Introduction to `plotdata` for physics students

Edward Sternin

January, 1994

Contents

1 What is `plotdata`?

`plotdata` is the name of a computer program that you will be using in many of your courses in Physics. It is difficult to find a successful physicist these days who does not use a computer in one way or another. The computer has become an indispensable tool: to be successful in many fields of human endeavor today not only you need to know the field, you also need to know how to apply computer tools to it efficiently. Physics is no exception, and `plotdata` is one such tool. You will have to spend some time learning how to use it but in the long run it will save you time and effort, and you will become more efficient in doing your assignments and lab reports.

As the name implies, `plotdata` excels at **plot**ting **data**. We will get to a simple example in a moment, when you will analyze the results of some measurement and generate a figure for a report or publication. `plotdata` is versatile enough to be used in other contexts as well, such as when you want to plot an analytical expression, or prepare a transparency or a poster with large–size text or mathematical formulas. In addition, `plotdata` has its own macro language, a way of grouping commands into "programs" and then executing (running) those programs with a single `plotdata` command. The syntax of this language is similar to Fortran, so it should prove familiar to most of you.

2 What am I supposed to know?

It is assumed that you:

- have an account on `spartan`, one of the central servers at Brock;
- know a little bit about Unix, its file and command structure, and some editor like `vi` or `pico`;
- can use a mouse, are familiar with the concept of windows, and with the process of clicking and dragging to manipulate windows (closing, resizing, *etc.*).

If you feel that the above does not apply to you, let your instructor know **now**. Otherwise, you may want to log on right away. You can then run `plotdata` and try different commands as we go along.

Briefly about the notation used in this document: `this font` is used to indicate a "computer word", *i.e.* a prompt or a message from a program, a file name, or a command you need to type in. In the latter case, a carriage return is implied at the end of every line of input.

One note of caution: Unix is fussy and distinguishes between lower and UPPER case letters, while `plotdata` often uses all capitals to name its special files. For example, `PLOTDATA.JOURNAL` is a log file of your session with a record of every `plotdata` command you typed in. If you want to examine or print that file, you need to refer to it in all uppercase, as in `vi PLOTDATA.JOURNAL`; trying to `vi plotdata.journal` will not work.

3 Hardware

plotdata runs on various Unix platforms, including the SGI/Irix that is spartan, and also on VAX/VMS platforms (such as csvax.cosc). We will assume you are using spartan. There are several things that need to be set properly (see Appendix A), but the system manager should have taken care of it, so to run plotdata all you need is to type

```
spartan % plotdata
```

at the Unix prompt (shown as spartan %).

plotdata is too big and complex to run on a personal computer, whether a Mac or an IBM PC compatible. It runs on a powerful host computer (spartan) and you access it through the so–called "X–terminal" communications protocol. This protocol of e**X**changing the information with a remote **terminal** was invented at MIT, and has been implemented for many different hardware platforms.

For example, in B203 (the lab across the hallway from the Physics Office, where you have your 2P20 labs) you will find several hardware X–terminals, powerful dedicated machines with high–resolution monitors and the ability to manage multiple windows connected to several different hosts. Using one of these X–terminals will give you the best performance.

In F317, the PCs must pretend to be like X–terminals; they do that by running an "X–terminal emulation" program called eXceed. To activate it, select the X-terminal entry under the blue start–up menu.

Additional PCs are available for use by Physics students in H300 when they are not required for the 2P31 or 3P92 labs. Instead of the blue menu, you activate eXceed by double–clicking on its icon, a big red **X**. You must do it from the MS Windows environment; if the PC you are using is in plain DOS mode of operation as indicated by the prompt such as D:\>, simply say win to start MS Windows first, and then double–click on the eXceed icon.

There are other ways to connect from a local PC to a remote host, and some of them may support the kind of graphics–intensive interaction that plotdata requires, but for the sake of simplicity we will stick to what is appropriate for X–terminals. Consult the manuals and/or your instructor about other options.

