# Chương 3

# VẼ ĐỒ THỊ TRÊN MAPLE

Maple cung cấp cho ta nhiều cách để biểu diễn dữ liệu cũng như các biểu thức toán học, một trong số đó là sử dụng đồ thị. Với đồ thị, ta có thể hình dung bài toán và các khái niệm một cách rõ ràng hơn.

- Maple có thể vẽ các loại đồ thị của hầu hết các hàm cơ bản cũng như các hàm ẩn, hàm có tham số. Những đồ thị này có thể hiển thị theo dạng 2-D, 3-D và có thể chuyển động.
- Maple chấp nhận nhiều loại hệ trục tọa độ khi vẽ đồ thị.
- Các vùng hiển thị đồ thị trong Maple rất sống động, vì vậy, ta có thể làm nhiều thao tác trên đó.
- Ngoài ra, Maple cung cấp nhiều tùy chọn trên đồ thị như: tùy chọn các loại trục tọa độ, tiêu đề, màu sắc, hình dáng... Điều này giúp ta hoàn toàn chủ động khi làm việc với các đồ thị.

Thông tin về các loại đồ thị có sẵn trong Maple được trình bày trong mục ***Plotting Guide*** của phần trợ giúp Help.

## 3.1. Các cách vẽ đồ thị

Maple cung cấp nhiều phương pháp khác nhau giúp ta dễ dạng tạo ra các đồ thị như:

- Dùng giao diện vẽ đồ thị tương tác (Interactive Plot Builder).
- Dùng menu ngữ cảnh (Context Menu)
- Kéo đồ thị từ vùng này đến một vùng đồ thị khác (dragging)
- Dùng lệnh

Mỗi phương pháp có một thuận lợi riêng và còn phụ thuộc vào loại đồ thị ta muốn vẽ cũng như các thiết lập riêng cho mỗi đồ thị.

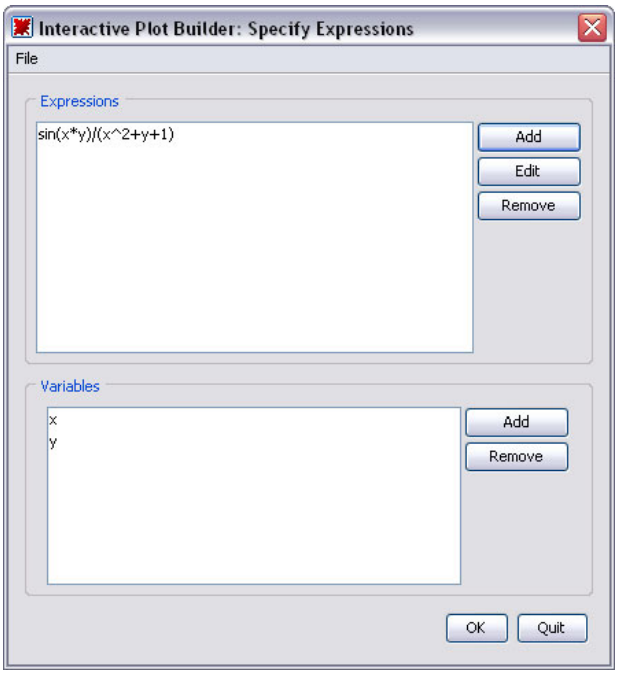
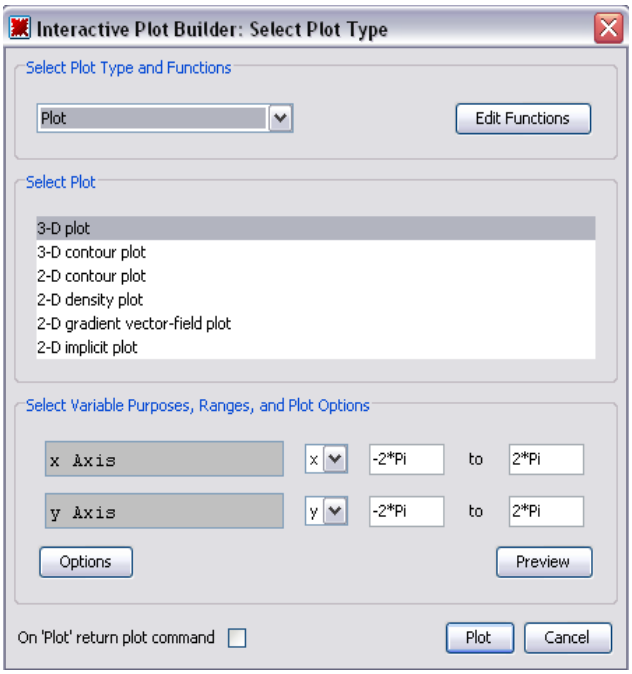
### 3.1.1. Giao diện vẽ đồ thị tương tác

Đây là một chương trình được tạo dựng sẵn trong thư viện của Maple giúp ta vẽ được nhiều loại đồ thị như: các đồ thị thông thường, đồ thị tương tác (hay đồ thị theo tham số), vận động của đồ thị và vận động của đồ thị tương tác. Với mỗi loại đồ thị ta có thể tạo ra:

- Đồ thị trong không gian 2 chiều (2-D), 3 chiều (3-D);
- Đồ thị 2-D trong tọa độ cực;

- Đồ thị phức 2-D, 3-D;
- Đồ thị mật độ 2-D;
- Đồ thị các trường vectơ gradient 2-D;
- Đồ thị hàm ẩn 2-D.

Để mở giao diện vẽ đồ thị, từ menu **Tools** ta chọn **Assistants** rồi chọn **Plot Builder.**

| Cửa sổ "***Specify Expressions***" | Cửa sổ "***Select Plot Type***" |
|---|---|
|  |  |





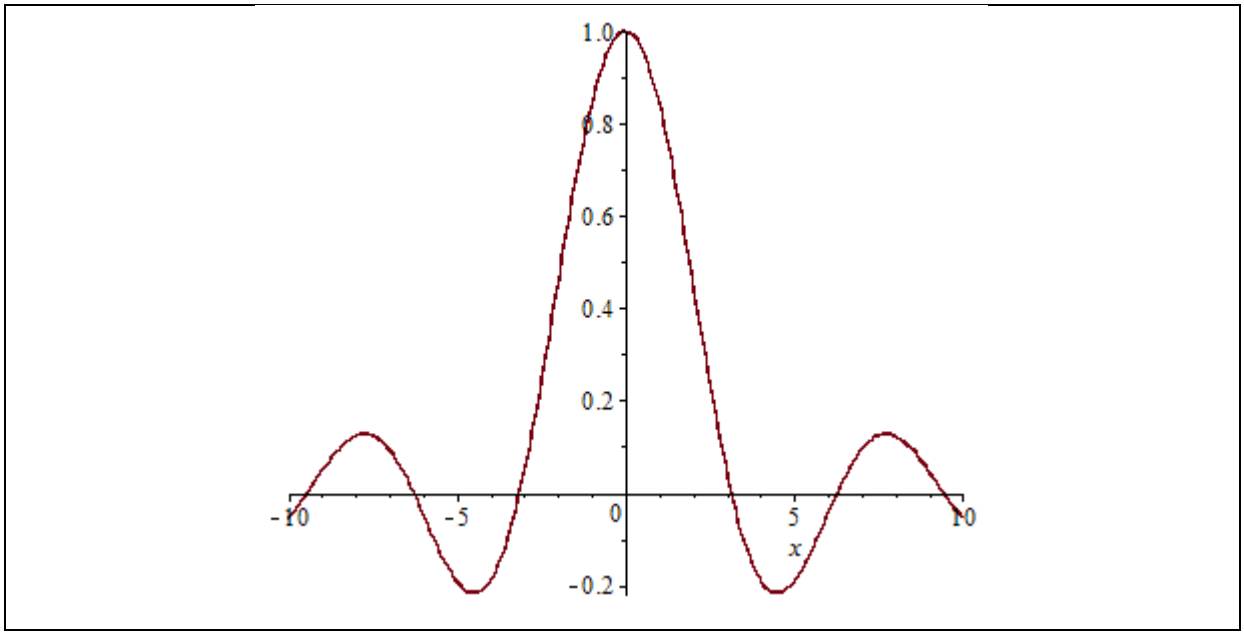
- Nhập biểu thức cần vẽ đồ thị và chọn biến để vẽ tại cửa sổ **Specify Expressions**. Nhấn **OK** để sang cửa sổ **Select Plot Type.**
- Cửa sổ **Select Plot Type:** chọn kiểu đồ thị và các dạng tương ứng của đồ thị, nhập vào các miền trên hệ trục tọa độ mà đồ thị hiển thị. Nhấn Plot để hiển thị đồ thị hoặc chọn **Options** để thiết lập các tùy chọn.

### a. Vẽ đồ thị hàm một biến

***Ví dụ 1:*** Vẽ đồ thị hàm số $y = \frac{\sin(x)}{x}$ trên khoảng $(-2\pi, 2\pi)$.

- Mở giao diện vẽ đồ thị**: Tools → Assistants → Plot Builder**. Lưu ý rằng khi Plot Builder mở ra thì Maple cũng chèn vào trong giao diện chính lệnh **plots[interactive]();** tại vị trí con trỏ.
- Trong cửa sổ **Specify Expressions**: nhấn **Add** để nhập biểu thức **sin(x)/x** rồi nhấn **OK** để mở cửa sổ **Select Plot Type.**
- Vẽ đồ thị: chọn loại đồ thị **2-D plot** và miền vẽ đồ thị theo biến ***x***, miền vẽ đồ thị là khoảng $(-2\pi, 2\pi)$ rồi nhấn **Plot** để vẽ.

```
> plots[interactive]()
```



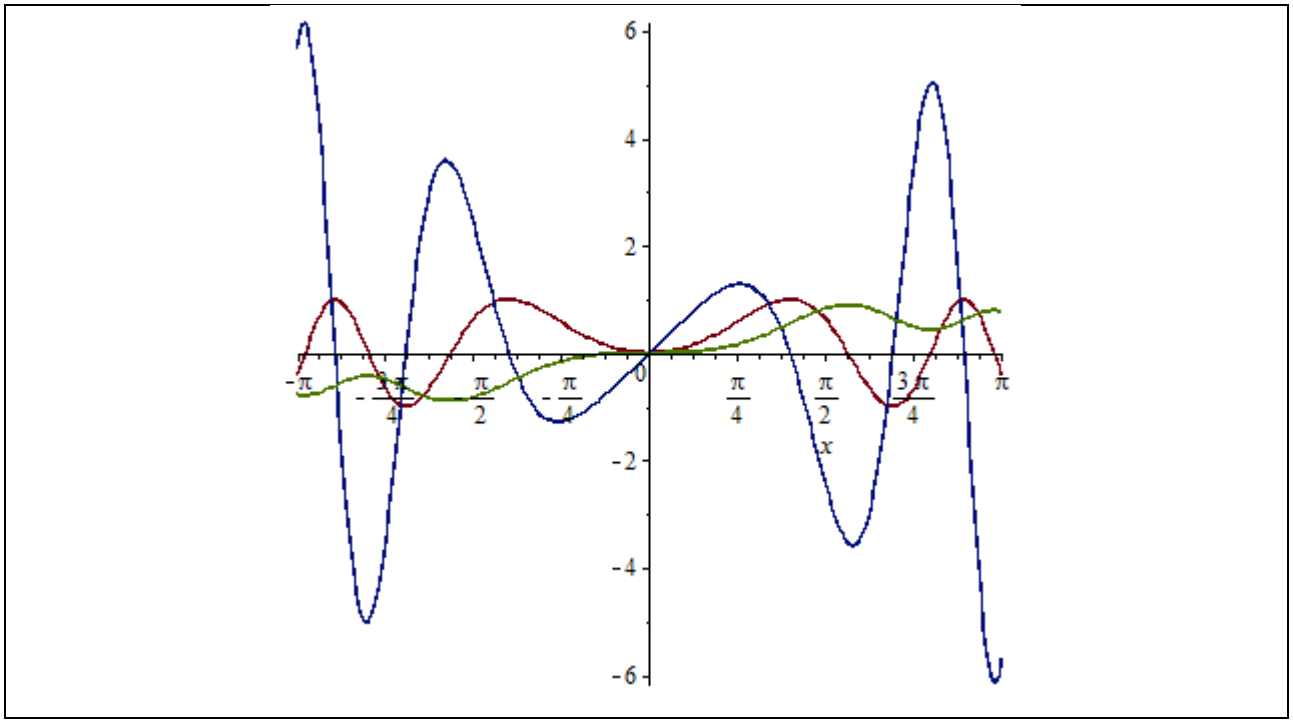

**b. Vẽ nhiều đồ thị trên cùng một hệ trục tọa độ**

Maple có thể vẽ đồ thị của nhiều biểu thức trên cùng một hệ trục giúp cho việc so sánh và đối chiếu các đồ thị dễ dàng hơn. Giao diện vẽ đồ thị có thể làm được điều này.

***Ví dụ 2***: Vẽ đồ thị các hàm số $y = f(x) = \sin(x^2), y = f'(x)$ và $y = \int f(x)dx$ trên cùng một hệ trục tọa độ.

- Mở giao diện vẽ đồ thị **Plot Builder**. Giao diện này chấp nhận việc nhập biểu thức vào dưới dạng 1-D Math và có thể thực hiện một số phép tính cơ bản. Chẳng hạn, nếu nhập vào $diff(sin(x\text{^}2), x)$ thì trong cửa sổ **Expression** sẽ hiển thị biểu thức $2 * cos(x\text{^}2) * x$.
- Trong cửa sổ **Specify Expressions**: thực hiện lần lượt việc nhập các biểu thức $sin(x\text{^}2), diff(sin(x\text{^}2), x)$, và $int(sin(x\text{^}2), x)$.
- Trong cửa sổ **Select Plot Type**:
  - thay đổi miền vẽ đồ thị ở **x Axis** là khoảng $-Pi\,..\,Pi$
  - nhấn **Options** để mở cửa sổ **Plot Options** nếu muốn thêm một số tùy chọn.
- Nhấn **Command** để trả về cú pháp lệnh vẽ các đồ thị trên.
- Để hiển thị đồ thị ta kích chuột phải vào lệnh mới tạo ra ở giao diện chính rồi chọn **Evaluate.**

$$plots_{display}\left( plot\left(\sin\left(x^2\right), x = -\pi..\pi\right), plot\left(2\cos\left(x^2\right)x, x = -\pi..\pi, color = \text{"Niagara Navy"}\right), plot\left(\frac{1}{2}\sqrt{2}\sqrt{\pi}\,\text{FresnelS}\left(\frac{\sqrt{2}\,x}{\sqrt{\pi}}\right), x = -\pi..\pi, color = \text{"Niagara LeafGreen"}\right)\right)$$



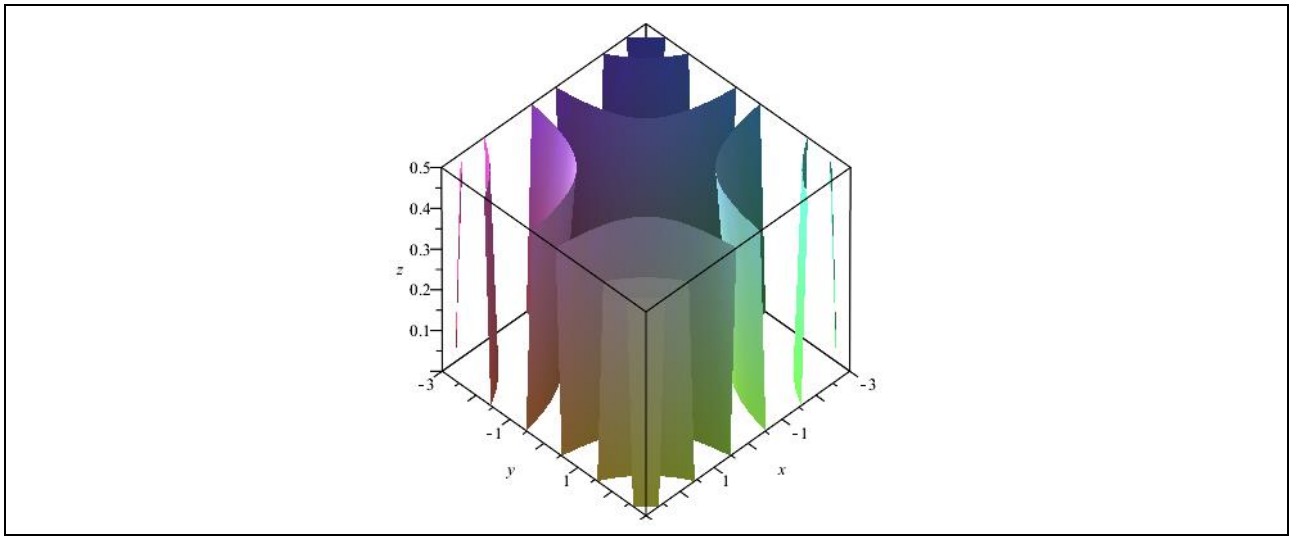

Theo mặc định, Maple hiển thị mỗi đồ thị một màu khác nhau. Ta cũng có thể chọn các dạng đường khi hiển thị đồ thị như: solid, dashed, hay dotted. Tham khảo trong plot/options ở trang trợ giúp help để tìm hiểu thêm về các tùy chọn.

### c. Vẽ đồ thị hàm nhiều biến

Maple có thể vẽ các đồ thị trong không gian 3 chiều và cho ta một số tùy chọn khi vẽ như: các kiểu ánh sáng, kiểu bề mặt của đồ thị, hoặc bóng của đồ thị...

***Ví dụ 3***: Vẽ đồ thị hàm số $z = f(x, y) = \cos(xy)$ trên miền $[-3, 3] \times [-3, 3]$.

- Mở giao diện vẽ đồ thị **Plot Builder.**
- Nhấn **Add** và nhập biểu thức $\boldsymbol{\cos(x * y)}$ rồi nhấn **OK**.
- Trong cửa sổ **Select Plot Type:**
  - Thay đổi miền vẽ đồ thị: **x Axis**: -3…3, **y Axis**: -3…3.
  - Nhấn **Options.**
- Trong cửa sổ **Plot Options:**
  - Ở cột **Variables**, thay đổi trong ô **Range from: 0** … **0.5**.
  - Ở cột **Label**, nhập **z**.
  - Ở ô **Style**, chọn **surface**.
  - Trong ô **Miscellaneous**, phần **Grid Size**, chọn **40, 40.**
  - Có thể nhấn **Preveiw** để xem trước hình ảnh đồ thị và chỉnh sửa các tùy chọn hoặc nhấn **Plot** để thực hiện vẽ đồ thị.

