Internal Report #82
SRL Plotting Procedures

Nancy Collins
Don Mitchell
Eric Holstege

Calif Institute of Technology
Pasadena, Ca

ABSTRACT

This document describes the plotting software available on the PDP 11/70. There is a library of subroutines (i.e. *plot(x,y)*, *print("string")*, etc.) to be called from user programs. There are plot output programs for the Calcomp and Versatec plotters and for the Hewlett-Packard graphics terminals.

Also included are sample plotting programs and their output, and examples of usage.

A plot multiplexer exists for accumulating multiple plots on a single pass through the data, and the documentation is included as Appendix A.

This document replaces "*SRL Internal Report #73*".

August 27, 1981

Internal Report #82
SRL Plotting Procedures

Nancy Collins
Don Mitchell
Eric Holstege

Calif Institute of Technology
Pasadena, Ca

Introduction

The plot software described in this paper allows the user to write programs that can produce sophisticated plots using a collection of simple subroutines.

The separation of plotting into two halves - the user program emitting coded plot commands, and the system program reading the commands and making the plot - allows a wide range of freedom for the user. The intermediate coded commands can be saved in a file for later plotting, or plotted first on an HP graphics terminal and later plotted on the hardcopy printers if they are correct, or plotted several times without having to go through the accumulation program again.

Since applications vary so widely, and the library subroutines allow such flexibility, few high-level routines are part of the system library. (It was discussed, but deemed impossible to find a "one-routine-does-all" that anyone could agree on.) Most users write their own plot set-up subroutines, constructed out of library subroutines, including their own preferences for reasonable default values.

1. Library Subroutines

The following subroutines are called from the user program and write coded plot instructions on the standard output. The standard output must either be redirected to a file or piped directly into the plot output program. Any messages intended for the user must be written on standard error.

To include the routines in user programs, compile them with the -lP option.

1.1. General Notes

In the following list of subroutines, variables starting with "x" or "y" must be replaced with floating point numbers. Variables starting with "a" represent integer values.

The C compiler interprets the constant "10" as in integer value, and the constant "10." as a floating point value. Strange things happen if a subroutine is passed the wrong type of argument. If you need to change the type of a variable, see "casts" in the C manual.

Upper case symbols refer to predefined values in the #include file "plot.h". Most library subroutines return the value *PLOT_ERROR* on an error.

1.2. Glossary

Screen Coordinates
or
Plotter Coordinates

Integer values which define locations on the physical plotting area. X coordinates range from 0 to 2047, Y coordinates range from 0 to 1599. This is the maximum resolution of the device.

World Coordinates
or
User Coordinates

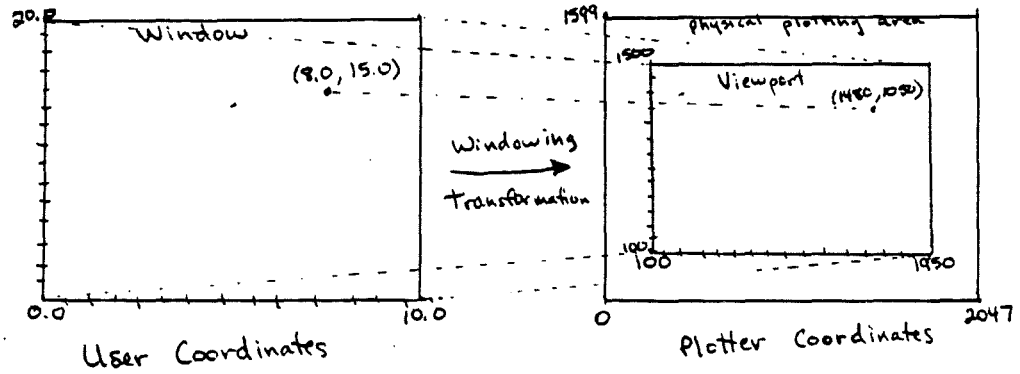
Floating point numbers which represent locations on the logical plotting area, in terms of user-defined values.

Viewport

Subarea of the total plotting screen where plotting may be done. Labels and headings may be printed outside the viewport, but no points or lines will be plotted.

Windowing Transformation
or
Window

The transformation of points from user coordinates to plotter coordinates. Translation and scaling are done, but no rotation.



