Appendix B: GMT file formats

1. Table data

   These files have *N* records which have *M* fields each. Most programs can read multicolumn files, but require that the *x* [and *y*] variable(s) be stored in the 1st [and 2nd] column (There are, however, some exceptions to this rule, such as *filter1d* and *sample1d*). GMT can read both ASCII and binary table data:

   ASCII: The first data record may be preceded by 1 or more header records. When using such files, make sure to use the **–H** option and set the parameter N_HEADER_RECS in the .gmtdefaults file (System default is 1 header record if **–H** is set). Fields within a record must be separated by spaces, tabs, or commas. When dealing with time- or (*x*,*y*)-series it is usually convenient to have each profile in separate files. However, this may sometimes prove impractical due to large numbers of profiles. An example is files of digitized lineations where the number of individual features may range into the thousands. One file per feature would in this case be unreasonable and furthermore clog up the directory. GMT provides a mechanism for keeping more than one profile in a file. Such files are called multiple segment files and are identical to the ones just outlined except that they have subheaders interspersed with data records that signal the start of a segment. The subheaders may be of any format, but all must have the same character in the first column. When using such files, you must specify the **–M** option. The unique character is by default '>', but you can override that by appending your chosen character to the –M option. E.g., **–MH** will look for subheaders starting with H, whereas **–M'*'** will check for asterisks (The quotes are necessary since * has special meaning to *UNIX*).

   binary: GMT programs also support binary tables to speed up input-ouput for i/o-intensive tasks like gridding and preprocessing. Files may have no header (hence the **–M** option cannot be used) and all data must either be single or double precision (no mixing allowed). The format and number of fields are specified with the **–b** option. Thus, for input you may set **–bi**[**s**][*n*], where **s** designates single precision (default is double) and *n* is the number of fields. For output, use **–bo**[**s**].

2. 2-D grdfile.

   The default 2-D binary netCDF grd-file in GMT has several attributes. The *grdedit* utility program will allow you to edit parts of the header of an existing grdfile. The following attributes are contained within the header record:

   Header record:

char title[80];	Descriptive title of the data set
char command[320];	Command line that produced the grdfile
char remark[160];	Any additional comments
char x_units[80];	Units of the x-dimension
char y_units[80];	Units of the y-dimension
char z_units[80];	Units of the z-dimension
int nx;	Number of nodes in the x-dimension
int ny;	Number of nodes in the y-dimension
double x_min;	Minimum x-value of region
double x_max;	Maximum x-value of region

double y_min;	Minimum y-value of region
double y_max;	Maximum y-value of region
double z_min;	Minimum z-value in data set
double z_max;	Maximum z-value in data set
double x_inc;	Node spacing in x-dimension
double y_inc;	Node spacing in y-dimension
double z_scale_factor;	Factor to multiply z-values after read
double z_add_offset;	Offset to add to scaled z-values
int node_offset;	0 for grid line registration, 1 for pixel registration

Data record:

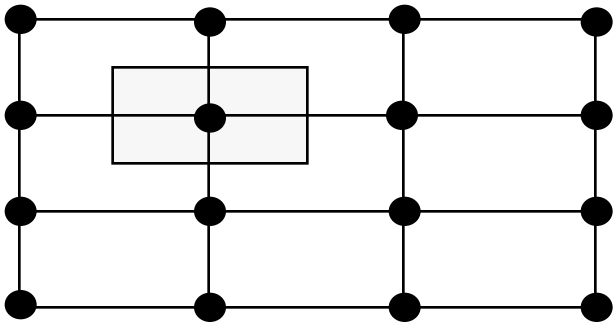
float z[nx*ny]	z-values stored in scanline format

GMT version 3 also allows other formats to be read. In addition to the default netcdf format it can use binary floating points, short integers, bytes, and bits, as well as 8-bit Sun rasterfiles (colormap ignored). Additional formats may be used by supplying read/write functions and linking these with the GMT libraries. The source file gmt_customio.c has the information that programmers will need to augment GMT to read custom grdfiles. We anticipate that the number of pre-programmed formats will increase as enterprising users implement what they need.

Scanline format means that the data are stored in rows (y = constant) going from the "top" (y = y_{max} (north)) to the "bottom" (y = y_{min} (south)). Data within each row are ordered from "left" ($x = x_{min}$ (west)) to "right" ($x = x_{max}$ (east)). The node_offset signals how the nodes are laid out. The grid is always defined as the intersections of all x ($x = x_{min}, x_{min} + x_{inc}, x_{min} + 2* x_{inc}, ..., x_{max}$) and y ($y = y_{min}, y_{min} + y_{inc}, y_{min} + 2* y_{inc}, ..., y_{max}$) lines. The two scenarios differ in which area each data point represents.

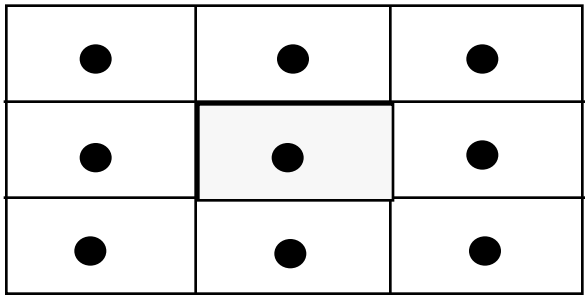
1) Grid line registration.

