Lecture 5

2D plots

1 Introduction

The built-in graphics capabilities of Scilab/Matlab are one of its strongest features. Compiled languages such as C and Fortran are generally text-based. Generating graphics requires either separate programs or libraries which are often operating-system specific. Scilab/Matlab, on the other hand, provides a complete environment in which graphical routines are an integral component and consistent across different operating systems.

There are many graphics routines. As always the Help command is a good way to see what is available. We are going to focus on a few of the most useful. This is one area where there is a fair amount of difference between Scilab and Matlab. We will try to emphasize those aspects in common, but our primary focus will be on Scilab.

2 The plot command (Scilab/Matlab)

Basic 2D plots can be generated using the plot command. The first argument is a vector of "x" data and the second argument is a vector of "y" data. These vectors must have the same size. For example

```
x = linspace(0,6,100);
y = sin(x);
plot(x,y);
```

Exercise 1: Plot $y=\cos(2x)$ over $-3\leq x\leq 3$ using 201 points.

To put multiple graphs on the same figure you can either execute multiple plot commands, or you can enter multiple *x,y* vector pairs in a single plot command as in

```
x = linspace(0,6,100);
y1 = sin(x);
y2 = cos(x);
plot(x,y1,x,y2);
```

Scilab/Matlab will automatically assign line attributes and/or colors as well as numerical axis labels. If you want to choose these yourself you can add a text argument after each x,y pair. For example

```
x = linspace(0,6,100);
y1 = sin(x);
y2 = cos(x);
y3 = 0.5-sin(x).^2;
y4 = 0.5-cos(x).^2;
plot(x,y1,'r-',x,y2,'g-',x,y3,'b-.',x,y4,'k:');
```

The letters denote colors and the symbols denote line types. Note that the single quote marks are required. Here are the basic color options

r	red
g	green
b	blue (default for first plot)
k	black
c	cyan
m	magenta
y	yellow
k	black
w	white

and the basic line types

-	solid line (default)
--	dashed line
:	dotted line
-.	dash-dot line

Line thickness can be specified using the following syntax.

```
plot(x,y2,'b:','linewidth',2);
```

This produces a red, dashed plot of thickness 3 and a blue dotted plot of thickness 2.

In place of a line, you can plot your data as discrete points using various symbols. For example

```
plot(x,y,'r*')
```

plots red asterisks at each (*x,y*) pair. The available symbols are

+	plus sign
o	circle
*	asterisk
.	point
x	cross
s	square
d	diamond
^	up triangle
v	down triangle
>	right triangle
<	left triangle
p	pentagram (star)

The symbol size can be specified as follows

```
plot(x,y,'r>','markersize',10);
```

A grid is often useful. To add a grid in Scilab use the `xgrid` command. The corresponding command in Matlab is `grid`.

3 Modifying a graphic window interactively (Scilab)

The default appearance of a plot is sufficient for many tasks. However there are situations where we might want to, say, add axis labels and a title and "enhance" the appearance of a plot in other ways. This is particularly the case when we are interested in saving a plot as an image file to be used in documents or presentations. As an example,

```
x = linspace(0,2*%pi,100);
y = sin(x);
plot(x,y);
```

Produces the graphic window shown in Fig. 1. We can move and resize this window with the

