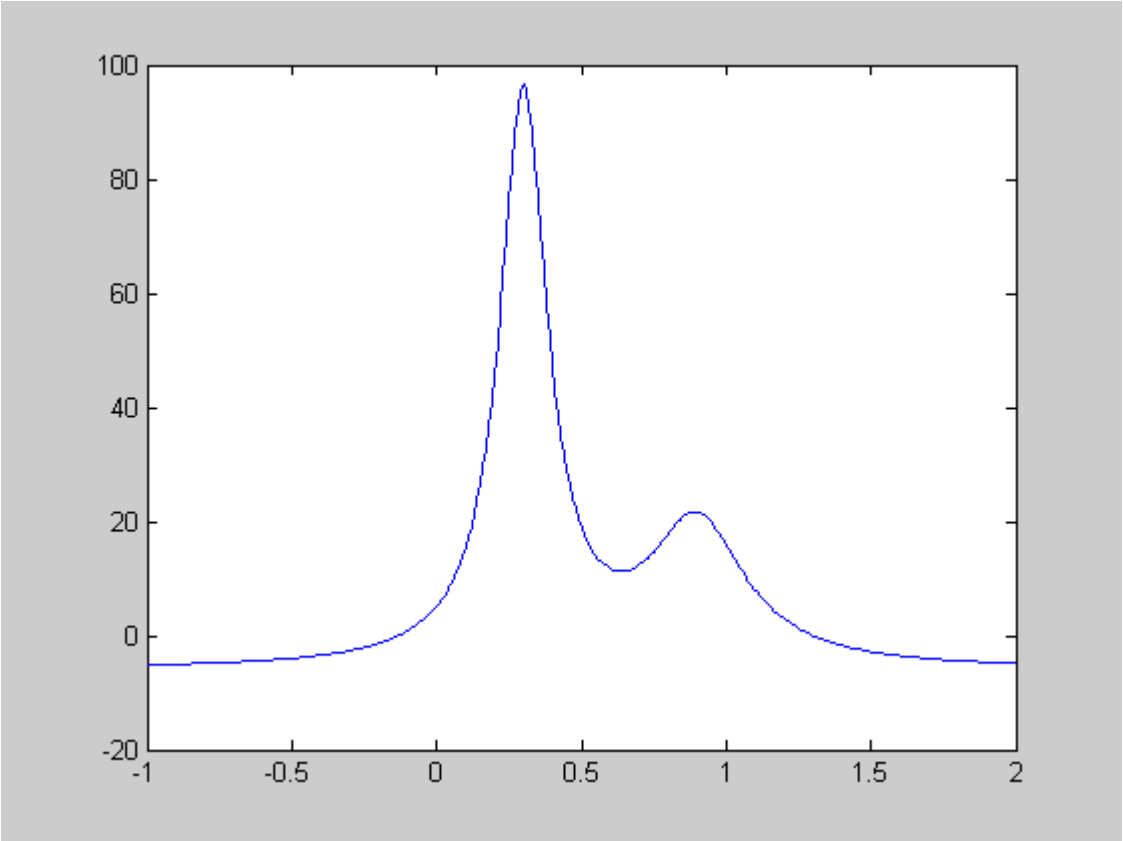
Diện tích vùng dưới hàm **humps(x)** có thể được xác định bằng tích phân hàm **humps(x)**, một cách xử lý xem như phép cầu phương. Để lấy tích phân hàm **humps** từ **0** đến **1**:

```
q = quad('humps',0,1)
q =
   29.8583
```

Hai hàm của **MATLAB** để tính cầu phương là:

Tích phân số	
quad	Phương pháp **Simson**
quad8	Phương pháp **Newton**

Lưu ý rằng đối số thứ nhất của hàm **quad** là xâu chữ đặt trong cặp dấu nháy chứa tên của một hàm. Điều này cho thấy tại sao gọi **quad** là một *hàm có đối số là hàm* - là một hàm tính toán trên các hàm khác.



Hình 8.1

8.2. Phương trình và tối ưu phi tuyến

Các hàm về hàm dùng cho phương trình và tối ưu phi tuyến gồm:

Phương trình và tối ưu phi tuyến	
fmin	Cực tiểu của hàm một biến
fmins	Cực tiểu của hàm nhiều biến (Tối ưu phi tuyến không ràng buộc)
fsolve	Lời giải của hệ phương trình phi tuyến (giá trị không của hàm nhiều biến)
fzero	Giá trị không của hàm một biến

Tiếp tục ví dụ, vị trí của giá trị cực tiểu của hàm **humps(x)** trong miền từ **0.5** đến **1** được tính với **fmin**:

```
xm = fmin('humps', .5, 1)

xm =

    0.6370
```

Giá trị của nó ở điểm cực tiểu là:

```
ym = humps(xm)

ym =

   11.2528
```

Theo đồ thị, rõ ràng **humps** có **2** giá trị **0**. Vị trí của giá trị **0** gần **x = 0** là:

```
xz1 = fzero('humps', 0)

xz1 =

   -0.1316
```

Vị trí của giá trị **0** gần **x = 1** là:

```
xz2 = fzero('humps', 1)

xz2 =

    1.2995
```

8.3. Phương trình vi phân

Các hàm của **MATLAB** để giải *phương trình vi phân thường* là:

Giải phương trình vi phân	
ode23	Phương pháp **Runge-Kutta** cấp 2/3
ode45	Phương pháp **Runge-Kutta-Fehlberg** cấp 4/5

Xét phương trình vi phân cấp **2 Van der Pol**

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

Có thể viết lại phương trình này như một hệ gồm cặp phương trình vi phân cấp một:

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

$$\dot{x}_2 = x_1$$

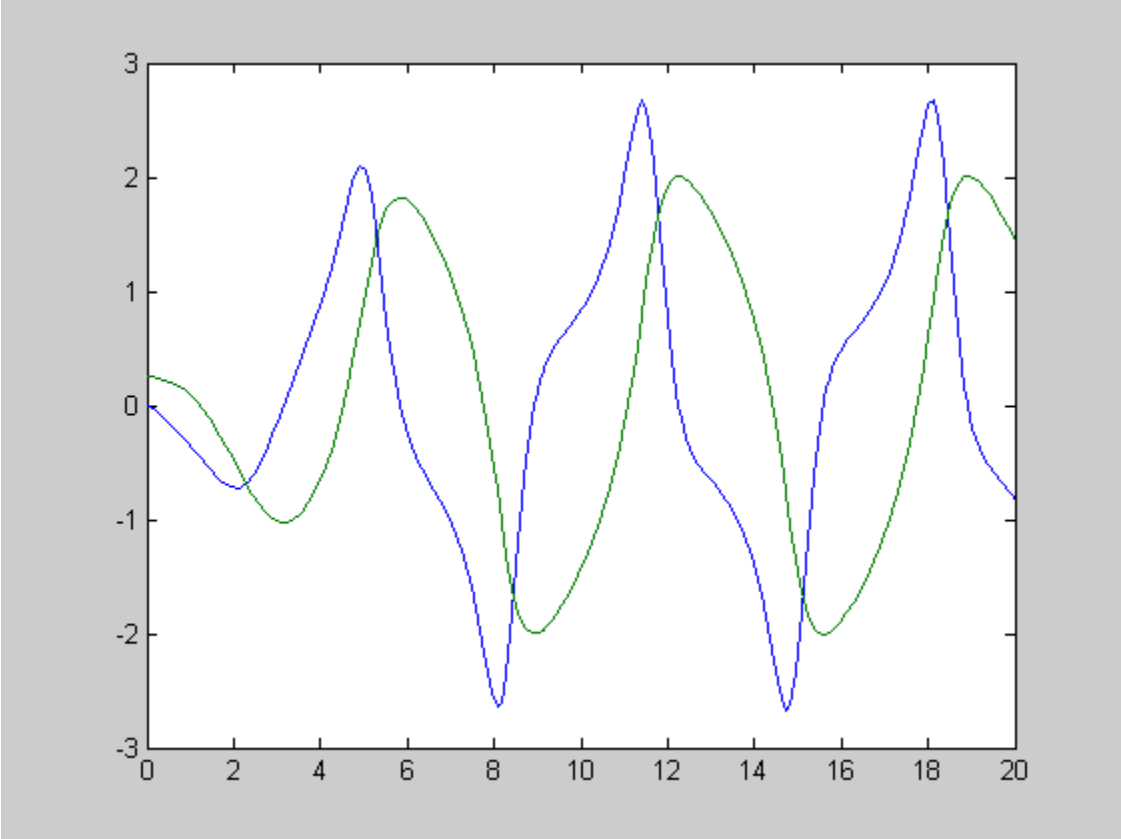
Bước thứ nhất đối với việc mô phỏng hệ này là tạo ra một tệp M-file hàm chứa các phương trình vi phân này. Có thể gọi nó là **vdpol.m**:

```
function xdot = vdpol(t,x)

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

xdot(2) = x(1);

xdot = xdot'; % phiên bản mới yêu cầu vectơ cột
```

Để mô phỏng phương trình vi phân **vdpol** xác định trên đoạn $0 \le t \le 20$, gọi hàm **ode23**

```
t0 = 0; tf = 20;

x0 = [ 0  0.25]'; % các điều kiện khởi đầu

[t,x] = ode23('vdpol', t0, tf, x0);

plot(t,x)
```

Kết quả là hình 8.2.