1.3. Subroutines

initplot()

Initializes internal variables. (This routine replaces *openplot()*). It sets the default viewport and window to the maximum plotting area (0,0,2047,1599).

endplot()

Flushes buffers, stops plot program from accumulating data, and causes existing data to be plotted. (Note: the commands *closeplot()* and *erase()* no longer exist.) *endplot()* resets the text size to 1, the text direction to *TD_R0* and the line type to *LT_SOLID*. It does not change any of the other plot settings, so if the viewport and window are the same, another *initplot()* is not necessary for the next plot.

setviewport(axl,ayl,axh,ayh)

Sets the area available for plotting, in terms of plotter coordinates. The call *setviewport(MAX_VIEWPORT)* sets the viewport to roughly the maximum plotting area. *MAX_X_VIEW* and *MAX_Y_VIEW* are also defined (as 2047 and 1599 respectivly).

setwindow(xl,yl,xh,yh)

Defines the user coordinate values of the corners of the viewport. This defines the transformation of each point from user to plotter coordinates.

setscreen(xl,yl,xh,yh)†

After a viewport and windowing transformation have been defined (by calling *setviewport()* and *setwindow()*), a new viewport can be defined in terms of user-coordinates. *setscreen()* uses 4 floating-point user-coordinates and the old windowing transformation to define a new viewport.

setextsize(an)

Sets the text size to *an*, where *an* is in the range 1-8. Returns the old text size.

setextdir(an)

Sets the default text direction, where *an* is one of: *TD_R0*, *TD_R90*, *TD_R180*, *TD_R270*. Returns the old text direction.

setextjust(an)

When a string is printed with *print()*, the relation of the string to the current "pen" position is controlled by the text justification. The text may be centered on the present position, or right or left justified. The current plotting routines ignore vertical (upper or lower) justification and center the string vertically. *an* is one of: *TJ_CL*, *TJ_CC*, *TJ_CR* for left, center and right justifications respectivly. Returns the old justification.

setlinetype(an)

Sets the line type, where *an* is one of *LT_SOLID* (the default), *LT_DASHED*, *LT_DOTTED*, *LT_DOTPLOT*. ‡

---

† I have never found an occasion when it was necessary to use this routine. *nsc*

‡ For making scatterplots, the routines *setlinetype(LT_DOTPLOT)*, and *draw(x,y)* can be used (and are faster) than *setplotchar('.')* and *plot(x,y)*. See section 3 for examples.

setplotchar(c)

If *'c'* is a printing ascii character, all subsequent calls to *plot(x,y)* will plot that character. Also available are a set of special characters (listed below). These characters will be centered at the plotting location. The subroutine returns the old plot character.

description	name
plus sign	C_PLUS
hollow square	C_SQUARE
hollow diamond	C_DIAMOND
asterisk	C_STAR
solid circle	C_SLD_CIRCLE
hollow circle	C_CIRCLE
solid square	C_SLD_SQUARE
diagonal cross	C_CROSS
solid diamond	C_SLD_DIAMOND

settics(alength,dir)

Sets the default length and direction of tic marks on the axis. The symbols *SHORT_TICS*, *MED_TICS* and *LONG_TICS* are defined to give reasonable lengths. Valid options for direction are *IN*, *OUT*, *BOTH*, *UP*, *DOWN*, *LEFT* and *RIGHT*. The defaults are *IN* and *MED_TICS*.

xaxis(x,y,length,divisions)

yaxis(x,y,length,divisions)

Draws an axis starting at *x*, *y* and extending *length* (all in user-coordinates). *divisions* is the number of sections the axis is divided into (an integer value). *divisions - 1* tic marks are actually drawn.

logxaxis(x,y,length,decades)

logyaxis(x,y,length,decades)

Draws a log axis starting at user-coordinates *x*, *y* and extending *length* (also in user-coordinates). *decades* is an integer value.

label(format)

llabel(format,which_tics,side)

The short form of *label()* puts numeric labels on the outer side of each tic mark on the last axis drawn, where *format* is the same as the C routine *printf()* format strings. The long form, *llabel()*, labels each *'which_tic'* (i.e. a value of 2 would label every other tic) and *OUT* or *IN* for *side*.

rawlabel(ap)

rawllabel(ap,which_tics,side)

These routines allow the user to select the string to be printed at each label location. *ap* is an array of character pointers, where each points to a null terminated string of less than 16 characters. These strings are printed at the same location regular labels would be.

title(string)

ltitle(left,center,right,dir)

Writes a title for the last axis drawn. The short form centers the title. The values for *dir* are *TD_R0*, *TD_R180* for x axes, and *TD_R90* and *TD_R270* for y axes.

hgrid()

vgrid()

Draws a grid with horizontal (vertical) lines. This routine takes the spacing of the tics from the last axis drawn vertical (horizontal) direction as the spacing of the lines. The type of line can be set with *setlinetype()*.

print(string)

Prints the string at the current plot position. The string must be null-terminated and cannot contain newlines.

move(x,y)

Moves the current "pen" location to user-coordinates *x*, *y*.

draw(x,y)

Draws a line of the current line type from present location to user-coordinates *x*, *y*. Returns 1 if endpoint was outside the viewport. If line type is dot-plot, moves to point *x*, *y* and draws a dot.

plot(x,y)

Draws the current plot symbol at location *x*, *y* in user-coordinates. Returns 1 if *x*, *y* is outside the viewport.

seterrbars(xleft,xright,ylower,yupper,axlen,aylen)

Sets the size of the error bars to be plotted with *ploterr()*. The first 4 lengths are floating-point user-coordinate lengths, and the last 2 are integer plotter-coordinate lengths which control the cross bars on the error bars.

ploterr(x,y)

Same as *plot()* except error bars are drawn on the symbol or character. The lengths are set from the last *seterrbars()* call.

out_of_range

Not a subroutine, but a global variable that contains the number of times a *plot()*, *ploterr()*, or *draw()* endpoint was outside the viewport. It is reset by calls to *initplot()*, *endplot()*, and *setviewport()*, but can be set by the user at any time.

