

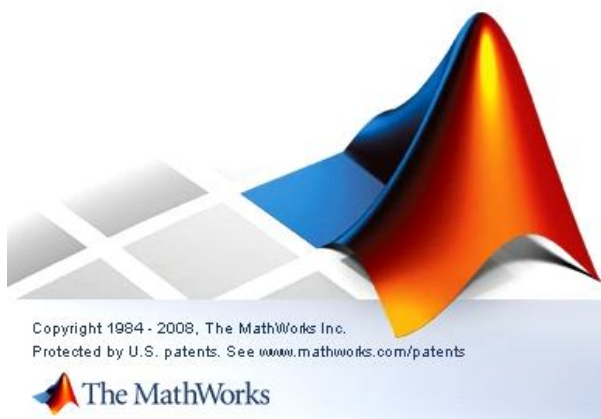
TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA CÔNG NGHỆ THÔNG TIN

ThS. Thái Duy Quý

BÀI GIẢNG TÓM TẮT
MATLAB CĂN BẢN

Dành cho sinh viên khối tự nhiên – công nghệ
(Lưu hành nội bộ)

MATLAB[®]
The Language of Technical Computing



Đà Lạt 2013

LỜI NÓI ĐẦU

Giáo trình “Matlab căn bản” được biên soạn theo chương trình đào tạo hệ thống tín chỉ của trường Đại Học Đà Lạt. Mục đích biên soạn giáo trình nhằm cung cấp cho sinh viên khối tự nhiên, đặc biệt là sinh viên ngành Vật lý hạt nhân những kiến thức cơ bản về các phương pháp xử lý trên ngôn ngữ Matlab.

Đây là học phần lần đầu tiên được triển khai, giảng dạy, tuy tác giả cũng có nhiều cố gắng trong công tác biên soạn nhưng chắc chắn giáo trình còn có nhiều thiếu sót. Tác giả xin trân trọng tiếp thu tất cả những ý kiến đóng góp của các bạn sinh viên, các đồng nghiệp trong lĩnh vực này để hoàn thiện giáo trình, phục vụ tốt hơn cho việc dạy và học cho sinh viên.

Đà Lạt, Tháng 08 năm 2013

Thái Duy Quý

MỤC LỤC

CHƯƠNG 1. GIỚI THIỆU MATLAB	4
1. Giới thiệu.....	4
2. Khởi động và chuẩn bị thư mục làm việc trong Matlab.....	4
3. Quản lý không gian làm việc của Matlab.....	5
4. Các thành phần của Matlab	6
5. Các phím tắt cơ bản trong Matlab.....	7
6. Các toán tử cơ bản của Matlab:	8
CHƯƠNG 2. MATLAB CƠ BẢN.....	11
1. Nhập xuất dữ liệu từ dòng lệnh.....	11
2. Nhập xuất dữ liệu từ bàn phím:	11
3. Nhập xuất dữ liệu từ file:.....	12
4. Các hàm toán học:.....	13
5. Các phép toán trên ma trận và vector:.....	16
6. Tạo số ngẫu nhiên:	18
7. Các lệnh dùng lập trình:	19
CHƯƠNG 3. XỬ LÝ ĐỒ THỊ TRONG MATLAB	22
1. Khái niệm chung.....	22
2. Các lệnh vẽ	22
3. Tạo hình vẽ.....	22
4. Đặc tả kiểu đường vẽ	23
5. Đặc tả màu và kích thước đường vẽ.....	23
6. Thêm đường vẽ vào đồ thị đã có.....	25
7. Chỉ vẽ các điểm số liệu	25
8. Vẽ các điểm và đường.....	26
9. Vẽ với hai trục y	26
10. Vẽ đường cong với số liệu 3D	27
11. Đặt các thông số cho trục	27
12. Ghi nhãn lên các trục tọa độ	28
13. Định vị văn bản trên hình vẽ.....	29
14. Đồ hoạ đặc biệt	30
15. Đồ hoạ 3D	37
16. Vẽ các vector	40
CHƯƠNG 4. LẬP TRÌNH GIAO DIỆN NGƯỜI DÙNG (GUI)	44
1. Cách thực hiện	44
2. Lập trình giao diện với Blank GUI	45
3. Kéo thả và thiết lập thuộc tính cho các điều khiển	46
4. Viết lệnh cho chương trình	47
5. Các tính chất của các điều khiển trong GUIDE Matlab	49
6. Tổng quan về hàm Callback trong lập trình GUI.....	50
7. Chương trình Calculator	52
CHƯƠNG 5. MỘT SỐ PHƯƠNG PHÁP XỬ LÝ TÍNH TOÁN TRÊN MATLAB.....	55
1. Tính định thức của ma trận.....	55
2. Nghịch đảo ma trận bằng cách dùng Minor.....	59
3. Nghịch đảo ma trận bằng thuật toán gauss-Jordan.....	60
4. Lập trình giao diện: Giải phương trình bậc 2	61
PHỤ LỤC	66


CHƯƠNG I. GIỚI THIỆU MATLAB

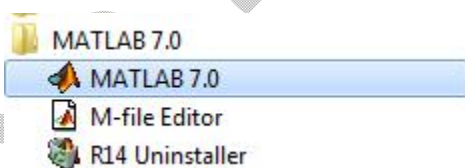
1. Giới thiệu

Matlab là từ viết tắt của Matrix Laboratory, Matlab là một ngôn ngữ lập trình cấp cao dạng thông dịch, nó là môi trường tính toán số được thiết kế bởi công ty MathWorks. Matlab cho phép thực hiện các phép tính toán số, ma trận, vẽ đồ thị hàm số hay biểu diễn thông tin (dưới dạng 2D hay 3D), thực hiện các thuật toán và giao tiếp với các chương trình của các ngôn ngữ khác một cách dễ dàng.

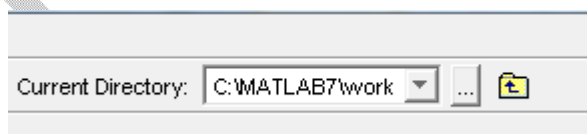
2. Khởi động và chuẩn bị thư mục làm việc trong Matlab

Trước khi khởi động Matlab, thì người dùng nên tạo một thư mục làm việc để chứa các file chương trình của mình (Ví dụ: D:\ThucHanh_DSP). Matlab sẽ thông dịch các lệnh được lưu trong file có dạng *.m

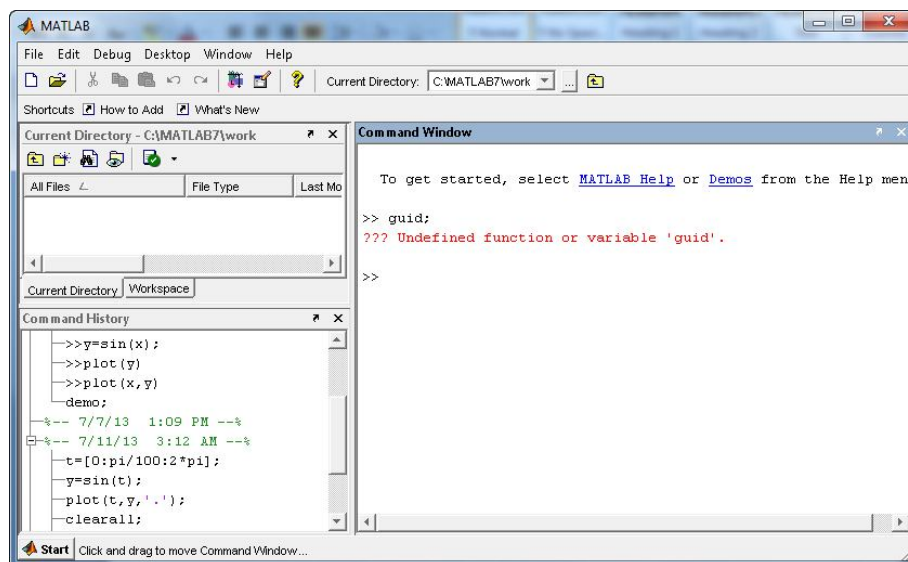
Sau khi đã cài đặt Matlab thì việc khởi chạy chương trình này chỉ đơn giản là nhấp vào biểu tượng của nó trên desktop , hoặc vào Start\All Programs\Matlab 7.0.4\ Matlab.



Sau khi đã khởi động xong Matlab, thì bước kế tiếp là chỉ thư mục làm việc của mình cho Matlab. Nhấp vào biểu tượng trên thanh công cụ và chọn thư mục làm việc của mình (ví dụ: D:\ThucHanh_Matlab).



3. Quản lý không gian làm việc của Matlab



Không gian làm việc của Matlab gồm có các phần cơ bản sau:

***Nút Start:** ở góc dưới bên trái của màn hình, cho phép chạy các ứng dụng mẫu (demos), các công cụ và cửa sổ chưa hiển thị khi khởi động Matlab.

Ví dụ : Start/Matlab/Demos và chạy một ứng dụng mẫu.

* **Cửa sổ lệnh:** Quá trình khởi động đưa người dùng đến Cửa sổ lệnh, nơi các dòng lệnh được biểu thị bằng dấu '>>'. Đây là dấu hiệu cho thấy Matlab đang chờ đánh một (câu) lệnh. Có thể xóa trắng toàn bộ cửa sổ lệnh bằng lệnh: >> *clc* hoặc vào Edit/ Clear Command Window. Khi thực hiện lệnh này, toàn bộ giá trị của các biến hiện có không thay đổi hay mất đi.

* **Cửa sổ không gian làm việc (workspace):** Nơi lưu giữ các biến và dữ liệu do người dùng nhập vào ngoại trừ những biến cục bộ thuộc về một M-file.

Dùng lệnh 'who' hoặc 'whos' để liệt kê các biến hiện có trong không gian làm việc. Để biết giá trị của biến, ta gõ tên biến tại dấu nhắc lệnh. Để xóa một hàm hoặc biến khỏi không gian làm việc, sử dụng lệnh 'clear':

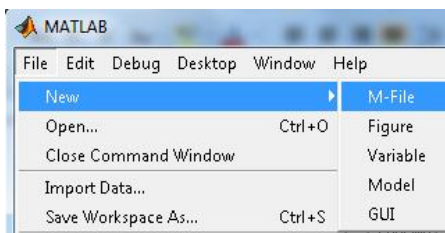
```
>> clear tên_biến;
```


* **Cửa sổ biên tập mảng (ma trận nói chung):** Khi đã có một mảng, có thể chỉnh sửa, biên tập lại bằng Array Editor. Công cụ này làm việc như một bảng tính (spreadsheet) cho ma trận.

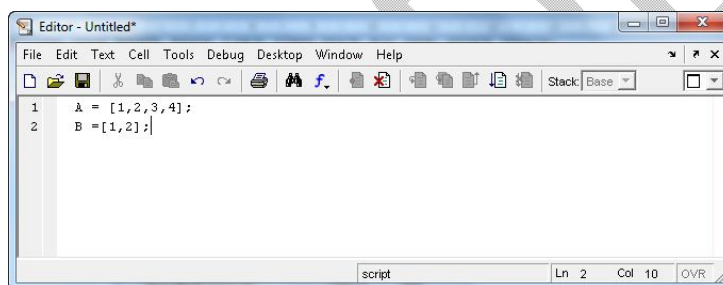
* Cửa sổ địa chỉ thư mục hiện thời: Thư mục hiện thời là nơi chương trình Matlab sẽ tìm các M-file, và các file không gian làm việc (.mat files) đã tải và lưu lại.

Để tạo một file.m trong thư mục làm việc bạn đọc có thể thực hiện:

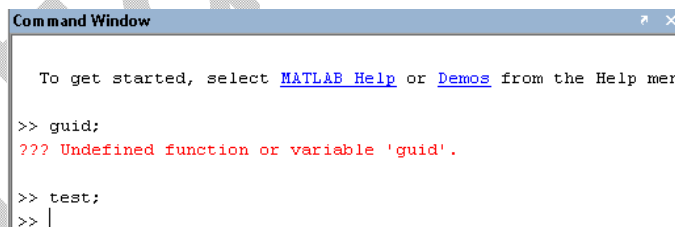
- Nhấp vào biểu tượng  hoặc vào **File|New|M-File**:



- Cửa sổ soạn thảo xuất hiện, gõ chương trình cần thiết vào file. Sau khi đã hoàn tất nhấn vào biểu tượng  để lưu vào thư mục hiện tại (D:\ThucHanh_Matlab).



Để thực thi tập lệnh có trong file.m trong thư mục làm việc thì người dùng chỉ cần gõ tên file đó và Matlab sẽ tự động thực thi các dòng lệnh có trong file.m này (ví dụ để thực thi các lệnh có trong file test.m, chỉ cần gõ lệnh test).



4. Các thành phần của Matlab

- *Ngôn ngữ Matlab*: là một ngôn ngữ ma trận/mảng cấp cao với các câu lệnh, hàm, cấu trúc dữ liệu, vào/ra, các tính năng lập trình hướng đối tượng. Nó cho phép lập trình các ứng dụng từ nhỏ đến các ứng dụng lớn và phức tạp.

- *Môi trường làm việc Matlab*: Đây là một bộ các công cụ và phương tiện mà bạn sử dụng với tư cách là người dùng hoặc người lập trình Matlab. Nó bao gồm các phương tiện cho việc quản lý các biến trong không gian làm việc Workspace cũng như xuất nhập khẩu dữ liệu. Nó cũng bao gồm các công cụ phát triển, quản lý, gỡ rối và định hình M-file, ứng dụng của Matlab.

- *Xử lý đồ họa*: Đây là hệ thống đồ họa của Matlab. Nó bao gồm các lệnh cao cấp cho trực quan hoá dữ liệu hai chiều và ba chiều, xử lý ảnh, ảnh động,... Nó cũng cung cấp các lệnh cấp thấp cho phép bạn tùy biến giao diện đồ họa cũng như xây dựng một giao diện đồ họa hoàn chỉnh cho ứng dụng Matlab của mình.

- *Thư viện toán học Matlab*: Đây là tập hợp khổng lồ các thuật toán tính toán từ các hàm cơ bản như cộng, sin, cos, số học phức... tới các hàm phức tạp hơn như nghịch đảo ma trận, tìm trị riêng của ma trận, phép biến đổi Fourier nhanh.

- *Giao diện chương trình ứng dụng Matlab API (Application Program Interface)*: Đây là một thư viện cho phép bạn viết các chương trình C và Fortran tương thích với Matlab. **Simulink**, một chương trình đi kèm với Matlab, là một hệ thống tương tác với việc mô phỏng các hệ thống động học phi tuyến. Nó là một chương trình đồ họa sử dụng chuột để thao tác cho phép mô hình hoá một hệ thống bằng cách vẽ một sơ đồ khối trên màn hình. Nó có thể làm việc với các hệ thống tuyến tính, phi tuyến, hệ thống liên tục theo thời gian, hệ gián đoạn theo thời gian, hệ đa biến...