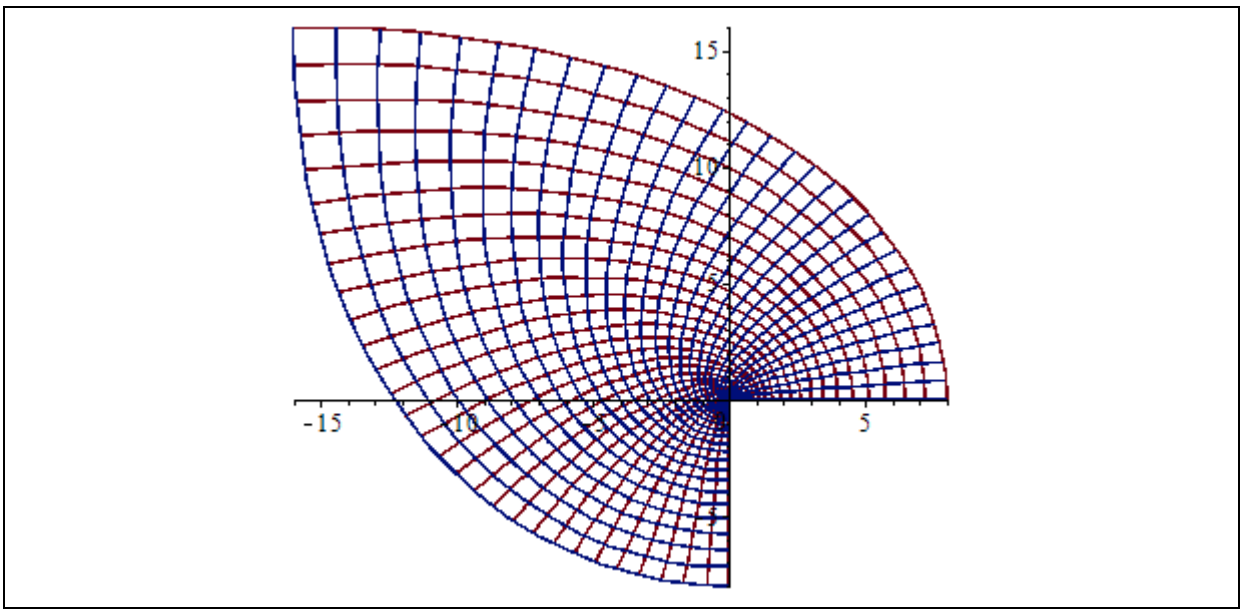



**d. Vẽ đồ thị hàm biến phức**

Maple có thể vẽ đồ thị bảo giác (conformal plot) của hàm số phức trong không gian 2 chiều hoặc trên mặt cầu Riemann 3 chiều.

***Ví dụ 4***: Vẽ đồ thị hàm số phức $y = z^3$ trên khoảng $(0, 2 + 2I)$.

- Mở giao diện vẽ đồ thị, nhấn **Add** nhập vào biểu thức **z^3**
- Trong cửa sổ **Select Plot Type:**
  - Chọn **2-D conformal plot of a complex-valued function**.
  - Thay đổi miền xác định của biến **z: 0** … **2+2*I**.
  - Nhấn **Plots Options.**
- Trong cửa sổ **Plot Options:**
  - Trong phần **Axes** chọn **normal.**
  - Trong phần **Miscellaneous**, ở ô **Grid Size** nhập **30, 30**.
  - Nhấn **Plot** để vẽ đồ thị.

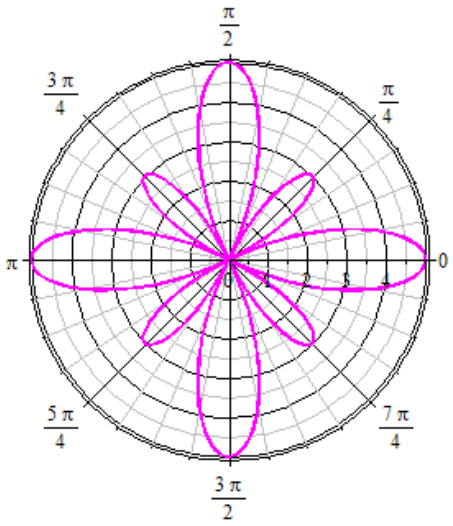


### e. Vẽ đồ thị trong tọa độ cực

Theo mặc định, việc vẽ đồ thị trong Maple luôn thực hiện trên tọa độ Cartesian. Tuy nhiên, nó cũng có các tùy chọn cho phép vẽ đồ thị trong các hệ tọa độ khác như: hyperbolic, parabolic, hệ tọa độ cực (polar),…trong không gian 2 chiều, hoặc bipolar cylindrical, bispherical, cylindrical,… trong không gian 3 chiều. Để biết thêm thông tin về các hệ tọa độ, tham khảo phần **coords** trong trang trợ giúp Help.

***Ví dụ 5***: Vẽ đồ thị $y = 1 + 4\cos(4\theta), \theta \in (0, 8\pi)$.

- Mở **Plot Builder**, nhấn **Add** nhập biểu thức: **1+4*cos(4*theta)** rồi nhấn **OK**.
- Trong cửa sổ **Select Plot Type:**
  - Chọn kiểu đồ thị **2-D polar plot.**
  - Thay đổi khoảng trong **Angle** là **0 .. 8*Pi**.
  - Nhấn **Options.**
- Trong cửa sổ **Plot Options:**
  - Ở phần **Color** chọn **Magenta**.
  - Nhấn **Plot** để vẽ đồ thị




### f. Vẽ đồ thị tương tác

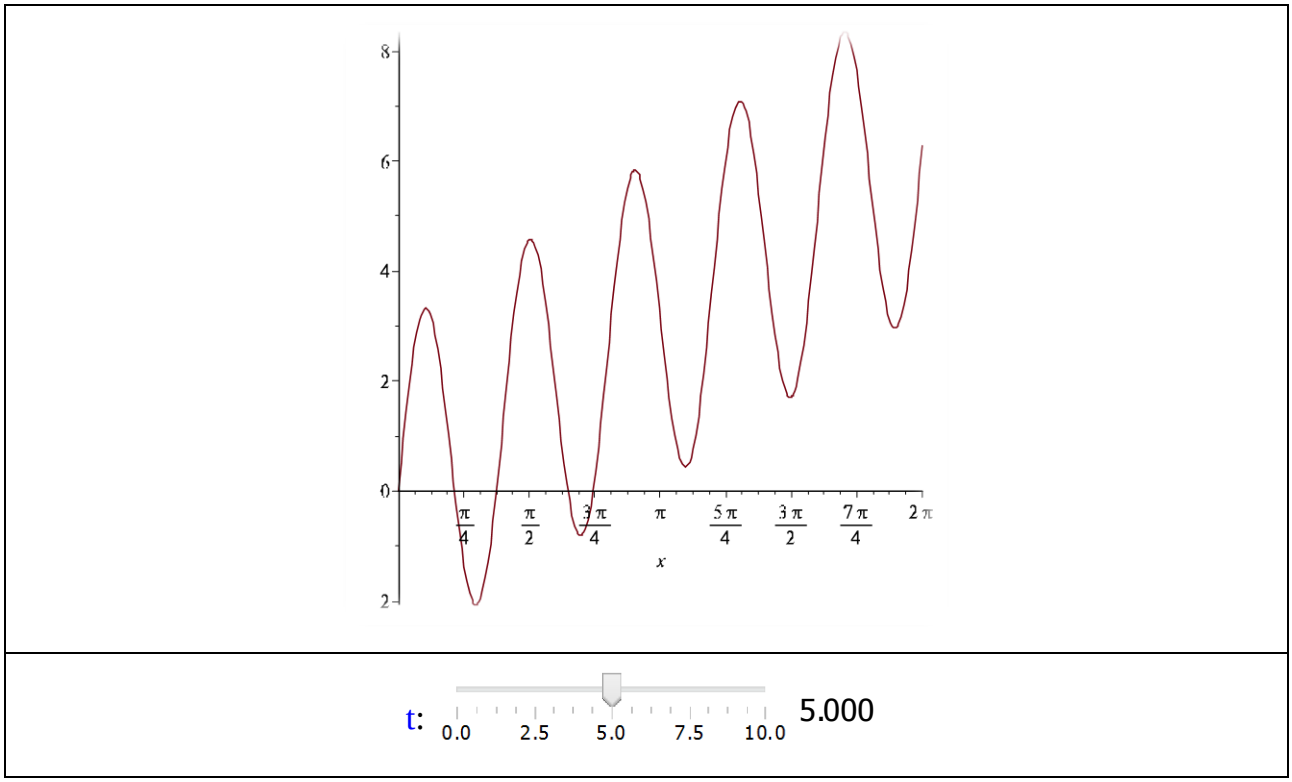
Với giao diện vẽ đồ thị **Plot Builder**, ta có thể vẽ các đồ thị tương tác, đó là các đồ thị mà người dùng có thể điều chỉnh được giá trị biến và làm thay đổi hình dạng đồ thị một cách trực tiếp. Để làm được điều này ta phải nhập một biểu thức theo 2 hoặc nhiều biến và chọn **Interactive Plot with 1 parameter** từ danh sách **Select Plot Type and Functions**.

***Ví dụ 6***: Vẽ đồ thị hàm số $y = x + 3\sin(xt), x \in [0, 2\pi], t \in [0, 10]$.

- Mở giao diện vẽ đồ thị **Plot Builder**, nhấn **Add** và nhập biểu thức **x+3*sin(x*t).**
- Trong cửa sổ **Select Plot Type**:
  - Ở phần **Select Plot and Functions**, chọn **Interactive Plot with 1 parameter**.

- Thay đổi miền của biến **x-Axis** là **0** .. **2*Pi.**
- Thay đổi khoảng giá trị của **t** là **0** .. **10**.
- Nhấn **Plot** để vẽ đồ thị vào giao diện đang làm việc.
  Chú ý rằng ta có thể thiết lập một số tùy chọn cho đồ thị bằng cách nhấn **Options** trước khi nhấn **Plot** để vẽ đồ thị**.**

➢ Sử dụng thanh trượt để điều chỉnh giá trị tham số **t** và xem sự thay đổi của đồ thị khi tham số thay đổi.




### 3.1.2. Menu ngữ cảnh (Context menu)

Menu ngữ cảnh của Maple chứa một danh sách các lệnh để thao tác, hiển thị hoặc tính toán các biểu thức. Tùy thuộc vào từng loại biểu thức mà menu ngữ cảnh hiển thị các lệnh tương ứng. Để vẽ đồ thị, ta kích chuột phải vào biểu thức và chọn một trong các lệnh:

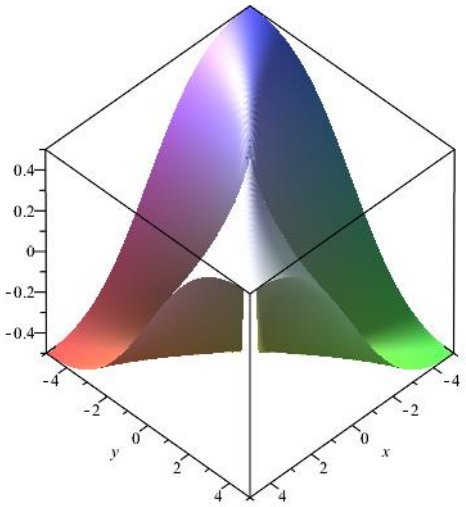
- 2-D or 3-D plot
- 2-D or 3-D implicit plot
- Plot Builder

***Ví dụ***: vẽ đồ thị hàm số $z = \frac{xy}{x^2+y^2}$

➢ Ở giao diện chính, nhập vào biểu thức: $\frac{xy}{x^2+y^2}$.

➢ Kích chuột phải vào biểu thức trên để mở menu ngữ cảnh

➢ Chọn **Plots→ 3-D Plot → x,y**.

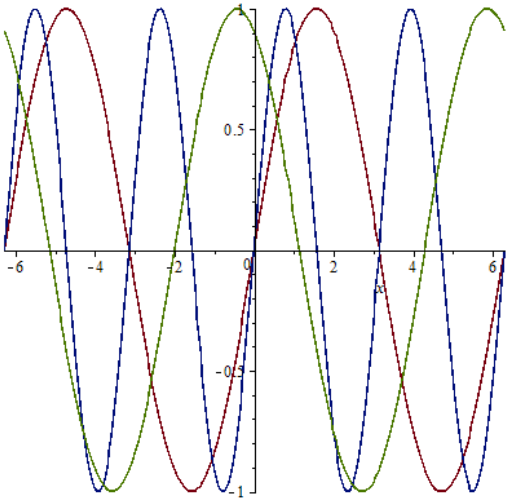



### 3.1.3. Kéo đồ thị đến vùng đồ thị khác

Bằng cách kéo và thả (drag-and-drop) ta có thể chèn một đồ thị vào vùng đồ thị đã có hoặc chèn một vùng đồ thị trống vào văn bản. Các vùng đồ thị trống có thể là 2 hoặc 3 chiều. Phương pháp này giúp ta dễ dàng thêm vào hoặc gỡ bỏ các đồ thị ra khỏi vùng đồ thị và nó độc lập với cú pháp lệnh vẽ đồ thị.

***Ví dụ***: Vẽ đồ thị các hàm $y = \sin(x)$ , $y = \sin(x + 2)$ và $y = \sin(2x)$ trên cùng một hệ trục tọa độ.

- Từ menu **Insert,** chọn **Plot → 2-D** để chèn một vùng đồ thị trống vào văn bản.
- Nhập vào biểu thức $\sin(x)$.
- Chọn toàn bộ biểu thức $\sin(x)$ rồi dùng chuột kéo vào vùng đồ thị trống trên, nếu muốn chỉ sao chép biểu thức thì ta nhấn thêm nút **Ctrl** khi kéo.
- Lặp lại các bước 2 và 3 đối với các biểu thức: $\sin(2 * x)$ và $\sin(x + 2)$.
- Để gỡ bỏ một đồ thị ta kéo đồ thị đó ra khỏi vùng đồ thị.



### 3.1.4. Các lệnh vẽ đồ thị

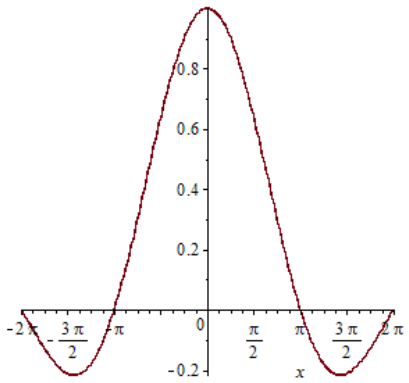
Ngoài các phương pháp nêu trên ta có thể dùng lệnh để vẽ đồ thị. Có 2 lệnh cơ bản là **plot** dùng vẽ các đồ thị hàm một biến và **plot3d** để vẽ các đồ thị trong không gian 3 chiều. Cú pháp lệnh cụ thể như sau:

$$\boldsymbol{plot(f(x), x = a..b, opts)}$$
$$\boldsymbol{plot3d(f(x,y), x = a..b, y = c..d, opts)}$$

trong đó: ***f(x), f(x,y)*** là các hàm (biểu thức) để vẽ đồ thị, ***opts*** là các tùy chọn của lệnh. Tập hợp các lệnh vẽ đồ thị được Maple xây dựng trong gói lệnh **plots**. Để có thể so sánh được ưu nhược điểm trong các cách vẽ đồ thị ta sẽ dùng lệnh vẽ lại các đồ thị ở các ví dụ trước.

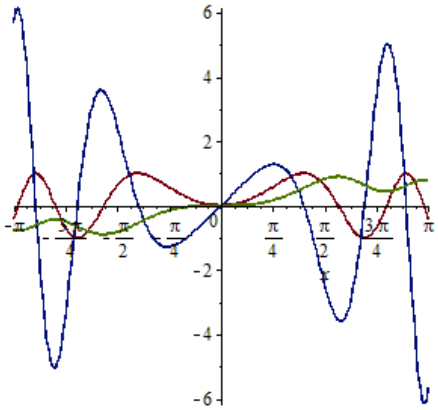
***Ví dụ 1***: Vẽ đồ thị hàm số $y = \frac{\sin(x)}{x}$ trên khoảng $(-2\pi, 2\pi)$.

```
plot(sin(x)/x, x = -2*Pi..2*Pi)
```



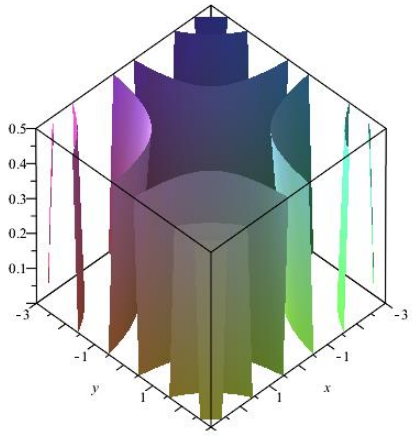

***Ví dụ 2***: Vẽ đồ thị các hàm số $y = f(x) = \sin(x^2), y = f'(x)$ và $y = \int f(x)dx$ trên cùng một hệ trục tọa độ.

```
plot([sin(x^2), diff(sin(x^2), x), int(sin(x^2), x)], x = -Pi..Pi)
```



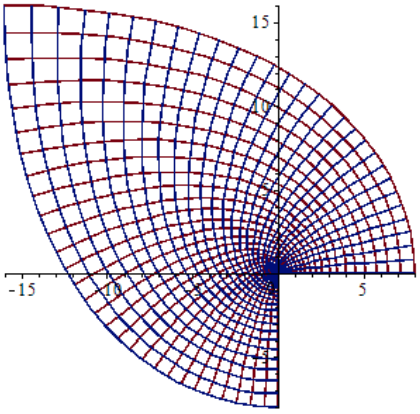
***Ví dụ 3***: Vẽ đồ thị hàm số $z = f(x, y) = \cos(xy)$ trên miền $[-3, 3] \times [-3, 3]$.

$plot3d(cos(x * y), x = -3..3, y = -3..3, view = 0..0.5, style = surface, grid = [40, 40])$



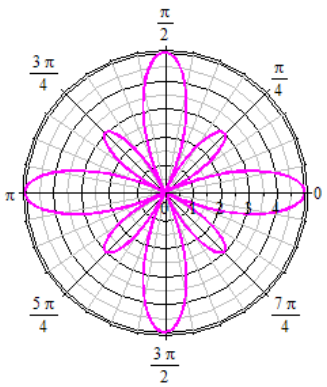

***Ví dụ 4***: Vẽ đồ thị hàm số phức $y = z^3$ trên khoảng $(0, 2 + 2I)$.

$plots[conformal](z\text{^}3, z = 0..2 + 2 * I, axes = normal, grid = [30, 30])$



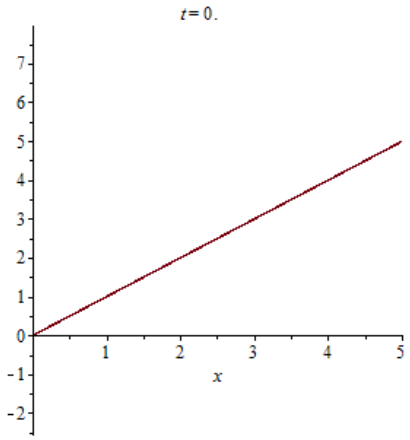

***Ví dụ 5***: Vẽ đồ thị $y = 1 + 4\cos(4\theta)$, $\theta \in (0, 8\pi)$.

$plots[polarplot](1 + 4 * cos(4 * theta), theta = 0..8 * Pi, color = magenta)$

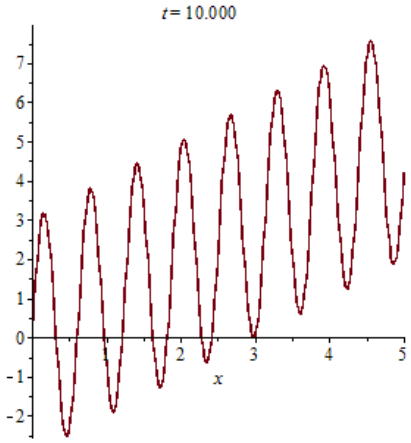


***Ví dụ 6***: Vẽ đồ thị hàm số $y = x + 3\sin(xt)$, $x \in [0, 2\pi], t \in [0, 10]$.

```
plots[animate](plot,[x + 3 * sin(x * t),x = 0..5],t = 0..10)
```



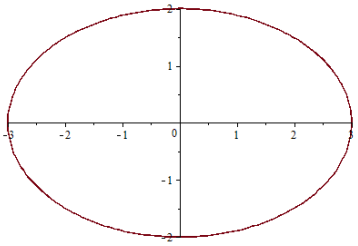

Kích chuột vào đồ thị ta sẽ thấy trên giao diện chính hiện ra thanh công cụ **Animation**, kéo thanh trượt hoặc nhấn  để thấy được sự thay đổi của đồ thị khi ***t*** biến thiên.