2. Plot Output Programs

2.1. Plot

This is a system program which expects coded plot commands as the input, either from a file or a pipe. It opens one of the plotters, accumulates and prints the plot.

"*plot*" with no options alternately tries to open the Calcomp and Versatec, making the plot on the first available device. "*plot*" uses raw disk areas to accumulate the plot, and waits if one of the four raw areas

is not available.

The options are:

-v make the plot program use the Versatec for plotting.

-c make the plot program use the Calcomp for plotting.

-s write the plot raster output on standard output instead of opening a plotter. (Someday may be useful for merging troff or other output with plot files.)

-o keep the plotter open between plots. Should only be done after plot has been accumulated in a file, and for plots which must be on consecutive pages, since it ties up the plotter.

-n causes program to ignore control-C's.

-4 shrinks each plot to one quarter of its original size and prints four plots per page.

2.2. Hpplot

This program needs to be run from an HP graphics terminal, or else you must log in on an HP, and then from another terminal redirect the standard output from your plotting program to that device. (Do a "*who am i*" on the HP to find out the name of the terminal, and the device will be "/dev/tty*x*" where "*x*" is the appropriate letter).

"*hpplot*" puts the HP in graphics mode and draws the plot as it reads the commands. When it is done with one plot (when it sees an "*endplot()*"), it beeps and waits for you to type any character on the HP keyboard. If there are more plots, it starts the next one, or else it resets the terminal back to its original mode.

3. Examples

3.1. A Scatter Plot

```
#include <plot.h>                       /* the plot header file */
#include <stdio.h>                      /* the standard i/o package */

main()
{
    int i;
    float x, y;
    char buffer[81], heading[50];

    initplot();                         /* initialize and begin */
    setviewport(115, 115, 2015, 1445);  /* leave some space for labels */
    setwindow(0.0, 0.0, 10.0, 10.0);    /* plot is scaled from 0-10 */
    xaxis(0.0, 0.0, 10.0, 5);           /* start at (0,0), 10 units long, 5 tics */
    label("%g");                        /* this format omits unneeded zeros */
    title("X axis");                    /* default is centered */
    yaxis(0.0, 0.0, 10.0, 10);          /* start at (0,0), 10 units long, 10 tics */
    label("%g");                        /* see the C manual for other formats */
    ltitle(" ","Y axis","no units",TD_R90); /* no lower title, rotated 90° */
    setlinetype(LT_DOTTED);             /* used dotted lines for gridding */
```

```
    hgrid();                                /* horizontal grid */
    vgrid();                                /* vertical grid */
    setlinetype(LT_SOLID);                  /* back to default */
    xaxis(0.0, 10.0, 10.0, 1);              /* close off top of box */
    yaxis(10.0, 0.0, 10.0, 1);              /* close off side - no tics */
    setlinetype(LT_DOTPLOT);                /* scatter plot */
    for(i=0; i<1000; i++) {                 /* generate 1000 random sets of points */
        x = rand() / 32767. * 10;           /* between 0 and 10 */
        y = rand() / 32767. * 10;
        draw(x,y);                          /* if linetype is dotplot, a draw makes
    }                                       /* a dot at the endpoint */
    setlinetype(LT_SOLID);                  /* back to regular plot */
    setplotchar('x');                       /* will use 'x' for character to plot */
    for(i=0; i<20; i++) {                   /* plot 20 random points */
        x = rand() / 32767. * 10;
        y = rand() / 32767. * 10;
        plot(x,y);                          /* plot point */
    }
    setextjust(TJ_CC);                      /* center text strings */
    setextsize(2);                          /* make text size larger */
    move(5.0, 10.5);                        /* move pen */
    fprintf(stderr,"Input title: ");        /* messages to the user must
                                               use stderr - not just printf() */
    gets(heading);                          /* get a string from standard input */
    sprintf(buffer,"Ex. 1: %s ", heading);  /* see C manual for
                                               more explanation of sprintf() */
    print(buffer);                          /* this is the plot routine */
                                            /* careful of print() and printf() */
                                            /* and no newline in string to be printed */
    endplot();                              /* flushes buffers, stops accumulation */
}
```