5. Các phím tắt cơ bản trong Matlab.

Trong quá trình soạn thảo lệnh, có thể dùng các phím tắt sau đây:

Ký hiệu phím	Phím tắt	Chức năng
↑	<i>Ctrl-P</i>	Gọi lại lệnh trước đó
↓	<i>Ctrl-N</i>	Gọi lệnh sau
←	<i>Ctrl-B</i>	Lùi lại một kí tự
→	<i>Ctrl-F</i>	Tiến lên một kí tự
Ctrl →	Ctrl-R	Sang phải một từ

Ctrl←	Ctrl-L	Sang phải một từ
home	Ctrl-A	Về đầu dòng
end	Ctrl-E	Về cuối dòng
esc	Ctrl-U	Xoá dòng
del	Ctrl-D	Xoá kí tự tại chỗ con nháy đứng
backspace	Ctrl-H	Xoá kí tự trước chỗ con nháy đứng

6. Các toán tử cơ bản của Matlab:

Các toán tử cơ bản:

+	Cộng	-	Trừ
*	Nhân	/	Chia phải
\	Chia trái	^	Luỹ thừa
‘	Chuyển vị ma trận hay số phức liên hợp		

Các toán tử quan hệ :

<	nhỏ hơn	<=	nhỏ hơn hay bằng
>	lớn hơn	>=	lớn hơn hoặc bằng
==	bằng	~=	không bằng

Các toán tử logic:

&	và		or	~	not
---	----	--	----	---	-----

Các hằng:

π	3.14159265	i	số ảo
j	tương tự i	ϵ	sai số 2-52
$realmin$	số thực nhỏ nhất 2-1022	$realmax$	số thực lớn nhất 21023
inf	vô cùng lớn	NaN	Not a number

Các lệnh cơ bản:

Lệnh	Chức năng
<i>Clear</i>	Xóa tất cả các biến trong bộ nhớ Matlab
<i>clc</i>	Lệnh xóa cửa sổ lệnh (command window)
<i>pause</i>	Chờ sự đáp ứng từ phía người dùng
=	Lệnh gán
%	Câu lệnh sau dấu này được xem là dòng chú thích
<i>input</i>	Lệnh lấy vào một giá trị. Ví dụ: x = input('Nhập giá trị cho x:');
<i>help</i>	lệnh yêu cầu sự giúp đỡ từ Matlab
<i>Save</i>	Lưu biến vào bộ nhớ Ví dụ: Save test A B C (lưu các biến A, B, C vào file test)
<i>Load</i>	Load biến từ file hay bộ nhớ. Ví dụ: Load test

Các lệnh điều khiển cơ bản:

Lệnh	Cú pháp/ Chức năng
If: Rẽ 2 nhánh	IF expression statements ELSEIF expression statements ELSE statements END
Switch: Lệnh rẽ nhiều nhánh	SWITCH switch_expr CASE case_expr, statement,..., statement CASE {case_expr1, case_expr2, case_expr3,...} statement,..., statement ... OTHERWISE,

	statement,..., statement END
Lệnh lặp For	FOR variable = expr, statement,..., statement END
Lệnh lặp While	WHILE expression statements END
Break	Thoát đột ngột khỏi vòng lặp WHILE hay FOR.
Continue	Bỏ qua các lệnh hiện tại, tiếp tục thực hiện vòng lặp ở lần lặp tiếp theo.
Return	lệnh trả về

Một số lệnh cơ bản trên đồ thị:

Lệnh	Chức năng
<i>clf</i>	xóa hình hiện tại
<i>plot(signal)</i>	vẽ dạng sóng tín hiệu signal
<i>stairs(signal)</i>	vẽ tín hiệu signal theo dạng cầu thang.
<i>stem(signal)</i>	vẽ chuỗi dữ liệu rời rạc
<i>bar(signal)</i>	vẽ dữ liệu theo dạng cột
<i>mesh(A)</i>	hiển thị đồ họa dạng 3D các giá trị ma trận

Chi tiết các lệnh sẽ được làm rõ trong các chương tiếp theo.

7. Kết chương

Trong chương này chúng ta đã làm quen với Matlab qua các thao tác cơ bản, Matlab là ngôn ngữ cấp cao, có các chức năng xử lý như một ngôn ngữ bình thường. Ta sẽ tìm hiểu kỹ hơn các chức năng ở các phần kế tiếp.

CHƯƠNG 2. MATLAB CƠ BẢN

1. Nhập xuất dữ liệu từ dòng lệnh

MATLAB không đòi hỏi phải khai báo biến trước khi dùng. MATLAB phân biệt chữ hoa và chữ thường. Các số liệu đưa vào môi trường làm việc của Matlab được lưu lại suốt phiên làm việc cho đến khi gặp lệnh *clear all*. Matlab cho phép ta nhập số liệu từ dòng lệnh.

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

- ✓ Ngăn cách các phần tử của ma trận bằng dấu “,” hay dấu trống
- ✓ Dùng dấu “;” để kết thúc một hàng.
- ✓ Bao các phần tử của ma trận bằng cặp dấu ngoặc vuông []

Ví dụ 2.1: Để nhập các ma trận sau:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 3 & -2 & 5 \\ 1 & 5 & 3 \end{bmatrix} \quad B = [1 \quad 4 \quad -2 \quad 1] \quad C = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}$$

ta dùng các lệnh:

$$A = [1 \ 2 \ 3; 3 \ -2 \ 4; 1 \ 5 \ 3]$$

$$B = [1 \ 4 \ 2 \ 1]$$

$$C = [1; 4; 7]$$

2. Nhập xuất dữ liệu từ bàn phím:

Lệnh *input* cho phép ta nhập số liệu từ bàn phím. Ví dụ: $x = \text{input}('Nhập x:')$

Lệnh *format* cho phép xác định dạng thức của dữ liệu.

Ví dụ 2.2.1:

format rat % số hữu tỉ

format long % số sẽ có 14 chữ số sau dấu phẩy

format long e % số dạng mũ

format hex % số dạng hex

format short e % số dạng mũ ngắn

format short % trở về số dạng ngắn (default)

Một cách khác để hiển thị giá trị của biến và chuỗi là đánh tên biến vào cửa sổ lệnh. Cũng

có thể dùng *disp* và *fprintf* để hiển thị các biến.

Ví dụ:

disp('Tri so cua x = '), disp(x)

Ví dụ 2.2.2: Viết chương trình *vidu_2.2.2.m* như sau:

```
1  %author: Thai Duy Quy
2 - clc
3 - R=input('Nhap vao ban kinh R = ');
4 - CV = R*2*pi;
5 - fprintf('Chu vi duong tron ban kinh %5.1f la %5.2f\n',R,CV);
6 - DT = R*R*pi;
7 - fprintf('Dien tich duong tron ban kinh %5.1f la %5.2f\n',R,DT);
```

Kết quả chạy *vidu_2.2.2* tại dòng lệnh như sau:

```
Nhap vao ban kinh R = 10
Chu vi duong tron ban kinh 10.0 la 62.83
Dien tich duong tron ban kinh 10.0 la 314.16
>> |
```

Trong trường hợp ta muốn nhập một chuỗi từ bàn phím, ta cần phải thêm kí tự *s* vào đối số. Ví dụ:

```
ans = input('Ban tra loi <co> hoac <khong>: ','s');
```

3. Nhập xuất dữ liệu từ file:

MATLAB có thể xử lý hai kiểu file dữ liệu: nhị phân (*.mat) và file ASCII (*.dat). Để lưu các ma trận A, B, C dưới dạng file nhị phân, ta dùng lệnh:

```
save ABC A B C
```

và nạp lại các ma trận A, B bằng lệnh:

```
load ABC A B
```

Nếu muốn lưu số liệu của ma trận B dưới dạng file ASCII ta viết:

```
save b.dat B /ascii
```

Ví dụ 2.3: Viết chương trình trong tập tin *vidu_2.3.m* như sau:

```
clear
```

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

```
B = [3; -2; 1];
```

$C(2) = 2; C(4) = 4$

`disp('Nhan phim bat ky de xem nhap/xuat du lieu tu file')`

`save ABC A B C` % lưu A,B & C dưới dạng MAT-file có tên 'ABC.mat'

`clear('A', 'C')` % xóa A và C khỏi bộ nhớ

`load ABC A C` % đọc MAT - file để nhập A và C vào bộ nhớ

`save b.dat B /ascii` % lưu B dưới dạng file ASCII có tên 'b.dat'

`clear B` % xóa B

`load b.dat` % đọc ASCII

`b`

`x = input('Nhập x:')`

`format short e`

`x`

`format rat, x`

`format long, x`

`format short, x`

4. Các hàm toán học:

a. Các hàm toán học cơ bản:

Ký hiệu	Tên hàm	Ký hiệu	Tên hàm
$\exp(x)$	hàm e^x	$\text{sqrt}(x)$	căn bậc hai của x
$\log(x)$	logarit tự nhiên	$\log_{10}(x)$	logarit cơ số 10
$\text{abs}(x)$	modun của số phức x	$\text{angle}(x)$	argument của số phức a
$\text{conj}(x)$	số phức liên hợp của x	$\text{imag}(x)$	phần ảo của x
$\text{real}(x)$	phần thực của x	$\text{sign}(x)$	dấu của x
Các hàm lượng giác		$\cos(x)$, $\sin(x)$, $\tan(x)$, $\text{acos}(x)$, $\text{asin}(x)$, $\text{atan}(x)$, $\cosh(x)$, $\coth(x)$, $\sinh(x)$, $\tanh(x)$, $\text{acosh}(x)$, $\text{acoth}(x)$, $\text{asinh}(x)$, $\text{atanh}(x)$	

b. Các hàm toán học tự tạo:

Matlab cho phép ta tạo hàm toán học và lưu nó vào một file để dùng như là hàm có sẵn của Matlab.

Ví dụ 2.4: Ta cần tạo hàm: $f_1(x) = \frac{1}{1+8x^2}$

$$\text{và hàm: } f_2(x) = \begin{bmatrix} f_1(x_1, x_2) \\ f_2(x_1, x_2) \end{bmatrix} = \begin{bmatrix} x_1^2 + 4x_2^2 + 5 \\ 2x_1^2 - 2x_1 - 3x_2 - 2.5 \end{bmatrix}$$

Muốn thế ta tạo ra file *f1.m* như sau:

```
function y = f1(x)
```

```
y = 1/(1+8*x^2);
```

và file *f2.m*:

```
function y = f2(x)
```

```
y(1) = x(1)*x(1)+4*x(2)*x(2) -5;
```

```
y(2) = 2*x(1)*x(1)-2*x(1)-3*x(2) -2.5;
```

Khi nhập lệnh *f1(2)* ta có giá trị của hàm *f1* tại $x = 2$. Khi nhập lệnh *f2([2 4])* ta có giá trị của hàm *f2* tại $x_1 = 2$ và $x_2 = 4$. Lệnh *feval('f1', 2)* và *feval('f2', [2 4])* cũng cho kết quả tương tự.

Bài tập: Viết chương trình tạo hàm sau đây:

$$f_3(x) = \begin{bmatrix} f_1(x_1, x_2, x_3) \\ f_2(x_1, x_2, x_3) \\ f_3(x_1, x_2, x_3) \end{bmatrix} = \begin{bmatrix} 2x_1^3 + 3x_2^2 + 4x_3^2 + 10 \\ x_1^3 - 5x_2 + 2x_3^2 + 10x_1 - 4 \\ 4x_1^3 - 5x_2^3 + 2x_3^3 + 10x_1^2 - 4x_3^2 - 12 \end{bmatrix}$$

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

```
f1 = inline('1./(1 + 8*x.^2)', 'x');
```

```
f1([0 1]), feval(f1, [0 1])
```

Ta cũng có thể viết:

```
f1 = '1./(1 + 8*x.^2)';
```

```
x = [0 1];
```

```
eval(f1)
```

Nếu hàm là đa thức ta chỉ cần nhập ma trận các hệ số từ số mũ cao nhất. Ví dụ với đa thức $P_4(x) = x^4 + 4x^3 + 2x + 1$ ta viết:

$$P = [1 \ 4 \ 0 \ 2 \ 1]$$

Để nhân hai đa thức ta dùng lệnh *conv*; để chia 2 đa thức ta dùng lệnh *deconv*. Muốn tính trị số của đa thức ta dùng lệnh *polyval* và lệnh *polyvalm* dùng khi đa thức là ma trận.

c. Các lệnh xử lý hàm:

Lệnh *fplot* vẽ đồ thị hàm toán học giữa các giá trị đã cho. Ví dụ:

```
fplot('f1', [-5 5])
```

```
grid on
```

Cho một hàm toán học một biến, ta có thể dùng lệnh *fminbnd* của Matlab để tìm cực tiểu địa phương của hàm trong khoảng đã cho. Ví dụ:

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

```
x = fminbnd(f, 0.3, 1)
```

Lệnh *fminsearch* tương tự hàm *fminbnd* dùng để tìm cực tiểu địa phương của hàm nhiều biến. Ta có hàm 3 biến lưu trong file *three_var.m* như sau:

```
function b = three_var(v)
```

```
x = v(1);
```

```
y = v(2);
```

```
z = v(3);
```

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

Bây giờ tìm cực tiểu đối với hàm này bắt đầu từ

$x = -0.6$, $y = -1.2$ và $z = 0.135$

bằng các lệnh:

```
v = [-0.6 -1.2 0.135];
```

```
a = fminsearch('three_var', v)
```

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

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

```
a = fzero(f, -0.2)
```

Zero found in the interval: [-0.10949, -0.264].

```
a =
```

```
-0.1316
```

5. Các phép toán trên ma trận và vector:

a. Khái niệm chung

Giả sử ta tạo ra các ma trận a và b bằng các lệnh:

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

```
b = [3 -2 1];
```

Ta có thể sửa đổi chúng:

```
A = [a; 7 8 9]
```

```
B = [b; [1 0 -1]]'
```

Toán tử ‘ $*$ ’ dùng để chuyển vị một ma trận thực và chuyển vị liên hợp một ma trận phức. Nếu chỉ muốn chuyển vị ma trận phức, ta dùng thêm toán tử ‘ $.$ ’ Nghĩa là phải viết ‘ $.'$ ’. Ví dụ:

```
C = [1 + 2*i 2 - 4*i; 3 + i 2 - 2*j];
```

```
X = C'
```

```
Y = C.'
```

b. Chỉ số

Phần tử ở hàng i cột j của ma trận $m \times n$ có kí hiệu là $A(i, j)$. Tuy nhiên ta cũng có thể tham chiếu tới phần tử của mảng nhờ một chỉ số, ví dụ $A(k)$ với $k = i + (j - 1)m$. Cách này thường dùng để tham chiếu vec tơ hàng hay cột. Trong trường hợp ma trận đầy đủ thì nó được xem là ma trận một cột dài tạo từ các cột của ma trận ban đầu. Như vậy nếu viết $A(5)$ có nghĩa là tham chiếu phần tử $A(2, 2)$.

Để xác định kích thước của một ma trận ta dùng lệnh *length*(trả về kích thước lớn nhất) hay *size*(số hàng và cột). Ví dụ:

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

```
length(c)
```


$$[m, n] = \text{size}(c)$$

c. Toán tử “:”

Toán tử “:” là một toán tử quan trọng của Matlab. Nó xuất hiện ở nhiều dạng khác nhau.

Ví dụ:

- ✓ Lệnh `1:10` tạo một vec tơ hàng chứa 10 số nguyên từ 1 đến 10.
- ✓ Lệnh `100:-7:50` tạo một dãy số từ 100 đến 51, giảm 7 đơn vị mỗi lần.
- ✓ Lệnh `0:pi/4:pi` tạo một dãy số từ 0 đến pi, cách đều nhau pi/4

Các biểu thức chỉ số tham chiếu tới một phần của ma trận. Viết `A(1:k, j)` là tham chiếu đến k phần tử đầu tiên của cột j . Ngoài ra toán tử “:” tham chiếu tới tất cả các phần tử của một hàng hay một cột.

Ví dụ: `B = A(:, [1 3 2])` tạo ra ma trận B từ ma trận A bằng cách đổi thứ tự các cột từ `[1 2 3]` thành `[1 3 2]`.

d. Tạo ma trận bằng hàm có sẵn

Matlab cung cấp một số hàm để tạo các ma trận cơ bản:

Tên hàm	Chức năng	Ví dụ
<code>zeros</code>	Tạo ra ma trận mà các phần tử đều là zeros	<code>z = zeros(2, 4)</code>
<code>ones</code>	tạo ra ma trận mà các phần tử đều là 1	<code>x = ones(2, 3)</code> <code>y = 5*ones(2, 2)</code>
<code>rand</code>	tạo ra ma trận mà các phần tử ngẫu nhiên phân bố đều	<code>d = rand(4, 4)</code>
<code>randn</code>	tạo ra ma trận mà các phần tử ngẫu nhiên phân bố trực giao	<code>e = randn(3, 3)</code>
<code>magic(n)</code>	tạo ra ma trận cấp n gồm các số nguyên từ 1 đến n^2 với tổng các hàng bằng tổng các cột n phải lớn hơn hay bằng 3	
<code>pascal(n)</code>	tạo ra ma trận xác định dương mà các phần tử lấy từ tam giác Pascal.	<code>pascal(4)</code>
<code>eye(n)</code>	tạo ma trận đơn vị.	<code>eye(3)</code>

$eye(m, n)$	tạo ma trận đơn vị mở rộng	$eye(3, 4)$
-------------	----------------------------	-------------

e. Lắp ghép

Ta có thể lắp ghép (*concatenation*) các ma trận có sẵn thành một ma trận mới.

Ví dụ:

$$a = ones(3, 3)$$

$$b = 5 * ones(3, 3)$$

$$c = [a + 2; b]$$

f. Xoá hàng và cột.

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

Ví dụ: Để xoá cột thứ 2 của ma trận b, ta viết:

$$b(:, 2) = []$$

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

g. Các lệnh xử lý ma trận:

- Cộng: $X = A + B$

- Trừ: $X = A - B$

- Nhân: $X = A * B$

$X.*A$ nhân các phần tử tương ứng với nhau

- Chia: $X = A/B$ lúc đó $X*B = A$

$X = A \setminus B$ lúc đó $A*X = B$

$X = A ./ B$ chia các phần tử tương ứng với nhau

- Lũy thừa: $X = A^2$

$$X = A^2$$

- Nghịch đảo: $X = inv(A)$

- Định thức: $d = det(A)$

6. Tạo số ngẫu nhiên:

Matlab có các lệnh tạo số ngẫu nhiên là *rand* và *randn* tạo ra các số ngẫu nhiên theo phân bố Gauss.