***Ví dụ 7***: Vẽ elip được cho dưới dạng phương trình tham số:

$$\begin{cases} x = 3\cos(t) \\ y = 2\sin(t) \end{cases}, t \in [0,2\pi]$$

```
plot([3 * cos(t),2 * sin(t),t = 0..2 * Pi],scaling = constrained)
```



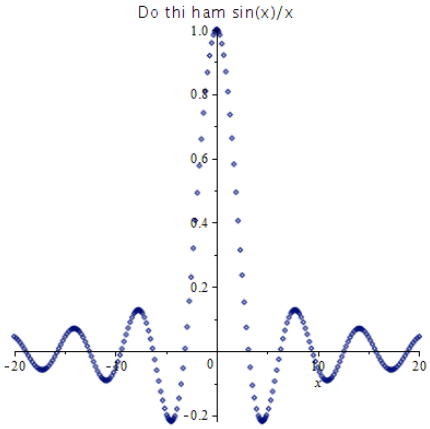
Khi dùng lệnh để vẽ đồ thị thì việc sử dụng các tùy chọn sẽ linh hoạt hơn so với các phương pháp trước. Để tìm hiểu thêm về các tùy chọn của lệnh vẽ đồ thị tham khảo mục plot/options và plot3d/options trong phần trợ giúp Help. Bảng sau giới thiệu sơ lượt một số tùy chọn:

| **Tùy chọn** | **Mô tả** |
|---|---|
| ***axes*** | định nghĩa các loại hệ trục như: **boxed**, **frame**, **none**, hoặc **normal** |
| ***caption*** | đặt chú giải cho đồ thị |
| ***color*** | xác định màu cho các đường đồ thị |
| ***font*** | xác định font chữ cho các văn bản hiển thị cùng đồ thị |
| ***glossiness (3-D)*** | điều chỉnh vùng sáng phản chiếu từ các mặt |
| ***gridlines (2-D)*** | thiết lập lưới ô cho đồ thị |
| ***lightmodel (3-D)*** | điều chỉnh kiểu ánh sáng chiếu vào đồ thị như: **none**, **light1**, **light2**, **light3**, hoặc **light4** |
| ***linestyle*** | thiết lập loại đường để vẽ đồ thị như: **dot**, **dash**, **dashdot**, **longdash**, **solid**, **spacedash**, và **spacedot** |
| ***legend (2-D)*** | thiết lập ghi chú cho đồ thị |
| ***numpoints*** | điều chỉnh toàn bộ số điểm sinh ra khi vẽ đồ thị |
| ***scaling*** | điều chỉnh tỉ lệ các trục: **constrained** hoặc **unconstrained** |
| ***shading (3-D)*** | thiết lập kiểu tô màu cho các mặt như: theo trục **xyz**, **xy**, **z**, **zgrayscale**, **zhue**, hoặc **none** |
| ***style*** | thiết lập kiểu vẽ như: **line**, **point**, **polygon**, hoặc **polygonoutline** cho đồ thị 2-D; **contour**, **point**, **surface**, **surfacecontour**, **surfacewireframe**, **wireframe**, hoặc **wireframeopaque** cho đồ thị 3-D. |
| ***symbol*** | thiết lập ký hiệu của các điểm trên đồ thị như: **asterisk**, **box**, **circle**, **cross**, **diagonalcross**, **diamond**, **point**, **solidbox**, **solidcircle**, hoặc **soliddiamond** cho đồ thị 2-D; **asterisk**, **box**, **circle**, **cross**, **diagonalcross**, **diamond**, **point**, **solidsphere** cho đồ thị 3-D. |

| | |
|---|---|
| ***title*** | đặt tiêu đề cho đồ thị |
| ***thickness*** | thiết lập độ dày của các đường trong đồ thị |
| ***transparency (3-D)*** | thiết lập độ trong suốt của các mặt |
| ***view*** | thiết lập giá trị lớn nhất và bé nhất của các trục tọa độ khi hiển thị đồ thị |

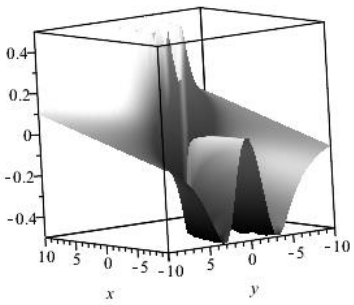
***Ví dụ 8***: Vẽ đồ thị hàm số $y = \frac{\sin(x)}{x}, x \in [-20, 20]$ với các tùy chọn tạo tiêu đề cho đồ thị, đổi kiểu vẽ đồ thị (style), thay đổi màu đồ thị.

$$plot(sin(x)/x, x = -20..20, title = "Do\ thi\ ham\ sin(x)/x", titlefont = [HELVETICA, 12], color = "Niagara\ 2", style = point)$$



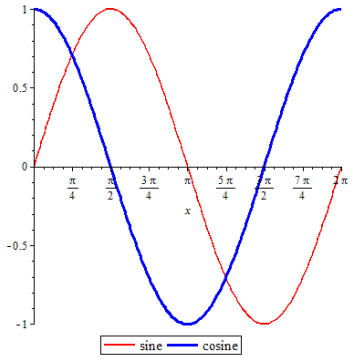

***Ví dụ 9***: Vẽ đồ thị hàm số $y = \frac{xy^2}{x^2+y^4}, x \in [-10, 10], y \in [-10,10]$ với các tùy chọn: numpoints, lightmodel, shading, orientation và style. Việc thiết lập số lượng điểm vẽ numpoints càng cao sẽ làm cho đồ thị trơn hơn và chính xác hơn.

$$plot3d(x * y\hat{}2/(y\hat{}4 + x\hat{}2), x = -10..10, y = -10..10, axes = boxed, numpoints = 1500, lightmodel = light3, shading = zgrayscale, orientation = [160, 20], style = patchnogrid)$$



***Ví dụ 10:*** Vẽ các đồ thị $y = \sin(x)$, $y = \cos(x)$, $x \in [0,2\pi]$ và ghi chú thích cho đồ thị.

```
plot([sin(x), cos(x)], x = 0..2 * Pi, color = [red, blue], thickness = [1,3], legend
    = ["sine", "cosine"])
```



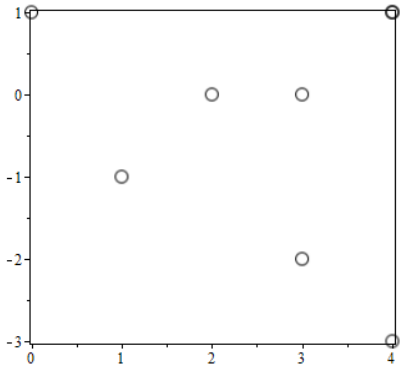

### 3.1.5. Một số dạng đồ thị đặc biệt

Gói lệnh **plots** chứa nhiều lệnh vẽ đồ thị đặc biệt như: ***animate***, ***contourplot***, ***densityplot***, ***fieldplot***, ***odeplot***, ***matrixplot***, ***spacecurve***, ***textplot***, ***tubeplot***... Dưới đây ta xét một số dạng đồ thị thường gặp.

#### a. Đồ thị điểm

Để vẽ đồ thị biểu diễn các điểm hoặc các dữ liệu số ta sử dụng lệnh ***pointplot***. Lưu ý rằng các điểm phải được sắp xếp theo một danh sách có dạng $[[x_1, y_1], [x_2, y_2], \dots, [x_n, y_n]]$. Theo mặc định, Maple không nối các điểm, để vẽ đường nối các điểm ta sử dụng thêm tùy chọn '***style = line'***.

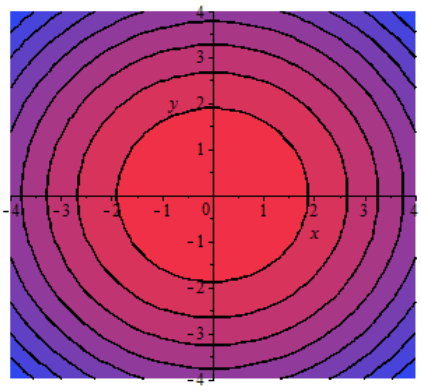
```
pointplot([[0, 1], [1, −1], [3, 0], [4, −3], [2, 0], [4, 1], [3, −2], [4, 1]], axes
    = BOXED, symbolsize = 25, symbol = circle)
```




#### b. Các đồ thị mức

Với một số dạng đồ thị khó thì đường mức hoặc mặt mức là một trong những phương tiện hữu hiệu giúp ta biết thêm nhiều tính chất của hàm số. Để vẽ các đồ thị mức ta sử dụng lệnh ***contourplot*** hoặc ***contourplot3d.***

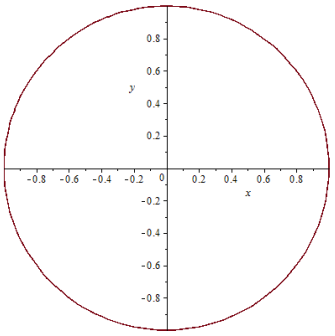
```
contourplot(x^2 + y^2, x = -4..4, y = -4..4, filled = true, numpoints
    = 750)
```



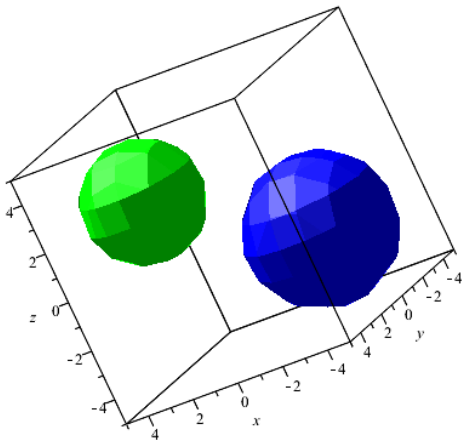
### c. Đồ thị hàm ẩn

Maple có thể vẽ được đồ thị các hàm ẩn bằng các lệnh ***implicitplot*** và ***implicitplot3d***.

```
with(plots):
implicitplot(x^2 + y^2 = 1, x = -1..1, y = -1..1)
```



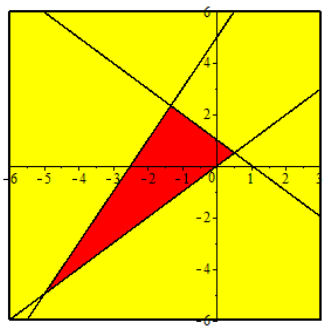
```
implicitplot3d([(x + 2)^2 + (y + 2)^2 + (z + 2)^2 = 9, (x - 2)^2 + (y - 2)^2 + (z - 2)^2
    = 6], x = -5..5, y = -5..5, z = -5..5, color
    = [blue, green], scaling = constrained, axes = boxed)
```



### d. Miền giao nhau của các bất phương trình

Muốn vẽ miền chấp nhận được của một hệ bất phương trình ta sử dụng lệnh ***inequal***.

$$inequal(\{5 + 2 * x >= y, x - y <= 0, x + y <= 1\}, x = -6..3, y = -6..6, optionsfeasible = (color = red), optionsexcluded = (color = yellow))$$

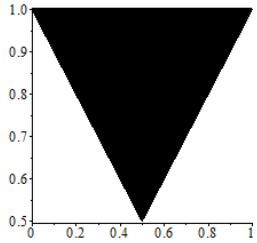


### e. Vẽ đa giác

Cho ***L*** là một danh sách các điểm. Khi đó, lệnh ***polygonplot(L)*** sẽ tạo ra một đa giác mà các đỉnh là các điểm trong ***L***.

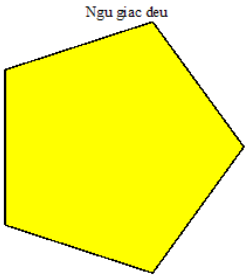
$$L := \left[[0,1],[1,1],\left[\frac{1}{2},\frac{1}{2}\right]\right]:$$

$$polygonplot(L)$$



$$dagiac := n \to \left[seq\left(\left[cos\left(2 * Pi * \frac{i}{n}\right), sin\left(2 * Pi * \frac{i}{n}\right)\right], i = 1..n\right)\right]:$$

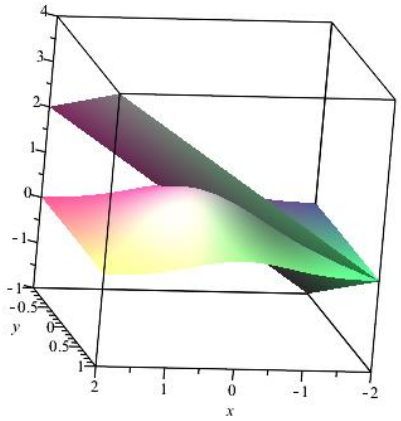
$$polygonplot(dagiac(5), scaling = constrained, axes = none, title = "Ngu giac deu", color = yellow)$$



### 3.1.6. Vẽ nhiều đồ thị trên cùng một vùng đồ thị

Như đã trình bày ở các ví dụ trước, để vẽ nhiều đồ thị trên cùng một vùng đồ thị ta đặt các biểu thức cần vẽ đồ thị vào trong một danh sách trong lệnh ***plot*** hoặc ***plot3d.***

$$plot3d([\exp(-x^2-y^2), x+y+1], x = -2..2, y = -1..1)$$



Ngoài ra, Maple còn cung cấp cho ta một lệnh khá hữu hiệu là ***display*** khi muốn hiển thị nhiều đồ thị, hình ảnh trên cùng một vùng đồ thị.

```
restart; with(plots): with(plottools):
tamgiac:= curve([[0,0], [1,0], [1/2,1/2 * sqrt(3)], [0,0]], color = red, thickness
          = 15, linestyle = 1)
a[0]:= display(arrow([0,0], [1,0], 0.2, 0.4, 0.3), color = blue, axes = none, scaling
          = constrained)
muiten := translate(scale(rotate(a[0], (1/2) * Pi), 1/4, 1/4), 1/2, 1/3)
text := textplot([1/2, (1/9) * sqrt(3), "One way"], color = blue, font
          = [HELVETICA, BOLDOBLIQUE, 20])
display({tamgiac, muiten, text}, scaling = constrained)
```

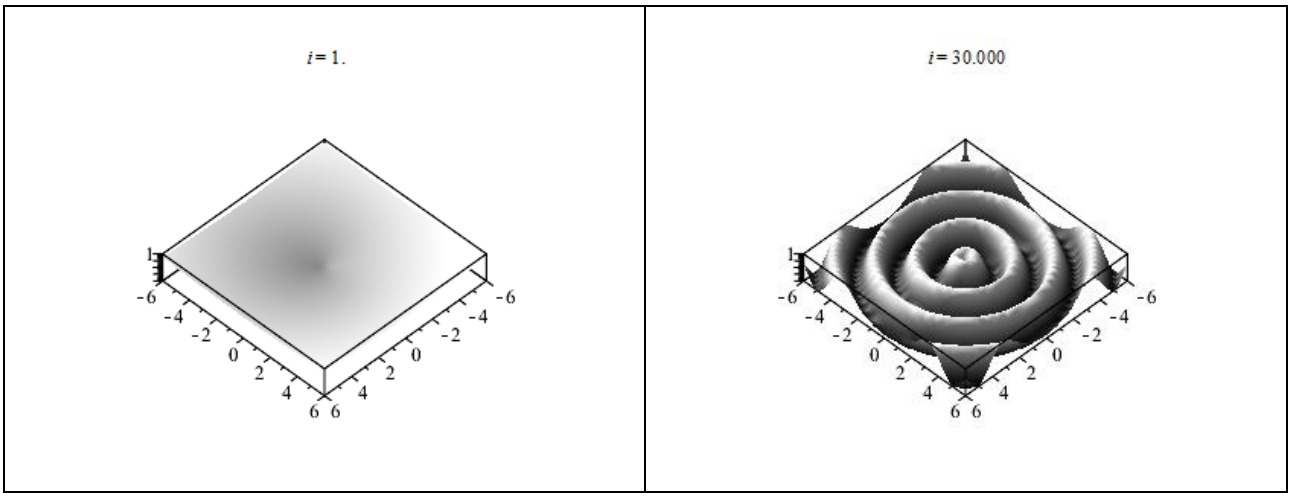
## 3.2. Sự vận động của đồ thị

Một trong những đặc tính nổi bật của Maple là ta có thể tạo ra các hình ảnh chuyển động của đồ thị. Tương tự như kỹ thuật làm phim hoạt hình, Maple tạo ra sự chuyển động này bằng cách ghép nối dãy các hình ảnh liên tiếp nhau. Muốn một đồ thị vận động được ta phải nhập biểu thức cần vẽ theo nhiều biến và cho một biến làm tham số, hình ảnh đồ thị sẽ thay đổi tùy thuộc vào giá trị tham số này. Có hai cách cơ bản để tạo một hình ảnh động cho đồ thị là dùng giao diện vẽ đồ thị tương tác (**Interactive Plot Builder)** hoặc dùng lệnh.

### 3.2.1. Sử dụng giao diện vẽ đồ thị tương tác

***Ví dụ:*** Vẽ đồ thị hàm số $z = \sin(\frac{i\sqrt{x^2+y^2}}{10})$ và tạo sự vận động của đồ thị khi tham số i thay đổi.

- Mở giao diện vẽ đồ thị: Tools/Asistant/Plot Buider, nhấn **Add** và nhập vào biểu thức: **sin(i*sqrt(x^2+y^2)/10).**
- Trong cửa sổ Select Plot Type:
  - Chọn **Animation** trong danh sách **Select Plot Type** and Functions.
  - Thay đổi ở **x Axis** là khoảng **-6** .. **6**.
  - Thay đổi ở **y Axis** là khoảng **-6** .. **6**.
  - Thay đổi tham số **Animation Parameter i** chạy trong khoảng **1** .. **30**.
  - Nhấn **Options**.
- Trong cửa sổ Plot Options:
  - Chọn **surface** trong ô **Style.**
  - Trong ô **Color**, chọn **red-turquoise** ở mục **Light Model**.
  - Chọn **z (grayscale)** ở mục **Shading**.
  - Ở ô **View**, chọn **Constrained Scaling.**
  - Nhấn **Plot** để hiển thị đồ thị.
- Nhấn nút **Play** trên thanh công cụ của **Animation** để xem đồ thị vận động.



### 3.2.2. Sử dụng lệnh animate

Bằng cách sử dụng lệnh ***animate*** kết hợp với các lệnh vẽ đồ thị như: ***plot, plot3d, polarplot,…*** để tạo sự vận động của đồ thị 2 chiều hoặc 3 chiều. Cú pháp lệnh như sau:

***animate(plotcommand, plotarguments, t=a..b, ops)***

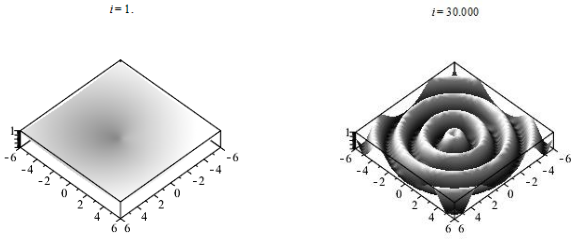
***animate(plotcommand, plotarguments, t=L, ops)***

trong đó:

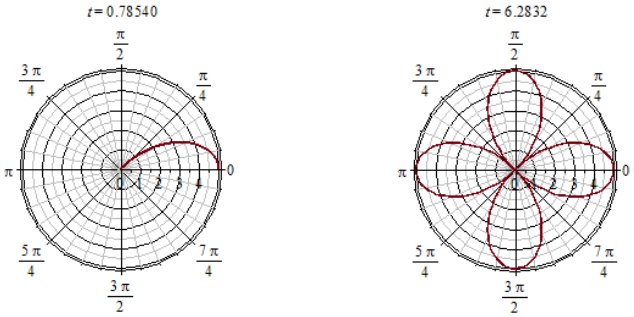
- ***plotcommand*:** lệnh vẽ đồ thị 2-D hoặc 3-D.
- ***plotarguments***: các tham số của lệnh vẽ đồ thị
- ***t=a..b***: tham số chuyển động và khoảng giá trị của nó
- ***t=L***: tham số chuyển động trong một danh sách các hằng số thực hoặc phức
- ***ops***: các tùy chọn (nếu có).

