THE GENERIC MAPPING TOOLS

GMT Documentation

Release 5.1.1

P. Wessel, W. H. F. Smith,
R. Scharroo, J. Luis, and F. Wobbe

March 01, 2014

Contents

The Generic Mapping Tools

Technical Reference and Cookbook

Pål (Paul) Wessel

SOEST, University of Hawai'i at Manoa

Walter H. F. Smith

Laboratory for Satellite Altimetry, NOAA/NESDIS

Remko Scharroo

EUMETSAT, Darmstadt, Germany

Joaquim F. Luis

Universidade do Algarve, Faro, Portugal

Florian Wobbe

Alfred Wegener Institute, Germany

Figure 1: The four horsemen of the GMT apocalypse: Remko Scharroo, Paul Wessel, Walter H.F. Smith, and Joaquim Luis at the GMT Developer Summit in Honolulu, Hawaii during February 14–18, 2011.

The Generic Mapping Tools (GMT) could not have been designed without the generous support of several people. We gratefully acknowledge A. B. Watts and the late W. F. Haxby for supporting our efforts on the original version 1.0 while we were their graduate students at Lamont-Doherty Earth Observatory. Doug Shearer and Roger Davis patiently answered many questions over e-mail. The subroutine `gauss` was written and supplied by Bill Menke. Further development of versions 2.0–2.1 at SOEST would not have been possible without the support from the HIGP/SOEST Post-Doctoral Fellowship program to Paul Wessel. Walter H. F. Smith gratefully acknowledges the generous support of the C. H. and I. M. Green Foundation for Earth Sciences at the Institute of Geophysics and Planetary Physics, Scripps Institution of Oceanography, University of California at San Diego. GMT series 3.x, 4.x, and 5.x owe their existence to grants EAR-93-02272, OCE-95-29431, OCE-00-82552, OCE-04-52126, and OCE-1029874 from the National Science Foundation, which we gratefully acknowledge.

We would also like to acknowledge feedback, suggestions and bug reports from Michael Barck, Manfred Brands, Allen Cogbill, Stephan Eickschen, Ben Horner-Johnson, John Kuhn, Angel Li, Andrew Macrae, Alex Madon, Ken McLean, Greg Neumann, Ameet Raval, Georg Schwarz, Richard Signell, Peter Schmidt, Dirk Stoecker, Eduardo Suárez, Mikhail Tchernychev, Malte Thoma, David Townsend, Garry Vaughan, William Weibel, and many others, including their advice on how to make GMT portable to a wide range of platforms. John Lillibridge and Stephan Eickschen provided the original examples 11 and 32, respectively; Hanno von Lom helped resolve early problems with DLL libraries for Win32;

Lloyd Parkes enabled indexed color images in PostScript; Kurt Schwehr maintains the packages; Wayne Wilson implemented the full general perspective projection; and William Yip helped translate GMT to POSIX ANSI C and incorporate netCDF 3. The SOEST RCF staff (Ross Ishida, Pat Townsend, and Sharon Stahl) provided valuable help on Linux and web server support.

Honolulu, HI, Silver Spring, MD, Faro, Portugal, Darmstadt and Bremerhaven, Germany, March 2014

CHAPTER 1

A Reminder

If you feel it is appropriate, you may consider paying us back by citing our EOS articles on GMT and technical papers on algorithms when you publish papers containing results or illustrations obtained using GMT. The EOS articles on GMT are

- Wessel, P., W. H. F. Smith, R. Scharroo, J. Luis, and F. Wobbe, Generic Mapping Tools: Improved Version Released, *EOS Trans. AGU*, 94(45), p. 409-410, 2013. doi:10.1002/2013EO450001.
- Wessel, P., and W. H. F. Smith, New, improved version of Generic Mapping Tools released, *EOS Trans. AGU*, 79(47), p. 579, 1998. doi:10.1029/98EO00426.
- Wessel, P., and W. H. F. Smith, New version of the Generic Mapping Tools released, *EOS Trans. AGU*, 76(33), 329, 1995. doi:10.1029/95EO00198.
- Wessel, P., and W. H. F. Smith, Free software helps map and display data, *EOS Trans. AGU*, 72(41), 445–446, 1991. doi:10.1029/90EO00319.

Some GMT programs are based on algorithms we have developed and published separately, such as

- Kim, S.-S., and P. Wessel, Directional median filtering for regional-residual separation of bathymetry, *Geochem. Geophys. Geosyst.*, 9, Q03005, 2008. doi:10.1029/2007GC001850. [`dimfilter`, **misc** supplement]
- Luis, J. F. and J. M. Miranda, Reevaluation of magnetic chrons in the North Atlantic between 35N and 47N: Implications for the formation of the Azores Triple Junction and associated plateau, *J. Geophys. Res.*, 113, B10105, 2008. doi:10.1029/2007JB005573. [`grdredpol`, **potential** supplement]
- Smith, W. H. F., and P. Wessel, Gridding with continuous curvature splines in tension, *Geophysics*, 55(3), 293–305, 1990. doi:10.1190/1.1442837. [`surface`]
- Wessel, P., Tools for analyzing intersecting tracks: The x2sys package, *Computers & Geosciences*, 36, 348–354, 2010. doi:10.1016/j.cageo.2009.05.009. [`x2sys` supplement]
- Wessel, P., A General-purpose Green's function-based interpolator, *Computers & Geosciences*, 35, 1247–1254, 2009. doi:10.1016/j.cageo.2008.08.012. [`greenspline`]
- Wessel, P. and J. M. Becker, Interpolation using a generalized Green's function for a spherical surface spline in tension, *Geophys. J. Int.*, 174, 21–28, 2008. doi:10.1111/j.1365-246X.2008.03829.x. [`greenspline`]

Finally, GMT includes some code supplied by others, in particular the Triangle code used for Delaunay triangulation. Its author, Jonathan Shewchuk, says

> “If you use Triangle, and especially if you use it to accomplish real work, I would like very much to hear from you. A short letter or email describing how you use Triangle will mean a lot to me. The more people I know are using this program, the more easily I can justify spending time on improvements and on the three-dimensional successor to Triangle, which in turn will benefit you.”

A few GMT users take the time to write us letters, telling us of the difference GMT is making in their work. We appreciate receiving these letters. On days when we wonder why we ever released GMT we pull these letters out and read them. Seriously, as financial support for GMT depends on how well we can “sell” the idea to funding agencies and our superiors, letter-writing is one area where GMT users can affect such decisions by supporting the GMT project.

CHAPTER 2

Copyright and Caveat Emptor!

Copyright ©1991–2014 by P. Wessel, W. H. F. Smith, R. Scharroo, J. Luis and F. Wobbe

The Generic Mapping Tools (GMT) is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation.

The GMT package is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the file `LICENSE.TXT` in the GMT directory or the for more details.

Permission is granted to make and distribute verbatim copies of this manual provided that the copyright notice and these paragraphs are preserved on all copies. The GMT package may be included in a bundled distribution of software for which a reasonable fee may be charged.

GMT does not come with any warranties, nor is it guaranteed to work on your computer. The user assumes full responsibility for the use of this system. In particular, the University of Hawaii School of Ocean and Earth Science and Technology, the National Oceanic and Atmospheric Administration, EUMETSAT, the Universidade do Algarve, Alfred Wegener Institute, the National Science Foundation, Paul Wessel, Walter H. F. Smith, Remko Scharroo, Joaquim F. Luis, Florian Wobbe or any other individuals involved in the design and maintenance of GMT are NOT responsible for any damage that may follow from correct or incorrect use of these programs.

CHAPTER 3

Preface

While GMT has served the map-making and data processing needs of scientists since 1988 [1], the current global use was heralded by the first official release in *EOS Trans. AGU* in the fall of 1991. Since then, GMT has grown to become a standard tool for many users, particularly in the Earth and Ocean Sciences but the global collective of GMT users is incredibly diverse. Development has at times been rapid, and numerous releases have seen the light of day since the early versions. For a detailed history of the changes from release to release, see file `ChangeLog` in the main GMT directory. For a nightly snapshot of ongoing activity, see the online page. For a historical perspective of the origins and development of GMT see the video podcast "20 Years with GMT – The Generic Mapping Tools" produced following a seminar given by Paul Wessel on the 20th anniversary of GMT; a link is available on the GMT website.

The success of GMT is to a large degree due to the input of the user community. In fact, most of the capabilities and options in GMT programs originated as user requests. We would like to hear from you should you have any suggestions for future enhancements and modification. Please send your comments to the GMT help list or create an issue in the bug tracker (see http://gmt.soest.hawaii.edu/projects/gmt/issues/).

[1] Version 1.0 was then informally released at the Lamont-Doherty Earth Observatory.

CHAPTER 4

New Features in GMT 5

GMT 5 represents a new branch of GMT development that mostly preserves the capabilities of the previous versions while adding over 200 new features to an already extensive bag of tricks. Our PostScript library `PSL` has seen a complete rewrite as well and produce shorter and more compact PostScript. However, the big news is aimed for developers who wish to leverage GMT in their own applications. We have completely revamped the code base so that high-level GMT functionality is now accessible via GMT "modules". These are high-level functions named after their corresponding programs (e.g., `GMT_grdimage`) that contains all of the functionality of that program within the function. While currently callable from C/C++ only (with some support for F77), we are making progress on the Matlab interface as well and there are plans to start on the Python version. Developers should consult the `GMT API` documentation for more details.

We recommend that users of GMT 4 consider learning the new rules and defaults since eventually (in some years) GMT 4 will be obsolete. To ease the transition to GMT 5 you may run it in compatibility mode, thus allowing the use of many obsolete default names and command switches (you will receive a warning instead). This is discussed below.

Below are six key areas of improvements in GMT 5.

4.1 New programs

First, a few new programs have been added and some have been promoted (and possibly renamed) from earlier supplements:

`gmt` This is the **only** program executable that is distributed with GMT 5. To avoid problems with namespace conflicts (e.g., there are other, non-GMT programs with generic names like triangulate, surface, etc.) all GMT 5 modules are launched from the gmt executable via "gmt module" calls (e.g, gmt pscoast). For backwards compatibility (see below) we also offer symbolic links with the old executable names that simply point to the gmt program, which then can start the correct module. Any module whose name starts with "gmt" can be abbreviated, e.g., "gmt gmtconvert" may be called as "gmt convert".

`gmt2kml` A `psxy` -like tool to produce KML overlays for Google Earth. Previously found in the misc supplement.

`gmtconnect` Connect individual lines whose end points match within given tolerance. Previously known as gmtstitch in the misc supplement (this name is recognized when GMT is running in compatibility mode).

`gmtget` Return the values of the specified GMT defaults. Previously only implemented as a shell script and thus not available on all platforms.

`gmtinfo` Report information about data tables. Previously known by the name minmax (this name is still recognized when GMT is running in compatibility mode).

`gmtsimplify` A line-reduction tool for coastlines and similar lines. Previously found in the misc supplement under the name gmtdp (this name is recognized when GMT is running in compatibility mode).

`gmtspatial` Perform various geospatial operations on lines and polygons.

`gmtvector` Perform basic vector manipulation in 2-D and 3-D.

`gmtwhich` Return the full path to specified data files.

`grdraster` Extracts subsets from large global grids. Previously found in the dbase supplement.

`kml2gmt` Extract GMT data tables from Google Earth KML files. Previously found in the misc supplement.

`sph2grd` Compute grid from list of spherical harmonic coefficients [We will add its natural complement grd2sph at a later date].

`sphdistance` Make grid of distances to nearest points on a sphere. Previously found in the sph supplement.

`sphinterpolate` Spherical gridding in tension of data on a sphere. Previously found in the sph supplement.

`sphtriangulate` Delaunay or Voronoi construction of spherical lon,lat data. Previously found in the sph supplement.

We have also added a new supplement called potential that contains these five modules:

`gmtgravmag3d:` Compute the gravity/magnetic anomaly of a body by the method of Okabe.

`gravfft:` Compute gravitational attraction of 3-D surfaces and a little more by the method of Parker.

`grdgravmag3d:` Computes the gravity effect of one (or two) grids by the method of Okabe.

`grdredpol:` Compute the Continuous Reduction To the Pole, also known as differential RTP.

`grdseamount:` Compute synthetic seamount (Gaussian or cone, circular or elliptical) bathymetry.

Finally, the spotter supplement has added one new module:

`grdpmodeler:` Evaluate a plate model on a geographic grid.

4.2 New common options

First we discuss changes that have been implemented by a series of new lower-case GMT common options:

- Programs that read data tables can now process the aspatial metadata in OGR/GMT files with the new **-a** option. These can be produced by **ogr2ogr** (a GDAL tool) when selecting the -f “GMT” output format. See Appendix P. The GMT Vector Data Format for OGR Compatibility for an explanation of the OGR/GMT file format. Because all GIS information is encoded via GMT comment lines these files can also be used in GMT 4 (the GIS metadata is simply skipped).
- Programs that read or write data tables can specify a custom binary format using the enhanced **-b** option.

- Programs that read data tables can control which columns to read and in what order (and optionally supply scaling relations) with the new **-i** option
- Programs that read grids can use new common option **-n** to control grid interpolation settings and boundary conditions.
- Programs that write data tables can control which columns to write and in what order (and optionally supply scaling relations) with the new **-o** option.
- All plot programs can take a new **-p** option for perspective view from infinity. In GMT 4, only some programs could do this (e.g., `pscoast`) and it took a program-specific option, typically **-E** and sometimes an option **-Z** would be needed as well. This information is now all passed via **-p** and applies across all GMT plotting programs.
- Programs that read data tables can control how records with NaNs are handled with the new **-s** option.
- All plot programs can take a new **-t** option to modify the PDF transparency level for that layer. However, as PostScript has no provision for transparency you can only see the effect if you convert it to PDF.

4.3 Updated common options

Some of the established GMT common options have seen significant improvements; these include:

- The completely revised **-B** option can now handle irregular and custom annotations (see Section Custom axes). It also has a new automatic mode which will select optimal intervals given data range and plot size. The 3-D base maps can now have horizontal gridlines on xz and yz back walls.
- The **-R** option may now accept a leading unit which implies the given coordinates are projected map coordinates and should be replaced with the corresponding geographic coordinates given the specified map projection. For linear projections such units imply a simple unit conversion for the given coordinates (e.g., km to meter).
- Introduced **-fp**[*unit*] which allows data input to be in projected values, e.g., UTM coordinates while **-Ju** is given.

While just giving - (the hyphen) as argument presents just the synopsis of the command line arguments, we now also support giving + which in addition will list the explanations for all options that are not among the GMT common set.

4.4 New default parameters

Most of the GMT default parameters have changed names in order to group parameters into logical groups and to use more consistent naming. However, under compatibility mode (see below) the old names are still recognized. New capabilities have been implemented by introducing new GMT default settings:

- *DIR_DCW* specifies where to look for the optional Digital Charts of the World database (for country coloring or selections).
- *DIR_GSHHG* specifies where to look for the required Global Self-consistent Hierarchical High-resolution Geography database.

- *GMT_COMPATIBILITY* can be set to 4 to allow backwards compatibility with GMT 4 command-line syntax or 5 to impose strict GMT5 syntax checking.
- *IO_NC4_CHUNK_SIZE* is used to set the default chunk size for the **lat** and **lon** dimension of the **z** variable of netCDF version 4 files.
- *IO_NC4_DEFLATION_LEVEL* is used to set the compression level for netCDF4 files upon output.
- *IO_SEGMENT_MARKER* can be used to change the character that GMT uses to identify new segment header records [>].
- *MAP_ANNOT_ORTHO* controls whether axes annotations for Cartesian plots are horizontal or orthogonal to the individual axes.
- *GMT_FFT* controls which algorithms to use for Fourier transforms.
- *GMT_TRIANGULATE* controls which algorithm to use for Delaunay triangulation.
- Great circle distance approximations can now be fine-tuned via new GMT default parameters *PROJ_MEAN_RADIUS* and *PROJ_AUX_LATITUDE*. Geodesics are now even more accurate by using the Vincenty [1975] algorithm instead of Rudoe's method.
- *GMT_EXTRAPOLATE_VAL* controls what splines should do if requested to extrapolate beyond the given data domain.
- *PS_TRANSPARENCY* allows users to modify how transparency will be processed when converted to PDF [Normal].

4.5 General improvements

Other wide-ranging changes have been implemented in different ways, such as

- All programs now use consistent, standardized choices for plot dimension units (**c**m, **i**nch, or **p**oint; we no longer consider **m**eter a plot length unit), and actual distances (choose spherical arc lengths in **d**egree, **m**inute, and **s**econd [was **c**], or distances in m**e**ter [Default], **f**oot [new], **k**m, **M**ile [was sometimes **i** or **m**], **n**autical mile, and s**u**rvey foot [new]).
- Programs that read data tables can now process multi-segment tables automatically. This means programs that did not have this capability (e.g., `filter1d`) can now all operate on the segments separately; consequently, there is no longer a **-m** option.
- While we support the scaling of z-values in grids via the filename convention name[=*ID*[/*scale*/*offset*[/*nan*]]] mechanism, there are times when we wish to scale the x,y domain as well. Users can now append **+u***unit* to their gridfile names, where *unit* is one of non-arc units listed in Table *distunits*. This will convert your Cartesian x and y coordinates *from* the given unit *to* meters. We also support the inverse option **+U***unit*, which can be used to convert your grid coordinates *from* meters *to* the specified unit.
- CPT files also support the **+u**|**U***unit* mechanism. Here, the scaling applies to the z values. By appending these modifiers to your CPT filenames you can avoid having two CPT files (one for meter and one for km) since only one is needed.
- Programs that read grids can now directly handle Arc/Info float binary files (GRIDFLOAT) and ESRI .hdr formats.
- Programs that read grids now set boundary conditions to aid further processing. If a subset then the boundary conditions are taken from the surrounding grid values.

- All text can now optionally be filled with patterns and/or drawn with outline pens. In the past, only `pstext` could plot outline fonts via **-S***pen*. Now, any text can be an outline text by manipulating the corresponding FONT defaults (e.g., *FONT_TITLE*).
- All color or fill specifications may append @*transparency* to change the PDF transparency level for that item. See **-t** for limitations on how to visualize this transparency.
- GMT now ships with 36 standard color palette tables (CPT), up from 24.

4.6 Program-specific improvements

Finally, here is a list of numerous enhancements to individual programs:

- `blockmean` added **-Ep** for error propagation and **-Sn** to report the number of data points per block.
- `blockmedian` added **-Er**[-] to return as last column the record number that gave the median value. For ties, we return the record number of the higher data value unless **-Er**- is given (return lower). Added **-Es** to read and output source id for median value.
- `blockmode` added **-Er**[-] but for modal value. Added **-Es** to read and output source id for modal value.
- `gmtconvert` now has optional PCRE (regular expression) support, as well as a new option to select a subset of segments specified by segment running numbers (**-Q**) and improved options to extract a subset of records (**-E**) and support for a list of search strings via **-S+f***patternfile*.
- `gmtinfo` has new option **-A** to select what group to report on (all input, per file, or per segment). Also, use **-If** to report an extended region optimized for fastest results in FFTs. and **-Is** to report an extended region optimized for fastest results in `surface`. Finally, new option **-D**[*inc*] to align regions found via **-I** with the center of the data.
- `gmtmath` with **-N***ncol* and input files will add extra blank columns, if needed. A new option **-E** sets the minimum eigenvalue used by operators LSQFIT and SVDFIT. Option **-L** applies operators on a per-segment basis instead of accumulating operations across the entire file. Many new operators have been added (BITAND, BITLEFT, BITNOT, BITOR, BITRIGHT, BITTEST, BITXOR, DIFF, IFELSE, ISFINITE, SVDFIT, TAPER). Finally, we have implemented user macros for long or commonly used expressions, as well as ability to store and recall using named variables.
- `gmtselect` Takes **-E** to indicate if points exactly on a polygon boundary are inside or outside, and **-Z** can now be extended to apply to other columns than the third.
- `grd2cpt` takes **-F** to specify output color model and **-G** to truncate incoming CPT to be limited to a given range.
- `grd2xyz` takes **-C** to write row, col instead of x,y. Append **f** to start at 1, else start at 0. Alternatively, use **-Ci** to write just the two columns *index* and *z*, where *index* is the 1-D indexing that GMT uses when referring to grid nodes.
- `grdblend` can now take list of grids on the command line and blend, and now has more blend choices (see **-C**). Grids no longer have to have same registration or spacing.
- `grdclip` has new option **-Si** to set all data >= low and <= high to the *between* value, and **-Sr** to set all data == old to the *new* value.
- `grdcontour` can specify a single contour with **-C+***contour* and **-A+***contour*.

- `grdcut` can use **-S** to specify an origin and radius and cut the corresponding rectangular area, and **-N** to extend the region if the new **-R** domain exceeds existing boundaries.
- `grdfft` can now accept two grids and let **-E** compute the cross-spectra. The **-N** option allows for many new and special settings, including ability to control data mirroring, detrending, tapering, and output of intermediate results.
- `grdfilter` can now do spherical filtering (with wrap around longitudes and over poles) for non-global grids. We have also begun implementing Open MP threads to speed up calculations on multi-core machines. We have added rectangular filtering and automatic resampling to input resolution for high-pass filters. There is also **-Ff***weightgrd* which reads the gridfile *weightgrd* for a custom Cartesian grid convolution. The *weightgrd* must have odd dimensions. Similarly added **-Fo***opgrd* for operators (via coefficients in the grdfile *opgrd*) whose weight sum is zero (hence we do not sum and divide the convolution by the weight sum).
- `grdgradient` now has **-Em** that gives results close to ESRI's "hillshade"" (but faster).
- `grdinfo` now has modifier **-Ts***dz* which returns a symmetrical range about zero. Also, if **-Ib** is given then the grid's bounding box polygon is written.
- `grdimage` with GDAL support can write a raster image directly to a raster file (**-A**) and may plot raster images as well (**-Dr**). It also automatically assigns a color table if none is given and can use any of the 36 GMT color tables and scale them to fit the grid range.
- `grdmask` has new option **-Ni**|I|p|P to set inside of polygons to the polygon IDs. These may come from OGR aspatial values, segment head **-L**ID, or a running number, starting at a specified origin [0]. Now correctly handles polygons with perimeters and holes. Added z as possible radius value in **-S** which means read radii from 3rd input column.
- `grdmath` added many new operators such as BITAND, BITLEFT, BITNOT, BITOR, BITRIGHT, BITTEST, BITXOR, DEG2KM, IFELSE, ISFINITE, KM2DEG, and TAPER. Finally, we have implemented user macros for long or commonly used expressions, as well as ability to store and recall using named variables.
- `grdtrack` has many new options. The **-A** option controls how the input track is resampled when **-C** is selected, the new **-C**, **-D** options automatically create an equidistant set of cross-sectional profiles given input line segments; one or more grids can then be sampled at these locations. The **-E** option allows users to quickly specify tracks for sampling without needed input tracks. Also added **-S** which stack cross-profiles generated with **-C**. The **-N** will not skip points that are outside the grid domain but return NaN as sampled value. Finally, **-T** will return the nearest non-NaN node if the initial location only finds a NaN value.
- `grdvector` can now take **-Si***scale* to give the reciprocal scale, i.e., cm/ unit or km/unit. Also, the vector heads in GMT have completely been overhauled and includes geo-vector heads that follow great or small circles.
- `grdview` will automatically assigns a color table if none is given and can use any of the 36 GMT color tables and scale them to fit the grid range.
- `grdvolume` can let **-S** accept more distance units than just km. It also has a modified **-T**[**c**|**h**] for ORS estimates based on max curvature or height. **-Cr** to compute the *outside* volume between two contours (for instances, the volume of water from a bathymetry grid).
- `greenspline` has an improved **-C** option to control how many eigenvalues are used in the fitting, and **–Sl** adds a linear (or bilinear) spline.
- `makecpt` has a new **-F** option to specify output color representation, e.g., to output the CPT table in h-s-v format despite originally being given in r/g/b, and **-G** to truncate incoming CPT to be

limited to a given range. It also adds **Di** to match the bottom/top values in the input cpt file.

- `mapproject` has a new **-N** option to do geodetic/geocentric conversions; it combines with **-I** for inverse conversions. Also, we have extended **-A** to accept **-Ao**| **O** to compute line orientations (-90/90). In **-G**, prepend - to the unit for (fast) flat Earth or + for (slow) geodesic calculations.
- `project` has added **-G**...[+] so if + is appended we get a segment header with information about the pole for the circle. Optionally, append /*colat* in **-G** for a small circle path.
- `ps2raster` has added a **-TF** option to create multi-page PDF files. There is also modification to **-A** to add user-specified margins, and it automatically detects if transparent elements have been included (and a detour via PDF might be needed).
- `psbasemap` has added a **-D** option to place a map-insert box.
- `psclip` has added an extended **-C** option to close different types of clip paths.
- `pscoast` has added a new option **-F** which lets users specify one or more countries to paint, fill, extract, or use as plot domain (requires DCW to be installed).
- `pscontour` is now similar to `grdcontour` in the options it takes, e.g., **-C** in particular. In GMT 4, the program could only read a CPT file and not take a specific contour interval.
- `pshistogram` now takes **-D** to place histogram count labels on top of each bar and **-N** to draw the equivalent normal distributions.
- `pslegend` no longer uses system calls to do the plotting. The modified **-D** allows for minor offsets, while **-F** offers more control over the frame and fill.
- `psrose` has added **-Wv***pen* to specify pen for vector (specified in **-C**). Added **-Zu** to set all radii to unity (i.e., for analysis of angles only).
- `psscale` has a new option **-T** that paints a rectangle behind the color bar. The **+n** modifier to **-E** draws a rectangle with NaN color and adds a label. The **-G** option will truncate incoming CPT to be limited to the specified z-range. Modification **-Np** indicates a preference to use polygons to draw the color bar.
- `pstext` can take simplified input via new option **-F** to set fixed font (including size), angle, and justification. If these parameters are fixed for all the text strings then the input can simply be *x y text*. It also has enhanced **-DJ** option to shorten diagonal offsets by $\sqrt{2}$ to maintain the same radial distance from point to annotation. Change all text to upper or lower case with **-Q**.
- `psxy` and `psxyz` both support the revised custom symbol macro language which has been expanded considerably to allow for complicated, multi-parameter symbols; see Appendix M. Custom Plot Symbols for details. Finally, we allow the base for bars and columns optionally to be read from data file by not specifying the base value.
- `sample1d` offers **-A** to control resampling of spatial curves (with **-I**).
- `spectrum1d` has added **-L** to control removal of trend, mean value or mid value.
- `surface` has added **-r** to create pixel-registered grids and knows about periodicity in longitude (given **-fg**). There is also **-D** to supply a “sort” break line.
- `triangulate` now offers **-S** to write triangle polygons and can handle 2-column input if **-Z** is given. Can also write triangle edges as line with **-M**.
- `xyz2grd` now also offers **-Af** (first value encountered), **-Am** (mean, the default), **-Ar** (rms), and **-As** (last value encountered).

Several supplements have new features as well:

- `img2grd` used to be a shell script but is now a C program and can be used on all platforms.
- `mgd77convert` added **-C** option to assemble *.mgd77 files from *.h77/*.a77 pairs.
- The spotter programs can now read GPlates rotation files directly as well as write this format. Also, `rotconverter` can extract plate circuit rotations on-the-fly from the GPlates rotation file.

Note: GMT 5 only produces PostScript and no longer has a setting for Encapsulated PostScript (EPS). We made this decision since (a) our EPS determination was always very approximate (no consideration of font metrics, etc.) and quite often wrong, and (b) `ps2raster` handles it exactly. Hence, users who need EPS plots should simply process their PostScript files via `ps2raster`.

4.7 Incompatibilities between GMT 5 and GMT 4

As features are added and bugs are discovered, it is occasionally necessary to break the established syntax of a GMT program option, such as when the intent of the option is non-unique due to a modifier key being the same as a distance unit indicator. Other times we see a greatly improved commonality across similar options by making minor adjustments. However, we are aware that such changes may cause grief and trouble with established scripts and the habits of many GMT users. To alleviate this situation we have introduced a configuration that allows GMT to tolerate and process obsolete program syntax (to the extent possible). To activate you must make sure *GMT_COMPATIBILITY* is set to 4 in your gmt.conf file. When not running in compatibility mode any obsolete syntax will be considered as errors. We recommend that users with prior GMT 4 experience run GMT 5 in compatibility mode, heed the warnings about obsolete syntax, and correct their scripts or habits accordingly. When this transition has been successfully navigated it is better to turn compatibility mode off and leave the past behind. Occasionally, users will supply an ancient GMT 3 syntax which may have worked in GMT 4 but is not honored in GMT 5.

Here are a list of known incompatibilities that are correctly processed with a warning under compatibility mode:

- GMT **default names**: We have organized the default parameters logically by group and renamed several to be easier to remember and to group. Old and new names can be found in Table *obsolete*. In addition, a few defaults are no longer recognized, such as N_COPIES, PS_COPIES, DOTS_PR_INCH, GMT_CPTDIR, PS_DPI, and PS_EPS, TRANSPARENCY. This also means the old common option **-c** for specifying PostScript copies is no longer available.
- **Units**: The unit abbreviation for arc seconds is finally **s** instead of **c**, with the same change for upper case in some clock format statements.
- **Contour labels**: The modifiers **+k***fontcolor* and **+s***fontsize* are obsolete, now being part of **+f***font*.
- **Ellipsoids**: Assigning *PROJ_ELLIPSOID* a file name is deprecated, use comma-separated parameters a, f^{-1} instead.
- **Custom symbol macros:** Circle macro symbol **C** is deprecated; use **c** instead.
- **Map scale**: Used by `psbasemap` and others. Here, the unit **m** is deprecated; use **M** for statute miles.
- **3-D perspective**: Some programs used a combination of **-E**, **-Z** to set up a 3-D perspective view, but these options were not universal. The new 3-D perspective in GMT 5 means you instead use the common option **-p** to configure the 3-D projection.

- **Pixel vs. gridline registration:** Some programs used to have a local **-F** to turn on pixel registration; now this is a common option **-r**.
- **Table file headers**: For consistency with other common i/o options we now use **-h** instead of **-H**.
- **Segment headers**: These are now automatically detected and hence there is no longer a **-m** (or the older **-M** option).
- **Front symbol**: The syntax for the front symbol has changed from **-Sf***spacing/size*[**+d**][**+t**][:*offset*] to **-Sf***spacing*[/*size*][**+r+l**][**+f+t+s+c+b**][**+o***offset*].
- **Vector symbol**: With the introduction of geo-vectors there are three kinds of vectors that can be drawn: Cartesian (straight) vectors with **-Sv** or **-SV**, geo-vectors (great circles) with **-S=**, and circular vectors with **-Sm**. These are all composed of a line (controlled by pen settings) and 0–2 arrow heads (control by fill and outline settings). Many modifiers common to all arrows have been introduced using the **+key**[*arg*] format. The *size* of a vector refers to the length of its head; all other quantities are given via modifiers (which have sensible default values). In particular, giving size as *vectorwidth/headlength/headwidth* is deprecated. See the `psxy` man page for a clear description of all modifiers.
- `blockmean`: The **-S** and **-Sz** options are deprecated; use **-Ss** instead.
- `filter1d`: The **-N***ncol/tcol* option is deprecated; use **-N***tcol* instead as we automatically determine the number of columns in the file.
- `gmtconvert`: **-F** is deprecated; use common option **-o** instead.
- `gmtdefaults`: **-L** is deprecated; this is now the default behavior.
- `gmtmath`: **-F** is deprecated; use common option **-o** instead.
- `gmtselect`: **-Cf** is deprecated; use common specification format **-C-** instead. Also, **-N**...**o** is deprecated; use **-E** instead.
- `grd2xyz`: **-E** is deprecated as the ESRI ASCII exchange format is now detected automatically.
- `grdcontour`: **-m** is deprecated as segment headers are handled automatically.
- `grdfft`: **-M** is deprecated; use common option **-fg** instead.
- `grdgradient`: **-L** is deprecated; use common option **-n** instead. Also, **-M** is deprecated; use common option **-fg** instead.
- `grdlandmask`: **-N**...**o** is deprecated; use **-E** instead.
- `grdimage`: **-S** is deprecated; use **-n***mode*[**+a**][**+t***threshold*] instead.
- `grdmath`: LDIST and PDIST now return distances in spherical degrees; while in GMT 4 it returned km; use DEG2KM for conversion, if needed.
- `grdproject`: **-S** is deprecated; use **-n***mode*[**+a**][**+t***threshold*] instead. Also, **-N** is deprecated; use **-D** instead.
- `grdsample`: **-Q** is deprecated; use **-n***mode*[**+a**][**+t***threshold*] instead. Also, **-L** is deprecated; use common option **-n** instead, and **-N***nx*>/<*ny* is deprecated; use **-I***nx*+>/<*ny*+ instead.
- `grdtrack`: **-Q** is deprecated; use **-n***mode*[**+a**][**+t***threshold*] instead. Also, **-L** is deprecated; use common option **-n** instead, and **-S** is deprecated; use common option **-sa** instead.
- `grdvector`: **-E** is deprecated; use the vector modifier **+jc** as well as the general vector specifications discussed earlier.

- `grdview`: **-L** is deprecated; use common option **-n** instead.
- `nearneighbor`: **-L** is deprecated; use common option **-n** instead.
- `project`: **-D** is deprecated; use **--***FORMAT_GEO_OUT* instead.
- `psbasemap`: **-G** is deprecated; specify canvas color via **-B** modifier **+g***color*.
- `pscoast`: **-m** is deprecated and have reverted to **-M** for selecting data output instead of plotting.
- `pscontour`: **-T***indexfile* is deprecated; use **-Q***indexfile*.
- `pshistogram`: **-T***col* is deprecated; use common option **-i** instead.
- `pslegend`: Paragraph text header flag > is deprecated; use P instead.
- `psmask`: **-D**...**+n***min* is deprecated; use **-Q** instead.
- `psrose`: Old vector specifications in Option **-M** are deprecated; see new explanations.
- `pstext`: **-m** is deprecated; use **-M** to indicate paragraph mode. Also, **-S** is deprecated as fonts attributes are now specified via the font itself.
- `pswiggle`: **-D** is deprecated; use common option **-g** to indicate data gaps. Also, **-N** is deprecated as all fills are set via the **-G** option.
- `psxy`: Old vector specifications in Option **-S** are deprecated; see new explanations.
- `psxyz`: Old vector specifications in Option **-S** are deprecated; see new explanations.
- `splitxyz`: **-G** is deprecated; use common option **-g** to indicate data gaps. Also, **-M** is deprecated; use common option **-fg** instead.
- `triangulate`: **-m** is deprecated; use **-M** to output triangle vertices.
- `xyz2grd`: **-E** is deprecated as the ESRI ASCII exchange format is one of our recognized formats. Also, **-A** (no arguments) is deprecated; use **-Az** instead.
- `grdraster`: Now in the main GMT core. The **H***skip* field in `grdraster.info` is no longer expected as we automatically determine if a raster has a GMT header. Also, to output *x,y,z* triplets instead of writing a grid now requires **-T**.
- `img2grd`: **-m***inc* is deprecated; use **-I***inc* instead.
- `psvelo`: Old vector specifications are deprecated; see new explanations.
- `mgd77convert`: **-4** is deprecated; use **-D** instead.
- `mgd77list`: The unit **m** is deprecated; use **M** for statute miles.
- `mgd77manage`: The unit **m** is deprecated; use **M** for statute miles. The **-Q** is deprecated; use **-n***mode*[**+a**][**+t***threshold*] instead
- `mgd77path`: **-P** is deprecated (clashes with GMT common options); use **-A** instead.
- `backtracker`: **-C** is deprecated as stage vs. finite rotations are detected automatically.
- `grdrotater`: **-C** is deprecated as stage vs. finite rotations are detected automatically. Also, **-T***lon/lat/angle* is now set via **-e***lon/lat/angle*.
- `grdspotter`: **-C** is deprecated as stage vs. finite rotations are detected automatically.
- `hotspotter`: **-C** is deprecated as stage vs. finite rotations are detected automatically.
- `originator`: **-C** is deprecated as stage vs. finite rotations are detected automatically.

- `rotconverter`: **-Ff** selection is deprecated, use **-Ft** instead.
- `x2sys_datalist`: The unit **m** is deprecated; use **M** for statute miles.

Old Name	New Name
ANNOT_FONT_PRIMARY	**FONT_ANNOT_PRIMARY**
ANNOT_FONT_SECONDARY	**FONT_ANNOT_SECONDARY**
ANNOT_FONT_SIZE_PRIMARY	**FONT_ANNOT_PRIMARY**
ANNOT_FONT_SIZE_SECONDARY	**FONT_ANNOT_SECONDARY**
ANNOT_MIN_ANGLE	**MAP_ANNOT_MIN_SPACING**
ANNOT_OFFSET_PRIMARY	**MAP_ANNOT_OFFSET_PRIMARY**
ANNOT_OFFSET_SECONDARY	**MAP_ANNOT_OFFSET_SECONDARY**
BASEMAP_AXES	**MAP_FRAME_AXES**
BASEMAP_FRAME_RGB	**MAP_DEFAULT_PEN**
BASEMAP_TYPE	**MAP_FRAME_TYPE**
CHAR_ENCODING	**PS_CHAR_ENCODING**
D_FORMAT	**FORMAT_FLOAT_OUT**
DEGREE_SYMBOL	**MAP_DEGREE_SYMBOL**
ELLIPSOID	**PROJ_ELLIPSOID**
FIELD_DELIMITER	**IO_COL_SEPARATOR**
FRAME_PEN	**MAP_FRAME_PEN**
FRAME_WIDTH	**MAP_FRAME_WIDTH**
GLOBAL_Y_SCALE	**PS_SCALE_X**
GLOBAL_X_SCALE	**PS_SCALE_X**
GRID_CROSS_SIZE_PRIMARY	**MAP_GRID_CROSS_SIZE_PRIMARY**
GRID_CROSS_SIZE_SECONDARY	**MAP_GRID_CROSS_SIZE_SECONDARY**
GRID_PEN_PRIMARY	**MAP_GRID_PEN_PRIMARY**
GRID_PEN_SECONDARY	**MAP_GRID_PEN_SECONDARY**
GRIDFILE_FORMAT	**IO_GRIDFILE_FORMAT**
GRIDFILE_SHORTHAND	**IO_GRIDFILE_SHORTHAND**
HEADER_FONT_SIZE	**FONT_TITLE**
HEADER_FONT	**FONT_TITLE**
HEADER_OFFSET	**MAP_TITLE_OFFSET**
HISTORY	**GMT_HISTORY**
HSV_MAX_SATURATION	**COLOR_HSV_MAX_S**
HSV_MAX_VALUE	**COLOR_HSV_MAX_V**
HSV_MIN_SATURATION	**COLOR_HSV_MIN_S**
HSV_MIN_VALUE	**COLOR_HSV_MIN_V**
INPUT_CLOCK_FORMAT	**FORMAT_CLOCK_IN**
INPUT_DATE_FORMAT	**FORMAT_DATE_IN**
INTERPOLANT	**GMT_INTERPOLANT**
INTERPOLANT	**GMT_INTERPOLANT**
INTERPOLANT	**GMT_INTERPOLANT**
LABEL_FONT	**FONT_LABEL**
LABEL_OFFSET	**MAP_LABEL_OFFSET**
LINE_STEP	**MAP_LINE_STEP**
MAP_SCALE_FACTOR	**PROJ_SCALE_FACTOR**
MEASURE_UNIT	**PROJ_LENGTH_UNIT**
NAN_RECORDS	**IO_NAN_RECORDS**
OBLIQUE_ANNOTATION	**MAP_ANNOT_OBLIQUE**
OUTPUT_CLOCK_FORMAT	**FORMAT_CLOCK_OUT**

Continued on next page

Table 4.1 – continued from previous page

Old Name	New Name
OUTPUT_CLOCK_FORMAT	**FORMAT_CLOCK_OUT**
OUTPUT_DATE_FORMAT	**FORMAT_DATE_OUT**
OUTPUT_DEGREE_FORMAT	**FORMAT_GEO_OUT**
PAGE_COLOR	**PS_PAGE_COLOR**
PAGE_ORIENTATION	**PS_PAGE_ORIENTATION**
PAPER_MEDIA	**PS_MEDIA**
PLOT_CLOCK_FORMAT	**FORMAT_CLOCK_MAP**
PLOT_DATE_FORMAT	**FORMAT_DATE_MAP**
PLOT_DEGREE_FORMAT	**FORMAT_GEO_MAP**
POLAR_CAP	**MAP_POLAR_CAP**
PS_COLOR	**COLOR_HSV_MAX_V**
TICK_LENGTH	**MAP_TICK_LENGTH_PRIMARY|SECONDARY**
TICK_PEN	**MAP_TICK_PEN_PRIMARY|SECONDARY**
TIME_FORMAT_PRIMARY	**FORMAT_TIME_PRIMARY_MAP**
TIME_FORMAT_SECONDARY	**FORMAT_TIME_SECONDARY_MAP**
UNIX_TIME_FORMAT	**FORMAT_TIME_STAMP**
UNIX_TIME_POS	**MAP_LOGO_POS**
UNIX_TIME	**MAP_LOGO**
VECTOR_SHAPE	**MAP_VECTOR_SHAPE**
VERBOSE	**GMT_VERBOSE**
WANT_LEAP_SECONDS	**TIME_LEAP_SECONDS**
X_ORIGIN	**MAP_ORIGIN_X**
XY_TOGGLE	**IO_LONLAT_TOGGLE**
Y_AXIS_TYPE	**MAP_ANNOT_ORTHO**
Y_ORIGIN	**MAP_ORIGIN_Y**
Y2K_OFFSET_YEAR	**TIME_Y2K_OFFSET_YEAR**

Note: While **TIME_LEAP_SECONDS** is a recognized keyword it is currently not implemented and has no effect. We reserve the right to enable this feature in the future.

CHAPTER 5

Switching between different GMT versions

We encourage all GMT users to start using version 5 immediately; it has been tested extensively by the GMT team and has benefitted from bug reports for the 4.5.x versions. Users who still worry about the new version breaking things may install GMT 4.5.x and 5 side by side.

Because GMT 5 is backwards compatible with the 4.5.x series (provided you configured it that way) yet maintains its parameters and history in separate files (e.g. `.gmtdefaults4`, versus `gmt.conf`) it is possible to install and use both versions on the same workstation. Switching between different GMT versions can be accomplished in several ways, two of which will be addressed here:

1. By using the `gmtswitch` utility to select the current working version. **Pro:** easy, interactive way to switch versions on the command line; works with previous GMT syntax. **Con:** editing of shell startup files required; needs write access in `$HOME`-directory; manual intervention necessary if symlink `$HOME/this_gmt` is broken.
2. By using the recommended `gmt <module>`-syntax in conjunction with a shell helper function that points to the desired GMT executable. **Pro:** no need to create symlinks and edit shell startup files; scripts are more portable. **Con:** different syntax required.

5.1 Setup of gmtswitch

Run `gmtswitch` after you have finished installing all GMT versions of interest. The first time you run `gmtswitch` it will try to find all the available versions installed on your file system. The versions found will be listed in the file `.gmtversions` in your home directory; each line is the full path to a GMT root directory (e.g., /usr/local/GMT4.5.9). You may manually add or remove entries there at any time. You are then instructed to make two changes to your environment (the details are shell-dependent but explained by `gmtswitch`):

1. `gmtswitch` creates and maintains a symbolic link `this_gmt` in your home directory that will point to a directory with one of the installed GMT versions.
2. Make sure `$HOME/this_gmt/bin` is in your executable PATH.

Make those edits, logout, and login again. The next time you run `gmtswitch` you will be able to switch between versions. Typing `gmtswitch` with no argument will list the available versions in a numerical menu and prompt you to choose one, whereas `gmtswitch` *version* will immediately switch to that version (*version* must be a piece of unique text making up the full path to a version, e.g., 4.5.9). If you use **bash**, **tcsh**, or **csh** you may have to type `hash -r` or `rehash` to initiate the path changes.

On Windows, the process is somewhat similar. The GMT bin directory has one batch file `gmtswitch.bat` that works by changing the Windows PATH variable so that the BIN directory of the preferred version always comes first. To do that the batch works in two alternative modes.

1. Permanent mode
2. Temporary mode

The permanent mode makes use of the free executable program EditPath to change the user path in the registry. It's called permanent because the changes remains until ... next change. Off course the editpath.exe binary must be in your system's path as well. WARNING: The path change will not be visible on the shell cmd where it was executed. For the change to be active you will need to open a new cmd window.

The second mode is temporary because the path to the selected GMT binary dir is prepended to previous path via a shell command line. This modification disappears when the shell cmd window where it was executes is deleted.

For further details the user should read the entire help section at the header of the `gmtswitch.bat`.

The `gmtswitch.bat` solution, however, has the drawback that the batch file must be located elsewhere and in a directory that is on the user's PATH, otherwise it wont be located after first use unless the other GMT bin directory has a similar batch file. A better solution is to install the Windows console enhancement that includes multiple tabs and configure the different tabs to start the different GMT versions. All it takes is in the Tab setting to call a batch that modifies the PATH locally. That PATH modifying batch will have a single line with something like:

```
set path=C:\programs\gmt5\bin;%PATH%
```

5.2 Version selection with helper function

A shell function can be used as a wrapper around the gmt executable. This even works when a gmt application is in the search PATH as it would shadow the real command. This method can easily be applied on the command line or in scripts when the recommended `gmt <module>`-syntax is used. Shell scripts using old-style GMT commands would have to be converted first. The syntax conversion can be accomplished with the `gmt5syntax` utility. A suitable bash wrapper function for GMT 5 would look like this:

```
function gmt () { /path/to/gmt5/bin/gmt "$@"; }
export -f gmt
```

Exporting the function is necessary to make it available to subshells and scripts. This wrapper function can be either set in your working shell or inside a GMT shell script. The latter is useful to switch to a certain GMT version on a per-script basis.

For GMT releases prior to GMT 5 which only provide the module commands, we need a slightly modified version of the wrapper script:

```
function gmt () { module=$1; shift; /path/to/gmt4/bin/${module} "$@"; }
export -f gmt
```

On the command line this might be too much typing to switch between versions. So we might as well put everything together in a script file `gmtfun`:

```
case $1 in
  4)
  function gmt () {
    module=$1; shift; /path/to/gmt4/bin/${module} "$@"
  }
  ;;
  5)
  function gmt() {
    /path/to/gmt5/bin/gmt "$@"
  }
  ;;
  *)
  return
  ;;
esac
export -f gmt
```

Source the file with either `. gmtfun 4` or `. gmtfun 5` to switch between versions.

CHAPTER 6

Introduction

Most scientists are familiar with the sequence: *raw data* $\rightarrow$ *processing* $\rightarrow$ *final illustration*. In order to finalize papers for submission to scientific journals, prepare proposals, and create overheads and slides for various presentations, many scientists spend large amounts of time and money to create camera-ready figures. This process can be tedious and is often done manually, since available commercial or in-house software usually can do only part of the job. To expedite this process we introduce the Generic Mapping Tools (GMT for short), which is a free [1], software package that can be used to manipulate columns of tabular data, time-series, and gridded data sets, and display these data in a variety of forms ranging from simple *x*–*y* plots to maps and color, perspective, and shaded-relief illustrations. GMT uses the PostScript page description language [*Adobe Systems Inc.*, 1990]. With PostScript, multiple plot files can easily be superimposed to create arbitrarily complex images in gray tones or 24-bit true color. Line drawings, bitmapped images, and text can be easily combined in one illustration. PostScript plot files are device-independent: The same file can be printed at 300 dots per inch (dpi) on an ordinary laserwriter or at 2470 dpi on a phototypesetter when ultimate quality is needed. GMT software is written as a set of UNIX tools [2] and is totally self-contained and fully documented. The system is offered free of charge and is distributed over the computer network (Internet) [*Wessel and Smith, 1991; 1995; 1998*].

The original version 1.0 of GMT was released in the summer of 1988 when the authors were graduate students at Lamont-Doherty Earth Observatory of Columbia University. During our tenure as graduate students, L-DEO changed its computing environment to a distributed network of UNIX workstations, and we wrote GMT to run in this environment. It became a success at L-DEO, and soon spread to numerous other institutions in the US, Canada, Europe, and Japan. The current version benefits from the many suggestions contributed by users of the earlier versions, and now includes more than 50 tools, more than 30 projections, and many other new, more flexible features. GMT provides scientists with a variety of tools for data manipulation and display, including routines to sample, filter, compute spectral estimates, and determine trends in time series, grid or triangulate arbitrarily spaced data, perform mathematical operations (including filtering) on 2-D data sets both in the space and frequency domain, sample surfaces along arbitrary tracks or onto a new grid, calculate volumes, and find trend surfaces. The plotting programs will let the user make linear, $\log_{10}$, and $x^a - y^b$ diagrams, polar and rectangular histograms, maps with filled continents and coastlines choosing from many common map projections, contour plots, mesh plots, monochrome or color images, and artificially illuminated shaded-relief and 3-D perspective illustrations.

GMT is written in the highly portable ANSI C programming language [*Kernighan and Ritchie*, 1988], is fully POSIX compliant [*Lewine*, 1991], has no Year 2000 problems, and may be used with any hardware running some flavor of UNIX, possibly with minor modifications. In writing GMT, we have followed the modular design philosophy of UNIX: The *raw data* $\rightarrow$ *processing* $\rightarrow$ *final illustration* flow is broken

[1] See GNU Lesser General Public License (http://www.gnu.org/copyleft/gpl.html) for terms on redistribution and modifications.

[2] The tools can also be installed on other platforms (see Appendix [app:L]).

down to a series of elementary steps; each step is accomplished by a separate GMT or UNIX tool. This modular approach brings several benefits: (1) only a few programs are needed, (2) each program is small and easy to update and maintain, (3) each step is independent of the previous step and the data type and can therefore be used in a variety of applications, and (4) the programs can be chained together in shell scripts or with pipes, thereby creating a process tailored to do a user-specific task. The decoupling of the data retrieval step from the subsequent massage and plotting is particularly important, since each institution will typically have its own data base formats. To use GMT with custom data bases, one has only to write a data extraction tool which will put out data in a form readable by GMT (discussed below). After writing the extractor, all other GMT modules will work as they are.

GMT makes full use of the PostScript page description language, and can produce color illustrations if a color PostScript device is available. One does not necessarily have to have access to a top-of-the-line color printer to take advantage of the color capabilities offered by GMT: Several companies offer imaging services where the customer provides a PostScript plot file and gets color slides or hardcopies in return. Furthermore, general-purpose PostScript raster image processors (RIPs) are now becoming available, letting the user create raster images from PostScript and plot these bitmaps on raster devices like computer screens, dot-matrix printers, large format raster plotters, and film writers [3]. Because the publication costs of color illustrations are high, GMT offers 90 common bit and hachure patterns, including many geologic map symbol types, as well as complete graytone shading operations. Additional bit and hachure patterns may also be designed by the user. With these tools, it is possible to generate publication-ready monochrome originals on a common laserwriter.

GMT is thoroughly documented and comes with a technical reference and cookbook which explains the purpose of the package and its many features, and provides numerous examples to help new users quickly become familiar with the operation and philosophy of the system. The cookbook contains the shell scripts that were used for each example; PostScriptfiles of each illustration are also provided. All programs have individual manual pages which can be installed as part of the on-line documentation under the UNIX **man** utility or as web pages. In addition, the programs offer friendly help messages which make them essentially self-teaching – if a user enters invalid or ambiguous command arguments, the program will print a warning to the screen with a synopsis of the valid arguments. All the documentation is available for web browsing and may be installed at the user's site.

The processing and display routines within GMT are completely general and will handle any (*x,y*) or (*x,y,z*) data as input. For many purposes the (*x,y*) coordinates will be (longitude, latitude) but in most cases they could equally well be any other variables (e.g., wavelength, power spectral density). Since the GMTplot tools will map these (*x,y*) coordinates to positions on a plot or map using a variety of transformations (linear, log-log, and several map projections), they can be used with any data that are given by two or three coordinates. In order to simplify and standardize input and output, GMT uses two file formats only. Arbitrary sequences of (*x,y*) or (*x,y,z*) data are read from multi-column ASCII tables, i.e., each file consists of several records, in which each coordinate is confined to a separate column [4]. This format is straightforward and allows the user to perform almost any simple (or complicated) reformatting or processing task using standard UNIX utilities such as **cut**, **paste**, **grep**, **sed** and **awk**. Two-dimensional data that have been sampled on an equidistant grid are read and written by GMT in a binary grid file using the functions provided with the netCDF library (a free, public-domain software library available separately from UCAR, the University Corporation of Atmospheric Research [*Treinish and Gough*, 1987]). This XDR (External Data Representation) based format is architecture independent, which allows the user to transfer the binary data files from one computer system to another [5]. GMT contains programs that will read ASCII (*x,y,z*) files and produce grid files. One such program, `surface`, includes new modifications to the gridding algorithm developed by *Smith and Wessel* [1990] using continuous splines in tension.

[3] One public-domain RIP is ghostscript, available from http://www.gnu.org/.

[4] Programs now also allow for fast, binary multicolumn file i/o.

[5] While the netCDF format is the default, many other formats are also possible.

Most of the programs will produce some form of output, which falls into four categories. Several of the programs may produce more than one of these types of output:

- 1-D ASCII Tables — For example, a (*x*,*y*) series may be filtered and the filtered values output. ASCII output is written to the standard output stream.
- 2-D binary (netCDF or user-defined) grid files – Programs that grid ASCII (*x*,*y*,*z*) data or operate on existing grid files produce this type of output.
- PostScript – The plotting programs all use the PostScriptpage description language to define plots. These commands are stored as ASCII text and can be edited should you want to customize the plot beyond the options available in the programs themselves.
- Reports – Several GMT programs read input files and report statistics and other information. Nearly all programs have an optional "verbose" operation, which reports on the progress of computation. All programs feature usage messages, which prompt the user if incorrect commands have been given. Such text is written to the standard error stream and can therefore be separated from ASCII table output.

GMT is available over the Internet at no charge. To obtain a copy, goto GMT home page http://gmt.soest.hawaii.edu/ and follow instructions. We also maintain two electronic mailing lists you may subscribe to in order to stay informed about bug fixes and upgrades.

For those without net-access that need to obtain GMT: Geoware makes and distributes CD-R and DVD-R media with the GMT package, compatible supplements, and several Gb of useful Earth and ocean science data sets. For more information send e-mail to geoware@geoware-online.com.

GMT has served a multitude of scientists very well, and their responses have prompted us to develop these programs even further. It is our hope that the new version will satisfy these users and attract new users as well. We present this system to the community in order to promote sharing of research software among investigators in the US and abroad.

6.1 References

- Adobe Systems Inc., *PostScript Language Reference Manual*, 2nd edition, 764, Addison-Wesley, Reading, Massachusetts, 1990.
- Kernighan, B. W., and D. M. Ritchie, *The C programming language*, 2nd edition, 272, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.
- Lewine, D., POSIX programmer's guide, 1st edition, 607, O'Reilly & Associates, Sebastopol, California, 1991.
- Treinish, L. A., and M. L. Gough, A software package for the data-independent management of multidimensional data, *EOS Trans. AGU*, 68(28), 633–635, 1987. doi:10.1029/EO068i028p00633.

GMT Overview and Quick Reference

7.1 GMT summary

The following is a summary of all the programs supplied with GMT and a very short description of their purpose. For more details, see the individual UNIX manual pages or the online web documentation. For a listing sorted by program purpose, see Section GMT quick reference.

`blockmean`	L_2 (*x*,*y*,*z*) table data filter/decimator
`blockmedian`	L_1 (*x*,*y*,*z*) table data filter/decimator
`blockmode`	Mode estimate (*x*,*y*,*z*) table data filter/decimator
`filter1d`	Filter 1-D table data sets (time series)
`fitcircle`	Finds the best-fitting great or small circle for a set of points
`gmt2kml`	Like `psxy` but plots KML for use in Google Earth
`gmtconnect`	Connect segments into more complete lines or polygons
`gmtconvert`	Convert data tables from one format to another
`gmtdefaults`	List the current default settings
`gmtget`	Retrieve selected parameters in current `gmt.conf` file
`gmtinfo`	Get information about table data files
`gmtmath`	Mathematical operations on table data
`gmtselect`	Select subsets of table data based on multiple spatial criteria
`gmtset`	Change selected parameters in current `gmt.conf` file
`gmtsimplify`	Line reduction using the Douglas-Peucker algorithm
`gmtspatial`	Geospatial operations on lines and polygons
`gmtvector`	Basic operations on vectors in 2-D and 3-D
`gmtwhich`	Find full path to specified data files
`grd2cpt`	Make color palette table from a grid files
`grd2rgb`	Convert Sun raster or grid file to red, green, blue component grids
`grd2xyz`	Conversion from 2-D grid file to table data
`grdblend`	Blend several partially over-lapping grid files onto one grid
`grdclip`	Limit the *z*-range in gridded data sets
`grdcontour`	Contouring of 2-D gridded data sets
`grdcut`	Cut a sub-region from a grid file
`grdedit`	Modify header information in a 2-D grid file
`grdfft`	Perform operations on grid files in the frequency domain
`grdfilter`	Filter 2-D gridded data sets in the space domain
`grdgradient`	Compute directional gradient from grid files
`grdhisteq`	Histogram equalization for grid files

Continued on next page

Table 7.1 – continued from previous page

grdimage	Produce images from 2-D gridded data sets
grdinfo	Get information about grid files
grdlandmask	Create masking grid files from shoreline data base
grdmask	Reset grid nodes in/outside a clip path to constants
grdmath	Mathematical operations on grid files
grdpaste	Paste together grid files along a common edge
grdproject	Project gridded data sets onto a new coordinate system
grdreformat	Converts grid files into other grid formats
grdsample	Resample a 2-D gridded data set onto a new grid
grdtrack	Sampling of 2-D gridded data set(s) along 1-D track
grdtrend	Fits polynomial trends to grid files
grdvector	Plotting of 2-D gridded vector fields
grdview	3-D perspective imaging of 2-D gridded data sets
grdvolume	Calculate volumes under a surface within specified contour
greenspline	Interpolation with Green's functions for splines in 1–3 D
kml2gmt	Extracts coordinates from Google Earth KML files
makecpt	Make color palette tables
mapproject	Transformation of coordinate systems for table data
nearneighbor	Nearest-neighbor gridding scheme
project	Project table data onto lines or great circles
ps2raster	Crop and convert PostScript files to raster images, EPS, and PDF
psbasemap	Create a basemap plot
psclip	Use polygon files to define clipping paths
pscoast	Plot (and fill) coastlines, borders, and rivers on maps
pscontour	Contour or image raw table data by triangulation
pshistogram	Plot a histogram
psimage	Plot Sun raster files on a map
pslegend	Plot a legend on a map
psmask	Create overlay to mask out regions on maps
psrose	Plot sector or rose diagrams
psscale	Plot gray scale or color scale on maps
pstext	Plot text strings on maps
pswiggle	Draw table data time-series along track on maps
psxy	Plot symbols, polygons, and lines on maps
psxyz	Plot symbols, polygons, and lines in 3-D
sample1d	Resampling of 1-D table data sets
spectrum1d	Compute various spectral estimates from time-series
sph2grd	Compute grid from spherical harmonic coefficients
sphdistance	Make grid of distances to nearest points on a sphere
sphinterpolate	Spherical gridding in tension of data on a sphere
sphtriangulate	Delaunay or Voronoi construction of spherical lon,lat data
splitxyz	Split *xyz* files into several segments
surface	A continuous curvature gridding algorithm
trend1d	Fits polynomial or Fourier trends to $y = f(x)$ series
trend2d	Fits polynomial trends to $z = f(x, y)$ series
triangulate	Perform optimal Delauney triangulation and gridding
xyz2grd	Convert an equidistant table *xyz* file to a 2-D grid file

7.2 GMT quick reference

Instead of an alphabetical listing, this section contains a summary sorted by program purpose. Also included is a quick summary of the standard command line options and a breakdown of the **-J** option for each of the over 30 projections available in GMT.

	Filtering of 1-D and 2-D Data
`blockmean`	L_2 (x,y,z) table data filter/decimator
`blockmedian`	L_1 (x,y,z) table data filter/decimator
`blockmode`	Mode estimate (x,y,z) table data filter/decimator
`filter1d`	Filter 1-D table data sets (time series)
`grdfilter`	Filter 2-D gridded data sets in the space domain
	Plotting of 1-D and 2-D Data
`grdcontour`	Contouring of 2-D gridded data sets
`grdimage`	Produce images from 2-D gridded data sets
`grdvector`	Plotting of 2-D gridded vector fields
`grdview`	3-D perspective imaging of 2-D gridded data sets
`psbasemap`	Create a basemap plot
`psclip`	Use polygon files to define clipping paths
`pscoast`	Plot (and fill) coastlines, borders, and rivers on maps
`pscontour`	Contour or image raw table data by triangulation
`pshistogram`	Plot a histogram
`psimage`	Plot Sun raster files on a map
`pslegend`	Plot a legend on a map
`psmask`	Create overlay to mask out regions on maps
`psrose`	Plot sector or rose diagrams
`psscale`	Plot gray scale or color scale on maps
`pstext`	Plot text strings on maps
`pswiggle`	Draw table data time-series along track on maps
`psxy`	Plot symbols, polygons, and lines on maps
`psxyz`	Plot symbols, polygons, and lines in 3-D
	Gridding of Data Tables
`greenspline`	Interpolation with Green's functions for splines in 1–3 D
`nearneighbor`	Nearest-neighbor gridding scheme
`sphinterpolate`	Spherical gridding in tension of data on a sphere
`surface`	A continuous curvature gridding algorithm
`triangulate`	Perform optimal Delauney triangulation and gridding
	Sampling of 1-D and 2-D Data
`gmtsimplify`	Line reduction using the Douglas-Peucker algorithm
`grdsample`	Resample a 2-D gridded data set onto a new grid
`grdtrack`	Sampling of 2-D gridded data set(s) along 1-D track
`sample1d`	Resampling of 1-D table data sets
	Projection and Map-transformation
`grdproject`	Project gridded data sets onto a new coordinate system
`mapproject`	Transformation of coordinate systems for table data
`project`	Project table data onto lines or great circles
	Retrieve Information
`gmtdefaults`	List the current default settings
`gmtget`	Retrieve selected parameters in current file
`gmtinfo`	Get information about table data files

Continued on next page

Table 7.2 – continued from previous page

	Filtering of 1-D and 2-D Data
`gmtset`	Change selected parameters in current file
`grdinfo`	Get information about grid files
	Mathematical Operations on Tables or Grids
`gmtmath`	Mathematical operations on table data
`makecpt`	Make color palette tables
`spectrum1d`	Compute various spectral estimates from time-series
`sph2grd`	Compute grid from spherical harmonic coefficients
`sphdistance`	Make grid of distances to nearest points on a sphere
`sphtriangulate`	Delaunay or Voronoi construction of spherical lon,lat data
	Convert or Extract Subsets of Data
`gmtconnect`	Connect segments into more complete lines or polygons
`gmtconvert`	Convert data tables from one format to another
`gmtselect`	Select subsets of table data based on multiple spatial criteria
`gmtspatial`	Geospatial operations on lines and polygons
`gmtvector`	Basic operations on vectors in 2-D and 3-D
`grd2rgb`	Convert Sun raster or grid file to red, green, blue component grids
`grd2xyz`	Conversion from 2-D grid file to table data
`grdblend`	Blend several partially over-lapping grid files onto one grid
`grdcut`	Cut a sub-region from a grid file
`grdpaste`	Paste together grid files along a common edge
`grdreformat`	Converts grid files into other grid formats
`splitxyz`	Split *xyz* files into several segments
`xyz2grd`	Convert an equidistant table *xyz* file to a 2-D grid file
	Determine Trends in 1-D and 2-D Data
`fitcircle`	Finds the best-fitting great or small circle for a set of points
`grdtrack`	Sampling of 2-D gridded data set(s) along 1-D track
`trend1d`	Fits polynomial or Fourier trends to $y = f(x)$ series
`trend2d`	Fits polynomial trends to $z = f(x, y)$ series
	Other Operations on 2-D Grids
`grd2cpt`	Make color palette table from a grid files
`grdclip`	Limit the *z*-range in gridded data sets
`grdedit`	Modify header information in a 2-D grid file
`grdfft`	Perform operations on grid files in the frequency domain
`grdgradient`	Compute directional gradient from grid files
`grdhisteq`	Histogram equalization for grid files
`grdlandmask`	Create masking grid files from shoreline data base
`grdmask`	Reset grid nodes in/outside a clip path to constants
`grdmath`	Mathematical operations on grid files
`grdvolume`	Calculate volumes under a surface within specified contour
	Miscellaneous Tools
`gmt2kml`	Like `psxy` but plots KML for use in Google Earth
`kml2gmt`	Extracts coordinates from Google Earth KML files
`ps2raster`	Crop and convert PostScript files to raster images, EPS, and PDF

GMT offers 31 map projections. These are specified using the **-J** common option. There are two conventions you may use: (a) GMT-style syntax and (b) **Proj4**-style syntax. The projection codes for the GMT-style are tabulated below.

WITH GMT PROJECTION CODES	
-J (upper case for *width*, lower case for *scale*) Map projection	
-JAlon$_0$/lat$_0$[/*horizon*]/*width*	Lambert azimuthal equal area
-JBlon$_0$/lat$_0$/lat$_1$/lat$_2$*width*	Albers conic equal area
-JClon$_0$/lat$_0$*width*	Cassini cylindrical
-JCyl_stere/[lon$_0$[/lat$_0$/]]*width*	Cylindrical stereographic
-JDlon$_0$/lat$_0$/lat$_1$/lat$_2$*width*	Equidistant conic
-JElon$_0$/lat$_0$[/*horizon*]/*width*	Azimuthal equidistant
-JFlon$_0$/lat$_0$[/*horizon*]/*width*	Azimuthal gnomonic
-JGlon$_0$/lat$_0$[/*horizon*]/*width*	Azimuthal orthographic
-JGlon$_0$/lat$_0$*alt*/*azim*/*tilt*/*twist*/*W*/*H*/*width*	General perspective
-JHlon$_0$*width*	Hammer equal area
-JIlon$_0$*width*	Sinusoidal equal area
-JJlon$_0$*width*	Miller cylindrical
-JKflon$_0$*width*	Eckert IV equal area
-JKslon$_0$*width*	Eckert VI equal area
-JLlon$_0$/lat$_0$/lat$_1$/lat$_2$*width*	Lambert conic conformal
-JM[lon$_0$[/lat$_0$/]]*width*	Mercator cylindrical
-JN[lon$_0$/]*width*	Robinson
-JOalon$_0$/lat$_0$*azim*/*width*	Oblique Mercator, 1: origin and azimuth
-JOblon$_0$/lat$_0$/lon$_1$/lat$_1$*width*	Oblique Mercator, 2: two points
-JOclon$_0$/lat$_0$/lon$_p$/lat$_p$*width*	Oblique Mercator, 3: origin and pole
-JP[**a**]*width*[/*origin*]	Polar [azimuthal] (θ, r) (or cylindrical)
-JPoly[lon$_0$[/lat$_0$/]]*width*	(American) polyconic
-JQ[lon$_0$[/lat$_0$/]]*width*	Equidistant cylindrical
-JR[lon$_0$/]*width*	Winkel Tripel
-JSlon$_0$/lat$_0$[/*horizon*]/*width*	General stereographic
-JT[lon$_0$[/lat$_0$/]]*width*	Transverse Mercator
-JU*zone*/*width*	Universal Transverse Mercator (UTM)
-JV[lon$_0$/]*width*	Van der Grinten
-JW[lon$_0$/]*width*	Mollweide
-JX*width*[**l**|**p***exp*|**T**|**t**][/*height*[**l**|**p***exp*|**T**|**t**]][**d**]	Linear, log$_{10}$, $x^a - y^b$, and time
-JYlon$_0$/lat$_0$*width*	Cylindrical equal area

The projection codes for the **Proj4**-style are tabulated below; these all accept a map *scale*.

WITH Proj4 PROJECTION CODES	
-J (lower case for *scale* only) Map projection	
-Jaea/lon$_0$/lat$_0$/lat$_1$/lat$_2$*scale*	Albers conic equal area
-Jaeqd/lon$_0$/lat$_0$[/*horizon*]/*scale*	Azimuthal equidistant
-Jcass/lon$_0$/lat$_0$*scale*	Cassini cylindrical
-Jcea/lon$_0$/lat$_0$*scale*	Cylindrical equal area
-Jcyl_stere/[lon$_0$[/lat$_0$/]]*scale*	Cylindrical stereographic
-Jeqc/[lon$_0$[/lat$_0$/]]*scale*	Equidistant cylindrical
-Jeqdc/lon$_0$/lat$_0$/lat$_1$/lat$_2$*scale*	Equidistant conic
-Jgnom/lon$_0$/lat$_0$[/*horizon*]/*scale*	Azimuthal gnomonic
-Jhammer/lon$_0$*scale*	Hammer equal area
-Jeck4/lon$_0$*scale*	Eckert IV equal area
-Jeck6/lon$_0$*scale*	Eckert VI equal area
-Jlaea/lon$_0$/lat$_0$[/*horizon*]/*scale*	Lambert azimuthal equal area

Continued on next page

Table 7.4 – continued from previous page

WITH Proj4 PROJECTION CODES	
-Jlcc/lon$_0$/lat$_0$/lat$_1$/lat$_2$*scale*	Lambert conic conformal
-Jmerc/[lon$_0$[/lat$_0$/]]*scale*	Mercator cylindrical
-Jmill/lon$_0$*scale*	Miller cylindrical
-Jmoll/[lon$_0$/]*scale*	Mollweide
-Jnsper/lon$_0$/lat$_0$*alt/azim/tilt/twist/W/H/scale*	General perspective
-Jomerc/lon$_0$/lat$_0$*azim/scale*	Oblique Mercator, 1: origin and azimuth
-Jomerc/lon$_0$/lat$_0$/lon$_1$/lat$_1$*scale*	Oblique Mercator, 2: two points
-Jomercp/:lon$_0$/lat$_0$/lon$_p$/lat$_p$*scale*	Oblique Mercator, 3: origin and pole
-Jortho/lon$_0$/lat$_0$[*/horizon*]*/scale*	Azimuthal orthographic
-Jpolar/[**a**]*scale*[*/origin*]	Polar [azimuthal] (θ, r) (or cylindrical)
-Jpoly/[lon$_0$[/lat$_0$/]]*scale*	(American) polyconic
-Jrobin/[lon$_0$/]*scale*	Robinson
-Jsinu/lat$_0$*scale*	Sinusoidal equal area
-Jstere/lon$_0$/lat$_0$[*/horizon*]*/scale*	General stereographic
-Jtmerc/[lon$_0$[/lat$_0$/]]*scale*	Transverse Mercator
-Jutm/*zone/scale*	Universal Transverse Mercator (UTM)
-Jvandg/[lon$_0$/]*scale*	Van der Grinten
-Jwintri/[lon$_0$/]*scale*	Winkel Tripel
-Jxy*xscale*[**l**|**p***exp*|**T**|**t**][*/yscale*[**l**|**p***exp*|**T**|**t**]][**d**]	Linear, $\log_{10}$, $x^a - y^b$, and time

Finally, the rest of the GMT common options are given below:

STANDARDIZED COMMAND LINE OPTIONS	
-B*information*	Specify map frame and axes parameters
-K	Append more PS later
-O	This is an overlay plot
-P	Select Portrait orientation
-R*west/east/south/north*[*/zmin/zmax*][**r**]	Specify Region of interest
-U[[*just*]*/dx/dy/*][*label*]	Plot time-stamp on plot
-V	Run in verbose mode
-X[**a**| **c**| **r**]*off*[**u**]	Shift plot origin in *x*-direction
-Y[**a**| **c**| **r**]*off*[**u**]	Shift plot origin in *y*-direction
-a*name=col,...*	Associates aspatial data with columns
-b[**i**| **o**][*ncol*][**t**]	Select binary input or output
-c*copies*	Set number of plot copies [1]
-f[**i**| **o**]*colinfo*	Set formatting of ASCII input or output
-g[+]**x**| **X**| **y**| **Y**| **d**| **D***gap*[**u**]	Segment data by detecting gaps
-h[**i**| **o**][*n_headers*]	ASCII [input| output] tables have header record[s]
-i*columns*	Selection of input columns
-o*columns*	Selection of output columns
-n[*type*][**+a**][**+b***BC*] [**+c**][**+t***threshold*]	Set grid interpolation mode
-p*azim/elev*[*/zlevel*][**+w***lon0/lat0*[*/z0*]][**+v***x0/y0*]	Control 3-D perspective view
-r	Sets grid registration
-s[**z**| *cols*]	Control treatment of NaN records
-t*transparency*	Set layer PDF transparency
-:[**i**| **o**]	Expect *y*/*x* input rather than *x*/*y*

General Features

This section explains features common to all the programs in GMT and summarizes the philosophy behind the system. Some of the features described here may make more sense once you reach the cook-book section where we present actual examples of their use.

8.1 GMT units

While GMT has default units for both actual Earth distances and plot lengths (dimensions) of maps, it is recommended that you specifically indicate the units of your arguments by appending the unit character, as discussed below. This will aid you in debugging, let others understand your scripts, and remove any uncertainty as to what unit you thought you wanted.

8.1.1 Distance units

d	Degree of arc	**M**	Statute mile
e	Meter [Default]	**n**	Nautical mile
f	Foot	**s**	Second of arc
k	Kilometer	**u**	US Survey foot
m	Minute of arc		

For Cartesian data and scaling the data units do not normally matter (they could be kg or Lumens for all we know) and are never entered. Geographic data are different as distances can be specified in a variety of ways. GMT programs that accept actual Earth length scales like search radii or distances can therefore handle a variety of units. These choices are listed in Table *distunits*; simply append the desired unit to the distance value you supply. A value without a unit suffix will be consider to be in meters. For example, a distance of 30 nautical miles should be given as 30**n**.

8.1.2 Distance calculations

The calculation of distances on Earth (or other planetary bodies) depends on the ellipsoidal parameters of the body (via *PROJ_ELLIPSOID*) and the method of computation. GMT offers three alternatives that trade off accuracy and computation time.

Flat Earth distances

Quick, but approximate "Flat Earth" calculations make a first-order correction for the spherical nature of a planetary body by computing the distance between two points A and B as

$$d_f = R\sqrt{(\theta_A - \theta_B)^2 + \cos\left[\frac{\theta_A + \theta_B}{2}\right]\Delta\lambda^2},$$

where *R* is the representative (or spherical) radius of the planet, θ is latitude, and the difference in longitudes, $\Delta\lambda = \lambda_A - \lambda_B$, is adjusted for any jumps that might occur across Greenwich or the Dateline. As written, the geographic coordinates are given in radians. This approach is suitable when the points you use to compute d_f do not greatly differ in latitude and computation speed is paramount. You can specify this mode of computation by using the **-** prefix to the specified distance (or to the unit itself in cases where no distance is required and only a unit is expected). For instance, a search radius of 50 statute miles using this mode of computation might be specified via **-S-**50**M**.

Great circle distances

This is the default distance calculation, which will also approximate the planetary body by a sphere of mean radius *R*. However, we compute an exact distance between two points A and B on such a sphere via the Haversine equation

$$d_g = 2R\sin^{-1}\sqrt{\sin^2\frac{\theta_A - \theta_B}{2} + \cos\theta_A\cos\theta_B\sin^2\frac{\lambda_A - \lambda_B}{2}},$$

This approach is suitable for most situations unless exact calculations for an ellipsoid is required (typically for a limited surface area). For instance, a search radius of 5000 feet using this mode of computation would be specified as **-S**5000**f**.

Note: There are two additional GMT defaults that control how great circle (and Flat Earth) distances are computed. One concerns the selection of the "mean radius". This is selected by *PROJ_MEAN_RADIUS*, which selects one of several possible representative radii. The second is *PROJ_AUX_LATITUDE*, which converts geodetic latitudes into one of several possible auxiliary latitudes that are better suited for the spherical approximation. While both settings have default values to best approximate geodesic distances (*authalic* mean radius and latitudes), expert users can choose from a range of options as detailed in the `gmt.conf` man page.

Geodesic distances

For the most accurate calculations we use a full ellipsoidal formulation. Currently, we are using Vincenty's [1975] formula [1]. You select this mode of computation by using the **+** prefix to the specified distance (or to the unit itself in cases where no distance is required). For instance, a search radius of 20 km using this mode of computation would be set by **-S+**20**k**.

8.1.3 Length units

GMT programs can accept dimensional quantities and plot lengths in **c**m, **i**nch, or **p**oint (1/72 of an inch) [2]. There are two ways to ensure that GMT understands which unit you intend to use:

[1] Vicenty, T. (1975), Direct and inverse solutions of geodesics on the ellipsoid with application of nested equations, *Surv. Rev., XXII(176)*, 88–93.

[2] PostScript definition. In the typesetting industry a slightly different definition of point (1/72.27 inch) is used, presumably to cause needless trouble.

1. Append the desired unit to the dimension you supply. This way is explicit and clearly communicates what you intend, e.g., **-X**4**c** means the length being passed to the **-X** switch is 4 cm.
2. Set the parameter *PROJ_LENGTH_UNIT* to the desired unit. Then, all dimensions without explicit unit will be interpreted accordingly.

The latter method is less secure as other users may have a different unit set and your script may not work as intended. We therefore recommend you always supply the desired unit explicitly.

8.2 GMT defaults

8.2.1 Overview and the gmt.conf file

There are about 100 parameters which can be adjusted individually to modify the appearance of plots or affect the manipulation of data. When a program is run, it initializes all parameters to the GMTdefaults [3], then tries to open the file `gmt.conf` in the current directory [4]. If not found, it will look for that file in a sub-directory `/.gmt` of your home directory, and finally in your home directory itself. If successful, the program will read the contents and set the default values to those provided in the file. By editing this file you can affect features such as pen thicknesses used for maps, fonts and font sizes used for annotations and labels, color of the pens, dots-per-inch resolution of the hardcopy device, what type of spline interpolant to use, and many other choices. A complete list of all the parameters and their default values can be found in the `gmt.conf` manual pages. Figures *GMT Parameters a*, *b*, and *c* show the parameters that affect plots. You may create your own `gmt.conf` files by running `gmtdefaults` and then modify those parameters you want to change. If you want to use the parameter settings in another file you can do so by specifying `+<defaultfile>` on the command line. This makes it easy to maintain several distinct parameter settings, corresponding perhaps to the unique styles required by different journals or simply reflecting font changes necessary to make readable overheads and slides. Note that any arguments given on the command line (see below) will take precedent over the default values. E.g., if your `gmt.conf` file has x offset = 1**i** as default, the **-X**1.5**i** option will override the default and set the offset to 1.5 inches.

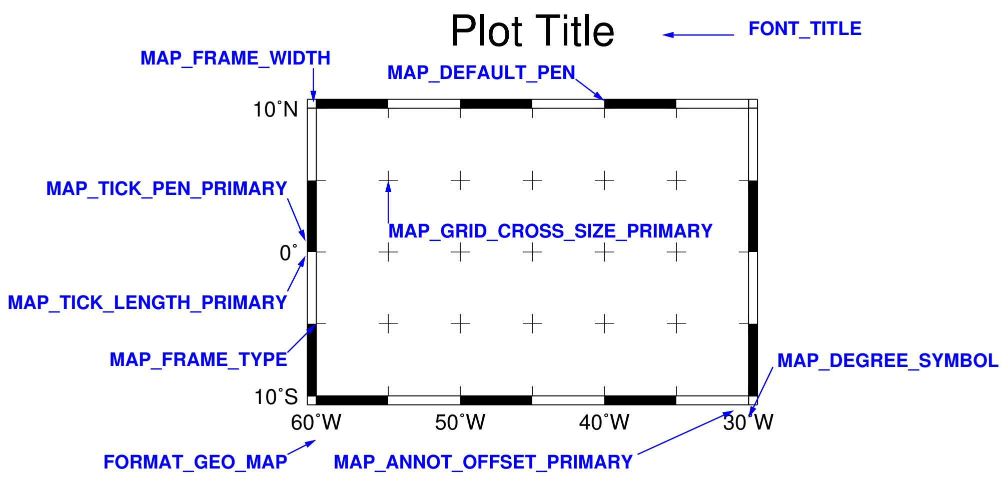


Figure 8.1: Some GMT parameters that affect plot appearance.

There are at least two good reasons why the GMT default options are placed in a separate parameter file:

[3] Choose between SI and US default units by modifying in the GMT share directory.

[4] To remain backwards compatible with GMT 4 we will also look for but only if cannot be found.

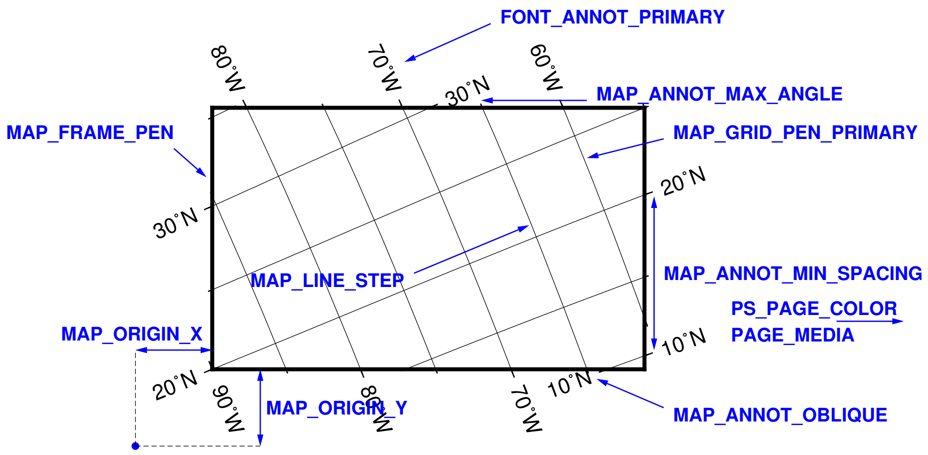


Figure 8.2: More GMT parameters that affect plot appearance.

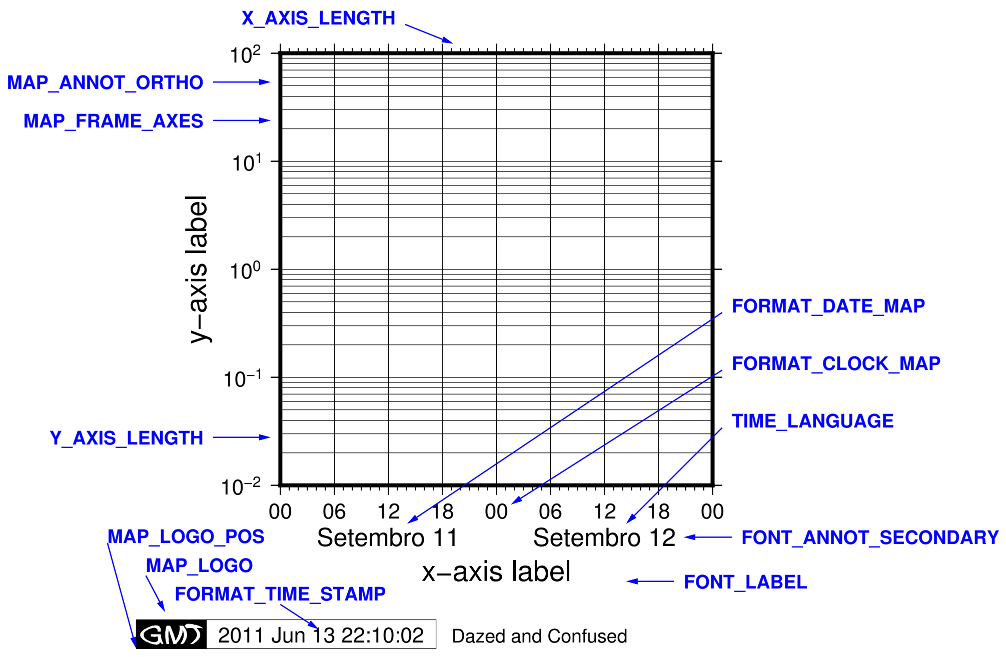


Figure 8.3: Even more GMT parameters that affect plot appearance.

1. It would not be practical to allow for command-line syntax covering so many options, many of which are rarely or never changed (such as the ellipsoid used for map projections).
2. It is convenient to keep separate `gmt.conf` files for specific projects, so that one may achieve a special effect simply by running GMT commands in the directory whose `gmt.conf` file has the desired settings. For example, when making final illustrations for a journal article one must often standardize on font sizes and font types, etc. Keeping all those settings in a separate `gmt.conf` file simplifies this process and will allow you to generate those illustrations with the same settings later on. Likewise, GMT scripts that make figures for PowerPoint presentations often use a different color scheme and font size than output intended for laser printers. Organizing these various scenarios into separate `gmt.conf` files will minimize headaches associated with micro-editing of illustrations.

8.2.2 Changing GMT defaults

As mentioned, GMT programs will attempt to open a file named `gmt.conf`. At times it may be desirable to override that default. There are several ways in which this can be accomplished.

- One method is to start each script by saving a copy of the current `gmt.conf`, then copying the desired `gmt.conf` file to the current directory, and finally reverting the changes at the end of the script. Possible side effects include premature ending of the script due to user error or bugs which means the final resetting does not take place (unless you write your script very carefully.)
- To permanently change some of the GMT parameters on the fly inside a script the `gmtset` utility can be used. E.g., to change the primary annotation font to 12 point Times-Bold in red we run

  ```
  gmt gmtset FONT_ANNOT_PRIMARY 12p,Times-Bold,red
  ```

  These changes will remain in effect until they are overridden.
- If all you want to achieve is to change a few parameters during the execution of a single command but otherwise leave the environment intact, consider passing the parameter changes on the command line via the **--***PAR=value* mechanism. For instance, to temporarily set the output format for floating points to have lots of decimals, say, for map projection coordinate output, append **--***FORMAT_FLOAT_OUT*=%.16lg to the command in question.
- Finally, GMT provides to possibility to override the settings only during the running of a single script, reverting to the original settings after the script is run, as if the script was run in "isolation". The isolation mode is discussed in Section Running GMT in isolation mode.

In addition to those parameters that directly affect the plot there are numerous parameters than modify units, scales, etc. For a complete listing, see the `gmt.conf` man pages. We suggest that you go through all the available parameters at least once so that you know what is available to change via one of the described mechanisms.

8.3 Command line arguments

Each program requires certain arguments specific to its operation. These are explained in the manual pages and in the usage messages. Most programs are "case-sensitive"; almost all options must start with an upper-case letter. We have tried to choose letters of the alphabet which stand for the argument so that they will be easy to remember. Each argument specification begins with a hyphen (except input file names; see below), followed by a letter, and sometimes a number or character string immediately after the letter. *Do not* space between the hyphen, letter, and number or string. *Do* space between options. Example:

```
gmt pscoast -R0/20/0/20 -Ggray -JM6i -Wthin -B5 -V -P > map.ps
```

8.4 Standardized command line options

Most of the programs take many of the same arguments like those related to setting the data region, the map projection, etc. The 24 switches in Table *switches* have the same meaning in all the programs (although some programs may not use all of them). These options will be described here as well as in the manual pages, as is vital that you understand how to use these options. We will present these options in order of importance (some are use a lot more than others).

-B	Define tickmarks, annotations, and labels for basemaps and axes
-J	Select a map projection or coordinate transformation
-K	Allow more plot code to be appended to this plot later
-O	Allow this plot code to be appended to an existing plot
-P	Select Portrait plot orientation [Default is landscape]
-R	Define the extent of the map/plot region
-U	Plot a time-stamp, by default in the lower left corner of page
-V	Select verbose operation; reporting on progress
-X	Set the *x*-coordinate for the plot origin on the page
-Y	Set the *y*-coordinate for the plot origin on the page
-a	Associate aspatial data from OGR/GMT files with data columns
-b	Select binary input and/or output
-c	Specify the number of plot copies
-f	Specify the data format on a per column basis
-g	Identify data gaps based on supplied criteria
-h	Specify that input/output tables have header record(s)
-i	Specify which input columns to read
-n	Specify grid interpolation settings
-o	Specify which output columns to write
-p	Control perspective views for plots
-r	Set the grid registration to pixel [Default is gridline]
-s	Control output of records containing one or more NaNs
-t	Change layer PDF transparency
-:	Assume input geographic data are (*lat,lon*) and not (*lon,lat*)

8.4.1 Data domain or map region: The -R option

The **-R** option defines the map region or data domain of interest. It may be specified in one of three ways, two of which are shown in Figure *Map region*:

1. **-R***xmin*/*xmax*/*ymin*/*ymax*. This is the standard way to specify Cartesian data domains and geographical regions when using map projections where meridians and parallels are rectilinear.
2. **-R***xlleft*/*ylleft*/*xuright*/*yuright***r**. This form is used with map projections that are oblique, making meridians and parallels poor choices for map boundaries. Here, we instead specify the lower left corner and upper right corner geographic coordinates, followed by the suffix **r**. This form guarantees a rectangular map even though lines of equal longitude and latitude are not straight lines.

3. **-R***gridfile*. This will copy the domain settings found for the grid in specified file. Note that depending on the nature of the calling program, this mechanism will also set grid spacing and possibly the grid registration (see Section Grid registration: The -r option).

