COLOR
A Program for Painting Color Contour Maps from a Regular Array

Robert L. Parker

INTRODUCTION

Complex monochrome contour maps consisting of level lines are often hard to interpret; the problem can be alleviated by resorting to colored regions or highlighting with gray shading. The program *color* provides contour maps in PostScript format. The colors are kept constant between specified level lines rather than making any attempt at continuous variation. The program operates in the same way as *plotxy* using the same command structure.

Numerical values are given in a rectangular array, saved as a diskfile. The user types a series of commands instructing the program to perform the tasks desired. Most of the parameters have default values so that to get a quick look at an array is very easy. Suppose the file MAT contains data arranged in 30 rows each of 10 numbers; then to obtain a color map one only need enter

```
file MAT
read 30 10
palette 2
show
plot
stop
```

Here the ASCII input file is specified by **file**; the program reads the data with **read**. The default orientation of the array is that of a numerical listing with row 1 at the top of picture. Next the color are specified with a **palette** command. If this is omitted, the map will be painted in gray tones. Because no explicit levels were provided, the program generates its own and produces the contour map with **plot**. Naturally **stop** halts the program having flushed all the buffers. A PostScript plotfile named *mypost* has been generated which is automatically displayed at the terminal running *gv*; it may be printed on a color PostScript printer; all printers in IGPP handle PostScript format.

All parameters and conditions continue to apply until explicitly changed by the user. Thus unless **file** is changed further **read** commands take data from the original data file. Or, suppose you wish to plot several variants of the same array; there is no need to read the array again. Notes and captions are provided in an elementary way. The catalog below describes the commands provided.

COMMAND CATALOG

All commands must begin in column 1. Each may be abbreviated to its first 4 letters although the full word is given in the catalog. The command line may be up to 80 characters in length. When a number or numbers follow a command, there must be at least one space after the command word. Numbers may be separated by a comma or spaces. Any line beginning with a blank is ignored.

above *z h s b*

The most flexible but also the most tedious way of specifying the colors you want in each region is with this command. The numbers z, h, s, b define a level, and a color

description. Values in the map **above** the level z (and below the next higher level you give) receive the color defined by the Hue-Saturation-Brightness triple h, s, b. For more on what this means see **NOTES**. There must be a separate **above** command for each color in the contour map. Order does not matter. It is simpler to use one of the predefined palettes: see **palette**. Values below the lowest level mentioned in this command are assigned an extrapolated color close the color immediately above it. See also **levels** and **interval**.

affine *a b*

Immediately after input, the array values are transformed according to: znew= z * a + b, which is an affine mapping. All subsequent commands act on the transformed values, not the originals.

axes *x1 x2 y1 y2*
axes *x1 x2 y1 y2 hidden*

The bottom and left sides of the contour map can be annotated with linear axes covering the intervals specified. These intervals have nothing to do with the dimension of the array being contoured and can be set arbitrarily. Labeling of the axes is not provided explicitly but can be accomplished through the **note** command.

The command is also used to supply a coordinate system for additional lines that may be added on top of the contour map. (see **lines** and **symbol**). The limits are also written on the map at plot time, but this notation can be suppressed by including the word *hidden* as indicated. The axes do not participate in the transformations of the **mapping** command.

border [*on off mask*]

If a line surrounding the final map is desired, the user can invoke **border** either with a blank command field, or with on. To cancel the border, when it has been set in an earlier plot, set the tag to off. By default maps have borders. If the code mask is used, the region between the normal bounding rectangle and an inscribed ellipse (or circle if width and height are equal) is set to be white, and is therefore masked out. An elliptical border is always supplied in this case. This option is useful for certain kinds of geographical contour maps, in Lambert equal-area or Hammer-Aitoff projections. See **mapping**.

dash
dash *z1 z2 z3 ...*

See **outline**: this command draws a dashed contour outline instead of a solid line, but is otherwise identical with **outline**. If a contour level appears in both commands, the dashed option takes precedence. Also see **heavy**, **weight**.

