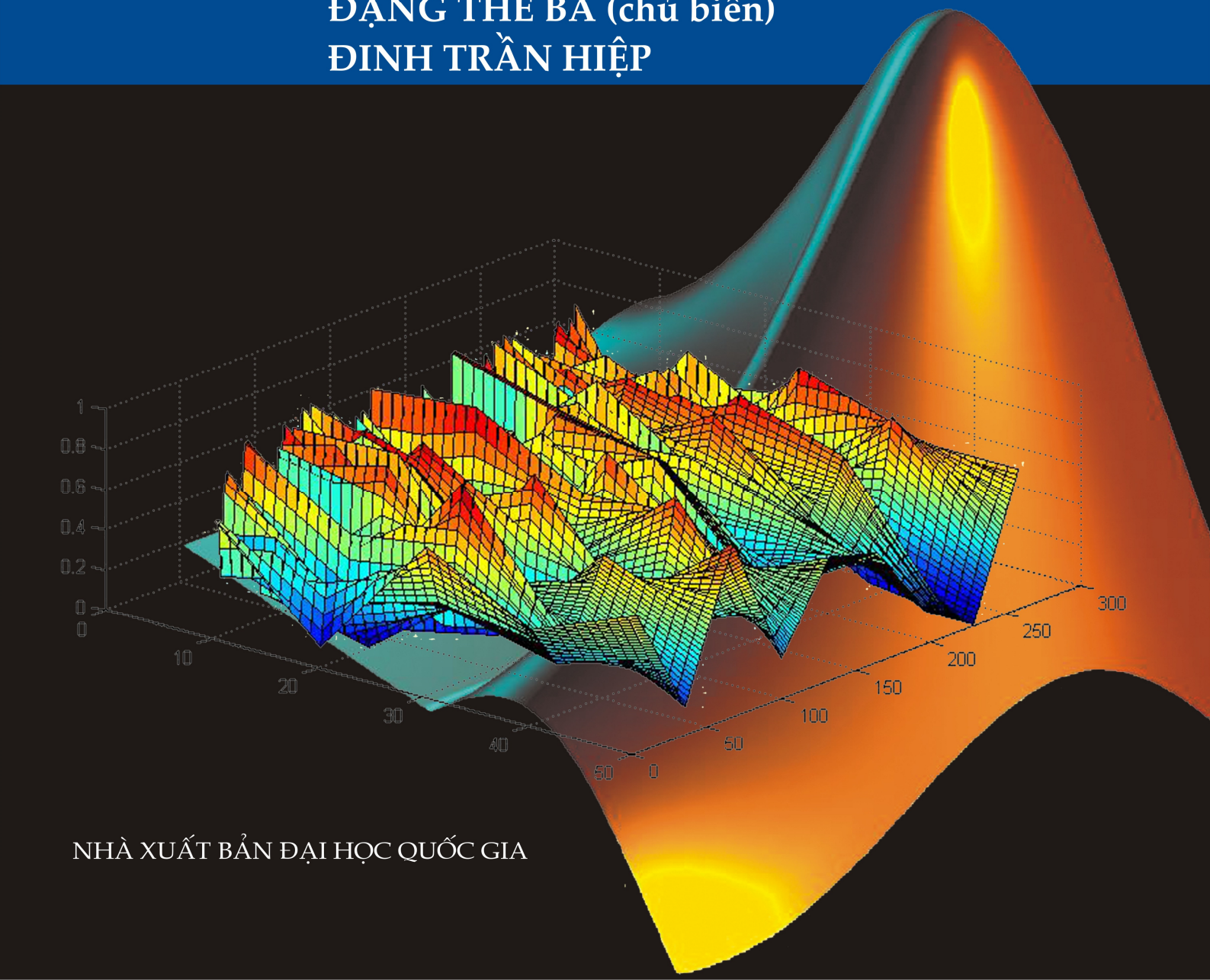
ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

MATLAB
VÀ ỨNG DỤNG TRONG CƠ KỸ THUẬT

ĐẶNG THẾ BA (chủ biên)
ĐINH TRẦN HIỆP



NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA

MỤC LỤC

Lời nói đầu

MATLAB là một gói phần mềm mạnh đang được sử dụng rộng rãi cho nhiều mục đích của nhiều đối tượng khác nhau, đặc biệt là trong khoa học và kỹ thuật. MATLAB có thể sử dụng theo hai cách khác nhau: Thứ nhất là sử dụng các lệnh lần lượt đưa vào dấu nhắc và thu dược kết quả (thông dịch); thứ hai là viết các lệnh thành một file (có thể là file lệnh - scrip file hoặc dạng hàm – function file) và các lệnh được thực hiện tự động ngay sau khi gọi thực hiện thông qua gọi tên file.

MATLAB được trang bị sẵn nhiều hàm để thực hiện các nhiệm vụ khác nhau thuộc nhiều lĩnh vực, từ đơn giản thực hiện các phép tính toán học đến các nhiệm vụ phức tạp như tìm nghiệm của phương trình vi phân, tối ưu hóa, xử lý ảnh…Thêm nữa, MATLAB còn được trang bị các câu lệnh cấu trúc dùng cho lập trình cho phép người dùng tự phát triển các chương trình theo mục đích riêng cho những bài toán cụ thể gặp phải trong thực tế.

Cuốn giáo trình này được biên soạn cho sinh viên khoa Cơ học Kỹ thuật và Tự động hóa, trường Đại học Công nghệ, Đại học Quốc gia Hà Nội nhằm giúp sinh viên làm quen với MATLAB. sử dụng MATLAB để giải các bài toán đơn giản và tiến tới lập trình tính toán mô phỏng cho các bài toán cơ bản phức tạp hơn. Khi đã nắm được những kiến thức cơ bản về khả năng, phương thức làm việc của MATLAB và kỹ năng lập trình, sinh viên có thể xây dựng các chương trình phức tạp hơn trong MATLAB để giải quyết hiệu quả các vấn đề thực tế liên quan đến ngành, chuyên ngành đã học. Cuốn sách được dùng kết hợp với các bài giảng và các buổi thực hành để trang bị những kiến thức cần thiết cho các nhà khoa học và kỹ sư tương lai để có thể phát triển và nâng cao khả năng tính toán, xử lý và phân tích những vấn đề chuyên môn đa dạng với sự trợ giúp của máy tính.

Nội dung giáo trình gồm các chương sau:

- Chương 1: Mở đầu,
- Chương 2: Vector và ma trận
- Chương 3: Tính toán số với MATLAB

- Chương 4: Bộ công cụ ký hiệu
- Chương 5: Mảng hỗn hợp và dữ liệu có cấu trúc
- Chương 6: Đồ thị trong MATLAB
- Chương 7: Lập trình trong MATLAB
- Chương 8: Phân tích động học cơ cấu
- Chương 9: Lý thuyết điều khiển tự động

Kết thúc mỗi chương, sinh viên có thể tự ôn tập và củng cố kiến thức đã học qua mục tổng hợp các lệnh trong chương và bài tập có lời giải tương ứng. Đối với những bài tập không có lời giải chi tiết, sinh viên có thể trao đổi với giảng viên trong các buổi thực hành.

Ngoài ra để cung cấp thông tin định hướng cho sinh viên muốn tìm hiểu thêm những công cụ dùng cho những ứng dụng chuyên sâu (Toolbox), một số bộ công cụ phổ biến sẽ được giới thiệu tóm tắt trong phần phụ lục.

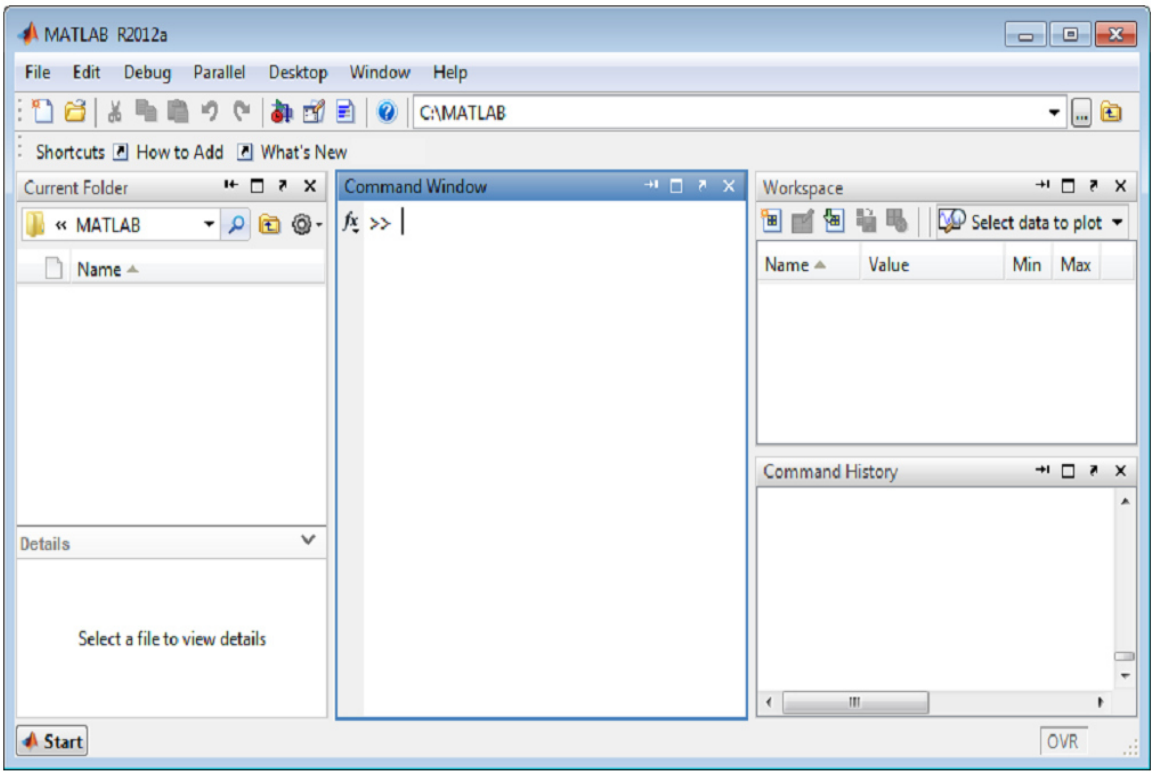
PHẦN I. MATLAB CƠ BẢN

Chương 1. MỞ ĐẦU

1.1 Giới thiệu MATLAB

MATLAB (**MAT**rix **LAB**oratory) là phần mềm của công ty MathWorks có trụ sở chính ở Natick, Massachusetts, Mỹ (www.mathworks.com), được ứng dụng rộng rãi trong các lĩnh vực khoa học, kỹ thuật, đặc biệt là điều khiển tự động và xử lý tín hiệu số. MATLAB được phát triển bởi Cleve Moler vào cuối những năm 70 của thế kỷ trước trên cơ sở thư viện nguồn của LINPACK và EISPACK FORTRAN. Những phiên bản đầu tiên của MATLAB chủ yếu phục vụ cho tính toán xử lý ma trận và giải các phương trình đại số tuyến tính. Ngày nay, MATLAB còn được biết đến như là một công cụ có khả năng xử lý đồ họa mạnh mẽ (Theo *PC Magazine Encyclopedia*).

Giao diện MATLAB



Hình 1. 1. Giao diện MATLAB

Khi khởi động, MATLAB sẽ hiện ra giao diện mặc định như trên, trong đó bao gồm các cửa số:

- Current Folder: Thư mục hiện hành, cho phép truy xuất tới các file chứa trong nó.
- Command Window: Cửa sổ lệnh, người dùng có thể nhập lệnh trên cửa sổ này từ sau dấu nhắc (>>).
- Workspace: Không gian bộ nhớ của MATLAB, dùng để lưu trữ dữ liệu của các biến trong một phiên làm việc của MATLAB.
- Command History: Lịch sử lệnh đã thực hiện, cho phép xem hoặc thực hiện lại những lệnh đã nhập vào Command Window trước đó.

Sau đây là một số ví dụ đơn giản thực hiện trong Command Window: tại dấu nhắc >> nhập câu lệnh sau:

```
>> a = 1
```

MATLAB sẽ lưu biến a có giá trị 1 vào Workspace và hiển thị kết quả trên màn hình:

```
a =
     1
```

Tương tự tạo các biến b, c có giá trị lần lượt là:

```
>> b = 2
b =
     2
>> c = a + b
c =
     3
```

Khi chúng ta không chỉ rõ giá trị trả về của phép tính, MATLAB sẽ sử dụng biến *ans* (viết tắt của *answer*) để lưu kết quả này.

```
>> b - a
ans = 1
```

Khi câu lệnh kết thúc bởi dấu chấm phẩy, MATLAB sẽ thực hiện phép tính nhưng không hiển thị kết quả lên màn hình.

```
>> d = a*b;
```

Hai phím mũi tên lên xuống ↑ ↓ có thể được sử dụng để gọi lại những câu lệnh đã sử dụng trước đó. Có thể sử dụng hai phím này tại dấu nhắc >> hoặc sau khi nhập một số ký tự đầu tiên của các ký tự đã sử dụng. Ví dụ: để gọi lại lệnh b = 2 chỉ cần nhập b và sau đó nhấn phím mũi tên.

Dấu % được sử dụng để viết chú thích cho câu lệnh. Mọi ký tự phía sau dấu % sẽ không được MATLAB xử lý. Ví dụ lấy a nhân với b và gán giá trị cho d:

```
>> d = a*b
```

1.2 Một số kiến thức cơ bản

1.2.1 Các phép toán thông dụng

Trong MATLAB các phép toán cộng, trừ, nhân, chia… được biểu diễn thông qua các ký tự đã quy định trước. Bảng 1.1 thể hiện các phép toán thông dụng và cách nhập các phép toán này trong MATLAB.

MATLAB	Ý nghĩa
`a+b`	$a + b$
`a-b`	$a - b$
`a*b`	Ab
`a/b`	$\frac{a}{b}$
`a^b`	a^b
`sin(x)`	$sin(x)$
`sqrt(x)`	$\sqrt{x}$
`cos(x)`	$cos(x)$
`exp(x)`	e^x
`tan(x)`	$tan(x)$

MATLAB	Ý nghĩa
`log(x)`	$ln(x)$
`asin(x)`	$sin^{-1}(x)$
`log10(x)`	$log_{10}(x)$
`acos(x)`	$cos^{-1}(x)$
`abs(x)`	$\|x\|$
`atan(x)`	$tan^{-1}(x)$
`rand`	*tạo số thực ngẫu nhiên trong khoảng (0, 1)*
`round(x)`	*làm tròn tới số nguyên gần nhất*
`floor(x)`	*làm tròn tới số nguyên bé hơn*
`ceil(x)`	*làm tròn tới số nguyên lớn hơn*

Bảng 1. 1. Các phép toán thông dụng trong MATLAB

1.2.2 Các lệnh quản lý

Trong MATLAB khi gặp trường hợp thời gian thực hiện lệnh quá lâu, có thể sử dụng tổ hợp phím "**ctrl + c**" để ngừng các tính toán đang thực thi. Câu lệnh mới có thể được nhập vào tại dấu nhắc xuất hiện sau khi thực hiện tổ hợp phím vừa nêu, lệnh ngắt này đặc biệt hiệu quả khi lập trình với vòng lặp.

Khi muốn xóa một hay nhiều biến, ta có thể sử dụng lệnh ***clear*** với các cú pháp thông dụng như sau

```
>> clear  bien
>> clear all
```

Trong đó

- *bien* và *all* lần lượt là tên các biến cụ thể hoặc tất cả các biến có trong Workspace
- Cú pháp thứ nhất cho phép xóa biến cụ thể khỏi Workspace còn cú pháp thứ hai cho phép người sử dụng xóa toàn bộ tất cả các biến, hàm hiện hành

Khi muốn hiện danh sách các biến có trong Workspace ta có thể dùng ***who*** hoặc ***whos*** theo cú pháp sau:

```
>>who
```

Hoặc

```
>>whos
```

Trong đó lệnh ***who*** chỉ liệt kê tên các biến, còn lệnh ***whos*** liệt kê cả tên và các thông tin cụ thể về biến.

Ví dụ: Nhập dòng lệnh >> a = 1; b = 2; c = 3;

Trong Workspace xuất hiện lần lượt các biến a, b, c với giá trị lần lượt là 1, 2, 3.

Quan sát sự khác biệt khi sử dụng "who" và "whos":

```
>>who
Your variables are:
a  b  c
>>whos
Name      Size      Bytes     Class     Attributes
  a       1x1       8         double
  b       1x1       8         double
  c       1x1       8         double
```

Xóa các biến trong Workspace bằng lệnh clear:

```
>>clear a
>>a
??? Undefined function or variable 'a'
>>clear all    % xóa nốt các biến b và c
```

Trong đó biến *a* được xóa khỏi Workspace trước tiên, tiếp theo là các biến *b* và *c*. Để tiện quan sát ta có thể "làm sạch" Command Window bằng lệnh "**clc**".

1.2.3 Các lệnh nhập xuất dữ liệu

Nhập dữ liệu từ bàn phím: ***input***

Cú pháp:

```
>>bien = input('text')
>>bien = input('text' 's')
```

Trong đó: *bien* là tên biến bất kỳ do người sử dụng đặt, *text* là đoạn văn bản sẽ hiển thị thành câu nhắc trên màn hình, *s* là tham số để MATLAB hiểu và xử lý dữ liệu nhập vào ở dạng xâu ký tự.

Ở cả hai cú pháp trên, đoạn văn bản sẽ được hiển thị trên màn hình, đồng thời MATLAB sẽ chờ người sử dụng nhập ký tự từ bàn phím tuy nhiên khi sử dụng cú pháp thứ nhất, người dùng chỉ có thể nhập vào một số, còn ở cú pháp thứ hai, người dùng có thể nhập vào một xâu ký tự sau khi đã thông báo cho MATLAB bằng tham số *s*. Trường hợp thực hiện lệnh input khi người dùng không nhập giá trị mà bấm trực tiếp phím Enter thì giá trị trả về sẽ là một mảng rỗng.

Ví dụ:

```
>> Lop = input('Ban hoc lop nao: ', 's')
Ban hoc lop nao: Co Dien Tu
Lop = Co Dien Tu
>>x=10;
>>SoSV = input('Lop ban co bao nhieu sinh vien: ')
Lop ban co bao nhieu sinh vien: 80
SoSV = 80
>>SVG = input('Bao nhieu sinh vien duoc A+ mon MATLAB:
')
Bao nhieu sinh vien duoc A+ mon MATLAB: x
SVG = 10
```

Nhập dữ liệu từ file: ***load***

Trong MATLAB có rất nhiều cách để tải dữ liệu từ file vào Workspace, trong đó đơn giản nhất là lệnh ***load*** cho phép tải toàn bộ nội dung file vào Workspace. Yêu cầu khi sử dụng lệnh ***load*** là nội dung file được tải phải ở dạng ma trận chứa các số, mỗi số ở các dòng phân cách nhau bằng ký tự trống, dấu phẩy hoặc tab. File có thể chứa các dòng chú thích trong MATLAB (các dòng bắt đầu bằng ký tự %).

Cú pháp:

```
>>load name.ext
```

Trong đó *name.ext* là tên file chứa dữ liệu cần tải. Nếu trong phần tên file không có phần mở rộng *ext*, MATLAB sẽ tìm trong thư mục hiện hành file có tên *name.mat*, nếu không tìm thấy *name.mat* hoặc tên file không có phần mở rộng không phải là *mat* (ví dụ: *txt*, *dat*) thì MATLAB sẽ coi file đó là file chứa dữ liệu ASCII.

Lệnh ***load*** như đã nói ở trên sẽ cho phép tải toàn bộ nội dung trong file *name.ext* vào trong Workspace.

Ví dụ: Tạo trong thư mục hiện hành file *diem.txt* là một ma trận 5 hàng 3 cột có nội dung như sau:

```
10      8       8.8
7       9       8.2
5       8       6.8
9       6       7.2
9       10      9.6
```

Sau đó tải dữ liệu từ file vào Workspace rồi copy vào các vector x, y, z.

Các câu lệnh cần thực hiện có thể như sau:

```
>>load diem.txt
```

tải dữ liệu từ file vào ma trận

```
>>x = diem(:,1);
```

copy cột 1 của diem vao vector x

```
>>y = diem(:,2);
```

copy cột 2 của diem vào vector y

```
>>z = diem(:,3);
```

copy cột 3 của diem vào vector z

Nhập dữ liệu từ file: ***fscanf***

Cho phép tải dữ liệu từ file ma trận mà các kiểu dữ liệu của các phần tử là không giống nhau.

Cú pháp:

```
>>A = fscanf(fid, format)
```

Trong đó file được xác định bởi tham số *fid* (file identifier) là số nguyên xác định file có từ lệnh ***fopen*** (cú pháp: `fid = fopen(filename)`), *format* là định dạng dữ liệu. Toàn bộ dữ liệu có định dạng *format* sẽ được copy từ file xác định bởi *fid* vào trong Workspace và sắp xếp theo dạng cột.

Ví dụ file *BDTH.txt* có nội dung như sau:

```
10      8      8.8     A
7       9      7.6     B
5       7      6.2     C
4       5      4.6     D
3       3      3       F
```

```
>>fid = fopen('BDTH.txt');
```

```
>>B=fscanf(fid,'%d %d %f %c',[4 inf])
```

lưu ma trận trong file "BDTH.txt" vào ma trận B có 4 hàng %(tương ứng với số cột trong ma trận gốc)

```
>>A=B'
```

A là ma trận chuyển vị của B

Thực hiện phép tính và quan sát kết quả thu được

```
A =
     10.0000    8.0000    8.8000   65.0000
     7.0000     8.0000    7.6000   66.0000
     5.0000     7.0000    6.2000   67.0000
     4.0000     5.0000    4.6000   68.0000
     3.0000     3.0000    3.0000   70.0000
```

Lưu ý: Sử dụng ***fscanf*** để đọc file có cả các ký tự và các số như ví dụ trên, ma trận trả về sẽ là ma trận số, trong đó giá trị trả về của các ký tự trong ma trận là giá trị tương ứng trong bảng mã ASCII.

Thử kiểm tra với giá trị cuối cùng của hàng 1 trong ma trận thu được:

```
>>char(A(1,4))
ans =
     A
```

Xuất dữ liệu ra màn hình: ***disp***

Cú pháp:

```
>>disp(X)
```

Trong đó *X* là mảng dữ liệu. Lệnh cho phép xuất ra màn hình nội dung của mảng *X* mà không đưa ra tên của mảng này.

Trong trường hợp *X* là một chuỗi ký tự, disp cũng sẽ xuất ra nội dung chuỗi đó.

Ví dụ:

```
>>disp(A)
```

xuất ra ma trận A thu được từ ví dụ trên

```
>>nhap = 'Kiem tra lenh disp'
>>disp(nhap)
Kiem tra lenh disp
```

Xuất dữ liệu theo định dạng ra file hoặc ra màn hình: ***fprinf***

Cú pháp:

```
>>fprintf(fileID,formatSpec,A1,...,An)
>>fprintf(formatSpec,A1,...,An)
```

Trong đó *fileID* là số nguyên xác định file thu được khi sử dụng lệnh ***fopen***, *formatSpec* là định dạng áp dụng cho các phần tử của mảng *A1, A2, …, An*.

Cú pháp thứ nhất cho phép xuất ra file dữ liệu dưới dạng cột áp dụng định dạng *formatSpec* đối với các phần tử của mảng *A1, A2, …, An*.

Cú pháp thứ hai áp dụng định dạng *formatSpec* cho dữ liệu và xuất ra màn hình.

(Các định dạng cho việc xuất, nhập dữ liệu xem bảng phần 1.4)

Ví dụ sau trình bày cách mở file *vd1.txt* và ghi vào nội dung dưới đây:

Diem cua sinh vien 1 la: 8

Diem cua sinh vien 2 la: 9

Diem cua sinh vien 3 la: 10

```
>> A=[1 2 3;8 9 10];
>> fileID=fopen('vd1.txt','w');
>> formatSpec='Diem cua sinh vien %d la: %d\n';
>> fprintf(fileID,formatSpec,A);
>> fclose(fileID);
```

Nội dung trên cũng có thể xuất trực tiếp ra màn hình:

```
>> fprintf(formatSpec,A);
```

Lệnh ***fprintf*** còn cho phép định dạng sẵn số ký tự dùng để biểu diễn các biến. Ví dụ %5d sẽ sử dụng một khoảng rộng 5 ký tự để biểu diễn một số nguyên, %10s sẽ sử dụng một khoảng rộng 10 ký tự để biểu diễn một xâu. Đối với số thực, số chữ số phần thập phân cũng được xác định, ví dụ %6.2f đồng nghĩa với việc

MATLAB sẽ sử dụng một khoảng rộng 6 ký tự (bao gồm dấu thập phân và phần thập phân) với hai chữ số biểu diễn phần thập phân.

Lưu ý: Nếu phần biểu diễn rộng hơn so với yêu cầu, các ký tự đầu tiên sẽ là các ký tự trống, nếu phần biểu diễn phần thập phân rộng hơn so với yêu cầu, các ký tự cuối cùng sẽ là số 0.

Lưu dữ liệu vào file: ***save***

Cú pháp:

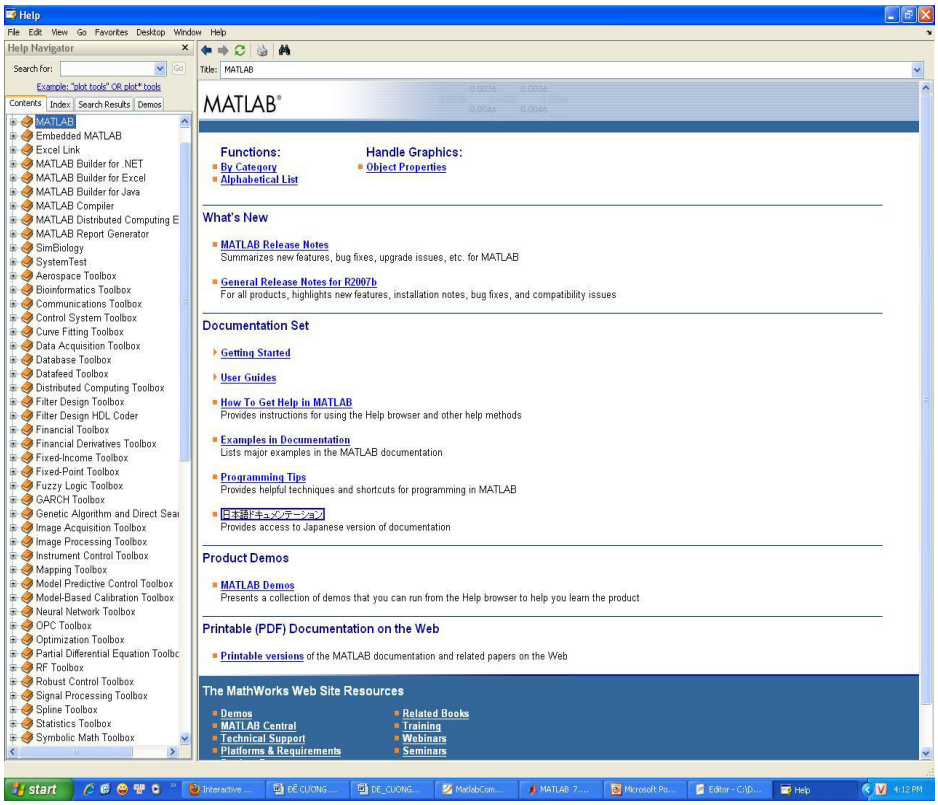
```
>>save filename
>>save filename x
>>save filename x y z
```

Trong đó *filename* là tên file dữ liệu người sử dụng muốn dùng để lưu các biến *x, y, z* có trong Workspace.

Có thể lưu một hoặc nhiều biến từ Workspace vào file bằng cách thêm tên biến (ví dụ x, y, z…) vào sau chuỗi ký tự biểu diễn tên file.

1.2.4 Help trong MATLAB

Về nguyên tắc ta có thể truy xuất mọi thông tin về MATLAB sử dụng chức năng trợ giúp ***Help***. Dưới đây là một số cách sử dụng đối với chức năng này.



Hình 1. 2. Help Browser

- Sử dụng trợ giúp khi biết tên hàm: trong cửa sổ lệnh, khi gõ `>>help tên_ham`, file M-help tương ứng sẽ hiển thị trong cửa sổ lệnh, bao gồm mô tả tóm tắt chức năng của hàm, cú pháp và các hàm liên quan.
- Tìm kiếm thông tin liên quan về một từ khóa: trong cửa sổ lệnh, khi gõ `>>lookfor tu_khoa`, MATLAB sẽ hiển thị tất cả các chủ đề liên quan đến từ khóa vừa nhập, lookfor đặc biệt hữu ích khi không nhớ chính xác cách viết của tên hàm.
- Sử dụng Help Browser: trình duyệt tích hợp với cửa sổ MATLAB cho phép tìm kiếm và đưa ra kết quả dưới dạng tài liệu HTML. Help Browser có thể mở theo các cách: sử dụng phím help trên thanh công cụ, gõ helpbrowser hoặc doc trong cửa sổ lệnh, chọn Help\Menu help trên thanh thực đơn. Cửa sổ Help Browser bao gồm hai phần chính, Help Navigator ở bên trái cho phép tìm kiếm thông tin, bên phải là nội dung thông tin tìm được. Trong cửa sổ Help Navigator, thông tin có thể được tìm theo các phương thức:
  - Contents: Các mục tìm kiếm sắp xếp theo danh sách.
  - Index: Tìm kiếm theo từ khóa.
  - Search results: danh sách các kết quả tìm được.
  - Demos: các chương trình giới thiệu có sẵn về các đề mục trong MATLAB.

1.3 Sơ lược về M-file

Bên cạnh việc cho phép người sử dụng thao tác trực tiếp trên cửa số lệnh, MATLAB cung cấp một chế độ soạn thảo văn bản qua đó người dùng có thể thực hiện một tập hợp các lệnh lặp lặp đi lặp lại hoặc tự xây dựng cho mình các hàm phù hợp với từng yêu cầu cụ thể. Người sử dụng có thể soạn thảo và lưu lại dưới dạng file văn bản có phần mở rộng là *.m*.

M-file có thể được tạo đơn giản bằng việc chọn ***New*** trên danh mục ***File*** và sau đó chọn ***Script***, hoặc gõ lệnh ***edit*** trên cửa số lệnh. Sau khi đã soạn thảo nội dung cho M-file, người sử dụng có thể chạy lần lượt tất cả các nội dung có trong M-file bằng cách gõ tên của file lên cửa sổ lệnh, hoặc có thể copy một phần của file rồi dán lên cửa sổ lệnh để thực thi.

Ở chương mở đầu này, tác giả chỉ giới thiệu sơ lược về M-file với một số ví dụ đơn giản, trong trường hợp sinh viên đã có kiến thức cơ bản về MATLAB, có thể chuyển sang mục 6.1 để tìm hiểu kỹ hơn về M-file.

Sau đây là một ví dụ về việc tính tổng, hiệu và thương của hai số *a* và *b* sử dụng M-file để thực hiện.

qh_2so.m

```
a = 5
b = 3
tong = a +b
hieu = a - b
tich = a*b
```

M-file *qh_2so.m* chứa 5 dòng lệnh trong đó 2 dòng đầu tiên gán các giá trị cho hai biến *a* và *b*, 3 dòng tiếp theo lần lượt tính tổng, hiệu, tích tương ứng của hai số trên. Khi gõ qh_2so trên cửa sổ lệnh, MATLAB sẽ hiển thị như sau:

```
>>qh_2so
     a = 5
     b = 3
     tong = 8
     hieu = 2
     tich = 15
```

Lợi ích đầu tiên của việc sử dụng M-file có thể thấy ở đây là nếu ta có nhiều cặp giá trị *(a, b)* thì chỉ việc thay giá trị tương ứng vào *qh_2so* và chạy lại file chứ không phải lặp lại việc thực hiện các lệnh tính toán ngoài cửa số lệnh. Bên cạnh đó, như đã nói ở trên, MATLAB cho phép người dùng tự xây dựng các hàm với các tham số đầu vào cũng như giá trị trả về tương ứng, sau đây là một ví dụ đơn giản về việc xây dựng hàm tính tổng của hai số:

tong.m

```
function z = tong(a,b)
z = a+b;
```

```
end
```

Cấu trúc cơ bản đối với một hàm trong MATLAB là dòng khai báo hàm bắt đầu bằng từ khóa ***function***, tiếp đến là các thông tin liên quan về giá trị đầu ra và đầu vào, kết thúc hàm bằng từ khóa ***end***, phần nội dung ở giữa là các lệnh phù hợp theo yêu cầu bài toán. Hàm *tong.m* vừa xây dựng có thể được gọi trong cửa sổ lệnh như sau:

```
>>t1 = tong(3,2)
t1 =
     5
>>t2 = tong(6,4)
t2 =
     10
```

1.4 Tóm tắt chương 1

Một số ký tự đặc biệt	
:	Tạo khoảng cách giữa các phần tử cách đều, biểu diễn một hàng/cột của ma trận
( )	Ngoặc bao tham số hàm, chỉ số mảng, vector
[ ]	Ngoặc bao phần tử mảng, vector
.	Dấu ngăn cách phần nguyên và phần thập phân
…	Ký hiệu dòng liên tục
,	Ngăn cách các câu lệnh và các phần tử trên cùng một hàng
;	Ngắt cột và không hiển thị kết quả sau khi thực hiện lệnh
%	Thể hiện phần chú thích và các định dạng đặc biệt
Nhóm lệnh quản lý	
clc	Xóa màn hình của cửa sổ lệnh
clear	Xóa biến khỏi bộ nhớ
help	Trợ giúp khi biết chính xác tên hàm
lookfor	Trợ giúp khi không biết chính xác tên hàm

quit	Thoát khỏi chương trình MATLAB
who	Hiển thị các biến hiện hành (danh sách rút gọn)
whos	Hiển thị các biến hiện hành (danh sách đầy đủ)
Một số biến và hằng đặc biệt	
ans	Kết quả gần nhất thu được
i,j	Đơn vị phần ảo, tuy nhiên i và j cũng thường xuyên được sử dụng như là chỉ số trong các vòng lặp, để tránh xung đột xảy ra, trong vòng lặp nên sử dụng ii hoặc jj làm chỉ số
inf	Vô cùng
nan	Kết quả số không được định nghĩa (Not a Number)
pi	Số π
Nhóm lệnhnhập xuất dữ liệu	
input	Nhập dữ liệu từ bàn phím
load	Đọc dữ liệu từ file, định dạng biến giống nhau
fscanf	Đọc dữ liệu từ file, định dạng biến khác nhau
disp	Xuất ra màn hình nội dung của một mảng, một xâu
fprintf	Lưu dữ liệu có định dạng ra file hoặc ra màn hình
Định dạng cho fprintf và fscanf	
%c	Ký tự đơn
%s	Xâu ký tự
%d	Giá trị nguyên
%f	Giá trị dấu chấm động
\n	Xuống dòng
\t	Chèn tab theo phương ngang
\v	Chèn tab theo phương dọc
Định dạng số hiển thị	
format short	Hiển thị 4 chữ số sau dấu phẩy, mặc định
format long	Hiển thị 15 chữ số sau dấu phẩy
format short e	Giống định dạng short nhưng có thêm phần lũy thừa

format long e	Giống định dạng long nhưng có thêm phần lũy thừa
format bank	Định dạng theo quy định ngân hàng cho dollars và cents, hai chữ số sau dấu phẩy

Bảng 1. 2. Tóm tắt các hàm sử dụng trong chương 1

1.5 Bài tập chương 1

Bài 1.1 Nhập lệnh

Nhập lần lượt các lệnh sau, trước khi nhập hãy dự đoán kết quả in ra màn hình

a) `>>a = 1 + 2 + 3 + 4 + 5 +6`

b) `>>b = 1 + 2 + 3 + 4 + 5 + 6;`

c) `>>b +3`

d) `>>1 + 2 + 3 + 4 + 5 + 6`

e) `>>c = 10*ans`

Đáp án:

a) a = 21 giá trị của tổng 1 + 2 + 3 + 4 + 5 +6 lưu vào biến a, kết quả *a* = 21 hiển thị trên cửa sổ lệnh.

b) b = 21, kết quả của phép tính không hiển thị trên màn hình cửa sổ lệnh do có dấu **;** ở cuối câu lệnh, tuy nhiên giá trị của tổng 1 + 2 + 3 + 4 + 5 +6 vẫn được lưu vào biến *b.*

c) ans = 24, kết quả phép tính *b + 3* hiển thị trên màn hình cửa sổ lệnh, trong trường hợp phép tính không xác định biến lưu kết quả như câu này, kết quả sẽ tự động gán vào biến *ans*.

d) ans = 21

e) c = 210 Kết quả câu này phụ thuộc vào câu *d* do có sử dụng biến *ans* trong phép tính, do ở câu *d* ta có *ans = 21* nên *c = 210*.

Bài 1.2 Các phép tính cơ bản

Thực hiện các phép tính sau trong MATLAB

a) $\frac{3+2^2}{16+5^2}$

b) $4^{2/2}$

c) $\sin\frac{\pi}{4}$

d) $\log(e)$

Đáp án:

a) 0.1707
b) 2.5198
c) 0.7071
d) $\log(\exp(1)) = 1$

Lưu ý: thứ tự ưu tiên các phép tính trong MATLAB. Trong MATLAB, các phép tính được thực hiện theo thứ tự như quy định trong đại số, trong đó *lũy thừa* thực hiện trước, tiếp theo là các phép *nhân, chia* rồi đến *cộng, trừ*. Tuy nhiên khi xuất hiện ngoặc ( ), phép tính trong ngoặc sẽ được ưu tiên trước. Đối với những phép tính có cùng mức ưu tiên, trình tự tính toán tiến hành từ trái qua phải.

Ví dụ: Phép tính dưới đây được thực hiện trong MATLAB như sau:

$$\frac{1}{2+3^2}+\frac{4}{5}\times\frac{6}{7}$$

```
>>1/(2 + 3^2) + 4/5 * 6/7

ans =

    0.7766
```

Tuy nhiên trong trường hợp không có ngoặc

```
>>1/2 + 3^2 + 4/5 * 6/7

ans =

    10.1857
```

Bài 1.3 Tính toán với biến

Gán cho x giá trị 2, thực hiện các phép tính sau trong MATLAB

a) $\frac{x^2}{6}$
b) e^{1+x^2}
c) $\frac{x}{\sqrt{1+x^2}}$
d) $x^3\sin(x^2)$
e) $x^{1/2}$

f) $\frac{\tan^{-1}(x)}{1+x^2}$

Đáp án: trước tiên cần nhập >>x = 2

a) 1.3333
b) 148.4132
c) 0.8944
d) -6.0544
e) 1.2599
f) 0.2214

Bài 1.4 Làm tròn

Kiểm tra giá trị của các phép tính sau:

a) round(6/9)
b) floor(6/9)
c) ceil(6/9)

Đáp án:

a) 1
b) 0
c) 1

Bài 1.5 Biến và hằng đặc biệt

Gõ các lệnh sau vào MATLAB và dự đoán kết quả thu được

a) exp(i*pi)
b) 1/0
c) Inf/Inf

Đáp án:

a) $e^{i\pi} = \cos(\pi) + i\sin(\pi) = -1 + 0i$

MATLAB: ans =

-1.0000 + 0.0000i

b) Inf
c) NaN

Bài 1.6 Save và Load

Dùng lệnh save tạo file *bai1_6* chứa hai biến *sv* và *diem* có giá trị như sau:

```
>>sv = [1;2;3;4;5];
>>diem = [6.25;7;8.50;9.75;10];
```

Xóa Workspace, dùng lệnh load để tải lại các biến *sv, diem* vào Workspace

Đáp án:

```
>>save bai1_6 sv diem
>>clear all
>>load bai1_6
```

Có thể sử dụng load để tải vào Workspace các biến cụ thể từ file theo cú pháp **load filename var**, ví dụ ở bài này có thể dùng >>load vd1_6 diem để chỉ load biến *diem* từ file vào Workspace.

Bài 1.7 fopen và fscanf

Sử dụng một chương trình soạn thảo bất kỳ, tạo file *bai1_7.txt* có nội dung là một ma trận như sau:

1 2 3 4

5 6 7 8

0 0 0 0

Sử dụng **fopen** và **fscanf** để hiện thị lại ma trận trên lên màn hình cửa sổ lệnh.

Đáp án: để sử dụng **fscanf** trước tiên ta cần thông số **fid** lấy từ lệnh **fopen**

```
>>fid = fopen('bai1_7.txt');
```

Lưu ý: khi dùng fopen, tên file đưa vào không có phần mở rộng, MATLAB sẽ tự động tìm .mat file có tên như tên đưa vào, nếu **fopen** không thể mở được file, **fid** sẽ nhận giá trị = -1.

```
>>fscanf(fid, '%d', [4,3]);
```

MATLAB làm việc với ma trận theo cột, máy tính xử lý file theo dòng. File *bai1_7.txt* có ma trận 3 hàng 4 cột tương đương với số dòng của file là 3. Khi thực thi, **fscanf** đọc từng số của file, lấy 4 số đầu của hàng 1, tiếp đến 4 số của hàng 2, cuối cùng là 4 số của hàng 3, đưa tất cả 12 số này vào một ma trận một cột. Sinh viên có thể tự kiểm tra lại bằng cách nhập lệnh >>fscanf(fid, '%d') thay vì >>fscanf(fid, '%d', [4,3]). Ở ví dụ này ta sử dụng [4,3] để yêu cầu MATLAB tải dữ liệu từ file vào ma trận 4 hàng 3 cột. Như vậy 4 phần tử của dòng thứ nhất trong file được đưa vào 4 vị trí tương ứng của cột 1 ma trận mới tạo, 4 phần tử của dòng thứ hai được đưa từ file vào 4 vị trí tương ứng của cột 2… Ma trận thu được sau bước này sẽ có dạng:

1 5 0

2 6 0

3 7 0

4 8 0

Để thu được ma trận như trong file, ta cần thực hiện bước chuyển vị ma trận

```
>>ans'
```

Bài 1.8 fprintf in ra màn hình

Load lại hai biến *sv, diem* ở bài 1.6 vào Workspace, dùng lệnh **fprintf** in kết quả sau ra màn hình trong đó phần điểm có hai chữ số sau dấu thập phân.

1 6.25

2 7.00

3 8.50

4 9.75

5 10.00

Đáp án:

```
>>ds = [sv'; diem'];
>>formatSpec='%d\t%.2f\n';
>>fprintf(formatSpec,ds);
```

Hoặc

```
>>fprintf('%d\t%.2f\n',ds)
```

Bài 1.9 fprintf xuất ra file

Sử dụng lệnh **fprintf**tạo file *bai1_9.txt* có nội dung sau (sử dụng biến ds trong bài 1.8):

Bang diem sinh vien

STT Diem

1 6.25

2 7.00

3 8.50

4 9.75

5 10.00

Đáp án:

```
>>fid=fopen('bai1_9.txt','w');
>> fprintf(fid,'Bang diem sinh vien\n');
>> fprintf(fid,'STT\tDiem\n');
>> fprintf(fid,'%d\t%.2f\n',ds);
```

```
>> fclose(fid);
```

Bài 1.10 Input

Tạo biến *r* có giá trị nhập vào từ bàn phím là bán kính của hình cầu theo cm. In ra màn hình thể tích và diện tích bề mặt hình cầu theo mẫu sau:

The tich hinh cau la V cm khoi

Dien tich hinh cau la S cm vuong

trong đó giá trị thể tích và diện tích có hai chữ số sau dấu thập phân

Công thức tính thể tích và diện tích bề mặt tương ứng: $V = \frac{4}{3}\pi \times r^2$, $S = 4\pi \times r^2$

Đáp án:

```
>>r = input('Nhap ban kinh hinh cau (cm): ');
>>tt = 4/3*pi*r^3;
>>dt = 4*pi*r^2;
>>fprintf('\nThe tich hinh cau la %.2f cm khoi\nDien
tich hinh cau la %.2f cm vuong\n',tt,dt);
```

Bài 1.11 Đổi đơn vị

Tạo biến *pounds* để lưu trọng lượng theo pounds có giá trị nhập vào từ bàn phím, đổi ra kilograms và gán giá trị cho biến *kilos*. Giá trị quy đổi là:

1 kilogramm = 2.2 pounds

Bài 1.12 Đổi đơn vị (tiếp)

Tạo biến *ftemp* để lưu nhiệt độ theo Fahrenheit (F) có giá trị nhập vào từ bàn phím, đổi ra độ Celcius (C) và lưu kết quả vào biến *ctemp*. Giá trị quy đổi là:

$$C = (F - 32) \times \frac{5}{9}$$

Bài 1.13 Định dạng hiển thị

Tìm lựa chọn cho lệnh **format** để có thể hiển thị kết quả sau

>>5/16 + 2/7

ans =

67/112

Bài 1.14 Nhiệt độ cảm nhận và nhiệt độ thực tế

Gió thường làm cho không khí lạnh hơn so với thực tế. Công thức tính nhiệt độ cảm nhận khi biết nhiệt độ thực tế T (độ Celcius) và vận tốc gió V (km/h) (hai giá trị T, V nhập vào từ bàn phím):

$$WT = 0.045 \times (5.2735 \times \sqrt{V} + 10.45 - 0.2778 \times V) \times (T - 33.0) + 33$$

Tạo biến T, V lưu nhiệt độ cảm nhận và tốc độ gió có giá trị nhập từ bàn phím, tính giá trị nhiệt độ cảm nhận, in ra màn hình kết quả như sau:

Nhiet do thuc te: (do C)

Toc do gio: (km/h)

Nhiet do cam nhan: (do C)

Kiểm tra với T = 20 (do C), V = 40(km/h)

Bài 1.15 rand

Hàm rand cho phép tạo số thực trong khoảng (0, 1)

a) Tạo số thực trong khoảng (0, 10)
b) Tạo số thực trong khoảng (0, 20)
c) Tạo số thực trong khoảng (20,50)
d) Tạo số nguyên trong khoảng (0, 10)
e) Tạo số nguyên trong khoảng (0,11)
f) Tạo số nguyên trong khoảng (50, 100)

Bài 1.16 Biểu diễn số thực

Sử dụng **fprintf** biểu diễn số thực 12345.6789

a) Trong khoảng rộng 10 ký tự với 4 chữ số phần thập phân
b) Trong khoảng rộng 10 ký tự với 2 chữ số phần thập phân
c) Trong khoảng rộng 6 ký tự với 4 chữ số phần thập phân
d) Trong khoảng rộng 2 ký tự với 2 chữ số phần thập phân

Bài 1.17 Biểu diễn ký tự

Tạo hai biến sau

x = 12.34

y = 4.56

Điền vào chỗ trống trong lệnh fprintf dưới đây để có thể hiển thị các kết quả

a) >>fprintf(

0123456789

0000012.340

b) >>fprintf(

0123456789
12

c) >>fprintf(
0123456789
4.6

d) >>fprintf(
0123456789
4.6 !

(Gợi ý: Để chuyển đổi một chữ thành số tương ứng trong bảng mã ASCII sử dụng lệnh **double** hoặc **int32**, ví dụ >>double('a') sẽ đưa ra kết quả 97. Để chuyển ngược lại dùng hàm **char**, ví dụ >>char(97) sẽ đưa ra kết quả a.

Bài 1.18 Bài toán thu nhập

Trung bình con người sử dụng 8 đến 10% thu nhập cho thực phẩm. Tạo biến lưu giá trị thu nhập một năm của một người nhập từ bàn phím (dollars), in ra màn hình số tiền bình quân người đó sử dụng trong một tháng để mua thực phẩm (biểu diễn bằng số thập phân có hai chữ số sau dấu thập phân):

Thu nhap mot nam cua ban la:

Moi thang ban su dung tu X den Y dollars de mua thuc pham

Bài 1.19 Vận tốc máy bay

Vận tốc máy bay thường ghi theo miles/h hoặc m/s. Tạo biến lưu vận tốc máy bay theo miles/h nhập vào từ bàn phím và in ra màn hình vận tốc m/s. Biết rằng 1h = 3600s, 1mile = 5280 feet, 1 foot = 0.3048m (feet là số nhiều của foot)

Nhap van toc may bay theo (miles/hour):

Van toc may bay theo meters/second la:

Bài 1.20 Diện tích hình chữ nhật

Tạo biến lưu chiều dài và chiều rộng của hình chữ nhật theo cm nhập từ bàn phím, in ra màn hình diện tích hình chữ nhật theo m^2 (biểu diễn bằng số thập phân chính xác đến 0.01)

Nhap chieu dai hinh chu nhat theo cm:

Nhap chieu rong hinh chu nhat theo cm:

Dien tich hinh chu nhat la S (met vuong)

Chương 2. VECTOR VÀ MA TRẬN

2.1 Vector và ma trận trong MATLAB

Vector và ma trận trong MATLAB được dùng để lưu một tập hợp các giá trị cùng kiểu. Một vector có thể là vector hàng hoặc vector cột. Ma trận có thể hiểu như một bảng các giá trị. Chiều của ma trận thường biểu diễn dưới dạng $m \times n$ trong đó m là số hàng, n là số cột tương ứng của ma trận. Nếu một vector có n phần tử, chiều của vector hàng tương ứng được biểu diễn là $1 \times n$, vector cột được biểu diễn là $n \times 1$.

Một giá trị vô hướng có chiều là 1×1. Vector và giá trị vô hướng thực ra là trường hợp đặc biệt của ma trận. Hình dưới từ trái qua phải là biểu diễn tương ứng của giá trị vô hướng, vector cột, vector hàng, ma trận.

1

1
2
3

1	2	3	4

1	2	3
4	5	6

MATLAB được viết ra với mục đích làm việc với các ma trận, do đó trong MATLAB rất dễ dàng tạo các biến vector và ma trận cũng như chứa rất nhiều hàm và phép toán xử lý vector, ma trận.

2.2 Thiết lập ma trận trong MATLAB

Thiết lập vector hàng

Có nhiều cách để thiết lập vector hàng trong MATLAB, trong đó trực tiếp nhất là nhập giá trị các phần tử của vector vào trong ngoặc vuông **[ ]**, ngăn cách giữa các phần tử bởi ký tự trống hoặc dấu phẩy. Ví dụ: Cả hai câu lệnh dưới đây cùng tạo ra vector v.

```
>>v = [ 1 2 3 4];
>>v = [ 1, 2, 3, 4]
v =
```

```
     1    2    3    4
```

Trong trường hợp các phần tử trong ma trận có giá trị cách đều nhau, dấu **:** có thể sử dụng để thiết lập vector, trong trường hợp này không cần sử dụng ngoặc vuông **[ ]** nữa. Ví dụ:

```
>>v1 = 1 : 5
v1 =
     1    2    3    4    5
```

Dấu **:** cũng có thể sử dụng để thiết lập vector mà các phần tử cách nhau một khoảng cho trước theo công thức (số đầu : khoảng cách : số cuối).

Ví dụ: Tạo vector gồm các số nguyên từ 1 đến 9 cách đều 2:

```
>>v2 = 1 : 2 : 9
v2 =
     1    3    5    7    9
```

Ví dụ: Dự đoán và kiểm tra kết quả bằng MATLAB khi khoảng cách nhập vào dẫn đến việc vượt quá khoảng định nghĩa bởi số đầu : số cuối

```
>>1 : 2 : 6
```

Lệnh này sẽ tạo một vector bao gồm các phần tử 1, 3, 5. Tăng thêm 2 vào 5 sẽ vượt qua 6, vì thế vector sẽ dừng ở 5. Kết quả hiển thị sẽ là:

```
1    3    5
```

Tương tự ta có hàm ***linspace*** (lineary spaced), cho phép tạo vector có các phần tử cách đều theo cú pháp sau:

>>linspace (x, y, n)

Trong đó *n* là số phần tử sẽ được tạo ra nằm trong khoảng giới hạn giữa *x* và *y*. Đối với trường hợp *n = 1* hàm ***linspace*** sẽ trả về giá trị *y*.

Ví dụ: Lệnh sau sẽ tạo vector có 5 phần tử trong khoảng 3 đến 15:

```
>>ls = linspace (3, 15, 5)
ls =
     3    6    9    12    15
```

Vector cũng có thể được thiết lập dựa trên những biến có sẵn. Ví dụ: Vector mới được thiết lập dưới đây bao gồm tất cả các phần tử của vector *v2* và vector *ls*:

```
>>newvec = [v2   ls]
newvec =
     1    3    5    7    9    3    6    9    12    15
```

Truy xuất và thay đổi các phần tử trong một vector

Trong MATLAB, các phần tử trong một vector được đánh số lần lượt theo thứ tự, bắt đầu từ 1. Ví dụ: Chỉ số của các phần tử trong vector *newvec* được thể hiện dưới đây:

1	2	3	4	5	6	7	8	9	10
1	3	5	7	9	3	6	9	12	15

Mỗi phần tử trong vector có thể truy xuất đến bằng cách sử dụng tên vector và chỉ số của phần tử đó ở trong ngoặc **( )**. Ví dụ: Phần tử thứ 5 của vector *newvec* là 9:

```
>>newvec (5)
ans =
     9
```

Sử dụng dấu **:** cũng có thể truy xuất đến các phần tử trong một vector. Ví dụ: Câu lệnh sau sẽ lấy từ phần tử thứ tư đến phần tử thứ sáu của vector *newvec* và lưu giá trị vào vector *b*:

```
>>b = newvec (4 : 6)
b =
     7    9    3
```

Bên cạnh việc sử dụng dấu **:** các phần tử trong vector này có thể được truy xuất bởi các vector khác, gọi là vector chỉ số. Ví dụ: Câu lệnh sau sẽ lấy ra phần tử thứ nhất, thứ năm và thứ mười của vector *newvec*:

```
>>newvec ([1   5   10])
ans = 1   9   15
```

Giá trị của mỗi phần tử trong một vector có thể thay đổi bằng cách gán giá trị khác cho phần tử của vector đó ở vị trí tương ứng. Ví dụ: Phần tử thứ hai của vector *b* sau khi thực hiện câu lệnh dưới đây sẽ chứa giá trị là 11:

```
>>b (2) = 11
b =
     7   11   3
```

Khi gán giá trị cho một phần tử trong một vector mà chỉ số của phần tử đó không tồn tại, ta có thể mở rộng vector đó. Ví dụ vector *b* ở trên có 3 phần tử. Khi ta gán giá trị cho phần tử thứ tư của *b*, *b* trở thành vector có 4 phần tử:

```
>>b (4) = 1
b =
     7   11   3   1
```

Nếu chỉ số nhập vào và chỉ số phần tử cuối cùng của vector không phải là hai số liên tiếp, tất cả các phần tử ở giữa sẽ nhận giá trị 0. Ví dụ: Lệnh sau tiếp tục mở rộng vector *b*:

```
>>b (6) = 13
b =
     7   11    3    1    0    13
```

Thiết lập vector cột

Cách thông dụng để tạo vector cột là nhập giá trị các phần tử vào trong ngoặc vuông **[ ]**, ngăn cách bởi dấu **;**

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

Không thể sử dụng dấu **:** để tạo vector cột một cách trực tiếp. Tuy nhiên có thể áp dụng phép chuyển vị lên mọi vector hàng để tạo ra vector cột tương ứng.

```
>>r = 1 : 3;
>>c = r'
c =
     1
     2
     3
```

Thiết lập ma trận

Thiết lập một ma trận tương đương với việc tạo ra các vector hàng và vector cột trong ma trận đó. Trong đó các giá trị trên cùng một hàng ngăn cách nhau bởi dấu phẩy hoặc ký tự trống, và các hàng khác nhau ngăn cách bởi dấu chấm phẩy.

```
>>mat = [4   3   1; 2   5   6]
mat =
     4   3   1
     2   5   6
```

Lưu ý: Số phần tử ở mỗi hàng phải luôn bằng nhau. Trường hợp ma trận nhập vào có số phần tử ở các hàng là khác nhau, kết quả nhận được sẽ là thông báo lỗi.

```
>>mat = [3   5   7; 1   2]
??? Error using ==> vertcat
CAT arguments dimensions are not consistent
```

Để phân biệt các dòng trong ma trận, bên cạnh việc sử dụng dấu **;** còn có thể sử dụng phím **Enter**

```
>>newmat = [2   6   88
33   5   2]
newmat =
     2   6   88
     33  5   2
```

Ma trận gồm các số ngẫu nhiên có thể được thiết lập sử dụng hàm ***rand*** theo cú pháp sau:

```
>>rand(n)
>>rand(m,n)
```

Trong đó *m, n* là các tham số biểu diễn chiều của ma trận tương ứng tạo thành.