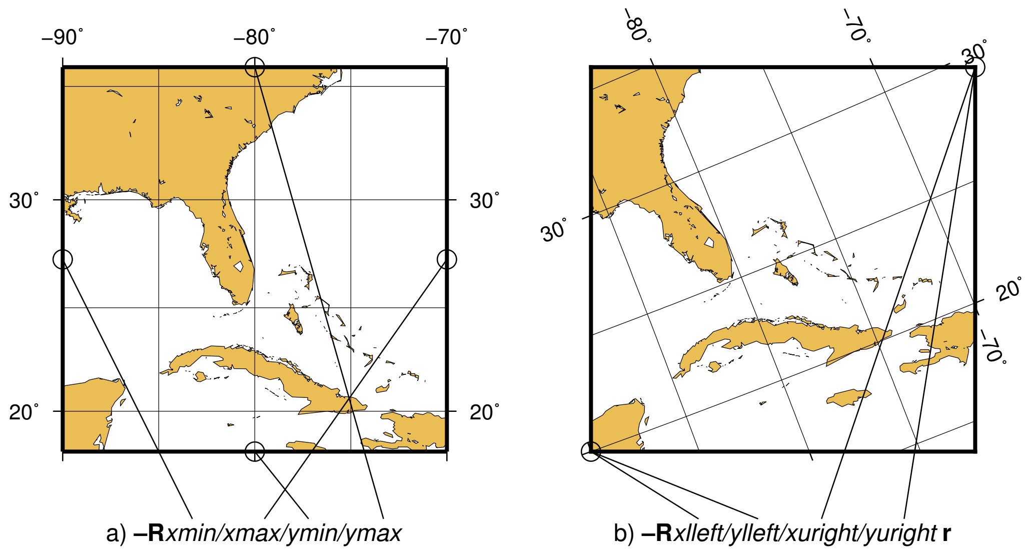

–90˚
–80˚
–70˚
30˚
30˚
20˚
20˚
a) –R*xmin/xmax/ymin/ymax*
-80˚
-70˚
30˚
30˚
20˚
-70˚
b) –R*xlleft/ylleft/xuright/yuright* r

Figure 8.4: The plot region can be specified in two different ways. (a) Extreme values for each dimension, or (b) coordinates of lower left and upper right corners.

For rectilinear projections the first two forms give identical results. Depending on the selected map projection (or the kind of expected input data), the boundary coordinates may take on several different formats:

Geographic coordinates: These are longitudes and latitudes and may be given in decimal degrees (e.g., -123.45417) or in the []*ddd*[:*mm*[:*ss*[.*xxx*]]][**W**|**E**|**S**|**N**] format (e.g., 123:27:15W). Note that **-Rg** and **-Rd** are shorthands for "global domain" **-R***0/360/-90/90* and **-R***-180/180/-90/90*, respectively.

When used in conjunction with the Cartesian Linear Transformation (**-Jx** or **-JX**) —which can be used to map floating point data, geographical coordinates, as well as time coordinates— it is prudent to indicate that you are using geographical coordinates in one of the following ways:

- Use **-Rg** or **-Rd** to indicate the global domain.
- Use **-Rg***xmin/xmax/ymin/ymax* to indicate a limited geographic domain.
- Add **W**, **E**, **S**, or **N** to the coordinate limits or add the generic **D** or **G**. Example: **-R***0/360G/-90/90N*.

Alternatively, you may indicate geographical coordinates by supplying **-fg**; see Section Data type selection: The -f option.

Projected coordinates: These are Cartesian projected coordinates compatible with the chosen projection and are given with a leading length *unit*, (e.g., **k**-200/200/-300/300 for a 400 by 600 km rectangular area centered on the projection center (0, 0). These coordinates are internally converted to the corresponding geographic (longitude, latitude) coordinates for the lower left and upper right corners. This form is convenient when you want to specify a region directly in the projected units (e.g., UTM meters). For allowable units, see Table *distunits*.

Calendar time coordinates: These are absolute time coordinates referring to a Gregorian or ISO calendar. The general format is [*date*]**T**[*clock*], where *date* must be in the *yyyy*[*-mm*[*-dd*]] (year, month, day-of-month) or *yyyy*[*-jjj*] (year and day-of-year) for Gregorian calendars and *yyyy*[*-***W***ww*[*-d*]] (year, week, and day-of-week) for the ISO calendar. If no *date* is given we assume the current day. The **T** flag is required if a *clock* is given.

The optional *clock* string is a 24-hour clock in *hh*[*:mm*[*:ss*[*.xxx*]]] format. If no *clock* is given it implies 00:00:00, i.e., the start of the specified day. Note that not all of the specified entities need be present in the data. All calendar date-clock strings are internally represented as double precision seconds since proleptic Gregorian date Monday January 1 00:00:00 0001. Proleptic means we assume that the modern calendar can be extrapolated forward and backward; a year zero is used, and Gregory's reforms [5] are extrapolated backward. Note that this is not historical.

Relative time coordinates: These are coordinates which count seconds, hours, days or years relative to a given epoch. A combination of the parameters *TIME_EPOCH* and *TIME_UNIT* define the epoch and time unit. The parameter *TIME_SYSTEM* provides a few shorthands for common combinations of epoch and unit, like **j2000** for days since noon of 1 Jan 2000. The default relative time coordinate is that of UNIX computers: seconds since 1 Jan 1970. Denote relative time coordinates by appending the optional lower case **t** after the value. When it is otherwise apparent that the coordinate is relative time (for example by using the **-f** switch), the **t** can be omitted.

Other coordinates: These are simply any coordinates that are not related to geographic or calendar time or relative time and are expected to be simple floating point values such as []*xxx.xxx*[E: | e| D| d[]xx], i.e., regular or exponential notations, with the enhancement to understand FORTRAN double precision output which may use D instead of E for exponents. These values are simply converted as they are to internal representation. [6]

8.4.2 Coordinate transformations and map projections: The -J option

This option selects the coordinate transformation or map projection. The general format is

- **-J***δ*[*parameters/*]*scale*. Here, *δ* is a *lower-case* letter of the alphabet that selects a particular map projection, the *parameters* is zero or more slash-delimited projection parameter, and *scale* is map scale given in distance units per degree or as 1:xxxxx.
- **-J**Δ[*parameters/*]*width*. Here, Δ is an *upper-case* letter of the alphabet that selects a particular map projection, the *parameters* is zero or more slash-delimited projection parameter, and *width* is map width (map height is automatically computed from the implied map scale and region).

Since GMT version 4.3.0, there is an alternative way to specify the projections: use the same abbreviation as in the mapping package **Proj4**. The options thus either look like:

- **-J***abbrev/*[*parameters/*]*scale*. Here, **abbrev** is a *lower-case* abbreviation that selects a particular map projection, the *parameters* is zero or more slash-delimited projection parameter, and *scale* is map scale given in distance units per degree or as 1:xxxxx.
- **-J***Abbrev/*[*parameters/*]*width*. Here, **Abbrev** is an *capitalized* abbreviation that selects a particular map projection, the *parameters* is zero or more slash-delimited projection parameter, and *width* is

[5] The Gregorian Calendar is a revision of the Julian Calendar which was instituted in a papal bull by Pope Gregory XIII in 1582. The reason for the calendar change was to correct for drift in the dates of significant religious observations (primarily Easter) and to prevent further drift in the dates. The important effects of the change were (a) Drop 10 days from October 1582 to realign the Vernal Equinox with 21 March, (b) change leap year selection so that not all years ending in "00" are leap years, and (c) change the beginning of the year to 1 January from 25 March. Adoption of the new calendar was essentially immediate within Catholic countries. In the Protestant countries, where papal authority was neither recognized not appreciated, adoption came more slowly. England finally adopted the new calendar in 1752, with eleven days removed from September. The additional day came because the old and new calendars disagreed on whether 1700 was a leap year, so the Julian calendar had to be adjusted by one more day.

[6] While UTM coordinates clearly refer to points on the Earth, in this context they are considered "other". Thus, when we refer to "geographical" coordinates herein we imply longitude, latitude.

map width (map height is automatically computed from the implied map scale and region).

The projections available in GMT are presented in Figure *The 30+ map projections and coordinate transformations available in GMT*. For details on all GMT projections and the required parameters, see the `psbasemap` man page. We will also show examples of every projection in the next Chapters, and a quick summary of projection syntax was given in Chapter *GMT overview and quick reference*.

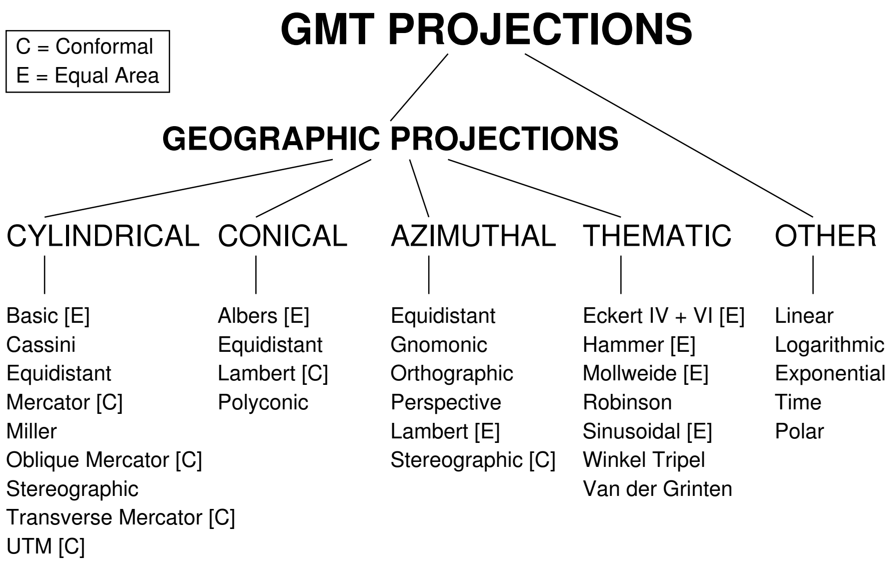


Figure 8.5: The 30+ map projections and coordinate transformations available in GMT

8.4.3 Map frame and axes annotations: The -B option

This is potentially the most complicated option in GMT, but most examples of its usage are actually quite simple. We distinguish between to sets of information: Frame settings and Axes parameters. These are set separately by their own **-B** invocations; hence multiple **-B** specifications may be specified. The frame settings covers things such as which axes should be plotted, canvas fill, plot title, and what type of gridlines be drawn, whereas the Axes settings deal with annotation, tick, and gridline intervals, axes labels, and annotation units.

The Frame settings are specified by

- **-B**[*axes*][**+b**][**+g***fill*][**+o***lon/lat*][**+t***title*]

Here, the optional *axes* dictates which of the axes should be drawn and possibly annotated. By default, all 4 map boundaries (or plot axes) are plotted (denoted **W**, **E**, **S**, **N**). To change this selection, append the codes for those you want (e.g., **WSn**). In this example, the lower case **n** denotes to draw the axis and (major and minor) tick marks on the "northern" (top) edge of the plot. The upper case **WS** will annotate the "western" and "southern" axes with numerals and plot the any axis labels in addition to draw axis/tick-marks. For 3-D plots you can also specify **Z** or **z**. By default a single vertical axes will then be plotted at the most suitable map corner. You can override this by appending any combination of corner ids **1234**, where **1** represents the lower left corner and the order goes counter-clockwise. Append **+b** to draw the outline of the 3-D box defined by **-R**; this modifier is also needed to display gridlines in the x–z, y–z planes. You may paint the map canvas by appending the **+g***fill* modifier [Default is no fill]. If gridlines are specified via the Axes parameters (discussed below) then by default these are referenced to the North pole. If, however, you wish to produce oblique gridlines about another pole you can append **+o***lon/lat* to change this behavior (the modifier is ignored if no gridlines are requested). Finally, you may optionally add **+t***title* to place a title that will appear centered above the plot frame.

The Axes settings are specified by

- **-B**[**p**|**s**][**x**|**x**|**z**]*intervals*[**+l***label*][**+p***prefix*][**+u***unit*]

but you may also split this into two separate invocations for clarity, i.e.,

- **-B**[**p**|**s**][**x**|**y**|**z**][**+l***label*][**+p***prefix*][**+u***unit*]
- **-B**[**p**|**s**][**x**|**y**|**z**]*intervals*

The first optional flag following **-B** selects **p** (rimary) [Default] or **s** (econdary) axes information (which is mostly used for time axes annotations; see examples below). The next optional flags specifies which axes you are providing information for. This can be an individual axis (e.g., just **x**) or a combination (e.g., **xz**). If none are given then we default to **xy**. Thus, if you wish to give different annotation intervals or labels for the various axes then you must repeat the **B** option for each axis. To add a label to an axis, just append **+l***label*. If the axis annotation should have a leading text prefix (e.g., dollar sign for those plots of your net worth) you can append **+p***prefix*. For geographic maps the addition of degree symbols, etc. is automatic (and controlled by the GMT default setting *FORMAT_GEO_MAP*). However, for other plots you can add specific units by adding **+u***unit*. If any of these text strings contain spaces or special UNIX characters you will need to enclose them in quotes. The *intervals* specification is a concatenated string made up of substrings of the form

[**t**]*stride*[*phase*][**u**].

The **t** flag sets the axis item of interest; the available items are listed in Table *inttype*. Normally, equidistant annotations occur at multiples of *stride*; you can phase-shift this by appending *phase*, which can be a positive or negative number.

Flag	Description
a	Annotation and major tick spacing
f	Minor tick spacing
g	Grid line spacing

Note that the appearance of certain time annotations (month-, week-, and day-names) may be affected by the *TIME_LANGUAGE*, *FORMAT_TIME_PRIMARY_MAP*, and *FORMAT_TIME_SECONDARY_MAP* settings.

For automated plots the region may not always be the same and thus it can be difficult to determine the appropriate *stride* in advance. Here GMT provides the opportunity to autoselect the spacing between the major and minor ticks and the grid lines, by not specifying the *stride* value. For example, **-Bafg** will select all three spacings automatically for both axes. In case of longitude–latitude plots, this will keep the spacing the same on both axes. You can also use **-Bafg/afg** to autoselect them separately.

In the case of automatic spacing, when the *stride* argument is omitted after **g**, the grid line spacing is chosen the same as the minor tick spacing; unless **g** is used in consort with **a**, then the grid lines are spaced the same as the annotations.

The unit flag **u** can take on one of 18 codes; these are listed in Table *units*. Almost all of these units are time-axis specific. However, the **m** and **s** units will be interpreted as arc minutes and arc seconds, respectively, when a map projection is in effect.

Flag	Unit	Description
Y	year	Plot using all 4 digits
y	year	Plot using last 2 digits
O	month	Format annotation using **FORMAT_DATE_MAP**
o	month	Plot as 2-digit integer (1–12)
U	ISO week	Format annotation using **FORMAT_DATE_MAP**
u	ISO week	Plot as 2-digit integer (1–53)
r	Gregorian week	7-day stride from start of week (see **TIME_WEEK_START**)
K	ISO weekday	Plot name of weekday in selected language
k	weekday	Plot number of day in the week (1–7) (see **TIME_WEEK_START**)
D	date	Format annotation using **FORMAT_DATE_MAP**
d	day	Plot day of month (1–31) or day of year (1–366) (see **FORMAT_DATE_MAP**
R	day	Same as **d**; annotations aligned with week (see **TIME_WEEK_START**)
H	hour	Format annotation using **FORMAT_CLOCK_MAP**
h	hour	Plot as 2-digit integer (0–24)
M	minute	Format annotation using **FORMAT_CLOCK_MAP**
m	minute	Plot as 2-digit integer (0–60)
S	seconds	Format annotation using **FORMAT_CLOCK_MAP**
s	seconds	Plot as 2-digit integer (0–60)

As mentioned, there may be two levels of annotations. Here, "primary" refers to the annotation that is closest to the axis (this is the primary annotation), while "secondary" refers to the secondary annotation that is plotted further from the axis. The examples below will clarify what is meant. Note that the terms "primary" and "secondary" do not reflect any hierarchical order of units: The "primary" annotation interval is usually smaller (e.g., days) while the "secondary" annotation interval typically is larger (e.g., months).

Geographic basemaps

Geographic basemaps may differ from regular plot axis in that some projections support a "fancy" form of axis and is selected by the *MAP_FRAME_TYPE* setting. The annotations will be formatted according to the *FORMAT_GEO_MAP* template and *MAP_DEGREE_SYMBOL* setting. A simple example of part of a basemap is shown in Figure *Geographic map border*.

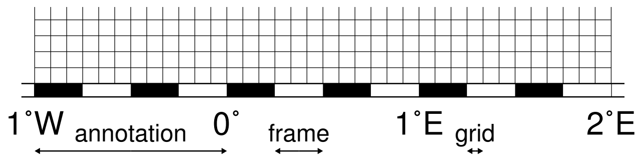


Figure 8.6: Geographic map border using separate selections for annotation, frame, and grid intervals. Formatting of the annotation is controlled by the parameter *FORMAT_GEO_MAP* in your `gmt.conf`.

The machinery for primary and secondary annotations introduced for time-series axes can also be utilized for geographic basemaps. This may be used to separate degree annotations from minutes- and seconds-annotations. For a more complicated basemap example using several sets of intervals, including different intervals and pen attributes for grid lines and grid crosses, see Figure *Complex basemap*.

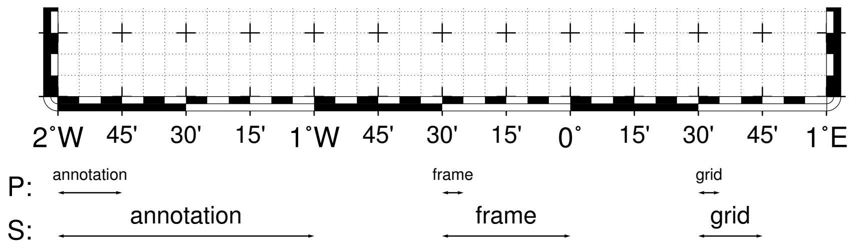


Figure 8.7: Geographic map border with both primary (P) and secondary (S) components.

Cartesian linear axes

For non-geographic axes, the *MAP_FRAME_TYPE* setting is implicitly set to plain. Other than that, cartesian linear axes are very similar to geographic axes. The annotation format may be controlled with the *FORMAT_FLOAT_OUT* parameter. By default, it is set to "%g", which is a C language format statement for floating point numbers [7], and with this setting the various axis routines will automatically determine how many decimal points should be used by inspecting the *stride* settings. If *FORMAT_FLOAT_OUT* is set to another format it will be used directly (.e.g, "%.2f" for a fixed, two decimals format). Note that for these axes you may use the *unit* setting to add a unit string to each annotation (see Figure *Axis label*).

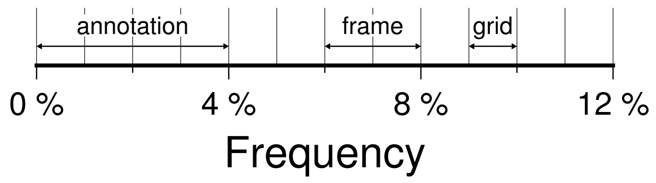


Figure 8.8: Linear Cartesian projection axis. Long tickmarks accompany annotations, shorter ticks indicate frame interval. The axis label is optional. For this example we used `-R0/12/0/0.95 -JX3i/0.3i -Ba4f2g1+lFrequency+u" %" -BS`

Cartesian $\log_{10}$ axes

Due to the logarithmic nature of annotation spacings, the *stride* parameter takes on specific meanings. The following concerns are specific to log axes (see Figure *Logarithmic projection axis*):

- *stride* must be 1, 2, 3, or a negative integer -n. Annotations/ticks will then occur at 1, 1-2-5, or 1,2,3,4,...,9, respectively, for each magnitude range. For *-n* the annotations will take place every *n*'th magnitude.
- Append **l** to *stride*. Then, $\log_{10}$ of the annotation is plotted at every integer $\log_{10}$ value (e.g., *x = 100* will be annotated as "2") [Default annotates *x* as is].
- Append **p** to *stride*. Then, annotations appear as 10 raised to $\log_{10}$ of the value (e.g., 10^{-5}).

Cartesian exponential axes

Normally, *stride* will be used to create equidistant (in the user's unit) annotations or ticks, but because of the exponential nature of the axis, such annotations may converge on each other at one end of the axis. To avoid this problem, you can append **p** to *stride*, and the annotation interval is expected to be

[7] Please consult the man page for *printf* or any book on C.

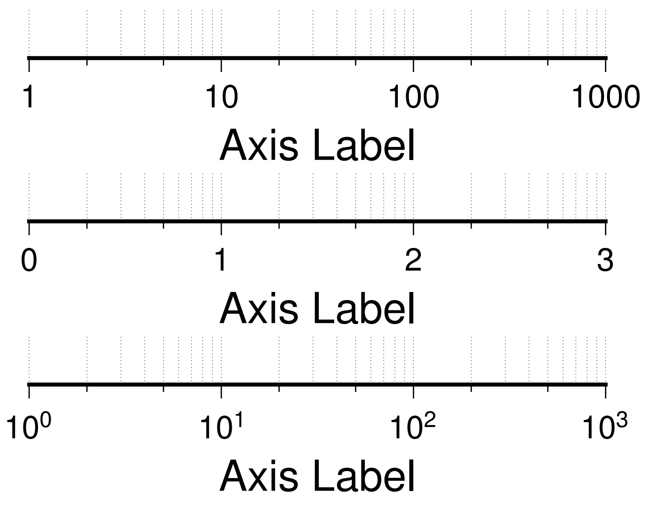


Figure 8.9: Logarithmic projection axis using separate values for annotation, frame, and grid intervals. (top) Here, we have chosen to annotate the actual values. Interval = 1 means every whole power of 10, 2 means 1, 2, 5 times powers of 10, and 3 means every 0.1 times powers of 10. We used -R1/1000/0/1 -JX3il/0.25i -Ba1f2g3. (middle) Here, we have chosen to annotate $\log_{10}$ of the actual values, with -Ba1f2g3l. (bottom) We annotate every power of 10 using $\log_{10}$ of the actual values as exponents, with -Ba1f2g3p.

in transformed units, yet the annotation itself will be plotted as un-transformed units (see Figure *Power projection axis*). E.g., if *stride* = 1 and power = 0.5 (i.e., sqrt), then equidistant annotations labeled 1, 4, 9, ... will appear.

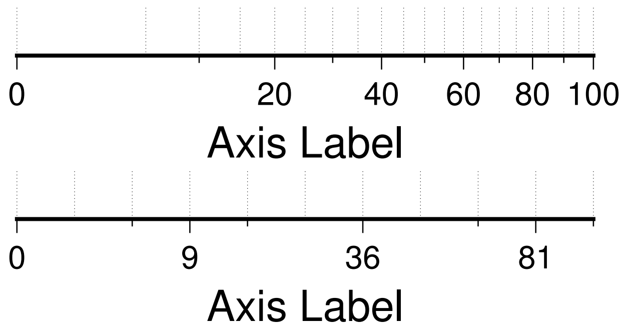


Figure 8.10: Exponential or power projection axis. (top) Using an exponent of 0.5 yields a $sqrt(x)$ axis. Here, intervals refer to actual data values, in -R0/100/0/0.9 -JX3ip0.5/0.25i -Ba20f10g5. (bottom) Here, intervals refer to projected values, although the annotation uses the corresponding unprojected values, as in -Ba3f2g1p.

Cartesian time axes

What sets time axis apart from the other kinds of plot axes is the numerous ways in which we may want to tick and annotate the axis. Not only do we have both primary and secondary annotation items but we also have interval annotations versus tickmark annotations, numerous time units, and several ways in which to modify the plot. We will demonstrate this flexibility with a series of examples. While all our examples will only show a single x-axis (south, selected via **-BS**), time-axis annotations are supported for all axes.

Our first example shows a time period of almost two months in Spring 2000. We want to annotate the month intervals as well as the date at the start of each week:

```
gmt set FORMAT_DATE_MAP=-o FONT_ANNOT_PRIMARY +9p
gmt psbasemap -R2000-4-1T/2000-5-25T/0/1 -JX5i/0.2i -Bpa7Rf1d -Bsa1O -BS -P > GMT_-B_time1.ps
```

These commands result in Figure *Cartesian time axis*. Note the leading hyphen in the *FORMAT_DATE_MAP* removes leading zeros from calendar items (e.g., 02 becomes 2).

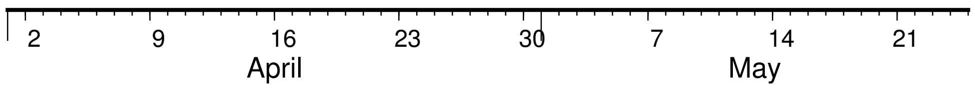


Figure 8.11: Cartesian time axis, example 1

The next example shows two different ways to annotate an axis portraying 2 days in July 1969:

```
gmt set FORMAT_DATE_MAP "o dd" FORMAT_CLOCK_MAP hh:mm FONT_ANNOT_PRIMARY +9p
gmt psbasemap -R1969-7-21T/1969-7-23T/0/1 -JX5i/0.2i -Bpa6Hf1h -Bsa1K -BS -P -K > GMT_-B_time2.ps
gmt psbasemap -R -J -Bpa6Hf1h -Bsa1D -BS -O -Y0.65i >> GMT_-B_time2.ps
```

The lower example (Figure *Cartesian time axis, example 2*) chooses to annotate the weekdays (by specifying **a**1**K**) while the upper example choses dates (by specifying **a**1**D**). Note how the clock format only selects hours and minutes (no seconds) and the date format selects a month name, followed by one space and a two-digit day-of-month number.

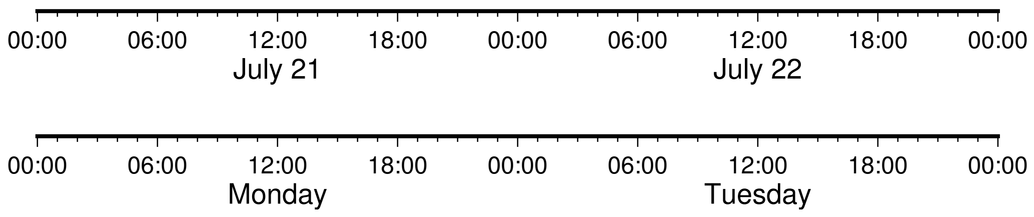


Figure 8.12: Cartesian time axis, example 2

The third example (Figure *Cartesian time axis, example 3*) presents two years, annotating both the years and every 3rd month.

```
gmt set FORMAT_DATE_MAP o FORMAT_TIME_PRIMARY_MAP Character FONT_ANNOT_PRIMARY +9p
gmt psbasemap -R1997T/1999T/0/1 -JX5i/0.2i -Bpa3Of1o -Bsa1Y -BS -P > GMT_-B_time3.ps
```

Note that while the year annotation is centered on the 1-year interval, the month annotations must be centered on the corresponding month and *not* the 3-month interval. The *FORMAT_DATE_MAP* selects month name only and *FORMAT_TIME_PRIMARY_MAP* selects the 1-character, upper case abbreviation of month names using the current language (selected by *TIME_LANGUAGE*).

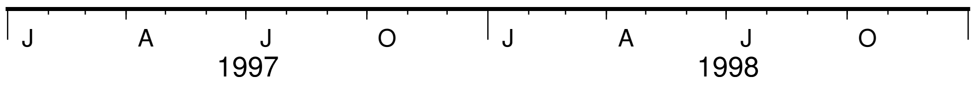


Figure 8.13: Cartesian time axis, example 3

The fourth example (Figure *Cartesian time axis, example 4*) only shows a few hours of a day, using relative time by specifying **t** in the **-R** option while the *TIME_UNIT* is **d** (for days). We select both primary and secondary annotations, ask for a 12-hour clock, and let time go from right to left:

```
gmt set FORMAT_CLOCK_MAP=-hham FONT_ANNOT_PRIMARY +9p
gmt psbasemap -R0.2t/0.35t/0/1 -JX-5i/0.2i -Bpa15mf5m -Bsa1H -BS -P > GMT_-B_time4.ps
```

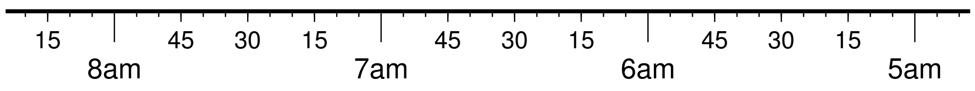


Figure 8.14: Cartesian time axis, example 4

The fifth example shows a few weeks of time (Figure *Cartesian time axis, example 5*). The lower axis shows ISO weeks with week numbers and abbreviated names of the weekdays. The upper uses Gregorian weeks (which start at the day chosen by *TIME_WEEK_START*); they do not have numbers.

```
gmt set FORMAT_DATE_MAP u FORMAT_TIME_PRIMARY_MAP Character \
       FORMAT_TIME_SECONDARY_MAP full FONT_ANNOT_PRIMARY +9p
gmt psbasemap -R1969-7-21T/1969-8-9T/0/1 -JX5i/0.2i -Bpa1K -Bsa1U -BS -P -K > GMT_-B_time5.ps
gmt set FORMAT_DATE_MAP o TIME_WEEK_START Sunday FORMAT_TIME_SECONDARY_MAP Chararacter
gmt psbasemap -R -J -Bpa3Kf1k -Bsa1r -BS -O -Y0.65i >> GMT_-B_time5.ps
```

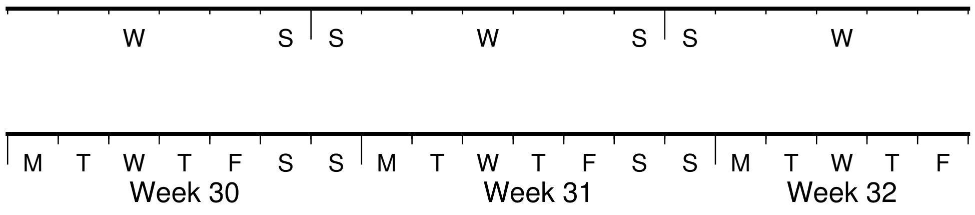


Figure 8.15: Cartesian time axis, example 5

Our sixth example (Figure *Cartesian time axis, example 6*) shows the first five months of 1996, and we have annotated each month with an abbreviated, upper case name and 2-digit year. Only the primary axes information is specified.

```
gmt set FORMAT_DATE_MAP "o yy" FORMAT_TIME_PRIMARY_MAP Abbreviated
gmt psbasemap -R1996T/1996-6T/0/1 -JX5i/0.2i -Ba1Of1d -BS -P > GMT_-B_time6.ps
```

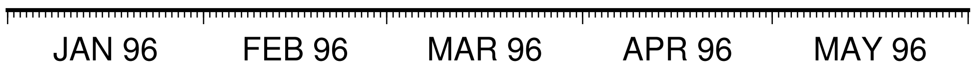


Figure 8.16: Cartesian time axis, example 6

Our seventh and final example (Figure *Cartesian time axis, example 7*) illustrates annotation of year-days. Unless we specify the formatting with a leading hyphen in *FORMAT_DATE_MAP* we get 3-digit integer days. Note that in order to have the two years annotated we need to allow for the annotation of small fractional intervals; normally such truncated interval must be at least half of a full interval.

```
gmt set FORMAT_DATE_MAP jjj TIME_INTERVAL_FRACTION 0.05 FONT_ANNOT_PRIMARY +9p
gmt psbasemap -R2000-12-15T/2001-1-15T/0/1 -JX5i/0.2i -Bpa5Df1d -Bsa1Y -BS -P > GMT_-B_time7.ps
```

Custom axes

Irregularly spaced annotations or annotations based on look-up tables can be implemented using the *custom* annotation mechanism. Here, we given the **c** (custom) type to the **-B** option followed by a filename that contains the annotations (and tick/grid-lines specifications) for one axis. The file can contain any number of comments (lines starting with #) and any number of records of the format

coord type [*label*]

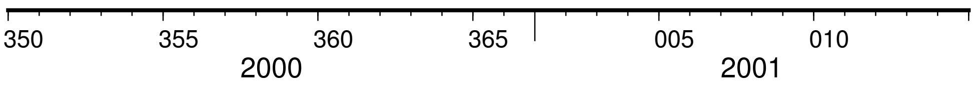


Figure 8.17: Cartesian time axis, example 7

The *coord* is the location of the desired annotation, tick, or grid-line, whereas *type* is a string composed of letters from **a** (annotation), **i** interval annotation, **f** frame tick, and **g** gridline. You must use either **a** or **i** within one file; no mixing is allowed. The coordinates should be arranged in increasing order. If *label* is given it replaces the normal annotation based on the *coord* value. Our last example (Figure *Custom and irregular annotations*) shows such a custom basemap with an interval annotations on the *x*-axis and irregular annotations on the *y*-axis.

```
cat << EOF > xannots.txt
416.0 ig Devonian
443.7 ig Silurian
488.3 ig Ordovician
542 ig Cambrian
EOF
cat << EOF > yannots.txt
0 a
1 a
2 f
2.71828 ag e
3 f
3.1415926 ag @~p@~
4 f
5 f
6 f
6.2831852 ag 2@~p@~
EOF
gmt psbasemap -R416/542/0/6.2831852 -JX-5i/2.5i -Bpx25f5g25+u" Ma" -Bpycyannots.txt \
              -BWS+glightblue -P -K > GMT_-B_custom.ps
gmt psbasemap -R416/542/0/6.2831852 -JX-5i/2.5i -Bsxcxannots.txt -BWS -O \
              --MAP_ANNOT_OFFSET_SECONDARY=10p --MAP_GRID_PEN_SECONDARY=2p >> GMT_-B_custom.ps
rm -f [xy]annots.txt
```

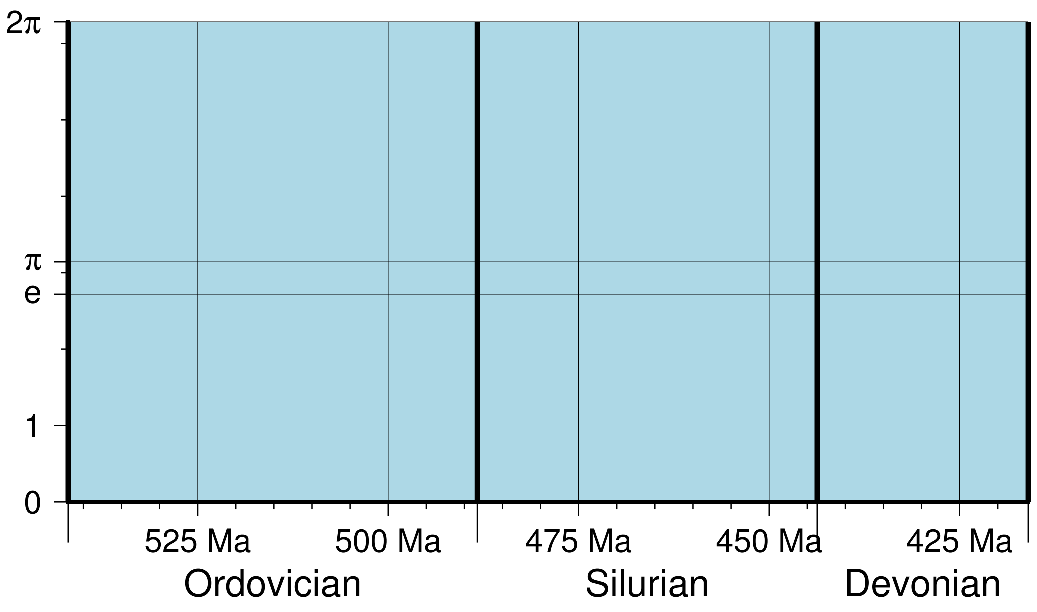


Figure 8.18: Custom and irregular annotations, tick-marks, and gridlines.

8.4.4 Portrait plot orientation: The -P option

The **-P** option selects Portrait plotting mode [8]. In general, a plot has an *x*-axis increasing from left to right and a *y*-axis increasing from bottom to top. If the paper is turned so that the long dimension of the paper is parallel to the *x*-axis then the plot is said to have *Landscape* orientation. If the long dimension of the paper parallels the *y*-axis the orientation is called *Portrait* (think of taking pictures with a camera and these words make sense). The default Landscape orientation is obtained by translating the origin in the *x*-direction (by the width of the chosen paper *PS_MEDIA*) and then rotating the coordinate system counterclockwise by 90. By default the *PS_MEDIA* is set to Letter (or A4 if SI is chosen); this value must be changed when using different media, such as 11” x 17” or large format plotters (Figure *Plot orientation*).

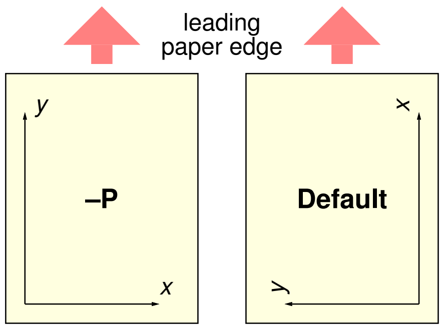


Figure 8.19: Users can specify Landscape [Default] or Portrait -P) orientation.

8.4.5 Plot overlays: The -K -O options

The **-K** and **-O** options control the generation of PostScript code for multiple overlay plots. All PostScript files must have a header (for initializations), a body (drawing the figure), and a trailer (printing it out) (see Figure *Multiple overlay plots*). Thus, when overlaying several GMT plots we must make sure that the first plot call omits the trailer, that all intermediate calls omit both header and trailer, and that the final overlay omits the header. The **-K** omits the trailer which implies that more PostScript code will be appended later [Default terminates the plot system]. The **-O** selects Overlay plot mode and omits the header information [Default initializes a new plot system]. Most unexpected results for multiple overlay plots can be traced to the incorrect use of these options. If you run only one plot program, ignore both the **-O** and **-K** options; they are only used when stacking plots.

8.4.6 Timestamps on plots: The -U option

The **-U** option draws UNIX System time stamp. Optionally, append an arbitrary text string (surrounded by double quotes), or the code **c**, which will plot the current command string (Figure *Time stamp*).

8.4.7 Verbose feedback: The -V option

The **-V** option selects verbose mode, which will send progress reports to standard error. Even more verbose levels are **-Vl** (long verbose) and **-Vd** (debug). Normal verbosity level produces only error and warning messages. This is the default or can be selected by using **-Vn**. If compiled with backward-compatibility support, the default is **-Vc**, which includes warnings about deprecated usage. Finally, **-Vq**

[8] For historical reasons, the GMT default is Landscape; see `gmt.conf` to change this.

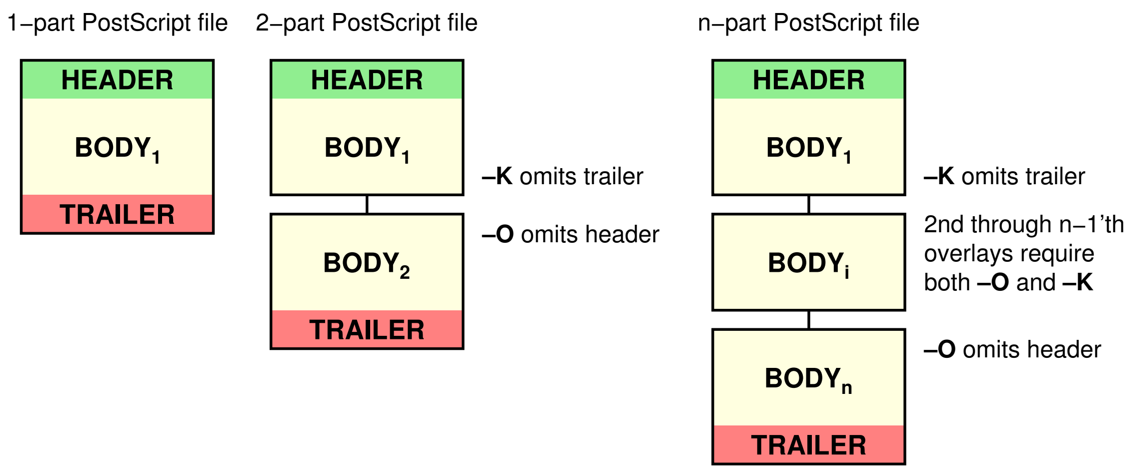


Figure 8.20: A final PostScript file consists of any number of individual pieces.

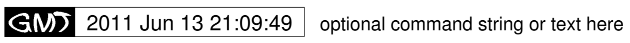


Figure 8.21: The -U option makes it easy to date a plot.

can be used to run without any warnings or errors. This option can also be set by specifying the default *GMT_VERBOSE*, as **quiet**, **normal**, **compat**, **verbose**, **long_verbose**, or **debug**, in order of increased verbosity.

8.4.8 Plot positioning and layout: The -X -Y options

The **-X** and **-Y** options shift origin of plot by (*xoff*,*yoff*) inches (Default is (*MAP_ORIGIN_X*, *MAP_ORIGIN_Y*) for new plots [9] and (0,0) for overlays (**-O**)). By default, all translations are relative to the previous origin (see Figure *Plot positioning*). Supply offset as **c** to center the plot in that direction relative to the page margin. Absolute translations (i.e., relative to a fixed point (0,0) at the lower left corner of the paper) can be achieve by prepending "a" to the offsets. Subsequent overlays will be co-registered with the previous plot unless the origin is shifted using these options. The offsets are measured in the current coordinates system (which can be rotated using the initial **-P** option; subsequent **-P** options for overlays are ignored).

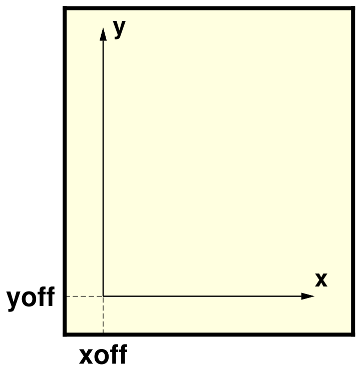


Figure 8.22: Plot origin can be translated freely with -X -Y.

[9] Ensures that boundary annotations do not fall off the page.

8.4.9 OGR/GMT GIS i/o: The -a option

GMT relies on external tools to translate geospatial files such as shapefiles into a format we can read. The tool **ogr2ogr** in the GDAL package can do such translations and preserve the aspatial metadata via a new OGR/GMT format specification (See Appendix P. The GMT Vector Data Format for OGR Compatibility). For this to be useful we need a mechanism to associate certain metadata values with required input and output columns expected by GMT programs. The **-a** option allows you to supply one or more comma-separated associations *col=name*, where *name* is the name of an aspatial attribute field in a OGR/GMT file and whose value we wish to as data input for column *col*. The given aspatial field thus replaces any other value already set. Note that *col = 0* is the first data columns. Note that if no aspatial attributes are needed then the **-a** option is not needed – GMT will still process and read such data files.

OGR/GMT input with -a option

If you need to populate GMT data columns with (constant) values specified by aspatial attributes, use **-a** and append any number of comma-separated *col=name* associations. E.g., *2=depth* will read the spatial *x,y* columns from the file and add a third (*z*) column based on the value of the aspatial field called *depth*. You can also associate aspatial fields with other settings such as labels, fill colors, pens, and values used to look-up colors. Do so by letting the *col* value be one of **D**, **G**, **L**, **T**, **W**, or **Z**. This works analogously to how standard multi-segment files can pass such options via its segment headers (See Appendix B. GMT file formats).

OGR/GMT output with -a option

You can also make GMT table-writing tools output the OGR/GMT format directly. Again, specify if certain GMT data columns with constant values should be stored as aspatial metadata using the *col=name*[:*type*], where you can optionally specify what data type it should be (double, integer, string, logical, byte, or datetime) [double is default]. As for input, you can also use the special *col* entries of **D**, **G**, **L**, **T**, **W**, or **Z** to have values stored as options in segment headers be used as the source for the name aspatial field. Finally, for output you must append +**g***geometry*, where *geometry* can be any of [**M**]**POINT**|**LINE**|**POLY**; the **M** represent the multi-versions of these three geometries. Use upper-case +**G** to signal that you want to split any line or polygon features that straddle the Dateline.

8.4.10 Binary table i/o: The -b option

All GMT programs that accept table data input may read ASCII, native binary, or netCDF data. Native binary files may have a header section and the **-h***n* option (see Section Header data records: The -h option) can be used to skip the first *n* bytes. The data record can be in any format, mixing different data types and even containing byte-swapped items. When using native binary data the user must be aware of the fact that GMT has no way of determining the actual number of columns in the file. You must therefore pass that information to GMT via the binary **-bi** option, where *n* is the actual number of data columns and **t** must be one of **c** (signed 1-byte character, int8_t), **u** (unsigned 1-byte character, uint8_t), **h** (signed 2-byte int, int16_t), **H** (unsigned 2-byte int, uint16_t), **i** (signed 4-byte int, int32_t), **I** (unsigned 4-byte int, uint32_t), **l** (signed 8-byte int, int64_t), **L** (unsigned 8-byte int, uint64_t), **f** (4-byte single-precision float), and **d** (8-byte double-precision float). In addition, use **x** to skip *n* bytes anywhere in the record. For a mixed-type data record you can concatenate several [*n*]**t** combinations, separated by commas. You may append **w** to any of the items to force byte-swapping. Alternatively, append **+L**|**B** to indicate that the entire data file should be read or written as little- or big-endian, respectively. Here, *n* is the number of each item in your binary file. Note that *n* may be larger than *m*, the number of columns that the GMT program requires to do its task. If *n* is not given then it defaults to *m* and all columns are assumed to be of the single specified type **t** [**d** (double), if not set]. If $n < m$ an error is generated. Multiple segment files are allowed and the segment headers are assumed to be records where all the fields equal NaN.

For binary output, use the **-bo** option; see **-bi** for further details.

Because of its meta data, reading netCDF tables (i.e., netCDF files containing 1-dimensional arrays) is quite a bit less complex than reading native binary files. When feeding netCDF tables to programs like `psxy`, the program will automatically recognize the format and read whatever amount of columns are needed for that program. To steer which columns are to be read, the user can append the suffix **?***var1***/***var2***/**... to the netCDF file name, where *var1*, *var2*, etc. are the names of the variables to be processed. No **-bi** option is needed in this case.

Currently, netCDF tables can only be input, not output. For more information, see Appendix [app:B].

8.4.11 Number of Copies: The -c option

The **-c** option specifies the number of plot copies [Default is 1]. This value is embedded in the PostScript file and will make a printer issue the chosen number of copies without respooling.

8.4.12 Data type selection: The -f option

When map projections are not required we must explicitly state what kind of data each input or output column contains. This is accomplished with the **-f** option. Following an optional **i** (for input only) or **o** (for output only), we append a text string with information about each column (or range of columns) separated by commas. Each string starts with the column number (0 is first column) followed by either **x** (longitude), **y** (latitude), **T** (absolute calendar time) or **t** (relative time). If several consecutive columns have the same format you may specify a range of columns rather than a single column, i.e., 0–4 for the first 5 columns. For example, if our input file has geographic coordinates (latitude, longitude) with absolute calendar coordinates in the columns 3 and 4, we would specify **fi0y**,1**x**,3–4**T**. All other columns are assumed to have the default, floating point format and need not be set individually. The shorthand **-f**[**i**|**o**]**g** means **-f**[**i**|**o**]0x,1y (i.e., geographic coordinates). A special use of **-f** is to select **-fp**[*unit*], which *requires* **-J** and lets you use *projected* map coordinates (e.g., UTM meters) as data input. Such coordinates are automatically inverted to longitude, latitude during the data import. Optionally, append a length *unit* (see Table *distunits*) [meter]. For more information, see Sections Input data formats and Output data formats.

8.4.13 Data gap detection: The -g option

GMT has several mechanisms that can determine line segmentation. Typically, data segments are separated by multiple segment header records (see Appendix B. GMT file formats). However, if key data columns contain a NaN we may also use that information to break lines into multiple segments. This behavior is modified by the parameter **IO_NAN_RECORDS** which by default is set to *skip*, meaning such records are considered bad and simply skipped. If you wish such records to indicate a segment boundary then set this parameter to *pass*. Finally, you may wish to indicate gaps based on the data values themselves. The **-g** option is used to detect gaps based on one or more criteria (use **-g+** if *all* the criteria must be met; otherwise only one of the specified criteria needs to be met to signify a data gap). Gaps can be based on excessive jumps in the *x*- or *y*-coordinates (**-gx** or **-gy**), or on the distance between points (**-gd**). Append the *gap* distance and optionally a unit for actual distances. For geographic data the optional unit may be arc **d**egree, **m**inute, and **s**econd, or m**e**ter [Default], **f**eet, **k**ilometer, **M**iles, or

nautical miles. For programs that map data to map coordinates you can optionally specify these criteria to apply to the projected coordinates (by using upper-case **-gX**, **-gY** or **-gD**). In that case, choose from **c**entimeter, **i**nch or **p**oint [Default unit is controlled by **PROJ_LENGTH_UNIT**]. Note: For **-gx** or **-gy** with time data the unit is instead controlled by *TIME_UNIT*.

8.4.14 Header data records: The -h option

The **-h**[**i**|**o**][*n_recs*] option lets GMT know that input file(s) have *n_recs* header records [0]. If there are more than one header record you must specify the number after the **-h** option, e.g., **-h**4. Note that blank lines and records that start with the character # are automatically considered header records and skipped. Thus, *n_recs* refers to general text lines that do *not* start with # and thus must specifically be skipped in order for the programs to function properly. The default number of such header records if **-h** is used is one of the many parameters in the `gmt.conf` file (**IO_N_HEADER_RECS**, by default 0), but can be overridden by **-h***n_header_recs*. Normally, programs that both read and write tables will output the header records that are found on input. Use **-hi** to suppress the writing of header records. You can use the **-h** options modifiers to to tell programs to output extra header records for titles, remarks or column names identifying each data column.

When **-b** is used to indicate binary data the **-h** takes on a slightly different meaning. Now, the *n_recs* argument is taken to mean how many *bytes* should be skipped (on input) or padded with the space character (on output).

8.4.15 Input columns selection: The -i option

The **-i***columns* option allows you to specify which input file data columns to use and in what order. By default, GMT will read all the data columns in the file, starting with the first column (0). Using **-i** modifies that process. For instance, to use the 4th, 7th, and 3rd data column as the required *x,y,z* to `blockmean` you would specify **-i**3,6,2 (since 0 is the first column). The chosen data columns will be used as is. Optionally, you can specify that input columns should be transformed according to a linear or logarithmic conversion. Do so by appending [**l**][**s***scale*][**o***offset*] to each column (or range of columns). All items are optional: The **l** implies we should first take $\log_{10}$ of the data [leave as is]. Next, we may scale the result by the given *scale* [1]. Finally, we add in the specified *offset* [0].

8.4.16 Grid interpolation parameters: The -n option

The **-n***type* option controls parameters used for 2-D grids resampling. You can select the type of spline used (**-nb** for B-spline smoothing, **-nc** for bicubic [Default], **-nl** for bilinear, or **-nn** for nearest-node value). For programs that support it, antialiasing is by default on; optionally, append **+a** to switch off antialiasing. By default, boundary conditions are set according to the grid type and extent. Change boundary conditions by appending **+b***BC*, where *BC* is either **g** for geographic boundary conditions or one (or both) of **n** and **p** for natural or periodic boundary conditions, respectively. Append **x** or **y** to only apply the condition in one dimension. E.g., **-nb+nxpy** would imply natural boundary conditions in the *x* direction and periodic conditions in the *y* direction. Finally, append **+t***threshold* to control how close to nodes with NaN the interpolation should go. A *threshold* of 1.0 requires all (4 or 16) nodes involved in the interpolation to be non-NaN. 0.5 will interpolate about half way from a non-NaN value; 0.1 will go about 90% of the way, etc.

8.4.17 Output columns selection: The -o option

The **-o***columns* option allows you to specify which columns to write on output and in what order. By default, GMT will write all the data columns produced by the program. Using **-o** modifies that process. For instance, to write just the 4th and 2nd data column to the output you would use **-o**3,1 (since 0 is the first column).

8.4.18 Perspective view: The -p option

All plotting programs that normally produce a flat, two-dimensional illustration can be told to view this flat illustration from a particular vantage point, resulting in a perspective view. You can select perspective view with the **-p** option by setting the azimuth and elevation of the viewpoint [Default is 180/90]. When **-p** is used in consort with **-Jz** or **-JZ**, a third value can be appended which indicates at which *z*-level all 2-D material, like the plot frame, is plotted (in perspective) [Default is at the bottom of the z-axis]. For frames used for animation, you may want to append **+** to fix the center of your data domain (or specify a particular world coordinate point with **+w***lon0/lat*[*z*]) which will project to the center of your page size (or you may specify the coordinates of the *projected* view point with **+v***x0/y0*. When **-p** is used without any further arguments, the values from the last use of **-p** in a previous GMTcommand will be used.

8.4.19 Grid registration: The -r option

All 2-D grids in GMT have their nodes organized in one of two ways, known as *gridline*- and *pixel* registration. The GMT default is gridline registration; programs that allow for the creation of grids can use the **-r** option to select pixel registration instead.

Gridline 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 ($x_{inc} \cdot y_{inc}$) centered on each node (left side of Figure *Grid registration*). In the case of grid line registration the number of nodes are related to region and grid spacing by

$$\begin{array}{rcl} nx & = & (x_{max} - x_{min})/x_{inc} + 1 \\ ny & = & (y_{max} - y_{min})/y_{inc} + 1 \end{array}$$

which for the example in left side of Figure *Gridline registration* yields nx = ny = 4.

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 (right side of Figure *Grid registration*). In the case of pixel registration the number of nodes are related to region and grid spacing by

$$\begin{array}{rcl} nx & = & (x_{max} - x_{min})/x_{inc} \\ ny & = & (y_{max} - y_{min})/y_{inc} \end{array}$$

Thus, given the same region (**-R**) and grid spacing, the pixel-registered grids have one less column and one less row than the gridline-registered grids; here we find nx = ny = 3.

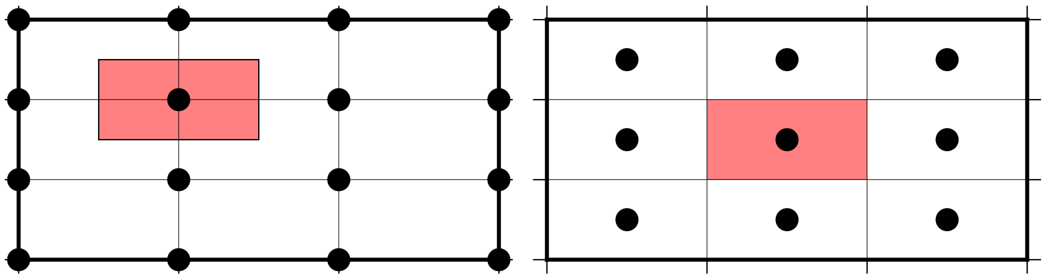


Figure 8.23: Gridline- and pixel-registration of data nodes.

8.4.20 NaN-record treatment: The -s option

We can use this option to suppress output for records whose z-value equals NaN (by default we output all records). Alternatively, append **r** to reverse the suppression, i.e., only output the records whose z-value equals NaN. Use **-sa** to suppress output records where one or more fields (and not necessarily z) equal NaN. Finally, you can supply a comma-separated list of all columns or column ranges to consider for this NaN test.

8.4.21 Layer PDF transparency: The -t option

While the PostScript language does not support transparency, PDF does, and via PostScript extensions one can manipulate the transparency levels of objects. The **-t** option allows you to change the transparency level for the current overlay by appending a percentage in the 0–100 range; the default is 0, or opaque. Transparency may also be controlled on a feature by feature basis when setting color or fill (see section Specifying area fill attributes).

8.4.22 Latitude/Longitude or Longitude/Latitude?: The -: option

For geographical data, the first column is expected to contain longitudes and the second to contain latitudes. To reverse this expectation you must apply the **-:** option. Optionally, append **i** or **o** to restrict the effect to input or output only. Note that command line arguments that may take geographic coordinates (e.g., **-R**) *always* expect longitude before latitude. Also, geographical grids are expected to have the longitude as first (minor) dimension.

8.5 Command line history

GMT programs "remember" the standardized command line options (See Section Standardized command line options) given during their previous invocations and this provides a shorthand notation for complex options. For example, if a basemap was created with an oblique Mercator projection, specified as

```
-Joc170W/25:30S/33W/56:20N/1:500000
```

then a subsequent `psxy` command to plot symbols only needs to state **-J**o in order to activate the same projection. In contrast, note that **-J** by itself will pick the most recently used projection. Previous commands are maintained in the file `gmt.history`, of which there will be one in each directory you run the programs from. This is handy if you create separate directories for separate projects since chances are that data manipulations and plotting for each project will share many of the same options. Note that an option spelled out on the command line will always override the previous entry in the `gmt.history` file and, if execution is successful, will replace this entry as the previous option argument in the `gmt.history` file. If you call several GMT modules piped together then GMT cannot guarantee that the `gmt.history` file is processed in the intended order from left to right. The only guarantee is that the file will not be clobbered since GMT uses advisory file locking. The uncertainty in processing order makes the use of shorthands in pipes unreliable. We therefore recommend that you only use shorthands in single process command lines, and spell out the full command option when using chains of commands connected with pipes.

8.6 Usage messages, syntax- and general error messages

Each program carries a usage message. If you enter the program name without any arguments, the program will write the complete usage message to standard error (your screen, unless you redirect it). This message explains in detail what all the valid arguments are. If you enter the program name followed by a *hyphen* (-) only you will get a shorter version which only shows the command line syntax and no detailed explanations. If you incorrectly specify an option or omit a required option, the program will produce syntax errors and explain what the correct syntax for these options should be. If an error occurs during the running of a program, the program will in some cases recognize this and give you an error message. Usually this will also terminate the run. The error messages generally begin with the name of the program in which the error occurred; if you have several programs piped together this tells you where the trouble is.

8.7 Standard input or file, header records

Most of the programs which expect table data input can read either standard input or input in one or several files. These programs will try to read *stdin* unless you type the filename(s) on the command line without the above hyphens. (If the program sees a hyphen, it reads the next character as an instruction; if an argument begins without a hyphen, it tries to open this argument as a filename). This feature allows you to connect programs with pipes if you like. If your input is ASCII and has one or more header records that do not begin with #, you must use the **-h** option (see Section Header data records: The -h option). ASCII files may in many cases also contain segment-headers separating data segments. These are called "multi-segment files". For binary table data the **-h** option may specify how many bytes should be skipped before the data section is reached. Binary files may also contain segment-headers separating data segments. These segment-headers are simply data records whose fields are all set to NaN; see Appendix [app:B] for complete documentation.

If filenames are given for reading, GMT programs will first look for them in the current directory. If the file is not found, the programs will look in two other directories pointed to by the *directory parameters* **DIR_DATA** and **DIR_USER** or by the environmental parameters **GMT_USERDIR** and **GMT_DATADIR** (if set). They may be set by the user to point to directories that contain data sets of general use, thus eliminating the need to specify a full path to these files. Usually, the **DIR_DATA** directory will hold data sets of a general nature (tables, grids), whereas the **DIR_USER** directory may hold miscellaneous data sets more specific to the user; this directory also stores GMT defaults and other configuration files. See *directory parameters* for details. Program output is always written to the current directory unless a full path has been specified.

8.8 Verbose operation

Most of the programs take an optional **-V** argument which will run the program in the "verbose" mode (see Section Verbose feedback: The -V option). Verbose will write to standard error information about the progress of the operation you are running. Verbose reports things such as counts of points read, names of data files processed, convergence of iterative solutions, and the like. Since these messages are written to *stderr*, the verbose talk remains separate from your data output. You may optionally choose among five models of *verbosity*; each mode adds more messages with an increasing level of details. The modes are

> **q** Complete silence, not even fatal error messages.
>
> **n** Warnings and progress messages [Default].
>
> **c** Warnings about deprecated usage (if compiled for compatibility).
>
> **l** Detailed progress messages.
>
> **d** Debugging messages.

The verbosity is cumulative, i.e., mode **l** means all messages of mode **n** as well. will be reported.

8.9 Program output

Most programs write their results, including PostScriptplots, to standard output. The exceptions are those which may create binary netCDF grid files such as `surface` (due to the design of netCDF a filename must be provided; however, alternative binary output formats allowing piping are available; see Section Grid file format specifications). Most operating systems let you can redirect standard output to a file or pipe it into another process. Error messages, usage messages, and verbose comments are written to standard error in all cases. You can usually redirect standard error as well, if you want to create a log file of what you are doing. The syntax for redirection differ among the main shells (Bash and C-shell) and is a bit limited in DOS.

8.10 Input data formats

Most of the time, GMT will know what kind of *x* and *y* coordinates it is reading because you have selected a particular coordinate transformation or map projection. However, there may be times when you must explicitly specify what you are providing as input using the **-f** switch. When binary input data are expected (**-bi**) you must specify exactly the format of the records. However, for ASCII input there are numerous ways to encode data coordinates (which may be separated by white-space or commas). Valid input data are generally of the same form as the arguments to the **-R** option (see Section Data domain or map region: The -R option), with additional flexibility for calendar data. Geographical coordinates, for example, can be given in decimal degrees (e.g., -123.45417) or in the []*ddd*[:*mm*[:*ss*[.*xxx*]]][**W**| **E**| **S**| **N**] format (e.g., 123:27:15W). With **-fp** you may even supply projected data like UTM coordinates.

Because of the widespread use of incompatible and ambiguous formats, the processing of input date components is guided by the template *FORMAT_DATE_IN* in your `gmt.conf` file; it is by default set to *yyyy-mm-dd*. Y2K-challenged input data such as 29/05/89 can be processed by setting *FORMAT_DATE_IN* to dd/mm/yy. A complete description of possible formats is given in the `gmt.conf` man page. The *clock* string is more standardized but issues like 12- or 24-hour clocks complicate matters as well as the presence or absence of delimiters between fields. Thus, the processing of input clock coordinates is guided by the template *FORMAT_CLOCK_IN* which defaults to *hh:mm:ss.xxx*.

GMT programs that require a map projection argument will implicitly know what kind of data to expect, and the input processing is done accordingly. However, some programs that simply report on minimum and maximum values or just do a reformatting of the data will in general not know what to expect, and furthermore there is no way for the programs to know what kind of data other columns (beyond the leading *x* and *y* columns) contain. In such instances we must explicitly tell GMT that we are feeding it data in the specific geographic or calendar formats (floating point data are assumed by default). We specify the data type via the **-f** option (which sets both input and output formats; use **-fi** and **-fo** to set input and output separately). For instance, to specify that the the first two columns are longitude and latitude, and that the third column (e.g., *z*) is absolute calendar time, we add **-fi**0x,1y,2T to the command line. For more details, see the man page for the program you need to use.

8.11 Output data formats

The numerical output from GMT programs can be binary (when **-bo** is used) or ASCII [Default]. In the latter case the issue of formatting becomes important. GMT provides extensive machinery for allowing just about any imaginable format to be used on output. Analogous to the processing of input data, several templates guide the formatting process. These are *FORMAT_DATE_OUT* and *FORMAT_CLOCK_OUT* for calendar-time coordinates, *FORMAT_GEO_OUT* for geographical coordinates, and *FORMAT_FLOAT_OUT* for generic floating point data. In addition, the user have control over how columns are separated via the *IO_COL_SEPARATOR* parameter. Thus, as an example, it is possible to create limited FORTRAN-style card records by setting *FORMAT_FLOAT_OUT* to %7.3lf and *IO_COL_SEPARATOR* to none [Default is tab].

8.12 PostScript features

PostScript is a command language for driving graphics devices such as laser printers. It is ASCII text which you can read and edit as you wish (assuming you have some knowledge of the syntax). We prefer this to binary metafile plot systems since such files cannot easily be modified after they have been created. GMT programs also write many comments to the plot file which make it easier for users to orient themselves should they need to edit the file (e.g., % Start of x-axis) [10]. All GMT programs create PostScript code by calling the `PSL` plot library (The user may call these functions from his/her own C or FORTRAN plot programs. See the manual pages for `PSL` syntax). Although GMT programs can create very individualized plot code, there will always be cases not covered by these programs. Some knowledge of PostScript will enable the user to add such features directly into the plot file. By default, GMT will produce freeform PostScript output with embedded printer directives. To produce Encapsulated PostScript (EPS) that can be imported into graphics programs such as **CorelDraw**, **Illustrator** or **InkScape** for further embellishment, simply run `ps2raster` **-Te**. See Appendix [app:C] for an extensive discussion of converting PostScript to other formats.

8.13 Specifying pen attributes

A pen in GMT has three attributes: *width*, *color*, and *style*. Most programs will accept pen attributes in the form of an option argument, with commas separating the given attributes, e.g.,

-W[*width*[**c|i|p**]],[*color*],[*style*[**c|i|p|**]]

[10] To keep PostScript files small, such comments are by default turned off; see *PS_COMMENTS* to enable them.

Width is by default measured in points (1/72 of an inch). Append **c**, **i**, or **p** to specify pen width in cm, inch, or points, respectively. Minimum-thickness pens can be achieved by giving zero width, but the result is device-dependent. Finally, a few predefined pen names can be used: default, faint, and {thin, thick, fat}[er|est], and obese. Table *pennames* shows this list and the corresponding pen widths.

faint	0	thicker	1.5p
default	0.25p	thickest	2p
thinnest	0.25p	fat	3p
thinner	0.50p	fatter	6p
thin	0.75p	fattest	12p
thick	1.0p	obese	18p

The *color* can be specified in five different ways:

1. Gray. Specify a *gray* shade in the range 0–255 (linearly going from black [0] to white [255]).
2. RGB. Specify *r*/*g*/*b*, each ranging from 0–255. Here 0/0/0 is black, 255/255/255 is white, 255/0/0 is red, etc.
3. HSV. Specify *hue-saturation-value*, with the former in the 0–360 degree range while the latter two take on the range 0–1 [11].
4. CMYK. Specify *cyan*/*magenta*/*yellow*/*black*, each ranging from 0–100%.
5. Name. Specify one of 663 valid color names. Use **man gmtcolors** to list all valid names. A very small yet versatile subset consists of the 29 choices *white*, *black*, and [light:|dark]{*red, orange, yellow, green, cyan, blue, magenta, gray*|*grey, brown*}. The color names are case-insensitive, so mixed upper and lower case can be used (like *DarkGreen*).

The *style* attribute controls the appearance of the line. A ”.” yields a dotted line, whereas a dashed pen is requested with “-”. Also combinations of dots and dashes, like ”.-” for a dot-dashed line, are allowed. The lengths of dots and dashes are scaled relative to the pen width (dots has a length that equals the pen width while dashes are 8 times as long; gaps between segments are 4 times the pen width). For more detailed attributes including exact dimensions you may specify *string*:*offset*, where *string* is a series of numbers separated by underscores. These numbers represent a pattern by indicating the length of line segments and the gap between segments. The *offset* phase-shifts the pattern from the beginning the line. For example, if you want a yellow line of width 0.1 cm that alternates between long dashes (4 points), an 8 point gap, then a 5 point dash, then another 8 point gap, with pattern offset by 2 points from the origin, specify **-W**0.1c,yellow,4_8_5_8:2p. Just as with pen width, the default style units are points, but can also be explicitly specified in cm, inch, or points (see *width* discussion above).

Table *penex* contains additional examples of pen specifications suitable for, say, `psxy`.

[11] For an overview of color systems such as HSV, see Appendix [app:I].

-W0.5p	0.5 point wide line of default color and style
-Wgreen	Green line with default width and style
-Wthin,red,-	Dashed, thin red line
-Wfat,.	Fat dotted line with default color
-W0.1c,120-1-1	Green (in h-s-v) pen, 1 mm thick
-Wfaint,100/0/0/0,..-	Very thin, cyan (in c/m/y/k), dot-dot-dashed line

In addition to these pen settings there are several PostScript settings that can affect the appearance of lines. These are controlled via the GMT defaults settings *PS_LINE_CAP*, *PS_LINE_JOIN*, and *PS_MITER_LIMIT*. They determine how a line segment ending is rendered, be it at the termination of a solid line or at the end of all dashed line segments making up a line, and how a straight lines of finite thickness should behave when joined at a common point. By default, line segments have rectangular ends, but this can change to give rounded ends. When *PS_LINE_CAP* is set to round the a segment length of zero will appear as a circle. This can be used to created circular dotted lines, and by manipulating the phase shift in the *style* attribute and plotting the same line twice one can even alternate the color of adjacent items. Figure *Line appearance* shows various lines made in this fashion. See the `gmt.conf` man page for more information.

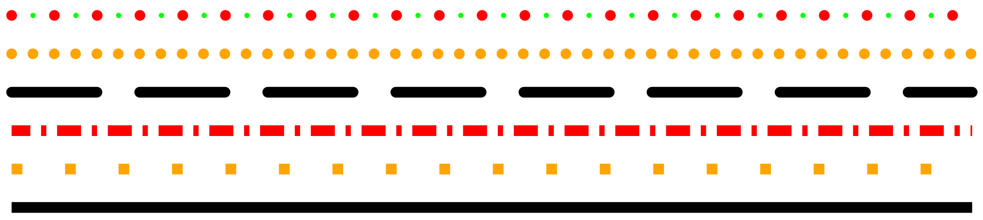


Figure 8.24: Line appearance can be varied by using *PS_LINE_CAP*

8.14 Specifying area fill attributes

Many plotting programs will allow the user to draw filled polygons or symbols. The fill specification may take two forms:

-G*fill*

-Gp*dpi/pattern*[:**B***color*[**F***color*]]

fill: In the first case we may specify a *gray* shade (0–255), RGB color (*r/g/b* all in the 0–255 range or in hexadecimal *#rrggbb*), HSV color (*hue-saturation-value* in the 0–360, 0–1, 0–1 range), CMYK color (*cyan/magenta/yellow/black*, each ranging from 0–100%), or a valid color *name*; in that respect it is similar to specifying the pen color settings (see pen color discussion under Section Specifying pen attributes).

pattern: The second form allows us to use a predefined bit-image pattern. *pattern* can either be a number in the range 1–90 or the name of a 1-, 8-, or 24-bit Sun raster file. The former will result in one of the 90 predefined 64 x 64 bit-patterns provided with GMT and reproduced in Appendix [app:E]. The latter allows the user to create customized, repeating images using standard Sun raster files [12]. The *dpi* parameter sets the resolution of this image on the page; the area fill is thus made up of a series of these "tiles". Specifying *dpi* as 0 will result in highest resolution obtainable given the present dpi setting in `gmt.history`. By specifying upper case **-GP** instead of **-Gp** the image will be bit-reversed, i.e., white and black areas will be interchanged (only applies to

[12] Convert other graphics formats to Sun ras format using ImageMagick's **convert** program.

1-bit images or predefined bit-image patterns). For these patterns and other 1-bit images one may specify alternative background and foreground colors (by appending :**B***color*[**F***color*]) that will replace the default white and black pixels, respectively. Setting one of the fore- or background colors to - yields a *transparent* image where only the back- *or* foreground pixels will be painted.

Due to PostScript implementation limitations the raster images used with **-G** must be less than 146 x 146 pixels in size; for larger images see `psimage`. The format of Sun raster files is outlined in Appendix [app:B]. Note that under PostScript Level 1 the patterns are filled by using the polygon as a *clip path*. Complex clip paths may require more memory than the PostScript interpreter has been assigned. There is therefore the possibility that some PostScript interpreters (especially those supplied with older laserwriters) will run out of memory and abort. Should that occur we recommend that you use a regular gray-shade fill instead of the patterns. Installing more memory in your printer *may or may not* solve the problem!

Table *fillex* contains a few examples of fill specifications.

-G128	Solid gray
-G127/255/0	Chartreuse, R/G/B-style
-G#00ff00	Green, hexadecimal RGB code
-G25-0.86-0.82	Chocolate, h-s-v-style
-GDarkOliveGreen1	One of the named colors
-Gp300/7	Simple diagonal hachure pattern in b/w at 300 dpi
-Gp300/7:Bred	Same, but with red lines on white
-Gp300/7:BredF-	Now the gaps between red lines are transparent
-Gp100/marble.ras	Using user image of marble as the fill at 100 dpi

8.15 Specifying Fonts

The fonts used by GMT are typically set indirectly via the GMT defaults parameters. However, some programs, like `pstext` may wish to have this information passed directly. A font is specified by a comma-delimited attribute list of *size*, *fonttype* and *fill*, each of which is optional. The *size* is the font size (usually in points) but **c**, **i** or **p** can be added to indicate a specific unit. The *fonttype* is the name (case sensitive!) of the font or its equivalent numerical ID (e.g., Helvetica-Bold or 1). *fill* specifies the gray shade, color or pattern of the text (see section Specifying area fill attributes above). Optionally, you may append **=***pen* to the *fill* value in order to draw the text outline with the specified *pen*; if used you may optionally skip the filling of the text by setting *fill* to **-**. If any of the attributes is omitted their default or previous setting will be retained. See Appendix G. PostScript fonts used by GMT for a list of all fonts recognized by GMT.

8.16 Stroke, Fill and Font Transparency

The PostScript language has no built-in mechanism for transparency. However, PostScriptextensions make it possible to request transparency, and tools that can render such extensions will produce transparency effects. We specify transparency in percent: 0 is opaque [Default] while 100 is fully transparent (i.e., nothing will show). As noted in section Layer PDF transparency: The -t option, we can control transparency on a layer-by-layer basis using the **-t** option. However, we may also set transparency as an attribute of stroke or fill (including for fonts) settings. Here, transparency is requested by appending @*transparency* to colors or pattern fills. The transparency *mode* can be changed by using the GMT default parameter *PS_TRANSPARENCY*; the default is Normal but you can choose among Color, ColorBurn, ColorDodge, Darken, Difference, Exclusion, HardLight, Hue, Lighten, Luminosity, Multiply, Normal, Overlay, Saturation, SoftLight, and Screen. For more information, see for instance (search online for) the Adobe pdfmark Reference Manual. Most printers and many PostScript viewers can neither print nor show transparency. They will simply ignore your attempt to create transparency and will plot any material as opaque. Ghostscript and its derivatives such as GMT's `ps2raster` support transparency (if compiled with the correct build option). Note: If you use **Acrobat Distiller** to create a PDF file you must first change some settings to make transparency effective: change the parameter /AllowTransparency to true in your *.joboptions file.

8.17 Color palette tables

Several programs, such as those which read 2-D gridded data sets and create colored images or shaded reliefs, need to be told what colors to use and over what *z*-range each color applies. This is the purpose of the color palette table (CPT file). These files may also be used by `psxy` and `psxyz` to plot color-filled symbols. For most applications, you will simply create a CPT file using the tool `makecpt` which will take an existing color table and resample it to fit your chosen data range, or use `grd2cpt` to build a CPT file based on the data distribution in one or more given grid files. However, in some situations you will need to make a CPT file by hand or using text tools like **awk** or **perl**.

Color palette tables (CPT) comes in two flavors: (1) Those designed to work with categorical data (e.g., data where interpolation of values is undefined) and (2) those designed for regular, continuously-varying data. In both cases the *fill* information follows the format given in Section Specifying area fill attributes. The z-values in CPT files can be scaled by using the **+u**|**U***unit* mechanism. Append these modifiers to your CPT filenames when used in GMT commands. The **+u***unit* modifier will scale z *from unit to* meters, while **+U***unit* does the inverse (scale z *from meters to unit*).

8.17.1 Categorical CPT files

Categorical data are information on which normal numerical operations are not defined. As an example, consider various land classifications (desert, forest, glacier, etc.) and it is clear that even if we assigned a numerical value to these categories (e.g., desert = 1, forest = 2, etc) it would be meaningless to compute average values (what would 1.5 mean?). For such data a special format of the CPT files are provided. Here, each category is assigned a unique key, a color or pattern, and an optional label (usually the category name) marked by a leading semi-colon. Keys must be monotonically increasing but do not need to be consecutive. The format is

key_1	*Fill*	[;*label*]
...		
key_n	*Fill*	[;*label*]

The *Fill* information follows the format given in Section Specifying area fill attributes. While not always applicable to categorical data, the background color (for *key*-values < key_1), foreground color (for *key*-values > key_n), and not-a-number (NaN) color (for *key*-values = NaN) are all defined in the `gmt.conf` file, but can be overridden by the statements

B	Fill_back
F	Fill_fore
N	Fill_nan

8.17.2 Regular CPT files

Suitable for continuous data types and allowing for color interpolations, the format of the regular CPT files is:

z_0	$\text{Color}_{\text{min}}$	z_1	$\text{Color}_{\text{max}}$	[**A**]	[;*label*]
...					
z_{n-2}	$\text{Color}_{\text{min}}$	z_{n-1}	$\text{Color}_{\text{max}}$	[**A**]	[;*label*]

Thus, for each "z-slice", defined as the interval between two boundaries (e.g., z_0 to z_1), the color can be constant (by letting Color_{max} = Color_{min} or -) or a continuous, linear function of z. If patterns are used then the second (max) pattern must be set to -. The optional flag **A** is used to indicate annotation of the color scale when plotted using `psscale`. The optional flag **A** may be **L**, **U**, or **B** to select annotation of the lower, upper, or both limits of the particular z-slice, respectively. However, the standard **-B** option can be used by `psscale` to affect annotation and ticking of color scales. Just as other GMT programs, the *stride* can be omitted to determine the annotation and tick interval automatically (e.g., **-Baf**). The optional semicolon followed by a text label will make `psscale`, when used with the **-L** option, place the supplied label instead of formatted z-values.

As for categorical tables, the background color (for z-values < z_0), foreground color (for z-values > z_{n-1}), and not-a-number (NaN) color (for z-values = NaN) are all defined in the `gmt.conf` file, but can be overridden by the statements

B	$\text{Fill}_{\text{back}}$
F	$\text{Fill}_{\text{fore}}$
N	Fill_{nan}

which can be inserted into the beginning or end of the CPT file. If you prefer the HSV system, set the `gmt.conf` parameter accordingly and replace red, green, blue with hue, saturation, value. Color palette tables that contain gray-shades only may replace the *r/g/b* triplets with a single gray-shade in the 0–255 range. For CMYK, give *c/m/y/k* values in the 0–100 range.

A few programs (i.e., those that plot polygons such as `grdview`, `psscale`, `psxy` and `psxyz`) can accept pattern fills instead of gray-shades. You must specify the pattern as in Section Specifying area fill attributes (no leading **-G** of course), and only the first pattern (for low z) is used (we cannot interpolate between patterns). Finally, some programs let you skip features whose z-slice in the CPT file has gray-shades set to -. As an example, consider

30	p200/16	80	-
80	-	100	-
100	200/0/0	200/255/255	0
200	yellow	300	green

where slice 30 < z < 80 is painted with pattern # 16 at 200 dpi, slice 80 < z < 100 is skipped, slice 100 < z < 200 is painted in a range of dark red to yellow, whereas the slice 200 < z < 300 will linearly yield colors from yellow to green, depending on the actual value of z.

Some programs like `grdimage` and `grdview` apply artificial illumination to achieve shaded relief maps. This is typically done by finding the directional gradient in the direction of the artificial light source and scaling the gradients to have approximately a normal distribution on the interval [-1,+1]. These intensities are used to add "white" or "black" to the color as defined by the z-values and the CPT file. An intensity of zero leaves the color unchanged. Higher values will brighten the color, lower values will darken it, all without changing the original hue of the color (see Appendix [app:I] for more details). The illumination is decoupled from the data grid file in that a separate grid file holding intensities in the [-1,+1] range must be provided. Such intensity files can be derived from the data grid using `grdgradient` and modified with `grdhisteq`, but could equally well be a separate data set. E.g., some side-scan sonar systems collect both bathymetry and backscatter intensities, and one may want to use the latter information to specify the illumination of the colors defined by the former. Similarly, one could portray magnetic anomalies superimposed on topography by using the former for colors and the latter for shading.

8.18 The Drawing of Vectors

GMT supports plotting vectors in various forms. A vector is one of many symbols that may be plotted by `psxy` and `psxyz`, is the main feature in `grdvector`, and is indirectly used by other programs. All vectors plotted by GMT consist of two separate parts: The vector line (controlled by the chosen pen attributes) and the optional vector head(s) (controlled by the chosen fill). We distinguish between three types of vectors:

1. Cartesian vectors are plotted as straight lines. They can be specified by a start point and the direction and length (in map units) of the vector, or by its beginning and end point. They may also be specified giving the azimuth and length (in km) instead.
2. Circular vectors are (as the name implies) drawn as circular arcs and can be used to indicate opening angles. It accepts an origin, a radius, and the beginning and end angles.
3. Geo-vectors are drawn using great circle arcs. They are specified by a beginning point and the azimuth and length (in km) of the vector, or by its beginning and end point.

There are numerous attributes you can modify, including how the vector should be justified relative to the given point (beginning, center, or end), where heads (if any) should be placed, if the head should just be the left or right half, if the vector attributes should shrink for vectors whose length are less than a given cutoff length, and the size and shape of the head. These attributes are detailed further in the relevant manual pages.

8.19 Character escape sequences

For annotation labels or text strings plotted with `pstext`, GMT provides several escape sequences that allow the user to temporarily switch to the symbol font, turn on sub- or superscript, etc., within words. These conditions are toggled on/off by the escape sequence @**x**, where **x** can be one of several types. The escape sequences recognized in GMT are listed in Table *escape*. Only one level of sub- or superscript is supported. Note that under Windows the percent symbol indicates a batch variable, hence you must use two percent-signs for each one required in the escape sequence for font switching.

@~	Turns symbol font on or off
@+	Turns superscript on or off
@-	Turns subscript on or off
@#	Turns small caps on or off
@_	Turns underline on or off
@%*fontno*%	Switches to another font; @%% resets to previous font
@:*size*:	Switches to another font size; @:: resets to previous size
@;*color*;	Switches to another font color; @;; resets to previous color
@!	Creates one composite character of the next two characters
@@	Prints the @ sign itself

Shorthand notation for a few special European characters has also been added:

Code	Effect	Code	Effect
@E	Æ	@e	æ
@O	Ø	@o	ø
@A	Å	@a	å
@C	Ç	@c	ç
@N	Ñ	@n	ñ
@U	Ü	@u	ü
@s	ß		

PostScript fonts used in GMT may be re-encoded to include several accented characters used in many European languages. To access these, you must specify the full octal code \xxx allowed for your choice of character encodings determined by the *PS_CHAR_ENCODING* setting described in the `gmt.conf` man page. Only the special characters belonging to a particular encoding will be available. Many characters not directly available by using single octal codes may be constructed with the composite character mechanism @!.

Some examples of escape sequences and embedded octal codes in GMT strings using the Standard+ encoding:

`2@~p@~r@+2@+h@-0@- E\363tv\363s` = $2\pi r^2 h_0$ Eötvös

`10@+-3 @Angstr@om` = 10^{-3} Ångstrøm

`Se@nor Gar@con` = Señor Garçon

`M@!\305anoa stra@se` = Manoa straße

`A@\#cceleration@\# (ms@+-2@+)` =

The option in `pstext` to draw a rectangle surrounding the text will not work for strings with escape sequences. A chart of characters and their octal codes is given in Appendix [app:F].

8.20 Grid file format specifications

GMT has the ability to read and write grids using more than one grid file format (see Table *grdformats* for supported format and their IDs). For reading, GMT will automatically determine the format of grid files, while for writing you will normally have to append =*ID* to the filename if you want GMT to use a different format than the default.

By default, GMT will create new grid files using the **nf** format; however, this behavior can be overridden by setting the *IO_GRIDFILE_FORMAT* defaults parameter to any of the other recognized values (or by appending =*ID*).

GMT can also read netCDF grid files produced by other software packages, provided the grid files satisfy the COARDS and Hadley Centre conventions for netCDF grids. Thus, products created under those conventions (provided the grid is 2-, 3-, 4-, or 5-dimensional) can be read directly by GMT and the netCDF grids written by GMT can be read by other programs that conform to those conventions. Three such programs are ncview, Panoply, and ncBrowse ; others can be found on the netCDF website.

In addition, users with some C-programming experience may add their own read/write functions and link them with the GMT library to extend the number of predefined formats. Technical information on this topic can be found in the source file `gmt_customio.c`. Users who are considering this approach should contact the GMT team.

ID	Explanation
	GMT 4 netCDF standard formats
nb	GMT netCDF format (8-bit integer, COARDS, CF-1.5)
ns	GMT netCDF format (16-bit integer, COARDS, CF-1.5)
ni	GMT netCDF format (32-bit integer, COARDS, CF-1.5)
nf	GMT netCDF format (32-bit float, COARDS, CF-1.5)
nd	GMT netCDF format (64-bit float, COARDS, CF-1.5)
	GMT 3 netCDF legacy formats
cb	GMT netCDF format (8-bit integer, depreciated)
cs	GMT netCDF format (16-bit integer, depreciated)
ci	GMT netCDF format (32-bit integer, depreciated)
cf	GMT netCDF format (32-bit float, depreciated)
cd	GMT netCDF format (64-bit float, depreciated)
	GMT native binary formats
bm	GMT native, C-binary format (bit-mask)
bb	GMT native, C-binary format (8-bit integer)
bs	GMT native, C-binary format (16-bit integer)
bi	GMT native, C-binary format (32-bit integer)
bf	GMT native, C-binary format (32-bit float)
bd	GMT native, C-binary format (32-bit float)
	Miscellaneous grid formats
rb	SUN raster file format (8-bit standard)
rf	GEODAS grid format GRD98 (NGDC)
sf	Golden Software Surfer format 6 (32-bit float)
sd	Golden Software Surfer format 7 (64-bit float)
af	Atlantic Geoscience Center AGC (32-bit float)
ei	ESRI Arc/Info ASCII Grid Interchange format (ASCII integer)
ef	ESRI Arc/Info ASCII Grid Interchange format (ASCII float)
gd	Import/export via GDAL [14]

Because some formats have limitations on the range of values they can store it is sometimes necessary to provide more than simply the name of the file and its ID on the command line. For instance, a native short integer file may use a unique value to signify an empty node or NaN, and the data may need translation and scaling prior to use. Therefore, all GMT programs that read or write grid files will decode the given filename as follows:

name[=*ID*[/*scale*/*offset*[/*nan*]]]

where everything in brackets is optional. If you are reading a grid then no options are needed: just continue to pass the name of the grid file. However, if you write another format you must append the =*ID* string, where *ID* is the format code listed above. In addition, should you want to (1) multiply the data by a scale factor, and (2) add a constant offset you must append the /*scale*/*offset* modifier. Finally, if you need to indicate that a certain data value should be interpreted as a NaN (not-a-number) you must append the /*nan* suffix to the scaling string (it cannot go by itself; note the nesting of the brackets!). The /*scale* and /*offset* modifiers may be left empty to select default values (scale = 1, offset = 0), or you may specify *a* for auto-adjusting the scale and/or offset of packed integer grids (=*ID*/*a* is a shorthand for =*ID*/*a*/*a*).

Some of the grid formats allow writing to standard output and reading from standard input which means you can connect GMT programs that operate on grid files with pipes, thereby speeding up execution and eliminating the need for large, intermediate grid files. You specify standard input/output by leaving out

[13] Requires building GMT with GDAL.
[14] Requires building GMT with GDAL.

the filename entirely. That means the "filename" will begin with "=*ID*". Note, that the netCDF format does not allow piping.

Everything looks clearer after a few examples:

- To write a native binary float grid file, specify the name as `my_file.f4=bf` .
- To read a native short integer grid file, multiply the data by 10 and then add 32000, but first let values that equal 32767 be set to NaN, use the filename `my_file.i2=bs/10/32000/32767`.
- To read a Golden Software "surfer" format 6 grid file, just pass the file name, e.g., `my_surferfile.grd`.
- To read a 8-bit standard Sun raster file (with values in the 0–255 range) and convert it to a 1 range, give the name as `rasterfile=rb/7.84313725e-3/-1` (i.e., 1/127.5).
- To write a native binary short integer grid file to standard output after subtracting 32000 and dividing its values by 10, give filename as `=bs/0.1/-3200`.
- To write an 8-bit integer netCDF grid file with an auto-adjusted offset, give filename as `=nb//a`.

Programs that both read and/or write more than one grid file may specify different formats and/or scaling for the files involved. The only restriction with the embedded grid specification mechanism is that no grid files may actually use the "=" character as part of their name (presumably, a small sacrifice).

One can also define special file suffixes to imply a specific file format; this approach represents a more intuitive and user-friendly way to specify the various file formats. The user may create a file called `gmt.io` in the current directory or home directory, or in the directory `~/.gmt` and define any number of custom formats. The following is an example of a `gmt.io` file:

# suffix format_id scale offset NaNxxxComments # GMT i/o shorthand file # It can have any number of comment line					
grd	nf	-	-	-	Default format
b	bf	-	-	-	Native binary floats
i2	bs	-	-	32767	2-byte integers with NaN value
ras	rb	-	-	-	Sun raster files
byte	bb	-	-	255	Native binary 1-byte grids
bit	bm	-	-	-	Native binary 0 or 1 grids
mask	bm	-	-	0	Native binary 1 or NaN masks
faa	bs	0.1	-	32767	Native binary gravity in 0.1 mGal
ns	ns	a	a	-	16-bit integer netCDF grid with auto-scale and auto-offset

These suffices can be anything that makes sense to the user. To activate this mechanism, set parameter *IO_GRIDFILE_SHORTHAND* to TRUE in your `gmt.conf` file. Then, using the filename `stuff.i2` is equivalent to saying `stuff.i2=bs///32767`, and the filename `wet.mask` means wet.mask=bm/1/0/0. For a file intended for masking, i.e., the nodes are either 1 or NaN, the bit or mask format file may be as small as 1/32 the size of the corresponding grid float format file.

8.21 Modifiers for changing the grid coordinates

A few GMT tools require that the two horizontal dimensions be specified in meters. One example is `grdfft` which must compute the 2-D Fourier transform of a grid and evaluate wavenumbers in the proper units (1/meter). There are two situations where the user may need to change the coordinates of the grid passed to such programs:

- You have a geographic grid (i.e., in longitude and latitude). Simply supply the **-fg** option and your grid coordinates will automatically be converted to meters via a "Flat Earth" approximation on the currently selected ellipsoid (Note: this is only possible in those few programs that require this capability. In general, **-fg** is used to specify table coordinates).

- You have a Cartesian grid but the units are not meters (e.g., they may perhaps be in km or miles). In this case you may append the file modifier **+u***unit*, where *unit* is one of non-arc units listed in Table *distunits*. For example, reading in the grid (which has distance units of km) and converting them to meters is done by specifying the filename as *filename***+u**k. On output, any derived grids will revert to their original units *unless* you specify another unit modifier to the output grid. This may be used, for instance, to save the original grid with distances in meters using some other unit.