heavy
heavy *z1 z2 z3 ...*

See **outline** and **dash**: provides for a heavy contour outlining the colored areas. The command allows the user to highlight certain levels. For example

```
interval 100
outline
heavy 0
```

sets equally space levels at 100 units, provides a solid outline for all of them, and gives the level zero a heavy weight. Also see **dash**, **outline**, and **weight**.

file *filename*

The rectangular data array to be contoured is held in a diskfile named 'filename'. This command identifies the file to the program and must be given before a **read** command can be issued. If several arrays are present in one file, there is no need to issue this command more than once. If you issue the command without a filename, the current file will be rewound to the beginning.

format *type*

The command specifies the format of the next **read**. The diskfile containing the array may be an ASCII formatted file or a binary (unformatted, sequential access) file. The default is formatted and then type is just blank. If type is *b* this specifies a binary file. See **read** for the structure of binary files. Note that precise formats cannot be specified here (unlike in *plotxy*), and that all formatted files are read with the FORTRAN statement READ (iunit, ∗) in the program.

height *h*

The height of the finished contour map in inches is specified with this command. If **height** is not issued or h is zero, then the size of the figure is determined from **width** and the natural proportions implied by the dimensions of the array. If neither **height** nor **width** is assigned, the program takes the larger side of the array to be 6 inches long and the other one in proportion according to the dimensions.

interval *dz*
interval *z1 z2*
interval *z1 z2 dz*

If you use **palette**, and omit the **level** command, the levels at which color changes occur will be left to the program to determine. Usually, the variation spans the range of numbers in the array, but if you desire, that range can be changed to the interval (z1, z2) using this command. This is useful if there are several different data sets are to be mapped with the same contour scheme. Alternatively, you can supply the exact values you want through **level** or **above**. If a positive value dz is provided, the color transitions will begin at z1 and march uniformly through the interval, advancing by dz at each step. If only positive dz is supplied all contour levels of the form integer*dz in the data range will appear. If dz is negative, this is interpreted to mean that -dz is the approximate number of evenly spaced contours, subject to a nice increment of the program's choosing.

note *(x, y) text*
note *(x, y, h) text*
note *(x, y, h, angle) text*

Write the text with the bottom left corner of the first character at coordinates x, y, in inches referred to the bottom left corner of the contour map. If the parameter h is present, it gives the height of the text; otherwise, or if h=0, **size** gives the character height in inches. The fourth, optional parameter is the angle in clockwise degrees that the note makes with the horizontal. There can be as many **note** commands as needed. Notes are **not** carried forward, however, to later contour maps, so that if required, the notes must be redefined for each new map.

label *text*

An identifying label comprised of the given text will be printed under the map. More elaborate captions, axis labels and titles can be obtained through **note**.

size *h* [strict]

Text takes the character height h in inches, although provision is made for variable heights of notes. Note, however, that if necessary, the numbers labeling the levels in the color key will be reduced in size to avoid overlapping. When large numerals are important (for clarity in overheads, etc), or you have used too many contour levels (microscopic numerals), this shrinking can be overridden by including the word *strict* as shown. Then to relieve overcrowding, not all the levels will receive a numerical label.

landscape

To orient the plot so that horizontal runs along the long axis of the paper include this command before the first **plot** command. Once turned on it cannot be revoked.

level *z1 z2 z3 . . .*

If you use **palette**, the values at which color transitions occur may be specified precisely by the numbers z1, z2, . . . The number of contour levels is simply the number of values supplied. If there are too many values to fit onto a single line, just repeat the command with the remainder of the list. See also **interval**.

lines *filename* [color=X] [wt=W]