4 A simple example

When you activate plotdata a welcome message followed by the first PLOTDATA: prompt is printed in the text window, and another window pops up, the one in which plotdata will display the graphics. Only one window can be active at any given time, so you may need to move the mouse inside the window that you want to "connect" your keyboard to. You may want to resize and/or move the text window around so that you can see both at the same time. Do not resize or close the graphics output window; it will get closed automatically when you exit plotdata.

Let's assume you have started plotdata successfully. At PLOTDATA: prompt type the following:

```
PLOTDATA: x={1,2,3,4,5,6,7,8,9}
PLOTDATA: y={0.05,0.10,0.14,0.19,0.25,0.30,0.34,0.40,0.45}
PLOTDATA: graph x,y
```

Your graphics window should be showing the graph of `y` *vs.* `x`! The above data could have been something like time and distance from one of your air track experiments. In fact, we should have also entered the error bars at each data point, like this:

```
PLOTDATA: dy={0.02,0.07,0.01,0.04,0.05,0.10,0.02,0.04,0.04}
PLOTDATA: clear
PLOTDATA: graph x,y,dy
```

We have used `clear` to clear the graph and start a new one with the next `graph` command.

You can immediately see that the dependence is roughly linear. Let us investigate closer. First of all, the line connecting the points is really inappropriate, since we only made a discrete set of measurements and really know nothing about what `y(x)` is like in between our data points. To fix that, we will change one of the settings of `plotdata`, that of the plotting character, or `pchar`.

```
PLOTDATA: pchar -10
PLOTDATA: clear
PLOTDATA: graph x,y,dy
```

Now each data point is indicated by a triangle (the plotting character No. 10), and there is no line connecting the data points (the minus sign in front of 10).

Now we are free to use a line to indicate a theoretical model that fits this data. From the way we generated our data (air track) we expect a linear dependence. Let's fit the data to a linear expression:

```
PLOTDATA: vary a
PLOTDATA: fit y=a*x
```

The output generated by `plotdata`, among other things, tells us that the best value for the parameter `a` is 0.0495 ± 0.0039. The error in the fit is given by the standard deviation (the `E2` parameter), and there are some other useful numbers reported which we will ignore for now.

Armed with this best fit, we can add a "theoretical" line to our experimental plot. We can recalculate what the values should have been if the experiment yielded exactly the linear dependence we expect, by "updating" a new vector, `f`, to contain the values that the last `fit` command had calculated:

```
PLOTDATA: update f
PLOTDATA: pchar 0
PLOTDATA: graph/noaxes x,f
```

Two points of interest here: we used `pchar 0` to turn the plotting character to none and the line through the points to on, and then we used a switch (`/noaxes`) on the `graph` command to add the second graph to the already existing one. Looks pretty cool.

To add the final touches, let's label things properly:

```
PLOTDATA: xlabel "Time, s"
```

```
PLOTDATA: ylabel "Distance, m"
PLOTDATA: replot
```

Figure 1 shows what your plot should look like at this point.

