

The PhilsPlot Graphics Library, version 3

September 2009

Introduction

A picture is worth a thousand words, and a movie a thousand pictures. Taking the time to put graphical output in a code is often well worth the effort both for improving understanding of the result and for its aid in debugging. This is especially true for simulations of systems which vary in time. We have written a very simple set of twelve functions for use in plotting and animating the output of your codes.

Terminology

In the following descriptions of the routines, we use the following terms in the definitions of many functions:

- **color**: A **color** argument is always an integer between 0 and 15 giving the color of the item to be drawn. 0 is black, 1 is white, 2 is red, 3 green, 4 blue, etc. You may want to write a small program you can keep around which plots a series of dots of various colors to remind yourself of the correspondences. You can selectively erase various parts of a plot by re-drawing the same item with color 0.
- **lstyle**: An **lstyle** specifies the type of a line. 1 gives a full solid line, 2 is dashed, 3 dot-dash-dot-dash, 4 dotted, and 5 dash-dot-dot-dot.
- **lweight**: An **lweight** specifies the weight (thickness) of a line and is a integer value which can range from 1 to 201. A value of 1 is the usual thin line; heavier lines go with larger values.
- **expand**: An **expand** is a **double** value which sets the size of characters plotted. 1.0 is about 1/40 of the size of the plot window; usually 2.0 to 3.0 works well.

List of Routines

open_plot

```
void open_plot(const char *geom)
```

`open_plot` opens a window on an X-capable display. The optional argument `geom` is a window geometry: `open_plot("700x500")` will open a 700 by 500 pixel window. This must be called first to use the rest of the routines. *Note*: Only one of `open_plot` and `png_plot` may be used at a time.

png_plot

```
void png_plot(const char *geom, const char *filename)
```

`png_plot` opens a png format file named `filename` with the specified geometry: `open_plot("700x500")` will open a 700 by 500 pixel file. This must be called first to use the rest of the routines. *Note:* Only one of `open_plot` and `png_plot` may be used at a time. Unlike for `open_plot`, however, when using `png_plot` one must call `close_plot` as the last plotting command to close the output file.

close_plot

```
void close_plot(void)
```

`close_plot` write out the final information for a `png_plot`, flushes the buffers and closes the file. It *must* be used with `png_plot`. It can be used with `open_plot`, in which case it performs no function. Getting into the habit of using `close_plot` means that you can simply change the `open_plot` call in your code to a `png_plot` call to get hardcopy with no other changes to your code.

flush_plot

```
void flush_plot(void)
```

All graphics are written on a “virtual” canvas in the computer; what you plot does not automatically appear on the screen. The `flush_plot` routine maps the graphics which are in the virtual canvas onto the screen. This is important for animation. Assume that all graphics appeared on the screen as soon as you called for them. You would start out erasing the frame and then draw the axes and all of the points and curves. As soon as you are done with the picture, you would then erase it and start again. The screen would appear always to be in a partially-completed state – as soon as a given screenfull was finished, it would be immediately erased! Instead, by leaving the previous screenfull in place until its replacement is fully drawn and then replacing it all at once (via a call to `flush_plot`), a completed screenfull is always visible, and the illusion of continuous motion is produced. Note that you must call `flush_plot` to get anything you draw to appear on the screen!

erase_plot

```
void erase_plot(void)
```

A call to `erase_plot` erases the virtual canvas in order to start drawing the next frame. Note that the erased screen will not take effect until you call `flush_plot`!

setlim_plot

```
void setlim_plot(double xmin, double xmax, double ymin, double ymax)
```

`setlim_plot` sets the limits of the plotting canvas to the given ranges. It does not draw axes; indeed, it doesn't draw anything!

box_plot

```
void box_plot(double xmin, double xmax, double ymin, double ymax,  
             double expand, int color,  
             const char *bottom_label, const char *left_label,  
             const char *top_label, const char *right_label)
```

`box_plot` sets the limits of the plotting canvas to the given ranges and draws a box with axes and labels. To leave a label blank, give an empty string ("") as the argument.

locate_plot

```
void locate_plot(double x, double y)
```

`locate_plot` sets the "pen" to the location (x,y) without drawing anything.

drawto_plot

```
void drawto_plot(double x, double y, int color, int lstyle, int lweight)
```