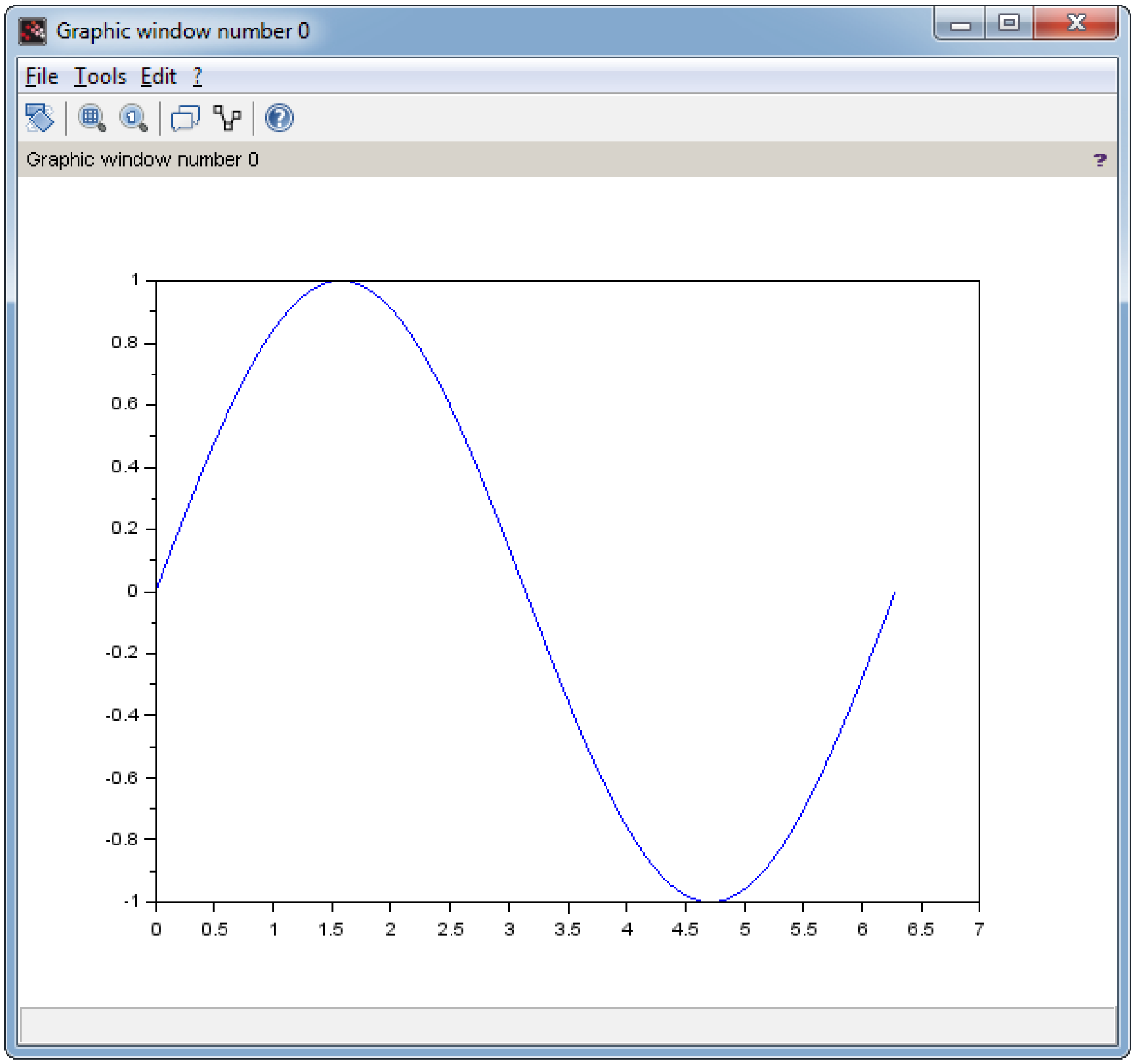
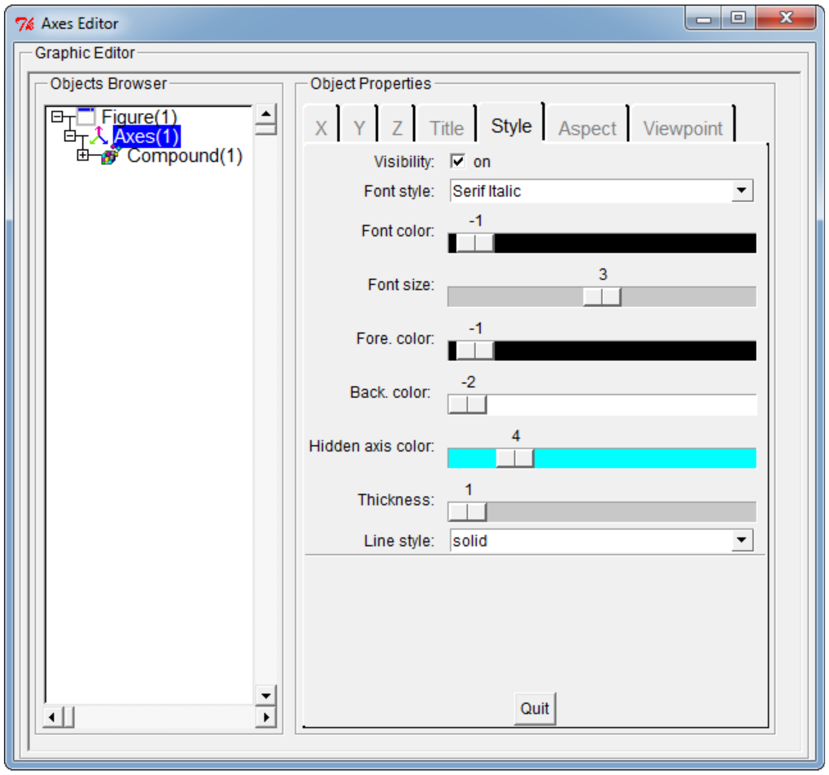