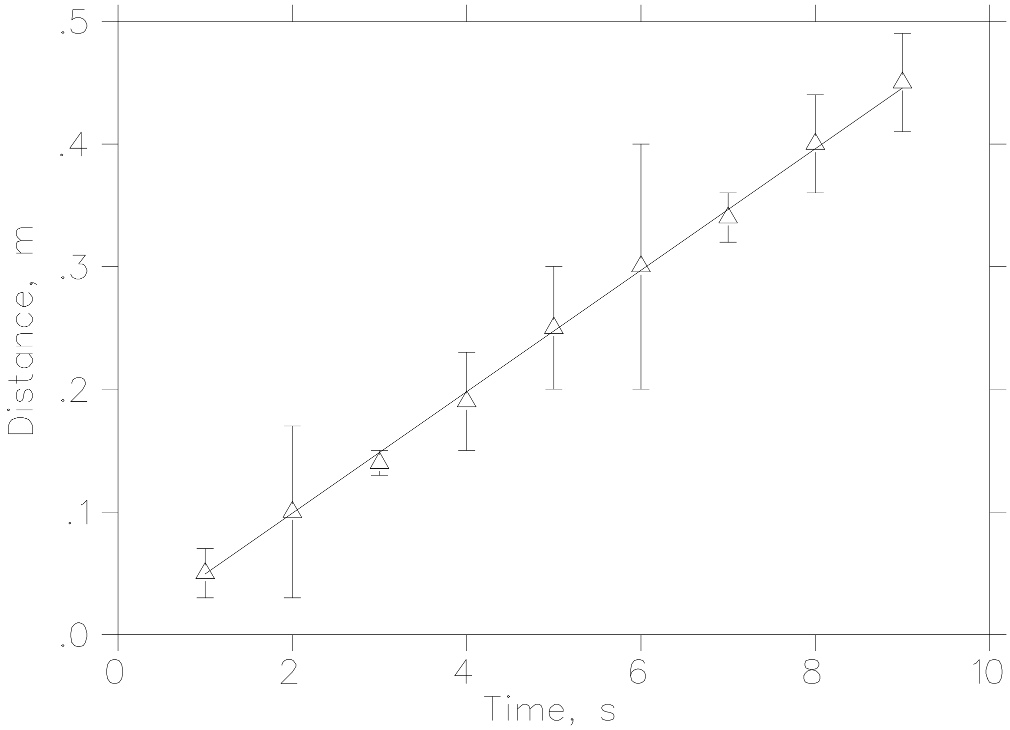

Figure 1: **A simple example.** Plotting data points with error bars. The fit to a linear equation is shown as a solid line.

Before we advance on to more elaborate things you must learn how to end your `plotdata` session. The magic word is `quit`. Try it now.

One remark: your graph may look slightly different. In creating Figure 1 I had doubled the size of the plotting character to 2% of the page width, up from the default 1%, using the command:

```
PLOTDATA: set %charsz 2
```

before the `graph` command. It's one of those slightly more advanced commands that we will get to soon.

4.1 Reading data into `plotdata`

When you have quit `plotdata` it "forgot" what vectors `x, y, dy` were, and if you want to look at them again for some reason, you need to re-type them. A more convenient approach would have been to enter your data into a file using your favourite editor first, and then to read that file into `plotdata` at will. In fact, you may already have created this file from the previous tutorial session, when you practiced using `vi` or `pico`. Otherwise, create this file (call it `file.dat`) now, using your favourite editor outside of `plotdata`. You may want to open a separate shell window for this purpose, putting your `plotdata` session aside for a moment. Enter the following data into this data file:

```
1.0  0.05  0.02
2.0  0.10  0.07
3.0  0.14  0.01
4.0  0.19  0.04
5.0  0.25  0.05
6.0  0.30  0.10
7.0  0.34  0.02
8.0  0.40  0.04
9.0  0.45  0.04
```

i.e. the same data as we used before, now entered as three columns.

Start `plotdata` again, or move your mouse to the `plotdata` window, and use a single `read` command to create all three vectors of our simple example:

```
PLOTDATA: read file.dat x,y,dy
PLOTDATA: list x,y,dy
```

Use the `list` command to verify that the data has been read in correctly. The rest of the commands required to generate the plot of Figure 1 remains exactly the same.

4.2 Programming with `plotdata` macros

When you `quit`, `plotdata` also forgot all of its settings, not just the data values, so you had to re-enter the label strings and all other commands. In general, computers are very good at repetitive tasks, and one should never have to type the same thing twice! In `plotdata` you can create "programs" of `plotdata` commands called macros.

For example, `quit` or move to another window, and use your favourite editor to create a plain text file named `GRAPH2D.PCM`, with the following lines in it:

```
read file.dat x,y,dy
vary a
fit y=a*x
upd f
xlab "Time, s"
ylab "Distance, m"
clear
pch -10
gra x,y,dy
pch 0
gra/noax x,f
```

These are exactly the same commands that we have used before, except we have abbreviated a few of them (non–ambiguous abbreviations are allowed in `plotdata`). When you re-start `plotdata` now, all you need to do to repeat the plot of Figure 1 is:

```
PLOTDATA: execute graph2d
```

Since we have used the default file extension in naming the macro (`.PCM`, note the capitalization), we do not need to type it in, `plotdata` supplies it automatically. The above line "executes" a macro contained in the file `graph2d.PCM` with all the right commands.