Khi truyền một tham số *n* cho hàm **rand**, ma trận tạo thành sẽ là một ma trận vuông $n \times n$, khi hàm **rand** có hai tham số *m, n*, ma trận tạo thành sẽ là ma trận *m* hàng *n* cột.

```
>>rand (2)
ans =
     0.3527   0.4068
     0.9982   0.1764
>>rand(1, 3)
ans =
     0.3754   0.4464   0.7862
```

Một số ma trận đặc biệt

Ma trận $m \times n$ mà tất cả các phần tử đều nhận giá trị 0, tạo bởi lệnh ***zeros(m, n)***

Ma trận $m \times n$ mà tất cả các phần tử đều nhận giá trị 1, tạo bởi lệnh ***ones(m, n)***

Ma trận $n \times n$ mà chỉ có các phần tử đường chéo nhận giá trị khác 0, tạo bởi lệnh ***diag(v)*** trong đó v là vector chứa các phần tử đường chéocủa ma trận đó.

Ma trận đơn vị $n \times n$ mà tất cả các phần tử trên đường chéo đều nhận giá trị 1, tạo bởi lệnh ***eye(n)***

Truy xuất và thay đổi các phần tử trong ma trận

Có thể truy xuất đến các phần tử đơn lẻ của ma trận sử dụng tên ma trận đi kèm với chỉ số hàng, chỉ số cột trong ngoặc **( )** (chỉ số hàng luôn đứng trước chỉ số cột). Ví dụ: Tạo một ma trận và truy xuất đến phần tử ở hàng thứ hai, cột thứ ba của ma trận đó.

```
>>mat = [2 : 4; 3 : 5]
mat =
     2   3   4
     3   4   5
>>mat (2, 3)
ans =
     5
```

Ví dụ: Truy xuất đến tất cả các phần tử nằm trên hàng một đến hàng hai, cột hai đến cột ba của ma trận *mat*.

```
>>mat (1 : 2, 2: 3)
ans =
     3   4
     4   5
```

Chỉ sử dụng dấu hai chấm **:** đồng nghĩa với việc lấy tất hàng hoặc cột của ma trận. Ví dụ: Truy xuất đến tất cả các cột của hàng thứ nhất (tất cả các phần tử trên hàng này).

```
>>mat (1, :)
ans =
     2   3   4
```

Ví dụ: Truy xuất đến tất cả các phần tử nằm trên cột hai của ma trận *mat*.

```
>>mat(:, 2)
ans =
     3
     4
```

Nếu chỉ sử dụng một chỉ số (**chỉ số tuyến tính**) để truy xuất đến các phần tử trong ma trận, MATLAB khi đó sẽ sắp xếp lại tất cả các phần tử ma trận đó vào một cột. Ví dụ ma trận *intmat* được tạo ở đây, phần tử thứ nhất và thứ hai của ma trận là hai phần tử tương ứng của cột thứ nhất, phần tử thứ ba và thứ tư của ma trận là phần tử thứ nhất và thứ hai của cột thứ hai.

```
>>intmat = [100   77; 28   14]
intmat =
     100   77
     28    14
>>intmat (1)
ans = 100
>>intmat (2)
ans = 28
>>intmat (3)
ans = 77
>>intmat (4)
ans = 14
```

Để thay đổi giá trị của các phần tử đơn lẻ trong ma trận, có thể truy xuất đến và gán cho phần tử đó giá trị mới. Ví dụ: Đối với ma trận *mat* tạo trước đó:

```
>>mat(1, 2);
ans =
     2   11   4
     3   4   5
```

Tương tự có thể thay đổi giá trị của cả một hàng hoặc một cột của ma trận:

```
>>mat (2, :) = 5:7
ans =
     2    11   4
     5     6   7
```

Để mở rộng ma trận, không thể chỉ thêm vào một phần tử vì khi đó sẽ làm số phần tử của mỗi hàng trở nên khác nhau mà cần thêm vào cả hàng hoặc cột. Ví dụ: Chèn thêm cột thứ tư vào ma trận *mat* ở trên:

```
>>mat (:, 4) = [9; 2]
mat =
     2   11   4   9
```

```
     5    6    7    2
```

Tương tự như đối với vector, nếu hàng hoặc cột thêm vào so với hàng hoặc cột cuối cùng của ma trận hiện thời không phải là liền nhau, MATLAB sẽ tự điền đầy ma trận bằng các giá trị 0. Ví dụ: Chèn thêm hàng thứ tư vào ma trận *mat* chỉ có hai hàng:

```
>>mat (4, :) = 2 : 2 : 8
ans =
     2   11    4    9
     5     6    7    2
     0     0    0    0
     2     4    6    8
```

Kích thước của ma trận

Các hàm ***length***, ***size*** trong MATLAB được sử dụng để xác định kích thước của vector và ma trận. Cú pháp gọi hàm như sau:

>>length(X)

>>size(X)

Trong đó *X* là vector, ma trận được quan tâm.

Hàm **length** trả về số phần tử của một vector. Hàm **size** trả về số hàng và số cột của ma trận hoặc vector. Ví dụ vector *vec* tạo dưới đây có bốn phần tử, như vậy giá trị trả về của hàm **length**(vec) sẽ có giá trị là 4. Vector *vec* cũng có thể coi là một ma trận một hàng bốn cột, như vậy kích thước của *vec* sẽ là 1 × 4.

```
>>vec = -2 : 1
vec = -2   -1    0    1
>>length (vec)
ans =
     4
>>size (vec)
ans =
     1    4
```

Hàm **size** trả về số hàng và số cột của một ma trận vì thế để có thể lưu các giá trị này vào các biến khác nhau ta có thể sử dụng một vector có hai biến ở vế

trái của câu lệnh. Biến *h* sẽ lưu giá trị đầu tiên trả về, tương ứng với số hàng của ma trận, biến *c* sẽ lưu giá trị số cột.

```
>>mat = [1 : 3; 5 : 7]'
mat =
     1  5
     2  6
     3  7
>>[h, c] = size (mat)
h =
     3
c =
     2
```

Khi sử dụng với ma trận, hàm **length** sẽ trả về số hàng hoặc số cột của ma trận đó, tùy vào giá trị nào lớn nhất.

```
>>length (mat)
ans =
     3
```

Ngoài ra trong MATLAB còn có hàm **numel** trả về tổng số phần tử trong một vector hoặc ma trận theo cú pháp sau:

```
N = numel(A)
```

Trong đó N là số phần tử trả về của mảng A

Đối với vector, hàm này có ý nghĩa tương đương hàm **length**, đối với ma trận, **numel** trả về giá trị là tích của số hàng và số cột của ma trận đó.

```
>>numel (vec)
ans =
     4
>>numel (mat)
ans =
     6
```

2.3 Các phép toán cơ bản đối với ma trận, vector

Phép cộng hai ma trận tương đương với việc cộng từng phần tử tương ứng của hai ma trận đó, điều này đồng nghĩa với việc hai ma trận phải có cùng kích thước. Phép cộng ma trận có thể biểu diễn như sau: $c_{ij} = a_{ij} + b_{ij}$. Trong MATLAB,

phép cộng hai ma trận có thể được thực thi sử dụng phép toán **+**. Tương tự **phép trừ** hai ma trận thực hiện bởi phép toán -, biểu diễn $c_{ij} = a_{ij} - b_{ij}$.

Phép nhân với một số là phép nhân tất cả các phần tử của một ma trận với một số, thực thi trong MATLAB bởi phép toán *****.

Phép nhân mảng cho phép nhân từng phần tử tương ứng của hai ma trận (chú ý không phải phép nhân ma trận), điều kiện là hai ma trận phải có cùng kích thước, được thực hiện trong MATLAB bởi phép toán **.***

```
>>A = [1 : 3; 4 : 6];
>>B = [100   10   1; 10   100   1];
>>C = A .* B
C =
     100  20   3
     40   500  6
```

Phép nhân ma trận A với ma trận B để thu được ma trận C chỉ được thực hiện khi số cột của ma trận A bằng với số hàng của ma trận B. Nếu ma trận A có kích thước $m \times n$, ma trận B phải có kích thước $n \times p$. Ma trận C thu được sẽ có số hàng bằng với số hàng của ma trận A và số cột bằng với số cột của ma trận B.

$$[A]_{m\times n}[B]_{n\times p} = [C]_{m\times p} \qquad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Phép lũy thừa ma trận A thực hiện liên tiếp phép nhân ma trận A với chính nó. Ma trận A cần phải là ma trận vuông để thực hiện phép tính này.

Ma trận chuyển vị của ma trận A là ma trận A có hàng và cột đổi vai trò cho nhau, ký hiệu A^T, trong MATLAB được thực hiện bởi phép toán **'**.

```
>>A = [1 : 3; 4 : 6];
>>AT = A'
AT =
      1   4
      2   5
      3   6
```

Ma trận nghịch đảo nếu kết quả của phép nhân ma trận A với một ma trân khác là một ma trận đơn vị, ta có thể nói ma trận kia chính là ma trận nghịch đảo của A. Trong MATLAB có thể tính ma trận nghịch đảo của ma trận A bằng hàm **inv**.

```
>>A = [1   2; 2   2]
A =
```

```
     1    2
     2    2
>>Ainv = inv(A)
Ainv =
     -1.0000   1.0000
      1.0000   -0.5000
>>A*Ainv
ans =
     1    0
     0    1
```

Nếu ma trận A là ma trận không khả nghịch, MATLAB sẽ đưa ra lời cảnh báo và cũng có thể đưa ra ma trận mà các phần tử đều là **Inf**.

```
>>inv([1   1; 0   0])
Warning: Matrix is singular to working precision
ans =
     Inf  Inf
     Inf  Inf
```

Định thức của ma trận vuông A trong MATLAB được tính theo hàm **det**.

Hạng của ma trận A trong MATLAB được tính theo hàm **rank**.

Phép nhân vô hướng của hai vector tương tự như phép nhân ma trận khi ta nhân vector hàng *a*với vector cột *b*, kết quả thu về là một giá trị vô hướng (một số). Trong MATLAB có thể thực hiện bằng cách sử dụng phép toán * giữa *a* và chuyển vị của *b*, hoặc sử dụng hàm **dot** theo cú pháp

```
>>C = dot(A, B);
```

Trong đó *C* là giá trị tích vô hướng của hai vector *A* và *B*

Ví dụ:

```
>>vec1 = [4   2   5   1];
>>vec2 = [3   6   1   2];
>>vec1 * vec2'
ans =
     31
>>dot(vec1, vec2)
ans =
     31
```

Phép nhân có hướng của hai vector trong không gian ba chiều là một vector vuông góc với hai vector trên, nói cách khác chính là vector pháp tuyến của mặt phẳng tạo bởi hai vector đó. Trong MATLAB phép toán này thực hiện bởi hàm **cross** theo cú pháp sau:

```
>>C = cross(A, B)
```

Trong đó C là vector hình thành bởi tích có hướng của hai vector A và B

```
>>vec1 = [4   2   5];
>>vec2 = [3   6   1];
>>cross(vec1, vec2)
ans =
     -28  11   18
```

2.4 Đại số tuyến tính

Xét hệ phương trình đại số tuyến tính $m \times n$ gồm m phương trình n ẩn số:

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + ... + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + ... + a_{2n}x_n = b_2$$

$$a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + ... + a_{3n}x_n = b_3$$

$$\cdots\cdots\cdots$$

$$a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + ... + a_{mn}x_n = b_m$$

Hệ phương trình này có thể biểu diễn dưới dạng ma trận Ax = b trong đó A là ma trận hệ số, x là vector cột chứa các ẩn, b là vector cột chứa các hệ số ở vế phải của ma trận.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{bmatrix}$$

A x b

Hệ phương trình trên có thể giải một cách đơn giản trong MATLAB sử dụng **phép chia trái (mldivide)**, hoặc nhân cả hai vế với ma trận nghịch đảo của A.

$$A^{-1}Ax = A^{-1}b$$
$$Ix = A^{-1}b$$
$$x = A^{-1}b$$

Ví dụ ta có hệ ba phương trình ba ẩn x_1, x_2, x_3

$$4x_1 - 2x_2 + 1x_3 = 7$$
$$1x_1 + 1x_2 + 5x_3 = 10$$
$$-2x_1 + 3x_2 - 1x_3 = 2$$

Viết lại hệ dưới dạng $Ax = b$

$$\begin{bmatrix} 4 & -1 & 1 \\ 1 & 1 & 5 \\ -2 & 3 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 10 \\ 2 \end{bmatrix}$$

Giải hệ dùng hàm **inv**

```
>>A = [4   -2   1; 1   1   5; -2   3   -1];
>>b = [7   10   2];
>>x = inv(A)*b
x =
     3.0244
     2.9512
     0.8049
>>x = A\b
x =
     3.0244
     2.9512
     0.8049
```

Lưu ý phân biệt phép chia trái, chia phải:

- C = A\B tương đương với C = A^{-1}*B, để thực hiện phép tính này, điều kiện là A phải là ma trận khả nghịch.
- C = A/B tương đương với C = A*B^{-1}, để thực hiện phép tính này, điều kiện là B phải là ma trận khả nghịch.
- (B/A) = (A'\B')'

Phép chia trái sử $x = A\backslash b$ dụng trong MATLAB để tìm nghiệm của hệ phương trình tuyến tính đôi khi đưa ra lời cảnh báo sau khi thực hiện lệnh như sau: >>Warning: Matrix is close to singular or badly scaled. Results maybe inaccurate. RCOND = … hoặc >>Warning: Matrix is singular to working precision.

Cảnh báo trên đồng nghĩa với việc kết quả thu được thường không chính xác. Tuy nhiên cũng có khi MATLAB không đưa ra cảnh báo nhưng kết quả thu được lại không đáng tin cậy do MATLAB đưa ra nghiệm bình phương tối thiểu của hệ (ví dụ trường hợp A không phải là ma trận vuông), vì vậy điều quan trọng khi sử dụng phép chia trái là luôn kiểm tra kết quả thu được bằng cách so sánh *A*x* và *b*.

Phép khử Gauss và Gauss – Jordan sử dụng trong việc giải hệ phương trình đại số tuyến tính đều bắt đầu bằng việc thiết lập ma trận $[A|b]$ và áp dụng các phép biến đổi lên các hàng của ma trận này (nhân tất cả các phần tử trong hàng với một số, đổi chỗ các hàng trong ma trận, thay thế một hàng bằng cách cộng hoặc trừ với các hàng khác) để thu về ma trận tam giác trên có cùng tập nghiệm. Phương pháp Gauss từ đây sẽ áp dụng thế ngược để tìm ra tập nghiệm của hệ phương trình trong khi phương pháp Gauss –Jordan sẽ tiếp tục khử để thu về ma trận đường chéo từ đó suy ra tập nghiệm của hệ phương trình.

Ví dụ:

```
>>a = [ 1   3   0; 2   1   3; 4   2   3]
a =
     1    3    0
     2    1    3
     4    2    3
>>b = [1   6   3]'
b =
     1
     6
     3
>>ab = [a   b]
ab =
     1    3    0    1
     2    1    3    6
     4    2    3    3
>>ab(2, :) = ab(2, :) - 2*ab(1, :)
ab =
     1    3    0    1
     0   -5    3    4
     4    2    3    3
>>ab(3, :) = ab(3, :) - 4*ab(1, :)
```

```
ab =
     1     3     0     1
     0     -5    3     4
     0     -10   3     -1
>>ab(3, :) = ab(3, :) – 2*ab(2, :)
ab =
     1     3     0     1
     0     -5    3     4
     0     0     -3    -9
>>ab(2, :) = ab(2, :) + ab(3, :)
ab =
     1     3     0     1
     0     -5    0     -5
     0     0     -3    -9
>>ab(1, :) = ab(1, :) + 3/5*ab(2, :)
ab =
     1     0     0     -2
     0     -5    0     -5
     0     0     -3    -9
```

Dạng bậc thang tối giản

Từ kết quả ở dạng đường chéo từ phép khử Gauss-Jordan, ma trận được đưa về dạng bậc thang tối giản, tương đương với tất cả các hệ số trên đường chéo thu được đều là 1, hay nói cách khác vector cột b chính là kết quả cần tìm. Trong MATLAB ta có thể đưa ma trận $[A|b]$ về dạng bậc thang tối giản $[I|b']$ để giải hệ phương trình đại số tuyến tính bằng hàm **rref**. Ví dụ đối với ma trận ở trên:

```
>>a = [ 1   3   0; 2   1   3; 4   2   3];
>>b = [1   6   3]';
>>ab = [a b];
>>rref(ab)
ans =
     1     0     0     -2
     0     1     0     1
```

```
     0     0     1     3
```

Nghiệm của hệ chính là giá trị của cột cuối cùng $x_1 = -2$, $x_2 = 1$, $x_3 = 3$

```
>>x = ans(:, end)
x =
     -2
     1
     3
```

2.5 Tóm tắt chương 2

Các lệnh đối với mảng	
linspace	Tạo mảng có các phần tử cách đều
find	Tìm chỉ số của các phần tử khác không trong mảng
max	Tìm giá trị lớn nhất trong mảng
min	Tìm giá trị nhỏ nhất trong mảng
prod	Tính tích các phần tử trong mảng
sum	Tính tổng các phần tử trong mảng
length	Tìm tổng số phần tử trong mảng
size	Tìm kích cỡ của mảng
cross	Tích có hướng của hai vector trong 3D
dot	Tích vô hướng của hai vector
Một số ma trận đặc biệt	
eye	Tạo ma trận đơn vị
ones	Tạo ma trận mà tất cả các phần tử đều nhận giá trị 1
zeros	Tạo ma trận mà tất cả các phần tử đều nhận giá trị 0
diag	Tạo ma trận đường chéo
Các lệnh giải hệ phương trình đại số tuyến tính	
det	Tính định thức ma trận
inv	Tính ma trận nghịch đảo
rank	Tính hạng của ma trận
rref	Đưa về dạng bậc thang tối giản

Bảng 2. 1. Tóm tắt các hàm sử dụng trong chương 2

2.6 Bài tập chương 2

Bài 2.1

Dùng dấu **:** tạo các vector hàng sau:

3 4 5 6

a) 1.0000 1.5000 2.0000 2.5000 3.0000
b) 5 4 3 2

Đáp án

```
a) >>3 : 6
b) >>1 : .5 : 3
c) >>5 : -1 : 2
```

Bài 2.2

Dùng hàm **linspace** tạo các vector sau:

```
a) >>4   6   8
b) >>-3   -6   -9   -12   -15
c) >>9   7   5
```

Đáp án

```
a) >>linspace(4, 8, 3)
b) >>linspace(-3, -15, 5)
c) >>linspace(9, 5, 3)
```

Bài 2.3

Dùng **:** và **'** tạo vector cột chứa các giá trị trong khoảng từ -1 đến 1 cách đều nhau 0.2.

Đáp án

```
>>[ -1 : .2 : 1]'
```

Bài 2.4

a) Tạo ma trận 2 × 3 chứa các số nguyên ngẫu nhiên từ 5 đến 20.
b) Tạo biến *hang* chứa số nguyên ngẫu nhiên từ 1 đến 3, tạo biến *cot* chứa số nguyên ngẫu nhiên từ 3 đến 5, tạo ma trận mà tất cả các phần tử đều nhận giá trị không có kích cỡ *hang* × *cot*.

Đáp án

```
a) >>round(5 + 15*rand(2, 3))
b) >>hang = round(3*rand);
   >>cot = round(3 + 2*rand);
   >>zeros(hang, cot)
```

Bài 2.5

Không nhập tất cả các phần tử, tạo ma trận sau:

mat =

7	8	9	10
12	10	8	6

Viết các lệnh truy xuất đến:

a) Phần tử ở hàng một cột ba của ma trận.
b) Toàn bộ hàng hai của ma trận.
c) Hai cột đầu của ma trận.

Đáp án

```
a) >>mat = (7:10; 12:-2:6);
   >>mat(1,3);
b) >>mat(2, :);
c) >>mat(:, 1:3);
```

Bài 2.6

Cho các ma trận sau:

$$A=\begin{bmatrix}1 & 2\\ 3 & 4\end{bmatrix} \qquad B=\begin{bmatrix}5 & 6\\ 7 & 8\end{bmatrix} \qquad C=\begin{bmatrix}1 & 2\\ 1 & 2\\ 1 & 2\\ 1 & 2\end{bmatrix}$$

Tạo ma trận D có ma trận A ở trên ma trận B tạo thành hai cột đầu, C là hai cột kế tiếp.

Đáp án

```
>>A = [1   2; 3   4];
>>B = [5   6; 7   8];
>>C = [1   1   1   1; 2   2   2   2]'
>>AB = [A;B];
>>D = [AB C];
```

Bài 2.7

Tạo các ma trận sau:

a) Ma trận đơn vị 6 x 6.
b) Ma trận không 5 x 10.
c) Ma trận một 5 x 15.

d) Ma trận ngẫu nhiên 5 x 5.
e) Ma trận đường chéo mà các giá trị trên đường chéo chạy từ 1 đến 5.

Đáp án

```
a) >> eye (6)
b) >>zeros (5,10)
c) >>ones (5,15)
d) >>rand (5)
e) >>e0 = 1:5;
   >>diag(e0)
```

Bài 2.8

Cho ma trận *A, b* và hệ phương trình tuyến tính *Ax = b*

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & -3 & 2 \\ 3 & 1 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} 6 \\ 14 \\ -2 \end{bmatrix}$$

Giải hệ phương trình sử dụng lệnh A\b và lệnh **rref**

Đáp án:

```
>>A = [1   2   3; 2   -3   2; 3   1   -1];
>>b = [6   14   -2];
>>x = A\b
x =
      1.0000
     -2.0000
      3.0000
>>A*x
```

Kiểm tra kết quả thu được , A*x có giá trị bằng với b, chứng tỏ x là %nghiệm của hệ phương trình Ax = b

```
>>rref([A, b])
```

Nghiệm thu được là duy nhất, do ma trận bậc thang tối giản chứa ma %trận đơn vị và ma trận kết quả

```
ans =
      1   0   0   1
      0   1   0  -2
      0   0   1   3
```

Bài 2.9

Cho hệ phương trình sau:

$$6x_1 + 2x_2 + 3x_3 = 0$$
$$4x_1 + 1x_2 - 2x_3 = 0$$
$$2x_1 + 1x_2 + 5x_2 = 0$$

Nhập vào MATLAB ma trận hệ số *A* và ma trận cột *b* tương ứng, tìm nghiệm của ma trận bằng cách sử dụng phép chia trái và **rref**.

Đáp án

```
>>A = [6   2   3; 4   1   -2; 2   1   5];
>>b = zeros(3,1);
>>A\b
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 6.728624e-018.
x =
     1.0e+015 *
     -2.6271
     9.0072
     -0.7506
>>rref([A b])
ans =
1.0000    0         -3.5000              0
0         1.0000    12.0000              0
0         0         0                    1.0000
```

Thế ngược ta có $x_1 = 3.5x_3$ và $x_2 = -12x_3$. Có thể thấy hệ phương trình có biến tự do x_3 và có thể nhận bất kỳ giá trị nào. Ví dụ chọn $x_3 = 2$ khi đó ta có:

$x_1 = 7,\ x_2 = -24$

```
>>x = [7; -24; 2];
>>A*x
ans =
     0
     0
     0
```

Bài 2.10

Cho hai ma trận *A*, *b* và hệ phương trình tuyến tính *Ax = b*

$$A = \begin{bmatrix} 12 & 3 \\ 21 & -3 \end{bmatrix} \qquad B = \begin{bmatrix} 4 \\ -4 \end{bmatrix}$$

Giải hệ phương trình sử dụng phép chia trái, tập nghiệm thu được có phải là duy nhất?

Tìm nghiệm tổng quát sử dụng hàm **rref**.

Đáp án:

```
>>A = [1   2   3; 2   1   -3];
>>b = [-4;   4];
>>x = A\b;
x =
          0
     0.0000
    -1.3333
>>A*x
```

Kiểm tra x có phải là nghiệm của phương trình

```
>>rref([ A   b])
```

Hệ có vô số nghiệm do số biến nhiều hơn số phương trình

```
ans =
     1    0    -3    4
     0    1     3    -4
```

Bài 2.11

Cho ma trận sau:

$$A = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 7 & 2 & 0 & 1 \\ 3 & -2 & -1 & -1 \\ 0 & 1 & 4 & 8 \end{bmatrix}$$

Dự đoán kết quả và kiểm tra lại bằng MATLAB ý nghĩa của lệnh sau:

```
>>A( [1 3], [1 3] ) = 10 * ones (2)
```

Bài 2.12

Nhập vào ma trận sau:

$$A = \begin{bmatrix} 1 & 2 & -1 & 3 \\ 7 & 2 & 0 & 1 \\ 3 & -2 & -1 & -1 \\ 0 & 1 & 4 & 8 \end{bmatrix}$$

a) Tính $A^T = B$
b) Tính $C = A * B$
c) Tính $D = B * A$
d) Tính $3A + 5B^3$

Bài 2.13

Nhập vào các ma trận sau:

$$A = \begin{bmatrix} 3 & 2 & 1 \\ 0 & 5 & 2 \\ 1 & 0 & 3 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Tính trong MATLAB các phép tính sau, giải thích trong trường hợp MATLAB không thể thực hiện phép toán.

*A * B*

*B * A*

I + A

A . I*

Bài 2.14

Tạo ma trận *4 × 2* và lưu vào một biến. Thay thế hàng thứ hai của ma trận đó bằng vector cột chứa các giá trị 3 và 6.

Bài 2.15

Tạo vector *x* chứa 20 phần tử cách đều nhau trong khoảng $-\pi$ đến π. Tạo vector *y* chứa *sin(x)*.

Bài 2.16

Tạo ma trận *3 × 5* có các phần tử là các số nguyên ngẫu nhiên trong khoảng -5 đến 5. Xóa hàng thứ ba của ma trận.

Bài 2.17

Tạo vector *vec*. Tìm tất cả các lệnh cho phép truy xuất đến phần tử cuối cùng của *vec* (giả sử không biết *vec* có bao nhiêu phần tử).

Bài 2.18

Tạo ma trận *mat*. Tìm tất cả các lệnh cho phép truy xuất đến phần tử cuối cùng của *ma* (giả sử không biết số phần tử, số hàng, số cột của *mat*).

Bài 2.19

Viết các lệnh yêu cầu nhập vào từ bàn phím các giá trị *n, low, high* (*high* lớn hơn *low)*.

Từ các giá trị nhập vào tạo ma trận đường chéo $n \times n$ trong đó đường chéo chứa các giá trị nguyên ngẫu nhiên từ *low* đến *high*.

Bài 2.20

Viết các câu lệnh yêu cầu nhập vào từ bàn phím hai số nguyên dương lưu vào các biến *m, n*. Viết câu lệnh nhập vào số nguyên *k*. Viết câu lệnh tạo ma trận có kích cỡ $m \times n$ mà tất cả các phần tử đều có giá trị *k* (trong câu lệnh có sử dụng lệnh **ones**).

Bài 2.21

Cho hệ phương trình đại số tuyến tính $Ax = b$.

Nhập các ma trận *A, b* và giải sử dụng các lệnh chia trái và **rref**.

$$A = \begin{bmatrix} 11 & -1 \\ 12 & -1 \\ 11 & 1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \\ 3 \\ \sqrt{6} \end{bmatrix}$$

Bài 2.22

Cho hệ bốn phương trình bốn ẩn sau:

Nhập các ma trận vào MATLAB và giải.

$$\begin{aligned} 4x_1 - x_2 + 3x_4 &= 10 \\ -2x_1 + 3x_2 + x_3 - 5x_4 &= -3 \\ x_1 + x_2 - x_3 + 2x_4 &= 2 \\ 3x_1 + 2x_2 - 4x_3 &= 4 \end{aligned}$$

Bài 2.23

Giải hệ phương trình đại số tuyến tính:

$$\begin{aligned} 2x_1 + 2x_2 + \quad x_3 &= 2 \\ x_2 + 2x_3 &= 1 \\ x_1 + \ x_2 + 3x_3 &= 3 \end{aligned}$$

Bài 2.24

Giải hệ phương trình đại số tuyến tính:

$$\begin{aligned} x_1 - 2x_2 + x_3 &= 2 \\ 2x_1 - 5x_2 + 3x_3 &= 6 \\ x_1 + 2x_2 + 2x_3 &= 4 \end{aligned}$$

Bài 2.25

Cho hệ phương trình đại số tuyến tính $Ax = b$, tìm nghiệm của hệ.

$$A = \begin{bmatrix} 10 \\ 12 \\ 14 \end{bmatrix} \qquad b = \begin{bmatrix} 10 \\ 16 \\ 17 \end{bmatrix}$$

Bài 2.26

Cho hệ phương trình đại số tuyến tính $Ax = b$, tìm nghiệm của hệ.

$$A = \begin{bmatrix} 1 & 1 \\ 1 & -1 \\ 1 & -3 \end{bmatrix} \qquad b = \begin{bmatrix} 2 \\ 0 \\ 3 \end{bmatrix}$$

Chương 3. TÍNH TOÁN SỐ VỚI MATLAB

3.1 Số phức

Số phức được viết dưới dạng tổng quát *a + bi* trong đó *a* là phần thực, *b* là phần ảo, *i* có giá trị $\sqrt{1}$. Trong MATLAB *i* và *j* là những hằng xây dựng sẵn trả về giá trị $\sqrt{-1}$. Số phức trong MATLAB có thể biểu diễn theo *i* hoặc theo *j*, ví dụ *5 + 2i* hoặc *3 – 4j*. Không cần sử dụng ký hiệu phép nhân * giữa phần ảo và các hằng *i*, *j*.

Có sự khác biệt giữa *"3i"* và *"3*i"* ? Điều này phụ thuộc vào việc trước đó *i* có được dùng cho một biến nào khác hay không. Nếu *i* trước đó đã sử dụng như là một biến (ví dụ trong vòng lặp *for*) thì câu lệnh *3*i* sẽ sử dụng giá trị được gán cho biến *i* và như vậy kết quả sẽ không phải là một số phức. Vì thế, để tránh nhầm lẫn khi biểu diễn số phức trong MATLAB ta nên sử dụng *1i* hoặc *1j* thay vì chỉ *i* hoặc *j*. Kết quả trả về khi sử dụng *1i* hoặc *1j* luôn luôn là số phức bất kể trước đó *i* và *j* có được gán giá trị hay không.

Trong MATLAB có hàm ***complex*** sẽ trả về số phức gồm phần thực và phần ảo khi nhận được hai tham số tương ứng, khi chỉ nhận một tham số, ***complex*** coi như phần ảo không có giá trị và chỉ biểu diễn phần thực, cú pháp gọi hàm như sau:

```
>>C = complex (A, B)
```

Trong đó giá trị trả về *C* sẽ được biểu diễn dưới dạng phức *C = A + Bi*

Một số ví dụ biểu diễn số phức:

```
>>z1 = 4 + 2i
z1 =
     4.0000 + 2.0000i
>>z2 = sqrt( -5 )
z2 =
     0 + 2.2361i
>>z3 = complex (3,-3)
z3 =
     3.0000 - 3.0000i
```

```
>>z4 = 2 + 3j
z4 =
     2.00 + 3.0000i
>>z5 =  (-4)^1/2
z5 =
     0.0  + 2.0000i
```

Lưu ý là ngay cả khi dùng *j* để biểu diễn số phức trong câu lệnh thì MATLAB vẫn sử dụng *i* trong các kết quả trả về. Trong MATLAB còn có các hàm ***real*** và ***imag*** trả về phần thực và phần ảo của một số phức, cú pháp gọi hàm như sau

```
>>real (X)
>>imag (X)
```

Trong đó *X* là số phức, giá trị trả về của hai hàm ***real*** và ***imag*** sẽ là phần thực và phần ảo tương ứng của *X*.

```
>>real(z1)
ans =
     4
>>imag(z3)
ans =
     -3
```

Trường hợp muốn xuất ra màn hình giá trị của một số phức, lệnh ***disp*** sẽ hiển thị cả phần thực và phần ảo trong khi đó lệnh ***fprintf*** chỉ hiển thị phần thực trừ khi có lệnh in đồng thời cả hai thành phần của số phức.

```
>>disp(z1)
ans =
     4.0000 + 2.0000i
>>fprintf(' %f \n', z1)
4.0000
>>fprintf(' %f  %f  \n', real(z1), imag(z1))
4.0000  2.0000
>>fprintf(' %f  +  %fi \n', real(z1), imag(z1))
4.0000 + 2.0000i
```

Hàm ***isreal*** trả về giá trị logic 1 nếu đúng là số phức không có giá trị phần ảo, trả về giá trị logic 0 nếu số phức có giá trị phần ảo (kể cả giá trị phần ảo là 0).

```
>>isreal(z1)
ans =
     0
>>z6 = complex(3)
z6 =
     3
>>isreal(z6)
ans =
     0
>>isreal(3.3)
ans =
     1
```

Đối với *z6* mặc dù giá trị hiển thị trên cửa sổ lệnh là 3 nhưng biểu diễn trong MATLAB vẫn là 3.0000 + 0.0000i (xem Workspace) vì thế giá trị trả về của hàm ***isreal*** trong trường hợp này là 0 do *z6* vẫn chứa phần ảo.

Biểu diễn số phức bằng tọa độ cực và lũy thừa của loga tự nhiên

Mọi số phức $z = a + bi$ đều có thể biểu diễn bởi điểm *(a, b)* hoặc dưới dạng vector trên mặt phẳng phức trong đó trục hoành và trục tung biểu diễn phần thực và phần ảo của z. Do một vector có thể được biểu diễn trong hệ tọa độ Đề các hoặc hệ tọa độ cực, một số phức cũng có thể biểu diễn trong hệ tọa độ cực sử dụng các phép quy đổi sau:

Biến đổi hệ tọa độ cực sang hệ tọa độ Đề các:

$a = r\cos\theta$

$b = r\sin\theta$

Biến đổi hệ tọa độ Đề các sang hệ tọa độ cực:

$$r = |z| = \sqrt{a^2 + b^2}$$

$$\theta = \tan^{-1}(\frac{b}{a})$$

$$z = a + bi = r\cos\theta + (r\sin\theta)i$$

Ngoài ra, số phức có thể biểu diễn dưới dạng lũy thừa của loga tự nhiên: $e^{i\theta} = \cos\theta + i\sin\theta$, biểu diễn dạng phức dạng lũy thừa có thể thu được trong MATLAB sử dụng lệnh exp(j*theta). Ví dụ:

```
>>exp(j*pi/2)
ans =
```

```
     0.0000 + 1.0000i
```

Trong MATLAB ta có thể sử dụng những hàm xây dựng sẵn để tìm giá trị của *r* và θ là **abs** và **angle** theo cú pháp sau:

```
>>abs (X)
>>angle (X)
```

Trong đó *X* là số phức, ***abs*** và ***angle*** sẽ trả về giá trị *r* và θ tương ứng của *X*, xét ví dụ:

```
>>z1 = 3 + 4i;
>>r = abs(z1)
r =
     5
>>theta = angle(z1)
theta =
     0.9273
>>r*exp(i*theta)
ans =
     3.0000 + 4.0000i
```

Cộng, trừ, nhân, chia số phức: Cho hai số phức $z_1 = a + bi$, $z_2 = c + di$

$z1 + z2 = a + bi + c + di = (a+c) + (b +d)i$

$z1 – z2 = a + bi – c – di = (a –c) + (b – d)i$

$z1 * z2 = (a + bi) * (c + di) = ac + (bc + ad)i –bd = (ac – bd) + (bc + ad)i$

Ví dụ:

```
>>z1 = 3 + 4i;
>>z2 = 1 - 2i;
>>z1*z2
ans =
     11.0000 - 2.0000i
```

Phép chia hai số phức có thể được thực hiện dễ dàng khi biểu diễn hai số phức dưới dạng lũy thừa của loga tự nhiên rồi thực hiện phép chia sau đó.

Ví dụ:

$$z_3 = \frac{z_2}{z_1} = \frac{2 + 1.5j}{1.25 + 2.5j}$$

```
>>z1 = 2 + 1.5j;
```

```
>>z2 = 1.25 + 2.5j;
>>abs1 = abs(z1);
>>abs2 = abs(z2);
>>angle1 = angle(z1);
>>angle2 = angle(z2);
>>abs3 = abs2/abs1;
>>angle3 = angle2 - angle1;
```

Hai phép tính z2/z1 và abs3*exp(j*angle3) đều đưa ra kết quả giống nhau.

```
>>z3 = z2/z1
z3 = 1.0000 + 0.5000i
```

Số phức liên hợp và giá trị tuyệt đối

Cho số phức dưới dạng $z = a + bi$, số phức $\overline{z} = a - bi$ gọi là số phức liên hợp của z. Độ lớn hay giá trị tuyệt đối của số phức được tính theo công thức $|z| = a^2 + b^2$. Trong MATLAB có thể sử dụng hai hàm sẵn có để tính dạng liên hợp và giá trị tuyệt đối của số phức là **conj** và **abs**.

```
>>z1 = 3 + 4i
z1 =
     3.0000 + 4.0000i
>>conj (z1)
ans =
     3.0000 - 4.0000i
>>abs (z1)
ans =
     5
```

3.2 Hàm vô danh

Hàm vô danh là dạng hàm số đơn giản, được thể hiện chỉ cần sử dụng một dòng lệnh trong MATLAB. Ưu điểm của hàm vô danh là không cần phải lưu trong các file văn bản soạn thảo bằng Matlab editor M-file, đơn giản hóa chương trình sử dụng những phép tính đơn giản, giảm thiểu số M-file cần dùng cho một chương trình. Hàm vô danh có thể được tạo trong cửa sổ lệnh hoặc trong các hàm văn bản. Cú pháp cho hàm vô danh trình bày như sau:

Ten_ham = @(ten_bien) than_ham

Ví dụ:

```
>>dt_hinhtron = @(r) pi*r.^2;
>>dt_hinhtron(4)
ans =
     50.2655
```

Trong trường hợp không truyền tham số cho hàm, vẫn phải sử dụng dấu ngoặc ( ) lúc khai báo và lúc gọi hàm. Ví dụ: Hàm vô danh sau in ra màn hình số thực ngẫu nhiên có hai chữ số ở phần thập phân:

```
>>in_nn = @( ) fprintf(‘ %.2f  \n’, rand);
>>in_nn ( )
0.95
```

Trường hợp gọi hàm chỉ có tên hàm mà không có dấu ngoặc, MATLAB sẽ hiển thị thân hàm lên màn hình.

```
>>in_nn
in_nn =
     @ ( ) fprintf(‘ %.2f \n’, rand)
```

3.3 Cực trị và nghiệm của hàm số một biến số

Trong MATLAB **cực trị** của hàm số có thể xác định trong khoảng x_1, x_2 của hàm số đó xác định từ đồ thị dùng hàm ***fminbnd*** theo cú pháp sau:

```
>>fminbnd (fun, x1, x2)
```

Trong đó *fun* là hàm số khai báo và *x1, x2* là khoảng giá trị chứa cực tiểu. Giá trị trả về của hàm ***fminbnd*** là cực tiểu của hàm *fun* trong khoảng x_1, x_2

Ví dụ: Cho hàm số $f(x) = x^3 – 2x - 5$, tìm cực tiểu của hàm số trong khoảng (0,2) và giá trị hàm số tại vị trí cực tiểu.

```
>>f = @(x) x^3 – 2*x -5;
>>x1 = fminbnd (f, 0, 2)
x1 =
     0.8165
>>y1 = f(x1)
y1 =
     -6.0887
```

Ví dụ: Cho hàm số $f(x) = x^3 – x^2 – 3\tan^{-1}(x) +1$, hàm số này có cực tiểu trong khoảng (0, 2) có thể xác định như sau:

```
>>clear all
>>f = @ (x) x^3 - x^2 - 3*atan(x) + 1;
>>x1 = fminbnd(f, 0, 2)
x1 =
     1.0878
>>f(x1)
ans  =
     -1.3784
```

Như vậy trên (0, 2) hàm số có một cực tiểu tại x = 1.0878 với giá trị -1.3784. Hàm số này cũng có một cực đại trong khoảng (-1, 0). MATLAB không có hàm để tìm trực tiếp cực đại của hàm số tuy nhiên ta có thể sử dụng **fminbnd** để tìm cực tiểu của hàm *–f(x)* trên khoảng tương ứng.

```
>>finv = @(x) -f(x);
>>x2 = fminbnd(finv, -1, 0)
x2 =
     -0.5902
>>finv(x2)
ans =
     -2.0456
```

Như vậy hàm số có một cực đại tại x = -0.5902 với giá trị -2.0456

Nghiệm của hàm số một biến số

Định nghĩa toán học về nghiệm của của hàm số *f(x)* là giao điểm của đồ thị hàm số đó và trục *Ox* hay giá trị x_0 mà tại đó $f(x_0) = 0$. Do MATLAB sử dụng dấu chấm động nên rất khó xác định chính xác điểm không của hàm số, nên thay vì tìm giá trị biến mà tại đó hàm số bằng không, MATLAB tìm điểm mà tại đó hàm số đổi dấu.

Cú pháp: x = fzero (fun, x0)

Trong đó x_0 là lân cận điểm mà hàm số giao với trục *Ox* hoặc *x0* là vector chứa hai giá trị x_1 và x_2 mà $f(x_1)$ và $f(x_2)$ khác dấu nhau. Hàm ***fzero*** cho phép xác định nghiệm của hàm số $f(x) = 0$ tại lân cận *x0*.

Ví dụ: Xét hàm số $f(x) = x^2 - 2x - 3$

Đồ thị hàm số này giao với trục *Ox* tại hai điểm $x_1 = -1$ và $x_2 = 3$

Để dùng MATLAB tìm các giá trị x_1, x_2 trước tiên cần khai báo hàm:

```
>>f = @(x) x^2 - 2*x - 3;
```

```
>>x1 = fzero(f, -2)
```

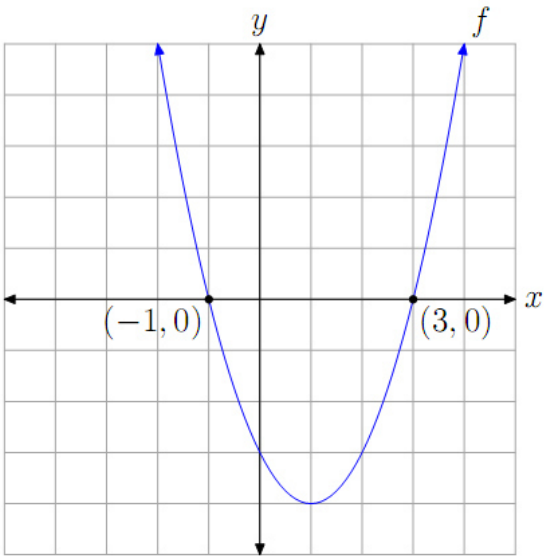
Tìm nghiệm tại lân cận của -2

```
x1 =
     -1
>>x2 = fzero(f, [2, 4])
```

Tìm nghiệm trong khoảng (2, 4)

Lưu ý f(2) và f(4) khác dấu

```
x2 =
     3
>>f(x1)
ans =
     0
>>f(x2)
ans =
     0
```



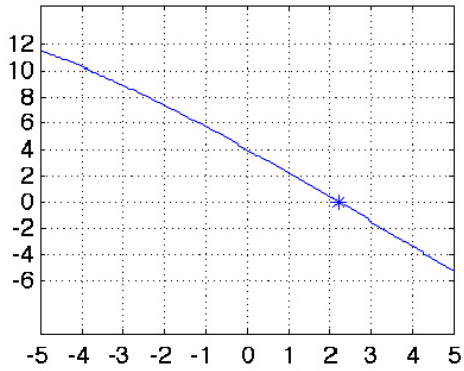
Hình 3. 1. Xác định nghiệm của hàm số tại điểm đồ thị giao với trục hoành

Ví dụ: Sử dụng MATLAB để giải phương trình sau $5 - 2x = e^{-0.25x}$

Trước tiên thực hiện phép biến đổi để có vế phải bằng không, sau đó khai báo hàm trong MATLAB

```
>>f = @(x) 5 - 2*x - exp(-0.25*x);
```

Ta thấy đồ thị hàm số cắt trục hoành trong khoảng [2, 3]



Hình 3. 2. Đồ thị hàm số f(x) = 5 - 2x - $e^{0.25x}$

```
>>x0 = fzero(f, [2, 3])
x0 =
     2.2124
>>f (x0)
ans =
     2.2204e-016
```

Lưu ý giá trị hàm tại x_0 xấp xỉ 2.22 × 10^{-16} tương đương không, vì vậy x =*2.2124* chính là nghiệm của hàm $5 - 2x - e^{-0.25x}$ cũng chính là nghiệm của phương trình $5 - 2x = e^{-0.25x}$

Lưu ý MATLAB định nghĩa nghiệm của phương trình $f(x) = 0$ tại nơi đồ thị hàm số $f(x)$ cắt trục hoành (nơi hàm số đổi dấu). Vì vậy trong trường hợp $f(x) = x^2$đạt giá trị không tại $x=0$, ***fzero*** sẽ không tìm được nghiệm nếu xác định lân cận không đúng. Ví dụ >>fzero(f, 1) hoặc fzero(f, -1) hay fzero(f, [-1, 1]).

3.4 Đa thức một biến số

MATLAB biểu diễn đa thức một biến số dưới dạng một vector hàng chứa các hệ số. Ví dụ: Đa thức $x^3 + 2x^2 - 4x + 3$ có thể biểu diễn bởi vector [1 2 -4 3], đa thức $2x^4 - x^2 + 5$ có thể biểu diễn bởi vector [2 0 -1 0 5], lưu ý hệ số không cho các số hạng x^3 và x.

Nếu v là một vector chứa các hệ số của đa thức và x là một số, hàm **polyval(v,x)** cho phép tính giá trị của đa thức biểu diễn bởi v tại x, x có thể là một vector, khi đó tập hợp các giá trị của đa thức cũng sẽ được lưu trong một vector, cú pháp gọi hàm như sau:

```
>>y = polyval (v, x)
```

Như đã trình bày ở trên, giá trị trả về y sẽ là giá trị của hàm số v tại x tương ứng.

Ví dụ:

```
>>polyval ([1   -2   0   1], [1   2   3   4])
ans  =
     [0   1   10   33]
```

Trong đó 0, 1, 10, 33 lần lượt là các giá trị của đa thức $x^3 - 2x^2 + 1$ tại các điểm $x = 1,\ 2,\ 3,\ 4$ tương ứng.

Hàm **roots** trong MATLAB cho phép tìm nghiệm của phương trình biểu diễn dưới dạng đa thức. Cú pháp gọi hàm như sau:

>>roots(C)

Trong đó *C* là vector chứa các hệ số của đa thức cần tìm nghiệm, giá trị trả về sẽ là tập hợp các nghiệm của đa thức cần tìm.

Ví dụ cho đa thức $f(x) = 4x^3 - 2x^2 - 8x + 3$, giải phương trình $f(x) = 0$.

```
>>Rs = roots([4   -2   -8   3])
Rs =
     -1.3660
      1.5000
      0.3660
```

Trường hợp đã biết tất cả các nghiệm của đa thức biểu diễn trong vector *Rs*, ta có thể biểu diễn lại đa thức đó dưới dạng vector của các hệ số sử dụng lệnh ***poly*** theo cú pháp sau:

>>f = poly(Rs)

Trong đó Rs là vector biểu diễn tập nghiệm của phương trình $f(x) = 0$ tương ứng, giá trị trả về của hàm sẽ là vector chứa các hệ số của $f(x)$.

Ví dụ: Biểu diễn lại đa thức có nghiệm chứa trong Rs được tính ở trên:

```
>>f = polys(Rs)
f =
     1.0000   -0.5000   -2.0000    0.7500
```

(Nhân tất cả các hệ số của đa thức với 4 để được đa thức ban đầu)

Trong MATLAB ta cũng có thể xác định tích của hai đa thức *a, b* bằng lệnh ***conv(a, b)*** với giá trị trả về là vector chứa các hệ số của đa thức tích. Tương tự phép chia hai đa thức cũng có thể thực hiện trong MATLAB sử dụng lệnh ***deconv(b, a)***. Trong đó, *b, a* lần lượt là các đa thức ở tử số và mẫu số. Cú pháp tương ứng như sau:

>>c = conv(a, b)

>>[q, r] = deconv(b, a)

Trong đó: *a, b* lần lượt là các đa thức thành phần, c là đa thức tích, *q* là thương của phép chia đa hai đa thức và *r* là phần dư trong phép chia đa thức *b* cho *a*.

Ví dụ: Xét hai đa thức

$A = 2x + 5$ $B = x^2 + 3x + 7$

```
>>A = [2   5];
>>B = [1   3   7];
```

Kết quả của phép tính *AB*

```
>>C = conv(A, B)
C =
     2    11    29    35
```

Kết quả của phép tính $\frac{C}{A}$

```
>>[D, r] = deconv(C, A)
D =
     1     3     7
r =
     0     0     0     0
```

3.5 Tích phân và đạo hàm

Để tính **tích phân** xác định, MATLAB có hàm ***quad***(fun, a, b) tính gần đúng tích phân của hàm *fun* trong khoảng *(a, b)* với sai số *1e-6* sử dụng công thức Simpson. Hàm *y = fun(x)* phải chấp nhận tham số *x* dưới dạng vector và trả về kết quả dưới dạng vector *y*. Vì thế khi định nghĩa hàm cần sử dụng các phép toán đối với vector **.* ./ .^**

Cú pháp gọi hàm như sau:

```
>>q = quad(fun, a, b)
```

Giá trị trả về của hàm *q* là kết quả tính gần đúng tích phân tương ứng của hàm *fun* như đã trình bày ở trên.

Ví dụ: Tính tích phân xác định của hàm $x^2 - 6x + 5$ trong khoảng *(1, 5)*.

```
>>f = @(x) x.^2 - 6*x + 5;
>>intf = quad(f, 1, 5)
```

```
intf =
     -10.6667
```

Ví dụ: Tính tích phân sau $\int_0^2 \frac{1}{x^3-2x-5}$

```
>>f = @(x) 1./(x.^3 - 2*x - 5);
>>quad(f, 0, 2)
ans =
     -0.4605
```

Đạo hàm của hàm số *y = f(x)* được viết dưới dạng $\frac{dy}{dx} f(x)$ hoặc *f'(x)* và định nghĩa bằng độ thay đổi của biến độc lập *y* theo *x*. Đạo hàm của hàm số tại một điểm chính là góc nghiêng của tiếp tuyến với hàm số tại điểm đó.

Để xác định đạo hàm của đa thức một biến trong MATLAB có thể sử dụng hàm **polyder** với cú pháp như sau:

>>polyder (P)

Trong đó *P* là vector chứa các hệ số của đa thức tương ứng.

Ví dụ: Cho đa thức $x^3 + 2x^2 - 4x + 3$ có thể biểu diễn bằng vector [1 2 -4 3], đạo hàm của đa thức này sẽ xác định như sau:

```
>>hs = [1   2   -4   3];
>>dhs = polyder(hs)
dhs =
     3   4   -4
```

có dạng vector [3 4 -4] biểu diễn cho đa thức $3x^2 + 4x - 4$.

Đạo hàm có thể biểu diễn dưới dạng giới hạn như sau và có thể tính gần đúng trong MATLAB sử dụng hàm **diff** theo cú pháp:

>>diff(X)

Trong đó *X* là vector chứa các hệ số của đa thức tương ứng.

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Xét hàm số $f(x) = x^2 - 6x + 5$. Giả sử chúng ta có tập hợp các giá trị của biến *x* biểu diễn bởi vector *x = [1 2 3 4 5]*. Như vậy tập hợp các giá trị của biến *y = f(x)* tương ứng là:

```
>>f = @(x) x.^2 - 6*x + 5;
>>x = [1   2   3   4   5];
```

```
>>y = f(x)
y =
     0    -3    -4    -3    0
```

Nếu coi *dy* là đạo hàm của *f(x)* theo *x* và xác định giá trị của đạo hàm tại các điểm *x = [1 2 3 4 5]* tương ứng:

```
>>dy = @(x) 2*x - 6
>>dy(x)
ans =
     -4    -2    0    2    4
```

So sánh giá trị đạo hàm của hàm số tại các điểm cho trước bằng cách tính gần đúng giới hạn sử dụng hàm **diff**.

```
>>diff(y)./diff(x)
ans =
     -3    -1  1    3
```

Lưu ý trong MATLAB việc tính **diff** của vector *a* gồm *n* phần tử được xác định bằng vector mới *da* có *n – 1* phần tử do đó số phần tử của phép tính đạo hàm theo **diff** ít hơn so với cách tính thông thường một phần tử. Giá trị chênh lệch giữa hai phép tính nhỏ dần khi khoảng cách giữa các phần tử trong vector *x* càng giảm (có thể thử lại khi nhập *x = 1: .1 : 5*).

$a = [a_1 \quad a_2 \quad a_3 \ldots a_{n-2} \quad a_{n-1} \quad a_n]$

$da = [a_2 - a_1 \quad a_3 - a_2 \ldots a_{n-1} - a_{n-2} \quad a_n - a_{n-1}]$

3.6 Phương trình vi phân

Trong MATLAB, phương trình vi phân có thể giải theo phương pháp số thông qua một số hàm xây dựng sẵn, trong nội dung giáo trình này chúng ta sẽ tập trung tìm hiểu hai hàm **ode23, ode45** là những phiên bản phát triển từ công thức Runge-Kutta bậc 2/3 và Runge-Kutta bậc 4/5.

Cú pháp tổng quát để giải phương trình vi phân với những hàm này như sau:

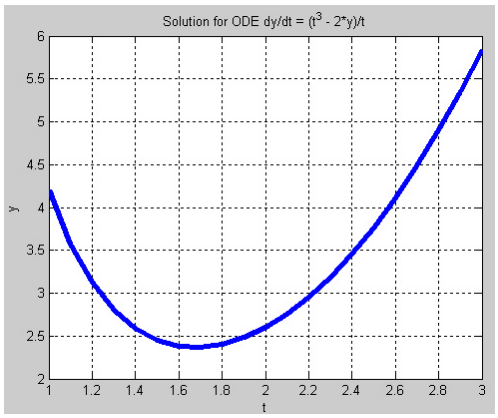
```
>>[t, y] = ode45('yprime', [t0, tF], y0)
>>[t,y ] = ode23('yprime', [t0, tF], y0)
```

Trong đó: 'yprime' là tên hàm biểu diễn phương trình vi phân, *t0* và *tF* là hai mốc thời gian đầu cuối cho đáp án, *y0* là giá trị của biến tại thời điểm ban đầu. Giá trị trả về của hàm là vector thời gian *t*, và ma trận *y* chứa những giá trị tương ứng của hàm tại các thời điểm khác nhau lấy từ vector thời gian *t*.

Đối với những hàm phức tạp hoặc đối với hệ phương trình vi phân, nên khai báo hàm trong script, trong chương này chúng ta sẽ chỉ xét những trường hợp có thể khai báo bằng hàm vô danh. Ví dụ: Giải phương trình vi phân

$$t\dot{y} + 2y = t^3 \rightarrow \frac{dy}{dt} = \frac{t^3 - 2y}{t}$$

trong khoảng $1 < t < 3$ và $y(0) = 4.2$



Hình 3. 3. Đồ thị (t, y) của phương trình vi phân dy/dt = (t^3 - 2y)/t

```
>>ode1 = @(t,y) (t^3 - 2*y)/t;
>>[t, y] = ode45(ode1, [1: 0.1: 3], 4.2);
```

Các cặp giá trị *(t,y)* thu được có thể biểu diễn bằng đồ thị 3.3.

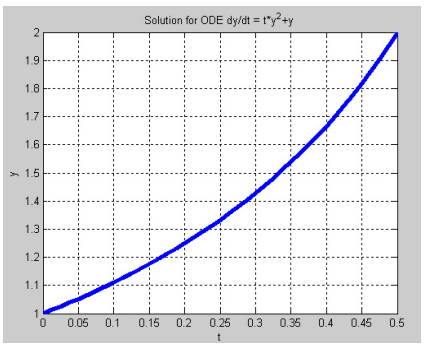
Giải phương trình vi phân sau:

$$\dot{y} = ty^2 + y$$

Trong khoảng $t \in [0,.5]$ với $y(0) = 1$

```
>>ode2 = @(t,y) (t*y^2 + y)
>>[t,y] = ode45(ode2, [0, .5], 1)
```

Các giá trị (t,y) thu được có thể biểu diễn dưới dạng đồ thị như sau:



Hình 3. 4. Đồ thị (t, y) của phương trình vi phân dy/dt = ty2 +y

Trên đây là các ví dụ về giải phương trình vi phân bậc nhất, đối với phương trình vi phân bậc cao hơn, cần đưa về hệ phương trình vi phân bậc nhất rồi giải, tuy nhiên khi đó khai báo hàm cần được thực hiện trong M-file.