`drawto_plot` draws a line from the current location to the location (x,y) with color `color`, type `lstyle`, and weight `lweight`.

putpoint_plot

```
void putpoint_plot(double x, double y, int nvertex, int style, int color,  
                  double expand, int memflag)
```

`putpoint_plot` plots a point at the position (x,y). The point has `nvertex` vertices (3 a triangle, 4 a square, ~ 10 a circle, etc.); a value of 1 or 2 gives a single dot. A `style` of 1 gives a filled polygon, 2 gives an outline, 3 fills the symbol with hatching, and 3 fills it with cross-hatching. `expand` controls the size of the point in the usual way, and `color` controls the color. If `memflag` is equal to 1, the point is remembered so that a subsequent call to `delpoint_plot` (below) will erase that point. In other words, if you draw three points with three calls to `putpoint` which have `memflag` = 1, three subsequent calls to `delpoint` will erase the first, second, and then third of the three points (first plotted, first erased). Points plotted with `memflag` not equal to 1 are forgotten and cannot subsequently be erased with `delpoint_plot`. They can still be erased by drawing them again with `color` = 0.

delpoint_plot

```
void delpoint(void)
```

`delpoint_plot` deletes points drawn with `putpoint_plot` (w/ `memflag=1`). See above.

curve_plot

```
void curve_plot(int npts, double *x, double *y,  
               int color, int lweight, int lstyle, int nlabel, double expand,  
               const char *clabel)
```

Given `npts` points in the arrays `x` and `y`, this function draws a curve connecting these points. The line will have color `color`, type `lstyle`, and weight `lweight`. The curve will have `nlabel` labels along its length specified by `clabel` with character size `expand`.

drawimage_plot

```
int drawimage_plot(int dimx, int dimy, double **a,  
                  double x1, double y1, double x2, double y2,  
                  double a1, double a2,  
                  const char *xlab, const char *ylab, const char *clab)
```

This routine takes a two-dimensional array of doubles, with first dimension `dimx` and second dimension `dimy` and plots it in a box as an image. The dimensions `x1`, `x2`, `y1`, `y2` are as for `box_plot`, as are the character strings for the bottom and left labels `xlab` and `ylab`, respectively. Each pixel in the image with a value between `a1` and `a2` is assigned a color from a *colormap* which runs from dark blue for values of `a1` to bright red for values of `a2`. A bar showing the numerical value associated with the colors in the colormap is plotted to the right of the image, with a label given in `clab`.

delay_plot

```
void delay_plot(int time)
```

Sometimes the plot goes by too fast. Calling `delay` with an integer argument `time` will pause the program for `time` milliseconds.

Compiling

First, don't forget to include the line:

```
#include "philspplot.h"
```

in every file which makes uses these functions. It provides the function prototypes so that the compiler will know what kinds of arguments the PhilsPlot functions expect.

Let us say that you have two files with code in them which needs to be compiled into an executable along with the PhilsPlot libraries: `main.c` and `sub.c`. You will then type, for example

```
gcc -I /home/pinto/include -o myprog.exe main.c sub.c -L/home/pinto/lib -lphilsplot \
-L/usr/X11R6/lib -lX11 -lpng -lg2c
```

(replace the “/home/pinto/” with the appropriate directory for your machine). The `-I` flag tells the compiler an additional place to look for files which have been `#include`-ed in your program. The `-L` flags tell the compiler additional places to look for libraries. The `-l` flags tell the compiler which libraries to *link* into your executable program. The `-lphilsplot` is the PhilsPlot library, the `-lX11` is the X-window system which actually draws to your screen, the `-lpng` is a library for making PNG graphics files (a format widely used on the web), and, finally, the `-lg2c` is a set of Fortran run-time libraries which are used by the PGPLOT library which was used in writing PhilsPlot.

You will need to set two *environment variables* to let the philsplot functions know where to find certain files. These variables are variables of the shell program you are using in your terminal window. You can also define a simple command *alias*, a short-cut to make compiling easier (or at least easier to type in!).

Your system may actually use a different set of libraries, depending upon what compilers and other software is installed. To make using PhilsPlot easier, in the include directory there are two *shell scripts* with the commands to set up your environment.