3.2. A Log Plot

```
#include <plot.h>                   /* the plot header file */

main()
{
    long x;
    float y;
    extern double log();            /* library subroutines must be declared */
                                    /* if they return non-integer values */
    double log10 = log(10.);        /* save this as a constant */

    initplot();                     /* initialize and begin */
    setviewport(115,115,2015,1445); /* actual area for plotting */
    setwindow(0.0,-1.0,200.0,5.0);  /* semi-log plot */
    xaxis(0.0,-1.0,200.0,5);        /* this is the linear axis */
    label("%g");
    logyaxis(0.0,-1.0,6.0,6);       /* use log values for axis */
    setextdir(TD_R90);              /* set direction for labels */
    label("10e%g");                 /* this makes good log labels */
    title("Ex. 2: Log Plot 1");     /* just 1 centered title */
```

```
        move(0.0,0.0);                  /* move to origin */
        for(x=0;x<200;x+=10) {          /* draw a curve */
            y = x * x + x;              /* y = x^2 + x */
            y = log(y) / log10;         /* convert it to base 10 */
            draw((float)x, y);          /* see casts in the C manual */
        }                               /* arguments must be floats */
        endplot();                      /* end of plot 1 */

        xaxis(0.0,-1.0,200.0,5);        /* same viewport and window */
                                        /* no initplot() needed for next plot */
        label("%g");
        logyaxis(0.0,-1.0,6.0,6);       /* use log values for axis */
        setextdir(TD_R90);              /* set direction for labels */
        label("10e%g");                 /* this makes good log labels */
        title("Ex. 2: Log Plot 2");     /* just 1 centered title */
        setplotchar(C_SLD_CIRCLE);      /* set character to solid circle */
        seterrbars(0.0,0.0,0.15,0.15,8,8); /* set length of err bars */
        move(0.0,0.0);                  /* move to origin */
        for(x=0;x<200;x+=10) {          /* draw a curve */
            y = x * x + x;              /* y = x^2 + x */
            y = log(y) / log10;         /* convert it to base 10 */
            draw((float)x, y);          /* see casts in the C manual */
            ploterr((float)x, y);       /* draw plot symbol with error bars */
        }
        endplot();                      /* end of plot 2 */
}
```

3.3. Multiple Plots on a Page

```
#include <plot.h>                       /* the plot header file */
#define MAX     60                      /* maximum number of histogram bins */

main()
{
    int i, lasti, bin[MAX+1];
    float x, mid, gauss();

    for(i=0;i<1000;i++) {               /* generate a random distribution */
        do {
        x = gauss();                    /* gaussian distribution */
        x = (x+5) * 6;                  /* spread and center around 30 */
        } while (x<0 || x>60);          /* try again if out of range */
        bin[(int)x] ++;                 /* increment the corresponding bin */
    }
    initplot();                         /* initialize */
    setviewport(200,120,950,1480);      /* about half the page */
    setwindow(0.0,0.0,60.0,100.0);      /* user coordinates */
    xaxis(0.0,0.0,60.0,6);
    label("%g");
    title("this is the X axis");
    yaxis(0.0,0.0,100.0,5);
    label("%g");
    title("this is the Y axis");
```

```
        setextsize(2);                          /* bigger size */
        title("Ex. 3: Gauss 1");                /* titles the last axis drawn */
        setextsize(1);                          /* back to normal */
        xaxis(0.0,100.0,60.0,6);                /* close off box */
        yaxis(60.0,0.0,100.0,5);
        move(0.0,0.0);                          /* go to origin */
        lasti = 0;
        for(i=0;i<MAX+1;i+=5) {                 /* do every 5th bin */
            mid = (i + lasti) / 2.;             /* get midpoint */
            draw(mid,(float)bin[lasti]);        /* horizontal line */
            draw(mid,(float)bin[i]);            /* vertical line */
            draw((float)i, (float)bin[i]);      /* horizontal again */
            lasti = i;                          /* save for next time */
        }
        initplot();                             /* reinitialize with no page feed */
        setviewport(1260,120,2010,1480);        /* second half of page */
        setwindow(0.0,0.0,60.0,100.0);          /* user coordinates */
        xaxis(0.0,0.0,60.0,6);
        label("%g");
        title("this is the X axis");
        yaxis(0.0,0.0,100.0,5);
        label("%g");                            /* this has to be a floating format */
        title("this is the Y axis");
        setextsize(2);                          /* bigger size */
        title("Ex. 3: Gauss 2");                /* titles the last axis drawn */
        setextsize(1);                          /* back to normal */
        xaxis(0.0,100.0,60.0,6);                /* close off box */
        yaxis(60.0,0.0,100.0,5);
        move(0.0,0.0);                          /* go to origin */
        lasti = 0;
        for(i=0;i<MAX+1;i++) {                  /* do every bin this time */
            mid = (i + lasti) / 2.;             /* get midpoint */
            draw(mid,(float)bin[lasti]);        /* horizontal line */
            draw(mid,(float)bin[i]);            /* vertical line */
            draw((float)i, (float)bin[i]);      /* horizontal again */
            lasti = i;                          /* save for next time */
        }
        endplot();
}

float
gauss()
{
        double u, v, s, t;
        extern double log(), sqrt(), ranf();
        do {
            u = 2.*ranf()-1.;
            v = 2.*ranf()-1.;
            s = u*u + v*v;
            } while (s >= 1.);
        t = -2.*log(s)/s;
        t = sqrt(t);
        return(u*t);                            /* v*t is also gaussian */
}
```