For convenience, we also support the inverse translation, i.e., **+U***unit*. This modifier can be used to convert your grid coordinates *from* meters *to* the specified unit. Example 28 shows a case where this is being used to change an UTM grid in meters to km. These modifiers are only allowed when map projections are not selected (or are Cartesian).

8.22 Modifiers for COARDS-compliant netCDF files

When the netCDF grid file contains more than one 2-dimensional variable, GMT programs will load the first such variable in the file and ignore all others. Alternatively, the user can select the required variable by adding the suffix ”?*varname*” to the grid file name. For example, to get information on the variable “slp” in file , use:

```
gmt grdinfo "file.nc?slp"
```

Since COARDS-compliant netCDF files are the default, the additional suffix “=nf” can be omitted.

In case the named grid is 3-dimensional, GMT will load the first (bottom) layer. If another layer is required, either add “[*index*]” or “(*level*)”, where *index* is the index of the third (depth) variable (starting at 0 for the first layer) and *level* is the numerical value of the third (depth) variable associated with the requested layer. To indicate the second layer of the 3-D variable “slp” use as file name: `file.nc?slp[1]`.

When you supply the numerical value for the third variable using “(*level*)”, GMT will pick the layer closest to that value. No interpolation is performed.

Note that the question mark, brackets and parentheses have special meanings on Unix-based platforms. Therefore, you will need to either *escape* these characters, by placing a backslash in front of them, or place the whole file name plus modifiers between single quotes or double quotes.

A similar approach is followed for loading 4-dimensional grids. Consider a 4-dimensional grid with the following variables:

```
lat(lat): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
lon(lon): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
depth(depth): 0, 10, 20, 30, 40, 50, 60, 70, 80, 90
time(time): 0, 12, 24, 36, 48
pressure(time,depth,lat,lon): (5000 values)
```

To get information on the 10x10 grid of pressure at depth 10 and at time 24, one would use:

```
gmt grdinfo "file.nc?pressure[2,1]"
```

or (only in case the coordinates increase linearly):

```
gmt grdinfo "file.nc?pressure(24,10)"
```

Programs that generally deal with columns of one-dimensional data, like or can use multi-dimensional netCDF files in a very similar way. If a variable in a netCDF file is one-dimensional, there is nothing more needed than name the variables on the command line. For example:

```
gmt psxy "file.nc?lon/lat" ...
gmt gmtconvert "file.nc?time/lat/lon"
```

If one or more of the selected variables are two-dimensional, and have the same leading dimension as the other selected variables they will be plotted in their entirety. For example, if a netCDF files contains 6 time steps recording temperature at 4 points, and the variable `tmp` is a 6 by 4 array, then the command `gmtconvert "file.nc?time/temp"` can result in:

```
2012-06-25T00:00:00 20.1 20.2 20.1 20.3
2012-06-25T12:00:00 24.2 23.2 24.5 23.5
2012-06-26T00:00:00 16.1 16.2 16.1 16.3
2012-06-26T12:00:00 22.1 23.0 23.9 23.5
2012-06-27T00:00:00 17.5 16.9 17.2 16.8
2012-06-27T12:00:00 27.2 27.2 27.5 27.5
```

If, for example, only the second temperature column is needed, use `gmt gmtconvert "file.nc?time/temp"` (indices start counting at 0).

The COARDS conventions set restrictions on the names that can be used for the units of the variables and coordinates. For example, the units of longitude and latitude are "degrees_east" and "degrees_north", respectively. Here is an example of the header of a COARDS compliant netCDF file (to be obtained using **ncdump**):

```
netcdf M2_fes2004 {
dimensions:
        lon = 2881 ;
        lat = 1441 ;
variables:
        float lon(lon) ;
                lon:long_name = "longitude" ;
                lon:units = "degrees_east" ;
                lon:actual_range = 0., 360. ;
        float lat(lat) ;
                lat:long_name = "latitude" ;
                lat:units = "degrees_north" ;
                lat:actual_range = -90., 90. ;
        short amp(lat, lon) ;
                amp:long_name = "amplitude" ;
                amp:unit = "m" ;
                amp:scale_factor = 0.0001 ;
                amp:add_offset = 3. ;
                amp:_FillValue = -32768s ;
        short pha(lat, lon) ;
                pha:long_name = "phase" ;
                pha:unit = "degrees" ;
                pha:scale_factor = 0.01 ;
                pha:_FillValue = -32768s ;
```

This file contains two grids, which can be plotted separately using the names `M2_fes2004.nc?amp` and `M2_fes2004.nc?pha`. The attributes `long_name` and `unit` for each variable are combined in GMT to a single unit string. For example, after reading the grid `y_unit` equals `latitude [degrees_north]`. The same method can be used in reverse to set the proper variable names and units when writing a grid. However, when the coordinates are set properly as geographical or time axes, GMT will take care of this. The user is, however, still responsible for setting the variable name and unit of the z-coordinate. The default is simply "z".

8.23 Modifiers to read and write grids and images via GDAL

If the support has been configured during installation, then GMT can read and write a variety of grid and image formats via GDAL. This extends the capability of GMT to handle data sets from a variety of sources.

8.23.1 Reading multi-band images

`grdimage` and `psimage` both lets the user select individual bands in a multi-band image file and treats the result as an image (that is the values, in the 0–255 range, are treated as colors, not data). To select individual bands you use the **+b***band-number* mechanism that must be appended to the image filename. Here, *band-number* can be the number of one individual band (the counting starts at zero), or it could be a comma-separated list of bands. For example

```
gmt psimage jpeg_image_with_three_bands.jpg+b0
```

will plot only the first band (i.e., the red band) of the jpeg image as a gray-scale image, and

```
gmt psimage jpeg_image_with_three_bands.jpg+b2,1,0
```

will plot the same image in color but where the RGB band order has been reversed.

Instead of treating them as images, all other GMT programs that process grids can read individual bands from an image but will consider the values to be regular data. For example, let `multiband` be the name of a multi-band file with a near infrared component in band 4 and red in band 3. We will compute the NDVI (Normalized Difference Vegetation Index), which is defined as NDVI = (NIR - R) / (NIR + R), as

```
gmt grdmath multiband=gd+b3 multiband=gd+b2 SUB multiband=gd+b3 \
            multiband=gd+b2 ADD DIV = ndvi.nc
```

The resulting grid `ndvi.nc` can then be plotted as usual.

8.23.2 Reading more complex multi-band IMAGES or GRIDS

It is also possible to access to sub-datasets in a multi-band grid. The next example shows how we can extract the SST from the MODIS file `A20030012003365.L3m_YR_NSST_9` that is stored in the HDF "format". We need to run the GDAL program **gdalinfo** on the file because we first must extract the necessary metadata from the file:

```
gdalinfo A20030012003365.L3m_YR_NSST_9
Driver: HDF4/Hierarchical Data Format Release 4
Files: A20030012003365.L3m_YR_NSST_9
Size is 512, 512
Coordinate System is `'
Metadata:
 Product Name=A20030012003365.L3m_YR_NSST_9
 Sensor Name=MODISA
 Sensor=
 Title=MODISA Level-3 Standard Mapped Image
...
 Scaling=linear
 Scaling Equation=(Slope*l3m_data) + Intercept = Parameter value
 Slope=0.000717185
 Intercept=-2
 Scaled Data Minimum=-2
 Scaled Data Maximum=45
 Data Minimum=-1.999999
 Data Maximum=34.76
Subdatasets:
```

```
SUBDATASET_1_NAME=HDF4_SDS:UNKNOWN:"A20030012003365.L3m_YR_NSST_9":0
SUBDATASET_1_DESC=[2160x4320] l3m_data (16-bit unsigned integer)
SUBDATASET_2_NAME=HDF4_SDS:UNKNOWN:"A20030012003365.L3m_YR_NSST_9":1
SUBDATASET_2_DESC=[2160x4320] l3m_qual (8-bit unsigned integer)
```

Now, to access this file with GMT we need to use the =gd mechanism and append the name of the sub-dataset that we want to extract. Here, a simple example using `grdinfo` would be

```
grdinfo A20030012003365.L3m_YR_NSST_9=gd?HDF4_SDS:UNKNOWN:"A20030012003365.L3m_YR_NSST_9:0"

HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: Title: Grid imported via GDAL
HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: Command:
HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: Remark:
HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: Gridline node registration used
HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: Grid file format: gd = Import through GDAL (convert tc
HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: x_min: 0.5 x_max: 4319.5 x_inc: 1 name: x nx: 4320
HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: y_min: 0.5 y_max: 2159.5 y_inc: 1 name: y ny: 2160
HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: z_min: 0 z_max: 65535 name: z
HDF4_SDS:UNKNOWN:A20030012003365.L3m_YR_NSST_9:0: scale_factor: 1 add_offset: 0
```

Be warned, however, that things are not yet completed because while the data are scaled according to the equation printed above ("Scaling Equation=(Slope*l3m_data) + Intercept = Parameter value"), this scaling is not applied by GDAL on reading so it cannot be done automatically by GMT. One solution is to do the reading and scaling via `grdmath` first, i.e.,

```
gmt grdmath A20030012003365.L3m_YR_NSST_9=gd?HDF4_SDS:UNKNOWN:"A20030012003365.L3m_YR_NSST_9:0" \
            0.000717185 MUL -2 ADD = sst.nc
```

then plot the `sst.nc` directly.

8.23.3 Writing grids and images

Saving images in the common raster formats is possible but, for the time being, only from `grdimage` and even that is restricted to raster type information. That is, vector data (for instance, coast lines) or text will not be saved. To save an image with `grdimage` use the **-A***outimg*=*driver* mechanism, where *driver* is the driver code name used by GDAL (e.g. GTiff).

For all other programs that create grids, it is also possible to save them using GDAL. To do it one need to use the =gd appended with the necessary information regarding the driver and the data type to use. Generically, =**gd**[/*scale*/*offset*[/*nan*]][:<*driver*>[/*dataType*]] where *driver* is the same as explained above and *dataType* is a 2 or 3 chars code from: u8|u16|i16|u32|i32|float32, and where i|u denotes signed|unsigned. If not provided the default type is float32. Both driver names and data types are case insensitive. Note, that you have to specify *nan* for integer types unless you whish that all NaN data values are replaced by zero.

8.24 The NaN data value

For a variety of data processing and plotting tasks there is a need to acknowledge that a data point is missing or unassigned. In the "old days" such information was passed by letting a value like -9999.99 take on the special meaning of "this is not really a value, it is missing". The problem with this scheme is that -9999.99 (or any other floating point value) may be a perfectly reasonable data value and in such a scenario would be skipped. The solution adopted in GMT is to use the IEEE concept Not-a-Number (NaN) for this purpose. Mathematically, a NaN is what you get if you do an undefined mathematical operation like 0/0; in ASCII data files they appear as the textstring NaN. This value is internally stored with a particular bit pattern defined by IEEE so that special action can be taken when it is encountered by programs. In particular, a standard library function called `isnan` is used to test if a floating point is a

NaN. GMT uses these tests extensively to determine if a value is suitable for plotting or processing (if a NaN is used in a calculation the result would become NaN as well). Data points whose values equal NaN are not normally plotted (or plotted with the special NaN color given in `gmt.conf`). Several tools such as `xyz2grd`, `gmtmath`, and `grdmath` can convert user data to NaN and vice versa, thus facilitating arbitrary masking and clipping of data sets. Note that a few computers do not have native IEEE hardware support. At this point, this applies to some of the older Cray super-computers. Users on such machines may have to adopt the old '-9999.99' scheme to achieve the desired results.

Data records that contain NaN values for the *x* or *y* columns (or the *z* column for cases when 3-D Cartesian data are expected) are usually skipped during reading. However, the presence of these bad records can be interpreted in two different ways, and this behavior is controlled by the *IO_NAN_RECORDS* defaults parameter. The default setting (*gap*) considers such records to indicate a gap in an otherwise continuous series of points (e.g., a line), and programs can act upon this information, e.g., not to draw a line across the gap or to break the line into separate segments. The alternative setting (*bad*) makes no such interpretation and simply reports back how many bad records were skipped during reading; see Section Data gap detection: The -g option for details.

8.25 Directory parameters

GMT versions prior to GMT 5 relied on several environment variables ($GMT_SHAREDIR, $GMT_DATADIR, $GMT_USERDIR, and $GMT_TMPDIR), pointing to folders with data files and program settings. Beginning with version 5, these locations are configurable with the `gmtset` utility. The environment variables are still supported but are overridden by the *directory parameters* DIR_DATA, DIR_USER, and DIR_TMP in `gmt.conf`.

Variable $GMT_SHAREDIR was sometimes required in previous GMT versions to locate the GMT share directory where all run-time support files such as coastlines, custom symbols, PostScript macros, color tables, and much more reside. If this parameter is not set (default), GMT will make a reasonable guess of the location of its share folder. Setting this variable is usually not required and recommended only under special circumstances.

Variable $GMT_DATADIR and parameter DIR_DATA may point to one or more directories where large and/or widely used data files can be placed. All GMT programs look in these directories when a file is specified on the command line and it is not present in the current directory. This allows maintainers to consolidate large data files and to simplify scripting that use these files since the absolute path need not be specified. Separate multiple directories with colons (:) – under Windows use semi-colons (;). Any directory name that ends in a trailing slash (/) will be searched recursively (not under Windows).

Variable $GMT_USERDIR and parameter DIR_USER may point to a directory where the user places custom configuration files (e.g., an alternate `coastline.conf` file, preferred default settings in `gmt.conf`, custom symbols and color palettes, math macros for `gmtmath` and `grdmath`, and shorthands for gridfile extensions via `gmt_io`). Users may also place their own data files in this directory as GMT programs will search for files given on the command line in both DIR_DATA and DIR_USER.

Variable $GMT_TMPDIR and parameter DIR_TMP may indicate the location, where GMT will write its state parameters via the two files `gmt.history` and `gmt.conf`. If DIR_TMP is not set, these files are written to the current directory. See Section Isolation mode for more information.

Note that files whose full path is given will never be searched for in any of these directories.

CHAPTER 9

GMT Coordinate Transformations

GMT programs read real-world coordinates and convert them to positions on a plot. This is achieved by selecting one of several coordinate transformations or projections. We distinguish between three sets of such conversions:

- Cartesian coordinate transformations
- Polar coordinate transformations
- Map coordinate transformations

The next chapter will be dedicated to GMT map projections in its entirety. Meanwhile, the present chapter will summarize the properties of the Cartesian and Polar coordinate transformations available in GMT, list which parameters define them, and demonstrate how they are used to create simple plot axes. We will mostly be using `psbasemap` (and occasionally `psxy`) to demonstrate the various transformations. Our illustrations may differ from those you reproduce with the same commands because of different settings in our `gmt.conf` file.) Finally, note that while we will specify dimensions in inches (by appending **i**), you may want to use cm (**c**), or points (**p**) as unit instead (see the `gmt.conf` man page).

9.1 Cartesian transformations

GMT Cartesian coordinate transformations come in three flavors:

- Linear coordinate transformation
- Log_{10} coordinate transformation
- Power (exponential) coordinate transformation

These transformations convert input coordinates *(x,y)* to locations *(x', y')* on a plot. There is no coupling between *x* and *y* (i.e., *x' = f(x)* and *y' = f(y)*); it is a **one-dimensional** projection. Hence, we may use separate transformations for the *x*- and *y*-axes (and *z*-axes for 3-D plots). Below, we will use the expression *u' = f(u)*, where *u* is either *x* or *y* (or *z* for 3-D plots). The coefficients in *f(u)* depend on the desired plot size (or scale), the chosen *(x,y)* domain, and the nature of *f* itself.

Two subsets of linear will be discussed separately; these are a polar (cylindrical) projection and a linear projection applied to geographic coordinates (with a 360 periodicity in the *x*-coordinate). We will show examples of all of these projections using dummy data sets created with `gmtmath`, a "Reverse Polish Notation" (RPN) calculator that operates on or creates table data:

```
gmt gmtmath -T0/100/1  T SQRT = sqrt.d
gmt gmtmath -T0/100/10 T SQRT = sqrt.d10
```

9.1.1 Cartesian linear transformation (-Jx -JX)

There are in fact three different uses of the Cartesian linear transformation, each associated with specific command line options. The different manifestations result from specific properties of three kinds of data:

1. Regular floating point coordinates
2. Geographic coordinates
3. Calendar time coordinates

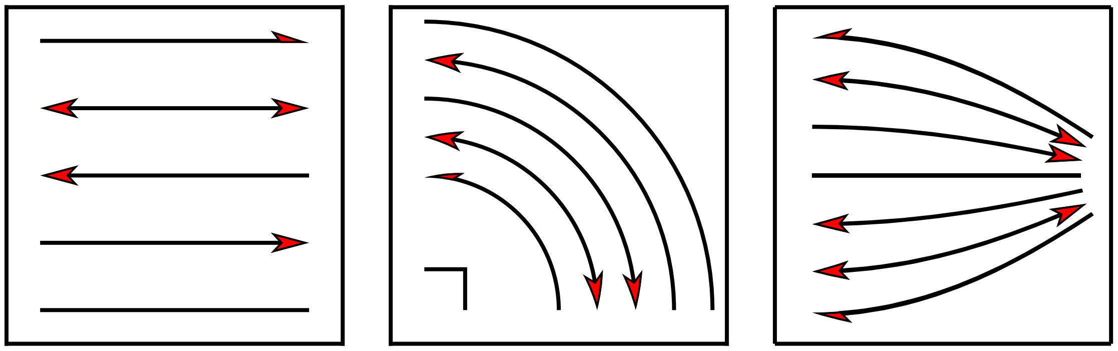


Figure 9.1: Examples of Cartesian (left), circular (middle), and geo-vectors (right) for different attribute specifications. Note that both full and half arrow-heads can be specified, as well as no head at all.

Regular floating point coordinates

Selection of the Cartesian linear transformation with regular floating point coordinates will result in a simple linear scaling *u’* = *au* + *b* of the input coordinates. The projection is defined by stating scale in inches/unit (**-Jx**) or axis length in inches (**-JX**). If the *y*-scale or *y*-axis length is different from that of the *x*-axis (which is most often the case), separate the two scales (or lengths) by a slash, e.g., **-Jx**0.1i/0.5i or **-JX**8i/5i. Thus, our $y = \sqrt{x}$ data sets will plot as shown in Figure *Linear transformation of Cartesian coordinates*.

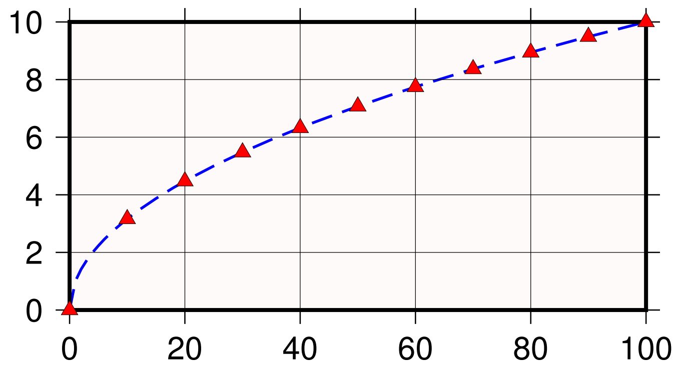


Figure 9.2: Linear transformation of Cartesian coordinates.

The complete commands given to produce this plot were

```
gmt psxy -R0/100/0/10 -JX3i/1.5i -Bag -BWSne+gsnow -Wthick,blue,- -P -K sqrt.d > GMT_linear.ps
gmt psxy -R -J -St0.1i -N -Gred -Wfaint -O sqrt.d10 >> GMT_linear.ps
```

Normally, the user's *x*-values will increase to the right and the *y*-values will increase upwards. It should be noted that in many situations it is desirable to have the direction of positive coordinates be reversed. For example, when plotting depth on the *y*-axis it makes more sense to have the positive direction downwards. All that is required to reverse the sense of positive direction is to supply a negative scale (or axis length). Finally, sometimes it is convenient to specify the width (or height) of a map and let the other dimension be computed based on the implied scale and the range of the other axis. To do this, simply specify the length to be recomputed as 0.

Geographic coordinates

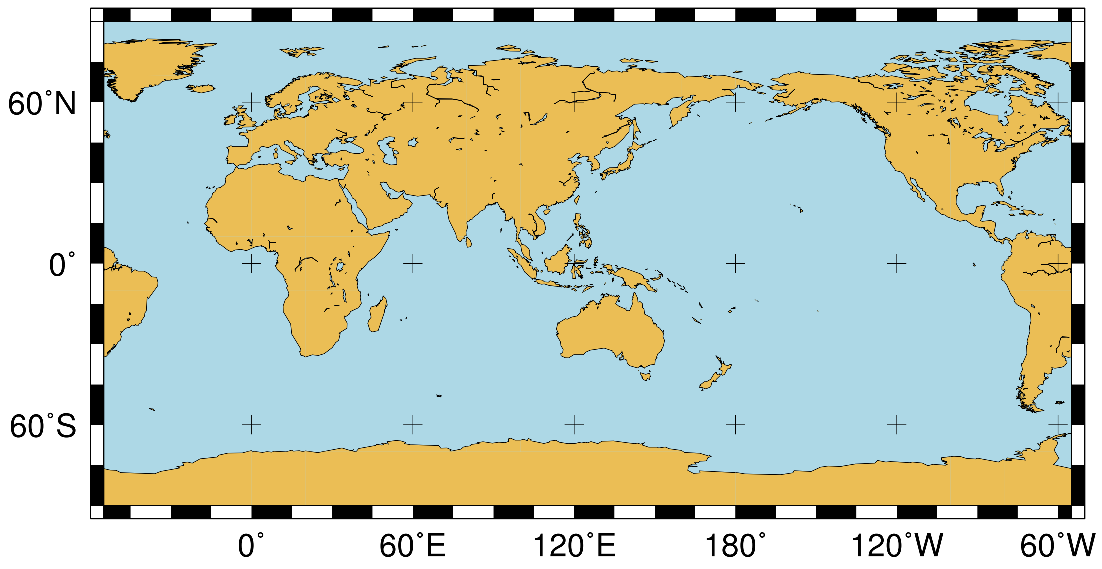


Figure 9.3: Linear transformation of map coordinates.

While the Cartesian linear projection is primarily designed for regular floating point *x*,*y* data, it is sometimes necessary to plot geographical data in a linear projection. This poses a problem since longitudes have a 360 periodicity. GMT therefore needs to be informed that it has been given geographical coordinates even though a linear transformation has been chosen. We do so by adding a **g** (for geographical) or **d** (for degrees) directly after **-R** or by appending a **g** or **d** to the end of the **-Jx** (or **-JX**) option. As an example, we want to plot a crude world map centered on 125E. Our command will be

```
gmt set MAP_GRID_CROSS_SIZE_PRIMARY 0.1i MAP_FRAME_TYPE FANCY FORMAT_GEO_MAP ddd:mm:ssF
gmt pscoast -Rg-55/305/-90/90 -Jx0.014i -Bagf -BWSen -Dc -A1000 -Glightbrown -Wthinnest \
        -P -Slightblue > GMT_linear_d.ps
```

with the result reproduced in Figure *Linear transformation of map coordinates*.

Calendar time coordinates

Several particular issues arise when we seek to make linear plots using calendar date/time as the input coordinates. As far as setting up the coordinate transformation we must indicate whether our input data have absolute time coordinates or relative time coordinates. For the former we append **T** after the axis scale (or width), while for the latter we append **t** at the end of the **-Jx** (or **-JX**) option. However, other command line arguments (like the **-R** option) may already specify whether the time coordinate

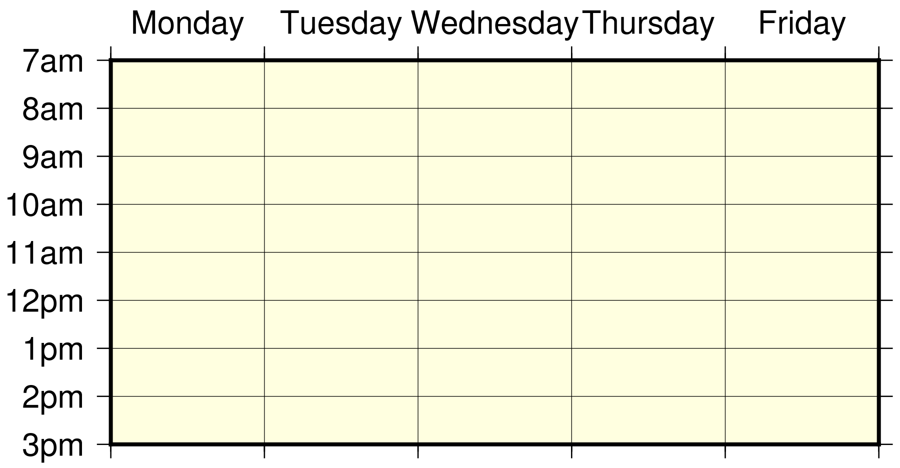


Figure 9.4: Linear transformation of calendar coordinates.

is absolute or relative. An absolute time entry must be given as [*date*]**T**[*clock*] (with *date* given as *yyyy*[-*mm*[-*dd*]], *yyyy*[-*jjj*], or *yyyy*[-**W***ww*[-*d*]], and *clock* using the *hh*[:*mm*[:*ss*[.*xxx*]]] 24-hour clock format) whereas the relative time is simply given as the units of time since the epoch followed by **t** (see *TIME_UNIT* and *TIME_EPOCH* for information on specifying the time unit and the epoch). As a simple example, we will make a plot of a school week calendar (Figure *Linear transformation of calendar coordinates*).

When the coordinate ranges provided by the **-R** option and the projection type given by **-JX** (including the optional **d**, **g**, **t** or **T**) conflict, GMT will warn the users about it. In general, the options provided with **-JX** will prevail.

```
gmt set FORMAT_DATE_MAP o TIME_WEEK_START Sunday FORMAT_CLOCK_MAP=-hham \
        FORMAT_TIME_PRIMARY_MAP full
gmt psbasemap -R2001-9-24T/2001-9-29T/T07:0/T15:0 -JX4i/-2i -Bxa1Kf1kg1d \
              -Bya1Hg1h -BWsNe+glightyellow -P > GMT_linear_cal.ps
```

9.1.2 Cartesian logarithmic projection

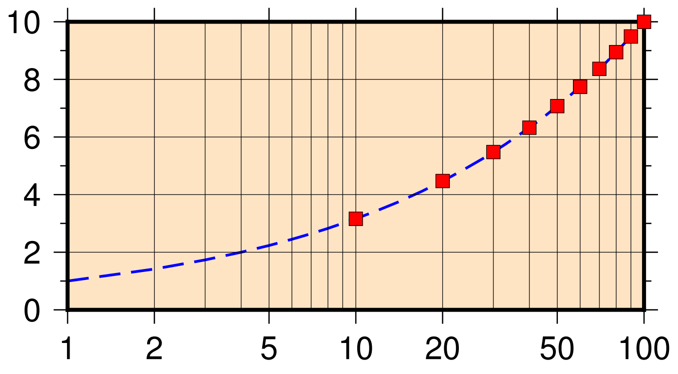


Figure 9.5: Logarithmic transformation of x–coordinates.

The $\log_{10}$ transformation is simply $u' = a \log_{10}(u) + b$ and is selected by appending an **l** (lower case L) immediately following the scale (or axis length) value. Hence, to produce a plot in which the *x*-axis is logarithmic (the *y*-axis remains linear, i.e., a semi-log plot), try (Figure *Logarithmic transformation*)

```
gmt psxy -R1/100/0/10 -Jx1.5il/0.15i -Bx2g3 -Bya2f1g2 -BWSne+gbisque \
         -Wthick,blue,- -P -K -h sqrt.d > GMT_log.ps
gmt psxy -R -J -Ss0.1i -N -Gred -W -O -h sqrt.d10 >> GMT_log.ps
```

Note that if x- and y-scaling are different and a $\log_{10} - \log_{10}$ plot is desired, the **l** must be appended twice: Once after the x-scale (before the /) and once after the y-scale.

9.1.3 Cartesian power projection

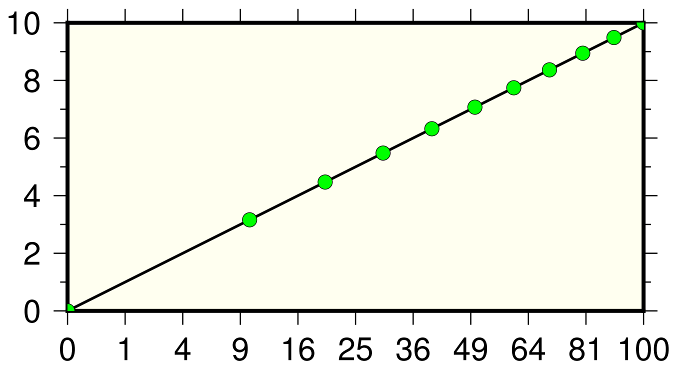


Figure 9.6: Exponential or power transformation of x–coordinates.

This projection uses $u' = au^b + c$ and allows us to explore exponential relationships like x^p versus y^q. While p and q can be any values, we will select *p = 0.5* and *q = 1* which means we will plot x versus $\sqrt{x}$. We indicate this scaling by appending a **p** (lower case P) followed by the desired exponent, in our case 0.5. Since *q = 1* we do not need to specify **p**1 since it is identical to the linear transformation. Thus our command becomes (Figure *Power transformation*)

```
gmt psxy -R0/100/0/10 -Jx0.3ip0.5/0.15i -Bxa1p -Bya2f1 -BWSne+givory \
         -Wthick -P -K sqrt.d > GMT_pow.ps
gmt psxy -R -J -Sc0.075i -Ggreen -W -O sqrt.d10 >> GMT_pow.ps
```

9.2 Linear projection with polar coordinates (-Jp -JP)

This transformation converts polar coordinates (angle θ and radius r) to positions on a plot. Now $x' = f(\theta, r)$ and $y' = g(\theta, r)$, hence it is similar to a regular map projection because x and y are coupled and x (i.e., θ) has a 360 periodicity. With input and output points both in the plane it is a **two-dimensional** projection. The transformation comes in two flavors:

1. Normally, θ is understood to be directions counter-clockwise from the horizontal axis, but we may choose to specify an angular offset [whose default value is zero]. We will call this offset θ_0. Then, $x' = f(\theta, r) = ar\cos(\theta - \theta_0) + b$ and $y' = g(\theta, r) = ar\sin(\theta - \theta_0) + c$.
2. Alternatively, θ can be interpreted to be azimuths clockwise from the vertical axis, yet we may again choose to specify the angular offset [whose default value is zero]. Then, $x' = f(\theta, r) = ar\cos(90 - (\theta - \theta_0)) + b$ and $y' = g(\theta, r) = ar\sin(90 - (\theta - \theta_0)) + c$.

Consequently, the polar transformation is defined by providing

- scale in inches/unit (**-Jp**) or full width of plot in inches (**-JP**)
- Optionally, insert **a** after **p| P** to indicate CW azimuths rather than CCW directions
- Optionally, append /*origin* in degrees to indicate an angular offset [0]
- Optionally, append **r** to reverse the radial direction (here, *south* and *north* must be elevations in 0–90 range).

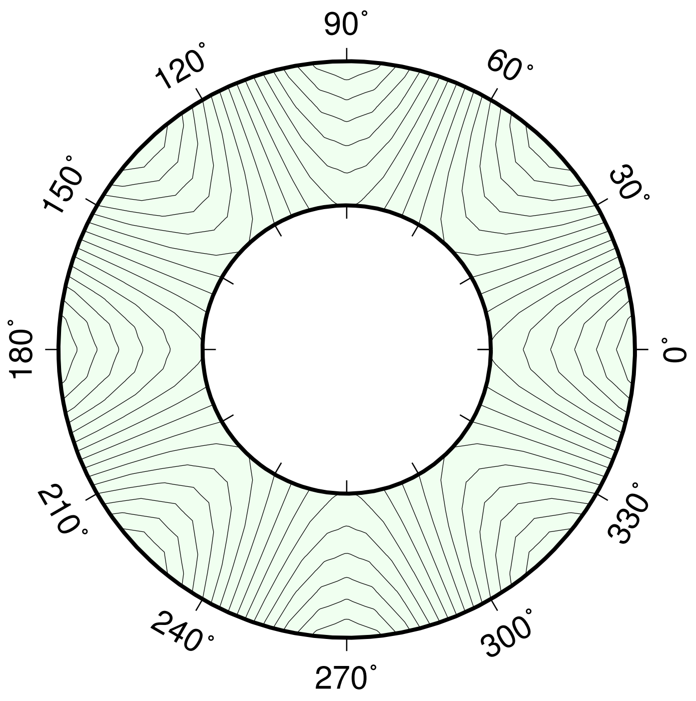


Figure 9.7: Polar (Cylindrical) transformation of (θ, r) coordinates.

- Optionally, append **z** to annotate depths rather than radius.

As an example of this projection we will create a gridded data set in polar coordinates $z(\theta, r) = r^2 \cdot \cos 4\theta$ using `grdmath`, a RPN calculator that operates on or creates grid files.

```
gmt grdmath -R0/360/2/4 -I6/0.1 X 4 MUL PI MUL 180 DIV COS Y 2 POW MUL = tt.nc
gmt grdcontour tt.nc -JP3i -B30 -BNs+ghoneydew -P -C2 -S4 --FORMAT_GEO_MAP=+ddd > GMT_polar.ps
```

We used `grdcontour` to make a contour map of this data. Because the data file only contains values with $2 \leq r \leq 4$, a donut shaped plot appears in Figure *Polar transformation*.

CHAPTER 10

GMT Map Projections

GMT implements more than 30 different projections. They all project the input coordinates longitude and latitude to positions on a map. In general, *x' = f(x,y,z)* and *y' = g(x,y,z)*, where *z* is implicitly given as the radial vector length to the *(x,y)* point on the chosen ellipsoid. The functions *f* and *g* can be quite nasty and we will refrain from presenting details in this document. The interested read is referred to *Snyder* [1987] [1]. We will mostly be using the `pscoast` command to demonstrate each of the projections. GMT map projections are grouped into four categories depending on the nature of the projection. The groups are

1. Conic map projections
2. Azimuthal map projections
3. Cylindrical map projections
4. Miscellaneous projections

Because *x* and *y* are coupled we can only specify one plot-dimensional scale, typically a map *scale* (for lower-case map projection code) or a map *width* (for upper-case map projection code). However, in some cases it would be more practical to specify map *height* instead of *width*, while in other situations it would be nice to set either the *shortest* or *longest* map dimension. Users may select these alternatives by appending a character code to their map dimension. To specify map *height*, append **h** to the given dimension; to select the minimum map dimension, append **-**, whereas you may append **+** to select the maximum map dimension. Without the modifier the map width is selected by default.

In GMT version 4.3.0 we noticed we ran out of the alphabet for 1-letter (and sometimes 2-letter) projection codes. To allow more flexibility, and to make it easier to remember the codes, we implemented the option to use the abbreviations used by the **Proj4** mapping package. Since some of the GMT projections are not in **Proj4**, we invented some of our own as well. For a full list of both the old 1- and 2-letter codes, as well as the **Proj4**-equivalents see the quick reference cards in Section GMT quick reference. For example, **-JM**15c and **-JMerc**/15c have the same meaning.

10.1 Conic projections

10.1.1 Albers conic equal-area projection (-Jb -JB)

This projection, developed by Albers in 1805, is predominantly used to map regions of large east-west extent, in particular the United States. It is a conic, equal-area projection, in which parallels are unequally spaced arcs of concentric circles, more closely spaced at the north and south edges of the map. Meridians,

[1] Snyder, J. P., 1987, Map Projections A Working Manual, U.S. Geological Survey Prof. Paper 1395.

on the other hand, are equally spaced radii about a common center, and cut the parallels at right angles. Distortion in scale and shape vanishes along the two standard parallels. Between them, the scale along parallels is too small; beyond them it is too large. The opposite is true for the scale along meridians. To define the projection in GMT you need to provide the following information:

- Longitude and latitude of the projection center.
- Two standard parallels.
- Map scale in inch/degree or 1:xxxxx notation (**-Jb**), or map width (**-JB**).

Note that you must include the "1:" if you choose to specify the scale that way. E.g., you can say 0.5 which means 0.5 inch/degree or 1:200000 which means 1 inch on the map equals 200,000 inches along the standard parallels. The projection center defines the origin of the rectangular map coordinates. As an example we will make a map of the region near Taiwan. We choose the center of the projection to be at 125 E/20 N and 25 N and 45 N as our two standard parallels. We desire a map that is 5 inches wide. The complete command needed to generate the map below is therefore given by:

```
gmt set MAP_GRID_CROSS_SIZE_PRIMARY 0
gmt pscoast -R110/140/20/35 -JB125/20/25/45/5i -Bag -Dl -Ggreen -Wthinnest \
            -A250 -P > GMT_albers.ps
```

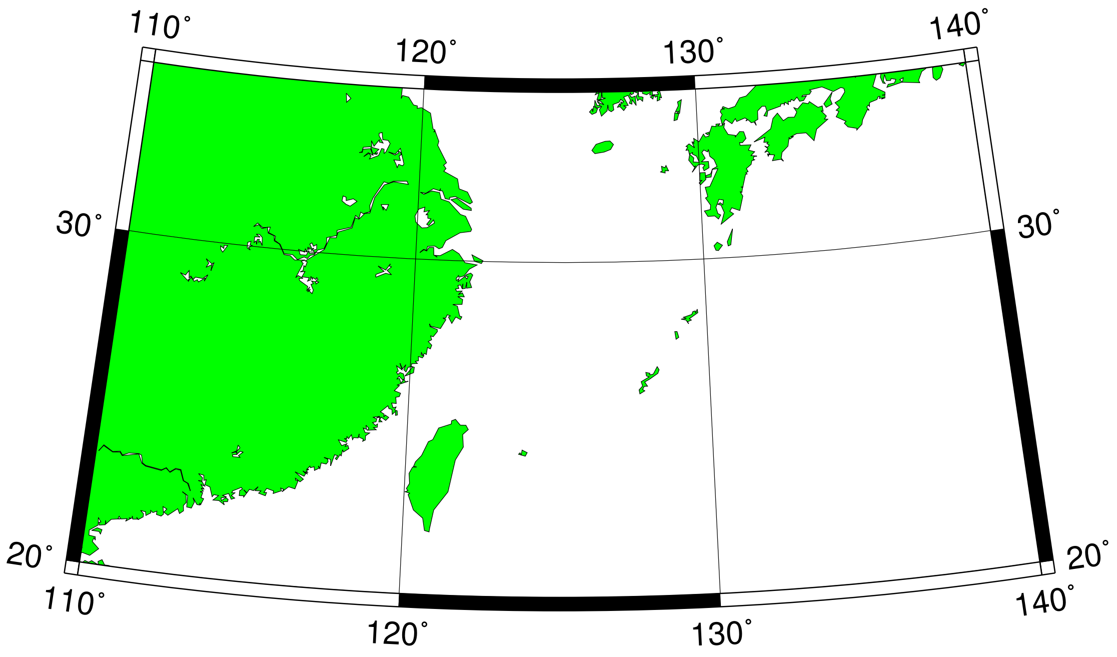


Figure 10.1: Albers equal-area conic map projection.

10.1.2 Equidistant conic projection (-Jd -JD)

The equidistant conic projection was described by the Greek philosopher Claudius Ptolemy about A.D. 150. It is neither conformal or equal-area, but serves as a compromise between them. The scale is true along all meridians and the standard parallels. To select this projection in GMT you must provide the same information as for the other conic projection, i.e.,

- Longitude and latitude of the projection center.
- Two standard parallels.
- Map scale in inch/degree or 1:xxxxx notation (**-Jd**), or map width (**-JD**).

The equidistant conic projection is often used for atlases with maps of small countries. As an example, we generate a map of Cuba:

```
gmt set FORMAT_GEO_MAP ddd:mm:ssF MAP_GRID_CROSS_SIZE_PRIMARY 0.05i
gmt pscoast -R-88/-70/18/24 -JD-79/21/19/23/4.5i -Bag -Di -N1/thick,red \
            -Glightgreen -Wthinnest -P > GMT_equidistant_conic.ps
```

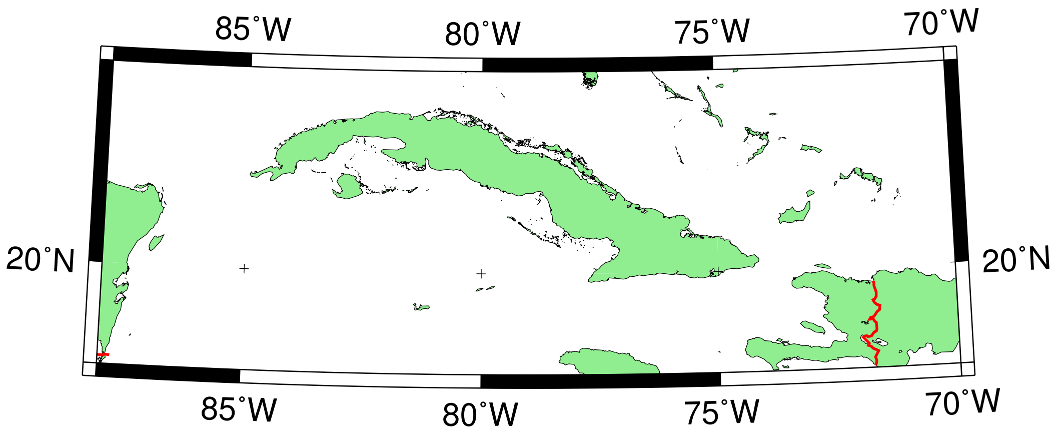


Figure 10.2: Equidistant conic map projection.

10.1.3 Lambert conic conformal projection (-Jl -JL)

This conic projection was designed by the Alsatian mathematician Johann Heinrich Lambert (1772) and has been used extensively for mapping of regions with predominantly east-west orientation, just like the Albers projection. Unlike the Albers projection, Lambert's conformal projection is not equal-area. The parallels are arcs of circles with a common origin, and meridians are the equally spaced radii of these circles. As with Albers projection, it is only the two standard parallels that are distortion-free. To select this projection in GMT you must provide the same information as for the Albers projection, i.e.,

- Longitude and latitude of the projection center.
- Two standard parallels.
- Map scale in inch/degree or 1:xxxxx notation (**-Jl**), or map width (**-JL**).

The Lambert conformal projection has been used for basemaps for all the 48 contiguous States with the two fixed standard parallels 33N and 45N. We will generate a map of the continental USA using these parameters. Note that with all the projections you have the option of selecting a rectangular border rather than one defined by meridians and parallels. Here, we choose the regular WESN region, a "fancy" basemap frame, and use degrees west for longitudes. The generating commands used were

```
gmt set MAP_FRAME_TYPE FANCY FORMAT_GEO_MAP ddd:mm:ssF MAP_GRID_CROSS_SIZE_PRIMARY 0.05i
gmt pscoast -R-130/-70/24/52 -Jl-100/35/33/45/1:50000000 -Bag -Dl -N1/thick,red \
            -N2/thinner -A500 -Gtan -Wthinnest,white -Sblue -P > GMT_lambert_conic.ps
```

The choice for projection center does not affect the projection but it indicates which meridian (here 100W) will be vertical on the map. The standard parallels were originally selected by Adams to provide a maximum scale error between latitudes 30.5N and 47.5N of 0.5–1%. Some areas, like Florida, experience scale errors of up to 2.5%.

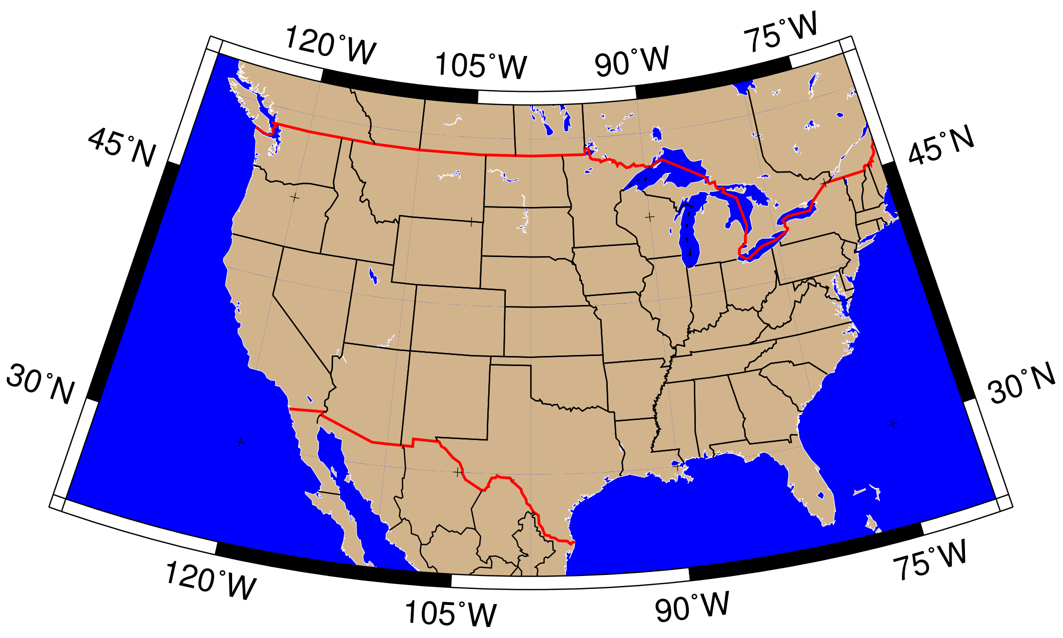


Figure 10.3: Lambert conformal conic map projection.

10.1.4 (American) polyconic projection (-Jpoly -JPoly)

The polyconic projection, in Europe usually referred to as the American polyconic projection, was introduced shortly before 1820 by the Swiss-American cartographer Ferdinand Rodulph Hassler (1770–1843). As head of the Survey of the Coast, he was looking for a projection that would give the least distortion for mapping the coast of the United States. The projection acquired its name from the construction of each parallel, which is achieved by projecting the parallel onto the cone while it is rolled around the globe, along the central meridian, tangent to that parallel. As a consequence, the projection involves many cones rather than a single one used in regular conic projections.

The polyconic projection is neither equal-area, nor conformal. It is true to scale without distortion along the central meridian. Each parallel is true to scale as well, but the meridians are not as they get further away from the central meridian. As a consequence, no parallel is standard because conformity is lost with the lengthening of the meridians.

Below we reproduce the illustration by *Snyder* [1987], with a gridline every 10 and annotations only every 30 in longitude:

```
gmt pscoast -R-180/-20/0/90 -JPoly/4i -Bx30g10 -By10g10 -Dc -A1000 -Glightgray \
            -Wthinnest -P > GMT_polyconic.ps
```

10.2 Azimuthal projections

10.2.1 Lambert Azimuthal Equal-Area (-Ja -JA)

This projection was developed by Lambert in 1772 and is typically used for mapping large regions like continents and hemispheres. It is an azimuthal, equal-area projection, but is not perspective. Distortion is zero at the center of the projection, and increases radially away from this point. To define this projection in GMT you must provide the following information:

- Longitude and latitude of the projection center.

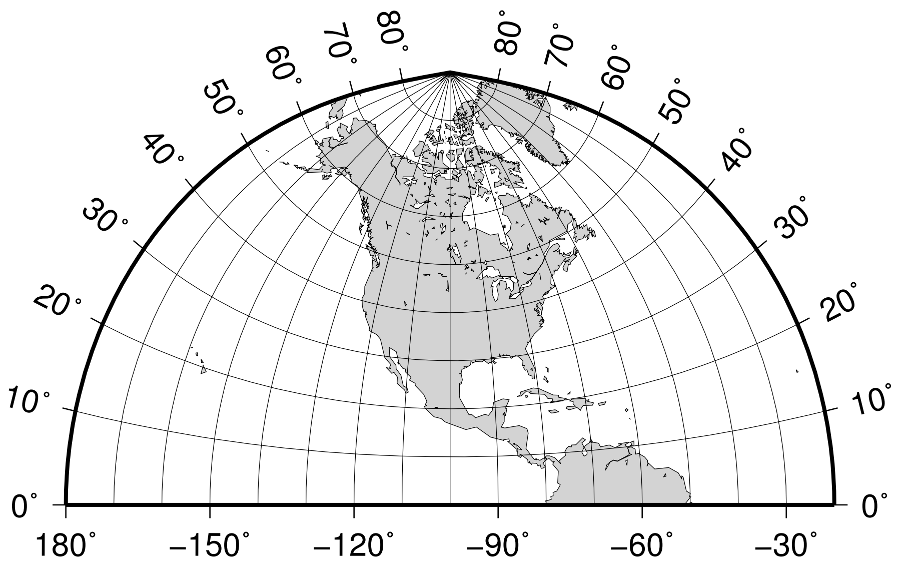


Figure 10.4: (American) polyconic projection.

- Optionally, the horizon, i.e., the number of degrees from the center to the edge (<= 180, default is 90).
- Scale as 1:xxxxx or as radius/latitude where radius is the projected distance on the map from projection center to an oblique latitude (**-Ja**), or map width (**-JA**).

Two different types of maps can be made with this projection depending on how the region is specified. We will give examples of both types.

Rectangular map

In this mode we define our region by specifying the longitude/latitude of the lower left and upper right corners instead of the usual *west, east, south, north* boundaries. The reason for specifying our area this way is that for this and many other projections, lines of equal longitude and latitude are not straight lines and are thus poor choices for map boundaries. Instead we require that the map boundaries be rectangular by defining the corners of a rectangular map boundary. Using 0E/40S (lower left) and 60E/10S (upper right) as our corners we try

```
gmt set FORMAT_GEO_MAP ddd:mm:ssF MAP_GRID_CROSS_SIZE_PRIMARY 0
gmt pscoast -R0/-40/60/-10r -JA30/-30/4.5i -Bag -Dl -A500 -Gp300/10 \
            -Wthinnest -P > GMT_lambert_az_rect.ps
```

Note that an "r" is appended to the **-R** option to inform GMT that the region has been selected using the rectangle technique, otherwise it would try to decode the values as *west, east, south, north* and report an error since *'east'* < *'west'*.

Hemisphere map

Here, you must specify the world as your region (**-Rg** or **-Rd**). E.g., to obtain a hemisphere view that shows the Americas, try

```
gmt pscoast -Rg -JA280/30/3.5i -Bg -Dc -A1000 -Gnavy -P > GMT_lambert_az_hemi.ps
```

To geologists, the Lambert azimuthal equal-area projection (with origin at 0/0) is known as the *equal-area* (Schmidt) stereonet and used for plotting fold axes, fault planes, and the like. An *equal-angle*

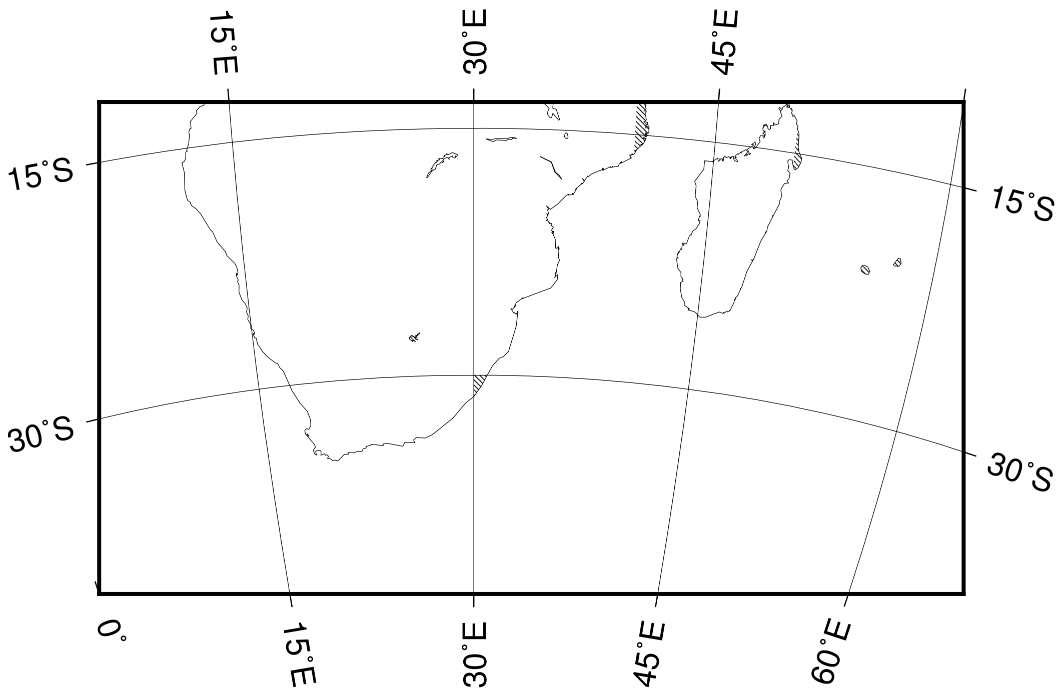


Figure 10.5: Rectangular map using the Lambert azimuthal equal-area projection.

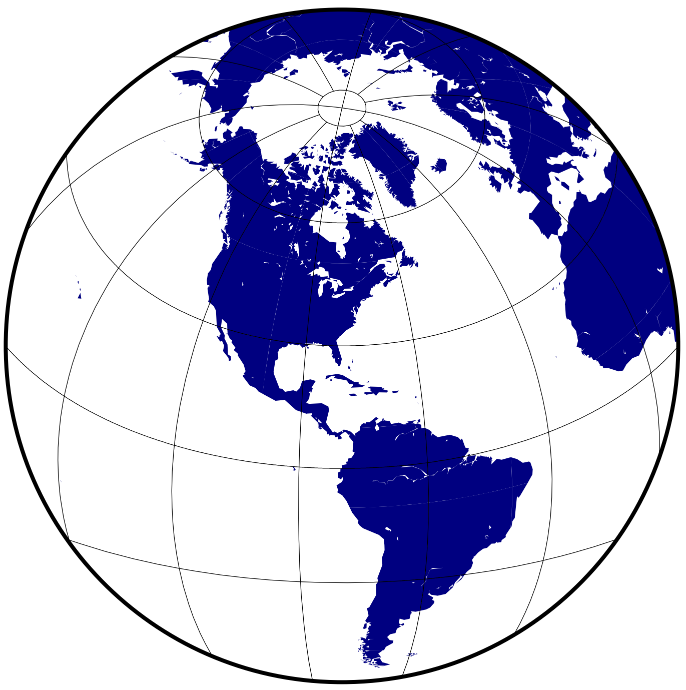


Figure 10.6: Hemisphere map using the Lambert azimuthal equal-area projection.

(Wulff) stereonet can be obtained by using the stereographic projection (discussed later). The stereonets produced by these two projections appear below.

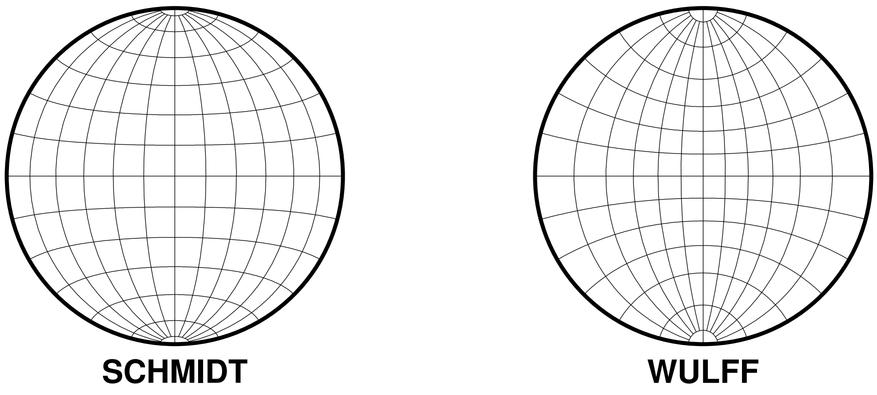


Figure 10.7: Equal-Area (Schmidt) and Equal-Angle (Wulff) stereo nets.

10.2.2 Stereographic Equal-Angle projection (-Js -JS)

This is a conformal, azimuthal projection that dates back to the Greeks. Its main use is for mapping the polar regions. In the polar aspect all meridians are straight lines and parallels are arcs of circles. While this is the most common use it is possible to select any point as the center of projection. The requirements are

- Longitude and latitude of the projection center.
- Optionally, the horizon, i.e., the number of degrees from the center to the edge (< 180, default is 90).
- Scale as 1:xxxxx (true scale at pole), slat/1:xxxxx (true scale at standard parallel slat), or radius/latitude where radius is distance on map in inches from projection center to a particular [possibly oblique] latitude (**-Js**), or simply map width (**-JS**).

A default map scale factor of 0.9996 will be applied by default (although you may change this with *PROJ_SCALE_FACTOR*). However, the setting is ignored when a standard parallel has been specified since the scale is then implicitly given. We will look at two different types of maps.

Polar Stereographic Map

In our first example we will let the projection center be at the north pole. This means we have a polar stereographic projection and the map boundaries will coincide with lines of constant longitude and latitude. An example is given by

```
gmt pscoast -R-30/30/60/72 -Js0/90/4.5i/60 -B10g -Dl -A250 -Groyalblue \
            -Sseashell -P > GMT_stereographic_polar.ps
```

Rectangular stereographic map

As with Lambert's azimuthal equal-area projection we have the option to use rectangular boundaries rather than the wedge-shape typically associated with polar projections. This choice is defined by selecting two points as corners in the rectangle and appending an "r" to the **-R** option. This command produces a map as presented in Figure *Polar stereographic*:

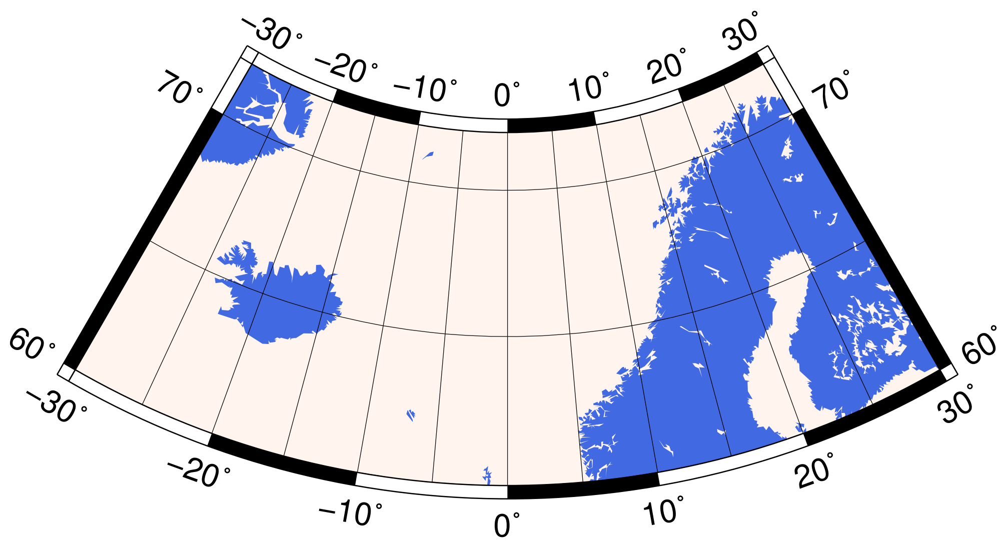


Figure 10.8: Polar stereographic conformal projection.

```
gmt set MAP_ANNOT_OBLIQUE 30
gmt pscoast -R-25/59/70/72r -JS10/90/11c -B20g -Dl -A250 -Gdarkbrown -Wthinnest \
            -Slightgray -P > GMT_stereographic_rect.ps
```

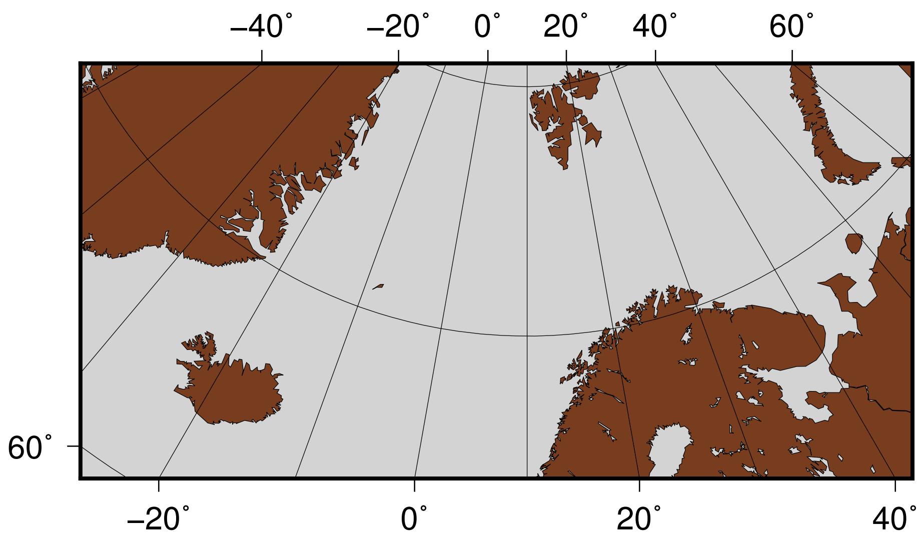


Figure 10.9: Polar stereographic conformal projection with rectangular borders.

General stereographic map

In terms of usage this projection is identical to the Lambert azimuthal equal-area projection. Thus, one can make both rectangular and hemispheric maps. Our example shows Australia using a projection pole at 130E/30S. The command used was

```
gmt set MAP_ANNOT_OBLIQUE 0
gmt pscoast -R100/-42/160/-8r -JS130/-30/4i -Bag -Dl -A500 -Ggreen -Slightblue \
            -Wthinnest -P > GMT_stereographic_general.ps
```

By choosing 0/0 as the pole, we obtain the conformal stereonet presented next to its equal-area cousin in the Section Lambert Azimuthal Equal-Area (-Ja -JA) on the Lambert azimuthal equal-area projection (Figure *Stereonets*).

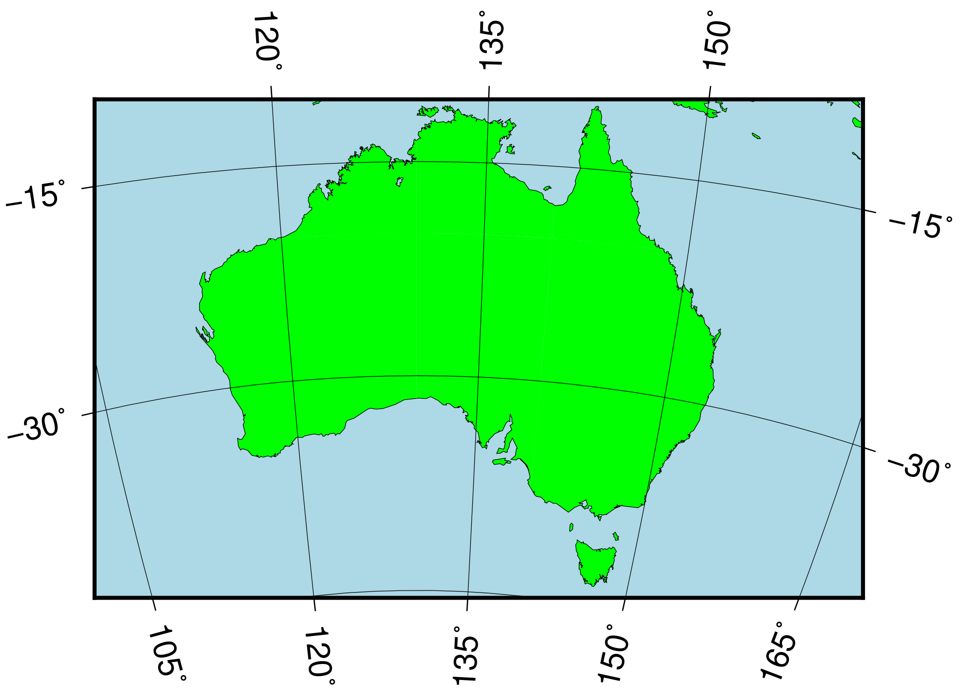


Figure 10.10: General stereographic conformal projection with rectangular borders.

10.2.3 Perspective projection (-Jg -JG)

The perspective projection imitates in 2 dimensions the 3-dimensional view of the earth from space. The implementation in GMT is very flexible, and thus requires many input variables. Those are listed and explained below, with the values used in Figure *Perspective projection* between brackets.

- Longitude and latitude of the projection center (4E/52N).
- Altitude of the viewer above sea level in kilometers (230 km). If this value is less than 10, it is assumed to be the distance of the viewer from the center of the earth in earth radii. If an "r" is appended, it is the distance from the center of the earth in kilometers.
- Azimuth in degrees (90, due east). This is the direction in which you are looking, measured clockwise from north.
- Tilt in degrees (60). This is the viewing angle relative to zenith. So a tilt of 0 is looking straight down, 60 is looking from 30 above the horizon.
- Twist in degrees (180). This is the boresight rotation (clockwise) of the image. The twist of 180 in the example mimics the fact that the Space Shuttle flies upside down.
- Width and height of the viewpoint in degrees (60). This number depends on whether you are looking with the naked eye (in which case you view is about 60 wide), or with binoculars, for example.
- Scale as 1:xxxxx or as radius/latitude where radius is distance on map in inches from projection center to a particular [possibly oblique] latitude (**-Jg**), or map width (**-JG**) (5 inches).

The imagined view of northwest Europe from a Space Shuttle at 230 km looking due east is thus accomplished by the following `pscoast` command:

```
gmt pscoast -Rg -JG4/52/230/90/60/180/60/60/5i -Bx2g2 -By1g1 -Ia -Di -Glightbrown \
            -Wthinnest -P -Slightblue --MAP_ANNOT_MIN_SPACING=0.25i > GMT_perspective.ps
```

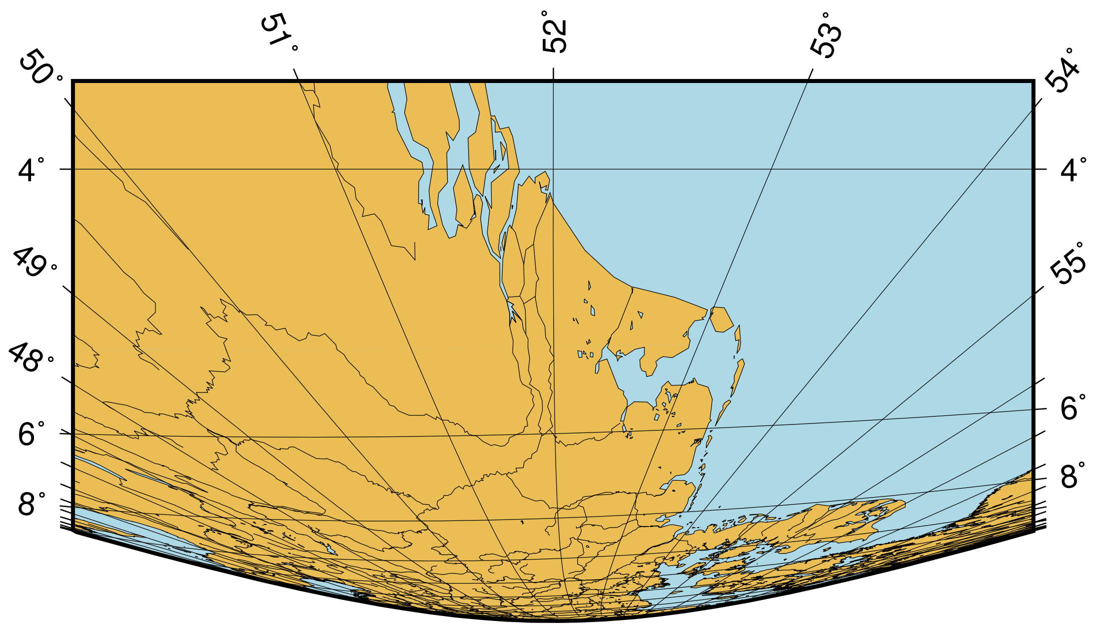


Figure 10.11: View from the Space Shuttle in Perspective projection.

10.2.4 Orthographic projection (-Jg -JG)

The orthographic azimuthal projection is a perspective projection from infinite distance. It is therefore often used to give the appearance of a globe viewed from outer space. As with Lambert's equal-area and the stereographic projection, only one hemisphere can be viewed at any time. The projection is neither equal-area nor conformal, and much distortion is introduced near the edge of the hemisphere. The directions from the center of projection are true. The projection was known to the Egyptians and Greeks more than 2,000 years ago. Because it is mainly used for pictorial views at a small scale, only the spherical form is necessary.

To specify the orthographic projection the same options **-Jg** or **-JG** as the perspective projection are used, but with fewer variables to supply:

- Longitude and latitude of the projection center.
- Optionally, the horizon, i.e., the number of degrees from the center to the edge (<= 90, default is 90).
- Scale as 1:xxxxx or as radius/latitude where radius is distance on map in inches from projection center to a particular [possibly oblique] latitude (**-Jg**), or map width (**-JG**).

Our example of a perspective view centered on 75W/40N can therefore be generated by the following `pscoast` command:

```
gmt pscoast -Rg -JG-75/41/4.5i -Bg -Dc -A5000 -Gpink -Sthistle -P > GMT_orthographic.ps
```

10.2.5 Azimuthal Equidistant projection (-Je -JE)

The most noticeable feature of this azimuthal projection is the fact that distances measured from the center are true. Therefore, a circle about the projection center defines the locus of points that are equally far away from the plot origin. Furthermore, directions from the center are also true. The projection, in the polar aspect, is at least several centuries old. It is a useful projection for a global view of locations at various or identical distance from a given point (the map center).

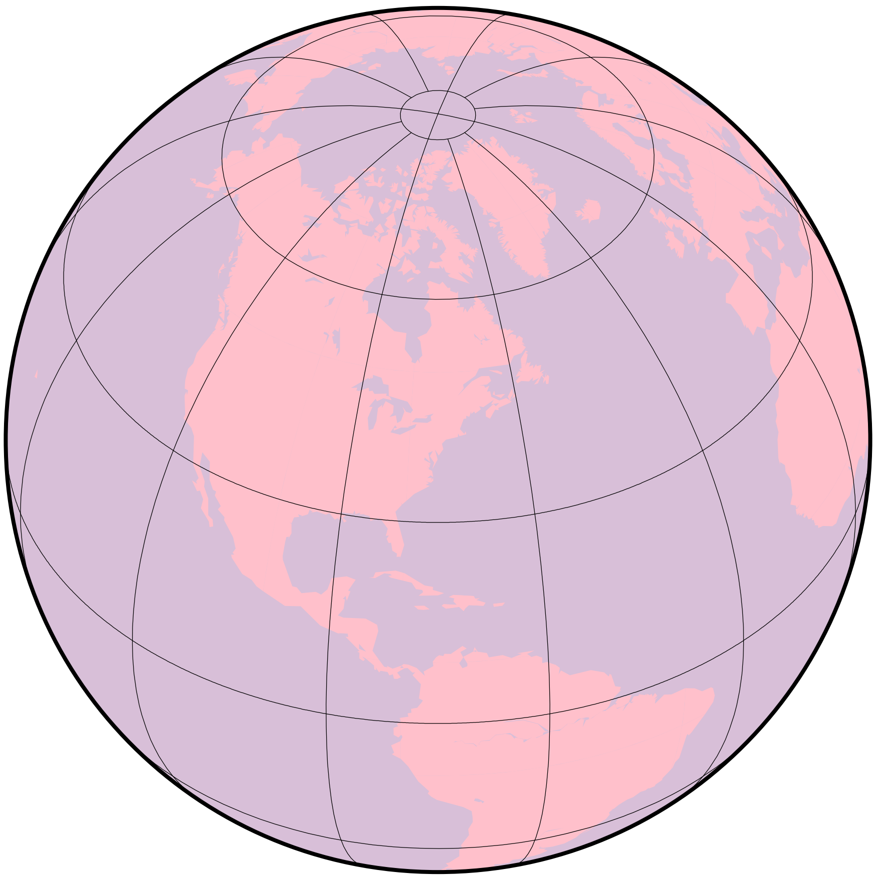


Figure 10.12: Hemisphere map using the Orthographic projection.

To specify the azimuthal equidistant projection you must supply:

- Longitude and latitude of the projection center.
- Optionally, the horizon, i.e., the number of degrees from the center to the edge (<= 180, default is 180).
- Scale as 1:xxxxx or as radius/latitude where radius is distance on map in inches from projection center to a particular [possibly oblique] latitude (**-Je**), or map width (**-JE**).

Our example of a global view centered on 100W/40N can therefore be generated by the following `pscoast` command. Note that the antipodal point is 180 away from the center, but in this projection this point plots as the entire map perimeter:

```
gmt pscoast -Rg -JE-100/40/4.5i -Bg -Dc -A10000 -Glightgray -Wthinnest -P \
            > GMT_az_equidistant.ps
```

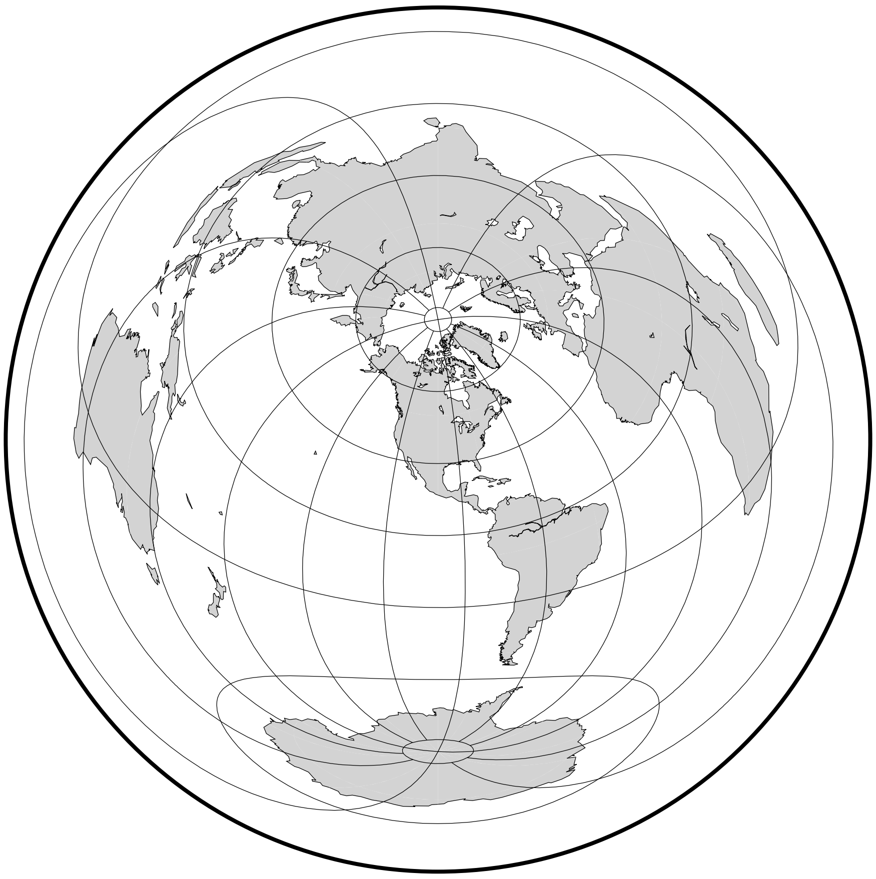


Figure 10.13: World map using the equidistant azimuthal projection.

10.2.6 Gnomonic projection (-Jf -JF)

The Gnomonic azimuthal projection is a perspective projection from the center onto a plane tangent to the surface. Its origin goes back to the old Greeks who used it for star maps almost 2500 years ago. The projection is neither equal-area nor conformal, and much distortion is introduced near the edge of the hemisphere; in fact, less than a hemisphere may be shown around a given center. The directions from the center of projection are true. Great circles project onto straight lines. Because it is mainly used for pictorial views at a small scale, only the spherical form is necessary.

To specify the Gnomonic projection you must supply:

- Longitude and latitude of the projection center.
- Optionally, the horizon, i.e., the number of degrees from the center to the edge (< 90, default is 60).
- Scale as 1:xxxxx or as radius/latitude where radius is distance on map in inches from projection center to a particular [possibly oblique] latitude (**-Jf**), or map width (**-JF**).

Using a horizon of 60, our example of this projection centered on 120W/35N can therefore be generated by the following `pscoast` command:

```
gmt pscoast -Rg -JF-120/35/60/4.5i -B30g15 -Dc -A10000 -Gtan -Scyan -Wthinnest \
            -P > GMT_gnomonic.ps
```

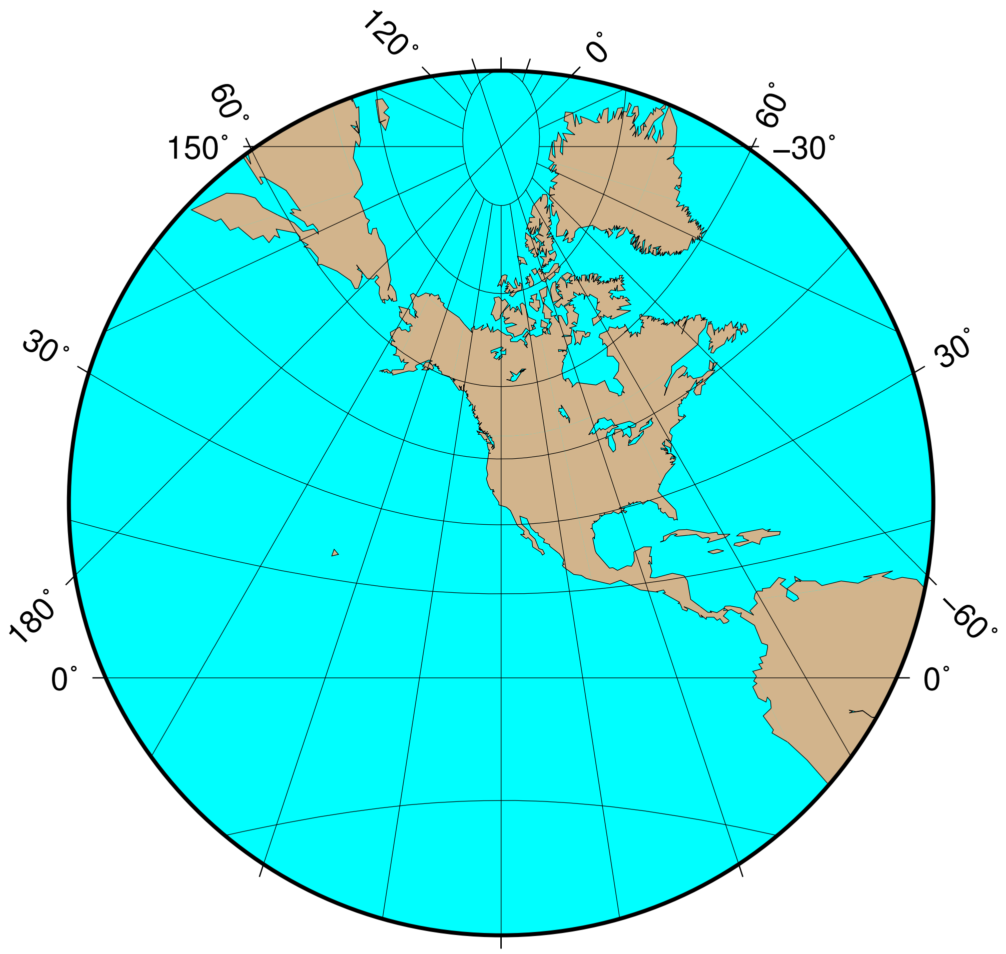


Figure 10.14: Gnomonic azimuthal projection.

10.3 Cylindrical projections

Cylindrical projections are easily recognized for its shape: maps are rectangular and meridians and parallels are straight lines crossing at right angles. But that is where similarities between the cylindrical projections supported by GMT (Mercator, transverse Mercator, universal transverse Mercator, oblique Mercator, Cassini, cylindrical equidistant, cylindrical equal-area, Miller, and cylindrical stereographic projections) stops. Each have a different way of spacing the meridians and parallels to obtain certain desirable cartographic properties.

10.3.1 Mercator projection (-Jm -JM)

Probably the most famous of the various map projections, the Mercator projection takes its name from the Flemish cartographer Gheert Cremer, better known as Gerardus Mercator, who presented it in 1569. The projection is a cylindrical and conformal, with no distortion along the equator. A major navigational feature of the projection is that a line of constant azimuth is straight. Such a line is called a rhumb line or *loxodrome*. Thus, to sail from one point to another one only had to connect the points with a straight line, determine the azimuth of the line, and keep this constant course for the entire voyage [2]. The Mercator projection has been used extensively for world maps in which the distortion towards the polar regions grows rather large, thus incorrectly giving the impression that, for example, Greenland is larger than South America. In reality, the latter is about eight times the size of Greenland. Also, the Former Soviet Union looks much bigger than Africa or South America. One may wonder whether this illusion has had any influence on U.S. foreign policy.

In the regular Mercator projection, the cylinder touches the globe along the equator. Other orientations like vertical and oblique give rise to the Transverse and Oblique Mercator projections, respectively. We will discuss these generalizations following the regular Mercator projection.

The regular Mercator projection requires a minimum of parameters. To use it in GMT programs you supply this information (the first two items are optional and have defaults):

- Central meridian [Middle of your map].
- Standard parallel for true scale [Equator]. When supplied, central meridian must be supplied as well.
- Scale along the equator in inch/degree or 1:xxxxx (**-Jm**), or map width (**-JM**).

Our example presents a world map at a scale of 0.012 inch pr degree which will give a map 4.32 inch wide. It was created with the command:

```
gmt set MAP_FRAME_TYPE fancy
gmt pscoast -R0/360/-70/70 -Jm1.2e-2i -Bxa60f15 -Bya30f15 -Dc -A5000 -Gred \
            -P > GMT_mercator.ps
```

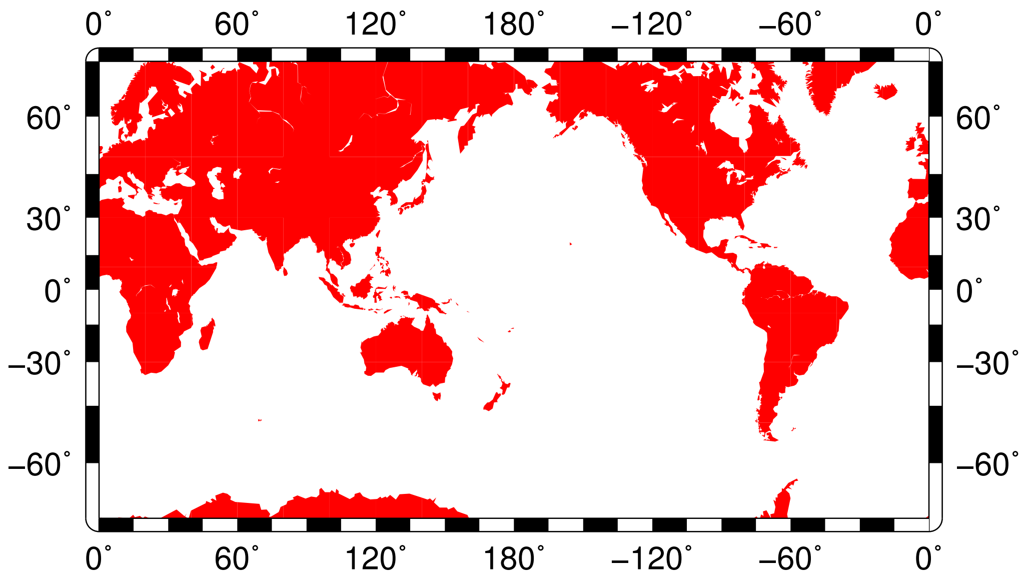


Figure 10.15: Simple Mercator map.

[2] This is, however, not the shortest distance. It is given by the great circle connecting the two points.

While this example is centered on the Dateline, one can easily choose another configuration with the **-R** option. A map centered on Greenwich would specify the region with **-R**-180/180/-70/70.

10.3.2 Transverse Mercator projection (-Jt -JT)

The transverse Mercator was invented by Lambert in 1772. In this projection the cylinder touches a meridian along which there is no distortion. The distortion increases away from the central meridian and goes to infinity at 90 from center. The central meridian, each meridian 90 away from the center, and equator are straight lines; other parallels and meridians are complex curves. The projection is defined by specifying:

- The central meridian.
- Optionally, the latitude of origin (default is the equator).
- Scale along the equator in inch/degree or 1:xxxxx (**-Jt**), or map width (**-JT**).

The optional latitude of origin defaults to Equator if not specified. Although defaulting to 1, you can change the map scale factor via the *PROJ_SCALE_FACTOR* parameter. Our example shows a transverse Mercator map of south-east Europe and the Middle East with 35E as the central meridian:

```
gmt pscoast -R20/30/50/45r -Jt35/0.18i -Bag -Dl -A250 -Glightbrown -Wthinnest \
            -P -Sseashell > GMT_transverse_merc.ps
```

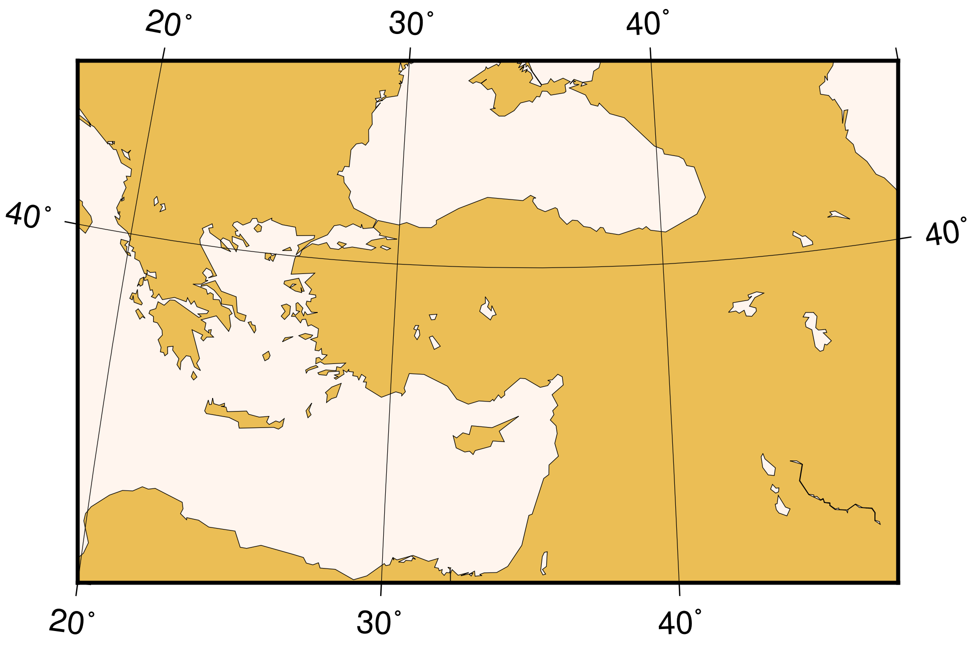


Figure 10.16: Rectangular Transverse Mercator map.

The transverse Mercator can also be used to generate a global map - the equivalent of the 360 Mercator map. Using the command

```
gmt pscoast -R0/360/-80/80 -JT330/-45/3.5i -Ba30g -BWSne -Dc -A2000 \
            -Slightblue -G0 -P > GMT_TM.ps
```

we made the map illustrated in Figure *Global transverse Mercator*. Note that when a world map is given (indicated by **-R***0/360/s/n*), the arguments are interpreted to mean oblique degrees, i.e., the 360 range is understood to mean the extent of the plot along the central meridian, while the “south” and “north” values represent how far from the central longitude we want the plot to extend. These values correspond to latitudes in the regular Mercator projection and must therefore be less than 90.

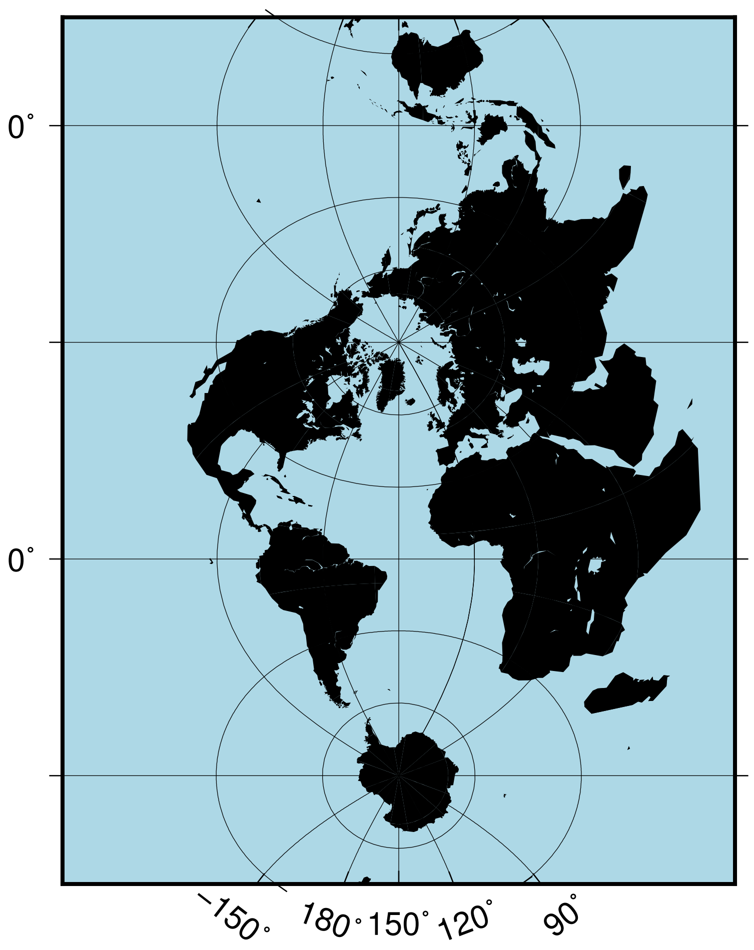


Figure 10.17: A global transverse Mercator map.