To analyze another set of data from a similar experiment you would have to edit the macro file `graph2d.PCM`, replace the file name `file.dat` with a new one, and then repeat the above. Actually, there is a way to build a "universal" macro! Change the first line in your macro to `read ?1 x,y,dy`. By specifying `?1` in place of the file name you are telling `plotdata` to get the file name from you at the time of execution of the macro, like this:

```
PLOTDATA: execute graph2d file.dat
```

meaning "read the data from the file whose name is contained in the parameter No. 1 (`?1`) on the command line into vectors `x, y`, and `dy`". And the name we supply on the above line is the same `file.dat` as before. For another set of data you could generate the graph with a single `plotdata` command, by simply specifying a different filename instead of `file.dat`!

Test your understanding of the above macro by modifying it to allow passing the plotting character code as the **second** parameter. Within the macro you would then refer to it as `?2` and from `plotdata` you would call it up like this:

```
PLOTDATA: exe graph2d file.dat 10
```

to use triangles again (note the abbreviation of the word `execute`); or

```
PLOTDATA: @graph2d file.dat 4
```

to use crosses (for example) at the data points. The symbol `@` is another shorthand for `execute`.

4.3 Getting help

Which `pchar` numbers generate which plotting characters? In general, there are two ways to get an answer to a question like this: issue the command `help`, or `help graph` if you want help on `graph` (also try `help introduction`!); or to open a hardcopy manual (several copies should be available from your instructor).

The only "exceptions" to this rule are the five things covered by the `display` command: fonts, special characters, hatch patterns, line types, and, yes, the plotting characters. For example, to get a table of possible `pchar` symbols and their codes on your screen you would use `display pchar` rather than `help pchar`; the latter simply gives you a brief description of the syntax of the `pchar` command but not the table of codes.

There are two manuals for `plotdata`: one is the *User's Guide* which takes the same tutorial approach that we have been using here, in much more detail, of course; it guides you by showing many examples. Once you have read the introductory chapters, the best way to use it is to flip through its pages until you find an example that looks like something that you have in mind, and use the commands of that example as an inspiration for generating your own graphs. The other is the *Reference Manual* which is the detailed description of all commands and their options.

Returning back to our original question: as an alternative to `display pchar` you can look up the table of `pchar` symbols in the *Reference Manual* on page 140.

4.4 Hardcopy

Finally, you are happy with the way your graph looks, so you want to print it out on paper. Many types of hardcopy devices are supported within `plotdata`, and the one you will be using is the `postscript` type. PostScript is the name of a "page description language" that is understood by many high–quality printers out there, including the ones at Brock. The printers are known to `plotdata` by their Unix names, and the ones you need to know about are:

`bj-physics` the printer of choice, this is the BubbleJet printer in B203. Your output is available for pickup whenever B203 is open, which is most of the time. At the moment, Department of Physics is subsidizing the cost of course–related printing on this printer; please, do not abuse this privilege as this will result in an immediate termination of this policy.

`ps` this is the central printer of the University's Computing Services. It is located in F301 and your output is available for pickup during the normal working hours.

`psd` the same printer! — except when addressed by this name, the multipage output sent to it gets printed in **duplex** mode, *i.e.* on both sides of the page.

`lw-physics` this is the printer in the Physics Office. Prior to printing there, please obtain secretary's permission so that your print jobs do not interfere with her work. Again, locked after hours.

`f316` as the name suggests, this printer is located in F316, and is only convenient when you are next door, in F317. Since some evening classes use those rooms, there usually is a TA on duty to give you your output.

With the exception of `bj-physics`, the printed output costs $0.10 per page, which is collected when you pick up your output.