3.7 Tóm tắt chương 3

Số phức	
abs(x)	Xác định giá trị tuyệt đối của biến x
angle(x)	Góc của số phức x
conj(x)	Trả về số phức liên hợp của x
imag(x)	Phần ảo của x
real(x)	Phần thực của x
Hàm số và đa thức một biến số	
fminbnd	Xác định cực tiểu của hàm số trong một khoảng cho trước
fzero	Nghiệm của hàm *f(x) = 0* xác định trong khoảng cho trước
polyval	Tính giá trị đa thức
roots	Tính nghiệm của đa thức
poly	Trả về vector hệ số của đa thức nếu biết vector nghiệm
conv	Thực hiện phép nhân hai đa thức
deconv	Thực hiện phép chia hai đa thức
Tích phân, đạo hàm, vi phân số	
quad	Tích phân hàm số
polyder	Đạo hàm đa thức một biến số
diff	Tính gần đúng đạo hàm của hàm số một biến số
ode23	Hàm tính vi phân theo công thức Runge-Kutta
ode45	Hàm tính vi phân theo công thức Runge-Kutta

Bảng 3. 1. Tóm tắt các hàm sử dụng trong chương 3

3.8 Bài tập chương 3

Bài 3.1 Thực hiện các phép tính sau

a. $(3+5i)+(2-3i)$
b. $(3+5i)+6$
c. $7i-(4+5i)$
d. $3(2+4i)$
e. $(5+3i)i$
f. $(2-7i)(3+4i)$
g. $\frac{1}{i}$
h. $\frac{3}{1+i}$
i. $\frac{4+7i}{2+5i}$

Đáp án:

a. $5+2i$
b. $9+5i$
c. $-4+2i$
d. $6+12i$
e. $-3+5i$
f. $34-13i$
g. $-i$
h. $1.5-1.5i$
i. $1.4828-0.2069i$

Bài 3.2 Rút gọn

a. $\frac{1+i}{1-i}-(1+2i)(2+2i)+\frac{3-i}{1+i}$
b. $2i(i-1)+(\overline{\sqrt{3}+i})^3+(1+i)\overline{(1+i)}$

Đáp án:

a. $3-7i$
b. $-10i$

Bài 3.3

a. Viết lại các số phức sau dưới dạng $[r,\theta]$
   i. $7+2i$

ii. $3-i$

iii. $-4+6i$

b. Viết lại các số phức sau dưới dạng $a+bi$

i. $[3,\frac{\pi}{4}]$

ii. $[5,\pi]$

iii. $[\sqrt{2},\frac{-2\pi}{3}]$

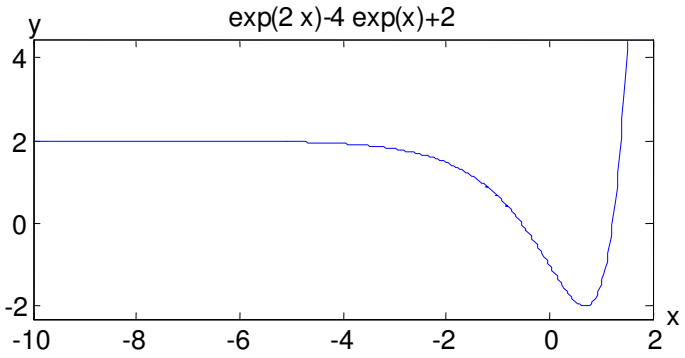
c. Viết lại các số phức sau dưới dạng lũy thừa.

i. $2(\cos\frac{\pi}{3}+i\cos\frac{\pi}{3})$

ii. $[5,\frac{2\pi}{3}]$

iii. $1-i\sqrt{3}$

Bài 3.4 Cho hàm số sau:



Hình 3. 5. Hình bài 3.4

$f(x)=e^{2x}-4e^{x}+2$

Tìm nghiệm *f(x) = 0* và cực trị của hàm trong khoảng biểu diễn của đồ thị.

Bài 3.5

Cho: $H(s)=\frac{n(s)}{d(s)}$

Với:

$n(s)=s^4+6s^3+5s^2+4s+3$

$d(s)=s^5+7s^4+6s^3+5s^2+4s+7$

Tìm:

a. *n(10), n(-5), n(-3), n(-1)*
b. *d(10), d(-5), d(-3), d(-1)*
c. *H(10), H(-5), H(-3), H(-1)*

Bài 3.6 Cho:

$$p_1(s) = s^3 + 5s^2 + 3s + 10$$

$$p_2(s) = s^4 + 7s^3 + 5s^2 + 8s + 15$$

$$p_3(s) = s^5 + 15s^4 + 10s^3 + 6s^2 + 3s + 9$$

a. Tìm: $p_1(2), p_2(2), p_3(3)$
b. Tìm: $p_1(s)p_2(s)p_3(s)$
c. Tìm: $p_1(s)p_2(s)/p_3(s)$

Bài 3.7 Tìm nghiệm $p(x) = 0$ của các đa thức sau:

a. $p_1(x) = x^7 + 8x^6 + 5x^5 + 4x^4 + 3x^3 + 2x^2 + x + 1$
b. $p_2(x) = x^5 - 13x^4 + 10x^3 + 12x^2 + 8x - 15$
c. $p_3(x) = x4 + 7x^3 + 12x^2 - 25x + 8$

Bài 3.8

Sử dụng hàm ***quad*** tính các tích phân sau:

a. $y_1 = \int_{\pi/2}^{\pi} \sin x dx$

b. $y_2 = \int_{\pi/2}^{\pi} \frac{\sin x}{x} dx$

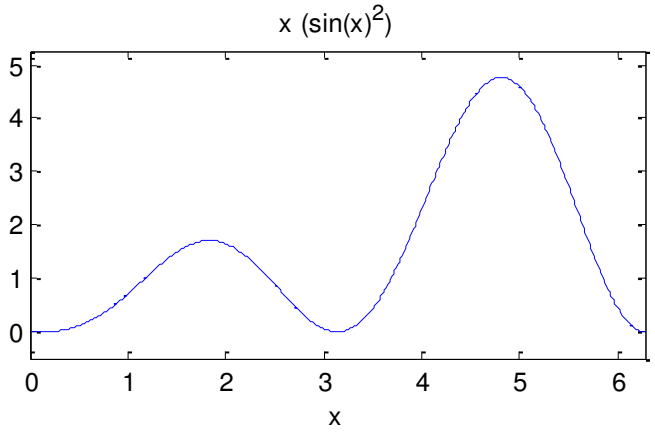
c. $y_3 = \int_{1}^{3} (x^3 + x^2 + \frac{1}{x^2}) dx$

Bài 3.9

Cho hàm số $f(x) = x\sin^2(x)$ và đồ thị biểu diễn trên khoảng $[0, 2\pi]$

a. Xác định các điểm cực đại của hàm số trên khoảng biểu diễn và giá trị hàm số tương ứng.
b. Tính tích phân sau sử dụng hàm ***quad***.

$$\int_0^{2\pi} f(x)dx$$



Hình 3. 6. Hình bài 3.9

Bài 3.10

Xét một vật chuyển động với vận tốc $v(t) = t^3 - 2t^2 + 2$ m/s xét trên khoảng thời gian $0(\text{s}) \le \text{t} \le 5(\text{s})$. Xác định gia tốc của vật ở thời điểm *t = 2.5 (s)*.

Đáp án:

Khai báo biến t chứa các giá trị thời gian cách nhau $\Delta t = 0.1(s)$, tính *v* theo *t*

```
>>t = 0: 0.1: 5;
>>v = t.^3 - 2*t.^2 + 2;
```

Tính đạo hàm theo *t* của *v*

```
>>dv = diff(v)./diff(t);
```

Gia tốc tại thời điểm *t = 2.5 (s)*

```
>>dv(26)
```

ans =

9.3100

Như vậy với $\Delta t = 0.1(s)$, gia tốc tại thời điểm *t = 2.5(s)* là 9.31 m/s^2

So sánh với giá trị gia tốc tính theo đạo hàm:

$$v'(t) = 3t^2 - 4t$$

$$v'(2.5) = 3 \cdot (2.5)2 - 4 \cdot 2.5 = 8.75 \text{ m/s}^2$$

Giá trị gia tốc tính bằng phương pháp sử dụng hàm ***diff*** chỉ đưa ra đánh giá gần đúng, để thu về kết quả chính xác hơn sinh viên có thể thử với $\Delta t = 0.01(s)$ và $\Delta t = 0.001(s)$

Bài 3.11

Sử dụng hàm ***ode23*** và ***ode45*** để giải các phương trình vi phân sau:

a. $\dot{x} = 1 - 2tx$ với *x(0) = 1* trên khoảng [0, 3];

b. $\dot{x} = \dfrac{tx}{1 + x^2}$ với *x(0) = 3* trên khoảng [0, 2];

Chương 4. BỘ CÔNG CỤ KÝ HIỆU

4.1 Giới thiệu biến ký hiệu

Bộ công cụ ký hiệu toán học là tập hợp các công cụ cho phép thao tác và xử lý các phép toán sử dụng biến ký hiệu. Bộ công cụ này có thể được ứng dụng trong đồ họa cũng như các phương pháp tính toán số trong môi trường MATLAB như: đạo hàm, tích phân, chuỗi Taylor, đại số tuyến tính, rút gọn, giải hệ phương trình, các hàm toán học đặc biệt, các phép biến đổi Fourier, Laplace…

Bộ công cụ biến ký hiệu trong MATLAB phát triển dựa trên cơ sở một phiên bản đặc biệt của Maple Kernel, tương thích với MATLAB 6 và Maple 8 trở lên.

Với sự xuất hiện của bộ công cụ biến ký hiệu, trong MATLAB xuất hiện thêm một kiểu dữ liệu gọi là *đối tượng ký hiệu*, được định nghĩa là một kiểu dữ liệu có cấu trúc. Trong đó, lưu một chuỗi ký tự đại diện cho một biến ký hiệu có thể là một biến đơn lẻ, một ma trận hoặc một biểu thức. Ví dụ sau đây chỉ ra sự khác biệt giữa hai loại kiểu dữ liệu *double* và đối tượng ký hiệu tương ứng, câu lệnh sau trả về kết quả là một số thập phân sử dụng dấu chấm động.

```
>>sqrt(2)
ans =
     1.4142
```

Mặt khác khi ta chuyển giá trị 2 sang dạng đối tương ký hiệu sử dụng lệnh **sym**sau đó thực hiện phép căn bậc hai, kết quả thu được sẽ là:

```
>>a = sqrt(sym(2))
ans =
     2^(1/2)
```

MATLAB trả về kết quả dạng biểu thức cho phép căn bậc hai mà không đưa ra giá trị số, kết quả này sẽ được lưu lại trong một xâu ký tự biểu diễn cho 2^(1/2). Giá trị số của biểu thức này có thể thu được sử dụng lệnh **double** theo cú pháp:

```
>>double (X)
```

Trong đó X là biến có kiểu cần chuyển về dạng *double*, nếu biến *X* đã có dạng *double*, lệnh trên không có tác dụng.

```
>>double(a)
ans =
     1.4142
```

Khi tạo một phân số có chứa đối tượng ký hiệu, MATLAB sẽ lưu lại tử số và mẫu số của phân số đó. Ví dụ:

```
>>sym(2)/sym(5)
ans =
     2/5
```

Kết quả thu được trong phép toán sử dụng đối tượng ký hiệu sẽ được biểu diễn dưới dạng biểu thức, khác với các kiểu dữ liệu tiêu chuẩn khác. Ví dụ: Trong phép cộng hai phân số có kiểu dữ liệu *double*, MATLAB trả về kết quả dạng số thập phân:

```
>>2/5 + 1/3
ans =
     0.7333
```

Trong khi đó nếu thực hiện phép cộng hai phân số có kiểu biến ký hiệu, MATLAB sẽ quy đồng mẫu số rồi thực hiện phép cộng, kết quả trả về sẽ có dạng phân số:

```
>>sym(2)/sym(5) + sym(1)/sym(3)
ans =
     11/15
```

4.2 Biến và biểu thức

Trong MATLAB, các biến và biểu thức ký hiệu có thể được tạo sử dụng lệnh **sym** theo cú pháp sau:

```
>>S = sym (A);
>>x = sym('x');
```

Trong đó các biến *S* và *x* được sử dụng để lưu trữ giá trị trả về của hàm *sym*, cú pháp thứ nhất cho phép tạo một biến có cấu trúc *sym* (cấu trúc ký hiệu) từ tham số đầu vào A, cú pháp thứ hai cho phép tạo biến ký hiệu có tên *'x'* và lưu kết quả vào *x*.

Ví dụ các lệnh:

```
>>x = sym('x')
>>a = sym('alpha')
```

cho phép tạo các biến ký hiệu mà khi in ra màn hình, chúng được biểu diễn tương ứng là *x* và *alpha*.

Trong toán học và nghệ thuật, hai đại lượng được coi là có tỉ số vàng nếu tỉ lệ giữa chúng tương đương với tỉ lệ giữa tổng của chúng với đại lượng lớn hơn. Ví dụ cho hai đại lượng *a* và *b* với *a* > *b*, hai đại lượng này được coi là có tỉ số vàng nếu: $\frac{a}{b}=\frac{a+b}{a}=\frac{1+\sqrt{5}}{2}$

Giả sử dùng biến ký hiệu để biểu diễn tỉ số vàng $\rho=\frac{1+\sqrt{5}}{2}$, ta có thể sử dụng lệnh:

```
>>rho = sym(' ( 1 + sqrt(5) )/2 ')
```

sau đó thực hiện các phép toán với biến ký hiệu *rho* vừa khai báo, ví dụ:

```
>>f = rho^2 - rho - 1
f =
     (1/2+1/2*5^(1/2))^2-3/2-1/2*5^(1/2)
```

Kết quả là một biểu thức mà ta có thể rút gọn được sử dụng lệnh ***simplify***.

```
>>simplify(f)
ans =
     0
```

Lệnh simplify cho phép rút gọn các phần tử có dạng ký hiệu, sử dụng theo cú pháp sau:

```
>>simplify(x)
```

Ví dụ: Nhập một hàm số bậc hai $f = ax^2 + bx + c$

```
>>f = sym('a*x^2 + b*x + c')
```

Lệnh trên sẽ gán cho biến *f* biểu thức ký hiệu $ax^2 + bx + c$. Tuy nhiên, trong trường hợp này bộ công cụ biến ký hiệu trong MATLAB không tạo các biến tương ứng với các thành phần *a, b, c, x* trong đa thức vừa nhập. Các biến, tham số trên cần được khai báo để có thể sử dụng:

```
>>a = sym('a');
>>b = sym('b');
>>c = sym('c');
>>x = sym('x');
```

Hoặc đơn giản hơn:

```
>>syms a b c x;
```

```
>>f = sym('a*x^2 + b*x + c');
```

Cũng giống như đối với đa thức biểu diễn bởi vector hàng hay hàm số vô danh, một đa thức được khai báo bởi lệnh **sym** có thể xác định giá trị của nó tại một điểm sử dụng lệnh **subs** theo cú pháp sau:

```
>>subs(S)
>>subs(S, new)
>>subs(S, old, new)
```

Trong đó *old* là các giá trị trong biểu thức *S* sẽ bị thay thế bởi các giá trị *new*. Trường hợp sử dụng cú pháp thứ nhất, tất cả các biến ký hiệu trong *S* sẽ bị thay thế bởi các biến trả về của các hàm được gọi hoặc các giá trị có sẵn trong Workspace. Trong với cú pháp thứ hai hoặc thứ ba, hoặc là tất cả các biến ký hiệu sẽ bị thay thế bởi *new* hoặc chỉ các biến ký hiệu được liệt kê trong *old* sẽ bị thay thế bởi *new* ở trong biểu thức *S*.

Ví dụ: Thay giá trị $x = 2$ vào biểu thức $f = 2x^2 - 3x + 1$

```
>>syms x f;
>>f = 2*x^2 - 3*x + 1;
>>f2 = subs(f,2)
f2 =
     3
```

Đối với đa thức có một biến số, vẫn có thể sử dụng lệnh **subs** để tính giá trị đa thức khi ta chỉ định rõ biến nào trong đa thức được thay thế bởi một giá trị cho trước.

```
>>syms x y ;
>>f = x^2*y + 5*x*sqrt(y);
```

thay giá trị x = 3 vào f

```
>>subs(f, x, 3)
ans =
     9*y + 15*y^(1/2)
```

thay giá trị y = 3 vào f

```
>>subs(f, y, 3)
ans =
     3*x^2 + 5*x*3^(1/2)
```

tính giá trị f tại x = 0 và y = 0

```
>>subs(f, {x, y}, {0,0})
ans =
```

```
0
```

Chú ý: Khi sử dụng hàm **subs** mà không chỉ rõ ra biến số ký hiệu nào sẽ được thay giá trị, MATLAB sẽ chọn **biến ký hiệu mặc định** theo quy tắc sau: đối với biến một ký tự, MATLAB sẽ chọn biến gần với x nhất trong bảng chữ cái, nếu có hai biến có cùng khoảng cách tới **x**, biến được chọn sẽ là biến xếp sau trong bảng chữ cái.

Bộ công cụ ký hiệu còn cho phép ta giải phương trình, hệ phương trình sử dụng hàm **solve** theo cú pháp sau:

>>solve(pt1, pt2, pt3, …, ptN)

>>solve(pt1, pt2, pt3,…, ptN, bien1, bien2, bien3, …, bienN)

Trong đó *pt1, pt2, pt3, …, ptN* là các phương trình cần giải còn *bien1, bien2, bien3, …, bienN* là các ẩn tương ứng.

Xét ví dụ đối với đa thức trong mục 3.4:

$$f(x) = 4x^3 - 2x^2 - 8x + 3$$

Đa thức này có thể được biểu diễn dưới dạng vector:

```
>>clear all
>>v = [4   -2   -8   3]
```

và có thể được biểu diễn dưới dạng hàm biến ký hiệu nhờ lệnh **poly2sym**:

```
>>f = poly2sym(v)
f =
     4*x^3 - 2*x^2 - 8*x + 3
```

Lưu ý trong trường hợp này, biến mới xuất hiện trong Workspace chỉ có *f* và *x* vẫn là biến chưa được khai báo.

```
>>x
??? Undefined function or variable 'x'
```

Ngược lại MATLAB cũng cho phép biến đa thức biến ký hiệu sang dạng vector hệ số với lệnh **sym2poly**.

Như đã biết *f(x) = 0* có 3 nghiệm và có thể giải bằng lệnh **solve** khi *f* được khai báo như một biến ký hiệu.

```
>>x=solve(f)
x =
     3/2
     1/2*3^(1/2)-1/2
     -1/2-1/2*3^(1/2)
```

Nghiệm của *f(x) = 0* ở đây được đưa ra dưới dạng biểu thức và có thể chuyển về giá trị số sử dụng lệnh **double** hoặc **eval**.

```
>>eval(x)
ans =
     1.5000
     0.3660
         -1.3660
```

Lưu ý: Trong trường hợp trên, nếu như tham số sử dụng trong lệnh **solve** là một đa thức, một hàm số chứ không ở dạng phương trình, **solve** sẽ cho đa thức bằng không rồi giải. Sau đây là ví dụ đối với trường hợp tham số đưa vào ở dạng phương trình.

```
>>syms y
>>solve('2*y^2 + y = 6')
ans =
     3/2
     -2
```

Trong trường hợp có nhiều hơn một biến trong một phương trình, cần chỉ rõ phương trình sẽ được giải theo biến nào. Trường hợp không chỉ rõ trong câu lệnh mà phương trình có biến *x* thì MATLAB sẽ luôn ưu tiên giải theo *x*.

Hai câu lệnh sau là ví dụ cho việc **solve** giải phương trình theo biến ưu tiên *x* và giải theo biến chỉ định:

```
>>solve (' a*x^2 + b*x ')
ans =
     0
          -b/a
>>solve('a*x^2 + b*x','b')
ans =
     -a*x
```

Lệnh **solve** cũng có thể sử dụng để giải hệ phương trình nhiều ẩn trong đó kết quả được lưu dưới dạng dữ liệu có cấu trúc. Ví dụ hệ phương trình dưới đây có thể giải như sau:

$$\begin{Bmatrix} 4x-2y+z=7 \\ x+y+5z=10 \\ -2x+3y-z=2 \end{Bmatrix}$$

```
>>solve('4*x - 2*y + z = 7', 'x + y + 5*z = 10', '-2*x +
3*y - z = 2')
ans =
     x: [1 x 1 sym]
     y: [1 x 1 sym]
     z: [1 x 1 sym]
```

Để truy xuất tới các nghiệm thành phần được lưu trong kiểu dữ liệu có cấu trúc, cần sử dụng dấu chấm "."

```
>> x = ans.x
x =
     124/41
>> y = ans.y
y =
     121/41
>> z = ans.z
z =
     33/41
```

4.3 Ma trận biến ký hiệu

MATLAB chấp nhận ma trận có thể có phần tử là biến ký hiệu, các phép toán đối với ma trận này hoàn toàn tương tự như các phép toán đối với ma trận thông thường. Ví dụ nhập ma trận $A=\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}$

```
>>syms a11 a12 a13 a21 a22 a23;
>>A = [a11 a12 a13; a21 a22 a23];
```

Một ma trận được coi là ma trận tuần hoàn nếu như mỗi hàng của ma trận chính là hàng trước đó của ma trận đó và phần tử đầu tiên của hàng được đặt ở vị trí cuối cùng. Ví dụ: Có thể tạo một ma trận tuần hoàn từ biến ký hiệu *a, b, c* như sau:

```
>>syms a b c
>>A = [a b c; b c a; c a b]
A =
     [a   b    c]
     [b   c    a]
```

```
     [c   a    b]
```

Đối với ma trận tuần hoàn, giá trị tổng các phần tử của mỗi hàng và cột là như nhau, ví dụ kiểm tra hàng một và cột hai của ma trận A:

```
>>sum(A(1,:) )
ans =
     a + b + c
>>sum( A(:,2) )
ans =
     a + b + c
```

Tạo thêm các biến ký hiệu *alpha, beta* và thay thế cho các phần tử trong A:

```
>>syms alpha beta
>>A(2,3) = beta;
>>subs(A,b,alpha);
>>A
A =
     [a        alpha          c]
     [alpha         c         beta]
     [c        a         alpha]
```

Ví dụ: Xét ma trận quay G với góc quay *t*

$$G = \begin{bmatrix} \cos(t) & \sin(t) \\ -\sin(t) & \cos(t) \end{bmatrix}$$

```
>>syms t;
>>G = [cos(t)   sin(t);   -sin(t)   cos(t)];
```

Sử dụng hai lần ma trận quay G cho phép quay mặt phẳng với một góc bằng hai lần góc *t* cho trước:

```
>>A = G*G
A =
     [ cos(t)^2-sin(t)^2,   2*cos(t)*sin(t)]
     [  -2*cos(t)*sin(t), cos(t)^2-sin(t)^2]
```

Cũng giống như tối giản biểu thức bằng lệnh **simplify**, ma trận cũng có thể được đưa về dạng đơn giản sử dụng lệnh **simple** theo cú pháp sau:

```
>>simple(S)
```

Trong đó S là một biến ký hiệu. Lệnh ***simple*** sẽ tìm dạng tối giản nhất của biểu thức hoặc ma trận biến ký hiệu.

Ví dụ:

```
>>simple(A)
ans =
     [  cos(2*t),  sin(2*t)]
     [ -sin(2*t),  cos(2*t)]
```

4.3 Tích phân và đạo hàm đa thức biến ký hiệu

Đối với đa thức biến ký hiệu, trong MATLAB ta có thể sử dụng hai hàm **int** và **diff** khi muốn tích phân hay đạo hàm những đa thức này. Chi tiết về hai lệnh này có thể tìm hiểu thêm trong MATLAB Help:

```
>>help sym/int
>>help sym/diff
```

Cú pháp:

```
>>int (S, v, a, b)
>> diff(S, 'v', n)
>> diff(S, n, 'v')
```

Trong đó ở cú pháp thứ nhất, MATLAB cho phép tính tích phân của *S* theo *v* trong khoảng từ *a* đến *b*. Ở cú pháp thứ hai và thứ ba, MATLAB cho phép người sử dụng tính đạo hàm *n* lần của *S* theo *v*

Ví dụ: Tính tích phân: $\int (3x^2 - 1)dx$

```
>>syms x;
>>int(3*x^2-1)
ans =
     x^3 - x
```

Xác định tích phân trên khi biết hai giới hạn *[a, b] = [2, 4]*

```
>>int(3*x^2 - 1, 2, 4)
ans =
     54
```

Ví dụ: Tính đạo hàm hàm số sau: $f = x^3 + 2x^2 - 4x + 3$

```
>>syms x f
>>f = x^3 + 2*x^2 - 4*x + 3
```

```
f =
     x^3 + 2*x^2 - 4*x + 3
>>diff(f)
ans =
     3*x^2 + 4*x - 4
```

Lưu ý: Khi muốn xác định đạo hàm của hằng số sử dụng lệnh **diff**, trước tiên cần định nghĩa hằng số đó kiểu biến ký hiệu. Ví dụ:

```
>>c = sym('5');
>>diff(c)
ans =
     0
```

Nếu chỉ nhập từ cửa sổ lệnh

```
>>diff(5)
```

Kết quả trả về sẽ có dạng:

```
ans =
     [ ]
```

Đối với đa thức có nhiều biến, cần chỉ rõ trong lệnh để xác định sẽ đạo hàm đa thức theo biến nào. Ví dụ:

```
>>syms s t
>>f = sin(s*t)
```

Đạo hàm theo *t*:

```
>>diff(f, t)
ans =
     cos(s*t)*s
```

Đạo hàm theo *s*:

```
>>diff(f, s)
ans =
     cos(s*t)*t
```

Hàm **diff** cũng có thể nhận giá trị đầu vào là ma trận biến ký hiệu, khi đó phép đạo hàm sẽ được thực hiện bằng cách đạo hàm từng phần tử của ma trận đầu vào. Ví dụ: Tìm đạo hàm của A.

$$A = \begin{bmatrix} \cos(\mathrm{ax}) & \sin(ax) \\ -\sin(ax) & \cos(ax) \end{bmatrix}$$

```
>>syms a x
>>A = [cos(a*x)      sin(a*x); -sin(a*x) cos(a*x)];
>>B = diff(A)
B =
     [ -sin(a*x)*a,  cos(a*x)*a]
     [ -cos(a*x)*a, -sin(a*x)*a]
```

4.4 Phương trình vi phân và hàm số biến ký hiệu

Trong MATLAB, phương trình vi phân của hàm số biến ký hiệu có thể được giải sử dụng hàm **dsolve**, trong đó ký tự *D* được sử dụng để biểu diễn các biến vi phân, ký hiệu *D2, D3,…, DN* tương ứng với bậc vi phân. Ví dụ *D2y* là biểu diễn của d^2y/dt^2. Điều kiện ban đầu nếu không khai báo trong phương trình, MATLAB sẽ đưa ra đáp án bao gồm các hằng *C1, C2*. Cú pháp gọi hàm như sau:

```
>>dsolve('eqn1', eqn2', ...)
```

Trong đó *eqn1, eqn2...* tương ứng là các phương trình biến ký hiệu thể hiện phương trình vi phân và các điều kiện ban đầu.

Ví dụ: Giải phương trình vi phân sau:

$$\dot{y} = 1 + y^2$$

trong đó *y* là biến phụ thuộc vào biến độc lập *t*

```
>>dsolve('Dy = 1 + y^2')
ans =
     tan( t + C1 )
```

Khi điều kiện ban đầu được khai báo trong phương trình, kết quả trả về sẽ là:

```
>>dsolve('Dy = 1 + y^2', 'y(0) = 1')
ans =
     tan (t + ¼*pi)
```

Lưu ý: Trong trường hợp này mặc dù *y* biểu diễn theo *t* và trong Workspace ta có thể quan sát thấy biến *y* nhưng không có biến *t*, vì thế hàm diff(y, t) sẽ trả về khai báo lỗi.

Ví dụ: Phương trình vi phân bậc hai với hai điều kiện ban đầu:

$$\frac{d^2y}{dt^2} = \cos(2t) - y \qquad y(0) = 1 \qquad \frac{dy}{dt}(0) = 0$$

```
>>y=dsolve('D2y = cos (2*t) - y', 'y(0) = 1', 'Dy(0) =
0', 't')
```

Không giống như khi sử dụng phương pháp số, phương trình vi phân bậc cao không cần phải đưa về hệ phương trình vi phân bậc nhất để giải, chỉ cần sử dụng ký hiệu phù hợp với các bậc vi phân tương ứng.

Ví dụ phương trình vi phân bậc ba:

$$\frac{d^3u}{dx^3} = u \qquad u(0) = 1 \qquad \frac{du}{dx}(0) = -1 \qquad \frac{d^2u}{dx^x}(0) = \pi$$

```
>>u = dsolve('D3u = u', 'u(0) = 1', 'Du(0) = -1',
'D2u(0) = pi', 'x')
```

Hàm **dsolve** còn cho phép giải hệ phương trình vi phân với một hoặc nhiều biến. Ví dụ: Cho hệ phương trình vi phân bậc nhất:

$$\dot{f} = 3f + 4g \qquad \dot{g} = -4f + 3g$$

```
>>S=dsolve('Df = 3*f + 4*g', 'Dg = -4*f + 3*g')
```

Kết quả trả về được lưu lại trong cấu trúc S, để truy xuất tới f và g có thể sử dụng lệnh:

```
>>f = S.f
f =
     exp(3*t)*(C1*sin(4*t)+C2*cos(4*t))
>>g = S.g
g =
     exp(3*t)*(C1*cos(4*t)-C2*sin(4*t))
```

4.5 Tóm tắt chương 4

Tóm tắt các hàm đối với biến ký hiệu	
sym	Khởi tạo một biến ký hiệu
syms	Tạo một hay nhiều biến ký hiệu
double	Chuyển biểu thức biến ký hiệu về dạng số
eval	Chuyển biểu thức biến ký hiệu về dạng số
numden	Đưa về dạng tử số và mẫu số của một đa thức ở dạng phân số
poly2sym	Chuyển đa thức dạng vector hệ số sang đa thức biến ký hiệu
sym2poly	Chuyển đa thức biến ký hiệu sang đa thức dạng vector hệ số

subs	Thay thế giá trị của biến trong biểu thức
simplify	Đơn giản hóa biểu thức
simple	Đơn giản hóa ma trận
diff	Đạo hàm đa thức
int	Tích phân đa thức
solve	Tìm nghiệm của phương trình, hệ phương trình
dsolve	Giải phương trình vi phân, hệ phương trình vi phân

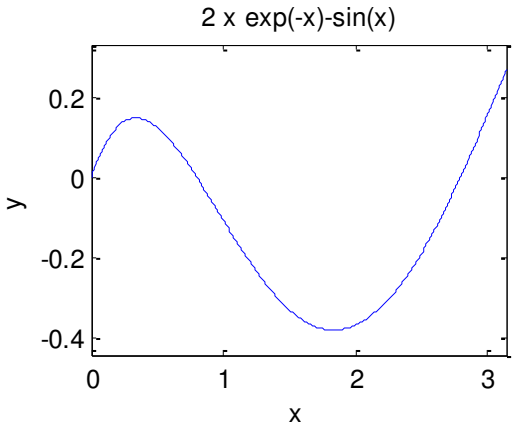
Bảng 4. 1. Tóm tắt các hàm sử dụng trong chương 4

4.6 Bài tập chương 4

Bài 4.1

Cho hàm số $f(x) = 2xe^{-x} - \sin(x)$ trên khoảng $[0, \pi]$, xác định nghiệm của phương trình $f(x) = 0$ và cực trị hàm số trên khoảng $[0, \pi]$

Đáp án



Hình 4. 1. Hình bài 4.1

Từ đồ thị có thể thấy phương trình $f(x) = 0$ có hai nghiệm và hàm số có hai cực trị trên $[0, \pi]$. Như đã biết ta có thể tìm nghiệm của phương trình và cực trị của hàm số sử dụng **fzero** và **fminbnd** đối với tham số hàm có dạng chuỗi ký tự.

```
>>syms x
>>y = 2*x*exp(-x) - sin(x)
```

Tìm nghiệm thứ nhất trên khoảng [0, 1]

```
>>x01 = fzero( char(y), [.5, 1])
```

```
x01 =
     0.8030
```

Tìm nghiệm thứ hai trên khoảng [2.5, 3]

```
>>x02 = fzero( char(y), [2.5, 3])
x02 =
     2.7923
```

Xác định cực tiểu của hàm số trên khoảng [1.5, 2]

```
>>xmin1 = fminbnd( char(y), [1.5, 2])
xmin1 =
     1.8409
```

Xác định cực đại của hàm số trên khoảng [0, .5]

```
>>xmax2 = fminbnd( char(-y), 0, .5)
xmax2 =
     0.3384
```

Bài 4.2

Cho đa thức $x^2y^3 + \sin(\pi xy)$, tính giá trị của đa thức khi thay $x = 2$ và $y = 3$ tương ứng.

Đáp án:

```
>>syms x  y
>>z = x^2*y^3 + sin(pi*x*y)
z =
     x^2*y^3 + sin(pi*x*y)
>>z23 = subs(z, {x, y}, {2, 3})
z23 =
     108
```

Bài 4.3

Kiểm tra kết quả của phép tính tích phân bất định sau bằng cách đạo hàm kết quả và so sánh với biểu thức trong dấu tích phân:

$$\int \frac{1}{1+x^4} dx$$

Đáp án:

```
>>syms x
```

```
>>y = int( 1/(1+x^4),x)
```

Kết quả hiển thị trên màn hình rất phức tạp và không thể dùng **simplify** để rút gọn. Để kiểm tra tính chính xác của kết quả trả về, ta sẽ đạo hàm *y* và so sánh với $\frac{1}{1+x^4}$

```
>>b = diff(y, x)
>>[t, n] = numden( b - 1/(1+x^4));
>>t
t =
      0
```

Tử số của phép tính $b-\frac{1}{1+x^4}$ nhận giá trị không chứng tỏ kết quả tích phân là đúng.

Bài 4.4

Tính các tích phân sau:

a. $\int_1^3 (\cos(x-1)+\frac{1}{x})dx$

b. $\int_0^\infty e^{-2x}\cos(3x)dx$

c. $\int_0^1 \log(x)dx$

d. $\int_0^4 \left|\left(x^2-2\right)^3\right|dx$

Đáp án:

a. 2.0079
b. 0.1538
c. -0.4343
d. 1.3461e+003

Lưu ý trong MATLAB, >>log(x) biểu diễn loga tự nhiên của *x*, muốn tính loga cơ số 10 của *x* cần nhập >>log10(x)

Bài 4.5

Giải thích ý nghĩa kết quả hiện trên màn hình khi thực hiện lệnh >>sqrt(x^2)

Đáp án:

```
>>sqrt(x^2)
ans =
      (x^2)^(1/2)
```

MATLAB sử dụng ký hiệu tiêu chuẩn để thể hiện các biểu thức biến ký hiệu, trong đó phép khai căn bậc hai được biểu diễn tương ứng với lũy thừa bậc (1/2).

Bài 4.6

Cho hàm số :

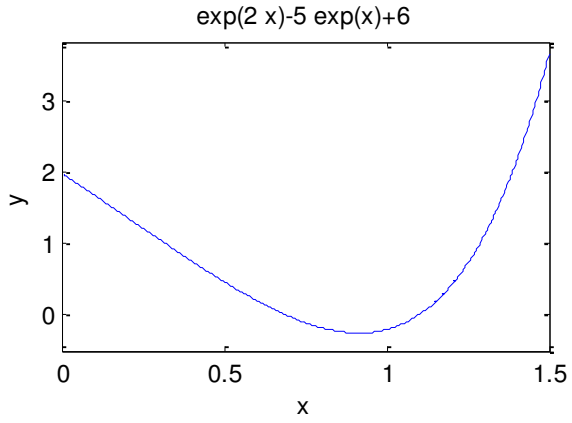
$$f(x) = \frac{\ln(1+x) + \frac{x^2}{2}}{(1+x)^2}$$

Tính $f'(x)$ và $f'(1)$

Đáp án:

```
>>syms x;
>>y = (log(1+x) + x^2/2)/((1+x)^2);
>>dy = diff(y);
>>dy1=subs(dy,1)
dy1 =
      0.0767
```

Bài 4.7



Hình 4. 2. Hình bài 4.7

$$f(x) = e^{2x} - 5e^{x} + 6$$

Có đồ thị trong khoảng [0, 1.5] như trong hình vẽ.

a. Tìm nghiệm *f(x) = 0*
b. Xác định cực tiểu của hàm số

Xác định giá trị hàm số và giá trị đạo hàm của hàm số tại điểm cực tiểu vừa tính.

Đáp án:

```
>>syms x;
>>y = exp(2*x) - 5*exp(x) + 6
```

Từ đồ thị có thể thấy phương trình *f(x) = 0* có hai nghiệm tại lân cận cực trị của hàm số, vì thế cực trị hàm số sẽ được tính trước.

```
>>xmin = fminbnd(char (y), .5, 1)
xmin =
     0.9163
```

Giá trị hàm số và giá trị đạo hàm của hàm số tại *xmin*.

```
>>ymin = subs(y, xmin)
ymin =
     -0.2500
>>dymin=subs(diff(y), xmin)
dymin =
     -6.5272e-006
```

Tìm hai nghiệm của phương trình *f(x) = 0*.

```
>>x1 = fzero(char(y), [xmin, xmin+1])
x1 =
     1.0986
>>x2 = fzero(char(y), [xmin-1, xmin])
x2 =
     0.6931
```

Hai nghiệm này cũng có thể xác định bằng cách sử dụng hàm **solve**.

```
>>nghiem = solve(y);
```

Bài 4.8

Cho hàm số:

$$f(x) = \sin\left(\frac{x^2 + 2x - 3}{x - 1}\right)$$

Tính đạo hàm cấp 10 và đưa kết quả về dạng tối giản.

Đáp án:

```
>>syms x;
```

```
>>y = sin((x^2 + 2*x - 3)/(x-1));
>>dy = diff(y,x,10);
>>dy = simplify(dy)
dy =
     -sin(x+3)
```

Bài 4.9

Cho:

$$f(x) = 3x^2 - 2x - 5$$

Trong MATLAB giải *f(x) = 0* theo ba cách khác nhau.

Bài 4.10

Cho hệ phương trình:

$$2x_1 + 2x_2 + x_3 = 2$$

$$x_2 + 2x_3 = 1$$

$$x_1 + x_2 + 3x_3 = 3$$

Giải hệ sử dụng bộ công cụ ký hiệu, kiểm tra kết quả bằng một phương pháp khác.

Bài 4.11

Cho hệ phương trình:

$$4x_1 - x_2 + 3x_4 = 10$$

$$-2x_1 + 3x_2 + x_3 - 5x_4 = -3$$

$$x_1 + x_2 - x_3 + 2x_4 = 2$$

$$3x_1 + 2x_2 - 4x_3 = 4$$

Giải hệ sử dụng bộ công cụ ký hiệu, kiểm tra kết quả bằng một phương pháp khác.

Bài 4.12

Cho hệ phương trình:

$$2(Va - Vb) + 5(Va - Vc) - e^{-t} = 0$$

$$2(Vb - Va) + 2Vb + 3(Vb - Vc) = 0$$

$$Vc = 2\sin(t)$$

Sử dụng bộ công cụ ký hiệu giải hệ, *Va, Vb, Vc* biểu diễn theo *t*.

Bài 4.13

Cho:

$$y = 2x^3 - x^2 + 4x - 5$$

Tính *y'(x)* và *y'(1)* theo hai cách khác nhau.

Bài 4.14

Giải hệ phương trình vi phân sau:

$\dot{x}_1 = 5x_1 - 2x_2$

$\dot{x}_2 = 7x_1 - 4x_2$

Với các điều kiện ban đầu:

$x_1(0) = 2$

$x_2(0) = 8$

Bài 4.15

Giải phương trình vi phân:

a. $\frac{d^2y}{dt} = -4y$ với các điều kiện ban đầu $y(0) = 1$ và $y'(\frac{\pi}{3}) = 0$

b. $\frac{d^2y}{dx^2} + 8\frac{dy}{dx} + 2y = \cos(x)$ với các điều kiện ban đầu $y(0) = 0$ và $y'(0) = 1$

Chương 5. MẢNG HỖN HỢP VÀ DỮ LIỆU CÓ CẤU TRÚC

Dữ liệu có cấu trúc (*data structures*) là các biến chứa nhiều hơn một giá trị. Có rất nhiều loại biến như thế này trong MATLAB mà vector và ma trận được giới thiệu ở chương 2 là ví dụ. Ta có thể thấy một biến vector hay một biến ma trận chứa rất nhiều giá trị liên quan một cách logic với nhau và có một điểm đặc biệt là tất cả các giá trị đó đều cùng một loại và trong một số trường hợp nào đó được sử dụng để biểu diễn cùng một đối tượng (ví dụ: ma trận chứa điểm của các môn học của một sinh viên thì các phần tử trong ma trận đều có thể có kiểu dữ liệu *double*…).

Một mảng hỗn hợp (*cell array*) cũng là một loại dữ liệu có cấu trúc, tuy nhiên có thể chứa các giá trị có các kiểu khác nhau. Mảng hỗn hợp có thể là vector hay ma trận. Một ví dụ điển hình của việc sử dụng mảng hỗn hợp là khi lưu trữ các chuỗi ký tự có độ dài khác nhau, mảng hỗn hợp cũng có thể được sử dụng để lưu các biến con trỏ trỏ tới các dữ liệu được lưu trữ khác.

Một trường hợp khác của dữ liệu có cấu trúc đó là các biến cấu trúc (*structures*) lưu giữ các giá trị có quan hệ logic với nhau, tuy nhiên các giá trị này không nhất thiết phải có cùng kiểu cũng như cùng sử dụng để biểu diễn một đối tượng. Các giá trị khác nhau trong biến cấu trúc được lưu giữ trong các trường (*fields*).

Một điển hình về việc sử dụng biến cấu trúc là khi xây dựng cơ sở dữ liệu sử dụng trong quản lý. Ví dụ một giáo viên muốn quản lý tất cả các sinh viên trong lớp học của mình theo tên, mã số sinh viên và các điểm thành phần của môn học. Trong trường hợp này, có thể sử dụng MATLAB để lưu giữ các dữ liệu trên bằng việc dùng biến cấu trúc.

Mảng hỗn hợp và biến cấu trúc đều có thể được sử dụng để lưu giữ các giá trị có kiểu khác nhau trong một biến đơn lẻ. Điểm khác biệt lớn nhất giữa mảng hỗn hợp và biến cấu trúc là mảng hỗn hợp quản lý giống như vector và ma trận, nghĩa là có thể truy xuất tới các phần tử trong mảng bằng cách sử dụng các hệ số vị trí cũng như có thể sử dụng vòng lặp để duyệt mảng còn biến cấu trúc thì không. Đối với biến cấu trúc, người sử dụng có thể duyệt và truy xuất tới các thành phần dữ liệu trong biến sử dụng tên của các trường thành phần tương ứng.

5.1 Mảng hỗn hợp

Mảng hỗn hợp là một yếu tố đặc biệt trong MATLAB mà không phải ngôn ngữ lập trình thông dụng nào cũng có. Như đã trình bày ở trên, mảng hỗn hợp được định nghĩa là một mảng, nghĩa là có thể truy xuất tới các phần tử thành phần sử dụng tham số vị trí, tuy nhiên các phần tử trong mảng lại có thể có các kiểu dữ liệu khác nhau.

5.1.1 Thiết lập mảng hỗn hợp

Có rất nhiều cách để thiết lập một mảng hỗn hợp trong MATLAB. Ví dụ ta sẽ tạo một mảng hỗn hợp gồm 4 phần tử, trong đó một phần tử chứa giá trị là một số kiểu *integer*, một phần tử chứa một ký tự, một phần tử chứa một vector, và một phần tử chứa một chuỗi ký tự. Như vậy mảng hỗn hợp mà ta định tạo có thể là một vector hàng hoặc vector cột gồm 4 phần tử, hoặc cũng có thể là một ma trận 2 hàng 2 cột. Cú pháp thiết lập vector hàng, cột hay ma trận cũng hoàn toàn tương tự như ở chương 2, nghĩa là các giá trị trong cùng một hàng có thể được phân tách với nhau bởi ký tự trống hoặc dấu phẩy, các hàng phân biệt với nhau bởi dấu chấm phẩy. Tuy nhiên thay vì sử dụng ngoặc vuông, mảng hỗn hợp sẽ được thiết lập bằng cách sử dụng ngoặc nhọn **{ }**.

Mảng hỗn hợp là một vector hàng

```
>>cellrowvec = {23, 'a', 1:2:9, 'hello'}
cellrowvec =
     [23] 'a'  [1x5 double]        'hello'
```

Mảng hỗn hợp là một vector cột

```
>>cellcolvec = {23; 'a'; 1:2:9; 'hello'}
cellcolvec =
     [23]
     'a'
     [1x5 double]
     'hello'
```

Mảng hỗn hợp là một ma trận 2x2

```
>>cellmat = {23     'a'; 1:2:9     'hello'}
cellmat  =
     [23       ]    'a'
     [1x5 double    ]    'hello'
```

Một cách khác để thiết lập mảng hỗn hợp là cấp giá trị cho từng phần tử riêng biệt và mở rộng mảng theo từng phần tử một. Tuy nhiên đây là cách làm không hiệu quả và khá tốn thời gian. Trong trường hợp chúng ta đã biết trước kích cỡ của mảng cần tạo, ta có thể sử dụng hàm ***cell***. Ví dụ sử dụng hàm ***cell*** để tạo một mảng hỗn hợp có dạng là một ma trận 2x2 như sau:

```
>>mycellmat = cell(2,2)
mycellmat =
     [ ]   [ ]
     [ ]   [ ]
```

5.1.2 Truy xuất và hiển thị thuộc tính của các phần tử mảng hỗn hợp

Cũng giống như khi làm việc với vector hoặc ma trận, ta có thể truy xuất tới các phần tử trong mảng hỗn hợp bằng cách sử dụng hệ số vị trí. Tuy nhiên trong mảng hỗn hợp, ta có thể truy xuất ra kiểu dữ liệu hoặc giá trị của phần tử mảng tương ứng. Khi truy xuất tới các phần tử trong mảng, ta có thể sử dụng dấu ngoặc tròn **( )** để chỉ truy xuất tới phần tử mà không thể hiện nội dung, giá trị của phần tử đó (truy xuất vị trí) hoặc ngoặc nhọn **{ }** để truy xuất tới giá trị của phần tử đó (truy xuất nội dung).

Ví dụ:

```
>>cellrowvec{2}
ans =
     a
>>cellmat{1,1}
ans =
     23
```

Bằng cách truy xuất này, có thể gán giá trị cho các phần tử tương ứng trong mảng hỗn hợp.

Ví dụ:

```
>> mycellmat{1,1} = 23
mycellmat =
     [23]      [ ]
      [   ]          [ ]
```

Khi sử dụng truy xuất hệ số vị trí bằng cách dùng dấu ngoặc tròn, giá trị của phần tử tương ứng trong mảng hỗn hợp sẽ không được thể hiện nếu phần tử đó là biến có cấu trúc.

Ví dụ:

```
>>cellcolvec (2)
ans =
     'a'
>>cellmat (2,1)
ans =
     [1 x 5 double]
>>cellmat {2,1}
ans =
     1    3    5    7    9
```

Do phần tử truy xuất tới là một vector, cũng có thể sử dụng kết hợp ngoặc nhọn và ngoặc tròn để truy xuất tới phần tử nằm bên trong một phần tử của mảng hỗn hợp như sau:

```
>>cellmat{2,1}(4)
ans =
     7
```

Tương tự như đối với ma trận, có thể truy xuất tới một tập hợp các phần tử trong mảng hỗn hợp như sau:

```
>>cellcolvec {2:3}
ans =
     a
ans =
     1    3    5    7    9
```

Lưu ý: Nếu không gán giá trị trả về của lệnh trên cho một biến bất kỳ, MATLAB sẽ sử dụng biến mặc định ans, như vậy sẽ chỉ có giá trị của cellcolvec{3} được lưu lại. Để tránh trường hợp này, có thể dùng một vector hoặc một mảng hỗn hợp mới để lưu giá trị trả về như sau:

```
>>[c1, c2] = cellcolvec{2:3}
c1 =
     a
c2 =
     1    3    5    7    9
>>cellcolvec (2:3)
ans =
```

```
    'a'
    [1x5 double]
```

Để hiển thị các mảng hỗn hợp, ta có thể sử dụng lệnh ***celldisp*** để hiển thị nội dung, giá trị của tất cả các phần tử hoặc sử dụng lệnh ***cellplot*** để hiển thị lại các phần tử trong mảng dưới dạng đồ thị (tuy nhiên không hiển thị nội dung).

Ví dụ:

```
>>celldisp(cellrowvec)
cellrowvec{1} =
    23
cellrowvec{2} =
    a
cellrowvec{3} =
    1    3    5    7    9
cellrowvec{4} =
    hello
```

Bên cạnh đó, có rất nhiều hàm áp dụng cho vector, ma trận cũng có thể được sử dụng trên mảng hỗn hợp, ví dụ một số hàm về kích thước và chiều như sau:

```
>>length(cellrowvec)
ans =
    4
>>size(cellrowvec)
ans =
    4    1
>>cellrowvec{end}
ans =
    hello
```

Ta cũng có thể xóa một phần tử trong mảng hỗn hợp hoặc xóa các hàng, cột thành phần của mảng hỗn hợp dưới dạng ma trận sử dụng tham số vị trí như sau:

```
>>cellrowvec
cellrowvec =
    [23] 'a'  [1x5 double]        'hello'
>>length(cellrowvec)
ans =
```

```
     4
>>cellrowvec(2) = [ ]
cellrowvec =
     [23]      [1x5 double]        'hello'
>>length(cellrowvec)
ans =
     3
>>cellmat
cellmat =
     [             23]        'a'
     [    1x5 double]         'hello'
>>cellmat(1,:) = [ ]
cellmat =
     [1x5 double]        'hello'
```

5.1.3 Lưu chuỗi ký tự trong mảng hỗn hợp

Một tiện ích của việc sử dụng mảng hỗn hợp đó là cho phép người sử dụng lưu các chuỗi ký tự với độ dài khác nhau. Ta cũng có thể sử dụng vòng lặp ***for*** để lần lượt duyệt tới các ký tự trong chuỗi.

Ví dụ:

```
>>names = {'Sue', 'Cathy', 'Xavier'}
names =
     'Sue'      'Cathy'    'Xavier'
>>for i = 1 : length(names)
disp (length(names{i}))
end
     3
     5
     6
```

MATLAB cũng cho phép người sử dụng chuyển từ mảng hỗn hợp chứa các ký tự sang ma trận chứa các biến kiểu ***char*** và ngược lại. Tuy nhiên, khi sử dụng hàm ***char*** để tạo ma trận chứa các chuỗi ký tự có độ dài khác nhau, MATLAB tự động thêm vào các ký tự trống để đảm bảo kích cỡ của ma trận, khi chuyển sang mảng hỗn hợp sử dụng lệnh ***cellstr***, các ký tự trống này sẽ bị loại bỏ. Cú pháp gọi lệnh như sau:

```
>>C = cellstr(S)
```

Trong đó S là mảng ký tự. Giá trị trả về C sẽ bao gồm các hàng của mảng ký tự S được lưu vào các phần tử mảng hỗn hợp riêng biệt.

```
>>greetmat = char ('Hello', 'Goodbye');
>>cellgreets = cellstr (greetmat)
cellgreets =
     'Hello'
     'Goodbye'
>>length(cellgreets{1}) + length(cellgreets{2})
ans =
     12
>>whos greetmat
Name      Size      Byte Class          Attributes
greetmat  2x7       28   char
```

Bên cạnh đó, người sử dụng cũng có thể kiểm tra nếu toàn bộ phần tử của một mảng hỗn hợp là chuỗi ký tự hay không sử dụng hàm **iscellstr**, MATLAB trả về giá trị logic 1 nếu đúng và ngược lại.

```
>>iscellstr(names)
ans = 1
>>iscellstr(cellcolvec)
ans = 0
```

5.2 Biến có cấu trúc

Biến có cấu trúc cho phép lưu trữ dữ liệu ở các kiểu khác nhau có quan hệ logic với nhau vào các trường dữ liệu của biến. Ưu điểm của việc sử dụng biến cấu trúc đó là người sử dụng có thể quản lý các trường dữ liệu phụ thuộc theo tên, tuy nhiên các biến dữ liệu không phải là mảng do đó không thể sử dụng vòng lặp để duyệt các phần tử trong biến.

5.2.1 Thiết lập và hiệu chỉnh các biến cấu trúc

Các biến cấu trúc có thể thiết lập đơn giản bằng cách gán giá trị cho các trường thành phần trong biến hoặc sử dụng hàm ***struct***, cú pháp gọi hàm như sau:

```
>>S = struct('field1', val1, 'field2', val2, ...)
```

Trong đó *field1, field2,...* là tên các trường thành phần, *val1, val2, ...* là các giá trị tương ứng với các trường thành phần đó. Giá trị trả về *S* sẽ là một biến cấu trúc bao gồm các trường *field1, field2, ...* với các giá trị *val1, val2, ...* tương ứng.

Ví dụ: Thiết lập một danh sách các mặt hàng của một siêu thị máy tính, trong đó mỗi một mặt hàng có các thông tin như số sản phẩm, giá thành mua vào (USD), giá thành bán ra, mã hàng. Một biến cấu trúc chứa thông tin sản phẩm có dạng như sau có thể thành lập: biến có tên là *package* với các trường thành phần lần lượt là *item_no, cost, price, code*.

Package

item_no	cost	Price	Code
123	19.99	39.95	G

Ta có thể thiết lập biến cấu trúc sử dụng hàm **struct** như sau:

```
>>package = struct ('item_no', 123, 'cost', 19.99, 'price', 39.95, 'code', 'g')
package =
     item_no: 123
     cost: 19.99
     price: 39.95
     code: 'g'
```

Một cách khác để thiết lập biến cấu trúc đó là sử dụng dấu chấm và gán giá trị trực tiếp cho các trường thành phần trong biến, tuy nhiên cách này không hiệu quả như khi sử dụng hàm ***struct***.

```
>>package.item_no = 123;
>>package.cost = 19.99;
>>package.price = 39.95;
>>package.code = 'g';
```

Tương tự, cũng có thể bổ sung thêm trường thành phần vào trong biến cấu trúc bằng cách sử dụng lệnh gán, ta cũng có thể gán cả biến cấu trúc cho một biến khác như sau:

```
>>newpack = package;
>>newpack.item_no = 111;
>>newpack.price = 34.95
newpack =
     item_no: 111
```

```
        cost: 19.90
        price: 34.95
        code: 'g'
```

Biến cấu trúc có thể in ra toàn bộ hoặc chỉ in các trường thành phần bằng cách sử dụng hàm ***disp***, và chỉ có thể in ra các trường thành phần khi sử dụng hàm ***fprintf***.

```
>>disp(package)
        item_no: 123
        cost: 19.90
        price: 39.95
        code: 'g'
>>disp(package.cost)
        19.90
>>fprintf('%d %c \n', package.item_no, package.code)
        123 g
```

Ta cũng có thể sử dụng hàm ***rmfield*** để loại bỏ trường thành phần trong biến cấu trúc. Lưu ý là giá trị trả về của hàm này sẽ là một biến mới có trường thành phần đã bị loại bỏ chứ không có sự thay đổi đối với biến cũ, cú pháp gọi hàm như sau:

```
>>S = rmfield(S, 'field')
```

Trong đó *field* là trường muốn loại bỏ của biến cấu trúc *S*. Giá trị trả về là biến cấu trúc ban đầu đã loại bỏ trường *field* tương ứng.

```
>>rmfield (newpack, 'code')
ans =
        item_no: 111
        cost: 19.90
        price: 34.95
```

Để thay đổi giá trị của biến *newpack*, giá trị trả về của ***rmfield*** phải được gán luôn cho biến *newpack*.

```
>>newpack = rmfield (newpack, 'code')
newpack =
        item_no: 111
        cost: 19.90
        price: 34.95
```

Tương tự như với mảng hỗn hợp, MATLAB cũng cung cấp cho người sử dụng một số hàm liên quan để kiểm tra thuộc tính của các biến có cấu trúc. Hàm ***isstruct*** trả về giá trị logic 1 nếu biến số của hàm đúng là một biến có cấu trúc và ngược lại, hàm ***isfield*** trả về giá trị logic 1 nếu một tên trường là một trường thành phần trong một biến có cấu trúc và ngược lại.

```
>>isstruct (package)
ans =
     1
>>isfield (package, 'cost')
ans =
     1
```

Ta cũng có thể sử dụng hàm ***fieldnames*** để liệt kê các trường thành phần của một biến có cấu trúc như sau:

```
>>pack_fields = fieldnames (package)
pack_fields =
     'item_no'
     'cost'
     'price'
     'code'
```

Do tên của các trường thành phần trong biến cấu trúc thường có độ dài khác nhau nên giá trị trả về của hàm ***fieldnames*** sẽ là một mảng hỗn hợp có các phần tử là tên của các trường thành phần tương ứng.