Continuous lines may be superposed on the contour map by reading the x-y coordinates from the named file, which is always read to the end-of-file and must contain numerical x-y pairs in the desired order, one pair per line. The coordinates for the plotting are defined by the **axes** command. To break the line, insert a point lying outside the plot limits set by **axes**. The color of the line may be specified by setting X to one of the eight colors: black, white, red, blue, green, yellow, orange, or purple. Optionally, the line weight is given by W, an integer that is the line width in 1/1000 inches; W=6 is the default. The command may be repeated with different files to accumulate information from several sources. As usual the x-y line data will be carried forward onto subsequent plots, unless they are all cancelled by calling **lines** without a filename. If further data are input in later plots, any previous set of x-y data is overwritten. This command is not as flexible as the one available in *contour*. When **mapping** is used the coordinates may be automatically transformed, or plotted directly onto the map as read. See also **symbol**.

symbol *filename kind h* [color=X]

It is often useful to mark points on the map with symbols. This command identifies a disk file, like the one used in the **lines** command, and causes a symbol to be plotted at each point. The file is always read to the end. Only one symbol is associated with a particular file, its type defined by *kind* and its height *h* in inches. You may read more than one file with further **symbol** commands to get a variety of symbols. The code *kind* conforms approximately to that used in *plotxy*; the order is unfortunately not very logical: 0 square; 1 triangle; 2 octagon; 3 diamond; 4 plus; 5 asterisk; 6 cross; 7 5-ray; 8 Y upside down; 9 pentagon; 10 triangle, base up; 11 hexagon; 12 Y; 13 bar; 14 6-ray; 15 dot; 16 heptagon; 17 circle; 18 large circle (double height); 19 small filled circle; 20 small filled square; 21 small filled triangle. Colors are set by choosing X from the list: black, white, red, blue, green, yellow, orange, purple.

All the symbols are carried forward to the next map if no new **symbol** commands are issued; but previously defined symbols will be cancelled if a new symbol command is included in the commands for a later map – then only the new set will be plotted. All previously defined symbols are deleted if **symbol** is entered without

arguments. See also *lines*.

nobar [*off*]

Normally a key to the colors is drawn to the right of the map. To prevent this, enter **nobar**. To reinstate the bar use the second form with the word "off" after the command.

orient *options*

There are eight different orientations on a page of an array with sides parallel to the edges of the paper; the user can specify which one in two different ways. First by giving the name of the desired orientation chosen from th following list; only the first letter (lower case) of the name is needed. Call the array A, then

numerical: Oriented as numerical listing of A.
vertical: A is plotted reflected in a vertical line in the page.
horizontal: A is plotted reflected in a horizontal line.
upside-down: Plot A after 180 degree rotation about an axis normal to the page.
clockwise: A is rotated clockwise by 90 degrees before plotting.
anticlockwise: A is rotated 90 degrees anticlockwise.
transpose: The matrix transpose A^T is plotted.
kombo: Combination of c first, then h; no easier description.

The second method is to supply a set of operations in the option list, that reorient the original array. The letter N denotes the normal (or numerical) arrangement of the array, oriented as it would be seen in a numerical listing of the input file, just like n. N is the default orientation, assumed if the **orient** is not given. T means the array is transposed before plotting. These two orientations can be followed by one or more 90-degree clockwise rotations denoted by R. In addition you can reflect the array about a horizontal line with H; or reflect about a vertical line with V. All of these letters represent operations performed on the array after it has been read. So you may type NRRVH which means the original array is is rotated clockwise twice, then reflected in the vertical axis, then in the horizontal. (It is an exercise to find the final state after this chain.) The actions on the array are performed in the order written, left-to-right, not the normal mathematical order of operators. Thus RV first rotates the array, then reflects it. Having a redundant set of multiple operations means that you can specify the orientation in the most comfortable way, but in fact it is never necessary to use more than two operations to achieve any orientation.