For those of you using the `tcsh` shell, add the following line to your `.cshrc` file:

```
source /home/pinto/include/philsplot.csh
```

For those of you using the `bash` shell, you will need to add these lines to your `.bashrc` file to perform the same function:

```
source /home/pinto/include/philsplot.sh
```

Again, replace “/home/pinto” with the appropriate directory for your machine. These scripts set the environment variables `PGPLOT_DIR` and `PGPLOT_DEV`, and define two aliases, `pc` and `pcpp`.

Now, to compile a program written in the C language, you can simply type

```
pc -o myprog.exe main.c sub.c
```

and the file `myprog.exe` will be created (assuming no errors in your code are found by the compiler!).

Similarly, to compile a C++ program, you would use something like

```
pcpp -o myprog.exe main.cpp sub.cpp
```

Finally, if your program complains:

```
%PGPLOT, PGSCF: no graphics device has been selected
```

or some similar error about not having selected a graphics device, you don't have your DISPLAY environment variable set. In most cases this should be automatic, but if it is not, assuming you are on a machine named pc123.as.arizona.edu, set the environment variable with (in the tcsh shell)

```
setenv DISPLAY pc123.as.arizona.edu:0.0
```

or, in the bash shell,

```
DISPLAY=pc123.as.arizona.edu:0.0 ; export DISPLAY
```

(use whatever is the correct name for your own machine, of course).

Examples

These examples are available for download from the class web site. Here is a sample of code to plot two functions:

```
// example1.c
#include <stdio.h>
#include <math.h>
#include "philspplot.h" // must include this header file

double sinc(double x) {
    if (x!=0.0)
        return sin(x)/x;
    else
        return 1.0;
}

int main(void) {
    double x1, x2, y1, y2, expand;
    int color, n, i;

    x1 = -4*M_PI-0.1; x2 = 4.0*M_PI+0.1; // set limits of plot
    y1 = -1.1; y2 = 1.1;

    n = 101;

    double x[n], y[n]; // get arrays to hold curve

    for (i=0; i<n; i++) x[i] = x1 + ((double)i/(double)n)*(x2-x1);
    for (i=0; i<n; i++) y[i] = cos(x[i]);
}
```

```
open_plot("700x700");

expand = 1.1;
color = 1;

box_plot(x1, x2, y1, y2, expand, color,
         "x axis", "y axis", "top label", "right side");

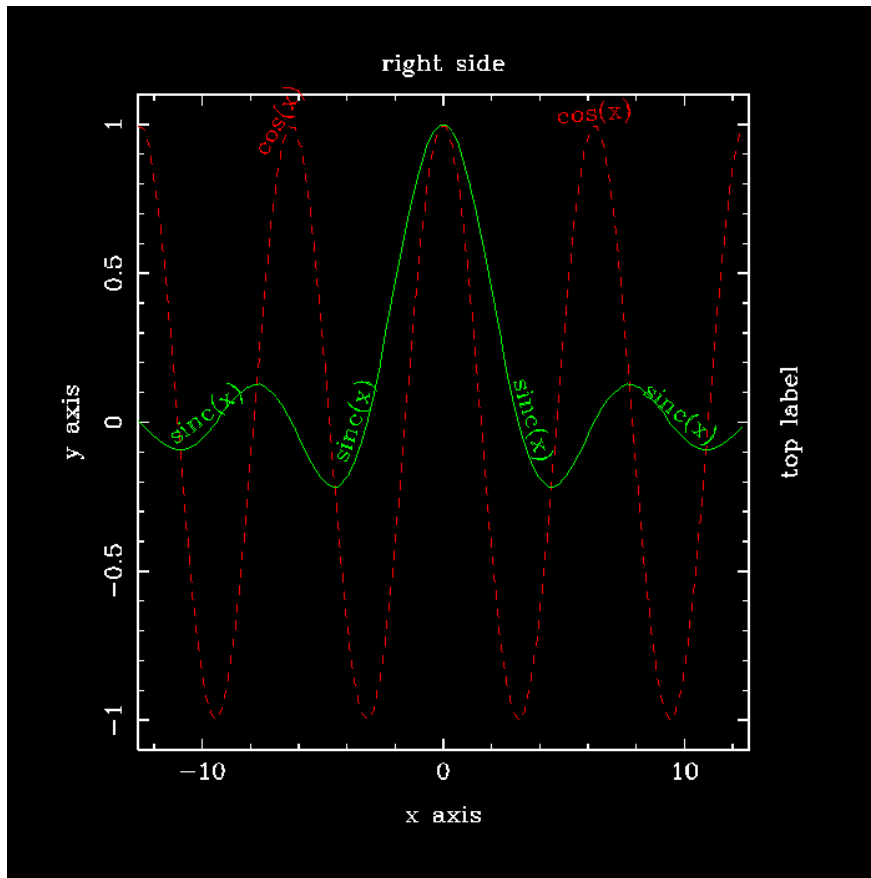
color = 2;
curve_plot(n, x, y, color, 1, 2, 2, 1.1, "cos(x)");

for(i=0; i<n; i++) y[i] = sinc(x[i]);
color = 3;
curve_plot(n, x, y, color, 1, 1, 4, 1.1, "sinc(x)");

flush_plot();

printf("type the enter key when finished: ");
getchar();
}
```