3.4. Compiling a User Program

```
$ cc yourprog.c -lP  -o yourprog
```
(compiles your program with library "P" and puts the output in "yourprog")

3.5. Running the Program

```
$ yourprog | plot
```
(sends your output directly to the first available plotter)

```
$ yourprog | plot -4
```
(prints four plots per page)

```
$ yourprog > savfil
```
(saves your output in "savfil" and does no plotting)

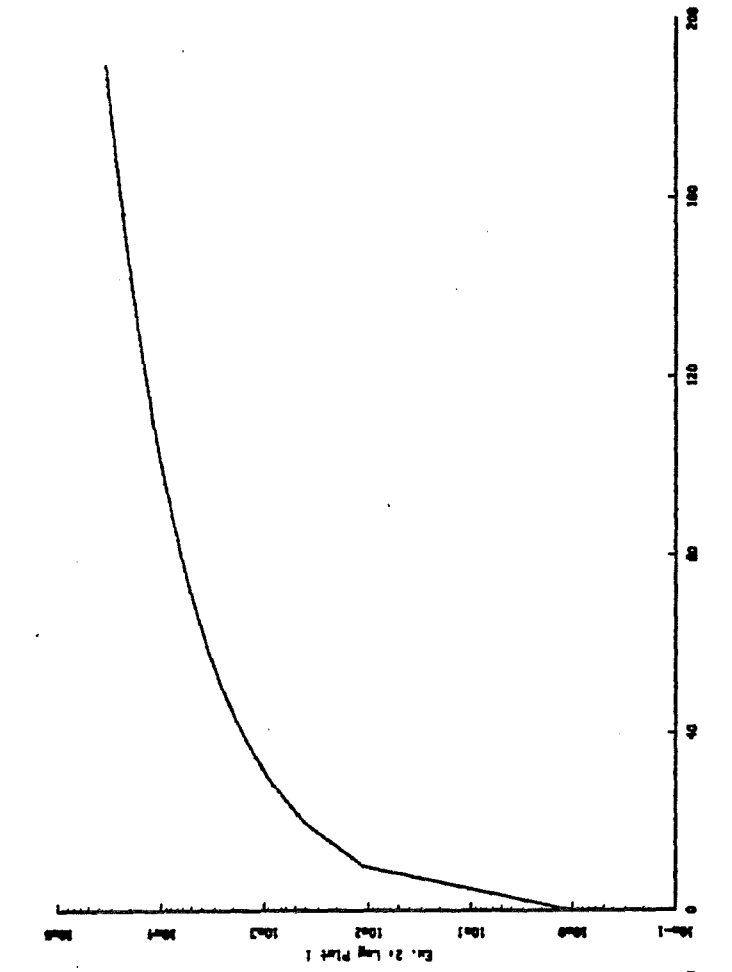
```
$ plot -c4 < savfil
```
(plots the already created output on the Calcomp, four to a page)

```
$ yourprog | tee savfil | plot
```
(saves a copy of the output in "savfil" and
also sends a copy to be plotted)