5.2.2 Vector biến cấu trúc

Như đã trình bày trong ví dụ mục trước, khi quản lý các sản phẩm của một siêu thị máy tính, ta có thể sử dụng biến cấu trúc để lưu thông tin của mỗi sản phẩm. Để quản lý các sản phẩm khác nhau trong siêu thị này, ta có thể sử dụng vector biến cấu trúc như sau:

packages

	item_no	cost	price	code
1	123	19.99	39.95	g
2	456	5.99	49.99	l
3	587	11.11	33.33	w

Trong ví dụ này *packages* là một vector có 3 phần tử, mỗi phần tử là một biến cấu trúc chứa các trường tương ứng *item_no, cost, price, code*. Có nhiều cách để thiết lập một vector chứa các biến cấu trúc như thế này, trong đó đơn giản nhất là tạo phần tử đầu tiên của vector, rồi sau đó bổ sung thêm các phần tử vào trong vector 1x1 ban đầu như sau:

```
>>packages = struct ('item_no', 123, 'cost', 19.99,
'price', 39.95, 'code', 'g');

>>packages (2) = struct ('item_no', 456, 'cost', 5.99,
'price', 49.99, 'code', 'l');

>>packages (3) = struct ('item_no', 587, 'cost', 11.11,
'price', 33.33, 'code', 'w');
```

Lưu ý trong trường hợp biết trước số phần tử của vector, phương pháp hiệu quả nhất là thiết lập vector bằng cách nhập phần tử cuối cùng trước để MATLAB cấp phát bộ nhớ cho vector này, sau đó người sử dụng có thể bổ sung lần lượt các phần tử từ đầu như sau:

```
>>packages (3) = struct ('item_no', 587, 'cost', 11.11,
'price', 33.33, 'code', 'w');

>>packages (1) = struct ('item_no', 123, 'cost', 19.99,
'price', 39.95, 'code', 'g');

>>packages (2) = struct ('item_no', 456, 'cost', 5.99,
'price', 49.99, 'code', 'l');
```

Một phương pháp thông dụng khác trong MATLAB đó là thiết lập một phần tử biến cấu trúc rồi sau đó sử dụng hàm **repmat** để tái tạo vector với kích thước mong muốn, cú pháp gọi hàm như sau:

```
>>B = repmat(A, M, N)
```

Trong đó *B* là ma trận, vector có kích cỡ *M* x *N* mà các phần tử thành phần đều là *A*.

Ví dụ:

```
>>packages = repmat (struct ('item_no', 123, 'cost',
19.99, 'price', 39.95, 'code', 'g'), 1, 3);

>>packages (2) = struct ('item_no', 456, 'cost', 5.99,
'price', 49.99, 'code', 'l');

>>packages (3) = struct ('item_no', 587, 'cost', 11.11,
'price', 33.33, 'code', 'w');
```

Lưu ý khi gọi biến cấu trúc bằng cách nhập tên biến, MATLAB sẽ chỉ hiển thị kích thước của vector biến cấu trúc cũng như tên của các trường thành phần như ví dụ sau:

```
>>packages
```

```
packages =
     1 x 3 truct array with fields:
          item_no
          cost
          price
          code
```

Biến *packages* lúc này là một vector cấu trúc, nghĩa là mỗi phần tử là một biến cấu trúc. Để hiển thị, truy xuất phần tử trong vector cấu trúc, ta phải sử dụng tham số vị trí tương ứng của phần tử đó, ví dụ truy xuất tới phần tử thứ 2 trong vector cấu trúc trên:

```
>>packages (2)
ans =
     item_no: 456
     cost: 5.99
     price: 49.99
     code: 'l'
```

Để truy xuất tới trường thành phần của từng phần tử, ta có thể sử dụng kết hợp tham số vị trí tương ứng của phần tử đó và tên của trường cần truy xuất, phân cách bởi dấu chấm như sau:

```
>>packages(1).code
ans =
     g
```

Có thể thấy ở đây *packages* là một biến có cấu trúc bậc ba đơn giản, trong đó bậc cao nhất là vector có cấu trúc *packages*, mỗi một phần tử của vector này là một biến cấu trúc, cấp thấp nhất trong biến này là các trường thành phần. Ta có thể sử dụng vòng lặp ***for*** để hiển thị tất cả các phần tử trong biến vector *packages* như sau:

```
>>for i = 1: length (packages)
disp (packages (i))
end
     item_no: 123
     cost: 19.99
     price: 39.95
     code: 'g'

     item_no: 456
```

```
    cost: 5.99
    price: 49.99
    code: 'l'

    item_no: 587
    cost: 11.11
    price: 33.33
    code: 'w'
```

Để truy xuất tới các trường thành phần trong một vector biến cấu trúc, MATLAB có lợi thế hơn so với phần lớn các ngôn ngữ lập trình thông dụng khi không cần thiết phải sử dụng vòng lặp duyệt qua tất cả các phần tử khác trong vector cũng như sử dụng dấu chấm để truy xuất tới các trường thành phần. Tất cả các giá trị trong một trường của tất cả các phần tử trong một vector cấu trúc đều có thể truy xuất trong MATLAB với chỉ một câu lệnh. Tuy nhiên nếu không gán giá trị trả về cho một biến cụ thể, các giá trị này sẽ được ghi đè lên nhau trong biến *ans* và người sử dụng chỉ có thể truy xuất tới giá trị cuối cùng:

```
>>packages.cost
ans =
    19.99
ans =
    5.99
ans =
    11.11
```

Để khắc phục, ta có thể lưu các giá trị này trong một vector khác như sau:

```
>>pc = [packages.cost]
pc =
    19.99          5,99      11.11
```

Với phương pháp này, MATLAB cho phép áp dụng hàm lên tất cả các trường tương ứng trong cùng một vector cấu trúc. Ví dụ: Để tính tổng giá trị của tất cả các trường có tên *cost* ta có thể dùng hàm ***sum***.

```
>>sum([packages.cost])
ans =
    37.09
```

Đối với các vector cấu trúc, cả vector hoặc phần tử có cấu trúc của vector, hoặc một trường trong vector đều có thể sử dụng với vai trò là các tham số trong hàm.

Trên đây là ví dụ về vector chứa các biến có cấu trúc. Tiếp theo ta sẽ xét một trường hợp phức tạp hơn khi một vector có cấu trúc mà một trường thành phần lại là một vector. Ví dụ khi thực hiện quản lý dữ liệu về sinh viên trong một lớp học bao gồm các thông số như tên sinh viên (*name*), mã số sinh viên (*id_no*), điểm thành phần (*quiz*) trong đó trường điểm thành phần là một vector chứa 4 điểm thành phần như sau:

student

	name	id_no	quiz			
			1	2	3	4
1	C, Joe	999	10.0	9.5	0.0	10.0
2	Hernandez, Pete	784	10.0	10.0	9.0	10.0
3	Brownnose, Violet	332	7.5	6.0	8.5	7.5

Ta có thể định nghĩa biến cấu trúc *student* như sau:

```
>>student (3) = struct ('name', 'Brownnose, Violet', 'id_no', 332, 'quiz', [7.5  6.0  8.5   7.5]);
>>student (2) = struct ('name', 'Hernandez, Pete', 'id_no', 784, 'quiz', [10.0  10.0  9.0  10.0]);
>>student (1) = struct ('name', 'C, Joe', 'id_no', 999, 'quiz', [10.0  9.5  0.0  10.0]);
```

Sau khi thiết lập biến *student*, ta có thể truy xuất đến từng thành phần trong biến. MATLAB sẽ hiển thị thông tin về biến cũng như tên các trường thành phần trong biến như sau:

```
>>student
student =
1 x 3 struct array with fields :
    name
    id_no
    quiz
```

Ta có thể truy xuất tới các giá trị thành phần như sau:

```
>>student (1)
```

```
ans =
     name: 'C, Joe'
     id_no: 999
     quiz: [10 9.5  0    10]
>>student (1). quiz
ans =
     10   9.5  0    10
>>student (1). quiz (2)
ans =
     9.5
>>student (3). name (1)
ans =
     B
```

Với những biến có cấu trúc phức tạp, điều quan trọng nhất là người sử dụng phải nắm được các thành phần trong biến để xử lý. Ví dụ đối với biến *student* là một biến vector cấu trúc, trong đó *student (1)* là một phần tử của vector *student* và là một biến có cấu trúc với các trường *name, id_no, quiz,* trong đó *student (1). quiz* là một vector kiểu double, *student(1). quiz(2)* là một phần tử của vector *quiz, student(3). name (1)* là ký tự đầu tiên của tên sinh viên thứ ba.

Ta cũng có thể sử dụng vòng lặp để duyệt vector cấu trúc *student*, qua đó thực hiện nhiệm vụ sau: in ra tên và điểm trung bình của từng sinh viên.

```
>>for i = 1: length(student)
qsum = sum([student(i).quiz]);
no_quizzes = length(student(i).quiz);
ave = qsum/no_quizzes;
fprintf('%-20s %.1f\n', student(i).name, ave);
end
C, Joe                  7.4
Hernandez, Pete         9.8
Brownnose, Violet       7.4
```

5.2.3 Cấu trúc lồng nhau và vector biến cấu trúc lồng nhau

Cấu trúc lồng nhau là cấu trúc mà trong đó ít nhất một trường thành phần của nó cũng là một cấu trúc. Xét ví dụ dưới đây về một cấu trúc đoạn thẳng có thể bao hàm các trường chứa tọa độ của các điểm mút biểu diễn trong mặt phẳng xOy.

Cấu trúc *lineseg* bao gồm hai trường *endpoint1* và *endpoint2*, trong đó mỗi trường thành phần này lại là một cấu trúc 2 trường chứa tọa độ *x* và *y*.

Lineseg

endpoint1		endpoint2	
x	Y	x	y
2	4	1	6

Phương pháp đơn giản và hiệu quả nhất để thiết lập biến *lineseg* trong MATLAB là sử dụng hàm **struct** khai báo trực tiếp như sau:

```
>>lineseg = struct('endpoint1', struct('x', 2, 'y', 4), 'endpoint2', struct('x',1,'y',6));
```

Ta cũng có thể khai báo từng bước để thiết lập biến *lineseg* bằng cách khai báo hai trường dạng cấu trúc *endpoint1* và *endpoint2* trước như sau:

```
>>pointone = struct('x', 2, 'y', 4);
>>pointtwo = struct('x', 1, 'y', 6);
>>lineseg = struct ('endpoin1', pointone, 'endpoint2', pointtwo);
```

Cách kém hiệu quả nhất trong MATLAB để khai báo các cấu trúc lồng nhau đó là khai báo theo từng trường thành phần một.

```
>>lineseg.endpoint1.x = 2;
>>lineseg.ednpoint1.y = 4;
>>lineseg.endpoint2.x = 1;
>>lineseg.ednpoint2.y = 6;
```

Khi biến cấu trúc lồng đã được thiết lập, việc truy xuất tới các thành phần trong biến hoàn toàn tương tự như khi truy xuất tới các biến cấu trúc thông thường.

```
>>lineseg
lineseg =
     endpoint1: [1 x 1 struct]
```

```
        endpoint2: [1 x 1 struct]
>>lineseg.endpoint1
ans =
        x: 2
        y: 4
>>lineseg.endpoint1.x
ans =
        2
```

Kết hợp vector và biến cấu trúc lồng nhau ta sẽ được một vector biến cấu trúc trong đó một số trường của các phần tử thành phần cũng là kiểu cấu trúc. Xét ví dụ sử dụng một vector cấu trúc lồng để biểu diễn các sản phẩm có dạng hình trụ tròn của một công ty với các kích thước, vật liệu, khối lượng khác nhau như sau:

Cyls

	code	dimensions		weight
		Rad	height	
1	x	3	6	7
2	a	4	2	5
3	c	3	6	9

Ta có thể thấy là *cyls* là một vector chứa ba phần tử biến cấu trúc với các trường *code, dimensions, weight* trong đó *dimensions* lại là một cấu trúc với hai trường thành phần *rad* và *height*. Biến *cyls* có thể được thiết lập như sau:

```
>>cyls(3) = struct ('code', 'c', 'dimension',
struct('rad', 3, 'height', 6), 'weight', 9);
>>cyls(1) = struct ('code', 'x', 'dimension',
struct('rad', 3, 'height', 6), 'weight', 7);
>> cyls(2) = struct ('code', 'a', 'dimension',
struct('rad', 4, 'height', 2), 'weight', 5);
```

Mặt khác, biến *cyls* cũng có thể được thiết lập bằng cách gán lần lượt giá trị cho các phần tử thành phần:

```
>> cyls(3).code = 'c';
>> cyls(3).dimensions.rad = 3;
>> cyls(3).dimensions.height = 6;
```

```
>> cyls(3).weight = 9;
>> cyls(1).code = 'x';
>> cyls(1).dimensions.rad = 3;
>> cyls(1).dimensions.height = 6;
>> cyls(1).weight = 7;
>> cyls(2).code = 'a';
>> cyls(2).dimensions.rad = 4;
>> cyls(2).dimensions.height = 2;
>> cyls(2).weight = 5;
```

Cần phân biệt rõ các lớp cụ thể trong biến *cyls* như sau:

- *cyls* là biến chính chứa toàn bộ thông tin liên quan đến sản phẩm được quan tâm, đồng thời là một vector cấu trúc.
- *cyls(1)* là phần tử thành phần trong vector *cyls*.
- *cyls(2).code* là một trong các trường thành phần trong cấu trúc và có dạng ký tự.
- *cyls(3).dimensions* là một trường thành phần trong cấu trúc nhưng bản thân nó cũng là một cấu trúc với hai trường thành phần *rad* và *height*.
- *cyls(1).dimensions.rad* là trường thành phần trong cấu trúc *dimensions*, giá trị được lưu ở dạng double.

5.3 Tóm tắt chương 5

Các hàm thông dụng đối với mảng hỗn hợp và biến cấu trúc	
cell	Thiết lập một mảng hỗn hợp
celldisp	Hiển thị nội dung các phần tử trong mảng hỗn hợp
cellplot	Hiển thị tên các phần tử trong mảng hỗn hợp dưới dạng đồ thị
cellstr	Chuyển mảng kiểu ký tự sang kiểu mảng hỗn hợp
iscellstr	Kiểm tra một mảng hỗn hợp chỉ chứa phần tử kiểu chuỗi ký tự
struct	Thiết lập biến cấu trúc
isstruct	Kiểm tra một biến có phải biến cấu trúc
fieldnames	Hiển thị tên các trường thành phần của một biến cấu trúc
repmat	Thiết lập vector biến cấu trúc với kích thước mong muốn

Bảng 5. 1. Tóm tắt các hàm sử dụng trong chương 5

5.3 Bài tập chương 5

Bài 5.1

Tạo một mảng hỗn hợp chứa các phần tử là các chuỗi ký tự như sau:

exclaimcell = {'Bravo', 'Fantastic job'};

In một phần tử ngẫu nhiên trong mảng.

Bài 5.2

Tạo mảng hỗn hợp sau:

>>ca = {'abc', 11, 3:2:9, zeros(2)}

Tạo một mảng hỗn hợp mới có dạng ma trận 2x2 chứa các phần tử của mảng *ca* vừa tạo, viết lệnh truy xuất tới cột cuối cùng trong mảng mới.

Bài 5.3

Sử dụng hàm ***cell*** thiết lập mảng hỗn hợp 2x2 sau đó gán giá trị bất kỳ cho các phần tử thành phần. Chèn một hàng vào giữa để mảng mới tạo thành có dạng ma trận 3x2.

Bài 5.4

Tạo mảng hỗn hợp lưu trữ các thông tin sau của một sinh viên bất kỳ: tên sinh viên, mã số sinh viên, điểm trung bình chung và in thông tin này ra màn hình.

Bài 5.5

Yêu cầu tương tự như bài **5.4** nhưng lưu thông tin vào một biến cấu trúc sau đó in ra màn hình.

Bài 5.6

Cho một vector cấu trúc được định nghĩa như sau:

```
>>kit(2).sub.id = 123;
>>kit(2).sub.wt = 4.4;
>>kit(2).sub.code = 'a';
>>kit(2).name = 'xyz';
>>kit(2).lens = [4 7];
>>kit(1).name = 'rst';
>>kit(1).lens = 5:6;
>>kit(1).sub.id = 33;
>>kit(1).sub.wt = 11.11;
```

```
>>kit(1).sub.code = 'q';
```

Trong các câu lệnh sau, câu lệnh nào là hợp lệ và nêu giá trị trả về tương ứng; câu lệnh nào không hợp lệ, giải thích.

```
>>kit(1).sub
>>kit(2).lens(1)
>>kit(1).code
>>kit(2).sub.id == kit(1).sub.id
>>strfind(kit(1).name, 's')
```

Bài 5.7

Lập một vector cấu trúc có dạng như dưới đây với thông tin của hai phần tử còn lại trong vector sinh viên tự khai báo.

```
>>subjects
subjects =
     1x3 struct array with fields:
          name
          sub_id
          height
          weight
>>subjects(1)
ans =
     name: 'Joey'
     sub_id: 111
     height: 6.7
     weight: 222.2
```

Giả sử không biết trước số phần tử của vector cấu trúc *subjects*, viết một đoạn chương trình tìm phần tử mà giá trị của các trường *height* hoặc *weight* nhỏ hơn giá trị trung bình của trường *height* hoặc *weight* của tất cả các phần tử trong vector, in ra màn hình thông tin phần tử tìm được.

Bài 5.8

Cho một file dữ liệu có dạng sau:

44	7	7.5	8
33	5.5	6	6.5

37	8	8	8
24	6	7	8

Sử dụng hàm ***load*** để tải file vào một ma trận trong MATLAB, từ ma trận vừa tạo, viết đoạn chương trình để tạo vector cấu trúc như sau:

Students

	id_no	Quiz		
		1	2	3
1	44	7	7.5	8
2	33	5.5	6	6.5
3	37	8	8	8
4	24	6	7	8

Nói cách khác, *students* là vector cấu trúc có 4 phần tử chứa thông tin về mã số sinh viên *id_no* và điểm thành phần môn học *quiz* của 4 sinh viên tương ứng, trong đó mã số sinh viên được biểu diễn trong cột đầu tiên, điểm thành phần được biểu diễn trong 3 cột còn lại của file dữ liệu.

Viết đoạn chương trình in ra mã số sinh viên và điểm trung bình theo cột như sau:

44	7.5
33	6.0
37	8.0
24	7.0

Bài 5.9

Tạo một biến cấu trúc lồng để lưu giữ thông tin về một cá nhân bao gồm: tên, địa chỉ, số điện thoại trong đó các trường thành phần địa chỉ và số điện thoại có cấu trúc riêng.

Bài 5.10

Tìm hiểu hai hàm xây dựng sẵn trong MATLAB là **cell2struct** (chuyển từ mảng hỗn hợp sang vector cấu trúc) và **struct2cell** (chuyển từ vector cấu trúc sang mảng hỗn hợp).

Chương 6. ĐỒ THỊ TRONG MATLAB

6.1 Đồ thị trong không gian hai chiều

6.1.1 Các lệnh cơ bản

MATLAB cung cấp cho người dùng thư viện gồm nhiều hàm xây dựng sẵn để thiết lập và hiệu chỉnh đồ họa trong không gian hai chiều cũng như ba chiều. Danh sách những câu lệnh cơ bản để làm việc với chế độ đồ họa trong MATLAB có thể tham khảo với lệnh **help** trong cửa sổ lệnh.

Các lệnh đồ họa 2D cơ bản

```
>>help graph2d
```

Các lệnh đồ họa 3D cơ bản

```
>>help graph3d
```

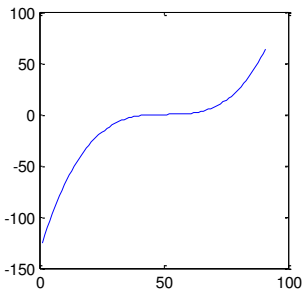
Bước quan trọng đầu tiên khi làm việc với chế độ đồ họa trong MATLAB đó là chuẩn bị nguồn dữ liệu. Dữ liệu nguồn ở đây có thể được thiết lập đơn giản bằng cách nhập từ bàn phím, sử dụng giá trị trả về của các hàm, hoặc được thiết lập bằng cách tải từ một hay nhiều file.

Lệnh đồ họa 2D cơ bản nhất trong MATLAB là **plot**, hàm cho phép thể hiện dưới dạng đồ thị của một hay nhiều tập dữ liệu. Ví dụ cho y là một vector bao gồm các phần tử là lũy thừa bậc 3 của các số cách đều 0.1 từ -5 đến 4:

```
>>y = (-5 : 0.1 : 4).^3;
```

Các phần tử của y sẽ được biểu diễn dưới dạng đồ thị như sau:

```
>>plot(y)
```



Hình 6. 1. Ví dụ lệnh plot

Lưu ý rằng trong đồ thị trên giá trị tương ứng trên trục hoành chính là số thứ tự của các phần tử trong vector *y*. Đồ thị trên hoàn toàn tương đương với đồ thị có hai tham số đầu vào *x* và *y* trong đó *x* là các phần tử cách đều 1 nhận các giá trị từ 1 đến 91 (91 tương đương với số phần tử của *y*).

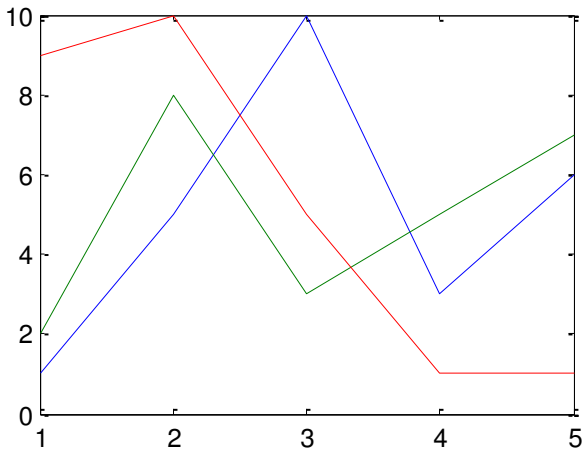
```
>>x = 1 : length(y);
>>plot (x, y)
```

Đối với lệnh plot chỉ có một tham số đầu vào ***plot(y)***, *y* có thể là vector hoặc ma trận, tùy vào dạng của tham số đầu vào mà đồ thị được thiết lập có sự khác biệt theo quy tắc sau:

- Nếu *y* là vector, đồ thị có dạng tập hợp của các điểm (x_i, y_i) trong đó x_i là số thứ tự của y_i trong vector *y*.
- Nếu *y* là ma trận *m* x *n*, đồ thị thu về là tập hợp *n* đường, trong đó mỗi đường là tập hợp các điểm (x_i, y_i) với $x_i = 1 \ldots m$, y_i các phần tử trong mỗi cột ứng với x_i tương ứng.

Ví dụ:

```
>>A = round(10*rand(5,3))
A =
     1     2     9
     5     8    10
    10     3     5
     3     5     1
     6     7     1
>>plot(A)
```



Hình 6. 2. Plot từ ma trận

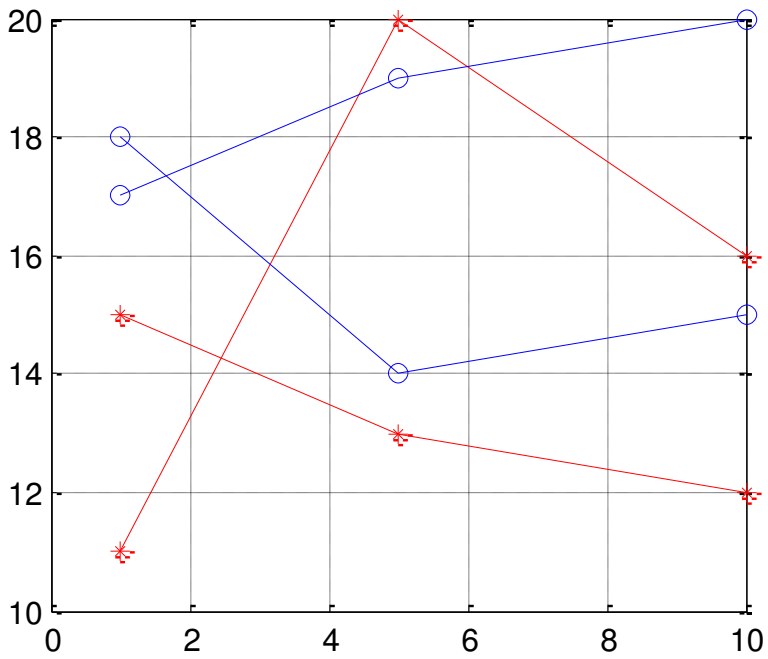
Đối với lệnh có hai tham số đầu vào ***plot(x, y)***:

- Nếu *x* và *y* là hai vector có cùng độ dài, MATLAB sẽ hiển thị đồ thị *y* theo *x*.
- Nếu *x* là vector và *y* là ma trận, đồ thị hiển thị hàng hoặc cột của *y* theo các phần tử của *x* tùy vào kích cỡ ma trận. Ví dụ cho vector *x*, ma trận *y1* và *y2*.

```
>>x = [1   5   10];
>>y1 = [11   20   16;   15   13   12];
>>y2 =  [18   17;   14   19;   15   20];
```

Do *y1* là ma trận có số cột bằng với chiều dài của *x*, các phần tử trên mỗi hàng của *y* sẽ được hiển thị theo các phần tử của *x*, số đường hiển thị bằng với số hàng của *y1*. Do ma trận *y2* có số hàng bằng với chiều dài của *x*, các phần tử trên mỗi cột của *y* sẽ được thể hiện theo các phần tử của *x*, số đường hiển thị bằng với số cột của *y2*.

```
>>plot(x, y1, '*-r');
>>hold on
>>plot(x, y2, 'o-b');
```



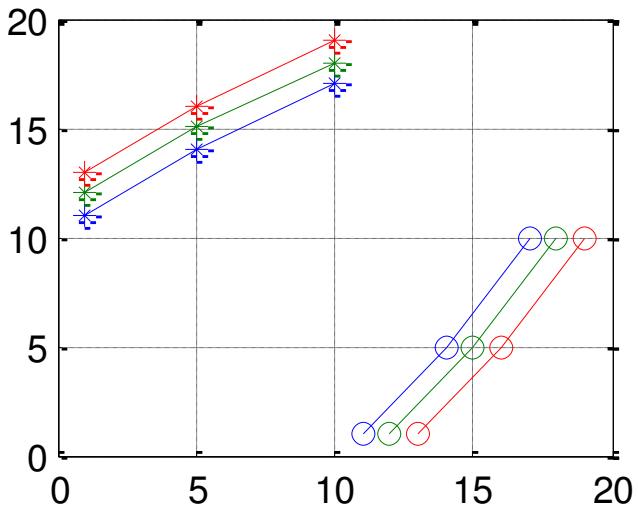
Hình 6. 3. Đồ thị (x, y1) và (x, y2)

- Nếu *x* là ma trận và *y* là vector, đồ thị sẽ hiển thị các phần tử của *y* theo các phần tử ở hàng hoặc cột của *x*.

- Nếu một trong hai tham số là ma trận vuông, đồ thị sẽ hiển thị các phần tử của tham số này theo các phần tử ở cột của tham số ma trận vuông.

Ví dụ:

```
>>y3=[11 12 13; 14 15 16; 17 18 19];
>>plot(x, y3, '*-')
>>hold on
>>plot(y3, x, 'o-')
```



Hình 6. 4. Đồ thị (x, y3) và (y3, x)

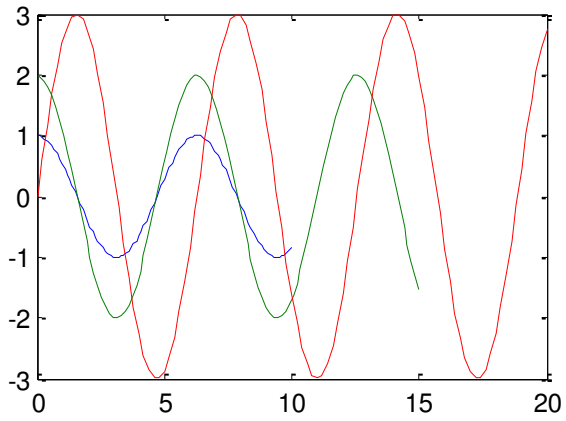
- Nếu x và y là hai ma trận có cùng kích cỡ, đồ thị sẽ hiển thị các phần tử ở từng cột của y theo các phần tử ở các cột của x tương ứng.

Hàm ***plot*** chấp nhận số tham số đầu vào có thể lớn hơn hai, tuy nhiên quy tắc nêu trên vẫn áp dụng cho các cặp tham số tương ứng. Ví dụ:

```
>>x1 = 0 : .1 : 10;
>>y1 = cos(x1);
>>x2 = 1.5*x1;
>>y2 = 2*cos(x2);
>>x3 = 2*x1;
>>y3 = 3*sin(x3);
>>plot(x1,y1,x2,y2,x3,y3);
```

Lệnh plot trên tương đương với:

```
>>X = [x1'   x2'   x3'];
>>Y = [y1'   y2'   y3'];
>>plot(X, Y)
```



Hình 6. 5. Plot giá trị 3 cặp vector trên cùng một đồ thị

Đồ thị của các hàm trên được thể hiện bằng các màu khác nhau, đó là do chế độ mặc định hiển thị màu trong MATLAB. Khi thể hiện nhiều đồ thị, các đường biểu diễn sẽ lần lượt được hiển thị với các màu tương ứng: xanh da trời (**b**lue), xanh lá cây (**g**reen), đỏ (**r**ed), lam (**c**yan), hồng thẫm (**m**agenta), vàng (**y**ellow) và đen (blac**k**). Quan sát trên đồ thị ta có thể thấy (x1, y1), (x2, y2), (x3, y3) được biểu diễn bằng các đường màu xanh da trời, xanh lá cây và đỏ tương ứng.

Màu		**Dạng ký hiệu**	
Ký tự	*Thể hiện*	*Ký hiệu*	*Thể hiện*
B	Xanh da trời	.	Điểm
G	Xanh lá cây	o	Hình tròn
R	Đỏ	x	Dấu x
C	Lam	+	Dấu +
M	Hồng thẫm	*	Dấu sao
Y	Vàng	s	Hình vuông
K	Đen	d	Kim cương
Dạng đường		v	Tam giác xuống
Ký tự	*Thể hiện*	^	Tam giác lên
-	Nét liền	<	Tam giác trái
:	Hai chấm	>	Tam giác phải
-.	Chấm gạch	p	Sao năm cánh
--	Nét đứt	h	Sao sáu cánh

Bảng 6. 1. Màu - dạng đường - dạng ký hiệu trong đồ thị

Chế độ đồ thị trong MATLAB cho phép hiển đường biểu diễn với các màu sắc, dạng đường, và ký hiệu khác nhau. Để thực hiện điều này, trong câu lệnh ***plot*** có thể thêm vào một chuỗi từ 1 đến 4 ký tự để đặt các thuộc tính **màu-dạng đường-dạng ký hiệu** cho đường biểu diễn. Các ký tự tượng trưng cho các thuộc tính của đường biểu diễn tóm tắt trong bảng trên.

Đối với ***plot***, MATLAB yêu cầu các tham số phải được khai báo và đồ thị sẽ được thể hiện tương ứng với dải giá trị của các tham số đầu vào. Bên cạnh đó, MATLAB cũng cung cấp cho người dùng một lệnh vẽ khác với nhiều tiện ích hơn, đó là ***ezplot***. Đối với ***ezplot***, khoảng biểu diễn của đồ thị mặc định sẽ là -2π đến 2π và có thể thay đổi phụ thuộc vào tham số đầu vào, tên của đồ thị sẽ là tên của chuỗi ký tự khai báo trong phần tham số hàm và được hiển thị trên hình vẽ. Đặc biệt, ***ezplot*** cho phép vẽ đồ thị biểu diễn hàm số khai báo dưới dạng chuỗi ký tự.

Cú pháp lệnh như sau:

```
>>ezplot(fun);
```

```
>>ezplot(fun, [a, b]);
```

Trong đó *fun* là hàm số được vẽ đồ thị. Ở cú pháp thứ nhất, đồ thị hàm số sẽ được thể hiện trên khoảng mặc định là -2π đến 2π, ở cú pháp thứ hai, người sử dụng có thể thể hiện đồ thị trên khoảng *a, b* tùy chọn.

Ví dụ sau đây so sánh hai hàm ***plot*** và ***ezplot***: vẽ đồ thị hàm số $y = x^2 + 2x + 1$.

```
>>x = -2*pi: .1 : 2*pi;
>>y = x.^2 + 2.*x + 1;
>>plot(x,y)
```

```
>>ezplot('x^2 + 2*x + 1');
```

Khi sử dụng hàm ***plot***, đầu tiên cần khai báo vector *x*, từ hàm số đã cho tìm giá trị vector *y*, sau đó thực hiện lệnh vẽ đồ thị. Tuy nhiên đối với ***ezplot*** chỉ cần thực hiện một câu lệnh đơn giản, kết quả đồ thị thu được từ hai cách là tương đương. Sử dụng hàm **ezplot** trong MATLAB có thể coi là cách đơn giản nhất để thể hiện đồ thị của một hàm số kiểu chuỗi ký tự hoặc biến ký hiệu, tuy nhiên về mặt chức năng, **ezplot** có nhiều hạn chế hơn so với **plot**.

Các thông số của đồ thị biểu diễn trong MATLAB có thể được thay đổi sử dụng các lệnh sau:

- Điều chỉnh hai trục tọa độ: ***axis*** trong đó cần lưu ý cú pháp của hai câu lệnh sau:
  - `>>axis ([Xmin Xmax Ymin Ymax])`
    - biểu diễn đồ thị trong giới hạn $Xmin \le X \le Xmax$, $Ymin \le Y \le Ymax$.
  - `>>axis equal`

  - thiết lập độ chia bằng nhau ở hai trục tọa độ.
- Đặt tên cho đồ thị: MATLAB cho phép đặt, thay đổi tên của đồ thị sử dụng lệnh **title** với cú pháp:
  - `>>title('Ten do thi')`
- Đặt tên cho hai trục tọa độ: Hai trục tọa độ của MATLAB có thể được đặt, thay đổi tên sử dụng các lệnh
  - `>>xlabel('text')` và `>>ylabel('text')` trong đó *text* là tên của các trục tọa độ hoành, tung tương ứng.
- Hiển thị lưới vùng đồ thị: hàm ***grid*** trong MATLAB cho phép bật/tắt chế độ hiển thị lưới trên vùng đồ thị sử dụng các lệnh:
  - `>>grid on` và `>>grid off` tương ứng, nếu không vừa lòng với kích cỡ ô lưới trên đồ thị, có thể sử dụng lệnh:
  - `>>grid minor` để thu được vùng đồ thị với các ô lưới dày đặc hơn.

6.1.2 Các lệnh tăng cường

Như đã biết, kết quả trả về của các lệnh ***plot, ezplot*** sẽ được lưu giữ trên các cửa sổ đồ họa *Figure***.** Trong thực tế, nhiều khi cần phân tích nhiều hình vẽ, đồ thị cùng một lúc nhưng chế độ mặc định của MATLAB không cho phép lưu giữ hình ảnh trên các cửa sổ đồ họa *Figure* giữa các lần vẽ. Có nghĩa là kết quả của lần thực hiện lệnh ***plot*** hoặc ***ezplot*** hiện tại sẽ thay thế kết quả của lệnh vẽ trước đó trên *Figure*. Để kết quả của lần vẽ trước không bị mất đi, người sử dụng MATLAB có thể dùng lệnh ***figure( )*** để tạo cửa sổ đồ họa mới và lưu kết quả trên đó hoặc dùng lệnh ***hold*** để thể hiện kết quả của lệnh vẽ hiện hành và trước đó trên cùng một cửa sổ đồ họa.

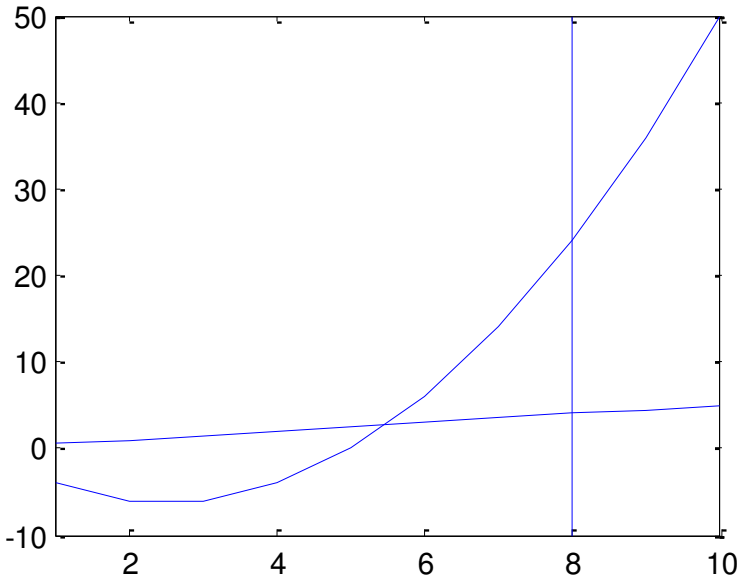
Lệnh ***figure(k)*** cho phép tạo cửa sổ đồ họa mới có tên là *Figure k* hoặc chuyển sang làm việc với *Figure k* nếu cửa sổ đồ họa này đã tồn tại trước đó, *Figure k* khi đó sẽ trở thành cửa sổ đồ họa hiện hành. Cửa sổ đồ họa hiện hành sẽ được đóng bằng cách gõ lệnh ***close*** trên cửa sổ lệnh. Lệnh ***close all*** sẽ đóng tất cả các cửa sổ đồ họa MATLAB đang hiển thị.

Lệnh ***hold*** có thể sử dụng theo ba cách sau:

- ***hold on*****:** kích hoạt chế độ đóng băng, người dùng có thể chèn thêm hình vẽ lên cửa sổ đồ họa mà không sợ những hình vẽ trước đó bị xóa đi.
- ***hold off*****:** tắt chế độ đóng băng, hình vẽ hiện hành sẽ thay thế hình vẽ được tạo trước đó trên cửa sổ đồ họa.
- ***hold*****:** chuyển giữa hai trạng thái bật/tắt của chế độ đóng băng.

Ví dụ sau cho phép vẽ lần lượt ba đồ thị của ba hàm số $y1 = x/2$, $y2 = x^2 - 5x$ và $x = 8$ trên cùng một cửa sổ đồ họa:

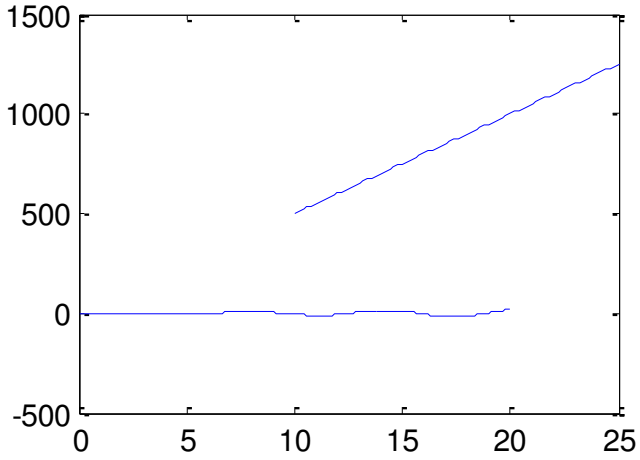
```
>>x = 1:10;
>>y1 = 0.5*x;
>>y2 = x^2 - 5*x;
>>plot(x,y1);
>>hold on;
>>plot(x,y2);
>>plot([8,8],[-10,50]);
```



Hình 6. 6. Vẽ 3 đồ thị trên cùng một cửa sổ đồ họa

Sau đây là một ví dụ khác về việc thể hiện đồ thị của hai hàm số trên cùng một đồ thị tuy nhiên giá trị thể hiện trên trục tung lại không tương đương nhau.

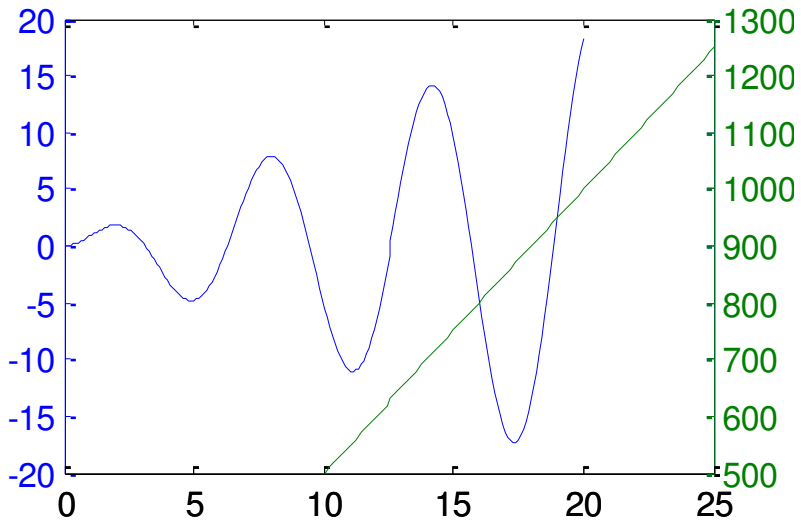
```
>>x1 = 0 : .1 : 20;
>>y1 = x1.*sin(x1);
>>x2 = 10 : .2 : 25;
>>y2 = 50*x2;
>>plot(x1, y1);
>>hold on;
>>plot(x2, y2);
```



Hình 6. 7. Hai đồ thị sử dụng cùng một trục tung

Do khoảng giá trị của *y1* và *y2* là không giống nhau nên khi biểu diễn trên cùng một đồ thị với một khoảng chia của trục tung, rất khó để nhận biết dạng đường của *y1*. Đối với trường hợp này, có thể sử dụng hàm ***plotyy*** để thể hiện đồ thị của hai hàm số với hai trục tung khác nhau.

>>plotyy(x1,y1,x2,y2);



Hình 6. 8. Hai đồ thị với hai trục tung tương ứng khác nhau

Một cách khác để thể hiện nhiều hơn một đồ thị trên cùng một trục tọa độ đó là sử dụng lệnh ***lines***, lệnh này cho phép chèn thêm đồ thị tạo bởi các giá trị của một cặp vector (hoặc 3 vector) lên một trục tọa độ có sẵn.

Ví dụ:

Thể hiện đồ thị của các đường sau trên cùng một trục tọa độ $0 \le x \le 5, -1 \le y \le 5$

$$y1 = \sin t$$

$$y2 = t$$

$$y3 = t - \frac{t^3}{3!} + \frac{t^5}{5!} - \frac{t^7}{7!}$$

$$0 \le t \le 2\pi$$

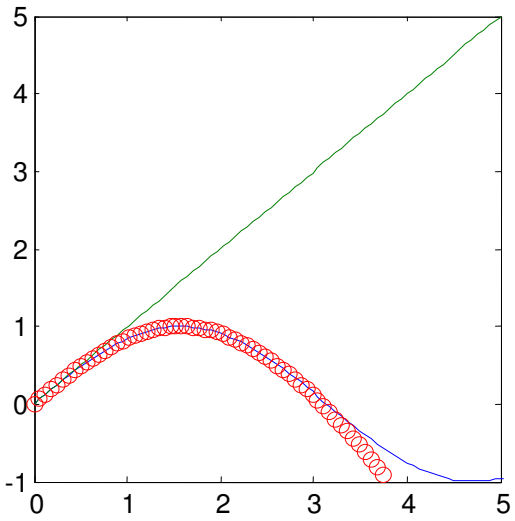
a. Chỉ sử dụng một lần lệnh ***plot***.
b. Sử dụng lệnh ***hold***.
c. Sử dụng lệnh ***line***.

Khai báo các biến:

```
>> t=linspace(0,2*pi,100);
>> y1=sin(t);
>> y2=t;
>> y3=t- (t.^3)/6+ (t.^5)/120 - (t.^7)/5040;
```

Vẽ 3 đồ thị trên cùng trục tọa độ sử dụng một lần lệnh ***plot***:

```
>> plot(t,y1,t,y2,'-',t,y3,'o')
>> axis([0 5 -1 5])
```



Hình 6. 9. Vẽ 3 đồ thị sử dụng một lần lệnh plot

Vẽ 3 đồ thị trên cùng một trục tọa độ sử dụng lệnh ***hold***:

```
>> plot(t,y1)
>> hold on
>> plot(t,y2,'-');
>> plot(t,y3,'o');
```

```
>> axis([0 5 -1 5]);
```

Vẽ 3 đồ thị trên cùng một hệ trục tọa độ sử dụng lệnh ***line***:

```
>> plot(t,y1)
>> line(t,y2,'linestyle','-')
>> line(t,y3,'marker','o')
>> axis([0 5 -1 5]);
```

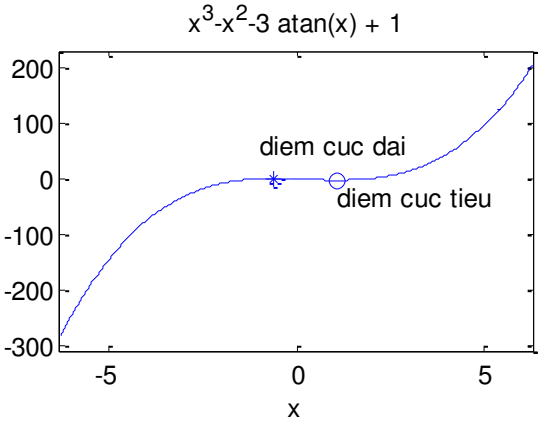
Một chức năng khác của đồ họa MATLAB là cho phép người sử dụng chèn một chuỗi ký tự lên trên trục tọa độ tại một điểm xác định. Lệnh chèn ký tự thông dụng nhất là **text** với cú pháp:

```
>>text (X, Y, 'string')
```

Trong đó *X, Y* là hoành độ và tung độ tương ứng biểu diễn vị trí của chuỗi ký tự trên trục tọa độ, *string* là nội dung chuỗi ký tự. Lưu ý là nếu *X, Y* ở dạng vector, lệnh **text** sẽ thể hiện nội dung chuỗi ký tự lên trên trục tọa độ tại tất cả các điểm cho bởi các phần tử của *X* và *Y* tương ứng. Bên cạnh đó, MATLAB cũng cho phép người dùng chèn chuỗi ký tự lên trục tọa độ mà vị trí của chuỗi ký tự sẽ được xác định bằng cách dùng trỏ chuột.

Ví dụ: Cho hàm số $f(x) = x^3 - x^2 - 3tan^{-1}x + 1$, tìm các cực trị của hàm số trong khoảng
$[-2\pi, 2\pi]$ và thể hiện trên đồ thị. Đây là một ví dụ đã được trình bày trong chương 3, hàm số có cực đại trên (-1, 0) và cực tiểu trên (0, 2) với các giá trị tương ứng.

```
>>xmin = 1.0878;
>>ymin = -1.3784;
>>xmax = -0.5902;
>>ymax = 2.0456;
>>ezplot('x^3-x^2-3*atan(x)+1');
>>hold on;
>>plot(xmin,ymin,'o');
>>text(xmin,ymin-30,'diem CT');
>>plot(xmax,ymax,'*');
>>gtext('diem CD');
```



Hình 6. 10. Thể hiện cực trị của hàm số trên đồ thị

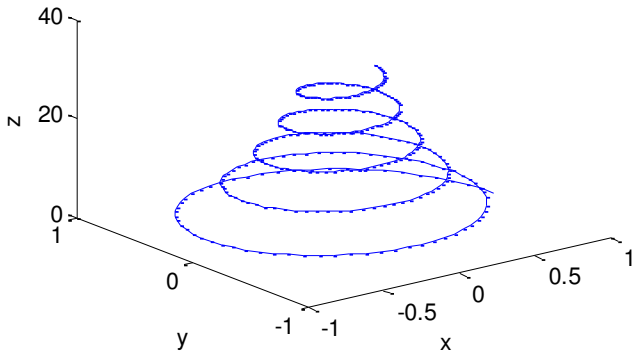
6.2 Đồ thị trong không gian ba chiều

6.2.1 Sử dụng plot3

MATLAB cung cấp cho người dùng một thư viện hàm phong phú để làm việc với chế độ đồ họa trong cả không gian hai chiều và ba chiều. Trong mục này, một số lệnh cơ bản để tái hiện lại dữ liệu trong không gian ba chiều cũng như một số kỹ thuật tạo bề mặt sẽ được trình bày với mục đích cung cấp cho người đọc một cái nhìn tổng quan về các hàm đồ họa trong MATLAB.

Tương tự như lệnh ***plot*** như trong không gian hai chiều, ta có thể sử dụng lệnh ***plot3*** để tái hiện lại dữ liệu trong không gian ba chiều với các chức năng tương tự, kèm theo một tham số thứ ba để biểu diễn trục *z*. Xét ví dụ sau với lệnh ***plot3***:

```
>>t = 0: .1 : 10*pi;
>>x = exp(-t/20).*cos(t);
>>y = exp(-t/20).*sin(t);
>>z = t;
>>plot3(x, y, z);
>>xlabel('x');
>>ylabel('y');
>>zlabel('z');
```



Hình 6. 11. Minh họa lệnh plot3

Hàm ***plot3*** có dạng tổng quát

```
>>plot3 (x, y, z, 'option')
```

Kết quả đầu ra phụ thuộc vào các tham số đầu vào như sau:

- Nếu *x, y, z* là các vector có cùng số phần tử, MATLAB tạo một đường trong không gian ba chiều nối các phần tử tương ứng của *x, y, z*.
- Nếu *x, y, z* là các ma trận có cùng số hàng và số cột, MATLAB sẽ tạo ra nhiều đường theo số cột của ma trận.
- Nếu một trong ba tham số là ma trận và các tham số còn lại là vector trong đó số phần tử của vector bằng với số hàng hoặc số cột của ma trận, MATLAB sẽ tùy thuộc vào đó để tạo ra các đường khác nhau. Nếu số phần tử của vector không bằng với số hàng hoặc số cột, MATLAB sẽ trả về thông báo lỗi.
- Các lựa chọn trong mục 'option' trong cú pháp gọi hàm ***plot3*** cũng giống như các lựa chọn cho hàm ***plot***.

Trong không gian ba chiều, góc quan sát của người dùng đối với hình thể hiện trên cửa sổ đồ họa có thể điều chỉnh sử dụng công cụ điều chỉnh ***rotate3D*** trên thanh công cụ hoặc sử dụng hàm ***view***. Cú pháp tổng quát cho hàm ***view*** này là

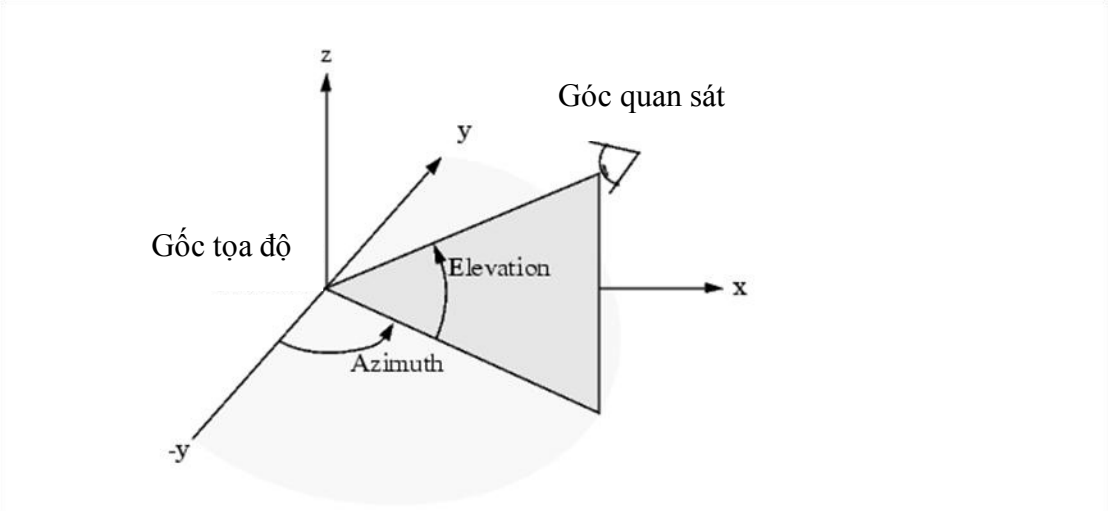
```
>>view(az, el)
```

hoặc

```
>>view([az, el])
```

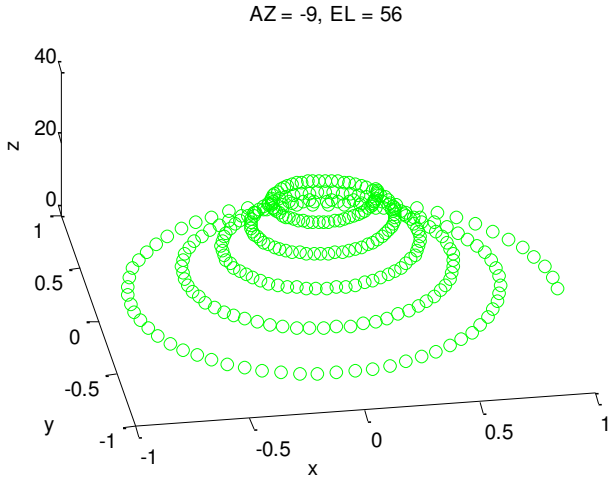
cho phép điều chỉnh chính xác góc quan sát theo các thông số *azimuth* và *elevation*. Trong đó, *azimuth* là góc quay trong mặt phẳng *xy* nhìn từ chiều âm của trục *y*, *elevation* thể hiện góc quan sát so với mặt phẳng *xy*, góc dương thể hiện quan sát phía trên mặt phẳng *xy*, góc âm thể hiện quan sát phía dưới mặt phẳng

xy. Đơn vị của hai thông số *azimuth* và *elevation* đều là độ. Trong MATLAB, các thông số này được mặc định lần lượt là AZ=-37.5^o và EL=30^o.