***Ví dụ***: tạo sự vận động cho đồ thị ở ví dụ trước bằng lệnh ***animate***

$$animate\left(plot3d, \left[\sin\left(\frac{i\sqrt{x^2+y^2}}{10}\right), x=-6..6, y=-6..6, style=patchnogrid, lightmodel=light3, shading=zgrayscale, scaling=constrained\right], i=1..30\right)$$




$$animate\left(polarplot, [5*cos(2\theta), \theta=0..t], t=\frac{\pi}{4}..2\pi, frames=50\right)$$



Ngoài ra, ta có thể thiết lập thêm một số tùy chọn cho lệnh để có thể tạo sự chuyển động uyển chuyển hơn của đồ thị. Một trong những tùy chọn hay dùng là tăng số lượng ảnh (frames). Theo mặc định, chuyển động của đồ thị trong không gian 2 chiều gồm 16 ảnh và trong không gian 3 chiều là 8 ảnh. Thiết lập lại ***frame*** để tăng số lượng ảnh, tuy nhiên khi tăng số ảnh thì yêu cầu về bộ nhớ và thời gian tính toán cũng tăng.

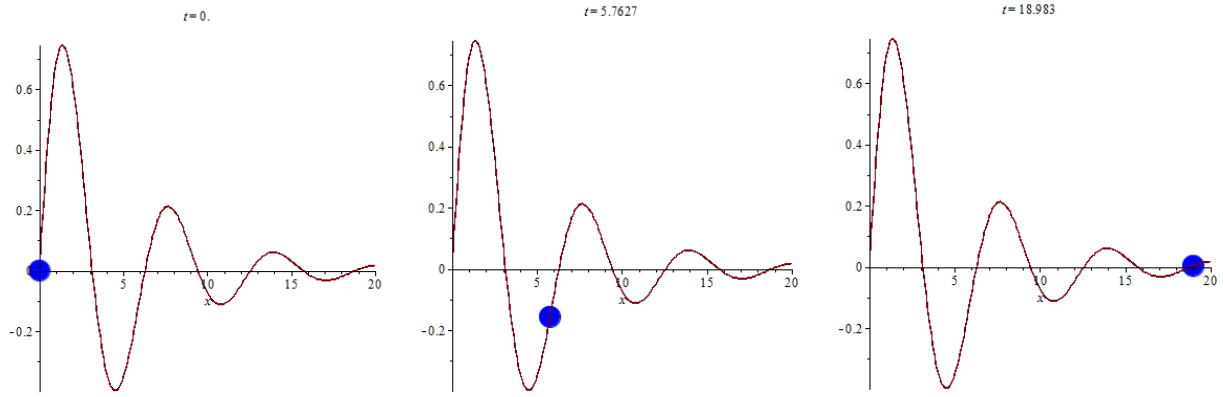
$$sinwave := plot\left(\sin(x)\, e^{-\frac{x}{5}}, x = 0..20\right)$$

$$\textbf{ball} := \textbf{proc}\,(x, y)$$

$$plots[pointplot]\left([[x, y]], symbol = solidcircle, symbolsize = 40, color = blue\right)$$

$$\textbf{end proc}:$$

$$plots[animate](ball, [t, sin(t) * exp(-(1/5) * t)], t = 0..20, frames = 60,$$
$$background = sinewave)$$

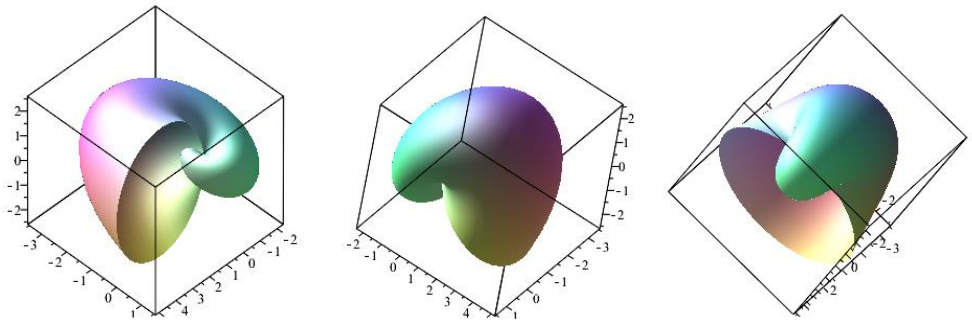



### 3.2.3. Chuyển động của các vật thể

Ta có thể sử dụng lệnh **plot3d** với tùy chọn **viewpoint** để tạo ra sự chuyển động của các vật thể trong không gian 3 chiều theo nhiều hướng khác nhau và ở các góc độ khác nhau tùy thuộc vào các tham số cũng như các loại hệ trục được thiết lập. Với lệnh này ta có thể tạo được các chuyển động như: bay qua, quay quanh, chuyển động sang bên, chuyển động hướng tới... trên các vật thể trong không gian 3 chiều. Hãy hình dung như đang điều khiển một *camera*, góc độ của máy quay thay đổi ta sẽ thấy hình ảnh của vật thể cũng thay đổi. Nói cách khác, chuyển động của vật thể do đường quay của máy quay thay đổi.
Trong lệnh này, ta có thể thiết lập hướng của camera để xem các mặt khác nhau của vật thể, thiết lập các đường quay để camera có thể xuyên qua hoặc quay quanh vật thể và có thể xác định tọa độ để di chuyển camera đến các điểm xác định bên cạnh vật thể, tạo các khoảng quay để camera đến gần hoặc xa vật thể. Thông tin chi tiết có thể xem tại **viewpoint** của trang trợ giúp Help.

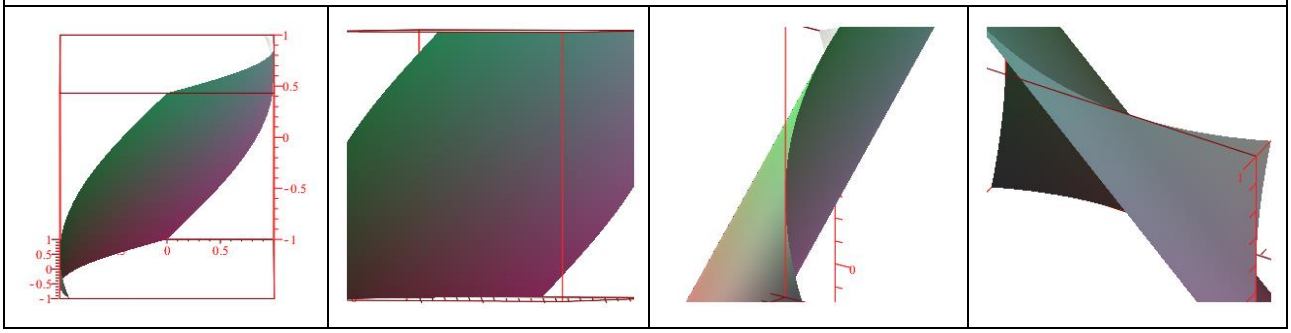
**Ví dụ 1**: Thiết lập các tùy chọn để di chuyển camera xung quanh một vật thể

$$plot3d(1.3^x * \sin(y), x = -1..2 * Pi, y = 0..Pi, coords = spherical, style = patch, viewpoint = [\text{circleleft}])$$



***Ví dụ 2***: Thiết lập một đường quay để camera tiến tới và di chuyển xung quanh vật thể.

$$plot3d(sin(x+y), x = -1..1, y = -1..1, shading = xyz, viewpoint = [path = [[50 * x, 90 * cos(x), 100 * sin(x)], x = -2 * Pi..Pi]])$$



## 3.3. Xuất ảnh

Ta có thể xuất hình ảnh các đồ thị ra theo nhiều định dạng khác nhau như: DXF, X3D (cho các hình ảnh 3-D), EPS, GIF, JPEG/JPG, POV, Windows BMP, and WMF. Với các hình động ta có thể xuất ra theo dạng file ảnh động GIF. Các loại ảnh xuất ra này ta có thể chèn vào trong các file trình diễn, các trang web, Microsoft Word, hoặc các phần mềm khác.

**Để xuất hình ảnh một đồ thị:**

- Kích chuột phải vào vùng đồ thị
- Chọn **Export**và sau đó chọn định dạng file xuất ra.

Hoặc:

- Chọn đồ thị cần xuất ảnh
- Nhấn **Plot** trên thanh menu rồi chọn **Export** và sau đó chọn định dạng file xuất ra.

# Chương 4

# LẬP TRÌNH CƠ BẢN VỚI MAPLE

Maple là công cụ dùng cho việc tính toán hình thức đồng thời cũng là một ngôn ngữ lập trình. Vì vậy, Maple cung cấp rất nhiều hàm toán học cũng như các hàm đáp ứng cho việc lập trình để từ đó người dùng có thể tự xây dựng các thủ tục hoặc chu trình tính toán.

Một trong những cách sử dụng hiệu quả các hàm do Maple cung cấp là kết hợp chúng với nhau. Có nhiều cách để làm được điều này, trong đó cách đơn giản nhất là lập các hàm lồng nhau. Ta có thể gọi các hàm lồng nhau nhờ cơ chế cho giá trị của các hàm, tức là các hàm sau khi thực hiện một thao tác theo các tham số thường cho các giá trị là kết quả của các phép tính, ta có thể lấy ngay kết quả của phép tính này để làm tham số cho các hàm tiếp theo, và cứ tiếp tục như vậy cho đến khi thu được kết quả mong muốn. Nếu làm tốt điều này ta sẽ phát huy được sức mạnh của hệ thống thư viện của Maple.

Để viết một chương trình (hàm, chu trình) trên Maple không cần bạn phải có một kỹ năng lập trình cao bởi vì Maple đã có sẵn một thư viện khổng lồ để hỗ trợ bạn. Không giống như các ngôn ngữ lập trình khác, trong Maple ta có thể sử dụng trực tiếp các hàm được tạo sẵn dùng cho việc tính toán hình thức mà không phải xây dựng lại nó. 90% trong hàng nghìn hàm của Maple là các chương trình được viết bằng ngôn ngữ lập trình Maple. Ta có thể đọc được mã lệnh các hàm này và có thể chỉnh sửa lại cho phù hợp với nhu cầu sử dụng hoặc có thể mở rộng để ứng dụng cho các bài toán khác. Điều này cho thấy Maple là một ngôn ngữ lập trình có tính tương tác cao với người sử dụng.

Toàn bộ hệ thống Maple có thể chia làm 3 thành phần cơ bản:

- **Kernel**: bao gồm một số các hàm cơ sở được viết bằng ngôn ngữ C và đã được dịch. Khi Maple khởi động thì các hàm này cũng được gọi ra (loading). Ta không thể xem được mã nguồn của các hàm này vì chúng đã được biên dịch.
- **Thư viện:**là nơi chứa các chương trình (hàm, thủ tục) được viết bằng ngôn ngữ Maple bao gồm: các hàm liên quan đến tính toán số học, đại số tuyến tính, thống kê, đồ thị,... và các gói hàm (packages) của nhiều lĩnh vực trong toán học.
- **Giao diện**: nơi ta có thể nhập hoặc cho ra các biểu thức, các hàm toán học...

## 4.1. Các cấu trúc dữ liệu cơ bản

### 4.1.1. Dãy các biểu thức (Expression Sequence)

Đây là một cấu trúc dữ liệu cơ bản của Maple bao gồm một nhóm các biểu thức được phân cách với nhau bởi dấu phẩy. Để truy nhập vào từng phần tử của dãy ta sử dụng công thức:

**<tên dãy>[số thứ tự của phần tử]**

***Ví dụ***:

$S := 2, y, \sin(x^2), I:$

$S[2]$ (lấy thành phần thứ 2 của dãy)

$y$

$S[-2]$ (thứ tự tính từ cuối dãy)

$\sin(x^2)$

$S[2..-2]$

$y, \sin(x^2)$

### 4.1.2. Tập hợp (Set)

Một tập hợp là một dãy các biểu thức được giới hạn bởi hai dấu ngoặc {}. Một tập hợp trên Maple có những tính chất như tập hợp trong toán học:

- Mỗi phần tử là duy nhất, các phần tử được lặp lại nhiều lần chỉ lưu 1 lần
- Không phân biệt thứ tự các phần tử trong tập hợp

Các phép toán trên tập hợp: cho 2 tập hợp $S_1$ và $S_2$

| | |
|---|---|
| $S_1$ ***union*** $S_2$ | $S_1 \cup S_2$ |
| $S_1$ ***intersect*** $S_2$ | $S_1 \cap S_2$ |
| $S_1$ ***minus*** $S_2$ | $S_1 \backslash S_2$ |
| $S_1$ ***subset*** $S_2$ | $S_1 \subseteq S_2$ |

***Ví dụ***:

$A := \{1, 2, 3, a, b\}$

$A := \{1, 2, 3, a, b\}$

$B := \{-3, a, -1, 2, a, b\}$

$B := \{-3, -1, 2, a, b\}$

$A$ ***union*** $B$

$\{-3, -1, 1, 2, 3, a, b\}$

$A$ ***intersect*** $B$

$\{2, a, b\}$

$A$ ***minus*** $B$

$\{1, 3\}$

$member(a, A)$ (kiểm tra phần tử $a$ có nằm trong tập $A$)

$$true$$

### 4.1.3. Danh sách (List)

Một danh sách là một dãy các biểu thức được gộp trong dấu ngoặc **[].** Lưu ý rằng **d**anh sách bảo toàn thứ tự các phần tử và cho phép lặp các phần tử. Để truy nhập vào từng phần tử của danh sách ta sử dụng công thức giống như ở dãy các biểu thức. Ví dụ:

$L1 := [x, y, z, y]$; $L2 := [a, b, c, d]$

$$L1 := [x, y, z, y]$$
$$L2 := [a, b, c, d]$$

$L := [op(L1), op(L2)]$

$$L := [x, y, z, y, a, b, c, d]$$

$s := seq(i/(i+1), i = 1..6)$

$$s := \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}$$

$M := [s]$

$$M := \left[\frac{1}{2}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}\right]$$

$M[2..5]$

$$\left[\frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}\right]$$

Trong các ví dụ trên thì hàm $op(L)$ cho ra các thành phần (operand) của danh sách $L$.

### 4.1.4. Mảng (Array)

Một mảng là một cấu trúc danh sách ở dạng tổng quát, gồm 2 đặc điểm quan trọng sau:

- Chỉ số của phần tử có thể là một số nguyên bất kỳ
- Chiều của mảng có thể lớn hơn 1 (danh sách chỉ có chiều là 1)

Để tạo một mảng ta sử dụng hàm **Array()** gồm 2 tham số:

- Khoảng chỉ số của mảng, chỉ số cho mỗi chiều
- Danh sách các phần tử

$a := Array(1..3, 1..3, [[1, 2, 3], [4, 5, 6], [7, 8, 9]])$

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$b := Array(1..2, 1..2, 1..2)$

$$b := \begin{bmatrix} 1..2\ x\ 1..2\ x\ 1..2\ Array \\ Data\ Type: anything \\ Storage: rectangular \\ Oder: Fortran\_order \end{bmatrix}$$

$a[2,3]$

$$6$$

$c := Array(1..2, 1..2)$

$$c := \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

$c[1,1] := 1;\ c[1,2] := 2:$

$c[2,1] := 3;\ c[2,2] := 4:$

$c$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Đặc biệt, nếu tạo các mảng một chiều hoặc hai chiều mà số chiều lớn hơn 10 thì Maple chỉ hiển thị dưới dạng thông báo. Để xem chi tiết mảng ta có thể kích đôi chuột vào thông báo hoặc kích chuột phải chọn **Browse**.

$Array(1..15, 1..15)$

$$b := \begin{bmatrix} 1..15\ x\ 1..15\ Array \\ Data\ Type: anything \\ Storage: rectangular \\ Oder: Fortran\_order \end{bmatrix}$$

### 4.1.5. Bảng (Table)

Bảng là khái niệm mở rộng của cấu trúc dữ liệu mảng 2 chiều, nhưng cấu trúc dữ liệu bảng được tạo bằng cấu trúc bảng băm. Bảng băm là một cấu trúc dữ liệu sử dụng hàm băm để ánh xạ từ giá trị xác định, được gọi là khóa (ví dụ như tên của một người), đến giá trị tương ứng (như số điện thoại của họ). Do đó, các chỉ mục của bảng có thể là giá trị bất kỳ, không chỉ các số nguyên như của mảng. Để tạo một bảng ta sử dụng hàm ***table()***.

$Greek := table([a = alpha, b = beta, c = gamma])$

$$Greek := table([a = \alpha, b = \beta, c = \gamma])$$

$Greek[b]$

$$\beta$$

$L := [a, b, c, d]:$

$table(L)$

$$table([1 = a, 2 = b, 3 = c, 4 = d])$$

$F := table([sin = cos, cos = -sin]):$

$F_{cos}\left(\frac{\pi}{2}\right)$

$$-1$$

### 4.1.6. Chuỗi ký tự (Strings)

Một chuỗi ký tự chính là một dãy các ký tự được tập hợp trong ngoặc kép " ". Ta có thể truy nhập vào từng phần tử của chuỗi bằng cách sử dụng công thức giống dãy. Maple xây dựng gói lệnh **StringTools** gồm nhiều lệnh thao tác hữu hiệu trên chuỗi.

$with(StringTools)$:

$S := \ "This\ is\ a\ sequence\ of\ characters."$

S := "This is a sequence of characters."

$S[9\,..\,18]$

"a sequence"

$length(S)$

33

$Random(9,'alnum')$ (tạo một chuỗi ngẫu nhiên)

"8dzrI9ema"

$Split("Create\ a\ list\ of\ strings\ from\ the\ words\ in\ a\ string")$

["Create", "a", "list", "of", "strings", "from", "the", "words", "in", "a", "string"]

$convert(a-b+c*d/e, string)$

$"a-b+c*d/e"$

Có thể tìm hiểu thêm về các lệnh trên chuỗi ký tự thông qua thư viện **StringTools** trong phần trợ giúp Help.

## 4.2. Các cấu trúc điều khiển

Như hầu hết các ngôn ngữ lập trình khác, Maple cũng xây dựng các cấu trúc điều khiển để quản lý quá trình thực hiện lệnh trong các hàm, chu trình. Có hai cấu trúc điều khiển cơ bản là cấu trúc điều kiện và cấu trúc lặp.

### 4.2.1. Cấu trúc điều kiện *if*

Với cấu trúc này, Maple chỉ thực hiện thao tác khi điều kiện thỏa mãn (thao tác ở đây có thể là một lệnh hoặc một nhóm lệnh). Điều kiện trong lệnh **if** phải là một biểu thức *Boolean* (***true***, ***false***, hoặc ***FAIL***).