Hình 8.2

****

Chương 9. ĐỒ THỊ

Dữ liệu về khoa học và kỹ thuật được xét đến ở dạng đồ thị trong MATLAB bằng cách dùng các lệnh về đồ họa để tạo ra hình vẽ trên màn hình. Có nhiều kiểu đồ họa khác nhau có thể chọn:

Đồ thị	
plot	Vẽ đường tuyến tính **x-y**
loglog	Vẽ loga **x-y**
semilogx	Vẽ bán loga **x-y** (loga trục **x**)
semilogy	Vẽ bán loga **x-y** (loga trục **y**)
polar	Vẽ tọa độ cực
mesh	Vẽ mặt lưới **3** chiều
contour	Vẽ đường mức
bar	Vẽ biểu đồ
stairs	Vẽ đồ thị bậc thang

Khi một đồ thị có trên màn hình thì có thể có nhãn, tiêu đề, hoặc các dòng lưới theo:

title	Tiêu đề đồ thị
xlabel	Nhãn trục **x**
ylabel	Nhãn trục **y**
text	Văn bản ở vị trí bất kỳ
gtext	Văn bản ở vị trí chuột
grid	Các dòng lưới

Có các lệnh để điều khiển việc chia trục và đồ thị:

axis	Chia trục
hold	Giữ hình vẽ trên màn hình
shg	Hiện màn hình đồ thị
clg	Xóa màn hình đồ thị
subplot	Chia màn hình đồ thị thành các cửa sổ con
ginput	Dấu chữ thập cho chuột

Và có các lệnh để đưa bản sao ra máy in:

print	Đưa đồ thị ra máy in
prtsc	In đồ thị màn hình
meta	Tạo siêu tệp đồ thị

9.1. Hình vẽ trong mặt phẳng x-y

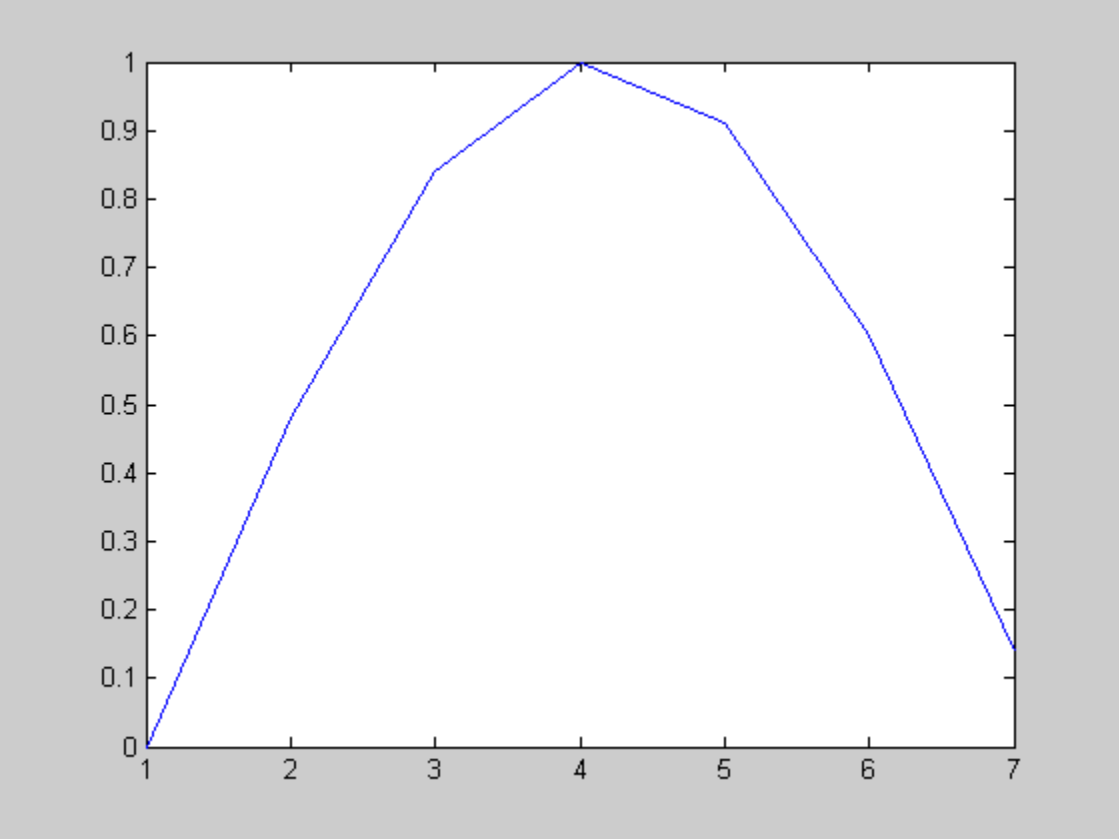
Lệnh **plot** tạo ra các hình vẽ mặt phẳng **x-y**. Khi lệnh **plot** là chủ thì các hình vẽ loga và cực được tạo ra bằng cách thay các từ **loglog, semilogx, semilogy**, hoặc **polar** cho từ **plot**. Cả **5** lệnh được dùng cùng một cách; chúng chỉ ảnh hưởng đến cách chia trục và cách hiển thị dữ liệu.

9.2. Dạng thức cơ bản

Nếu **Y** là một vectơ thì lệnh **plot(Y)** cho ra một hình vẽ gồm các phần tử của **Y** đối số là chỉ số của các phần tử của **Y**. Ví dụ, để vẽ các số **{0., .48, .84, 1, .91, .6, .14}**, nhập chúng vào một vectơ và thực hiện lệnh **plot**:

```
Y = [0.   .48  .84  1.   .91   .6   .14];

plot(Y)
```

Kết quả là hình 9.1.



Hình 9.1

Lưu ý rằng dữ liệu được chia trục tự động và các trục **X** và **Y** được vẽ. Ở điểm này, tùy theo phần cứng của máy sử dụng mà màn hình có đáp ứng các lệnh đánh vào hay không. **MATLAB** có hai màn hình, một màn hình đồ họa và một màn hình lệnh. Một số cấu hình phần cứng cho phép cả hai màn hình hiện đồng thời, trong khi một số khác chỉ hiện mỗi lúc một màn hình. Nếu màn hình lệnh không còn ở đó nữa thì có thể quay lại bằng cách ấn một phím bất kỳ.

Khi màn hình lệnh đã quay lại thì một tiêu đề đồ thị, nhãn trục **X** và nhãn trục **Y**, và các dòng lưới có thể đặt vào hình vẽ bằng cách nhập liên tục vào các lệnh

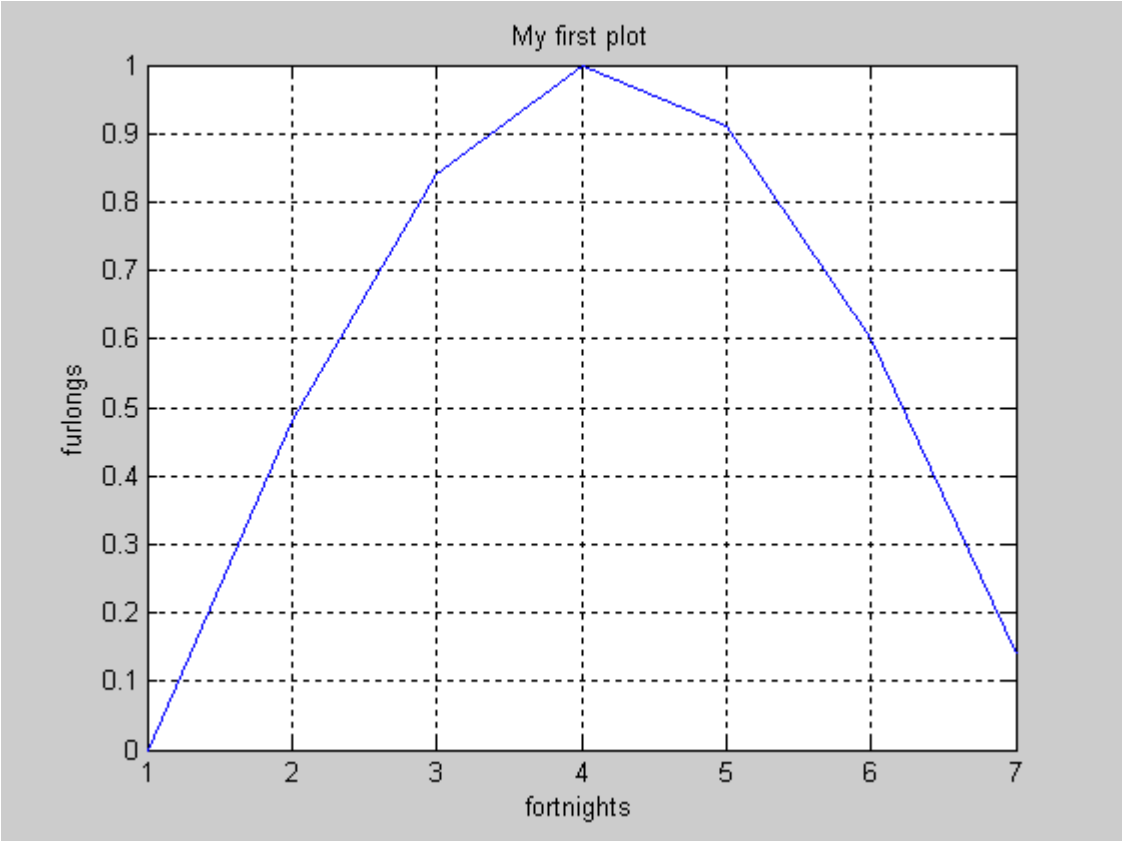
title('My first plot')

xlabel('fortnights')

ylabel(furlongs')

grid

Kết quả là hình 9.2.