Once you know the name of the printer you want to use, get the hardcopy output by following this example:

```
PLOTDATA: hardcopy
 HPL   HP Laserjet
 PS    PostScript
Enter:  device code >> ps
Device=PostScript
 HARDCOPY COMMANDS
 P   queue  | print
 S   file   | save file
```

```
Enter:  hardcopy command >> p bj-physics
```

The only things you have to type here are `hardcopy`, the command; `ps`, the printer type; and `p bj-physics`, the command to print to a given printer, `bj-physics` in this example. The rest of the text shown above is supplied by `plotdata` to prompt you.

As you can tell, you could also save the output to a file for printing later or for including it in your lab report, by choosing `s FILE.PS` at the last prompt.

5 Generating your own data

Instead of plotting experimental data, physicists often like to "doodle" with mathematical expressions. Sometimes it's an attempt to visualize a solution to some differential equation, to see if it makes sense; or it's a first step in finding the nulls of a complicated polynomial; possibly it's a quick way to look up a family of standard mathematical functions. Programs like `maple` or `mathematica` do these things extremely well, but in many cases `plotdata` has sufficient capabilities, and is much faster.

As an example, execute the following commands:

```
PLOTDATA: generate x 0,0.1,10
PLOTDATA: y=x**2*exp(-x)
PLOTDATA: graph x,y
```

and you can see right away that the function $y(x) = x^2 e^{-x}$ has a maximum value of about 0.55 near $x = 2$. The vector `x` got generated from 0 to 10 in steps of 0.1, then we calculated `y(x)` using the standard Fortran syntax, and plotted `y` *vs.* `x` in the usual way. Look for the zeroes of the first derivative, in this way:

```
PLOTDATA: set lintyp 9
PLOTDATA: derivative x y
PLOTDATA: replot
```

Both the function and its derivative show up on the same plot, as in Figure 2. For the derivative, we set the line type (`set lintyp`) to something other than the default value of 0, the solid line type (use `display lines` to see what line types are available).

Numerical integration, differentiation, spline interpolation, a large library of special functions (would you like to know what the fourth–order Laguerre polynomial looks like? – try `y=laguerre(4,x)`) — these are all things that `plotdata` does with ease. It is pointless to try to provide you with a comprehensive list of its capabilities, this is what the manuals are for. Remember, you don't have to know all of `plotdata`'s commands, just get the feel for where things are in the manual, to be able to look them up quickly when needed. For example, the special functions known to `plotdata` are listed in Appendix C of the *User's Manual*.