10.3.3 Universal Transverse Mercator (UTM) projection (-Ju -JU)

A particular subset of the transverse Mercator is the Universal Transverse Mercator (UTM) which was adopted by the US Army for large-scale military maps. Here, the globe is divided into 60 zones between 84S and 84N, most of which are 6 wide. Each of these UTM zones have their unique central meridian. Furthermore, each zone is divided into latitude bands but these are not needed to specify the projection for most cases. See Figure *Universal Transverse Mercator* for all zone designations.

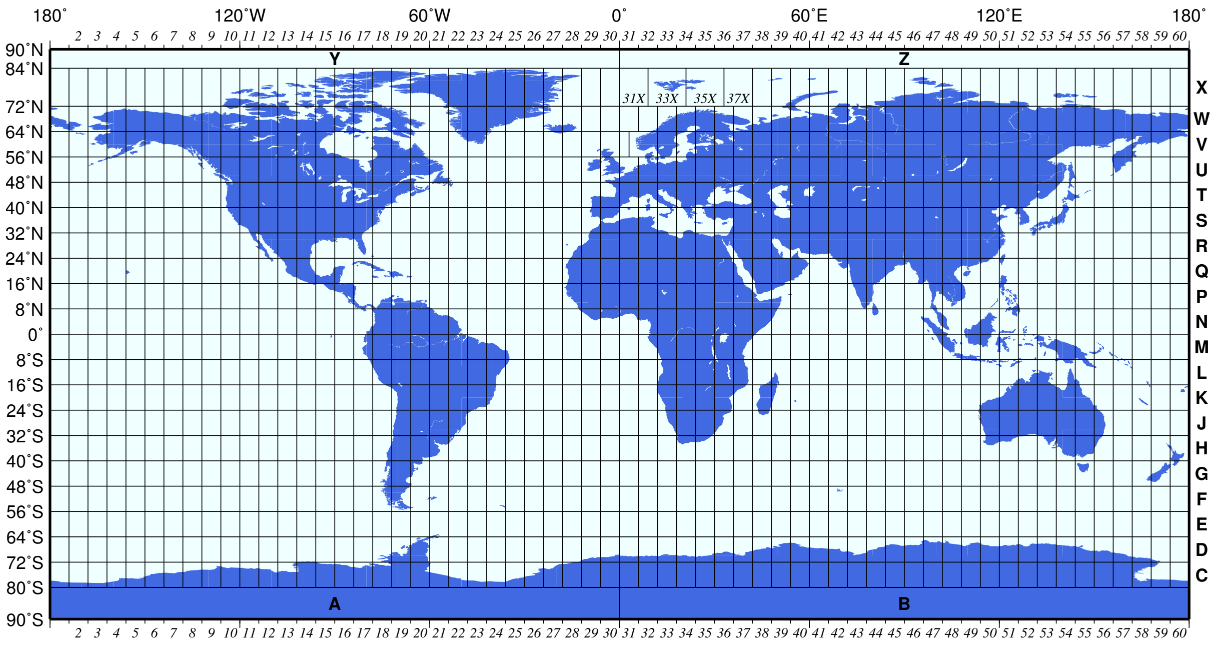


Figure 10.18: Universal Transverse Mercator zone layout.

GMT implements both the transverse Mercator and the UTM projection. When selecting UTM you must specify:

- UTM zone (A, B, 1–60, Y, Z). Use negative values for numerical zones in the southern hemisphere or append the latitude modifiers C–H, J–N, P–X) to specify an exact UTM grid zone.
- Scale along the equator in inch/degree or 1:xxxxx (**-Ju**), or map width (**-JU**).

In order to minimize the distortion in any given zone, a scale factor of 0.9996 has been factored into the formulae. (although a standard, you can change this with *PROJ_SCALE_FACTOR*). This makes the UTM projection a *secant* projection and not a *tangent* projection like the transverse Mercator above. The scale only varies by 1 part in 1,000 from true scale at equator. The ellipsoidal projection expressions are accurate for map areas that extend less than 10 away from the central meridian. For larger regions we use the conformal latitude in the general spherical formulae instead.

10.3.4 Oblique Mercator projection (-Jo -JO)

Oblique configurations of the cylinder give rise to the oblique Mercator projection. It is particularly useful when mapping regions of large lateral extent in an oblique direction. Both parallels and meridians are complex curves. The projection was developed in the early 1900s by several workers. Several parameters must be provided to define the projection. GMT offers three different definitions:

1. Option **-Joa** or **-JOa**:
    - Longitude and latitude of projection center.

  - Azimuth of the oblique equator.
  - Scale in inch/degree or 1:xxxxx along oblique equator (**-Joa**), or map width (**-JOa**).

2. Option **-Job** or **-JOb**:
  - Longitude and latitude of projection center.
  - Longitude and latitude of second point on oblique equator.
  - Scale in inch/degree or 1:xxxxx along oblique equator (**-Job**), or map width (**-JOb**).

3. Option **-Joc** or **-JOc**:
  - Longitude and latitude of projection center.
  - Longitude and latitude of projection pole.
  - Scale in inch/degree or 1:xxxxx along oblique equator (**-Joc**), or map width (**-JOc**).

Our example was produced by the command

```
gmt pscoast -R270/20/305/25r -JOc280/25.5/22/69/4.8i -Bag -Di -A250 -Gburlywood \
            -Wthinnest -P -Tf301.5/23/0.4i/2 -Sazure --FONT_TITLE=8p \
            --MAP_TITLE_OFFSET=0.05i > GMT_obl_merc.ps
```

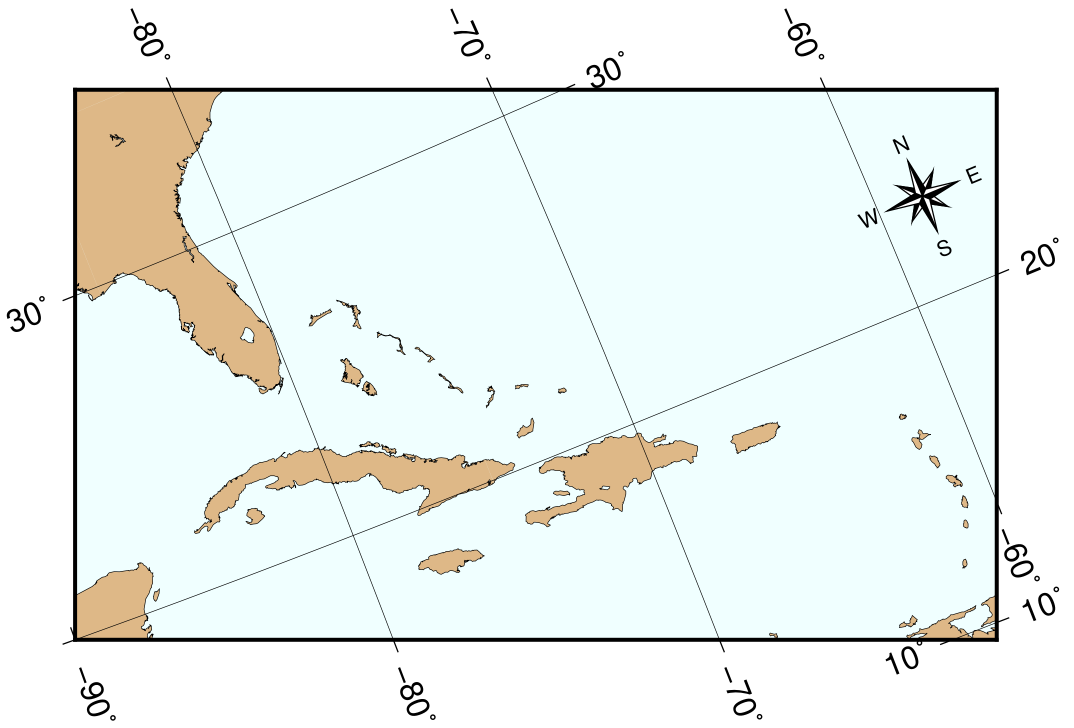


Figure 10.19: Oblique Mercator map using **-Joc**. We make it clear which direction is North by adding a star rose with the **-T** option.

It uses definition 3 for an oblique view of some Caribbean islands. Note that we define our region using the rectangular system described earlier. If we do not append an "r" to the **-R** string then the information provided with the **-R** option is assumed to be oblique degrees about the projection center rather than the usual geographic coordinates. This interpretation is chosen since in general the parallels and meridians are not very suitable as map boundaries.

10.3.5 Cassini cylindrical projection (-Jc -JC)

This cylindrical projection was developed in 1745 by César-François Cassini de Thury for the survey of France. It is occasionally called Cassini-Soldner since the latter provided the more accurate mathematical analysis that led to the development of the ellipsoidal formulae. The projection is neither conformal nor equal-area, and behaves as a compromise between the two end-members. The distortion is zero along the central meridian. It is best suited for mapping regions of north-south extent. The central meridian, each meridian 90 away, and equator are straight lines; all other meridians and parallels are complex curves. The requirements to define this projection are:

- Longitude and latitude of central point.
- Scale in inch/degree or as 1:xxxxx (**-Jc**), or map width (**-JC**).

A detailed map of the island of Sardinia centered on the 845'E meridian using the Cassini projection can be obtained by running the command:

```
gmt pscoast -R7:30/38:30/10:30/41:30r -JC8.75/40/2.5i -Bafg -Lf9.5/38.8/40/60 -Gspringgreen \
            -Dh -Sazure -Wthinnest -Ia/thinner -P --FONT_LABEL=12p > GMT_cassini.ps
```

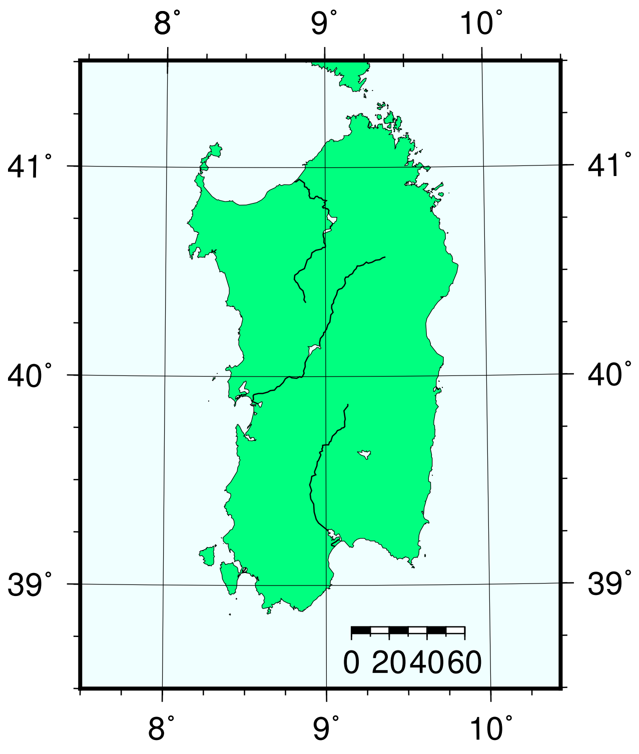


Figure 10.20: Cassini map over Sardinia.

As with the previous projections, the user can choose between a rectangular boundary (used here) or a geographical (WESN) boundary.

10.3.6 Cylindrical equidistant projection (-Jq -JQ)

This simple cylindrical projection is really a linear scaling of longitudes and latitudes. The most common form is the Plate Carrée projection, where the scaling of longitudes and latitudes is the same. All meridians and parallels are straight lines. The projection can be defined by:

- The central meridian [Middle of your map].

- Standard parallel [Equator].
- Scale in inch/degree or as 1:xxxxx (**-Jq**), or map width (**-JQ**).

The first two of these are optional and have defaults. When the standard parallel is defined, the central meridian must be supplied as well.

A world map centered on the dateline using this projection can be obtained by running the command:

```
gmt pscoast -Rg -JQ4.5i -B60f30g30 -Dc -A5000 -Gtan4 -Slightcyan -P > GMT_equi_cyl.ps
```

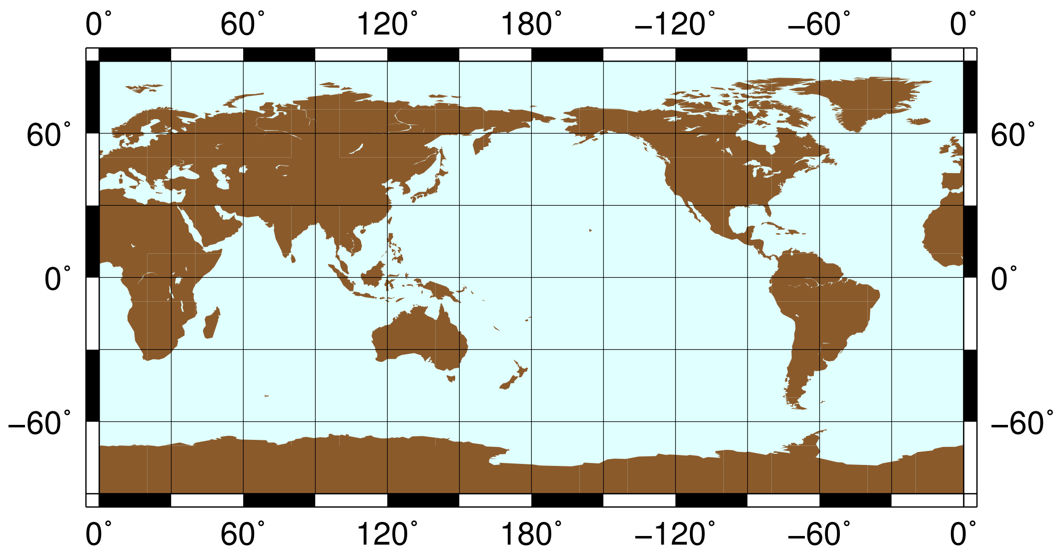


Figure 10.21: World map using the Plate Carr'{e}e projection.

Different relative scalings of longitudes and latitudes can be obtained by selecting a standard parallel different from the equator. Some selections for standard parallels have practical properties as shown in Table *JQ*.

Grafarend and Niermann, minimum linear distortion	61.7
Ronald Miller Equirectangular	50.5
Ronald Miller, minimum continental distortion	43.5
Grafarend and Niermann	42
Ronald Miller, minimum overall distortion	37.5
Plate Carrée, Simple Cylindrical, Plain/Plane	0

10.3.7 Cylindrical equal-area projections (-Jy -JY)

This cylindrical projection is actually several projections, depending on what latitude is selected as the standard parallel. However, they are all equal area and hence non-conformal. All meridians and parallels are straight lines. The requirements to define this projection are:

- The central meridian.
- The standard parallel.
- Scale in inch/degree or as 1:xxxxx (**-Jy**), or map width (**-JY**)

While you may choose any value for the standard parallel and obtain your own personal projection, there are seven choices of standard parallels that result in known (or named) projections. These are listed in

Table *JY*.

Balthasart	50
Gall-Peters	45
Hobo-Dyer	3730’ (= 37.5)
Trystan Edwards	3724’ (= 37.4)
Caster	3704’ (= 37.0666)
Behrman	30
Lambert	0

For instance, a world map centered on the 35E meridian using the Behrman projection (Figure *Behrman cylindrical projection*) can be obtained by running the command:

```
gmt pscoast -R-145/215/-90/90 -JY35/30/4.5i -B45g45 -Dc -A10000 -Sdodgerblue \
            -Wthinnest -P > GMT_general_cyl.ps
```

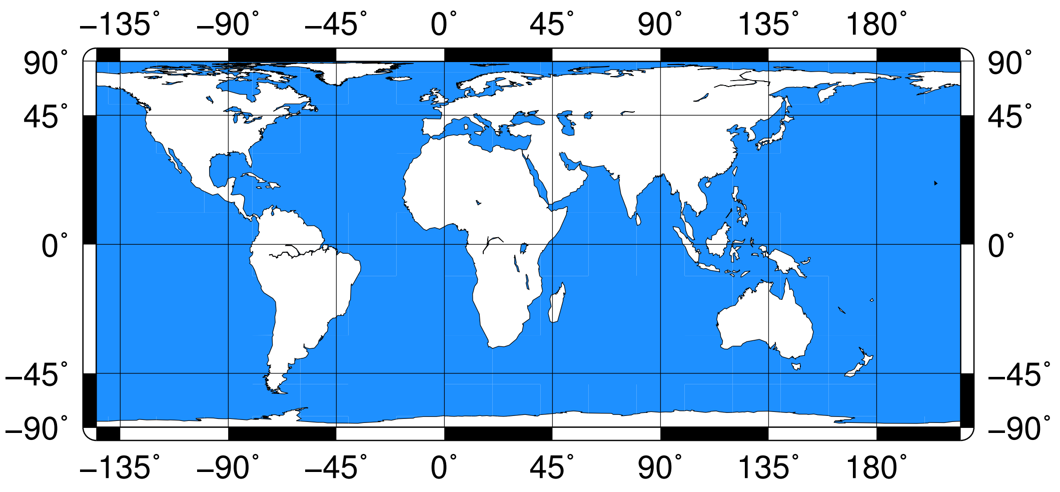


Figure 10.22: World map using the Behrman cylindrical equal-area projection.

As one can see there is considerable distortion at high latitudes since the poles map into lines.

10.3.8 Miller Cylindrical projection (-Jj -JJ)

This cylindrical projection, presented by Osborn Maitland Miller of the American Geographic Society in 1942, is neither equal nor conformal. All meridians and parallels are straight lines. The projection was designed to be a compromise between Mercator and other cylindrical projections. Specifically, Miller spaced the parallels by using Mercator’s formula with 0.8 times the actual latitude, thus avoiding the singular poles; the result was then divided by 0.8. There is only a spherical form for this projection. Specify the projection by:

- Optionally, the central meridian (default is the middle of your map).
- Scale in inch/degree or as 1:xxxxx (**-Jj**), or map width (**-JJ**).

For instance, a world map centered on the 90E meridian at a map scale of 1:400,000,000 (Figure *Miller projection*) can be obtained as follows:

```
gmt pscoast -R-90/270/-80/90 -Jj1:400000000 -Bx45g45 -By30g30 -Dc -A10000 \
            -Gkhaki -Wthinnest -P -Sazure > GMT_miller.ps
```

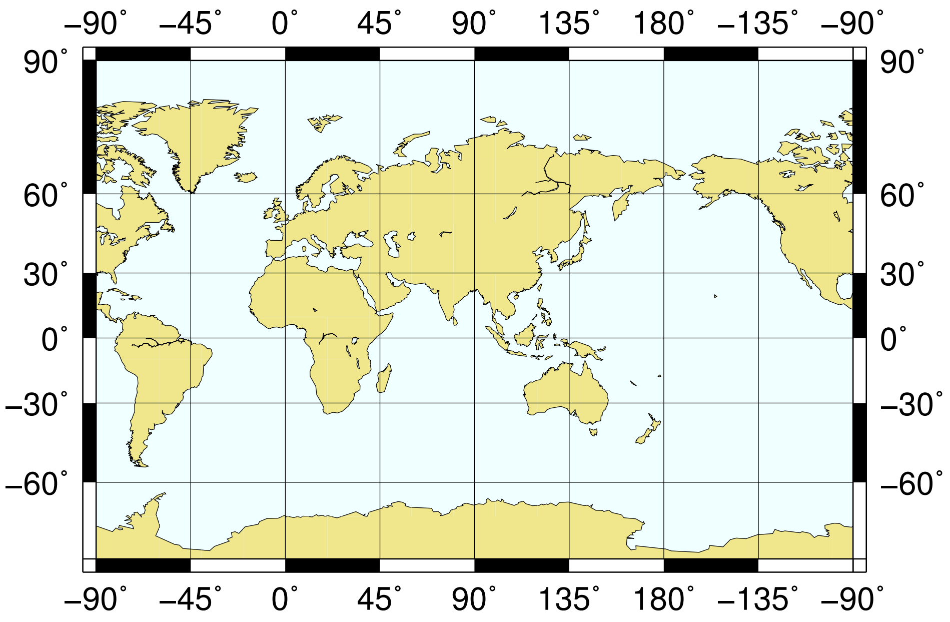


Figure 10.23: World map using the Miller cylindrical projection.

10.3.9 Cylindrical stereographic projections (-Jcyl_stere -JCyl_stere)

The cylindrical stereographic projections are certainly not as notable as other cylindrical projections, but are still used because of their relative simplicity and their ability to overcome some of the downsides of other cylindrical projections, like extreme distortions of the higher latitudes. The stereographic projections are perspective projections, projecting the sphere onto a cylinder in the direction of the antipodal point on the equator. The cylinder crosses the sphere at two standard parallels, equidistant from the equator. The projections are defined by:

- The central meridian (uses the middle of the map when omitted).
- The standard parallel (default is the Equator). When used, central meridian needs to be given as well.
- Scale in inch/degree or as 1:xxxxx (**-Jcyl_stere**), or map width (**-JCyl_stere**)

Some of the selections of the standard parallel are named for the cartographer or publication that popularized the projection (Table *JCylstere*).

Miller's modified Gall	66.159467
Kamenetskiy's First	55
Gall's stereographic	45
Bolshoi Sovietskii Atlas Mira or Kamenetskiy's Second	30
Braun's cylindrical	0

A map of the world, centered on the Greenwich meridian, using the Gall's stereographic projection (standard parallel is 45, Figure *Gall's stereographic projection*), is obtained as follows:

```
gmt set FORMAT_GEO_MAP dddA
gmt pscoast -R-180/180/-60/80 -JCyl_stere/0/45/4.5i -Bxa60f30g30 -Bya30g30 -Dc -A5000 \
            -Wblack -Gseashell4 -Santiquewhite1 -P > GMT_gall_stereo.ps
```

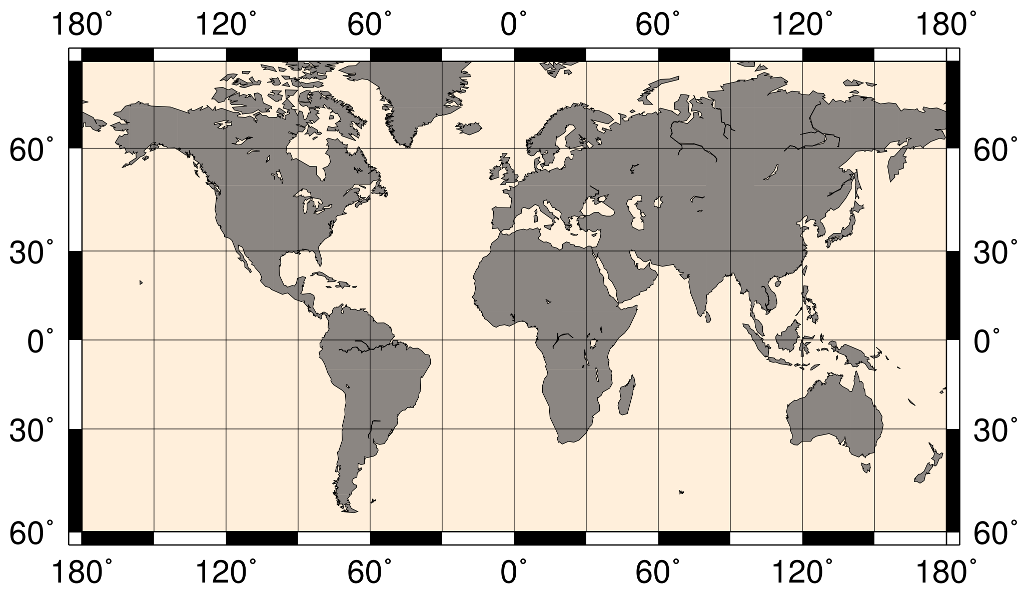


Figure 10.24: World map using Gall's stereographic projection.

10.4 Miscellaneous projections

GMT supports 8 common projections for global presentation of data or models. These are the Hammer, Mollweide, Winkel Tripel, Robinson, Eckert IV and VI, Sinusoidal, and Van der Grinten projections. Due to the small scale used for global maps these projections all use the spherical approximation rather than more elaborate elliptical formulae.

In all cases, the specification of the central meridian can be skipped. The default is the middle of the longitude range of the plot, specified by the (R) option.

10.4.1 Hammer projection (-Jh -JH)

The equal-area Hammer projection, first presented by the German mathematician Ernst von Hammer in 1892, is also known as Hammer-Aitoff (the Aitoff projection looks similar, but is not equal-area). The border is an ellipse, equator and central meridian are straight lines, while other parallels and meridians are complex curves. The projection is defined by selecting:

- The central meridian [Middle of your map].
- Scale along equator in inch/degree or 1:xxxxx (**-Jh**), or map width (**-JH**).

A view of the Pacific ocean using the Dateline as central meridian is accomplished thus

```
gmt pscoast -Rg -JH4.5i -Bg -Dc -A10000 -Gblack -Scornsilk -P > GMT_hammer.ps
```

10.4.2 Mollweide projection (-Jw -JW)

This pseudo-cylindrical, equal-area projection was developed by the German mathematician and astronomer Karl Brandan Mollweide in 1805. Parallels are unequally spaced straight lines with the meridians being equally spaced elliptical arcs. The scale is only true along latitudes 4044' north and south. The projection is used mainly for global maps showing data distributions. It is occasionally referenced under the name homalographic projection. Like the Hammer projection, outlined above, we need to specify

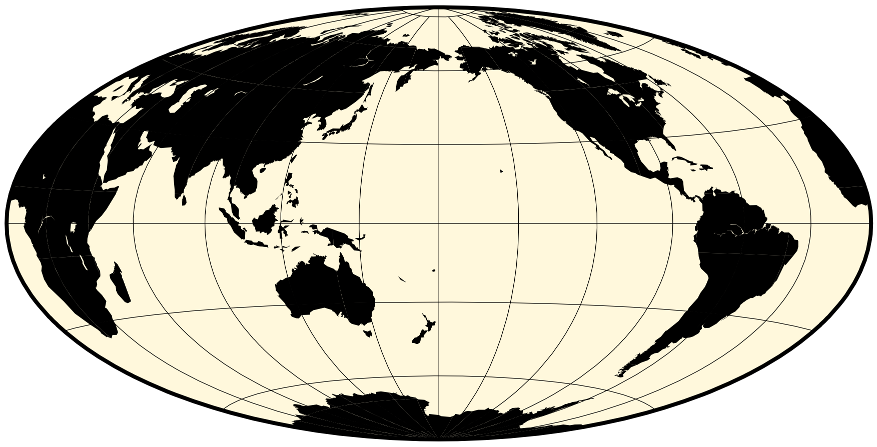


Figure 10.25: World map using the Hammer projection.

only two parameters to completely define the mapping of longitudes and latitudes into rectangular *x*/*y* coordinates:

- The central meridian [Middle of your map].
- Scale along equator in inch/degree or 1:xxxxx (**-Jw**), or map width (**-JW**).

An example centered on Greenwich can be generated thus:

```
gmt pscoast -Rd -JW4.5i -Bg -Dc -A10000 -Gtomato1 -Sskyblue -P > GMT_mollweide.ps
```

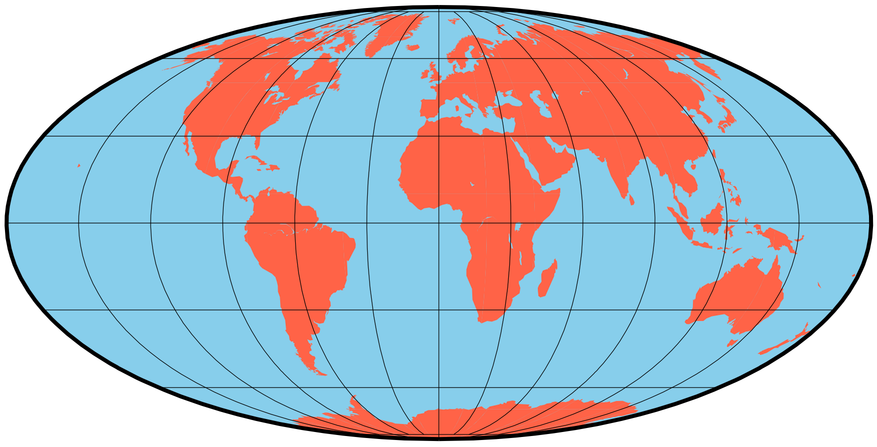


Figure 10.26: World map using the Mollweide projection.

10.4.3 Winkel Tripel projection (-Jr -JR)

In 1921, the German mathematician Oswald Winkel a projection that was to strike a compromise between the properties of three elements (area, angle and distance). The German word “tripel” refers to this junction of where each of these elements are least distorted when plotting global maps. The projection was popularized when Bartholomew and Son started to use it in its world-renowned “The Times Atlas of the World” in the mid 20th century. In 1998, the National Geographic Society made the Winkel Tripel as its map projection of choice for global maps.

Naturally, this projection is neither conformal, nor equal-area. Central meridian and equator are straight lines; other parallels and meridians are curved. The projection is obtained by averaging the coordinates of the Equidistant Cylindrical and Aitoff (not Hammer-Aitoff) projections. The poles map into straight lines 0.4 times the length of equator. To use it you must enter

- The central meridian [Middle of your map].
- Scale along equator in inch/degree or 1:xxxxx (**-Jr**), or map width (**-JR**).

Centered on Greenwich, the example in Figure *Winkel Tripel projection* was created by this command:

```
gmt pscoast -Rd -JR4.5i -Bg -Dc -A10000 -Gburlywood4 -Swheat1 -P > GMT_winkel.ps
```

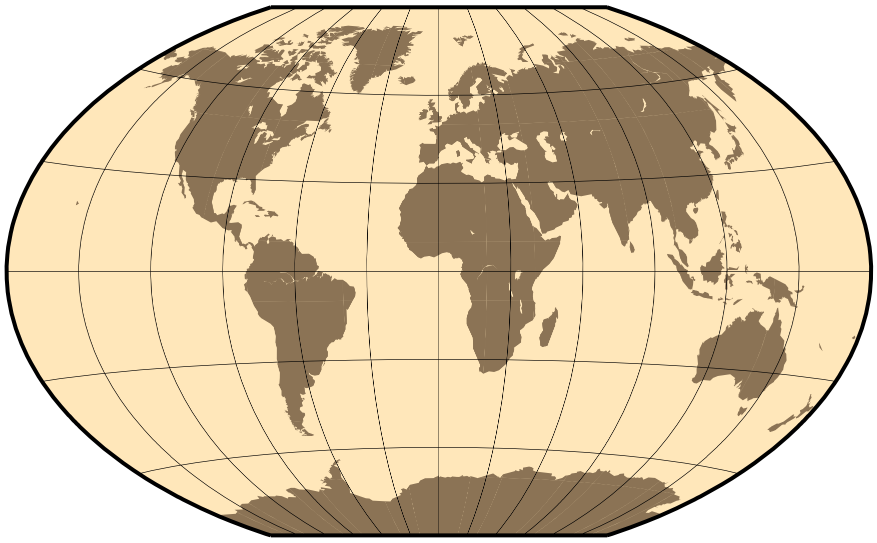


Figure 10.27: World map using the Winkel Tripel projection.

10.4.4 Robinson projection (-Jn -JN)

The Robinson projection, presented by the American geographer and cartographer Arthur H. Robinson in 1963, is a modified cylindrical projection that is neither conformal nor equal-area. Central meridian and all parallels are straight lines; other meridians are curved. It uses lookup tables rather than analytic expressions to make the world map "look" right [3]. The scale is true along latitudes 38. The projection was originally developed for use by Rand McNally and is currently used by the National Geographic Society. To use it you must enter

- The central meridian [Middle of your map].
- Scale along equator in inch/degree or 1:xxxxx (**-Jn**), or map width (**-JN**).

Again centered on Greenwich, the example below was created by this command:

```
gmt pscoast -Rd -JN4.5i -Bg -Dc -A10000 -Ggoldenrod -Ssnow2 -P > GMT_robinson.ps
```

10.4.5 Eckert IV and VI projection (-Jk -JK)

The Eckert IV and VI projections, presented by the German cartographer Max Eckert-Greiffendorff in 1906, are pseudocylindrical equal-area projections. Central meridian and all parallels are straight lines;

[3] Robinson provided a table of *y*-coordinates for latitudes every 5. To project values for intermediate latitudes one must interpolate the table. Different interpolants may result in slightly different maps. GMT uses the interpolant selected by the parameter *GMT_INTERPOLANT* in the file.

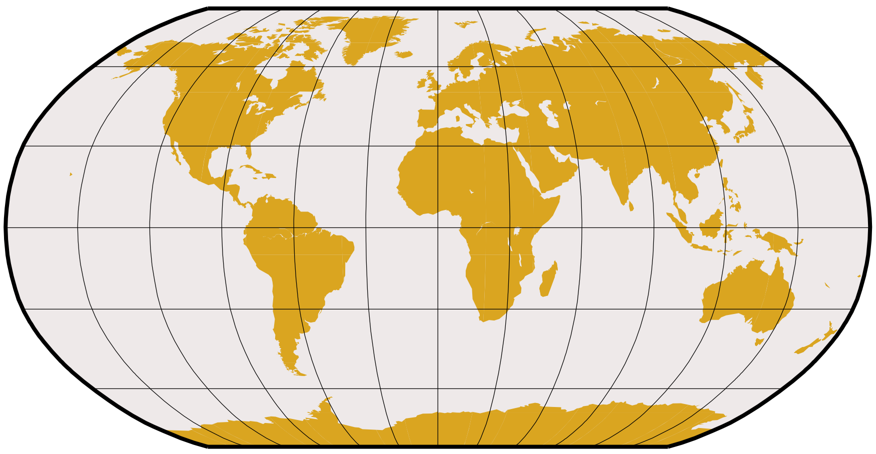


Figure 10.28: World map using the Robinson projection.

other meridians are equally spaced elliptical arcs (IV) or sinusoids (VI). The scale is true along latitudes 4030’ (IV) and 4916’ (VI). Their main use is in thematic world maps. To select Eckert IV you must use **-JKf** (**f** for “four”) while Eckert VI is selected with **-JKs** (**s** for “six”). If no modifier is given it defaults to Eckert VI. In addition, you must enter

- The central meridian [Middle of your map].
- Scale along equator in inch/degree or 1:xxxxx (**-Jk**), or map width (**-JK**).

Centered on the Dateline, the Eckert IV example below was created by this command:

```
gmt pscoast -Rg -JKf4.5i -Bg -Dc -A10000 -Wthinnest -Givory -Sbisque3 -P > GMT_eckert4.ps
```

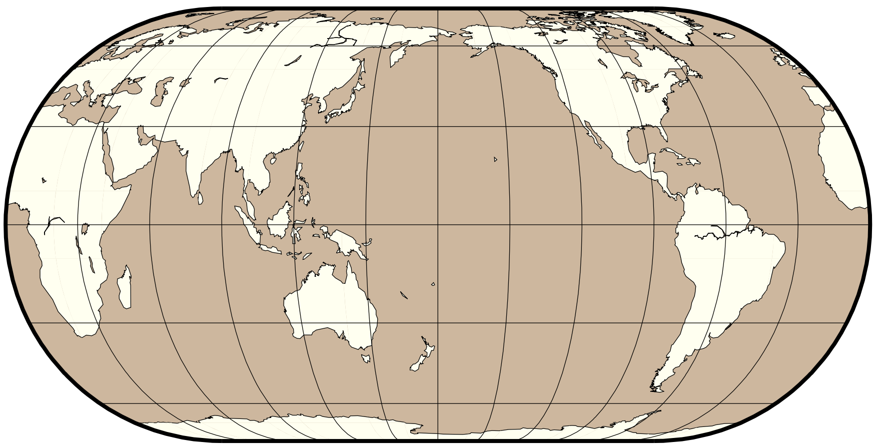


Figure 10.29: World map using the Eckert IV projection.

The same script, with **s** instead of **f**, yields the Eckert VI map:

10.4.6 Sinusoidal projection (-Ji -JI)

The sinusoidal projection is one of the oldest known projections, is equal-area, and has been used since the mid-16th century. It has also been called the “Equal-area Mercator” projection. The central meridian is a straight line; all other meridians are sinusoidal curves. Parallels are all equally spaced straight lines, with scale being true along all parallels (and central meridian). To use it, you need to select:

- The central meridian [Middle of your map].

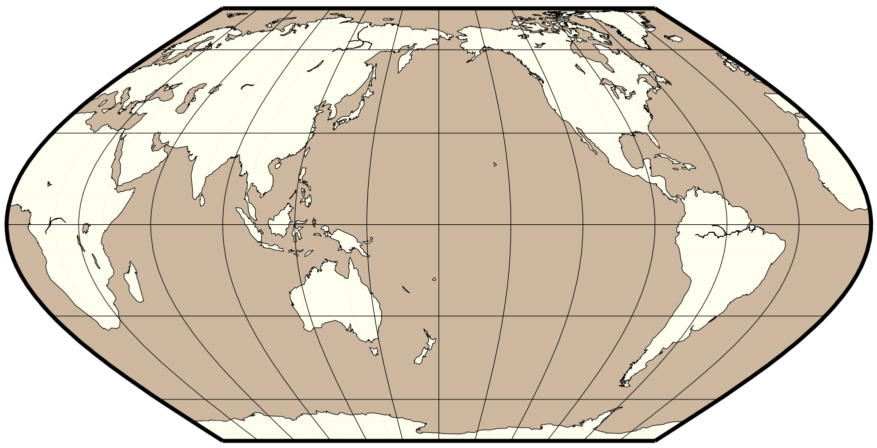


Figure 10.30: World map using the Eckert VI projection.

- Scale along equator in inch/degree or 1:xxxxx (**-Ji**), or map width (**-JI**).

A simple world map using the sinusoidal projection is therefore obtained by

```
gmt pscoast -Rd -JI4.5i -Bxg30 -Byg15 -Dc -A10000 -Ggray -P > GMT_sinusoidal.ps
```

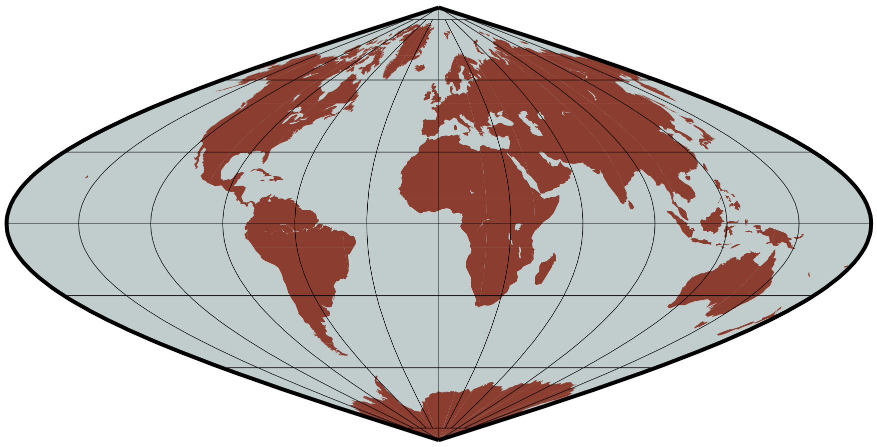


Figure 10.31: World map using the Sinusoidal projection.

To reduce distortion of shape the interrupted sinusoidal projection was introduced in 1927. Here, three symmetrical segments are used to cover the entire world. Traditionally, the interruptions are at 160W, 20W, and 60E. To make the interrupted map we must call `pscoast` for each segment and superpose the results. To produce an interrupted world map (with the traditional boundaries just mentioned) that is 5.04 inches wide we use the scale 5.04/360 = 0.014 and offset the subsequent plots horizontally by their widths (140·0.014 and 80·0.014):

```
gmt pscoast -R200/340/-90/90 -Ji0.014i -Bxg30 -Byg15 -A10000 -Dc \
            -Gblack -K -P > GMT_sinus_int.ps
gmt pscoast -R-20/60/-90/90 -Ji0.014i -Bxg30 -Byg15 -Dc -A10000 \
            -Gblack -X1.96i -O -K >> GMT_sinus_int.ps
gmt pscoast -R60/200/-90/90 -Ji0.014i -Bxg30 -Byg15 -Dc -A10000 \
            -Gblack -X1.12i -O >> GMT_sinus_int.ps
```

The usefulness of the interrupted sinusoidal projection is basically limited to display of global, discontinuous data distributions like hydrocarbon and mineral resources, etc.

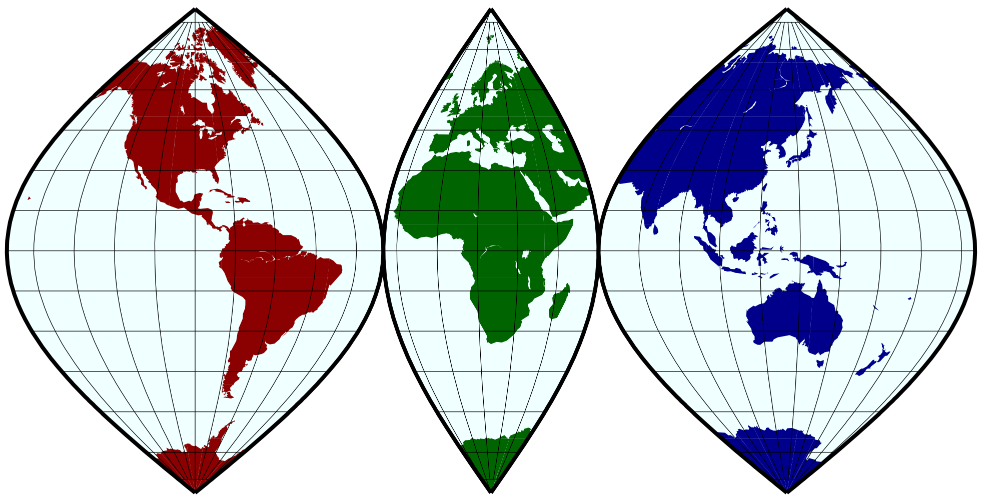


Figure 10.32: World map using the Interrupted Sinusoidal projection.

10.4.7 Van der Grinten projection (-Jv -JV)

The Van der Grinten projection, presented by Alphons J. van der Grinten in 1904, is neither equal-area nor conformal. Central meridian and Equator are straight lines; other meridians are arcs of circles. The scale is true along the Equator only. Its main use is to show the entire world enclosed in a circle. To use it you must enter

- The central meridian [Middle of your map].
- Scale along equator in inch/degree or 1:xxxxx (**-Jv**), or map width (**-JV**).

Centered on the Dateline, the example below was created by this command:

```
gmt pscoast -Rg -JV4i -Bxg30 -Byg15 -Dc -Glightgray -A10000 \
            -Wthinnest -P > GMT_grinten.ps
```

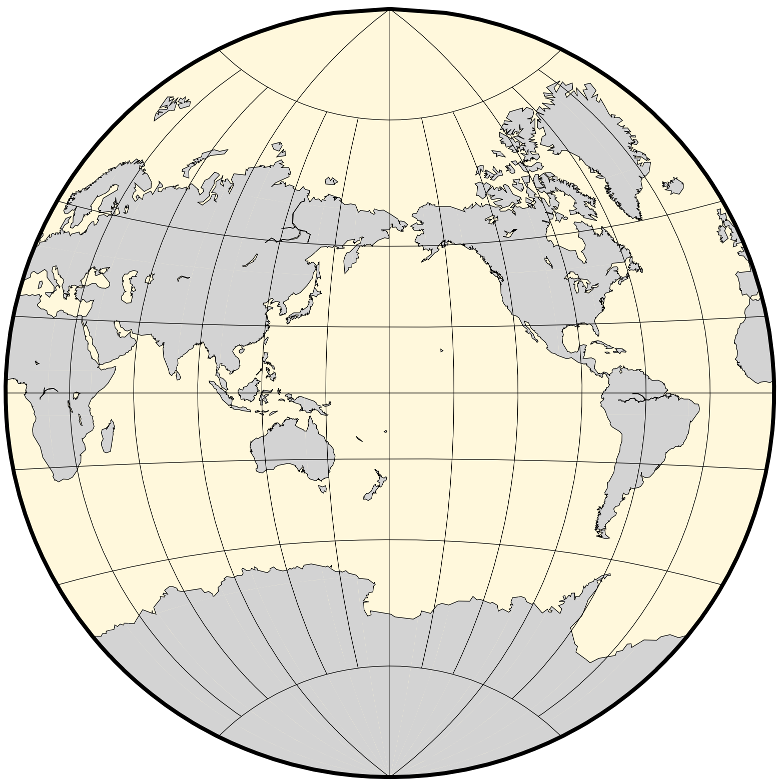


Figure 10.33: World map using the Van der Grinten projection.

CHAPTER 11

Creating GMT Graphics

In this section we will be giving numerous examples of typical usage of GMT programs. In general, we will start with a raw data set, manipulate the numbers in various ways, then display the results in diagram or map view. The resulting plots will have in common that they are all made up of simpler plots that have been overlaid to create a complex illustration. We will mostly follow the following format:

1. We explain what we want to achieve in plain language.
2. We present an annotated Bourne shell script that contains all commands used to generate the illustration.
3. We explain the rationale behind the commands.
4. We present the illustration, 50% reduced in size, and without the timestamp (**-U**).

A detailed discussion of each command is not given; we refer you to the manual pages for command line syntax, etc. We encourage you to run these scripts for yourself. See Appendix [app:D] if you would like an electronic version of all the shell-scripts (both **sh** and **csh** scripts are available, as or DOS batch files; only the **sh**-scripts are discussed here) and support data used below. Note that all examples explicitly specifies the measurement units, so although we use inches you should be able to run these scripts and get the same plots even if you have cm as the default measure unit. The examples are all written to be "quiet", that is no information is echoed to the screen. Thus, these scripts are well suited for background execution.

Note that we also end each script by cleaning up after ourselves. Because there are several AWK implementations such as **gawk** and **nawk**, which are not available everywhere, we refer to `$AWK` in the scripts below. This variable must be set prior to running the example scripts.

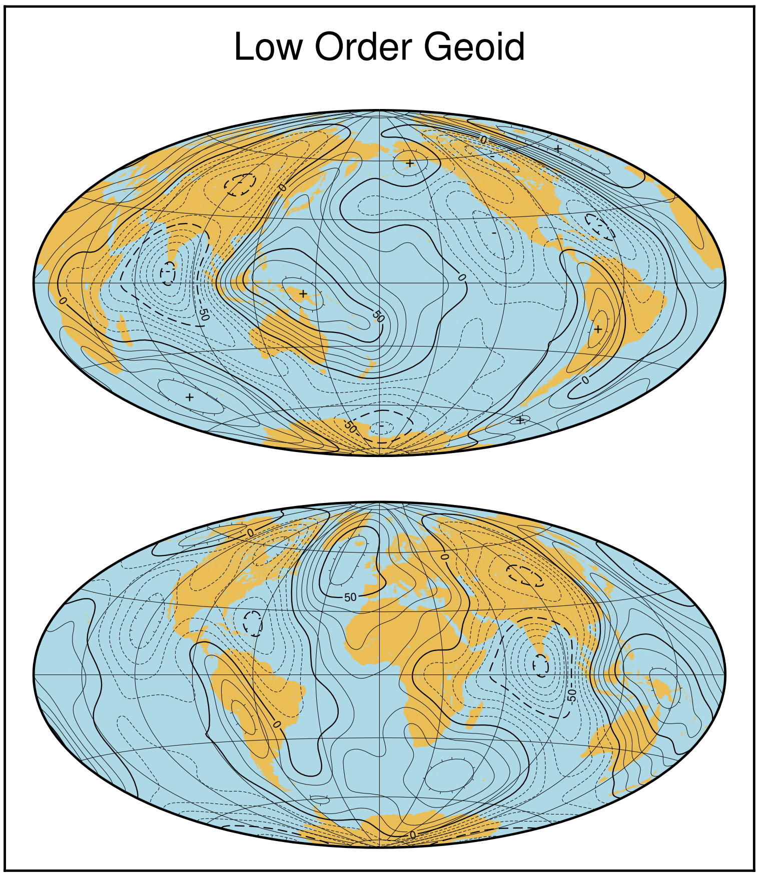
Finally, be aware that for practical purposes the output PostScript file name is stored as the variable `ps`.

11.1 The making of contour maps

We want to create two contour maps of the low order geoid using the Hammer equal area projection. Our gridded data file is called `osu91a1f_16.nc` and contains a global 1 by 1 gridded geoid (we will see how to make gridded files later). We would like to show one map centered on Greenwich and one centered on the dateline. Positive contours should be drawn with a solid pen and negative contours with a dashed pen. Annotations should occur for every 50 m contour level, and both contour maps should show the continents in light brown in the background. Finally, we want a rectangular frame surrounding the two maps. This is how it is done:

```
#!/bin/bash
#               GMT EXAMPLE 01
#
# Purpose:        Make two contour maps based on the data in the file osu91a1f_16.nc
# GMT progs:        gmtset, grdcontour, psbasemap, pscoast
# Unix progs:        rm
#
ps=example_01.ps
gmt gmtset MAP_GRID_CROSS_SIZE_PRIMARY 0 FONT_ANNOT_PRIMARY 10p
gmt psbasemap -R0/6.5/0/7.5 -Jx1i -B0 -P -K > $ps
gmt pscoast -Rg -JH0/6i -X0.25i -Y0.2i -O -K -Bg30 -Dc -Glightbrown -Slightblue >> $ps
gmt grdcontour osu91a1f_16.nc -J -C10 -A50+f7p -Gd4i -L-1000/-1 -Wcthinnest,- -Wathin,- \
            -O -K -T0.1i/0.02i >> $ps
gmt grdcontour osu91a1f_16.nc -J -C10 -A50+f7p -Gd4i -L-1/1000 -O -K -T0.1i/0.02i >> $ps
gmt pscoast -Rg -JH6i -Y3.4i -O -K -B+t"Low Order Geoid" -Bg30 -Dc -Glightbrown \
            -Slightblue >> $ps
gmt grdcontour osu91a1f_16.nc -J -C10 -A50+f7p -Gd4i -L-1000/-1 -Wcthinnest,- -Wathin,- \
            -O -K -T0.1i/0.02i:-+ >> $ps
gmt grdcontour osu91a1f_16.nc -J -C10 -A50+f7p -Gd4i -L-1/1000 -O -T0.1i/0.02i:-+ >> $ps
rm -f gmt.conf
```

The first command draws a box surrounding the maps. This is followed by two sequences of `pscoast`, `grdcontour`, `grdcontour`. They differ in that the first is centered on Greenwich; the second on the dateline. We use the limit option (**-L**) in `grdcontour` to select negative contours only and plot those with a dashed pen, then positive contours only and draw with a solid pen [Default]. The **-T** option causes tick marks pointing in the downhill direction to be drawn on the innermost, closed contours. For the upper panel we also added - and + to the local lows and highs. You can find this illustration as



11.2 Image presentations

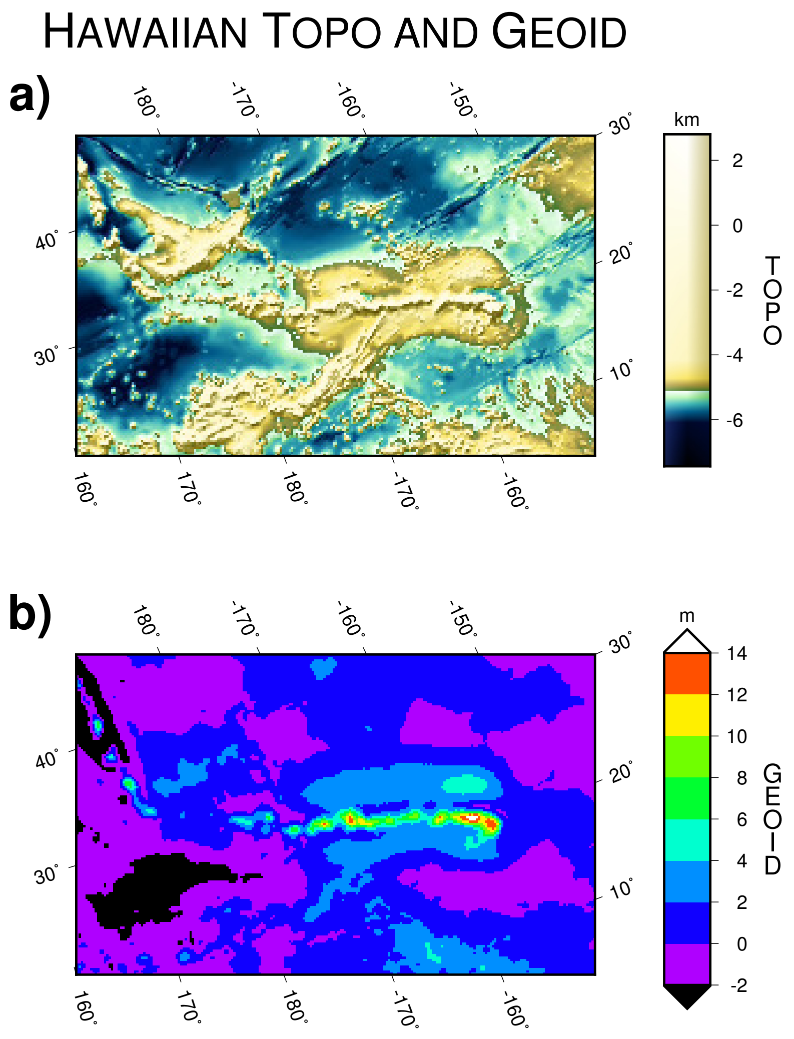
As our second example we will demonstrate how to make color images from gridded data sets (again, we will defer the actual making of grid files to later examples). We will use `grdraster` to extract 2-D grid files of bathymetry and Geosat geoid heights and put the two images on the same page. The region of interest is the Hawaiian islands, and due to the oblique trend of the island chain we prefer to rotate our geographical data sets using an oblique Mercator projection defined by the hotspot pole at (68W, 69N). We choose the point (190, 25.5) to be the center of our projection (e.g., the local origin), and we want to image a rectangular region defined by the longitudes and latitudes of the lower left and upper right corner of region. In our case we choose (160, 20) and (220, 30) as the corners. We use `grdimage` to make the illustration:

```
#!/bin/bash
#                 GMT EXAMPLE 02
#
# Purpose:         Make two color images based gridded data
# GMT progs:         gmtset, grd2cpt, grdgradient, grdimage, makecpt, psscale, pstext
# Unix progs:         rm
#
ps=example_02.ps
gmt gmtset FONT_TITLE 30p MAP_ANNOT_OBLIQUE 0
gmt makecpt -Crainbow -T-2/14/2 > g.cpt
gmt grdimage HI_geoid2.nc -R160/20/220/30r -JOc190/25.5/292/69/4.5i -E50 -K -P \
            -B10 -Cg.cpt -X1.5i -Y1.25i > $ps
gmt psscale -Cg.cpt -D5.1i/1.35i/2.88i/0.4i -O -K -Ac -Bx2+lGEOID -By+lm -E >> $ps
gmt grd2cpt HI_topo2.nc -Crelief -Z > t.cpt
gmt grdgradient HI_topo2.nc -A0 -Nt -GHI_topo2_int.nc
gmt grdimage HI_topo2.nc -IHI_topo2_int.nc -R -J -B+t"H@#awaiian@# T@#opo and @#G@#eoid" \
            -B10 -E50 -O -K -Ct.cpt -Y4.5i --MAP_TITLE_OFFSET=0.5i >> $ps
gmt psscale -Ct.cpt -D5.1i/1.35i/2.88i/0.4i -O -K -I0.3 -Ac -Bx2+lTOPO -By+lkm >> $ps
gmt pstext -R0/8.5/0/11 -Jx1i -F+f30p,Helvetica-Bold+jCB -O -N -Y-4.5i >> $ps << END
-0.4 7.5 a)
-0.4 3.0 b)
END
rm -f HI_topo2_int.nc ?.cpt gmt.conf
```

The first step extracts the 2-D data sets from the local data base using `grdraster` that may be adapted to reflect the nature of your data base format. It automatically figures out the required extent of the region given the two corners points and the projection. The extreme meridians and parallels enclosing the oblique region is **-R**159:50/220:10/3:10/47:35. This is the area extracted by `grdraster`. For your convenience we have commented out those lines and provided the two extracted files so you do not need `grdraster` to try this example. By using the embedded grid file format mechanism we saved the topography using kilometers as the data unit. We now have two grid files with bathymetry and geoid heights, respectively. We use `makecpt` to generate a linear color palette file `geoid.cpt` for the geoid and use `grd2cpt` to get a histogram-equalized cpt file `topo.cpt` for the topography data. To emphasize the structures in the data we calculate the slopes in the north-south direction using `grdgradient`; these will be used to modulate the color image. Next we run `grdimage` to create a color-code image of the Geosat geoid heights, and draw a color legend to the right of the image with `psscale`. Similarly, we run `grdimage` but specify **-Y**4.5i to plot above the previous image. Adding scale and label the two plots a) and b) completes the illustration.

11.3 Spectral estimation and xy-plots

In this example we will show how to use the GMT programs `fitcircle`, `project`, `sample1d`, `spectrum1d`, `psxy`, and `pstext`. Suppose you have (lon, lat, gravity) along a satellite track in a file called `sat.xyg`, and (lon, lat, gravity) along a ship track in a file called `ship.xyg`. You want to make a cross-spectral analysis of these data. First, you will have to get the two data sets into equidistantly


HAWAIIAN TOPO AND GEOID
a)
180°
-170°
-160°
-150°
30°
20°
10°
40°
30°
160°
170°
180°
-170°
-160°
km
2
0
-2
-4
-6
TOPO
b)
180°
-170°
-160°
-150°
30°
20°
10°
40°
30°
160°
170°
180°
-170°
-160°
m
14
12
10
8
6
4
2
0
-2
GEOID

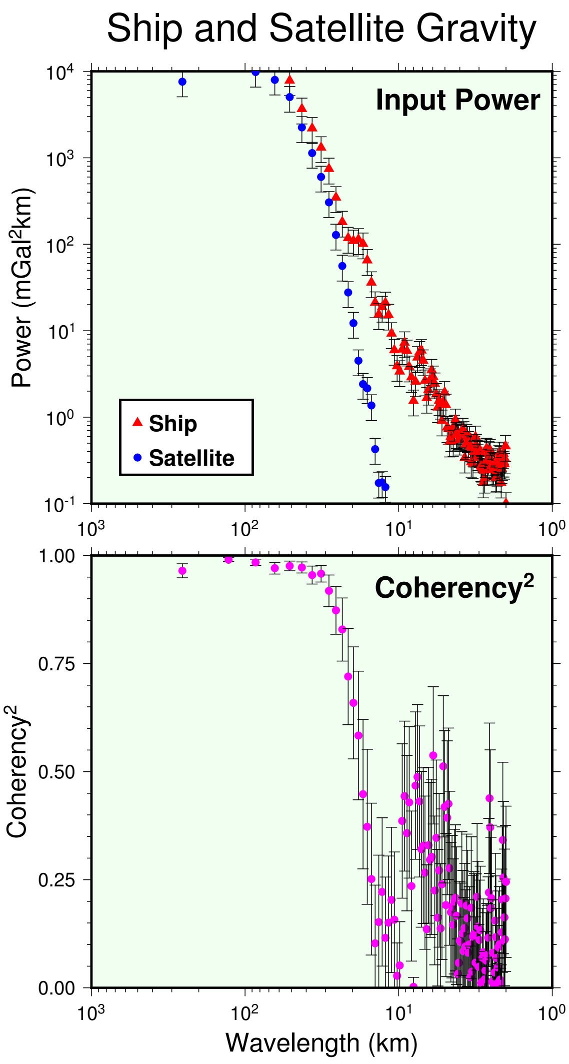
sampled time-series form. To do this, it will be convenient to project these along the great circle that best fits the sat track. We must use `fitcircle` to find this great circle and choose the L_2 estimates of best pole. We project the data using `project` to find out what their ranges are in the projected coordinate. The `gmtinfo` utility will report the minimum and maximum values for multi-column ASCII tables. Use this information to select the range of the projected distance coordinate they have in common. The script prompts you for that information after reporting the values. We decide to make a file of equidistant sampling points spaced 1 km apart from -1167 to +1169, and use the UNIX utility **awk** to accomplish this step. We can then resample the projected data, and carry out the cross-spectral calculations, assuming that the ship is the input and the satellite is the output data. There are several intermediate steps that produce helpful plots showing the effect of the various processing steps (`example_03[a-f].ps`), while the final plot `example_03.ps` shows the ship and sat power in one diagram and the coherency on another diagram, both on the same page. Note the extended use of `pstext` and `psxy` to put labels and legends directly on the plots. For that purpose we often use **-Jx**1i and specify positions in inches directly. Thus, the complete automated script reads:

```
#!/bin/bash
#               GMT EXAMPLE 03
#
# Purpose:        Resample track data, do spectral analysis, and plot
# GMT progs:        filter1d, fitcircle, gmtinfo, project, sample1d
# GMT progs:        spectrum1d, trend1d, pshistogram, psxy, pstext
# Unix progs:        $AWK, cat, echo, head, paste, rm, tail
#
# This example begins with data files "ship.xyg" and "sat.xyg" which
# are measurements of a quantity "g" (a "gravity anomaly" which is an
# anomalous increase or decrease in the magnitude of the acceleration
# of gravity at sea level).  g is measured at a sequence of points "x,y"
# which in this case are "longitude,latitude".  The "sat.xyg" data were
# obtained by a satellite and the sequence of points lies almost along
# a great circle.  The "ship.xyg" data were obtained by a ship which
# tried to follow the satellite's path but deviated from it in places.
# Thus the two data sets are not measured at the same of points,
# and we use various GMT tools to facilitate their comparison.
# The main illustration (example_03.ps) are accompanied with 5 support
# plots (03a-f) showing data distributions and various intermediate steps.
#
# First, we use "gmt fitcircle" to find the parameters of a great circle
# most closely fitting the x,y points in "sat.xyg":
#
ps=example_03.ps
gmt fitcircle sat.xyg -L2 > report
cposx=`grep "L2 Average Position" report | cut -f1`
cposy=`grep "L2 Average Position" report | cut -f2`
pposx=`grep "L2 N Hemisphere" report | cut -f1`
pposy=`grep "L2 N Hemisphere" report | cut -f2`
#
# Now we use "gmt project" to gmt project the data in both sat.xyg and ship.xyg
# into data.pg, where g is the same and p is the oblique longitude around
# the great circle.  We use -Q to get the p distance in kilometers, and -S
# to sort the output into increasing p values.
#
gmt project  sat.xyg -C$cposx/$cposy -T$pposx/$pposy -S -Fpz -Q > sat.pg
gmt project ship.xyg -C$cposx/$cposy -T$pposx/$pposy -S -Fpz -Q > ship.pg
#
# The gmtinfo utility will report the minimum and maximum values for all columns.
# We use this information first with a large -I value to find the appropriate -R
# to use to plot the .pg data.
#
R=`cat sat.pg ship.pg | gmt info -I100/25`
gmt psxy $R -UL/-1.75i/-1.25i/"Example 3a in Cookbook" -BWeSn \
        -Bxa500f100+l"Distance along great circle" -Bya100f25+l"Gravity anomaly (mGal)" \
        -JX8i/5i -X2i -Y1.5i -K -Wthick sat.pg > example_03a.ps
gmt psxy -R -JX -O -Sp0.03i ship.pg >> example_03a.ps
#
# From this plot we see that the ship data have some "spikes" and also greatly
# differ from the satellite data at a point about p ~= +250 km, where both of
# them show a very large anomaly.
```

```
#
# To facilitate comparison of the two with a cross-spectral analysis using "gmt spectrum1d",
# we resample both data sets at intervals of 1 km.  First we find out how the data are
# typically spaced using $AWK to get the delta-p between points and view it with
# "gmt pshistogram".
#
$AWK '{ if (NR > 1) print $1 - last1; last1=$1; }' ship.pg | gmt pshistogram  -W0.1 -Gblack \
        -JX3i -K -X2i -Y1.5i -B0 -B+t"Ship" -UL/-1.75i/-1.25i/"Example 3b in Cookbook" \
        > example_03b.ps
$AWK '{ if (NR > 1) print $1 - last1; last1=$1; }' sat.pg  | gmt pshistogram  -W0.1 -Gblack \
        -JX3i -O -X5i -B0 -B+t"Sat" >> example_03b.ps
#
# This experience shows that the satellite values are spaced fairly evenly, with
# delta-p between 3.222 and 3.418.  The ship values are spaced quite unevenly, with
# delta-p between 0.095 and 9.017.  This means that when we want 1 km even sampling,
# we can use "gmt sample1d" to interpolate the sat data, but the same procedure applied
# to the ship data could alias information at shorter wavelengths.  So we have to use
# "gmt filter1d" to resample the ship data.  Also, since we observed spikes in the ship
# data, we use a median filter to clean up the ship values.  We will want to use "paste"
# to put the two sampled data sets together, so they must start and end at the same
# point, without NaNs.  So we want to get a starting and ending point which works for
# both of them.  This is a job for gmt gmtmath UPPER/LOWER.
#
head -1 ship.pg > tmp
head -1 sat.pg >> tmp
sampr1=`gmt gmtmath tmp -Ca -Sf -o0 UPPER CEIL =`
tail -1 ship.pg > tmp
tail -1 sat.pg >> tmp
sampr2=`gmt gmtmath tmp -Ca -Sf -o0 LOWER FLOOR =`
#
# Now we can use sampr1|2 in gmt gmtmath to make a sampling points file for gmt sample1d:
gmt gmtmath -T$sampr1/$sampr2/1 -N1/0 T = samp.x
#
# Now we can resample the gmt projected satellite data:
#
gmt sample1d sat.pg -Nsamp.x > samp_sat.pg
#
# For reasons above, we use gmt filter1d to pre-treat the ship data.  We also need to sample
# it because of the gaps > 1 km we found.  So we use gmt filter1d | gmt sample1d.  We also
# use the -E on gmt filter1d to use the data all the way out to sampr1/sampr2 :
#
gmt filter1d ship.pg -Fm1 -T$sampr1/$sampr2/1 -E | gmt sample1d -Nsamp.x > samp_ship.pg
#
# Now we plot them again to see if we have done the right thing:
#
gmt psxy $R -JX8i/5i -X2i -Y1.5i -K -Wthick samp_sat.pg \
        -Bxa500f100+l"Distance along great circle" -Bya100f25+l"Gravity anomaly (mGal)" \
        -BWeSn -UL/-1.75i/-1.25i/"Example 3c in Cookbook" > example_03c.ps
gmt psxy -R -JX -O -Sp0.03i samp_ship.pg >> example_03c.ps
#
# Now to do the cross-spectra, assuming that the ship is the input and the sat is the output
# data, we do this:
#
gmt gmtconvert -A samp_ship.pg samp_sat.pg -o1,3 | gmt spectrum1d -S256 -D1 -W -C > /dev/null
#
# Now we want to plot the spectra. The following commands will plot the ship and sat
# power in one diagram and the coherency on another diagram, both on the same page.
# Note the extended use of gmt pstext and gmt psxy to put labels and legends directly on the
# plots. For that purpose we often use -Jx1i and specify positions in inches directly:
#
gmt psxy spectrum.coh -Bxa1f3p+l"Wavelength (km)" -Bya0.25f0.05+l"Coherency@+2@+" \
        -BWeSn+g240/255/240 -JX-4il/3.75i -R1/1000/0/1 -P -K -X2.5i -Sc0.07i -Gmagenta \
        -Ey/0.5p -Y1.5i > $ps
echo "3.85 3.6 Coherency@+2@+" | gmt pstext -R0/4/0/3.75 -Jx1i -F+f18p,Helvetica-Bold+jTR \
        -O -K >> $ps
gmt psxy spectrum.xpower -Bxa1f3p -Bya1f3p+l"Power (mGal@+2@+km)" \
        -BWeSn+t"Ship and Satellite Gravity"+g240/255/240 \
        -Gred -ST0.07i -O -R1/1000/0.1/10000 -JX-4il/3.75il -Y4.2i -K -Ey/0.5p >> $ps
gmt psxy spectrum.ypower -R -JX -O -K -Gblue -Sc0.07i -Ey/0.5p >> $ps
echo "3.9 3.6 Input Power" | gmt pstext -R0/4/0/3.75 -Jx -F+f18p,Helvetica-Bold+jTR -O -K >> $ps
gmt psxy -R -Jx -O -K -Gwhite -L -Wthicker >> $ps << END
0.25        0.25
```

```
1.4         0.25
1.4         0.9
0.25         0.9
END
echo "0.4 0.7" | gmt psxy -R -Jx -O -K -ST0.07i -Gred >> $ps
echo "0.5 0.7 Ship" | gmt pstext -R -Jx -F+f14p,Helvetica-Bold+jLM -O -K >> $ps
echo "0.4 0.4" | gmt psxy -R -Jx -O -K -Sc0.07i -Gblue >> $ps
echo "0.5 0.4 Satellite" | gmt pstext -R -Jx -F+f14p,Helvetica-Bold+jLM -O >> $ps
#
# Now we wonder if removing that large feature at 250 km would make any difference.
# We could throw away a section of data with $AWK or sed or head and tail, but we
# demonstrate the use of "gmt trend1d" to identify outliers instead.  We will fit a
# straight line to the samp_ship.pg data by an iteratively-reweighted method and
# save the weights on output.  Then we will plot the weights and see how things
# look:
#
gmt trend1d -Fxw -N2r samp_ship.pg > samp_ship.xw
gmt psxy $R -JX8i/4i -X2i -Y1.5i -K -Sp0.03i \
        -Bxa500f100+l"Distance along great circle" -Bya100f25+l"Gravity anomaly (mGal)" \
        -BWeSn -UL/-1.75i/-1.25i/"Example 3d in Cookbook" samp_ship.pg > example_03d.ps
R=`gmt info samp_ship.xw -I100/1.1`
gmt psxy $R -JX8i/1.1i -O -Y4.25i -Bxf100 -Bya0.5f0.1+l"Weight" -BWesn -Sp0.03i \
        samp_ship.xw >> example_03d.ps
#
# From this we see that we might want to throw away values where w < 0.6.  So we try that,
# and this time we also use gmt trend1d to return the residual from the model fit (the
# de-trended data):
gmt trend1d -Fxrw -N2r samp_ship.pg | $AWK '{ if ($3 > 0.6) print $1, $2 }' \
        | gmt sample1d -Nsamp.x > samp2_ship.pg
gmt trend1d -Fxrw -N2r samp_sat.pg  | $AWK '{ if ($3 > 0.6) print $1, $2 }' \
        | gmt sample1d -Nsamp.x > samp2_sat.pg
#
# We plot these to see how they look:
#
R=`cat samp2_sat.pg samp2_ship.pg | gmt info -I100/25`
gmt psxy $R -JX8i/5i -X2i -Y1.5i -K -Wthick \
        -Bxa500f100+l"Distance along great circle" -Bya50f25+l"Gravity anomaly (mGal)" \
        -BWeSn -UL/-1.75i/-1.25i/"Example 3e in Cookbook" samp2_sat.pg > example_03e.ps
gmt psxy -R -JX -O -Sp0.03i samp2_ship.pg >> example_03e.ps
#
# Now we do the cross-spectral analysis again.  Comparing this plot (example_03e.ps) with
# the previous one (example_03d.ps) we see that throwing out the large feature has reduced
# the power in both data sets and reduced the coherency at wavelengths between 20--60 km.
#
gmt gmtconvert -A samp2_ship.pg samp2_sat.pg -o1,3 | gmt spectrum1d -S256 -D1 -W -C > /dev/null
#
gmt psxy spectrum.coh -Bxa1f3p+l"Wavelength (km)" -Bya0.25f0.05+l"Coherency@+2@+" -BWeSn \
        -JX-4il/3.75i -R1/1000/0/1 -UL/-2.25i/-1.25i/"Example 3f in Cookbook" -P -K -X2.5i \
        -Sc0.07i -Gblack -Ey/0.5p -Y1.5i > example_03f.ps
echo "3.85 3.6 Coherency@+2@+" | gmt pstext -R0/4/0/3.75 -Jx -F+f18p,Helvetica-Bold+jTR -O \
        -K >> example_03f.ps
cat > box.d << END
2.375         3.75
2.375         3.25
4         3.25
END
gmt psxy -R -Jx -O -K -Wthicker box.d >> example_03f.ps
gmt psxy -Bxa1f3p -Bya1f3p+l"Power (mGal@+2@+km)" -BWeSn+t"Ship and Satellite Gravity" \
        spectrum.xpower -ST0.07i -O -R1/1000/0.1/10000 -JX-4il/3.75il -Y4.2i -K -Ey/0.5p \
        >> example_03f.ps
gmt psxy spectrum.ypower -R -JX -O -K -Gblack -Sc0.07i -Ey/0.5p >> example_03f.ps
echo "3.9 3.6 Input Power" | gmt pstext -R0/4/0/3.75 -Jx -F+f18p,Helvetica-Bold+jTR -O \
        -K >> example_03f.ps
gmt psxy -R -Jx -O -K -Wthicker box.d >> example_03f.ps
gmt psxy -R -Jx -O -K -Glightgray -L -Wthicker >> example_03f.ps << END
0.25         0.25
1.4         0.25
1.4         0.9
0.25         0.9
END
echo "0.4 0.7" | gmt psxy -R -Jx -O -K -ST0.07i -Gblack >> example_03f.ps
echo "0.5 0.7 Ship" | gmt pstext -R -Jx -F+f14p,Helvetica-Bold+jLM -O -K >> example_03f.ps
echo "0.4 0.4" | gmt psxy -R -Jx -O -K -Sc0.07i -Gblack >> example_03f.ps
echo "0.5 0.4 Satellite" | gmt pstext -R -Jx -F+f14p,Helvetica-Bold+jLM -O >> example_03f.ps
#
rm -f box.d report tmp samp* *.pg *.extr spectrum.*
```

The final illustration (Figure *Example 03*) shows that the ship gravity anomalies have more power than altimetry derived gravity for short wavelengths and that the coherency between the two signals improves dramatically for wavelengths > 20 km.



11.4 A 3-D perspective mesh plot

This example will illustrate how to make a fairly complicated composite figure. We need a subset of the ETOPO5 bathymetry [1] and Geosat geoid data sets which we will extract from the local data bases using `grdraster`. We would like to show a 2-layer perspective plot where layer one shows a contour map of the marine geoid with the location of the Hawaiian islands superposed, and a second layer showing the 3-D mesh plot of the topography. We also add an arrow pointing north and some text. The first part of this script shows how to do it:

[1] These data are available on CD-ROM from NGDC (http://www.ngdc.noaa.gov/).

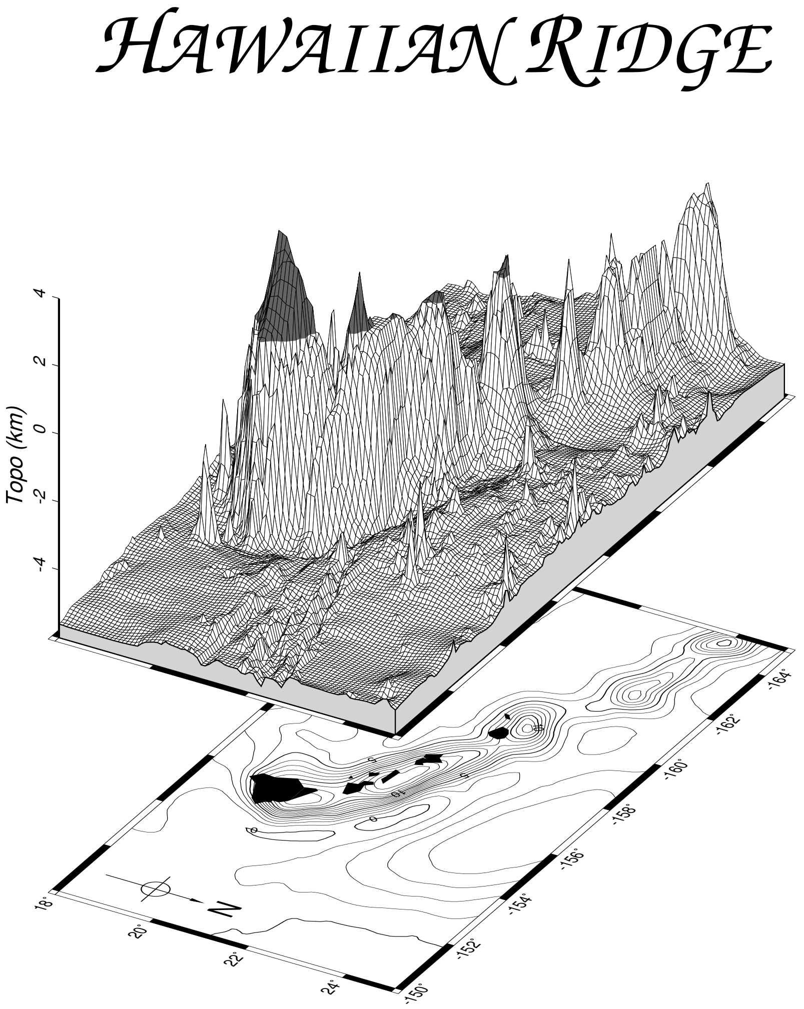
```
#!/bin/bash
#               GMT EXAMPLE 04
#
# Purpose:        3-D mesh and color plot of Hawaiian topography and geoid
# GMT progs:        grdcontour, grdgradient, grdimage, grdview, psbasemap, pscoast, pstext
# Unix progs:        echo, rm
#
ps=example_04.ps
echo '-10  255   0  255' > zero.cpt
echo '  0  100  10  100' >> zero.cpt
gmt grdcontour HI_geoid4.nc -R195/210/18/25 -Jm0.45i -p60/30 -C1 -A5+o -Gd4i -K -P \
        -X1.25i -Y1.25i > $ps
gmt pscoast -R -J -p -B2 -BNEsw -Gblack -O -K -T209/19.5/1i >> $ps
gmt grdview HI_topo4.nc -R195/210/18/25/-6/4 -J -Jz0.34i -p -Czero.cpt -O -K \
        -N-6+glightgray -Qsm -B2 -Bz2+l"Topo (km)" -BneswZ -Y2.2i >> $ps
echo '3.25 5.75 H@#awaiian@# R@#idge' | gmt pstext -R0/10/0/10 -Jx1i \
        -F+f60p,ZapfChancery-MediumItalic+jCB -O >> $ps
rm -f zero.cpt
#
ps=example_04c.ps
gmt grdgradient HI_geoid4.nc -A0 -Gg_intens.nc -Nt0.75 -fg
gmt grdgradient HI_topo4.nc -A0 -Gt_intens.nc -Nt0.75 -fg
gmt grdimage HI_geoid4.nc -Ig_intens.nc -R195/210/18/25 -JM6.75i -p60/30 -Cgeoid.cpt -E100 \
        -K -P -X1.25i -Y1.25i > $ps
gmt pscoast -R -J -p -B2 -BNEsw -Gblack -O -K >> $ps
gmt psbasemap -R -J -p -O -K -T209/19.5/1i --COLOR_BACKGROUND=red --FONT=red \
        --MAP_TICK_PEN_PRIMARY=thinner,red >> $ps
gmt psscale -R -J -p240/30 -D3.375i/-0.5i/5i/0.3ih -Cgeoid.cpt -I -O -K -Bx2+l"Geoid (m)" >> $ps
gmt grdview HI_topo4.nc -It_intens.nc -R195/210/18/25/-6/4 -J -JZ3.4i -p60/30 -Ctopo.cpt \
        -O -K -N-6+glightgray -Qc100 -B2 -Bz2+l"Topo (km)" -BneswZ -Y2.2i >> $ps
echo '3.25 5.75 H@#awaiian@# R@#idge' | gmt pstext -R0/10/0/10 -Jx1i \
        -F+f60p,ZapfChancery-MediumItalic+jCB -O >> $ps
rm -f *_intens.nc
```

The purpose of the color palette file `zero.cpt` is to have the positive topography mesh painted light gray (the remainder is white). The left side of Figure shows the complete illustration.

The second part of the script shows how to make the color version of this figure that was printed in our first article in *EOS Trans. AGU* (8 October 1991). Using `grdview` one can choose to either plot a mesh surface (left) or a color-coded surface (right). We have also added artificial illumination from a light-source due north, which is simulated by computing the gradient of the surface grid in that direction though the `grdgradient` program. We choose to use the **-Qc** option in `grdview` to achieve a high degree of smoothness. Here, we select 100 dpi since that will be the resolution of our final raster (The EOS raster was 300 dpi). Note that the size of the resulting output file is directly dependent on the square of the dpi chosen for the scanline conversion and how well the resulting image compresses. A higher value for dpi in **-Qc** would have resulted in a much larger output file. The CPT files were taken from Section [sec:example$_0$2].

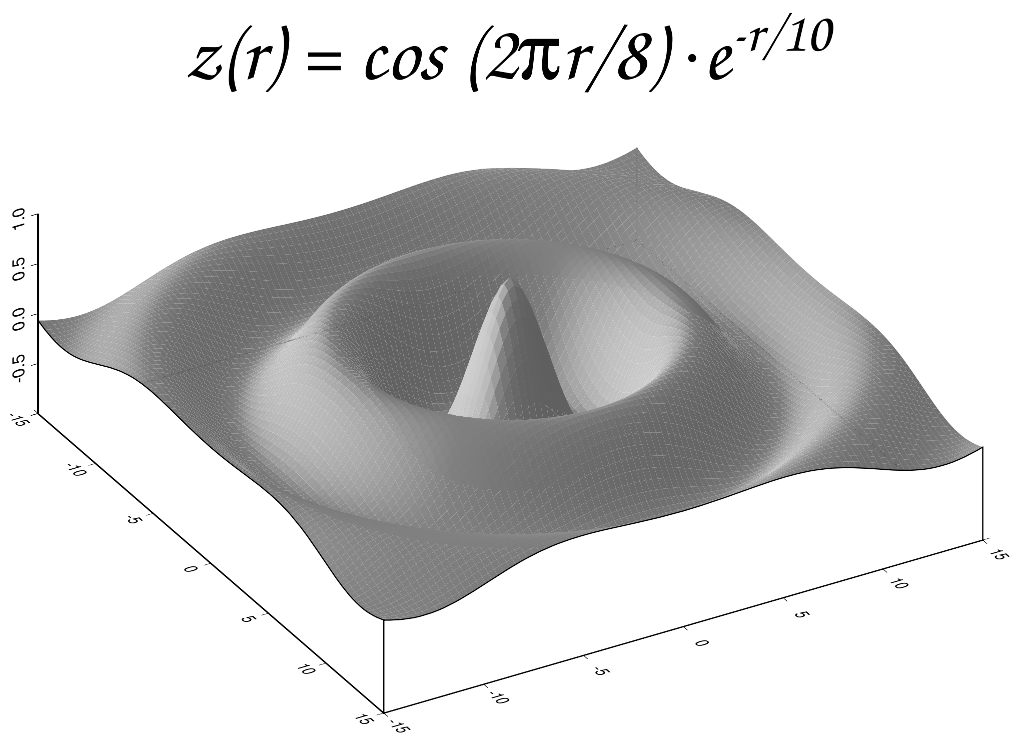
11.5 A 3-D illuminated surface in black and white

Instead of a mesh plot we may choose to show 3-D surfaces using artificial illumination. For this example we will use `grdmath` to make a grid file that contains the surface given by the function $z(x, y) = \cos(2\pi r/8) \cdot e^{-r/10}$, where $r^2 = (x^2 + y^2)$. The illumination is obtained by passing two grid files to `grdview`: One with the z-values (the surface) and another with intensity values (which should be in the 1 range). We use `grdgradient` to compute the horizontal gradients in the direction of the artificial light source. The `gray.cpt` file only has one line that states that all z values should have the gray level 128. Thus, variations in shade are entirely due to variations in gradients, or illuminations. We choose to illuminate from the SW and view the surface from SE:


HAWAIIAN RIDGE
Topo (km)
4
2
0
-2
-4
N
18°
20°
22°
24°
-150°
-152°
-154°
-156°
-158°
-160°
-162°
-164°

```
#!/bin/bash
#               GMT EXAMPLE 05
#
# Purpose:        Generate grid and show monochrome 3-D perspective
# GMT progs:        grdgradient, grdmath, grdview, pstext
# Unix progs:         echo, rm
#
ps=example_05.ps
gmt grdmath -R-15/15/-15/15 -I0.3 X Y HYPOT DUP 2 MUL PI MUL 8 DIV COS EXCH NEG 10 DIV \
        EXP MUL = sombrero.nc
echo '-5 128 5 128' > gray.cpt
gmt grdgradient sombrero.nc -A225 -Gintensity.nc -Nt0.75
gmt grdview sombrero.nc -JX6i -JZ2i -B5 -Bz0.5 -BSEwnZ -N-1+gwhite -Qs -Iintensity.nc -X1.5i \
        -Cgray.cpt -R-15/15/-15/15/-1/1 -K -p120/30 > $ps
echo "4.1 5.5 z(r) = cos (2@~p@~r/8) @~\327@~e@+-r/10@+" | gmt pstext -R0/11/0/8.5 -Jx1i \
        -F+f50p,ZapfChancery-MediumItalic+jBC -O >> $ps
rm -f gray.cpt sombrero.nc intensity.nc
```

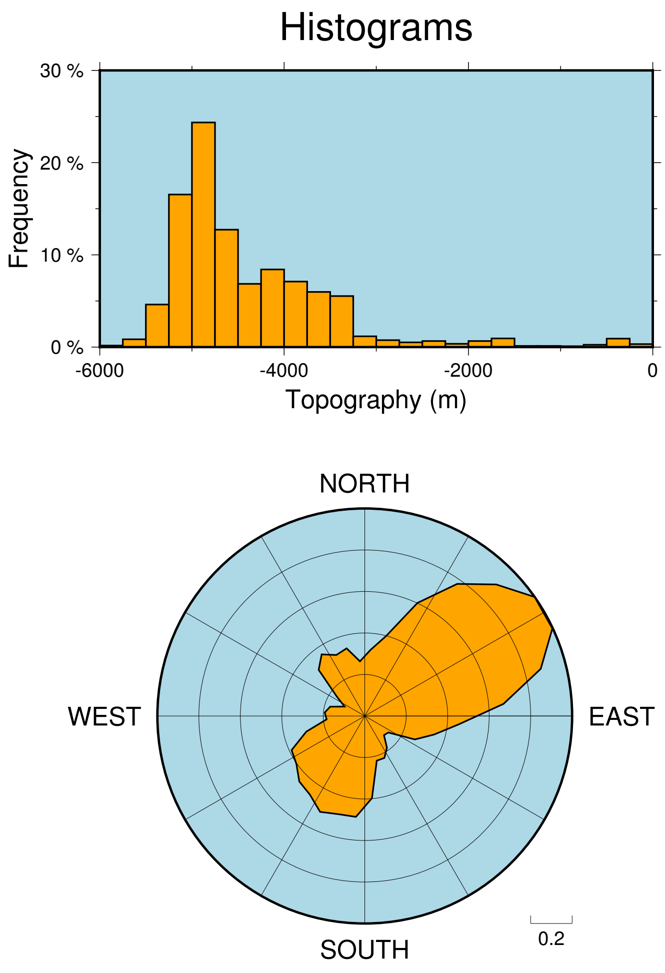
The variations in intensity could be made more dramatic by using `grdmath` to scale the intensity file before running `grdview`. For very rough data sets one may improve the smoothness of the intensities by passing the output of `grdgradient` to `grdhisteq`. The shell-script above will result in a plot like the one in Figure *Example 05*.