In this registration, the nodes are centered on the grid line intersections and the data points represent the average value in a cell of dimensions ($x_{inc} * y_{inc}$) centered on the nodes:



2) Pixel registration

Here, the nodes are centered in the grid cells, i.e., the areas between grid lines, and the data points represent the average values within each cell:



Thus, inspecting the figures we see that in the case of grid registration the number of nodes are related to region and grid spacing by

$$nx = (x_{max} - x_{min}) / x_{inc} + 1 = 4$$
$$ny = (y_{max} - y_{min}) / x_{ync} + 1 = 4$$

while for pixel registration we find

$$nx = (x_{max} - x_{min}) / x_{inc} = 3$$
$$ny = (y_{max} - y_{min}) / y_{inc} = 3$$

The default registration in GMT is grid line registration. Most programs can handle both types, and for some programs like *grdimage* a pixel registered file makes more sense. Utility programs like *grdsample* and *grdproject* will allow you to convert from one format to the other.

Boundary Conditions for operations on grids

GMT has the option to specify boundary conditions in some programs that operate on grids (*grdsample* **–L**; *grdgradient* **–L**; *grdtrack* **–L**; *nearneighbor* **–L**; *grdview* **–L**). The boundary conditions come into play when interpolating or computing derivatives near the limits of the region covered by the grid. The default boundary conditions used are those which are "natural" for the boundary of a minimum curvature interpolating surface. If the user knows that the data are periodic in *x* (and/or *y*), or that the data cover a sphere with *x,y* representing *longitude,latitude*, then there are better choices for the boundary conditions. Periodic conditions on *x* (and/or *y*) are chosen by specifying *x* and or *y* as the boundary condition flags; global spherical cases are specified using the *g* (geographical) flag. Behavior of these conditions is as follows:
Periodic conditions on *x* indicate that the data are periodic in the distance (x_{max} - x_{min}) and thus repeat values after every $N = (x_{max} - x_{min})/ x_{inc}$. Note that this implies that in a grid-registered file the values in the first and last columns are equal, since these are located at $x = x_{min}$ and $x = x_{max}$, and there are $N + 1$ columns in the file. This is not the case in a pixel-registere file, where there are only N and the first and last columns are located at $x_{min} + x_{inc}/2$ and $x_{max} - x_{inc}/2$. If *y* is periodic all the same holds for *y*.

Geographical conditions indicate the following:

1. If (x_{max} - x_{min}) ≥ 360 and also 180 modulo $x_{inc} = 0$ then a periodic condition is used on *x* with a period of 360; else a default condition is used on the *x* boundaries.

2n. If condition 1 is true and also $y_{max} = 90$ then a "north pole condition" is used at y_{max}, else a default condition is used there.
2s. If condition 1 is true and also $y_{min} = -90$ then a "south pole condition" is used at y_{min}, else a default condition is used there.

"Pole conditions" use a 180˚ phase-shift of the data, requiring 180 modulo $x_{inc} = 0$. Default boundary conditions are

$$^2 f = 0 \text{ and } \frac{\partial}{\partial n} \; ^2 f = 0$$

on the boundary, where $f(x, y)$ is represented by the values in the grid file, $\partial/\partial n$ is the derivative in the direction normal to the boundary, and

$$^2 = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \right)$$

is the two-dimensional Laplacian operator.

3. Sun raster files

The Sun raster file format consists of a header followed by a series unsigned short integers that represents the bit-pattern. Bits are scanline oriented, and each row must contain an even number of bytes (due to the short int building block). The predefined 1-bit patterns in GMT have dimensions of 64 by 64, but other sizes will be accepted when using the **–Gp|P** option. The Sun header structure is

```
int  ras_magic;          /* magic number */
int  ras_width;          /* width (pixels) of image */
int  ras_height;         /* height (pixels) of image */
int  ras_depth;          /* depth (1, 8, 24, 32 bits) of pixel */
int  ras_length;         /* length (bytes) of image */
int  ras_type;           /* type of file; see RT_* below */
int  ras_maptype;        /* type of colormap; see RMT_* below */
int  ras_maplength;      /* length (bytes) of following map */
```

After the header, the color map (if ras_maptype is not RMT_NONE) follows for ras_maplength bytes, followed by an image of ras_length bytes.

Some definitions that are related:

```
#define RAS_MAGIC           0x59a66a95
#define RT_STANDARD         1   /* Raw pixrect image in 68000 byte order */
#define RT_BYTE_ENCODED     2   /* Run-length compression of bytes */
#define RT_FORMAT_RGB       3   /* [X]RGB instead of [X]BGR */
#define RMT_NONE            0   /* ras_maplength is expected to be 0 */
#define RMT_EQUAL_RGB       1   /* red[ras_maplength/3],green[],blue[] */
```

Numerous public-domain programs exist, such as *xv* and *convert* (in the ImageMagick package), that will translate between various rasterfile formats such as tiff, gif, jpeg, and Sun raster. Raster patterns may be created with GMT plotting tools by generating *PostScript* plots that can be rasterized by *ghostscript* and translated into the right raster format. Make sure to use RT_STANDARD format.