If you want to plot several different orientations of the same array, you must **read** the array each time. See the examples at the end of this document.

outline
outline *z1 z2 z3 ...*
outline *off*

In the first form, run a dark contour line along each color boundary. In the second form, specific contour levels are given, and the outline runs only along those levels. If a level in the list is not in the contour set, it will not be created. To turn off the outlining, use the third form, with *off* as a parameter. Boundaries indicated in this command will also be marked in the color key at the side of the map. See **heavy** and **dash** for variants on the basic theme. See also **weight**.

output *filename*

An alternative plotfile name to the default *mypost* may be specified with this command; it must be invoked before the first call to **plot** and after that it will be ignored.

palette *p*
palette *0 d q*

Because of the tedium of specifying the color of each region individually with **above**, a number of predefined schemes have been provided and can be invoked with this command. The integer p may be 0 (for a gray scale), 1, 2, etc up to the maximum provided, currently 7. Here is rough description of the current collection: p=1, Blue-Red ranges from deep blue at the low end to dark red at the top; values in the middle of the range are pale; p=2, Bright Spectrum, progresses roughly in spectral order from deep blue, through green, yellow, orange, brown, red and violet; p=3, Rainbow, is a more muted version of p=2; p=4, Harvard, runs from dark blue, light blue, yellow, orange then brown - no green; p=5, Sand, yellow through light brown, all pale colors; p=6, Cartographic, blue though green, yellow, brown all very pale; p=7, Oceanographic, dark deep blue through pale blue.

If you ask for more levels than the palette provides, the HSB coordinates will be interpolated and that may or may not result in distinct colors in the final plot. For more on color choices see **NOTES**.

To enter a customized color palette use the **table** command to read values from a file.

When p=0 a gray scale is provided. This can be tuned with the optional numbers d and q. The intensity of the gray tone is controlled by the range of values in the array or the values set by **interval**. The number d controls the density of the blackest parts of the plot; d should be no greater than unity in magnitude. If d is positive the darkness runs from zero (pure white) for the smallest array value to a maximum of d (where 1 is completely black) corresponding to maximum values. If contour lines are to be superposed with **outline**, d should be no more than 0.7 to prevent the gray tone obscuring the contour lines. When d is negative, light tones are used for high values in the array and dark ones for the smaller values.

When a linear mapping is made between gray level and function magnitude the overall aspect of the picture is rather dark. To lighten the plot the second argument q allows stretching of the gray scale through a simple power law. Thus when q is 2 (the default) the darkness value is squared, which has the effect of reducing the number of very dark regions; q need not be an integer but must be positive. You should experiment with these arguments for the best effect.

Obviously, the commands **interval**, **levels**, **above** are alternative ways of specifying the transition levels. If more than one of them is present, the order of precedence is: **above**, **levels**, **interval**, meaning that the **above** command will cancel the effect of the other two if present, and so on. This holds irrespective of the order in which the commands are entered.

plot
plot *x y*

This command instructs the program to perform the contouring of the array. The numbers x and y are usually omitted; see below. All the specifications up to this point are used or defaults assumed if the user is silent on essential matters. What is done is merely to write plotting instructions to the plotfile *mypost* which must be sent

to a PostScript graphics output device at a later stage for inspection.

Another contour map may be plotted without restarting the program by reading more data and calling **plot** again. If new array data are input, they supersede the earlier numbers, otherwise the old array will be used. The next map will be placed tastefully above the previous one unless you want something special. The numbers x and y in the argument can be used to specify the coordinates in inches of the new plot origin (bottom-left corner of the map) relative to the old origin. If x and y are given for the first map, the bottom-left corner of the map will positioned at these coordinates with respect to the bottom-left corner of the paper.

show

Under the Mac X operating system Fortran can launch a shell process; **show** starts the PostScript display program gv that sends the contour map image to the terminal.

read *m n*

The array of values to be contoured is read from the diskfile whose name has been supplied with **file**. The rectangular array of values must be organized in a series of m rows, and each row is n numbers long in the standard mathematical way. The rows are read one at a time, the top row first in as a numerical listing. However, in an ASCII text file, a row of the array does not have to appear on a single line, but may continue onto as many lines are necessary: only the order of the numbers in the file is important, not where the line breaks occur (indeed, one number per line is acceptable). A binary file (specified by **format** b) must be written in FORTRAN with

```
      DO 1100 I=1, M
1100  WRITE (iunit) (A(I,J), J=1,N)
```

where m = M and n = N.