11.6 Plotting of histograms

GMT provides two tools to render histograms: `pshistogram` and `psrose`. The former takes care of regular histograms whereas the latter deals with polar histograms (rose diagrams, sector diagrams, and wind rose diagrams). We will show an example that involves both programs. The file `fractures.yx` contains a compilation of fracture lengths and directions as digitized from geological maps. The file `v3206.t` contains all the bathymetry measurements from *Vema* cruise 3206. Our complete figure (Figure *Example 06*) was made running this script:

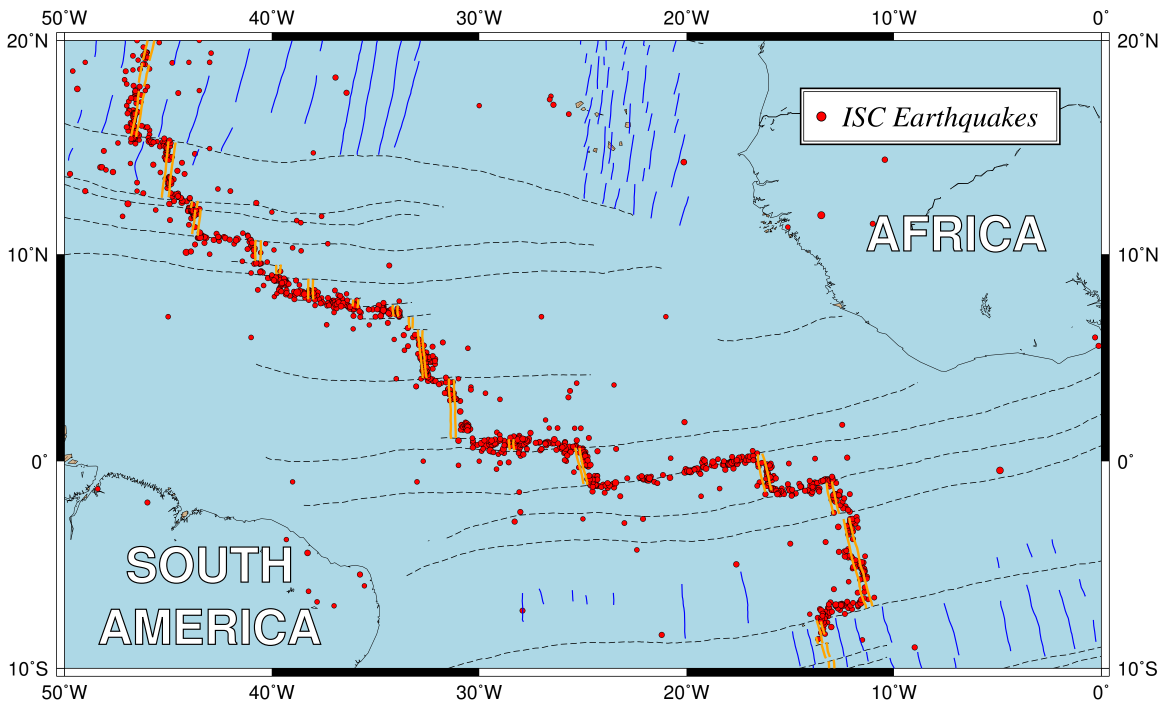
```
#!/bin/bash
#               GMT EXAMPLE 06
#
# Purpose:        Make standard and polar histograms
# GMT progs:        pshistogram, psrose
# Unix progs:        rm
#
ps=example_06.ps
gmt psrose fractures.d -: -A10r -S1.8in -P -Gorange -R0/1/0/360 -X2.5i -K -Bx0.2g0.2 \
        -By30g30 -B+glightblue -W1p > $ps
gmt pshistogram -Bxa2000f1000+l"Topography (m)" -Bya10f5+l"Frequency"+u" %" \
        -BWSne+t"Histograms"+glightblue v3206.t -R-6000/0/0/30 -JX4.8i/2.4i -Gorange -O \
        -Y5.0i -X-0.5i -L1p -Z1 -W250 >> $ps
```



11.7 A simple location map

Many scientific papers start out by showing a location map of the region of interest. This map will typically also contain certain features and labels. This example will present a location map for the equatorial Atlantic ocean, where fracture zones and mid-ocean ridge segments have been plotted. We also would like to plot earthquake locations and available isochrons. We have obtained one file, `quakes.xym`, which contains the position and magnitude of available earthquakes in the region. We choose to use magnitude/100 for the symbol-size in inches. The digital fracture zone traces (`fz.xy`) and isochrons (0 isochron as `ridge.xy`, the rest as `isochrons.xy`) were digitized from available maps [23]. We create the final location map (Figure *Example 07*) with the following script:

```
#!/bin/bash
#               GMT EXAMPLE 07
#
# Purpose:        Make a basemap with earthquakes and isochrons etc
# GMT progs:        pscoast, pstext, psxy
# Unix progs:        echo, rm
#
ps=example_07.ps
gmt pscoast -R-50/0/-10/20 -JM9i -K -Slightblue -GP300/26:FtanBdarkbrown -Dl -Wthinnest \
        -B10 --FORMAT_GEO_MAP=dddF > $ps
gmt psxy -R -J -O -K fz.xy -Wthinner,- >> $ps
gmt psxy quakes.xym -R -J -O -K -h1 -Sci -i0,1,2s0.01 -Gred -Wthinnest >> $ps
gmt psxy -R -J -O -K isochron.xy -Wthin,blue >> $ps
gmt psxy -R -J -O -K ridge.xy -Wthicker,orange >> $ps
gmt psxy -R -J -O -K -Gwhite -Wthick -A >> $ps << END
-14.5        15.2
 -2        15.2
 -2        17.8
-14.5        17.8
END
gmt psxy -R -J -O -K -Gwhite -Wthinner -A >> $ps << END
-14.35        15.35
 -2.15        15.35
 -2.15        17.65
-14.35        17.65
END
echo "-13.5 16.5" | gmt psxy -R -J -O -K -Sc0.08i -Gred -Wthinner >> $ps
echo "-12.5 16.5 ISC Earthquakes" | gmt pstext -R -J -F+f18p,Times-Italic+jLM -O -K >> $ps
gmt pstext -R -J -O -F+f30,Helvetica-Bold,white=thin >> $ps << END
-43 -5 SOUTH
-43 -8 AMERICA
 -7 11 AFRICA
END
```

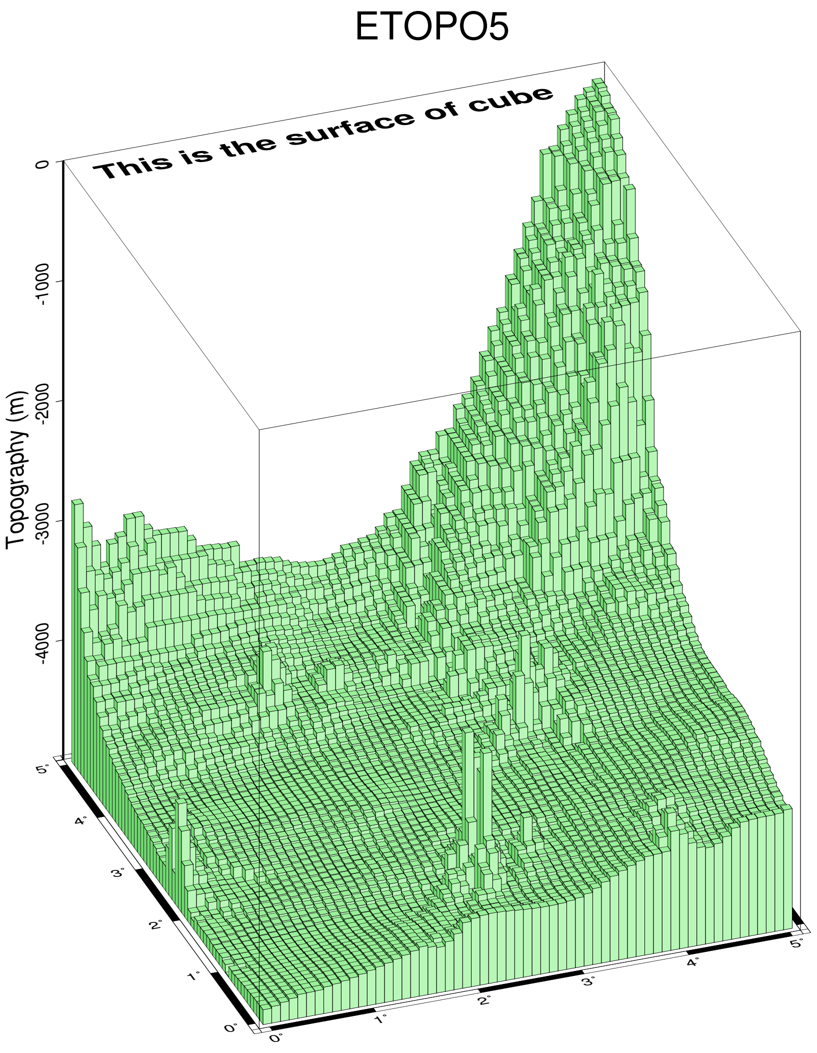


The same figure could equally well be made in color, which could be rasterized and made into a slide for a meeting presentation. The script is similar to the one outlined above, except we would choose a color for land and oceans, and select colored symbols and pens rather than black and white.

11.8 A 3-D histogram

The program `psxyz` allows us to plot three-dimensional symbols, including columnar plots. As a simple demonstration, we will convert a gridded netCDF of bathymetry into an ASCII *xyz* table and use the height information to draw a 2-D histogram in a 3-D perspective view. Our gridded bathymetry file is called `guinea_bay.nc` and covers the region from 0 to 5 E and 0 to 5 N. Depth ranges from -5000 meter to sea-level. We produce the Figure *Example 08* by running this script:

```
#!/bin/bash
#               GMT EXAMPLE 08
#
# Purpose:       Make a 3-D bar plot
# GMT progs:       grd2xyz, pstext, psxyz
# Unix progs:        echo, rm
#
ps=example_08.ps
gmt grd2xyz guinea_bay.nc | gmt psxyz -B1 -Bz1000+l"Topography (m)" -BWSneZ+b+tETOPO5 \
        -R-0.1/5.1/-0.1/5.1/-5000/0 -JM5i -JZ6i -p200/30 -So0.0833333ub-5000 -P \
        -Wthinnest -Glightgreen -K > $ps
echo '0.1 4.9 This is the surface of cube' | gmt pstext -R -J -JZ -Z0 \
        -F+f24p,Helvetica-Bold+jTL -p -O >> $ps
```



11.9 Plotting time-series along tracks

A common application in many scientific disciplines involves plotting one or several time-series as as "wiggles" along tracks. Marine geophysicists often display magnetic anomalies in this manner, and seismologists use the technique when plotting individual seismic traces. In our example we will show how a set of Geosat sea surface slope profiles from the south Pacific can be plotted as "wiggles" using the `pswiggle` program. We will embellish the plot with track numbers, the location of the Pacific-Antarctic Ridge, recognized fracture zones in the area, and a "wiggle" scale. The Geosat tracks are stored in the file `tracks.txt`, the ridge in `ridge.xy`, and all the fracture zones are stored in the multiple segment file `fz.xy`. The profile id is contained in the segment headers and we wish to use the last data point in each of the track segments to construct an input file for `pstext` that will label each profile with the track number. We know the profiles trend approximately N40E so we want the labels to have that same orientation (i.e., the angle with the baseline must be 50). We do this by extracting the last record from each track and select segment header as textstring when running the output through `pstext`. Note we offset the positions by -0.05 inch with **-D** in order to have a small gap between the profile and the label:

```
#!/bin/bash
#               GMT EXAMPLE 09
#
# Purpose:        Make wiggle plot along track from geoid deflections
# GMT progs:        gmtconvert, pswiggle, pstext, psxy
# Unix progs:
#
ps=example_09.ps
gmt pswiggle tracks.txt -R185/250/-68/-42 -K -Jm0.13i -Ba10f5 -BWSne+g240/255/240 -G+red \
        -G-blue -Z2000 -Wthinnest -S240/-67/500/@~m@~rad --FORMAT_GEO_MAP=dddF > $ps
gmt psxy -R -J -O -K ridge.xy -Wthicker >> $ps
gmt psxy -R -J -O -K fz.xy -Wthinner,- >> $ps
# Take label from segment header and plot near coordinates of last record of each track
gmt gmtconvert -El tracks.txt | gmt pstext -R -J -F+f10p,Helvetica-Bold+a50+jRM+h \
        -D-0.05i/-0.05i -O >> $ps
```

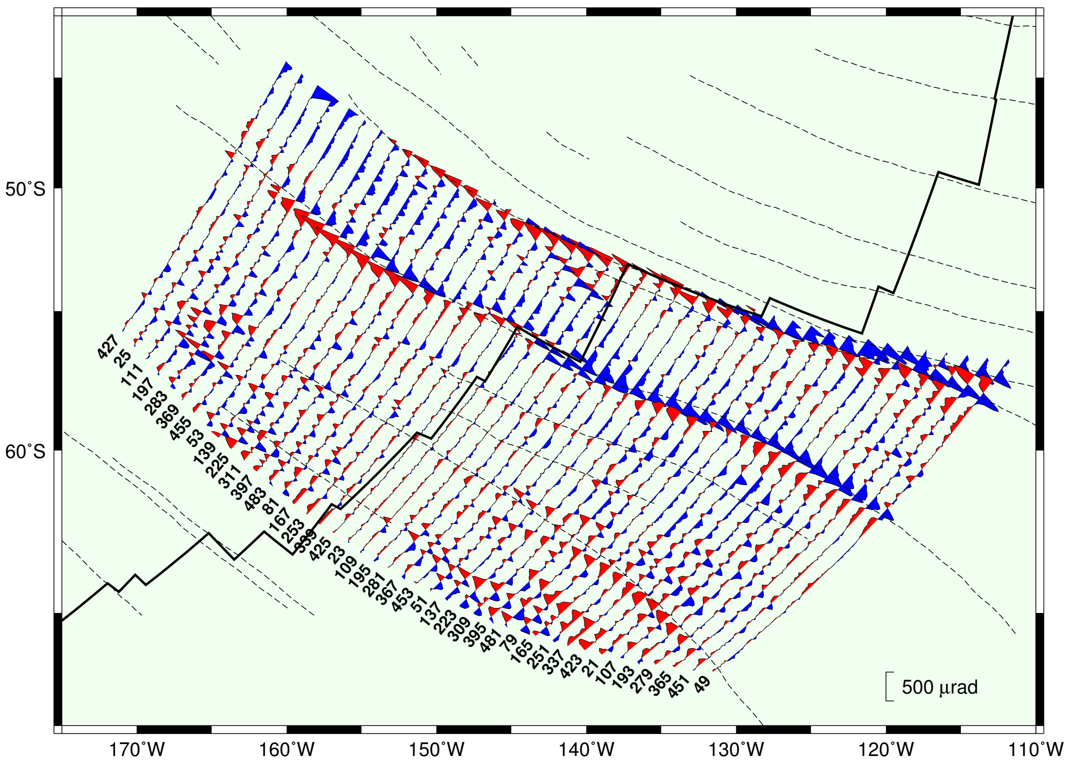
The output shows the sea-surface slopes along 42 descending Geosat tracks in the Eltanin and Udintsev fracture zone region in a Mercator projection.

11.10 A geographical bar graph plot

Our next and perhaps most business-like example presents a three-dimensional bar graph plot showing the geographic distribution of all the languages of the world. The input data was taken from Ethnologue. We decide to plot a 3-D column centered on each continent with a height that is proportional to the languages used. We choose a plain linear projection for the basemap and add the columns and text on top. Eventually we make it a bit more fancy by splitting the columns up in different colors indicating how commonly the languages are used, from institutional languages to languages threatened by extinction. The script also shows how to effectively use transparency of the boxes around the numbers and in the shade surrounding the legend.

Our script that produces Figure *Example 10* reads:

```
#!/bin/bash
#               GMT EXAMPLE 10
#
# Purpose:        Make 3-D bar graph on top of perspective map
# GMT progs:        pscoast, pstext, psxyz, pslegend
# Unix progs:         $AWK
#
ps=example_10.ps
gmt pscoast -Rd -JX8id/5id -Dc -Sazure2 -Gwheat -Wfaint -A5000 -p200/40 -K > $ps
```



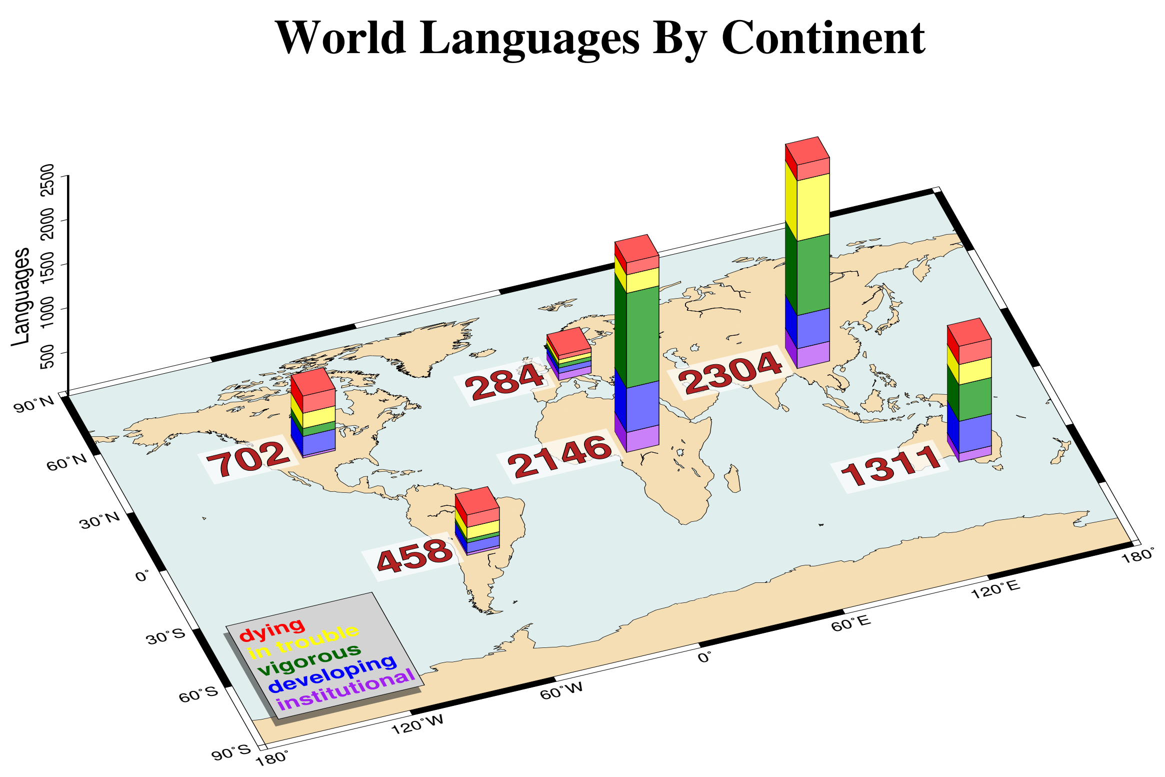
```
$AWK '{print $1, $2, $3+$4+$5+$6+$7}’ languages.txt \
        | gmt pstext -R -J -O -K -p -Gwhite@30 -D-0.25i/0 \
        -F+f30p,Helvetica-Bold,firebrick=thinner+jRM >> $ps
gmt psxyz languages.txt -R-180/180/-90/90/0/2500 -J -JZ2.5i -So0.3i -Gpurple -Wthinner \
        --FONT_TITLE=30p,Times-Bold --MAP_TITLE_OFFSET=-0.7i -O -K -p --FORMAT_GEO_MAP=dddF \
        -Bx60 -By30 -Bza500+lLanguages -BWSneZ+t"World Languages By Continent" >> $ps
$AWK '{print $1, $2, $3+$4, $3}’ languages.txt \
        | gmt psxyz -R -J -JZ -So0.3ib -Gblue -Wthinner -O -K -p >> $ps
$AWK '{print $1, $2, $3+$4+$5, $3+$4}’ languages.txt \
        | gmt psxyz -R -J -JZ -So0.3ib -Gdarkgreen -Wthinner -O -K -p >> $ps
$AWK '{print $1, $2, $3+$4+$5+$6, $3+$4+$5}’ languages.txt \
        | gmt psxyz -R -J -JZ -So0.3ib -Gyellow -Wthinner -O -K -p >> $ps
$AWK '{print $1, $2, $3+$4+$5+$6+$7, $3+$4+$5+$6}’ languages.txt \
        | gmt psxyz -R -J -JZ -So0.3ib -Gred -Wthinner -O -K -p >> $ps
gmt pslegend -R -J -JZ -D-170/-80/1.35i/0/BL -O --FONT=Helvetica-Bold \
        -F+glightgrey+pthinner+s-4p/-6p/grey20@40 -p legend.txt >> $ps
```

11.11 Making a 3-D RGB color cube

In this example we generate a series of 6 color images, arranged so that they can be cut out and assembled into a 3-D color cube. The six faces of the cube represent the outside of the R-G-B color space. On each face one of the color components is fixed at either 0 or 255 and the other two components vary smoothly across the face from 0 to 255. The cube is configured as a right-handed coordinate system with *x*-*y*-*z* mapping R-G-B. Hence, the 8 corners of the cube represent the primaries red, green, and blue, plus the secondaries cyan, magenta and yellow, plus black and white.

The 6 color faces are generated by feeding `grdimage` three grids, one for each color component (R, G, and B). In some cases the X or Y axes of a face are reversed by specifying a negative width or height in order to change the variation of the color value in that direction from ascending to descending, or vice versa.

A number of rays emanating from the white and black corners indicate the Hue value (ranging from 0 to 360). The dashed and dotted lines near the white corner reflect saturation levels, running from 0 to



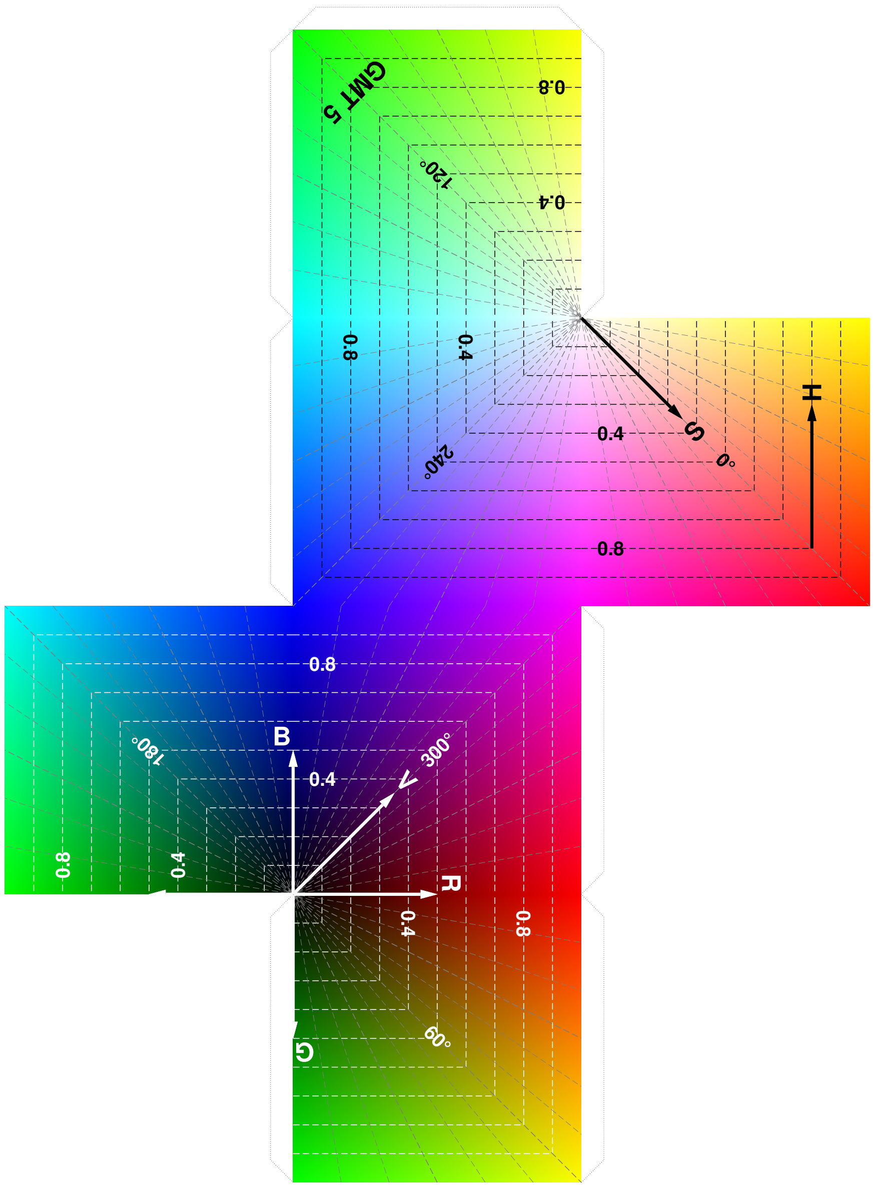
1 (in black font). On these 3 faces the brightness is a constant value of 1. On the other 3 faces of the cube, around the black corner, the white decimal numbers indicate brightnesses between 0 and 1, with saturation fixed at 1.

Here is the shell script to generate the RGB cube in Figure *Example 11*:

```
#!/bin/bash
#               GMT EXAMPLE 11
#
# Purpose:        Create a 3-D RGB Cube
# GMT progs:        gmtset, grdimage, grdmath, pstext, psxy
# Unix progs:        rm
ps=example_11.ps

# Use gmt psxy to plot "cut-along-the-dotted" lines.

gmt gmtset MAP_TICK_LENGTH_PRIMARY 0

gmt psxy cut-here.dat -Wthinnest,. -R-51/306/0/1071 -JX3.5i/10.5i -X2.5i -Y0.5i -P -K > $ps

# First, create grids of ascending X and Y and constant 0.
# These are to be used to represent R, G and B values of the darker 3 faces of the cube.

gmt grdmath -I1 -R0/255/0/255 X = x.nc
gmt grdmath -I1 -R Y = y.nc
gmt grdmath -I1 -R 0 = c.nc

gmt gmtset FONT_ANNOT_PRIMARY 12p,Helvetica-Bold

gmt grdimage x.nc y.nc c.nc -JX2.5i/-2.5i -R -K -O -X0.5i >> $ps
gmt psxy -Wthinner,white,- rays.dat -J -R -K -O >> $ps
gmt pstext --FONT=white -J -R -K -O -F+f+a >> $ps << END
128 128 12p -45 60\217
102  26 12p -90 0.4
204  26 12p -90 0.8
10  140 16p 180 G
END
echo 0 0 0 128 | gmt psxy -N -Sv0.15i+s+e -Gwhite -W2p,white -J -R -K -O >> $ps

gmt grdimage x.nc c.nc y.nc -JX2.5i/2.5i -R -K -O -Y2.5i >> $ps
gmt psxy -Wthinner,white,- rays.dat -J -R -K -O >> $ps
gmt pstext --FONT=white -J -R -K -O -F+f+a >> $ps << END
128 128 12p  45 300\217
26  102 12p   0 0.4
26  204 12p   0 0.8
140  10 16p -90 R
100 100 16p -45 V
END
echo 0 0 128 0 | gmt psxy -N -Sv0.15i+s+e -Gwhite -W2p,white -J -R -K -O >> $ps
echo 0 0 90 90 | gmt psxy -N -Sv0.15i+s+e -Gwhite -W2p,white -J -R -K -O >> $ps

gmt grdimage c.nc x.nc y.nc -JX-2.5i/2.5i -R -K -O -X-2.5i >> $ps
gmt psxy -Wthinner,white,- rays.dat -J -R -K -O >> $ps
gmt pstext --FONT=white -J -R -K -O -F+f+a >> $ps << END
128 128 12p 135 180\217
102  26 12p  90 0.4
204  26 12p  90 0.8
10  140 16p   0 B
END
echo 0 0 0 128 | gmt psxy -N -Sv0.15i+s+e -Gwhite -W2p,white -J -R -K -O >> $ps
echo 0 0 128 0 | gmt psxy -N -Sv0.15i+s+e -Gwhite -W2p,white -J -R -K -O >> $ps

# Second, create grids of descending X and Y and constant 255.
# These are to be used to represent R, G and B values of the lighter 3 faces of the cube.

gmt grdmath -I1 -R 255 X SUB = x.nc
gmt grdmath -I1 -R 255 Y SUB = y.nc
gmt grdmath -I1 -R 255       = c.nc

gmt grdimage x.nc y.nc c.nc -JX-2.5i/-2.5i -R -K -O -X2.5i -Y2.5i >> $ps
gmt psxy -Wthinner,black,- rays.dat -J -R -K -O >> $ps
gmt pstext -J -R -K -O -F+f+a >> $ps << END
128 128 12p 225 240\217
102  26 12p 270 0.4
204  26 12p 270 0.8
END

gmt grdimage c.nc y.nc x.nc -JX2.5i/-2.5i -R -K -O -X2.5i >> $ps
gmt psxy -Wthinner,black,- rays.dat -J -R -K -O >> $ps
gmt pstext -J -R -K -O -F+f+a >> $ps << END
128 128 12p -45 0\217
26  102 12p   0 0.4
26  204 12p   0 0.8
100 100 16p  45 S
204  66 16p  90 H
END
echo 0 0 90 90 | gmt psxy -N -Sv0.15i+s+e -Gblack -W2p -J -R -K -O >> $ps
echo 204 204 204 76 | gmt psxy -N -Sv0.15i+s+e -Gblack -W2p -J -R -K -O >> $ps

gmt grdimage x.nc c.nc y.nc -JX-2.5i/2.5i -R -K -O -X-2.5i -Y2.5i >> $ps
gmt psxy -Wthinner,black,- rays.dat -J -R -K -O >> $ps
gmt pstext -J -R -O -F+f+a >> $ps << END
128 128 12p 135 120\217
26  102 12p 180 0.4
26  204 12p 180 0.8
200 200 16p 225 GMT 5
END

rm -f *.nc gmt.conf
```

11.12 Optimal triangulation of data

Our next example (Figure *Example 12*) operates on a data set of topographic readings non-uniformly distributed in the plane (Table 5.11 in Davis: *Statistics and Data Analysis in Geology*, J. Wiley). We use `triangulate` to perform the optimal Delaunay triangulation, then use the output to draw the resulting network. We label the node numbers as well as the node values, and call `pscontour` to make a contour map and image directly from the raw data. Thus, in this example we do not actually make grid


GMT 5
0.8
0.4
120°
0.8
0.4
240°
0.4
S
0°
H
0.8
0.8
B
180°
0.4
300°
V
R
0.8
0.4
0.4
0.8
60°
G

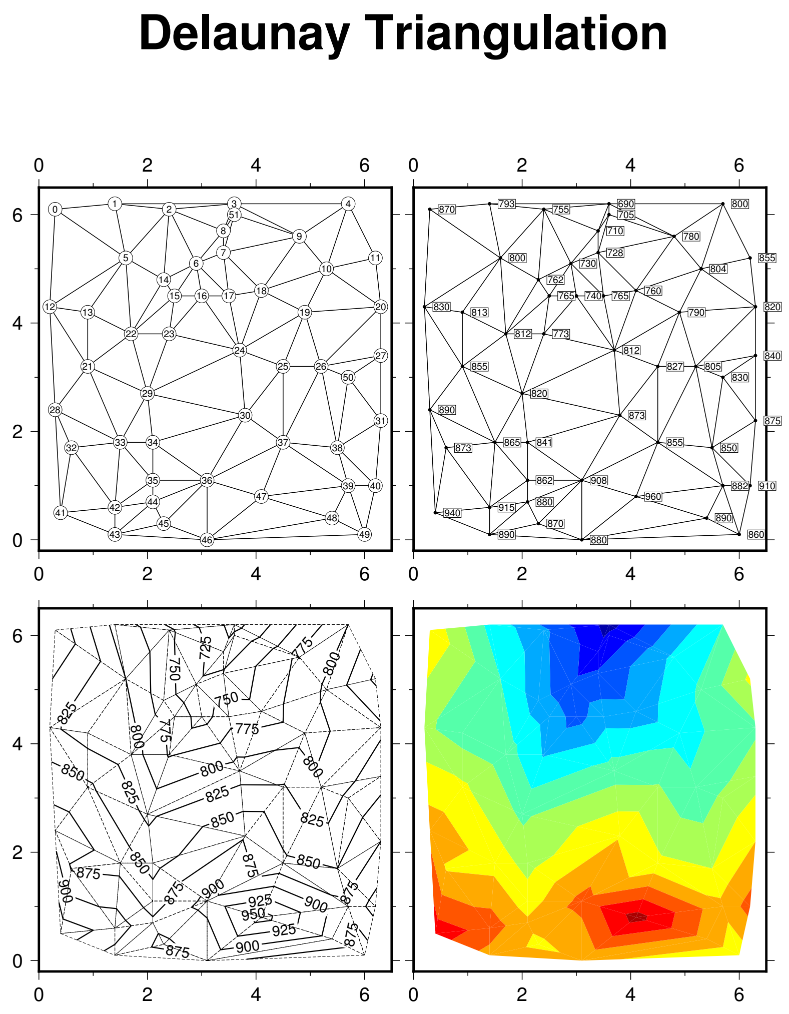
files but still are able to contour and image the data. We use a color palette table `topo.cpt` (created via `gmtinfo` and `makecpt`). The script becomes:

```
#!/bin/bash
#                GMT EXAMPLE 12
#
# Purpose:        Illustrates Delaunay triangulation of points, and contouring
# GMT progs:        makecpt, gmtinfo, pscontour, pstext, psxy, triangulate
# Unix progs:        $AWK, echo, rm
#
# First draw network and label the nodes
#
ps=example_12.ps
gmt triangulate table_5.11 -M > net.xy
gmt psxy -R0/6.5/-0.2/6.5 -JX3.06i/3.15i -B2f1 -BWSNe net.xy -Wthinner -P -K -X0.9i -Y4.65i > $ps
gmt psxy table_5.11 -R -J -O -K -Sc0.12i -Gwhite -Wthinnest >> $ps
$AWK '{print $1, $2, NR-1}' table_5.11 | gmt pstext -R -J -F+f6p -O -K >> $ps
#
# Then draw network and print the node values
#
gmt psxy -R -J -B2f1 -BeSNw net.xy -Wthinner -O -K -X3.25i >> $ps
gmt psxy -R -J -O -K table_5.11 -Sc0.03i -Gblack >> $ps
gmt pstext table_5.11 -R -J -F+f6p+jLM -O -K -Gwhite -W -C0.01i -D0.08i/0i -N >> $ps
#
# Then contour the data and draw triangles using dashed pen; use "gmt gmtinfo" and "gmt makecpt" to make a
# color palette (.cpt) file
#
T=`gmt info -T25/2 table_5.11`
gmt makecpt -Cjet $T > topo.cpt
gmt pscontour -R -J table_5.11 -B2f1 -BWSne -Wthin -Ctopo.cpt -Lthinnest,- -Gd1i -X-3.25i -Y-3.65i \
        -O -K >> $ps
#
# Finally color the topography
#
gmt pscontour -R -J table_5.11 -B2f1 -BeSnw -Ctopo.cpt -I -X3.25i -O -K >> $ps
echo "3.16 8 Delaunay Triangulation" | \
        gmt pstext -R0/8/0/11 -Jx1i -F+f30p,Helvetica-Bold+jCB -O -X-3.25i >> $ps
#
rm -f net.xy topo.cpt
```

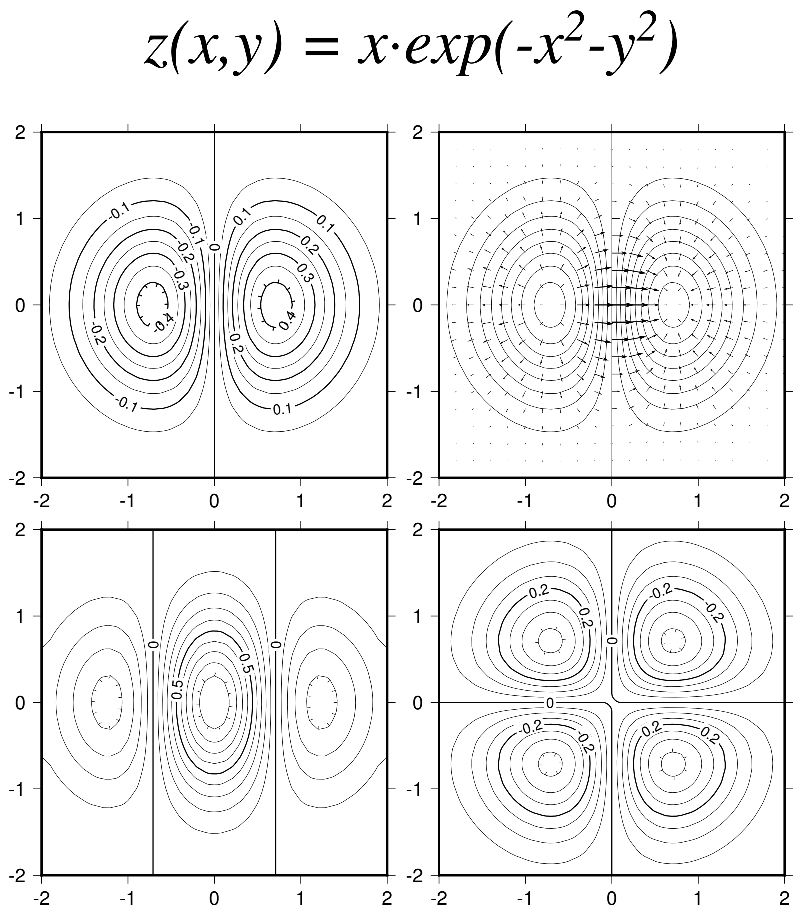
11.13 Plotting of vector fields

In many areas, such as fluid dynamics and elasticity, it is desirable to plot vector fields of various kinds. GMT provides a way to illustrate 2-component vector fields using the `grdvector` utility. The two components of the field (Cartesian or polar components) are stored in separate grid files. In this example we use `grdmath` to generate a surface $z(x, y) = x \cdot \exp(-x^2 - y^2)$ and to calculate ∇z by returning the *x*- and *y*-derivatives separately. We superpose the gradient vector field and the surface *z* and also plot the components of the gradient in separate windows. A `pstext` call to place a header finishes the plot Figure *Example 13*:

```
#!/bin/bash
#                GMT EXAMPLE 13
#
# Purpose:        Illustrate vectors and contouring
# GMT progs:        grdmath, grdcontour, grdvector, pstext
# Unix progs:        echo, rm
#
ps=example_13.ps
gmt grdmath -R-2/2/-2/2 -I0.1 X Y R2 NEG EXP X MUL = z.nc
gmt grdmath z.nc DDX = dzdx.nc
gmt grdmath z.nc DDY = dzdy.nc
gmt grdcontour dzdx.nc -JX3i -B1 -BWSne -C0.1 -A0.5 -K -P -Gd2i -S4 -T0.1i/0.03i > $ps
gmt grdcontour dzdy.nc -J -B -C0.05 -A0.2 -O -K -Gd2i -S4 -T0.1i/0.03i -Xa3.45i >> $ps
gmt grdcontour z.nc -J -B -C0.05 -A0.1 -O -K -Gd2i -S4 -T0.1i/0.03i -Y3.45i >> $ps
gmt grdcontour z.nc -J -B -C0.05 -O -K -Gd2i -S4 -X3.45i >> $ps
gmt grdvector dzdx.nc dzdy.nc -I0.2 -J -O -K -Q0.1i+e+n0.25i -Gblack -W1p -S5i \
```


Delaunay Triangulation

```
        --MAP_VECTOR_SHAPE=0.5 >> $ps
echo "3.2 3.6 z(x,y) = x@~\327@~exp(-x@+2@+-y@+2@+)" \
        | gmt pstext -R0/6/0/4.5 -Jx1i -F+f40p,Times-Italic+jCB -O -X-3.45i >> $ps
rm -f z.nc dzdx.nc dzdy.nc
```



11.14 Gridding of data and trend surfaces

This example shows how one goes from randomly spaced data points to an evenly sampled surface. First we plot the distribution and values of our raw data set (same as in Section [sec:example$_1$2]). We choose an equidistant grid and run `blockmean` which preprocesses the data to avoid aliasing. The dashed lines indicate the logical blocks used by `blockmean`; all points inside a given bin will be averaged. The logical blocks are drawn from a temporary file we make on the fly within the shell script. The processed data is then gridded with the `surface` program and contoured every 25 units. A most important point here is that `blockmean`, `blockmedian`, or `blockmode` should always be run prior to running `surface`, and both of these steps must use the same grid interval. We use `grdtrend` to fit a bicubic trend surface to the gridded data, contour it as well, and sample both grid files along a diagonal transect using `grdtrack`. The bottom panel compares the gridded (solid line) and bicubic trend (dashed line) along the transect using `psxy` (Figure *Example 14*):

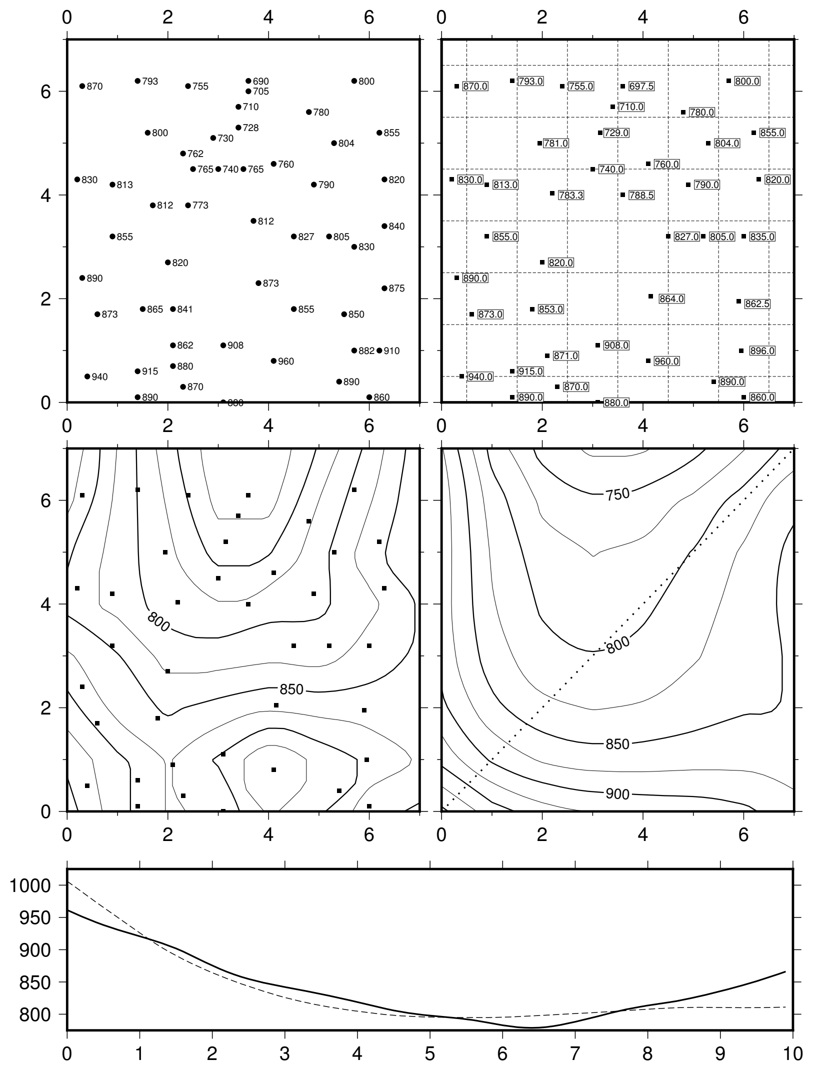
```
#!/bin/bash
#               GMT EXAMPLE 14
#
# Purpose:       Showing simple gridding, contouring, and resampling along tracks
# GMT progs:       blockmean, grdcontour, grdtrack, grdtrend, gmtinfo, project
# GMT progs:       gmtset, pstext, psbasemap, psxy, surface
# Unix progs:       rm
#
```

```
ps=example_14.ps

# First draw network and label the nodes

gmt gmtset MAP_GRID_PEN_PRIMARY thinnest,-
gmt psxy table_5.11 -R0/7/0/7 -JX3.06i/3.15i -B2f1 -BWSNe -Sc0.05i -Gblack -P -K -Y6.45i > $ps
gmt pstext table_5.11 -R -J -D0.1c/0 -F+f6p+jLM -O -K -N >> $ps
gmt blockmean table_5.11 -R0/7/0/7 -I1 > mean.xyz

# Then draw gmt blockmean cells

gmt psbasemap -R0.5/7.5/0.5/7.5 -J -O -K -Bg1 -X3.25i >> $ps
gmt psxy -R0/7/0/7 -J -B2f1 -BeSNw mean.xyz -Ss0.05i -Gblack -O -K >> $ps
# Reformat to one decimal for annotation purposes
gmt gmtconvert mean.xyz --FORMAT_FLOAT_OUT=%.1f | \
        gmt pstext -R -J -D0.15c/0 -F+f6p+jLM -O -K -Gwhite -W -C0.01i -N >> $ps

# Then gmt surface and contour the data

gmt surface mean.xyz -R -I1 -Gdata.nc
gmt grdcontour data.nc -J -B2f1 -BWSne -C25 -A50 -Gd3i -S4 -O -K -X-3.25i -Y-3.55i >> $ps
gmt psxy -R -J mean.xyz -Ss0.05i -Gblack -O -K >> $ps

# Fit bicubic trend to data and compare to gridded gmt surface

gmt grdtrend data.nc -N10 -Ttrend.nc
gmt project -C0/0 -E7/7 -G0.1 -N > track
gmt grdcontour trend.nc -J -B2f1 -BwSne -C25 -A50 -Glct/cb -S4 -O -K -X3.25i >> $ps
gmt psxy -R -J track -Wthick,. -O -K >> $ps

# Sample along diagonal

gmt grdtrack track -Gdata.nc -o2,3 > data.d
gmt grdtrack track -Gtrend.nc -o2,3 > trend.d
gmt psxy `gmt info data.d trend.d -I0.5/25` -JX6.3i/1.4i data.d -Wthick -O -K -X-3.25i -Y-1.9i \
        -Bx1 -By50 -BWSne >> $ps
gmt psxy -R -J trend.d -Wthinner,- -O >> $ps

rm -f mean.xyz track *.nc *.d gmt.conf
```

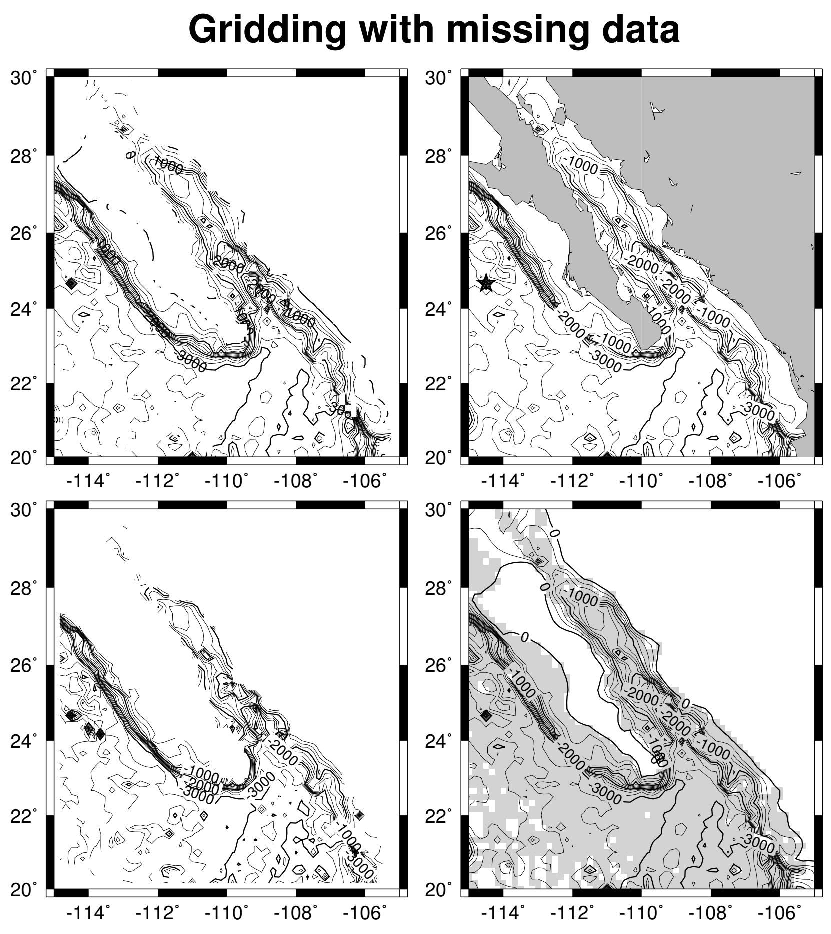
11.15 Gridding, contouring, and masking of unconstrained areas

This example (Figure *Example 15*) demonstrates some off the different ways one can use to grid data in GMT, and how to deal with unconstrained areas. We first convert a large ASCII file to binary with `gmtconvert` since the binary file will read and process much faster. Our lower left plot illustrates the results of gridding using a nearest neighbor technique (`nearneighbor`) which is a local method: No output is given where there are no data. Next (lower right), we use a minimum curvature technique (`surface`) which is a global method. Hence, the contours cover the entire map although the data are only available for portions of the area (indicated by the gray areas plotted using `psmask`). The top left scenario illustrates how we can create a clip path (using `psmask`) based on the data coverage to eliminate contours outside the constrained area. Finally (top right) we simply employ `pscoast` to overlay gray land masses to cover up the unwanted contours, and end by plotting a star at the deepest point on the map with `psxy`. This point was extracted from the grid files using `grdinfo`.

```
#!/bin/bash
#               GMT EXAMPLE 15
#
# Purpose:        Gridding and clipping when data are missing
# GMT progs:        blockmedian, gmtconvert, grdclip, grdcontour, grdinfo, gmtinfo
# GMT progs:        nearneighbor, pscoast, psmask, pstext, surface
# Unix progs:        echo, rm
#
ps=example_15.ps
gmt gmtconvert ship.xyz -bo > ship.b
```


750
800
850
900
1000
950
900
850
800

```
#
region=`gmt info ship.b -I1 -bi3d`
gmt nearneighbor $region -I10m -S40k -Gship.nc ship.b -bi
gmt grdcontour ship.nc -JM3i -P -B2 -BWSne -C250 -A1000 -Gd2i -K > $ps
#
gmt blockmedian $region -I10m ship.b -b3d > ship_10m.b
gmt surface $region -I10m ship_10m.b -Gship.nc -bi
gmt psmask $region -I10m ship.b -J -O -K -T -Glightgray -bi3d -X3.6i >> $ps
gmt grdcontour ship.nc -J -B -C250 -L-8000/0 -A1000 -Gd2i -O -K >> $ps
#
gmt psmask $region -I10m ship_10m.b -bi3d -J -B -O -K -X-3.6i -Y3.75i >> $ps
gmt grdcontour ship.nc -J -C250 -A1000 -L-8000/0 -Gd2i -O -K >> $ps
gmt psmask -C -O -K >> $ps
#
gmt grdclip ship.nc -Sa-1/NaN -Gship_clipped.nc
gmt grdcontour ship_clipped.nc -J -B -C250 -A1000 -L-8000/0 -Gd2i -O -K -X3.6i >> $ps
gmt pscoast $region -J -O -K -Ggray -Wthinnest >> $ps
gmt grdinfo -C -M ship.nc | gmt psxy -R -J -O -K -Sa0.15i -Wthick -i11,12 >> $ps
echo "-0.3 3.6 Gridding with missing data" | gmt pstext -R0/3/0/4 -Jx1i \
        -F+f24p,Helvetica-Bold+jCB -O -N >> $ps
rm -f ship.b ship_10m.b ship.nc ship_clipped.nc
```

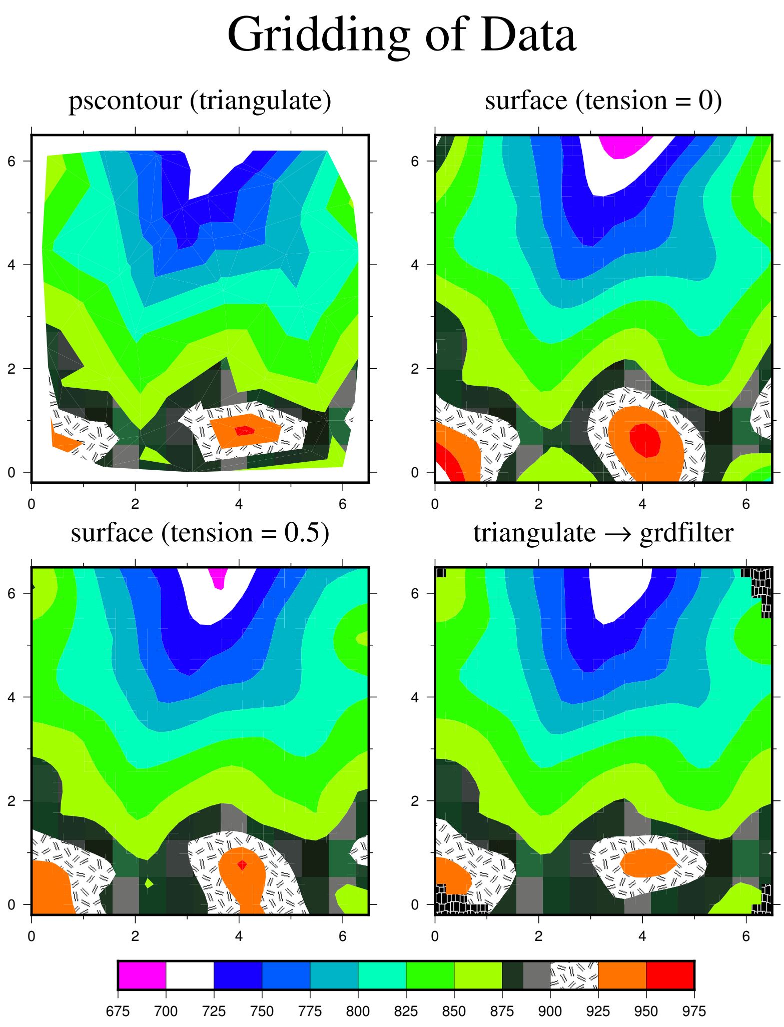


11.16 Gridding of data, continued

`pscontour` (for contouring) and `triangulate` (for gridding) use the simplest method of interpolating data: a Delaunay triangulation (see Section [sec:example$_1$2]) which forms *z(x, y)* as a union of planar triangular facets. One advantage of this method is that it will not extrapolate *z(x, y)* beyond the convex hull of the input (*x, y*) data. Another is that it will not estimate a *z* value above or below the local bounds on any triangle. A disadvantage is that the *z(x, y)* surface is not differentiable, but has sharp kinks at triangle edges and thus also along contours. This may not look physically reasonable, but it can be filtered later (last panel below). `surface` can be used to generate a higher-order (smooth and differentiable) interpolation of *z(x, y)* onto a grid, after which the grid may be illustrated (`grdcontour`, `grdimage`, `grdview`). `surface` will interpolate to all (*x, y*) points in a rectangular region, and thus will extrapolate beyond the convex hull of the data. However, this can be masked out in various ways (see Section [sec:example$_1$5]).

A more serious objection is that `surface` may estimate *z* values outside the local range of the data (note area near x = 0.8, y = 5.3). This commonly happens when the default tension value of zero is used to create a "minimum curvature" (most smooth) interpolant. `surface` can be used with non-zero tension to partially overcome this problem. The limiting value *tension = 1* should approximate the triangulation, while a value between 0 and 1 may yield a good compromise between the above two cases. A value of 0.5 is shown here (Figure *Example 16*). A side effect of the tension is that it tends to make the contours turn near the edges of the domain so that they approach the edge from a perpendicular direction. A solution is to use `surface` in a larger area and then use `grdcut` to cut out the desired smaller area. Another way to achieve a compromise is to interpolate the data to a grid and then filter the grid using `grdfft` or `grdfilter`. The latter can handle grids containing "NaN" values and it can do median and mode filters as well as convolutions. Shown here is `triangulate` followed by `grdfilter`. Note that the filter has done some extrapolation beyond the convex hull of the original *x, y* values. The "best" smooth approximation of *z(x, y)* depends on the errors in the data and the physical laws obeyed by *z*. GMT cannot always do the "best" thing but it offers great flexibility through its combinations of tools. We illustrate all four solutions using a CPT file that contains color fills, predefined patterns for interval (900,925) and NaN, an image pattern for interval (875,900), and a "skip slice" request for interval (700,725).

```
#!/bin/bash
#               GMT EXAMPLE 16
#
# Purpose:        Illustrates interpolation methods using same data as Example 12.
# GMT progs:        gmtset, grdview, grdfilter, pscontour, psscale, pstext, surface, triangulate
# Unix progs:        echo, rm
#
# Illustrate various means of contouring, using triangulate and surface.
#
ps=example_16.ps
gmt gmtset FONT_ANNOT_PRIMARY 9p
#
gmt pscontour -R0/6.5/-0.2/6.5 -Jx0.45i -P -K -Y5.5i -Ba2f1 -BWSne table_5.11 -Cex16.cpt -I > $ps
echo "3.25 7 pscontour (triangulate)" | gmt pstext -R -J -O -K -N -F+f18p,Times-Roman+jCB >> $ps
#
gmt surface table_5.11 -R -I0.2 -Graws0.nc
gmt grdview raws0.nc -R -J -B -Cex16.cpt -Qs -O -K -X3.5i >> $ps
echo "3.25 7 surface (tension = 0)" | gmt pstext -R -J -O -K -N -F+f18p,Times-Roman+jCB >> $ps
#
gmt surface table_5.11 -R -I0.2 -Graws5.nc -T0.5
gmt grdview raws5.nc -R -J -B -Cex16.cpt -Qs -O -K -Y-3.75i -X-3.5i >> $ps
echo "3.25 7 surface (tension = 0.5)" | gmt pstext -R -J -O -K -N -F+f18p,Times-Roman+jCB >> $ps
#
gmt triangulate table_5.11 -Grawt.nc -R -I0.2 > /dev/null
gmt grdfilter rawt.nc -Gfiltered.nc -D0 -Fc1
gmt grdview filtered.nc -R -J -B -Cex16.cpt -Qs -O -K -X3.5i >> $ps
echo "3.25 7 triangulate @~\256@~ grdfilter" | gmt pstext -R -J -O -K -N \
        -F+f18p,Times-Roman+jCB >> $ps
echo "3.2125 7.5 Gridding of Data" | gmt pstext -R0/10/0/10 -Jx1i -O -K -N \
        -F+f32p,Times-Roman+jCB -X-3.5i >> $ps
gmt psscale -D3.25i/0.35i/5i/0.25ih -Cex16.cpt -O -Y-0.75i >> $ps
#
rm -f *.nc gmt.conf
```


Gridding of Data
pscontour (triangulate)
surface (tension = 0)
surface (tension = 0.5)
triangulate → grdfilter
0
2
4
6
675
700
725
750
775
800
825
850
875
900
925
950
975

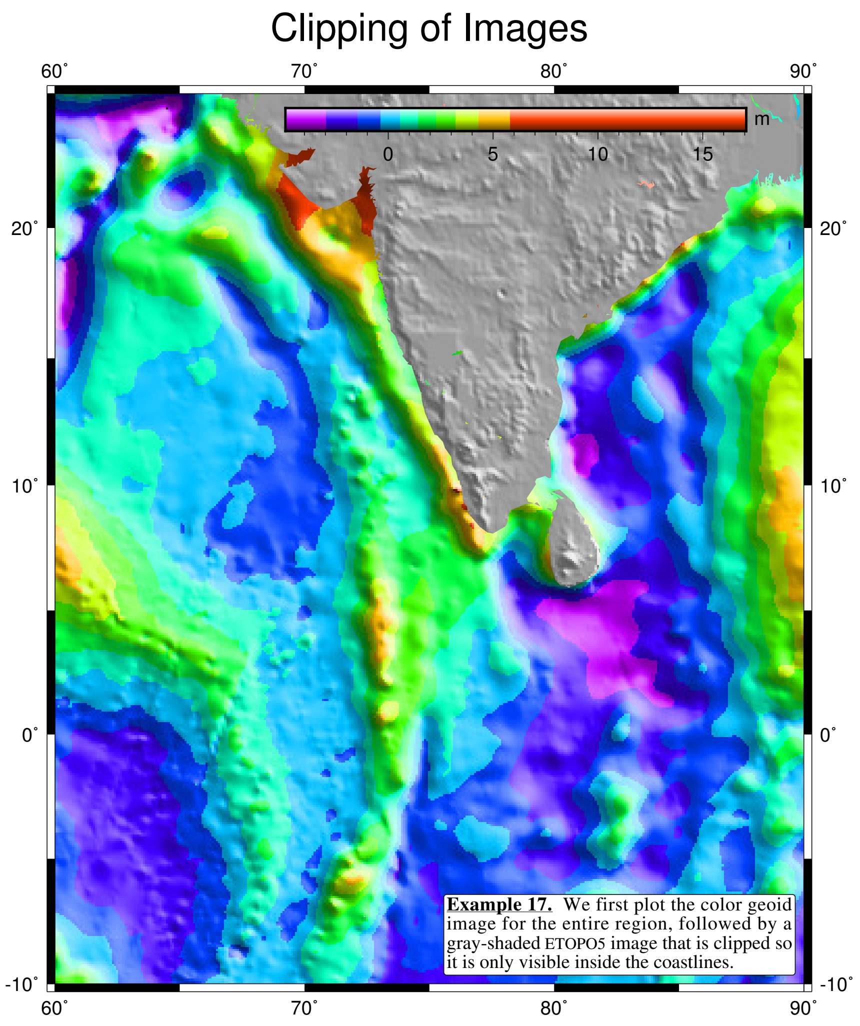
11.17 Images clipped by coastlines

This example demonstrates how `pscoast` can be used to set up clip paths based on coastlines. This approach is well suited when different gridded data sets are to be merged on a plot using different color palette files. Merging the files themselves may not be doable since they may represent different data sets, as we show in this example. Here, we lay down a color map of the geoid field near India with `grdimage`, use `pscoast` to set up land clip paths, and then overlay topography from the ETOPO5 data set with another call to `grdimage`. We finally undo the clippath with a second call to `pscoast` with the option **-Q** (Figure *Example 17*):

We also plot a color legend on top of the land. So here we basically have three layers of "paint" stacked on top of each other: the underlaying geoid map, the land mask, and finally the color legend. This legend makes clear how `grd2cpt` distributed the colors over the range: they are not of equal length put are associated with equal amounts of area in the plot. Since the high amounts (in red) are not very prevalent, that color spans a long range.

For this image it is appropriate to use the **-I** option in `psscale` so the legend gets shaded, similar to the geoid grid. See Appendix [app:M] to learn more about color palettes and ways to draw color legends.

```
#!/bin/bash
#                 GMT EXAMPLE 17
#
# Purpose:         Illustrates clipping of images using coastlines
# GMT progs:         grd2cpt, grdgradient, grdimage, pscoast, pstext
# Unix progs:         rm
#
ps=example_17.ps

# First generate geoid image w/ shading

gmt grd2cpt india_geoid.nc -Crainbow > geoid.cpt
gmt grdgradient india_geoid.nc -Nt1 -A45 -Gindia_geoid_i.nc
gmt grdimage india_geoid.nc -Iindia_geoid_i.nc -JM6.5i -Cgeoid.cpt -P -K > $ps

# Then use gmt pscoast to initiate clip path for land

gmt pscoast -Rindia_geoid.nc -J -O -K -Dl -Gc >> $ps

# Now generate topography image w/shading

echo "-10000 150 10000 150" > gray.cpt
gmt grdgradient india_topo.nc -Nt1 -A45 -Gindia_topo_i.nc
gmt grdimage india_topo.nc -Iindia_topo_i.nc -J -Cgray.cpt -O -K >> $ps

# Finally undo clipping and overlay basemap

gmt pscoast -R -J -O -K -Q -B10f5 -B+t"Clipping of Images" >> $ps

# Put a color legend on top of the land mask

gmt psscale -D4i/7.6i/4i/0.2ih -Cgeoid.cpt -Bx5f1 -By+lm -I -O -K >> $ps

# Add a text paragraph

gmt pstext -R -J -O -M -Gwhite -Wthinner -TO -D-0.1i/0.1i -F+f12,Times-Roman+jRB >> $ps << END
> 90 -10 12p 3i j
@_@%5%Example 17.@%%@_  We first plot the color geoid image
for the entire region, followed by a gray-shaded @#etopo5@#
image that is clipped so it is only visible inside the coastlines.
END

# Clean up

rm -f geoid.cpt gray.cpt *_i.nc
```



Example 17. We first plot the color geoid image for the entire region, followed by a gray-shaded ETOPO5 image that is clipped so it is only visible inside the coastlines.

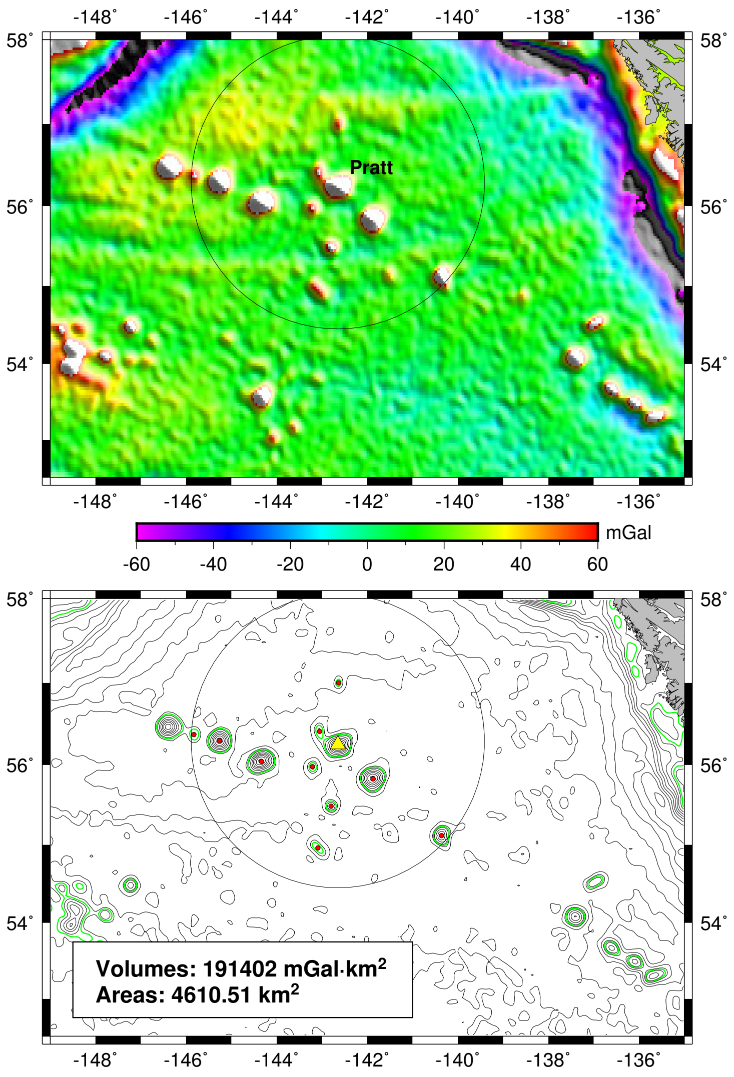
11.18 Volumes and Spatial Selections

To demonstrate potential usage of the new programs `grdvolume` and `gmtselect` we extract a subset of the Sandwell & Smith altimetric gravity field [2] for the northern Pacific and decide to isolate all seamounts that (1) exceed 50 mGal in amplitude and (2) are within 200 km of the Pratt seamount. We do this by dumping the 50 mGal contours to disk, then making a simple AWK script `center.awk` that returns the mean location of the points making up each closed polygon, and then pass these locations to `gmtselect` which retains only the points within 200 km of Pratt. We then mask out all the data outside this radius and use `grdvolume` to determine the combined area and volumes of the chosen seamounts. Our illustration is presented in Figure *Example 18*.

```
#!/bin/bash
#               GMT EXAMPLE 18
#
# Purpose:        Illustrates volumes of grids inside contours and spatial
#               selection of data
# GMT progs:        gmtset, gmtselect, gmtspatial, grdclip, grdcontour, grdgradient, grdimage
# GMT progs:        grdmath, grdvolume, makecpt, pscoast, psscale, pstext, psxy
# Unix progs:        $AWK, cat, rm
#
ps=example_18.ps

# Use spherical gmt projection since SS data define on sphere
gmt gmtset PROJ_ELLIPSOID Sphere FORMAT_FLOAT_OUT %g

# Define location of Pratt seamount and the 400 km diameter
echo "-142.65 56.25 400" > pratt.d

# First generate gravity image w/ shading, label Pratt, and draw a circle
# of radius = 200 km centered on Pratt.

gmt makecpt -Crainbow -T-60/60/120 -Z > grav.cpt
gmt grdgradient AK_gulf_grav.nc -Nt1 -A45 -GAK_gulf_grav_i.nc
gmt grdimage AK_gulf_grav.nc -IAK_gulf_grav_i.nc -JM5.5i -Cgrav.cpt -B2f1 -P -K -X1.5i \
        -Y5.85i > $ps
gmt pscoast -RAK_gulf_grav.nc -J -O -K -Di -Ggray -Wthinnest >> $ps
gmt psscale -D2.75i/-0.4i/4i/0.15ih -Cgrav.cpt -Bx20f10 -By+l"mGal" -O -K >> $ps
$AWK '{print $1, $2, "Pratt"}' pratt.d | gmt pstext -R -J -O -K -D0.1i/0.1i \
        -F+f12p,Helvetica-Bold+jLB >> $ps
gmt psxy pratt.d -R -J -O -K -SE- -Wthinnest >> $ps

# Then draw 10 mGal contours and overlay 50 mGal contour in green

gmt grdcontour AK_gulf_grav.nc -J -C20 -B2f1 -BWSEn -O -K -Y-4.85i >> $ps
# Save 50 mGal contours to individual files, then plot them
gmt grdcontour AK_gulf_grav.nc -C10 -L49/51 -Dsm_%d_%c.txt
gmt psxy -R -J -O -K -Wthin,green sm_*.txt >> $ps
gmt pscoast -R -J -O -K -Di -Ggray -Wthinnest >> $ps
gmt psxy pratt.d -R -J -O -K -SE- -Wthinnest >> $ps
rm -f sm_*_O.txt        # Only consider the closed contours

# Now determine centers of each enclosed seamount > 50 mGal but only plot
# the ones within 200 km of Pratt seamount.

# First determine mean location of each closed contour and
# add it to the file centers.d

gmt gmtspatial -Q -fg sm_*_C.txt > centers.d

# Only plot the ones within 200 km

gmt gmtselect -C200k/pratt.d centers.d -fg | gmt psxy -R -J -O -K -SC0.04i -Gred -Wthinnest >> $ps
gmt psxy -R -J -O -K -ST0.1i -Gyellow -Wthinnest pratt.d >> $ps

# Then report the volume and area of these seamounts only
# by masking out data outside the 200 km-radius circle
```

[2] See http://topex.ucsd.edu/marine_grav/mar_grav.html.

```
# and then evaluate area/volume for the 50 mGal contour

gmt grdmath -R `$AWK '{print $1, $2}' pratt.d` SDIST = mask.nc
gmt grdclip mask.nc -Sa200/NaN -Sb200/1 -Gmask.nc
gmt grdmath AK_gulf_grav.nc mask.nc MUL = tmp.nc
area=`gmt grdvolume tmp.nc -C50 -Sk | cut -f2`
volume=`gmt grdvolume tmp.nc -C50 -Sk | cut -f3`

gmt psxy -R -J -A -O -K -L -Wthin -Gwhite >> $ps << END
-148.5        52.75
-141        52.75
-141        53.75
-148.5        53.75
END
gmt pstext -R -J -O -F+f14p,Helvetica-Bold+jLM >> $ps << END
-148 53.08 Areas: $area km@+2@+
-148 53.42 Volumes: $volume mGal\264km@+2@+
END

# Clean up

rm -f grav.cpt sm_*.txt *_i.nc tmp.nc mask.nc pratt.d center* gmt.conf
```



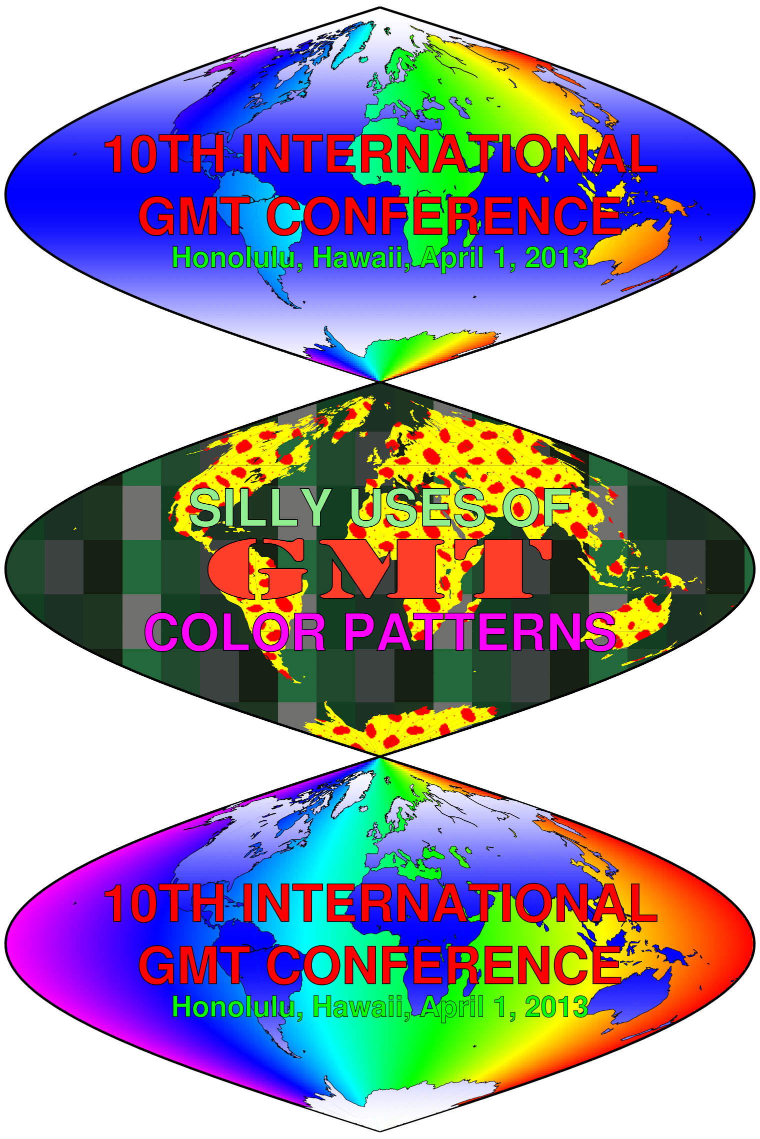
11.19 Color patterns on maps

GMT 3.1 introduced color patterns and this examples give a few cases of how to use this new feature. We make a phony poster that advertises an international conference on GMT in Honolulu. We use `grdmath`, `makecpt`, and `grdimage` to draw pleasing color backgrounds on maps, and overlay `pscoast` clip paths to have the patterns change at the coastlines. The middle panel demonstrates a simple `pscoast` call where the built-in pattern # 86 is drawn at 100 dpi but with the black and white pixels replaced with color combinations. At the same time the ocean is filled with a repeating image of a circuit board (provides in Sun raster format). The text GMT in the center is an off-line PostScript file that was overlaid using `psimage`. The final panel repeats the top panel except that the land and sea images have changed places (Figure *Example 19*).

```
#!/bin/bash
#               GMT EXAMPLE 19
#
# Purpose:        Illustrates various color pattern effects for maps
# GMT progs:        gmtset, grdimage, grdmath, makecpt, pscoast, pstext, psimage
# Unix progs:        rm
#
ps=example_19.ps

# First make a worldmap with graded blue oceans and rainbow continents

gmt grdmath -Rd -I1 -r Y COSD 2 POW = lat.nc
gmt grdmath -Rd -I1 -r X = lon.nc
echo "0 white 1 blue" > lat.cpt
gmt makecpt -Crainbow -T-180/180/360 -Z > lon.cpt
gmt grdimage lat.nc -JI0/6.5i -Clat.cpt -P -K -Y7.5i -B0 -nl > $ps
gmt pscoast -R -J -O -K -Dc -A5000 -Gc >> $ps
gmt grdimage lon.nc -J -Clon.cpt -O -K -nl >> $ps
gmt pscoast -R -J -O -K -Q >> $ps
gmt pscoast -R -J -O -K -Dc -A5000 -Wthinnest >> $ps
echo "0 20 10TH INTERNATIONAL" | gmt pstext -R -J -O -K -F+f32p,Helvetica-Bold,red=thinner >> $ps
echo "0 -10 GMT CONFERENCE" | gmt pstext -R -J -O -K -F+f32p,Helvetica-Bold,red=thinner >> $ps
echo "0 -30 Honolulu, Hawaii, April 1, 2013" | gmt pstext -R -J -O -K \
        -F+f18p,Helvetica-Bold,green=thinnest >> $ps

# Then show example of color patterns and placing a PostScript image

gmt pscoast -R -J -O -K -Dc -A5000 -Gp100/86:FredByellow -Sp100/circuit.ras -B0 -Y-3.25i >> $ps
echo "0 30 SILLY USES OF" | gmt pstext -R -J -O -K -F+f32p,Helvetica-Bold,lightgreen=thinner >> $ps
echo "0 -30 COLOR PATTERNS" | gmt pstext -R -J -O -K -F+f32p,Helvetica-Bold,magenta=thinner >> $ps
gmt psimage -C3.25i/1.625i/CM -W3i GMT_covertext.eps -O -K >> $ps

# Finally repeat 1st plot but exchange the patterns

gmt grdimage lon.nc -J -Clon.cpt -O -K -Y-3.25i -B0 -nl >> $ps
gmt pscoast -R -J -O -K -Dc -A5000 -Gc >> $ps
gmt grdimage lat.nc -J -Clat.cpt -O -K -nl >> $ps
gmt pscoast -R -J -O -K -Q >> $ps
gmt pscoast -R -J -O -K -Dc -A5000 -Wthinnest >> $ps
echo "0 20 10TH INTERNATIONAL" | gmt pstext -R -J -O -K -F+f32p,Helvetica-Bold,red=thinner >> $ps
echo "0 -10 GMT CONFERENCE" | gmt pstext -R -J -O -K -F+f32p,Helvetica-Bold,red=thinner >> $ps
echo "0 -30 Honolulu, Hawaii, April 1, 2013" | gmt pstext -R -J -O \
        -F+f18p,Helvetica-Bold,green=thinnest >> $ps

rm -f l*.nc l*.cpt gmt.conf
```

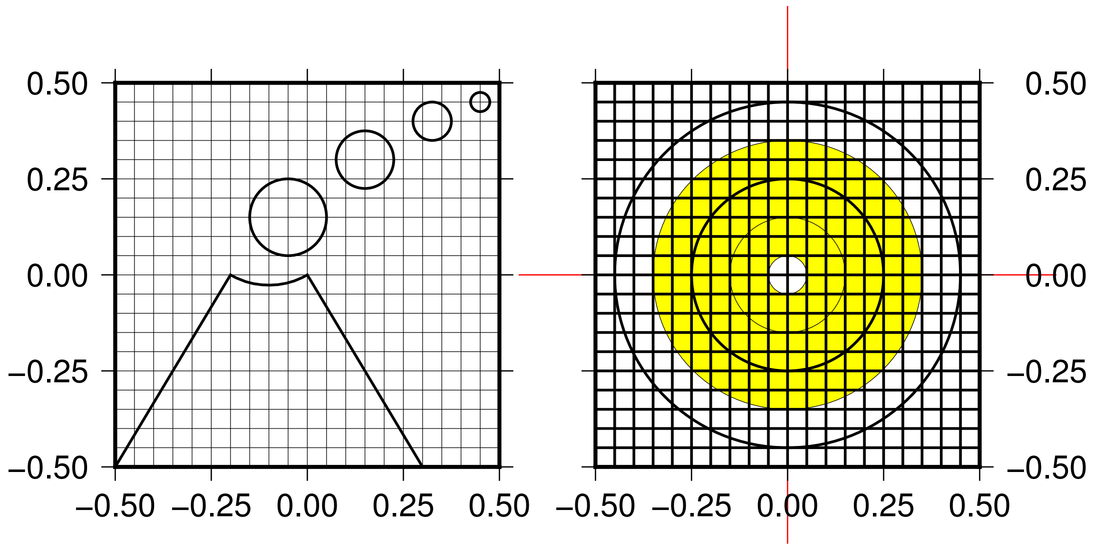
11.20 Custom plot symbols

One is often required to make special maps that shows the distribution of certain features but one would prefer to use a custom symbol instead of the built-in circles, squares, triangles, etc. in the GMT plotting programs `psxy` and `psxyz`. Here we demonstrate one approach that allows for a fair bit of flexibility


10TH INTERNATIONAL
GMT CONFERENCE
Honolulu, Hawaii, April 1, 2013
SILLY USES OF
GMT
COLOR PATTERNS
10TH INTERNATIONAL
GMT CONFERENCE
Honolulu, Hawaii, April 1, 2013

in designing ones own symbols. The following recipe is used when designing a new symbol.

1. Use `psbasemap` (or engineering paper!) to set up an empty grid that goes from -0.5 to +0.5 in both *x* and *y*. Use ruler and compass to draw your new symbol using straight lines, arcs of circles, and stand-alone geometrical objects (see `psxy` man page for a full description of symbol design). In this Section we will create two new symbols: a volcano and a bulls eye.



1. After designing the symbol we will encode it using a simple set of rules. In our case we describe our volcano and bulls eye using these three freeform polygon generators:

   x_0 y_0 *r* **C** [**-G***fill*] [**-W***pen*] Draw x_0 y_0 **M** [**-G***fill*] [**-W***pen*] Start new element at x_0, y_0

   x_1 y_1 **D** Draw straight line from current point to x_1, y_1 around (x_0, y_0)

   x_0 y_0 *r* α_1 α_2 **A** Draw arc segment of radius *r* from angle α_1 to α_2

   We also add a few stand-alone circles (for other symbols, see `psxy` man page):

   x_0 y_0 *r* **C** [**-G***fill*] [**-W***pen*] Draw x_0 y_0 *r* **c** [**-G***fill*] [**-W***pen*] Draw single circle of radius *r* around x_0, y_0

   The optional **-G** and **-W** can be used to hardwire the color fill and pen for segments (use **-** to disallow fill or line for any specific feature). By default the segments are painted based on the values of the command line settings.

   Manually applying these rules to our volcano symbol results in a definition file `volcano.def`:

   Without much further discussion we also make a definition file `bullseye.def` for a multi-colored bulls eye symbol. Note that the symbol can be created beyond the -0.5 to +0.5 range, as shown by the red lines. There is no limit in GMT to the size of the symbols. The center, however, will always be at (0,0). This is the point to which the coordinates in `psxy` refers.

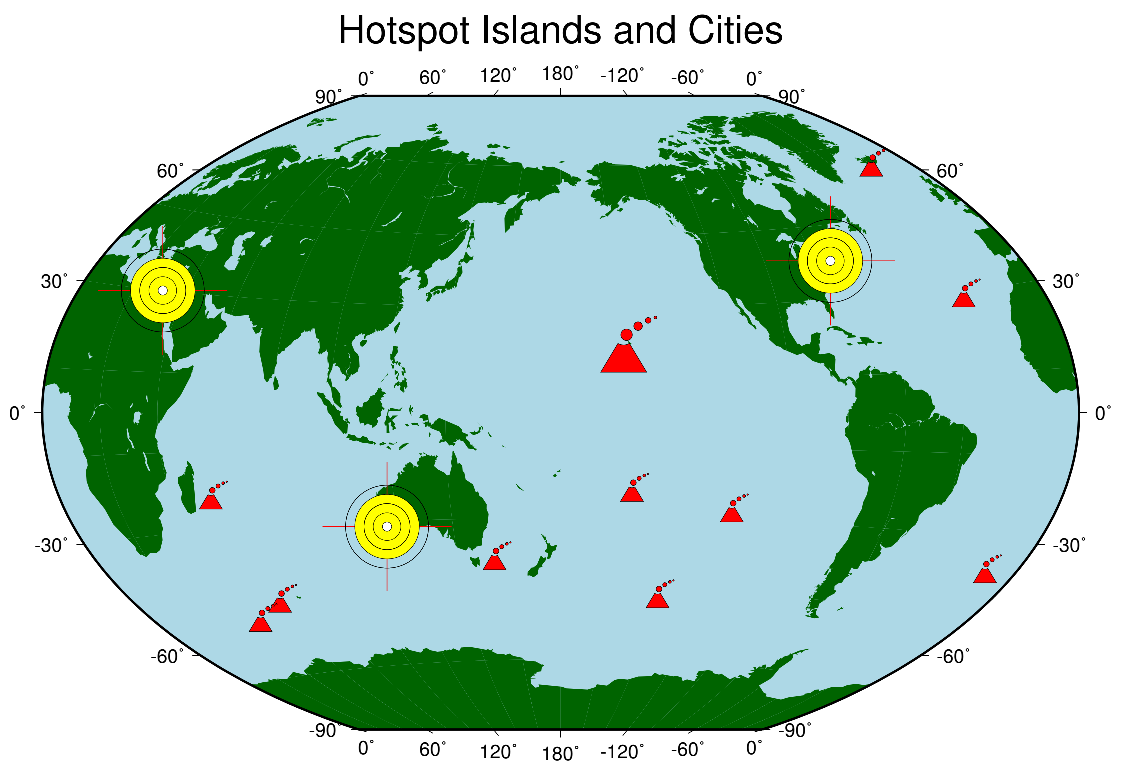
   The values refer to positions and dimensions illustrated in the Figure above.

2. Given proper definition files we may now use them with `psxy` or `psxyz`.

We are now ready to give it a try. Based on the hotspot locations in the file `hotspots.d` (with a 3rd column giving the desired symbol sizes in inches) we lay down a world map and overlay red volcano symbols using our custom-built volcano symbol and `psxy`. We do something similar with the bulls eye symbols. Without the **-G** option, however, they get the colors defined in `bullseye.def`.

Here is our final map script that produces Figure *Example 20*:

```
#!/bin/bash
#                GMT EXAMPLE 20
#
# Purpose:        Extend GMT to plot custom symbols
# GMT progs:        pscoast, psxy
# Unix progs:        rm
#
# Plot a world-map with volcano symbols of different sizes
# on top given locations and sizes in hotspots.d
ps=example_20.ps

cat > hotspots.d << END
55.5        -21.0        0.25
63.0        -49.0        0.25
-12.0        -37.0        0.25
-28.5        29.34        0.25
48.4        -53.4        0.25
155.5        -40.4        0.25
-155.5        19.6        0.5
-138.1        -50.9        0.25
-153.5        -21.0        0.25
-116.7        -26.3        0.25
-16.5        64.4        0.25
END

gmt pscoast -Rg -JR9i -Bx60 -By30 -B+t"Hotspot Islands and Cities" -Gdarkgreen -Slightblue \
        -Dc -A5000 -K > $ps

gmt psxy -R -J hotspots.d -Skvolcano -O -K -Wthinnest -Gred >> $ps

# Overlay a few bullseyes at NY, Cairo, and Perth

cat > cities.d << END
286        40.45        0.8
31.15        30.03        0.8
115.49        -31.58        0.8
END

gmt psxy -R -J cities.d -Skbullseye -O >> $ps

rm -f hotspots.d cities.d
```



Given these guidelines you can easily make your own symbols. Symbols with more than one color can be obtained by making several symbol components. E.g., to have yellow smoke coming out of red volcanoes we would make two symbols: one with just the cone and caldera and the other with the bubbles. These would be plotted consecutively using the desired colors. Alternatively, like in `bullseye.def`, we may specify colors directly for the various segments. Note that the custom symbols (Appendix [app:N]), unlike the built-in symbols in GMT, can be used with the built-in patterns (Appendix [app:E]). Other approaches are also possible, of course.