Fig. 1

mouse. We can also use the Edit menu to select Axes properties. This opens the Axes Editor (Fig. 2). In this window there are tabs labeled

```
X   Y   Z   Title   Style   Aspect   Viewpoint
```

Figures 2, 3 and 4 show the windows corresponding to the Style, X and Aspect tabs, respectively. In the Style window (Fig. 2) we can select the font type, color and size used for the numerical



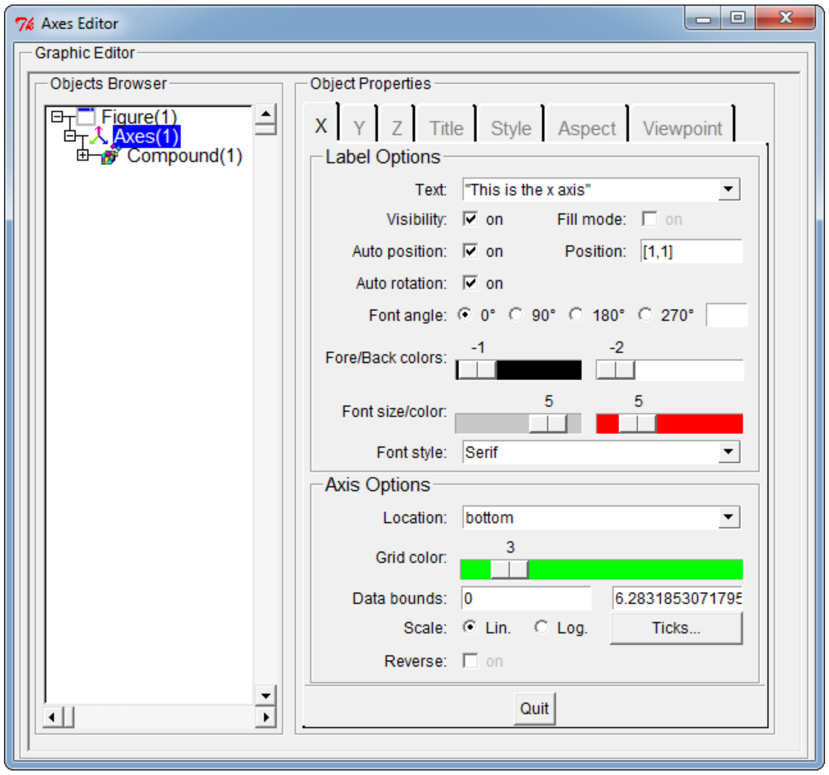
Visibility: ✓ on
Font style: Serif Italic
Font color: -1
Font size: 3
Fore. color: -1
Back. color: -2
Hidden axis color: 4
Thickness: 1
Line style: solid

Fig. 2

labels on the axes, in addition to other properties. These can best be learned by generating a plot and playing around with the various settings.