and here is the output:



To animate a plot, we need to make more careful use of `flush_plot`. Here are the steps in an animation:

1. Draw a complete picture in the computer's memory, using whatever functions are necessary.
2. Write the complete picture out to the screen using `flush_plot`.
3. Pause for some interval so the animation doesn't go by too quickly by using `delay_plot(100)`, where the integer argument is the number of milliseconds to wait.
4. Erase the whole picture using `erase_plot`, or just parts of it by re-drawing them with `color=0`.
5. Repeat.

Here are a number of examples of how to use the PhilsPlot library. Type them in and try running them; for the animations, we cannot give example output in a document like this – you have to see them in action!

Consider the following code to animate a point tracing out the curve of $\sin x$:

```

// example2.c
#include <stdio.h>
#include <math.h>
#include "philsplot.h"

int main(void) {
    double x1, x2, y1, y2, x, y;

    // set limits of plot
    x1 = -4*M_PI-0.1; x2 = 4.0*M_PI+0.1;
    y1 = -1.1; y2 = 1.1;

    // initialize the plot
    open_plot("700x700");

    // first point to plot
    x = x1; y=sin(x1);

    // animation loop:
    while(x<x2) {

        // draw the box with axes and labels
        box_plot(x1, x2, y1, y2, 1.1, 1,
                "x axis", "y axis", "top label", "right side");

        // plot the point (last arg is mem=0)
        putpoint_plot(x,y,4,1,2,2.5,0);

        // flush to screen
        flush_plot();

        // wait a bit
        delay_plot(20);

        // erase the plot (in memory, not the screen!)
        erase_plot();

        // update the position
        x = x + 0.05;
        y = sin(x);
    }

    printf("type the enter key when finished: ");
    getchar();
}

```

We can write this another way, one which will be faster if we have a lot to plot. The box in the previous example remained the same in all plots. We could simply erase the point, using `delpoint_plot`, having set the `mem` argument to true (1) so that the library remembers the points for you.

```
// example3.c
#include <stdio.h>
#include <math.h>
#include "philplot.h"

int main(void) {
    double x1, x2, y1, y2, x, y;

    // set limits of plot
    x1 = -4*M_PI-0.1; x2 = 4.0*M_PI+0.1;
    y1 = -1.1; y2 = 1.1;

    // initialize the plot
    open_plot("700x700");

    // draw the box once
    box_plot(x1, x2, y1, y2, 1.1, 1,
             "x axis", "y axis", "top label", "right side");

    // first point to plot
    x = x1; y=sin(x1);

    // animation loop:
    while(x<x2) {

        putpoint_plot(x,y,4,1,2,2.5,1);
        flush_plot();
        delay_plot(20);
        delpoint_plot();

        x = x + 0.05;
        y = sin(x);
    }

    printf("type the enter key when finished: ");
    getchar();
}
```

Note that the point will erase any parts of the box which it over-writes; if you are fastidious, you could alter the limits so this does not occur, but we will mainly be interested in seeing the behavior of our solution, not with producing Hollywood-level animations!