- *rand(m, n)*: tạo ra ma trận các số ngẫu nhiên phân bố đồng nhất.

- *randn(m, n)*: tạo ra ma trận các số ngẫu nhiên theo phân bố chuẩn Gauss.

Ví dụ:

rand(3, 3)

randn(3, 3)

7. Các lệnh dùng lập trình:

a. Các phát biểu điều kiện *if*, *else*, *elseif*

Cú pháp của *if*:

if <biểu thức điều kiện>

 <phát biểu>

end

Nếu <biểu thức điều kiện> cho kết quả đúng thì phần lệnh trong thân của *if* được thực hiện. Các phát biểu *else* và *elseif* cũng tương tự.

Ví dụ 2.7: Ta xét chương trình *doantui*. *m* để đoán tuổi như sau:

clc

disp('Xin chao! Han hanh duoc lam quen');

*x = fix(30*rand);*

disp('Tuoi toi trong khoang 0 - 30');

gu = input('Xin nhap tuoi cua ban: ');

if gu < x

disp('Ban tre hon toi');

elseif gu > x

disp('Ban lon hon toi');

else

disp('Ban bang tuoi toi');

end

b. Lệnh rẽ nhánh *switch*

Cú pháp của *switch* như sau :

```

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

```

c. Lệnh lặp while

Vòng lặp while dùng khi không biết trước số lần lặp. Cú pháp như sau:

```

while <biểu thức>
    <phát biểu>
end

```

Xét chương trình in ra chuỗi “Xin chào” lên màn hình với số lần nhập từ bàn phím *xinchao.m* như sau:

```

clc
disp('xin chào');
gu = input('Nhập số lần in: ');
i = 0;
while i ~= gu
    disp(['Xin chào' i]);
    i = i + 1
end

```

d. Lệnh lặp for

Vòng lặp for dùng khi biết trước số lần lặp. Cú pháp như sau:

```

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

```

Ta xây dựng chương trình đoán số *doanso.m*:

```

clc
x = fix(100*rand);

```

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

8. Kết chương

Chương này giúp sinh viên làm quen với một số khái niệm cơ bản về cách lập trình, cách khai báo biến trong Matlab cũng như các cấu trúc cơ bản của ngôn ngữ lập trình bậc cao.

Chương tiếp theo sẽ giúp sinh viên làm quen với các lệnh vẽ đồ thị.

CHƯƠNG 3. XỬ LÝ ĐỒ THỊ TRONG MATLAB

1. Khái niệm chung

Xử lý đồ thị là một trong những công cụ khá mạnh của Matlab. Ngôn ngữ này cung cấp sẵn các hàm cho phép ta vẽ đồ thị 2D và 3D. Trong chương này ta sẽ làm quen với một số lệnh vẽ cho phép thực hiện vẽ đồ thị.

2. Các lệnh vẽ

Matlab cung cấp một loạt hàm để vẽ biểu diễn các vector số liệu cũng như giải thích và in các đường cong này.

- *plot*: đồ họa 2-D với số liệu 2 trục vô hướng và tuyến tính
- *plot3*: đồ họa 3-D với số liệu 2 trục vô hướng và tuyến tính
- *polar*: đồ họa trong hệ tọa độ cực
- *loglog*: đồ họa với các trục logarit
- *semilogx*: đồ họa với trục x logarit và trục y tuyến tính
- *semilogy*: đồ họa với trục y logarit và trục x tuyến tính
- *plotyy*: đồ họa với trục y có nhãn ở bên trái và bên phải

3. Tạo hình vẽ

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

Ví dụ:

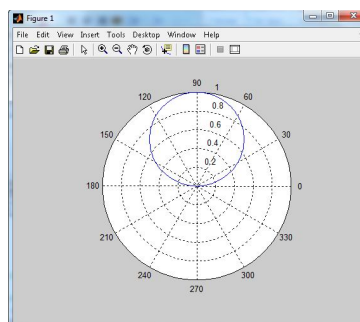
```
t = [0: pi/100: 2*pi]
```

```
y = sin(t);
```

```
plot(t, y)
```

```
grid on
```

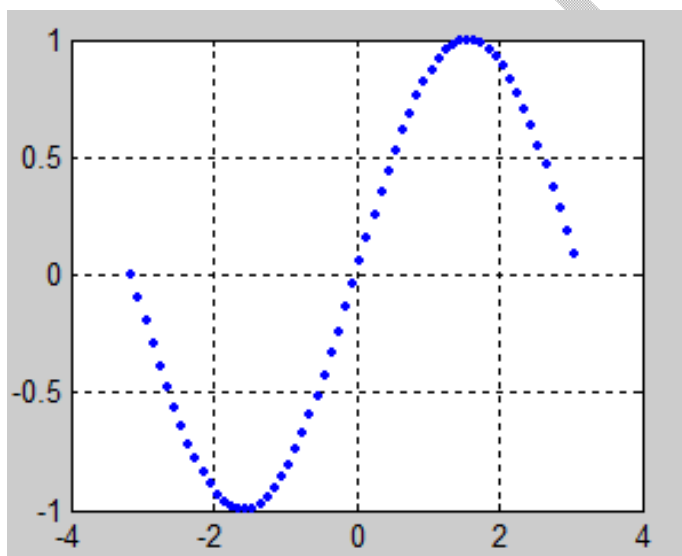
```
polar(t, y)
```



4. Đặc tả kiểu đường vẽ

Ta có thể dùng các kiểu đường vẽ khác nhau khi vẽ hình. Muốn thế ta chuyển kiểu đường vẽ cho hàm plot. Ta viết chương trình tạo ra đồ thị hàm hình *sin*:

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



5. Đặc tả màu và kích thước đường vẽ

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

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

Màu được xác định bằng các tham số:

Mã	Màu	Mã	Màu
R	Red	M	Magenta
G	Green	Y	Yellow
B	Blue	K	Black

C	cyan	W	White
---	------	---	-------

Các dạng điểm đánh dấu xác định bằng:

Mã	Kiểu đánh dấu	Mã	Kiểu đánh dấu
+	Dấu cộng	.	Điểm
o	Vòng tròn	x	Chữ thập
*	Dấu sao	s	Hình vuông
d	Hạt kim cương	v	Điểm tam giác hướng xuống
^	Điểm tam giác hướng lên	<	Tam giác sang trái
>	Tam giác sang phải	h	Lục giác
p	Ngũ giác		

Các dạng đường thẳng xác định bằng:

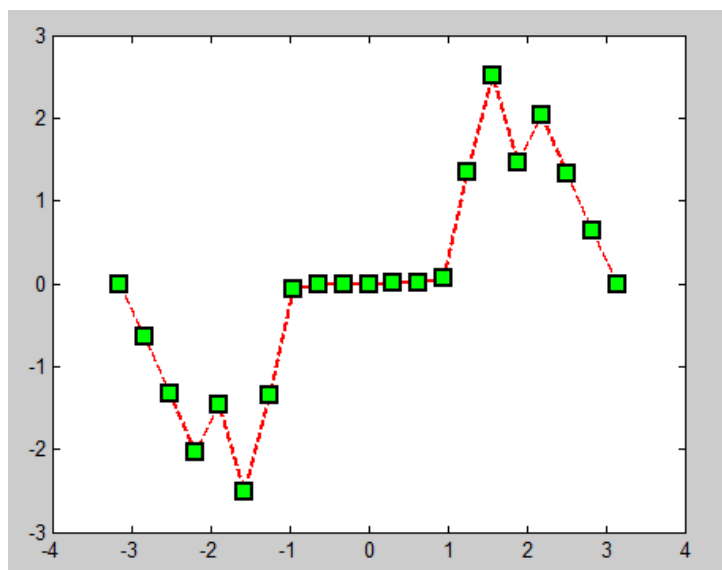
Mã	Kiểu đường	Mã	Kiểu đường
-	Đường liền	:	Đường chấm chấm
--	Đường nét đứt	-.	Đường chấm gạch

Ta xét chương trình *dothi.m* như sau:

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

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

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

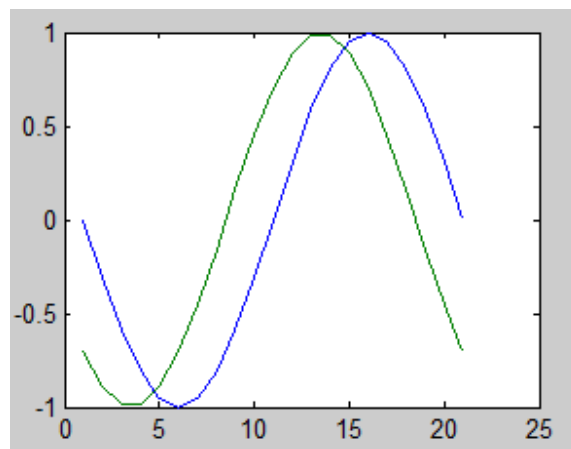


6. Thêm đường vẽ vào đồ thị đã có

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

Ví dụ:

```
plot(sin(x));
hold all
plot(sin(x+(pi/4)));
```



7. Chỉ vẽ các điểm số liệu

Để vẽ các điểm đánh dấu mà không nối chúng lại với nhau ta dùng đặc tả nói rằng không có các đường nối giữa các điểm, nghĩa là ta gọi hàm plot chỉ với đặc tả màu và điểm đánh dấu. Ví dụ: xét chương trình như sau:

```
x = -pi : pi/10 : pi;
```

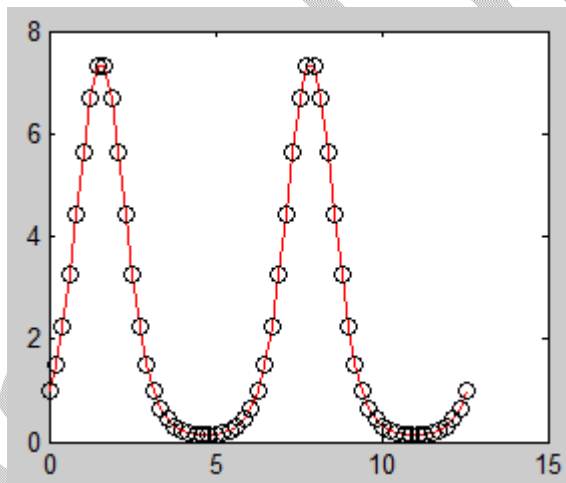
```
y = tan(sin(x)) - sin(tan(x));
plot(x, y, 's', 'MarkerEdgeColor', 'k')
```

8. Vẽ các điểm và đường

Để vẽ cả các điểm đánh dấu và đường nối giữa chúng ta cần mô tả kiểu đường và kiểu điểm. Ta xét đoạn chương trình như sau:

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

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

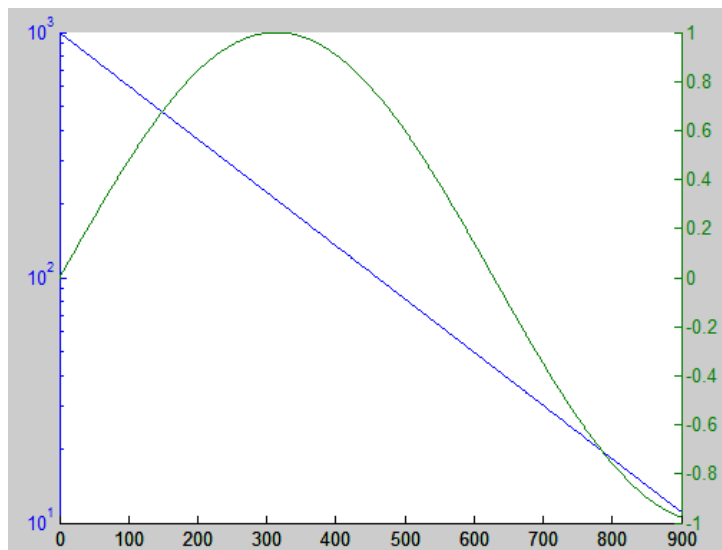


9. Vẽ với hai trục y

Lệnh `plotyy` cho phép tạo một đồ thị có hai trục y. Ta cũng có thể dùng `plotyy` để cho giá trị trên hai trục y có kiểu khác nhau nhằm tiện so sánh. Ta xét chương trình *dothi.m* như sau:

```
t = 0:900;
A = 1000;
b = 0.005;
a = 0.005;
z2 = sin(b*t);
z1 = A*exp(-a*t);
```

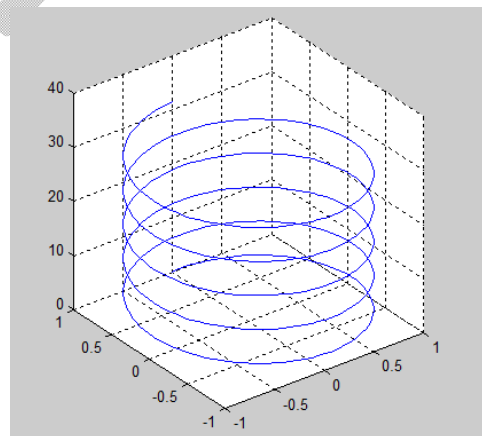
```
[haxes, hline1, hline2] = plotyy(t, z1, t, z2, 'semilogy', 'plot');
```



10. Vẽ đường cong với số liệu 3D

Nếu x, y, z là 3 vec tơ có cùng độ dài thì `plot3` sẽ vẽ đường cong 3D. Ta viết chương trình `duongcong3D.m` như sau:

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



11. Đặt các thông số cho trục

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

- `axis` đặt lại các giá trị trên trục tọa độ

- *axes* tạo một trục tọa độ mới với các đặc tính được mô tả
- *get* và *set*: cho phép xác định và đặt các thuộc tính của trục tọa độ đang có.
- *gca*: trở về trục tọa độ cũ

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

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

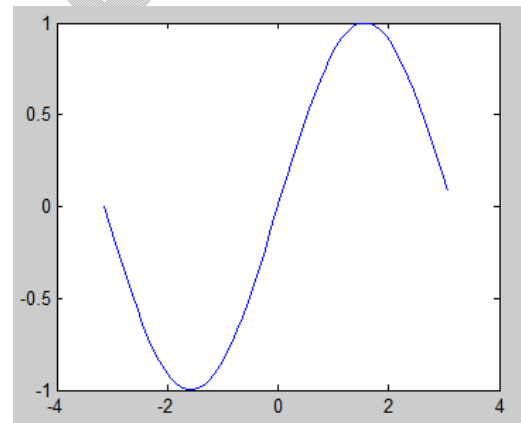
Ta xét chương trình *thongso.m* như sau:

```
x = 0:0.025:pi/2;
plot(x, tan(x), '-ro')
axis([0 pi/2 0 5])
```

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

Ví dụ: xét chương trình như sau:

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



12. Ghi nhãn lên các trục tọa độ

Matlab cung cấp các lệnh ghi nhãn lên đồ họa gồm :

- *title*: thêm nhãn vào đồ họa

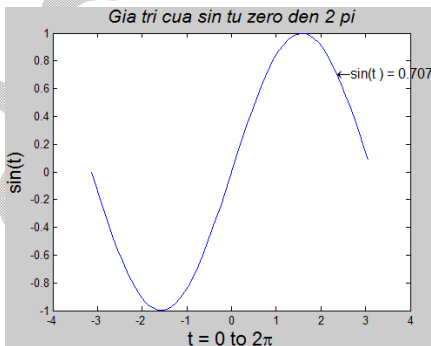
- *xlabel*: thêm nhãn vào trục x
- *ylabel*: thêm nhãn vào trục y
- *zlabel*: thêm nhãn vào trục z
- *legend*: thêm chú giải vào đồ thị
- *text*: hiển thị chuỗi văn bản ở vị trí nhất định

- *gtext*: đặt văn bản lên đồ hoạ nhờ chuột
- *\bf*: bold font
- *\it*: italics font
- *\sl*: oblique font (chữ nghiêng)
- *\rm*: normal font

Các kí tự đặc biệt xem trong String properties của Help.

Ta dùng các lệnh *xlabel*, *ylabel*, *zlabel* để thêm nhãn vào các trục toạ độ. Ta có thể thêm văn bản vào bất kì chỗ nào trên hình vẽ nhờ hàm *text*. Ta có chương trình như sau:

```
x = -pi: .1: pi;
y = sin(x);
plot(x, y)
xlabel('t = 0 to 2\pi', 'FontSize', 16)
ylabel('sin(t)', 'FontSize', 16)
title('\it{Gia tri cua sin tu zero đến 2 pi}', 'FontSize', 16)
text(3*pi/4, sin(3*pi/4), '\leftarrow sin(t) = 0.707', 'FontSize', 12)
```



13. Định vị văn bản trên hình vẽ

Ta có thể sử dụng đối tượng văn bản để ghi chú các trục ở vị trí bất kì. Matlab định vị văn bản theo đơn vị dữ liệu trên trục.