In the X window (Fig. 3) we can specify a text label for the x axis. This is entered into the Text box. Note that it must be enclosed in double quotes. We can also choose the font type, color and size of the label. The Location pull-down menu allows us to reposition the *x* axis. The Grid color slider selected causes an *x* grid to appear of the chosen color. The Data bounds boxes initially contain the minimum and maximum *x* values of the plotted data. Suppose we wanted our plot to scale such that it displayed the *x* axis region $-1 \le x \le 10$ (for instance we might want to compare it to another plot). We manually enter -1 and 10 in the Data bounds boxes to achieve this. We can also select Linear (default) or Logarithmic axis scaling. (Log scaling can only be used if $x > 0$ for all data.)

The Aspect tab window is shown in Fig. 4. Checking the Isoview box causes the plot to be scaled so that one unit has the same length along the *y* axis as it does along the *x* axis. This is very useful if our (*x*,*y*) data represent distance measurements, say in cm. Then the resulting plot is geometrically accurate. If the Isoview box is unchecked (default) the *x* and *y* data will be separately scaled to fill up graphic window.

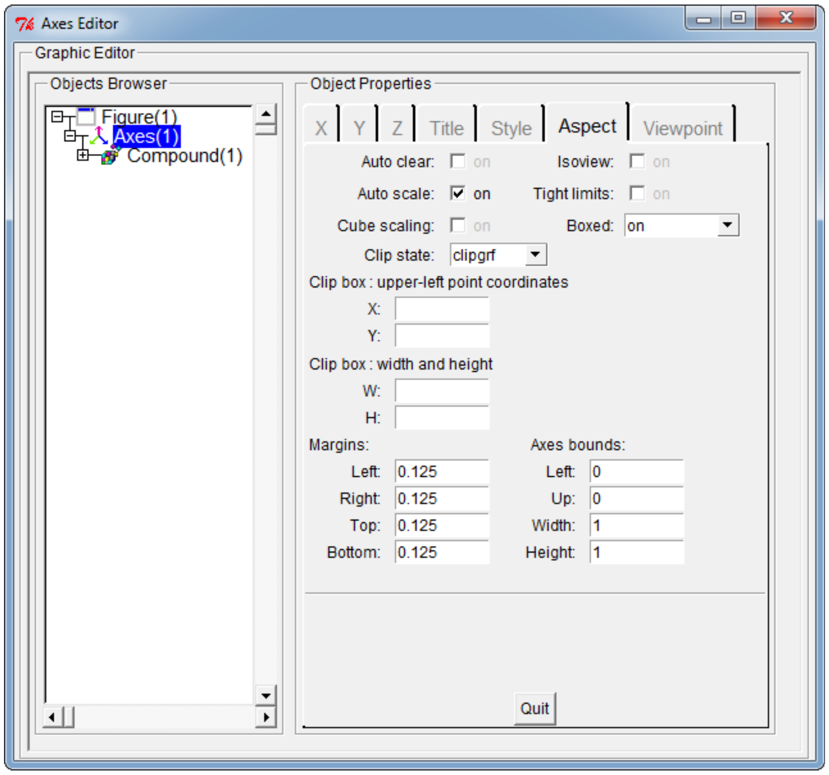


Text: "This is the x axis"
Visibility: ☑ on Fill mode: ☐ on
Auto position: ☑ on Position: [1,1]
Auto rotation: ☑ on
Font angle: ◉ 0° ○ 90° ○ 180° ○ 270°
Fore/Back colors: -1 -2
Font size/color: 5 5
Font style: Serif
Location: bottom
Grid color: 3
Data bounds: 0 6.283185307179
Scale: ◉ Lin. ○ Log.
Reverse: ☐ on

Fig. 3

In our example the data have x values over the range $0 \le x \le 2\pi = 6.283$. By default Scilab will display x values over a range such as $0 \le x \le 7$ so that the plot begins and ends on a “major tic mark.” Clicking the Tight bounds box overrides this behavior. The Margins boxes specify the space between the plotted axes and the edges of the figure. These values are fractions of the plot width or height, and we can change them as desired.