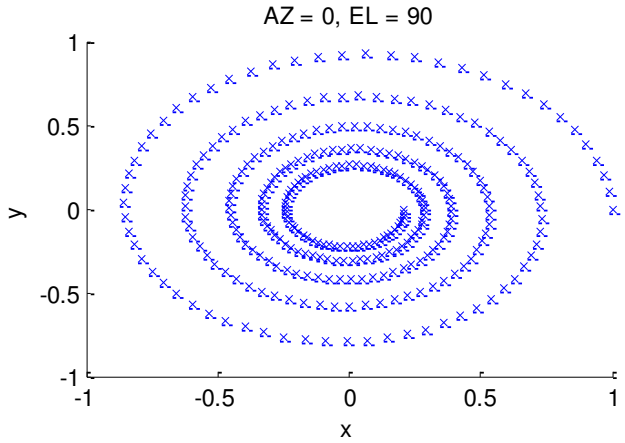
Hình 6. 12. Ý nghĩa hai thông số az, el trong lệnh view

Ví dụ sau thể hiện các góc quan sát khác nhau đối với đồ thị của các giá trị *x, y, z* xây dựng ở trên:



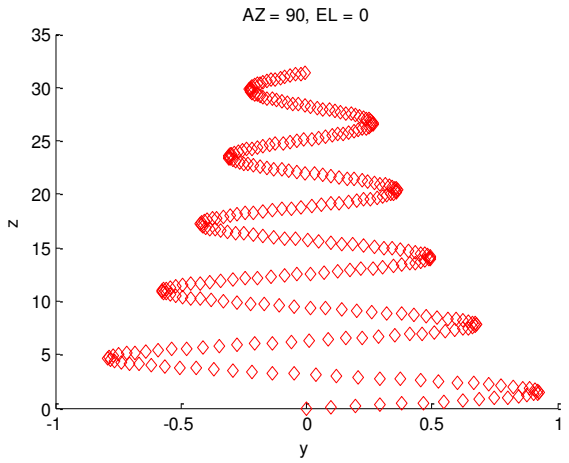
Hình 6. 13. Đồ thị 3 chiều với (AZ, EL) = (-9, 56)

```
>>plot3(x,y,z,'og');
>>xlabel('x');
>>ylabel('y');
>>zlabel('z');
>>view(-9,56);
>>title('Az=-9,El=56');
```



Hình 6. 14. Đồ thị 3 chiều với (AZ, EL) = (0,90)

```
>>plot3(x,y,z,'xb');
>>xlabel('x');
>>ylabel('y');
>>zlabel('z');
>>view(0,90);
>>title('Az=0,El=90');
```



Hình 6. 15. Đồ thị với (AZ, EL) = (90, 0)

```
>>plot3(x,y,z,'dr');
>>xlabel('x');
>>ylabel('y');
>>zlabel('z');
>>view(90,0);
>>title('Az=90,El=0');
```

6.2.2 Tạo lưới và mặt phẳng 3 chiều

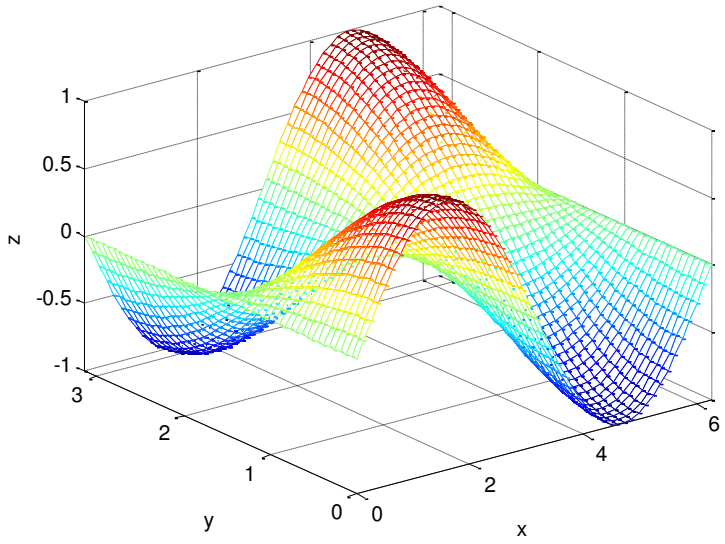
Trường hợp thường gặp khi xử lý đồ thị trong không gian 3 chiều là các dữ liệu phụ thuộc vào cả hai giá trị *x* và *y* và thường được biểu diễn dưới dạng *z=f(x,y)*. Bên cạnh việc sử dụng vòng lặp để tạo các biến *z* phụ thuộc vào cặp giá trị *x, y* cho trước, hàm ***meshgrid*** trong MATLAB đem lại kết quả tương đương nếu *x, y* được khai báo dưới dạng vector. Cú pháp gọi hàm:

`>>[X, Y]=meshgrid(x,y)` sẽ tạo ra các ma trận *X, Y* tạo bởi các vector *x, y* tương ứng. Trong đó, *X* là ma trận có số hàng bằng số phần tử của *y* và các hàng là sự lặp lại của vector *x*, *Y* là ma trận có số cột bằng số phần tử của *x* và các cột là sự lặp lại của vector *y*. Hàm ***meshgrid*** có thể minh họa cụ thể qua ví dụ sau:

```
>>x=[-1   0   1];
>>y=[5   6   7   8   9];
>>[X, Y]=meshgrid(x,y)
X =
     -1     0     1
     -1     0     1
     -1     0     1
     -1     0     1
     -1     0     1
Y =
     5     5     5
     6     6     6
     7     7     7
     8     8     8
     9     9     9
```

Từ các dữ liệu cho trước, một mặt phẳng dạng lưới có thể thể hiện trong không gian ba chiều sử dụng lệnh ***mesh***. Giả sử tồn tại ba ma trận *X, Y, Z* kích cỡ *M-N*, mỗi một điểm xác định bởi các tọa độ (x(i,j), y(i,j), z(i,j)) tương ứng với phần tử ở hàng *i* cột *j* của các ma trận X, Y, Z là các mắt lưới và sẽ được nối với các mắt lưới ở các hàng, cột lân cận để tạo nên mặt phẳng lưới. Các mắt lưới phía trong (không nằm trên các hàng, cột đầu tiên hoặc cuối cùng của ma trận) sẽ được nối với 4 mắt lưới lân cận, các mắt lưới nằm trên cạnh của mặt phẳng sẽ có 3 mắt lưới lân cận, các mắt lưới ở góc mặt phẳng sẽ có 2 mắt lưới lân cận tương ứng. Bên cạnh đó, để tăng hiệu quả quan sát, lệnh ***mesh*** biểu diễn các khu vực có độ cao khác nhau bằng các màu sắc khác nhau. Xét ví dụ sau:

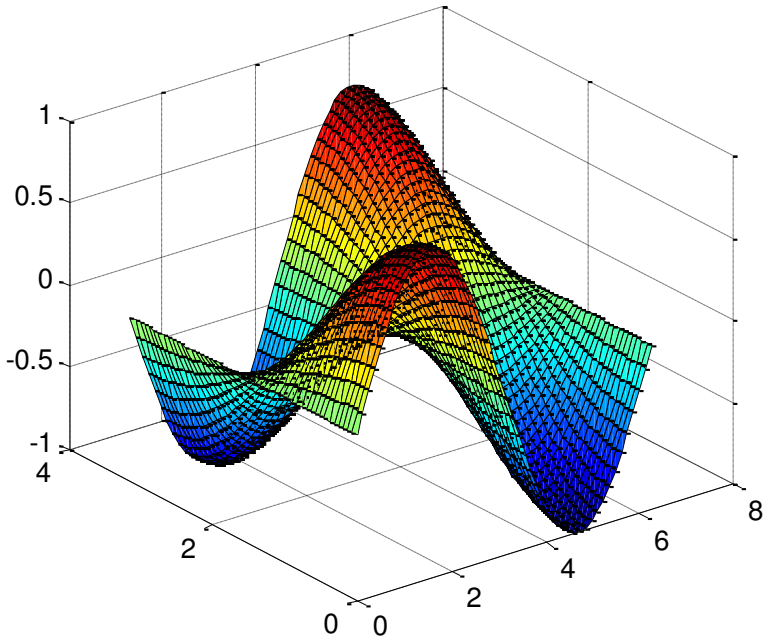
```
>>[X,Y]=meshgrid(linspace(0,2*pi,50),
linspace(0,pi,50));
>>Z=sin(X).*cos(Y);
>>mesh(X, Y, Z);
>>xlabel('x'); ylabel('y'); zlabel('z');
>>axis([0   2*pi   0   pi   -1   1])
```



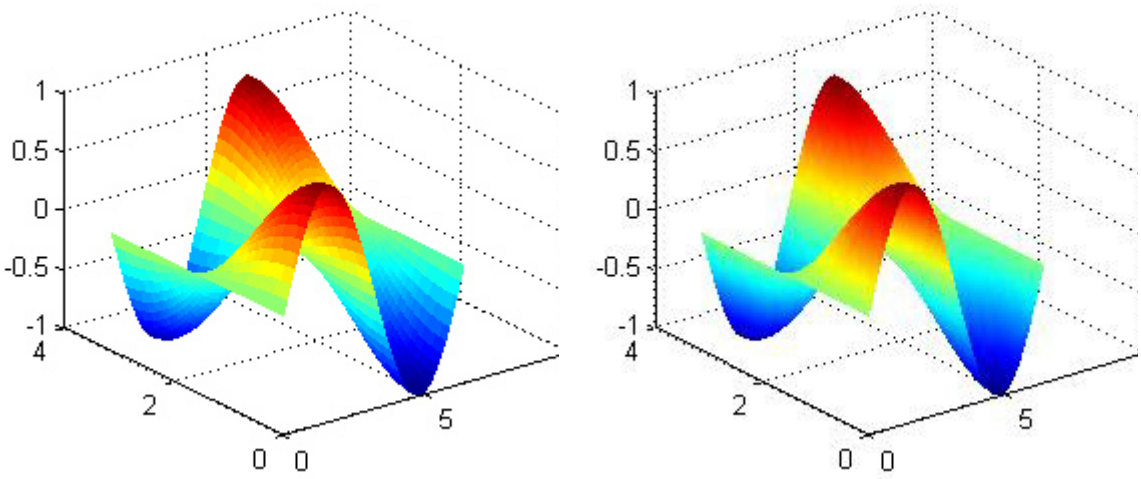
Hình 6. 16. Đồ thị 3 chiều vẽ bởi lệnh mesh

Bên cạnh việc sử dụng lệnh ***mesh*** để tạo bề mặt trong không gian ba chiều từ dữ liệu cho trước, MATLAB cung cấp cho người sử dụng lệnh ***surf*** có tính năng tương đương. Tuy nhiên, các ô lưới sẽ được tô màu và có thể lựa chọn chế độ hiển thị sử dụng lệnh ***shading*** như ví dụ sau:

```
>>surf(X, Y, Z);
>>figure( );
>>subplot(1,2,1);
>>surf(X,Y,Z);
>>shading flat;
>>subplot(1,2,2);
>>surf(X,Y,Z);
>>shading interp;
```



Hình 6. 17. Thể hiện màu trên các ô lưới với lệnh surf



Hình 6. 18. Shading flat và interp

Trong đó ***shading flat*** sẽ không hiển thị đường bao của các ô lưới nhưng màu sắc trong mỗi ô là cố định do đó ta vẫn có thể quan sát được các "đường" trên bề mặt như hình trên, ***shading interp*** biến đổi màu sắc của các ô lưới theo ô lân cận, dẫn đến bề mặt tạo thành trở nên mịn hơn.

6.3 Tóm tắt chương 6

Một số lệnh cơ bản với đồ thị trong không gian hai chiều	
plot	Vẽ đồ thị xy
ezplot	Vẽ nhanh đồ thị xy
line	Vẽ thêm đồ thị lên trục tọa độ hiện hành
figure	Mở cửa sổ đồ họa mới
title	Tạo tiêu đề cho đồ thị
xlabel	Đặt tên cho trục hoành
ylabel	Đặt tên cho trục tung
hold	Bật tắt chế độ vẽ nhiều đồ thị trên một trục tọa độ
grid	Bật tắt chế độ lưới
axis	Hiệu chỉnh hai trục tọa độ
text	Chèn ký tự lên đồ thị theo tọa độ nhập từ bàn phím
gtext	Chèn ký tự lên đồ thị theo tọa độ xác định bằng chuột
legend	Chèn chú thích trong đồ thị
subplot	Chia cửa sổ đồ họa thành nhiều cửa sổ phụ khác nhau
close	Đóng cửa sổ đồ họa hiện hành
close all	Đóng tất cả cửa sổ đồ họa
area	Vẽ và tô màu vùng được vẽ
bar	Tạo biểu đồ cột
fill	Tô màu phần diện tích bao bởi đa giác
fplot	Vẽ đồ thị hàm số một biến số
loglog	Vẽ đồ thị với độ chia loga ở cả hai trục x và y
plotyy	Vẽ đồ thị với hai trục tung
polar	Vẽ đồ thị trong hệ tọa độ cực
quiver	Vẽ vector, trường vector
stairs	Vẽ đồ thị dạng bậc thang

Một số lệnh cơ bản với đồ thị trong không gian ba chiều	
plot3	Vẽ đồ thị xyz
zlabel	Đặt tên cho trục z
meshgrid	Tạo lưới dữ liệu
mesh	Tạo mặt phẳng lưới từ dữ liệu cho trước
surf	Tạo mặt phẳng lưới có độ bóng
shading	Điều chỉnh độ bóng cho mặt phẳng
view	Chọn góc quan sát đồ thị

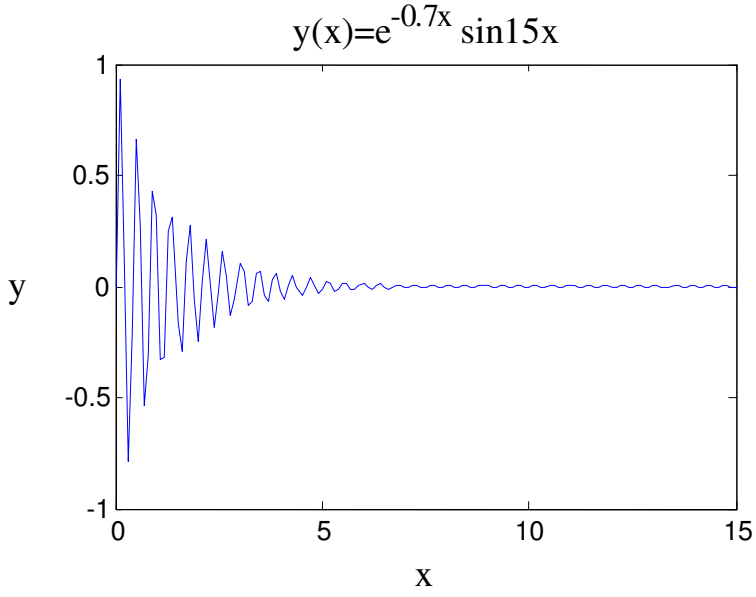
Bảng 6. 2. Tóm tắt các hàm sử dụng trong chương 6

6.4 Bài tập chương 6

Bài 6.1

Vẽ đồ thị của hàm số $y(x)=e^{-0.7x}\sin(15x)$ với $0\le x\le 15$, tạo vector chứa các giá trị của x cách đều nhau 0.1

Đáp án:



Hình 6. 19. Hình bài 6.1

```
>> x=0:0.1:15;
```

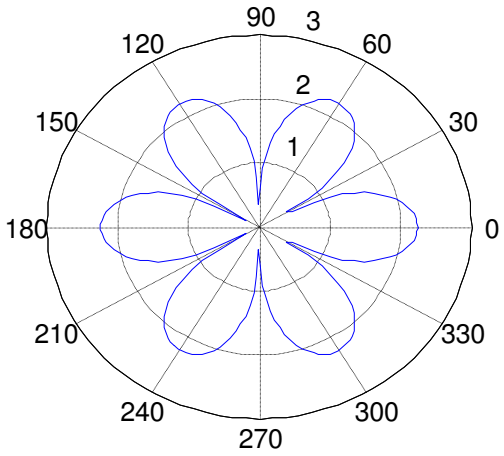
```
>> y=exp(-0.7*x).*sin(15*x);
>> plot(x,y)
>> xlabel('x')
>> ylabel('y')
>> title('y(x)=e^-^0^.^7^x sin15x');
```

Bài 6.2

Vẽ đồ thị hàm số sau sử dụng lệnh ***polar***.

$r^2 = 5\cos 3t$ với $0 \le t \le 2\pi$

Đáp án:



Hình 6. 20. Hình bài 6.2

```
>> t=linspace(0,2*pi,200);
>> r=sqrt(abs(5*cos(3*t)));
>> polar(t,r)
```

Bài 6.3

Dùng lệnh ***fill*** tô màu vùng giới hạn bởi *x, y* trong đó:

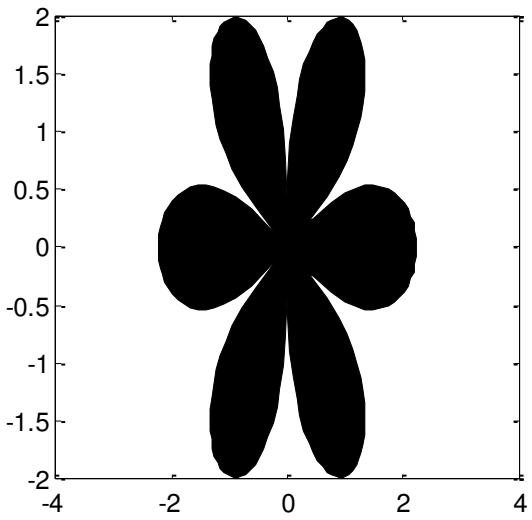
$r^2 = 5\cos 3t \qquad 0 \le t \le 2\pi$

$x = r\cos t \qquad y = r\sin t$

Đáp án:

```
>> t=linspace(0,2*pi,200);
>> r=sqrt(abs(5*cos(3*t)));
```

```
>> x=r.*cos(t);
>> y=r.*sin(t);
>> fill(x,y,'k')
>> axis('square')
```



Hình 6. 21. Hình bài 6.3

Bài 6.4

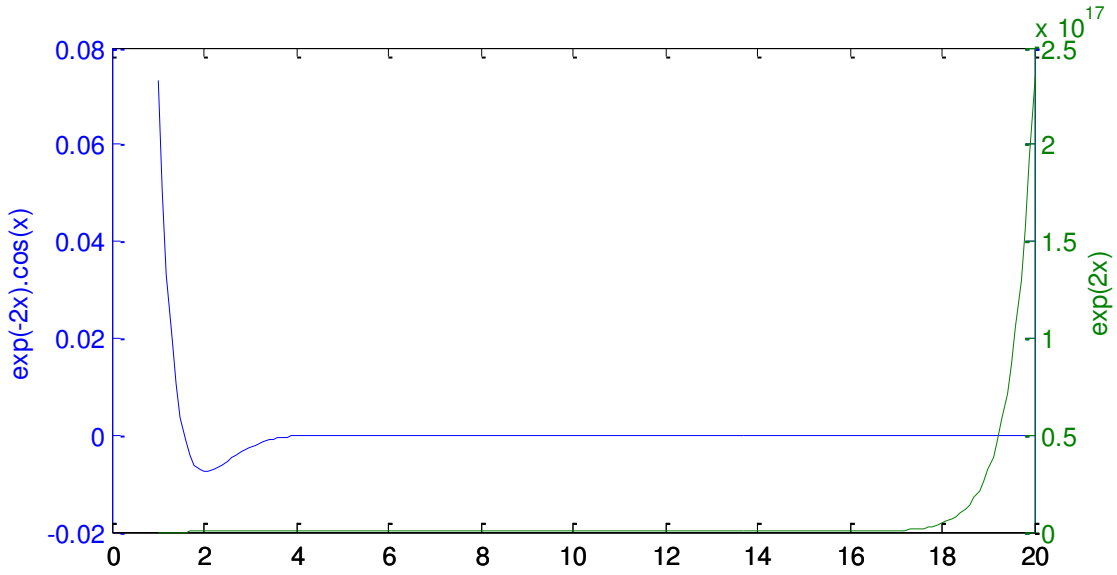
Sử dụng lệnh ***plotyy*** để vẽ hai đồ thị hàm số trên cùng trục tọa độ.

$$y1 = e^{-2x}\cos x$$

$$y = e^{2x} \qquad 0 \le x \le 20$$

Đáp án:

```
>> x=1:0.1:20;
>> y1=exp(-2*x).*cos(x);
>> y2=exp(2*x);
>> Ax=plotyy(x,y1,x,y2);
>> hy1=get(Ax(1),'ylabel');
>> hy2=get(Ax(2),'ylabel');
>> set(hy1,'string','exp(-2x).cos(x)')
>> set(hy2,'string','exp(2x)')
```

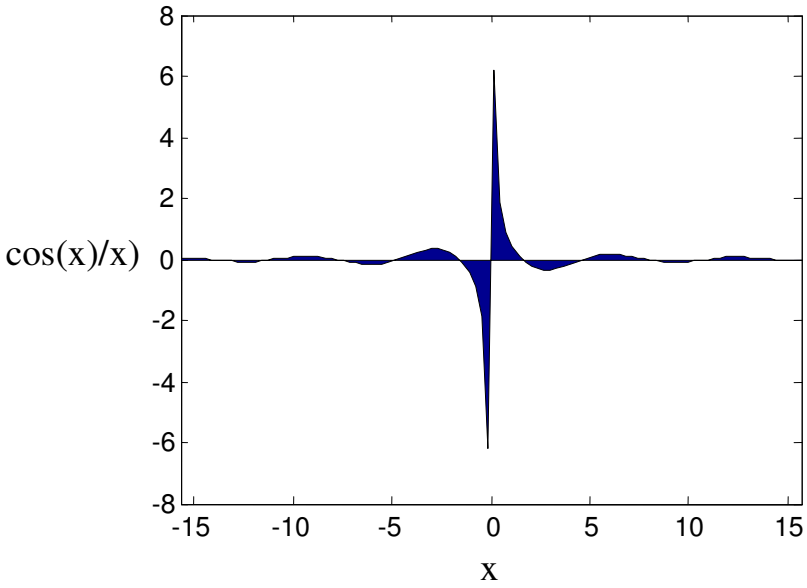


Hình 6. 22. Hình bài 6.4

Bài 6.5

Sử dụng lệnh ***area*** tô màu vùng không gian xác định bởi đồ thị hàm số với trục tung và trục hoành:

$$y = \frac{\cos x}{x} \qquad -5\pi \le x \le 5\pi$$



Hình 6. 23. Hình bài 6.5

Đáp án:

```
>> x=linspace(-5*pi,5*pi,100);
>> y=cos(x)./x;
>> area(x,y);
>> xlabel('x')
>> ylabel('cos(x)/x)')
```

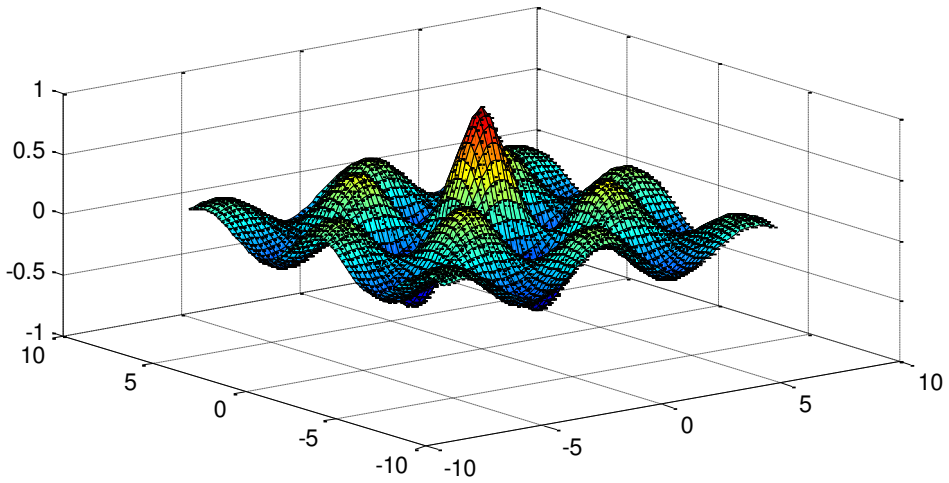
Bài 6.6

Dùng lệnh ***surf*** để vẽ đồ thị hàm số sau trong không gian 3 chiều.

$$z = \cos(x)\cos(y)e^{-\frac{\sqrt{x^2+y^2}}{5}}$$

$$|x| \le 7 \qquad |y| \le 7$$

Đáp án:



Hình 6. 24. Hình bài 6.6

```
>> u=-7:0.2:7;
>> [X,Y]=meshgrid(u,u);
>> Z=cos(X).*cos(Y).*exp(-sqrt(X.^2+Y.^2)/5);
>> surf(X,Y,Z)
```

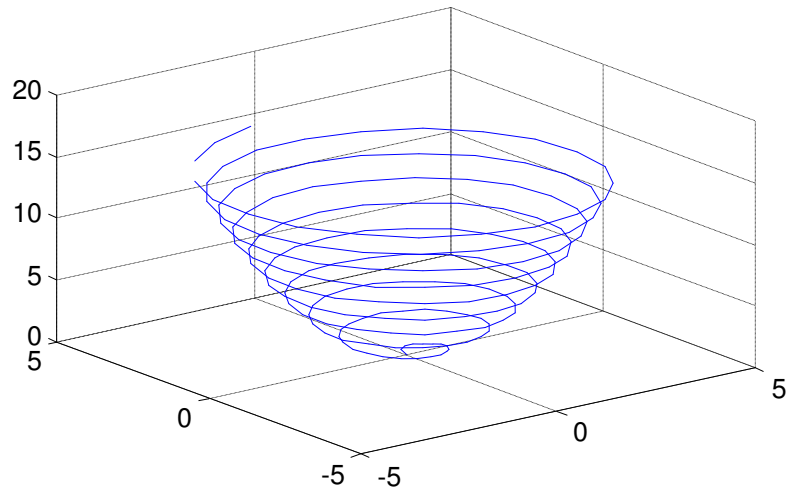
Bài 6.7

Vẽ đồ thị khi x, y, z là các hàm theo biến *t* như sau ($0 \le t \le 6\pi$).

$$x = \sqrt{t}\sin(3t) \qquad y = \sqrt{t}\cos(3t) \qquad z = 0.8t$$

Đáp án:

```
>> t=0:0.1:6*pi;
>> x=sqrt(t).*sin(3*t);
>> y=sqrt(t).*cos(3*t);
>> z=0.8*t;
>> plot3(x,y,z)
>> grid
```



Hình 6. 25. Hình bài 6.7

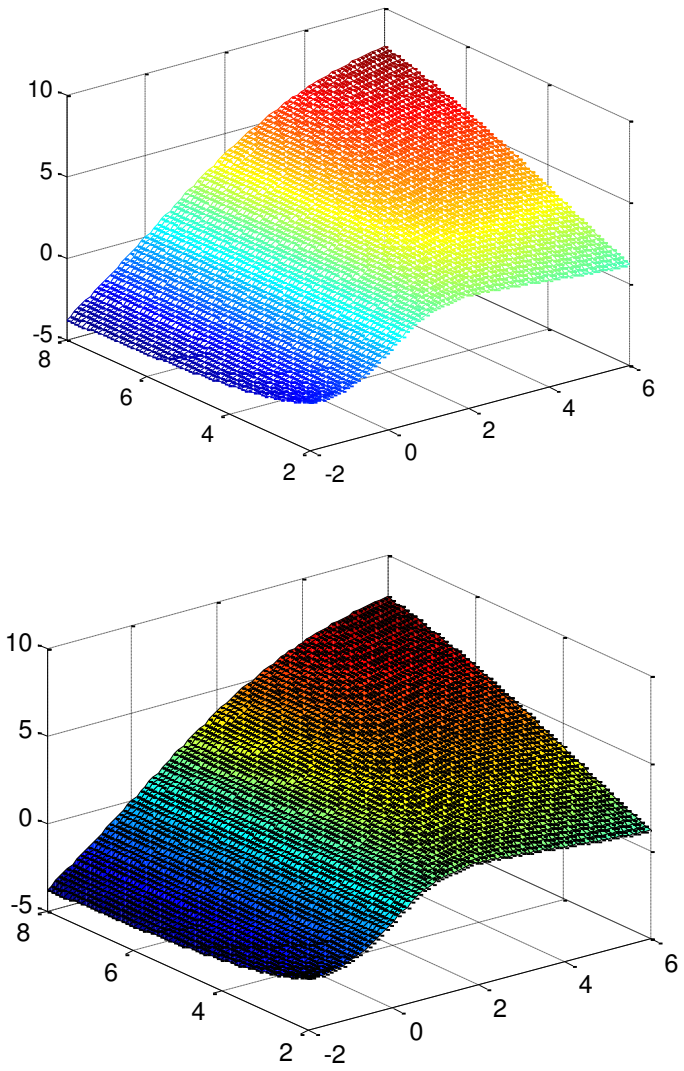
Bài 6.8

Sử dụng lệnh ***mesh*** và ***surface*** để vẽ đồ thị hàm số $z = \frac{2xy^2}{x^2 + y^2}$ trên khoảng $-2 \le x \le 6$ và $2 \le y \le 8$

Đáp án:

```
>> x=-2:0.1:6;
>> y=2:0.1:8;
>> [x,y]=meshgrid(x,y);
>> z=2*x.*y.^2./(x.^2+y.^2);
>> mesh(x,y,z)
>>figure(2)
```

```
>> surf(x,y,z)
```



Hình 6. 26. Hình bài 6.8

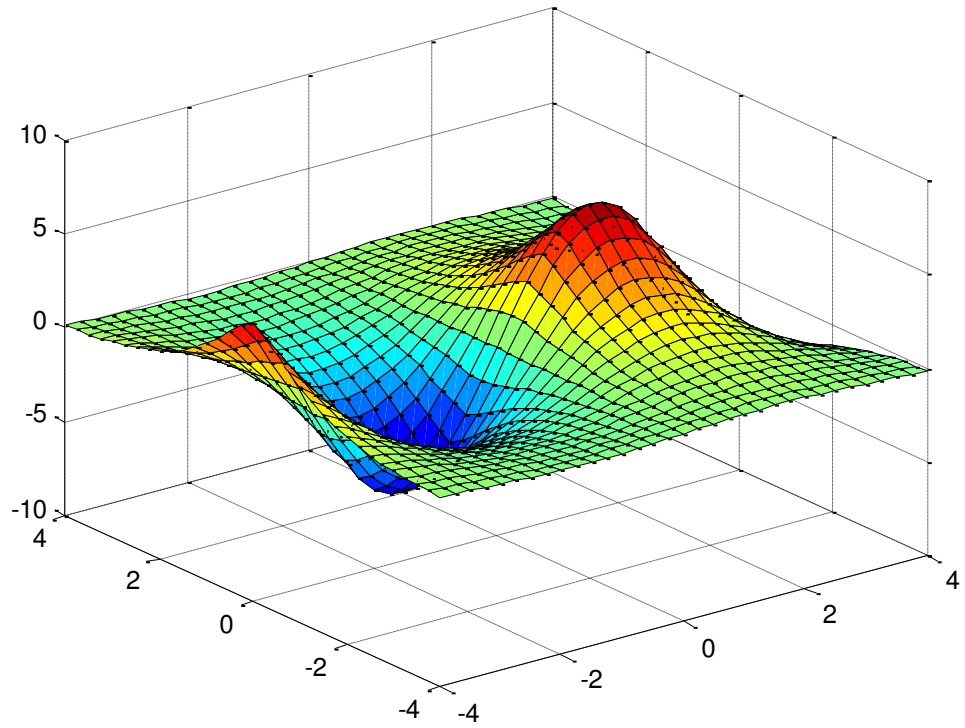
Bài 6.9

Sử dụng lệnh ***surf*** vẽ đồ thị hàm số $z = 2^{-1.5\sqrt{x^2+y^2}} \sin(x)\cos(0.5y)$ trên khoảng $-4 \le x \le 4$ và $-4 \le y \le 4$ và dùng hiệu ứng ***shading interp*** để tạo độ bóng.

Đáp án:

```
>> x=-4:0.25:4;
>> y=-4:0.25:4;
>> [x,y]=meshgrid(x,y);
```

```
>> z=2.^(-1.5*sqrt(x.^2+y.^2))*cos(0.5*y).*sin(x);
>> surf(x,y,z);
>> shading interp;
```



Hình 6. 27. Hình bài 6.9

Bài 6.10

a. Dùng lệnh ***fplot*** để vẽ đồ thị hàm số sau:
$f(t) = t\cos t \qquad 0 \le t \le 10\pi$

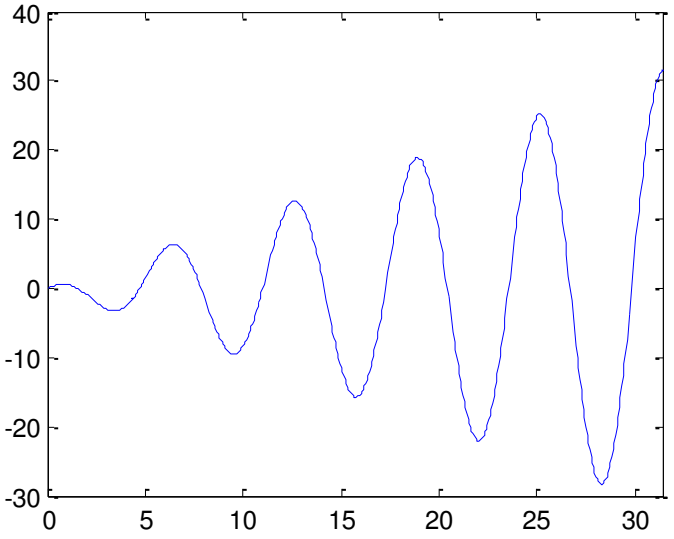
b. Dùng lệnh ***stairs*** để vẽ đồ thị hàm số $y = f(t)$
$r^3 = 3\sin 7t \qquad y = r\sin t \qquad 0 \le t \le \pi$

c. Dùng lệnh ***bar*** để vẽ đồ thị hàm số $y = f(t)$
$r^3 = 3\sin 4t \qquad y = r\sin t \qquad 0 \le t \le \pi$

Đáp án:

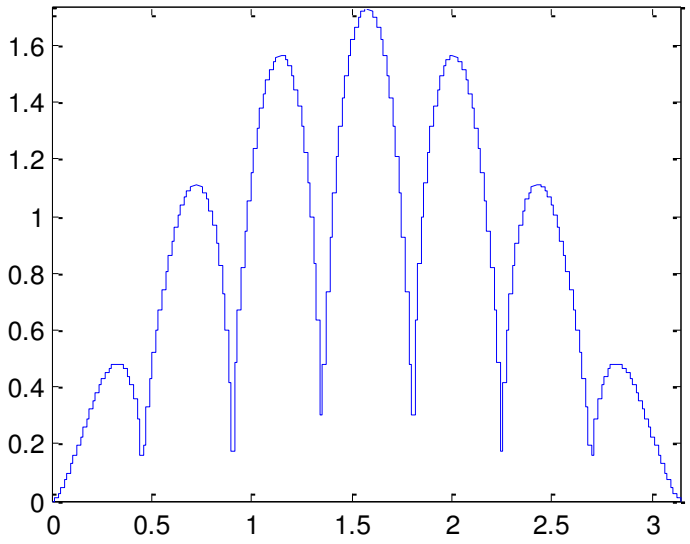
a.

```
>> fplot('x.*cos(x)',[0,10*pi]);
```



Hình 6. 28. Hình bài 6.10a

b.

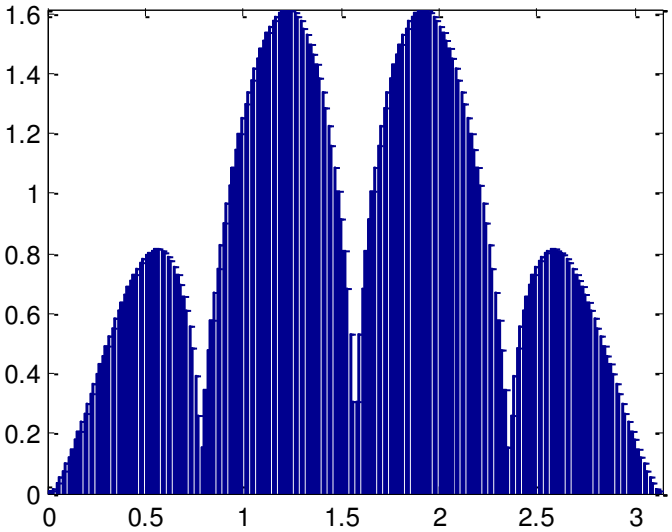


Hình 6. 29. Hình bài 6.10b

```
>> t=linspace(0,pi,200);
>> r=sqrt(abs(3*sin(7*t)));
>> y=r.*sin(t);
>> stairs(t,y)
>> axis([0 pi 0 inf])
```

c.

```
>> t=linspace(0,pi,200);
>> r=sqrt(abs(3*sin(4*t)));
>> y=r.*sin(t);
>> bar(t,y)
>> axis([0 pi 0 inf])
```



Hình 6. 30. Hình bài 6.10c

Chương 7. LẬP TRÌNH TRONG MATLAB

7.1 Giới thiệu M-file

Có bốn cách để lập trình trong MATLAB. Cách đầu tiên và cũng là cách đơn giản nhất đã được đề cập trong năm chương đầu của giáo trình này là nhập lệnh trực tiếp trong cửa sổ lệnh. Khi đó MATLAB sẽ được sử dụng như một máy tính bỏ túi và phù hợp với các bài toán có độ phức tạp không cao. Cách thứ hai là tạo một văn bản **script M-file** có chứa các câu lệnh, khi gọi văn bản trong cửa sổ lệnh, các câu lệnh chứa trong văn bản sẽ lần lượt thực hiện giống như khi ta gõ trực tiếp. Cách thứ ba là sử dụng **function M-file** để tạo các file hàm cho phép xử lý các giá trị đầu vào để đưa ra các giá trị đầu ra thích hợp. Cách thứ tư sẽ không được đề cập đến trong giáo trình này đó là tích hợp các chương trình được viết bởi ngôn ngữ C hoặc FORTRAN vào trong MATLAB sử dụng .mex file. Script và function M-file đều có phần tên chung là M-file vì các file này đều được lưu với phần mở rộng là ".m".

MATLAB có sự khác biệt lớn so với các ngôn ngữ lập trình tiêu chuẩn khác như C, C++, Java. Trước tiên MATLAB có thể hiểu như là một chương trình phiên dịch, có nghĩa là tất cả các chương trình MATLAB đều được đọc, phiên dịch và thực thi theo từng dòng một, sử dụng ngôn ngữ thân thiện với người dùng. Điều này dẫn đến việc thời gian tính toán trong MATLAB sẽ lâu tuy nhiên sẽ rất dễ dàng trong việc phát hiện lỗi cũng như viết các chương trình mẫu hoặc chương trình thử nghiệm. Các ngôn ngữ khác như C, C++, Fortran, Java… là ngôn ngữ lập trình cần được biên dịch. Các chương trình được dịch sang ngôn ngữ máy để xử lý trong file thực thi. Tốc độ tính toán đối với các chương trình này là rất nhanh tuy nhiên khó tìm ra lỗi trong quá trình dịch hay thực thi. Một điểm thuận lợi thứ hai khi lập trình trong MATLAB đó là không cần phải khai báo trước các biến cũng như kiểu dữ liệu tương ứng sẽ sử dụng. Trong MATLAB tất cả các biến đều ở dạng ma trận hoặc dạng chuỗi ký tự trong khi ở những ngôn ngữ lập trình khác, việc định nghĩa trước biến, kích cỡ, kiểu dữ liệu là vô cùng quan trọng. Hai điểm khác biệt khác rất dễ nhận thấy khi so sánh MATLAB với các ngôn ngữ lập trình khác đó là trong MATLAB không sử dụng dữ liệu kiểu con trỏ và MATLAB không phải là ngôn ngữ lập trình hướng đối tượng. Dưới đây là giới thiệu hai dạng ứng dụng phổ biến của M-file trong MATLAB, đó là ***văn bản*** và ***hàm do người dùng tự định nghĩa.***

7.1.1 Văn bản

Để tạo một văn bản trong MATLAB 7.5.0 (R2007b), nhấp chuột vào File, chọn New sau đó chọn M-file. Một cửa sổ mới gọi là **Editor** sẽ xuất hiện. Trong cửa sổ này, nhập chuỗi các câu lệnh mong muốn (số dòng của văn bản sẽ hiển thị ở bên trái, không có dấu nhắc >>) rồi lưu lại với tên có phần mở rộng *.m* (mặc định trong MATLAB). Ví dụ sau minh họa cho việc tạo một văn bản có tên *script1.m* để tính diện tích hình tròn với bán kính cho trước. Trong ví dụ này, nội dung của văn bản sẽ được thể hiện trong ô hình chữ nhật với tên file ở trên

script1.m

```
radius = 5
area = pi * (radius^2)
```

Để xem nội dung của một văn bản đã được lưu, có thể sử dụng một trong hai cách sau đây. Cách thứ nhất là sử dụng lệnh ***type***, nội dung văn bản sẽ hiện lên trong cửa sổ lệnh. Cách thứ hai là mở file trong cửa sổ **Editor**. Trong cửa sổ lệnh, khi gõ tên file của M-file thì tất cả các lệnh chứa trong file đó sẽ được thực thi.

```
>>type script1
radius = 5
area = pi * (radius^2)
>>script1
radius =
     5
area = 78.5398
```

Một điều quan trọng khi làm việc với các file văn bản và hàm đó là các văn bản cần phải trình bày rõ ràng, dễ hiểu để tiện cho việc tham khảo cũng như kiểm tra lỗi trong quá trình chạy. Một trong những cách làm cho việc trình bày văn bản được rõ ràng là sử dụng dấu % để thể hiện chú thích cho các đoạn, câu lệnh mà không ảnh hưởng đến kết quả. Ví dụ văn bản trước có thể điều chỉnh một chút về nội dung như sau:

script1b.m

```
% Chuong trinh tinh dien tich hinh tron
% Cho truoc ban kinh hinh tron

radius = 5
```

```
area = pi * (radius^2)
```

Lệnh ***help*** trong MATLAB cũng có thể sử dụng đối với M-file và sẽ hiển thị khối chú thích đầu tiên (được hiểu là những dòng chú thích kế tiếp nhau) trên cửa sổ lệnh. Ví dụ đối với văn bản *script1b.m* vừa lưu:

```
>>help script1b
Chuong trinh tinh dien tich hinh tron
```

Trường hợp giữa hai dòng chú thích %Chuong trinh… và %Cho truoc… không phải là một dòng trống, MATLAB sẽ hiểu hai dòng này thuộc cùng một khối chú thích và sẽ hiển thị lên màn hình khi lệnh ***help*** cho văn bản này được gọi.

Trong lập trình, việc xuất nhập dữ liệu ở các dạng khác nhau là rất quan trọng. Các lệnh cơ bản về xuất nhập dữ liệu đã được giới thiệu ở chương mở đầu và sẽ được nhắc lại cụ thể hơn ở chương này trong các bài tập tương ứng. Ví dụ: Từ file *script1b.m* viết chương trình tính diện tích hình tròn với bán kính nhập từ bàn phím.

script2.m

```
%Chuong trinh tinh dien tich hinh tron
%Dua vao ban kinh duoc nhap tu ban phim
%Yeu cau nhap ban kinh hinh tron tu ban phim
fprintf('Luu y: Don vi ban kinh la centimet\n')
radius = input('Nhap ban kinh: ');
area = pi * (radius^2);

%In ra cac gia tri ban kinh va dien tich hinh tron
fprintf('Ban kinh hinh tron la %.2f cm, \n', radius)
fprintf('Dien tich hinh tron la %.2f cm2\n', area)
```

Khi chạy chương trình, màn hình sẽ hiện ra các nội dung sau:

```
>> script2
Luu y: Don vi ban kinh la centimet
Nhap ban kinh: 3.9
Ban kinh hinh tron la 3.90 cm,
Dien tich hinh tron la 47.78 cm2
```

7.1.2 Hàm do người dùng tự định nghĩa

Trong MATLAB, bên cạnh các hàm đã xây dựng sẵn, các hàm vô danh, người dùng còn có thể tự định nghĩa các hàm sử dụng M-file được giới thiệu sau

đây. Do hàm tạo trong Editor nên các bước mở cửa sổ Editor cũng tương tự như đối với văn bản cho MATLAB phiên bản 7.5.0 (File → New → M-file), ở các phiên bản mới hơn, có thể chọn File → New → Function cho hàm và File → New → Script cho văn bản. Các hàm do người dùng tự định nghĩa có thể có một hay nhiều tham số, trả về một hay nhiều giá trị, tuy nhiên về cơ bản, mọi hàm trong MATLAB cần phải có những yếu tố sau:

- Dòng đầu tiên của hàm với các thông tin:
  - Từ khóa ***function***.
  - Tên các giá trị trả về của hàm (nếu có).
  - Tên hàm.
  - Tên các tham số của hàm (nếu có).
- Khối dòng chú thích trình bày ý nghĩa của hàm, khối này sẽ hiện ra khi sử dụng lệnh ***help***.
- Thân hàm: bao gồm tất cả các lệnh xử lý của hàm.
- Từ khóa ***end*** ở dòng cuối cùng của hàm.

Ví dụ: Hàm *calcarea* tính và trả về giá trị diện tích hình tròn lưu trong file *calcarea.m* như sau:

calcarea.m

```
function area = calcarea(rad)
% Ham calcarea tinh dien tich hinh tron neu biet ban kinh
% Cu phap goi ham: calcarea(ban_kinh)
% Tra ve gia tri dien tich cua hinh tron co ban kinh tuong ung

area = pi * rad * rad;
end
```

Bán kính hình tròn sẽ được lưu vào tham số *rad*, hàm sẽ tính diện tích hình tròn và lưu vào biến *area*. Lưu ý: do hàm có giá trị trả về nên phần thân hàm phải có lệnh để truyền giá trị cho biến đầu ra *area* của hàm. Để xem nội dung của hàm bên cạnh việc sử dụng **Editor** có thể dùng lệnh ***type*** như đã giới thiệu ở trên.

Gọi hàm từ cửa sổ lệnh: Sau đây là ví dụ của việc gọi hàm *calcarea* trong đó giá trị trả về lưu trong biến mặc định *ans*:

```
>>calcarea (4)
ans =
      50.2655
```

Nói chung việc gọi hàm sẽ thực thi thành công khi nhập tên của file chứa hàm tuy nhiên để tránh rắc rối và dễ quản lý, nên chọn tên file và tên hàm trùng nhau. Kết quả trả về của hàm có thể lưu trong một biến có tên trùng hoặc khác với tên của giá trị trả về trong hàm.

Giá trị trả về của hàm cũng có thể in ra màn hình hoặc xuất ra file sử dụng lệnh ***disp*** hoặc ***fprintf***:

```
>>disp(calcarea(4))
50.2655
>>fprintf('Dien tich hinh tron la %.1f\n', calcarea(4))
Dien tich hinh tron là 50.3
```

Các hàm do người dùng định nghĩa cũng có thể được gọi từ văn bản giống như các hàm xây dựng sẵn trong MATLAB. Ví dụ tính diện tích hình tròn trên đây là một trường hợp đơn giản chỉ cần sử dụng một tham số đầu vào. Tuy nhiên trong thực tế chúng ta thường gặp phải các bài toán phức tạp yêu cầu nhiều tham số đầu vào cũng như các tính toán trung gian đối với biến cục bộ để có thể đưa ra giá trị trả về mong muốn như ví dụ sau: Tính tiền công bằng USD cho việc gia công một hình trụ tròn biết các kích thước của hình trụ cũng như giá thành gia công cho một m^2 tính bằng VND và tỉ giá 1 USD = ? VND. Chúng ta có thể giải bài toán bằng cách tính toán diện tích xung quanh của hình trụ từ đó tính ra giá thành gia công bằng VND rồi quy đổi ra USD với tỉ giá cho trước. Diện tích xung quanh của hình trụ được tính theo công thức:

$$dt = 2\pi rh + 2\pi r^2$$

giacong.m

```
function chiphi=giacong(r, h, giaVND, tigia)
% Ham tinh so tien tra cho viec gia cong mot hinh tru tron bang USD
% Cu phap goi ham: giacong(ban kinh, chieu cao, gia theoVND, ti gia quy doi)
% Ham tra ve chi phi tinh bang USD
% Ban kinh va chieu cao tinh bang m
% Chi phi co don vi USD/m2
% Tinh dien tich xung quanh cua hinh tru
dt = 2*pi*r*h + 2*pi*r^2;
% Tinh chi phi gia cong bang VND
cpVND = dt*giaVND;
% Tinh chi phi gia cong bang USD
```

```
chiphi = cpvnd/tigia;
end
```

Hàm có thể được gọi như sau, ví dụ hình trụ bán kính 0.1m, chiều cao 0.5m, tiền công là 200 000VND/m^2 và 20 000VND đổi được 1 USD.

```
>>giacong(.1, .5, 200000, 20000)
ans =
       3.7699
>>fprintf('Tien gia cong hinh tru la: %.2f USD\n',giacong(.1,.5,200000,20000))
Tien gia cong hinh tru la 3.77 USD
```

Hàm do người dùng tự định nghĩa trong MATLAB có thể được phân biệt theo số lượng biến đầu vào và kết quả đầu ra như sau

- Hàm tính toán và trả về một giá trị
- Hàm tính toán và trả về nhiều hơn một giá trị
- Hàm chỉ thực hiện một số nhiệm vụ nhất định mà không trả về giá trị nào

Về tổng quan, như đã trình bày ở trên, mọi hàm do người dùng tự định nghĩa trong MATLAB đều có từ khóa ***function*** ở đầu, trong trường hợp hàm có trả về giá trị thì (tập hợp các) tên của giá trị trả về sẽ được thể hiện trên cùng dòng với từ khóa ***function***, theo sau là phép gán =). Tham số đầu vào được biểu diễn trong ngoặc đơn, trong trường hợp có nhiều tham số đầu vào, sử dụng dấu , để phân biệt.

Ví dụ trên là một minh họa cho hàm trả về một giá trị, đối với hàm trả về nhiều hơn một giá trị, cú pháp tổng quát của một hàm dạng này như sau

ten_ham.m

```
function [tap hop cac gia tri tra ve] = ten_ham(tap hop cac tham so vao)
%Dong chu thich mo ta noi dung ham
Cac cau lenh thanh phan
Tat ca cac gia tri tra ve liet ke o dong dau deu phai duoc gan gia tri
end
```

Lưu ý là đối với dạng hàm này, trong phần thân hàm bắt buộc phải có lệnh gán giá trị cho tất cả các giá trị trả về đã được liệt kê ở phần khai báo hàm.

Ví dụ hàm sau sẽ tính và trả về hai giá trị chu vi và diện tích của hình tròn với tham số đầu vào là bán kính của hình tròn tương ứng.

areacirc.m

```
function [area, circum] = areacirc(rad)
% ham areacirc tra ve gia tri chu vi va dien tich hinh
% tron
% cu phap goi ham: areacirc(radius)
area = pi * rad .* rad;
circum = 2 * pi * rad;
end
```

Do hàm có hai giá trị trả về, ở dòng khai báo hàm, hai giá trị trả về này sẽ được biểu diễn trong ngoặc vuông, phân biệt với nhau bởi dấu phẩy. Lưu ý là do hàm có hai giá trị trả về nên cần phải lưu các giá trị này vào các biến riêng biệt khác nhau, trong ví dụ này, diện tích sẽ được lưu vào biến *a* và chu vi được lưu vào biến *c*

```
>>[a c] = areacirc(4)
a =
       50.2655
c =
       25.1327
```

Trong trường hợp không lưu vào các biến riêng lẻ, chỉ có giá trị đầu tiên của hàm được trả về, ví dụ:

```
>>disp(areacirc(4))
       50.2655
```

Sinh viên cần lưu ý về thứ tự lưu các giá trị trả về như trong dòng khai báo hàm, trong trường hợp này, giá trị trả về đầu tiên sẽ là diện tích, tiếp theo là chu vi, tuy nhiên thứ tự gán kết quả cho các giá trị trả về trong thân hàm không quan trọng. Ở bài này, sinh viên có thể kiểm tra kết quả khi tham số đầu vào không phải là một số mà là một vector.

Tương tự như các hàm xây dựng sẵn trong MATLAB, khi dùng hàm ***help*** đối với hàm vừa tạo, màn hình sẽ hiển thị nội dung trong các dòng chú thích ngay phía dưới dòng khai báo hàm, ví dụ

```
>>help areacirc
ham areacirc tra ve gia tri chu vi va dien tich hinh
tron
cu phap goi ham: areacirc(radius)
```

Việc gọi hàm cũng hoàn toàn tương tự như đối với các hàm xây dựng sẵn trong MATLAB, nghĩa là người sử dụng có thể gọi trực tiếp trong cửa sổ lệnh hoặc trong văn bản…

Đối với các hàm không có giá trị trả về mà chỉ thực hiện một số nhiệm vụ nhất định, cú pháp tổng quát như sau

ten_ham.m

```
function ten_ham(tap hop cac tham so vao)
%Dong chu thich mo ta noi dung ham
Cac cau lenh thanh phan
end
```

Lưu ý ở dòng khai báo hàm, ta có thể thấy không có khai báo các giá trị trả về cũng như không có phép gán =, ví dụ hàm sau chỉ thực hiện nhiệm vụ in các tham số đầu vào đã nhập trong cùng một câu

printem.m

```
function printem(a,b)
% printem in hai tham so dau vao trong cung mot cau
% cu phap goi ham: printem(so1, so2)
fprintf('So thu nhat %.1f va so thu hai %.1f\n',a,b)
end
```

Ví dụ gọi hàm

```
>>printem(3.3, 2)
So thu nhat 3.3 va so thu hai 2.0
```

Lưu ý do hàm không có giá trị trả về nên nếu gán kết quả của hàm cho một biến, MATLAB sẽ hiển thị thông báo lỗi, ví dụ

```
>>x = printem(3, 5) %Error!!
??? Error using ==> printem
Too many output arguments.
```

Đối với các ví dụ trên, ta đều thấy các hàm đều có tham số đầu vào, tuy nhiên trong một số trường hợp, tùy vào bài toán cụ thể, sự xuất hiện của tham số đầu vào là không cần thiết, ví dụ hàm sau sẽ in ra một số ngẫu nhiên với hai chữ số ở phần thập phân

printrand.m

```
function printrand()
% printrand in ra man hinh mot so ngau nhien
% cu phap goi ham: printrand hoac printrand()
fprintf('So ngau nhien la %.2f\n',rand)
end
```

Ví dụ gọi hàm như sau

```
>>printrand()
So ngau nhien la 0.94
```

Do không có tham số đưa vào hàm nên không có tham số trong dấu ngoặc khi khai báo hoặc gọi hàm, nói cách khác, đối với những dạng hàm này, không cần sử dụng dấu ngoặc đơn trong dòng khai báo hàm hoặc khi gọi hàm.

7.2 Các lệnh lựa chọn

Hai câu lệnh lựa chọn cơ bản trong MATLAB là ***if*** và ***switch*** trong đó ***if*** còn có các lệnh rẽ nhánh ***else*** và ***elseif***. Lệnh ***if*** sẽ kiểm tra các giá trị **đúng** hoặc **sai** của điều kiện để thực hiện các câu lệnh tiếp theo trong đó điều kiện được biểu diễn bởi các phép toán logic và các phép toán so sánh được giới thiệu dưới đây.

7.2.1 Các phép toán quan hệ

Các phép toán quan hệ có thể là các phép toán so sánh giữa hai biểu thức cùng kiểu dữ liệu hoặc là các phép toán logic, giá trị trả về của chúng là **đúng** hoặc **sai**. Các phép toán so sánh trong MATLAB được biểu diễn trong bảng sau:

>	Lớn hơn
<	Nhỏ hơn
>=	Lớn hơn hoặc bằng
<=	Nhỏ hơn hoặc bằng
==	Bằng
~=	Khác

Cú pháp biểu diễn các phép toán so sánh là giống nhau ở các ngôn ngữ lập trình, có thể có sự khác nhau về ký hiệu. Lưu ý ở đây phép so sánh bằng sử dụng ký hiệu "**==**" do ký hiệu "**=**" đã được sử dụng cho phép gán. Trong MATLAB cũng như các ngôn ngữ lập trình khác, **đúng** được biểu diễn bởi giá trị logic "**1**", **sai** được biểu diễn bởi giá trị logic "**0**", xem ví dụ dưới đây:

```
>>3 < 5
ans =
     1
>>2 > 9
ans =
     0
```

Giá trị trả về của phép toán so sánh mặc dù là các giá trị logic nhưng vẫn có thể sử dụng trong tính toán toán học với các giá trị 1 và 0. Ví dụ:

```
>>5 < 7
ans =
     1
>>ans + 3
ans =
     4
```

Các phép toán logic thường gặp là:

| hoặc ||	Logic OR
& hoặc &&	Logic AND
~	Logic NOT

Bên cạnh đó MATLAB còn có phép toán ***xor*** chỉ nhận hai tham số đầu vào và trả về giá trị logic **đúng** khi một và chỉ một tham số là **đúng**. Ví dụ:

```
>>xor(3<5, 'a' > 'c')
ans =
     1
>>xor(3<5, 'a' < 'c')
ans =
     0
```

Bảng giá trị cho các phép toán logic được giới thiệu trong chương này đối với hai biến *x, y* được trình bày trong bảng dưới:

x	y	~x	x || y	x && y	xor (x, y)
1	1	0	1	1	0
1	0	0	1	0	1
0	0	1	0	0	0

7.2.2 IF

If cho phép lựa chọn thực hiện hoặc không thực hiện một hoặc nhóm câu lệnh. Cú pháp tổng quát cho câu lệnh điều kiện này là:

```
if dieu_kien
     hanh_dong
end
```

Trong đó *dieu_kien* là biểu thức quan hệ chỉ nhận giá trị **đúng** hoặc **sai**, *hanh_dong* là một hoặc nhóm câu lệnh sẽ được thực hiện khi *dieu_kien* nhận giá trị **đúng**. Ví dụ sau đây sẽ kiểm tra một giá trị là dương hay âm, sau đó chuyển giá trị âm sang giá trị dương bằng cách lấy giá trị tuyệt đối.

```
if   num < 0
     num = abs(num)
end
```

Có thể sử dụng ***if*** trong cửa sổ lệnh tuy nhiên tốt nhất vẫn nên thực hiện trong M-file để dễ kiểm soát cũng như thực hiện lệnh. Mở rộng ví dụ trên, ta có thể kiểm tra một số nhập vào từ bàn phím là âm hay dương và thực hiện phép tính khai căn với số nhập vào nếu đó là số dương, hoặc khai căn với trị tuyệt đối của số đó nếu số nhập vào âm.

vidukhaican.m

```
% Yeu cau nhap vao mot so tu ban phim, tra ve gia tri
can bac hai
num = input('Hay nhap mot so: ');
% Neu so duoc nhap vao nhan gia tri am, tim tri tuyet
doi cua so do
if num<0
   disp('Ban da nhap vao so am, chuong trinh se khai
can gia tri tuyet doi');
   num = abs (num)
end
fprintf('Can    bac    hai    cua    %.1f    la    %.1f
\n',num,sqrt(num))
```

Chạy thử chương trình với các số nhập vào là 9 và -9, cho biết kết quả.