We can leave a larger number of the points on the screen by recognizing that `delpoint_plot` operates by deleting the *oldest* point still on the screen. Consider the result of

```
// example4.c
#include <stdio.h>
#include <math.h>

int main(void) {
    int count;
    double x1, x2, y1, y2, x, y;

    // set limits of plot
    x1 = -4*M_PI-0.1; x2 = 4.0*M_PI+0.1;
    y1 = -1.1; y2 = 1.1;

    // initialize the plot
    open_plot("700x700");

    // draw the box once
    box_plot(x1, x2, y1, y2, 1.1, 1,
             "x axis", "y axis", "top label", "right side");

    // first point to plot
    x = x1; y=sin(x1);

    // animation loop:
    count = 0;
    while (x<x2) {

        putpoint_plot(x,y,4,1,2,2.5,1);
        flush_plot();
        delay_plot(20);
        if(count>7) delpoint_plot();

        x = x + 0.05;
        y = sin(x);
        count++;
    }

    printf("type the enter key when finished: ");
    getchar();
}
```

```
}
```

Finally, here is an example of how to use the image-plotting function. Do you recognize the image?

```
// example5.c
#include <stdio.h>
#include <math.h>
#include "philplot.h"

#define MAX(a,b) ((a)>(b))? (a):(b)
#define MIN(a,b) ((a)<(b))? (a):(b)

int main(int argc, char *argv[]) {
    double xmin, xmax, ymin, ymax;
    int color, style;
    double expand;

    open_plot("800x600");

    int i, j, it;
    int xdim, ydim;
    double x0, y0, x1, x2, y1, y2, x, y, xt;
    double amax, amin;

    xdim = 900; ydim = 600;

    double a[xdim][ydim];

    x1 = -2.05; x2 = 2.05; y1 = -1.25; y2 = 1.25;

    amin = 1e30; amax = -amin;

    for(i=0; i<xdim; i++) {
        x0 = x1 + (x2-x1)*(double)i/(double)(xdim-1);
        for(j=0; j<ydim; j++) {
            y0 = y1 + (y2-y1)*(double)j/(double)(ydim-1);

            x = x0; y = y0;
            it = 0;
            while(x*x+y*y<=4 && it < 100) {
                xt = x*x - y*y - 0.913;
                y = 2*x*y + 0.266;
                x = xt;
                it++;
            }
        }
    }
}
```

```

    }

    a[i][j] = it + (log(2*log(2)) - log(log(sqrt(x*x+y*y))))/log(2);
    if(it==100) a[i][j] = 0;
    amax = MAX(a[i][j],amax);
    amin = MIN(a[i][j],amax);
  }
}

drawimage_plot(xdim, ydim, a, x1, y1, x2, y2, 0.0, amax/5,
               "real z", "imag z", "iterations");

flush_plot();

printf("type the enter key when finished: ");
getchar();
}

```

We can even animate this. Note we haven't used `delay_plot` since this code takes long enough to prepare the image that there is no need for further delay:

```

// example6.c
#include <stdio.h>
#include <math.h>
#include "philplot.h"

#define MAX(a,b) ((a)>(b))? (a):(b)
#define MIN(a,b) ((a)<(b))? (a):(b)

int main(int argc, char *argv[]) {
    int i, j, it, xdim, ydim;
    double x0, y0, x1, x2, y1, y2, x, y, xt, cr, ci;;

    xdim = 450; ydim = 300;
    double a[xdim][ydim];

    x1 = -1.85; x2 = 1.85; y1 = -1.0; y2 = 1.0;
    ci = 0.124; cr = -0.8;

    open_plot("800x600");

    while (cr < -0.735) {

        for (i=0; i<xdim; i++) {
            x0 = x1 + (x2-x1)*(double)i/(double)(xdim-1);
            for (j=0; j<ydim; j++) {
                y0 = y1 + (y2-y1)*(double)j/(double)(ydim-1);

```

```

x = x0; y = y0; it = 0;
while(x*x+y*y<=4 && it <200) {
    xt = x*x - y*y + cr;
    y = 2*x*y + ci;
    x = xt;
    it++;
}
a[i][j] = it + 0.47123 - log(log(sqrt(x*x+y*y)))/log(2);
}
}

drawimage_plot(xdim, ydim, a, x1, y1, x2, y2, 0.0, 100,
               "real z", "imag z", "iterations");

flush_plot();
erase_plot();

cr += 0.001;
}

printf("type enter to quit: ");
getchar();
}

```