In the Objects Browser portion of the Axes Editor window appears the hierarchy Figure, Axes, Compound. Expanding the Compound object (click on the + sign) shows a Polyline object. Selecting this brings up the Polyline Editor (Fig. 5). Here we can modify the properties of the plotted line(s). If we uncheck the Visibility box the corresponding line will become invisible. We can also modify the line type, width and color. Alternately (as shown in the figure) we can uncheck the Line mode box and check the Mark mode to cause the data to appear as discrete symbols. The marker type, color and size can be selected as desired. The result is the graphic illustrated in Fig. 7.



Auto clear: ☐ on
Isoview: ☐ on
Auto scale: ☑ on
Tight limits: ☐ on
Cube scaling: ☐ on
Boxed: on
Clip state: clipgrf
Clip box : upper-left point coordinates
X:
Y:
Clip box : width and height
W:
H:
Margins:
Left: 0.125
Right: 0.125
Top: 0.125
Bottom: 0.125
Axes bounds:
Left: 0
Up: 0
Width: 1
Height: 1

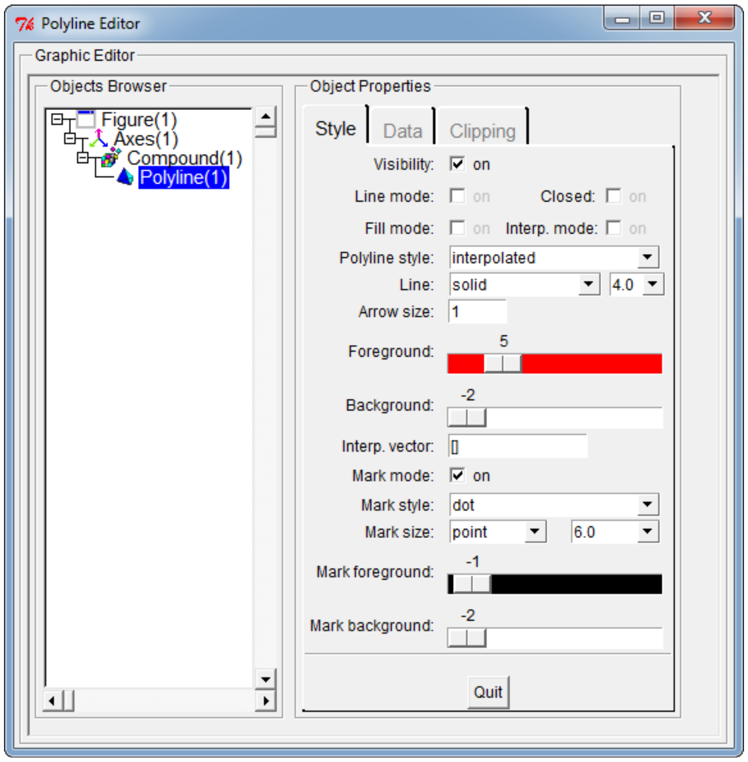
Fig. 4

4 Exporting graphics (Scilab)

Once we have a plot formatted to our liking we can export it as an image file which can then be imported into a word processor or other application. In the graphics windows we use the File menu to select Export to. Various types of files are supported. PNG image files are generally the most effect bit-mapped format for this purpose. The number of pixels depends on the size of the window (default is generally 610-by-460).

For higher quality we can use the SVG (scalable vector graphics) format. If an application supports this format it will allow the figure to be rendered at the highest possible resolution. Alternately software such as Inkscape (free and open-source, available at inkscape.org) can render an SVG file as a PNG file of arbitrary resolution.

Another approach is to use the “Copy to clipboard” option to directly copy and paste a figure. As in the direct PNG export case this produces a relatively low-resolution graphic, but if this is good enough then it's generally the fastest method.



Visibility: ☑ on
Line mode: ☐ on Closed: ☐ on
Fill mode: ☐ on Interp. mode: ☐ on
Polyline style: interpolated
Line: solid 4.0
Arrow size: 1
Foreground: 5
Background: -2
Interp. vector: []
Mark mode: ☑ on
Mark style: dot
Mark size: point 6.0
Mark foreground: -1
Mark background: -2

Fig. 5

5 Modifying a graphic window within a program (Scilab)

Using the graphic edit menus is fine for a "one off" figure, but often we want to a program to directly generate a plot with some desired formatting. This is especially true if we intend to use the program to generate plots of many different data sets. In that case it's too tedious to manually format each plot as we generate it. Instead we need to learn how to use programming commands to automatically modify the formatting of a figure.