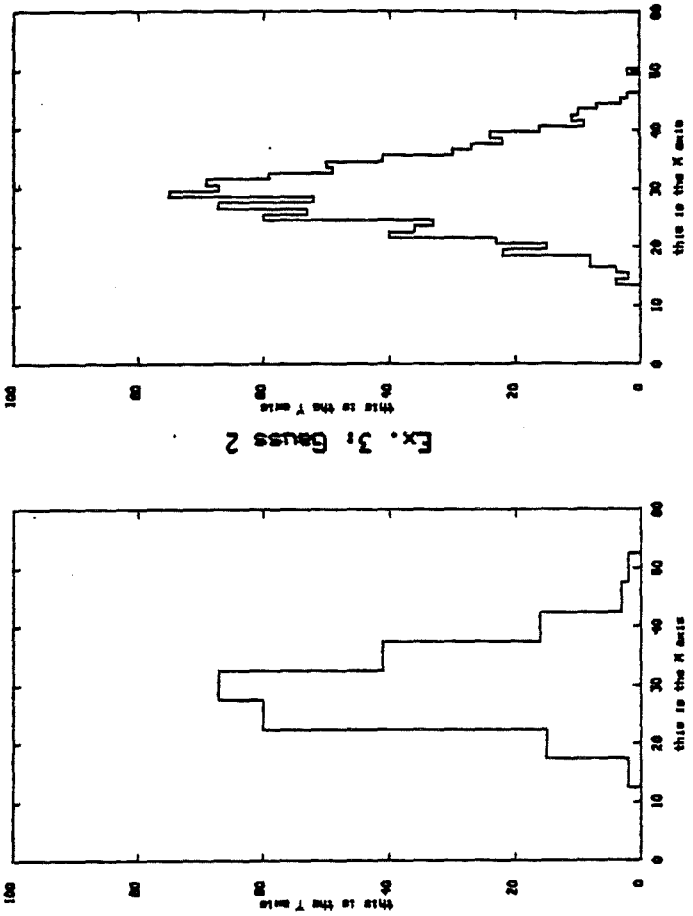
```
$ yourprog | hpplot
```
(if the terminal is a Hewlett-Packard, clears
the screen and draws the plot in graphics mode)

```
$ yourprog | hpplot > /dev/ttym
```
(you must have logged in on "ttym" - it must be an HP - and
then you can run this from any terminal)

```
$ yourprog | tee savfil | hpplot
```
(prints on the HP as the plot is accumulating, and also saves a
copy in "savfil" for later hardcopy printing)

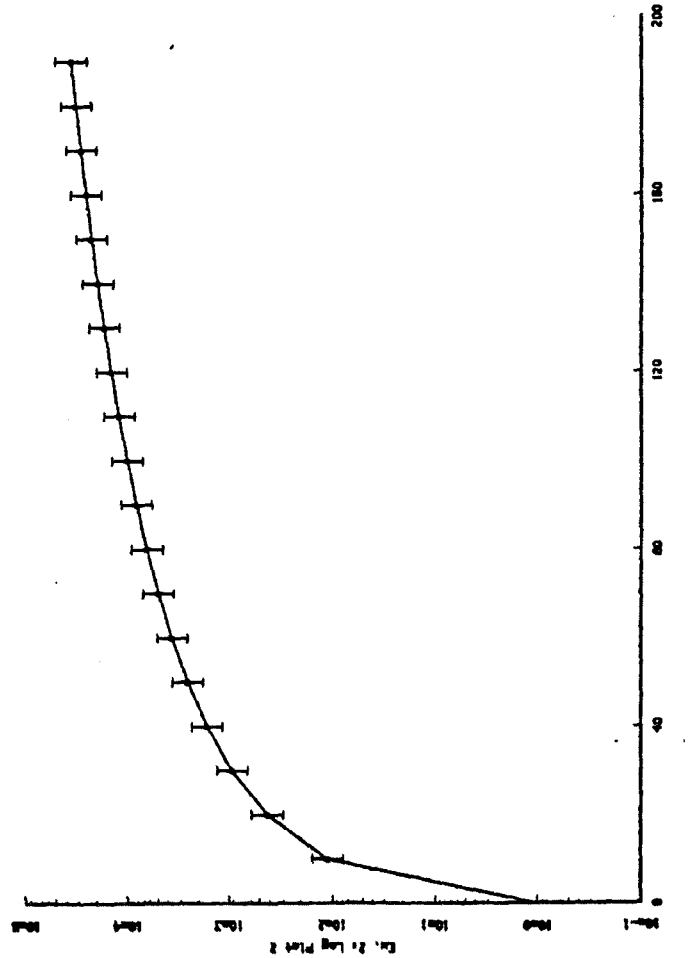


Ex. 1: Scatter Plot



Ex. 3: Gauss 2

Ex. 3: Gauss 1



Appendix A:
Plot Multiplexing

Don Mitchell

Calif Institute of Technology
Pasadena, Ca

ABSTRACT

Multiplexing plots allows a program to accumulate many separate plots during a single pass through the data. All plots should have the same windowing transformation (same viewport and same window), since these routines simply mark the output stream for later separation; no variables are saved or calculations are done.

1. Introduction

Multiplexing plots allows the user to accumulate and print up to 256 plots during one pass through the data. This is especially useful for sequential-access devices such as magnetic tape, where the rewind and rereading time are substantial.

The multiplexing fits well into the existing plot package. Even though many plots may be accumulated at one time, there is still only one output stream containing all the information. The multiplexing routines insert extra characters into the stream for each plot channel change. The output stream must be piped into a demultiplexer which separates the output and feeds the plot program with normal plot input.