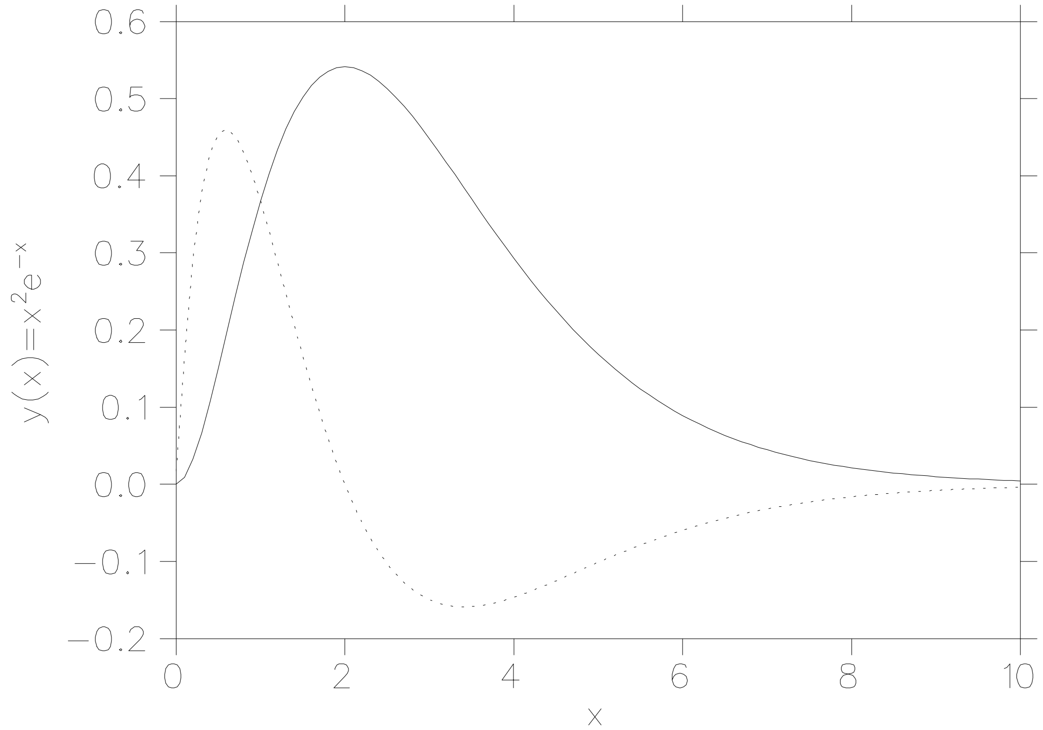

Figure 2: **A physicist's "doodle".** We plot here $y(x) = x^2 e^{-x}$ and its first derivative (the dotted line).

6 Changing the default settings

You may not have realized while going through our simple example that every detail of the graphs that you were creating could be modified. That's because `plotdata` provides a pretty reasonable set of default settings for things like line widths, character sizes, layout of the graph on the page, *etc.* However, should you be unhappy with any of these defaults, `plotdata` allows you to change them. The general method of changing something is the command

```
PLOTDATA: set parameter value
```

which has over a hundred different parameters that specify every minute detail of the appearance of the graph, from the frame placement on the page, to the angle of the axis tics. All parameters have names that are no more than six–character long (Fortran experts will recognize the origin of this convention), and an attempt has been made to keep the names logically related. All parameters that are integers begin with `n`; a set of parameters pertaining to x–axis is identical to the ones for the y–axis with letter `y` in place of `x`, *etc.* For example, `nlxinc` and `nlyinc` are the `n`umbers of `l`arge `x` or `y`–axis `inc`rements, *i.e.* the major divisions of the appropriate axis. Also, the way of modifying all parameters is exactly the same: if you specify `parameter` but not the new `value`, the current value is reported, you are prompted for the new one, and entering a carriage return alone retains the old value. In addition, all parameters that define length or size of something can be specified in graph units or in percent of the page size; you choose the latter by prepending `%` to the `parameter` name.

Some useful examples:

`set %xlaxis 18` Modifies the placement of the lower end of the x–axis on the page, here at 18% of the total page width from the left edge. The default value is 15%, and if you have a complex axis' label with subscripts and superscripts you may want to leave a bit more room for it.

Other parameters in this "family": `xuaxis`, `ylaxis`, and `yuaxis`.

`set nlxinc 5` Specifies the number of the `l`arge (major) divisions of the `x`–axis. See also `nsxinc`, the number of `s`mall (secondary) tics inside each major division; also `nlyinc` and `nsyinc`. The default values depend on the range of the data.

`set %charsz 2` Modifies the size of the plotting character (`pchar`). Similarly, `xnumsz` and `ynumsz` refer to the size of the numbers at the tics of the axes; `xlabsz` and `ylabsz` to the size of the text of axis labels; `txthit` to the height of general strings of text that can be placed at arbitrary locations on the graph using the command `text`.

`set xtica 90` and `set ytica -90` will "flip" the axis tics to the inward direction, opposite to the default outward one. The angle is specified relative to the direction of the respective axis, and need not be $\pm 90°$. Similarly, the angle of the `text`'s text string is determined by `txtang`.

`set lintyp 9` Selects the line type for the next `graph` command. There are 10 predefined types (use `display lines` to see what they are) and user–defined line types can be created.

`set box 0` Turns off the axis box. Many parameters have only two values, on (=1) and off (=0). The most frequently used ones are: `xaxis` and `yaxis` which control whether the appropriate axis is drawn at all; `xcross` and `ycross` which force the axis zero–crossing to be shown on the graph; there are many others.