5.1 *scf and clf commands*

A new (blank) figure can be produced using the `scf` (set current figure) command. This has a single argument which is the desired graphic window number. The commands

```
scf(1);
scf(2);
```

generate separate blank graphic windows 1 and 2. Plot commands are directed to the last set figure. Therefore

```
scf(1);
plot(x1,y1);
scf(2);
plot(x2,y2);
```

would plot `(x1,y1)` in Figure 1 and `(x2,y2)` in Figure 2. The `scf` command returns a "handle" to a structure defining the figure properties. So

```
fg = scf(1);
```

Assigns to variable `fg` a structure with the following properties.

```
Handle of type "Figure" with properties:
   ========================================
   children: "Axes"
   figure_position = [865,219]
   figure_size = [626,586]
   axes_size = [610,460]
   auto_resize = "on"
   viewport = [0,0]
   figure_name = "Graphic window number %d"
   figure_id = 1
   info_message = ""
   color_map = matrix 32x3
   pixel_drawing_mode = "copy"
   anti_aliasing = "off"
   immediate_drawing = "on"
   background =  -2
   visible = "on"
   rotation_style = "unary"
   event_handler = ""
   event_handler_enable = "off"
   user_data = []
   resizefcn = ""
   closerequestfcn = ""
   resize = "on"
   toolbar = "figure"
   toolbar_visible = "on"
   menubar = "figure"
   menubar_visible = "on"
   infobar_visible = "on"
   dockable = "on"
   layout = "none"
   layout_options = "OptNoLayout"
   default_axes = "on"
   icon = ""
   tag = ""
```

Properties can be modified with commands such as

```
   fg.figure_position = [100,150];
```

For example, the following commands

```
   fg = scf(1);
   fg.axes_size = [400,300];
   fg.figure_position = [0,0];
   fg = scf(2);
   fg.axes_size = [400,300];
   fg.figure_position = [416,0];
   fg = scf(3);
   fg.axes_size = [400,300];
   fg.figure_position = [0,426];
   fg = scf(4);
   fg.axes_size = [400,300];
   fg.figure_position = [416,426];
```