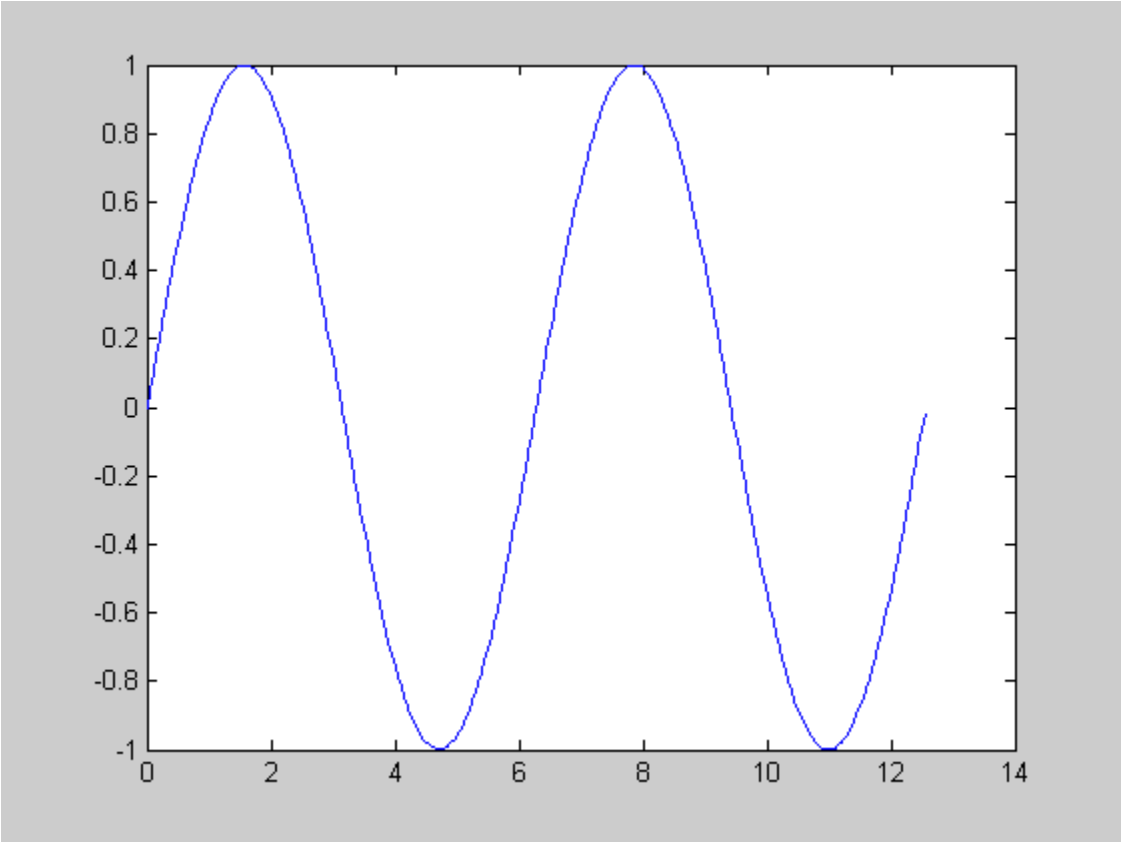
Hình 9.2

Hàm **gtext('text')** cho phép chuột hoặc các phím mũi tên định vị bằng một dấu chữ thập trên đồ thị, ở điểm mà văn bản sẽ đặt khi có phím hoặc nút chuột được nhấn.

Nếu **X** và **Y** là các vectơ cùng độ dài, thì lệnh **plot(X,Y)** vẽ hình vẽ **x-y** gồm các phần tử của **X** đối số là các phần tử của **Y**. Ví dụ,

```
t = 0:.05:4*pi;

y = sin(t);

plot(t,y)
```

Kết quả là hình 9.3.



Hình 9.3

9.3. Nhiều đường

Có hai cách để vẽ nhiều đường trên một đồ thị đơn. Thứ nhất là cho lệnh **plot** với **2** đối số, như **plot(X,Y)**, ở đây hoặc là **X**, hoặc là **Y**, hoặc là cả hai là ma trận. Sau đó:

[1] Nếu **Y** là ma trận và **X** là vectơ, thì **plot(X,Y)** vẽ liên tục các dòng hoặc các cột của **Y** đối số là vectơ **X**, dùng kiểu đường khác nhau cho mỗi dòng hoặc cột. Việc "định hướng" dòng hay cột của **Y** được chọn để có cùng số phần tử như vectơ **X**. Nếu **Y** là ma trận vuông thì tự chọn hướng cột.

[2] Nếu **X** là ma trận và **Y** là vectơ, thì các quy tắc trên được áp dụng, ngoại trừ các đường từ **X** được vẽ đối số là vectơ **Y**.

[3] Nếu cả **X** và **Y** là ma trận cùng cỡ, thì **plot(X,Y)** vẽ các cột của **X** đối số là các cột của **Y**.

[4] Nếu không chỉ định **X**, như **plot(Y)**, ở đây **Y** là ma trận, thì các đường được vẽ cho mỗi cột của **Y** đối số là chỉ số dòng.

Cách thứ hai và dễ dàng hơn để vẽ nhiều đường trên một đồ thị đơn là dùng lệnh **plot** với nhiều đối số:

plot(X1, Y1, X2, Y2, ..., Xn, Yn)

Các biến **X1, Y1, X2, Y2, ...** là các cặp vectơ. Mỗi cặp **x-y** đượoc vẽ, phát sinh ra nhiều đường trên đồ thị. Phương pháp nhiều đối số có điều thuận lợi là cho phép các vectơ có độ dài khác nhau hiển thị trên cùng một đồ thị. Như trước đây, mỗi cặp dùng một kiểu đường khác nhau.

9.4. Kiểu đường và kiểu điểm

9.4.1. Kiểu

Kiểu đường dùng trong đồ thị có thể điều khiển nếu không thỏa mãn kiểu ngầm định . Cũng có thể vẽ điểm bằng các ký hiệu khác nhau. Ví dụ,

plot(X,Y,'x')

vẽ một hình vẽ điểm bằng cách dùng các dấu **x** trong khi

plot(X1,Y1,':',X2,Y2,'+')

dùng đường chấm chấm cho đường cong thứ nhất và dấu + cho đường cong thứ hai. Các kiểu đường và kiểu điểm khác là:

Kiểu đường		Kiểu điểm	
đặc	-	dấu chấm	.
gạch	--	dấu cọng	+
chấm	:	dấu sao	*
chấm gạch	-.	dấu tròn	o
		dấu x	x

9.4.2. Màu

Trong hệ thống có cung cấp màu, thì màu đường và màu điểm có thể chỉ định theo cách tương tự kiểu đường và kiểu điểm. Ví dụ, các lệnh

plot(X,Y,'r')

plot(X,Y,'+g')

dùng màu đỏ cho đồ thị thứ nhất và dấu + màu xanh cho đồ thị thứ hai. Các màu khác là:

Màu	
đỏ	**r**
xanh lá cây	**g**
xanh nước biển	**b**
trắng	**w**
ẩn	**i**

Nếu thiết bị phần cứng không cung cấp màu, thì các màu khác nhau trên màn hình làm cho các kiểu đường vẽ sẽ khác nhau.

9.5. Dữ liệu ảo và phức

Khi đối số của lệnh **plot** là phức (có phần ảo khác không), thì phần ảo được bỏ qua ngoại trừ khi **plot** được cho một đối số phức đơn. Đối với trường hợp đặc biệt này, thì kết quả là hình vẽ tắt của hàm phần thực đối số là phần ảo. Do đó lệnh **plot(Z)**, khi **Z** là một vectơ phức hoặc ma trận phức thì tương đương lệnh **plot(real(Z),imag(Z))**.

Để vẽ nhiều đường trong mặt phẳng phức thì không có cách vẽ tắt, và các phần thực và ảo phải chỉ định rõ ràng.

9.6. Hình vẽ loga, cực, và biểu đồ

Cách dùng các lệnh **loglog, semilogx, semilogy**, và **polar** là giống như lệnh **plot**. Các lệnh này cho phép dữ liệu được vẽ theo các kiểu khác nhau, nghĩa là trong các hệ tọa độ khác nhau:

- **polar(theta, rho)** là hình vẽ trong hệ tọa cực của góc **theta**, theo đơn vị **radian**, đối số là bán kính **rho**. Sau đó dùng lệnh **grid** để vẽ các lưới cực.

- **loglog** là hình vẽ dùng đơn vị chia trục $\log_{10}$ - $\log_{10}$.

- **semilogx** là hình vẽ dùng đơn vị chia trục bán loga. Trục **x** là **log10** trong khi trục **y** là tuyến tính.
- **semilogy** là hình vẽ dùng đơn vị chia trục bán loga. Trục **y** là **log10** trong khi trục **x** là tuyến tính.

Lệnh **bar(x)** hiển thị biểu đồ thanh của các phần tử của vectơ **x**, lệnh **bar** không chấp nhận nhiều đối số. Tương tự, nhưng bỏ qua các đường đứng là lệnh **stairs**, cho ra hình vẽ bậc thang là hữu ích cho việc vẽ biểu đồ hệ dữ liệu mẫu.

9.7. Vẽ mặt lưới 3 chiều và đường mức

Lệnh **mesh(Z)** tạo ra hình vẽ phối cảnh **3** chiều của các phần tử trong ma trận **Z**. Mặt lưới được xác định bởi các tọa độ **Z** của các điểm bên trên lưới chữ nhật trong mặt phẳng **x-y**. Hình vẽ được định dạng bằng cách nối các điểm kề nhau bằng các đoạn thẳng.