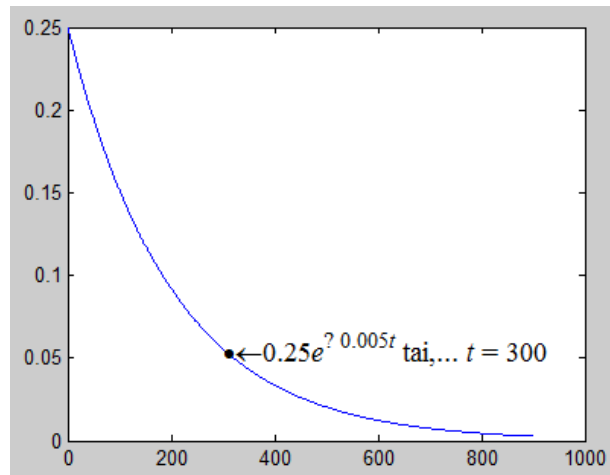
Ví dụ để vẽ hàm $y = Ae^{\alpha t}$ với $A = 0.25$ với $t = 0$ đến 900 và $\alpha = 0.005$ ta viết chương trình như sau:

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

```

plot(t, 0.25*exp(-0.005*t))
plot(t,y)
text(300,.25*exp(-.005*300),...
    '\bullet\leftarrow\fontname{times}0.25\{ite\}^{\{- 0.005\{itt\}} tai,... \{itt\} = 300',
    'FontSize', 14)

```



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

14. Đồ họa đặc biệt

a. Khối và vùng

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

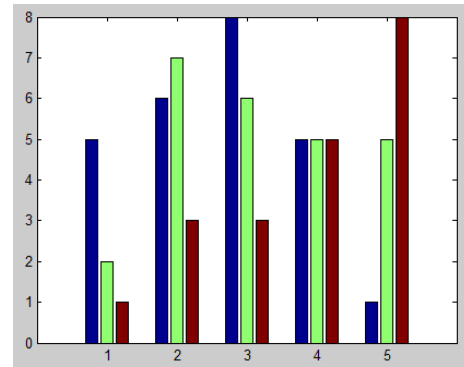
- *bar*: hiển thị các cột của ma trận $m \times n$ như là m nhóm, mỗi nhóm có n bar.
- *barh*: hiển thị các cột của ma trận $m \times n$ như là m nhóm, mỗi nhóm có n bar nằm ngang.
- *bar3*: hiển thị các cột của ma trận $m \times n$ như là m nhóm, mỗi nhóm có n bar dạng 3D.
- *bar3h*: hiển thị các cột của ma trận $m \times n$ như là m nhóm, mỗi nhóm có n bar dạng 3D nằm ngang.

Mặc định, mỗi phần tử của ma trận được biểu diễn bằng một bar. Ta xét chương trình như sau:

```

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

```



b. Mô tả dữ liệu trên trục.

Ta dùng các hàm *xlabel* và *ylabel* để mô tả các dữ liệu trên trục. Ta xét chương trình như sau:

```

nhdo = [29 23 27 25 20 23 23 27];
ngay = 0: 5: 35;
bar(ngay, nhdo)
xlabel('Ngày')
ylabel('Nhiệt độ (^{o}C)')
set(gca, 'YLim', [15 30], 'Layer', 'top')
grid on
set(gca, 'YLim', [15 30])

```

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

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

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

c. Xếp chồng đồ thị

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

Ví dụ:

```

TCE = [515 420 370 250 135 120 60 20];
nhdo = [29 23 27 25 20 23 23 27];
ngay = 0:5:35;

```

```
bar(ngay, nhdo)
xlabel('Ngày')
ylabel('Nhiệt độ (^oC)')
```

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

```
h1 = gca;
và tạo trục thứ 2 ở vị trí trục thứ nhất trước nhất vẽ bộ số liệu thứ 2:
h2 = axes('Position',get(h1,'Position'));
plot(days,TCE,'LineWidth',3)
```

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

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

Để ghi chú lên đồ thị ta viết:

```
text(11,380,'Mat do','Rotation',-55,'FontSize',16)
ylabel('TCE Mat do (PPM)')
title('Xếp chồng đồ thị','FontSize',16)
```

d. Đồ họa vùng.

Hàm area hiển thị đường cong tạo từ một vec tơ hay từ một cột của ma trận. Nó vẽ các giá trị của một cột của ma trận thành một đường cong riêng và tô đầy vùng không gian giữa các đường cong và trục x ta xét chương trình như sau:

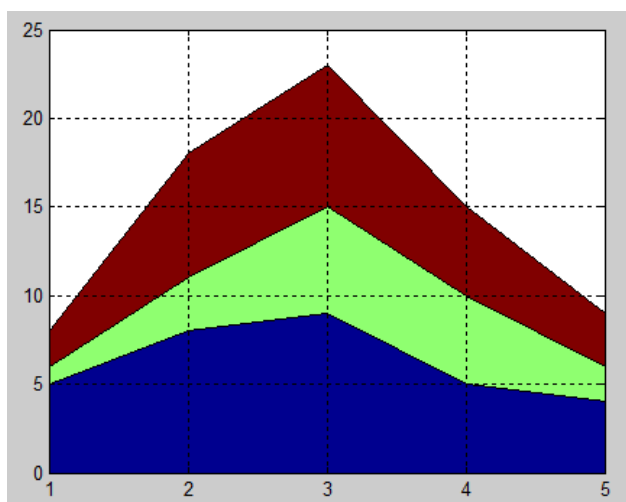
```
Y = [ 5 1 2
      8 3 7
      9 6 8
      5 5 5
      4 2 3];
area(Y)
```


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

```
set(gca,'Layer','top')
```

```
set(gca,'XTick',1:5)
```

```
grid on
```



e. Đồ thị Pie

Đồ thị pie hiển thị theo tỉ lệ phần trăm của một phần tử của một vec tơ hay một ma trận so với tổng các phần tử. Các lệnh *pie* và *pie3* tạo ra đồ thị 2D và 3D ta xét chương trình như sau:

```
X = [19.3 22.1 51.6; 34.2 70.3 82.4; 61.4 82.9 90.8; 50.5 54.9 59.1; 29.4 36.3 47.0];
```

```
x = sum(X);
```

```
explode = zeros(size(x));
```

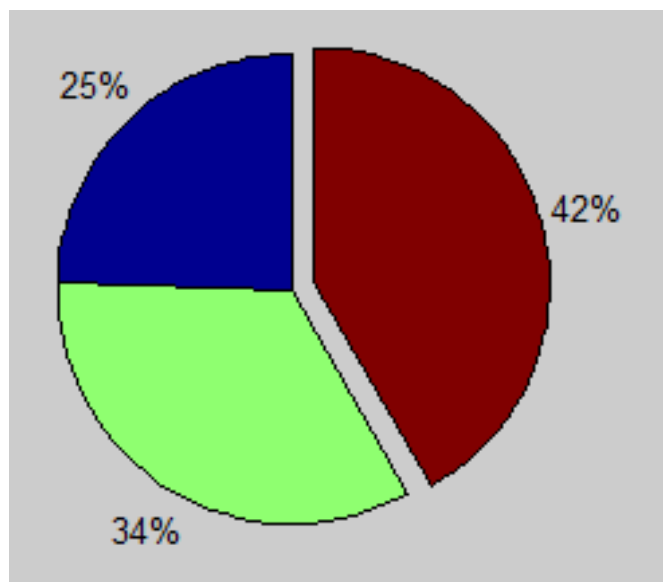
```
[c,offset] = max(x);
```

```
explode(offset) = 1;
```

```
h = pie(x,explode)
```

```
%A = [ 1 3 6];
```

```
%pie3(A)
```



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

```
x = [.19 .22 .41];
```

```
pie(x)
```

f. Làm hình chuyển động:

Có thể tạo ra hình chuyển động bằng 2 cách:

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

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

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

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

```
axis equal
```

```
M = moviein(16, gcf);
```

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

```

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

```

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

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

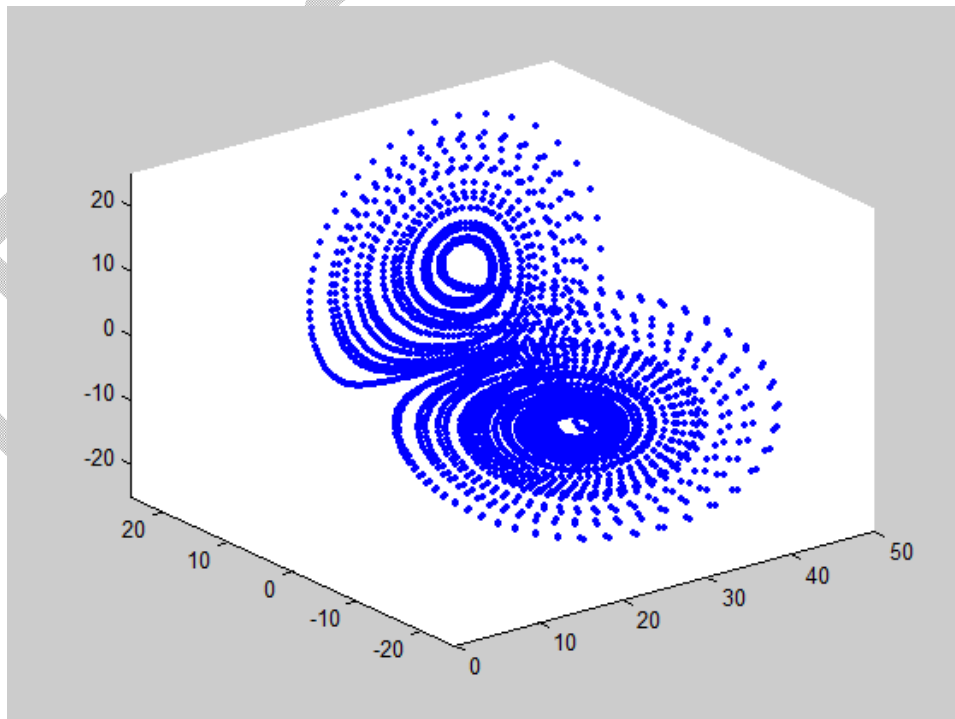
ngăn hàm `plot` đưa tỉ lệ các trục về axis normal mỗi khi nó được gọi. Hàm `getframe` không đổi số trả lại các điểm ảnh của trục hiện hành ở hình hiện có. Mỗi khung hình gồm các số liệu trong một vec tơ cột. Hàm `getframe(gcf)` chụp toàn bộ phần trong của một cửa sổ hiện hành. Sau khi tạo ra hình ảnh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

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

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

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

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



15. Đồ họa 3D

a. Các lệnh cơ bản.

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

b. Đồ thị các hàm hai biến

Bước thứ nhất để thể hiện hàm 2 biến $z = f(x, y)$ là tạo ma trận *x* và *y* chứa các tọa độ trong miền xác định của hàm.

Hàm *meshgrid* sẽ biến đổi vùng xác định bởi 2 vec tơ *x* và *y* thành ma trận *x* và *y*. Sau đó ta dùng ma trận này để đánh giá hàm.

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

```
[x,y] = meshgrid(-8:.5:8);
r = sqrt(x.^2 + y.^2) + 0.005;
```

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

```
z = sin(r)./r;
mesh(z)
```

c. Đồ thị đường đẳng mức.

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

- *clabel*: tạo các nhãn sử dụng ma trận *contour* và hiển thị nhãn
- *contour*: hiển thị các đường đẳng mức tạo bởi một giá trị cho trước của ma trận *Z*.
- *contour3*: hiển thị các mặt đẳng mức tạo bởi một giá trị cho trước của ma trận *Z*.
- *contourc*: hiển thị đồ thị *contour* 2D và tô màu vùng giữa 2 các đường *contourc* hàm cấp thấp để tính ma trận *contour*

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

Ví dụ:

```
[X,Y,Z] = peaks;
```

```
contour(X,Y,Z,20)
```

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

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

```
Z = peaks;
[C,h] = contour(Z,10);
clabel(C,h)
title({'Cac contour co nhan','clabel(C,h)'})
```

Hàm contourf hiển thị đồ thị đường đẳng mức trên một mặt phẳng và tô màu vùng còn lại giữa các đường đẳng mức. Để kiểm soát màu tô ta dùng hàm caxis và colormap. Ta viết chương trình ct1_26.m:

```
Z = peaks;
[C, h] = contourf(Z, 10);
caxis([-20 20])
colormap autumn;
title({'Contour co to mau', 'contourf(Z, 10)'})
```

Các hàm contour(z, n) và contour(z, v) cho phép ta chỉ rõ số lượng mức contour hay một mức contour cần vẽ nào đó với z là ma trận số liệu, n là số đường contour và v là vec tơ các mức contour. MATLAB không phân biệt giữa vec tơ một phần tử hay đại lượng vô hướng. Như vậy nếu v là vec tơ một phần tử mô tả một contour đơn ở một mức hàm contour sẽ coi nó là số lượng đường contour chứ không phải là mức contour. Nghĩa là, contour(z, v) cũng như contour(z, n). Để hiển thị một đường đẳng mức ta cần cho v là một vector có 2 phần tử với cả hai phần tử bằng mức mong muốn. Ví dụ để tạo ra một đường đẳng mức 3D của hàm peaks ta viết chương trình ct1_27.m:

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

```
Z = peaks(X, Y);
```

```
contour3(X, Y, Z)
```

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

```
v = [1 1]
```

```
contour3(X, Y, Z, v)
```

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

Ví dụ: tạo một M-file có tên *Vidu_02.m* như sau:

```
disp('Chuot phai tro cac diem tren duong ve')
```

```
disp('Chuot trai tro diem cuoi cua duong ve')
```

```
axis([0 10 0 10])
```

```
hold on
```

```
x = [];
```

```
y = [];
```

```
n = 0;
```

```
but = 1;
```

```
while but ~= 1
```

```
    [xi,yi,but] = ginput(1);
```

```
    plot(xi, yi, 'go')
```

```
    n = n + 1;
```

```
    x(n, 1) = xi;
```

```
    y(n, 1) = yi;
```

```
end
```

```
t = 1:n;
```

```
ts = 1: 0.1: n;
```

```
xs = spline(t, x, ts);
```

```
ys = spline(t, y, ts);
```

```
plot(xs, ys, 'c-');
hold off
```

16. Vẽ các vector

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

Các hàm vẽ vec tơ gồm:

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

a. Hàm *compass*.

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

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

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

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

và tạo ra ghi chú trên đồ thị:

```
gc = {'Huong gio va suc gio tai san bay Da Nang'}
text(-28, 15, gc)
```

b. Hàm *feather*

Hàm *feather* hiển thị các vector bắt đầu từ một đường thẳng song song với trục x. Ví dụ để tạo ra các vec tơ có góc từ 90 đến 00 và cùng độ dài ta viết chương trình như sau:


```
theta = 90:-10: 0;
```

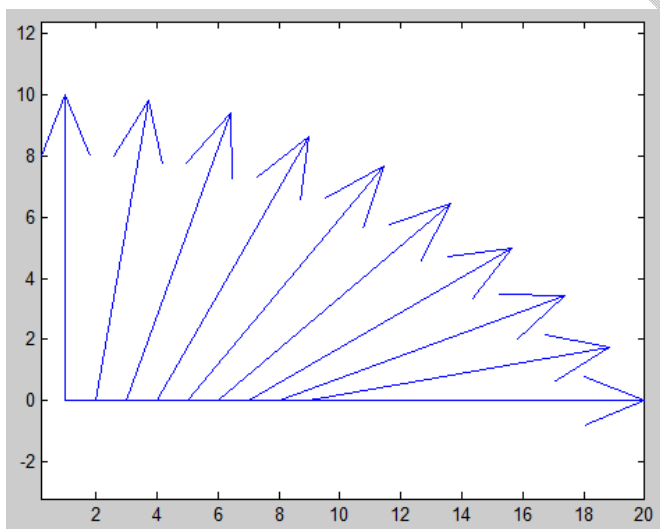
```
r = ones(size(theta));
```

trước khi vẽ, chuyển các số liệu sang toạ độ vuông góc và tăng độ lớn thành r để dễ nhìn:

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

```
feather(u, v)
```

```
axis equal
```



Nếu đối số là số phức z thì *feather* coi phần thực là x và phần ảo là y . Ta xét chương trình như sau:

```
t = 0: 0.3: 10;
```

```
s = 0.05 + i;
```

```
Z = exp(-s*t);
```

```
feather(Z)
```

c. Hàm quiver

Hàm quiver hiển thị các vec tơ ở các điểm đã cho trong mặt phẳng. Các vec tơ này được xác định bằng các thành phần x và y . Ví dụ để tạo ra 10 contour của hàm peaks ta dùng chương trình như sau:

```
n = -2.0: .2: 2.0;
```

```
[X,Y,Z] = peaks(n);
```

```
contour(X, Y, Z, 10)
```

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

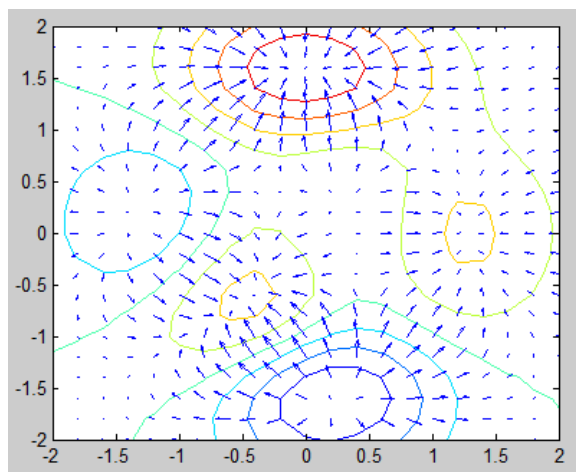
```
[U, V] = gradient(Z, .2);
```

Đặt *hold on* để thêm đường contour:

```
hold on
```

```
quiver(X,Y,U,V)
```

```
hold off
```



d. Hàm *quiver3*.