Trong khi lệnh ***if*** chỉ cho phép lựa chọn có hay không thực thi một hay nhóm câu lệnh, để lựa chọn giữa nhiều hay các nhóm câu lệnh ta có thể sử dụng ***if-else***, các câu lệnh ***if*** lồng nhau, ***switch***.

Cú pháp cho câu lệnh ***if-else*** như sau:

```
if dieu_kien
     hanh_dong_1
else
     hanh_dong_2
end
```

Các *hanh_dong* sẽ được thực hiện dựa trên việc kiểm tra *dieu_kien*. Nếu *dieu_kien* nhận giá trị **đúng**, *hanh_dong_1* sẽ được thực thi và sau khi hoàn thành, câu lệnh ***if-else*** sẽ kết thúc. Nếu *dieu_kien* nhận giá trị **sai**, chương trình sẽ bỏ qua *hanh_dong_1* để thực hiện *hanh_dong_2* và sau đó kết thúc câu lệnh ***if-else***. Ví dụ:

testifelse.m

```
a=rand;
fprintf('So ngau nhien vua nhan: %.1f\n',a);
if a<=.5
    fprintf('la mot so nho hon hoac bang 0.5\n')
else
    fprintf('la mot so lon hon 0.5\n')
end
```

Một trong những ứng dụng của ***if-else*** là để kiểm tra lỗi nhập dữ liệu trong văn bản. Ví dụ ở đầu chương chúng ta có chương trình nhập bán kính hình tròn rồi tính diện tích, bây giờ ta sẽ chèn thêm dòng lệnh để kiểm tra xem bán kính nhập vào có hợp lệ hay không (ví dụ: bán kính nhỏ hơn không là không hợp lệ).

checkradius.m

```
% Chuong trinh tinh dien tich hinh tron
% Kiem tra tinh hop le cua gia tri ban kinh nhap vao
radius = input('Hay nhap vao ban kinh: ');
if radius <= 0
fprintf('Xin loi, %.2f khong phai la ban kinh hop
le\n',radius)
else
area = calcarea(radius);
fprintf('Hinh tron co ban kinh %.2f,',radius)
fprintf(' co dien tich la %.2f\n',area)
end
```

Qua ví dụ trên có thể thấy việc sử dụng ***if-else*** cho phép lựa chọn thực hiện giữa hai câu lệnh hay hai nhóm câu lệnh, trong trường hợp số lệnh, nhóm lệnh

phải lựa chọn nhiều hơn hai, có thể sử dụng các câu lệnh ***if*** lồng nhau. Xét ví dụ với hàm số *y=f(x)* dưới đây:

$y = 1$ nếu $x < -1$

$y = x^2$ nếu $-1 \le x \le 2$

$y = 4$ nếu $x > 2$

Biểu diễn tương ứng trong MATLAB:

```
if x < -1
     y = 1;
else
```

Đến đây chỉ xét trường hợp x>=-1

Sử dụng if-else để lựa chọn giữa hai khoảng còn lại

```
     if x <= 2
          y = x^2;
     else
```

Chỉ còn lại trường hợp x>2

```
          y = 4;
     end
end
```

Trong phần lớn các ngôn ngữ lập trình, khi có nhiều câu lệnh, nhóm câu lệnh để lựa chọn, ta có thể sử dụng các câu lệnh ***if*** lồng nhau. Tuy nhiên, MATLAB cho phép giải quyết vấn đề trên bằng một câu lệnh khác ***elseif***, khi có trên hai lựa chọn trở lên.

```
if dieu_kien_1
     hanh_dong_1
elseif dieu_kien_2
     hanh_dong_2
elseif dieu_kien_3
     hanh_dong_3
else
     hanh_dong_khac
end
```

Xét ví dụ viết hàm đổi điểm số ra điểm chữ sau đây, trong đó điểm số là số nguyên được nhập từ bàn phím, điểm trên 9 đến 10 tương ứng với điểm 'A', điểm

trên 8 đến 9 tương ứng với điểm 'B', điểm trên 7 đến 8 tương ứng với điểm 'C', điểm trên 6 đến 7 tương ứng với điểm 'D', dưới 6 tương đương với 'F', nếu điểm nhập vào lớn hơn 10 hoặc nhỏ hơn 0 thì trả về giá trị 'X'.

function doidiem.m

```
function diemchu = doidiem(diemso)
%Ham doidiem cho phep doi diem chu ra diem so
%Trong do diem so duoc tu ban phim
%Cu phap goi ham: doidiem(diem so)
%Gia tri tra ve la chu cai

%Kiem tra xem diem nhap vao co hop le
if diemso < 0 || diemso > 10
diemchu = 'X';
%Doi diem nhap vao ra diem chu tuong ung
elseif diemso > 9 && diemso <= 10
diemchu = 'A';
elseif diemso > 8 && diemso <= 9
diemchu = 'B';
elseif diemso > 7 && diemso <=8
diemchu = 'C';
elseif diemso > 6 && diemso <=7
diemchu = 'D';
else
diemchu = 'F';
end
end
```

Một số ví dụ gọi hàm vừa tạo:

```
>>diemso = 8.2;
>>diemchu = doidiem(diemso)
diemchu =
     B
>>diemchu = doidiem(5.8)
diemchu =
     F
```

```
>>diemchu = doidiem(11)
diemchu =
     X
```

7.2.3 SWITCH

Lệnh ***switch*** thường sử dụng trong trường hợp các câu lệnh ***if*** lồng nhau hoặc sử dụng nhiều câu lệnh ***elseif***. ***Switch*** được sử dụng khi so sánh xem một biểu thức có giá trị **bằng** các giá trị cụ thể cho trước.

Cú pháp của ***switch***:

```
switch bieu_thuc
     case truong_hop_1
          hanh_dong_1
     case truong_hop_2
          hanh_dong_2
     case truong_hop_3
          hanh_dong_3
     otherwise
          hanh_dong_n
end
```

Một câu lệnh ***switch*** bắt đầu với từ khóa ***switch*** và kết thúc với từ khóa ***end***. Trước tiên *biểu thức* trong ***switch*** sẽ được so sánh theo thứ tự với các giá trị của (*trường hợp 1, trường hợp 2, trường hợp 3…)*. Nếu giá trị của *biểu thức* trùng với giá trị của *trường hợp 1*, *hành động 1* sẽ được thực thi và sau đó kết thúc câu lệnh ***switch***. Nếu giá trị của *biểu thức* trùng với trường hợp *i*, *hành động i* sẽ được thực thi và kết thúc câu lệnh ***switch***. Nếu giá trị của *biểu thức* không trùng với trường hợp nào, *hành động n* phía dưới từ khóa *otherwise* sẽ được thực thi. Xét ví dụ sau đây nếu người sử dụng chỉ nhập vào từ bàn phím các giá trị 1, 3, 5 thì MATLAB sẽ in ra màn hình "một", "ba", "năm" tương ứng, trường hợp giá trị nhập vào không phải là 1, 3, 5 MATLAB sẽ hiển thị thông báo lỗi.

otherwisetest.m

```
%Vi du su dung otherwise trong switch de hien thi thong
bao loi
sonhap=input('Nhap vao mot trong cac gia tri 1, 3, 5:
');


switch sonhap
```

```
    case 1
        disp('So nhap vao la so mot!!');
    case 3
        disp('So nhap vao la so ba!!');
    case 5
        disp('So nhap vao la so nam!!')
    otherwise
        disp('So nhap vao khong dung, lam theo huong
dan lan sau!!')
end
```

7.3 Vòng lặp

Đối với bài toán tính diện tích hình tròn với bán kính cho trước như đã ví dụ ở trên, có thể không cần sử dụng đến M-file để lập trình mà có thể thực hiện phép tính đơn giản trong MATLAB: $S = \pi r^3$. Tuy nhiên, ví dụ như khi bài toán yêu cầu tính diện tích các hình tròn có bán kính lần lượt từ 0.1, 0.2, …99.9, 100cm cũng như khi gặp phải những bài toán yêu cầu thực hiện lại nhiều lần một phép tính, ta cần sử dụng đến vòng lặp.

Vòng lặp trong lập trình cho phép thực hiện lại nhiều lần một hay một nhóm các hành động. Có hai loại vòng lặp cơ bản thường gặp: vòng lặp giới hạn và vòng lặp có điều kiện. Vòng lặp giới hạn là vòng lặp mà trong đó các hành động được lặp lại với một số lần xác định trước. Vòng lặp có điều kiện cho phép lặp lại các hành động cho đến khi điều kiện không còn nhận giá trị **đúng**. Trong MATLAB hai loại vòng lặp trên tương ứng với các vòng lặp ***for*** (vòng lặp giới hạn) và ***while*** (vòng lặp có điều kiện).

7.3.1 FOR

Vòng lặp ***for*** thường sử dụng khi cần lặp lại nhiều lần một lệnh hay nhóm câu lệnh trong M-file văn bàn hay hàm và số lần lặp xác định trước, trong đó lần lặp thứ *n* được xác định bởi giá trị của **biến lặp**. Các biến lặp thường gặp trong các ngôn ngữ lập trình là *i, j, k, l…* tuy nhiên trong MATLAB do *i* và *j* đã được sử dụng để biểu diễn số phức nên cần lưu ý khi sử dụng *i* và *j* làm biến lặp.

Cú pháp cho vòng lặp ***for***:

```
for   bien_lap = khoang_lap
     hanh_dong
```

```
end
```

Trong đó *khoảng_lặp* chứa các giá trị từ nhỏ nhất đến lớn nhất của *biến_lặp*. *khoảng_lặp* có thể được định nghĩa bằng vector tuy nhiên thông thường *khoảng_lặp* được biểu diễn sử dụng dấu hai chấm **:**

Ví dụ: In một cột số nhận các giá trị từ 1 đến 5:

```
for i = 1 : 5
     fprintf('%d \n', i)
end
```

Đầu tiên biến lặp *i* sẽ được gán giá trị đầu tiên của *khoảng_lặp* là 1, sau đó *hành_động* của vòng lặp *for* sẽ được thực thi (ở đây là lệnh ***fprintf*** in ra giá trị của *i*, sau đó xuống dòng). Tiếp đó *i* sẽ được gán giá trị 2, MATLAB in giá trị tương ứng của *i* ra màn hình, xuống dòng… cho đến khi *hành_động* thực hiện xong với *i* nhận giá trị cuối cùng bằng 5, vòng lặp kết thúc.

Một trong những ứng dụng phổ biến của việc sử dụng vòng lặp ***for*** là tính tổng hoặc tích. MATLAB có hàm ***sum*** và ***prod*** để tính tổng và tích của các phần tử trong một vector, ta có thể sử dụng ***for*** để thực hiện công việc tương đương mà không cần dùng đến các hàm xây dựng sẵn. Ví dụ: Cho vector $A = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10]$, tính tổng và tích của tất cả các phần tử trong vector A.

```
>>A = 1:10;
>>sumA = 0;
>>prodA = 1;
>>for i = 1 : length(A)
     sumA = sumA + i;
     prodA = prodA * I;
end
```

So sánh *sumA* và *prodA* với kết quả các phép tính *sum(A)* và *prod(A)*.

Vòng lặp ***for*** cũng có thể kết hợp với câu lệnh lựa chọn ***if*** như trong ví dụ dưới đây: Chỉ tính tổng những phần tử trong A nhận giá trị nhỏ hơn 5 và tính tích của các phần tử còn lại của A.

```
>>sum5 = 0;
>>postrest = 1;
>>for i = 1 : length(A)
     if i < 5
          sum5 = sum5 + i;
     else
```

```
            postrest = postrest * i;
        end
    end
```

Như đã thấy ở ví dụ trên, *hành_ động* ở trong vòng lặp ***for*** có thể là bất kì câu lệnh hợp lệ nào, vì thế ta có thể nhiều vòng lặp ***for*** lồng trong nhau theo cú pháp sau:

```
    for   bien_lap_1 = khoang_lap_1
        %hanh dong 1 tuong duong voi viec khai bao them
vong lap
        for bien_lap_2 = khoang_lap_2
            hanh_dong_2
        end
    end
```

Các vòng lặp ***for*** lồng nhau thường gặp trong ma trận khi phải duyệt các phần tử theo hàng và cột. Ví dụ hãy in ra màn hình 15 ngôi sao xếp theo 3 hàng liên tiếp (mỗi hàng 5 ngôi)

```
    >>h = 3; c = 5;
    >>for i = 1 : h
        for j = 1 : c
            fprintf('*');
        end
        fprintf('\n');
    end
    *****
    *****
    *****
```

7.3.2 WHILE

Câu lệnh **while** là câu lệnh cho phép lặp có điều kiện trong MATLAB, được sử dụng để lặp lại một hành động khi không biết chính xác số lần lặp. Cú pháp tổng quát:

```
    while   dieu_kien
        hanh_dong
    end
```

Trước tiên *dieu_kien* sẽ được kiểm tra, nếu *dieu_kien* nhận giá trị logic **đúng**, *hanh_dong* sẽ được thực thi. Sau đó MATLAB thực hiện lại phép kiểm tra, nếu *dieu_kien* vẫn nhận giá trị **đúng**, *hanh_dong* lại tiếp tục được thực thi…Vòng lặp có thể trở thành vòng lặp vô cùng nếu *dieu_kien* không nhận giá trị **sai** như ví dụ sau đây:

```
>>dieu_kien=1;
>>while dieu_kien
fprintf('Su dung to hop phim Ctrl+C de thoat khoi vong lap\n')
end
```

Để tránh trường hợp vòng lặp vô cùng như trên, *hanh_dong* phải dẫn đến sự thay đổi trong *dieu_kien* để giá trị logic của nó trở thành **sai**. Ví dụ cho một chuỗi vô hạn các số sau: 1 2 6 24 … trong đó phần tử đầu tiên của chuỗi nhận giá trị 1, phần tử thứ hai nhận giá trị 2, phần tử thứ *n* nhận giá trị bằng tích của phần tử thứ *n-1* nhân với vị trí của phần tử đó trong chuỗi *n*. Viết hàm tìm phần tử đầu tiên của chuỗi lớn hơn giá trị tham số đầu vào của hàm đó.

```
timso.m
function a = timso(b)
%Tim so tra ve gia tri phan tu dau tien cua chuoi lon hon tham so dau vao cua ham
i = 1;
so = 1;
while so <= b
 i = i + 1;
 so = so * i;
end
a = so;
end
```

Ví dụ: Cho tham số đầu vào của hàm bằng 700.

```
>>timso(700)
ans =
     720
```

Trong chương trình này ta sử dụng hai biến *i* và *so*, trong đó *i* biểu diễn số thứ tự của phần tử trong chuỗi, *so* đại diện cho các phần tử trong chuỗi bắt đầu từ phần tử thứ nhất, *dieu_kien* được chọn cho hàm là phần tử thứ nhất nhỏ hơn giá trị đầu vào của hàm. Chạy vòng lặp **while** lần đầu tiên, *dieu_kien* vẫn nhận giá trị

đúng (1 < 700), *so* được gán giá trị của phần tử thứ hai trong chuỗi theo công thức (1 * 2 = 2),*dieu_kien* một lần nữa được kiểm tra và vẫn trả về giá trị **đúng**. Quá trình kiểm tra điều kiện và gán được lặp lại đến khi *so* nhận giá trị lớn hơn tham số đầu vào của hàm, chương trình kết thúc.

Một ví dụ khác về việc ứng dụng vòng lặp **while** đó là lặp lại việc nhập giá trị từ bàn phím theo định dạng cho trước.

```
% Chuong trinh yeu cau nguoi dung nhap mot so duong tu
ban phim,
% neu do la mot so duong, thuc thi vong lap bao gom cac
hanh dong: in so do ra
% man hinh va yeu cau nhap mot so duong khac cho den
khi so nhap vao am
inputnum=input('Nhap vao mot so duong: ');
while inputnum >= 0
fprintf('Ban da nhap vao so: %d\n\n',inputnum)
inputnum = input('Nhap vao mot so duong: ');
end
fprintf('So nhap vao khong hop le!\n')
```

7.4 Tóm tắt chương 7

Các phép toán quan hệ và các phép toán logic	
==	Bằng
~=	Khác
<	Nhỏ hơn
<=	Nhỏ hơn hoặc bằng
>	Lớn hơn
>=	Lớn hơn hoặc bằng
& hoặc &&	Logic AND
| hoặc ||	Logic OR
~	Logic NOT

Xor	Logic EXCLUSIVE OR
Các hàm điều khiển chương trình	
break	Ngắt vòng lặp
case	Câu lệnh rẽ nhánh trong cấu trúc **switch**
else	Câu lệnh rẽ nhánh trong cấu trúc **if**
elseif	Câu lệnh rẽ nhánh trong cấu trúc **if**
end	Kết thúc các cấu trúc lệnh **for, while, end**
error	Hiển thị thông báo lỗi
for	Lặp lại tập hợp lệnh theo một số lần cho trước
if	Thực hiện câu lệnh theo điều kiện
otherwise	Phần mặc định của cấu trúc **switch**
switch	Cấu trúc cho phép thực hiện lệnh khi so sánh với các điều kiện cho trước
warning	Hiển thị lời cảnh báo
while	Lặp lại tập hợp lệnh theo một số lần không xác định

Bảng 7. 1. Tóm tắt các hàm sử dụng trong chương 7

7.5 Bài tập chương 7

Bài 7.1

Viết hàm *sumstep* tính và trả về giá trị tổng của các số từ 1 đến *n* với khoảng cách giữa các số là *k*, trong đó *n* và *k* là tham số đầu vào của hàm. Kiểm tra kết quả bằng cách gọi hàm với *n = 11, k = 3* như sau

```
>>sumstep(11, 3)
```

Bài 7.2

Viết hàm *prodby* nhận vào giá trị của một số nguyên dương *n*, trả về giá trị tích của các số lẻ trong khoảng [1, n].

Bài 7.3

Viết chương trình *printE* yêu cầu nhập vào một vector, in ra theo thứ tự các phần tử của vector đó theo mẫu sau:

Nhap vao mot vector: vec

Phan tu thu 1 la

Phan tu thu 2 la

Phan tu thu 3 la

Phan tu thu 4 la

Bài 7.4

Tạo file *sp.txt* chứa ma trận 10x2 trong đó cột thứ nhất chứa các số thực ngẫu nhiên trong khoảng [2, 3], cột thứ hai chứa các số thực ngẫu nhiên trong khoảng [10, 11], các phần tử của ma trận lưu vào file với hai chữ số ở phần thập phân.

Bài 7.5

File *sp.txt* chứa các thông số về chiều dài *L* và cân nặng *W* của loạt sản phẩm cần kiểm tra. Trong đó các sản phẩm đạt yêu cầu có chiều dài trong khoảng [2.20, 2.79] và cân nặng trong khoảng [10.10, 10.89]. Viết chương trình kiểm tra thông số sản phẩm theo dữ liệu trong file *sp.txt*, in ra màn hình các sản phẩm bị loại theo mẫu:

STT L W

Bài 7.6

Phát triển chương trình của bài 7.5, nội dung in ra màn hình có dạng sau:

Danh sach cac san pham bi loai

****

STT L W

Ket_luan

Trong đó nội dung của Ket_luan phụ thuộc vào số sản phẩm bị loại, nếu không có sản phẩm nào hoặc tất cả các sản phẩm đều bị loại, dòng "STT L W" sẽ không in ra và Ket_luan có nội dung: "Khong co san pham nao bi loai!!!" hoặc "Tat ca cac san pham deu bi loai!!!", nếu số sản phẩm *n* bị loại trong khoảng (0, 10), in ra theo cấu trúc trên và Ket_luan có nội dung "Co n san pham bi loai!!!".

Bài 7.7

Tạo một vector A gồm mười phần tử là các số nguyên ngẫu nhiên trong khoảng [-10, 10]. Sử dụng vòng lặp và **if** (nếu cần) thực hiện các yêu cầu sau:

a. Trừ mỗi phần tử cho 3.
b. Kiểm tra xem vector có bao nhiêu phần tử nhận giá trị dương.
c. Tạo vector B có các phần tử là trị tuyệt đối của các phần tử tương ứng A.

d. Tìm phần tử lớn nhất, phần tử có giá trị tuyệt đối lớn nhất trong A.

Bài 7.8

Tạo ma trận ngẫu nhiên 3x5, các phần tử là các số nguyên trong khoảng [0, 10], sử dụng vòng lặp và **if** (nếu cần) thực hiện các yêu cầu sau:

a. Tìm phần tử lớn nhất trong mỗi cột.
b. Tìm phần tử lớn nhất trong mỗi hàng.
c. Tìm phần tử lớn nhất trong cả ma trận.

Bài 7.9

Viết chương trình in ra chính xác nội dung dưới đây, sử dụng vòng lặp.

1 x 8 + 1 = 9

12 x 8 + 2 = 98

123 x 8 + 3 = 987

1234 x 8 + 4 = 9876

12345 x 8 + 5 = 98765

123456 x 8 + 6 = 987654

1234567 x 8 + 7 = 9876543

12345678 x 8 + 8 = 98765432

123456789 x 8 + 9 = 987654321

Bài 7.10

Viết chương trình in ra chính xác nội dung dưới đây, sử dụng vòng lặp.

1

2 4

3 6 9

4 8 12 16

5 10 15 20 25

Bài 7.11

Viết chương trình tính thể tích hình nón trong đó yêu cầu người dùng nhập vào bán kính và chiều cao của hình nón, kiểm tra xem số nhập vào có phải là số dương hay không, tính và in ra thể tích hình nón theo công thức $V = \frac{\pi}{3} r^2 h$

Bài 7.12

Viết hàm *dtcv* yêu cầu nhập vào bán kính, trả về diện tích và chu vi đường tròn, báo lỗi nếu bán kính nhập vào có giá trị âm.

Bài 7.13

Viết hàm *recpol* nhận vào hai giá trị *x, y(x, y > 0)* trong tọa độ Đê các, trả về giá trị tọa độ cực *r, θ* theo công thức $r = \sqrt{(x^2 + y^2)}$, $\theta = \arctan(\frac{y}{x})$

Bài 7.14

Biết khoảng cách giữa hai điểm (x_1, y_1), (x_2, y_2) tính theo công thức:

$$d = \sqrt{(x - x_1)^2 + (y - y_1)^2}$$

Diện tích tam giác tính theo công thức:

$$A = \sqrt{s(s-a)(s-b)(s-c)}$$

Trong đó *a, b, c* là các cạnh của tam giác, *s* là nửa tổng của ba cạnh tam giác. Viết hàm tính khoảng cách của hai điểm bất kỳ theo công thức trên. Viết chương trình yêu cầu người dùng nhập vào tọa độ ba đỉnh của một tam giác, tính và in ra màn hình diện tích tam giác, trong đó sử dụng hàm vừa viết để tính độ dài các cạnh.

Bài 7.15

Cho ma trận *mat*

mat =

4	2	4	3	2
1	3	1	0	5
2	4	4	0	2

Viết hàm *sumprint* trong đoạn code sau để có nội dung in ra màn hình theo mẫu:

```
[r, c] = size(mat);
for i = 1 : r
   sumprint(mat(i, :))
end
```

để có nội dung in ra màn hình theo mẫu sau:

```
Tong bay gio la 15
Tong bay gio la 25
Tong bay gio la 37
```

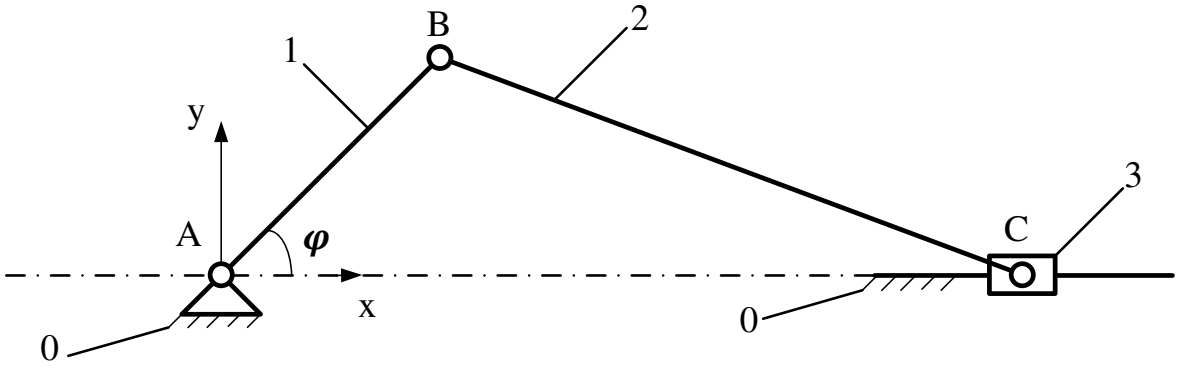
PHẦN II. MATLAB ỨNG DỤNG

Chương 8. PHÂN TÍCH ĐỘNG HỌC CƠ CẤU

Mục này giới thiệu các bài toán xác định vị trí khâu, khớp trong cơ cấu, khảo sát chuyển động của cơ cấu khi khâu dẫn quay theo một góc cho trước, khảo sát vận tốc, gia tốc tức thời của các khâu, khớp của cơ cấu. Các bài toán sẽ được hướng dẫn với ví dụ cơ cấu tay quay con trượt. Bên cạnh đó sinh viên có thể tham khảo bài tập với các cơ cấu khác có đáp số đi kèm.

8.1 Xác định vị trí các khâu, khớp trong cơ cấu

Cho cơ cấu tay quay con trượt như trong hình dưới trong đó AB = 0.5m, BC=1m. Khâu dẫn 1 tạo với phương ngang một góc $\varphi = \varphi_1 = 45^o$. Xác định vị trí các khớp B, C và góc tạo bởi khâu 2 và phương ngang.



Hình 8. 1. Cơ cấu tay quay con trượt

Hướng dẫn.

Khi bắt đầu một chương trình MATLAB, nên đưa vào những dòng lệnh sau để xóa tất cả các biến hiện hành, xóa nội dung trên cửa sổ lệnh, đóng các cửa sổ đồ họa đang mở để tránh ảnh hưởng đến kết quả tính toán cũng như dễ phát hiện lỗi trong quá trình lập trình.

```
clear all %xoa tat ca cac bien hien hanh
```

```
clc %xoa noi dung hien thi tren cua so lenh
close all %dong tat ca cua so do hoa dang mo
```

Khai báo các giá trị đầu vào, lưu ý số đo góc tạo bởi khâu 1 và phương ngang cần đổi ra radian.

```
%Khai bao tham so dau vao
AB = 0.5;
BC = 1;
phi1 = pi/4;
```

Vị trí khớp A

Hệ trục tọa độ xOy được đặt như hình vẽ, do A trùng với gốc tọa độ nên vị trí khớp A tương ứng là:

$$xA = 0; yA = 0;$$

```
%Vi tri khop A
xA = 0;
yA = 0;
```

Vị trí khớp B

Vị trí của khớp B có thể xác định qua hai ẩn xB và yB. Do độ dài AB cố định và góc tạo bởi khâu 1 và phương ngang đã biết, hai ẩn trên được tính theo công thức:

$$\text{xB} = \text{AB}\cos\varphi = (0.5)\cos 45^{o} = 0.353533\text{m}$$

$$\text{yB} = \text{AB}\sin\varphi = (0.5)\sin 45^{o} = 0.353533\text{m}$$

```
%Vi tri khop B
xB = AB*cos(phi1);
yB = AB*sin(phi1);
```

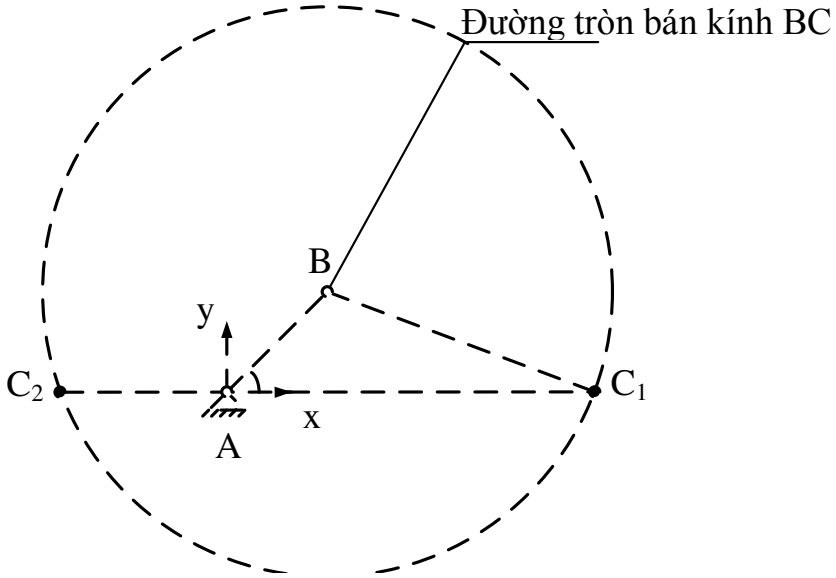
Vị trí khớp C

Vị trí của khớp C được xác định qua hai ẩn xC và yC, trong đó $yC = 0$. Do độ dài của BC không đổi ta có phương trình:

$$(xB - xC)^2 + (yB - yC)^2 = BC^2$$

Đây là phương trình bậc 2 có ẩn xC chưa biết. Phương trình này sẽ có hai nghiệm trong đó chỉ có một nghiệm là đáp án của bài toán. Yêu cầu đặt ra là tìm điều kiện để loại nghiệm của phương trình trên. Có thể thấy là trong trường hợp $0° \le \varphi \le 90°$, xC luôn lớn hơn xB. Phương trình để xác định tọa độ của C trong

MATLAB có thể giải bằng lệnh ***solve*** với ẩn khai báo là **xCn**, bước chọn nghiệm có thể sử dụng câu lệnh **ifelse** để giải như sau:



Hình 8. 2. Hai phương án giá trị của C

```
%Vi tri khop C
yC = 0;
ptC = '(xB-xCn)^2+(yB-yC)^2-BC^2';
nghiemC = solve(ptC,'xCn');
%Hai nghiem cua phuong trinh
xC1 = eval(nghiemC(1));
xC2 = eval(nghiemC(2));
%Tim C thoa man dieu kien xC > xB
if xC1 > xB
   xC = xC1;
else
   xC = xC2;
end
```

Sau khi chạy chương trình ta thu được hoành độ điểm C:

$$xC = 1.2890$$

Xác định góc tạo bởi khâu 2 với phương ngang

Sau khi xác định được tọa độ điểm C, góc φ_2 tạo bởi khâu 2 với phương ngang có thể xác định theo tọa độ hai điểm đầu cuối của khâu 2:

$$\varphi_2 = \arctan\frac{yB - yC}{xB - xC}$$

Các bước tính ◎2 được thực hiện trong MATLAB như sau:

```
%Tinh phi2
phi2r = atan((yB-yC)/(xB-xC));%Tinh theo radian
phi2 = phi2r*180/pi;%Tinh theo do
```

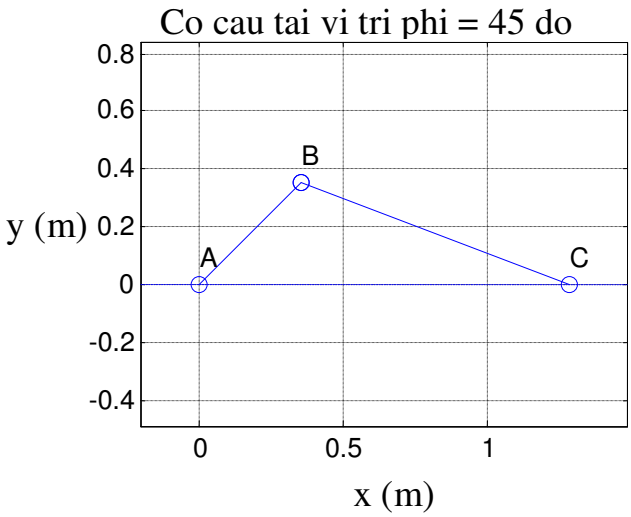
Kết quả thu được có thể được in ra màn hình theo các câu lệnh sau:

```
%In ket qua ra man hinh
fprintf('xB = %g (m) \n', xB)
fprintf('yB = %g (m) \n', yB)
fprintf('xC = %g (m) \n', xC)
fprintf('yC = %g (m) \n', yC)
fprintf('phi2 = %g (do) \n',phi2)
```

Vẽ lại cơ cấu

Cơ cấu có thể được vẽ lại trên MATLAB bằng cách sử dụng lệnh ***plot*** và các lệnh hỗ trợ khác như sau:

```
%Ve lai co cau
hold on
axis equal
title('Co cau tai vi tri phi = 45 do');
xlabel('x (m)');
ylabel('y (m)');
plot([xA-.2,xC+.2],[yA,yC],'--','LineWidth',2);
plot([xA,xB],[yA,yB],'-o','LineWidth',2);
plot([xB,xC],[yB,yC],'-o','LineWidth',2);
text(xA,yA+.1,'A');
text(xB,yB+.1,'B');
text(xC,yC+.1,'C');
```



Hình 8. 3. Biểu diễn lại vị trí của cơ cấu

8.2 Khảo sát chuyển động và tạo phim mô phỏng chuyển động của cơ cấu

8.2.1 Khảo sát chuyển động

Để khảo sát chuyển động của cơ cấu khi khâu dẫn 1 quay trọn vẹn một vòng, có thể sử dụng vòng lặp để tính toán vị trí các khớp tại các góc tương ứng. Khi lập trình cần chú ý điều kiện để chọn nghiệm đảm bảo điều kiện đơn giản dễ thực hiện nhưng có tính chính xác cao. Xét ví dụ khi khâu dẫn 1 của cơ cấu tay quay con trượt quay trọn vẹn một vòng $0° \le \varphi \le 360°$ có thể thấy xB nhận các giá trị dương khi $0° \le \varphi \le 90°$ hoặc $270° \le \varphi \le 360°$ và nhận giá trị âm với các giá trị còn lại của φ. Tuy nhiên ta cũng có thể sử dụng một điều kiện khác đơn giản hơn đó là với mọi giá trị của φ, *xB* luôn nhỏ hơn *xC*.

Trong ví dụ này, ta sử dụng vòng lặp ***for*** với bước nhảy của vòng lặp là *pi/4*. Để tiện cho việc quan sát đồ thị, sinh viên có thể thay thế bằng các bước nhảy có giá trị nhỏ hơn.

```
%Khao sat chuyen dong cua co cau
for phi=0:pi/4:2*pi
    xB = AB*cos(phi);
    yB = AB*sin(phi);
    yC = 0;
    ptC = '(xB-xCn)^2+(yB-yC)^2-BC^2';
    nghiemC = solve(ptC,'xCn');
%Hai nghiem cua phuong trinh
```

```
    xC1 = eval(nghiemC(1));
    xC2 = eval(nghiemC(2));
%Tim C thoa man dieu kien xC > xB
    if xC1 > xB
        xC = xC1;
    else
        xC = xC2;
    end
    hold on
    axis equal
    plot([xA,xB],[yA,yB],'-o','LineWidth',2);
    plot([xB,xC],[yB,yC],'-o','LineWidth',2);
    text(xA,yA+.1,'A');
    text(xB,yB+.12,'B');
    text(xC,yC-.08,'C');
end
title('Khao sat chuyen dong cua co cau')
xlabel('x(m)')
ylabel('y(m)')
```

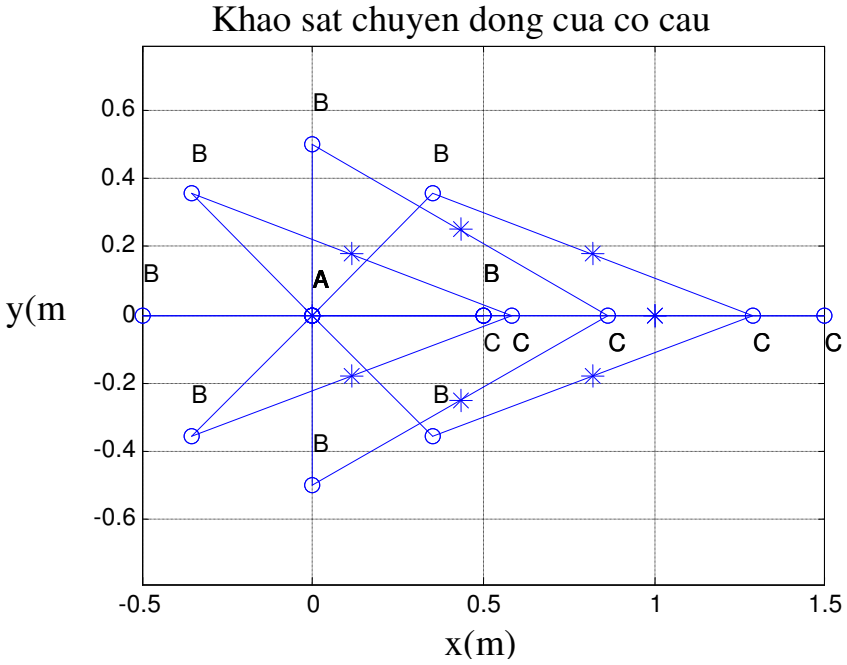
Khi đã xác định được vị trí các khớp của cơ cấu trong chuyển động, ta cũng có thể suy ra quỹ đạo của một điểm bất kỳ trên khâu trong chuyển động đó. Ví dụ xét điểm P là trọng tâm của thanh BC, tại mọi thời điểm khâu chuyển động, tọa độ của P được tính tương ứng theo tọa độ của hai đầu B, C là:

$$x_P(\varphi) = x_B(\varphi) + x_C(\varphi)$$

$$y_P(\varphi) = y_B(\varphi) + y_C(\varphi)$$

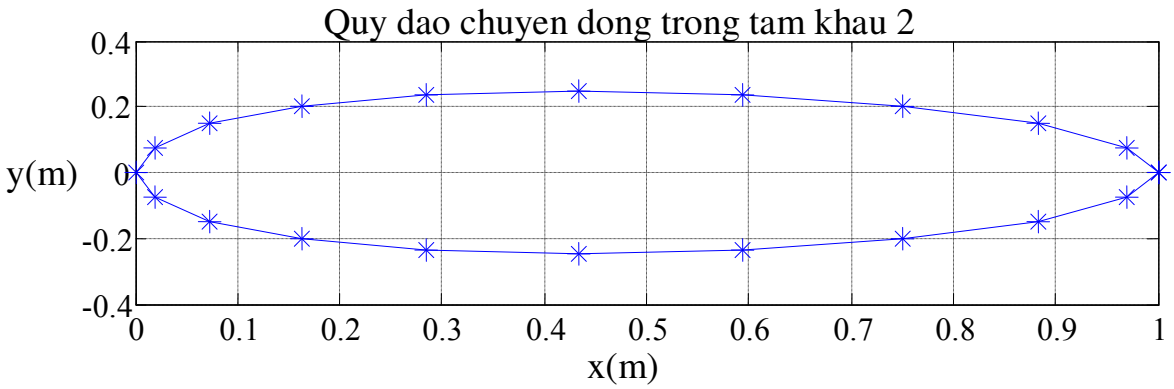
Trong vòng lặp, ta có thể thêm vào một số dòng lệnh để tính tọa độ trọng tâm P và lưu các tọa độ này vào một mảng giá trị.

```
i = i+1; %i la bien chay nhan gia tri 0 o ngoai vong
lap
xP(i) = (xB + xC)/2;
yP(i) = (yB + yC)/2;
plot(xP(i),yP(i),'*','MarkerSize',10);
```



Hình 8. 4. Khảo sát chuyển động của cơ cấu

Tọa độ trọng tâm P của khâu BC trong chuyển động của cơ cấu có thể được thể hiện riêng như sau (hình vẽ được thể hiện với bước nhảy vòng lặp là *pi/10*):



Hình 8. 5. Quỹ đạo chuyển động của trọng tâm khâu BC

```
plot(xP,yP,'*-','MarkerSize',10);
axis([0 1 -.4 .4]);
title('Quy dao chuyen dong trong tam khau 2');
xlabel('x(m)');
ylabel('y(m)');
```

8.2.2 Tạo clip mô phỏng chuyển động

Một trong những cách thông dụng để tạo những đoạn phim ngắn trong MATLAB là sử dụng hàm ***getframe*** để chụp lại nội dung thể hiện trên cửa sổ đồ họa hiện hành rồi sau đó sử dụng hàm ***movie*** để cho các khung hình đã được chụp lại lần lượt thể hiện ra trên màn hình tạo nên chuyển động.

Hàm ***getframe*** thường được sử dụng trong vòng lặp, giá trị trả về của hàm là vector một hàng, vì thế cả hai cú pháp gọi hàm dưới đây đều hợp lệ:

```
>>M(i) = getframe;
```

hoặc

```
M(:, j) = getframe;
```

Trong đó M là vector chứa các khung hình của phim thể hiện chuyển động, *i*, *j* là số thứ tự của khung hình trong vector.

Một số lưu ý khi sử dụng hàm ***getframe***

Đảm bảo hệ trục tọa độ ở các khung hình khác nhau là không đổi, nếu không các hình vẽ sẽ được thể hiện trên các hệ trục tọa độ có giới hạn cũng như độ chia khác nhau dẫn đến kết quả của phim sẽ khó theo dõi.

Sử dụng dấu ***;*** ở cuối lệnh ***getframe*** để không hiển thị các giá trị thành phần của ***M*** khi chạy vòng lặp.

Khi vòng lặp đang được chạy, bảo đảm cửa sổ đồ họa hiện hành không bị che khuất bởi các cửa sổ khác (cửa sổ lệnh, cửa sổ đồ họa khác…) nếu không nội dung của các cửa sổ đồ họa này cũng sẽ được lưu vào các khung hình của phim.

Đối với cơ cấu tay quay con trượt trong ví dụ này, ở ngoài vòng lặp *%khao sat chuyen dong cua co cau* đã trình bày ở trên có thể tạo một biến chạy *incr = 1* ngoài vòng lặp, và thêm vào bên trong vòng lặp phía dưới các lệnh vẽ dòng lệnh sau:

```
axis ([-1.5 1.5 -.6 .6]);
M(incr) = getframe;
close;
incr = incr +1;
```

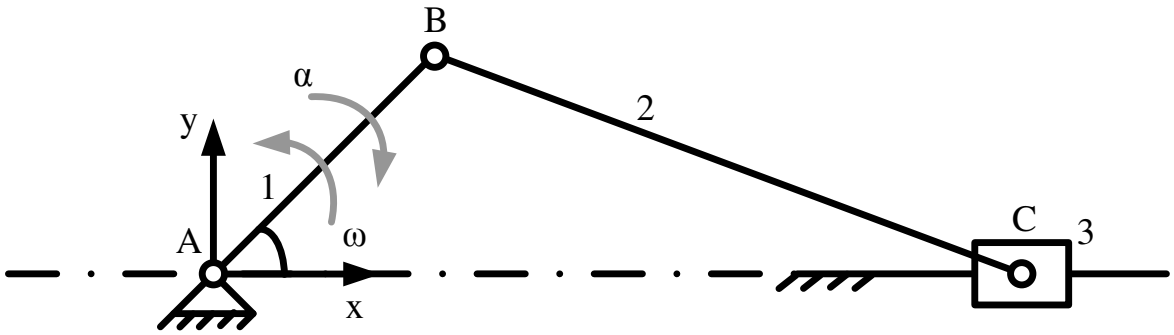
Sau khi biến M đã lưu các khung hình của phim thể hiện chuyển động của cơ cấu, hàm ***movie*** có thể được sử dụng để trình chiếu lại đoạn phim theo cú pháp sau

>>movie(M, n)

Trong đó *n* là số lần người sử dụng muốn chương trình lặp lại đoạn phim, ví dụ với *n = 10,* chương trình sẽ chiếu đoạn phim lặp đi lặp lại 10 lần.

8.3 Vận tốc và gia tốc

Đối với bài toán tay quay con trượt đang xét, vận tốc góc và gia tốc góc tức thời của khâu 1 lần lượt là $\omega = 1$ rad/s, $\alpha = -1$ rad/s^2, tính vận tốc và gia tốc dài của các khớp B, C, tính vận tốc góc và gia tốc góc của khâu 2.



Hình 8. 6. Bài toán vận tốc, gia tốc với cơ cấu tay quay con trượt

Vận tốc của khớp B

Vận tốc của điểm B trên khâu 1 có thể được tính như sau:

$$v_B = v_{B_1} = v_A + v_{BA} = v_A + \omega \times r_{BA}$$

Trong đó v_{B_1} là vận tốc của điểm B trên khâu 1, v_{BA} là vận tốc của B trong chuyển động quay tương đối so với điểm A, ω là vận tốc góc của khâu 1, r_{BA} được tính theo công thức $r_{BA} = r_B - r_A$ trong đó $r_A = \begin{bmatrix} x_A & y_A & 0 \end{bmatrix}^T$, $r_B = \begin{bmatrix} x_B & y_B & 0 \end{bmatrix}^T$, r_A và r_B lần lượt là tọa độ của A, B

Vận tốc khớp C

Do điểm C thuộc hai khâu 2 và 3 nên $v_{C_2} = v_{C_3}$

Mặt khác do khâu 3 chỉ trượt theo phương ngang nên $v_{Cy} = 0$, do đó vận tốc điểm C có thể được biểu diễn theo $v_C = v_{C_2} = v_{C_3} = \begin{bmatrix} v_{Cx} & 0 & 0 \end{bmatrix}^T$

Vận tốc điểm C tính theo phương trình

$$v_{C_2} = v_B + v_{CB} = v_B + \omega_2 \times r_{CB} = v_B + \omega_2 \times (r_C - r_B)$$

$$\begin{bmatrix} v_{Cx} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} v_{Bx} \\ v_{By} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega_2 \end{bmatrix} \times \begin{bmatrix} x_C - x_B \\ y_C - y_B \\ 0 \end{bmatrix}$$

Qua đó ta có hai phương trình với hai ẩn là v_{Cx} và ω_2 chưa biết:

$$v_{Cx} = v_{Bx} - \omega_2(y_C - y_B)$$
$$0 = v_{By} + \omega_2(x_C - x_B)$$

Sau khi tìm được ω_2, vận tốc quay tương đối giữa C và B có thể được tính theo công thức:

$$v_{CB} = \omega_2 \times r_{CB}$$

Để giải bài toán trong MATLAB trước tiên ta cần khai báo các biến r_A, r_B, r_C theo tọa độ x_A, x_B, x_C và các biến ký hiệu *vCx* và *omega2z*.

```
rA=[xA yA 0]';
rB=[xB yB 0]';
rC=[xC yC 0]';
omega1=[0 0 1]'; %rad/s
syms vCx omega2z real;
vC=[vCx 0 0]'; %m/s
omega2=[0 0 omega2z]'; %rad/s
%Tinh van toc diem B
vB=cross(omega1,rB-rA); %m/s
%Tinh van toc diem C
%Lap phuong trinh
ptvC=vC - vB - cross(omega2,rC-rB);
%Tach ra hai phuong trinh chua cac an vCx va omega2z
ptvC1=ptvC(1);
ptvC2=ptvC(2);
nv=solve(ptvC1,ptvC2);
vCxs=eval(nv.vCx);
omega2s=eval(nv.omega2z);
vC=[vCxs 0 0]'; %m/s
omega2=[0 0 omega2s]'; %rad/s
%Tinh van toc quay tuong doi giua C va B
vCB=cross(omega2, rC-rB); %m/s
```

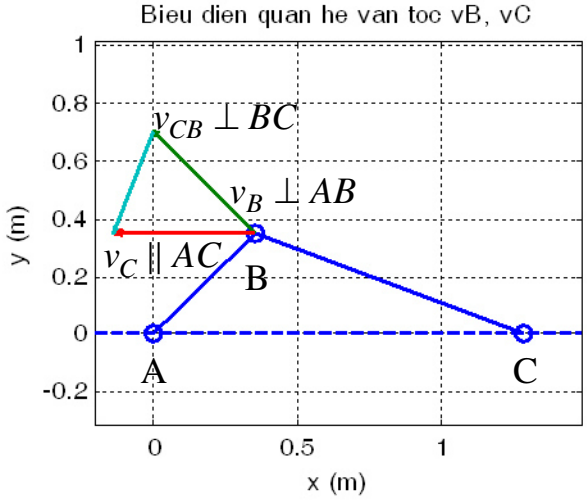
Biểu diễn họa đồ vận tốc trong MATLAB

Trong MATLAB, người sử dụng có thể biểu diễn các vector sử dụng hàm ***quiver*** theo cú pháp sau:

```
>>quiver(x, y, u, v, S)
```

Trong đó x, y là điểm đặt của gốc vector; u, v là độ dài vector chiếu xuống trục hoành và trục tung, S là tỉ lệ xích biểu diễn vector trên cửa sổ đồ họa. Đối với ví dụ tay quay con trượt này, họa độ vận tốc có thể được biểu diễn như sau:

```
hold on
axis equal %dam bao the hien quan he vuong goc tren
hinh
title('Bieu dien quan he van toc vB, vC');
xlabel('x (m)');
ylabel('y (m)');
plot([xA-.2,xC+.2],[yA,yC],'--','LineWidth',2);
plot([xA,xB],[yA,yB],'-o','LineWidth',2);
plot([xB,xC],[yB,yC],'-o','LineWidth',2);
%Ve vector vB, vC co goc dat tai diem B va C tuong ung
quiver(xB,yB,vB(1),vB(2),1,'LineWidth',2);
quiver(xB,yB,vC(1),vC(2),1,'LineWidth',2);
%Ve vector vBC co goc trung voi ngon cua vector vB
quiver(xB+vB(1),yB+vB(2),vCB(1),vCB(2),1,'LineWidth',2)
;
```



Hình 8. 7. Họa đồ vận tốc của cơ cấu

Gia tốc khớp B

Gia tốc điểm B trên khâu 1 tính theo công thức:

$$a_B = a_{B_1} = a_{B_2} = a_A + \alpha_1 \times r_{BA} + \omega_1 \times (\omega_1 \times r_{BA})$$

Với hai thành phần tiếp tuyến a_B^t (vuông góc với BA và có hướng xác định theo hướng của α và thành phần pháp tuyến a_B^n (song song với BA và hướng từ B đến tâm quay A)

Các bước tính gia tốc khớp B trong MATLAB trình bày như sau:

```
alpha1=[0 0 -1]'; %rad/s^2
aA=[0 0 0]'; %m/s^2
aB=cross(alpha1,rB-rA)+cross(omega1,cross(omega1,rB-
rA));
aBt=cross(alpha1,rB-rA);
aBn=cross(omega1,cross(omega1,rB-rA));
```

Gia tốc khớp C

Gia tốc điểm C có thể tính được khi khảo sát chuyển động trên khâu 2:

$$a_C = a_{C_2} = a_{B_2} + \alpha_2 \times r_{CB} + \omega_2 \times (\omega_2 \times r_{CB})$$

Trong đó gia tốc góc α_2 của khâu 2 là ẩn số chưa biết, do khâu 3 chỉ chuyển động trượt theo phương ngang:

$$a_C = a_{C_2} = \begin{bmatrix} a_{Cx} & 0 & 0 \end{bmatrix}^T$$

Biểu diễn phương trình tính gia tốc điểm C dưới dạng vector:

$$\begin{bmatrix} a_{Cx} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} a_{Bx} \\ a_{By} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \alpha_2 \end{bmatrix} \times \begin{bmatrix} xC - xB \\ yC - yB \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega_2 \end{bmatrix} \times \left(\begin{bmatrix} 0 \\ 0 \\ \omega_2 \end{bmatrix} \times \begin{bmatrix} xC - xB \\ yC - yB \\ 0 \end{bmatrix} \right)$$

Ta được hệ hai phương trình với hai ẩn a_{Cx} và α_2:

$$a_{Cx} = a_{Bx} - \alpha_2 (yC - yB) - \omega_2^2 (xC - xB)$$

$$0 = a_{By} + \alpha_2 (xC - xB) - \omega_2^2 (yC - yB)$$

Sau khi tính được α_2 ta có thể suy ra hai thành phần tiếp tuyến và pháp tuyến của gia tốc tương đối giữa C và B:

$$a_{CB}^t = \alpha_2 \times r_{CB}$$

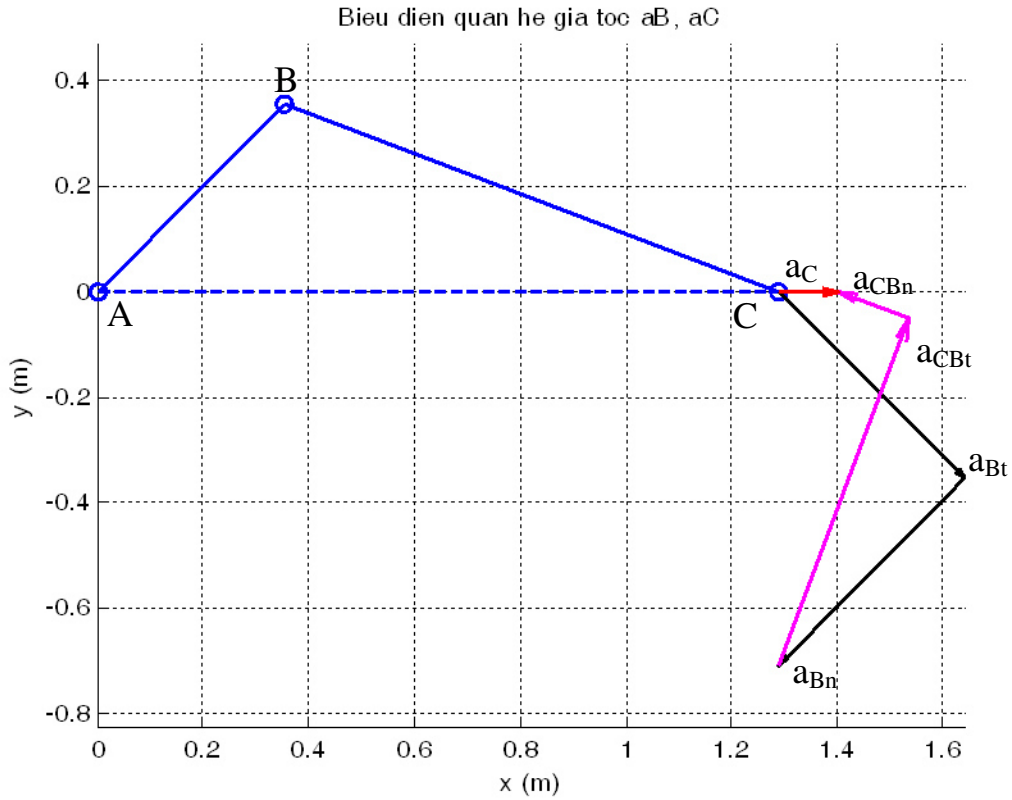
$$a_{CB}^n = \omega_2 \times (\omega_2 \times r_{CB})$$

Các bước tính toán trong MATLAB được thực hiện như sau:

```
syms alpha2zaCxreal;%Khai bao cac an aCx va alpha2z
alpha2=[0 0 alpha2z]';
aC=[aCx 0 0]';
%Lap phuong trinh
ptaC=aC-aB-cross(alpha2,rC-rB)-
cross(omega2,cross(omega2,rC-rB));
%Tach hai phuong trinh chua aCx va alpha2z
ptaC1=ptaC(1);
ptaC2=ptaC(2);
na=solve(ptaC1,ptaC2);
aCxs=eval(na.aCx);
alpha2s=eval(na.alpha2z);
aC=[aCxs 0 0]';
alpha2=[0 0 alpha2s]';
aCBt=cross(alpha2,rC-rB);
aCBn=cross(omega2,cross(omega2,rC-rB));
```

Biểu diễn họa đồ gia tốc trong MATLAB

Họa đồ gia tốc biểu diễn mối quan hệ giữa gia tốc của điểm B và gia tốc của điểm C có thể được biểu diễn như sau:



Hình 8. 8. Họa đồ gia tốc của cơ cấu

```
hold on
axis equal
title('Bieu dien quan he gia toc aB, aC');
xlabel('x (m)');
ylabel('y (m)');
%Ve lai co cau
plot([xA,xC],[yA,yC],'--','LineWidth',2);
plot([xA,xB],[yA,yB],'-o','LineWidth',2);
plot([xB,xC],[yB,yC],'-o','LineWidth',2);
%Ve vector aBt tai diem C
quiver(xC,yC,aBt(1),aBt(2),1,'k','LineWidth',2);
%Ve vector aBn tai ngon cua vector aBt
xs=xC+aBt(1);
ys=yC+aBt(2);
quiver(xs,ys,aBn(1),aBn(2),1,'k','LineWidth',2);
```

```
%Ve vector aCBt tai ngon cua vector aBn
xs=xs+aBn(1);
ys=ys+aBn(2);
quiver(xs,ys,aCBt(1),aCBt(2),1,'m','LineWidth',2);
%Ve vector aCBn tai ngon cua vector aCBt
xs=xs+aCBt(1);
ys=ys+aCBt(2);
quiver(xs,ys,aCBn(1),aCBn(2),1,'m','LineWidth',2,'MaxHeadSi
ze',.8);
%Ve vector aC tai C
quiver(xC,yC,aC(1),aC(2),1,'r','LineWidth',2,'MaxHeadSize',
.8);
```

8.4 Bài tập chương 8

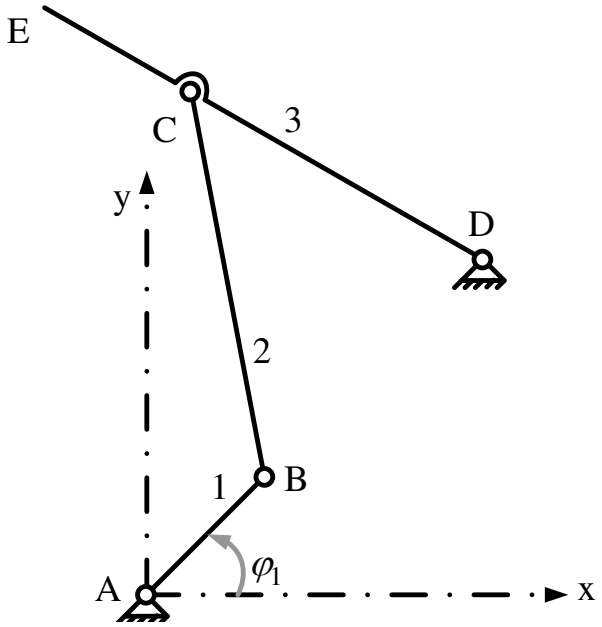
Bài 8.1 Cơ cấu 4 khớp quay

Cho cơ cấu như hình vẽ.