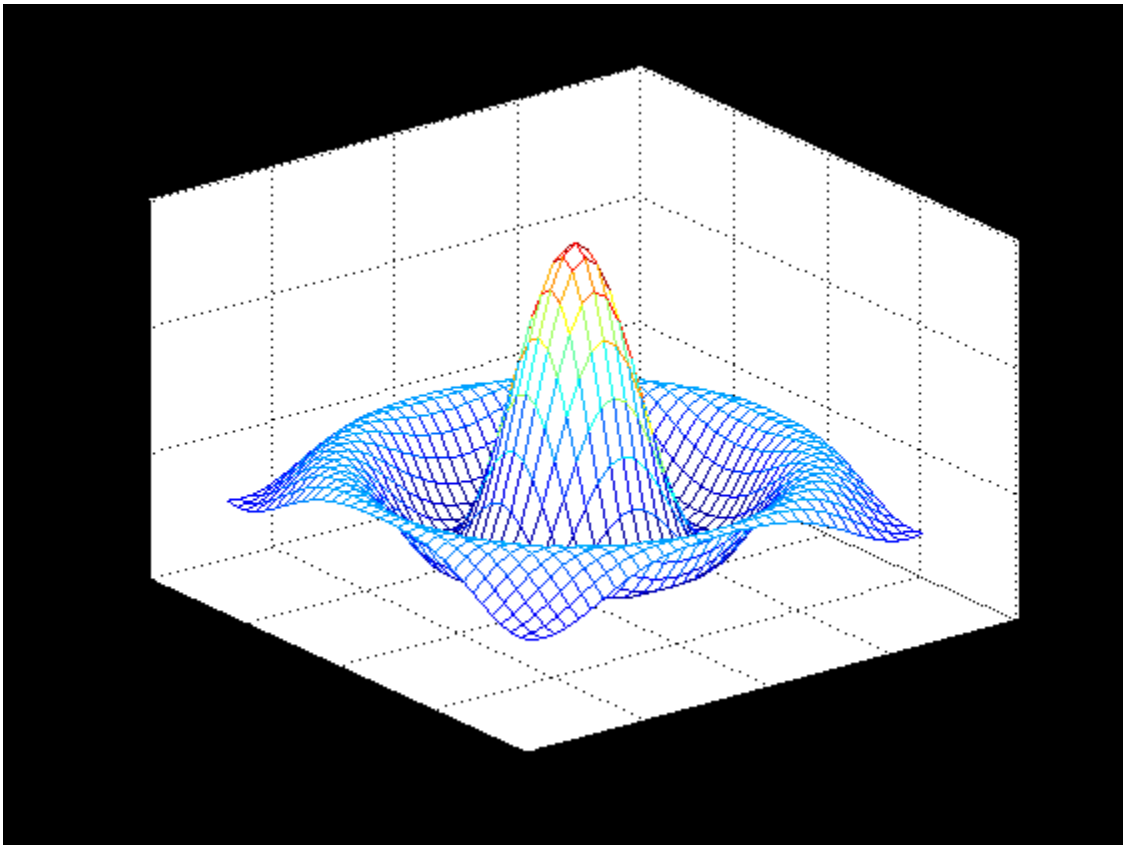
Lệnh **mesh** có thể dùng để xem các ma trận lớn, mà nếu in ra ở dạng thức số thì quá lớn. Nó cũng có thể dùng để vẽ các hàm hai biến.

Bước thứ nhất trong việc hiển thị hàm hai biến **z = f(x,y)** là phát sinh các ma trận **X** và **Y** gồm các dòng và các cột lặp tương ứng trên miền giá trị của hàm. Sau đó hàm có thể được tính toán trực tiếp và vẽ.

Xét hàm **sin(r)/r** hay **sinc** mà kết quả là mặt mũ phớt rộng vành mà mọi người ưa nhìn Một cách tạo ra là:

```
x = -8:.5:8;

y = x';

X = ones(size(y))*x;

Y = y*ones(size(x));

R = sqrt(X .^2 + Y  .^2) + eps;

Z = sin(R)./R;

mesh(Z)
```

Lệnh thứ nhất xác định miền giá trị **x** mà trên đó hàm được ước lượng. Lệnh thứ ba tạo ra ma trận **X** gồm các dòng lặp. Sau khi phát sinh ma trận **Y** tương ứng, ma trận **R** được tạo ra chứa khoảng cách từ tâm của ma trận, đó là gốc. Việc định dạng hàm **sinc** và áp dụng lệnh **mesh** kết quả là hình 9.4.



Hình 9.4

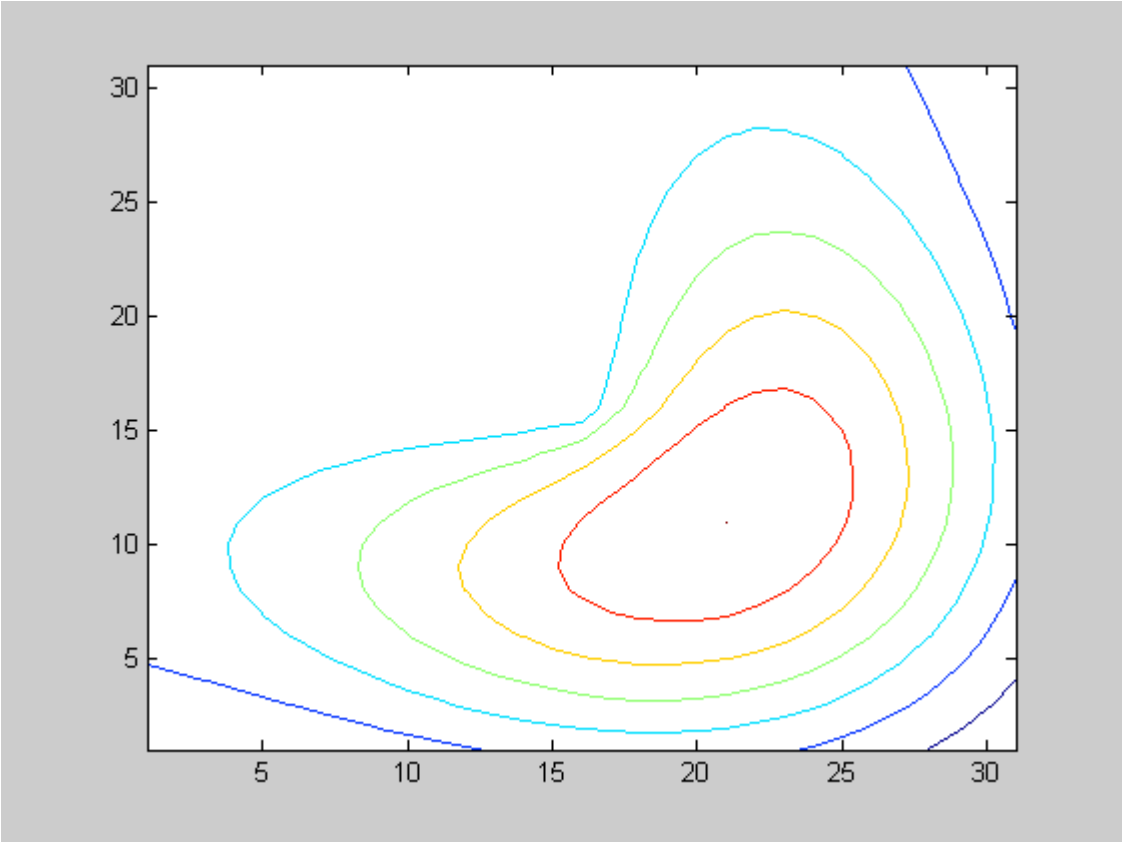
Một ma trận đơn vị trông như mặt lưới gì ? Hãy thử lệnh **mesh(eye(14))**. Với một phương pháp dễ dàng phát sinh các ma trận đặc biệt **X** và **Y** đòi hỏi để ước lượng hàm hai biến, xem lệnh **meshdom** trong phần tham khảo.

Xen vào vẽ lưới là vẽ đường mức để xem nội dung của ma trận. Một đường mức của màng dạng **L** trong sách hướng dẫn này là

z = membrane(1, 15, 9, 2);

contour(z)

Kết quả làhình 9.5.



Hình 9.5

9.8. Điều khiển màn hình

MATLAB có **2** màn hình, một cửa sổ đồ họa và một của sổ lệnh. Cấu hình phần cứng có thể cho phép cả hai màn hình thấy được đồng thời trên **2** cửa sổ khác nhau, hoặc có chỉ cho phép thấy mỗi lúc một cửa sổ. Một số lệnh dùng để chuyển qua lại giữa **2** cửa sổ, và/hoặc xóa các cửa sổ theo yêu cầu:

shg	Hiện cửa sổ đồ họa
any key	Quay ngược lại cửa sổ lệnh
clc	Xóa cửa sổ lệnh
clg	Xóa cửa sổ đồ họa
home	Đưa con trỏ lệnh về đầu dòng

Ví dụ, nếu trong lúc **MATLAB** đang làm việc mà chỉ có màn hình lệnh trên màn hình, thì vào lệnh **shg** sẽ gọi lại hình vẽ cuối cùng đã vẽ trên màn hình đồ họa.

Ngầm định, với cấu hình phần cứng không hiển thị cả hai màn hình lệnh và màn hình đồ họa đồng thời sẽ, tạm dừng trong chế độ vẽ sau khi vẽ xong và chờ ấn phím.

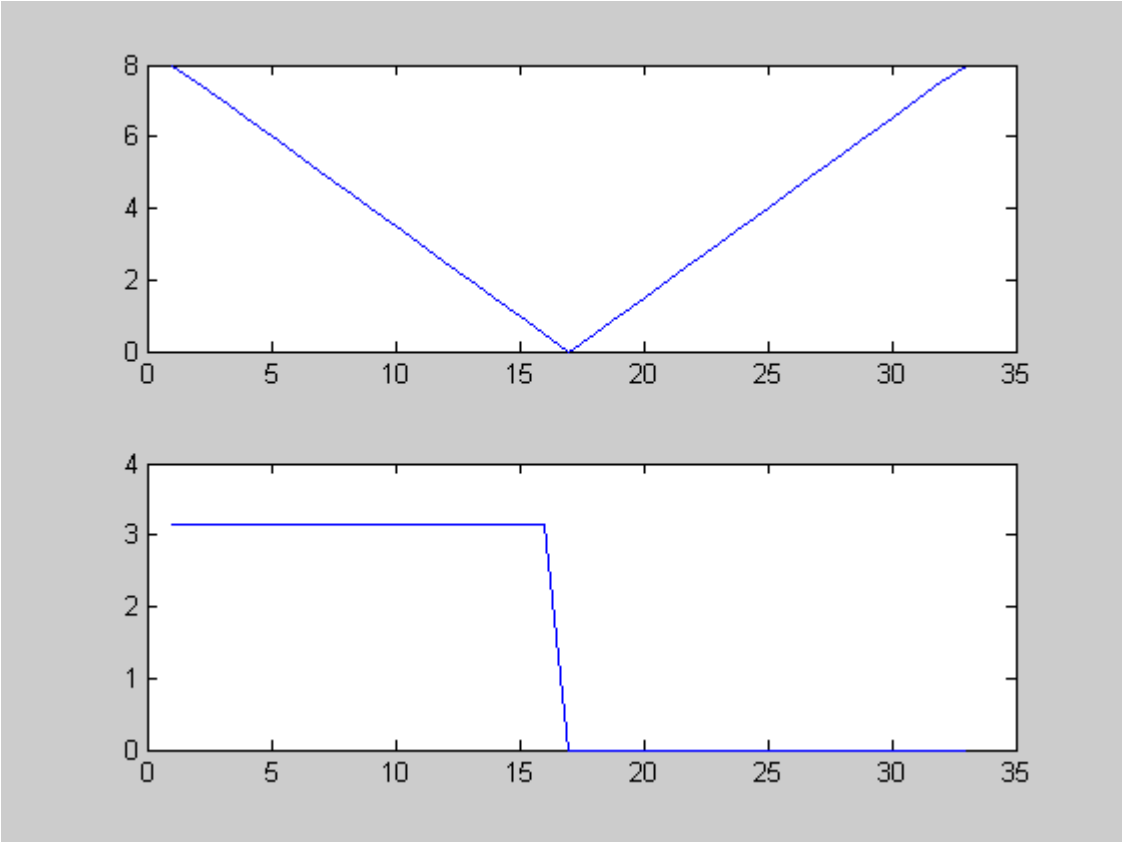
Có thể tách cửa sổ đồ họa thành nhiều phần, nhằm để xem một số hình vẽ cùng một lúc. Lệnh **subplot(m,n,p)** cắt cửa sổ đồ họa thành ***mxn*** lưới và dùng hộp thứ ***p*** cho hình vẽ tiếp sau. Ví dụ,

subplot(2, 1, 1), plot(abs(y))

subplot(2, 1, 2), plot(angle(y))

cắt màn hình thành hai, vẽ độ dài của vectơ phức trong nữa trên, và vẽ góc pha trong nữa dưới. Xem hình 9.6.