11.21 Time-series of RedHat stock price

As discussed in Section [sec:timeaxis], the annotation of time-series is generally more complicated due to the extra degrees of freedom afforded by the dual annotation system. In this example we will display the trend of the stock price of RedHat (RHAT) from their initial public offering until late 2006. The data file is a comma-separated table and the records look like this:

```
Date,Open,High,Low,Close,Volume,Adj.Close*
12-Mar-04,17.74,18.49,17.67,18.02,4827500,18.02
11-Mar-04,17.60,18.90,17.37,18.09,7700400,18.09
```

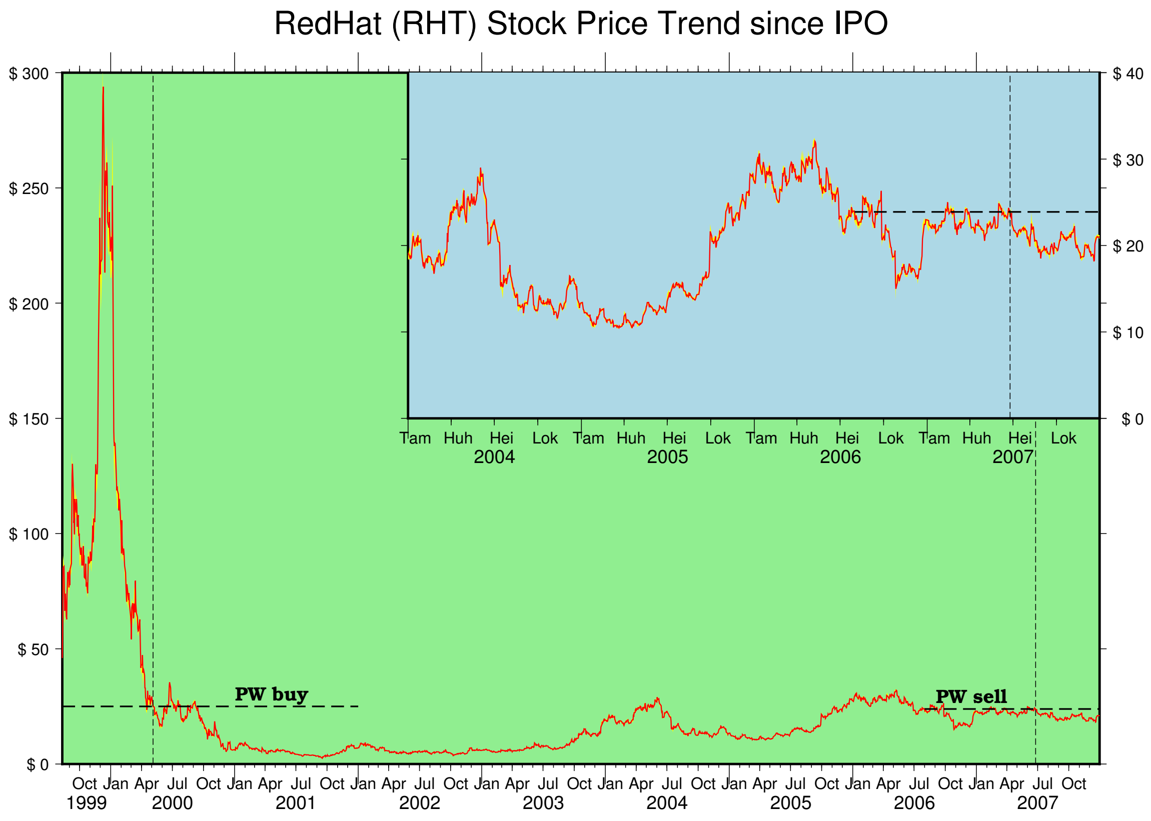
Hence, we have a single header record and various prices in USD for each day of business. We will plot the trend of the opening price as a red line superimposed on a yellow envelope representing the low-to-high fluctuation during each day. We also indicate when and at what cost Paul Wessel bought a few shares, and zoom in on the developments since 2004; in the inset we label the time-axis in Finnish in honor of Linus Thorvalds. Because the time coordinates are Y2K-challenged and the order is backwards (big units of years come *after* smaller units like days) we must change the default input/output formats used by GMT. Finally, we want to prefix prices with the $ symbol to indicate the currency. Here is how it all comes out:

```
#!/bin/bash
#               GMT EXAMPLE 21
#
# Purpose:       Plot a time-series
# GMT progs:       gmtset, gmtconvert, gmtinfo, psbasemap, psxy
# Unix progs:       cut, echo
#
ps=example_21.ps

# File has time stored as dd-Mon-yy so set input format to match it

gmt gmtset FORMAT_DATE_IN dd-o-yy FORMAT_DATE_MAP o FONT_ANNOT_PRIMARY +10p
gmt gmtset FORMAT_TIME_PRIMARY_MAP abbreviated PS_CHAR_ENCODING ISOLatin1+

# Pull out a suitable region string in yyy-mm-dd format

gmt info -fT -I50 -C RHAT_price.csv > RHAT.info
w=`cut -f1 RHAT.info`
e=`cut -f2 RHAT.info`
s=`cut -f3 RHAT.info`
n=`cut -f4 RHAT.info`
R="-R$w/$e/$s/$n"

# Lay down the basemap:

gmt psbasemap $R -JX9i/6i -K -Bsx1Y -Bpxa3Of1o -Bpy50+p"$ " \
        -BWSen+t"RedHat (RHT) Stock Price Trend since IPO"+glightgreen > $ps

# Plot main window with open price as red line over yellow envelope of low/highs

gmt gmtset FORMAT_DATE_OUT dd-o-yy
gmt gmtconvert -o0,2 -f0T RHAT_price.csv > RHAT.env
gmt gmtconvert -o0,3 -f0T -I -T RHAT_price.csv >> RHAT.env
gmt psxy -R -J -Gyellow -O -K RHAT.env >> $ps
gmt psxy -R -J RHAT_price.csv -Wthin,red -O -K >> $ps

# Draw P Wessel's purchase price as line and label it.  Note we temporary switch
# back to default yyyy-mm-dd format since that is what gmt info gave us.

echo "05-May-00         0" > RHAT.pw
echo "05-May-00         300" >> RHAT.pw
gmt psxy -R -J RHAT.pw -Wthinner,- -O -K >> $ps
echo "01-Jan-99         25" > RHAT.pw
echo "01-Jan-02         25" >> RHAT.pw
gmt psxy -R -J RHAT.pw -Wthick,- -O -K >> $ps
gmt gmtset FORMAT_DATE_IN yyyy-mm-dd
echo "$w 25 PW buy" | gmt pstext -R -J -O -K -D1.5i/0.05i -N -F+f12p,Bookman-Demi+jLB >> $ps
gmt gmtset FORMAT_DATE_IN dd-o-yy

# Draw P Wessel's sales price as line and label it.

echo "25-Jun-07         0" > RHAT.pw
echo "25-Jun-07         300" >> RHAT.pw
gmt psxy -R -J RHAT.pw -Wthinner,- -O -K >> $ps
echo "01-Aug-06         23.8852" > RHAT.pw
echo "01-Jan-08         23.8852" >> RHAT.pw
gmt psxy -R -J RHAT.pw -Wthick,- -O -K >> $ps
gmt gmtset FORMAT_DATE_IN yyyy-mm-dd
echo "$e 23.8852 PW sell" | gmt pstext -R -J -O -K -Dj0.8i/0.05i -N \
        -F+f12p,Bookman-Demi+jRB >> $ps
gmt gmtset FORMAT_DATE_IN dd-o-yy

# Get smaller region for insert for trend since 2004

R="-R2004T/$e/$s/40"

# Lay down the basemap, using Finnish annotations and place the insert in the upper right

gmt psbasemap --TIME_LANGUAGE=fi $R -JX6i/3i -Bpxa3Of3o -Bpy10+p"$ " -BESw+glightblue -Bsx1Y \
        -O -K -X3i -Y3i >> $ps

# Again, plot close price as red line over yellow envelope of low/highs

gmt psxy -R -J -Gyellow -O -K RHAT.env >> $ps
gmt psxy -R -J RHAT_price.csv -Wthin,red -O -K >> $ps

# Draw P Wessel's sales price as dashed line

gmt psxy -R -J RHAT.pw -Wthick,- -O -K >> $ps

# Mark sales date

echo "25-Jun-07         0" > RHAT.pw
echo "25-Jun-07         300" >> RHAT.pw
gmt psxy -R -J RHAT.pw -Wthinner,- -O >> $ps

# Clean up after ourselves:

rm -f RHAT.* gmt.conf
```

which produces the plot in Figure *Example 21*, suggesting Wessel has missed a few trains if he had hoped to cash in on the Internet bubble...

11.22 World-wide seismicity the last 7 days

The next example uses the command-line tool **wget** to obtain a data file from a specified URL. In the example script this line is commented out so the example will run even if you do not have **wget** (we use the supplied `neic_quakes.d` which normally would be created by **wget**); remove the comment to get the actual current seismicity plot using the live data. The main purpose of this script is not to show how to plot a map background and a few circles, but rather demonstrate how a map legend may be composed

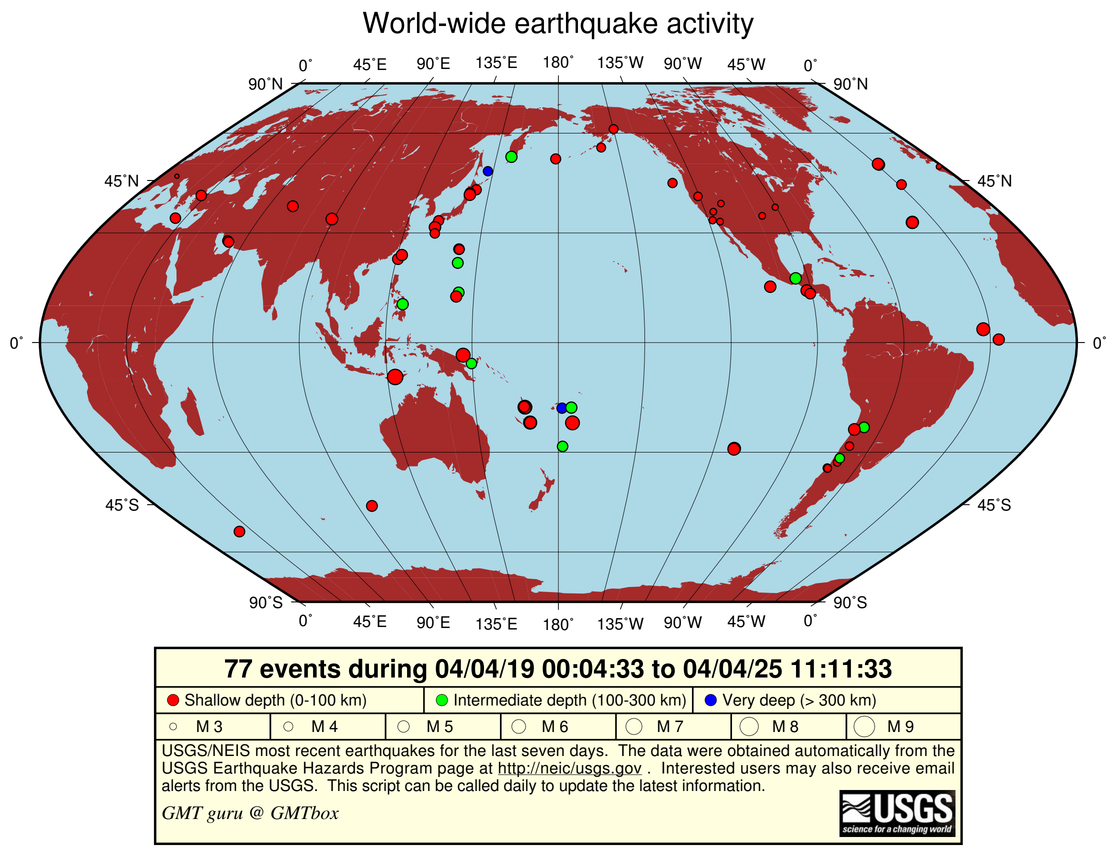


using the new tool `pslegend`. Some scripting is used to pull out information from the data file that is later used in the legend. The legend will normally have the email address of the script owner; here that command is commented out and the user is hardwired to "GMT guru". The USGS logo, taken from their web page and converted to a Sun raster file, is used to spice up the legend.

The script produces the plot in Figure *Example 22*, giving the URL where these and similar data can be obtained.

```
#!/bin/bash
#               GMT EXAMPLE 22
#
# Purpose:       Automatic map of last 7 days of world-wide seismicity
# GMT progs:       gmtset, pscoast, psxy, pslegend
# Unix progs:       cat, sed, awk, wget|curl
#
ps=example_22.ps
gmt gmtset FONT_ANNOT_PRIMARY 10p FONT_TITLE 18p FORMAT_GEO_MAP ddd:mm:ssF

# Get the data (-q quietly) from USGS using the wget (comment out in case
# your system does not have wget or curl)

#wget http://neic.usgs.gov/neis/gis/bulletin.asc -q -O neic_quakes.d
#curl http://neic.usgs.gov/neis/gis/bulletin.asc -s > neic_quakes.d

# Count the number of events (to be used in title later. one less due to header)

n=`cat neic_quakes.d | wc -l`
n=`expr $n - 1`

# Pull out the first and last timestamp to use in legend title

first=`sed -n 2p neic_quakes.d | $AWK -F, '{printf "%s %s\n", $1, $2}'`
last=`sed -n '$p' neic_quakes.d | $AWK -F, '{printf "%s %s\n", $1, $2}'`

# Assign a string that contains the current user @ the current computer node.
# Note that two @@ is needed to print a single @ in gmt pstext:
```

```
#set me = "$user@@`hostname`"
me="GMT guru @@ GMTbox"

# Create standard seismicity color table

cat > neis.cpt << END
0        red        100        red
100        green        300        green
300        blue        10000        blue
END

# Start plotting. First lay down map, then plot quakes with size = magintude/50":

gmt pscoast -Rg -JK180/9i -B45g30 -B+t"World-wide earthquake activity" -Gbrown -Slightblue \
        -Dc -A1000 -K -Y2.75i > $ps
$AWK -F, '{ print $4, $3, $6, $5*0.02}' neic_quakes.d \
        | gmt psxy -R -JK -O -K -Cneis.cpt -Sci -Wthin -h >> $ps
# Create legend input file for NEIS quake plot

cat > neis.legend << END
H 16 1 $n events during $first to $last
D 0 1p
N 3
V 0 1p
S 0.1i c 0.1i red 0.25p 0.2i Shallow depth (0-100 km)
S 0.1i c 0.1i green 0.25p 0.2i Intermediate depth (100-300 km)
S 0.1i c 0.1i blue 0.25p 0.2i Very deep (> 300 km)
V 0 1p
D 0 1p
N 7
V 0 1p
S 0.1i c 0.06i - 0.25p 0.3i M 3
S 0.1i c 0.08i - 0.25p 0.3i M 4
S 0.1i c 0.10i - 0.25p 0.3i M 5
S 0.1i c 0.12i - 0.25p 0.3i M 6
S 0.1i c 0.14i - 0.25p 0.3i M 7
S 0.1i c 0.16i - 0.25p 0.3i M 8
S 0.1i c 0.18i - 0.25p 0.3i M 9
V 0 1p
D 0 1p
N 1
END

# Put together a reasonable legend text, and add logo and user's name:

cat << END >> neis.legend
P
T USGS/NEIS most recent earthquakes for the last seven days.  The data were
T obtained automatically from the USGS Earthquake Hazards Program page at
T @_http://neic/usgs.gov @_.  Interested users may also receive email alerts
T from the USGS.
T This script can be called daily to update the latest information.
G 0.4i
# Add USGS logo
I USGS.ras 1i RT
G -0.3i
L 12 6 LB $me
END

# OK, now we can actually run gmt pslegend.  We center the legend below the map.
# Trial and error shows that 1.7i is a good legend height:

gmt pslegend -Dx4.5i/-0.4i/7i/1.7i/TC -O -F+p+glightyellow neis.legend  >> $ps

# Clean up after ourselves:

rm -f neis.* gmt.conf
```



77 events during 04/04/19 00:04:33 to 04/04/25 11:11:33

● Shallow depth (0-100 km) ● Intermediate depth (100-300 km) ● Very deep (> 300 km)

○ M 3 ○ M 4 ○ M 5 ○ M 6 ○ M 7 ○ M 8 ○ M 9

USGS/NEIS most recent earthquakes for the last seven days. The data were obtained automatically from the USGS Earthquake Hazards Program page at http://neic/usgs.gov . Interested users may also receive email alerts from the USGS. This script can be called daily to update the latest information.

GMT guru @ GMTbox

11.23 All great-circle paths lead to Rome

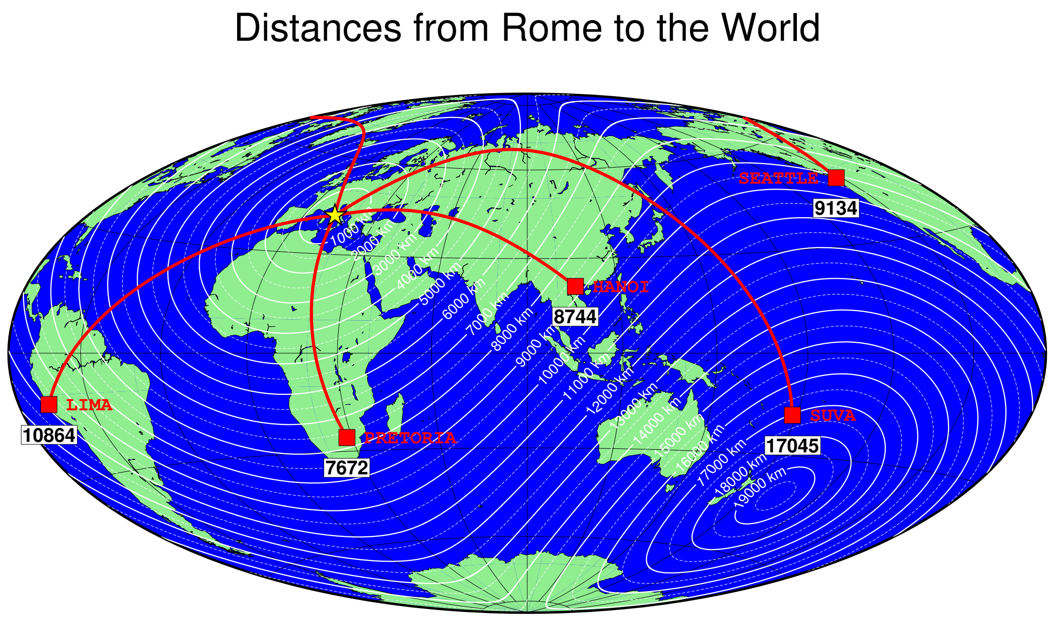
While motorists recently have started to question the old saying “all roads lead to Rome”, aircraft pilots have known from the start that only one great-circle path connects the points of departure and arrival [3]. This provides the inspiration for our next example which uses `grdmath` to calculate distances from Rome to anywhere on Earth and `grdcontour` to contour these distances. We pick five cities that we connect to Rome with great circle arcs, and label these cities with their names and distances (in km) from Rome, all laid down on top of a beautiful world map. Note that we specify that contour labels only be placed along the straight map-line connecting Rome to its antipode, and request curved labels that follows the shape of the contours.

The script produces the plot in Figure *Example 23*; note how interesting the path to Seattle appears in this particular projection (Hammer). We also note that Rome’s antipode lies somewhere near the Chatham plateau (antipodes will be revisited in Section [sec:example$_2$5]).

```
#!/bin/bash
#                 GMT EXAMPLE 23
#
# Purpose:         Plot distances from Rome and draw shortest paths
# GMT progs:         grdmath, grdcontour, pscoast, psxy, pstext, grdtrack
# Unix progs:         echo, cat, awk
#
ps=example_23.ps

# Position and name of central point:

lon=12.50
lat=41.99
name="Rome"
```

[3] Pedants who wish to argue about the “other” arc going the long way should consider using it.

```
# Calculate distances (km) to all points on a global 1x1 grid

gmt grdmath -Rg -I1 $lon $lat SDIST = dist.nc

# Location info for 5 other cities + label justification

cat << END > cities.d
105.87        21.02        HANOI                LM
282.95        -12.1        LIMA                LM
178.42        -18.13        SUVA                LM
237.67        47.58        SEATTLE                RM
28.20        -25.75        PRETORIA        LM
END

gmt pscoast -Rg -JH90/9i -Glightgreen -Sblue -A1000 -Dc -Bg30 \
        -B+t"Distances from $name to the World" -K -Wthinnest > $ps

gmt grdcontour dist.nc -A1000+v+u" km"+fwhite -Glz-/z+ -S8 -C500 -O -K -J \
        -Wathin,white -Wcthinnest,white,- >> $ps

# For each of the cities, plot great circle arc to Rome with gmt psxy

while read clon clat city; do
        (echo $lon $lat; echo $clon $clat) | gmt psxy -R -J -O -K -Wthickest,red >> $ps
done < cities.d

# Plot red squares at cities and plot names:
gmt psxy -R -J -O -K -Ss0.2 -Gred -Wthinnest cities.d >> $ps
$AWK '{print $1, $2, $4, $3}' cities.d | gmt pstext -R -J -O -K -Dj0.15/0 \
        -F+f12p,Courier-Bold,red+j -N >> $ps
# Place a yellow star at Rome
echo "$lon $lat" | gmt psxy -R -J -O -K -Sa0.2i -Gyellow -Wthin >> $ps

# Sample the distance grid at the cities and use the distance in km for labels

gmt grdtrack -Gdist.nc cities.d \
        | $AWK '{printf "%s %s %d\n", $1, $2, int($NF+0.5)}' \
        | gmt pstext -R -J -O -D0/-0.2i -N -Gwhite -W -C0.02i -F+f12p,Helvetica-Bold+jCT >> $ps

# Clean up after ourselves:

rm -f cities.d dist.nc
```



11.24 Data selection based on geospatial criteria

Although we are not seismologists, we have yet another example involving seismicity. We use seismicity data for the Australia/New Zealand region to demonstrate how we can extract subsets of data using geospatial criteria. In particular, we wish to plot the epicenters given in the file `oz_quakes.d` as red or green circles. Green circles should only be used for epicenters that satisfy the following three criteria:

1. They are located in the ocean and not on land
2. They are within 3000 km of Hobart
3. They are more than 1000 km away from the International Dateline

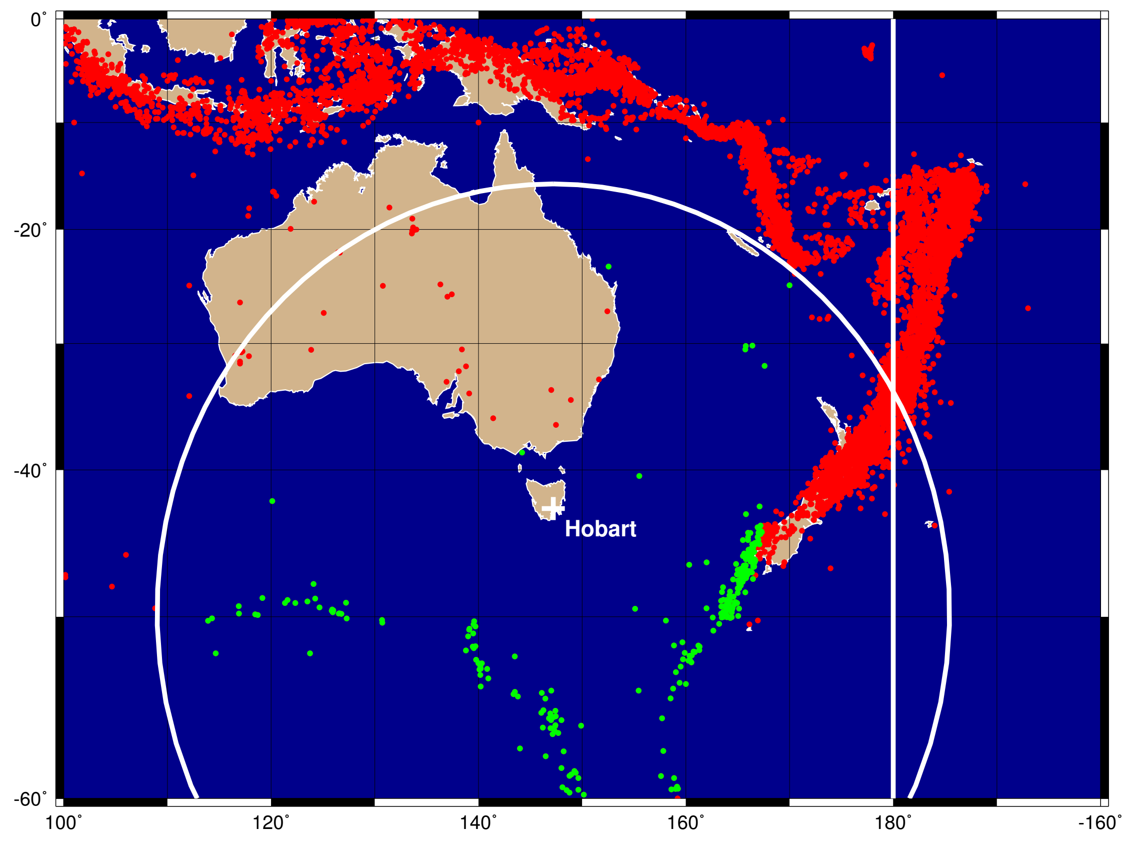
All remaining earthquakes should be plotted in red. Rather that doing the selection process twice we simply plot all quakes as red circles and then replot those that pass our criteria. Most of the work here is done by `gmtselect`; the rest is carried out by the usual `pscoast` and `psxy` workhorses. Note for our purposes the Dateline is just a line along the 180 meridian.

The script produces the plot in Figure *Example 24*. Note that the horizontal distance from the dateline seems to increase as we go south; however that is just the projected distance (Mercator distortion) and not the actual distance which remains constant at 1000 km.

```
#!/bin/bash
#               GMT EXAMPLE 24
#
# Purpose:        Extract subsets of data based on geospatial criteria
# GMT progs:        gmtselect, pscoast, psxy, gmtinfo
# Unix progs:        echo, cat, awk
#
# Highlight oceanic earthquakes within 3000 km of Hobart and > 1000 km from dateline
ps=example_24.ps
echo "147:13 -42:48 6000 Hobart" > point.d
cat << END > dateline.d
> Our proxy for the dateline
180       0
180       -90
END
R=`gmt info -I10 oz_quakes.d`
gmt pscoast $R -JM9i -K -Gtan -Sdarkblue -Wthin,white -Dl -A500 -Ba20f10g10 -BWeSn > $ps
gmt psxy -R -J -O -K oz_quakes.d -Sc0.05i -Gred >> $ps
gmt gmtselect oz_quakes.d -L1000k/dateline.d -Nk/s -C3000k/point.d -fg -R -Il \
        | gmt psxy -R -JM -O -K -Sc0.05i -Ggreen >> $ps
gmt psxy point.d -R -J -O -K -SE- -Wfat,white >> $ps
$AWK '{print $1, $2, $4}' point.d | gmt pstext -R -J -O -K -F+f14p,Helvetica-Bold,white+jLT \
        -D0.1i/-0.1i >> $ps
gmt psxy -R -J -O -K point.d -Wfat,white -S+0.2i >> $ps
gmt psxy -R -J -O dateline.d -Wfat,white -A >> $ps
rm -f point.d dateline.d
```

11.25 Global distribution of antipodes

As promised in Section [sec:example$_2$3], we will study antipodes. The antipode of a point at (ϕ, λ) is the point at $(-\phi, \lambda + 180)$. We seek an answer to the question that has plagued so many for so long: Given the distribution of land and ocean, how often is the antipode of a point on land also on land? And what about marine antipodes? We use `grdlandmask` and `grdmath` to map these distributions and calculate the area of the Earth (in percent) that goes with each of the three possibilities. To make sense of our `grdmath` equations below, note that we first calculate a grid that is +1 when a point and its antipode is on land, -1 if both are in the ocean, and 0 elsewhere. We then seek to calculate the area distribution of dry antipodes by only pulling out the nodes that equal +1. As each point represent an area approximated by $\Delta\phi \times \Delta\lambda$ where the $\Delta\lambda$ term's actual dimension depends on $\cos(\phi)$, we need to allow



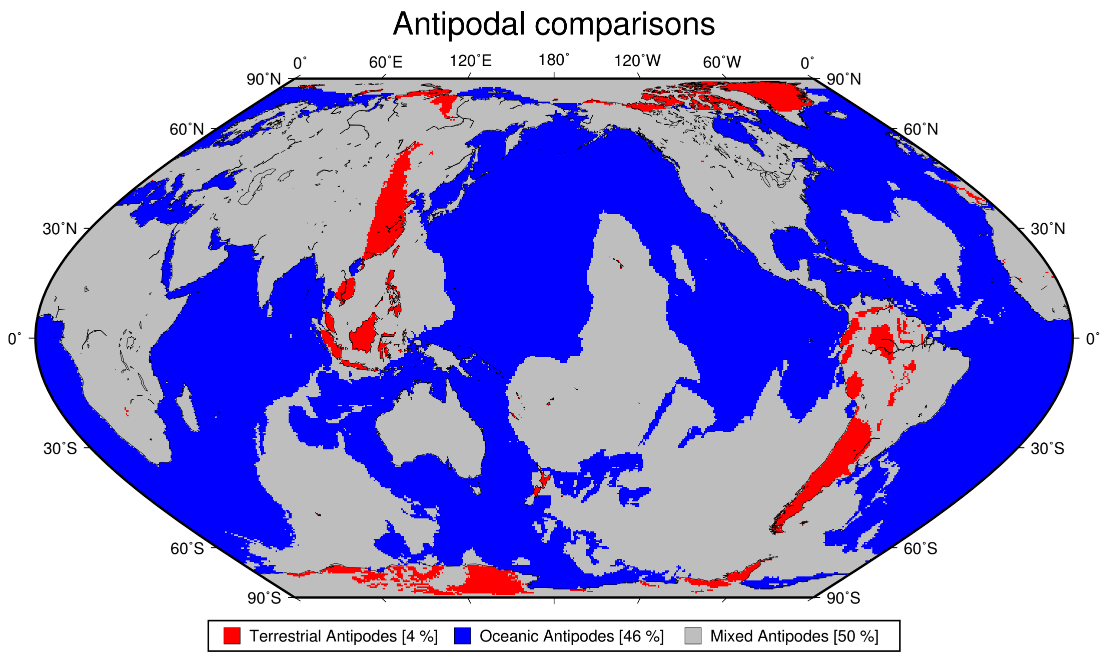
for that shrinkage, normalize our sum to that of the whole area of the Earth, and finally convert that ratio to percent. Since the $\Delta\lambda$, $\Delta\phi$ terms appear twice in these expressions they cancel out, leaving the somewhat intractable expressions below where the sum of $\cos(\phi)$ for all ϕ is known to equal $2N_y/\pi$:

In the end we obtain a funny-looking map depicting the antipodal distribution as well as displaying in legend form the requested percentages (Figure *Example 25*). Note that the script is set to evaluate a global 30 minute grid for expediency (*D = 30*), hence several smaller land masses that do have terrestrial antipodes do not show up. If you want a more accurate map you can set the parameter *D* to a smaller increment (try 5 and wait a few minutes).

The call to `grdimage` includes the `-Sn` to suspend interpolation and only return the value of the nearest neighbor. This option is particularly practical for plotting categorical data, like these, that should not be interpolated.

```
#!/bin/bash
#               GMT EXAMPLE 25
#
# Purpose:        Display distribution of antipode types
# GMT progs:        gmtset, grdlandmask, grdmath, grd2xyz, gmtmath, grdimage, pscoast, pslegend
# Unix progs:         cat
#
# Create D minutes global grid with -1 over oceans and +1 over land
ps=example_25.ps
D=30
gmt grdlandmask -Rg -Im -Dc -A500 -N-1/1/1/1/1 -r -Gwetdry.nc
# Manipulate so -1 means ocean/ocean antipode, +1 = land/land, and 0 elsewhere
gmt grdmath -fg wetdry.nc DUP 180 ROTX FLIPUD ADD 2 DIV = key.nc
# Calculate percentage area of each type of antipode match.
gmt grdmath -Rg -Im -r Y COSD 60 $D DIV 360 MUL DUP MUL PI DIV DIV 100 MUL = scale.nc
gmt grdmath -fg key.nc -1 EQ 0 NAN scale.nc MUL = tmp.nc
gmt grd2xyz tmp.nc -s -ZTLf > key.b
ocean=`gmt gmtmath -bi1f -Ca -S key.b SUM UPPER RINT =`
gmt grdmath -fg key.nc 1 EQ 0 NAN scale.nc MUL = tmp.nc
gmt grd2xyz tmp.nc -s -ZTLf > key.b
land=`gmt gmtmath -bi1f -Ca -S key.b SUM UPPER RINT =`
```

```
gmt grdmath -fg key.nc 0 EQ 0 NAN scale.nc MUL = tmp.nc
gmt grd2xyz tmp.nc -s -ZTLf > key.b
mixed=`gmt gmtmath -bi1f -Ca -S key.b SUM UPPER RINT =`
# Generate corresponding color table
cat << END > key.cpt
-1.5        blue        -0.5        blue
-0.5        gray        0.5        gray
0.5        red        1.5        red
END
# Create the final plot and overlay coastlines
gmt gmtset FONT_ANNOT_PRIMARY +10p FORMAT_GEO_MAP dddF
gmt grdimage key.nc -JKs180/9i -Bx60 -By30 -BWsNE+t"Antipodal comparisons" -K -Ckey.cpt -Y1.2i -nn > $ps
gmt pscoast -R -J -O -K -Wthinnest -Dc -A500 >> $ps
# Place an explanatory legend below
gmt pslegend -R0/9/0/0.5 -Jx1i -O -Dx4.5i/0/6i/TC -Y-0.2i -F+pthick >> $ps << END
N 3
S 0.15i s 0.2i red  0.25p 0.3i Terrestrial Antipodes [$land %]
S 0.15i s 0.2i blue 0.25p 0.3i Oceanic Antipodes [$ocean %]
S 0.15i s 0.2i gray 0.25p 0.3i Mixed Antipodes [$mixed %]
END
rm -f *.nc key.* gmt.conf
```



11.26 General vertical perspective projection

Next, we present a recent extension to the **-JG** projection option which allows the user to specify a particular altitude (this was always at infinity before), as well as several further parameters to limit the view from the chosen vantage point. In this example we show a view of the eastern continental US from a height of 160 km. Below we add a view with a specific tilt of 55 and azimuth 210; here we have chosen a boresight twist of 45. We view the land from New York towards Washington, D.C.

At this point the full projection has not been properly optimized and the map annotations will need additional work. Also, note that the projection is only implemented in `pscoast` and `grdimage`. We hope to refine this further and extend the availability of the full projection to all of the GMT mapping programs.

```
#!/bin/bash
#               GMT EXAMPLE 26
#
```

```
# Purpose:        Demonstrate general vertical perspective projection
# GMT progs:        pscoast
# Unix progs:        rm
#
ps=example_26.ps

# first do an overhead of the east coast from 160 km altitude point straight down

latitude=41.5
longitude=-74.0
altitude=160.0
tilt=0
azimuth=0
twist=0
Width=0.0
Height=0.0

PROJ=-JG///////4i

gmt pscoast -Rg $PROJ -X1i -B5g5 -Glightbrown -Slightblue -W -Dl -N1/1p,red -N2,0.5p -P -K \
        -Y5i > $ps

# now point from an altitude of 160 km with a specific tilt and azimuth and with a wider restricted
# view and a boresight twist of 45 degrees

tilt=55
azimuth=210
twist=45
Width=30.0
Height=30.0

PROJ=-JG///////5i

gmt pscoast -R $PROJ -B5g5 -Glightbrown -Slightblue -W -Ia/blue -Di -Na -O -X1i -Y-4i >> $ps
```

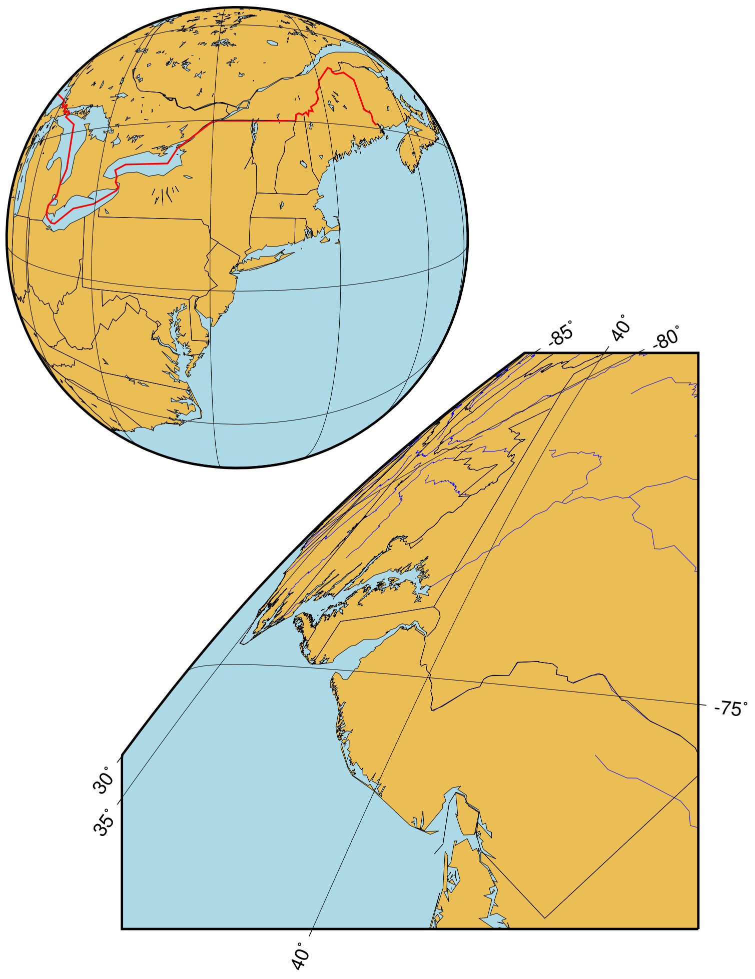
11.27 Plotting Sandwell/Smith Mercator img grids

Next, we show how to plot a data grid that is distributed in projected form. The gravity and predicted bathymetry grids produced by David Sandwell and Walter H. F. Smith are not geographical grids but instead given on a spherical Mercator grid. The GMT supplement img has tools to extract subsets of these large grids. If you need to make a non-Mercator map then you must extract a geographic grid using `supplements/img/img2grd` and then plot it using your desired map projection. However, if you want to make a Mercator map then you can save time and preserve data quality by avoiding to re-project the data set twice since it is already in a Mercator projection. This example shows how this is accomplished. We use the **-M** option in `supplements/img/img2grd` to pull out the grid in Mercator units (i.e., do *not* invert the Mercator projection) and then simply plot the grid using a linear projection with a suitable scale (here 0.25 inches per degrees of longitude). To overlay basemaps and features that has geographic longitude/latitude coordinates we must remember two key issues:

1. This is a *spherical* Mercator grid so we must use –**PROJ_ELLIPSOID**=Sphere with all commands that involve projections (or use `gmtset` to change the setting).
2. Select Mercator projection and use the same scale that was used with the linear projection.

This map of the Tasman Sea shows the marine gravity anomalies with land painted black. A color scale bar was then added to complete the illustration.

```
#!/bin/bash
#                GMT EXAMPLE 27
#
# Purpose:        Illustrates how to plot Mercator img grids
# GMT progs:        makecpt, mapproject, grdgradient, grdimage, grdinfo, pscoast
# GMT supplement: img2grd (to read Sandwell/Smith img files)
```


-85°
40°
-80°
-75°
30°
35°
40°

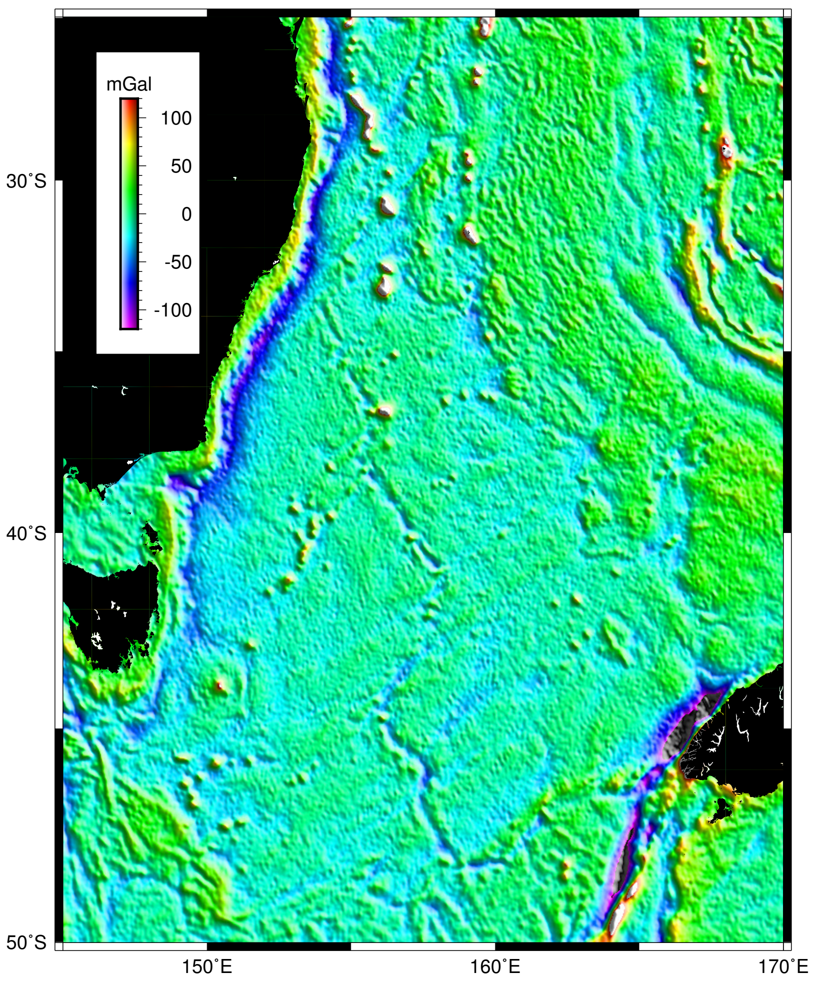
```
# Unix progs:        rm, grep, $AWK
#
ps=example_27.ps

# Gravity in tasman_grav.nc is in 0.1 mGal increments and the grid
# is already in projected Mercator x/y units.
# First get gradients.

gmt grdgradient tasman_grav.nc -Nt1 -A45 -Gtasman_grav_i.nc

# Make a suitable cpt file for mGal

gmt makecpt -T-120/120/240 -Z -Crainbow > grav.cpt

# Since this is a Mercator grid we use a linear projection

gmt grdimage tasman_grav.nc=ns/0.1 -Itasman_grav_i.nc -Jx0.25i -Cgrav.cpt -P -K > $ps

# Then use gmt pscoast to plot land; get original -R from grid remark
# and use Mercator gmt projection with same scale as above on a spherical Earth

R=`gmt grdinfo tasman_grav.nc | grep Remark | $AWK '{print $NF}'`

gmt pscoast $R -Jm0.25i -Ba10f5 -BWSne -O -K -Gblack --PROJ_ELLIPSOID=Sphere \
        -Cwhite -Dh+ --FORMAT_GEO_MAP=dddF >> $ps

# Put a color legend on top of the land mask justified with 147E,31S

pos=`echo 147E 31S | gmt mapproject -R -J -Di --PROJ_ELLIPSOID=Sphere | \
        $AWK '{printf "%si/%si\n", $1, $2}'`
gmt psscale -D$pos/2i/0.15i -Cgrav.cpt -Bx50f10 -By+lmGal -I -O -T+gwhite+p1p >> $ps

# Clean up

rm -f grav.cpt *_i.nc
```

11.28 Mixing UTM and geographic data sets

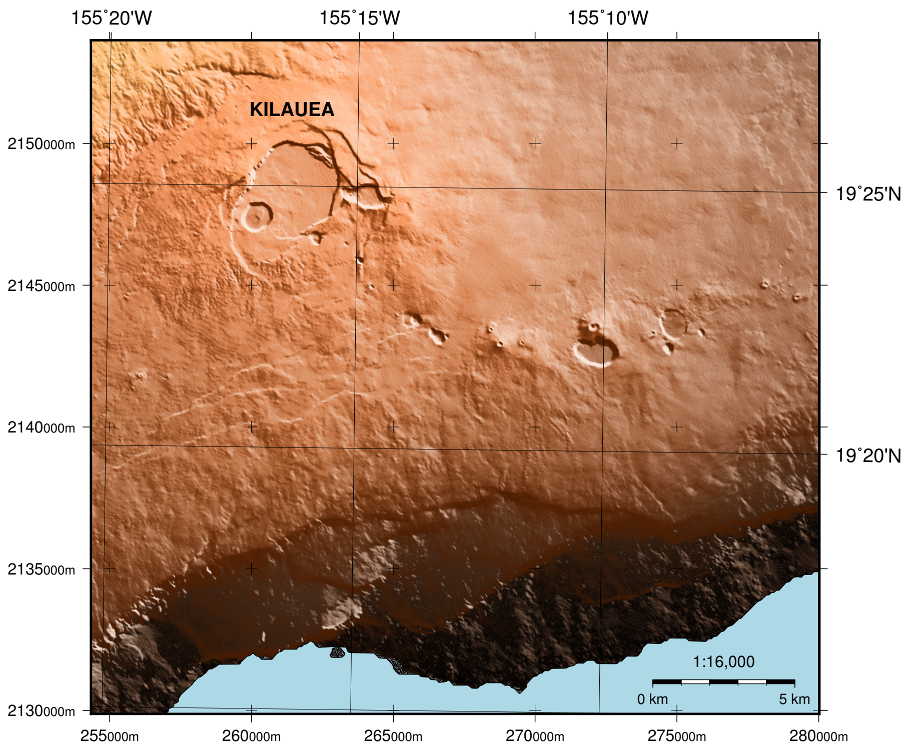
Next, we present a similar case: We wish to plot a data set given in UTM coordinates (meter) and want it to be properly registered with overlying geographic data, such as coastlines or data points. The mistake many GMT rookies make is to specify the UTM projection with their UTM data. However, that data have already been projected and is now in linear meters. The only sensible way to plot such data is with a linear projection, yielding a UTM map. In this step one can choose to annotate or tick the map in UTM meters as well. To plot geographic (lon/lat) data on the same map you simply have to specify the region using the UTM meters but supply the actual UTM projection parameters. Make sure you use the same scale with both the linear and UTM projection.

Our script illustrates how we would plot a UTM grid (with coordinates in meters) of elevations near Kilauea volcano on the Big Island of Hawaii and overlay geographical information (with longitude, latitude coordinates). We first lay down the UTM grid using the linear projection. Then, given we are in UTM zone 5Q, we use the UTM domain and the UTM projection when overlaying the coastline and light blue ocean. We do some trickery by converting the UTM domain to km so that we can add custom annotations to the map. Finally, we place a scale bar and label Kilauea crater to complete the figure.

```
#!/bin/bash
#                GMT EXAMPLE 28
#
# Purpose:        Illustrates how to mix UTM data and UTM gmt projection
# GMT progs:        makecpt, grdgradient, grdimage, grdinfo, grdmath, pscoast, pstext, mapproject
# Unix progs:        rm, echo
#
ps=example_28.ps
```


mGal
100
50
0
-50
-100
30˚S
40˚S
50˚S
150˚E
160˚E
170˚E

```
# Get intensity grid and set up a color table
gmt grdgradient Kilauea.utm.nc -Nt1 -A45 -GKilauea.utm_i.nc
gmt makecpt -Ccopper -T0/1500/100 -Z > Kilauea.cpt
# Lay down the UTM topo grid using a 1:16,000 scale
gmt grdimage Kilauea.utm.nc -IKilauea.utm_i.nc -CKilauea.cpt -Jx1:160000 -P -K \
        --FORMAT_FLOAT_OUT=%.10g --FONT_ANNOT_PRIMARY=9p > $ps
# Overlay geographic data and coregister by using correct region and gmt projection with the same scale
gmt pscoast -RKilauea.utm.nc -Ju5Q/1:160000 -O -K -Df+ -Slightblue -W0.5p -B5mg5m -BNE \
        --FONT_ANNOT_PRIMARY=12p --FORMAT_GEO_MAP=ddd:mmF >> $ps
echo 155:16:20W 19:26:20N KILAUEA | gmt pstext -R -J -O -K -F+f12p,Helvetica-Bold+jCB >> $ps
gmt psbasemap -R -J -O -K --FONT_ANNOT_PRIMARY=9p -Lf155:07:30W/19:15:40N/19:23N/5k+l1:16,000+u \
        --FONT_LABEL=10p >> $ps
# Annotate in km but append ,000m to annotations to get customized meter labels
gmt psbasemap -RKilauea.utm.nc+Uk -Jx1:160 -B5g5+u"@:8:000m" -BWSne -O --FONT_ANNOT_PRIMARY=10p \
        --MAP_GRID_CROSS_SIZE_PRIMARY=0.1i --FONT_LABEL=10p >> $ps
# Clean up
rm -f Kilauea.utm_i.nc Kilauea.cpt tmp.txt
```



11.29 Gridding spherical surface data using splines

Finally, we demonstrate how gridding on a spherical surface can be accomplished using Green's functions of surface splines, with or without tension. Global gridding does not work particularly well in Cartesian coordinates hence the chosen approach. We use greenspline to produce a crude topography grid for Mars based on radii estimates from the Mariner 9 and Viking Orbiter spacecrafts. This data comes from *Smith and Zuber* [Science, 1996] and is used here as a small (N = 370) data set we can use to demonstrate spherical surface gridding. Since greenspline must solve a N by N matrix system your system memory may impose limits on how large data sets you can handle; also note that the spherical surface spline in tension is particularly slow to compute.

Our script must first estimate the ellipsoidal shape of Mars from the parameters given by *Smith and Zuber* so that we can remove this reference surface from the gridded radii. We run the gridding twice: First with no tension using *Parker*'s [1990] method and then with tension using the *Wessel and Becker* [2008] method. The grids are then imaged with grdimage and grdcontour and a color scale is placed between them.

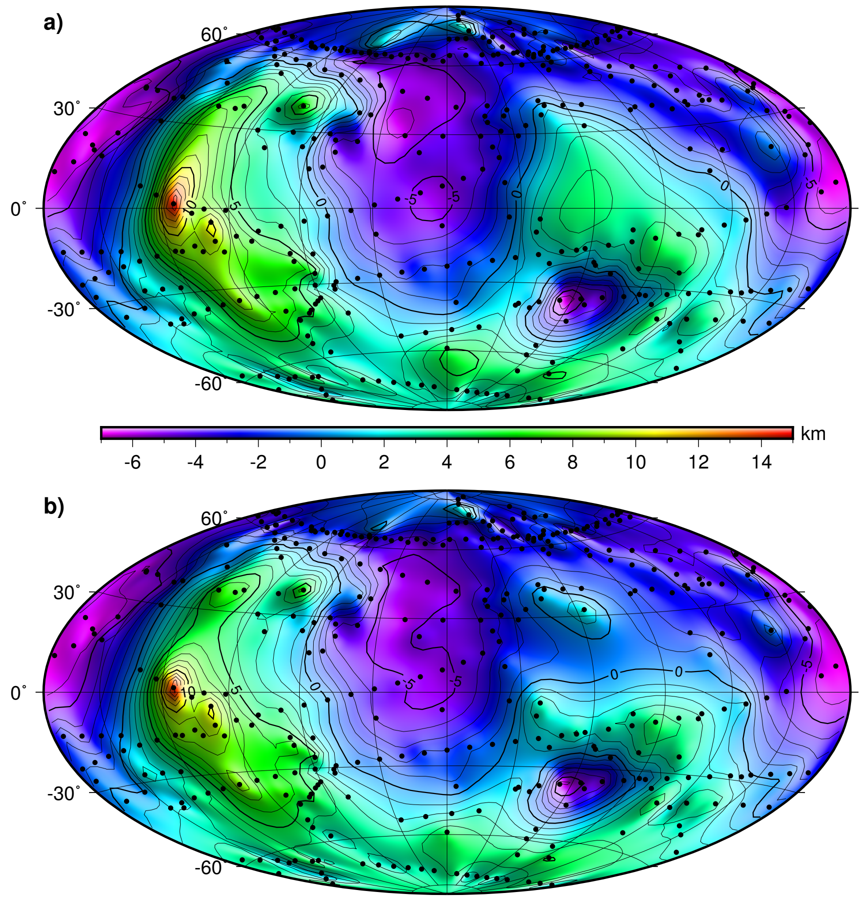
```
#!/bin/bash
#               GMT EXAMPLE 29
#
# Purpose:        Illustrates spherical surface gridding with Green's function of splines
# GMT progs:        makecpt, grdcontour, grdgradient, grdimage, grdmath greenspline, psscale, pstext
# Unix progs:        rm, echo
#
ps=example_29.ps

# This example uses 370 radio occultation data for Mars to grid the topography.
# Data and information from Smith, D. E., and M. T. Zuber (1996), The shape of
# Mars and the topographic signature of the hemispheric dichotomy, Science, 271, 184-187.

# Make Mars PROJ_ELLIPSOID given their three best-fitting axes:
a=3399.472
b=3394.329
c=3376.502
gmt grdmath -Rg -I4 -r X COSD $a DIV DUP MUL X SIND $b DIV DUP MUL ADD Y COSD DUP MUL MUL Y \
        SIND $c DIV DUP MUL ADD SQRT INV = PROJ_ELLIPSOID.nc

#  Do both Parker and Wessel/Becker solutions (tension = 0.9975)
gmt greenspline -RPROJ_ELLIPSOID.nc mars370.in -D4 -Sp -Gmars.nc
gmt greenspline -RPROJ_ELLIPSOID.nc mars370.in -D4 -Sq0.9975 -Gmars2.nc
# Scale to km and remove PROJ_ELLIPSOID
gmt grdmath mars.nc 1000 DIV PROJ_ELLIPSOID.nc SUB = mars.nc
gmt grdmath mars2.nc 1000 DIV PROJ_ELLIPSOID.nc SUB = mars2.nc
gmt makecpt -Crainbow -T-7/15/22 -Z > mars.cpt
gmt grdgradient mars2.nc -fg -Ne0.75 -A45 -Gmars2_i.nc
gmt grdimage mars2.nc -Imars2_i.nc -Cmars.cpt -B30g30 -BWsne -JH0/7i -P -K -E200 \
        --FONT_ANNOT_PRIMARY=12p -X0.75i > $ps
gmt grdcontour mars2.nc -J -O -K -C1 -A5 -Glz+/z- >> $ps
gmt psxy -Rg -J -O -K -Sc0.045i -Gblack mars370.in  >> $ps
echo "0 90 b)" | gmt pstext -R -J -O -K -N -D-3.5i/-0.2i -F+f14p,Helvetica-Bold+jLB >> $ps
gmt grdgradient mars.nc -fg -Ne0.75 -A45 -Gmars_i.nc
gmt grdimage mars.nc -Imars_i.nc -Cmars.cpt -B30g30 -BWsne -J -O -K -Y4.2i -E200 \
        --FONT_ANNOT_PRIMARY=12p >> $ps
gmt grdcontour mars.nc -J -O -K -C1 -A5 -Glz+/z- >> $ps
gmt psxy -Rg -J -O -K -Sc0.045i -Gblack mars370.in  >> $ps
gmt psscale -Cmars.cpt -O -K -D3.5i/-0.15i/6i/0.1ih -I --FONT_ANNOT_PRIMARY=12p -Bx2f1 -By+lkm >> $ps
echo "0 90 a)" | gmt pstext -R -J -O -N -D-3.5i/-0.2i -F+f14p,Helvetica-Bold+jLB >> $ps
# Clean up
rm -f *.nc mars.cpt
```

11.30 Trigonometric functions plotted in graph mode

Finally, we end with a simple mathematical illustration of sine and cosine, highlighting the *graph* mode for linear projections and the new curved vectors for angles.

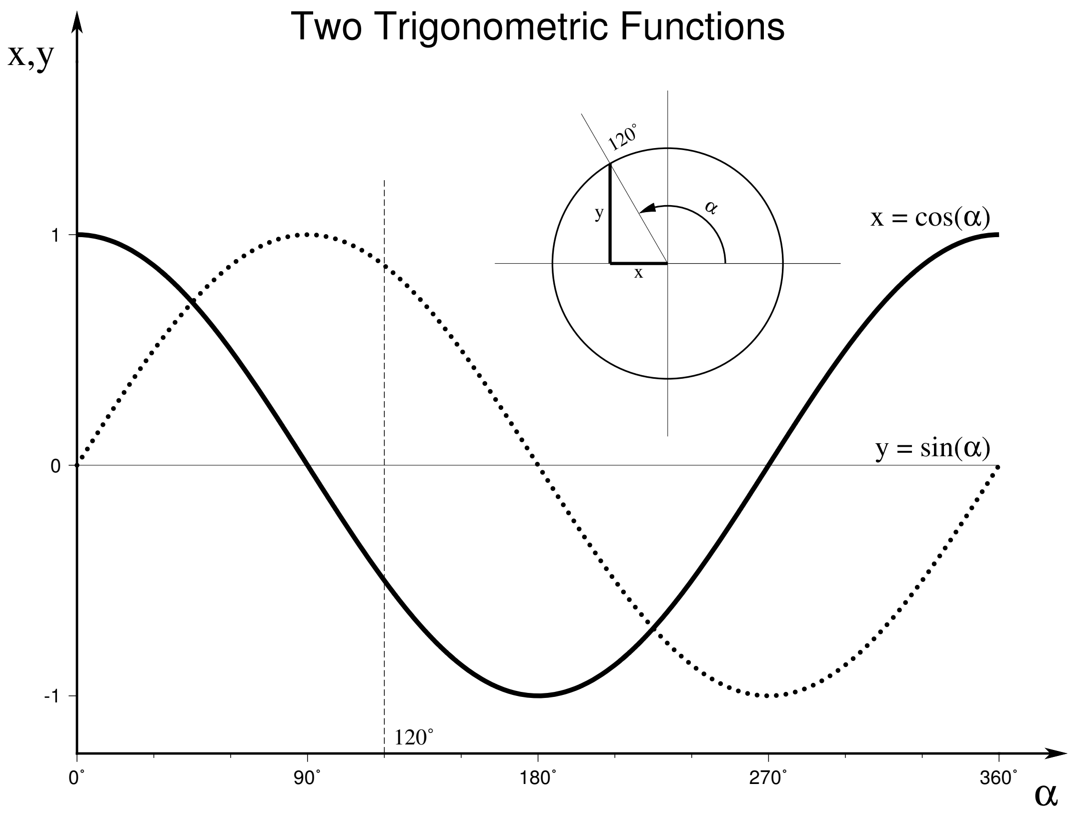
The script simply draws a graph basemap, computes sine and cosine and plots them as lines, then indicates on a circle that these quantities are simply the projections of an unit vector on the x- and y-axis, at the given angle.

```
#!/bin/bash
#               GMT EXAMPLE 30
#
# Purpose:        Show graph mode and math angles
# GMT progs:        gmtmath, psbasemap, pstext and psxy
# Unix progs:        echo, rm
#
# Draw generic x-y axes with arrows
ps=example_30.ps

gmt psbasemap -R0/360/-1.25/1.75 -JX8i/6i -Bx90f30+u"\\312" -By1g10 -BWS+t"Two Trigonometric Functions" \
        -K --MAP_FRAME_TYPE=graph --MAP_VECTOR_SHAPE=0.5 > $ps

# Draw sine an cosine curves
```



```
gmt gmtmath -T0/360/0.1 T COSD = | gmt psxy -R -J -O -K -W3p >> $ps
gmt gmtmath -T0/360/0.1 T SIND = | gmt psxy -R -J -O -K -W3p,0_6:0 --PS_LINE_CAP=round >> $ps

# Indicate the x-angle = 120 degrees
gmt psxy -R -J -O -K -W0.5p,- << EOF >> $ps
120         -1.25
120         1.25
EOF

gmt pstext -R -J -O -K -Dj0.2c -N -F+f+j << EOF >> $ps
360 1 18p,Times-Roman RB x = cos(@%12%a@%%)
360 0 18p,Times-Roman RB y = sin(@%12%a@%%)
120 -1.25 14p,Times-Roman LB 120\\312
370 -1.35 24p,Symbol LT a
-5 1.85 24p,Times-Roman RT x,y
EOF

# Draw a circle and indicate the 0-70 degree angle

echo 0 0 | gmt psxy -R-1/1/-1/1 -Jx1.5i -O -K -X3.625i -Y2.75i -Sc2i -W1p -N >> $ps
gmt psxy -R -J -O -K -W1p << EOF >> $ps
> x-gridline  -Wdefault
-1         0
1         0
> y-gridline  -Wdefault
0         -1
0         1
> angle = 0
0         0
1         0
> angle = 120
0         0
-0.5         0.866025
> x-gmt projection -W2p
-0.3333         0
```

```
0         0
> y-gmt projection -W2p
-0.3333        0.57735
-0.3333        0
EOF

gmt pstext -R -J -O -K -Dj0.05i -F+f+a+j << EOF >> $ps
-0.16666 0 12p,Times-Roman 0 CT x
-0.3333 0.2888675 12p,Times-Roman 0 RM y
0.22 0.27 12p,Symbol -30 CB a
-0.33333 0.6 12p,Times-Roman 30 LB 120\\312
EOF

echo 0 0 0.5i 0 120 | gmt psxy -R -J -O -Sm0.15i+e -W1p -Gblack >> $ps
```



11.31 Using non-default fonts in PostScript

[sec:non-default-fonts-example]

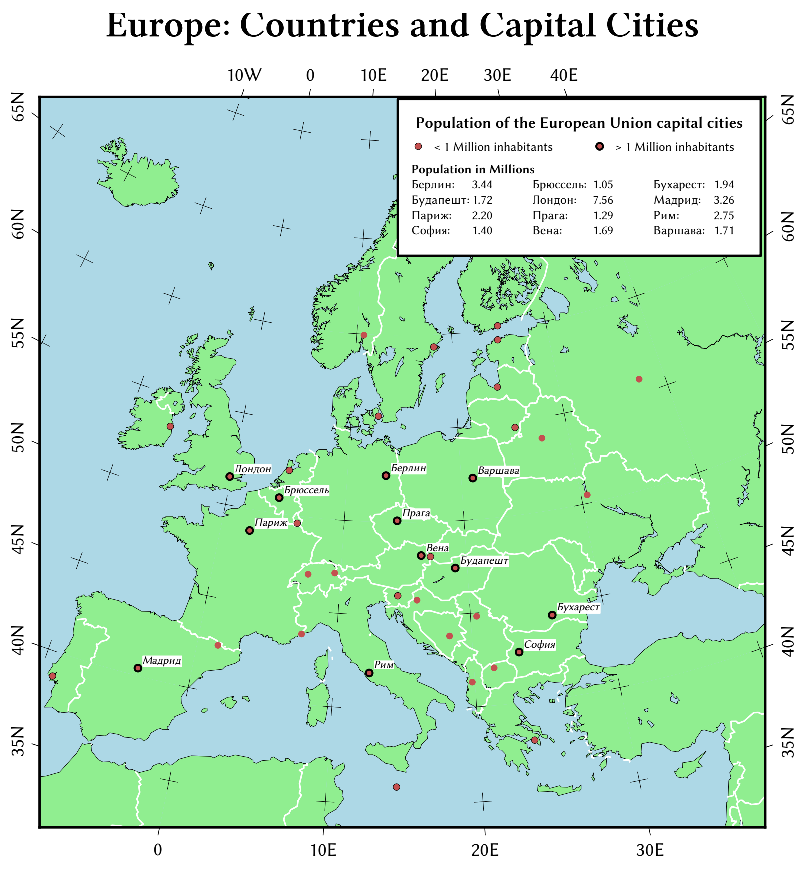
This example illustrates several possibilities to create GMTplots with non-default fonts. As these fonts are not part of the standard PostScript font collection they have to be embedded in the PS- or PDF-file with Ghostscript. See also Appendix [sec:non-default-fonts] for further information. The script includes the following steps:

- create a `CUSTOM_font_info.d` file;
- set the GMT parameters `MAP_DEGREE_SYMBOL`, `PS_CHAR_ENCODING`, and `FONT`;
- replace the default Helvetica font in the GMT-PostScript-File with sed;
- create a PostScript-File with outlined fonts (optional);
- convert GMT's PostScript output to PDF or any image format (optional).

The script produces the plot in Figure *Example 31*. All standard fonts have been substituted by the free OpenType fonts Linux Libertine (title) and Linux Biolinum (annotations). Uncomment the appropriate lines in the script to make a PostScript-file with outlined fonts or to convert to a PDF-file.

```
#!/bin/bash
#                GMT EXAMPLE 31
#
# Purpose:        Illustrate usage of non-default fonts in PostScript
# GMT progs:        gmtset, pscoast, psxy, pstext, pslegend
# Unix progs:        gs, awk, cat, rm
#
file=example_31
ps=.ps
ps_outlined=_outlined.ps
eps_outlined=_outlined.eps

# create file CUSTOM_font_info.d in current working directory
# and add PostScript font names of Linux Biolinum and Libertine
$AWK '{print $1, 0.700, 0}' << EOF > CUSTOM_font_info.d
LinBiolinumO
LinBiolinumOI
LinBiolinumOB
LinLibertineOB
EOF

# common settings
gmt gmtset FORMAT_GEO_MAP ddd:mm:ssF \
MAP_DEGREE_SYMBOL colon \
MAP_TITLE_OFFSET 20p \
MAP_GRID_CROSS_SIZE_PRIMARY 0.4c \
PS_LINE_JOIN round \
PS_CHAR_ENCODING ISO-8859-5 \
FONT LinBiolinumO \
FONT_TITLE 24p,LinLibertineOB \
MAP_ANNOT_OBLIQUE 42

# map of countries
gmt pscoast -Dl -R-7/31/64/66/r -JL15/50/40/60/16c -P \
        -Bx10g10 -By5g5 -B+t"Europe\072 Countries and Capital Cities" -A250 \
        -Slightblue -Glightgreen -W0.25p -N1/1p,white -K > $ps

# mark capitals
gmt psxy europe-capitals-ru.csv -R -J -i0,1 \
-Sc0.15c -G196/80/80 -O -K >> $ps

# small EU cities
$AWK 'BEGIN {FS=","} $4 !="" && $4 <= 1000000 {print $1, $2}' europe-capitals-ru.csv | \
gmt psxy -R -J -Sc0.15c -W0.25p -O -K >> $ps

# big EU cities
$AWK 'BEGIN {FS=","} $4 > 1000000 {print $1, $2}' europe-capitals-ru.csv | \
gmt psxy -R -J -Sc0.15c -W1.25p -O -K >> $ps

# label big EU cities
$AWK 'BEGIN {FS=","} $4 > 1000000 {print $1, $2, $3}' europe-capitals-ru.csv | \
gmt pstext -R -J -F+f7p,LinBiolinumOI+jBL -Dj0.1c -Gwhite -C5% -Qu -TO -O -K >> $ps

# construct legend
cat << EOF > legend.txt
G -0.1c
H 10 LinBiolinumOB Population of the European Union capital cities
G 0.15c
N 2
S 0.15c c 0.15c 196/80/80 0.25p 0.5c < 1 Million inhabitants
S 0.15c c 0.15c 196/80/80 1.25p 0.5c > 1 Million inhabitants
N 1
G 0.15c
L 8 LinBiolinumOB L Population in Millions
N 6
EOF

# append city names and population to legend
```

```
$AWK 'BEGIN {FS=","; f="L 8 LinBiolinumO L"}
  $4 > 1000000 {printf "%s %s:\n%s %.2f\n", f, $3, f, $4/1e6}' \
  europe-capitals-ru.csv >> legend.txt

# reduce annotation font size for legend
gmt gmtset FONT_ANNOT_PRIMARY 8p

# plot legend
gmt pslegend -R -J -Dx7.9c/12.6c/8.0c/BL \
-C0.3c/0.4c -L1.2 -F+p+gwhite -O legend.txt >> $ps

# make a PostScript and a PDF file with outlined fonts
# unfortunately gmt ps2raster won't be able to crop that file correctly anymore
# use Heiko Oberdiek's pdfcrop (http://code.google.com/p/pdfcrop2/) instead
# or crop with gmt ps2raster -A -Te before
#
# a. remove GMT logo and crop EPS:
#gmt ps2raster -P -Au -Te -C-sFONTPATH="/fonts" -Fex31CropNoLogo $ps
# b. make PS with outlined fonts:
#gs -q -sPAPERSIZE=a3 -dNOCACHE -dSAFER -dNOPAUSE -dBATCH -dNOPLATFONTS \
#  -sDEVICE=pswrite -sFONTPATH="/fonts" -sOutputFile=$ps_outlined ex31CropNoLogo.eps
# c. make croppepd EPS:
#gs -q -dNOCACHE -dSAFER -dNOPAUSE -dBATCH -dEPSCrop -sDEVICE=epswrite \
#  -sOutputFile=$eps_outlined $ps_outlined
# d. make cropped PDF:
#gmt ps2raster -P -A -Tf $ps_outlined
# uncomment to do conversation to PDF and PNG
# you will get a PDF with subsetted TrueType/PostScript fonts embedded
# which you can still edit with your favorite vector graphics editor
#export GS_FONTPATH="/fonts"
#gmt ps2raster -P -A -Tf $ps
#gmt ps2raster -P -A -Tg -E110 $ps
# clean up
rm -f gmt.history gmt.conf CUSTOM_font_info.d legend.txt ex31CropNoLogo.eps

exit 0
```



11.32 Draping an image over topography

In some cases, it is nice to “drape” an arbitrary image over a topographic map. We have already seen how to use `psimage` to plot an image anywhere in out plot. But here are aim is different, we want to manipulate an image to shade it and plot it in 3-D over topography. This example was originally created by Stephan Eickschen for a flyer emphasizing the historical economical and cultural bond between Brussels, Maastricht and Bonn. Obviously, the flag of the European Union came to mind as a good “background”.

To avoid adding large files to this example, some steps have been already done. First we get the EU flag directly from the web and convert it to a grid with values ranging from 0 to 255, where the higher values will become yellow and the lower values blue. This use of `grdreformat` requires GDAL support. `grdedit` then adds the right grid dimension.

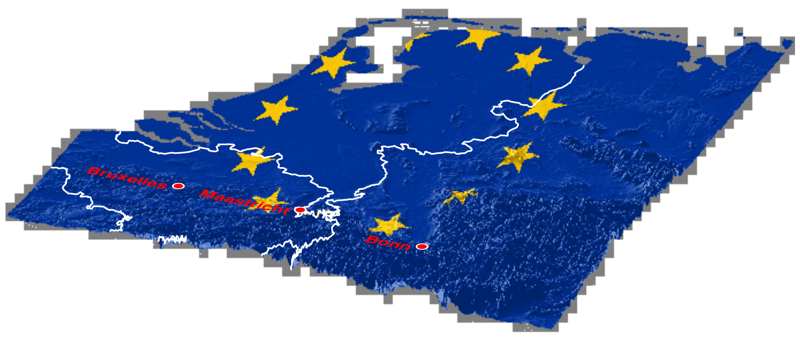
The second step is to reformat the GTOPO30 DEM file to a netCDF grid as well and then subsample it at the same pixels as the EU flag. We then illuminate the topography grid so we can use it later to emphasize the topography. The colors that we will use are those of the proper flag. Lower values will become blue and the upper values yellow.

The call the `grdview` plots a topography map of northwest continental Europe, with the flagged draped over it and with shading to show the little topography there is. `pscoast` is used in conjunction with `grdtrack` and GMTpsxyz to plot borders “at altitude”. Something similar is done at the end to plot some symbols and names for cities.

The script produces the plot in Figure *Example 32*. Note that the PNG image of the flag can be downloaded directly in the call the `grdreformat`, but we have commented that out in the example because it requires compilation with GDAL support. You will also see the `grdcut` command commented out because we did not want to store the 58 MB DEM file, whose location is mentioned in the script.

```
#!/bin/bash
#                GMT EXAMPLE 32
#
# Purpose:        Illustrate draping of an image over topography
# GMT progs:        grdcut, grdedit, grdgradient, grdreformat, grdtrack, grdview
# GMT progs:        pscoast, pstext, psxyz
# Unix progs:        cat, rm
# Credits:        Original by Stephan Eickschen
#
ps=example_32.ps

# Here we get and convert the flag of Europe directly from the web through grdreformat using
# GDAL support. We take into account the dimension of the flag (1000x667 pixels)
# for a ratio of 3x2.
# Because GDAL support will not be standard for most users, we have stored
# the result, euflag.nc in this directory.

Rflag=-R3/9/50/54
# gmt grdreformat \
#   http://upload.wikimedia.org/wikipedia/commons/thumb/b/b7/Flag_of_Europe.svg/1000px-Flag_of_Europe.svg.png
#   euflag.nc=ns
# gmt grdedit euflag.nc -fg $Rflag

# Now get the topography for the same area from GTOPO30 and store it as topo.nc.
# The DEM file comes from http://eros.usgs.gov/#/Find_Data/Products_and_Data_Available/gtopo30/w020n90
# We make an gradient grid as well, which we will use to "illuminate" the flag.

# gmt grdcut W020N90.DEM $Rflag -Gtopo.nc=ns
gmt grdgradient topo.nc -A0/270 -Gillum.nc -Ne0.6

# The color map assigns "Reflex Blue" to the lower half of the 0-255 range and
# "Yellow" to the upper half.
cat << EOF > euflag.cpt
0        0/51/153        127        0/51/153
```

```
127        255/204/0        255        255/204/0
EOF

# The next step is the plotting of the image.
# We use gmt grdview to plot the topography, euflag.nc to give the color, and illum.nc to give
# the shading.

Rplot=$Rflag/-10/790
gmt grdview topo.nc -JM13c $Rplot -Ceuflag.cpt -Geuflag.nc -Iillum.nc -Qc -JZ1c -p157.5/30 -P -K > $ps

# We now add borders. Because we have a 3-D plot, we want them to be plotted "at elevation".
# So we write out the borders, pipe them through grdtack and then plot them with psxyz.

gmt pscoast $Rflag -Df -M -N1 | gmt grdtrack -Gtopo.nc -sa | gmt psxyz $Rplot -J -JZ -p -W1p,white \
        -O -K >> $ps

# Finally, we add dots and names for three cities.
# Again, gmt grdtrack is used to put the dots "at elevation".

cat << EOF > cities.txt
05:41:27 50:51:05 Maastricht
04:21:00 50:51:00 Bruxelles
07:07:03 50:43:09 Bonn
EOF

gmt grdtrack -Gtopo.nc -sa cities.txt | gmt psxyz -i0,1,3 $Rplot -J -JZ -p -Sc7p -W1p,white -Gred \
        -K -O >> $ps
gmt pstext $Rplot -J -JZ -p -F+f12p,Helvetica-Bold,red+jRM -Dj0.1i/0.0i -O cities.txt >> $ps

# cleanup

rm -f gmt.conf euflag.cpt illum.nc cities.txt
```



11.33 Stacking automatically generated cross-profiles

The script produces the plot in Figure *Example 33*. Here we demonstrate how `grdtrack` can be used to automatically create a suite of crossing profiles of uniform spacing and length and then sample one or more grids along these profiles; we also use the median stacking option to create a stacked profile, showed above the map, with the gray area representing the variations about the stacked median profile.

```
#!/bin/bash
#               GMT EXAMPLE 33
#               $Id $
#
# Purpose:      Illustrate grdtrack's new cross-track and stacking options
# GMT progs:    makecpt, gmtconvert, grdimage, grdgradient, grdtrack, pstext, psxy
# GMT progs:    pscoast, pstext
# Unix progs:   cat, rm
```

```
#
ps=example_33.ps

# Extract a subset of ETOPO1m for the East Pacific Rise
# gmt grdcut etopo1m_grd.nc -R118W/107W/49S/42S -Gspac.nc
gmt makecpt -Crainbow -T-5000/-2000/500 -Z > z.cpt
gmt grdgradient spac.nc -A15 -Ne0.75 -Gspac_int.nc
gmt grdimage spac.nc -Ispac_int.nc -Cz.cpt -JM6i -P -Baf -K -Xc --FORMAT_GEO_MAP=dddF > $ps
# Select two points along the ridge
cat << EOF > ridge.txt
-111.6        -43.0
-113.3        -47.5
EOF
# Plot ridge segment and end points
gmt psxy -Rspac.nc -J -O -K -W2p,blue ridge.txt >> $ps
gmt psxy -R -J -O -K -Sc0.1i -Gblue ridge.txt >> $ps
# Generate cross-profiles 400 km long, spaced 10 km, samped every 2km
# and stack these using the median, write stacked profile
gmt grdtrack ridge.txt -Gspac.nc -C400k/2k/10k -Sm+sstack.txt > table.txt
gmt psxy -R -J -O -K -W0.5p table.txt >> $ps
# Show upper/lower values encountered as an envelope
gmt gmtconvert stack.txt -o0,5 > env.txt
gmt gmtconvert stack.txt -o0,6 -I -T >> env.txt
gmt psxy -R-200/200/-3500/-2000 -Bxafg1000+l"Distance from ridge (km)" -Byaf+l"Depth (m)" -BWSne \
        -JX6i/3i -O -K -Glightgray env.txt -Y6.5i >> $ps
gmt psxy -R -J -O -K -W3p stack.txt >> $ps
echo "0 -2000 MEDIAN STACKED PROFILE" | gmt pstext -R -J -O -K -Gwhite -F+jTC+f14p -Dj0.1i >> $ps
gmt psxy -R -J -O -T >> $ps
# cleanup
rm -f gmt.conf z.cpt spac_int.nc ridge.txt table.txt env.txt stack.txt
```

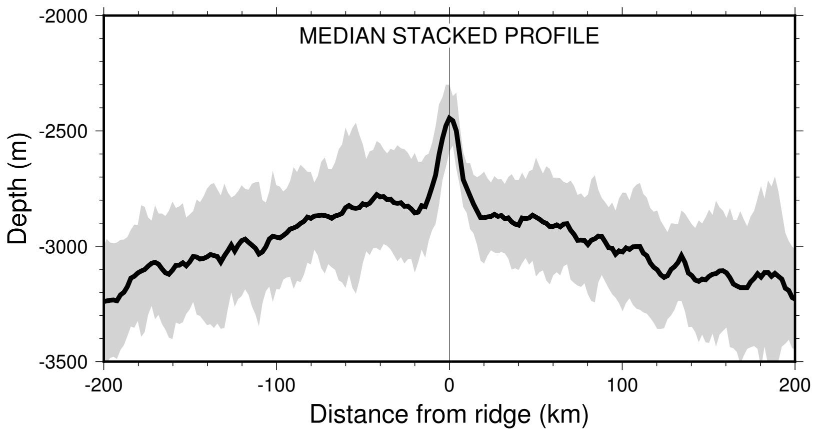
11.34 Using country polygons for plotting and shading

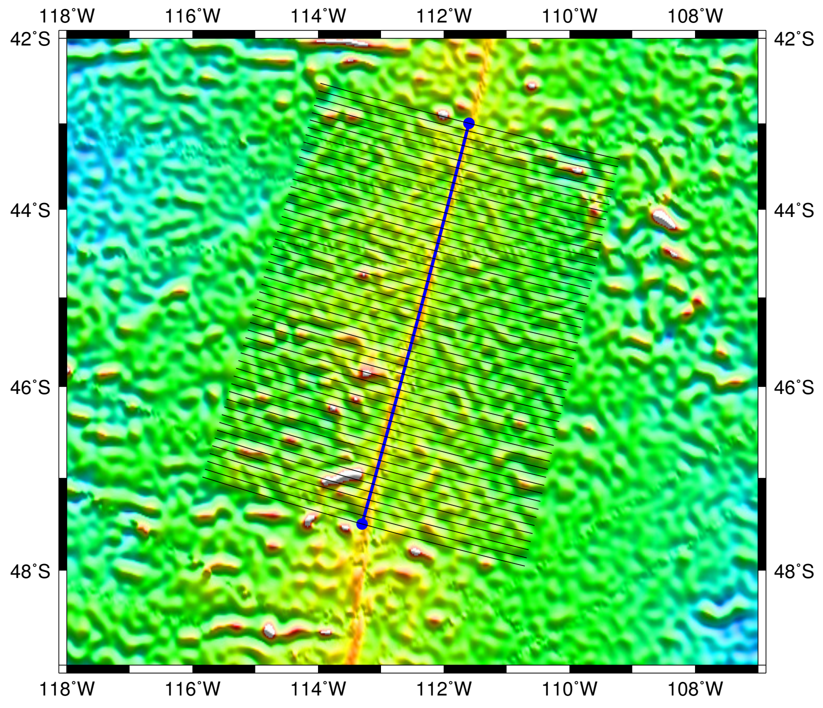
The script produces the plot in Figure *Example 34*. Here we demonstrate how `pscoast` can be used to extract and plot country polygons. We show two panels; one in which we do a basic basemap and another where we lay down a color topography image and then place a transparent layer identifying the future Franco-Italian Union whose untimely breakup in 2045 the historians will continue to debate for some time.