Các thông số của cơ cấu được cho như sau: AB = 0.15m, BC = 0.35m, CD=0.30m, CE = 0.15m, xA = yA = 0, xD = yD = 0.30m, $\varphi_1 = 45°$, khâu dẫn 1 quay với vận tốc góc không đổi 60 vòng/phút.

Tính xB, yB, xC, yC, xE, yE, φ_2, φ_3

Tính vận tốc, gia tốc các khâu, khớp tại thời điểm $\varphi_1 = 45°$



Hình 8. 9. Cơ cấu 4 khớp quay

Đáp án

$$rB = \begin{bmatrix} 0.106066 & 0.106066 & 0 \end{bmatrix}^T m$$

$$rC = \begin{bmatrix} 0.0400698 & 0.449788 & 0 \end{bmatrix}^T m$$

$$rE = \begin{bmatrix} -0.0898952 & 0.524681 & 0 \end{bmatrix}^T m$$

$$\varphi_2 = \text{-}79.1312\ ^\circ$$

$$\varphi_3 = \text{-}29.9532\ ^\circ$$

$$\omega_1 = \begin{bmatrix} 0 & 0 & 6.28319 \end{bmatrix}^T rad/s$$

$$\omega_2 = \begin{bmatrix} 0 & 0 & -3.43639 \end{bmatrix}^T rad/s$$

$$\omega_3 = \begin{bmatrix} 0 & 0 & -3.43639 \end{bmatrix}^T rad/s$$

$$\alpha_1 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T rad/s^2$$

$$\alpha_2 = \begin{bmatrix} 0 & 0 & -8.97883 \end{bmatrix}^T rad/s^2$$

$$\alpha_3 = \begin{bmatrix} 0 & 0 & 22.6402 \end{bmatrix}^T rad/s^2$$

$$v_B = v_{B_1} = v_{B_2} = \begin{bmatrix} -0.666432 & 0.666432 & 0 \end{bmatrix}^T m/s$$

$$v_C = \begin{bmatrix} 0.514728 & 0.893221 & 0 \end{bmatrix}^T m/s$$

$$v_E = \begin{bmatrix} 0.772092 & 1.33983 & 0 \end{bmatrix}^T m/s$$

$$a_B = a_{B_1} = a_{B_2} = \begin{bmatrix} -4.18732 & -4.18732 & 0 \end{bmatrix}^T m/s^2$$

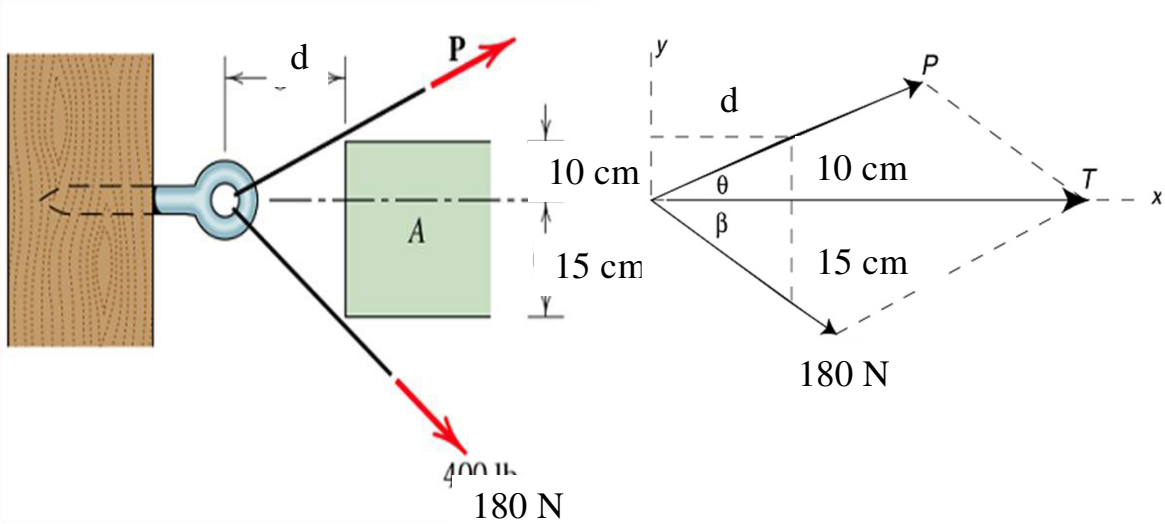
$$a_C = \begin{bmatrix} -0.321767 & -7.65368 & 0 \end{bmatrix}^T m/s^2$$

$$a_E = \begin{bmatrix} -0.48265 & -11.4805 & 0 \end{bmatrix}^T m/s^2$$

Một số bài tập ứng dụng MATLAB trong môn cơ học lý thuyết

Bài 8.2

Để rút một chiếc đinh ra khỏi gỗ cần tác dụng một lực theo phương ngang, do bị vướng vật cản A nên cần buộc cáp vào đinh và tác dụng một lực 180 N và một lực P như hình vẽ. Xác định P để hợp lực T tác dụng lên đinh chỉ theo phương ngang, đồng thời tính hợp lực T. Vẽ đồ thị biểu diễn P và T theo d với d nằm trong khoảng từ 5cm đến 60cm



Hình 8. 10. Hình đề bài 8.2

Đáp án:

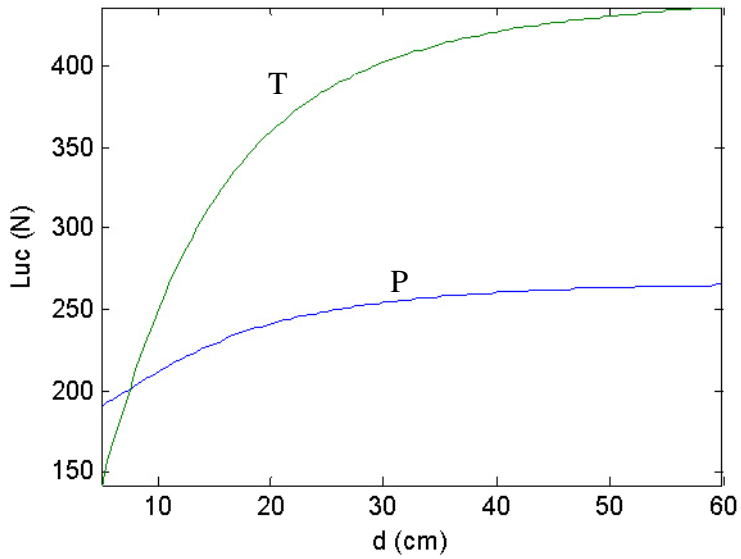
Các lực kéo dây cáp và hợp lực T biểu diễn như hình trên. Phương trình cân bằng lực có thể viết như sau:

$$T = \sum F_x = P\cos\theta + 180\cos\beta$$

$$0 = \sum F_y = P\sin\theta - 180\sin\beta$$

Các bước tính trong MATLAB:

```
>>syms d P T
>>theta = atan(10/d);
>>beta = atan(15/d);
>> pt1='P*cos(theta)+180*cos(beta)-T';
>> pt2='P*sin(theta)-180*sin(beta)';
>> ds=solve(pt1,pt2,'P,T');
>>ds.P
ans =
180*sin(beta)/sin(theta)
>>ds.T
ans =
180*(sin(beta)*cos(theta)+cos(beta)*sin(theta))/sin(thet
a)
```



Hình 8. 11. Đồ thị bài 8.2

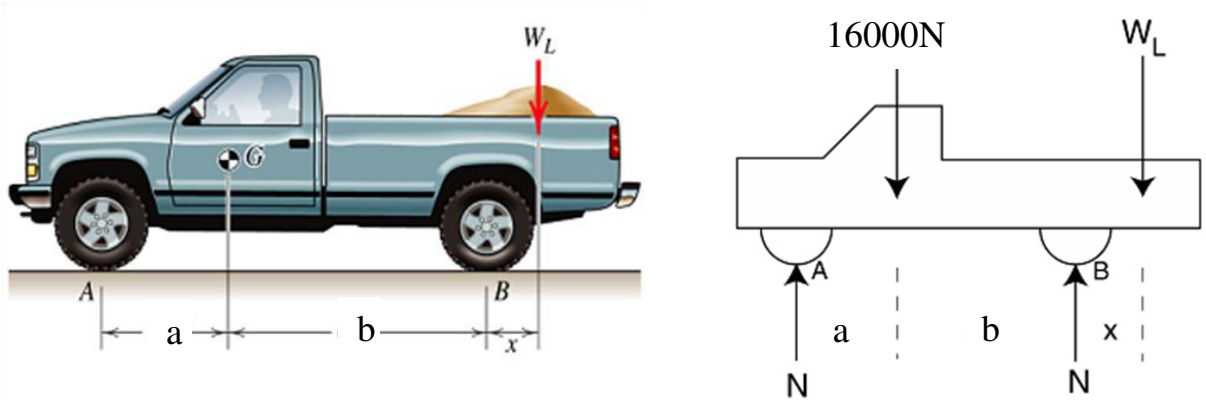
Vẽ đồ thị của P và T theo d

```
>>d = 5: .5: 60;
>> beta=atan(15./d);
>> theta=atan(10./d);
>>P = 180*sin(beta)./sin(theta);
```

```
>>T =
180*(sin(beta).*cos(theta)+cos(beta).*sin(theta))./sin(t
heta);
>>plot(d, P, d, T);
>>xlabel('d (cm)'); ylabel('Luc (N)');
```

Bài 8.3

Trong điều kiện không tải, xe có trọng lượng 16000N đặt tại trọng tâm G. Đặt tải W_L có trọng tâm ở phía sau của bánh xe sau một đoạn là x (cm). Tìm mối quan hệ giữa W_L và x nếu biết phản lực lên bánh trước và bánh sau là bằng nhau. Vẽ đồ thị biểu diễnW_L theo x với x nằm trong khoảng 0 đến 100 cm. Biết a = 115 cm, b = 170 cm.



Hình 8. 12. Hình bài 8.3

Đáp án:

Phương trình cân bằng moment và phương trình cân bằng lực tác dụng lên xe có thể viết như sau:

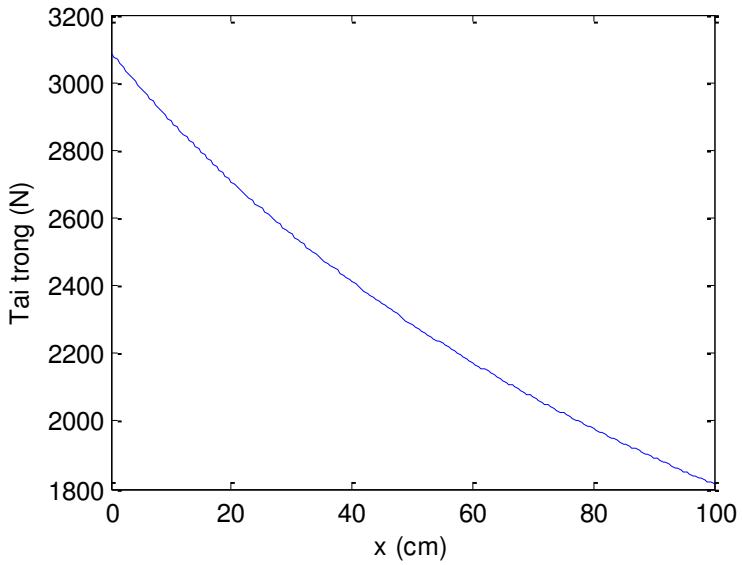
$$\sum M_B = 0 = 16000.170 - N.285 - W_L x$$

$$\sum F_y = 0 = N + N - 16000 - W_L$$

Các bước giải trong MATLAB như sau:

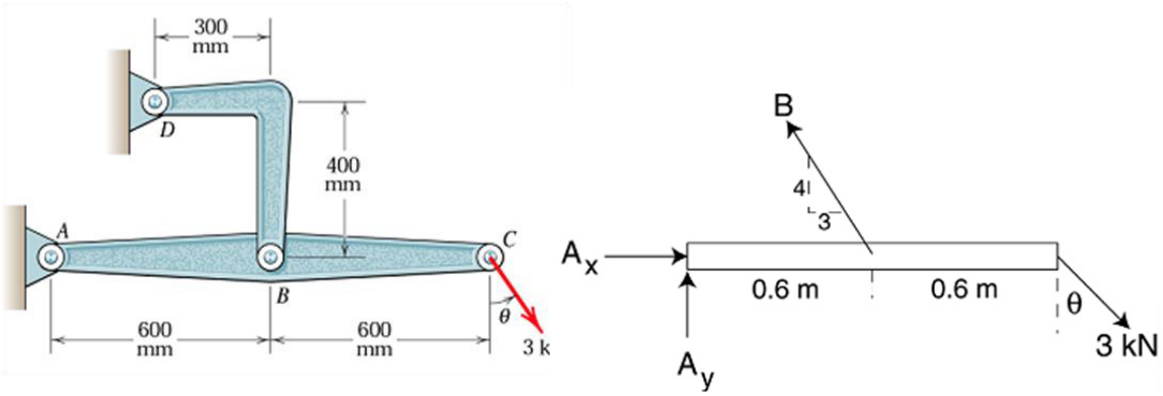
```
>>syms x N W
>> pt1 = '16000*170 - N*285 - W*x';
>> pt2 = 'N + N - 16000 - W';
>> ds=solve(pt1, pt2, 'W,N');
>>ds.W
```

```
ans =
880000/(285+2*x)
>>x = 0: .5: 100;
>>W = 880000./(285 + 2*x);
>>plot(x,W);
>> ylabel('Tai trong (N)'); xlabel('x (cm)');
```



Hình 8. 13. Đồ thị bài 8.3

Bài 8.4



Hình 8. 14. Hình bài 8.4

của góc θ từ -90° đến 90°. Tính các lực tại A và B theo θ, xác định giá trị cực đại của các lực này với góc θ tương ứng.

Đáp án:

Phương trình cân bằng lực và moment cho hệ viết như sau:

$$\sum M_A = \frac{4}{5}B.(0.6) - 3\cos\theta(1.2) = 0$$

$$\sum F_x = A_x - \frac{3}{5}B + 3\sin\theta = 0$$

$$\sum F_y = A_y + \frac{4}{5}B - 3\cos\theta = 0$$

Các bước tính trong MATLAB như sau:

```
>> syms Ax Ay B theta
>> pt1='0.8*B*0.6-3*cos(theta)*1.2';
>> pt2='Ax-0.6*B+3*sin(theta)';
>> pt3='Ay+0.8*B-3*cos(theta)';
>> ds=solve(pt1,pt2,pt3,'Ax,Ay,B');
```

$$Ax = \frac{9}{2}\cos\theta - 3\sin\theta$$

$$Ay = -3\cos\theta$$

$$B = \frac{15}{2}\cos\theta$$

Vẽ đồ thị của B, A theo θ

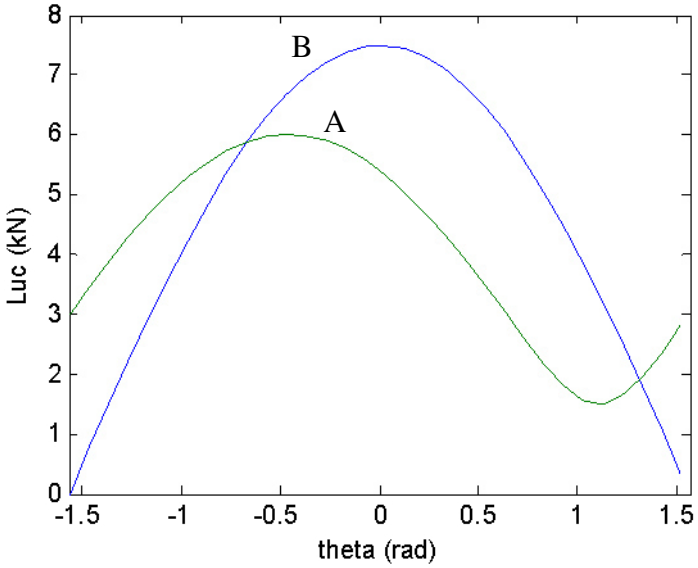
```
>> theta=-pi/2:0.1:pi/2;
>> B=15/2*cos(theta);
>> Ax=9/2*cos(theta)-3*sin(theta);
>> Ay=-3*cos(theta);
>> A=sqrt(Ax.^2+Ay.^2);
>> plot(theta,B,theta,A)
>> xlabel('theta (rad)');
>> ylabel('Luc (kN)');
```

Từ giá trị $B = \frac{15}{2}\cos\theta$ có thể dễ dàng thấy được B đạt cực đại khi $\theta = 0$

Giá trị cực đại của A có thể được tìm như sau: tìm các giá trị của θ sao cho đạo hàm của A nhận giá trị 0, kết hợp với đồ thị để suy ra θ cần tìm.

```
>> fA=sqrt(ds.Ax^2+ds.Ay^2);
>> dfA=diff(fA);
>> ntheta=solve(dfA)
```

```
ntheta =
  2.67794504458898712224838715181 83
 -.46364760900080611621425623146121
 -2.03444393579570273544557792310 10
  1.10714871779409050301706546017 85
```



Hình 8. 15. Đồ thị bài 8.4

Quan sát đồ thị ta có thể thấy được A đạt cực đại tại lân cận của $\theta = 0.5$

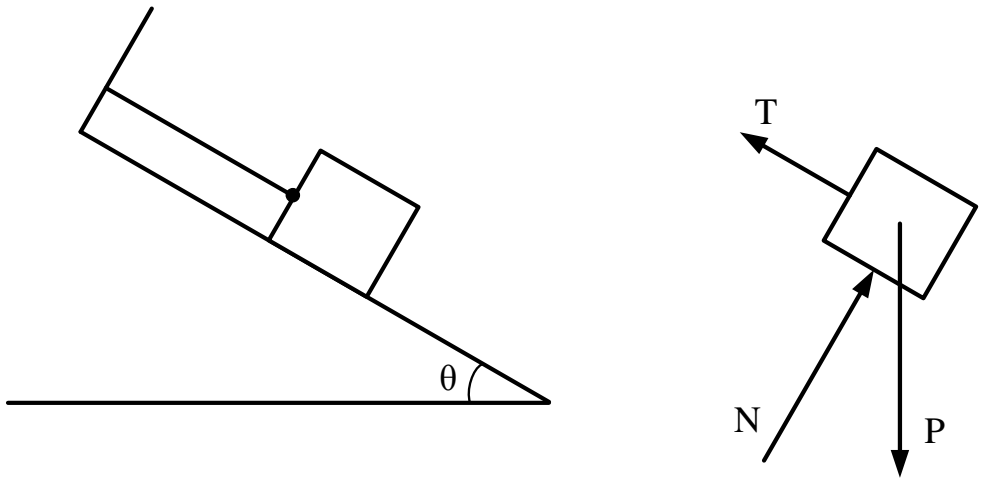
```
>> Amax=subs(fA,ntheta(2))
Amax =
6.0000
>> thetaA=double(ntheta(2)*180/pi)
thetaA =
-26.5651
>> Bmax=subs(ds.B,0)
Bmax =
    7.5000
```

Bài 8.5

Một vật nặng được giữ cân bằng trên mặt phẳng nghiêng một góc θ so với phương ngang. Tính lực căng T và phản lực N của đất tác dụng lên vật theo θ. Với giá trị nào của θ thì T = N. Biết vật có khối lượng 100 kg.

Đáp án:

Các lực tác dụng lên vật được biểu diễn như hình dưới. Phương trình cân bằng lực chiếu lên các mặt phẳng song song và vuông góc với mặt phẳng nghiêng lần lượt có dạng:



Hình 8. 16. Hình bài 8.5

$$\sum F_i = P\sin\theta - T = 0$$

$$\sum F_k = N - P\cos\theta = 0$$

Các bước giải trong MATLAB như sau:

```
>> syms P N T theta
>> pt1='P*sin(theta)-T';
>> pt2='N-P*cos(theta)';
>> ds=solve(pt1,pt2,'N,T');
```

Tính giá trị của T và N tại $\theta = 30°$

```
>> N=subs(ds.N,{P,theta},{100,pi/6});
>> T=subs(ds.T,{P,theta},{100,pi/6});
```

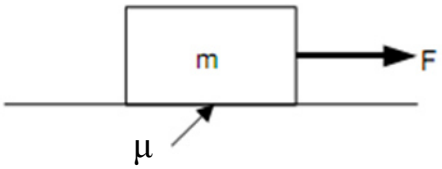
Tìm giá trị của θ để T = N:

```
>> pt3=ds.N-ds.T;
>> ntheta=solve(pt3,'theta')
ntheta =
1/4*pi
```

Bài 8.6

Hệ số ma sát trượt μ được xác định theo công thức $\mu = F / mg$, trong đó *F (N)* là lực kéo tác dụng lên vật, *m (kg)* là khối lượng của vật, *g = 9.81 m/s²* là gia tốc trọng trường. Bảng dưới đây liệt kê kết quả các lần thí nghiệm với các lực tác dụng và khối lượng vật khác nhau. Xác định hệ số ma sát trong mỗi lần kiểm tra và hệ số ma sát trung bình của tất cả các lần kiểm tra.

Lần	1	2	3	4	5
Khối lượng m (kg)	2	4	5	10	20
Lực kéo F (N)	12.4	23.2	30.5	60.8	116.5



Hình 8. 17. Hình bài 8.6

Khai báo khối lượng và lực theo vector:

```
>>m = [2   4   5   10   20];
>>F = [12.4   23.2   30.5   60.8   116.5];
```

Hệ số ma sát trượt *mu*:

```
>>mu = F./(9.81 * m)
mu =
    0.6320    0.5912    0.6218    0.6198    0.5938
```

Hệ số ma sát trượt trung bình:

```
>>mutb = mean(mu)
mutb =
    0.6117
```

Bài 8.7

Một chất điểm được ném xuống theo phương thẳng đứng vào trong lòng một chất lỏng với vận tốc ban đầu là $v = 30(m/s)$. Do lực cản của chất lỏng, chất điểm có gia tốc $a = -(0.7v^3)(m/s^2)$. Vẽ đồ thị vận tốc theo thời gian và quãng đường theo thời gian của chất điểm.

Đáp án:

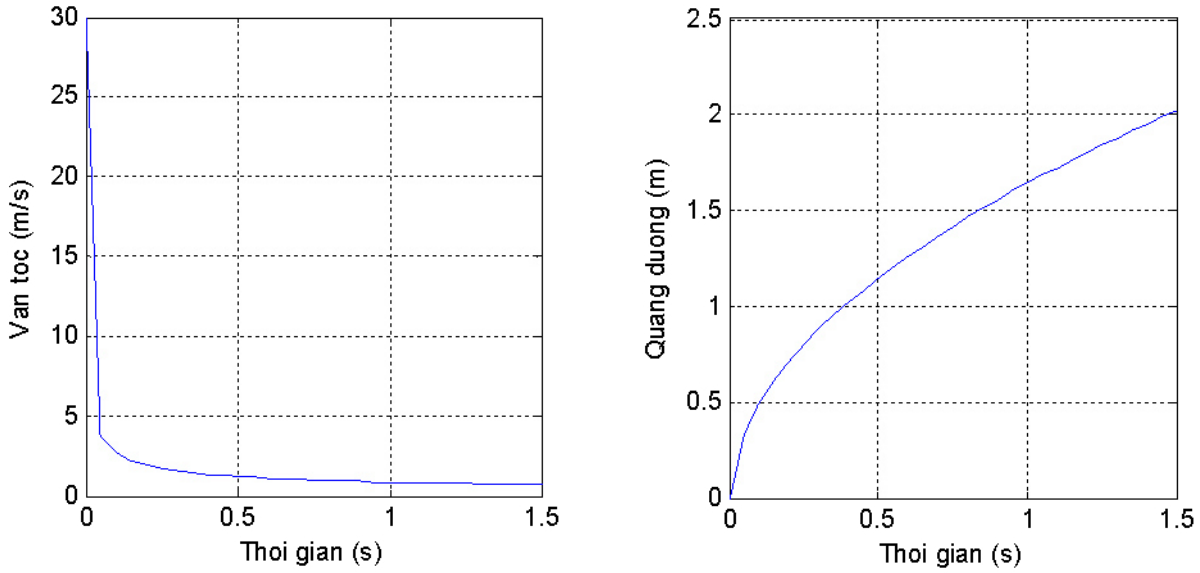
Phương trình biểu diễn vận tốc và quãng đường di chuyển của chất điểm theo thời gian có thể được tính bằng việc giải các phương trình vi phân bậc nhất sau:

$$\frac{dv}{dt} = -(0.7v^3) \quad v(0) = 30$$

$$\frac{ds}{dt} = v \quad s(0) = 0$$

Các bước giải trong MATLAB như sau:

```
>> v=dsolve('Dv + 0.7*v^3','v(0)=30')
v =
30/(1260*t+1)^(1/2)
>> s = dsolve('Ds - 30/(1260*t+1)^(1/2)','s(0)=0')
s =
1/21*(1260*t+1)^(1/2)-1/21
```



Hình 8. 18. Đồ thị bài 8.7

Vẽ đồ thị biểu diên vận tốc, quãng đường của chất điểm theo thời gian $0 \le t \le 1.5$

```
>>t = 0: 0.05: 1.5;
>>vt = 30./(1260*t+1).^(1/2);
>>st = (1/21)*(1260*t+1).^(1/2)-1/21;
>> plot(t,vt)
```

```
>> grid
>> xlabel('Thoi gian (s)');
>> ylabel('Van toc (m/s)');
>>figure(2);
>>plot(t,st);
>>grid
>> xlabel('Thoi gian (s)');
>> ylabel('Quang duong(m)');
```

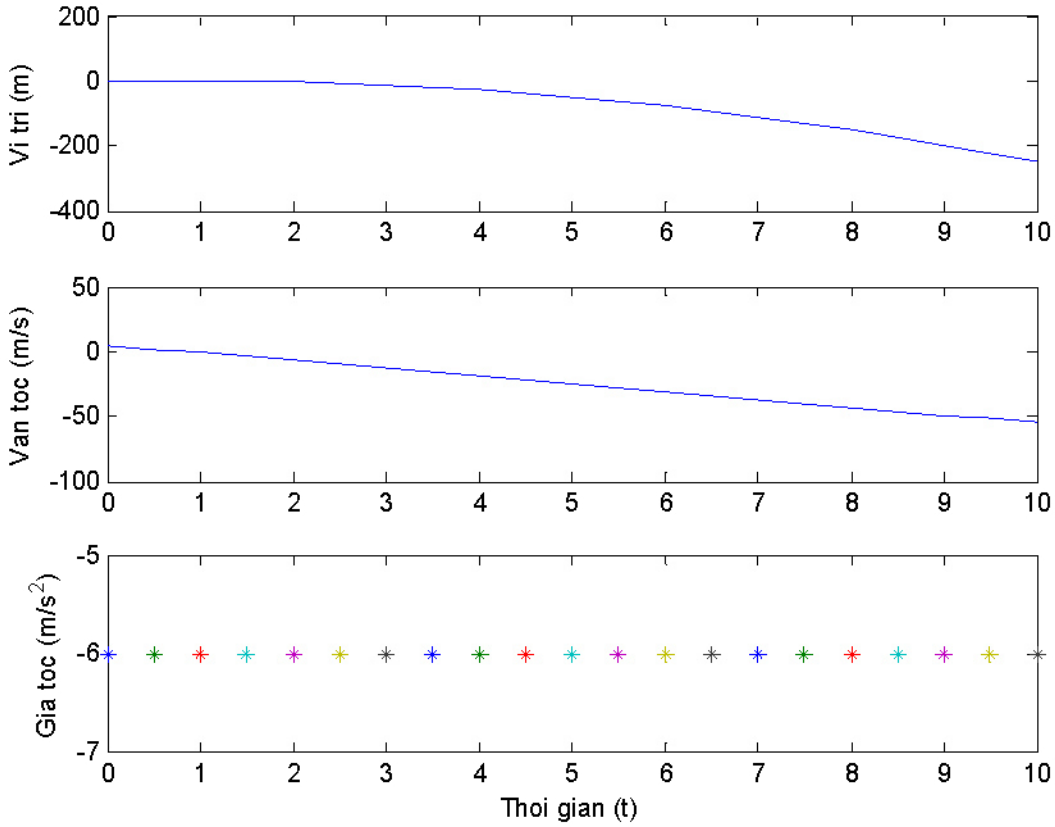
Bài 8.8

Vị trí của một ô tô được biểu diễn bởi hàm theo thời gian như sau $s = 5t - 3t^2$, vẽ đồ thị biểu diễn vị trí, vận tốc và gia tốc của ô tô trong khoảng thời gian $0 < t < 10$(s).

Đáp án:

```
>> s=sym('5*t - 3*t^2');
>> v=diff(s,'t');
>> a=diff(v);
>> st=inline(s)
st =
     Inline function:
     st(t) = 5.*t - 3.*t.^2
>> vt=inline(v)
vt =
     Inline function:
     vt(t) = 5-6.*t
>> at=inline(a)
at =
     Inline function:
     at(x) = -6
>> t=0:0.5:10;
>> subplot(3,1,1)
>> plot(t,st(t))
>> ylabel('Vi tri (m)');
```

```
>> subplot(3,1,2)
>> plot(t,vt(t))
>> ylabel('Van toc (m/s)');
>> subplot(3,1,3)
>> plot(t,at(t),'*')
>> ylabel('Gia toc (m/s^2)');
>> xlabel('Thoi gian (t)');
```



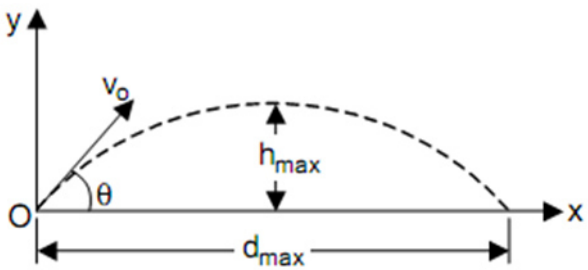
Hình 8. 19. Đồ thị bài 8.8

Bài 8.9

Chuyển động của một vật ném xiên được biểu diễn như trong hình vẽ. Trong đó, vận tốc ban đầu và góc nghiêng tạo bởi vận tốc ban đầu với phương ngang đã biết. Viết hàm tính độ cao và độ xa cực đại của chuyển động ném xiên. Sử dụng hàm để tính h_{max} và d_{max} khi $v_0 = 250m/s$ $\theta = 40°$

Đáp án:

```
function [hmax, dmax] = nemxien(v0,theta)
syms t td
%Doi gia tri goc nhap vao tu do sang radian
theta = theta*pi/180;
%Tinh gia tri ban dau v0 theo hai phuong x, y
v0x =v0*cos(theta);
v0y = v0*sin(theta);
%Viet phuong trinh tinh van toc hai phuong x, y theo t
vy = v0y - 9.81*t;
vx = v0x;
%Phuong trinh bieu dien chuyen dong tren hai phuong x, y
theo t
y = v0y*t - 1/2*9.81*t^2;
x = v0x*t;
%Tim thoi gian vat dat do cao cuc dai
thmax = solve(vy,'t');
hmax = double(subs(y,'t',thmax));
%Tim thoi gian vat dat do xa cuc dai
td = solve(y,'t');
if double(td(1)) > 0
    tdmax = td(1);
else
    tdmax = td(2);
end
dmax = double(subs(x,'t',tdmax));
end
```



Hình 8. 20. Hình bài 8.9

Thử lại hàm với $v_0 = 250m/s\ \theta = 40°$

```
>>>> [h,d]=nemxien(250,40)
h =
  1.3162e+00
d =
 6.2743e+003
```

Bài 8.10

Một chất điểm khối lượng 10kg được ném từ dưới đất theo phương thẳng đứng với vận tốc ban đầu 50m/s. Tìm độ cao cực đại mà chất điểm có thể đạt được biết lực cản không khí tại mọi thời điểm là $0.01v^2$ trong đó v là vận tốc tức thời của chất điểm. Vẽ đồ thị biểu diễn độ cao đạt được theo vận tốc của chất điểm.

Đáp án:

Hình 8. 21. Hình bài 8.10

Áp dụng định luật II Newton theo phương thẳng đứng:

$$\sum F_y = ma_y = -F - mg = -0.01v^2 - mg$$

$$a_y = \frac{dv}{dt} = \frac{-0.01v^2 - mg}{m}$$

Sinh viên có thể giải bài toán theo phương pháp đã sử dụng trong bài II.1.7 (sử dụng lệnh ***dsolve*** để tìm các hàm *v(t)* và *s(t)*, sau đó vẽ đồ thị *s(t)* theo *v(t)* với *t* là vector có các phần tử chạy từ 0 đến *tmax* (*tmax* là thời điểm vật đạt độ cao cực đại, vận tốc tức thời theo phương thẳng đứng của chất điểm bằng không)).

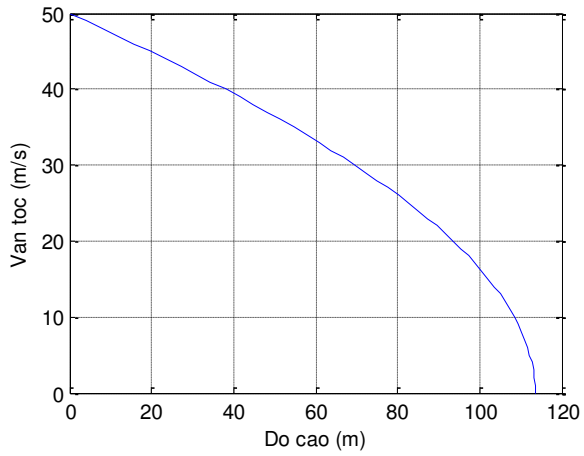
Trong bài tập này sinh viên có thể sử dụng phương pháp số dùng hàm ***ode45*** để giải như sau:

$$\frac{ds}{dt} = v \ , \ \frac{dv}{dt} = a$$

$$\frac{ds}{v} = \frac{dv}{a} \leftrightarrow ds = \frac{vdv}{a} \leftrightarrow \frac{ds}{dv} = \frac{v}{a} = \frac{mv}{-0.01v^2 - mg}$$

Do a là hàm theo v (theo đầu bài), ta có thể sử dụng hàm ***ode45*** để giải như sau:

```
>> s0=0;
>> v0=50;
>> vspan = [v0 : -1 : 0];
>> ptvs=@(v,s) 10*v/(-0.01*v^2 - 10*9.81);
>> [v,s]=ode45(ptvs,vspan,s0);
>>grid
>>xlabel('Do cao (m)'); ylabel('Van toc (m/s)');
```



Hình 8. 22. Đồ thị bài 8.10

Chương 9. LÝ THUYẾT ĐIỀU KHIỂN TỰ ĐỘNG

Trong chương này, sinh viên sẽ được giới thiệu một số phương pháp để giải các vấn đề cơ bản trong lý thuyết điều khiển tự động sử dụng các lệnh được xây dựng sẵn trong MATLAB và bộ công cụ khảo sát, thiết kế hệ thống điều khiển.

9.1 Biến đổi Laplace

Bên cạnh lệnh ***root*** đã được giới thiệu ở chương 3 dùng để tìm nghiệm của đa thức được khai báo dưới dạng vector mà các phần tử của vector là các hệ số của ma trận. Trong mục này, chúng ta sẽ làm quen với một lệnh khác của MATLAB được áp dụng để giải các bài toán biến đổi Laplace trong lý thuyết điều khiển tự động, đó là lệnh ***residue***. Lệnh ***residue*** cho phép khai triển phân thức của đa thức tử số *B(s)* và đa thức mẫu số *A(s)* thành tổng của các phân thức thành phần. Cú pháp:

```
>>[R, P, K] = residue(B, A)
```

Trường hợp không có nghiệm lặp:

$$\frac{B(s)}{A(s)} = \frac{R(1)}{s-P(1)} + \frac{R(2)}{s-P(2)} + ... + \frac{R(n)}{s-P(n)} + K(s)$$

Trong đó *R* là vector hệ số của các phân thức thành phần, *K* là vector hệ số của phân thức $\frac{B(s)}{A(s)}$ khi bậc của đa thức *B(S)* lớn hơn hoặc bằng bậc của đa thức *A(S)*, *P* là vector chứa nghiệm ở mẫu số của các phân thức thành phần.

Trường hợp tồn tại nghiệm lặp $P(j) = ... = P(j+m-1)$ với m là số lần lặp, khi đó các phân thức thành phần sẽ bao gồm:

$$... + \frac{R(j)}{s-P(j)} + \frac{R(j+1)}{(s-P(j))^2} + ... + \frac{R(j+m-1)}{(s-P(n))^m} + ...$$

Lệnh ***residue*** cũng được sử dụng để biến đổi tổng của các phân thức thành phần biểu diễn bởi các vector *R, P, K* thành phân thức của đa thức B và A.

```
>>[B, A] = residue(R, P, K)
```

Ví dụ: Tìm biến đổi ngược của hàm $L(f(t))(s) = \frac{1}{s^2-1}$

Trước khi sử dụng lệnh **residue** ta cần khai báo hai đa thức A, B.

```
>>B = 1;
>>A = [1   0   -1];
```

```
>>[R, P, K] = residue(B, A)
R =
     -0.5000
     0.5000
P =
     -1
     1
K =
     [ ]
```

Kết quả trả về cho thấy K là một mảng rỗng, phân thức đã cho tương đương với:

$$\frac{1}{s^2-1} = \frac{-0.5000}{s-(-1.0000)} + \frac{0.5000}{s-1.0000}$$

Qua đó ta suy ra được biến đổi ngược của hàm Laplace đã cho là $\frac{e^t - e^{-t}}{2}$

Bên cạnh việc sử dụng lệnh ***residue,*** biến đổi Laplace cũng có thể được tính toán trong MATLAB sử dụng bộ công cụ biến ký hiệu với hai lệnh ***laplace*** và ***ilaplace***.

Ví dụ:

Tìm biến đổi Laplace của hàm *f(t)* sau đây:

$$f(t) = -1.25 + 3.5te^{-2t} + 1.25e^{-2t}$$

```
>>syms t   s;
>>f = -1.25 + 3.5*t*exp(-2*t) + 1.25*exp(-2*t);
>>F = laplace(f, t, s)
F =
(s-5)/s/(s+2)^2
```

Tương đương với $F(s) = \frac{s-5}{s(s+2)^2}$

F(s) cũng có thể biểu diễn dưới dạng dễ quan sát như trên trong MATLAB sử dụng hàm ***pretty***.

```
>>pretty(F)
```

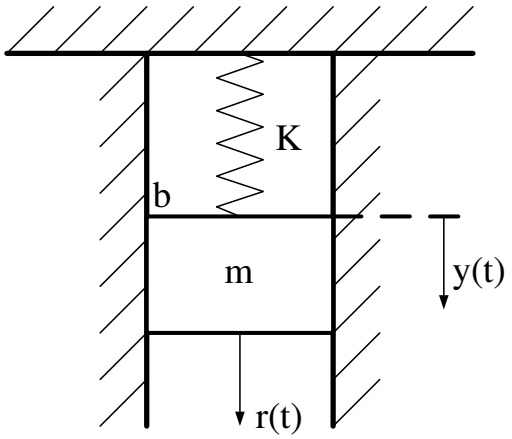
```
      s - 5
      ----------
               2
      s (s + 2)
```

Khi cần tìm biến đổi ngược của hàm *F(s)* ta có thể sử dụng lệnh ***ilaplace***.

```
>> Ft=ilaplace(F)
Ft =
-5/4+1/4*exp(-2*t)*(14*t+5)
>> simplify(Ft)
ans =
-5/4+7/2*t*exp(-2*t)+5/4*exp(-2*t)
>> pretty(ans)
                    - 5/4 + 7/2 t exp(-2 t) + 5/4 exp(-2
t)
```

Ứng dụng biến đổi Laplace để giải phương trình vi phân.

Cho hệ thống cơ khí như hình dưới



Hình 9. 1. Hệ thống lò xo vật nặng

Trong đó vật nặng chịu tác động của một lực *r(t) = 1* với $t \geq 0$, được gắn với một lò xo có hệ số *k*, hệ số ma sát giữa tường và vật nặng là *b*, vị trí của vật nặng theo thời gian t biểu diễn bởi *y(t)*. Giả sử $y(0) = 0$, $y'(0) = 2$, $m = 1$, $b = 3$, $k = 2$

Định luật II Newton viết cho hệ có dạng:

$$m\frac{d^2}{dt^2}y(t)+b\frac{d}{dt}y(t)+ky(t)=r(t)$$

$$\frac{d^2}{dt^2}y(t)+3\frac{d}{dt}y(t)+2y(t)=1(t)$$

Hàm *1(t)(unit step function)* trong MATLAB có thể được biểu diễn bởi hàm ***heaviside(t)***

Đáp ứng *y(t)*của hệ có thể giải trong MATLAB áp dụng biến đổi Laplace như sau

```
%Khai bao cac bien
syms Y   s   t;
%Khai bao ham rt
rt=heaviside(t);
%Bien doi Laplace cua rt
Rs=laplace(rt);
%Bien doi Laplace cua D2y=s^2*Y(s)-s*Y(0)-Dy(0)
Y2=s^2*Y-s*0-2;
%Bien doi Laplace cua Dy=s*Y(s)-y(0)
Y1=s*Y-0;
%Bieu dien Laplace cua dap ung y(t)
Ys=solve(Y2 + 3*Y1 + 2*Y - Rs, Y);
yt=ilaplace(Ys);
```

Kết quả tính: $y(t)=\frac{1}{2}-\frac{3}{2}e^{-2t}+e^{-t}$

9.2 Biến đổi z

Bộ công cụ biến ký hiệu trong MATLAB cho phép người sử dụng thực hiện các phép biến đổi tín hiệu rời rạc sang miền Z và ngược lại với các lệnh ***ztrans*** và ***iztrans***.

Cú pháp

```
>>Fz = ztrans(f)
```

Trong đó *Fz* là biến đổi z của hàm ký hiệu *f*

Ví dụ: Tìm biến đổi z của hàm sau.

$$x(n) = \frac{1}{4^n}u(n)$$

Khai báo biến ký hiệu n:

```
>>syms n
```

Khai báo hàm f:

```
>>f = (1/4^n)*heaviside(n);
```

Tìm biến đổi z của f:

```
>>Fz=ztrans(f)
Fz=
4*z/(4*z-1)
```

Ví dụ: Tìm tín hiệu *x(n)* biết:

$$X(z) = \frac{2z}{2z-1}$$

Các bước giải bài toán trong MATLAB như sau:

```
>>syms z;
>>Xz = (2*z)/(2*z-1);
>>xn = iztrans(Xz)
xn =
      (1/2)^n
```

Phép biến đổi Z ngược cũng có thể được thực hiện trong MATLAB với hàm ***residuez*** cho phép khai triển phân thức của đa thức tử số *B(z)* và đa thức mẫu số *A(z)* thành các phân thức sơ cấp, cú pháp tương tự lệnh ***residue***.

$$\frac{B(z)}{A(z)} = \frac{r(1)}{1-p(1)z^{-1}} + ... + \frac{r(n)}{1-p(n)z^{-1}} + k(1) + k(2)z^{-1} + ...$$

Lưu ý: Ở đây *B(z)* và *A(z)* biểu diễn chiều giảm dần hệ số lũy thừa của *z*.

$$B(z) = b_0 + b_1z^{-1} + b_2z^{-2} + ... + b_mz^{-m}$$

$$A(z) = a_0 + a_1z^{-1} + a_2z^{-2} + ... + a_nz^{-n}$$

Ví dụ: Tìm biến đổi Z ngược của tín hiệu sau:

$$X(z) = \frac{1}{(1-z^{-1})(1-0.5z^{-1})}$$

Viết lại *X(z)*dưới dạng lũy thừa không âm của *z*:

$$X(z) = \frac{z^2}{(z-1)(z-0.5)}$$

Các bước giải bài toán trong MATLAB như sau:

Khai báo biến ký hiệu *z*:

```
>>syms z;
```

Khai báo đa thức tử số và mẫu số dưới dạng vector chứa các hệ số:

```
>>B = 1;
>>A = [1   -1.5   0.5];
>>[R, P, K] = residuez(B, A)
R =
      2
     -1
P =
     1.0000
     0.5000
K =
     [ ]
```

Như vậy *X(z)*có thể được khai triển thành tổng của các phân thức sơ cấp như sau:

$$X(z) = \frac{2}{1-z^{-1}} - \frac{1}{1-0.5z^{-1}}$$

Khử lũy thừa âm trong biểu thức:

$$X(z) = \frac{2z}{z-1} - \frac{z}{z-0.5}$$

Tra bảng để suy ra tín hiệu cần tìm:

$$x(n) = 2u(n) - (0.5)^n u(n)$$

Kiểm tra lại kết quả bằng lệnh ***ztrans***:

```
>>syms n
>>ztrans(2*heaviside(n) - (1/2)^n*heaviside(n))
ans =
     2*z/(z-1)-2*z/(2*z-1)
```

Vẽ đồ thị điểm cực – điểm không trong MATLAB

Lệnh ***zplane*** trong MATLAB cho phép người sử dụng đồ thị điểm cực – điểm không của phân thức trong đó đa thức tử số và đa thức mẫu số được khai báo dưới dạng vector chứa các hệ số của đa thức, các điểm cực được ký hiệu trên đồ thị bằng điểm ***x***, các điểm không được ký hiệu trên đồ thị bằng điểm ***o***.

Ví dụ: Vẽ đồ thị điểm cực – điểm không.

$$H(z) = \frac{z^{-1} + \frac{1}{2}z^{-2}}{1 + \frac{3}{5}z^{-1} + \frac{2}{25}z^{-2}}$$

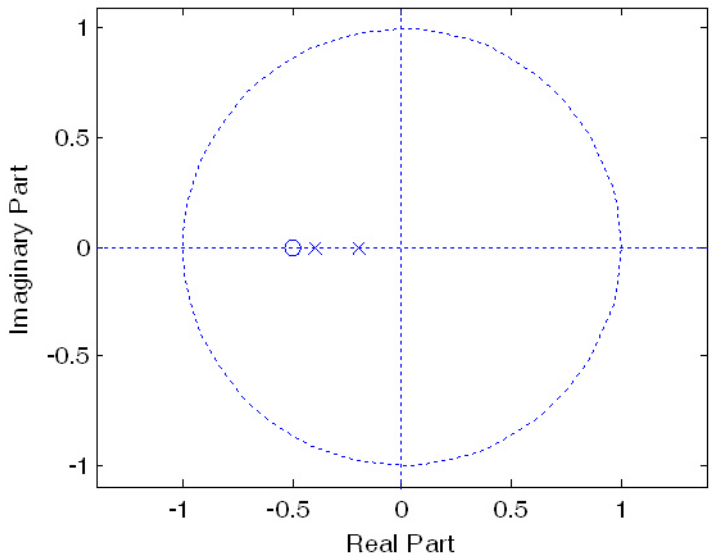
Các bước giải trong MATLAB như sau:

Khai báo đa thức tử số B và đa thức mẫu số A.

```
>>B = [1    1/2];
>>A = [2    3/5    2/25];
```

Sử dụng lệnh ***zplane*** để vẽ đồ thị điểm cực – điểm không:

```
>>zplane(B, A)
```



Hình 9. 2. Đồ thị điểm cực điểm không

9.2 Xây dựng các hệ tuyến tính – dừng

Trong mục này, người đọc sẽ được giới thiệu các phương thức mô tả hệ tuyến tính dừng trong điều khiển tử động với các mô hình tương ứng và các phép biến đổi từ mô hình này sang mô hình khác:

- Mô hình hàm truyền.
- Mô hình điểm không – điểm cực.
- Mô hình không gian trạng thái.

9.2.1 Mô hình hàm truyền

a. Hệ có một đầu vào và một đầu ra

Hàm truyền được xây dựng dưới dạng một phân thức bao gồm đa thức tử số B và đa thức mẫu số A sử dụng lệnh ***tf*** theo một trong các cú pháp sau:

```
>>sys = tf(B, A);
>>sys = tf(B, A, 'variable', 'p');
>>sys = tf(B, A, Ts);
>>S = tf('s');
```

định nghĩa dưới dạng hàm hữu tỉ của s

```
>>Z = tf('z',Ts);
```

định nghĩa dưới dạng hàm hữu tỉ của z

Trong đó ***sys*** là hệ tuyến tính dừng

B và ***A*** là các đa thức ở tử số và mẫu số có dạng:

$$B(s) = b_m s^m + b_{m-1} s^{m-1} + ... + b_1 s + b_0$$

$$A(s) = a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + a_0$$

và được biểu diễn dưới dạng vector của các hệ số thành phần.

Ts là chu kỳ lấy mẫu của hệ rời rạc.

variable định nghĩa biến biểu diễn.

Ví dụ:

```
>>B1 = [1   2   2];
>>A1 = [1   2   2   1];
>>B2 = [1   -1];
>>A2 = [1   3   2   1];
>>B3 = [1   0];
>>A3 = [1   -1.5   0.5];
>>s1 = tf(B1, A1)
Transfer function:
```

```
s^2 + 2 s + 2
---------------------
s^3 + 2 s^2 + 2 s + 1
>>s2 = tf(B2, A2,'variable','p')
Transfer function:
        p - 1
---------------------
p^3 + 3 p^2 + 2 p + 1
>>s3 = tf(B3, A3, 1)
Transfer function:
        z
-----------------
z^2 - 1.5 z + 0.5
Sampling time: 1
```

Ví dụ: Định nghĩa hàm truyền dưới dạng hàm hữu tỉ của **s** hoặc **z**.

```
>>s = tf('s')
Transfer function:
s
>> hs=(s^2+2*s+2)/(s^3+2*s^2+2*s+1)
Transfer function:
    s^2 + 2 s + 2
---------------------
s^3 + 2 s^2 + 2 s + 1
>>z = tf('z', 1)
Transfer function:
z
Sampling time: 1
>> hz=z/(z^2-1.5*z+0.5)
 Transfer function:
        z
-----------------
z^2 - 1.5 z + 0.5
Sampling time: 1
```

Đối với các hệ đã được định nghĩa bằng lệnh ***tf*** như trên, ta có thể sử dụng lệnh ***tfdata*** để trả về các vector biểu diễn đa thức mẫu số ***A*** và đa thức tử số ***B*** theo cú pháp:

```
>>[B, A] = tfdata(sys, 'v');
```

Ví dụ:

```
>>[B1n, A1n] = tfdata(s1, 'v')
B1n =
     0     1     2     2
A1n =
     1     2     2     1
>>[B2n, A2n] = tfdata(s2, 'v')
B2n =
     0     0     1    -1
A2n =
     1     3     2     1
```

b. Hệ có nhiều đầu vào và nhiều đầu ra

Đối với hệ có nhiều đầu vào và nhiều đầu ra, người sử dụng có thể khai báo hệ trong MATLAB như sau:

```
>>sys = tf(num, den)
>>sys = tf(num, den, 'variable', 'p')
>>sys = tf(num, den, Ts)
```

Trong đó ***num***, ***den*** là hai trường kích cỡ *Ny* × *Nu* với *Ny* là số đầu ra, *Nu* là số đầu vào, ***num*** chứa các đa thức tử số, ***den*** chứa các đa thức mẫu số. Các trường được khai báo theo cấu trúc sau:

{[…] […] […]…;[…] […] […]…;[…] […] […]…; …}

Ví dụ: Xét hệ có hai đầu vào và hai đầu ra.

```
>>nums2 = {2 [1 0.5]; [1 0] 2};
>>denum2 = {[4 4 1] [2 1 2]; [1 1] [1 2 1]};
>>Gp22 = tf(nums2, denum2)
Transfer function from input 1 to output...
          2
#1:  ---------------
     4 s^2 + 4 s + 1
```

```
          s
#2:  -----
      s + 1
Transfer function from input 2 to output...
          s + 0.5
#1:  -------------
     2 s^2 + s + 2
               2
#2:  -------------
     s^2 + 2 s + 1
```

9.2.2 Mô hình điểm không – điểm cực

a. Hệ có một đầu vào và một đầu ra

Một hệ tuyến tính cũng có thể được biểu diễn theo mô hình điểm không – điểm cực sử dụng lệnh ***zpk*** theo cú pháp sau:

```
>>sys = zpk(Z, P, K);
>>sys = zpk(Z, P, K, Ts);
>>S = zpk('s')
```

mô hình điểm không – điểm cực với biến Laplace

```
>>Z = zpk('z', Ts)
```

mô hình điểm không – điểm cực với biến z

Trong đó ***sys*** là hệ tuyến tính.

Z, P, K lần lượt là các vector điểm không Z, điểm cực P và hệ số khuếch đại K.

Ts là chu kỳ lấy mẫu.

Ví dụ:

```
>> Z1 = [ ];
>> P1 = [-1 -.5 -2];
>> K1 = [2];
>> sz1 = zpk(Z1, P1, K1)
```

```
Zero/pole/gain:
         2
-------------------
(s+1) (s+0.5) (s+2)
>> Z2 = [-0.5];
>> P2 = [-1+i -1-i -0.5];
>> K2 = 10;
>> sz2 = zpk(Z2, P2, K2)
Zero/pole/gain:
      10 (s+0.5)
----------------------
(s+0.5) (s^2 + 2s + 2)
```

Tương tự nhu đối với mô hình hàm truyền, ta có thể sử dụng lệnh ***zpkdata*** để trả về các giá trị điểm không, điểm cực và hệ số khuếch đại của mô hình theo cú pháp sau:

```
>>[Z, P, K] = zpkdata(sys, 'v')
```

Ví dụ:

```
>> [Z2n, P2n, K2n] = zpkdata(sz2, 'v')
Z2n =
   -0.5000
P2n =
  -1.0000 + 1.0000i
  -1.0000 - 1.0000i
  -0.5000
K2n =
    10
```

b. Hệ có nhiều đầu vào và nhiều đầu ra

Cú pháp tương tự như đối với hệ có một đầu vào và một đầu ra, lưu ý ở đây ***Z*** và ***P*** là hai trường có kích cỡ *Ny* × *Nu* trong đó *Z{i, j}* và *P{i, j}* biểu diễn các điểm không và điểm cực của hàm truyền từ đầu vào *j* tới đầu ra *i*; *K* là ma trận hai chiều chứa các hệ số khuếch đại cho mỗi kênh vào/ra.