Lệnh **subplot(1, 1, 1)**, hoặc đúng **subplot**, trở về cửa sổ đơn ngầm định là toàn màn hình.



Hình 9.6

9.9. Cách chia đơn vị trục tọa độ

Trong trường hợp nào đó, có thể muốn đè lên đặc tính chia trục ngầm định của lệnh vẽ và chọn giới hạn vẽ. Việc thực hiện lệnh **axis**, chính nó giữ lại cách chia đơn vị trục hiện thời cho các hình vẽ sau. Vào lệnh **axis** lần nữa tiếp tục chia tự động. Hàm **axis** trả về vectơ dòng gồm **4** phần tử chứa **[x_min, x_max, y_min, y_max]** từ hình vẽ cuối cùng. Lệnh **axis(V)**, ở đây **V** là vectơ **4** phần tử, đặt cách chia trục vào các giới hạn chỉ định.

Dùng lần thứ hai lệnh **axis** là để điều khiển tỉ lệ phân giải của hình vẽ trên màn hình. Lệnh **axis('square')** đặt vùng vẽ trên màn hình là hình vuông. Với tỉ số phân giải vuông thì một đường thẳng có hệ số góc **1** đúng là **45 độ**, không bị lệch bởi hình dáng không đều của màn hình. Cũng vậy, các đường tròn, như **plot(sin(t),cos(t))**, giống đường tròn thay vì đường ô-van. Lệnh **axis('normal')** đặt tỉ lệ phân giải về giá trị chuẩn.

Lệnh **hold** giữ hình vẽ hiện thời trên màn hình. Lệnh **plot** tiếp theo sẽ thêm vào hình vẽ, dùng các giới hạn về trục đã thiết lập và duy trì các đường cong đã vẽ trước đó. Lệnh **hold** vẫn còn hiệu lực cho đến khi được gọi lại.

9.10. Bản sao phần cứng

Ba lệnh **prtsc, print** và **meta**, cung cấp các khả năng về phần cứng nói chung:

prtsc **prtsc** khởi động liệt kê cửa sổ đồ họa, như **Shift-Prtsc**, và cho phép thực hiện trong tệp M-file hoặc trong vòng lặp **for**. Nói chung, kết quả này theo hình vẽ phân giải thấp, vì điểm ảnh trên màn hình chuyển thành điểm ảnh trên máy in.

meta **meta** mở một siêu tệp đồ họa, dùng tên tệp chỉ định, và ghi hình vẽ hiện thời vào đó để xử lý về sau. Lệnh meta tiếp sau nối vào tên tệp đã chỉ định trước đó. Siêu tệp có thể được xử lý sau đó, dùng các chương trình xử lý hình vẽ.

print **print** đưa bản sao phân giải cao của hình vẽ hiện thời ra máy in. Một số máy có giới hạn về bộ nhớ thì lệnh này không thực hiện được.

Cách dễ nhất để nhận hình vẽ trên màn hình trên mọi máy tính cá nhân là giữ phím **Shift** và ấn phím **Prtsc**. Hoạt động này đưa hình ảnh trong màn hình đồ họa ra máy in.

Xem phần đầu về đặc tả máy của sách hướng dẫn này để biết thêm thông tin về cách nhận bản sao phần cứng.

****

Chương 10. ĐIỀU KHIỂN LUỒNG

MATLAB có các lệnh điều khiển luồng như đã tìm thấy trong hầu hết các ngôn ngữ máy tính. Các lệnh điều khiển luồng đưa **MATLAB** sang cấp độ khác máy tính tay, cho phép nó được dùng như một ngôn ngữ bậc cao về ma trận.

10.1. Vòng lặp FOR

MATLAB có phiên bản riêng của nó về vòng lặp "**DO**" hoặc "**FOR**" tìm thấy trong các ngôn ngữ máy tính. Nó cho phép một câu lệnh, một nhóm lệnh, được lặp lại một số lần cố định xác định trước. Ví dụ

```
for  i = 1:n, x(i) = 0, end
```

gán **0** vào ***n*** phần tử đầu của **x**. Nếu ***n*** nhỏ hơn **1** thì lệnh vẫn hợp pháp, nhưng câu lệnh bên trong không được thực hiện. Nếu **x** chưa có, hoặc có ít hơn ***n*** phần tử thì không gian thêm vào được tự động phân phối.

Có thể tổ hợp các vòng lặp và thường thụt vào để dễ đọc.

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

Dấu chấm phẩy cuối câu lệnh bên trong vòng lặp để hủy việc in lặp ra màn hình, trong khi lệnh **A** sau vòng lặp hiển thị kết quả cuối cùng.

Một điển quan trong là: mỗi vòng lặp **for** phải gắn với từ khóa **end**. Nếu đơn giản vào lệnh

```
for  i = 1:n, x(i) = 0
```

thì hệ thống sẽ kiên nhẫn chờ nhập các lệnh còn lại trong thân vòng lặp. Không có gì xảy ra đánh vào **end**.

Một ví dụ khác, giả sử

```
t =

  -1
   0
   1
   3
   5
```

và muốn phát sinh một ma trận **Vandermonde**, ma trận có các cột là lũy thừa các phần tử của **t.**

```
A =

    1    -1     1    -1     1

    0     0     0     0     1

    1     1     1     1     1

   81    27     9     3     1

  625   125    25     5     1
```

ở đây vòng lặp kép rõ ràng nhất.

```
n =  max(size(t));

for  j = 1:n

   for  i = 1:n

        A(i,j) = t(i)^(n-j);

   end

end
```

Nhưng vòng lặp đơn với các phép toán trên vectơ có ý nghĩa hơn và cũng minh họa cho vấn đề vòng lặp **for** có thể chạy ngược.

```
A(:,n) = ones(n,1);

for  j = n-1:-1:1

   A(:,j) = t .* A(:,j+1);
```

```
end
```

Dạng tổng quát của vòng lặp **for** là

```
for  v = expression
   statements
end
```

expression đúng là một ma trận, vì đúng là trong **MATLAB**. Các cột của ma trận được gán từng cột vào biến **v** và rồi các lệnh **statements** được thực hiện. Một cách rõ ràng hơn của việc hoàn thành cùng công việc này là

```
E = expression;
[m,n] = size(E);
for  j = 1:n
   v = E(:,j);
   statements
end
```

Thông thường **expression** là loại như **m:n**, hoặc **m:i:n**, đó là ma trận chỉ có một dòng, và bởi vậy các cột của nó đơn giản là các đại lượng vô hướng. Trong trường hợp đặc biệt này, vòng lặp **for** giống như các vòng lặp "**FOR**" hay "**DO**" của các ngôn ngữ lập trình khác.

10.2. Vòng lặp WHILE

MATLAB cũng có phiên bản về vòng lặp "**WHILE**", cho phép một lệnh hoặc nhóm lệnh lặp lại với số lần không xác định, dưới điều khiển của một điều kiện logic. Sau đây là bài toán đơn giản minh họa cho vòng lăp **while**. Số nguyên ***n*** đầu tiên là số nào để **n!** (***n*** giai thừa) là một số gồm **100** chữ số ? Vòng lặp **while** sau đây sẽ tìm ra nó. Nếu chưa biết câu trả lời thì có thể chạy các lệnh này

```
n = 1;
while(prd(1:n)<1.e100, n = n+1; end
n
```

Một minh họa tính toán có tính ứng dụng cao hơn về vòng lặp **while** là tính hàm mũ của một ma trận, trong **MATLAB** gọi là **expm(A)**. Một định nghĩa có thể có của hàm mũ là chuỗi lũy thừa,

expm(A) = 1 + A + A^2/2! + A^3/3! + ...

Lý do để dùng cách tính toán thực sự này là với số phần tử của **A** không lớn lắm. ý tưởng này là tính tổng nhiều hạng tử của chuỗi này là cần thiết để cho ra kết quả không đổi nếu một số hạng tử nữa thêm vào độ chính xác số học của máy. Trong vòng lặp sau, **A** là ma trận đã cho, **E** sẽ thành lũy thừa mong muốn, **F** là hạng tử riêng trong chuỗi, và **k** là chỉ số của hạng tử đó. Các lệnh trong vòng lặp được lặp lại cho đến khi **F** là quá nhỏ để việc thêm nó vào **E** không làm thay đổi giá trị của **E**.

```
E = zeros(A);

F = eye(A);

k =1;

while norm(E+F-E,1)>0

   E = E + F;

   F = A*F/k;

   k = k+1;

end
```

Thay vào đó nếu muốn tính mảng hoặc mũ từng phần tử **exp(A)** thì đúng là phải thay đổi giá trị khởi tạo của **F** từ **eye(A)** thành **ones(A)** và thay đổi tích ma trận **A*F** thành tích mảng **A .*F**.