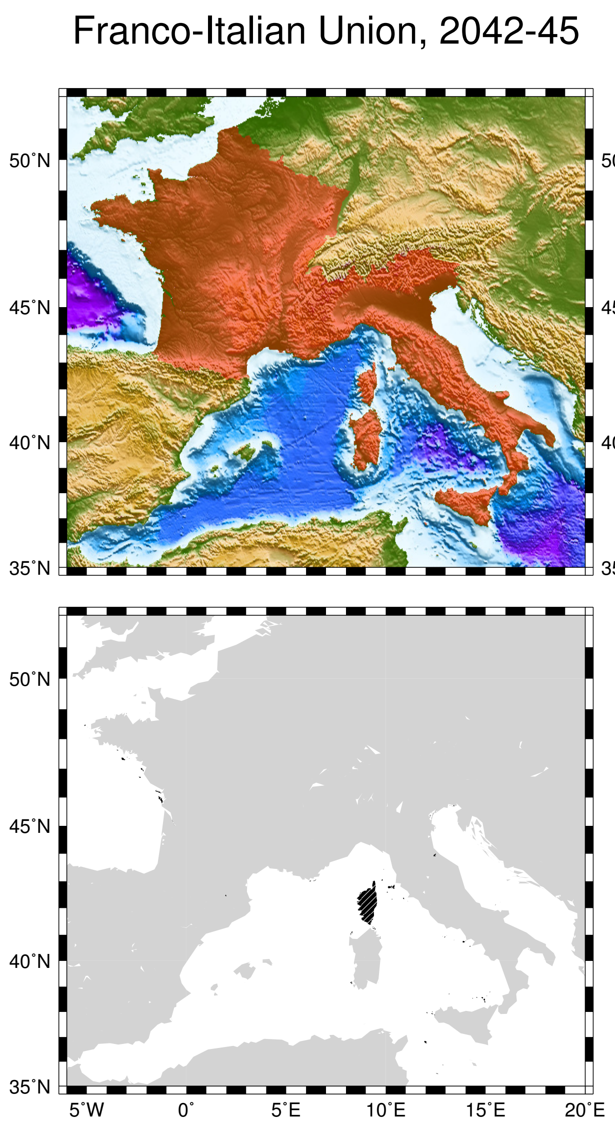
```
#!/bin/bash
#               GMT EXAMPLE 34
#
# Purpose:      Illustrate pscoast with DCW country polygons
# GMT progs:    pscoast, makecpt, grdimage, grdgradient
# Unix progs:   rm
#
ps=example_34.ps
gmt gmtset FORMAT_GEO_MAP dddF
gmt pscoast -JM4.5i -R-6/20/35/52 -FFR,IT+gP300/8 -Glightgray -Baf -BWSne -P -K -X2i > $ps
# Extract a subset of ETOPO2m for this part of Europe
# gmt grdcut etopo2m_grd.nc -R -GFR+IT.nc=ns
gmt makecpt -Cglobe -T-5000/5000/500 -Z > z.cpt
gmt grdgradient FR+IT.nc -A15 -Ne0.75 -GFR+IT_int.nc
gmt grdimage FR+IT.nc -IFR+IT_int.nc -Cz.cpt -J -O -K -Y4.5i \
        -Baf -BWsnE+t"Franco-Italian Union, 2042-45" >> $ps
gmt pscoast -J -R -FFR,IT+gred@60 -O >> $ps
# cleanup
rm -f gmt.conf FR+IT_int.nc z.cpt
```

11.35 Spherical triangulation and distance calculations

The script produces the plot in Figure *Example 35*. Here we demonstrate how `sphtriangulate` and `sphdistance` are used to compute the Delauney and Voronoi information on a sphere, using a


MEDIAN STACKED PROFILE
Depth (m)
-2000
-2500
-3000
-3500
-200
-100
0
100
200
Distance from ridge (km)

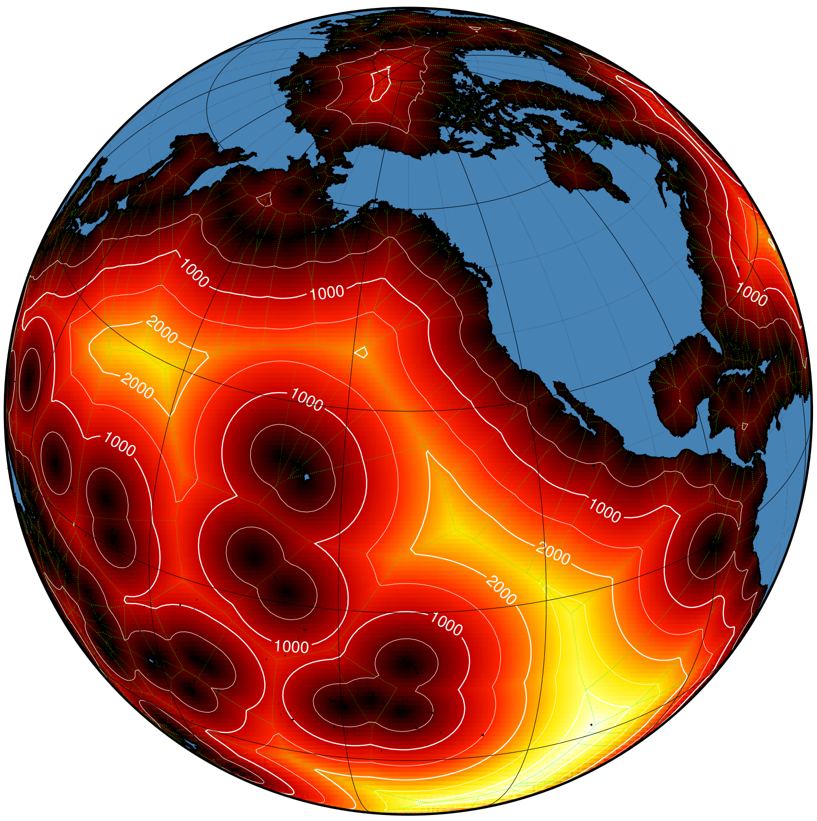

118˚W
116˚W
114˚W
112˚W
110˚W
108˚W
42˚S
44˚S
46˚S
48˚S


Franco-Italian Union, 2042-45
50˚N
45˚N
40˚N
35˚N
5˚W
0˚
5˚E
10˚E
15˚E
20˚E

decimated GSHHG crude coastline. We show a color image of the distances, highlighted with 500-km contours, and overlay the Voronoi polygons in green. Finally, the continents are placed on top.

```
#!/bin/bash
#               GMT EXAMPLE 35
#               $Id$
#
# Purpose:      Illustrate sphtriangulate and sphdistance with GSHHG crude data
# GMT progs:    pscoast, psxy, makecpt, grdimage, grdcontour, sphtriangulate, sphdistance
# Unix progs:   rm
#
ps=example_35.ps

# Get the crude GSHHS data, select GMT format, and decimate to ~20%:
# gshhs $GMTHOME/src/coast/gshhs/gshhs_c.b | $AWK '{if ($1 == ">" || NR%5 == 0) print $0}' > gshhs_c.txt
# Get Voronoi polygons
gmt sphtriangulate gshhs_c.txt -Qv -D > tt.pol
# Compute distances in km
gmt sphdistance -Rg -I1 -Qtt.pol -Gtt.nc -Lk
gmt makecpt -Chot -T0/3500/500 -Z > t.cpt
# Make a basic image plot and overlay contours, Voronoi polygons and coastlines
gmt grdimage tt.nc -JG-140/30/7i -P -K -Ct.cpt -X0.75i -Y2i > $ps
gmt grdcontour tt.nc -J -O -K -C500 -A1000+f10p,Helvetica,white -L500 -GL0/90/203/-10,175/60/170/-30,-50/30/22
gmt psxy -R -J -O -K tt.pol -W0.25p,green,. >> $ps
gmt pscoast -R -J -O -W1p -Gsteelblue -A0/1/1 -B30g30 -B+t"Distances from GSHHG crude coastlines" >> $ps
# cleanup
rm -f gmt.conf tt.pol tt.nc t.cpt
```

Distances from GSHHG crude coastlines



11.36 Spherical gridding using Renka's algorithms

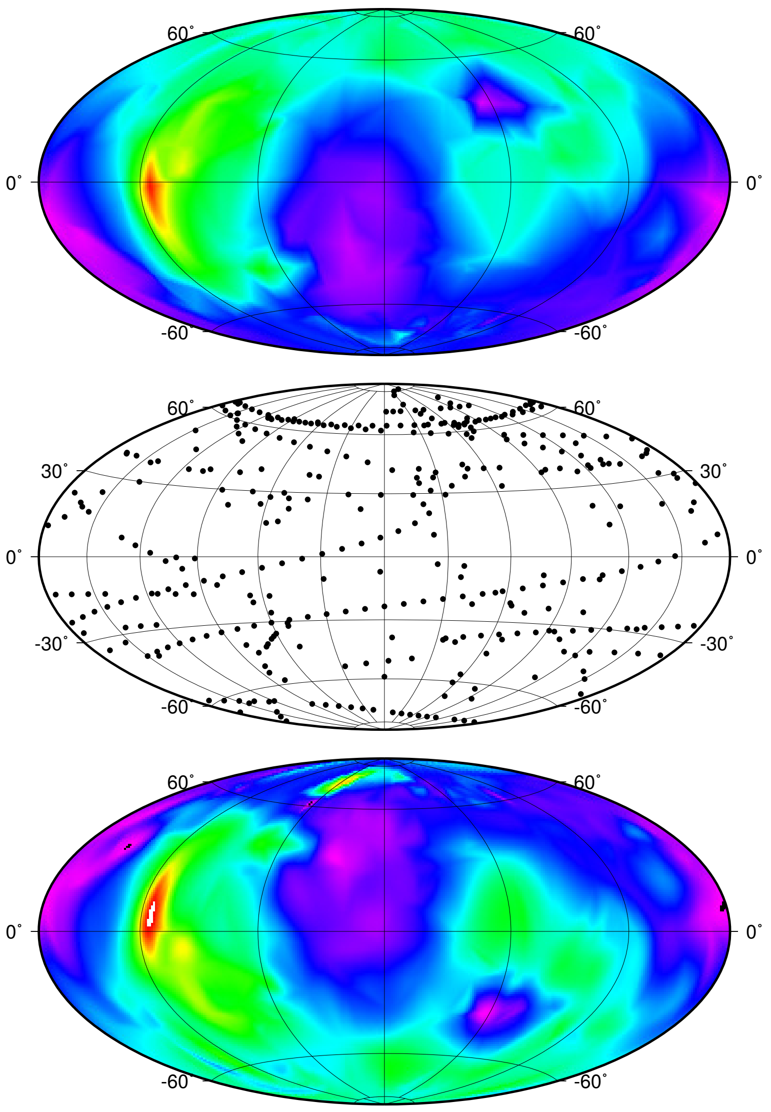
The next script produces the plot in Figure *Example 36*. Here we demonstrate how `sphinterpolate` can be used to perform spherical gridding. Our example uses early measurements of the radius of Mars from Mariner 9 and Viking Orbiter spacecrafts. The middle panels shows the data distribution while the top and bottom panel are images of the interpolation using a piecewise linear interpolation and a smoothed spline interpolation, respectively. For spherical gridding with large volumes of data we recommend `sphinterpolate` while for small data sets (such as this one, actually) you have more flexibility with `greenspline`.

```
#!/bin/bash
#               GMT EXAMPLE 36
#               $Id$
#
# Purpose:      Illustrate sphinterpolate with Mars radii data
# GMT progs:    psxy, makecpt, grdimage, sphinterpolate
# Unix progs:   rm
#
ps=example_36.ps
# Interpolate data of Mars radius from Mariner9 and Viking Orbiter spacecrafts
gmt makecpt -Crainbow -T-7000/15000/1000 -Z > tt.cpt
# Piecewise linear interpolation; no tension
gmt sphinterpolate mars370.txt -Rg -I1 -Q0 -Gtt.nc
gmt grdimage tt.nc -JH0/6i -Bag -Ctt.cpt -P -Xc -Y7.25i -K > $ps
gmt psxy -Rg -J -O -K mars370.txt -Sc0.05i -G0 -B30g30 -Y-3.25i >> $ps
# Smoothing
gmt sphinterpolate mars370.txt -Rg -I1 -Q3 -Gtt.nc
gmt grdimage tt.nc -J -Bag -Ctt.cpt  -Y-3.25i -O -K >> $ps
gmt psxy -Rg -J -O -T >> $ps
# cleanup
rm -f gmt.conf tt.cpt tt.nc
```

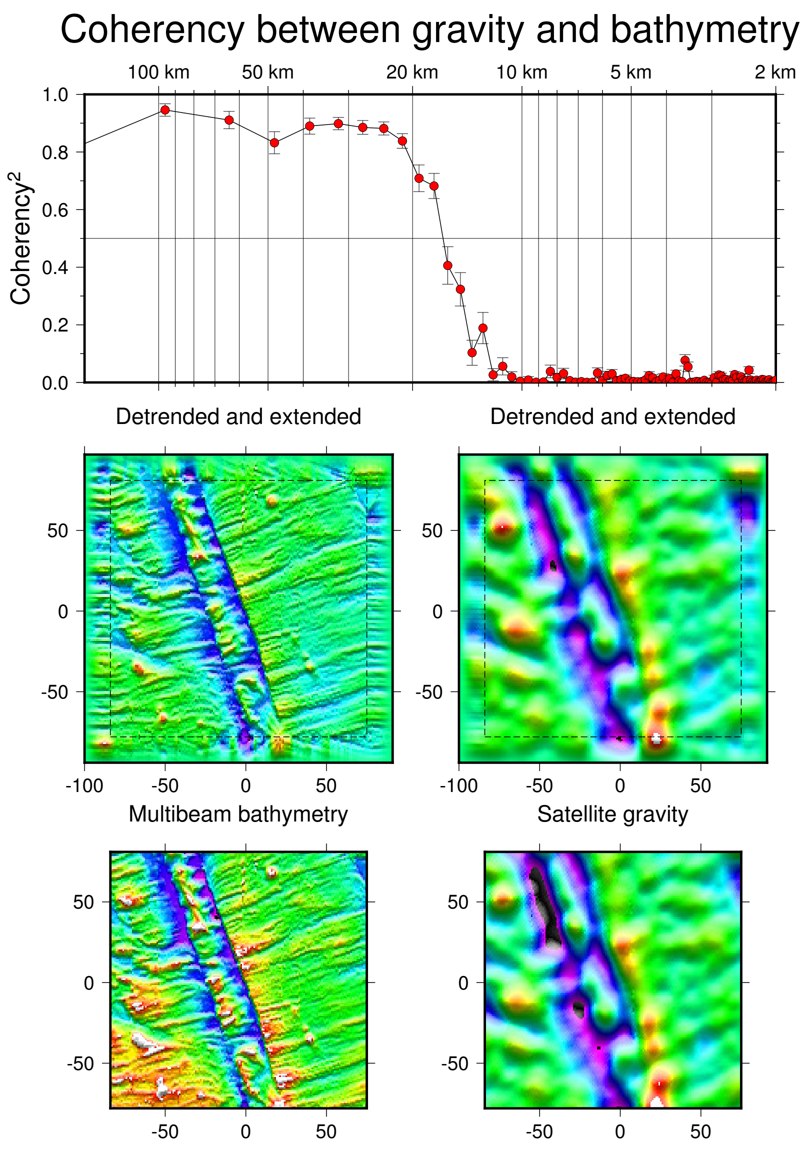
11.37 Spectral coherence between gravity and bathymetry grids

The next script produces the plot in Figure *Example 37*. We demonstrate how `grdfft` is used to compute the spectral coherence between two data sets, here multibeam bathymetry and satellite-derived gravity. The grids are detrended and tapered before the Fourier transform is computed; the intermediate plots show the grids being extended and padded to a suitable dimension.

```
#!/bin/bash
#               GMT EXAMPLE 37
#               $Id$
#
# Purpose:      Illustrate 2-D FFT and coherence between gravity and bathymetry grids
# GMT progs:    psbasemap, psxy, makecpt, grdfft, grdimage, grdinfo, grdgradient
# Unix progs:   rm
#
ps=example_37.ps

# Testing gmt grdfft coherence calculation with Karen Marks example data
# Prefix of two .nc files

G=grav.V18.par.surf.1km.sq
T=mb.par.surf.1km.sq
gmt gmtset FONT_TITLE 14p

gmt makecpt -Crainbow -T-5000/-3000/100 -Z > z.cpt
gmt makecpt -Crainbow -T-50/25/5 -Z > g.cpt
gmt grdinfo $T.nc -Ib > bbox
gmt grdgradient $G.nc -A0 -Nt1 -G_int.nc
gmt grdgradient $T.nc -A0 -Nt1 -G_int.nc
scl=1.4e-5
sclkm=1.4e-2
gmt grdimage $T.nc -I_int.nc -Jxi -Cz.cpt -P -K -X1.474i -Y1i > $ps
gmt psbasemap -R-84/75/-78/81 -Jxi -O -K -Ba -BWSne+t"Multibeam bathymetry" >> $ps
gmt grdimage $G.nc -I_int.nc -Jxi -Cg.cpt -O -K -X3.25i >> $ps
gmt psbasemap -R-84/75/-78/81 -Jxi -O -K -Ba -BWSne+t"Satellite gravity" >> $ps

gmt grdfft $T.nc $G.nc -Ewk -N192/192+d+wtmp > cross.txt
gmt grdgradient _tmp.nc -A0 -Nt1 -G_tmp_int.nc
```


60°
60°
0°
0°
-60
-60°
60°
60°
30°
30°
0°
0°
-30°
-30°
-60
-60°
60°
60°
0°
0°
-60
-60°

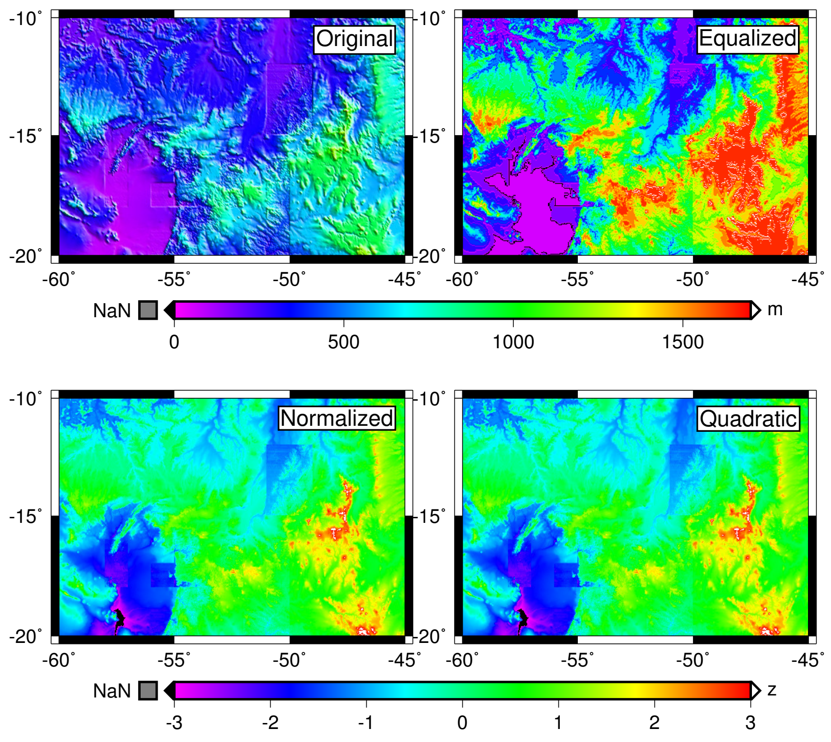
```
gmt grdgradient _tmp.nc -A0 -Nt1 -G_tmp_int.nc

gmt makecpt -Crainbow -T-1500/1500/100 -Z > z.cpt
gmt makecpt -Crainbow -T-40/40/5 -Z > g.cpt

gmt grdimage _tmp.nc -I_tmp_int.nc -Jxi -Cz.cpt -O -K -X-3.474i -Y3i >> $ps
gmt psxy -R_tmp.nc -J bbox -O -K -L -W0.5p,- >> $ps
gmt psbasemap -R-100/91/-94/97 -Jxi -O -K -Ba -BWSne+t"Detrended and extended" >> $ps

gmt grdimage _tmp.nc -I_tmp_int.nc -Jxi -Cg.cpt -O -K -X3.25i >> $ps
gmt psxy -R_tmp.nc -J bbox -O -K -L -W0.5p,- >> $ps
gmt psbasemap -R-100/91/-94/97 -Jxi -O -K -Ba -BWSne+t"Detrended and extended" >> $ps

gmt gmtset FONT_TITLE 24p
gmt psxy -R2/160/0/1 -JX-6il/2.5i -Bxa2f3g3+u" km" -Byafg0.5+l"Coherency@+2@+" -BWsNe+t"Coherency between grav
gmt psxy -R -J cross.txt -O -K -i0,15,16 -Sc0.075i -Gred -W0.25p -Ey >> $ps
gmt psxy -R -J -O -T >> $ps
rm -f cross.txt *_tmp.nc *_int.nc ?.cpt bbox
```



11.38 Histogram equalization of bathymetry grids

The next script produces the plot in Figure *Example 38*. This example shows how to use histogram equalization to enhance various ranges of a grid depending on its frequency distribution. The key tool used here is `grdhisteq`.

```
#!/bin/bash
#               GMT EXAMPLE 38
#               $Id$
#
# Purpose:      Illustrate histogram equalization on topography grids
# GMT progs:    psscale, pstext, makecpt, grdhisteq, grdimage, grdinfo, grdgradient
# Unix progs:   rm
#
ps=example_38.ps

gmt makecpt -Crainbow -T0/1700/100 -Z > t.cpt
gmt makecpt -Crainbow -T0/15/1 > c.cpt
gmt grdgradient topo.nc -Nt1 -fg -A45 -Gitopo.nc
gmt grdhisteq topo.nc -Gout.nc -C16
gmt grdimage topo.nc -Iitopo.nc -Ct.cpt -JM3i -Y5i -K -P -B5 -BWSne > $ps
echo "315 -10 Original" | gmt pstext -Rtopo.nc -J -O -K -F+jTR+f14p -T -Gwhite -W1p -Dj0.1i >> $ps
gmt grdimage out.nc -Cc.cpt -J -X3.5i -K -O -B5 -BWSne >> $ps
echo "315 -10 Equalized" | gmt pstext -R -J -O -K -F+jTR+f14p -T -Gwhite -W1p -Dj0.1i >> $ps
gmt psscale -D0i/-0.4i/5i/0.15ih -O -K -Ct.cpt -Ba500 -By+lm -E+n >> $ps
gmt grdhisteq topo.nc -Gout.nc -N
gmt makecpt -Crainbow -T-3/3/0.1 -Z > c.cpt
gmt grdimage out.nc -Cc.cpt -J -X-3.5i -Y-3.3i -K -O -B5 -BWSne >> $ps
echo "315 -10 Normalized" | gmt pstext -R -J -O -K -F+jTR+f14p -T -Gwhite -W1p -Dj0.1i >> $ps
gmt grdhisteq topo.nc -Gout.nc -N
gmt grdimage out.nc -Cc.cpt -J -X3.5i -K -O -B5 -BWSne >> $ps
echo "315 -10 Quadratic" | gmt pstext -R -J -O -K -F+jTR+f14p -T -Gwhite -W1p -Dj0.1i >> $ps
gmt psscale -D0i/-0.4i/5i/0.15ih -O -Cc.cpt -Bx1 -By+lz -E+n >> $ps
rm -f itopo.nc out.nc ?.cpt
```



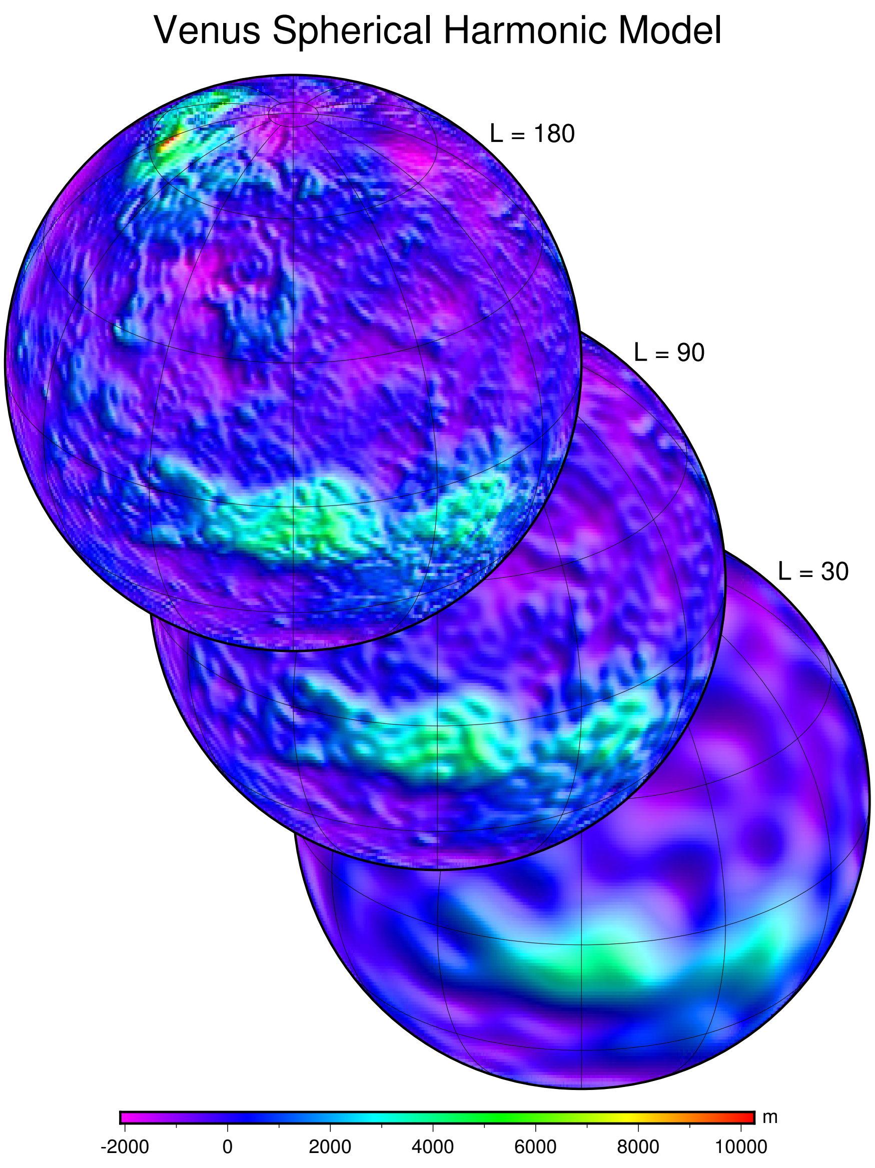
11.39 Evaluation of spherical harmonics coefficients

The next script produces the plot in Figure *Example 39*. We use a spherical harmonic model for the topography of Venus and evaluate the resulting global grid for three sets of upper order/degrees, here 30, 90, and 180; the original file (see below) goes to order and degree 720. We use the coefficients to evaluate the grids and make perspective globes of the different resolutions. The key tool used here is `sph2grd`.

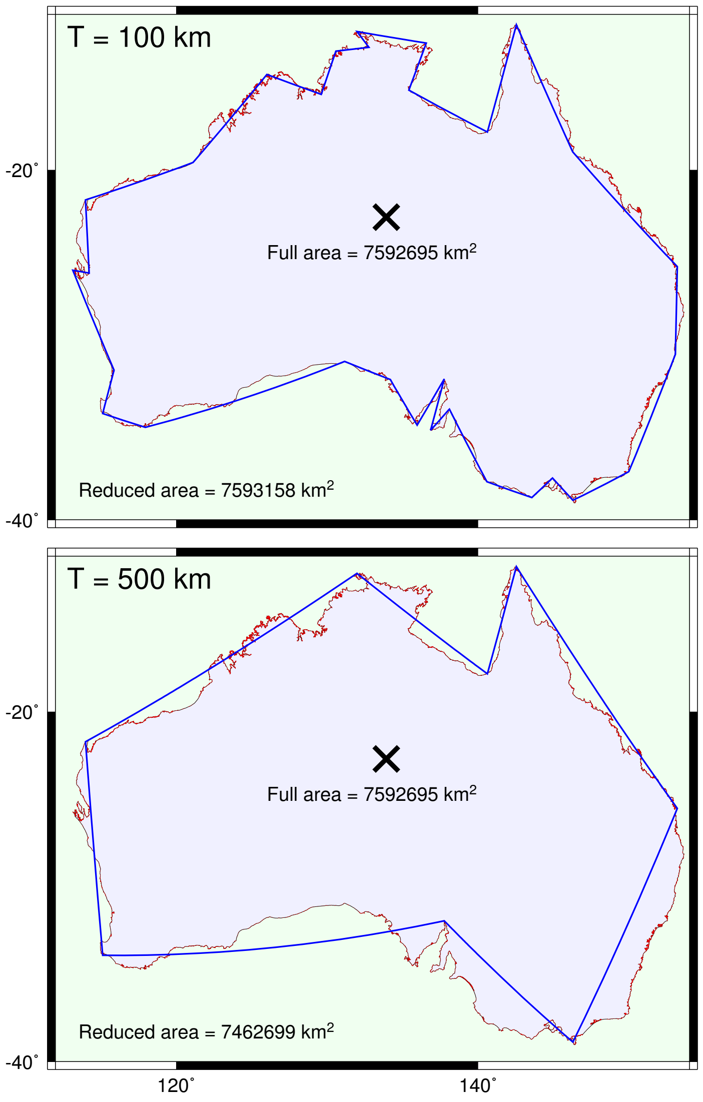
```
#!/bin/bash
#               GMT EXAMPLE 39
#               $Id$
#
# Purpose:      Illustrate evaluation of spherical harmonic coefficients
# GMT progs:    psscale, pstext, makecpt, grdimage, grdgradient, sph2grd
# Unix progs:   rm
#
ps=example_39.ps

# Evaluate the first 180, 90, and 30 order/degrees of Venus spherical
# harmonics topography model, skipping the L = 0 term (radial mean).
# File truncated from http://www.ipgp.fr/~wieczor/SH/VenusTopo180.txt.zip
# Wieczorek, M. A., Gravity and topography of the terrestrial planets,
#   Treatise on Geophysics, 10, 165-205, doi:10.1016/B978-044452748-6/00156-5, 2007

gmt sph2grd VenusTopo180.txt -I1 -Rg -Ng -Gv1.nc -F1/1/25/30
gmt sph2grd VenusTopo180.txt -I1 -Rg -Ng -Gv2.nc -F1/1/85/90
gmt sph2grd VenusTopo180.txt -I1 -Rg -Ng -Gv3.nc -F1/1/170/180
gmt grd2cpt v3.nc -Crainbow -E16 -Z > t.cpt
gmt grdgradient v1.nc -Nt0.75 -A45 -Gvint.nc
gmt grdimage v1.nc -Ivint.nc -JG90/30/5i -P -K -Bg -Ct.cpt -X3i -Y1.1i > $ps
echo 4 4.5 L = 30 | gmt pstext -R0/6/0/6 -Jx1i -O -K -Dj0.2i -F+f16p+jLM -N >> $ps
gmt psscale -Ct.cpt -O -K -D1.25i/-0.2i/5.5i/0.1ih -Bxaf -By+lm >> $ps
gmt grdgradient v2.nc -Nt0.75 -A45 -Gvint.nc
gmt grdimage v2.nc -Ivint.nc -JG -O -K -Bg -Ct.cpt -X-1.25i -Y1.9i >> $ps
echo 4 4.5 L = 90 | gmt pstext -R0/6/0/6 -Jx1i -O -K -Dj0.2i -F+f16p+jLM -N >> $ps
gmt sph2grd VenusTopo180.txt -I1 -Rg -Ng -Gv3.nc -F1/1/170/180
gmt grdgradient v3.nc -Nt0.75 -A45 -Gvint.nc
gmt grdimage v3.nc -Ivint.nc -JG -O -K -Bg -Ct.cpt -X-1.25i -Y1.9i >> $ps
echo 4 4.5 L = 180 | gmt pstext -R0/6/0/6 -Jx1i -O -K -Dj0.2i -F+f16p+jLM -N >> $ps
echo 3.75 5.4 Venus Spherical Harmonic Model | gmt pstext -R0/6/0/6 -Jx1i -O -F+f24p+jCM -N >> $ps
rm -f v*.nc t.cpt
```

11.40 line simplification and area calculations

```
#!/bin/bash
#               GMT EXAMPLE 40
#               $Id $
#
# Purpose:      Illustrate line simplification and area calculations
# GMT progs:    psbasemap, pstext, psxy, gmtsimplify, gmtspatial
# Unix progs:   awk, rm
#
ps=example_40.ps

gmt gmtspatial GSHHS_h_Australia.txt -fg -Qk > centroid.txt
gmt psbasemap -R112/154/-40/-10 -JM5.5i -P -K -B20 -BWSne+g240/255/240 -Xc > $ps
gmt psxy GSHHS_h_Australia.txt -R -J -O -Wfaint -G240/240/255 -K >> $ps
gmt psxy GSHHS_h_Australia.txt -R -J -O -Sc0.01c -Gred -K >> $ps
gmt gmtsimplify GSHHS_h_Australia.txt -T500k > T500k.txt
gmt gmtspatial GSHHS_h_Australia.txt -fg -Qk | awk '{printf "Full area = %.0f km@+2@+\n", $3}' > area.txt
gmt gmtspatial T500k.txt -fg -Qk | awk '{printf "Reduced area = %.0f km@+2@+\n", $3}' > area_T500k.txt
gmt psxy -R -J -O -K -W1p,blue T500k.txt >> $ps
gmt psxy -R -J -O -K -Sx0.3i -W3p centroid.txt >> $ps
echo 112 -10 T = 500 km | gmt pstext -R -J -O -K -Dj0.1i/0.1i -F+jTL+f18p >> $ps
gmt pstext -R -J -O -K area.txt -F+14p+cCM >> $ps
gmt pstext -R -J -O -K area_T500k.txt -F+14p+cLB -Dj0.2i >> $ps
```


Venus Spherical Harmonic Model
L = 180
L = 90
L = 30
m
-2000
0
2000
4000
6000
8000
10000

```
gmt psbasemap -R -J -O -K -B20+lightgray -BWsne+g240/255/240 -Y4.7i >> $ps
gmt psxy GSHHS_h_Australia.txt -R -J -O -Wfaint -G240/240/255 -K >> $ps
gmt psxy GSHHS_h_Australia.txt -R -J -O -Sc0.01c -Gred -K >> $ps
gmt gmtsimplify GSHHS_h_Australia.txt -T100k > T100k.txt
gmt gmtspatial T100k.txt -fg -Qk | awk '{printf "Reduced area = %.0f km@+2@+\n", $3}' > area_T100k.txt
gmt psxy -R -J -O -K -W1p,blue T100k.txt >> $ps
gmt psxy -R -J -O -K -Sx0.3i -W3p centroid.txt >> $ps
echo 112 -10 T = 100 km | gmt pstext -R -J -O -K -Dj0.1i/0.1i -F+jTL+f18p >> $ps
gmt pstext -R -J -O -K area.txt -F+14p+cCM >> $ps
gmt pstext -R -J -O -K area_T100k.txt -F+14p+cLB -Dj0.2i >> $ps
gmt psxy -R -J -O -T >> $ps
rm -f centroid.txt area*.txt T*.txt
```


T = 100 km
Full area = 7592695 km^2
Reduced area = 7593158 km^2
-20˚
-40˚
T = 500 km
Full area = 7592695 km^2
Reduced area = 7462699 km^2
-20˚
-40˚
120˚
140˚

CHAPTER 12

Creating GMT Animations

Unlike the previous chapter, in this chapter we will explore what is involved in creating animations (i.e., movies). Of course, an animation is nothing more than a series of individual images played back in an orderly fashion. Here, these images will have been created with GMT. To ensure a smooth transition from frame to frame we will be following some general guidelines when writing our scripts. Since there is no "movie" mode in GMT we must take care of all the book-keeping in our script. Thus, animations may require a bit of planning and may use more advanced scripting than the previous static examples. Note: This is a new chapter introduced with the 4.4.0 version and should be considered work in progress.

Most, if not all, animation scripts must deal with several specific phases of movie making:

1. Define parameters that determine the dimension of the final movie.
2. Pre-calculate all variables, data tables, grids, or background map layers that are *independent* of your time variable.
3. Have a frame-number loop where each frame is created as a PostScript plot, then rasterized to a TIFF file of chosen dimension.
4. Convert the individual frames to a single movie of suitable format.
5. Clean up temporary files and eventually the individual frames.

We will discuss these phases in more detail before showing our first example.

1. There are several coordinates that you need to consider when planning your movie. The first is the coordinates of your data, i.e., the *user coordinates*. As with all GMT plots you will transform those to the second set of *plot coordinates* in inches (or cm) by applying a suitable region and map projection. As before, you normally do this with a particular paper size in mind. When printed you get a high-resolution plot in monochrome or color. However, movies are not device-independent and you must finally consider the third set of *pixel coordinates* which specifies the resolution of the final movie. We control the frame size by selecting a suitable *dpi* setting that will scale your physical dimensions to the desired frame size in pixels. If you decide up front on a particular resolution (e.g., 480 by 320 pixels) then you should specify a paper size and *dpi* so that their product yields the desired pixel dimensions. For instance, here it might make sense to plan your plotting on a 4.8 by 3.2 inch "paper" and use 100 *dpi* to convert it to pixels, but you are free to use any combination that multiplies to the desired dimensions. After deciding on frame size you need to consider how many frames your movie should have. This depends on lots of things such as how patient you are, how many frames per second you need and the time range of your animation. We recommend you use variables to specify the items that go into computing the number of frames so that you can easily test your script with a few frames before changing settings and running the full Hollywood production overnight.

2. Depending on what you want to display, there are usually many elements that do not change between frames. Examples include a coastline base map for background, an overlay of text legends, perhaps some variables that hold information that will be used during the movie, and possibly subsets of larger data sets. Since movie-making can take a long time if you are ambitious, it is best to compute or plot all the elements that can be done outside your main frame-loop rather than waste time doing the same thing over and over again. You are then ready for the main loop.
3. Initialize a frame counter to 0 and have a loop that continues until your frame counter equals the desired number of frames. You must use your frame counter to create a unique file name for each frame image so that the series of images can be lexically arranged. We recommend using the GMT shell function **gmt_set_framename** to format the frame counter with an adequate number of leading zeros; see our examples for details. The bulk of your main loop involves create the single PostScript plot for this particular frame (time). This can be trivial or a serious scripting exercise depending on what you want to show. We will give a few examples with increasing complexity. Once the PostScript plot is created you need to rasterize it; we recommend you use `ps2raster` to generate a TIFF image at the agreed-upon resolution. We also recommend that you place all frame images in a sub-directory. You may increment your frame counter using **gmt_set_framenext**.
4. Once you have all your frames you are ready to combine them into an animation. There are two general approaches. (a) If your image sequence is not too long then you can convert the images into a single animated GIF file. This file can be included in PowerPoint presentations or placed on a web page and will play back as a movie by pausing the specified amount between frames, optionally repeating the entire sequence one or more times. (b) For more elaborate projects you will need to convert the frames into a proper movie format such as Quicktime, AVI, MPEG-2, MPEG-4, etc., etc. There are both free and commercial tools that can help with this conversion and they tend to be platform-specific. Most movie tools such as iMovie or MovieMaker can ingest still images and let you specify the frame duration. Under OS X we prefer to use Quicktime. [1] Free tools exist to call the Quicktime library functions from the command line as we prefer to do in our scripts. Another choice is to use the free mencoder. You will find yourself experimenting with compression settings and movie formats so that the final movie has the resolution and portability you require.
5. Finally, when all is done you should delete any temporary files created. However, since creating the frames may take a lot of time it is best to not automatically delete the frame sub directory. That way you can redo the frames-to-movie conversion with different settings until you are satisfied.

12.1 Animation of the sine function

Our first animation is not very ambitious: We wish to plot the sine function from 0-360 and take snap shots every 20. To get a smooth curve we must sample the function much more frequently; we settle on 10 times more frequently than the frame spacing. We place a bright red circle at the leading edge of the curve, and as we move forward in time (here, angles) we dim the older circles to a dark red color. We add a label that indicates the current angle value. Once the 18 frames are completed we convert them to a single animated GIF file and write a plain HTML wrapper with a simple legend. Opening the HTML page in `anim01.html` the browser will display the animation.

```
#!/bin/bash
#               GMT ANIMATION 01
#
# Purpose:      Make web page with simple animated GIF of sine function
```

[1] While Quicktime is free you must upgrade to QuickTime Pro (USD 30) to use the authoring functions.

```
# GMT progs:    gmt gmtset, gmt gmtmath, gmt psbasemap, gmt pstext, gmt psxy, gmt ps2raster
# Unix progs:   printf, mkdir, rm, mv, echo, convert, cat
# Note:         Run with any argument to build movie; otherwise 1st frame is plotted only.
#
# 1. Initialization
# 1a) Assign movie parameters
width=4i
height=2i
dpi=125
n_frames=18
name=anim_01
ps=.ps
# 1b) Do frame-independent calculations and setup
angle_step=`gmt gmtmath -Q 360  DIV =`
angle_inc=`gmt gmtmath -Q  10 DIV =`
gmt psbasemap -R0/360/-1.2/1.6 -JX3.5i/1.65i -P -K -X0.35i -Y0.25i \
        -BWSne+glightgreen -Bxa90g90f30+u\\312 -Bya0.5f0.1g1 \
        --PS_MEDIA=x --FONT_ANNOT_PRIMARY=9p > $$.map.ps
# 2. Main frame loop
mkdir -p $$
frame=0
while [  -le  ]; do
        # Create file name using a name_##.tif format
        file=`gmt_set_framename  `
        cp -f $$.map.ps $$.ps
        angle=`gmt gmtmath -Q   MUL =`
        if [  -gt 0 ]; then         # First plot has no curves
#                Plot smooth blue curve and dark red dots at all angle steps so far
                gmt gmtmath -T0// T SIND = $$.sin.d
                gmt psxy -R -J -O -K -W1p,blue $$.sin.d >> $$.ps
                gmt gmtmath -T0// T SIND = $$.sin.d
                gmt psxy -R -J -O -K -Sc0.1i -Gdarkred $$.sin.d >> $$.ps
        fi
        #        Plot red dot at current angle and annotate
        sin=`gmt gmtmath -Q  SIND =`
        gmt psxy -R -J -O -K -Sc0.1i -Gred >> $$.ps <<< " "
        printf "0 1.6 a = %03d"  | gmt pstext -R -J -F+f14p,Helvetica-Bold+jTL -O -K \
                -N -Dj0.1i/0.05i >> $$.ps
        gmt psxy -R -J -O -T >> $$.ps
        if [ $# -eq 0 ]; then
                mv $$.ps
                gmt_cleanup .gmt
                gmt_abort ": First frame plotted to .ps"
        fi
#        RIP to TIFF at specified dpi
        gmt ps2raster -E -Tt $$.ps
        mv -f $$.tif $$/.tif
        echo "Frame  completed"
        frame=`gmt_set_framenext `
done
cp $$.ps t.ps
# 3. Create animated GIF file and HTML for web page
convert -delay 20 -loop 0 $$/_*.tif .gif
cat << END > .html
<HTML>
<TITLE>GMT Trigonometry: The sine movie</TITLE>
<BODY bgcolor="#ffffff">
<CENTER>
<H1>GMT Trigonometry: The sine movie</H1>
<IMG src=".gif">
</CENTER>
<HR>
We demonstrate how the sine function <I>y = sin(a)</I> varies with <I>a</I> over
the full 360-degree interval.  We plot a bright red circle at each
new angle, letting previous circles turn dark red.  The underlying
sine curve is sampled at 10 times the frame sampling rate in order to reproduce
a smooth curve.  Our animation uses Imagemagick's convert tool to make an animated GIF file
with a 0.2 second pause between frames, set to repeat forever.
<HR>
<I>.sh: Created by  on `date`</I>
</BODY>
</HTML>
```

```
END
# 4. Clean up temporary files
gmt_cleanup .gmt
```

Make sure you understand the purpose of all the steps in our script. In this case we did some trial-and-error to determine the exact values to use for the map projection, the region, the spacing around the frame, etc. so that the final result gave a reasonable layout. Do this planning on a single PostScript plot before running a lengthy animation script.

12.2 Examining DEMs using variable illumination

Our next animation uses a gridded topography for parts of Colorado (US); the file is distributed with the tutorial examples. Here, we want to use `grdimage` to generate a shaded-relief image sequence in which we sweep the illumination azimuth around the entire horizon. The resulting animation illustrates how changing the illumination azimuth can bring out subtle features (or artifacts) in the gridded data. The red arrow points in the direction of the illumination.

```
#!/bin/bash
#               GMT ANIMATION 02
#
# Purpose:      Make web page with simple animated GIF of a DEM grid
# GMT progs:    gmt gmtset, gmt gmtmath, gmt grdgradient, gmt makecpt, gmt grdimage gmt psxy, gmt ps2raster
# Unix progs:   awk, mkdir, rm, mv, echo, convert, cat
# Note:         Run with any argument to build movie; otherwise 1st frame is plotted only.
#
# 1. Initialization
# 1a) Assign movie parameters
width=3.5i
height=4.15i
dpi=72
n_frames=36
name=anim_02
ps=.ps
# 1b) setup
del_angle=`gmt gmtmath -Q 360  DIV =`
gmt makecpt -Crainbow -T500/4500/5000 -Z > $$.cpt
# 2. Main loop
mkdir -p $$
frame=0
while [  -lt  ]; do
        # Create file name using a name_##.tif format
        file=`gmt_set_framename  `
        angle=`gmt gmtmath -Q   MUL =`
        dir=`gmt gmtmath -Q  180 ADD =`
        gmt grdgradient us.nc -A -Nt2 -fg -G$$.us_int.nc
        gmt grdimage us.nc -I$$.us_int.nc -JM3i -P -K -C$$.cpt -BWSne -B1 -X0.35i -Y0.3i \
        --PS_MEDIA=x --FONT_ANNOT_PRIMARY=9p > $$.ps
        gmt psxy -Rus.nc -J -O -K -Sc0.8i -Gwhite -Wthin >> $$.ps <<< "256.25 35.6"
        gmt psxy -Rus.nc -J -O -Sv0.1i+e -Gred -Wthick >> $$.ps <<< "256.25 35.6  0.37"
        if [ $# -eq 0 ]; then
                mv $$.ps
                gmt_cleanup .gmt
                gmt_abort ": First frame plotted to .ps"
        fi
#        RIP to TIFF at specified dpi
        gmt ps2raster -E -Tt $$.ps
        mv -f $$.tif $$/.tif
        echo "Frame  completed"
        frame=`gmt_set_framenext  `
done
# 3. Create animated GIF file and HTML for web page
convert -delay 10 -loop 0 $$/*.tif .gif
cat << END > .html
<HTML>
<TITLE>GMT shading: A tool for feature detection</TITLE>
```

```
<BODY bgcolor="#ffffff">
<CENTER>
<H1>GMT shading: A tool for feature detection</H1>
<IMG src=".gif">
</CENTER>
<HR>
We make illuminated images of topography from a section of Colorado and
vary the azimuth of the illumination (see arrow).  As the light-source sweeps around
the area over 360 degrees we notice that different features in the data
become hightlighted.  This is because the illumination is based on data
gradients and such derivatives will high-light short-wavelength signal.
Again, our animation uses Imagemagick's convert tool to make an animated GIF file
with a 0.1 second pause between the 36 frames.
<HR>
<I>.sh: Created by  on `date`</I>
</BODY>
</HTML>
END
# 4. Clean up temporary files
gmt_cleanup .gmt
```

As you can see, these sorts of animations are not terribly difficult to put together, especially since our vantage point is fixed. In the next example we will move the "camera" around and must therefore deal with how to frame perspective views.

12.3 Orbiting a static map

Our third animation keeps a fixed gridded data set but moves the camera angle around the full 360. We use `grdview` to generate a shaded-relief image sequence using the new enhanced **-E** option. No additional information is plotted on the image. As before we produce an animated GIF image and a simple HTML wrapper for it.

```
#!/bin/bash
#               GMT ANIMATION 03
#
# Purpose:      Make web page with simple animated GIF of Iceland topo
# GMT progs:    gmt gmtset, gmt gmtmath, gmt psbasemap, gmt psxy, gmt ps2raster
# Unix progs:   awk, mkdir, rm, mv, echo, convert, cat
# Note:         Run with any argument to build movie; otherwise 1st frame is plotted only.
#
# 1. Initialization
# 1a) Assign movie parameters
lon=-20
lat=65
dpi=100
x0=1.5
y0=0.75
px=4
py=2.5
el=35
az=0
name=anim_03
ps=.ps
mkdir -p $$
frame=0
gmt grdclip -Sb0/-1 -G$.nc Iceland.nc
gmt grdgradient -fg -A45 -Nt1 $.nc -G$$.nc
gmt makecpt -Crelief -Z > $$.cpt
while [  -lt 360 ]; do
        file=`gmt_set_framename  `
        if [ $# -eq 0 ]; then        # If a single frame is requested we pick this view
                az=135
        fi
        gmt grdview $.nc -R-26/-12/63/67 -JM2.5i -C$$.cpt -Qi -Bx5g10 -By5g5 -P -X0.5i -Y0.5i \
                -p/+w/+v/ --PS_MEDIA=ixi > $$.ps
        if [ $# -eq 0 ]; then
```

```
                mv $$.ps
                gmt_cleanup .gmt
                gmt_abort ": First frame plotted to .ps"
        fi
        gmt ps2raster $$.ps -Tt -E
        mv $$.tif $$/.tif
        az=`expr  + 5`
        echo "Frame  completed"
        frame=`gmt_set_framenext `
done
convert -delay 10 -loop 0 $$/*.tif .gif
cat << END > .html
<HTML>
<TITLE>GMT 3-D perspective of Iceland</TITLE>
<BODY bgcolor="#ffffff">
<CENTER>
<H1>GMT 3-D perspective of Iceland</H1>
<IMG src=".gif" border=1>
</CENTER>
<HR>
Here we show ETOPO2 topography of Iceland as we move the view
point around the island.
<I>.sh: Created by  on `date`</I>
</BODY>
</HTML>
END
# 4. Clean up temporary files
gmt_cleanup .gmt
```

12.4 Flying over topography

Our next animation simulates what an imaginary satellite might see as it passes in a great circle from New York to Miami at an altitude of 160 km. We use the general perspective view projection with `grdimage` and use `project` to create a great circle path between the two cities, sampled every 5 km. The main part of the script will make the DVD-quality frames from different view points, draw the path on the ground, and add frame numbers to each frame. As this animation generates 355 frames we can use 3rd party tools to turn the image sequence into a MPEG-4 movie [2]. Note: At the moment, `grdview` cannot use general perspective view projection to allow "fly-through" animations like Fledermaus; we expect to add this functionality in a future version.

```
#!/bin/bash
#               GMT ANIMATION 04
#
# Purpose:      Make DVD-res Quicktime movie of NY to Miami flight
# GMT progs:    gmt gmtset, gmt gmtmath, gmt psbasemap, gmt pstext, gmt psxy, gmt ps2raster
# Unix progs:   awk, mkdir, rm, mv, echo, qt_export, cat
# Note:         Run with any argument to build movie; otherwise 1st frame is plotted only.
#
# 1. Initialization
# 1a) Assign movie parameters
REGION=-Rg
altitude=160.0
tilt=55
azimuth=210
twist=0
Width=36.0
Height=34.0
px=7.2
py=4.8
dpi=100
name=anim_04
ps=.ps
```

[2] QuickTime Pro can do this, as can most video-editing programs.

```
# Set up flight path
gmt project -C-73.8333/40.75 -E-80.133/25.75 -G5 -Q > $$.path.d
frame=0
mkdir -p frames
gmt grdgradient USEast_Coast.nc -A90 -Nt1 -G$.nc
gmt makecpt -Cglobe -Z > $$.cpt
while read lon lat dist; do
        file=`gmt_set_framename  `
        ID=`echo  | $AWK '{printf "%04d\n", $1}'`
        gmt grdimage -JG////////7i+ \
                 -P -Y0.1i -X0.1i USEast_Coast.nc -I$.nc -C$$.cpt \
                --PS_MEDIA=ixi -K > $$.ps
        gmt psxy -R -J -O -K -W1p $$.path.d >> $$.ps
        gmt pstext -R0//0/ -Jx1i -F+f14p,Helvetica-Bold+jTL -O >> $$.ps <<< "0 4.6 "
        if [ $# -eq 0 ]; then
                mv $$.ps
                gmt_cleanup .gmt
                gmt_abort ": First frame plotted to .ps"
        fi
        gmt ps2raster $$.ps -Tt -E
        mv $$.tif frames/.tif
        echo "Frame  completed"
        frame=`gmt_set_framenext  `
done < $$.path.d
if [ $# -eq 0 ]; then
        echo "anim_04.sh: Made  frames at 480x720 pixels placed in subdirectory frames"
#        qt_export $$/anim_0_123456.tiff --video=h263,24,100, _movie.m4v
fi
# 4. Clean up temporary files
gmt_cleanup .gmt
```

CHAPTER 13

A. GMT Supplemental Packages

These packages are for the most part written and supported by us, but there are some exceptions. They provide extensions of GMT that are needed for particular rather than general applications. The software is provided in a separate, supplemental archive (GMT_suppl.tar.gz (or .bz2)). Questions or bug reports for this software should be addressed to the person(s) listed in the `README` file associated with the particular program. It is not guaranteed that these programs are fully ANSI-C, Y2K, or POSIX compliant, or that they necessarily will install smoothly on all platforms, but most do. Note that the data sets some of these programs work on are not distributed with these packages; they must be obtained separately. The contents of the supplemental archive may change without notice; at this writing it contains these directories:

13.1 gshhg: GSHHG data extractor

This package contains `gshhg` which you can use to extract shoreline polygons from the Global Self-consistent Hierarchical High-resolution Shorelines (GSHHG) available separately from or the (GSHHG is the polygon data base from which the GMT coastlines derive). The package is maintained by Paul Wessel.

13.2 img: gridded altimetry extractor

This package consists of the program `img2grd` to extract subsets of the global gravity and predicted topography solutions derived from satellite altimetry [1]. The package is maintained by Walter Smith and Paul Wessel.

13.3 meca: seismology and geodesy symbols

This package contains the programs `pscoupe`, `psmeca`, `pspolar`, and `psvelo` which are used by seismologists and geodesists for plotting focal mechanisms (including cross-sections and polarities), error ellipses, velocity arrows, rotational wedges, and more. The package was developed by Kurt Feigl and Genevieve Patau but is now maintained by the GMT team.

[1] For data bases, see http://topex.ucsd.edu/marine_grav/mar_grav.html.

13.4 mgd77: MGD77 extractor and plotting tools

This package currently holds the programs `mgd77convert`, `mgd77info`, `mgd77list`, `mgd77magref`, `mgd77manage`, `mgd77path`, `mgd77sniffer`, and `mgd77track` which can be used to extract information or data values from or plot marine geophysical data files in the ASCII MGD77 or netCDF MGD77+ formats [2]). This package has replaced the old **mgg** package. The package is maintained by Paul Wessel.

13.5 misc: Miscellaneous tools

At the moment, this package contains the program `dimfilter`, which is an extension of `grdfilter` in that it allows for spatial directional filtering. The package is maintained by Paul Wessel.

13.6 potential: Geopotential tools

At the moment, this package contains the programs `gravfft`, which performs gravity, isostasy, and admittance calculation for grids, `grdredpol`, which compute the continuous reduction to the pole, AKA differential RTP for magnetic data, `grdseamount`, which computes synthetic bathymetry over various seamount shapes, and `gmtgravmag3d` and `grdgravmag3d`, which computes the gravity or magnetic anomaly of a body by the method of Okabe [3]. The package is maintained by Joaquim Luis and Paul Wessel.

13.7 segyprogs: plotting SEGY seismic data

This package contains programs to plot SEGY seismic data files using the GMT mapping transformations and postscript library. `pssegy` generates a 2-D plot (x:location and y:time/depth) while `pssegyz` generates a 3-D plot (x and y: location coordinates, z: time/depth). Locations may be read from predefined or arbitrary portions of each trace header. Finally, `segy2grd` can convert SEGY data to a GMT grid file. The package is maintained by Tim Henstock [4].

13.8 spotter: backtracking and hotspotting

This package contains the plate tectonic programs `backtracker`, which you can use to move geologic markers forward or backward in time, `grdpmodeler` which evaluates predictions of a plate motion model on a grid, `grdrotater` which rotates entire grids using a finite rotation, `hotspotter` which generates CVA grids based on seamount locations and a set of absolute plate motion stage poles (`grdspotter` does the same using a bathymetry grid instead of seamount locations), `originator`, which associates seamounts with the most likely hotspot origins, and `rotconverter` which does various operations involving finite rotations on a sphere. The package is maintained by Paul Wessel.

[2] The ASCII MGD77 data are available on CD-ROM from NGDC (http://www.ngdc.noaa.gov/).

[3] Okabe, M., 1979, Analytical expressions for gravity anomalies due to polyhedral bodies and translation into magnetic anomalies, *Geophysics, 44*, 730–741.

[4] Timothy J. Henstock, University of Southampton

13.9 x2sys: track crossover error estimation

This package contains the tools `x2sys_datalist`, which allows you to extract data from almost any binary or ASCII data file, and `x2sys_cross` which determines crossover locations and errors generated by one or several geospatial tracks. Newly added are the tools `x2sys_init`, `x2sys_binlist`, `x2sys_get`, `x2sys_list`, `x2sys_put`, `x2sys_report`, `x2sys_solve` and `x2sys_merge` which extends the track-management system employed by the mgg supplement to generic track data of any format. This package represents a new generation of tools and replaces the old **x_system** package. The package is maintained by Paul Wessel.

CHAPTER 14

B. GMT File Formats

14.1 Table data

These files have *N* records which have *M* fields each. All programs that handle tables can read multicolumn files. GMT can read both ASCII, native binary, and netCDF table data.

14.1.1 ASCII tables

Optional file header records

The first data record may be preceded by one or more header records. Any records that begins with '#' is considered a header or comment line and are always processed correctly. If your data file has leading header records that do *not* start with '#' then you must make sure to use the **-h** option and set the parameter *IO_N_HEADER_RECS* in the `gmt.conf` file (GMT default is one header record if **-h** is given; you may also use **-h***nrecs* directly). Fields within a record must be separated by spaces, tabs, or commas. Each field can be an integer or floating-point number or a geographic coordinate string using the [+|-]dd[:mm[:ss]][W:|S|N|E|w|s|n|e] format. Thus, 12:30:44.5W, 17.5S, 1:00:05, and 200:45E are all valid input strings. On output, fields will be separated by the character given by the parameter *IO_COL_SEPARATOR*, which by default is a TAB.

Optional segment header records

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 segment headers interspersed with data records that signal the start of a new segment. The segment headers may be of any format, but all must have the same character in the first column. The unique character is by default '>', but you can override that by modifying the *IO_SEGMENT_MARKER* default setting. Programs can examine the segment headers to see if they contain **-D** for a distance value, **-W** and **-G** options for specifying pen and fill attributes for individual segments, **-Z** to change color via a CPT file, **-L** for label specifications, or **-T** for general-purpose text descriptions. These settings (and occasionally others) will override the corresponding command line options. GMT also provides for two special values for *IO_SEGMENT_MARKER* that can make interoperability with other software packages easier. Choose the marker **B** to have blank lines recognized as segment breaks, or use **N** to have data records whose fields equal NaN mean segment breaks (e.g., as used by Matlab or Octave). When these markers are used then no other segment header will be considered. Note that *IO_SEGMENT_MARKER* can be set differently for input and output. Finally, if a segment represents a closed polygon that is a hole inside another polygon you indicate this with **-Ph**. This setting will be read and processed if converting a file to the OGR format.

14.1.2 Binary tables

GMT programs also support native binary tables to speed up input-output for i/o-intensive tasks like gridding and preprocessing. This is discussed in more detail in section Binary table i/o: The -b option.

14.1.3 NetCDF tables

More and more programs are now producing binary data in the netCDF format, and so GMT programs started to support tabular netCDF data (files containing one or more 1-dimensional arrays) starting with GMT version 4.3.0. Because of the meta data contained in those files, reading them is much less complex than reading native binary tables, and even than ASCII tables. GMT programs will read as many 1-dimensional columns as are needed by the program, starting with the first 1-dimensional it can find in the file. To specifically specify which variables are to be read, append the suffix **?***var1***/***var2***/**... to the netCDF file name or add the option **-bic***var1***/***var2***/**..., where *var1*, *var2*, etc.are the names of the variables to be processed. The latter option is particularly practical when more than one file is read: the **-bic** option will apply to all files. Currently, GMT only reads, but does not write, netCDF tabular data.

14.2 Grid files

GMT allows numerous grid 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 raster files (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 grid files. See Section Grid file format specifications for more information.

14.2.1 NetCDF files

By default, GMT stores 2-D grids as COARDS-compliant netCDF files. COARDS (which stands for Cooperative Ocean/Atmosphere Research Data Service) is a convention used by many agencies distributing gridded data for ocean and atmosphere research. Sticking to this convention allows GMT to read gridded data provided by other institutes and other programs. Conversely, other general domain programs will be able to read grids created by GMT. COARDS is a subset of a more extensive convention for netCDF data called CF-1.5 (Climate and Forecast, version 1.5). Hence, GMT grids are also automatically CF-1.5-compliant. However, since CF-1.5 has more general application than COARDS, not all CF-1.5 compliant netCDF files can be read by GMT.

The netCDF grid file in GMT has several attributes (See Table *netcdf-format*) to describe the content. The routine that deals with netCDF grid files is sufficiently flexible so that grid files slightly deviating from the standards used by GMT can also be read.

Atributte	Description
	Global attributes
Conventions	COARDS, CF-1.5 (optional)
title	Title (optional)
source	How file was created (optional)
node_offset	0 for gridline node registration (default), 1 for pixel registration
	x- and y-variable attributes
long_name	Coordinate name (e.g., “Longitude” and “Latitude”)
units	Unit of the coordinate (e.g., “degrees_east” and “degrees_north”)
actual range (or valid range)	Minimum and maximum *x* and *y* of region; if absent the first and last *x*- and *y*-values are queried
	z-variable attributes
long_name	Name of the variable (default: “z”)
units	Unit of the variable
scale_factor	Factor to multiply *z* with (default: 1)
add_offset	Offset to add to scaled *z* (default: 0)
actual_range	Minimum and maximum *z* (in unpacked units, optional) and *z*
_FillValue (or missing_value)	Value associated with missing or invalid data points; if absent an appropriate default value is assumed, depending on data type.

By default, the first 2-dimensional variable in a netCDF file will be read as the *z* variable and the coordinate axes *x* and *y* will be determined from the dimensions of the *z* variable. GMT will recognize whether the *y* (latitude) variable increases or decreases. Both forms of data storage are handled appropriately.

For more information on the use of COARDS-compliant netCDF files, and on how to load multi-dimensional grids, read Section Modifiers for COARDS-compliant netCDF files.

14.2.2 Chunking and compression with netCDF

GMT supports reading and writing of netCDF-4 files since release 5.0. For performance reasons with ever-increasing grid sizes, the default output format of GMT is netCDF-4 with chunking enabled for grids with more than 16384 cells. Chunking means that the data is not stored sequentially in rows along latitude but rather split up into tiles. Figure *Grid split into 3 by 3 chunks* illustrates the layout in a chunked netCDF file. To access a subset of the data (e.g., the four blue tiles in the lower left), netCDF only reads those tiles (“chunks”) instead of extracting data from long rows.

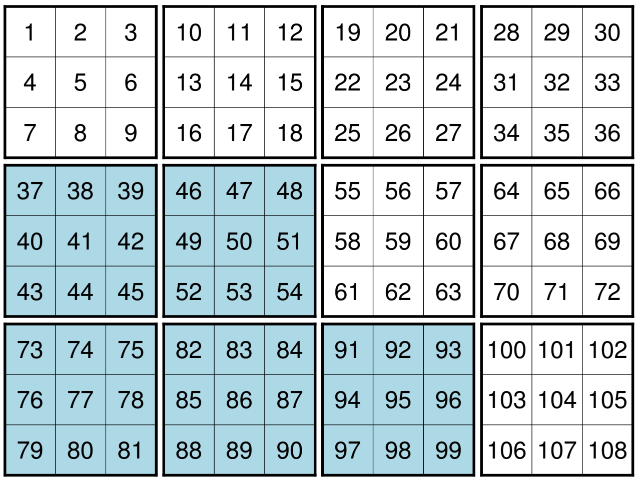


Figure 14.1: Grid split into 3 by 3 chunks

Gridded datasets in the earth sciences usually exhibit a strong spatial dependence (e.g. topography, potential fields, illustrated by blue and white cells in Figure *Grid split into 3 by 3 chunks*) and deflation can greatly reduce the file size and hence the file access time (deflating/inflating is faster than hard disk I/O). It is therefore convenient to deflate grids with spatial dependence (levels 1–3 give the best speed/size-tradeoff).

You may control the size of the chunks of data and compression with the configuration parameters *IO_NC4_CHUNK_SIZE* and *IO_NC 4_DEFLATION_LEVEL* as specified in `gmt.conf` and you can check the netCDF format with `grdinfo`.

Classic netCDF files were the de facto standard until netCDF 4.0 was released in 2008. Most programs supporting netCDF by now are using the netCDF-4 library and are thus capable of reading netCDF files generated with GMT 5, this includes official GMT releases since revision 4.5.8. In rare occasions, when you have to load netCDF files with old software, you may be forced to export your grids in the old classic format. This can be achieved by setting *IO_NC4_CHUNK_SIZE* to **c**lassic.

Further reading:

- Unidata NetCDF Workshop: NetCDF Formats and Performance
- Unidata NetCDF Workshop: What is Chunking?
- HDF NetCDF-4 Performance Report

14.2.3 Gridline and Pixel node registration

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 *registration* signals how the nodes are laid out. The grid is always defined as the intersections of all *x* ($x = x_{min}, x_{min} + x_{inc}, x_{min} + 2 \cdot x_{inc}, \ldots, x_{max}$) and *y* ($y = y_{min}, y_{min} + y_{inc}, y_{min} + 2 \cdot y_{inc}, \ldots, y_{max}$) lines. The two scenarios differ as to which area each data point represents. The default node registration in GMT is gridline node 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; `grdedit` can make changes to the grid header and convert a pixel- to a gridline-registered grid, or *vice versa*. The grid registration is determined by the common GMT **-r** option (see Section Grid registration: The -r option).

14.2.4 Boundary Conditions for operations on grids

GMT has the option to specify boundary conditions in some programs that operate on grids (e.g., `grdsample`, `grdgradient`, `grdtrack`, `nearneighbor`, and `grdview`, to name a few. The desired condition can be set with the common GMT option **-n**; see Section Grid interpolation parameters: The -n option. 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-registered 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 $(x_{max} - x_{min}) \geq 360$ 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.
2. 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.
3. 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

$$\nabla^2 f = \frac{\partial}{\partial n} \nabla^2 f = 0$$

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

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

is the two-dimensional Laplacian operator.

14.2.5 Native binary grid files

The old style native grid file format that was common in earlier version of GMT is still supported, although the use of netCDF files is strongly recommended. The file starts with a header of 892 bytes containing a number of attributes defining the content. The `grdedit` utility program will allow you to edit parts of the header of an existing grid file. The attributes listed in Table *grdheader* are contained within the header record in the order given (except the *z*-array which is not part of the header structure, but makes up the rest of the file). As this header was designed long before 64-bit architectures became available, the jump from the first three integers to the subsequent doubles in the structure does not occur on a 16-byte alignment. While GMT handles the reading of these structures correctly, enterprising programmers must take care to read this header correctly (see our code for details).

Parameter	Description
int *nx*	Number of nodes in the *x*-dimension
int *ny*	Number of nodes in the *y*-dimension
int *registration*	0 for grid line registration, 1 for pixel registration
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
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
char *title*[80]	Descriptive title of the data set
char *command*[320]	Command line that produced the grid file
char *remark*[160]	Any additional comments
TYPE *z*[nx*ny]	1-D array with *z*-values in scanline format

14.3 Sun raster files

The Sun raster file format consists of a header followed by a series of unsigned 1-byte integers that represents the bit-pattern. Bits are scanline oriented, and each row must contain an even number of bytes. 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 outline in Table *sunheader*.

Parameter	Description
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 related definitions are given in Table *sundef*.

Macro name	Description
RAS_MAGIC	0x59a66a95
RT_STANDARD	1 (Raw pixrect image in 68000 byte order)
RT_BYTE_ENCODED	2 (Run-length compression of bytes)
RT_FORMAT_RGB	3 ([X]RGB instead of [X]BGR)
RMT_NONE	0 (ras_maplength is expected to be 0)
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 raster file 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.

C. Including GMT Graphics into your Documents

Now that you made some nice graphics with GMT, it is time to add them to a document, an article, a report, your dissertation, a poster, a web page, or a presentation. Of course, you could try the old-fashioned scissors and glue stick. More likely, you want to incorporate your graphics electronically into the document. Depending on the application, the GMT PostScript file will need to be converted to Encapsulated PostScript (EPS), Portable Document Format (PDF), or some raster format (e.g., JPEG, PNG, or TIFF) in order to incorporate them into the document.

- When creating a document intended for printing (article, dissertation, or poster) it is best to preserve the scalable vector characteristics of the PostScript file. Many applications can directly incorporate PostScript in the form of EPS files. Modern programs will often allow the inclusion of PDF files. Either way, the sharpness of lines and fonts will be preserved and can be scaled up or down as required.
- When the aim is to display the graphics on a computer screen or present it using a projector, it is wise to convert the PostScript into a raster format. Although applications like PowerPoint can do this for you, you can best take the conversion into your own hands for the best results.

A large number of questions to the GMT-Help mailing list are related to these rendering issues, showing that something as seemingly straightforward as incorporating a PostScript file into a document is a far from trivial exercise. This Appendix will show how to include GMT graphics into documents and how to achieve the best quality results.

15.1 Making GMT Encapsulated PostScript Files

GMT produces freeform PostScript files. Note that a freeform PostScript file may contain special operators (such as `Setpagedevice`) that is specific to printers (e.g., selection of paper tray). Some previewers (among them, Sun's pageview) may not understand these valid instructions and may fail to image the file. Also, embedding freeform PostScript with such instructions in it into a larger document can cause printing to fail. While you could choose another viewer (we recommend **ghostview**) to view single plots prepared by GMT, it is generally wiser to convert PostScript to EPS output when you are creating a plot intended for inclusion into a larger document. Some programs (and some publishers as well) do not allow the use of instructions like `Setpagedevice` as part of embedded graphics.

An EPS file that is to be placed into another document needs to have correct bounding box parameters. These are found in the PostScript Document Comment %%BoundingBox. Applications that generate EPS files should set these parameters correctly. Because GMTmakes the PostScript files on the fly, often with several overlays, it is not possible to do so accurately. Therefore, if you need and EPS version with a "tight" BoundingBox you need to post-process your PostScript file. There are several ways in which this can be accomplished.

- Programs such as Adobe Illustrator, Aldus Freehand, and Corel Draw will allow you to edit the BoundingBox graphically.
- A command-line alternative is to use freely-available program **epstool** from the makers of Aladdin ghostscript. Running

  ```
  epstool -c -b myplot.ps
  ```

  should give a tight BoundingBox; **epstool** assumes the plot is page size and not a huge poster.
- Another option is to use **ps2epsi** which also comes with the ghostscript package. Running

  ```
  ps2epsi myplot.ps myplot.eps
  ```

  should also do the trick. The downside is that this program adds an "image" of the plot in the preamble of the EPS file, thus increasing the file size significantly. This image is a rough rendering of your PostScript graphics that some programs will show on screen while you are editing your document. This image is basically a placeholder for the PostScript graphics that will actually be printed.
- The preferred option is to use the GMT utility `ps2raster`. Its **-A** option will figure out the tightest BoundingBox, again using ghostscript in the background. For example, running

  ```
  ps2raster -A -Te myplot.ps
  ```

  will convert the PostScript file `myplot.ps` into an encapsulated PostScript file `myplot.eps` which is exactly cropped to the tightest possible BoundingBox.

If you do not want to modify your illustration but just include it in a text document: many word processors (such as Microsoft Word or Apple Pages) will let you include a PostScript file that you may place but not edit. Newer versions of those programs also allow you to include PDF versions of your graphics. Except for Pages, you will not be able to view the figure on-screen, but it will print correctly.

15.2 Converting GMT PostScript to PDF or raster images

Since Adobe's PDF (Portable Document Format) seems to have become the *de facto* standard for vector graphics, you are often well off converting GMT produced PostScript files to PDF. Being both vector formats (i.e., they basically describe all objects, text and graphics as lines and curves), such conversion sounds awfully straightforward and not worth a full section in this document. But experience has shown differently, since most converters cut corners by using the same tool (Aladdin's ghostscript) with basic default options that are not devised to produce the best quality PDF files.

For some applications it is practical or even essential that you convert your PostScript file into a raster format, such as GIF (Graphics Interchange Format), TIFF (Tagged Image File Format), PNG (Portable Network Graphics), or JPEG (Joint Photographic Experts Group). A web page is better served with a raster image that will immediately show on a web browser, than with a PostScript file that needs to be downloaded to view, despite the better printing quality of the PostScript image. A less obvious reason to convert your image to a raster format is to by-pass PowerPoint's rendering engine in case you want to embed the image into a presentation.

The are a number of programs that will convert PostScript files to PDF or raster formats, like Aladdin's **pstopdf**, pbmplus' **pstoimg**, or ImageMagick's **convert**, most of which run ghostscript behind the scenes. The same is true for viewers like **ghostview** and Apple's Preview*. So a lot of the times when people report that their PostScript plot does not look right but prints fine, it is the way ghostscript is used with its most basic settings that is to blame.

15.2.1 When converting or viewing PostScript goes awry

Here are some notorious pitfalls with ghostscript (and other rendering programs for that matter).

Rendering. When you are converting to a raster format, make sure you use a high enough resolution so that the pixels do not show when it is enlarged onto a screen or using a projector. The right choice of resolution depends on the application, but do not feel limited to the default 72 dpi (dots-per-inch) that is offered by most converters.

Image compression. There are *lossy* and *non-lossy* compressions. A compression algorithm is called "lossy" when information is lost in the conversion: there is no way back to get the full original. The effect can be seen when there are sharp color transitions in your image: the edges will get blurry in order to allow a more efficient compression. JPEG uses a lossy compression, PNG is non-lossy, and TIFF generally does not use compression at all. We therefore recommend you convert to PNG if you need to rasterize your plot, and leave JPEG to photographs.

Embedded image compression. When your GMT plot includes objects produced by `grdimage`, `psimage` or `pslegend`, they are seen as "images". The default options of ghostscript will use a *lossy* compression (similar to JPEG) on those images when converting them to PDF objects. This can be avoided, however, by inhibiting the compression altogether, or using the non-lossy *flate* compression, similar to the one used in the old **compress** program. This compression is fully reversible, so that your image does not suffer any loss.

Auto-rotation. The ghostscript engine has the annoying habit to automatically rotate an image produced with portrait orientation (using the **-P** option) so that the height is always larger than the width. So if you have an image that was printed in portrait mode but happens to have a width larger than height (for example a global map), it would suddenly get rotated. Again, this function needs to be switched off. Apple's Preview uses the ghostscript engine and suffers from the same annoying habit. Oddly enough, ghostscript does not force landscape plots to be "horizontal".

Anti-aliasing. This is not something to worry about when converting to PDF, but certainly when producing raster images (discussed below). *Anti-aliasing* in this context means that the rendering tries to avoid *aliasing*, for example, sampling only the blacks in a black-and-white hachure. It does so by first oversampling the image and then using "gray-shades" when a target pixel is only partially white or black.

Clearly, this can lead to some unwanted results. First, all edges and lines get blurry and second, the assumption of a white background causes the gray shades to stand out when transferring the image to background with a different color (like the popular sleep-inducing blue in PowerPoint presentations). A more surprising effect of anti-aliasing is that the seams between tiles that make up the land mask when using `pscoast` will become visible. The anti-aliasing somehow decides to blur the edges of all polygons, even when they are seamlessly connected to other polygons.

It is therefore wise to overrule the default anti-aliasing option and over-sample the image yourself by choosing a higher resolution.

Including fonts. When you are producing print-ready copy to publishers, they will often (and justifiably) ask that you include all fonts in your PDF document. Again, ghostscript (and all converters relying on that engine) will not do so by default.

15.2.2 Using `ps2raster`

The remedy to all the problems mentioned in the previous section is readily available to you in the form of the GMT utility `ps2raster`. It is designed to provide the best quality PDF and raster files using ghostscript as a rendering engine. The program `ps2raster` avoids anti-aliasing and lossy compression techniques that are default to ghostscript and includes the fonts into the resulting PDF file to ensure portability. By default the fonts are rendered at 720 dots-per-inch in a PDF file and images are sampled to 300 dpi, but that can be changed with the **-E** option. Simply run

```
gmt ps2raster -A -P -Tf *.ps
```

to convert all PostScript files to PDF while cropping it to the smallest possible BoundingBox. Or use the **-Tg** option to convert your files to PNG.

The **-P** option of `ps2raster` may also come in handy. When you have *not* supplied the **-P** option in your first GMT plot command, your plot will be in Landscape mode. That means that the plot will be rotated 90 degrees (anti-clockwise) to fit on a Portrait mode page when coming out of the printer. The **-P** option of `ps2raster` will undo that rotation, so that you do not have to do so within your document. This will only affect Landscape plots; Portrait plots will not be rotated.

15.3 Examples

15.3.1 GMT graphics in LaTeX

Nearly all illustrations in this GMT documentation were GMT-produced PostScript files. They were converted to PDF files using `ps2raster` and then included into a LaTeX document that was processed with **pdflatex** to create the PDF document you are reading.

To add the graphics into the LaTeX document we use the `\includegraphics` command supplied by the package. In the preamble of your LaTeX document you will need to include the line

```
\usepackage{graphicx}
```

The inclusion of the graphics will probably be inside a floating figure environment; something like this

```
\begin{figure}
   \includegraphics{myplot}
   \caption{This is my first plot in \LaTeX.}
   \label{fig:myplot}
\end{figure}
```

Note that the `\includegraphics` command does not require you to add the suffix `.pdf` to the file name. If you run **pdflatex**, it will look automatically for `myplot.pdf`. If you run **latex**, it will use `myplot.eps` instead.

You can scale your plot using the options `width=`, `height=`, or `scale=`. In addition, if your original graphics was produced in Landscape mode (i.e., you did *not* use GMT's **-P** option: not while plotting, nor in `ps2raster`), you will need to rotate the plot as well. For example,

```
\includegraphics[angle=-90,width=0.8\textwidth]{myplot}
```

will rotate the image 90 clockwise and scale it such that its width (after rotation) will be 80% of the width of the text column.

15.3.2 GMT graphics in PowerPoint

In Figure *Rendered images* we have attempted to include Figure *Example 20* into a PowerPoint presentation. First the PostScript file was converted to PDF (using `ps2raster`), then loaded into PowerPoint and the white background color was made transparent using the formatting toolbar (shown on the left

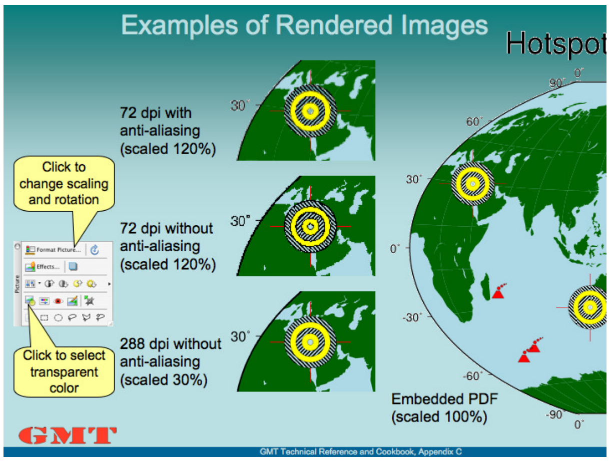
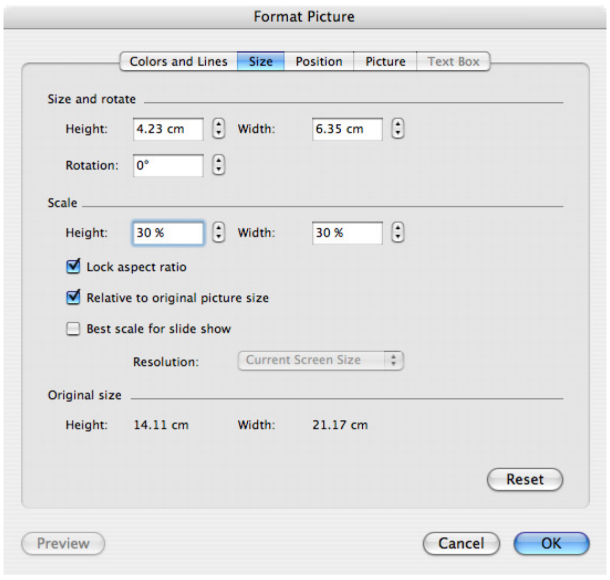

Examples of Rendered Images
Hotspot
72 dpi with anti-aliasing (scaled 120%)
Click to change scaling and rotation
72 dpi without anti-aliasing (scaled 120%)
288 dpi without anti-aliasing (scaled 30%)
Click to select transparent color
Embedded PDF (scaled 100%)
GMT
GMT Technical Reference and Cookbook, Appendix C

Figure 15.1: Examples of rendered images in a PowerPoint presentation



Size and rotate

Height: 4.23 cm Width: 6.35 cm

Rotation: 0°

Scale

Height: 30 % Width: 30 %

☑ Lock aspect ratio

☑ Relative to original picture size

☐ Best scale for slide show

Resolution: Current Screen Size

Original size

Height: 14.11 cm Width: 21.17 cm

Figure 15.2: PowerPoint’s Format Picture dialogue to set scale and rotation.

side of Figure *Rendered images*). Clearly, when we let PowerPoint do the rendering, we do not get the best result:

- The anti-aliasing causes the tiles that make up the land to stand out. This is because the anti-aliasing algorithm blurs all edges, even when the tiles join seamlessly.
- The background color was assumed to be white, hence the text is "smoothed" using gray shades. Instead, shades of blue which would be appropriate for the background we are using.

On the central column of Figure *Rendered images* we have included PNG versions of a portion of the same example. This shows the workings of anti-aliasing and different resolutions. All samples were obtained with **convert**. The one on the top uses all default settings, resulting in an anti-aliased image at 72 dpi resolution (very much like the PDF included directly into PowerPoint).

Just switching anti-aliasing off (middle) is clearly not an option either. It is true that we got rid of the gray blurring and the seams between the tiles, but without anti-aliasing the image becomes very blocky. The solution is to render the image at a higher resolution (e.g., 300 dpi) without anti-aliasing and then shrink the image to the appropriate size (bottom of the central column in Figure *Rendered images*). The scaling, rotation as well as the selection of the transparent color can be accomplished through the "Formatting" tool bar and the "Format Picture" dialogue box of PowerPoint (Figure *PowerPoint dialogue box*), which can be found by double clicking the included image (or selecting and right-clicking or control-clicking on a one-button mouse).

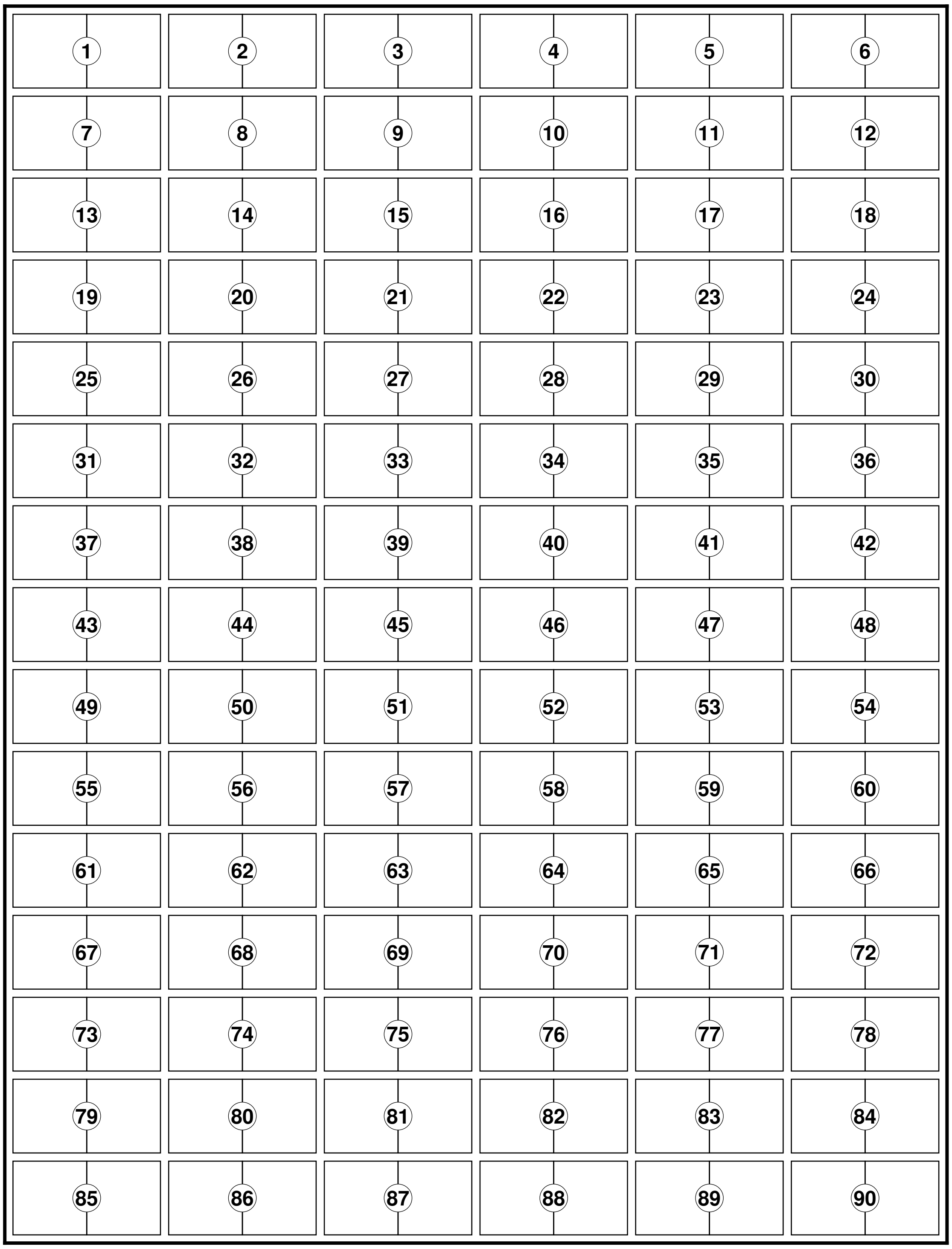
15.4 Concluding remarks

These examples do not constitute endorsements of the products mentioned above; they only represent our limited experience with adding PostScript to various types of documents. For other solutions and further help, please post messages to.

CHAPTER 16

E. Predefined Bit and Hachure Patterns in GMT

GMT provides 90 different bit and hachure patterns that can be selected with the **-Gp** or **-GP** option in most plotting programs. The left side of each image was created using **-Gp**, the right side shows the inverted version using **-GP**. These patterns are reproduced below at 300 dpi using the default black and white shades.


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90

CHAPTER 17

F. Chart of Octal Codes for Characters

The characters and their octal codes in the Standard and ISOLatin1 encoded fonts are shown in Figure *Octal codes for Standard and ISO*. Light red areas signify codes reserved for control characters. In order to use all the extended characters (shown in the light green boxes) you need to set *PS_CHAR_ENCODING* to Standard+ or ISOLatin1+ in your `gmt.conf` file [1].

The chart for the Symbol character set (GMT font number 12) and Pifont ZapfDingbats character set (font number 34) are presented in Figure *Octal codes for Symbol and ZapfDingbats* below. The octal code is obtained by appending the column value to the \?? value, e.g., ∂ is \266 in the Symbol font. The euro currency symbol is \240 in the Symbol font and will print if your printer supports it (older printer's firmware will not know about the euro).

[1] If you chose SI units during the installation then the default encoding is ISOLatin1+, otherwise it is Standard+.

Standard+

octal	0	1	2	3	4	5	6	7
\03x	🟥	¾	³	™	²	ý	ÿ	ž
\04x		!	"	#	$	%	&	’
\05x	(	)	*	+	,	-	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	–
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	ƒ
\20x	Ã	Ç	Đ	Ł	Ñ	Õ	Š	Þ
\21x	Ý	Ÿ	Ž	ã	¦	ç	©	°
\22x	÷	ð	¬	ł	−	µ	×	ñ
\23x	½	¼	¹	õ	±	®	š	þ
\24x	🟥	¡	¢	£	⁄	¥	ƒ	§
\25x	¤	'	“	«	‹	›	ﬁ	ﬂ
\26x	Á	–	†	‡	·	Â	¶	•
\27x	‚	„	”	»	…	‰	Ä	¿
\30x	À	`	´	ˆ	˜	¯	˘	˙
\31x	¨	É	˚	¸	Ê	˝	˛	ˇ
\32x	—	Ë	È	Í	Î	Ï	Ì	Ó
\33x	Ô	Ö	Ò	Ú	Û	Ü	Ù	á
\34x	â	Æ	ä	ª	à	é	ê	ë
\35x	è	Ø	Œ	º	í	î	ï	ì
\36x	ó	æ	ô	ö	ò	1	ú	û
\37x	ü	ø	œ	ß	ù	Å	å	ÿ

ISOLatin1+

octal	0	1	2	3	4	5	6	7
\03x	🟥	•	…	™	—	–	ﬁ	ž
\04x		!	"	#	$	%	&	’
\05x	(	)	*	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	@	A	B	C	D	E	F	G
\11x	H	I	J	K	L	M	N	O
\12x	P	Q	R	S	T	U	V	W
\13x	X	Y	Z	[	\	]	^	–
\14x	‘	a	b	c	d	e	f	g
\15x	h	i	j	k	l	m	n	o
\16x	p	q	r	s	t	u	v	w
\17x	x	y	z	{	|	}	~	š
\20x	Œ	†	‡	Ł	⁄	‹	Š	›
\21x	œ	Ÿ	Ž	ł	‰	„	“	”
\22x	1	`	´	ˆ	˜	¯	˘	˙
\23x	¨	‚	˚	¸	'	˝	˛	ˇ
\24x	🟥	¡	¢	£	¤	¥	¦	§
\25x	¨	©	ª	«	¬	-	®	¯
\26x	°	±	²	³	´	µ	¶	·
\27x	¸	¹	º	»	¼	½	¾	¿
\30x	À	Á	Â	Ã	Ä	Å	Æ	Ç
\31x	È	É	Ê	Ë	Ì	Í	Î	Ï
\32x	Đ	Ñ	Ò	Ó	Ô	Õ	Ö	×
\33x	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
\34x	à	á	â	ã	ä	å	æ	ç
\35x	è	é	ê	ë	ì	í	î	ï
\36x	ð	ñ	ò	ó	ô	õ	ö	÷
\37x	ø	ù	ú	û	ü	ý	þ	ÿ

Figure 17.1: Octal codes and corresponding symbols for StandardEncoding (left) and ISO-Latin1Encoding (right) fonts.

Symbol

octal	0	1	2	3	4	5	6	7
\04x		!	∀	#	∃	%	&	∋
\05x	(	)	∗	+	,	−	.	/
\06x	0	1	2	3	4	5	6	7
\07x	8	9	:	;	<	=	>	?
\10x	≅	Α	Β	Χ	Δ	Ε	Φ	Γ
\11x	Η	Ι	ϑ	Κ	Λ	Μ	Ν	Ο
\12x	Π	Θ	Ρ	Σ	Τ	Υ	ς	Ω
\13x	Ξ	Ψ	Ζ	[	∴	]	⊥	_
\14x	‾	α	β	χ	δ	ε	φ	γ
\15x	η	ι	ϕ	κ	λ	μ	ν	ο
\16x	π	θ	ρ	σ	τ	υ	ϖ	ω
\17x	ξ	ψ	ζ	{	|	}	∼	🟩
\24x	€	ϒ	′	≤	⁄	∞	ƒ	♣
\25x	♦	♥	♠	↔	←	↑	→	↓
\26x	°	±	″	≥	×	∝	∂	•
\27x	÷	≠	≡	≈	…	⏐	⎯	↵
\30x	ℵ	ℑ	ℜ	℘	⊗	⊕	∅	∩
\31x	∪	⊃	⊇	⊄	⊂	⊆	∈	∉
\32x	∠	∇	®	©	™	∏	√	⋅
\33x	¬	∧	∨	⇔	⇐	⇑	⇒	⇓
\34x	◊	〈	®	©	™	∑	⎛	⎜
\35x	⎝	⎡	⎢	⎣	⎧	⎨	⎩	⎪
\36x		〉	∫	⌠	⎮	⌡	⎞	⎟
\37x	⎠	⎤	⎥	⎦	⎫	⎬	⎭	🟩

ZapfDingbats

octal	0	1	2	3	4	5	6	7
\04x		✁	✂	✃	✄	☎	✆	✇
\05x	✈	✉	☛	☞	✌	✍	✎	✏
\06x	✐	✑	✒	✓	✔	✕	✖	✗
\07x	✘	✙	✚	✛	✜	✝	✞	✟
\10x	✠	✡	✢	✣	✤	✥	✦	✧
\11x	★	✩	✪	✫	✬	✭	✮	✯
\12x	✰	✱	✲	✳	✴	✵	✶	✷
\13x	✸	✹	✺	✻	✼	✽	✾	✿
\14x	❀	❁	❂	❃	❄	❅	❆	❇
\15x	❈	❉	❊	❋	●	❍	■	❏
\16x	❐	❑	❒	▲	▼	◆	❖	◗
\17x	❘	❙	❚	❛	❜	❝	❞	🟩
\24x	🟩	❡	❢	❣	❤	❥	❦	❧
\25x	♣	♦	♥	♠	①	②	③	④
\26x	⑤	⑥	⑦	⑧	⑨	⑩	❶	❷
\27x	❸	❹	❺	❻	❼	❽	❾	❿
\30x	➀	➁	➂	➃	➄	➅	➆	➇
\31x	➈	➉	➊	➋	➌	➍	➎	➏
\32x	➐	➑	➒	➓	➔	→	↔	↕
\33x	➘	➙	➚	➛	➜	➝	➞	➟
\34x	➠	➡	➢	➣	➤	➥	➦	➧
\35x	➨	➩	➪	➫	➬	➭	➮	➯
\36x		➱	➲	➳	➴	➵	➶	➷
\37x	➸	➹	➺	➻	➼	➽	➾	🟩

Figure 17.2: Octal codes and corresponding symbols for Symbol (left) and ZapfDingbats (right) fonts.

CHAPTER 18

G. PostScript Fonts Used by GMT

GMT uses the standard 35 fonts that come with most PostScript laserwriters. If your printer does not support some of these fonts, it will automatically substitute the default font (which is usually Courier). The following is a list of the GMT fonts:

#	Font Name	#	Font Name
0	Helvetica	**17**	**Bookman-Demi**
1	**Helvetica-Bold**	***18***	***Bookman-DemiItalic***
2	*Helvetica-Oblique*	19	Bookman-Light
3	***Helvetica-BoldOblique***	*20*	*Bookman-LightItalic*
4	Times-Roman	21	Helvetica-Narrow
5	**Times-Bold**	**22**	**Helvetica-Narrow-Bold**
6	*Times-Italic*	*23*	*Helvetica-Narrow-Oblique*
7	***Times-BoldItalic***	***24***	***Helvetica-Narrow-BoldOblique***
8	Courier	25	NewCenturySchlbk-Roman
9	**Courier-Bold**	*26*	*NewCenturySchlbk-Italic*
10	*Courier-Oblique*	**27**	**NewCenturySchlbk-Bold**
11	***Courier-BoldOblique***	***28***	***NewCenturySchlbk-BoldItalic***
12	Σψμβολ (Symbol)	29	Palatino-Roman
13	AvantGarde-Book	*30*	*Palatino-Italic*
14	*AvantGarde-BookOblique*	**31**	**Palatino-Bold**
15	**AvantGarde-Demi**	***32***	***Palatino-BoldItalic***
16	***AvantGarde-DemiOblique***	*33*	*ZapfChancery-MediumItalic*
		34	✺❁❐❆✤❉■✻✪❁▼▲ (ZapfDingbats)

Figure 18.1: The standard 35 PostScript fonts recognized by GMT.

For the special fonts Symbol (12) and ZapfDingbats (34), see the octal charts in Appendix [app:F]. When specifying fonts in GMT, you can either give the entire font name *or* just the font number listed in this table. To change the fonts used in plotting basemap frames, see the man page for `gmt.conf`. For direct plotting of text-strings, see the man page for `pstext`.

18.1 Using non-default fonts with GMT

To add additional fonts that you may have purchased or that are available freely in the internet or at your institution, see the instructions in the `CUSTOM_font_info.d` under the `share/pslib` directory and continue reading. GMT does not read or process any font files and thus does not know anything about installed fonts and their metrics. In order to use extra fonts in GMT you need to specify the PostScript name of the relevant fonts in the file `CUSTOM_font_info.d`. You can either edit the existing file distributed with GMT to make the changes global or you can create a new file in the current working directory, e.g.,

```
LinBiolinumO      0.700    0
LinLibertineOB    0.700    0
```

The format is a space delimited list of the PostScript font name, the font height-point size-ratio, and a boolean variable that tells GMT to re-encode the font (if set to zero). The latter has to be set to zero as additional fonts will most likely not come in standard PostScript encoding. GMT determines how tall typical annotations might be from the font size ratio so that the vertical position of labels and titles can be adjusted to a more uniform typesetting. Now, you can set the GMT font parameters to your non-standard fonts:

```
gmt set FONT              LinBiolinumO \
        FONT_TITLE        28p,LinLibertineOB \
        PS_CHAR_ENCODING  ISO-8859-1 \
        MAP_DEGREE_SYMBOL degree
```

After setting the encoding and the degree symbol, the configuration part for GMT is finished and you can proceed to create GMT-maps as usual. An example script is discussed in Example [sec:non-default-fonts-example].

18.1.1 Embedding fonts in PostScript and PDF

If you have Type 1 fonts in PFA (Printer Font ASCII) format you can embed them directly by copying them at the very top of your PostScript-file, before even the %!PS header comment. PFB (Printer Font Binary), TrueType or OpenType fonts cannot be embedded in PostScript directly and therefore have to be converted to PFA first.

However, you most likely will have to tell Ghostscript where to find your custom fonts in order to convert your GMT-PostScript-plot to PDF or an image with `ps2raster`. When you have used the correct PostScript-names of the fonts in `CUSTOM_font_info.d` you only need to point the `GS_FONTPATH` environment variable to the directory where the font files can be found and invoke `ps2raster` in the usual way. Likewise you can specify Ghostscript's `-sFONTPATH` option on the command line with `C -sFONTPATH=/path/to/fontdir`. Ghostscript, which is invoked by `ps2raster`, does not depend on file names. It will automatically find the relevant font files by their PostScript-names and embed and subset them in PDF-files. This is quite convenient as the document can be displayed and printed even on other computers when the font is not available locally. There is no need to convert your fonts as Ghostscript can handle all Type 1, TrueType and OpenType fonts. Note also, that you do not need to edit Ghostscript's Fontmap.

If you do not want or cannot embed the fonts you can convert them to outlines (shapes with fills) with Ghostscript in the following way:

```
gs -q -dNOCACHE -dSAFER -dNOPAUSE -dBATCH -dNOPLATFONTS \
   -sDEVICE=pswrite -sFONTPATH="/path/to/fontdir" \
   -sOutputFile=mapWithOutlinedFonts.ps map.ps
```

Note, that this only works with the *pswrite* device. If you need outlined fonts in PDF, create the PDF from the converted PostScript-file. Also, `ps2raster` cannot correctly crop Ghostscript converted PostScript-files anymore. Use Heiko Oberdiek's instead or crop with `ps2raster` **-A -Te** before (See Example [sec:non-default-fonts-example]).

18.1.2 Character encoding

Since PostScript itself does not support Unicode fonts, Ghostscript will re-encode the fonts on the fly. You have to make sure to set the correct *PS_CHAR_ENCODING* with `gmtset` and save your script file with the same character encoding. Alternatively, you can substitute all non ASCII characters with their corresponding octal codes, e.g., \265 instead of μ. Note, that PostScript fonts support only a small range of glyphs and you may have to switch the *PS_CHAR_ENCODING* within your script.

CHAPTER 19

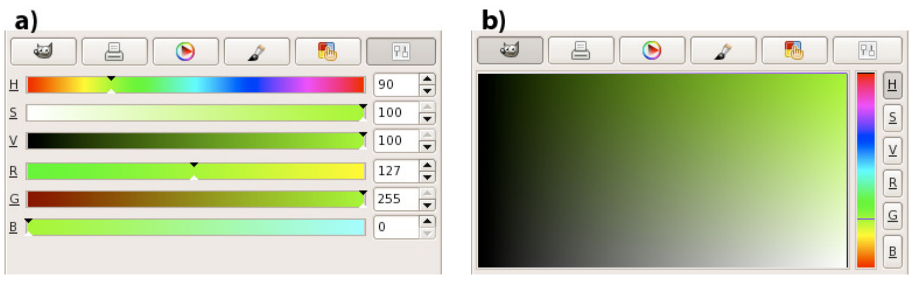
H. Color Space: The Final Frontier

In this Appendix, we are going to try to explain the relationship between the RGB, CMYK, and HSV color systems so as to (hopefully) make them more intuitive. GMT allows users to specify colors in CPT files in either of these three systems. Interpolation between colors is performed in either RGB or HSV, depending on the specification in the CPT file. Below, we will explain why this all matters.

19.1 RGB color system

Remember your (parents') first color television set? Likely it had three little bright colored squares on it: red, green, and blue. And that is exactly what each color on the tube is made of: varying levels of red, green and blue light. Switch all of them off, *r=g=b=0*, then you have black. All of them at maximum, *r=g=b=255*, creates white. Your computer screen works the same way.