The **orient** command can rotate or reflect the array after it has been read.

skip *n*

When reading from a diskfile in ASCII the program will skip the first n lines before reading in data. This allows the user to put header information in the file. Notice lines of input are skipped, not rows of the matrix. Skipping occurs immediately after the file is opened, so that when there are several arrays in the same file, no skipping is performed before the second and subsequent data sets are input.

stop

Flushes the plot buffers, closes the plotfile *mypost* and data diskfile then halts the program. You may create as many different contour maps as you desire in one execution of *color* by repeated **plot** commands.

table *filename*

Reads a substitute color table for palette 1 from the named file, which must be an ASCII file in form of a table with three columns comprising successive values of hue, saturation and brightness. Setting **palette** 1 after this command will get the new colors, the old ones having been overwritten.

verbose *n*

As execution proceeds *color* prints a minimum of information. Most printing can be suppressed by setting n to 0. With n set to 1 information is printed that can be useful at a later stage, particularly the actual color values used by the program, which you

may want to modify slightly.

weight *w1 w2 w3* The line weights for regular outlines (w1), dashed outlines (w2) and heavy outlines (w3) may be set in units of 0.001 inches. The defaults are w1=w2=4, w3=12. see **outline**.

width *w*

As with **height** this command sets the width of the map in inches. If w is zero or left unset, the width is computed from the height and the natural proportions of the rectangular array.

NOTES

Unlike black-and-white plotting, color rendition depends strongly upon hardware. PostScript provides a hardware-independent description of the colors, but what those colors actually look like varies from device to device. I have debugged and designed palettes on a 16-inch Sony color monitor. The transition to paper is somewhat unpredictable and some experimentation may be advisable for satisfactory results.

Using **above** you can describe a color precisely through its HSB (Hue-Saturation-Brightness) coordinates. Roughly speaking these have the following meanings. Hue is a number ranging from 0 to 1 that takes the color around the perimeter of a color disk: 0 is pure red, 0.333 is pure green; 0.667 is pure blue and 1.0 is back at red again. Yellow lies between red and green at about 0.13, purple between blue and red at about 0.8. These colors can be modified in their saturation, the second coordinate: roughly speaking, saturation 1 is the pure color, and decreasing saturation adds white or gray (depending on brightness). Thus 0 saturation is a gray tone independent of the hue. The third coordinate, brightness, is self-explanatory: red with a low brightness is brown; zero brightness is always black, independently of hue and saturation.

It is unwise to attempt the representation of too many levels through a large number of colors. I have found 20 to be a reasonable number. When contour levels are unevenly spaced, the color key is also drawn with uneven spacing to reflect this fact. But it can easily happen that closely spaced levels might cause neighboring numerical labels to overlap. *Color* avoids this by distorting the spacing in the color key to provide enough room for a label at every level. If the *strict* option is invoked in the **size** command, some levels will be unlabeled, which may lead to a serious loss of information, and therefore *strict* sizing is not recommended when contours levels are set unevenly.

Lettering in *color* is relatively unsophisticated, and is confined to a single font, Helvetica, with a very simple facility for sub- or superscripting. To obtain a superscript you can raise the level of the line by a half letter height by including \up\; then you must lower the level afterwards with\down\. Subscripts are produced by applying these two phrases in the reverse order. No change in character size is implied.

Greek letters can be included. The English name must be enclosed in back-slashes; upper case is specified through the first character of the name; the name can be abbreviated. Thus \a\ works for alpha, but \ome\ is required for omega. Also for upsilon use \ups\ to avoid interpretation as the superscript mechanism.