There are several things to be aware of when multiplexing plots. The scaling of all points is done in the user program by the library subroutines, and several global variables are saved by the routines. When changing plots, none of those variables or the windowing transform are changed. If all plots have the same window and viewport, points can be generated interchangeably and printed on different plots. If the scaling is not the same on all plots, everything must be reinitialized when changing plot channels (the window, viewport, linetype, textsize, etc). For moving and drawing lines, the last position of the "pen" is saved. If you move to a position but then change plot channel, that position will not be saved for the next time you change back to that plot. To draw continuous lines on separate plots, the user must save the position of the pen on that particular plot himself, and move there before he draws the line.

2. Library Subroutines

There are two new plot library subroutines to be called from the user program as the plots are being generated. They are in the standard plot library (-lP) and function in the same manner as all the other library routines.

channel(an)

Directs the plot output stream to channel *an*, where *an* is in the range 0-255. All subsequent output will go to this channel until the next *channel()* call. Each separate plot should be collected on a different channel.

broadcast(an,am)

This allows the output stream to be duplicated on a range of channels, instead of just one. The regular plot output will be duplicated on channels *an* through *am* inclusive until *channel()* or *broadcast()* is called again.

3. Demux Program

The coded plot commands (the standard output from the user plot routines) must be piped through the program "*demux*" before they are sent to "*plot*" for plotting. "*demux*" is the plot demultiplexer - it sorts out the stream of plot commands and prints each plot sequentially. "*demux*" reads standard input and writes on standard output, so input should be piped or redirected from a file. The output can be piped to "*plot*" directly or saved in another file for later plotting.