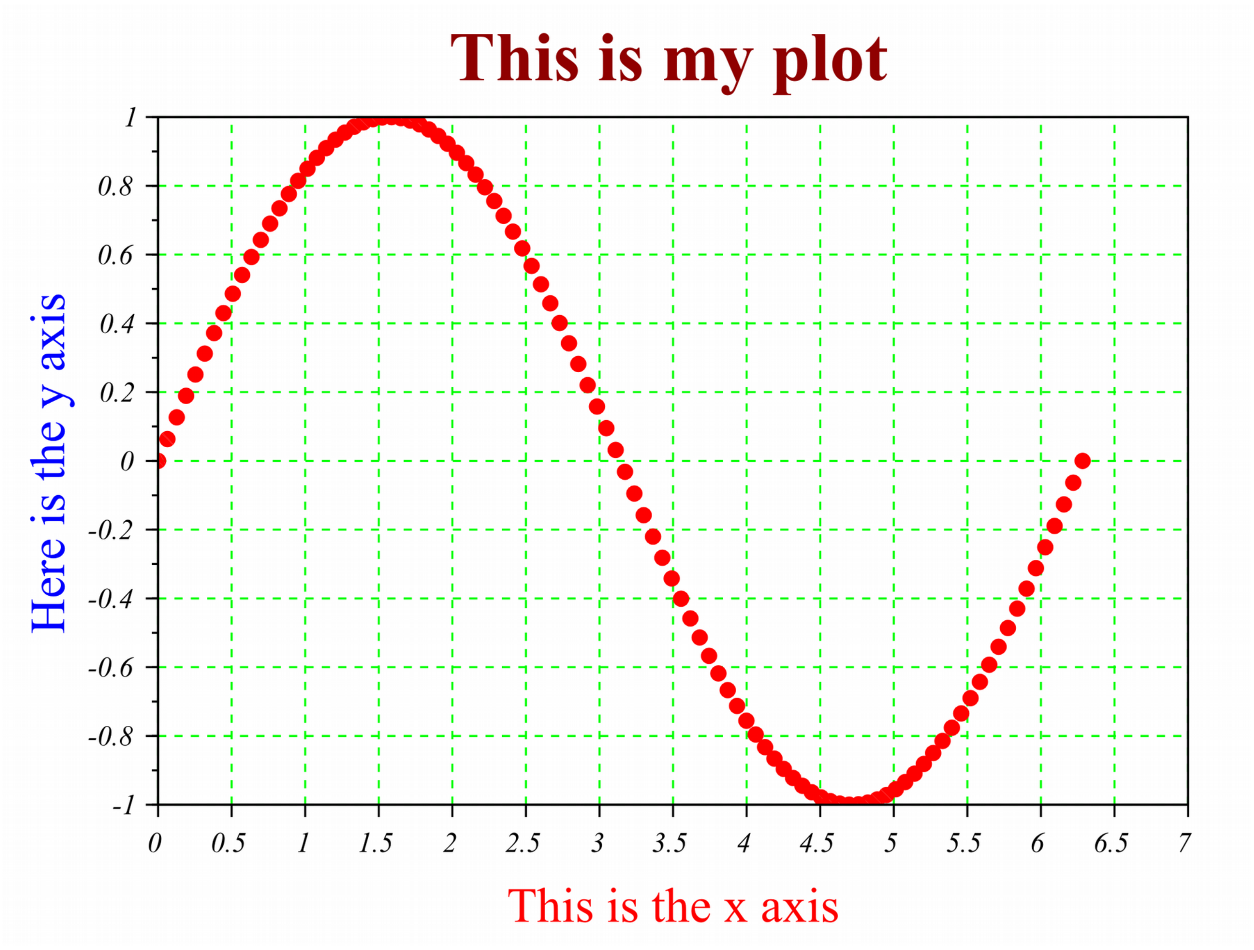


Fig. 6

generates four graphic windows (1,2,3,4) arranged in a 2-by-2 array as shown in Fig. 7.

We can then select graphic window 2 and draw a plot in it with the following commands

```
scf(2);
x = linspace(0,2*%pi,50);
y = sin(x);
plot(x,y,'r-','linewidth',3);
xgrid;
```

We have seen that repeated plot commands add curves to a figure. If we want to “start over” we need to clear the figure using the `clf()` command. This applies to the currently selected figure. If we want to clear a specific figure, say graphic window 3, we can specify `clf(3)`.