GRID TRANSFORMATIONS

The following describes some special functions for drawing contour maps in a transformed plane. The easiest way to understand the functions is to consider the example of a map projection: suppose the array represents data on a regular grid of latitude and longitude, but the user wants the final picture in some map projection. This is accomplished by transforming the contour outlines just before they are output. For the Aitoff projection (n=2, below) it is almost always best to include the **mask** command.

mapping *n* [+data]

Define the kind of mapping to be used: n=1 means uniform mapping from the grid, that is, no transformation is done; n=2 transforms the grid onto an Hammer-Aitoff equal-area map projection; n=3 maps x and y separately through a mapping read from a diskfile (see **tensor**). In the case of the Hammer-Aitoff mapping, it is assumed that x coordinate in the original array runs from 0 to 360 degrees of longitude, and that the y coordinate spans −90 to 90 in latitude; if **lines** is used the axes will be automatically set with the x interval (-2,2), and the y interval (-1,1). The mapping remains in force until changed.

If the optional +data tag is included, and n=2, the numbers in the files of the **lines** and **symbol** commands are taken to be latitude-longitude degrees rather than (x, y) coordinates referred to the Cartesian axes, and the data mapping is performed internally. Otherwise, those values will not be mapped.

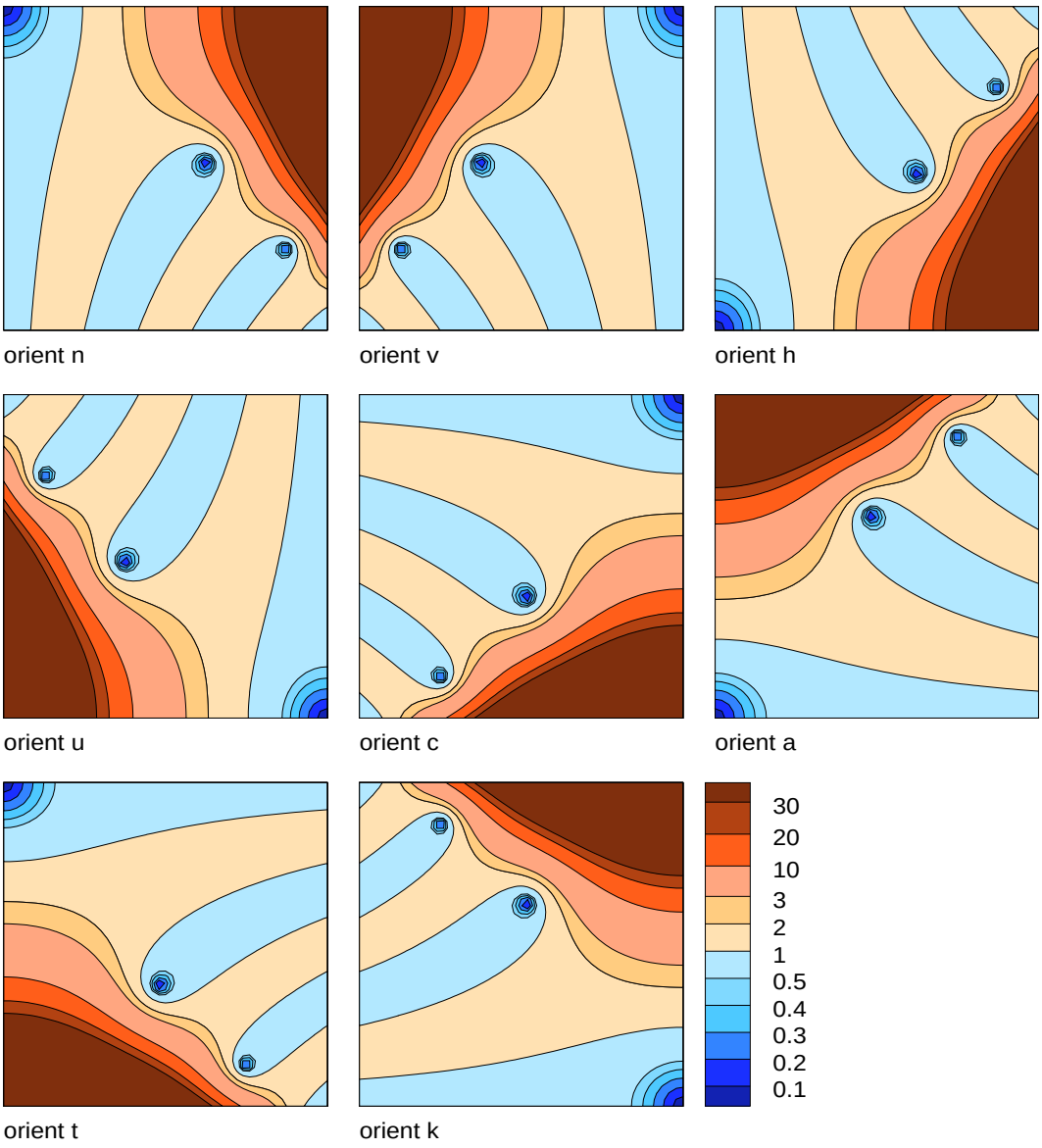
tensor *filename*

In **mapping** with n=3 the spacing in the rectangular array between the adjacent columns of numbers (the x spacing) need not be constant, but is defined by the user in the file named in the command; and similarly with rows (y spacing). If the input array has kx columns across (at plot time – recall it may have been rotated first), there must be kx-1 numbers in the file giving the relative spacing between adjacent columns, starting at the left; these must be followed by ky-1 numbers giving the relative spacing in y. The values give the ratio of the separations not the actual intervals since the size of the final plot is determined by the commands **width** and **height**. The reason for the name is that this is a tensor-product transformation.

EXAMPLES

If you are reading the PDF document you will be able to examine examples of plots made by *color* on the next few pages. First comes an illustration of the various orientations. The *color* script for the top left panel is

```
nobar
width 2
palette 4
outline
levels 0.1 0.2 0.3 0.4 0.5 1 2 3 10 20 30
orient n
note (0 −0.2)orient n
file zerf.dat
read  51 51
plot 0.75 8
```



Subsequent commands are of the form

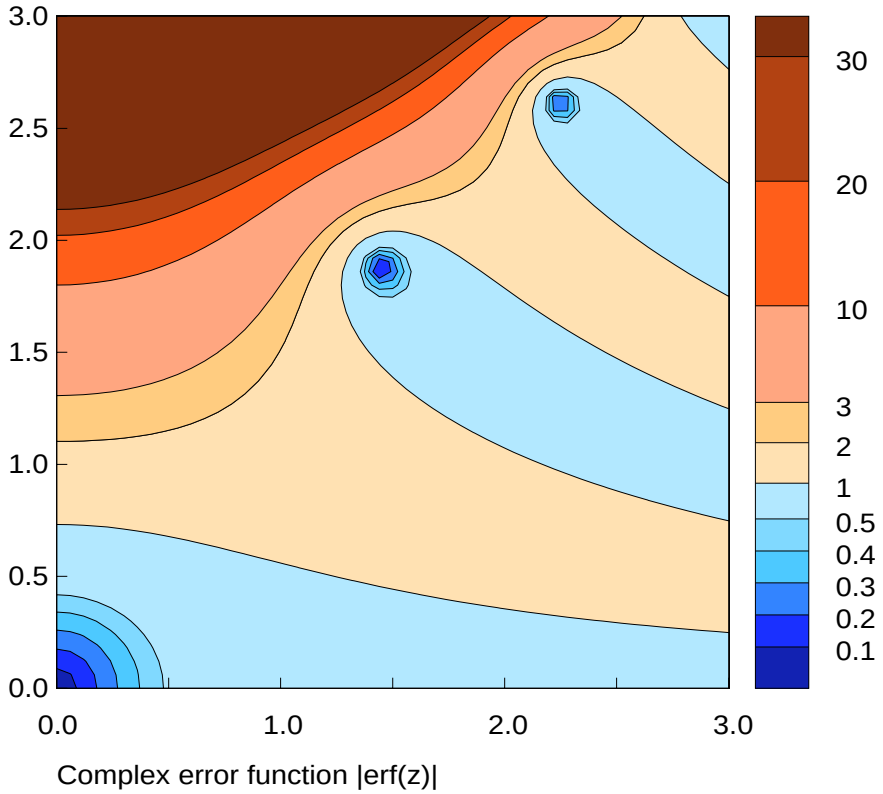
```
note (0 –0.2)orient v
orient v
file
read 51 51
plot 2.2  0
```