Dạng tổng quát của vòng lặp **while** là

```
while expression

    statements

end
```

Các lệnh **statements** được thực hiện lặp khi tất cả các phần tử trong ma trận biểu thức **expression** khác không. Ma trận biểu thức thường dùng nhất là một biểu thức quan hệ **1-1**, bởi vậy khác không tương ứng với **TRUE**. Khi ma trận

biểu thức không phải là một đại lượng vô hướng thì có thể thu gọn nó lại bằng các hàm **any** và **all**.

10.3. Các lệnh IF và BREAK

Sau đây là cặp ví dụ minh họa cho lệnh **if** của **MATLAB**. Đầu tiên trình bày cách tính có thể rơi vào **3** trường hợp, tùy thuộc vào dấu và tính chẵn lẻ của **n**.

```
if  n<0
    A = negative(n)
elseif  rem(n,2) == 0
    A = even(n)
else
    A = odd(n)
end
```

Ví dụ thứ hai là một bài toán hấp dẫn trong lý thuyết số. Lấy một số nguyên dương bất kỳ. Nếu chẵn thì chia cho **2**; nếu lẻ thì nhân **3** cọng **1**. Lặp lại tiến trình trên cho đến khi số nguyên bằng **1**. Vấn đề lý thú chưa được chứng minh là: Có số nguyên nào để tiến trình không được kết thúc không ? Chương trình **MATLAB** minh họa cho các câu lệnh **while** và **if**. Cũng trình bày hàm **input** - nhắc người dùng nhập dữ liệu từ bàn phím, và lệnh **break** - cung cấp một việc nhảy ra ngoài vòng lặp.

```
% Bài toán cổ "3n+1" trong lý thuyết số.
while 1
   n = input(' Nhập n, âm để thoát. ');
   if n<=0, break, end
   while n >1
      if rem(n,2)==0
        n = n/2
      else
```

```
            n = 3*n+1
          end;
        end
      end
```

Có thể chương trình này chạy mãi mãi.

****

Chương 11. SIÊU TỆP M-FILE NGUYÊN BẢN VÀ HÀM

MATLAB thường dùng chế độ dòng lệnh; khi nhập một dòng lệnh đơn thì MATLAB thực hiện ngay lập tức và hiển thị kết quả. **MATLAB** cũng có khả năng thực hiện một dãy các lệnh lưu trong một tệp. Hai chế độ này tạo thành một môi trường thông dịch.

Các tệp chứa các lệnh của **MATLAB** gọi là siêu tệp M-file vì chúng có tên mở rộng là "**.m**" ("**.m**" là lựa chọn của **Macintosh**). Ví dụ, một tệp tên là **bessel.m** có thể chứa các lệnh của **MATLAB** để tính các hàm **Bessel**.

Một M-file gồm một dãy các lệnh chuẩn của **MATLAB**, có thể chứa các tham chiếu đến các M-file khác. Một M-file có thể gọi đệ quy đến chính nó.

Một cách dùng M-file là một dãy dài tùy ý các lệnh. Các tệp như thế gọi là các tệp nguyên bản. Một kiểu thứ hai của M-file cung cấp khả năng mở rộng **MATLAB**. Gọi là tệp hàm, chúng cho phép các hàm mới thêm vào các hàm đã có. Nhiều tính năng của **MATLAB** nhận được từ khả năng này để tạo ra các hàm mới để giải các bài toán do người dùng chỉ định.

Cả hai kiểu M-file, nguyên bản và hàm, là các tệp văn bản ASCII bình thường, và được tạo ra bằng cách dùng một trình soạn thảo văn bản hay trình xử lý từ, tùy chọn.

11.1. Tệp nguyên bản

Khi nguyên bản được gọi, **MATLAB** đơn giản thực hiện các lệnh trong tệp, thay cho việc đợi nhập từ bàn phím. Các lệnh trong tệp nguyên bản thực hiện toàn cục trên dữ liệu trong vùng làm việc. Các nguyên bản thường hữu ích cho việc vận hành các phân tích, giải toán, hoặc thực hiện các thiết trí đòi hỏi quá nhiều lệnh mà trở nên cồng kềnh trong chế độ tương tác.

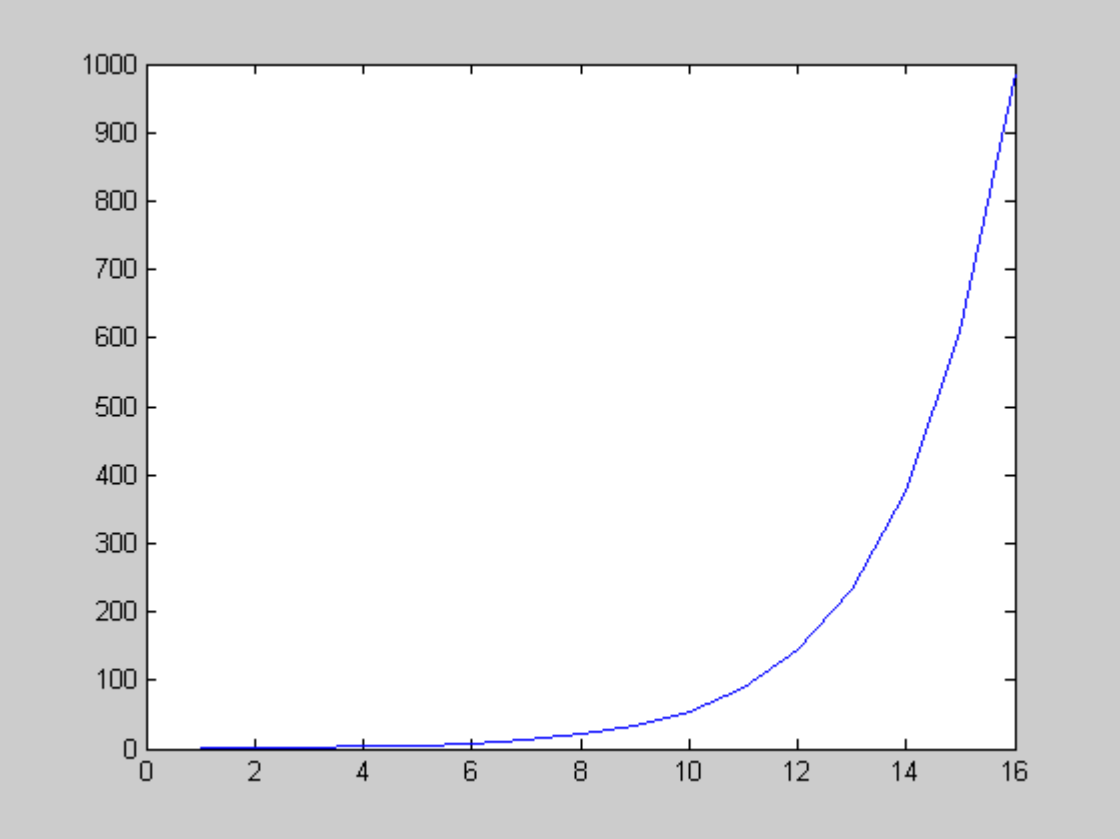
Ví dụ, giả sử các lệnh của **MATLAB**

```
% Một M-file để tính 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)

được chứa trong một tệp tên là **fibno.m**. Vào lệnh **fibno** làm cho **MATLAB** thực hiện các lệnh, tính **16** số **Fibonnaci** đầu tiên và tạo ra hình vẽ như hình 11.1.



Hình 11.1

Sau khi thực hiện tệp xong, các biến **f** và **i** còn lại trong vùng làm việc.

Các chương trình mẫu của **MATLAB** là các ví dụ tốt cho cách sử dụng các M-file để thực hiện nhiều nhiệm vụ phức tạp hơn. Tên nguyên bản **startup.m** được tự động thi hành khi **MATLAB** được gọi. Các hằng vật lý, các thừa số chuyển đổi kỹ thuật, hoặc các thứ khác muốn định nghĩa trước trong vùng làm việc có thể đặt trong các tệp này. Trên hệ thống mạng hoặc nhiều người dùng, thì có một nguyên bản tên **matlab.m** được dành riêng để dùng cho quản lý hệ thống. Nó có thể dùng để cài đặt các định nghĩa và các thông điệp rộng rãi.

11.2. Tệp hàm

Nếu dòng thứ nhất của một M-file chứa từ "**function**", thì tệp là một tệp hàm. Một hàm khác với một nguyên bản là có thể truyền các đối số, và các biến định nghĩa và thực hiện bên trong tệp là cục bộ của hàm và không thao tác toàn cục trong vùng làm việc. Các tệp hàm là hữu ích cho việc mở rộng **MATLAB**, đó là tạo ra các hàm **MATLAB** mới bằng cách dùng chính ngôn ngữ **MATLAB**.

Sau đây là một ví dụ đơn giản. Tệp **mean.m** chứa các lệnh:

```
function  y = mean(x)

% MEAN   Giá trị trung bình. Đối với vectơ , MEAN(x)

% trả về giá trị trung bình. Đối với ma trận, MEAN(x) là một
%vectơ dòng chứa các giá trị trung bình của mỗi cột.
    [m,n] = size(x);

    if m== 1

        m = n;        % xử lý vectơ dòng.

    end

    y = sum(x)/m;
```

Tệp này định nghĩa một hàm mới tên là **mean**. Hàm mới **mean** được dùng như mọi hàm **MATLAB** khác. Ví dụ, nếu **Z** là một vectơ gồm các số từ **1** đến **99**,