***Cú pháp:***

**if** *<biểu thức điều kiện>* **then**
    *<dãy lệnh 1>*
**else**
    *<dãy lệnh 2>*
**end if;**

***Ý nghĩa:*** *<dãy lệnh 1>* sẽ thực hiện nếu *<biểu thức điều kiện>* đúng, ngược lại sẽ thực hiện *<dãy lệnh 2>*.

Các <*biểu thức điều kiện*> trong Maple có thể được tạo thành bởi:

- Các toán tử quan hệ: <, <=, =, >=, >, <>
- Các toán tử logic: ***and, or, xor, implies, not***
- Các giá trị logic: ***true, false, FAIL***

Các câu lệnh **if** có thể lồng trong nhau, nghĩa là sau lệnh **else** ta có thể tiếp tục một câu lệnh **if** khác và viết gọn là **elif**.

***Ví dụ:***

```
x := 1173:
if isprime(x) then
    print(x `la so nguyen to`)
else
    ifactor(x);
end if;
                                (3)(17)(23)
y := 11:
if not type(y, integer) then
    printf("%a khong phai la so nguyen", y)
elif irem(y,2)=1 then
    printf("%a la so nguyen le.", y)
else
    printf("%a la so nguyen chan.", y)
fi;
11 la so nguyen le.
```

***Chú ý***: để xuống dòng trong một cấu trúc lệnh ta nhấn đồng thời tổ hợp phím ***Shift*** và ***Enter***, kết thúc lệnh **if** có thể dùng "**fi**" thay cho "**end if**".

### 4.2.2. Cấu trúc lặp

Để lặp lại một dãy các thao tác ta dùng cấu trúc lặp. Có 3 kiểu lặp cơ bản:

- Lặp theo biến đếm (**for/from**)
- Lặp theo thành phần của biểu thức (**for/in**)
- Lặp theo điều kiện (**while**)

**a. Vòng lặp *for/from***

***Cú pháp***:

**for** <*biến*> **from** <*start*> **by** <*change*> **to** <*finish*> **do**

<*dãy các thao tác*>

**end do;**

***Ý nghĩa***: lặp lại dãy các thao tác khi <*biến*> chạy từ <*start*> đến <*finish*> và mỗi vòng chạy biến tăng lên <*change*> đơn vị.

Chú ý rằng nếu từ khóa "**from**" và "**by**" bị bỏ qua trong lệnh **for** thì biến sẽ bắt đầu từ 1 và sau mỗi lần lặp biến tăng lên 1 đơn vị.

***Ví dụ:***

```
for i from 1 by 2 to 5 do
      print(i)
end do;
```

```
                                    1
                                    3
                                    5
```

```
for n from 10 by -1 to 3 do
  if isprime(n) then
      print(n);
  end if;
od;
```

```
                                    7
                                    5
                                    3
```

```
for n to 5 do
   if n mod 2=0 then
      printf("%d la so chan\n",n)
   fi;
od;
```

```
2 la so chan
4 la so chan
```

**b. Vòng lặp *for/in***

***Cú pháp***:

**for** <*biến*> **in** <*biểu thức*> **do**
    <*dãy các thao tác*>
**end do;**

***Ý nghĩa***: lặp lại dãy các thao tác khi <*biến*> lần lượt được gán cho từng thành phần của <*biểu thức*>, vòng lặp kết thúc khi không còn thành phần nào để gán.

***Ví dụ:***

```
dt:=0:
for i in [1,x,y,q^2,3] do
  dt:=dt+i
end do:
dt
```

$$q^2 + x + y + 4$$

### c. Vòng lặp *while*

***Cú pháp***:

**while** *<biểu thức điều kiện>* **do**
    *<dãy các thao tác>*
**end do;**

***Ý nghĩa***: vòng lặp while lặp lại dãy các thao tác cho đến khi biểu thức điều kiện có giá trị ***false*** hoặc ***FAIL***.

***Ví dụ***:

```
x := 15:
while x > 0 do
  irem(x, 2);
  x := iquo(x, 2);
end do;
```

$$1$$
$$x := 7$$
$$1$$
$$x := 3$$
$$1$$
$$x := 1$$
$$1$$
$$x := 0$$

```
a = 28; b := 12;
while b > 0 do
    c :=a mod b;
    a := b;
    b := c;
    if b = 0 then printf("Uoc chung lon nhat la: %d", a) end if
end do;
```

$$4$$
$$12$$
$$4$$
$$0$$
$$4$$
$$0$$

Uoc chung lon nhat la: 4

### d. Các vòng lặp vô hạn

Các vòng lặp mà không có điều kiện dừng được gọi là các vòng lặp vô hạn. Chẳng hạn, một vòng lặp while mà <biểu thức điều kiện> luôn đúng. Để tránh vòng lặp vô hạn ta có thể dùng các lệnh ngắt lặp giữa chừng như: **break**, **quit**, hoặc **return** lồng

vào trong cấu trúc lặp. Ngoài ra ta có thể ngắt lặp bằng biểu tượng  trên thanh công cụ.

## 4.3. Các hàm, chu trình (procedure)

Một hàm (chu trình) trong Maple là một chương trình (program) gồm các thao tác và tùy chọn. Khi gọi thực hiện hàm ta sẽ thu được đồng thời các kết quả của các thao tác đó.

### 4.3.1. Định nghĩa hàm

***Cú pháp:*** **<tên hàm>:= proc([<các tham biến>])**

[**local** <các biến cục bộ>]
[**global** <các biến toàn cục>]
[**option** <các tùy chọn>]
*<dãy các thao tác>*
**end proc;**

trong đó: *<tên hàm>* do người dùng đặt theo quy tắc đặt tên của Maple;

*<các tham biến>* là các biến dùng để nhận dữ liệu đầu vào (input) khi gọi hàm;

*<các biến cục bộ>* được khai báo trong hàm và chỉ có tác dụng ở bên trong hàm;

*<các biến toàn cục>* được khai báo trong hàm nhưng có tác dụng cả bên ngoài.

***Ví dụ***:

```
kc:= proc(x, y)
  sqrt(x^2 + y^2);
end proc;
                    proc (x, y) sqrt(x^2+y^2) end proc
kc(3,4)
                                5
```

Các tham biến, biến cục bộ và biến toàn cục có thể được khai báo kiểu dữ liệu cần sử dụng. Các kiểu dữ liệu này chính là các kiểu mà Maple định nghĩa như: ***array, complex, equation, even, integer, list, name, negint*** *(số nguyên âm)****, odd, posint*** *(số nguyên dương)****, prime, set, string***… Để biết thêm về những kiểu dữ liệu có trong Maple ta có thể tham khảo **type** trong phần trợ giúp Help. Đặc biệt, nếu không khai báo kiểu dữ liệu cho biến thì Maple hiểu rằng biến đó có thể nhận bất kỳ kiểu dữ liệu nào (anything).

```
SUM := proc(n::posint)                          #n là số nguyên dương
      local i, tong;                            #i, tong là các biến cục bộ
      tong := 0;                                #i, tong::anything
      for i from 1 to n do
          tong := tong + i;
      end do;
      tong;                                     #hàm này tính tổng của n số tự
end proc:                                       #nhiên đầu tiên
SUM(10)
                                55
SUM(sqrt(10))                                   #lỗi đầu vào
Error, invalid input: SUM expects its 1st argument, n, to be of type posint, but
received 10^(1/2)
```

***Chú ý***: để ghi chú các câu lệnh trong một chu trình ta dùng dấu # trước các ghi chú. Khi gọi thực hiện chu trình thì những câu ghi chú này sẽ bị bỏ qua.

### 4.3.2. Giá trị trả về của hàm

Khi thực hiện một hàm, Maple chỉ trả về kết quả của câu lệnh cuối cùng. Nếu muốn hàm trả về kết quả nào khác ta có thể dùng lệnh ***return***. Ví dụ sau sử dụng lệnh ***return*** để nhận kết quả trả về của hàm ***f*** được định nghĩa như sau:

$$f(x) = \begin{cases} x, & x < 0 \\ 1 - x, & 0 \le x \le 1 \\ x^2, & x > 1 \end{cases}$$

```
f := proc(x)
    if x<0 then
       return x;
    end if;
    if x >= 0 and x <= 1 then
       return 1 - x;
    end if;
    if x>1then
       return x^2
    end if;
end proc:
f(−0.4), f(0.3), f(2);
                          -0.4, 0.7, 4
```

### 4.3.3. Các tham số đặc biệt args và nargs

Khi định nghĩa một hàm trong Maple, đôi khi không cần phải khai báo cụ thể tên của các tham số đầu vào vì Maple đã có công cụ để quản lý điều này. Tham số **args** là một

tham số đặc biệt quản lý danh sách các tham số đầu vào của hàm và **nargs** là tham số quản lý số tham số đầu vào. Hai tham số đặc biệt này làm các chu trình uyển chuyển hơn trong việc nhập các tham số đầu vào. Xét các ví dụ sau:

```
thu := proc()
    print("Cac tham so cua ham la", args);
    print("Tham so dau tien la", args[1]);
    print("So tham so cua ham la", nargs)
end proc:
thu(a, b)
```

```
"Cac tham so cua ham la", a, b
"Tham so dau tien la", a
"So tham so cua ham la", 2
```

```
Max_min := proc()
    local i, m, n;
    if nargs = 0 then return -infinity end if;
    m := args[1];
    n := m;
    for i from 2 to nargs do
        if m < args[i] then m := args[i]
        elif args[i] < n then n := args[i] end if
    end do;
    return [m, n]
end proc:
Max_min(12,-23,5,34,1)
```

```
[34,-23]
```

### 4.3.4. Các hàm đệ quy

Phương pháp đệ quy là một trong những phương pháp cơ bản của lập trình. Nhiều bài toán lập trình bằng đệ quy sẽ đơn giản hơn lập trình không đệ quy. Tuy nhiên, một nhược điểm của đệ quy là tốn không gian nhớ và có khi tính toán trùng lặp. Maple cho phép các hàm được viết theo kiểu đệ quy và có thêm tùy chọn ***remember*** giúp khắc phục nhược điểm trên.

***Ví dụ***: viết hàm cho ra số hạng thứ n của dãy Fibonacci.

```
restart
Fibo1 := proc (n::nonnegint)
    if n < 2 then
        n
    else
        Fibo1(n-1)+Fibo1(n-2)
    end if
end proc:
Fibo1(10)
```

```
55
```

```
seq(Fibo1(i), i = 1 .. 7)
                                  1, 1, 2, 3, 5, 8, 13
time(Fibo1(20))                          #thời gian tính toán của hàm Fibo1(20)
                                  0.031
```

Bây giờ ta viết lại hàm Fibo trên thêm tùy chọn ***remember***. Tùy chọn này cho phép các kết quả tính toán của hàm lưu vào trong một bảng nhớ (giống bảng quy hoạch động), nếu gọi đệ quy gặp kết quả nào đã tính rồi thì sẽ lấy từ bảng mà không phải tính lại. Tùy chọn này sẽ làm giảm đáng kể không gian lưu trữ và thời gian tính.

```
restart
Fibo2 := proc (n::nonnegint)
      option remember;
      if n < 2 then
         n
      else
         Fibo1(n-1)+Fibo1(n-2)
      end if
end proc:
time(Fibo2(20))
                                  0.
time(Fibo2(2000));
                                  0.015
```

### 4.3.5. Hiển thị mã nguồn của hàm trong thư viện Maple

Một trong những đặc điểm nổi bật của Maple là hầu hết các hàm xây dựng sẵn (built-in) đều được viết bằng ngôn ngữ lập trình Maple và mã lệnh của các hàm này có thể được hiển thị. Bằng cách xem mã lệnh của các hàm trong Maple ta có thể học thêm nhiều điều về phương pháp lập trình trên Maple. Để hiển thị mã nguồn của những hàm này ta phải đặt lại giá trị **interface** với tùy chọn **verboseproc** bằng 2 và dùng lệnh **print**(<tên hàm>) hoặc **op**(<tên hàm>).

```
print(lcm);
                                  proc(a,b) ... end proc
interface(verboseproc=2)
print(lcm);
proc (a, b)
      options remember,
      Copyright (c) 1990 by the University of Waterloo. All rights reserved.;
      local q, t;
      if nargs = 0 then
         1
      elif nargs = 1 then
         t := expand(a); sign(t)*t
```

```
    elif 2 < nargs then
        foldl(procname, args)
    elif type(a, 'integer') and type(b, 'integer') then
        ilcm(a, b)
    else gcd(a, b, 'q'); q*b
    end if
end proc
```

### 4.3.6. Vùng soạn thảo mã lệnh

Vùng soạn thảo mã lệnh là nơi giúp cho việc lập trình tiện lợi hơn ở giao diện chuẩn. Chẳng hạn như khi xuống dòng ta chỉ nhấn **Enter** thay vì **Shift-Enter**, muốn thụt đầu dòng một khoảng lớn ta có thể nhấn phím **Tab** thay vì dùng nhiều lần phím **Space** hay có thể thu gọn đoạn mã lệnh lại còn một dòng,… Để chèn một vùng soạn thảo mã lệnh vào trong giao diện chuẩn ta chọn menu **Insert→Code Edit Region**. Vùng soạn thảo mã lệnh có một số đặc tính giúp cho việc viết, đọc và sửa lỗi mã lệnh dễ dàng hơn như làm nổi bật cú pháp lệnh, tự động thụt đầu dòng theo từng cấu trúc lệnh, tự căn chỉnh câu lệnh theo khung vùng soạn thảo,…

```
1  Max_min := proc ()
2       local i, m, n;
3       if nargs = 0 then
4            return -infinity
5       end if;
6       m := args[1];
7       n := m;
8       for i from 2 to nargs do
9         if m < args[i] then
10           m := args[i]
11        elif args[i] < n then
12           n := args[i] end if
13      end do;
14      return [m, n]
15 end proc:
16
```

Sau khi nhập mã lệnh vào vùng soạn thảo, để thực hiện các mã lệnh đó ta nhấp chuột phải vào vùng soạn thảo và chọn **Execute Code** (hoặc **Edit\Execute Code**).

Ngoài ra, ta có thể thu gọn vùng soạn thảo mã lệnh trong giao diện chuẩn bằng cách nhấp chuột phải vào vùng soạn thảo và chọn **Collapse Code Edit Region**. Lúc này vùng soạn thảo sẽ thu gọn lại thành một biểu tượng có tiêu đề chính là dòng mã lệnh đầu tiên.

## 4.4. Lập trình với tập tin (file)

### 4.4.1. Định dạng đầu ra

Khi muốn in dữ liệu ra màn hình theo một định dạng nào đó ta sử dụng hàm ***printf***. Hàm này giống như lệnh ***printf*** của ngôn ngữ lập trình **C**. Cú pháp của hàm như sau:

**printf*(<định dạng>,<biểu thức>)***
trong đó, phần ***<định dạng>*** theo cấu trúc sau:
*%[flags][width][.precision][modifiers]code*

- tùy chọn ***flags*** có thể là dấu +, -, khoảng trắng hoặc 0
  - dấu + dùng cho các số dương hoặc âm muốn có dấu + hoặc – đứng trước
  - dấu – khi muốn dữ liệu được in ra canh lề bên phải
  - khoảng trắng khi muốn số in ra có dấu – đứng trước nếu là số âm và không có dấu nếu số dương
  - số 0 nếu muốn số được in ra có thêm các số 0 được thêm vào phía trước
- tùy chọn ***width*** chỉ độ rộng (số ký tự) của dữ liệu được in ra.
- tùy chọn ***precision*** chỉ độ chính xác của số được in ra hay số chữ số sau dấu chấm thập phân.
- tùy chọn ***modifiers*** chỉ kiểu của giá trị số được in gồm: **l** (long int), **L** (long long), **zc** hoặc **Z** (dùng cho số phức).
- ***code*** chỉ kiểu dữ liệu được in ra gồm:
  - các định dạng số nguyên: **d** (số nguyên), **o** (cơ số 8), **x** hoặc **X** (cơ số 16).
  - các định dạng số thực: **e** hoặc **E** (định dạng theo kiểu khoa học), **f** (có dấu chấm thập phân).
  - định dạng ký tự: **c**, định dạng chuỗi ký tự: **s.**
  - định dạng các biểu thức của Maple: **a** hoặc **A.**

***Ví dụ***:

$x := 2^{12}$

$$x := 4096$$

*printf(%d, x)*
4096
*printf("x=%d\n",x)* *#sử dụng \n để xuống dòng*
$x = 4096$
*printf("x=%10d\n",x)*
$x =$ 4096
*printf("x=%010d\n",x)*
$x = 0000004096$

$y := \frac{1}{x}$

$$y := \frac{1}{4096}$$

```
evalf(y)
                                0.0002441406250
printf("y=%e\n",y)
y=2.441406e-04
printf("y=%.5f\n", y)
y=0.00024
printf("Tich phan: %a.\n", int(1/t, t))
Tich phan: ln(t).
```

### 4.4.2. Đầu vào tương tác

Trong một số chương trình, đôi khi ta muốn dữ liệu đầu vào được nhập từ người sử dụng ngay tại vị trí dấu con trỏ. Maple đã xây dựng 2 hàm có thể thực hiện việc nhập dữ liệu một cách tương tác như trên là **readline** và **readstat**.

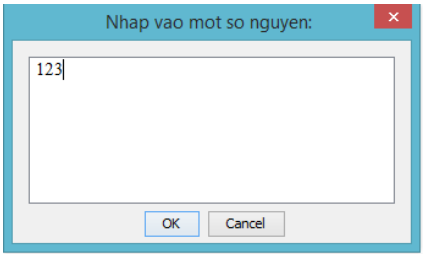
Hàm **readline(terminal)** cho phép chương trình nhận vào một chuỗi ký tự từ người sử dụng. Ví dụ sau mô tả một chương trình cần người dùng chọn thao tác làm việc bằng cách nhập vào số 1 hoặc 2.

```
luachon := proc ()
    local p;
    printf("Chon mot trong hai thao tac:\n");
    printf("1. Tinh dao ham\n");
    printf("2. Tinh tich phan\n");
    printf("Nhap vao 1 hoac 2: ");
    p := readline(terminal);
    if p = "1" or p = "2" then
        return p;
    else
        error "Nhap sai! Chi nhap 1 hoac 2."
    end if
end proc:
luachon()
Chon mot trong hai thao tac:
1. Tinh dao ham
2. Tinh tich phan
Nhap vao 1 hoac 2:
                                "1"
whattype(%)
                                string
```

Ở ví dụ trên, ta thấy rằng chương trình nhận dữ liệu vào dưới dạng chuỗi ký tự "1" chứ không phải dạng số. Muốn chương trình nhận dữ liệu vào dưới dạng số hoặc biểu thức của Maple ta phải dùng hàm **readstat**.

```
nhapso := proc ()
    local p;
    p := readstat("Nhap vao mot so nguyen: ");
    if type(p, integer) then
        return p
    else
        error "Nhap sai! Phai nhap vao mot so nguyen."
    end if
end proc:

nhapso()
#Khi thực hiện hàm nhapso() thì một cửa sổ nhập xuất hiện và ta nhập dữ liệu vào
```




123

### 4.4.3. Mở, đóng một tập tin

Có hai loại tập tin cơ bản là tập tin văn bản và tập tin nhị phân, thư viện I/O (Input/Output) của Maple chứa các lệnh có thể làm việc được với cả hai loại tập tin này. Để đọc hoặc ghi dữ liệu vào một tập tin thì trước hết ta phải mở tập tin đó, cú pháp mở một tập tin như sau:

**fopen(*<filename>*, *<mode>*)**

hoặc **fopen(*<filename>*, *<mode>*, *<type>*)**

trong đó: ***<filename>*** là tên tập tin được mở (bao gồm cả đường dẫn);
***<mode>*** là thao tác thực hiện trên tập tin gồm: **READ, WRITE, APPEND**;
***<type>*** là loại tập tin được mở là **TEXT** hay **BINARY**.