with appropriate changes to the **orient, note** and **plot** commands. The last plot also includes **nobar** *off*.

The function being contoured is the magnitude of the error function erf(z) in the first quadrant of the complex plane. There is insufficient room to plot axes. In the next plot we supply axes with

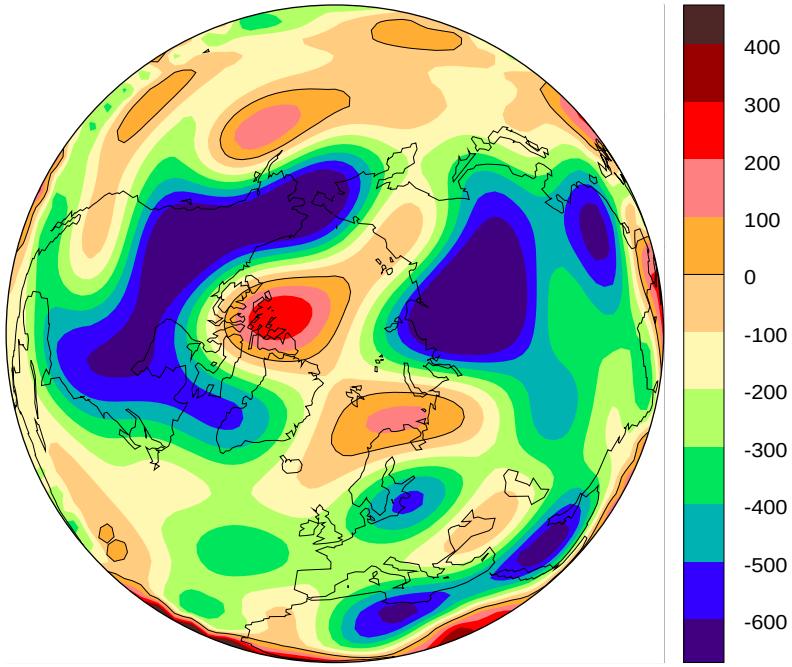
```
axes 0 3 0 3
```

and draw at a slightly larger scale. You may notice that the color bar at the right has expanded the intervals that are larger, but not in proportion to their true size, giving



a pleasing appearance. There was not enough space for this arrangement in the first set of graphs.

Next is a geographic map made from data generated in the program *magmap*.



Here is the script:

```
border mask
file polar.dat
read    40   40
axes   -1    1   -1    1  hidden
lines coast10
width 4
outline 0
palette 2
plot
```

We are looking down on the north pole at a picture of the radial magnetic field on the core-mantle boundary. Note the use of **border** *mask* to eliminate values in the array that have no place in the map. A coastline has been added for orientation purposes, although of course that boundary does not exist on the core. The values in the arrays *coast10* and *polar.dat* have been transformed appropriately outside *color*, since the program has only one bulit-in map projection, Hammer-Aitoff. The **outline** function is applied at only one level (the magnetic equator) to avoid too many lines that conflict with the coastline.