4. Examples

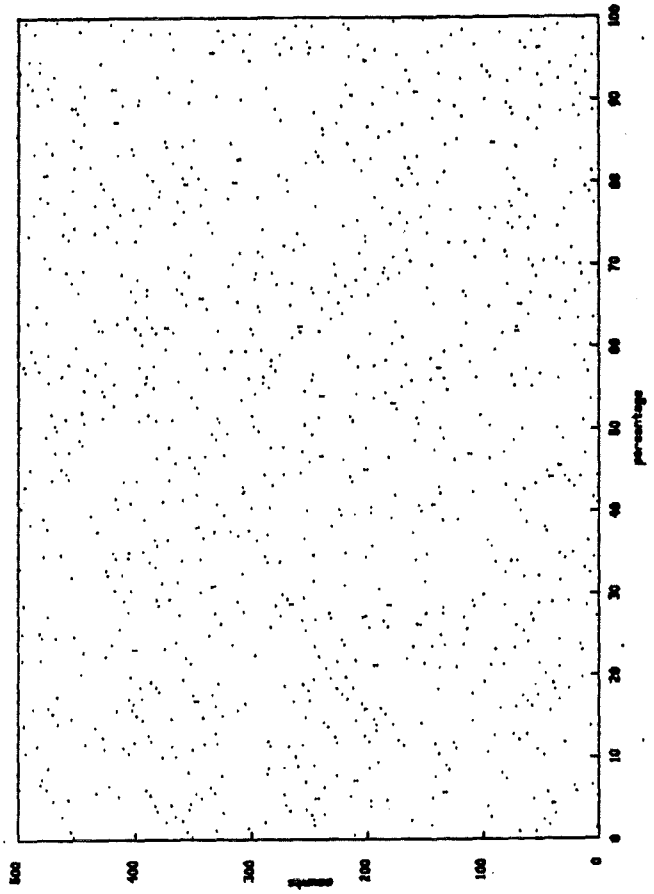
4.1. A Plot Program

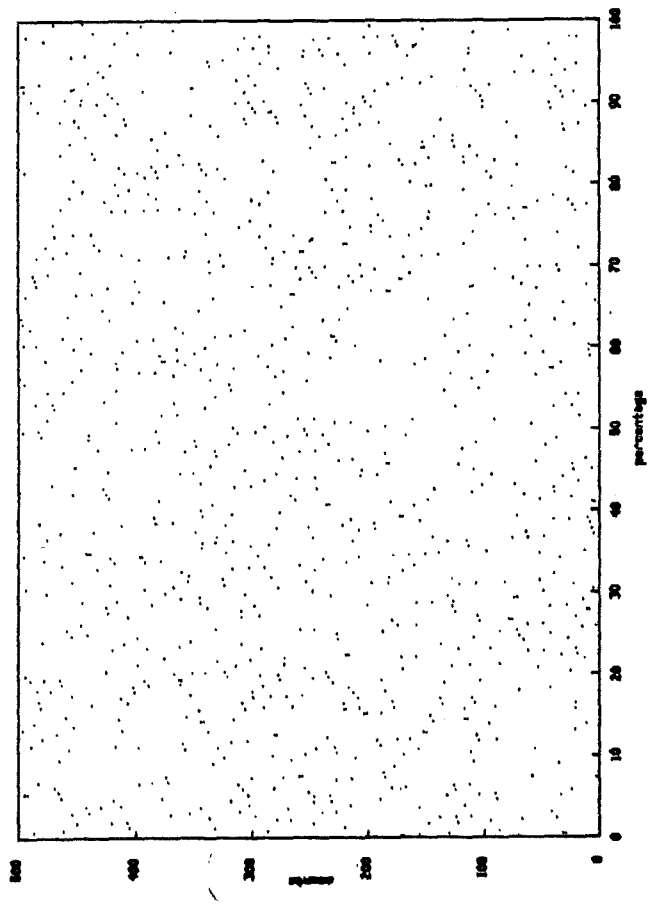
```
#include <plot.h>
#include <stdio.h>

main()
{
    int i, p;
    char buf[20];
    float x,y;

    initplot();                          /* initialize */
    setviewport(150,150,1900,1400);      /* all should have same viewport */
    setwindow(0.0,0.0,100.0,500.0);      /*  and same window */
    broadcast(0,2);                      /* send this to all 3 channels */
    xaxis(0.0,0.0,100.0,10);             /* draw an axis */
    label("%g");                         /* label tic marks */
    title("percentage");                 /* title the axis */
    yaxis(0.0,0.0,500.0,5);              /* draw y axis */
    label("%g");                         /* label tic marks */
    title("counts");                     /* title y axis */
    xaxis(0.0,500.0,100.0,1);            /* no tic marks */
    yaxis(100.0,0.0,500.0,1);            /* close off box */
    setextjust(TJ_CC);                   /* center printing */
```

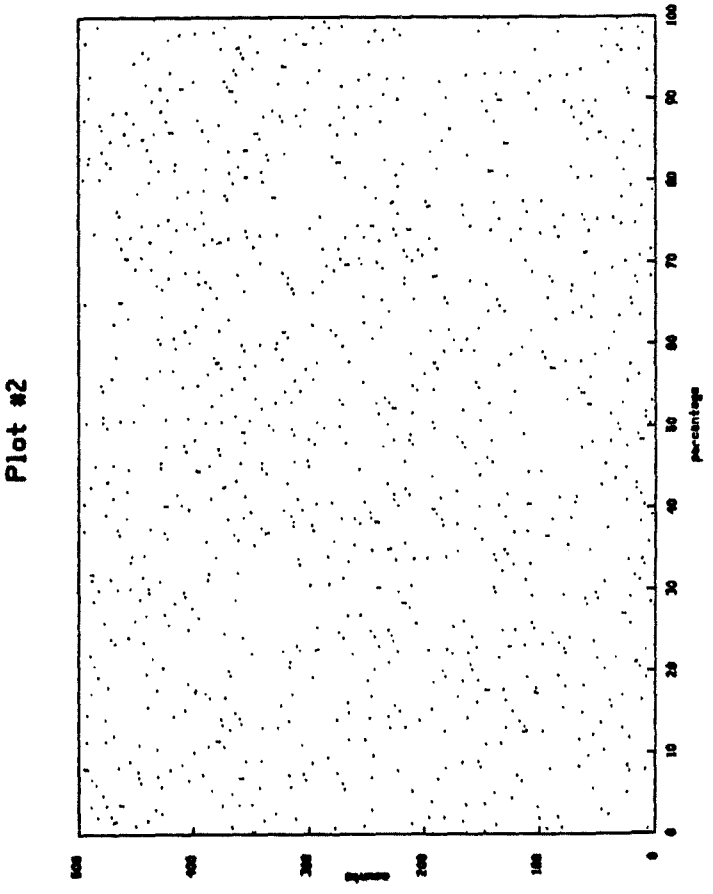
```
    setextsize(2);                  /* big for heading */
    setlinetype(TL_DOTPLOT);        /* all will be scatteplots */

    for(p=0;p<3;p++) {
        channel(p);                 /* change plots */
        move(50.0,550.0);           /* move up for title */
        sprintf(buf,"Plot #%d", i); /* make unique labels */
        print(buf);                 /* label each plot with it's number */
    }
    for(i=0;i<3000;i++) {
        p = rand()/32768. * 3;      /* select a random channel */
        channel(p);
        x = rand()/32767. * 100.;   /* random x and y */
        y = rand()/32767. * 500.;
        draw(x,y);                  /* plot a random point */
    }
    broadcast(0,2);                 /* all channels */
    endplot();                      /* must be called for each plot */
}
```

4.2. Running the program

```
$ yourprog | demux | plot

$ yourprog > savefile
$ demux < savefile | plot

$ demux < savefile > savefile2
$ plot < savefile2
```


Plot #1
500
400
300
200
100
0
10
20
30
40
50
60
70
80
90
100
percentage


Plot #0
500
400
300
200
100
0
10
20
30
40
50
60
70
80
90
100
percentage


Plot #2
500
400
300
200
100
0
10
20
30
40
50
60
70
80
90
100
percentage