***Chú ý***:

- Nếu mở một tập tin để đọc dữ liệu mà tập tin chưa tồn tại thì hàm **fopen()** sẽ báo lỗi. Nếu mở một tập tin để ghi dữ liệu mà tập tin chưa tồn tại thì Maple sẽ tạo ra một tập tin mới, trong trường hợp tập tin đã tồn tại thì Maple sẽ xóa hết dữ liệu cũ. Hàm **fopen()** cũng sẽ báo lỗi nếu tập tin đang định mở đã mở từ trước mà chưa được đóng lại.
- Khi mở tập tin phải chỉ rõ đường dẫn nơi chứa tập tin, nếu tập tin được mở ở cùng thư mục với tập tin Maple đang chạy thì không cần chỉ đường dẫn.

Sau khi đã hoàn thành thao tác với tập tin ta nên đóng tập tin lại. Để đóng một tập tin đang mở ta dùng hàm: **fclose(*<file>*)** với ***<file>*** có thể là tên tập tin, tên biến đại diện cho tập tin hoặc số thứ tự của tập tin khi mở.

```
f := fopen("Vidu.txt", WRITE, TEXT)
                                    0
fprintf(f, "Ghi du lieu vao file")
                                   20
fclose(f)
```

### 4.4.4. Ghi dữ liệu vào tập tin

#### a. Ghi dữ liệu có định dạng

***Cú pháp:*** **fprintf(*<file>, <định dạng>, <dữ liệu>*)**

Hàm này thực hiện việc ghi dữ liệu vào tập tin đang mở theo định dạng của loại dữ liệu. Định dạng của các loại dữ liệu giống như trong hàm **printf** đã nói ở trên.

```
x := 23
                                   23
y := -1/x
                                  1/23
f := fopen("Vidu.txt", WRITE, TEXT)
                                    0
fprintf(f, "m = %d,  n= %a", x, y)
                                   16
fclose(f)
```

#### b. Ghi dữ liệu số

***Cú pháp:*** **writedata(*<file>, <dữ liệu>*)**

hoặc **writedata(*<file>, <dữ liệu>, <định dạng>*)**

trong đó: ***<định dạng>*** có thể là **integer**, **float** hoặc **string**.

Hàm này cho phép ghi các dữ liệu kiểu số từ ma trận, vectơ, hoặc danh sách (list) vào tập tin văn bản (TEXT). Nếu **<*file*>=terminal** thì dữ liệu sẽ được viết ra giao diện.

```
A := matrix([[1.5, 2.2, 3.4], [2.7, 3.4, 5.6], [1.8, 3.1, 6.7]])
                                [1.5  2.2  3.4]
                                [2.7  3.4  5.6]
                                [1.8  3.1  6.7]
writedata(terminal, A, float)
1.5     2.2     3.4
2.7     3.4     5.6
1.8     3.1     6.7
writedata(terminal, A, integer)
1       2       3
2       3       5
1       3       6
fh := fopen("Vidu.txt", WRITE, TEXT)
```

```
                                    fh:=1
writedata(fh, A, integer)
fclose(fh)
```

**c. Ghi dữ liệu theo từng dòng**

***Cú pháp:*** **writeline(<*file*>, <*dữ liệu*>)**

Hàm này cho phép ghi các dữ liệu như chuỗi ký tự hoặc các biểu thức Maple vào tập tin, mỗi dữ liệu được ghi trên một dòng. Chú ý rằng nếu tập tin đang được mở thì hàm này sẽ thực hiện việc mở lại tập tin đó dưới chế độ WRITE và định dạng TEXT. Nếu **<*file*>=terminal** thì dữ liệu sẽ được viết ra giao diện.

```
f := fopen("Vidu.txt", WRITE, TEXT)
                                    f := 3
writeline(f, "Dong thu nhat", "Dong thu hai")
                                    27
fclose(f)
writeline(terminal, "Dong thu nhat", "Dong thu hai");
Dong thu nhat
Dong thu hai
                                    27
```

### 4.4.5. Đọc dữ liệu từ tập tin

**a. Đọc dữ liệu theo định dạng**

***Cú pháp:*** **fscanf(<*file*>, <*định dạng*>)**

Hàm này cho phép đọc dữ liệu vào tập tin theo định dạng. Định dạng của các loại dữ liệu giống như trong hàm **printf.** Lưu ý rằng kết quả trả về của hàm là một danh sách các dữ liệu đọc được. Không giống như hàm **printf** hay **fprintf**, định dạng "***%s***" trong hàm này chỉ đọc được một chuỗi ký tự không có khoảng trắng.

***Ví dụ***: Giả sử có tập tin "**Data.txt**" gồm 2 dòng:

- Dòng thứ nhất: 3 2/5
- Dòng thứ hai: "toan hoc"

```
f := fopen("Data.txt", READ, TEXT):
fscanf(f, "%d")
                                    [3]
y:=op(fscanf(f, "%a"))
                                    y := 2/5
fscanf(f, "%s")
                                    ["toan"]
fclose(f)
```

**b. Đọc dữ liệu số theo từng cột**

***Cú pháp:*** **readdata(*<file>*, *n*)**

hoặc **readdata(*<file>*, *<định dạng>*, *n*)**

trong đó: ***<định dạng>*** có thể là **integer**, **float** hoặc **string**.

Hàm này cho phép đọc dữ liệu số từ một tập tin văn bản (TEXT) trong Maple. Dữ liệu trong tập tin ***<file>*** phải là số nguyên hoặc số thực được sắp xếp theo từng cột và cách nhau bởi khoảng trắng. Nếu chỉ đọc một cột dữ liệu thì kết quả trả về của hàm là một danh sách các số trong cột. Nếu đọc nhiều cột dữ liệu thì hàm trả về một danh sách mà mỗi phần tử của nó là một danh sách các số nằm trên một hàng tương ứng với các cột được đọc. Nếu sử dụng lệnh mà không có định dạng dữ liệu thì Maple mặc định là đọc dữ liệu dưới dạng số thực.

Muốn đọc dữ liệu số từ tập tin và lưu nó dưới dạng ma trận hoặc vectơ thì có thể tham khảo các hàm **ImportMatrix** và **ImportVector**.

***Ví dụ***: Giả sử có tập tin **"Matran.txt"** gồm 3 dòng chứa 3 cột số như sau:

$$\begin{matrix} 2 & 3 & 5 \\ 3 & -2 & 1 \\ -1 & 4 & 7 \end{matrix}$$

```
readdata("Matran.txt", 1)
                        [2., 3., -1.]
readdata("Matran.txt", integer, 3)
                    [[2, 3, 5], [3, -2, 1], [-1, 4, 7]]
readdata("Matran.txt", integer)
                          [2, 3, -1]
readdata("Matran.txt", [integer, integer, float])
                    [[2, 3, 5.], [3, -2, 1.], [-1, 4, 7.]]
```

**c. Đọc dữ liệu theo từng dòng**

***Cú pháp:*** **readline(*<file>*)**

Hàm này cho phép đọc từng dòng dữ liệu trong tập tin ***<file>***. Kết quả trả về của hàm là chuỗi các ký tự nằm trên một dòng của tập tin, đồng thời con trỏ định vị trong tập tin sẽ dịch chuyển sang dòng mới. Nếu không có dòng nào được đọc thì hàm sẽ trả về giá trị 0. Trường hợp tập tin chưa được mở thì hàm này sẽ mở tập tin ở chế độ READ dưới dạng TEXT.

***Ví dụ***: Giả sử tập tin "Vidu.txt" gồm các dòng có nội dung sau:

- Dòng thứ nhất: "Cho ma tran A vuong cap 2:"
- Dòng thứ hai: 1 3

- Dòng thứ ba: 5 7
- Dòng thứ tư: “Dinh thuc cua ma tran la -8.”

Các lệnh sau sẽ lấy được hàng đầu tiên và hàng cuối cùng của tập tin.

```
first := readline("Vidu.txt")
                                    first := "Cho ma tran A vuong cap 2:"
line := readline("Vidu.txt")
                                                line := "1  3"
while line <> 0 do
    last := line;
    line := readline("Vidu.txt")
end do:
last
                                        "Dinh thuc cua ma tran la -8."
```

# Chương 5

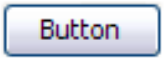
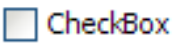
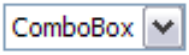
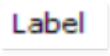
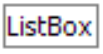
# CÁC THÀNH PHẦN ĐỒ HỌA VÀ MAPLET

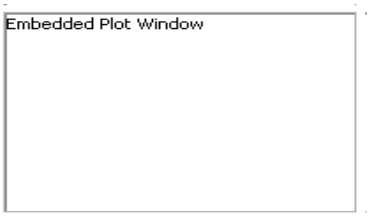
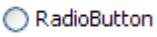
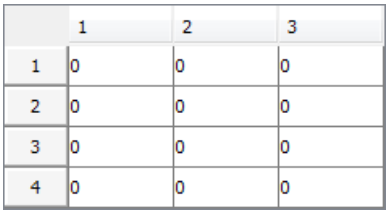
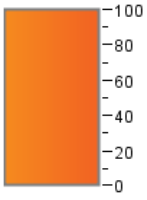
## 5.1. Giới thiệu các thành phần đồ họa

Maple cho phép nhúng các thành phần đồ họa đơn giản vào trong giao diện chuẩn. Các thành phần này có thể được thiết lập để tiến hành các hoạt động đã được lập trình, nó cho phép người sử dụng làm việc với Maple mà không cần hiểu rõ cú pháp lệnh.

### 5.1.1. Mô tả một số thành phần đồ họa

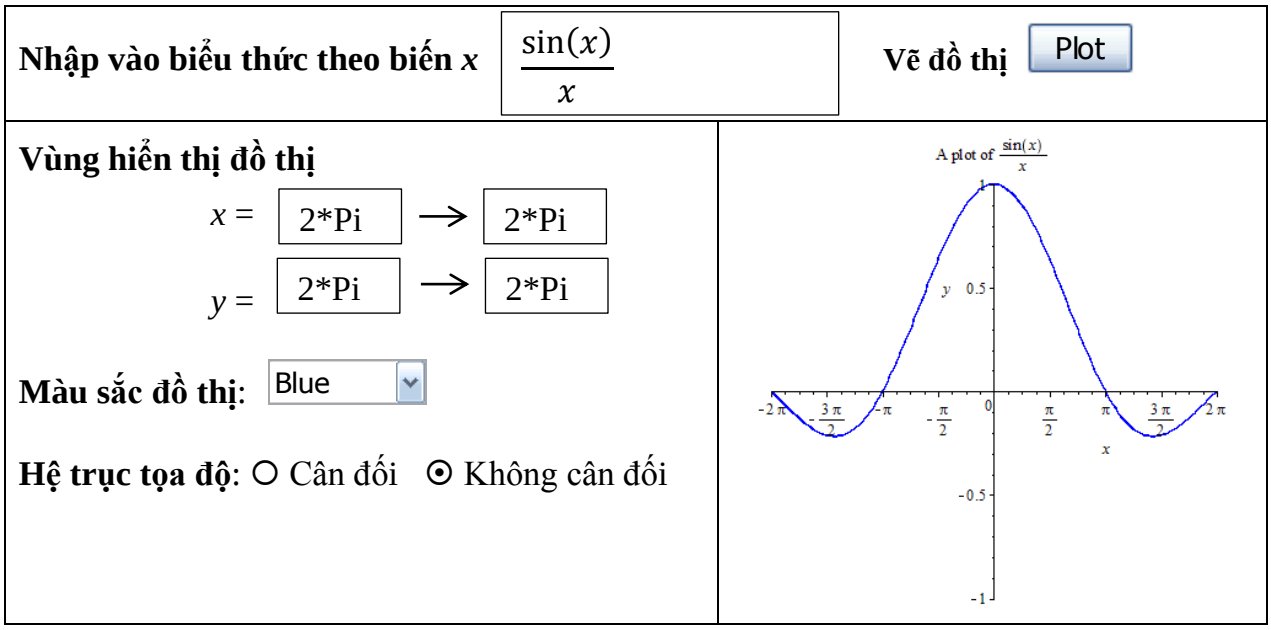
Bảng sau mô tả một số các thành phần đồ họa thường sử dụng:

| Tên và mô tả | Hình ảnh |
|---|---|
| **Button:** nút lệnh, dùng để thực hiện lệnh. |   |
| **Check Box**: dùng để chọn hoặc bỏ chọn. Thay đổi nhãn và nhập lệnh thi hành khi giá trị thay đổi. |   |
| **Combo Box**: hộp chọn, cho phép chọn một trong nhiều lựa chọn có trong hộp. Thay đổi danh sách thành phần và nhập lệnh thi hành khi giá trị thay đổi. |   |
| **Dial**: chọn hoặc hiển thị một giá trị nguyên hoặc thực. Thay đổi cách hiển thị và nhập lệnh thi hành khi giá trị thay đổi. |   |
| **Label**: dùng để hiển thị văn bản làm nhãn cho các thành phần khác. Giá trị có thể được cập nhật phụ thuộc vào mã lệnh hoặc các thành phần đồ họa khác. |   |
| **List Box**: hiển thị danh sách các thành phần. Thay đổi danh sách thành phần và nhập lệnh thi hành khi một hành phần được chọn. |   |
| **Mathematical Expression**: nhập hoặc hiển thị một biểu thức toán học. Giá trị có thể được cập nhật phụ thuộc vào mã lệnh hoặc các thành phần đồ họa khác. |  |

| | |
|---|---|
| **Meter**: lựa chọn hoặc hiển thị một giá trị nguyên hoặc thực. Thay đổi hiển thị và nhập lệnh thi hành khi giá trị thay đổi. |  |
| **Plot**: vùng hiển thị đồ thị và vận động của đồ thị. Giá trị có thể được cập nhật phụ thuộc vào mã lệnh hoặc một thành phần đồ họa khác. |  |
| **Rotary Gauge**: chọn hoặc hiển thị một giá trị nguyên hoặc thực. Thay đổi hiển thị và nhập lệnh thi hành khi giá trị thay đổi. |  |
| **Radio Button**: nút chọn tròn, sử dụng cùng với các nút chọn khác để lựa chọn. Thi hành mã lệnh khi giá trị thay đổi. |  |
| **Slider**: chọn hoặc hiển thị một giá trị nguyên hoặc thực. Thay đổi hiển thị và nhập lệnh thi hành khi giá trị thay đổi. |  |
| **Data Table**: bảng dữ liệu dùng để nhập hoặc liên kết với các dạng dữ liệu như ma trận, vectơ hoặc mảng trong giao diện. |  |
| **Text Area**: vùng để hiển thị văn bản hoặc nhập văn bản. Giá trị có thể được cập nhật phụ thuộc vào mã lệnh hoặc một thành phần đồ họa khác. |  |
| **Toggle Button:** dùng để chọn hoặc hiển thị một trong hai tùy chọn. Thanh đổi hình ảnh hiển thị, nhập mã lệnh thi hành khi giá trị thay đổi. |  |
| **Volume Gauge**: chọn hoặc hiển thị một giá trị nguyên hoặc thực. Thay đổi hiển thị và nhập lệnh thi hành khi giá trị thay đổi. |  |

### 5.1.2. Minh họa việc sử dụng kết hợp các thành phần đồ họa

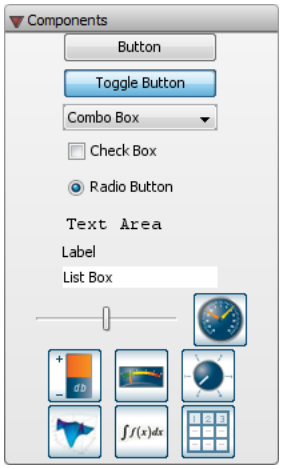
Sau đây ta tạo một chương trình nhỏ trong giao diện chuẩn cho phép người dùng nhập vào một hàm số theo biến *x* và hiển thị đồ thị của hàm số cùng với các tùy chọn như vùng hiển thị, màu sắc đồ thị và sự cân đối của hệ trục tọa độ. Giao diện này minh họa sự kết hợp của một số thành phần đồ họa như: Label, Text Area, Combo Box, Button, Radio Button, Math Expression và Plot.




## 5.2. Lập trình với các thành phần đồ họa

### 5.2.1. Thêm các thành phần đồ họa vào giao diện chuẩn

Các thành phần đồ họa có thể được thêm vào giao diện chuẩn bằng cách sử dụng bảng **Components** ở phía trái của giao diện hoặc có thể dùng cách sao chép các thành phần đã có từ một vùng soạn thảo này đến vùng khác.

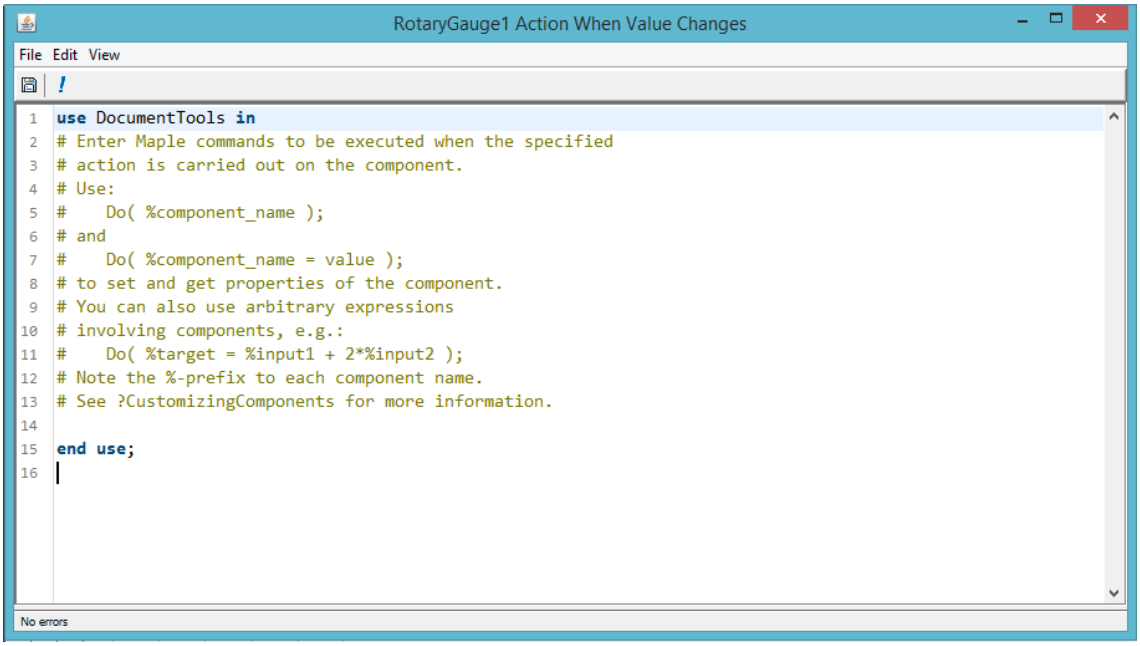


Ta có thể quản lý, sắp xếp các thành phần đồ họa được thêm vào bằng cách đưa các thành phần đó vào trong một bảng (như ví dụ trên). Để tạo một bảng trong giao diện chuẩn của Maple ta sử dụng menu **Insert\Table**.