`set menu` Presents the list of all possible parameter names and their current values. The list is long and wide; it may wrap lines on your screen and scroll a couple of pages by, but it's all there!

One comment: several `set` commands can be combined together, like this:

```
PLOTDATA: set
Enter: name { value } >> xlaxis 18
Enter: name { value } >> xuaxis 95
Enter: name { value } >> box 0
Enter: name { value } >> lintyp 7
Enter: name { value } >>
PLOTDATA:
```

with a blank line at the end terminating the current `set` command. In a macro file, of course, you would not enter the text of the prompt `Enter: ... >>`, but a blank line to terminate a multiple `set` command would still be needed.

There are a few commands that affect the `settings`, but are important enough to deserve their own `plotdata` command. We have already seen `pchar`; in a similar category is `scales` command which allows you to specify explicitly the x and y–axis scales and number of divisions on each (`plotdata` defaults to scales that include all of the data, and determines an appropriate number of divisions automatically; most of the time it does a pretty good job). In principle, you could achieve the effects of a single `scales` command with a series of `set` commands, explicitly changing such parameters as `xauto`, `xmin`, `xmax`, `nlxinc`, `nsxinc`, `nxdig`, `nxdec`, *etc.*, but this is much less convenient. For more information, try `help scales`.

7 Surfaces, contours, and other 3D plots

Powerful three–dimensional capabilities of `plotdata` allow us to look at the data that represents functions of two variables, $z = z(x, y)$ rather than simply $y = y(x)$ that we have been discussing so far. This 3D data can be read in from a file or generated via `plotdata` commands and macros.

Outside `plotdata`, use your favourite editor to create the data file `m.dat` with the following data in it:

0.00	0.01	0.16	0.18	0.16	0.01	0.00
0.16	0.20	0.32	0.36	0.32	0.20	0.16
0.07	0.54	0.68	1.00	0.68	0.54	0.07
0.16	0.20	0.32	0.36	0.32	0.20	0.16
0.00	0.01	0.16	0.18	0.16	0.01	0.00

Bring this data into `plotdata` by:

```
PLOTDATA: read/matrix m.dat m 7
PLOTDATA: listm m
```

where `listm` confirms that the data has been read in as a 7×5 matrix. There are several ways to look at this data, all of them representing different renditions of the same 3D surface. You may want to try several of the different ways shown below until you find the rendition that you like the best:

```
PLOTDATA: surface m
PLOTDATA: surface m 1 0 45 -65
PLOTDATA: density m
PLOTDATA: density/diffusion/profile m
PLOTDATA: contour m
Enter specific contour levels (Yes or No) no
Enter:  number_of_contours { min { inc { label_size { legend_size }}}}
>> 10
Enter specific contour levels ( Yes or No ) no
```

Don't forget to `clear` between different commands, to have each try show up separately. In the last example above, `contour m`, `no`, `10`, and `no` are the things that you type, the rest are the prompts from `plotdata`. As usual, to interpret what the different parameters are and to find out about other possibilities, you should use the `help` command and/or look things up in an appropriate manual.

7.1 Generating 3D data

By default, the objects that `plotdata` manipulates are vectors, *i.e.* inherently one–dimensional things. However, one can create a one–dimensional equivalent of a two–dimensional object by "unravelling" the $m \times n$ matrix like this:

$$\begin{pmatrix} z(x_1,y_1) & z(x_1,y_2) & \dots & z(x_1,y_m) \\ z(x_2,y_1) & z(x_2,y_2) & \dots & z(x_2,y_m) \\ \dots & \dots & \dots & \dots \\ z(x_n,y_1) & z(x_n,y_2) & \dots & z(x_n,y_m) \end{pmatrix} \longrightarrow$$
$$\longrightarrow \{z(x_1,y_1),\dots,z(x_1,y_m),z(x_2,y_1),\dots,z(x_2,y_m),\dots,z(x_n,y_1),\dots,z(x_n,y_m)\}$$