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

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

Ta viết chương trình *chuyendong.m*, trước hết ta gán vận tốc ban đầu và gia tốc a :

```
v0 = 20; % Van toc ban dau
```

```
a = -32; % gia toc
```

Tiếp theo tính z tại các thời điểm:

```
t = 0:1:1;
```

```
z = v0*t + 1/2*a*t.^2;
```

Tính vị trí theo hướng x và y :

```
vx = 2;
```

```
x = vx*t;
```

```
vy = 3;
```

```
y = vy*t;
```

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

```
u = gradient(x);
```

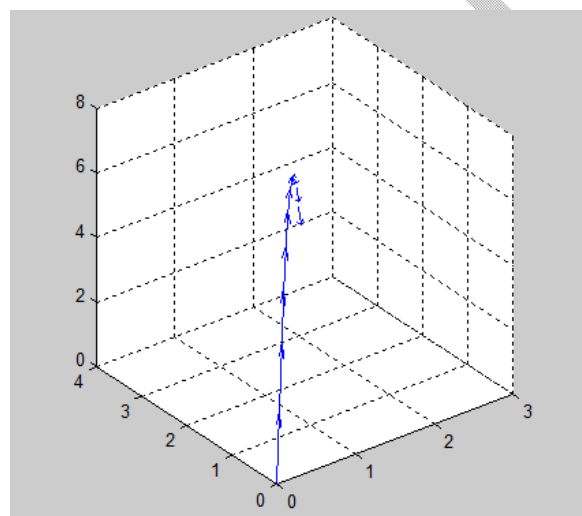
```
v = gradient(y);
```

```
w = gradient(z);
```

```
scale = 0;
```

```
quiver3(x, y, z, u, v, w, scale)
```

```
axis square
```



17. Kết chương

Trong chương này, sinh viên đã làm quen với một số lệnh cơ bản trong vẽ đồ thị dựa trên dữ liệu. Chương cũng giúp sinh viên làm quen với một số lệnh vẽ đồ thị 3D và nâng cao.

Chương tiếp theo sẽ giúp sinh viên làm quen với phương pháp lập trình giao diện đồ họa người dùng trên Matlab

CHƯƠNG 4. LẬP TRÌNH GIAO DIỆN NGƯỜI DÙNG (GUI)

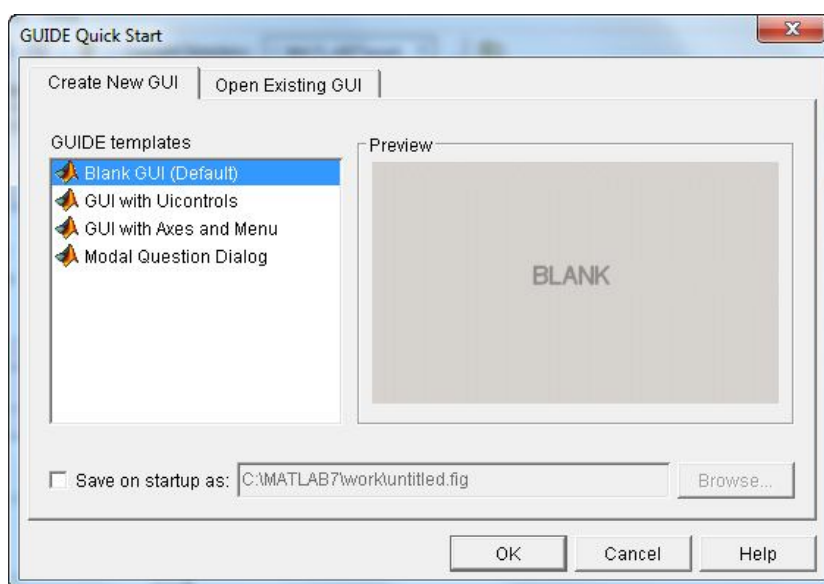
Chương này sẽ hướng dẫn người dùng lập trình bằng cửa sổ, đây là dạng lập trình giao diện người dùng thường thấy ở các ngôn ngữ cấp cao hiện nay.

1. Cách thực hiện

Mở phần mềm Matlab, gõ lệnh sau vào cửa sổ Command:

`>> guide`

Cửa sổ **GUIDE Quick Start** hiện ra như sau:



Trong cửa sổ **GUIDE Quick Start** có nhiều lựa chọn theo các khuôn mẫu như sau:

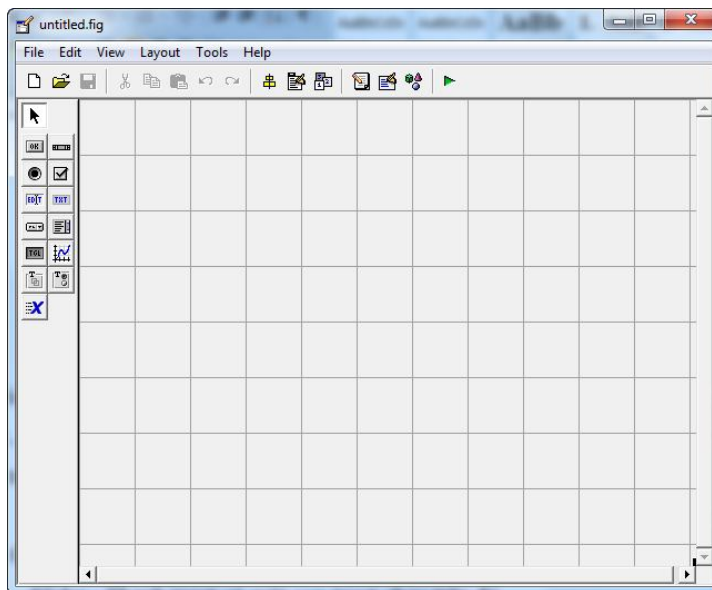
Thẻ Create New GUI: Tạo một hộp thoại GUI mới theo một trong các loại sau:

- *Blank GUI (Default):* Hộp thoại trống, không có một điều khiển uicontrol nào cả.
- *GUI with Uicontrols:* Hộp thoại với một vài uicontrol như button,...
- *GUI with Axes and Menu:* Hộp thoại với một uicontrol axes và button, các menu để hiển thị đồ thị.
- *Modal Question Dialog:* Hộp thoại đặt câu hỏi Yes, No.

Thẻ Open Existing GUI: giúp người dùng mở một project có sẵn đã tạo trước.

Trong hướng dẫn này, khi tạo một project mới sẽ chọn *Blank GUI*

2. Lập trình giao diện với Blank GUI



Giao diện rất giống với các chương trình lập trình giao diện như Visual Basic, Visual C++, C# ... Di chuột qua các biểu tượng ở bên trái sẽ thấy tên của các điều khiển.

Sau đây là một số điều khiển thường dùng:

- **Push Button:** Là các nút bấm như nút OK, Cancel mà ta vẫn bấm (tương tự Button trong các ngôn ngữ bậc cao)
- **Slider:** Thanh trượt có một con trượt chạy trên đó.
- **Radio Button:** Chọn lựa 1 điều kiện trong một tập điều kiện.
- **Check Box:** Chọn lựa nhiều điều kiện trong một tập điều kiện.
- **Edit Text:** Là một ô cho người dùng nhập văn bản
- **Static Text:** Là ô hiển thị văn bản của người dùng
- **Pop-up Menu:** Người dùng chọn một đối tượng trong danh sách sổ xuống.
- **List Box:** Cho phép người dùng chọn 1 đối tượng trong danh sách hiện ra.
- **Axes:** Thực hiện vẽ hệ trục tọa độ.
- **Panel:** Là một dạng phân ô của các thành phần trên giao diện.
- **Button Group:** Nhóm các button.
- **ActiveX Control:** Các Control do người dùng nhúng thêm vào.
- **Toggle Button:** Dạng nút bấm bật - tắt.

Phía trên cùng là menu, trọng nhất là menu Tools có:

- **Run (Ctrl + T):** Chạy chương trình đã viết, sẽ báo lỗi nếu chương trình chứa lỗi. Để chạy được chương trình, hệ thống bắt buộc người dùng phải lưu lại ứng dụng.
- **Align Object:** dùng để sắp xếp các điều khiển và căn lề cho các đối tượng.
- **Grid and Rulers:** Hiển thị lưới và thước trong giao diện.
- **Menu Editor:** Tạo menu cho ứng dụng.
- **Tab Order Editor:** sắp xếp thứ tự các xuất hiện chuột lên đối tượng khi nhấn phím Tab lúc chạy ứng dụng.
- **Gui Options:** Một số lựa chọn cho giao diện GUI.

Khi ta lưu lại (vào *File\Save* hoặc nhấn *Ctrl + S*) với một tên nào đó (ví dụ: *vidu*), khi đó hệ thống đồng thời xuất hiện hai cửa sổ là **cửa sổ soạn thảo** và cửa sổ thiết kế. Trong thư mục vừa lưu sẽ có hai tập tin:

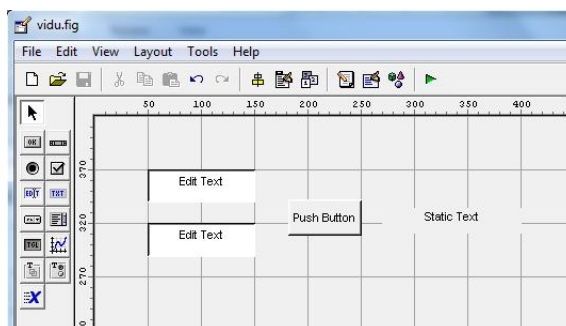
- *vidu.fig*: Tập tin này chứa giao diện của chương trình, là nơi cho người dùng thiết kế bằng cách kéo thả các điều khiển để tạo ứng dụng.
- *vidu.m*: tập tin này chứa các mã thực thi cho chương trình như các hàm khởi tạo, các hàm callback...

3. Kéo thả và thiết lập thuộc tính cho các điều khiển

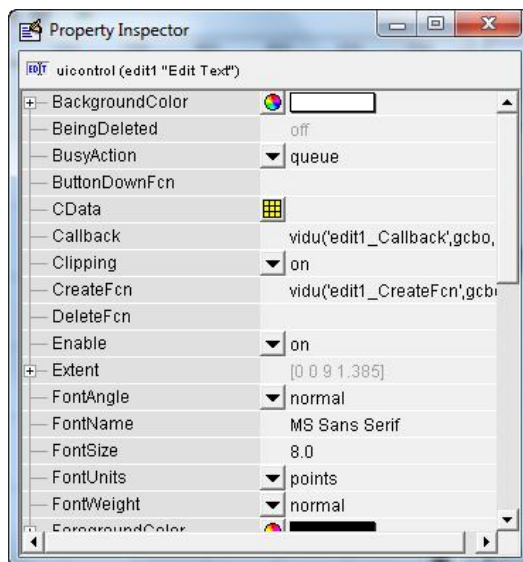
Ví dụ: Thực hiện chương trình cho người dùng nhập vào hai số, khi nhấn nút sẽ hiển thị kết quả phép tính cộng của 2 số vừa nhập.

Thực hiện các bước như sau:

- Tạo một Blank GUI: Vào *File\New\GUI* chọn *Blank GUI*, đặt tên là *vidu.fig*
- Kéo thả vào trong giao diện 2 *Edit Text*, 1 *Static Box* và 1 *Push Button*.



Click đôi vào Edit Text để xuất hiện cửa sổ các thuộc tính của điều khiển như sau:



Một số thuộc tính quan trọng nhất của *Edit Box* bao gồm:

- Tag: đây là tên của điều khiển. Dùng tên này có thể thao tác đến các thuộc tính của đối tượng. Ở đây ta đặt tên là: *txtX*.
- String: là xâu kí tự hiện lên *Edit Box*. Có thể để trống hoặc thiết lập bằng 0.

Tương tự, thay đổi thuộc tính tag của Edit Box thứ 2 thành *txtY*, *Static Box* cũng tương tự thành *txtKetQua*.

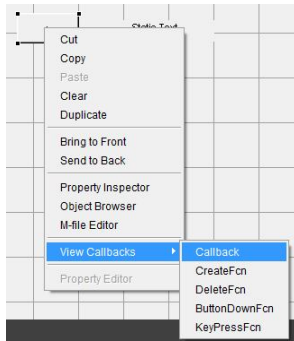
Với **Push Button**: thuộc tính tag đặt là *cmdCalculate*, String đặt là “*ket qua*”

4. Viết lệnh cho chương trình

Chương trình có tác dụng khi nhấn vào nút **Push Button** sẽ hiện lên kết quả ở *Static Box*. Vì thế sẽ phải viết vào hàm nào mà khi nhấn vào Push Button sẽ gọi. Chính là hàm *Callback*. Điều khiển nào cũng có hàm callback, giống như hàm ngắt trong vi điều khiển vậy.

Click chuột phải vào nút “+” chọn *View Callbacks\ Callback*:

Trong phần này còn một số hàm nữa sẽ giới thiệu sau.



```
% --- Executes on button press in cmdCalculate.
function cmdCalculate_Callback(hObject, eventdata, handles)
% hObject    handle to cmdCalculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

Nhìn vào định nghĩa của hàm trong Editor ta sẽ thấy là: hàm này được thực hiện khi nhấn vào nút buttonCalculate.

Hàm có một số tham số:

- *hObject* : handle của điều khiển buttonCalculate
- *eventdata*: dữ liệu khi sự kiện xảy ra.
- *handles*: là một cấu trúc chứa tất cả các điều khiển và dữ liệu người dùng, dùng để truy xuất các điều khiển khác.

Qua thuộc tính tag của các điều khiển ta sẽ truy suất đến thuộc tính string của các điều khiển txtX, txtY, txtKetQua bằng lệnh *get* và *set*.