### 5.2.2. Thiết lập thuộc tính cho các thành phần đồ họa

Các thành phần đồ họa có thể được lập trình để thực hiện các thao tác đặc trưng của chúng. Ví dụ các nút lệnh (button) và thanh trượt (slider) có thể được lập trình để hiển thị thông tin khi chúng được nhấn hay kéo, vùng hiển thị đồ thị (plot) có thể được lập trình để hiển thị các điểm,… Để một thành phần đồ họa thực hiện đúng nhiệm vụ đặt ra ta phải chỉnh sửa các thuộc tính của nó và trong nhiều trường hợp phải cung cấp thêm các mã lệnh cần thiết cho nhiệm vụ đó. Thao tác chung khi chỉnh sửa thuộc tính của các thành phần đồ họa là:

- Kích chuột phải vào thành phần để hiển thị menu ngữ cảnh.
- Chọn **Component Properties…** để hiển thị cửa sổ thuộc tính của thành phần đó.
- Nhập vào các giá trị và các thuộc tính cần thiết vào các trường.
- Để nhập các mã lệnh điều khiển việc thực hiện nhiệm vụ của thành phần đồ họa ta chọn **Edit Value Change Action**… (hoặc **Edit Cliked Action**…) trong menu ngữ cảnh. Cửa sổ sau xuất hiện cho phép ta nhập các mã lệnh vào, các lệnh này sẽ được thực hiện khi sự kiện liên quan đến thành phần này xảy ra.

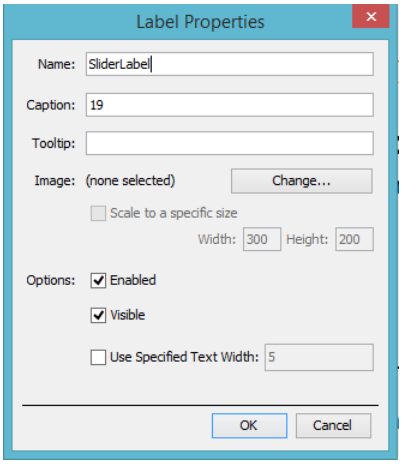


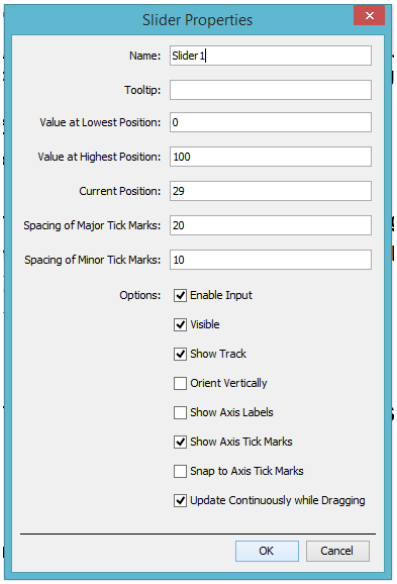

***Ví dụ 1:*** Hiển thị giá trị của Slider

Ví dụ này hướng dẫn tạo một thanh trượt (Slider) và một nhãn (Label) vào giao diện chuẩn, sau đó hiển thị giá trị của thanh trượt khi nó bị thay đổi bởi thao tác kéo chuột.

- Đặt con trỏ ở vị trí muốn thêm Slider vào.
- Trong bảng **Components**, chọn Slider.
- Chọn tiếp Label trong bảng **Components** để thêm một nhãn vào kế bên Slider.


- Kích chuột phải vào Label chọn *Component Properties*. Hộp thoại *Label Properties* xuất hiện. Nhập tên *SliderLabel* vào ô Name và nhấn OK.
- Kích chuột phải vào Slider và chọn *Component Properties*. Nhập tên *Slider1* vào ô Name.




- Nhập giá trị vào ô *Value at Lowest Position* là **0** và ô *Value at Highest Position* là **100,** để thiết lập giá trị thấp nhất và giá trị cao nhất của Slider.
- Nhập vào ô *Major Tick Marks* là **20** and *Minor Tick Marks* là **10** để thiết lập giá trị các vạch lớn, nhỏ trong Slider.
- Đánh dấu chọn vào ô *Update Continuously while Dragging***.** Nhấn OK để đóng hộp thoại.
- Tiếp tục chọn *Edit Value Changed Action* trong menu ngữ cảnh của Slider. Cửa sổ nhập mã lệnh xuất hiện với câu lệnh tổng quát là "**use** DocumentTools **in…end use"**. Câu lệnh này cho phép ta sử dụng trực tiếp các lệnh mà không cần phải gọi gói lệnh chứa nó.
- Nhập vào dòng lệnh sau (trước dòng **end use)**:

*Do(%SliderLabel(caption)=%Slider1(value));*

Sử dụng chuột để di chuyển thanh trượt và giá trị trượt đến được chỉ ra trong **Label**.

***Ví dụ 2:*** Tạo giao diện vẽ đồ thị hàm số $y = ax + b$.

Trong ví dụ này, ta tạo ra hai tham số $a, b$ bằng cách chọn giá trị trên các **Dial**, sau đó vẽ đồ thị hàm số $y = ax + b$ và tính $\frac{a}{b}$.

**Thiết kế giao diện**:

- Tạo một bảng trong giao diện chuẩn gồm 5 hàng 3 cột bằng menu **Insert\Table**.
- Thêm các thành phần đồ họa vào bảng gồm: hai **Dial** dùng cho việc nhập các tham số a, b, **Rotary gauge** để hiển thị thương $\frac{a}{b}$, **Mathematical expression** để hiển thị hàm số $y = ax + b$ và **Plot** để hiển thị đồ thị hàm số.
- Thêm các tiêu đề và sắp xếp lại các hàng, cột trong bảng để được giao diện sau:

<table>
<tr><td colspan="2">Nhập vào các tham số a, b</td><td rowspan="2">Đồ thị hàm số</td></tr>
<tr><td>Chọn a</td><td>Chọn b</td></tr>
<tr><td>  </td><td>  </td><td rowspan="3">Embedded Plot Window</td></tr>
<tr><td colspan="2">Thương $\frac{a}{b}$</td></tr>
<tr><td colspan="2">  </td></tr>
</table>

**Thiết lập thuộc tính cho các thành phần:**

Mở hộp thoại *Component Properties* của từng thành phần. Lưu ý rằng các thành phần này đều có tên mặc định, ta có thể đặt lại các tên này. Thiết lập thuộc tính của các thành phần lần lượt như sau:

- **Dial0**: thay đổi giá trị ở ô *Value at Highest Position* là 10, ô *Spacing of Major Tick Marks* là 1, và ô *Spacing of Minor Tick Marks* là 1.
- **Dial1**: giữ nguyên các giá trị mặc định.
- **RotaryGauge0**: thay đổi giá trị tại ô *Value at Highest Position* là 40, ô *Spacing of Major Tick Marks* là 5, và ô *Spacing of Minor Tick Marks* là 1.
- **Plot0**: giữ nguyên các giá trị mặc định.

- **MathContainer0**: thay đổi giá trị tại ô *Width in Pixels* là 200, và ô *Height in Pixels* là 45.

**Lập trình hoạt động của các thành phần:**

Các thành phần như Math Container, Rotary Gauge, Plot chỉ hoạt động khi giá trị của các Dial thay đổi. Vì vậy, mã lệnh thực hiện cần thiết lập trong các Dial để khi giá trị các Dial thay đổi thì các thành phần trên cũng thay đổi hiển thị cho phù hợp. Sử dụng các câu lệnh sau:

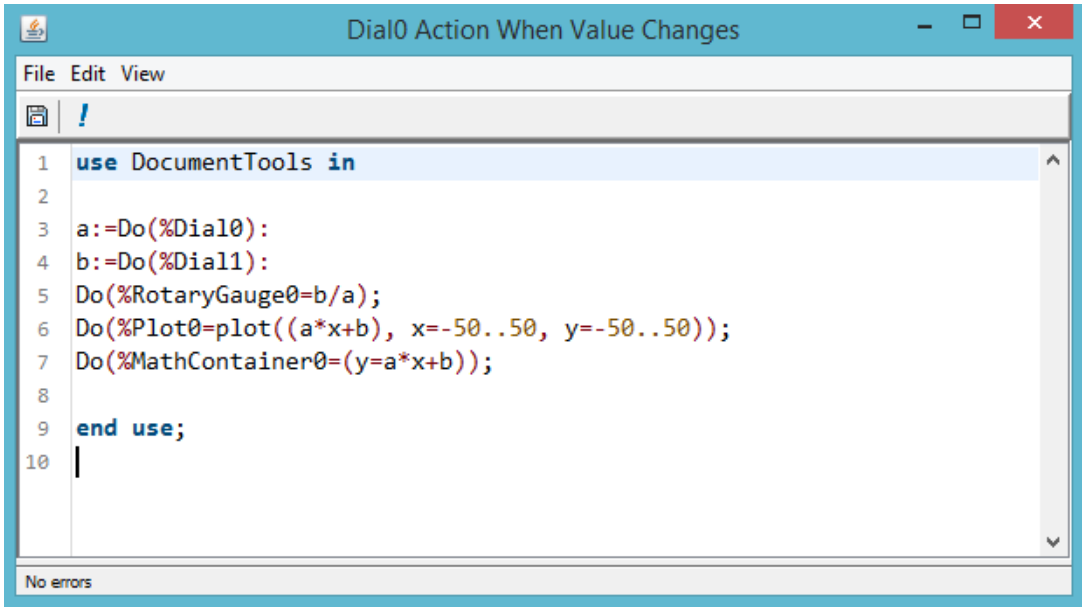
```
a:=Do(%Dial0):
b:=Do(%Dial1):
Do(%RotaryGauge0=a/b);
Do(%Plot0=plot((a*x+b), x=-50..50, y=-50..50));
Do(%MathContainer0=(y=a*x+b));
```

Trước khi đưa lệnh vào các Dial ta phải kiểm tra xem các lệnh có thực hiện đúng theo yêu cầu bài toán không. Để làm điều này, đầu tiên cần chạy gói lệnh DocumentTools:

```
with(DocumentTools);
```

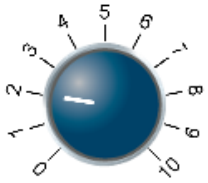
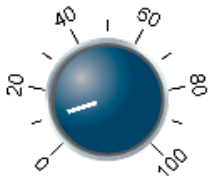
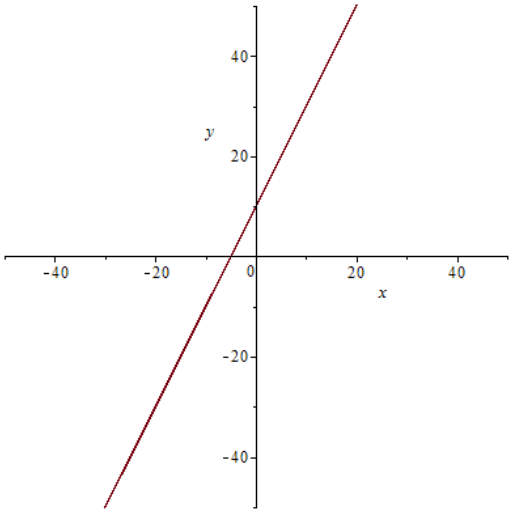
Sau đó lần lượt thực hiện các lệnh và xem kết quả. Nếu lệnh thực hiện chính xác ta lần lượt đưa các lệnh đó vào trong cửa sổ lệnh của Dial0 và Dial1.

- Nhấn chuột phải vào Dial0 chọn *Edit Value Changed Action* và nhập các lệnh trên vào cửa sổ *Action When Value Changes* (trước từ khóa **end use**):




- Thực hiện tương tự thao tác trên đối với Dial1.

Trở lại giao diện đã tạo, dùng chuột di chuyển các Dial để nhập các tham số a, b và kết quả sẽ hiển thị ở các thành phần còn lại.

<table>
<tr><td colspan="2">Nhập vào các tham số a, b</td><td rowspan="2">Đồ thị hàm số<br>$y = 2x + 10$</td></tr>
<tr><td>Chọn a</td><td>Chọn b</td></tr>
<tr><td>  </td><td>  </td><td rowspan="3">  </td></tr>
<tr><td colspan="2">Thương $\frac{a}{b}$</td></tr>
<tr><td colspan="2">  </td></tr>
</table>

## 5.3. Sử dụng các Maplet

Một Maplet là một giao diện ứng dụng cho phép người sử dụng tương tác với bộ máy của Maple thông qua các nút lệnh (button), vùng văn bản (text region), thanh trượt (slider bar) và các thành phần đồ họa khác. Maplet có thể được mở như một ứng dụng bình thường bên ngoài Maple. Với phiên bản này, Maple đã tích hợp sẵn khá nhiều Maplet với các chủ đề thường gặp ở một số lĩnh vực. Để truy cập các Maplet này ta vào menu **Tool\Tutors**. Mỗi người dùng có thể tự tạo các Maplet của riêng mình tùy từng mục đích, mỗi Maplet như thế phải đảm bảo được hai yêu cầu: giới thiệu chi tiết về vấn đề đang làm và giao diện đầy đủ chức năng.

### 5.3.1. Mở một tập tin maplet

Nếu một ứng dụng Maplet được lưu dưới dạng tập tin ***.maplet*** thì ứng dụng này có thể được mở trực tiếp trong Window như một chương trình độc lập bằng cách kích đôi chuột vào biểu tượng tập tin. Lúc này, trình **Maplet Viewer** của Maple sẽ khởi động và hiển thị giao diện của ứng dụng Maplet.

Nếu muốn xem và chỉnh sửa mã lệnh của Maplet được lưu dưới dạng tập tin ***.maplet*** ta thực hiện như sau:

- Mở chương trình Maple.

- Từ menu **File**, chọn **Open**. Hộp thoại **Open** xuất hiện.
- Trong danh sách **Files of Type**, chọn **.maplet.**
- Chỉ đường dẫn đến vị trí có tập tin ***.maplet** cần xem hoặc chỉnh sửa mã lệnh.
- Nhấn **Open**.

### 5.3.2. Mở Maplet trong giao diện chuẩn của Maple

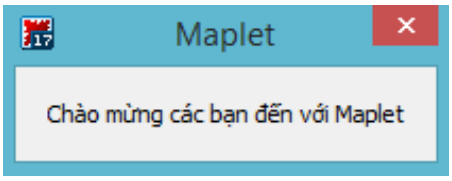
Để mở một ứng dụng Maplet được lập trình bởi các mã lệnh Maple trong giao diện chuẩn ta cần thực hiện lần lượt các mã lệnh Maple đó. Câu lệnh chính để hiển thị Maplet là *Maplets[Display]*. Với một Maplet phức tạp thì số lượng mã lệnh sẽ tương đối lớn. Để những ứng dụng Maplet như vậy chạy chính xác ta cần phải chắc chắn rằng các hàm được định nghĩa theo nó phải được thực hiện hết. Lưu ý rằng khi một ứng dụng Maplet đang chạy thì ta không thể làm việc được với giao diện chuẩn của Maple.

Thủ tục thông thường để mở một Maplet trong giao diện chuẩn bao gồm:

- Chạy các hàm được dùng trong Maplet:
  *Myproc:=proc…*
- Gọi gói lệnh Maplets[Elements]:
  *with(Maplets[Elements])*
- Gọi tên của ứng dụng Maplet:
  *Maplet_name:=Maplet( Maplet_definition );*
- Hiển thị ứng dụng Maplet:
  *Maplets[Display]( Maplet_name );*

***Ví dụ***: Thực hiện lần lượt các câu lệnh sau để mở một Maplet hiển thị dòng văn bản "Chào mừng các bạn đến với Maplet".

```
with(Maplets[Elements]):
MySimpleMaplet:= Maplet([["Chào mừng các bạn đến với Maplet"]]):
Maplets[Display](MySimpleMaplet);
```




## 5.4. Tạo các Maplet

Để tạo các Maplet ta có thể sử dụng giao diện **Maplet Builder** hoặc sử dụng các lệnh có trong gói lệnh **Maplets**. Maplet Builder cho phép ta kéo – thả các thành phần đồ họa như: button, slider, text region… vào Maplet đang tạo, thiết lập thuộc tính các thành phần hoặc

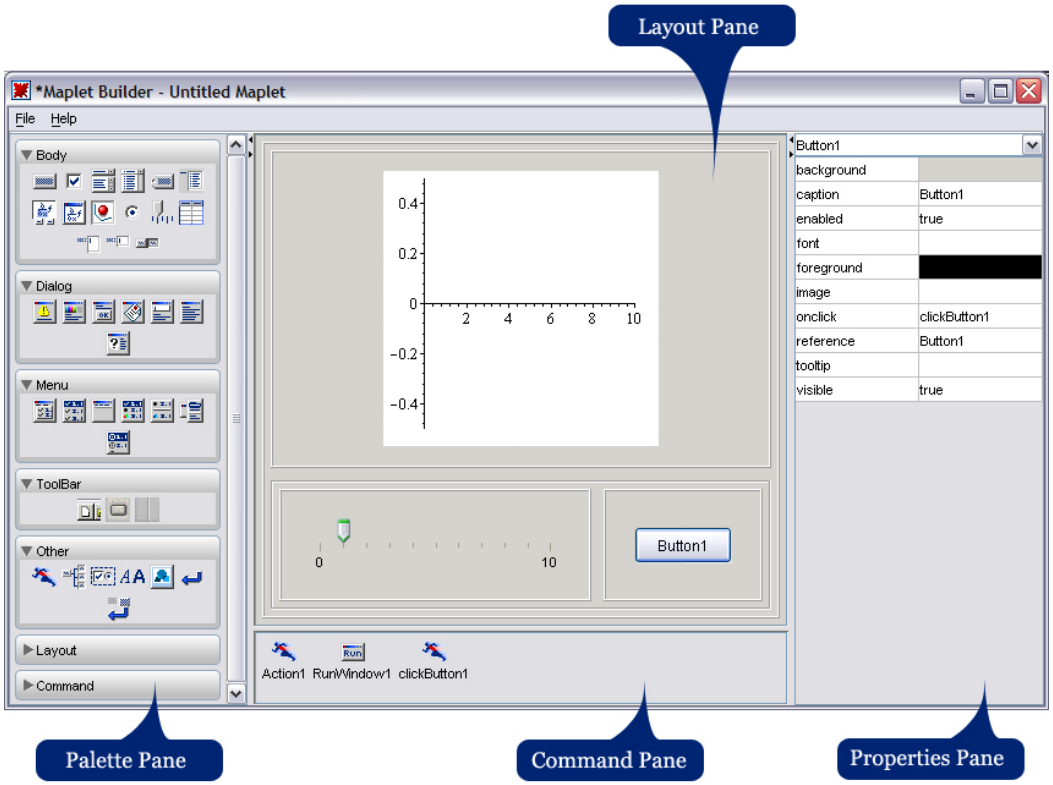
cập nhật hiển thị các thành phần. Maplet Builder được thiết kế để tạo ra các Maplet tương đối đơn giản. Nếu muốn tạo ra các Maplet phức tạp hơn với nhiều thao tác, điều khiển và tùy chọn thì nên sử dụng gói lệnh Maplets.

Thiết kế một ứng dụng Maplet giống như đang xây một ngôi nhà. Khi xây nhà, đầu tiên phải thi công phần cơ bản (nền móng, sàn và các tường xung quanh) rồi sau đó mới thêm vào cửa sổ, cửa chính... Tương tự, khi thiết kế một Maplet, trước hết ta phải xác định khung chính của nó phân thành mấy phần (gồm bao nhiêu hàng, bao nhiêu cột) rồi sau đó mới thêm các thành phần đồ họa vào.