The last line can then be thought of as a mapping $z(X,Y)$, where

$$\begin{aligned} X &= \{x_1,x_1,\dots,x_1,x_2,x_2,\dots,x_2,\dots,x_n,x_n,\dots,x_n\} \\ Y &= \{y_1,y_2,\dots,y_m,y_1,y_2,\dots,y_m,\dots,y_1,y_2,\dots,y_m\} \end{aligned}$$

This presupposes that we know $z(x,y)$ on a regular array of data points $\{(x_i,y_j)\}$. Here's one quick way of creating such a mapping:

```
PLOTDATA: generate j 0,,99 100
PLOTDATA: x=int(j/10)*10
PLOTDATA: y=j-x
PLOTDATA: z=(x-50)**2*exp(-0.1*(y-5)**2)
PLOTDATA: list x,y,z
PLOTDATA: density/boxes x,y,z
```

We generated three vectors of length 100, and then interpreted them as a three–dimensional object. The `/boxes` switch shows off one more way of rendering such a "surface".

To do the surface justice, however, it is best to perform the reverse: take a set of vector (1D) data and then create a regularly–spaced grid matrix out of it:

```
PLOTDATA: grid m x,y,z
PLOTDATA: surface m 0 0 25 -30
```

Note that we have used our **regular** arrays `x`, `y`, and `z` to generate a matrix `m`, but in general `grid` command will interpolate data as needed, so the input data representing a surface $z(x,y)$ need not be known at a regularly–spaced grid of points $\{(x_i,y_j)\}$. For our arrays we could have used the option `grid/nointerpolate` since the data is already regularly spaced.

8 Real programming: `if`'s and `do`'s

In addition to the vectors and matrices that `plotdata` manipulates as a whole, it knows about scalars, and, in fact, it knows how to divide up its vectors and matrices into their elements. In `plotdata`, one addresses say, the fifth element of a vector `v` as `v[5]` or `v(5)`, and an element of a matrix `m` as, say, `m[5,6]` or `m(5,6)`. Thus we have another way of creating the matrices of the previous section, by writing a short macro like this, and then `execut`ing it:

```
do j = 1,10
  do k = 1,10
    m[j,k]=(10*(j-6))**2*exp(-0.1*(k-6)**2)
  enddo
enddo
surface m
```

Please, pay attention to the spaces around the equal sign; they are mandatory here, at least in the current version of `plotdata`. An obscure error message will show up if you omit them.

You can see the immediate similarity with Fortran that was mentioned in the beginning. This similarity is, in fact, more extensive: `plotdata` macros can contain labels, logical `if...then` statements, and `goto`'s. These allow you to perform fairly complicated routines without leaving `plotdata`. There are many examples of such macros shown in the *User's Guide*. A word of caution: `plotdata` macros are interpreted line–by–line, and so an equivalent but pre-compiled Fortran program will run much faster. Thus you should reserve your `plotdata` programming for relatively simple things; when the computations get complicated and long it is best to leave `plotdata`; write, compile, and run a Fortran program; and then return to `plotdata` to look at the produced output. Always use the tools that are appropriate to the task at hand!

This brief tutorial should have given you a taste of the capabilities of `plotdata`. For a more comprehensive discussion you are, as always, referred to the manuals.

A On the Unix end

There are several things that need to be correctly set before you can run `plotdata`. The Unix shell (the supervisor that interprets and performs your Unix commands) on `spartan` needs to be told what type of terminal you are on, what type of hardcopy output device you have, and where on the system disk are the files required by `plotdata`. Hopefully, your `.login` file has made these determinations for you. To check, issue the Unix command `setenv`; it will report the list of the so–called environment variables that are currently set. If you do not see the right definitions (see below), here's what you may need to type at the shell prompt:

```
setenv TRIUMF_TERMINAL_TYPE x
setenv TRIUMF_PLOTTER_TYPE postscript
setenv TRIUMF_FONTS /usr/local/lib/triumf
setenv PLOTDATA_DIR /usr/local/lib/triumf
```