Ví dụ: Xét mô hình **ZPK** có hai đầu ra và một đầu vào.

```
>>H = zpk({[ ]; [2 3]}, {1; [0 -1]}, [-5; 1])
Zero/pole/gain from input to output...
```

```
        -5
#1:  -----
     (s-1)

     (s-2) (s-3)
#2:  -----------
       s (s+1)
```

9.2.3 Mô hình không gian trạng thái

Mô hình trạng thái tổng quát viết dưới dạng ma trận:

$$\frac{d}{dt}x = Ax + Bu$$

$$y = Cx + Du$$

Lệnh ***ss*** trong MATLAB cho phép biểu diễn hệ thống theo mô hình không gian trạng thái sử dụng cú pháp sau:

```
>>sys = ss(A, B, C, D)
>>sys = ss(M, N, C, D, Ts)
```

Trong đó ***A, M*** là ma trận trạng thái của hệ liên tục và rời rạc

B, N là ma trận đầu vào của hệ liên tục và rời rạc

C là ma trận đầu ra

D là ma trận liên thông của mô hình trạng thái

Ts là thời gian quét mẫu

Ví dụ: Xét hệ phương trình vi phân tuyến tính với các biến trạng thái h_1, h_2, biến đầu vào u và biến đầu ra q.

$$\frac{dh_1}{dt} = -2h_1 + 3h_2 + 2v$$

$$\frac{dh_2}{dt} = 4h_1 - h_2 - v$$

$$q = h_1 - 2h_2 + \frac{1}{2}v$$

Hệ có thể được viết lại dưới dạng ma trận:

$$\frac{d}{dt}x = Ax + Bu$$

$$y = Cx + Du$$

với các vector đầu vào ***u***, vector đầu ra ***y*** và vector trạng thái ***x***

$$x = \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} \qquad u = \begin{bmatrix} v \end{bmatrix} \qquad y = \begin{bmatrix} q \end{bmatrix}$$

$$A = \begin{bmatrix} -2 & 3 \\ 4 & -1 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \\ -1 \end{bmatrix} \qquad C = \begin{bmatrix} 1 & -2 \end{bmatrix} \qquad D = \frac{1}{2}$$

Trong MATLAB ta có thể thiết lập mô hình không gian trạng thái như sau:

```
>>A = [-2   3; 4   -1];
>>B = [2;   -1];
>>C = [1   -2];
>>D = [1/2];
>>ht_ss = ss(A, B, C, D)
a =
       x1  x2
   x1  -2   3
   x2   4  -1

b =
       u1
   x1   2
   x2  -1

c =
       x1  x2
   y1   1  -2
d =
        u1
   y1  0.5
Continuous-time model.
```

Khi đã có mô hình không gian trạng thái, người dùng có thể truy xuất lại các ma trận thành phần *A, B, C, D*sử dụng lệnh ***ssdata*** theo cú pháp:

```
>>[A, B, C, D] = ssdata(sys, 'v');
>>[M, N, C, D, Ts]
```

9.2.4 Chuyển đổi giữa các mô hình

Trong MATLAB, việc chuyển đổi từ mô hình này sang mô hình khác giữa các mô hình giới thiệu trong chương này là khá dễ dàng, xét ví dụ sau: Chuyển đổi từ mô hình không gian trạng thái sang mô hình hàm truyền.

Xét mô hình không gian trạng thái giới thiệu ở mục **2.2.2**, mô hình này có thể chuyển đổi sang mô hình hàm truyền sử dụng lệnh ***ss2tf*** theo cú pháp sau:

```
>>[num, den] = ss2tf(A, B, C, D);
```

Trong đó giá trị trả về ***num***, ***den*** lần lượt là các vector đa thức ở tử số và mẫu số của mô hình hàm truyền.

```
num =
     0.5000    5.5000  -18.0000
den =
1     3   -10
>>F = tf(num, den)
Transfer function:
0.5 s^2 + 5.5 s - 18
--------------------
   s^2 + 3 s - 10
```

Bên cạnh đó MATLAB cũng cho phép chuyển đổi tự động đối tượng từ dạng mô hình này sang mô hình khác sử dụng các lệnh định nghĩa mô hình giới thiệu ở trên ***ts***, ***zpk***, ***ss***.

Ví dụ:

```
>> F2 = tf(ht_ss)
Transfer function:
0.5 s^2 + 5.5 s - 18
--------------------
   s^2 + 3 s - 10
```

9.2.5 Đặc tính thời gian trễ trong hàm truyền

Để thể hiện đặc tính thời gian trễ trong các hàm truyền ta có thể đưa thêm tham số vào lệnh ***tf*** hoặc thay đổi thuộc tính của mô hình bằng cách sử dụng lệnh ***set*** như trong hai ví dụ sau:

Ví dụ: Sử dụng tham số trong lệnh **tf**.

```
>>Q = tf(num, den, 'Inputdelay', 2)
Transfer function:
                0.5 s^2 + 5.5 s - 18
exp(-2*s) *     --------------------
                     s^2 + 3 s - 10
```

Ví dụ: Sử dụng lệnh **set** để thay đổi thuộc tính hàm truyền.

```
>>set(Q, 'Inputdelay', 0.5)
>>Q
Transfer function:
                    0.5 s^2 + 5.5 s - 18
exp(-0.5*s) *       --------------------
                        s^2 + 3 s - 10
```

9.2.6 Ghép nối các mô hình

Các mô hình hệ thống tuyến tính – dừng có thể được ghép nối trong MATLAB theo các phương thức nối tiếp (lệnh ***series***), song song (lệnh ***parallel***), ghép có phản hồi (lệnh ***feedback***).

Ví dụ: Ghép nối tiếp hai mô hình G1 và G2.

```
>>G1 = tf([1], [10, 1])
Transfer function:
1
--------
10 s + 1
>> G2 = tf([1, 2], [2, 3, 2])
Transfer function:
     s + 2
---------------
2 s^2 + 3 s + 2
>>G3 = series(G1, G2)
```

```
Transfer function:
s + 2
-------------------------
20 s^3 + 32 s^2 + 23 s + 2
```

Bên cạnh đó MATLAB cũng cho phép người dùng sử dụng một số phép toán cơ bản để thực hiện việc ghép nối các mô hình như phép cộng (lệnh **+**), phép trừ (lệnh **-**), phép nhân (lệnh *), phép đảo (lệnh ***inv***).

Ví dụ: So sánh kết quả phép nhân ghép nối G1, G2 với lệnh **series**.

```
>>G3m = G1*G2
Transfer function:
          s + 2
-------------------------
20 s^3 + 32 s^2 + 23 s + 2
```

Ví dụ: So sánh kết quả tính hồi tiếp âm khi sử dụng phép nhân, phép đảo, phép cộng và khi sử dụng lệnh **feedback**.

```
>> G4=G1*inv(1+G1*G2)
Transfer function:
      20 s^3 + 32 s^2 + 23 s + 2
-------------------------------------
200 s^4 + 340 s^3 + 272 s^2 + 64 s + 4
%Dua ve dang rut gon
>> G4 = zpk(G4)
Zero/pole/gain:
0.1 (s+0.1) (s^2 + 1.5s + 1)
-----------------------------------------
(s+0.2244) (s+0.1) (s^2 + 1.376s + 0.8913)
% Su dung lenh feedback
>> G4m=zpk(feedback(G1, G2))
Zero/pole/gain:
0.1 (s^2 + 1.5s + 1)
---------------------------------
(s+0.2244) (s^2 + 1.376s + 0.8913)
```

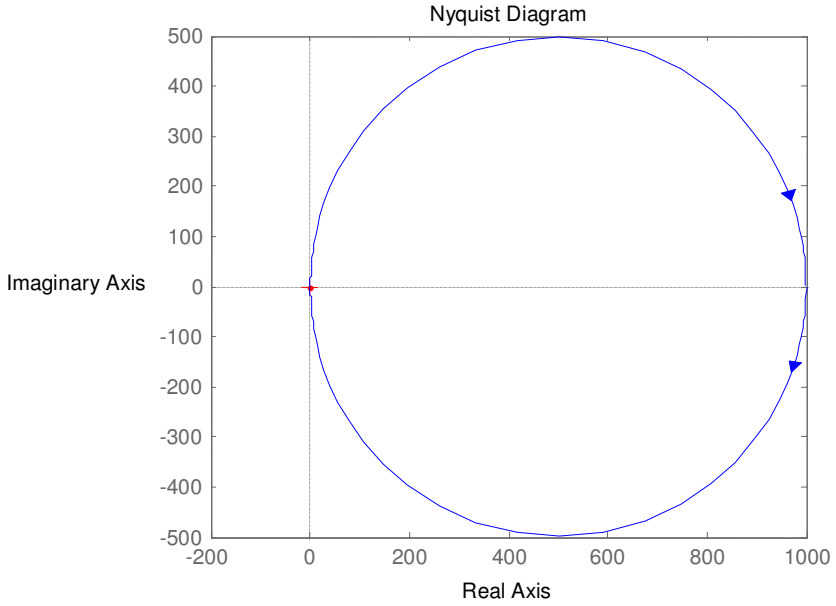
Có thể thấy là khi sử dụng ***feedback*** thì MATLAB tự động rút gọn *(s + 0.1)* ở cả đa thức tử số và đa thức mẫu số.

9.2.7 Giản đồ Bode, Nyquist, Nichols và đáp ứng của hệ

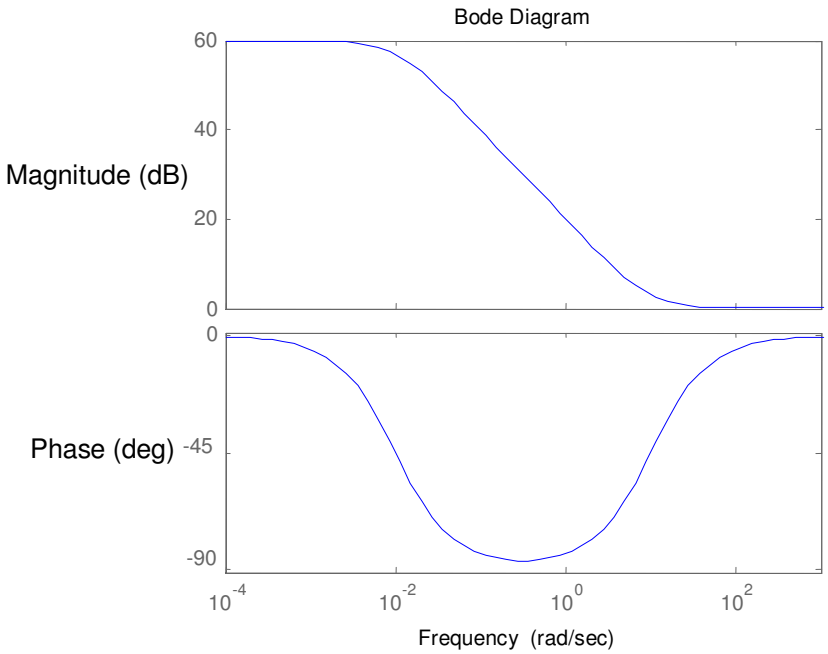
Trong MATLAB, người sử dụng có thể vẽ giản đồ Bode, Nyquist, Nichols của hệ bằng cách sử dụng các lệnh ***bode(G1)***, ***nyquist(G1)***, ***nichols(G1)*** tương ứng. Trong đó, *G1* là hệ cần khảo sát, đáp ứng xung và đáp ứng bước nhảy của hệ có thể tìm được thông qua các lệnh ***impuls(G1)*** và ***step(G1)***.

Ví dụ: Vẽ giản đồ Bode và Nyquist của hệ sau:

```
>>Gp=zpk([-10],[-0.01],1)
Zero/pole/gain:
(s+10)
--------
(s+0.01)
%Gian do Nyquist
>>nyquist(Gp)
%Gian do Bode
>>bode(Gp)
```

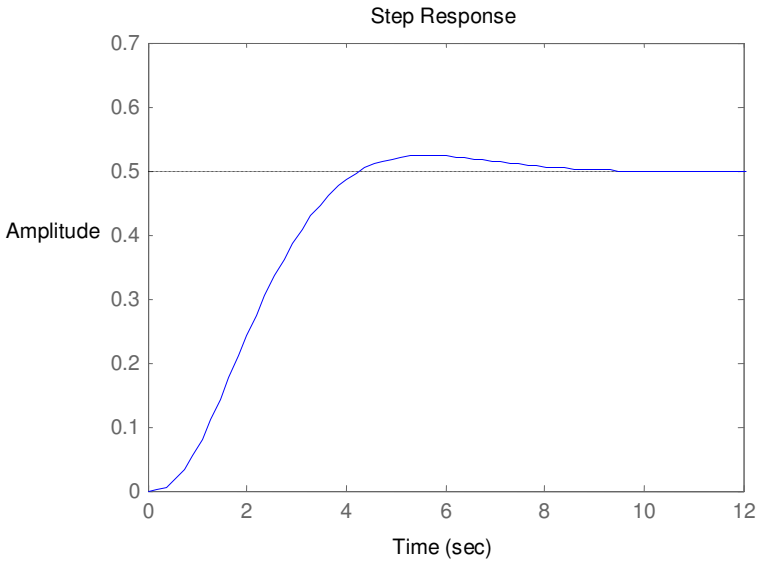


Hình 9. 3. Đồ thị Nyquist của Gp



Hình 9. 4. Đồ thị Bode của Gp

Ví dụ: Xác định đáp ứng xung.



Hình 9. 5. Đáp ứng bước nhảy của PROCESS

```
>>PROCESS = tf([1], [1 4 4 2])
```

```
Transfer function:
          1
--------------------
s^3 + 4 s^2 + 4 s + 2
>>step(PROCESS)
```

9.3 Tóm tắt chương 9

Biến đổi Laplace và biến đổi z	
laplace	Biến đổi Laplace thuận
ilaplace	Biến đổi Laplace ngược
heaviside(t)	Biểu diễn hàm *1(t)*
residue	Phân tích đa thức hữu tỉ thành tổng các đa thức thành phần
ztrans	Biến đổi z thuận
iztrans	Biến đổi z ngược
residuez	Khai triển đa thức ẩn z thành tổng các đa thức thành phần
zplane	Biểu diễn đồ thị điểm cực – điểm không
Xây dựng các hệ tuyến tính – dừng	
tf	Mô hình hàm truyền
zpk	Mô hình điểm không – điểm cực – hệ số khuếch đại
ss	Mô hình không gian trạng thái
tfdata	Trả về các yếu tố thành phần của mô hình ***tf***
zpkdata	Trả về các yếu tố thành phần của mô hình ***zpk***
ssdata	Trả về các yếu tố thành phần của mô hình ***ss***
ss2tf	Chuyển đổi từ mô hình không gian trạng thái sang hàm truyền
ss2zp	Chuyển đổi từ mô hình không gian trạng thái sang điểm không – cực
tf2ss	Chuyển đổi từ mô hình hàm truyền sang không gian trạng thái

tf2zp	Chuyển đổi từ mô hình hàm truyền sang điểm không - cực
zp2ss	Chuyển đổi từ mô hình điểm không – cực sang không gian trạng thái
zp2tf	Chuyển đổi từ mô hình điểm không – cực sang hàm truyền
series	Ghép nối tiếp hai mô hình
parallel	Ghép song song hai mô hình
feedback	Ghép nối có hồi tiếp
nyquist	Vẽ giản đồ Nyquist
bode	Vẽ giản đồ Bode
nichols	Vẽ giản đồ Nichols
impulse	Tìm đáp ứng xung của hệ
step	Tìm đáp ứng bước nhảy của hệ

Bảng 9. 1. Tóm tắt các hàm sử dụng trong chương 9

9.4 Bài tập chương 9

Bài 9.1

Phân tích đa thức hữu tỉ thành tổng các đa thức thành phần, tìm biến đổi Laplace ngược của:

$$F_1(s) = \frac{1}{s^4 + 5s^3 + 7s^2}$$

Đáp án:

```
>>b = [0 0 0 0 1];
>>a = [1 5 7 0 0];
>>[r, p, k] = residue(b,a)
r =
     0.0510 - 0.0648i
     0.0510 + 0.0648i
     -0.1020
     0.1429

p =
```

```
     -2.5000 + 0.8660i
     -2.5000 - 0.8660i
     0
     0
k =
     [ ]
```

Từ kết quả tính trong MATLAB ta có thể biểu diễn lại $F_1(s)$ như sau:

$$F_1(s) = \frac{r_1}{s-p_1} + \frac{r_2}{s-p_2} + \frac{r_3}{s-p_3} + \frac{r_4}{s-p_4}$$

$$F_1(s) = \frac{0.0510 - 0.0648i}{s-(2.5000+0.8660i)} + \frac{0.0510 + 0.0648i}{s-(-2.5000-0.8660i)} + \frac{-0.1020}{s-0} + \frac{0.1429}{s-0}$$

Biến đổi Laplace ngược của $F_1(s)$ có thể tính như sau:

```
>>syms s
>>f1 = 1/(s^4 + 5*s^3 + 7*s^2);
>>ilaplce(f1)
ans =
     -5/49+1/7*t+1/147*exp(-
5/2*t)*(15*cos(1/2*3^(1/2)*t)
     +11*3^(1/2)*sin(1/2*3^(1/2)*t))
```

Bài 9.2

Tìm biến đổi Laplace của các hàm sau:

a. $f_1(t) = 7t^3\cos(5t + 60°)$

b. $f_2(t) = -7te^{-5t}$

c. $f_3(t) = -3\cos(5t)$

d. $f_4(t) = t\sin(7t)$

e. $f_5(t) = 5e^{-2t}\cos(5t)$

Đáp án:

a.

```
>>syms t
>>f1t = 7*t^3*cos(5*t + (pi/3));
>>f1s = laplace(f1t)
```

```
f1s =
   21*(s^4+625-150*s^2-
20*3^(1/2)*s^3+500*3^(1/2)*s)/(s^2+25)^4
```

b.

```
>>f2t = -7*t*exp(-5*t);
>>f2s = laplace(f2t)
f2s =
   -7/(s+5)^2
```

c.

```
>>f3t = -3*cos(5*t);
>>f3s = laplace(f3t)
f3s =
   -3*s/(s^2+25)
```

d.

```
>>f4t = t*sin(7*t);
>>f4s = laplace(f4t)
f4s =
   14*s/(s^2+49)^2
```

e.

```
>>f5t = 5*exp(-2*t)*cos(5*t);
>>f5s = laplace(f5t)
f5s =
   5*(s+2)/(s^2+4*s+29)
```

Bài 9.3

Tìm biến đổi Z của các hàm sau:

a. $x_1(n) = \left(\frac{1}{2}\right)^n$

b. $x_2(n) = e^{2n}$

c. $x_3(n) = n$

Đáp án:

a.

```
>>syms n
```

```
>>x1n = (1/2)^n;
>>X1Z = ztrans(x1n)
X1Z =
   2*z/(2*z-1)
```

b.

```
>>x2n = exp(2*n);
>>X2Z = ztrans(x2n)
X2Z =
   z/exp(2)/(z/exp(2)-1)
```

c.

```
>>x3n = n;
>>X3Z = ztrans(x3n)
X3Z =
   z/(z-1)^2
```

Bài 9.4

a. Cho: $X_1(Z) = \frac{5z}{(z-1)^2} - \frac{2z}{(z-0.5)^2}$ tìm biến đổi Z ngược.

b. Cho: $X_2(Z) = \frac{10z}{z^2 - z + 1}$ tìm biến đổi Z ngược.

c. $X_3(Z) = \frac{z^2}{(z-1)(z-0.5)^2}$ sử dụng lệnh ***residuez*** phân tích $X_3(Z)$ thành các đa thức hữu tỉ thành phần rồi tìm biến đổi Z ngược.

d. $X_4(Z) = \frac{z^2(z+1)}{(z-1)(z^2 - z + 0.5)}$ sử dụng lệnh ***residuez*** phân tích $X_4(Z)$ thành các đa thức hữu tỉ thành phần.

Đáp án:

a.

```
>>syms z
>>X1Z = 5*z/(z-1)^2 - 2*z/(z-0.5)^2;
>>x1n = iztrans(X1Z)
x1n =
```

```
   5*n-4*(1/2)^n*n
```

$$x_1(n) = 5nu(n) - 4n(0.5)^n u(n)$$

b.

```
>>X2Z = 10*z/(z^2 - z + 1);
>>xn2 = iztrans(X2Z)
x2n =
   20/3*3^(1/2)*sin(1/3*pi*n)
```

$$x_2(n) = 11.547\sin(n.60°)$$

c.

```
>>a = conv(conv([1, -0.5], [1, -0.5]), [1, -1]);
>>b = [0, 1];
>>[r, p, k] = residuez(b, a)
r =
    4.0000
   -2.0000 - 0.0000i
   -2.0000 + 0.0000i
p =
    1.0000
    0.5000 + 0.0000i
    0.5000 - 0.0000i
k =
      [ ]
>>X3Z = 4/(1-z^(-1))-2/(1-0.5*z^(-1))-2/((1-0.5*z^(-
1))^2);
>>x3n = iztrans(X3Z)
>>x3n =
   4-4*(1/2)^n-2*(1/2)^n*n
```

$$X_3(Z) = \frac{4}{1-z^{-1}} + \frac{-2}{1-0.5z^{-1}} + \frac{-2}{(1-0.5z^{-1})^2}$$

$$x_3(n) = 4u(n) - 4(0.5)^n u(n) - 2n(0.5)^n u(n)$$

d.

```
>> b=conv([1 0 0],[1,1]);
```

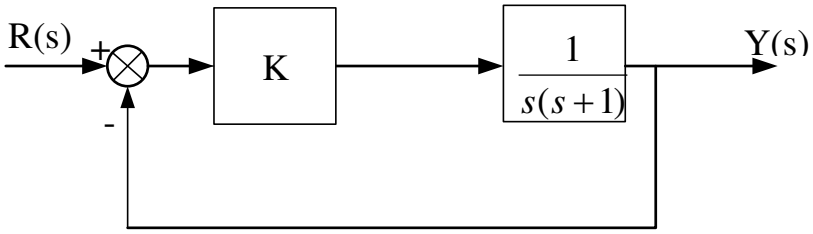
```
>> a=conv([1 -1],[1 -1 0.5]);
>> [r,p,k]=residuez(b,a)
r =
   4.0000
  -1.5000 - 0.5000i
  -1.5000 + 0.5000i
p =
   1.0000
   0.5000 + 0.5000i
   0.5000 - 0.5000i
k =
     0
```

$$X_4(Z)=\frac{4}{1-z^{-1}}+\frac{-1.5-0.5i}{1-(0.5+0.5i)z^{-1}}+\frac{-1.5+0.5i}{1-(-0.5-0.5i)z^{-1}}$$

Bài 9.5

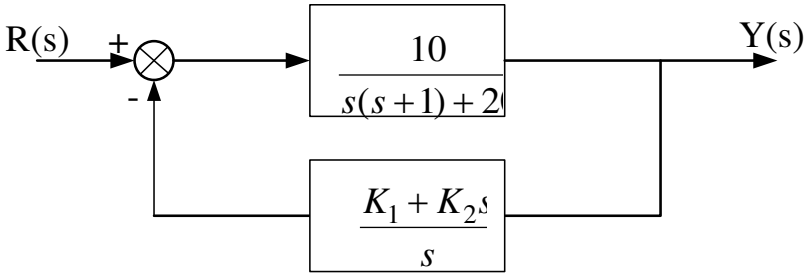
Tìm hàm truyền cho các hệ sau (*K = 10, K1 = 100, K2 = 150)*.

a.



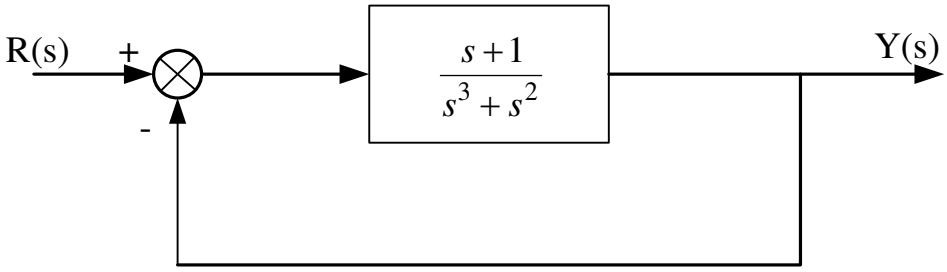
Hình 9. 6. Hình bài 9.5a

b.



Hình 9. 7. Hình bài 9.5b

c.



Hình 9. 8. Hình bài 9.5c

Đáp án:

a. Gọi: $G(s) = K\frac{1}{s(s+1)} = \frac{K}{s(s+1)}$

Ta có: $Y(s) = (R(s) - Y(s))G(s)$

$$\frac{Y(s)}{R(s)} = \frac{G(s)}{1+G(s)} = \frac{\frac{K}{s(s+1)}}{1+\frac{K}{s(s+1)}} = \frac{K}{s(s+1)+K} = \frac{K}{s^2+s+K}$$

Các bước giải trong MATLAB như sau:

```
>>syms s
>>Gs = tf(10, sym2poly(s*(s+1)));
>>Hs = feedback(Gs, 1)
Transfer function:
     10
------------
s^2 + s + 10
```

b. Hàm truyền hở:

```
>>Gs = tf(10, sym2poly(s*(s+1)+20));
```

Hàm truyền hồi tiếp:

```
>>Ms = tf([150, 100], [1, 0]);
```

Hàm truyền của hệ:

```
>>Hs = feedback(Gs, Ms)


Transfer function:
          10 s
```

```
-------------------------
s^3 + s^2 + 1520 s + 1000
```

c. Hàm truyền hở:

```
>>Gs = tf([1, 1], [1, 1, 0, 0]);
```

Hàm truyền của hệ:

```
>>Hs = feedback(Gs, 1)
Transfer function:
      s + 1
-----------------
s^3 + s^2 + s + 1
```

Bài 9.6

Xác định vị trí các điểm cực của hệ có hàm truyền như sau:

$$H(s)=\frac{s^3-6s^2+7s+15}{s^5+s^4-5s^3-9s^2+11s-12}$$

Đáp án:

Các điểm cực là nghiệm của đa thức mẫu số:

```
>>den = [1   1   -5   -9   11   -12];
>>A = roots(den)
A =
  -2.1586 + 1.2396i
  -2.1586 - 1.2396i
   2.3339
   0.4917 + 0.7669i
   0.4917 - 0.7669i
```

Bài 9.7

Xác định vị trí các điểm cực của hàm truyền đơn vị có hàm truyền hở:

$$G(s)=\frac{150}{(s+5)(s+7)(s+9)(s+11)}$$

Đáp án:

Khai báo hàm truyền hở

```
>>syms s
>>Gs = tf(150, sym2poly((s+5)*(s+7)*(s+9)*(s+11)))
```

```
Transfer function:
                150
-------------------------------------
s^4 + 32 s^3 + 374 s^2 + 1888 s + 3465
```

Hàm truyền của hệ:

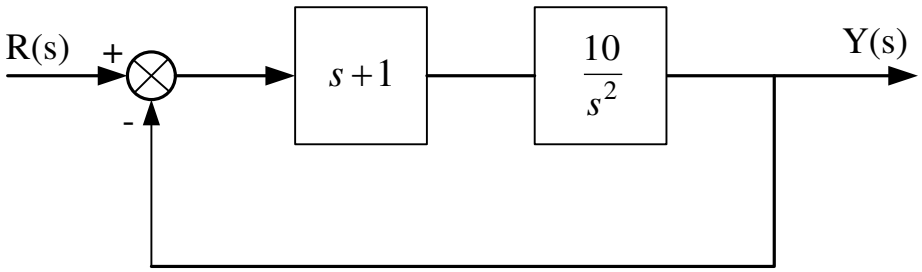
```
>>Hs = feedback(Gs, 1)
Transfer function:
                150
-------------------------------------
s^4 + 32 s^3 + 374 s^2 + 1888 s + 3615
```

Xác định các điểm cực:

```
>>pole(Hs)
ans =
 -10.9673 + 1.9506i
 -10.9673 - 1.9506i
  -5.0327 + 1.9506i
  -5.0327 - 1.9506i
```

Bài 9.8

Tìm đáp ứng đối với hàm bước đơn vị của hệ sau:



Hình 9. 9. Hình bài 9.8

Đáp án:

```
>>syms s   t
>>G1 = tf([1, 1], [1]);
>>G2 = tf([10], [1, 0, 0]);
```

Hàm truyền vòng hở:

```
>>Gs = series(G1, G2);
```

Hàm truyền của hệ:

```
>>Hs = feedback(Gs, 1)
Transfer function:
   10 s + 10
---------------
s^2 + 10 s + 10
```

Đáp ứng của hệ

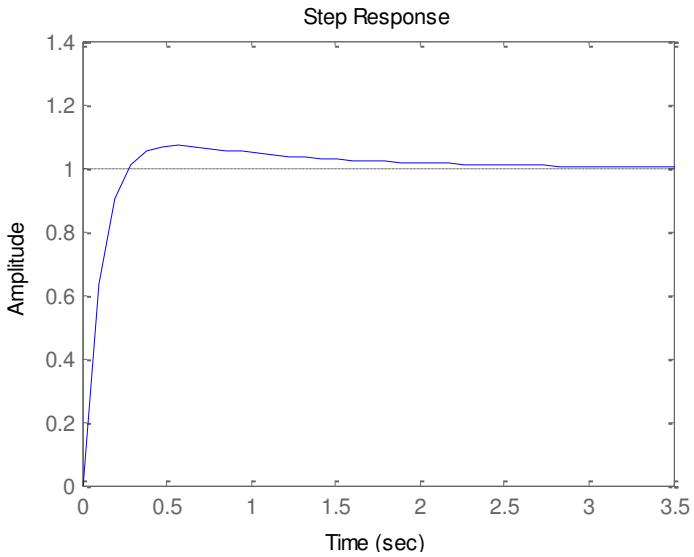
```
>>step(Hs)
```

Sinh viên có thể tự kiểm tra bằng cách tìm hàm Laplace của hàm bước đơn vị để suy ra giá trị của đầu vào *R(s)*.

```
>>Rs = laplace(heaviside(t))
Rs =
1/s
```

Tính Y(s) theo H(s) và R(s) đã biết:

$$Y(s) = R(s)H(s)$$



Hình 9. 10. Đồ thị bài 9.8

Đồ thị đáp ứng của hệ chính là đồ thị của hàm *y(t)* với *y(t)* là biến đổi Laplace ngược của Y(s).

Lưu ý ở đây muốn thực hiện bước tính Y(s) theo R(s) và H(s) sau đó biến đổi Laplace ngược để ra y(t) cần khai báo lại biến Ys là biểu thức biến ký hiệu.

Bài 9.9

Dùng MATLAB vẽ biểu đồ Bode của các hệ cho bởi hàm truyền như sau:

a. $G(s) = \dfrac{1}{2s+1}$

b. $G(s) = \dfrac{4}{s^2+s+4}$

Đáp án:

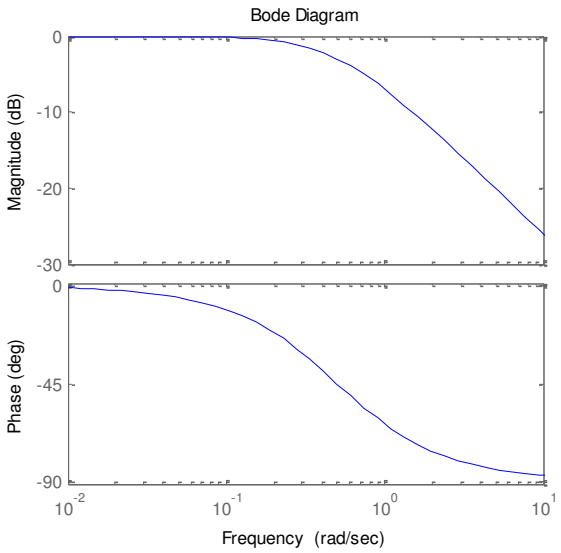
a.

Khai báo hàm truyền của hệ:

```
>>syms s
>>G = tf([1], [2 1])
Transfer function:
   1
-------
2 s + 1
```

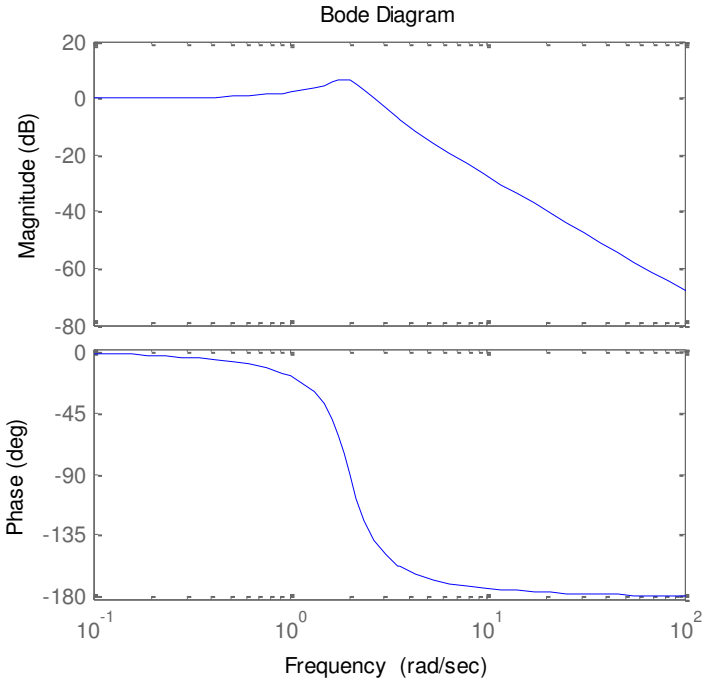
Vẽ đồ thị Bode

>>bode(G)



Hình 9. 11. Đồ thị bài 9.9a

b.



Hình 9. 12. Đồ thị bài 9.9b

```
>>G = tf([4], [1   1   4])

Transfer function:
     4
-----------
s^2 + s + 4
>>bode(G)
```

Bài 9.10

Dùng MATLAB vẽ đồ thị Nyquist và Nichols của hệ cho bởi hàm truyền sau:

$$G(s) = \frac{k(s+1)(s+3+7i)(s+3-7i)}{(s+1)(s+3)(s+5)(s+3+7i)(s+3-7i)} \quad (\text{k} = 30)$$

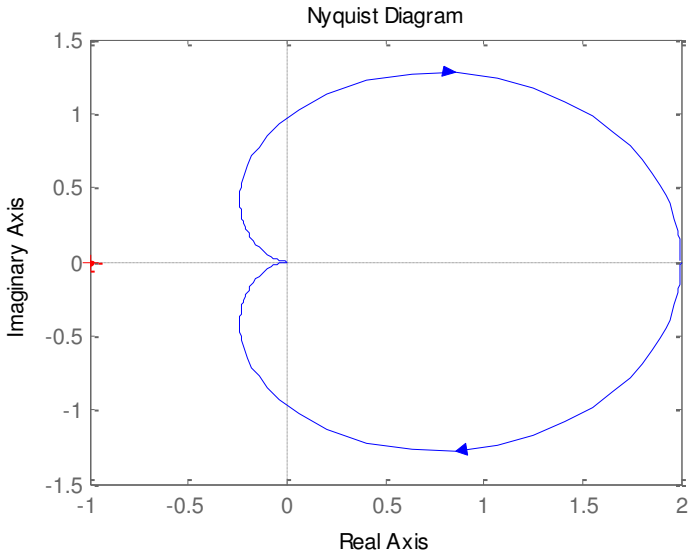
Đáp án:

Lưu ý hệ ở đây cũng có thể được khai báo theo vector điểm không, điểm cực rồi đưa về mô hình hàm truyền bằng lệnh ***zp2tf***.

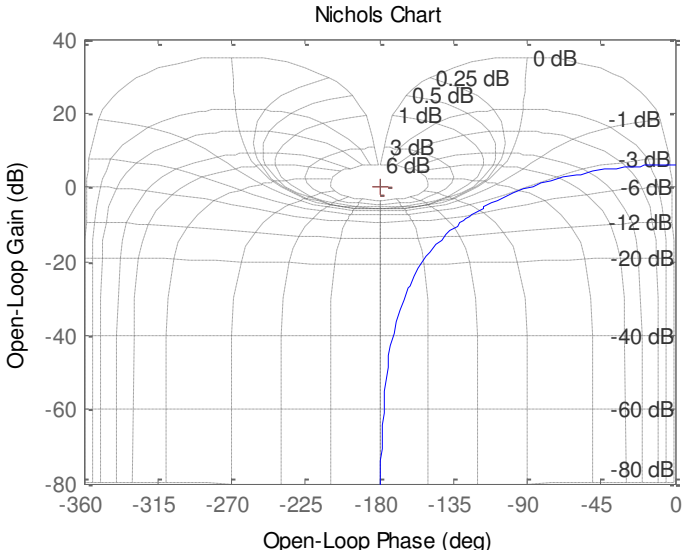
Khai báo theo mô hình hàm truyền với các đa thức tử số và đa thức mẫu số:

```
>>syms s
```

```
>>nG = sym2poly(30*(s+1)*(s+3+7i)*(s+3-7i));
>>dG = sym2poly((s+1)*(s+3)*(s+5)*(s+3+7i)*(s+3-7i));
>>G = tf(nG, dG)
```



Hình 9. 13. Đồ thị Nyquist của hệ



Hình 9. 14. Đồ thị Nichols của hệ

Khai báo theo điểm không, điểm cực rồi đưa về mô hình hàm truyền, đa thức tử số và đa thức mẫu số được biểu diễn dưới dạng các vector hệ số:

```
>>z = [-1   -3-7i   -3+7i]';
>>p = [-1   -3   -5   -3-7i   -3+7i]';
>>k = 30;
```

```
>>[num, den] = zp2tf(z, p, k)
num =
0              0             30            210           1920
1740
den =
1             15            135            675           1424
870
```

So sánh hàm truyền khai báo bằng lệnh ***tf***.

```
>>G
Transfer function:
       30 s^3 + 210 s^2 + 1920 s + 1740
-----------------------------------------------
s^5 + 15 s^4 + 135 s^3 + 675 s^2 + 1424 s + 870
```

Vẽ đồ thị Nyquist:

```
>>nyquist(G)
```

Hoặc có thể sử dụng lệnh

```
>>nyquist(num, den)
```

Vẽ đồ thị Nichol:

```
>>nichol(G)
>>ngrid
```

Hoặc cũng có thể sử dụng lệnh

```
>>nichols(num, den).
```

MỘT SỐ BỘ CÔNG CỤ ỨNG DỤNG CHUYÊN SÂU TRONG MATLAB

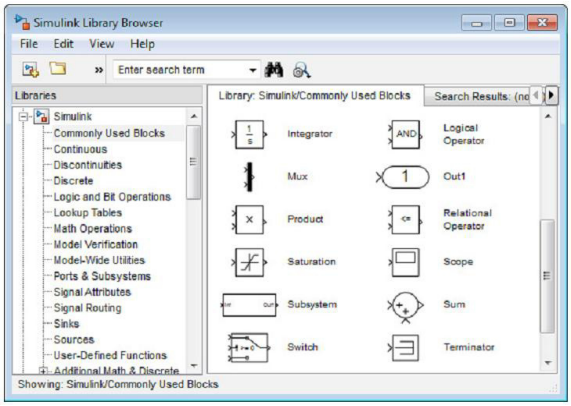
1. Simulink

Simulink là môi trường sơ đồ khối cho phép mô phỏng và thiết kế các mô hình. Bên cạnh chức năng mô phỏng, Simulink hỗ trợ người dùng trong việc thiết lập tự động các đoạn mã, kiểm thử liên tục và xác nhận các hệ thống nhúng.

Simulink cho phép xử lý đồ họa dưới dạng sơ đồ khối, thiết lập các thư viện khối cũng như cung cấp giải pháp mô hình, mô phỏng hệ động lực. Simulink được tích hợp với MATLAB và cho phép người sử dụng kết hợp các thuật toán MATLAB vào các mô hình cũng như xuất kết quả mô phỏng ra MATLAB cho các bước phân tích khác.

Một số tính năng chính của Simulink

- Xây dựng mô hình hệ thống phụ phân cấp với các khối định nghĩa sẵn.
- Mô phỏng động lực mô hình và kiểm tra kết qủa dưới dạng chạy mô phỏng.
- Phân tích kết quả mô phỏng: xem và gỡ lỗi mô phỏng
- Quản lý các dự án: dễ dàng quản lý các file cũng như dữ liệu thành phần của các dự án.
- Kết nối với phần cứng với mô hình cho các bước kiểm tra thời gian thực và phát triển các hệ thống nhúng.

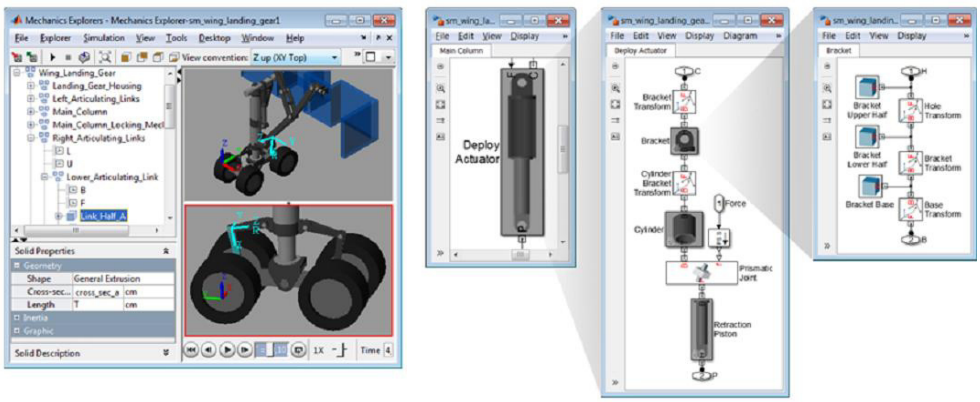


Hình P. 1. Thư viện Simulink

2. SimMechanics

SimMechanics cung cấp môi trường mô phỏng hệ nhiều vật cho các hệ thống cơ khí 3D như robot, hệ thống giảm xóc xe, các thiết bị xây dựng cũng như các hệ thống hạ cánh của máy bay. Người sử dụng mô hình hóa hệ nhiều vật với các khối đại diện cho các khâu, khớp, liên kết, và các lực thành phần, phần việc tiếp theo của SimMechanics là thiết lập và giải các phương trình động học cho toàn bộ cơ hệ. Mô hình xây dựng từ các chương trình thiết kế với sự hỗ trợ của máy tính CAD bao gồm các yếu tố như khối lượng, quán tính, khớp, liên kết và thiết kế hình học 3D đều có thể được đưa vào và xử lý trong SimMechanics. Một mô phỏng 3D của mô hình động lực học được thiết lập một cách tự động cho phép người sử dụng quan sát chuyển động của mô hình.

Người sử dụng có thể thiết lập các thông số đặc trưng cho mô hình với các biến cũng như các phương pháp biểu diễn trong MATLAB, đồng thời thiết kế hệ điều khiển cho mô hình hệ nhiều vật trong Simulink. Bên cạnh đó, các thành phần điện, thủy lực, khí nén… cũng có thể được đưa vào mô hình cơ khí bằng Simscape và kiểm tra tất cả các thành phần trong một mô hình mô phỏng riêng. Để triển khai các mô hình thiết kế trong các môi trường mô phỏng khác, bao gồm cả kỹ thuật phần cứng trong vòng lặp HIL (hardware-in-the-loop), SimMechanics hỗ trợ người dùng thiết lập các đoạn mã C với SimulinkCoder.

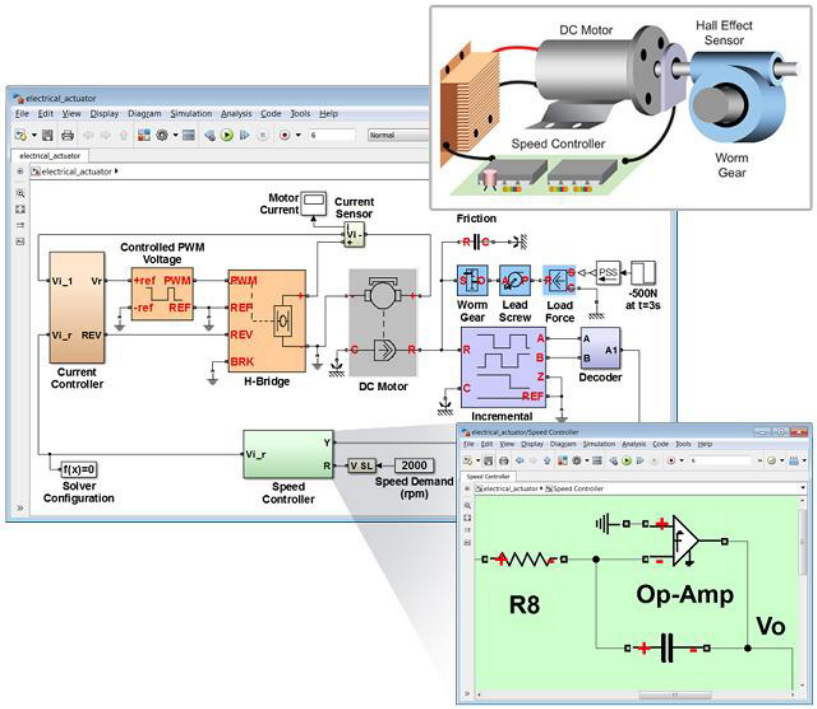


Hình P. 2. Mô hình bộ phận hạ cánh của máy bay trong SimMechanics

3. SimElectronics

SimElectronics cung cấp các thư viện thành phần cho phép mô hình và mô phỏng các hệ thống điện tử và cơ điện tử. Các thư viện bao gồm mô hình bán dẫn, động cơ, cảm biến và các cơ cấu chấp hành… Các mô hình này có thể được sử dụng để phát triển các hệ cơ điện chấp hành cũng như xây dựng các mô hình hành vi để đánh giá kiến trúc mạch tương tự trong Simulink.

Các mô hình SimElectronics có thể được sử dụng để phát triển các thuật toán điều khiển trong các hệ thống điện tử và cơ điện tử, bao gồm cả điện tử thân xe, cơ cấu tùy động máy bay, hệ thống khuếch đại. Các mô hình bán dẫn bao gồm các ảnh hưởng phi tuyến và nhiệt động lực, cho phép người sử dụng lựa chọn các yếu tố trong các bộ khuếch đại, bộ biến đổi tương tự - số, giai đoạn khóa lặp, và các mạch khác… Tương tự như SimMechanics, SimElectronics cũng hộ trợ người dùng trong việc đưa vào các thành phần bên ngoài như điện, thủy lực, khí nén… và triển khai các mô hình thiết kế trong các môi trường mô phỏng khác với Simscape và SimulinkCoder.

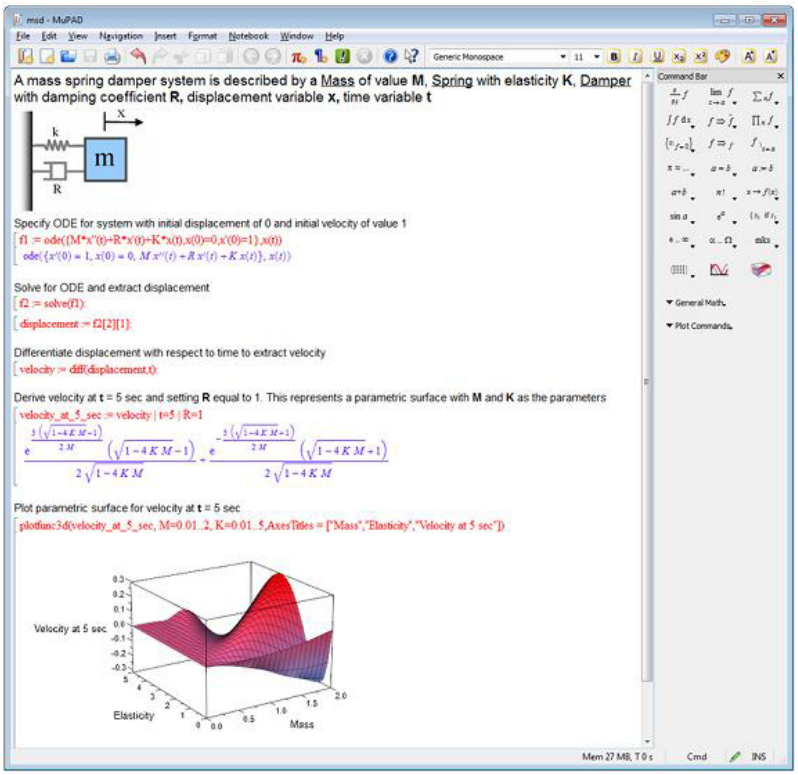


Hình P. 3. Mô hình cơ điện

4. Symbolic Math Toolbox

Bộ công cụ biến ký hiệu cung cấp các hàm để giải và biến đổi các biểu thức toán học biểu tượng và các phép toán số học. Người dùng có thể thực hiện các phép tính tích phân, đạo hàm, tối giản, biến đổi và giải phương trình. Người dùng đồng thời cũng có thể viết các đoạn mã cho MATLAB, Simulink, và Simscape từ các biểu thức toán học với sự trợ giúp của bộ công cụ này.

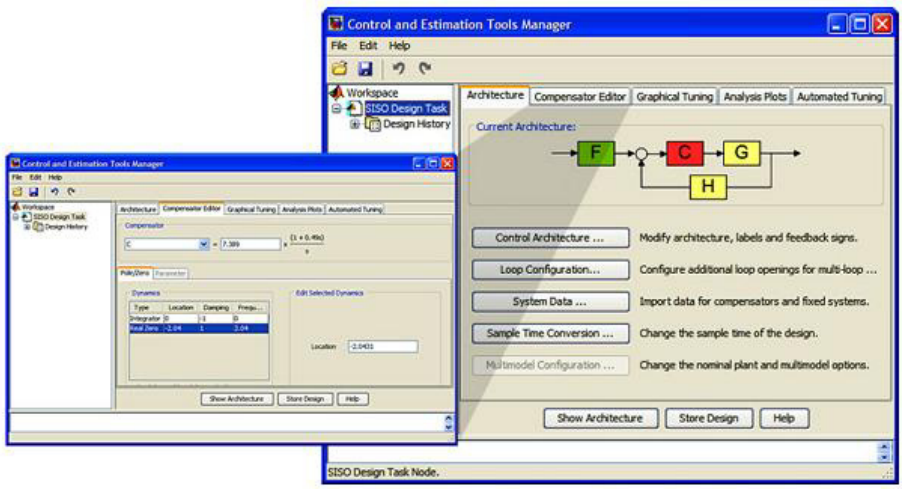
Bộ công cụ biến ký hiệu sử dụng ngôn ngữ MuPAD, được tối ưu hóa cho các thao tác và xử lý các biểu thức toán học ký hiệu. Nó cung cấp các thư viện hàm của MuPAD trong các lĩnh vực toán học thông dụng như giải tích và đại số tuyến tính và trong các lĩnh vực chuyên ngành như lý thuyết số và tổ hợp. Ta cũng có thể viết tùy chỉnh chức năng biểu tượng và các thư viện bằng ngôn ngữ MuPAD. Ứng dụng MuPAD Notebook cho phép người dùng thiết lập các văn bản toán học các đoạn ký tự nhúng, đồ họa, và bộ ký hiệu toán học. Những văn bản này có thể được chia sẻ dưới dạng HTML hoặc PDF. (giao diện MuPAD Notebook)



Hình P. 4. Giao diện MuPAD Notebook

5. Control System Toolbox

Bộ công cụ hệ thống điều khiển cung cấp các thuật toán tiêu chuẩn công nghiệp và các ứng dụng cho hệ thống phân tích, thiết kế, và điều chỉnh các hệ thống điều khiển tuyến tính. Người sử dụng có thể định nghĩa hệ thống dưới dạng hàm truyền, không gian trạng thái, mô hình điểm không – điểm cực, hoặc mô hình đáp ứng tần số. Các ứng dụng và hàm đi kèm như đáp ứng bước và giản đồ Bode, cho phép người sử dụng hình dung hệ thống trong miền thời gian và miền tần số. Bộ công cụ cũng cho phép điều chỉnh các thông số bù sử dụng bộ điều chỉnh PID, quỹ tích nghiệm số, thiết kế LQR/LQG, các kỹ thuật tương tác và tự động khác. Người dùng có thể kiểm thử thiết kế của mình bằng cách kiểm tra thời gian quá độ, độ vọt lố, thời gian xác lập, độ dự trữ biên độ và biên pha cũng như các yêu cầu khác.



Hình P. 5. Giao diện Control and Estimation Tools Manager

6. Image Processing Toolbox

Bộ công cụ xử lý ảnh cung cấp một tập hợp toàn diện các thuật toán tham khảo tiêu chuẩn, hàm và các ứng dụng phục vụ xử lý ảnh, phân tích, trực quan, và phát triển thuật toán. Người sử dụng có thể thực hiện các bước tăng cường ảnh, xóa mờ, phát hiện các đặc điểm, giảm nhiễu, phân vùng ảnh, biến đổi hình học và kết

hợp đa ảnh. Nhiều chức năng của bộ công cụ được đa luồng để tận dụng ưu điểm của các máy tính đa lõi và đa vi xử lý.

Bộ công cụ xử lý ảnh hỗ trợ làm việc trên một tập hợp đa dạng của các loại nguồn ảnh, bao gồm cả giải tương phản động mở rộng, độ phân giải mức gigapixel, chụp cắt lớp… Chức năng trực quan cho phép người dùng quan sát hình ảnh, kiểm tra các vùng của các điểm ảnh, điều chỉnh độ tương phản, tạo đường nét hoặc biểu đồ, thao tác trên các vùng quan tâm (Region of Interests ROIs). Với các thuật toán mà bộ công cụ hỗ trỡ, ta có thể khôi phục lại hình ảnh bị suy thoái, phát hiện và đo lường các thuộc tính, phân tích hình dạng và kết cấu cũng như điều chỉnh cân bằng màu sắc.

Hình P. 6. Ứng dụng Image Processing Toolbox trong việc xác định ô tô từ video giao thông

TÀI LIỆU THAM KHẢO

Nguyễn Phùng Quang. *MATLAB&SIMULINK dành cho kỹ sư điều khiển tự động,* Nhà xuất bản Khoa học và Kỹ thuật, Hà Nội, 2004.

Stormy Attaway. *MATLAB: A Practical Introduction to Programming and Problem Solving, 2nd Edition.* Butterworth - Heinemann Newton, MA, USA, 2011.

Patrick Marchand, O. Thomas Holland. *Graphics and GUIs with MATLAB.* CRC Press, Inc. Boca Raton, FL, USA, 2003.

Dan B. Marghitu. *Mechanisms and Robots Analysis with MATLAB.* Springer Publishing Company, Incorporated, 2009.

R. V. Dukkipati. *MATLAB: An Introduction with Applications.* New Age International Publishers, New Delhi, India, 2010.

R. V. Dukkipati, J. Srinivas. *Solving Engineering Mechanics Problems with MATLAB*. New Age International Publishers, New Delhi, India, 2007.

R. V. Dukkipat. *Analysis & Design of Control Systems Using MATLAB.* New Age International (P) Limited New Delhi, India, 2009.

Giáo trình MATLAB: Interactive Course của trường **Eindhoven University of Technology, Hà Lan.**

Giáo trình MATLAB: A Guide to Using MATLAB in Statics của trường **Arizona State University, Mỹ.**

Giáo trình MATLAB: Sample Problems from Solving Statics Problems in MATLAB của trường **Ohio State University, Mỹ.**

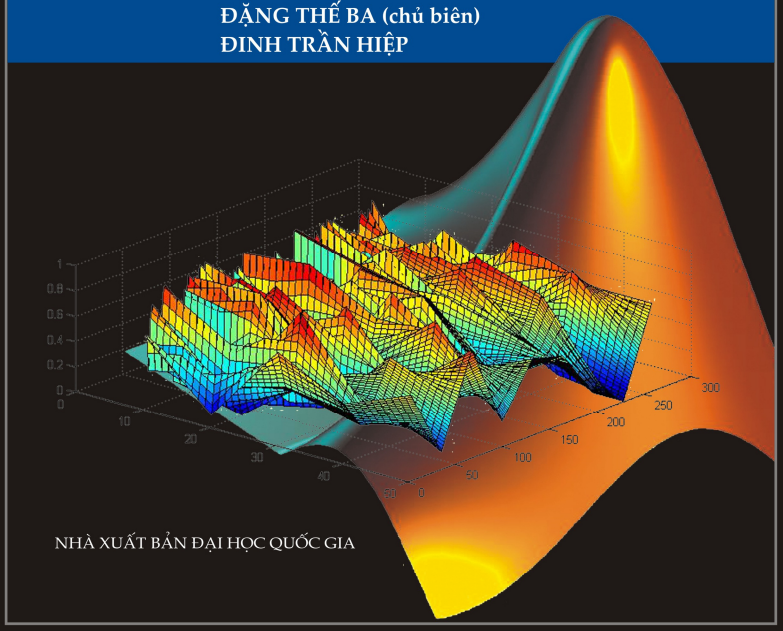
Giáo trình MATLAB của trường **Texas A&M University, Mỹ.**

DANH MỤC HÀM

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ

MATLAB
VÀ ỨNG DỤNG TRONG CƠ KỸ THUẬT

ĐẶNG THẾ BA (chủ biên)
ĐINH TRẦN HIỆP



NHÀ XUẤT BẢN ĐẠI HỌC QUỐC GIA