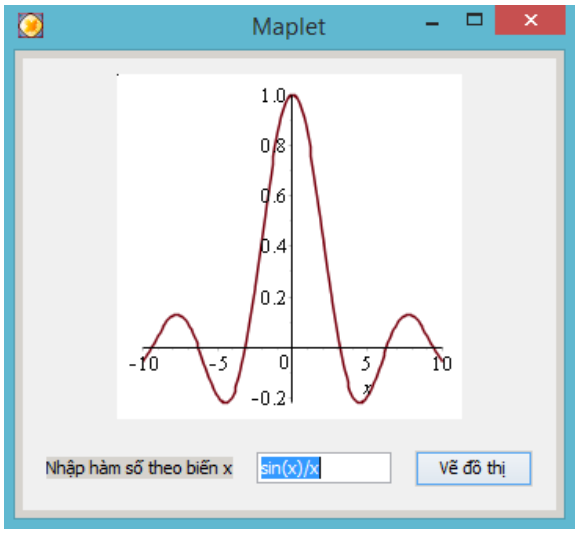
### 5.4.1. Sử dụng Maplet Builder

Để mở giao diện **Maplet Builder** ta vào menu **Tools**, chọn **Assistants\Maplet Builder**. Giao diện **Maplet Builder** xuất hiện gồm 4 vùng chính:

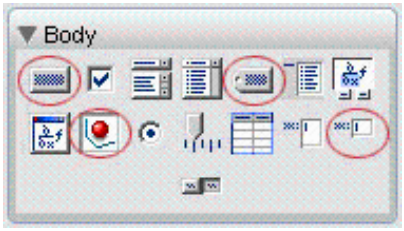
- Vùng **Palette** là nơi hiển thị các bảng được sắp xếp theo chức năng, mỗi bảng chứa các thành phần có thể đưa vào trong Maplet. Trong đó, bảng **Body** chứa hầu hết các thành phần thường được sử dụng.
- Vùng **Layout** là nơi hiển thị các thành phần đồ họa đã thêm vào Maplet.
- Vùng **Command** hiển thị các mã lệnh và các hoạt động được định nghĩa trong Maplet.
- Vùng **Properties** hiển thị các thuộc tính của một thành phần trong Maplet đang được tương tác.



***Ví dụ***: Thiết kế một Maplet sử dụng Maplet Builder
Trong ví dụ này, ta sẽ thiết kế một Maplet cho phép nhập vào một hàm số và hiển thị đồ thị của nó.




Để tạo được Maplet như trên ta sử dụng các thành phần đồ họa sau:

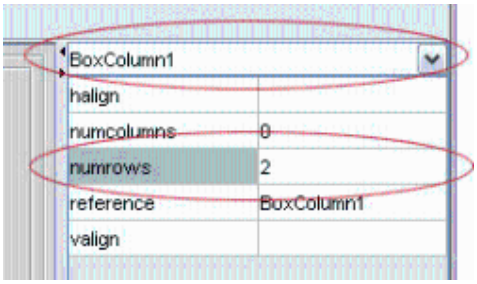
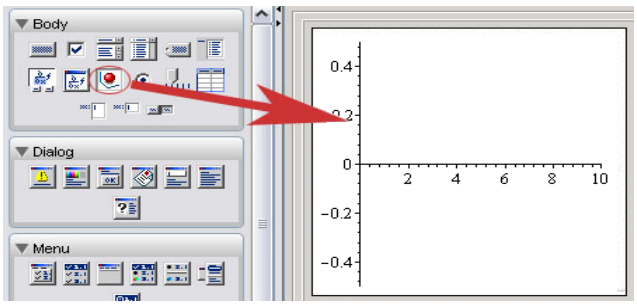



 Button

 Label

 Plotter

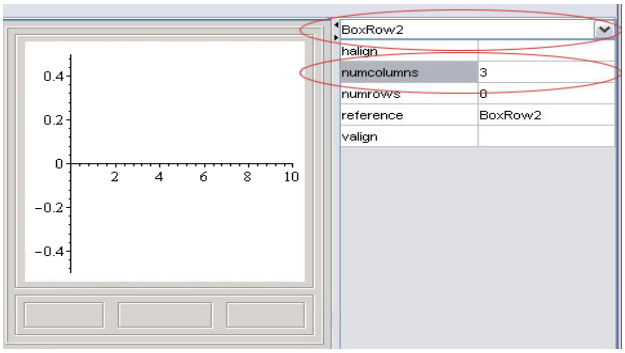
 TextField

| Hoạt động | Kết quả trong Maplet Builder |
|---|---|
| **1. Xác định số hàng và số cột của Maplet.**<br>Ở cửa sổ **Properties**:<br>• Chọn **BoxColumn1** từ danh sách sổ xuống<br>• Thay đổi giá trị ở trường **numrows** là **2**. |  |
| **2. Thêm vùng đồ thị vào Maplet.**<br>Từ bảng **Body**, kéo **Plotter** đến hàng thứ nhất của vùng **Layout**. |  |

**3. Phân hàng thứ hai của Maplet thành 3 cột.**

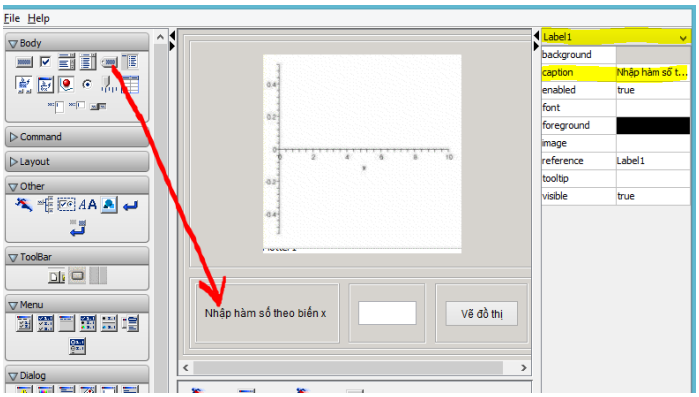
Ở cửa sổ **Properties**:

- Chọn **BoxRow2**.
- Thay đổi giá trị ở trường **numcolumns** là **3**.




**4. Thêm một Label vào Maplet.**

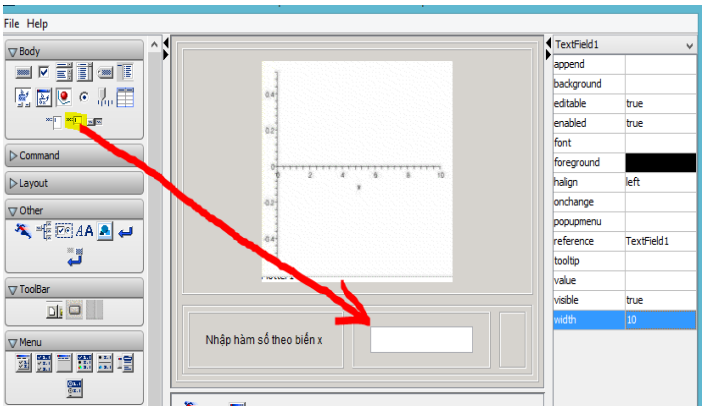
- Từ bảng **Body**, kéo **Label** bỏ vào cột đầu tiên của hàng 2
- Ở cửa sổ Properties:
  - Chọn **Label1**
  - Thay đổi văn bản ở trường **caption** là: "Nhập vào một hàm số theo biến *x*"




**5. Thêm một TextField vào Maplet.**

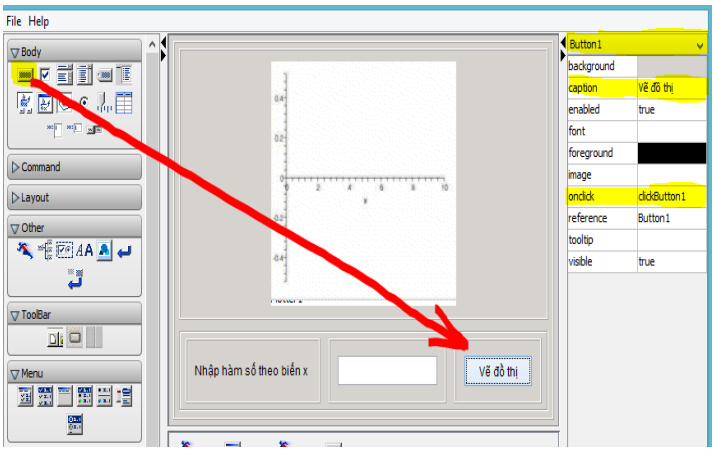
- Từ bảng **Body** kéo **TextField** vào cột giữa của hàng 2.
- Điều chỉnh lại độ rộng của TextField cho phù hợp.

Ta có thể điều chỉnh lại kích thước của Maplet Builder để có thể hiển thị hết toàn bộ vùng **Layout**.



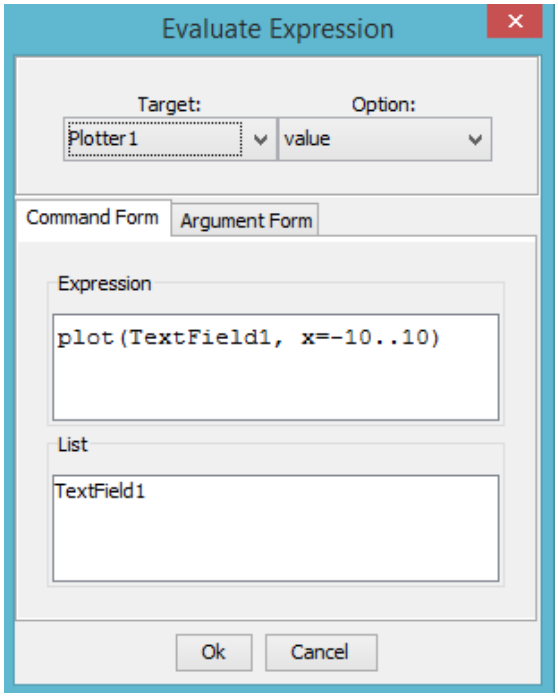

**6. Thêm một nút lệnh vào Maplet.**

- Từ bảng **Body** kéo **Button** vào cột cuối của hàng 2.
- Trong vùng **Properties**:
  - Chọn **Button1**.
  - Thay đổi văn bản ở trường **caption** là "Vẽ đồ thị".
  - Thay đổi ở trường **onclick** là **<Evaluate>**.



Trong cửa sổ **Evaluate Expression**:

- Ở danh sách **Target** chọn **Plotter1**.
- Nhập lệnh sau vào hộp **Expression** ở tab **Command Form** với lưu ý không nhập dấu ; ở cuối lệnh:
  *plot(TextField1, x=-10..10)*
- Nhấn **Ok**.




**7. Chạy Maplet:** từ menu **File** của Maplet Builder chọn **Run**. Hoặc có thể lưu Maplet vừa tạo dưới dạng tập tin ***.maplet*** rồi chạy trực tiếp tập tin này.

Để tìm hiểu thêm về Maplet Builder nên tham khảo ở phần trợ giúp Help. Ngoài ra, Maple cũng xây dựng nhiều ví dụ về các Maplet tạo bởi Maplet Buider ở trong Help. Sử dụng các lệnh sau để đi đến các phần trợ giúp trên:

```
?MapletBuilder
?MapletBuilder/examples
```

### 5.4.2. Gói lệnh Maplets

Khi thiết kế một Maplet phức tạp thì việc sử dụng lệnh trong gói lệnh **Maplets** là một lựa chọn hợp lý vì gói lệnh này cho ta nhiều điều khiển hơn giao diện Maplet Builder. Gói lệnh **Maplets[Elements]** nằm trong gói lệnh **Maples** có đầy đủ các thành phần để tạo nên một ứng dụng Maplet. Sau khi lập trình xong Maplet, sử dụng lệnh **Maplets[Display]** để chạy ứng dụng vừa tạo.

Để biết thêm thông tin về gói lệnh Maplets cũng như các ví dụ về cách tạo các Maplet bằng lệnh nên tham ở phần trợ giúp Help bằng các lệnh sau:

```
?MapletsPackage
?examples/ExampleMaplets
```

***Ví dụ***: Thiết kế một maplet sử dụng gói lệnh Maplets

Ví dụ này minh họa việc thiết kế Maplet vẽ đồ thị ở ví dụ trước bằng lệnh để thấy được sự tương quan giữa hai phương thức tạo Maplet.

```
with(Maplets[Elements]):
# Đặt tên và thiết kế Maplet
Dothi:=Maplet(
  BoxLayout(
        BoxColumn(
          # Hàng đầu
           BoxRow(
                # Xác định vùng đồ thị
                Plotter('reference' = Plotter1)
          # Kết thúc hàng đầu
             ),
          # Hàng thứ 2
           BoxRow(
                # Định nghĩa Label
                Label("Nhập hàm số theo biến x"),
                 # Định nghĩa Text Field
                TextField('reference' = TextField1),
                # Định nghĩa nút lệnh
                Button(caption="Vẽ đồ thị", Evaluate(value = 'plot (TextField1,
                                                        x = -10..10)', 'target' = Plotter1))
          # Kết thúc hàng thứ 2
             )
        # Đóng BoxColumn
         )
 # Đóng BoxLayout
   )
# Đóng Maplet
):
# Chạy Maplet vừa tạo
Maplets[Display](Dothi);
```

### 5.4.3. Một số lệnh thường dùng trong lập trình Maplet

#### a. CloseWindow

Một ứng dụng Maplet có thể có nhiều cửa sổ thành phần. Mỗi cửa sổ như thế có thể chứa các thành phần đồ họa. Để đóng một cửa sổ ta sử dụng lệnh *CloseWindow*. Xét ví dụ sau để hiểu rõ hơn về lệnh:

```
with(Maplets[Elements]):
mapletvd1 := Maplet(‘onstartup’ = ‘A1’,
        Window[‘W1’](“Cửa sổ thứ nhất”,
                [Button(“Mở cửa sổ mới”, RunWindow(‘W2’)),
                Button(“Thoát”, Shutdown())]),
        Window[‘W2’](“Cửa sổ thứ hai”,
                [Button(“Đóng cửa sổ”, CloseWindow(‘W2’)),
                Button(“Thoát”, Shutdown())]),
        Action[‘A1’](RunWindow(‘W1’))):
Maplets[Display](mapletvd1)
```

**b. Evaluate**

Sử dụng lệnh này để thực hiện một hàm hoặc một chu trình đã có và hiển thị kết quả trong thành phần nào đó của Maplet. Ví dụ:

```
with(Maplets[Elements]):
mapletvd2 := Maplet([["Nhập biểu thức", TextField['TF1']('width' = 30)],
                ["Đạo hàm theo biến x:", Button("Đạo hàm",
                Evaluate('TF1' = 'diff(TF1, x)')), Button("Thoát", Shutdown(['TF1']))]]):
Maplets[Display](mapletvd2)
```

**c. RunDialog**

Khi lập trình các Maplet, ta có thể thao tác với một số hộp thoại như: AlertDialog (hộp thoại cảnh báo), ColorDialog (hộp thoại chọn màu), ConfirmDialog (hộp thoại xác nhận), FileDialog (hộp thoại chọn tập tin), InputDialog (hộp thoại nhập dữ liệu), MessageDialog (hộp thoại thông báo), QuestionDialog (hộp thoại truy vấn). Để gọi hộp thoại nào trong số các hộp thoại trên ta sử dụng lệnh *RunDialog*. Ví dụ:

```
with(Maplets[Elements]):
mapletvd3 := Maplet(Window([[TextField['TF1']()],
                [Button("Đạo hàm theo biến x", Evaluate('TF1' = 'diff(TF1, x)')),
                Button("Trợ giúp", RunDialog('MD1')), Button("Thoát", Shutdown(['TF1']))]]),
                MessageDialog['MD1']("Sử dụng cú pháp \"?diff\" để thấy các hướng dẫn cho
                                                                lệnh đạo hàm diff")):
Maplets[Display](mapletvd3)
```

**d. RunWindow**

Một ứng dụng Maplet phức tạp thường chứa nhiều cửa sổ, mỗi cửa sổ gồm các thành phần đồ họa được lập trình để thực hiện một chức năng nào đó. Cửa sổ ban đầu khi vào

ứng dụng Maplet thường được gọi là cửa sổ chính, các cửa sổ còn lại được gọi là cửa sổ con. Để hiển thị một cửa sổ nào đó ta sử dụng lệnh *RunWindow*. Ví dụ:

```
with(Maplets[Elements]):
mapletvd4:=Maplet('onstartup' = 'A1',
        Window['W1']('title' = "Đạo hàm và tích phân", 'layout' = 'BL0'),
        BoxLayout['BL0'](BoxColumn(BoxRow("Chọn một thao tác"),
        BoxRow(Button("Đạo hàm", Action(CloseWindow('W1'),RunWindow('W2'))),
        Button("Tích phân", Action(CloseWindow('W1'), RunWindow('W3'))),
        Button("Thoát", Shutdown()))),
        Window['W2']('title' = "Đạo hàm", [["Nhập biểu thức:", TextField['TF1']()],
        [Button("Đạo hàm theo biến x", Evaluate('TF1' = 'diff(TF1, x)')),
                                        Button("Quay lui", RunWindow('W1'))]]),
        Window['W3']('title' = "Tích phân", [["Nhập hàm số:", TextField['TF2']()],
        [Button("Nguyên hàm theo biến x", Evaluate('TF2' = 'int(TF2, x)')),
                                        Button("Quay lui", RunWindow('W1'))]]),
        Action['A1'](RunWindow('W1'))):
Maplets[Display](mapletvd4)
```

**e. SetOption**

Mỗi thành phần trong ứng dụng Maplet đều có một số thuộc tính riêng, có thể tham khảo trong phần *Maplets\ElementOptions* ở trang trợ giúp Help để tìm hiểu thuộc tính của từng thành phần. Để thiết lập giá trị thuộc tính cho một thành phần trong Maplet ta sử dụng lệnh *SetOption*. Ví dụ:

```
with(Maplets[Elements]):
mapletvd5 := Maplet([[Label('caption' = "Nhập một biểu thức")],
        ["Input:", TextField['TF1'](20)],
        [Button("Đổi font chữ", SetOption(('TF1')('font') = 'F1')),
        Button("Đổi màu nền", SetOption(('TF1')('background') = 'red')),
        Button("Thoát", Shutdown(['TF1']))]], Font[F1]("courier", size = 14)):
Maplets[Display](mapletvd5)
```

**f. Shutdown**

Khi muốn đóng một ứng dụng Maplet ta sử dụng lệnh *Shutdown*. Với lệnh này ta có thể thêm tùy chọn giá trị trả về khi thoát khỏi ứng dụng. Ở các ví dụ trên, các nút lệnh "Thoát" đều sử dụng lệnh *Shutdown*.

### 5.4.4. Lưu Maplet

Ta có thể lưu mã lệnh Maplet trong tập tin Maple (***.mw**) như mã lệnh lập trình thông thường**.** Ngoài ra, nếu tập tin Maple chỉ chứa mã lệnh của một ứng dụng Maplet thì có thể xuất ra chương trình dưới dạng tập tin ***.maplet**. Các bước thực hiện như sau:

- Từ menu **File**, chọn **Export As**.
- Trong danh sách **Files of Type**, chọn **Maplet(.maplet)**.
- Chỉ đường dẫn đến thư mục chứa ứng dụng.
- Nhập tên ứng dụng cần lưu.
- Nhấn **Save**.

# TÀI LIỆU THAM KHẢO

[1] Maplesoft, *Maple User Manual*, Waterloo Maple Inc, 1996 – 2009.

[2] Frank Garvan, *The Maple Book*, Chapman & Hall/CRC, 2002

[3] L. Bernardin, P. Chin, P. DeMarco, K. O. Geddes, D. E. G. Hare, K. M. Heal, G. Labahn, J. P. May, J. McCarron, M. B. Monagan, D. Ohashi, and S. M. Vorkoetter, *Maple Programming Guide*, Waterloo Maple Inc, 2012.

[4] André Heck, *Introduction to Maple*, Springer-Verlag, 2003.

[5] Roger Kraft, *Programming in Maple*, Purdue University Calumet, 2002.