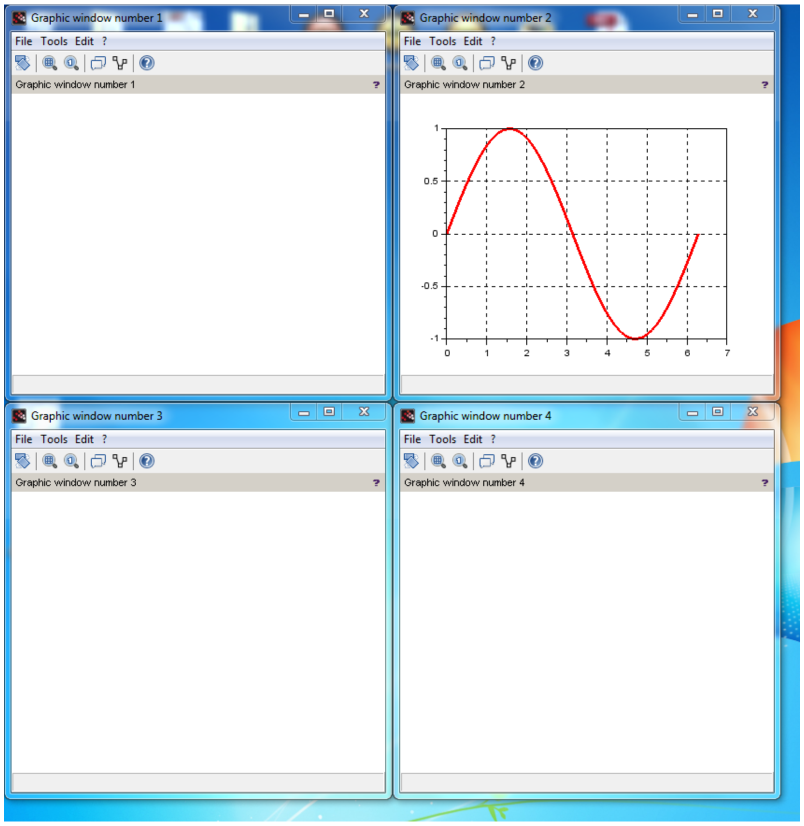


Fig. 7 Four graphics windows produced by above code.

5.2 Modifying axes properties

Most of the properties we want to modify are in the axes structure. To access this we use the `gca()` command. The statement

```
    ax = gca();
```

assigns to variable `ax` a “handle” to a structure specifying the axes properties of the currently selected figure. These properties are as follows.

```
Handle of type "Axes" with properties:
======================================
parent: Figure
children: "Compound"

visible = "on"
axes_visible = ["on","on","on"]
axes_reverse = ["off","off","off"]
grid = [-1,-1]
grid_position = "background"
grid_thickness = [1,1]
grid_style = [3,3]
x_location = "bottom"
y_location = "left"
title: "Label"
x_label: "Label"
y_label: "Label"
z_label: "Label"
auto_ticks = ["on","on","on"]
x_ticks.locations = matrix 11x1
y_ticks.locations = matrix 11x1
z_ticks.locations = []
x_ticks.labels = matrix 11x1
y_ticks.labels = matrix 11x1
z_ticks.labels = []
ticks_format = ["","",""]
ticks_st = [1,1,1;0,0,0]
box = "on"
filled = "on"
sub_ticks = [4,1]
font_style = 6
font_size = 1
font_color = -1
fractional_font = "off"

isoview = "off"
cube_scaling = "off"
view = "2d"
rotation_angles = [0,270]
log_flags = "nnn"
tight_limits = "off"
data_bounds = [0,-0.9999233;5,0.9995736]
zoom_box = []
margins = [0.125,0.125,0.125,0.125]
auto_margins = "on"
axes_bounds = [0,0,1,1]

auto_clear = "off"
auto_scale = "on"

hidden_axis_color = 4
hiddencolor = 4
line_mode = "on"
line_style = 1
thickness = 1
mark_mode = "off"
mark_style = 0
mark_size_unit = "tabulated"
mark_size = 0
mark_foreground = -1
mark_background = -2
foreground = -1
background = -2
arc_drawing_method = "lines"
clip_state = "clipgrf"
clip_box = []
user_data = []
tag =
```

Properties can be modified by assigning the corresponding variables new values. For example

```
ax.isoview = "on";
```

causes the *x* and *y* axes to have the same scaling. To change the font style and font size of the axes tic labels and add a black grid we might use

```
ax.font_size = 4;
ax.font_style = 3;
ax.grid = [1,1];
```

To add a label to the *x* axis with a desired font and font size we can execute the following commands

```
ax.x_label.text = "this is the x label";
ax.x_label.font_size = 5;
ax.x_label.font_style = 3;
```

Similarly for the *y* (and *z*) labels and title. To find out more about, say, `font_style` settings see

```
help graphics_fonts
```

6 Learning more

In this lecture we have focused on rectangular *x,y* plots. These are not the only types of plots that we may want to generate. Polar coordinate r,θ plots are common. Other examples are plots of vector fields (such as fluid velocity) and histograms. To learn more use the help window and follow the links

```
Help => Graphics => 2d_plot //Scilab
Help => graph2d %Matlab
```

You'll find routines such as polarplot, histplot, and many others. Once you are comfortable with rectangular plots you should find it easy to use the other plotting routines. The help sections on most of these have examples that you can run and code you can examine or use as a template for your own programs.