```
Z = 1:99;
```

giá trị trung bình tìm thấy bằng cách đánh vào

```
mean(Z)
```

kết quả là **ans =**

```
50
```

Hãy xét vài chi tiết của **mean.m**:

- Dòng thứ nhất khai báo tên hàm, các đối số nhập, và các đối số xuất. Không có dòng này thì tệp sẽ là tệp nguyên bản thay vì tệp hàm.
- Dấu **%** biểu hiện phần còn lại của dòng là lời chú thích và được bỏ qua.

- Vài dòng đầu cung cấp tư liệu M-file và được hiển thị nếu đánh vào

  help mean.

- Các biến **m, n**, và **y** là cục bộ của **mean** và sẽ không còn trong vùng làm việc khi **mean** thực hiện xong. (Hoặc nếu trước đó đã có thì không bị thay đổi.)

- Không cần phải đặt các số nguyên từ **1** đến **99** vào biến **x**. Thực ra, dùng **mean** với biến tên là **Z**. Vectơ **Z** chứa các số nguyên từ **1** đến **99** được truyền hoặc sao chép vào **mean** ở đây nó trở thành một biến cục bộ tên là **x**.

Một phiên bản có một ít phức tạp hơn của **mean** gọi là **stat** tính độ lệch chuẩn:

```
function  [mean, stdev] = stat(x)

[m,n] = size(x);

if m==1

   m = n;      % xử lý vectơ dòng

end

mean = sum(x)/m;

stdev = sqrt(sum(x.^2)/ m - mean.^);
```

stat minh họa cho khả năng trả về nhiều đối số xuất.

Một hàm tính hạng ma trận dùng nhiều đối số nhập:

```
function  r = rank(y,tol)

% hạng của một ma trận

s = svd(x);

if (nargin == 1)

    tol = max(size(x)) * s(1) * eps;

end

r = sum(s>tol);
```

Ví dụ này minh họa cách dùng biến thường xuyên **nargin** để tìm số đối số nhập. Biến **nargout**, mặc dù không được dùng ở đây nhưng chứa số đối số xuất.

Vài gợi ý trợ giúp :

Khi một tệp M-hàm được gọi lần đầu thì được biên dịch và đưa vào bộ nhớ. Sau đó có thể sử dụng cho các lần gọi sau mà không biên dịch lại. Nó còn trong bộ nhớ trừ khi không đủ bộ nhớ, trong trường hợp này có thể bị xóa tự động.

Lệnh **what** trình bày danh sách thư mục các tệp M-file có thể sử dụng trong thư mục hiện hành, lệnh **type** liệt các tệp M-file, và **!** dùng để gọi trình soạn thảo, cho phép tạo ra hoặc sửa đổi tệp M-file.

Nói chung, nếu nhập tên nào đó cho **MATLAB**, ví dụ đánh vào **whoopie**, thì **MATLAB** thông dịch qua các bước sau:

[1] Tìm xem **whoopie** có phải là một biến.

[2] Kiểm tra **whoopie** có phải là hàm cài sẵn.

[3] Tìm trong thư mục hiện hành có không một tệp có tên **whopie.m**.

[4] Tìm trong các thư mục chỉ định bởi biến môi trường

MATLABPATH có không một tệp có tên **whoopie.m**. (Xem phần giới thiệu cách cài đặt để học cách đặt biến môi trường **MATLABPATH**)

Do đó đầu tiên **MATLAB** thử dùng **whoopie** như một biến, nếu có, trước khi dùng **whoopie** như một hàm.

11.3. Các lệnh Echo, input, pause, keyboard

Thông thường, khi thực hiện M-file, các lệnh trong tệp không được hiển thị trên màn hình. Lệnh **echo** làm cho tệp M-file được thấy khi thực hiện, điều này hữu ích cho việc gỡ rối hoặc làm mẫu. Xem phần tham khảo để biết thêm chi tiết.

Hàm **input** nhận dữ liệu nhập từ người dùng. Lệnh

n = input('Có bao nhiêu quả táo')

cho người dùng câu văn bản nhắc, đợi người dùng nhập số hoặc biểu thức từ bàn phím. Một cách dùng **input** là xây dựng M-file điều khiển **menu**. Công cụ **demo** là một ví dụ cho trường hợp này.

Tương tự **input**, nhưng mạnh hơn, là hàm **keyboard**. Hàm này gọi bàn phím như một nguyên bản. Đặt trong các tệp M-file, thì đặc tính này giúp ích cho việc gỡ rối, hoặc cho việc thay đổi các biến trong thời gian thi hành.

Lệnh **pause** tạo ra thủ tục dừng và chờ người dùng ấn phím bất kỳ trước khi tiếp tục. Lệnh **pause(*n*)** tạm dừng ***n*** giây trước khi tiếp tục.

Cũng có thể định nghĩa các biến toàn cục, mặc dù chúng tôi không khuyên như thế. Xem phần tham khảo nếu có ý muốn.

11.4. Xâu chữ và macro xâu chữ

Các xâu chữ văn bản được nhập vào **MATLAB** trong cặp nháy đơn. Ví dụ,

```
s = 'Hello'
```

kết quả là

```
s =

    Hello
```

Xâu chữ được lưu trong một vectơ, mỗi phần tử một ký tự. Trong trường hợp này, lệnh

```
size(s)

ans =

     1     5
```

biểu hiện rằng **s** có **5** phần tử. Các ký tự được lưu giá trị ASCII của chúng và hàm **abs** trình bày giá trị này,

```
abs(s)

ans =

    72   101   108   108   111
```

Hàm **setstr** đặt các vectơ để hiển thị như văn bản thay vì trình bày các giá trị ASCII. Cũng hữu ích là lệnh **disp** đơn giản hiển thị văn bản có trong biến, và các hàm **isstr** và **strcmp** dò tìm và so sánh các xâu chữ tương ứng.

Các biến văn bản có thể nối lại thành xâu chữ lớn bằng cách dùng cặp ngoặc vuông:

```
s = [s, 'World']

s =

   Hello World
```

Các giá trị số được chuyển sang các xâu chữ bằng các hàm **sprintf, num2str**, và **int2str**. Các giá trị số sau khi chuyển sang xâu chữ thường được nối vào xâu chữ lớn để đặt tiêu đề cho hình vẽ có giá trị số:

```
f = 70; c =(f-32)/1.8;

title(['Nhiệt độ trong phòng là ',num2str(c),' độ C'])
```

eval là hàm làm việc với các biến xâu chữ để cài đặt một công cụ **macro** văn bản khá mạnh mẽ. **eval(t)** làm cho văn bản chứa trong **t** được ước lượng. Nếu **STRING** là văn bản nguồn cho nhiều biểu thức hoặc câu lệnh của **MATLAB** thì

```
t ='STRING';
```

mã hóa văn bản trong **t**. Đánh vào **t** in văn bản và **eval(t)** làm cho văn bản được thông dịch, hoặc là một lệnh hoặc là một nhân tử trong biểu thức. Ví dụ

```
t = '1/(i+j-1)';

for  i = 1:n

   for  j = 1:n

        a(i,j) = eval(t);

   end

end
```

phát sinh ma trân **Hilbert** cấp **n**. Một ví dụ khác trình bày văn bản đánh chỉ số,

```
S = ['x = 3                '

     'y = 4                '

     'z = sqrt(x*x+y*y) '];

for k = 1:3

   eval(S(k,:));
```

```
end
```

Các xâu chữ tạo thành các dòng của ma trận **A** cần phải có cùng độ dài. Sau đây là ví dụ cuối cùng trình bày cách **eval** có thể dùng lệnh **load** để nạp **10** tệp dữ liệu được đánh số liên tục:

```
fname = 'mydata';

for  i = 1:10

     eval(['load ',fname,int2str(i)]

end
```

Công cụ macro văn bản được ứng dụng hữu hiệu trong việc truyền tên hàm cho các tệp M-hàm. Để lấy ví dụ, xem tệp **funm.m** trong **MATLAB TOOLBOX**.

11.5. Chương trình bên ngoài

Có thể, và thương hữu ích, để tạo ra các chương trình độc lập bên ngoài riêng của mình hoạt động như các hàm **MATLAB** mới. Điều này có thể thực hiện bằng cách viết các tệp M-file để

[1] Lưu các biến trên đĩa,

[2] Chạy các chương trình bên ngoài (đọc các tệp dữ liệu, xử lý chúng, và ghi kết quả trở lại đĩa), và

[3] Nạp các tệp đã xử lý ngược về vùng làm việc.

Ví dụ, sau đây là một M-hàm giả định để tìm lời giải phương trình **Garfield** dùng chương trình **GAREQN** bên ngoài

```
function  y =  garfield(a,b,q,r)

save  gardata a b q r

!gareqn

load gardata
```

Nó yêu cầu đã viết một chương trình tên là **GAREQN** (bằng **Fortran** hoặc ngôn ngữ nào đó) để đọc tệp tên là **gardata.mat**, xử lý nó, và đặt kết quả trở ra tệp đó. Các chương trình con tiện ích mô tả trong phần sau có thể dùng để đọc và ghi các tệp **MAT**.

Công cụ này là một trong các lựa chọn để "liên kết chương trình riêng" vào **MATLAB**. Một lựa chọn khác là dùng công cụ tệp **MEX** - một kỹ thuật nhờ đó có thể liên kết vật lý chương trình có đối tượng mới vào **MATLAB**. Xem phần đặc tả máy để thấy công cụ này có thể dùng cho máy mình không.

11.6. Vấn đề về tốc độ và bộ nhớ

Các thao tác về vectơ và ma trận gắn liền của **MATLAB** thực hiện nhanh hơn các thao tác được dịch của nó. Điều này có nghĩa là để nhận tốc độ nhanh nhất ngoài **MATLAB** phải cố gắng vectơ hóa thuật toán trong tệp M-file. Bất kỳ đâu có thể được, các vòng lặp **for** và **while** nên chuyển sang các phép toán về vectơ hoặc ma trận. Ví dụ, một cách lấy **sin** của **1000** số từ **1** đến **10**:

```
i = 0;
for  t = 0:.01:10
    i = i+1;
    y(i)  = sin(t);
end
```

Một phiên bản vectơ hóa của cùng chương trình này là:

```
t = 0:.01:10;
y = sin(t);
```

Trên một máy, ví dụ thứ nhất chạy hết **15** giây, trong khi ví dụ thứ hai chỉ tốn **0.6** giây, nhanh gấp **25** lần. Không phải luôn luôn tối ưu được các chương trình phức tạp, nhưng khi tốc độ là quan trọng thì nên tìm cách vectơ hóa thuật toán.

Nếu không thể vectơ hóa mảnh chương trình, thì đây là một cách để làm cho vòng lặp **for** chạy nhanh hơn: *định vị trước mọi vectơ có kết quả xuất được lưu*. Ví dụ, việc đưa vào câu lệnh thứ nhất ở đây , dùng hàm **zeros**, làm cho vòng lặp **for** thực hiện nhanh đáng kể:

```
y = zeros(1,100);
for  i = 1:100
    y(i) = det(x^i);
end
```

Lý do là, nếu không định vị trước, thì **MATLAB** phải tăng kích thước của vectơ y lên **1** qua mỗi lần lặp. Nếu vectơ được định vị trước thì bước này được khử đi và việc thực hiện được nhanh hơn.

Đối với công việc thực hiện với các ma trận lớn trên các máy có bộ nhớ hạn chế, thì ý đồ định vị trước có một tiện lợi thứ hai: là sử dụng bộ nhớ hiệu lực hơn và giúp kiểm tra có chạy tràn bộ nhớ không. Nó trợ giúp vì bộ nhớ có khuynh hướng bị phân mảnh, bởi vậy có thể có nhiều vùng nhớ tự do, nhưng không đủ không gian liên tục để giữ một biến lớn. Việc định vị trước giúp thu gọn sự phân mảnh.

Trong vấn đề về bộ nhớ, nếu lệnh **who** hiển thị tổng số bộ nhớ tự do còn lại, thì có vài điều về số này có lẽ nên cẩn thận. Nếu xóa một biến trong vùng làm việc, thì con số hiển thị bởi lệnh **who** thường không tăng lên, *trừ khi nó là biến "cao nhất" trong vùng làm việc.*

Con số này biểu hiện thực sự tổng bộ nhớ tự do liên tục và chưa dùng. Việc xóa biến cao nhất làm cho bộ nhớ lớn hơn, nhưng xóa biến dưới biến cao nhất không có hiệu lực. Về mặt ứng dụng, toàn bộ ý nghĩa này là có thể có nhiều vùng nhớ tự do hơn lệnh **who** biểu hiện.

Các máy tính với bộ nhớ ảo không hiển thị tổng số vùng nhớ tự do còn lại vì không có các giới hạn phải chấp nhận của **MATLAB** hay của phần cứng.

Có một cách tối ưu mà **MATLAB** thực hiện giúp để biết khi viết M-file. Các đối số gọi hàm M-file không sao chép sang vùng làm việc cục bộ của hàm trừ khi xen vào nội dung của đối số vào bên trong M-hàm. Điều này có nghĩa là không có không tốn bộ nhớ cho việc truyền các biến lớn vào các hàm M-file.

****

Chương 12. VỀ TỆP TRÊN ĐĨA

load và **save** là các lệnh của **MATLAB** để lưu vào hoặc lấy ra từ đĩa nội dung của vùng làm việc. Các lệnh khác quan hệ đến tệp giúp cho việc quản lý đĩa, cho phép các chương trình bên ngoài chạy, và cung cấp khả năng nhập/xuất dữ liệu.

12.1. Thao tác về tệp

Các lệnh **dir, type, delete**, và **chdir** cài đặt tập hợp các lệnh về hệ điều hành chung để thao tác về tệp. Sau đây là bảng biểu hiện các lệnh này sắp xếp với các hệ điều hành khác, có lẽ người dùng làm quen với một trong chúng:

MATLAB	MS-DOS	UNIX	VAX/VMS
dir	dir	is	dir
type	type	cat	type
delete	del	rm	delete
chdir	chdir	cd	set default

Với hầu hết các lệnh này, đường dẫn, ký tự đại diện, và tên ổ đĩa dùng theo cách thông thường.

Lệnh **type** khác với lệnh **type** thông thường ở một điểm đặc biệt; nếu không cho kiểu tệp thì ngầm định là **.m**. Điều này thuận tiện cho việc hay dùng nhất của lệnh **type** là để liệt kê các tệp M-file trên màn hình.

Lệnh **diary** tạo ra nhật ký cho công việc của **MATLAB** trên đĩa (tuy nhiên không lưu các hình ảnh). Kết quả là tệp văn bản ASCII phù hợp với việc đưa vào các bản báo cáo và các tài liệu khác dùng trình xử lý từ bất kỳ.

Để biết thêm chi tiết về các lệnh này, xem phần tham khảo hoặc dùng công cụ trợ giúp nóng **help**.

12.2. Chạy chương trình bên ngoài

Ký tự chấm than, ! , là ký tự thoát và biểu hiện phần còn lại của dòng nhập là lệnh của hệ điều hành. Điều này hoàn toàn có ích cho việc gọi các trình tiện ích hoặc chạy các chương trình khác mà không ra khỏi MATLAB. Ví dụ

!f77 simpleprog

gọi trình biên dịch Fortran và

!edt darwin.m

gọi trình soạn thảo edt cho một tệp có tên là darwin.m. Sau khi chương trình này chạy xong, quyền điều khiển trả về cho MATLAB.

Cách xử lý đúng đắn về ! tùy thuộc vào từng loại máy cụ thể. Xem phần đặc tả máy để biết thêm thông tin.

12.3. Nhập và xuất dữ liệu

Dữ liệu từ các chương trình khác và bên ngoài có thể đưa vào **MATLAB** bằng nhiều cách. Tương tự, dữ liệu **MATLAB** có thể xuất ra bên ngoài. Cũng có thể có các chương trình thao tác dữ liệu trực tiếp trong các tệp **MAT**, dạng tệp **MATLAB** sử dụng.

Cách tốt nhất phụ thuộc vào số lượng dữ liệu đang có, dữ liệu có ở dạng máy có thể đọc được không, dạng gì, v.v... Sau đây là một số lựa chọn; hãy chọn một để tương thích.

[1] Nhập như một danh sách rõ ràng các phần tử. Nếu ít dữ liệu, nhỏ hơn **10x15** phần tử, thì dễ dàng nhập trực tiếp dữ liệu vào bằng cặp ngoặc vuông, **[** và **]**. Phương pháp này bất tiện đối với dữ liệu lớn vì không thể sửa dữ liệu nhập nếu bị lỗi.

[2] Tạo trong một M-file. Dùng trình soạn thảo văn bản để tạo ra một nguyên bản M-file chứa danh sách rõ ràng các phần tử. Cách này là tốt khi dữ liệu chưa ở dạng máy có thể đọc được và phải đánh chúng vào. Cơ bản là giống cách **1**, có thuận tiện là cho phép dùng trình soạn thảo để thay đổi dữ liệu hoặc sửa lỗi. Sau đó có thể chạy lại M-file để nhập lại dữ liệu.

[3] Nạp từ một tệp ASCII phẳng. Nếu dữ liệu được lưu ở dạng ASCII, với các dòng có độ dài cố định kết thúc bằng ký tự sang dòng,và các khoảng trống ngăn cách các số, thì tệp như thế được gọi là tệp phẳng.(Tệp ASCII phẳng có thể được soạn bằng trình soạn thảo văn bản thông thường.) Các tệp phẳng có thể được đọc trực tiếp vào **MATLAB** bằng lệnh **load**. Kết quả được đặt vào một biến có tên là tên tệp.

[4] Viết một chương trình bằng **Fortran** hoặc **C** để dịch dữ liệu sang dạng **MAT**-file.

Vài cách đưa dữ liệu ra bên ngoài là:

[1] Với các ma trận nhỏ, dùng lệnh **diary** để tạo ra một tệp nhật ký, rồi sau đó liệt kê các biến trong tệp này. Có thể dùng soạn thảo văn bản để thao tác trên tệp nhật ký sau này. Việc xuất dữ liệu của lệnh **diary** dùng trong suốt thời

gian làm việc của **MATLAB**, nó có ích cho việc đưa dữ liệu vào tài liệu và các bản báo cáo.

[2] Lưu vào một biến bằng lệnh **save**, với lựa chọn **/ascii**. Ví dụ,

```
A rand(4,3);
save temp.dat  A/ascii
```

tạo ra một tệp ASCII tên là **temp.dat** chứa nội dung:

```
0.2113    0.8096    0.4832
0.0824    0.8474    0.6135
0.7599    0.4524    0.2749
0.0087    0.8075    0.8807
```

[3] Viết một chương trình bằng **Fortran** hoặc **C** để dịch tệp **MAT** sang dạng đặc biệt riêng của mình.

Có thể muốn có các chương trình bên ngoài đọc hoặc ghi dữ liệu trực tiếp vào tệp **MAT** đã dùng các lệnh load và save. Dạng thức của tệp **MAT** được trình bày dưới lệnh **load** trong phần tham khảo.

Nếu chương trình viết bằng **Fortran** hoặc **C**, thì có vài phục vụ cung cấp trong **MATLAB TOOLBOX** giúp giao diện chương trình với các tệp **MAT**:

savemat.for	Ghi tệp **MAT**.
loadmat.for	Đọc tệp **MAT**.
testls1.for	Ví dụ sử dụng **savemat** và **loadmat**.
testls2.for	Ví dụ khác sử dụng **savemat** và **loadmat**.
loadmat.c	Nạp ma trận từ tệp **MAT**.
savemat.c	Lưu ma trận vào tệp **MAT**.
testls.c	Ví dụ dùng **loadmat.c** và **savemat.c**.

Việc cài đặt các phiên bản **Fortran** của các phục vụ này có thể khác nhau theo từng loại máy.

****

THÔNG TIN VỀ TÁC GIẢ
GIÁO TRÌNH "MATLAB"

1 Thông tin về tác giả :

+ Họ và tên **: PHAN THANH TAO**
+ Quê quán :
+ Cơ quan công tác :
  KHOA CÔNG NGHỆ THÔNG TIN
  Trường Đại học Bách khoa, Đại học Đà Nẵng
+ Email:

2 Phạm vi và đối tượng sử dụng :

+ Giáo trình dùng tham khảo cho các ngành
+ Có thể dùng ở các trường có đào tạo các chuyên ngành
+ Từ khóa :**.**
+ Yêu cầu kiến thức trước khi học môn này :