```
get(handles.tag_dieu_khien, 'ten_thuoc_tinh');
```

```
set(handles.tag_dieu_khien, 'ten_thuoc_tinh', gia_tri);
```

Một hàm quan trọng nữa biến từ chuỗi sang số: *str2num* và *num2str* để biến trở lại. Vậy chúng ta sẽ viết hàm như sau:

```
% --- Executes on button press in buttonCalculate.
function buttonCalculate_Callback(hObject, eventdata, handles)
% hObject    handle to buttonCalculate (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

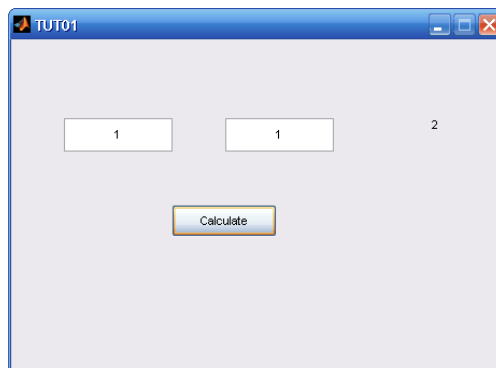
val1 = get(handles.editStr1, 'String');
val2 = get(handles.editStr2, 'String');

val1 = str2num(val1);
val2 = str2num(val2);

val3 = val1 + val2;

set(handles.staticStr3, 'String', num2str(val3));
```


Nhấn nút Run kiểm tra kết quả:



5. Các tính chất của các điều khiển trong GUIDE Matlab

Để hiện cửa sổ các tính chất **Property Inspector** của một điều khiển, có 3 cách sau:

- Nhấn đúp chuột vào mỗi điều khiển.
- Chọn điều khiển rồi vào menu View, chọn **Property Inspector**.
- Chọn điều khiển rồi nhấn vào biểu tượng **Property Inspector**, gần chỗ M-file editor.

Khi đó, cửa sổ **Property Inspector** sẽ hiện ra. Khi nhấn vào các điều khiển khác thì cửa sổ này sẽ hiện thông tin tương ứng cho điều khiển đó.

Một vài tính chất chung của các điều khiển mà các bạn nên chú ý:

Tính chất (property)	Giá trị (value)	Mô tả
Enable	on, inactive, off Mặc định là: on	Xác định khi nào thì điều khiển hiển thị lên giao diện. Đặt = off, thì điều khiển sẽ <u>không</u> xuất hiện.
Max	Mặc định là 1.	Giá trị lớn nhất, tùy thuộc vào từng điều khiển.
Min	Mặc định là 0	Giá trị nhỏ nhất, tùy thuộc vào từng điều khiển.
Position	Vector gồm 4 phần tử (left, bottom, width, height)	Kích thước của điều khiển và vị trí tương đối của nó với điều khiển chứa nó.
String		Chuỗi hiển thị
Units		Đơn vị đo lường dùng trong xác định vị trí.
Value	Vô hướng hoặc vector	Giá trị của component, tùy thuộc vào từng component.

Ngoài ra, thuộc tính cực kỳ quan trọng mà các bạn phải chú ý là **tag**. Thuộc tính này chính là tên của đối tượng, là duy nhất dùng để phân biệt đối tượng này và đối tượng khác.

6. Tổng quan về hàm Callback trong lập trình GUI

Sau khi tạo giao diện xong thì phần lập trình rất quan trọng, chính là lập trình các hành vi của các điều khiển để đáp ứng lại các sự kiện như nhấn phím, kéo thanh trượt, khi chọn menu, ... đó chính là các hàm Callback (giống như các hàm sự kiện trong các ngôn ngữ khác).

a. Thế nào là hàm Callback

Callback là một hàm mà khi viết miêu tả hành vi của một thành phần GUI xác định hoặc là của chính GUI figure, điều khiển các hành vi của chúng bằng cách thực hiện một số hành động được viết trong hàm, để đáp ứng lại một sự kiện của chính thành phần đó. Cách lập trình này thường gọi là: Lập trình lái sự kiện (event driven programming).

Ví dụ, khi bạn nhấn một Button thì vẽ đồ thị, tính tổng, ... Vậy thì khi nhấn phím thì hiển nhiên đã gọi hàm Callback nhấn phím của Button đó, và trong hàm Callback này sẽ thực hiện lệnh vẽ đồ thị, tính tổng, ... tương ứng.

b. Các loại hàm Callback

Mỗi thành phần có nhiều hàm Callback khác nhau, sau đây liệt kê các loại hàm Callback và các điều khiển có thể có hàm này.

Callback property	Sự kiện xảy ra	Thành phần có hàm này
ButtonDownFcn	Thực hiện khi người dùng nhấn chuột lên hoặc trong 5 pixels của component hoặc figure. Nếu là component thì thuộc tính Enable phải on (tắt nhiên rồi).	Axes,figure,button group,panel,user interfacecontrols
Callback	Hành động của các component, ví dụ như thực thi khi người dùng click lên Push Button hoặc chọn một thành phần menu.	Contextmenu, menu,userinterface controls
CloseRequestFcn	Thực thi trước khi figure đóng.	Figure
CreateFcn	Tạo các thành phần.Nó được dùng để khởi tạo các thành phần khi nó được	Axes,figure,button group,contextmenu,

	tạo ra. Nó thực thi sau khi thành phần hoặc figure được tạo, nhưng trước khi hiển thị lên trên giao diện người dùng.	menu,panel,user interfacecontrols
DeleteFcn	Xóa thành phần. Nó có thể được dùng để thực hiện hành động xóa bỏ trước khi component hoặc figure bị hủy bỏ.	Axes,figure,button group,contextmenu, menu,panel,user interfacecontrols
KeyPressFcn	Thực thi khi người dùng nhấn một phím trong keyboard và component hoặc figure của hàm callback đó đang được focus.	Figure,userinterface controls
KeyReleaseFcn	Thực thi khi người dùng nhả một phím đang bấm và figure vẫn đang được focus.	Figure
ResizeFcn	Thực thi khi người dùng thay đổi kích thước của panel, button group, hoặc figure với điều kiện thuộc tính Resize của figure = on.	Buttongroup,figure, panel
SelectiononChangeFcn	Thực thi khi người dùng lựa chọn một nút Radio Button khác hoặc toggle button khác trong thành phần Button Group.	Buttongroup
WindowButtonDownFcn	Thực thi khi bạn nhấn chuột (trái hoặc phải) trong khi con trỏ vẫn nằm trong vùng cửa sổ figure.	Figure
WindowButtonMotionFcn	Thực thi khi bạn di chuyển con trỏ trong vùng cửa sổ figure.	Figure
WindowButtonUpFcn	Ban đầu bạn nhấn chuột (trái, hoặc phải) thì khi nhả phím đó ra thì hàm	Figure

	này sẽ được gọi.	
WindowScrollWheelFcn	Thực thi khi nút cuộn của chuột cuộn trong khi figure vẫn trong tầm focus.	Figure

7. Chương trình Calculator

Có rất nhiều cách để tạo ra một chương trình "Máy tính bấm tay" hay Calculator. Ví dụ sau đây sẽ hướng dẫn tạo ra một chương trình máy tính bấm tay đơn giản sử dụng kỹ thuật tròng "Callback" trong lập trình giao diện GUI.

Sử dụng GUIDE, thiết kế giao diện của chương trình như sau:



Đặt thuộc tính cho các đối tượng như sau:

- Edit Text: Tag=edit1 (mặc định); FontSize=20; Enable=Inactive; HorizontalAligment=right;...
- Đặt thuộc tính Tag của tất cả các nút là: pushadd (ngoại trừ 3 nút: "=" và "%" và "C")
- Nút "=": Tag=pushequal
- Nút "%": Tag=pushpercent
- Nút "C": Tag=pushclear

Điều chỉnh kích thước và sử dụng công cụ "Align Objects" để sắp xếp vị trí các nút như hình trên.

Lưu lại fig-file với tên *mycalc.fig*, trong file *mycalc.m* tìm các hàm tương ứng và thêm vào các lệnh sau:

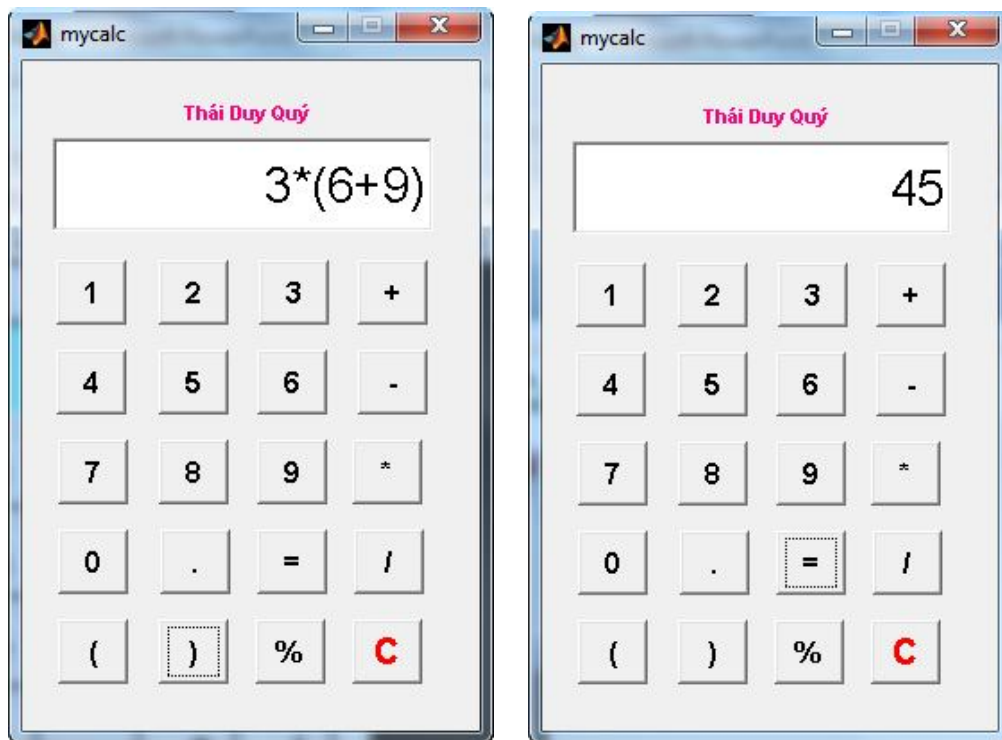
```
% --- Executes on button press in pushadd.
function pushadd_Callback(hObject, eventdata, handles)
textstr = get(handles.edit1, 'String');
addstr = get(hObject, 'String');
newstr = strcat(textstr, addstr);
set(handles.edit1, 'String', newstr)

% --- Executes on button press in pushequal.
function pushequal_Callback(hObject, eventdata, handles)
textstr = get(handles.edit1, 'String');
try
    set(handles.edit1, 'String', eval(textstr))
catch err
    set(handles.edit1, 'String', err.message)
end

% --- Executes on button press in pushpercent.
function pushpercent_Callback(hObject, eventdata, handles)
textstr = get(handles.edit1, 'String');
try
    set(handles.edit1, 'String', 100*eval(textstr))
catch err
    set(handles.edit1, 'String', err.message)
end

% --- Executes on button press in pushclear.
function pushclear_Callback(hObject, eventdata, handles)
set(handles.edit1, 'String', '')
```

Lưu lại file *mycalc.m* và chạy chương trình. Sử dụng chuột click các nút để nhập vào một phép tính, sau đó Click nút "=" để thu được kết quả:



Cuối cùng, Click nút "C" để bắt đầu một phép tính khác.

8. Kết chương

Trong chương này, sinh viên tìm hiểu các vấn đề liên quan đến lập trình giao diện người dùng. Thông qua tìm hiểu hàm Callback, kết hợp với ví dụ về chương trình Calculator sinh viên sẽ có cái nhìn rõ hơn về lập trình giao diện người dùng GUI.

Chương tiếp theo sẽ giới thiệu một số phương pháp lập trình xử lý với Matlab.

CHƯƠNG 5. MỘT SỐ PHƯƠNG PHÁP XỬ LÝ TÍNH TOÁN TRÊN MATLAB

1. Tính định thức của ma trận

Cho một ma trận vuông cấp n . Ta cần tìm định thức của nó. Trước hết nhắc lại một số tính chất quan trọng của định thức:

- Nếu nhân tất cả các phần tử của một hàng (hay cột) với k thì định thức được nhân với k .
- Định thức không đổi nếu ta cộng thêm vào một hàng tổ hợp tuyến tính của các hàng còn lại.
- Nếu đổi chỗ hai hàng cho nhau thì định thức đổi dấu

Trước khi đi đến định nghĩa về định thức ta tìm hiểu khái niệm về hoán vị và phép thế.

Cho một dãy số, nếu ta đổi chỗ các số trong dãy cho nhau thì ta đã thực hiện một phép hoán vị. Ví dụ 123, 132,... là các hoán vị của dãy số $\{1, 2, 3\}$. Trong hoán vị $\alpha_1\alpha_2\ldots\alpha_i\ldots\alpha_j\ldots\alpha_n$ ta nói α_i làm một nghịch thế với α_j nếu $i < j$ mà $\alpha_i > \alpha_j$. Ví dụ trong hoán vị 1432 số 4 làm với số 3 một nghịch thế, số 4 làm với số 2 một nghịch thế, số 3 làm với số 2 một nghịch thế. Một hoán vị gọi là chẵn nếu tổng số nghịch thế trong hoán vị đó là một số chẵn; một hoán vị gọi là lẻ trong trường hợp ngược lại. Như vậy 1432 là một hoán vị lẻ.

Cho một dãy số, nếu ta tạo ra một dãy số mới bằng cách đổi chỗ các phần tử cho nhau thì ta đã thực hiện một phép thế.

Ví dụ $p = \begin{pmatrix} 2 & 1 & 4 & 3 \\ 1 & 4 & 2 & 3 \end{pmatrix}$ là phép thế biến 2 thành 1, 1 thành 4, 4 thành 2 và 3 thành 3.

Một phép thế gọi là chẵn nếu tính chẵn lẻ của dòng trên và dòng dưới như nhau và lẻ trong trường hợp ngược lại. Phép thế trên là phép thế lẻ.

Cho ma trận vuông $[A]$ cấp n . Các phần tử của hàng thứ i là $a_{i,1}, a_{i,2}, \ldots, a_{i,n}$. Các phần tử của cột thứ j là $a_{1,j}, a_{2,j}, \ldots, a_{n,j}$. Ta xem hàng thứ i là một vector, kí hiệu là A_{i*} và cột thứ j cũng là một vec tơ, kí hiệu là A_{*j} . Với mỗi phép thế:

$$p = \begin{pmatrix} i_1 & i_2 & \dots & i_n \\ j_1 & j_2 & \dots & j_n \end{pmatrix} \quad (1)$$

ta lập tích:

$$a_{i_1 j_1} a_{i_2 j_2} \dots a_{i_n j_n} \quad (2)$$

Trước mỗi tích (2) ta đặt dấu + nếu và dấu - nếu phép thế (1) lẻ. Sau đó ta lập tổng của $n!$ tích có dấu như vậy, nghĩa là tổng:

$$\sum_p (-1)^{t(p)} a_{i_1 j_1} a_{i_2 j_2} \dots a_{i_n j_n} \quad (3)$$

trong đó:

$t(p) = 1$ nếu phép thế p lẻ

$t(p) = 0$ nếu phép thế p chẵn

Tổng (4) được gọi là định thức của ma trận vuông $[A]$, cấp n .

Ta xây dựng hàm *determinant()* để tính định thức của ma trận theo định nghĩa:

```
function d = determinant(A)
% DETERMINANT tính định thức theo định nghĩa.
[m, n] = size(A);
if ( m ~= n )
    fprintf( '\n' );
    fprintf( 'Chỉ ma trận vuông mới có định thức!\n' );
    return
end
p = zeros(1, n);
nf = prod([1:n]);
d = 0.0;
for i = 1:nf
    p = nextperm(p);
    s = permsign(p);
    x = diag(A([1:n],p));
    d = d + s*prod(x);
end
```



```
end
function psign = permsign(p)
% PERMSIGN tra ve dau phep the .
% +1, neu phep the chan,
% -1, neu phep the le.
n = length ( p );
psign = 1;
for i = 1:n-1
    j = i;
    while (p(j) ~= i)
        j = j + 1;
    end
    if ( j ~= i )
        temp = p(i);
        p(i) = p(j);
        p(j) = temp;
        psign = - psign;
    end
end
function q = nextperm(p)
n = length(p);
q = p;
if(n == 1)
    q = 1;
elseif (q == 0)
    q = [1:n];
else
    i = n - 1;
```

```
while (q(i) > q(i+1))
```

```
    i = i - 1;
```

```
    if (i == 0)
```

```
        break; %2
```

```
    end
```

```
end
```

```
if (i == 0)
```

```
    q = [1:n];
```

```
else
```

```
    j = n;
```

```
    while (q(j) < q(i))
```

```
        j = j - 1;
```

```
    end
```

```
    t = q(j);
```

```
    q(j) = q(i);
```

```
    q(i) = t;
```

```
    q(i+1:n) = q(n:-1:i+1);
```

```
end
```

```
end
```

Để tính định thức ta dùng chương trình *ctdeterminant.m*:

```
clear all, clc
```

```
%a = [1 2; 3 5];
```

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

```
d = determinant(a)
```

2. Nghịch đảo ma trận bằng cách dùng Minor

Cho ma trận $[A]$, ta có:

$$(a^{-1})_{i,j} = \frac{A_{j,i}}{\det[A]}$$

Trong đó: $(a^{-1})_{i,j}$ là phần tử ở hàng i , cột j của ma trận $[A]^{-1}$, $A_{i,j}$ là phần bù đại số của phần tử $a_{i,j}$ của ma trận $[A]$.

Ta xây dựng hàm `minorinv()` để thực hiện thuật toán trên:

```
function c = minorinv(a)
```

```
    % Tìm ma tran Nghich dao bang thuat toan minor
```

```
    n = size(a, 1);
```

```
    ms = det(a);
```

```
    for i = 1:n
```

```
        for k = 1:n
```

```
            b = cofactor(a, i, k);
```

```
            c(i, k) = b/ms;
```

```
        end
```

```
    end
```

```
    c = transpose(c);
```

Để tìm ma trận nghịch đảo ta dùng chương trình `ctminorinv.m`:

```
clear all, clc;
```

```
a = [1 3 5; 3 4 9; 5 9 6];
```

```
b = minorinv(a)
```

3. Nghịch đảo ma trận bằng thuật toán gauss-Jordan.

Cho ma trận $[A]$ và ma trận đơn vị $[E]$ tương ứng. Dạng của ma trận $[E]$ cấp 4, là:

$$E = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Như vậy, vấn đề là ta cần tìm ma trận $[A]^{-1}$. Phương pháp loại trừ để nhận được ma trận nghịch đảo $[A]^{-1}$ được thực hiện qua n giai đoạn, mỗi một giai đoạn gồm hai bước. Đối với giai đoạn thứ k :

- Chuẩn hoá phần tử a_{kk} bằng cách nhân hàng với nghịch đảo của nó.
- Làm cho bằng không các phần tử phía trên và phía dưới đường chéo cho đến cột thứ k .

Khi $k = n$ thì $[A]^{(k)}$ sẽ trở thành ma trận đơn vị và $[E]$ trở thành $[A]^{-1}$

Ta xây dựng một hàm nghịch đảo $\text{invmat}()$:

```
function x = invmat(a)
```

```
    % Nghich dao ma tran a 102
```

```
    %Cu phap: x = invmat(a)
```

```
    k = size(a, 1);
```

```
    n = k;
```

```
    b = eye(n);
```

```
    a = [a, b];
```

```
    i = 1;
```

```
    while i <= n
```

```
        if a(i, i) ~= 0
```

```
            c = a(i, i);
```

```
            a(i, i:2*n) = a(i, i:2*n)/c;
```

```
        end
```

```
    for k = 1:n
```

```
        if k ~= i
```

```

c = a(k, i);
a(k, i:2*n) = a(k, i:2*n) - a(i, i:2*n)*c;
end
end
i = i+1;
end
x(:, 1:k) = a(:, (1+k):(2*k));

```

Để nghịch đảo ma trận:

$$[A] = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

ta dùng chương trình *ctinvmat.m*:

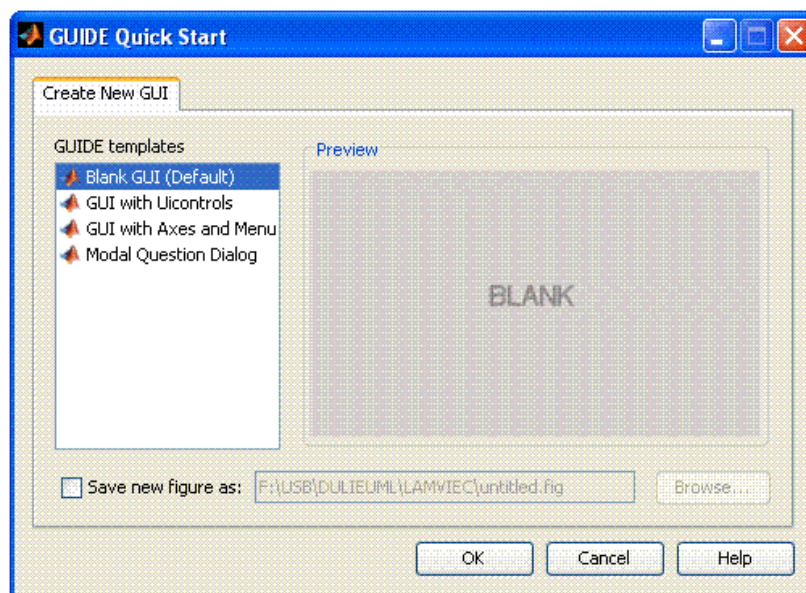
```

clear all, clc
a = [ 2 1 1; 1 2 1; 1 1 2];
b = invmat(a)

```

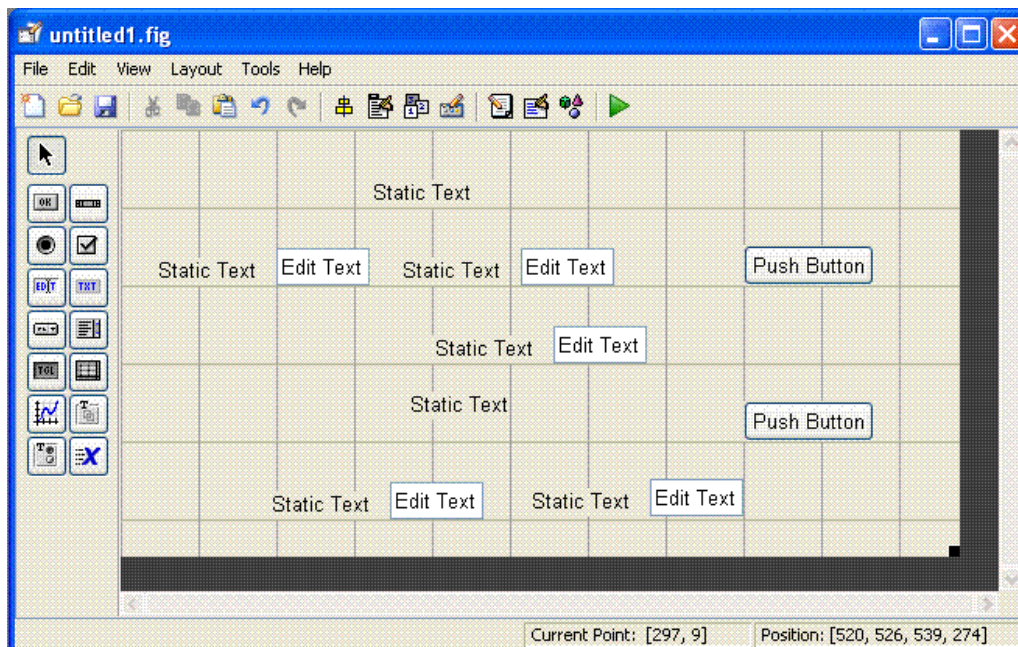
4. Lập trình giao diện: Giải phương trình bậc 2

Chạy Matlab, vào Menu File\New\GUI sẽ thấy hiện ra hộp thoại sau:



Chọn **Blank GUI**, nhấn OK.

Trong hộp thoại hiện ra, kéo thả các điều khiển **Push button**, **Edit Text** và **Static Text** vào figure như giao diện sau:



Mỗi điều khiển đều có nhiều thuộc tính, có 2 thuộc tính quan trọng nhất là: **Tag**, **String**. **Tag** là thuộc tính chỉ địa chỉ của điều khiển (dùng để gọi khi cần). **String** là thuộc tính chứa nội dung (sẽ được hiển thị ra ngoài) của điều khiển. Sau đây ta đặt các thuộc tính này:

- Push button 1:

- + Tag : start
- + String : Bat dau

- Push button 2:

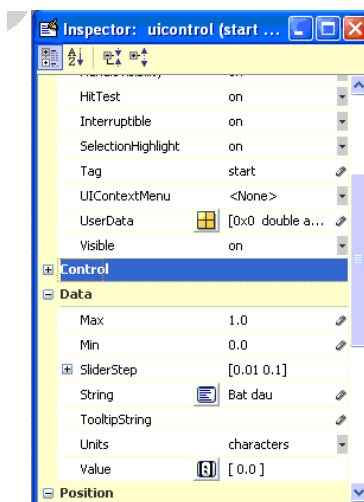
- + Tag : close
- + String : Close

- Edit Text 1:

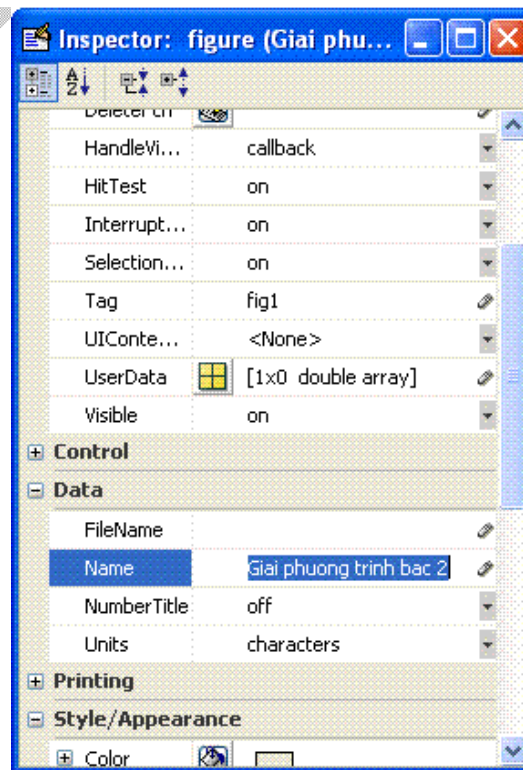
- + Tag : heso_a
- + String : (xóa trắng)

- Edit Text 2:

- + Tag : heso_b

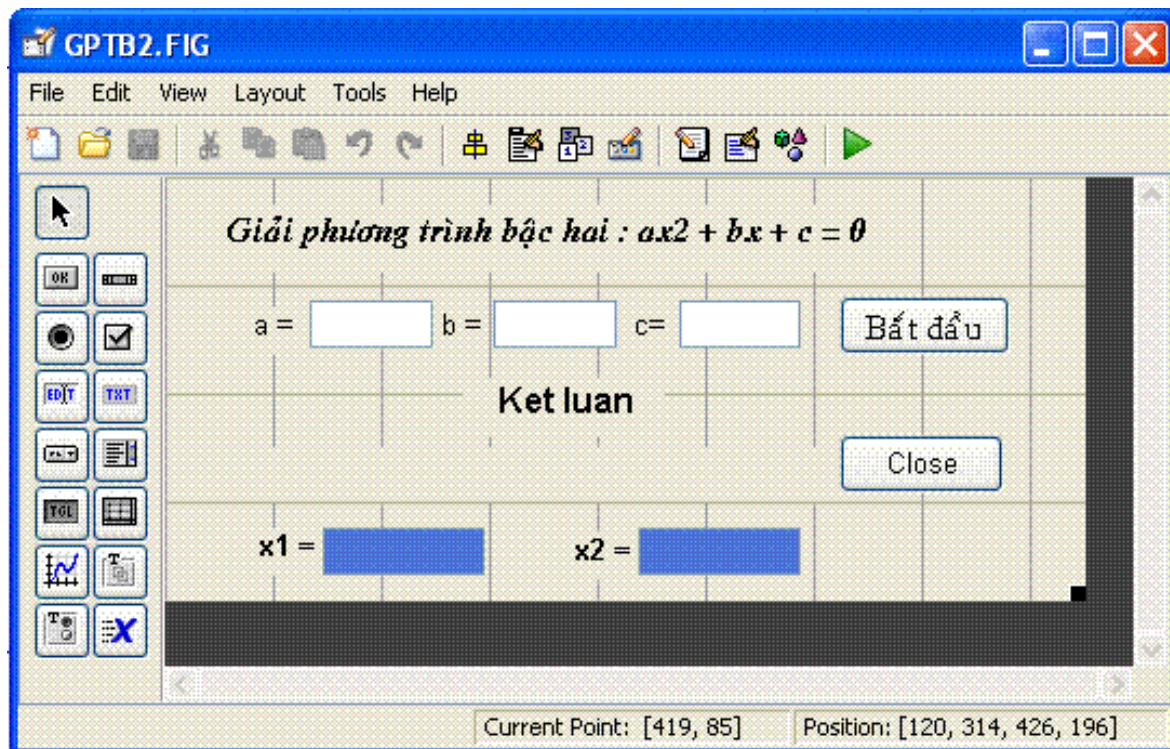


- + String : (xóa trắng)
- Edit Text 3:
 - + Tag : heso_c
 - + String : (xóa trắng)
- Edit Text 4:
 - + Tag : nghiem_x1
 - + String : (xóa trắng)
- Edit Text 5:
 - + Tag : nghiem_x2
 - + String : (xóa trắng)
- Static Text 1: (Đối với các Static Text thuộc tính **Tag** không quan trọng, trừ các trường hợp đặc biệt)
 - + String : Giai phương trình bậc 2: $ax^2 + bx + c = 0$
- Static Text 2:
 - + String : a =
- Static Text 3:
 - + String : b =
- Static Text 4:
 - + String : c =
- Static Text 5:
 - + String : Ket luan
- Static Text 6:
 - + Tag : kl
 - + String : (xóa trắng)
- Static Text 7:
 - + String : x1 =
- Static Text 8:



- + String : x2 =
- figure: (click đúp vào nền của figure):
- + Tag : fig1
- + Name : Giai phương trình bac 2

Sau khi đặt các thuộc tính, căn chỉnh (sử dụng chuột hoặc công cụ *Align Objects*) được *figure* có dạng như sau:



Lưu lại figure (Menu File\Save) dưới tên GPTB2.fig. Sau đó Matlab tự sinh file GPTB2.m; trong file này tìm hàm **start_callback** và đánh vào các dòng lệnh sau:


```

% -----
function varargout = start_Callback(h, eventdata, handles, varargin)
a=str2double(get(handles.heso_a,'string'));
b=str2double(get(handles.heso_b,'string'));
c=str2double(get(handles.heso_c,'string'));
delta=b^2-4*a*c;
x1=(sqrt(delta)-b)/(2*a);
x2=- (sqrt(delta)+b)/(2*a);
if delta>0
    set(handles.k1,'string','Phương trình có 2 nghiệm');
elseif delta==0
    set(handles.k1,'string','Phương trình có nghiệm kép');
else
    set(handles.k1,'string','Phương trình vô nghiệm');
    x1=[];
    x2=[];
end
set(handles.nghiem_x1,'string',x1);
set(handles.nghiem_x2,'string',x2);

```

Hàm **set** là hàm đặt thuộc tính cho điều khiển.

Hàm **get** là hàm lấy giá trị thuộc tính của điều khiển

Hàm **str2double** là hàm biến chuỗi thành số

Cuối cùng ghi lại file m (GPTB2.m) và chạy chương trình (nhấn F5 hoặc nút run hình tam giác màu xanh). Nhập vào các hệ số a, b, c và click nút **Bat dau** để xem kết quả.

5. Kết chương

Trong chương này, sinh viên đã làm quen với một số phương pháp lập trình phức tạp trên Matlab. Thông qua các ví dụ, sinh viên sẽ có điều kiện hiểu rõ hơn các phương pháp xử lý lập trình trên Matlab hiện nay.

PHỤ LỤC

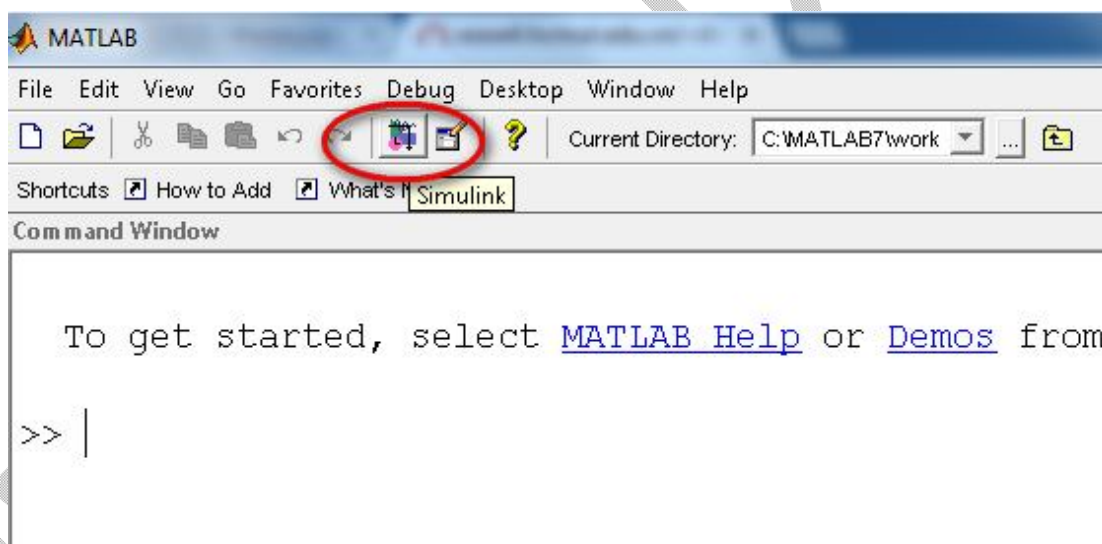
Hướng dẫn sử dụng công cụ Simulink

Simulink là một công cụ trong Matlab dùng để mô hình, mô phỏng và phân tích các hệ thống động với môi trường giao diện sử dụng bằng đồ họa. Việc xây dựng mô hình được đơn giản hóa bằng các hoạt động nhấp chuột và kéo thả. Simulink bao gồm một bộ thư viện khối với các hộp công cụ toàn diện cho cả việc phân tích tuyến tính và phi tuyến.

Simulink là một phần quan trọng của Matlab và có thể dễ dàng chuyển đổi qua lại trong quá trình phân tích, và vì vậy người dùng có thể tận dụng được ưu thế của cả hai môi trường.

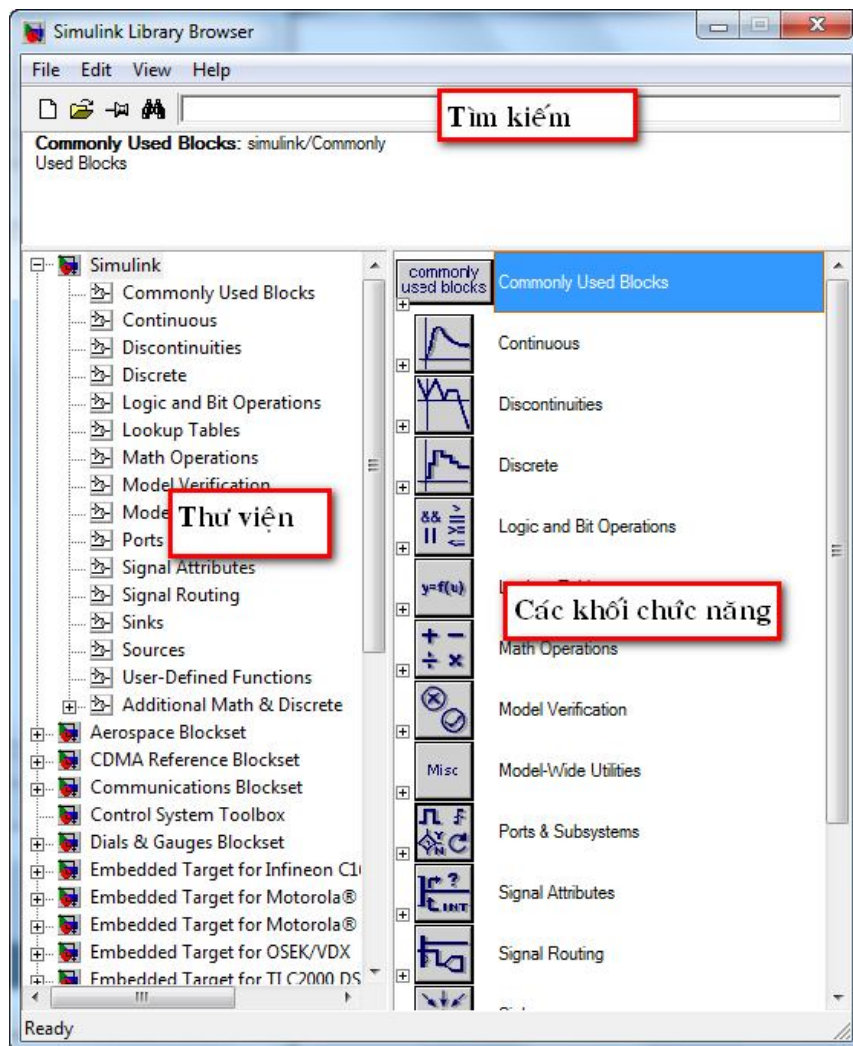
Có thể mở Simulink bằng 2 cách:

- Click vào biểu tượng như hình dưới (Simulink icon)



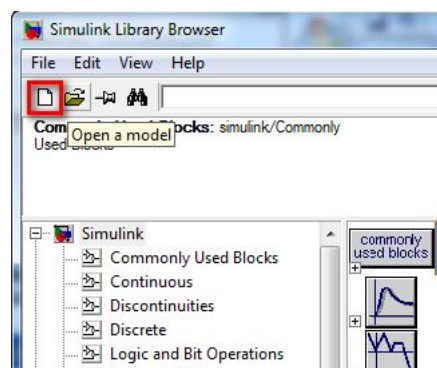
- Từ cửa sổ lệnh, đánh lệnh `simulink` và enter

Cửa sổ thư viện Simulink sẽ hiển thị:



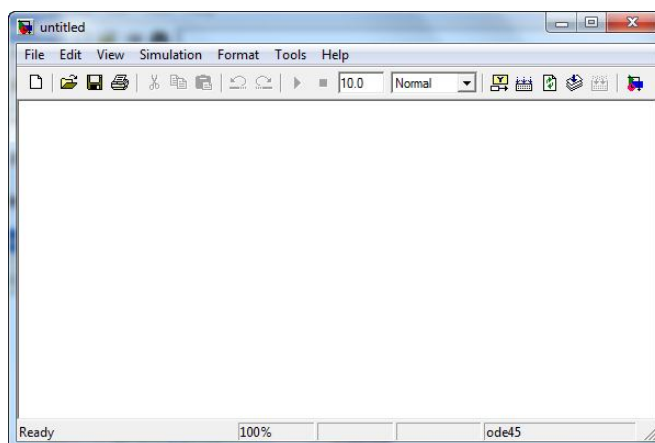
Tạo một mô hình mới bằng cách:

- Click vào icon **New model** hoặc gõ **Ctrl-N**

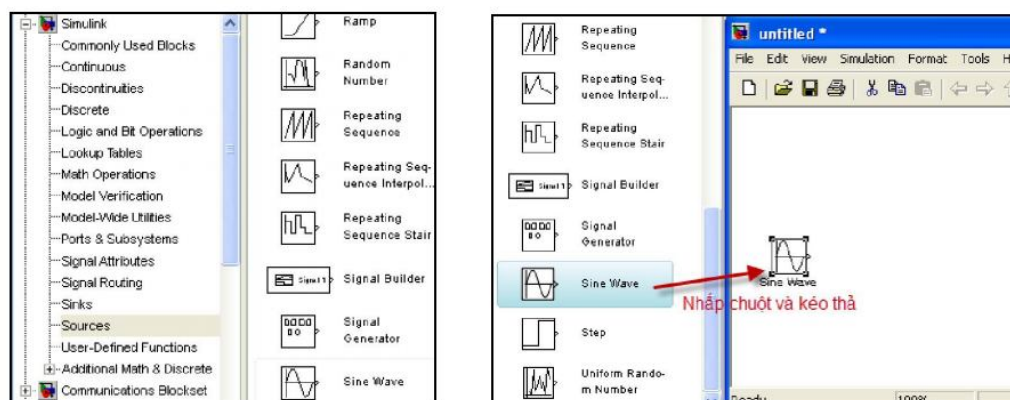


- **Menu File\New\Model**

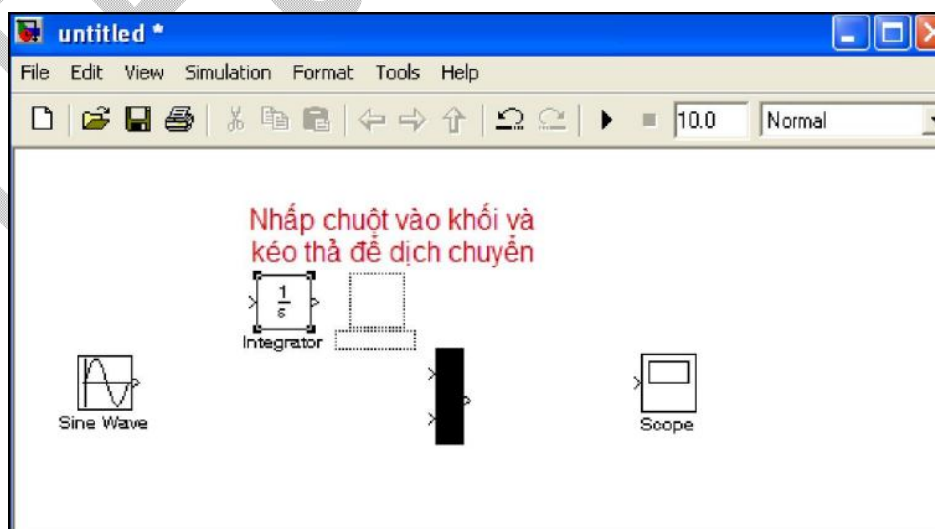
Cửa sổ xây dựng mô hình xuất hiện:



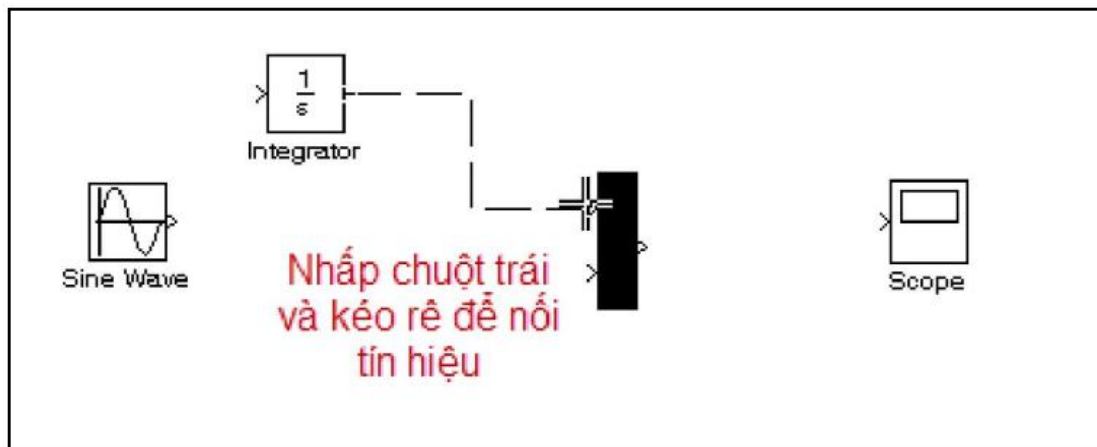
Tạo các khối: từ thư viện Simulink chọn khối cần dùng, nhấp chuột vào và kéo ra ra cửa sổ mô hình:



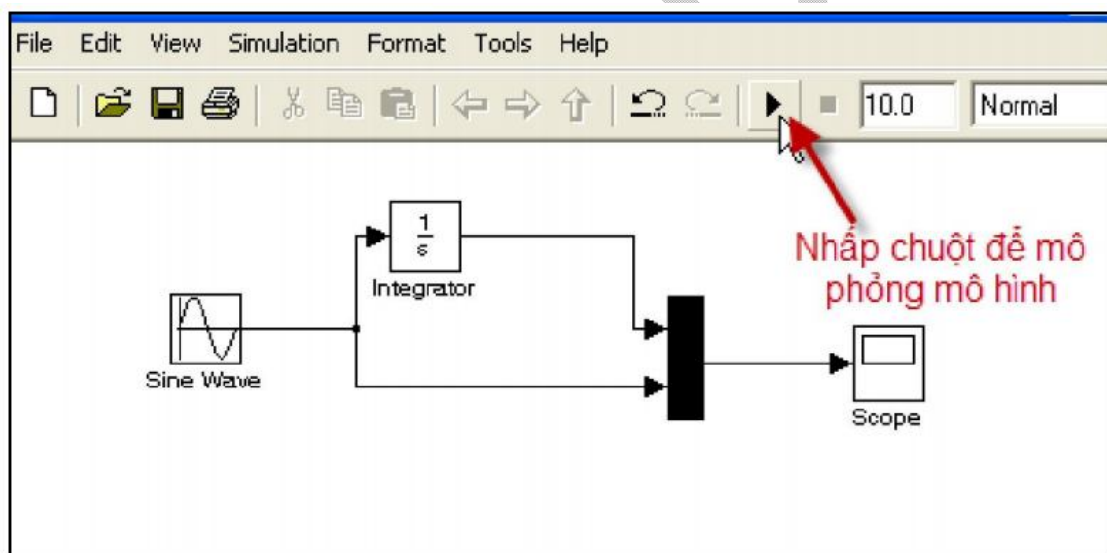
Lưu trữ mô hình bằng lệnh **Save (File\Save)** hoặc nhấp vào icon **Save**. Dịch chuyển các khối đơn giản bằng cách nhấp vào khối đó và kéo thả:



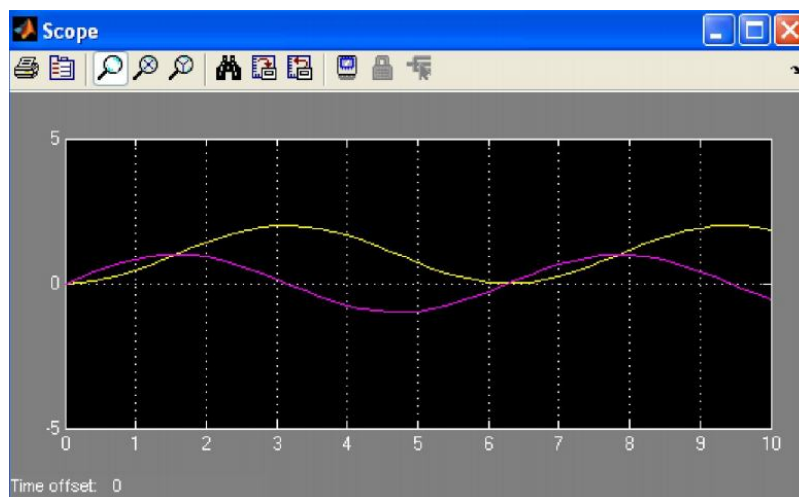
Nối tín hiệu: Đưa con chuột tới ngõ ra của khối (dấu “>”), khi đó con chuột sẽ có dạng “+”. Kéo rê chuột tới ngõ vào của một khối khác và thả ra để kết nối tín hiệu.



Mô phỏng mô hình: Dùng lệnh Start (**Menu Simulation\Start**) hoặc nhấp chuột vào icon Start

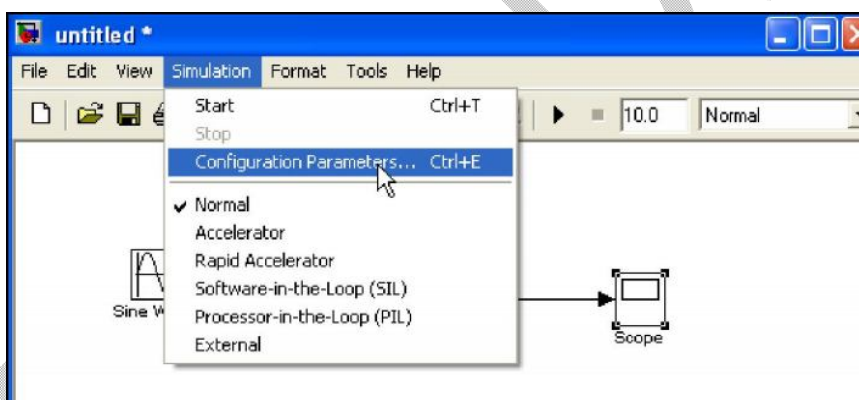


Xem tín hiệu từ Scope: nhấp đôi vào khối Scope:

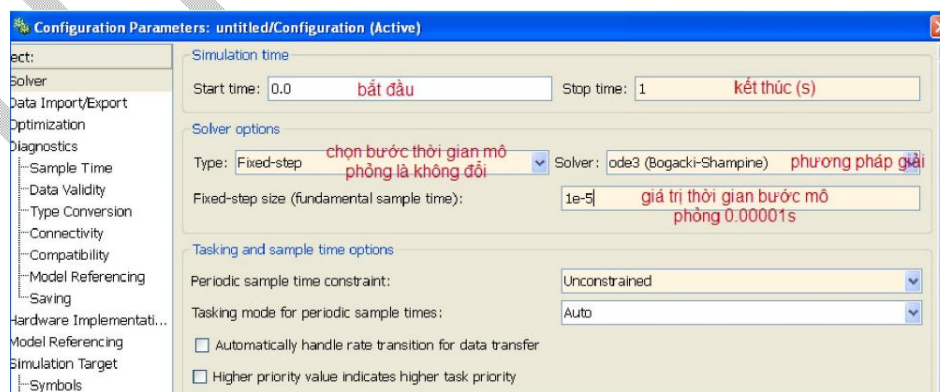


Chỉnh thông số của một khối bằng cách nhấp đôi vào khối cần chỉnh

Trước khi mô phỏng mô hình Simulink, chúng ta cần đặt các thông số mô phỏng bằng cách chọn menu Simulation ☐ Configuration Parameters



Ở cửa sổ Configuration Parameters, chúng ta có thể đặt một số thông số như Start time, Stop time (second – giây), và phương pháp giải Solver, Solver options,.. sau đó nhấn nút OK



TÀI LIỆU THAM KHẢO

- [1].Phan Thanh Tao, *Giáo trình Matlab*, Đại học Đà Nẵng, 2004
- [2].Trần Văn Chính, *Matlab toàn tập*, Đại học Bách Khoa Hà Nội, 2005.
- [3].Ebook, *The Student Edition of Matlab*, Mathworks, Inc;
- [4].Brian R. Hunt Ronald L. Lipsman JonathanM. Rosenberg, *A Guide to MATLAB for Beginners and Experienced Users*, Cambridge University Press, 2001.
- [5]. <http://www.mathworks.com>.

Và một số tài liệu tham khảo khác trên Internet.