A mix of levels of red, green, and blue creates basically any color imaginable. In GMT each color can be represented by the triplet *r7g7b*. For example, 127/255/0 (half red, full green, and no blue) creates a color called chartreuse. The color sliders in the graphics program GIMP are an excellent way to experiment with colors, since they show you in advance how moving one of the color sliders will change the color. As Figure *Chartreuse in GIMPa* shows: increase the red and you will get a more yellow color, while lowering the blue level will turn it into brown.

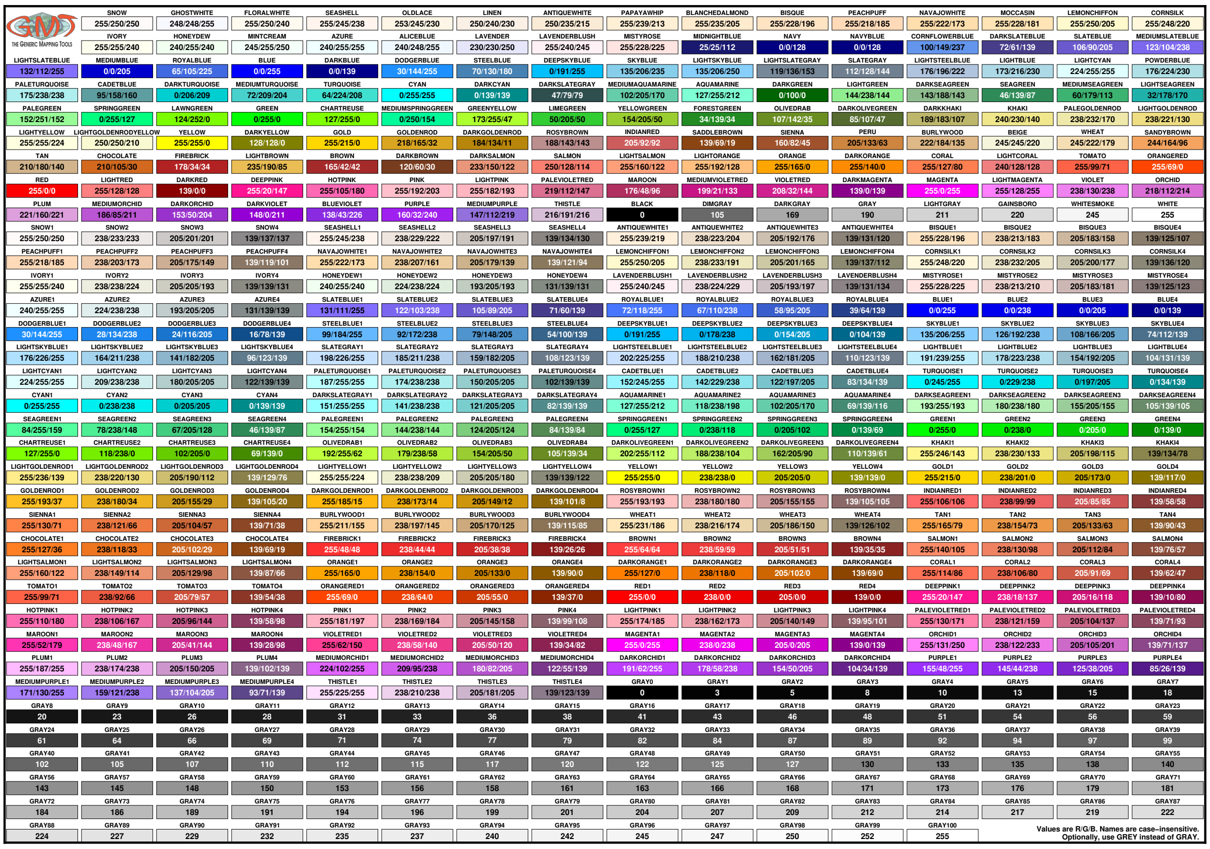


H 90
S 100
V 100
R 127
G 255
B 0

Figure 19.1: Chartreuse in GIMP. *(a)* Sliders indicate how the color is altered when changing the H, S, V, R, G, or B levels. *(b)* For a constant hue (here 90) value increases to the right and saturation increases up, so the pure color is on the top right.

Is chocolate your favorite color, but you do not know the RGB equivalent values? Then look them up in Figure *RGB chart* or type `man gmtcolors` for a full list. It's 210/105/30. But GMT makes it easy on you: you can specify pen, fill, and palette colors by any of the more than 500 unique colors found in that file.

Are you very web-savvy and work best with hexadecimal color codes as they are used in HTML? Even that is allowed in GMT. Just start with a hash mark (#) and follow with the 2 hexadecimal characters for red, green, and blue. For example, you can use `#79ff00` for chartreuse, `#D2691E` for chocolate.



	SNOW	GHOSTWHITE	FLORALWHITE	SEASHELL	OLDLACE	LINEN	ANTIQUEWHITE	PAPAYAWHIP	BLANCHEDALMOND	BISQUE	PEACHPUFF	NAVAJOWHITE	MOCCASIN	LEMONCHIFFON	CORNSILK
	255/250/250	248/248/255	255/250/240	255/245/238	253/245/230	250/240/230	250/235/215	255/239/213	255/235/205	255/228/196	255/218/185	255/222/173	255/228/181	255/250/205	255/248/220
	IVORY	HONEYDEW	MINTCREAM	AZURE	ALICEBLUE	LAVENDER	LAVENDERBLUSH	MISTYROSE	MIDNIGHTBLUE	NAVY	NAVYBLUE	CORNFLOWERBLUE	DARKSLATEBLUE	SLATEBLUE	MEDIUMSLATEBLUE
	255/255/240	240/255/240	245/255/250	240/255/255	240/248/255	230/230/250	255/240/245	255/228/225	25/25/112	0/0/128	0/0/128	100/149/237	72/61/139	106/90/205	123/104/238
LIGHTSLATEBLUE	MEDIUMBLUE	ROYALBLUE	BLUE	DARKBLUE	DODGERBLUE	STEELBLUE	DEEPSKYBLUE	SKYBLUE	LIGHTSKYBLUE	LIGHTSLATEGRAY	SLATEGRAY	LIGHTSTEELBLUE	LIGHTBLUE	LIGHTCYAN	POWDERBLUE
132/112/255	0/0/205	65/105/225	0/0/255	0/0/139	30/144/255	70/130/180	0/191/255	135/206/235	135/206/250	119/136/153	112/128/144	176/196/222	173/216/230	224/255/255	176/224/230
PALETURQUOISE	CADETBLUE	DARKTURQUOISE	MEDIUMTURQUOISE	TURQUOISE	CYAN	DARKCYAN	DARKSLATEGRAY	MEDIUMAQUAMARINE	AQUAMARINE	DARKGREEN	LIGHTGREEN	DARKSEAGREEN	SEAGREEN	MEDIUMSEAGREEN	LIGHTSEAGREEN
175/238/238	95/158/160	0/206/209	72/209/204	64/224/208	0/255/255	0/139/139	47/79/79	102/205/170	127/255/212	0/100/0	144/238/144	143/188/143	46/139/87	60/179/113	32/178/170
PALEGREEN	SPRINGGREEN	LAWNGREEN	GREEN	CHARTREUSE	MEDIUMSPRINGGREEN	GREENYELLOW	LIMEGREEN	YELLOWGREEN	FORESTGREEN	OLIVEDRAB	DARKOLIVEGREEN	DARKKHAKI	KHAKI	PALEGOLDENROD	LIGHTGOLDENROD
152/251/152	0/255/127	124/252/0	0/255/0	127/255/0	0/250/154	173/255/47	50/205/50	154/205/50	34/139/34	107/142/35	85/107/47	189/183/107	240/230/140	238/232/170	238/221/130
LIGHTYELLOW	LIGHTGOLDENRODYELLOW	YELLOW	DARKYELLOW	GOLD	GOLDENROD	DARKGOLDENROD	ROSYBROWN	INDIANRED	SADDLEBROWN	SIENNA	PERU	BURLYWOOD	BEIGE	WHEAT	SANDYBROWN
255/255/224	250/250/210	255/255/0	128/128/0	255/215/0	218/165/32	184/134/11	188/143/143	205/92/92	139/69/19	160/82/45	205/133/63	222/184/135	245/245/220	245/222/179	244/164/96
TAN	CHOCOLATE	FIREBRICK	LIGHTBROWN	BROWN	DARKBROWN	DARKSALMON	SALMON	LIGHTSALMON	LIGHTORANGE	ORANGE	DARKORANGE	CORAL	LIGHTCORAL	TOMATO	ORANGERED
210/180/140	210/105/30	178/34/34	235/190/85	165/42/42	120/60/30	233/150/122	250/128/114	255/160/122	255/192/128	255/165/0	255/140/0	255/127/80	240/128/128	255/99/71	255/69/0
RED	LIGHTRED	DARKRED	DEEPPINK	HOTPINK	PINK	LIGHTPINK	PALEVIOLETRED	MAROON	MEDIUMVIOLETRED	VIOLETRED	DARKMAGENTA	MAGENTA	LIGHTMAGENTA	VIOLET	ORCHID
255/0/0	255/128/128	139/0/0	255/20/147	255/105/180	255/192/203	255/182/193	219/112/147	176/48/96	199/21/133	208/32/144	139/0/139	255/0/255	255/128/255	238/130/238	218/112/214
PLUM	MEDIUMORCHID	DARKORCHID	DARKVIOLET	BLUEVIOLET	PURPLE	MEDIUMPURPLE	THISTLE	BLACK	DIMGRAY	DARKGRAY	GRAY	LIGHTGRAY	GAINSBORO	WHITESMOKE	WHITE
221/160/221	186/85/211	153/50/204	148/0/211	138/43/226	160/32/240	147/112/219	216/191/216	0	105	169	190	211	220	245	255
SNOW1	SNOW2	SNOW3	SNOW4	SEASHELL1	SEASHELL2	SEASHELL3	SEASHELL4	ANTIQUEWHITE1	ANTIQUEWHITE2	ANTIQUEWHITE3	ANTIQUEWHITE4	BISQUE1	BISQUE2	BISQUE3	BISQUE4
255/250/250	238/233/233	205/201/201	139/137/137	255/245/238	238/229/222	205/197/191	139/134/130	255/239/219	238/223/204	205/192/176	139/131/120	255/228/196	238/213/183	205/183/158	139/125/107
PEACHPUFF1	PEACHPUFF2	PEACHPUFF3	PEACHPUFF4	NAVAJOWHITE1	NAVAJOWHITE2	NAVAJOWHITE3	NAVAJOWHITE4	LEMONCHIFFON1	LEMONCHIFFON2	LEMONCHIFFON3	LEMONCHIFFON4	CORNSILK1	CORNSILK2	CORNSILK3	CORNSILK4
255/218/185	238/203/173	205/175/149	139/119/101	255/222/173	238/207/161	205/179/139	139/121/94	255/250/205	238/233/191	205/201/165	139/137/112	255/248/220	238/232/205	205/200/177	139/136/120
IVORY1	IVORY2	IVORY3	IVORY4	HONEYDEW1	HONEYDEW2	HONEYDEW3	HONEYDEW4	LAVENDERBLUSH1	LAVENDERBLUSH2	LAVENDERBLUSH3	LAVENDERBLUSH4	MISTYROSE1	MISTYROSE2	MISTYROSE3	MISTYROSE4
255/255/240	238/238/224	205/205/193	139/139/131	240/255/240	224/238/224	193/205/193	131/139/131	255/240/245	238/224/229	205/193/197	139/131/134	255/228/225	238/213/210	205/183/181	139/125/123
AZURE1	AZURE2	AZURE3	AZURE4	SLATEBLUE1	SLATEBLUE2	SLATEBLUE3	SLATEBLUE4	ROYALBLUE1	ROYALBLUE2	ROYALBLUE3	ROYALBLUE4	BLUE1	BLUE2	BLUE3	BLUE4
240/255/255	224/238/238	193/205/205	131/139/139	131/111/255	122/103/238	105/89/205	71/60/139	72/118/255	67/110/238	58/95/205	39/64/139	0/0/255	0/0/238	0/0/205	0/0/139
DODGERBLUE1	DODGERBLUE2	DODGERBLUE3	DODGERBLUE4	STEELBLUE1	STEELBLUE2	STEELBLUE3	STEELBLUE4	DEEPSKYBLUE1	DEEPSKYBLUE2	DEEPSKYBLUE3	DEEPSKYBLUE4	SKYBLUE1	SKYBLUE2	SKYBLUE3	SKYBLUE4
30/144/255	28/134/238	24/116/205	16/78/139	99/184/255	92/172/238	79/148/205	54/100/139	0/191/255	0/178/238	0/154/205	0/104/139	135/206/255	126/192/238	108/166/205	74/112/139
LIGHTSKYBLUE1	LIGHTSKYBLUE2	LIGHTSKYBLUE3	LIGHTSKYBLUE4	SLATEGRAY1	SLATEGRAY2	SLATEGRAY3	SLATEGRAY4	LIGHTSTEELBLUE1	LIGHTSTEELBLUE2	LIGHTSTEELBLUE3	LIGHTSTEELBLUE4	LIGHTBLUE1	LIGHTBLUE2	LIGHTBLUE3	LIGHTBLUE4
176/226/255	164/211/238	141/182/205	96/123/139	198/226/255	185/211/238	159/182/205	108/123/139	202/225/255	188/210/238	162/181/205	110/123/139	191/239/255	178/223/238	154/192/205	104/131/139
LIGHTCYAN1	LIGHTCYAN2	LIGHTCYAN3	LIGHTCYAN4	PALETURQUOISE1	PALETURQUOISE2	PALETURQUOISE3	PALETURQUOISE4	CADETBLUE1	CADETBLUE2	CADETBLUE3	CADETBLUE4	TURQUOISE1	TURQUOISE2	TURQUOISE3	TURQUOISE4
224/255/255	209/238/238	180/205/205	122/139/139	187/255/255	174/238/238	150/205/205	102/139/139	152/245/255	142/229/238	122/197/205	83/134/139	0/245/255	0/229/238	0/197/205	0/134/139
CYAN1	CYAN2	CYAN3	CYAN4	DARKSLATEGRAY1	DARKSLATEGRAY2	DARKSLATEGRAY3	DARKSLATEGRAY4	AQUAMARINE1	AQUAMARINE2	AQUAMARINE3	AQUAMARINE4	DARKSEAGREEN1	DARKSEAGREEN2	DARKSEAGREEN3	DARKSEAGREEN4
0/255/255	0/238/238	0/205/205	0/139/139	151/255/255	141/238/238	121/205/205	82/139/139	127/255/212	118/238/198	102/205/170	69/139/116	193/255/193	180/238/180	155/205/155	105/139/105
SEAGREEN1	SEAGREEN2	SEAGREEN3	SEAGREEN4	PALEGREEN1	PALEGREEN2	PALEGREEN3	PALEGREEN4	SPRINGGREEN1	SPRINGGREEN2	SPRINGGREEN3	SPRINGGREEN4	GREEN1	GREEN2	GREEN3	GREEN4
84/255/159	78/238/148	67/205/128	46/139/87	154/255/154	144/238/144	124/205/124	84/139/84	0/255/127	0/238/118	0/205/102	0/139/69	0/255/0	0/238/0	0/205/0	0/139/0
CHARTREUSE1	CHARTREUSE2	CHARTREUSE3	CHARTREUSE4	OLIVEDRAB1	OLIVEDRAB2	OLIVEDRAB3	OLIVEDRAB4	DARKOLIVEGREEN1	DARKOLIVEGREEN2	DARKOLIVEGREEN3	DARKOLIVEGREEN4	KHAKI1	KHAKI2	KHAKI3	KHAKI4
127/255/0	118/238/0	102/205/0	69/139/0	192/255/62	179/238/58	154/205/50	105/139/34	202/255/112	188/238/104	162/205/90	110/139/61	255/246/143	238/230/133	205/198/115	139/134/78
LIGHTGOLDENROD1	LIGHTGOLDENROD2	LIGHTGOLDENROD3	LIGHTGOLDENROD4	LIGHTYELLOW1	LIGHTYELLOW2	LIGHTYELLOW3	LIGHTYELLOW4	YELLOW1	YELLOW2	YELLOW3	YELLOW4	GOLD1	GOLD2	GOLD3	GOLD4
255/236/139	238/220/130	205/190/112	139/129/76	255/255/224	238/238/209	205/205/180	139/139/122	255/255/0	238/238/0	205/205/0	139/139/0	255/215/0	238/201/0	205/173/0	139/117/0
GOLDENROD1	GOLDENROD2	GOLDENROD3	GOLDENROD4	DARKGOLDENROD1	DARKGOLDENROD2	DARKGOLDENROD3	DARKGOLDENROD4	ROSYBROWN1	ROSYBROWN2	ROSYBROWN3	ROSYBROWN4	INDIANRED1	INDIANRED2	INDIANRED3	INDIANRED4
255/193/37	238/180/34	205/155/29	139/105/20	255/185/15	238/173/14	205/149/12	139/101/8	255/193/193	238/180/180	205/155/155	139/105/105	255/106/106	238/99/99	205/85/85	139/58/58
SIENNA1	SIENNA2	SIENNA3	SIENNA4	BURLYWOOD1	BURLYWOOD2	BURLYWOOD3	BURLYWOOD4	WHEAT1	WHEAT2	WHEAT3	WHEAT4	TAN1	TAN2	TAN3	TAN4
255/130/71	238/121/66	205/104/57	139/71/38	255/211/155	238/197/145	205/170/125	139/115/85	255/231/186	238/216/174	205/186/150	139/126/102	255/165/79	238/154/73	205/133/63	139/90/43
CHOCOLATE1	CHOCOLATE2	CHOCOLATE3	CHOCOLATE4	FIREBRICK1	FIREBRICK2	FIREBRICK3	FIREBRICK4	BROWN1	BROWN2	BROWN3	BROWN4	SALMON1	SALMON2	SALMON3	SALMON4
255/127/36	238/118/33	205/102/29	139/69/19	255/48/48	238/44/44	205/38/38	139/26/26	255/64/64	238/59/59	205/51/51	139/35/35	255/140/105	238/130/98	205/112/84	139/76/57
LIGHTSALMON1	LIGHTSALMON2	LIGHTSALMON3	LIGHTSALMON4	ORANGE1	ORANGE2	ORANGE3	ORANGE4	DARKORANGE1	DARKORANGE2	DARKORANGE3	DARKORANGE4	CORAL1	CORAL2	CORAL3	CORAL4
255/160/122	238/149/114	205/129/98	139/87/66	255/165/0	238/154/0	205/133/0	139/90/0	255/127/0	238/118/0	205/102/0	139/69/0	255/114/86	238/106/80	205/91/69	139/62/47
TOMATO1	TOMATO2	TOMATO3	TOMATO4	ORANGERED1	ORANGERED2	ORANGERED3	ORANGERED4	RED1	RED2	RED3	RED4	DEEPPINK1	DEEPPINK2	DEEPPINK3	DEEPPINK4
255/99/71	238/92/66	205/79/57	139/54/38	255/69/0	238/64/0	205/55/0	139/37/0	255/0/0	238/0/0	205/0/0	139/0/0	255/20/147	238/18/137	205/16/118	139/10/80
HOTPINK1	HOTPINK2	HOTPINK3	HOTPINK4	PINK1	PINK2	PINK3	PINK4	LIGHTPINK1	LIGHTPINK2	LIGHTPINK3	LIGHTPINK4	PALEVIOLETRED1	PALEVIOLETRED2	PALEVIOLETRED3	PALEVIOLETRED4
255/110/180	238/106/167	205/96/144	139/58/98	255/181/197	238/169/184	205/145/158	139/99/108	255/174/185	238/162/173	205/140/149	139/95/101	255/130/171	238/121/159	205/104/137	139/71/93
MAROON1	MAROON2	MAROON3	MAROON4	VIOLETRED1	VIOLETRED2	VIOLETRED3	VIOLETRED4	MAGENTA1	MAGENTA2	MAGENTA3	MAGENTA4	ORCHID1	ORCHID2	ORCHID3	ORCHID4
255/52/179	238/48/167	205/41/144	139/28/98	255/62/150	238/58/140	205/50/120	139/34/82	255/0/255	238/0/238	205/0/205	139/0/139	255/131/250	238/122/233	205/105/201	139/71/137
PLUM1	PLUM2	PLUM3	PLUM4	MEDIUMORCHID1	MEDIUMORCHID2	MEDIUMORCHID3	MEDIUMORCHID4	DARKORCHID1	DARKORCHID2	DARKORCHID3	DARKORCHID4	PURPLE1	PURPLE2	PURPLE3	PURPLE4
255/187/255	238/174/238	205/150/205	139/102/139	224/102/255	209/95/238	180/82/205	122/55/139	191/62/255	178/58/238	154/50/205	104/34/139	155/48/255	145/44/238	125/38/205	85/26/139
MEDIUMPURPLE1	MEDIUMPURPLE2	MEDIUMPURPLE3	MEDIUMPURPLE4	THISTLE1	THISTLE2	THISTLE3	THISTLE4	GRAY0	GRAY1	GRAY2	GRAY3	GRAY4	GRAY5	GRAY6	GRAY7
171/130/255	159/121/238	137/104/205	93/71/139	255/225/255	238/210/238	205/181/205	139/123/139	0	3	5	8	10	13	15	18
GRAY8	GRAY9	GRAY10	GRAY11	GRAY12	GRAY13	GRAY14	GRAY15	GRAY16	GRAY17	GRAY18	GRAY19	GRAY20	GRAY21	GRAY22	GRAY23
20	23	26	28	31	33	36	38	41	43	46	48	51	54	56	59
GRAY24	GRAY25	GRAY26	GRAY27	GRAY28	GRAY29	GRAY30	GRAY31	GRAY32	GRAY33	GRAY34	GRAY35	GRAY36	GRAY37	GRAY38	GRAY39
61	64	66	69	71	74	77	79	82	84	87	89	92	94	97	99
GRAY40	GRAY41	GRAY42	GRAY43	GRAY44	GRAY45	GRAY46	GRAY47	GRAY48	GRAY49	GRAY50	GRAY51	GRAY52	GRAY53	GRAY54	GRAY55
102	105	107	110	112	115	117	120	122	125	127	130	133	135	138	140
GRAY56	GRAY57	GRAY58	GRAY59	GRAY60	GRAY61	GRAY62	GRAY63	GRAY64	GRAY65	GRAY66	GRAY67	GRAY68	GRAY69	GRAY70	GRAY71
143	145	148	150	153	156	158	161	163	166	168	171	173	176	179	181
GRAY72	GRAY73	GRAY74	GRAY75	GRAY76	GRAY77	GRAY78	GRAY79	GRAY80	GRAY81	GRAY82	GRAY83	GRAY84	GRAY85	GRAY86	GRAY87
184	186	189	191	194	196	199	201	204	207	209	212	214	217	219	222
GRAY88	GRAY89	GRAY90	GRAY91	GRAY92	GRAY93	GRAY94	GRAY95	GRAY96	GRAY97	GRAY98	GRAY99	GRAY100			
224	227	229	232	235	237	240	242	245	247	250	252	255			

Values are R/G/B. Names are case-insensitive.
Optionally, use GREY instead of GRAY.

Figure 19.2: The 555 unique color names that can be used in GMT. Lower, upper, or mixed cases, as well as the british spelling of grey are allowed. A4, Letter, and Tabloid sized versions of this RGB chart can be found in the GMT documentation directory.

19.2 HSV color system

If you have played around with RGB color sliders, you will have noticed that it is not intuitive to make a chosen color lighter or darker, more saturated or more gray. It would involve changing three sliders. To make it easier to manipulate colors in terms of lightness and saturation, another coordinate system was invented: HSV (hue, saturation, value). Those terms can be made clear best by looking at the color sliders in Figure *Chartreuse in GIMPa*. Hue (running from 0 to 360) gives you the full spectrum of saturated colors. Saturation (from 0 to 1, or 100%) tells you how 'full' your color is: reduce it to zero and you only have gray scales. Value (from 0 to 1, or 100%) will bring you from black to a fully saturated color. Note that "value" is not the same as "intensity", or "lightness", used in other color geometries. "Brilliance" may be the best alternative word to describe "value". Apple calls it as "brightness", and hence refers to HSB for this color space.

Want more chartreuse or chocolate? You can specify them in GMT as 90-1-1 and 25-0.86-0.82, respectively.

19.3 The color cube

We are going to try to give you a geometric picture of color mixing in RGB and HSV by means of a tour of the RGB cube depicted in Figure *Example 11*. The geometric picture is most helpful, we think, since HSV are not orthogonal coordinates and not found from RGB by a simple algebraic transformation. So here goes: Look at the cube face with black, red, magenta, and blue corners. This is the $g = 0$ face. Orient the cube so that you are looking at this face with black in the lower left corner. Now imagine a right-handed cartesian (*rgb*) coordinate system with origin at the black point; you are looking at the $g = 0$ plane with r increasing to your right, g increasing away from you, and b increasing up. Keep this sense of (*rgb*) as you look at the cube.

Now tip the cube such that the black corner faces down and the white corner up. When looking from the top, you can see the hue, contoured in gray solid lines, running around in 360 counter-clockwise. It starts with shades of red (0), then goes through green (120) and blue (240), back to red.

On the three faces that are now on the lower side (with the white print) one of (*rgb*) is equal to 0. These three faces meet at the black corner, where $r = g = b = 0$. On these three faces the colors are fully saturated: $s = 1$. The dashed white lines indicate different levels of v, ranging from 0 to 1 with contours every 0.1.

On the upper three faces (with the black print), one of (*rgb*) is equal to the maximum value. These three faces meet at the white corner, where $r = g = b = 255$. On these three faces value is at its maximum: $v = 1$ (or 100%). The dashed black lines indicate varying levels of saturation: s ranges from 0 to 1 with contours every 0.1.

Now turn the cube around on its vertical axis (running from the black to the white corner). Along the six edges that zigzag around the "equator", both saturation and value are maximum, so $s = v = 1$. Twirling the cube around and tracing the zigzag, you will visit six of the eight corners of the cube, with changing hue (h): red (0), yellow (60), green (120), cyan (180), blue (240), and magenta (300). Three of these are the RGB colors; the other three are the CMY colors which are the complement of RGB and are used in many color hardcopy devices (see below). The only cube corners you did not visit on this path are the black and white corners. They lie on the vertical axis where hue is undefined and $r = g = b$. Any point on this axis is a shade of gray.

Let us call the points where $s = v = 1$ (points along the RYGCBM path described above) the "pure" colors. If we start at a pure color and we want to whiten it, we can keep h constant and $v = 1$ while decreasing s; this will move us along one of the cube faces toward the white point. If we start at a pure color and we want to blacken it, we can keep h constant and $s = 1$ while decreasing v; this will move us along one of the cube faces toward the black point. Any point in (*rgb*) space which can be thought of as a mixture of pure color + white, or pure color + black, is on a face of the cube.

The points in the interior of the cube are a little harder to describe. The definition for h above works at all points in (non-gray) (*rgb*) space, but so far we have only looked at (s, v) on the cube faces, not inside it. At interior points, none of (*rgb*) is equal to either 0 or 255. Choose such a point, not on the gray axis. Now draw a line through your point so that the line intersects the gray axis and also intersects the RYGCBM path of edges somewhere. It is always possible to construct this line, and all points on this line have the same hue. This construction shows that any point in RGB space can be thought of as a mixture of a pure color plus a shade of gray. If we move along this line away from the gray axis toward the pure color, we are "purifying" the color by "removing gray"; this move increases the color's saturation. When we get to the point where we cannot remove any more gray, at least one of (*rgb*) will have become zero and the color is now fully saturated; $s = 1$. Conversely, any point on the gray axis is completely undersaturated, so that $s = 0$ there. Now we see that the black point is special, s is both 0 and 1 at the same time. In other words, at the black point saturation in undefined (and so is hue). The convention is to use $h = s = v = 0$ at this point.

It remains to define value. To do so, try this: Take your point in RGB space and construct a line through it so that this line goes through the black point; produce this line from black past your point until it hits a face on which *v = 1*. All points on this line have the same hue. Note that this line and the line we made in the previous paragraph are both contained in the plane whose hue is constant. These two lines meet at some arbitrary angle which varies depending on which point you chose. Thus HSV is not an orthogonal coordinate system. If the line you made in the previous paragraph happened to touch the gray axis at the black point, then these two lines are the same line, which is why the black point is special. Now, the line we made in this paragraph illustrates the following: If your chosen point is not already at the end of the line, where *v = 1*, then it is possible to move along the line in that direction so as to increase (*rgb*) while keeping the same hue. The effect this has on a color monitor is to make the color more "brilliant", your hue will become "stronger"; if you are already on a plane where at least one of (*rgb*) = 255, then you cannot get a stronger version of the same hue. Thus, *v* measures brilliance or strength. Note that it is not quite true to say that *v* measures distance away from the black point, because *v* is not equal to $\sqrt{r^2 + g^2 + b^2}/255$.

Another representation of the HSV space is the color cone illustrated in Figure *The HSV color space*.

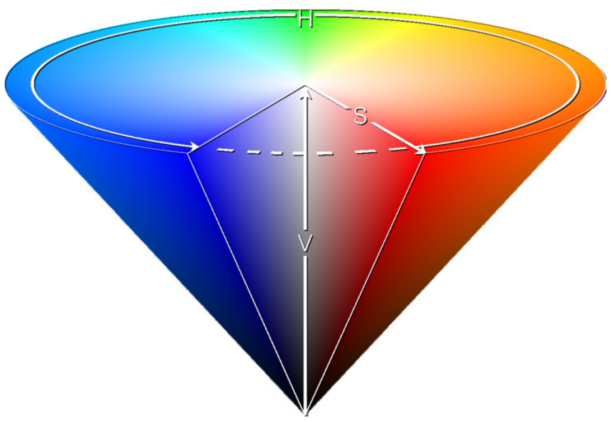


Figure 19.3: The HSV color space

19.4 Color interpolation

From studying the RGB cube, we hope you will have understood that there are different routes to follow between two colors, depending whether you are in the RGB or HSV system. Suppose you would make an interpolation between blue and red. In the RGB system you would follow a path diagonally across a face of the cube, from 0/0/255 (blue) via 127/0/127 (purple) to 255/0/0 (red). In the HSV system, you would trace two edges, from 240-1-1 (blue) via 300-1-1 (magenta) to 360-1-1 (red). That is even assuming software would be smart enough to go the shorter route. More likely, red will be recorded as 0-1-1, so hue will be interpolated the other way around, reducing hue from 240 to 0, via cyan, green, and yellow.

Depending on the design of your color palette, you may want to have it either way. By default, GMT interpolates in RGB space, even when the original color palette is in the HSV system. However, when you add the line `#COLOR_MODEL=+HSV` (with the leading '+' sign) in the header of the color palette file, GMT will not only read the color representation as HSV values, but also interpolate colors in the HSV system. That means that H, S, and V values are interpolated linearly between two colors, instead of their respective R, G, and B values.

The top row in Figure *Interpolating colors* illustrates two examples: a blue-white-red scale (the palette in Appendix [app:M]) interpolated in RGB and the palette interpolated in HSV. The bottom row of the

Figure demonstrates how things can go terribly wrong when you do the interpolation in the other system.

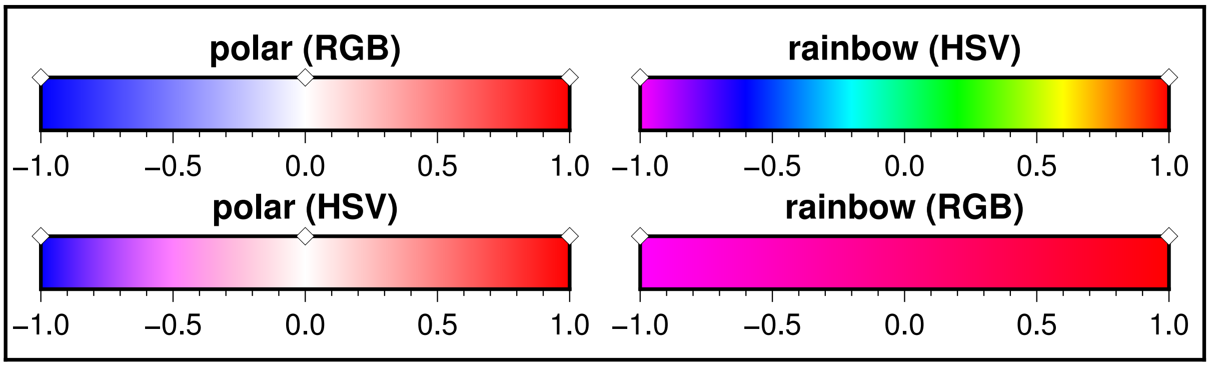


Figure 19.4: When interpolating colors, the color system matters. The polar palette on the left needs to be interpolated in RGB, otherwise hue will change between blue (240) and white (0). The rainbow palette should be interpolated in HSV, since only hue should change between magenta (300) and red (0). Diamonds indicate which colors are defined in the palettes; they are fixed, the rest is interpolated.

19.5 Artificial illumination

GMT uses the HSV system to achieve artificial illumination of colored images (e.g., **-I** option in `grdimage`) by changing the saturation s and value v coordinates of the color. When the intensity is zero (flat illumination), the data are colored according to the CPT file. If the intensity is non-zero, the color is either lightened or darkened depending on the illumination. The color is first converted to HSV (if necessary) and then darkened by moving (sv) toward (*COLOR_HSV_MIN_S*, *COLOR_HSV_MIN_V*) if the intensity is negative, or lightened by sliding (sv) toward (*COLOR_HSV_MAX_S*, *COLOR_HSV_MAX_V*) if the illumination is positive. The extremes of the s and v are defined in the `gmt.conf` file and are usually chosen so the corresponding points are nearly black ($s = 1$, $v = 0$) and white ($s = 0$, $v = 1$). The reason this works is that the HSV system allows movements in color space which correspond more closely to what we mean by “tint” and “shade”; an instruction like “add white” is easy in HSV and not so obvious in RGB.

19.6 Thinking in RGB or HSV

The RGB system is understandable because it is cartesian, and we all learned cartesian coordinates in school. But it doesn’t help us create a tint or shade of a color; we cannot say, “We want orange, and a lighter shade of orange, or a less vivid orange”. With HSV we can do this, by saying, “Orange must be between red and yellow, so its hue is about $h = 30$; a less vivid orange has a lesser s, a darker orange has a lesser v”. On the other hand, the HSV system is a peculiar geometric construction, more like a cone (Figure *The HSV color space*). It is not an orthogonal coordinate system, and it is not found by a matrix transformation of RGB; these make it difficult in some cases too. Note that a move toward black or a move toward white will change both s and v, in the general case of an interior point in the cube. The HSV system also doesn’t behave well for very dark colors, where the gray point is near black and the two lines we constructed above are almost parallel. If you are trying to create nice colors for drawing chocolates, for example, you may be better off guessing in RGB coordinates.

19.7 CMYK color system

Finally, you can imagine that printers work in a different way: they mix different paints to make a color. The more paint, the darker the color, which is the reverse of adding more light. Also, mixing more colored paints does not give you true black, so that means that you really need four colors to do it right. Open up your color printer and you'll probably find four cartridges: cyan, magenta, yellow (often these are combined into one), and black. They form the CMYK system of colors, each value running from 0 to 1 (or 100%). In GMT CMYK color coding can be achieved using *c/m/y/k* quadruplets.

Obviously, there is no unique way to go from the 3-dimensional RGB system to the 4-dimensional CMYK system. So, again, there is a lot of hand waving applied in the transformation. Strikingly, CMYK actually covers a smaller color space than RGB. We will not try to explain you the details behind it, just know that there is a transformation needed to go from the colors on your screen to the colors on your printer. It might explain why what you see is not necessarily what you get. If you are really concerned about how your color plots will show up in your PhD thesis, for example, it might be worth trying to save and print all your color plots using the CMYK system. Letting GMT do the conversion to CMYK may avoid some nasty surprises when it comes down to printing. To specify the color space of your PostScript file, set *PS_COLOR_MODEL* in the `gmt.conf` file to RGB, HSV, or CMYK.

I. Filtering of Data in GMT

The GMT programs `filter1d` (for tables of data indexed to one independent variable) and `grdfilter` (for data given as 2-dimensional grids) allow filtering of data by a moving-window process. (To filter a grid by Fourier transform use `grdfft`.) Both programs use an argument **-F**<*type*><*width*> to specify the type of process and the window's width (in 1-D) or diameter (in 2-D). (In `filter1d` the width is a length of the time or space ordinate axis, while in `grdfilter` it is the diameter of a circular area whose distance unit is related to the grid mesh via the **-D** option). If the process is a median, mode, or extreme value estimator then the window output cannot be written as a convolution and the filtering operation is not a linear operator. If the process is a weighted average, as in the boxcar, cosine, and gaussian filter types, then linear operator theory applies to the filtering process. These three filters can be described as convolutions with an impulse response function, and their transfer functions can be used to describe how they alter components in the input as a function of wavelength.

Impulse responses are shown here for the boxcar, cosine, and gaussian filters. Only the relative amplitudes of the filter weights shown; the values in the center of the window have been fixed equal to 1 for ease of plotting. In this way the same graph can serve to illustrate both the 1-D and 2-D impulse responses; in the 2-D case this plot is a diametrical cross-section through the filter weights (Figure *Impulse responses*).

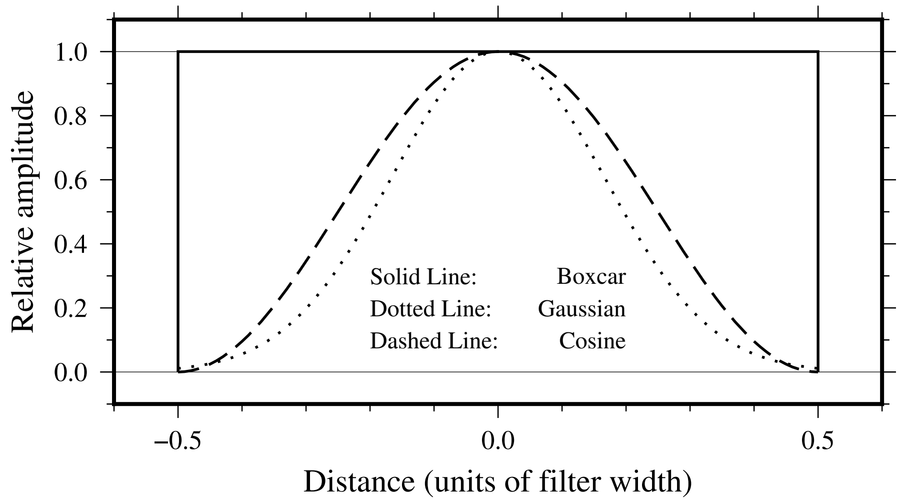


Figure 20.1: Impulse responses for GMT filters.

Although the impulse responses look the same in 1-D and 2-D, this is not true of the transfer functions; in 1-D the transfer function is the Fourier transform of the impulse response, while in 2-D it is the Hankel transform of the impulse response. These are shown in Figures *Transfer functions for 1D* and *2D*, respectively. Note that in 1-D the boxcar transfer function has its first zero crossing at $f = 1$, while in 2-D it is around $f \sim 1.2$. The 1-D cosine transfer function has its first zero crossing at *f* = *2*; so a cosine filter needs to be twice as wide as a boxcar filter in order to zero the same lowest frequency. As a general rule, the cosine and gaussian filters are "better" in the sense that they do not have the "side lobes" (large-amplitude oscillations in the transfer function) that the boxcar filter has. However, they are correspondingly "worse" in the sense that they require more work (doubling the width to achieve the same cut-off wavelength).

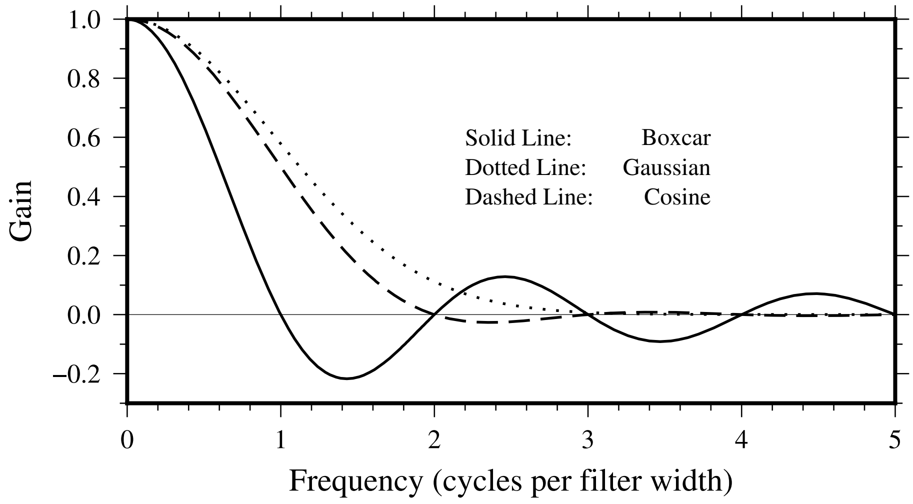


Figure 20.2: Transfer functions for 1-D GMT filters.

One of the nice things about the gaussian filter is that its transfer functions are the same in 1-D and 2-D. Another nice property is that it has no negative side lobes. There are many definitions of the gaussian filter in the literature (see page 7 of Bracewell [1]). We define σ equal to 1/6 of the filter width, and the impulse response proportional to $\exp[-0.5(t/\sigma)^2)$. With this definition, the transfer function is $\exp[-2(\pi\sigma f)^2]$ and the wavelength at which the transfer function equals 0.5 is about 5.34 σ, or about 0.89 of the filter width.

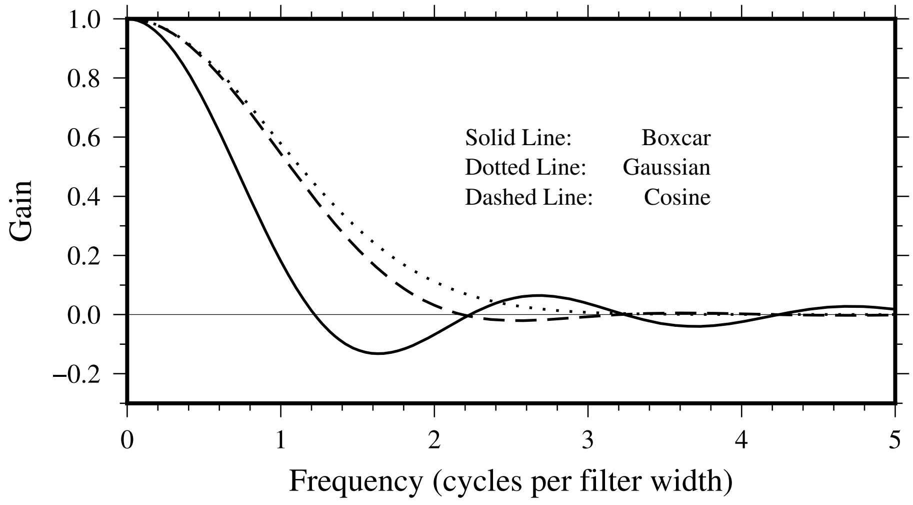


Figure 20.3: Transfer functions for 2-D (radial) GMT filters.

[1] R. Bracewell, *The Fourier Transform and its Applications*, McGraw-Hill, London, 444 p., 1965.

CHAPTER 21

J. The GMT High-Resolution Coastline Data

Starting with version 3.0, GMT use a completely new coastline database and the `pscoast` utility was been completely rewritten to handle the new file format. Many users have asked us why it has taken so long for GMT to use a high-resolution coastline database; after all, such data have been available in the public domain for years. To answer such questions we will take you along the road that starts with these public domain data sets and ends up with the database used by GMT.

21.1 Selecting the right data

There are two well-known public-domain data sets that could be used for this purpose. Once is known as the World Data Bank II or CIA Data Bank (WDB) and contains coastlines, lakes, political boundaries, and rivers. The other, the World Vector Shoreline (WVS) only contains shorelines between saltwater and land (i.e., no lakes). It turns out that the WVS data is far superior to the WDB data as far as data quality goes, but as noted it lacks lakes, not to mention rivers and borders. We decided to use the WVS whenever possible and supplement it with WDB data. We got these data over the Internet; they are also available on CD-ROM from the National Geophysical Data Center in Boulder, Colorado [1].

21.2 Format required by GMT

In order to paint continents or oceans it is necessary that the coastline data be organized in polygons that may be filled. Simple line segments can be used to draw the coastline, but for painting polygons are required. Both the WVS and WDB data consists of unsorted line segments: there is no information included that tells you which segments belong to the same polygon (e.g., Australia should be one large polygon). In addition, polygons enclosing land must be differentiated from polygons enclosing lakes since they will need different paint. Finally, we want `pscoast` to be flexible enough that it can paint the land *or* the oceans *or* both. If just land (or oceans) is selected we do not want to paint those areas that are not land (or oceans) since previous plot programs may have drawn in those areas. Thus, we will need to combine polygons into new polygons that lend themselves to fill land (or oceans) only (Note that older versions of `pscoast` always painted lakes and wiped out whatever was plotted beneath).

21.3 The long and winding road

The WVS and WDB together represent more than 100 Mb of binary data and something like 20 million data points. Hence, it becomes obvious that any manipulation of these data must be automated.

[1] National Geophysical Data Center, Boulder, Colorado

For instance, the reasonable requirement that no coastline should cross another coastline becomes a complicated processing step.

- To begin, we first made sure that all data were "clean", i.e., that there were no outliers and bad points. We had to write several programs to ensure data consistency and remove "spikes" and bad points from the raw data. Also, crossing segments were automatically "trimmed" provided only a few points had to be deleted. A few hundred more complicated cases had to be examined semi-manually.
- Programs were written to examine all the loose segments and determine which segments should be joined to produce polygons. Because not all segments joined exactly (there were non-zero gaps between some segments) we had to find all possible combinations and choose the simplest combinations. The WVS segments joined to produce more than 200,000 polygons, the largest being the Africa-Eurasia polygon which has 1.4 million points. The WDB data resulted in a smaller data base (~25% of WVS).
- We now needed to combine the WVS and WDB data bases. The main problem here is that we have duplicates of polygons: most of the features in WVS are also in WDB. However, because the resolution of the data differ it is nontrivial to figure out which polygons in WDB to include and which ones to ignore. We used two techniques to address this problem. First, we looked for crossovers between all possible pairs of polygons. Because of the crossover processing in step 1 above we know that there are no remaining crossovers within WVS and WDB; thus any crossovers would be between WVS and WDB polygons. Crossovers could mean two things: (1) A slightly misplaced WDB polygon crosses a more accurate WVS polygon, both representing the same geographic feature, or (2) a misplaced WDB polygon (e.g., a small coastal lake) crosses the accurate WVS shoreline. We distinguished between these cases by comparing the area and centroid of the two polygons. In almost all cases it was obvious when we had duplicates; a few cases had to be checked manually. Second, on many occasions the WDB duplicate polygon did not cross its WVS counterpart but was either entirely inside or outside the WVS polygon. In those cases we relied on the area-centroid tests.
- While the largest polygons were easy to identify by visual inspection, the majority remain unidentified. Since it is important to know whether a polygon is a continent or a small pond inside an island inside a lake we wrote programs that would determine the hierarchical level of each polygon. Here, level = 1 represents ocean/land boundaries, 2 is land/lakes borders, 3 is lakes/islands-in-lakes, and 4 is islands-in-lakes/ponds-in-islands-in-lakes. Level 4 was the highest level encountered in the data. To automatically determine the hierarchical levels we wrote programs that would compare all possible pairs of polygons and find how many polygons a given polygon was inside. Because of the size and number of the polygons such programs would typically run for 3 days on a Sparc-2 workstation.
- Once we know what type a polygon is we can enforce a common "orientation" for all polygons. We arranged them so that when you move along a polygon from beginning to end, your left hand is pointing toward "land". At this step we also computed the area of all polygons since we would like the option to plot only features that are bigger than a minimum area to be specified by the user.
- Obviously, if you need to make a map of Denmark then you do not want to read the entire 1.4 million points making up the Africa-Eurasia polygon. Furthermore, most plotting devices will not let you paint and fill a polygon of that size due to memory restrictions. Hence, we need to partition the polygons so that smaller subsets can be accessed rapidly. Likewise, if you want to plot a world map on a letter-size paper there is no need to plot 10 million data points as most of them will plot several times on the same pixel and the operation would take a very long time to complete. We chose to make 5 versions on the database, corresponding to different resolutions. The

decimation was carried out using the Douglas-Peucker (DP) line-reduction algorithm [2]. We chose the cutoffs so that each subset was approximately 20% the size of the next higher resolution. The five resolutions are called **f**ull, **h**igh, **i**ntermediate, **l**ow, and **c**rude; they are accessed in `pscoast`, `gmtselect`, and `grdlandmask` with the **-D** option [3]. For each of these 5 data sets (**f**, **h**, **i**, **l**, **c**) we specified an equidistant grid (1, 2, 5, 10, 20) and split all polygons into line-segments that each fit inside one of the many boxes defined by these grid lines. Thus, to paint the entire continent of Australia we instead paint many smaller polygons made up of these line segments and gridlines. Some book-keeping has to be done since we need to know which parent polygon these smaller pieces came from in order to prescribe the correct paint or ignore if the feature is smaller than the cutoff specified by the user. The resulting segment coordinates were then scaled to fit in short integer format to preserve precision and written in netCDF format for ultimate portability across hardware platforms [4].

- While we are now back to a file of line-segments we are in a much better position to create smaller polygons for painting. Two problems must be overcome to correctly paint an area:
  - We must be able to join line segments and grid cell borders into meaningful polygons; how we do this will depend on whether we want to paint the land or the oceans.
  - We want to nest the polygons so that no paint falls on areas that are “wet” (or “dry”); e.g., if a grid cell completely on land contains a lake with a small island, we do not want to paint the lake and then draw the island, but paint the annulus or “donut” that is represented by the land and lake, and then plot the island.

  GMT uses a polygon-assembly routine that carries out these tasks on the fly.

21.4 The Five Resolutions

We will demonstrate the power of the new database by starting with a regional hemisphere map centered near Papua New Guinea and zoom in on a specified point. The map regions will be specified in projected km from the projection center, e.g., we may want the map to go from km to km in the longitudinal and the latitudinal direction. Also, as we zoom in on the projection center we want to draw the outline of the next map region on the plot. To do that we use the **-D** option in `psbasemap`.

21.4.1 The crude resolution (-Dc)

We begin with an azimuthal equidistant map of the hemisphere centered on 130°21’E, 0°12’S, which is slightly west of New Guinea, near the Strait of Dampier. The edges of the map are all 9000 km true distance from the projection center. At this scale (and for global maps) the crude resolution data will usually be adequate to capture the main geographic features. To avoid cluttering the map with insignificant detail we only plot features (i.e., polygons) that exceed 500 km^2 in area. Smaller features would only occupy a few pixels on the plot and make the map look “dirty”. We also add national borders to the plot. The crude database is heavily decimated and simplified by the DP-routine: The total file size of the coastlines, rivers, and borders database is only 283 kbytes. The plot is produced by the script:

[2] Douglas, D.H., and T. K. Peucker, 1973, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, *Canadian Cartographer*, 10, 112–122.

[3] The full and high resolution files are in separate archives because of their size. Not all users may need these files as the intermediate data set is better than the data provided with version 2.2.4.

[4] If you need complete polygons in a simpler format, see the article on GSHHG (Wessel, P., and W. H. F. Smith, 1996, A Global, self-consistent, hierarchical, high-resolution shoreline database, *J. Geophys. Res. 101*, 8741–8743).

```
gmt set MAP_GRID_CROSS_SIZE_PRIMARY 0 MAP_ANNOT_OBLIQUE 22 MAP_ANNOT_MIN_SPACING 0.3i
gmt pscoast -Rk-9000/9000/-9000/9000 -JE130.35/-0.2/3.5i -P -Dc -A500 \
            -Gburlywood -Sazure -Wthinnest -N1/thinnest,- -B20g20 -BWSne -K > GMT_App_K_1.ps
gmt psbasemap -R -J -O -Dk2000+c130.35/-0.2+pthicker >> GMT_App_K_1.ps
```

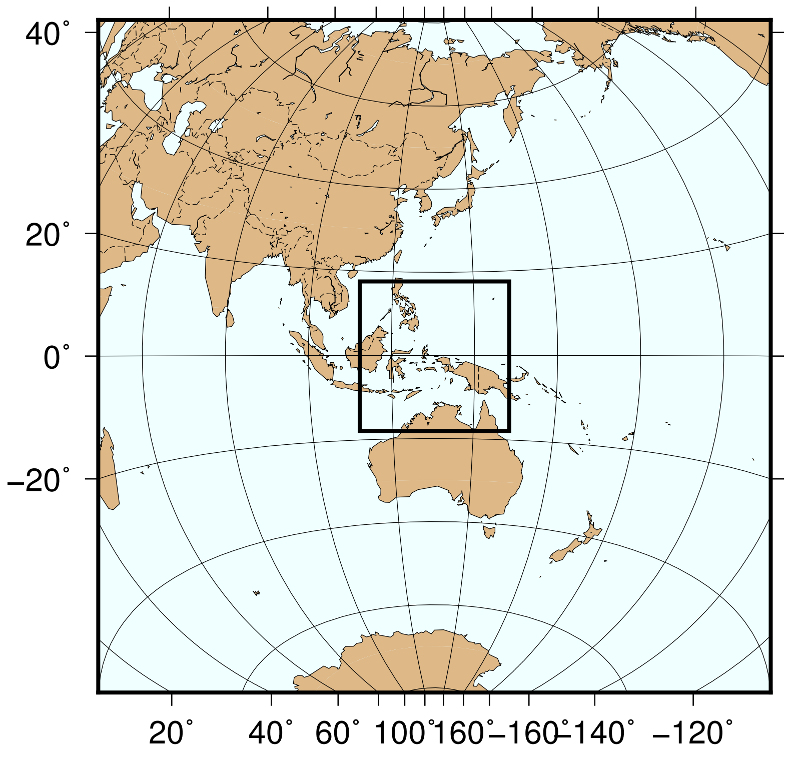


Figure 21.1: Map using the crude resolution coastline data.

Here, we use the *MAP_ANNOT_OBLIQUE* bit flags to achieve horizontal annotations and set *MAP_ANNOT_MIN_SPACING* to suppress some longitudinal annotations near the S pole that otherwise would overprint. The square box indicates the outline of the next map.

21.4.2 The low resolution (-Dl)

We have now reduced the map area by zooming in on the map center. Now, the edges of the map are all 2000 km true distance from the projection center. At this scale we choose the low resolution data that faithfully reproduce the dominant geographic features in the region. We cut back on minor features less than 100 km^2 in area. We still add national borders to the plot. The low database is less decimated and simplified by the DP-routine: The total file size of the coastlines, rivers, and borders combined grows to 907 kbytes; it is the default resolution in GMT. The plot is generated by the script:

```
gmt pscoast -Rk-2000/2000/-2000/2000 -JE130.35/-0.2/3.5i -P -Dl -A100 -Gburlywood \
            -Sazure -Wthinnest -N1/thinnest,- -B10g5 -BWSne -K > GMT_App_K_2.ps
gmt psbasemap -R -J -O -Dk500+c130.35/-0.2+pthicker >> GMT_App_K_2.ps
```

21.4.3 The intermediate resolution (-Di)

We continue to zoom in on the map center. In this map, the edges of the map are all 500 km true distance from the projection center. We abandon the low resolution data set as it would look too jagged at this scale and instead employ the intermediate resolution data that faithfully reproduce the dominant geographic features in the region. This time, we ignore features less than 20 km^2 in area. Although the

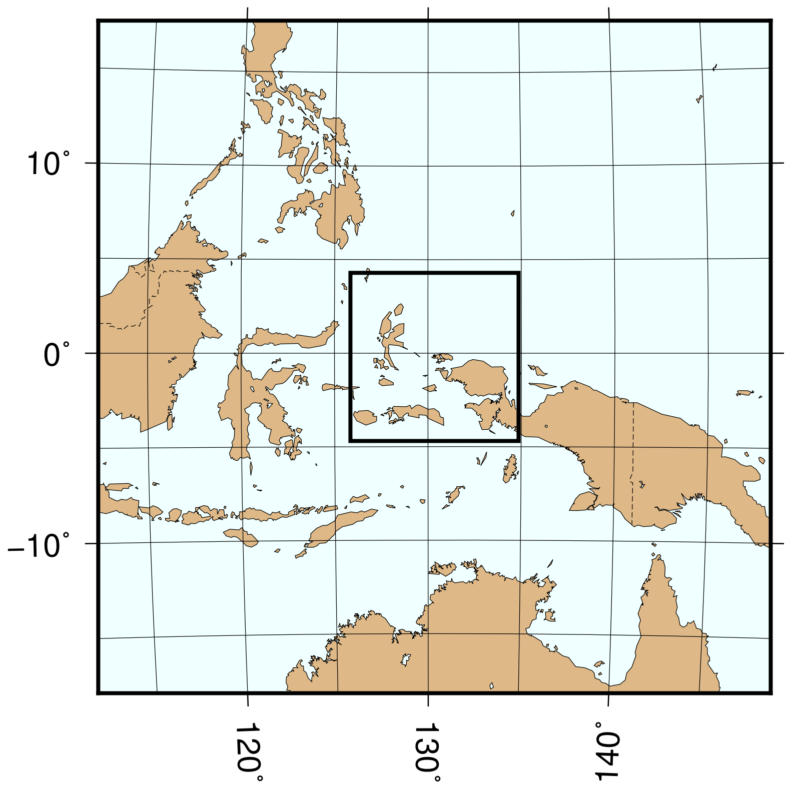


Figure 21.2: Map using the low resolution coastline data.

script still asks for national borders none exist within our region. The intermediate database is moderately decimated and simplified by the DP-routine: The combined file size of the coastlines, rivers, and borders now exceeds 3.35 Mbytes. The plot is generated by the script:

```
gmt pscoast -Rk-500/500/-500/500 -JE130.35/-0.2/3.5i -P -Di -A20 -Gburlywood \
            -Sazure -Wthinnest -N1/thinnest,- -B2g1 -BWSne -K > GMT_App_K_3.ps
echo 133 2 | gmt psxy -R -J -O -K -Sc1.4i -Gwhite >> GMT_App_K_3.ps
gmt psbasemap -R -J -O -K --FONT_TITLE=12p --MAP_TICK_LENGTH_PRIMARY=0.05i \
              -Tm133/2/1i::+45/10/5 --FONT_ANNOT_SECONDARY=8p >> GMT_App_K_3.ps
gmt psbasemap -R -J -O -Dk100+c130.35/-0.2+pthicker >> GMT_App_K_3.ps
```

21.4.4 The high resolution (-Dh)

The relentless zooming continues! Now, the edges of the map are all 100 km true distance from the projection center. We step up to the high resolution data set as it is needed to accurately portray the detailed geographic features within the region. Because of the small scale we only ignore features less than 1 km^2 in area. The high resolution database has undergone minor decimation and simplification by the DP-routine: The combined file size of the coastlines, rivers, and borders now swells to 12.3 Mbytes. The map and the final outline box are generated by these commands:

```
gmt pscoast -Rk-100/100/-100/100 -JE130.35/-0.2/3.5i -P -Dh -A1 -Gburlywood \
            -Sazure -Wthinnest -N1/thinnest,- -B30mg10m -BWSne -K > GMT_App_K_4.ps
gmt psbasemap -R -J -O -Dk20+c130.35/-0.2+pthicker >> GMT_App_K_4.ps
```

21.4.5 The full resolution (-Df)

We now arrive at our final plot, which shows a detailed view of the western side of the small island of Waigeo. The map area is approximately 40 by 40 km. We call upon the full resolution data set to

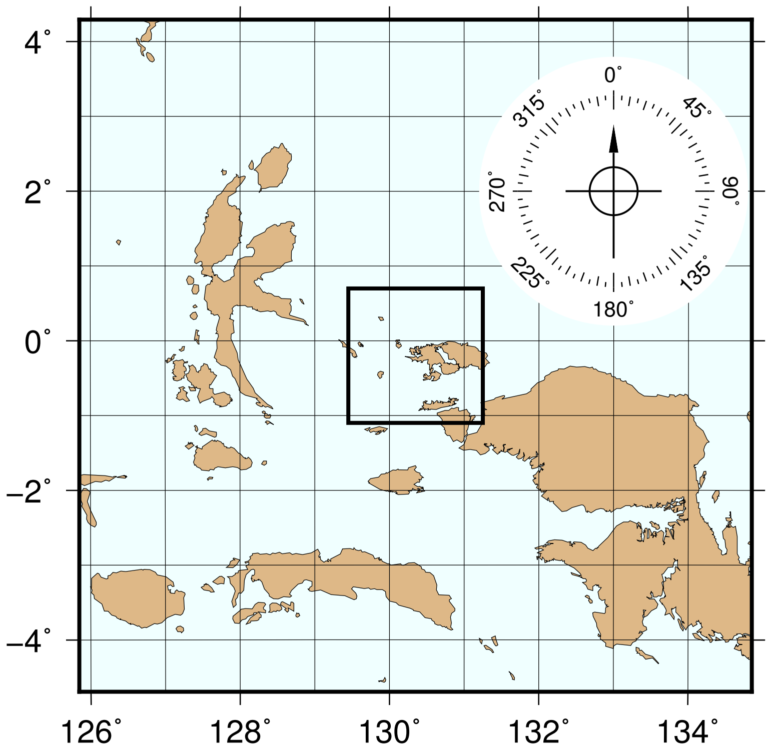


Figure 21.3: Map using the intermediate resolution coastline data. We have added a compass rose just because we have the power to do so.

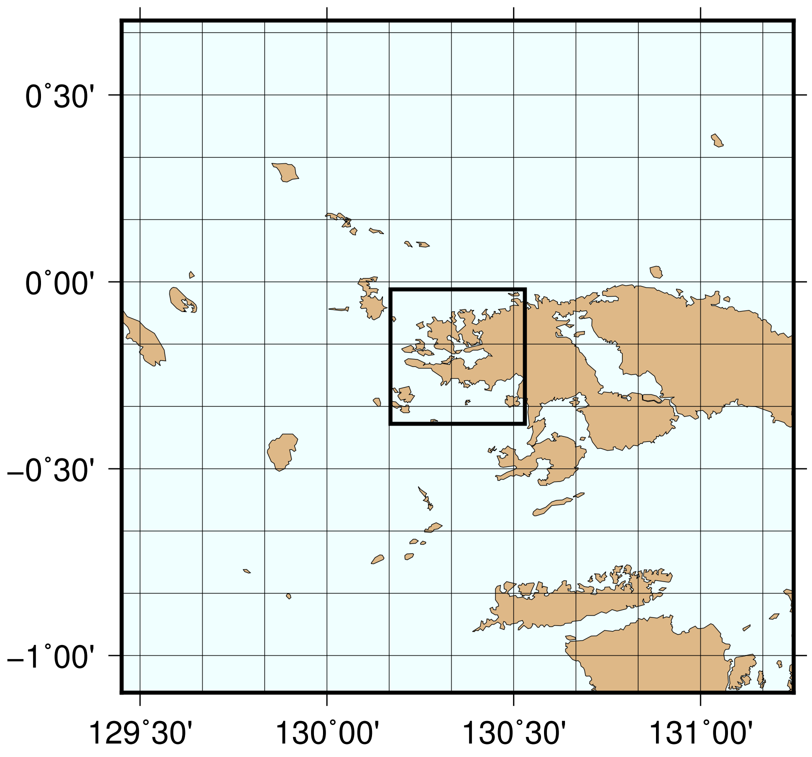


Figure 21.4: Map using the high resolution coastline data.

portray the richness of geographic detail within this region; no features are ignored. The full resolution has undergone no decimation and it shows: The combined file size of the coastlines, rivers, and borders totals a (once considered hefty) 55.9 Mbytes. Our final map is reproduced by the single command:

```
gmt pscoast -Rk-20/20/-20/20 -JE130.35/-0.2/3.5i -P -Df -Gburlywood \
            -Sazure -Wthinnest -N1/thinnest,- -B10mg2m -BWSne > GMT_App_K_5.ps
```

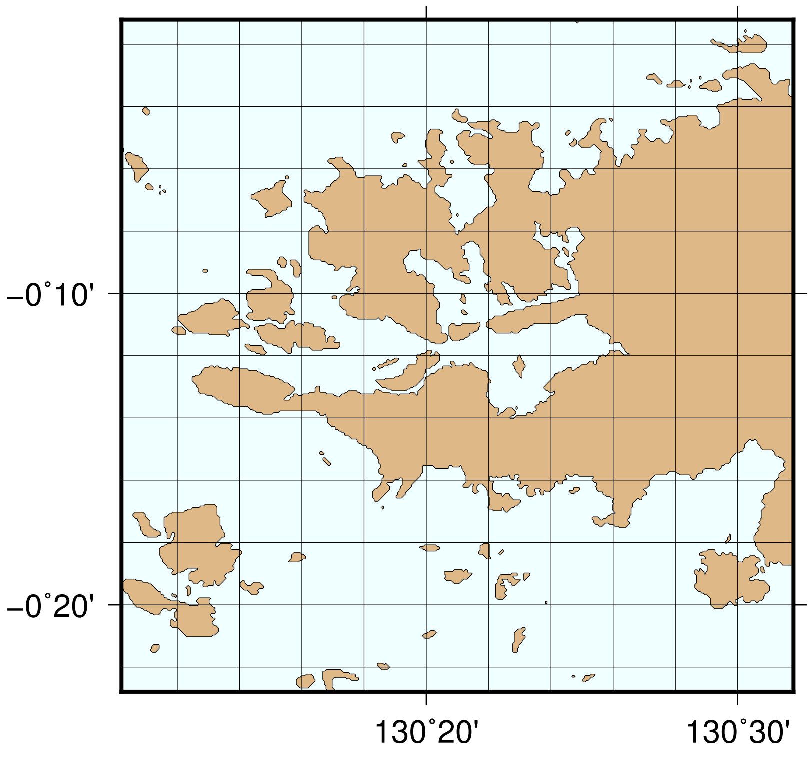


Figure 21.5: Map using the full resolution coastline data.

We hope you will study these examples to enable you to make efficient and wise use of this vast data set.

K. GMT on non-UNIX Platforms

22.1 Introduction

While GMT was ported to non-UNIX systems such as Windows, it is also true that one of the strengths of GMT lies its symbiotic relationship with UNIX. We therefore recommend that GMT be installed in a POSIX-compliant UNIX environment such as traditional UNIX-systems, Linux, or Mac OS X. If abandoning your non-UNIX operating system is not an option, consider one of these solutions:

WINDOWS: Choose among these three possibilities:

1. Install GMT under MinGW/MSYS (A collection of GNU utilities).
2. Install GMT under Cygwin (A GNU port to Windows).
3. Install GMT in Windows using Microsoft C/C++ or other compilers. Unlike the first two, this option will not provide you with any UNIX tools so you will be limited to what you can do with DOS batch files.

22.2 Cygwin and GMT

Because GMT works best in conjugation with UNIX tools we suggest you install GMT using the Cygwin product from Cygnus (now assimilated by Redhat, Inc.). This free version works on any Windows version and it comes with both the Bourne Again shell **bash** and the **tcsh**. You also have access to most standard GNU development tools such as compilers and text processing tools (**awk**, **grep**, **sed**, etc.). Note that executables prepared for Windows will also run under Cygwin.

Follow the instructions on the Cygwin page on how to install the package; note you must explicitly add all the development tool packages (e.g., **gcc** etc) as the basic installation does not include them by default. Once you are up and running under Cygwin, you may install GMT the same way you do under any other UNIX platform by either running the automated install via **install_gmt** or manually running configure first, then type make all. If you install via the web form, make sure you save the parameter file without DOS CR/LF endings. Use **dos2unix** to get rid of those if need be.

Finally, from Cygwin's User Guide: By default, no Cygwin program can allocate more than 384 MB of memory (program and data). You should not need to change this default in most circumstances. However, if you need to use more real or virtual memory in your machine you may add an entry in either the **HKEY_LOCAL_MACHINE** (to change the limit for all users) or **HKEY_CURRENT_USER** (for just the current user) section of the registry. Add the DWORD value **heap_chunk_in_mb** and set it to the desired memory limit in decimal Mb. It is preferred to do this in Cygwin using the **regtool** program included in the Cygwin package. (For more information about **regtool** or the other Cygwin utilities, see the Section called Cygwin Utilities in Chapter 3 of the Cygwin's User Guide or use the help option of each utility.) You should always be careful when using **regtool** since damaging your system registry can result in an unusable system. This example sets the local machine memory limit to 1024 Mb:

```
regtool -i set /HKLM/Software/Cygnus\ Solutions/Cygwin/heap_chunk_in_mb 1024
regtool -v list /HKLM/Software/Cygnus\ Solutions/Cygwin
```

For more installation details see the general README file.

22.3 MINGW|MSYS and GMT

Though one can install GMT natively using CMake, the simplest way of installing under MINGW|MSYS is to just install the Windows binaries and use them from the msys bash shell. As simple as that. Furthermore, GMT programs launch faster here than on Cygwin so this is the recommended way of running GMT on Windows.

CHAPTER 23

L. Of Colors and Color Legends

23.1 Built-in color palette tables

Figures *CPT files a* and *b* show the 36 built-in color palettes, stored in so-called CPT tables [1]. The programs `makecpt` and `grd2cpt` are used to access these master CPT tables and translate/scale them to fit the user's range of z-values. The top half of the color bars in the Figure shows the original color scale, which can be either discrete or continuous, though some (like **globe**) are a mix of the two. The bottom half the color bar are built by using `makecpt` **-T**-1/1/0.25, thus splitting the color scale into 8 discrete colors.

23.2 Labeled and non-equidistant color legends

[app:colorbars] The use of color legends has already been introduced in Chapter [ch:7] (examples 2, 16, and 17). Things become a bit more complicated when you want to label the legend with names for certain intervals (like geological time periods in the example below). To accomplish that, one should add a semi-colon and the label name at the end of a line in the CPT table and add the **-L** option to the `psscale` command that draws the color legend. This option also makes all intervals in the legend of equal length, even it the numerical values are not equally spaced.

Normally, the name labels are plotted at the lower end of the intervals. But by adding a *gap* amount (even when zero) to the **-L** option, they are centered. The example below also shows how to annotate ranges using **-Li** (in which case no name labels should appear in the CPT file), and how to switch the color bar around (by using a negative length).

For additional color tables, visit cpt-city.

[1] The 23rd palette is called *random* and produces a random set of colors suitable for categorical plots.

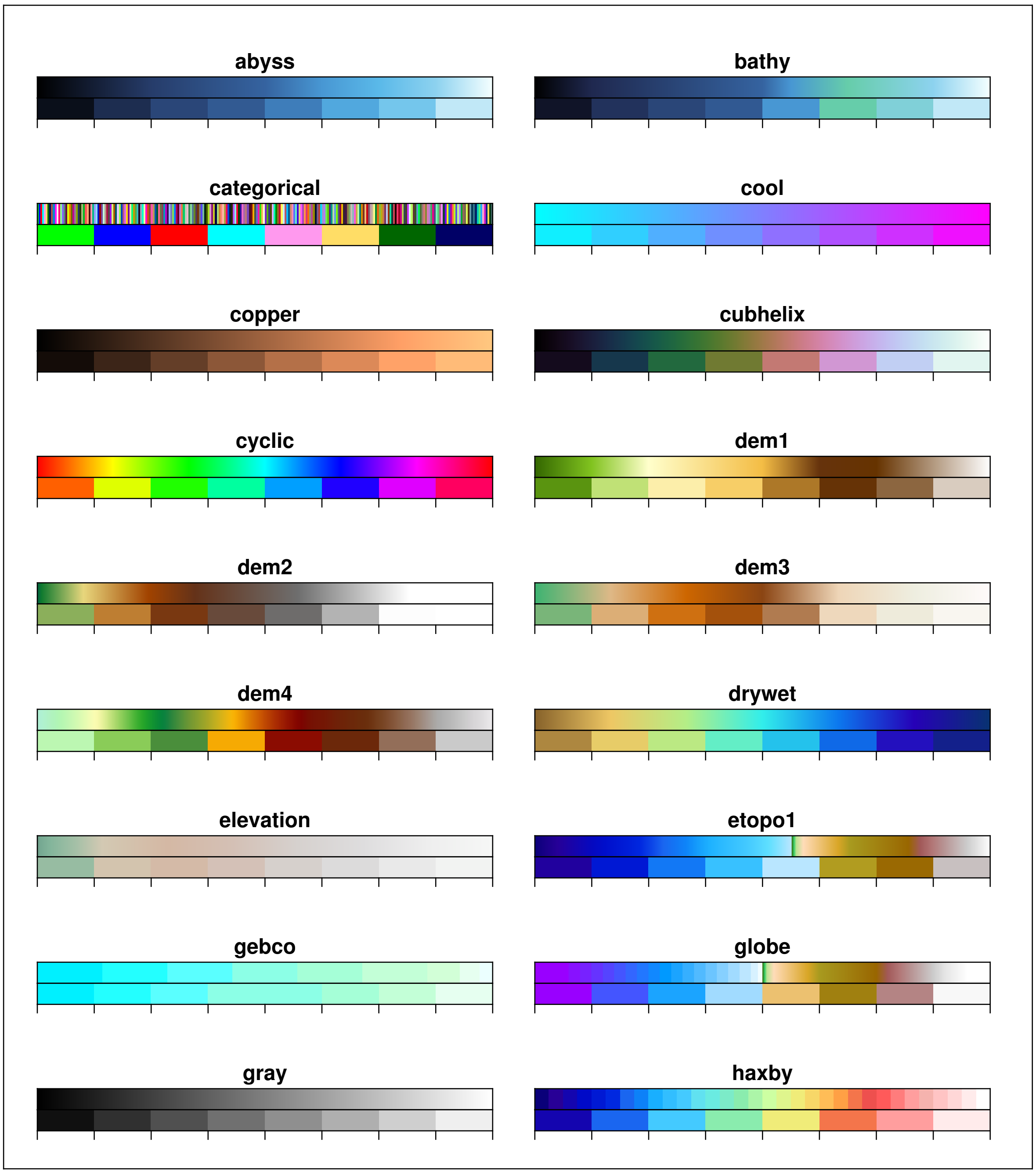


Figure 23.1: The first 18 of the standard 36 CPT files supported by GMT

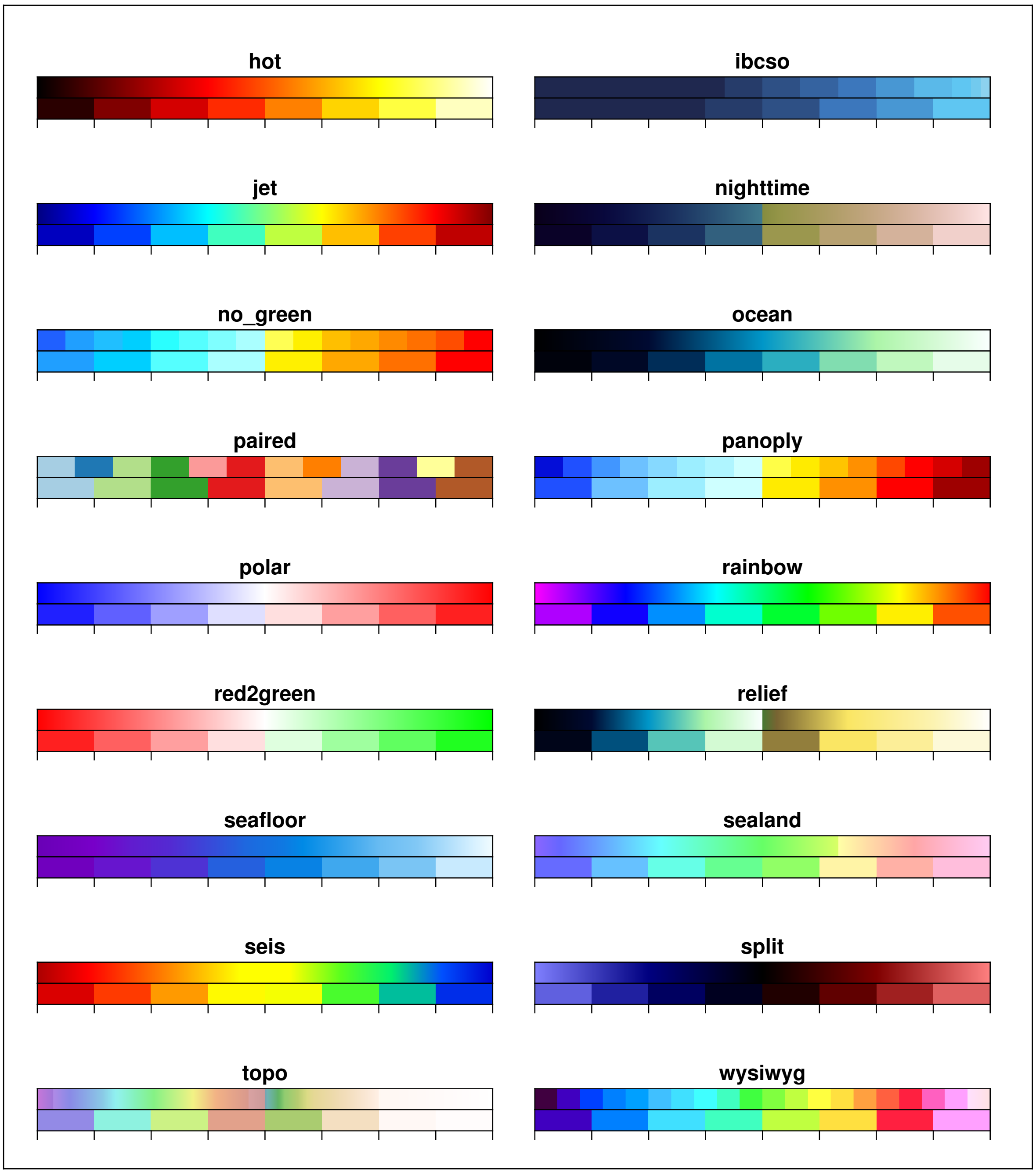
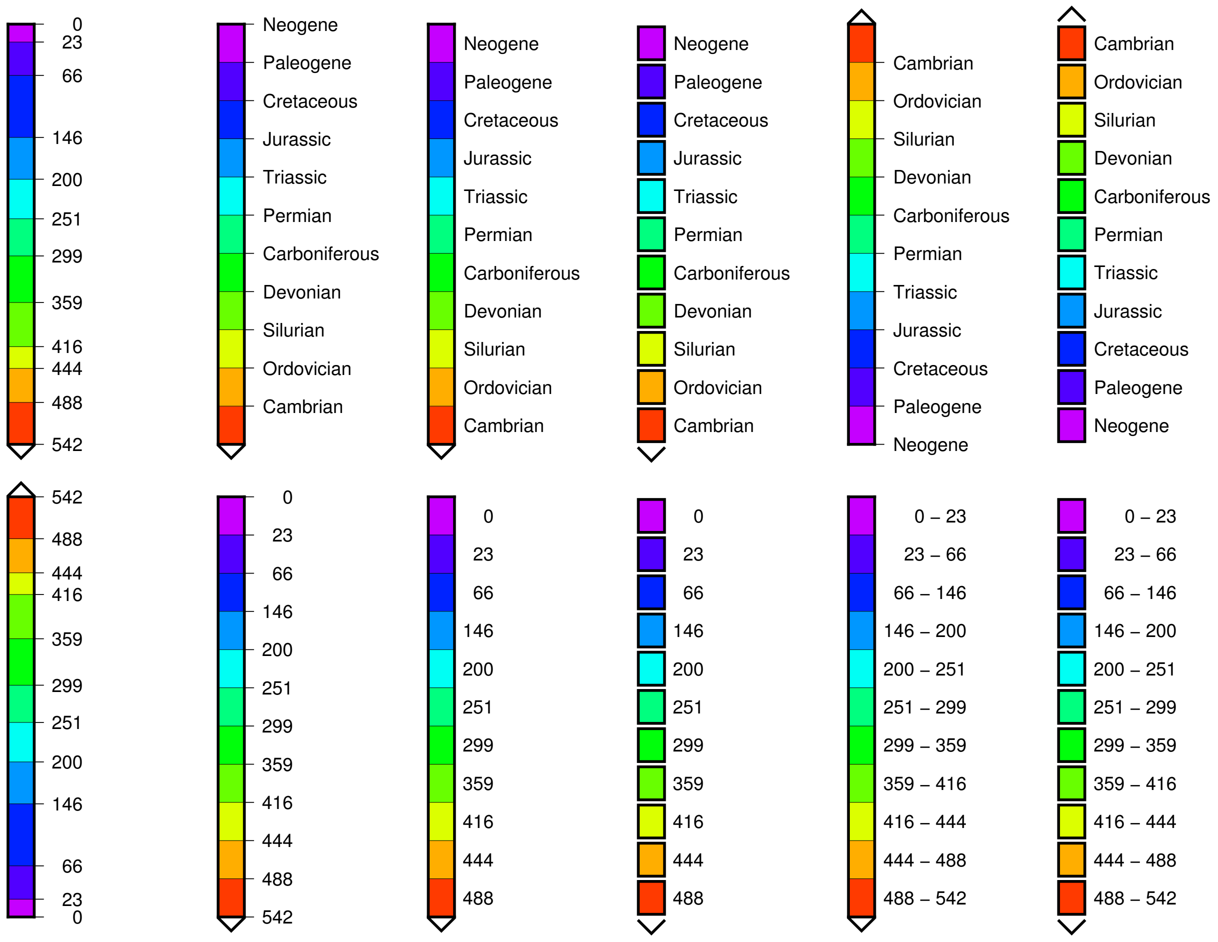


Figure 23.2: The second 18 of the standard 36 CPT files supported by GMT


0
23
66
146
200
251
299
359
416
444
488
542
Neogene
Paleogene
Cretaceous
Jurassic
Triassic
Permian
Carboniferous
Devonian
Silurian
Ordovician
Cambrian
Cambrian
Ordovician
Silurian
Devonian
Carboniferous
Permian
Triassic
Jurassic
Cretaceous
Paleogene
Neogene
0 – 23
23 – 66
66 – 146
146 – 200
200 – 251
251 – 299
299 – 359
359 – 416
416 – 444
444 – 488
488 – 542

M. Custom Plot Symbols

24.1 Background

The GMT tools `psxy` and `psxyz` are capable of using custom symbols as alternatives to the built-in, standard geometrical shapes like circles, triangles, and many others. One the command line, custom symbols are selected via the **-Sk***symbolname*[*size*] symbol selection, where *symbolname* refers to a special symbol definition file called `symbolname.def` that must be available via the standard GMT user paths. Several custom symbols comes pre-configured with GMT(see Figure *Custom symbols*)

You may find it convenient to examine some of these and use them as a starting point for your own design; they can be found in GMT's share/custom directory.

24.2 The macro language

To make your own custom plot symbol, you will need to design your own *.def files. This section defines the language used to build custom symbols. You can place these definition files in your current directory or your .gmt user directory. When designing the symbol, you are doing so in a relative coordinate system centered on (0,0). This point will be mapped to the actual location specified by your data coordinates. Furthermore, your symbol should be constructed within the domain $-\frac{1}{2}, +\frac{1}{2}, -\frac{1}{2}, +\frac{1}{2}$, resulting in a 1 by 1 relative canvas area. This 1 x 1 square will be scaled to your actual symbol size when plotted.

24.2.1 Comment lines

Your definition file may have any number of comment lines, defined to begin with the character #. These are skipped by GMT but provides a mechanism for you to clarify what your symbol does.

24.2.2 Symbol variables

Simple symbols, such as circles and triangles, only take a single parameter: the symbol size, which is either given on the command line (via **-Sk**) or as part of the input data. However, more complicated symbols, such as the ellipse or vector symbols, may require more parameters. If your custom symbol requires more than the single size parameter you must include the line

> **N**: *n_extra_parameters* [*types*]

before any other macro commands. It is an optional statement in that *n_extra_parameters* will default to 0 unless explicitly set. By default the extra parameters are considered to be quantities that should be

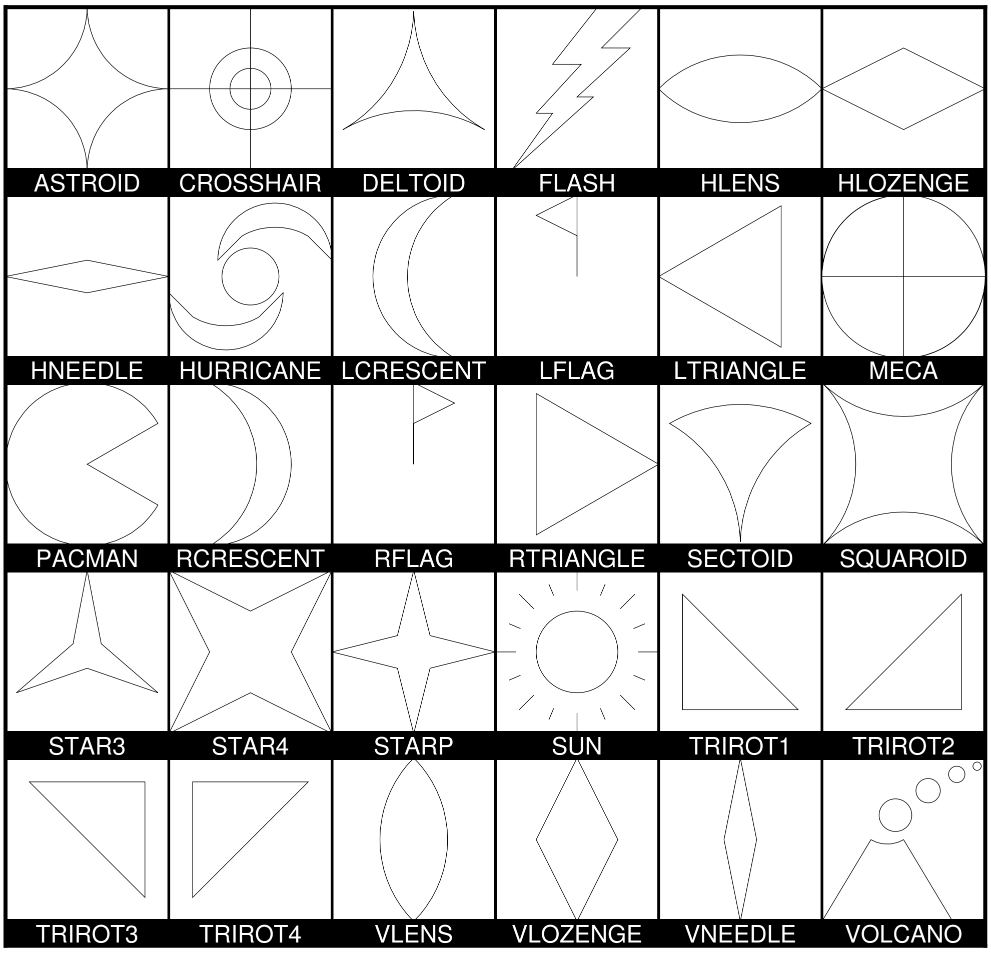


Figure 24.1: Custom plot symbols supported by GMT. Note that we only show the symbol outline and not any fill. These are all single-parameter symbols. Be aware that some symbols may have a hardwired fill or no-fill component, while others duplicate what is already available as standard built-in symbols.

passed directly to the symbol machinery. However, you can use the *types* argument to specify different types of parameters. The available types are

> **a** Geographic angle, to be converted to map angle given the current map projection.
>
> **l** Length, i.e., an additional length scale (in cm, inch, or point as per *PROJ_LENGTH_UNIT*) in addition to the given symbol size.
>
> **o** Other, i.e., a numerical quantity to be passed to the custom symbol as is.
>
> **s** String, i.e., a single column of text to be placed by the **l** command. Use octal \040 to include spaces while still remaining a single word.

To use the extra parameters in your macro you address them as $1, $2, etc.

24.2.3 Macro commands

The custom symbol language contains commands to rotate the relative coordinate system, draw free-form polygons and lines, change the current fill and/or pen, place text, and include basic geometric symbols as part of the overall design (e.g., circles, triangles, etc.). The available commands are listed in Table *custsymb*.

Name	Code	Purpose	Arguments
rotate	**R**	Rotate the coordinate system	α[**a**]
moveto	**M**	Set a new anchor point	x_0, y_0
drawto	**D**	Draw line from previous point	x, y
arc	**A**	Append circular arc to existing path	$x_c, y_c, r, \alpha_1, \alpha_2$
stroke	**S**	Stroke existing path only	
texture	**T**	Change current pen and fill	
star	**a**	Plot a star	x, y,*size*
circle	**c**	Plot a circle	x, y,*size*
diamond	**d**	Plot a diamond	x, y,*size*
ellipse	**e**	Plot a ellipse	x, y, α,*major*,*minor*
octagon	**g**	Plot an octagon	x, y,*size*
hexagon	**h**	Plot a hexagon	x, y,*size*
invtriangle	**i**	Plot an inverted triangle	x, y,*size*
letter	**l**	Plot a letter	x, y,*size*, *string*
marc	**m**	Plot a math arc	$x, y, r, \alpha_1, \alpha_2$
pentagon	**n**	Plot a pentagon	x, y,*size*
plus	**+**	Plot a plus sign	x, y,*size*
rect	**r**	Plot a rectangle	x, y, *width*, *height*
square	**s**	Plot a square	x, y,*size*
triangle	**t**	Plot a triangle	x, y,*size*
wedge	**w**	Plot a wedge	$x, y, r, \alpha_1, \alpha_2$
cross	**x**	Plot a cross	x, y,*size*
x-dash	**-**	Plot a x-dash	x, y,*size*
y-dash	**y**	Plot a y-dash	x, y,*size*

Note for **R**: if an **a** is appended then α is considered to be a map azimuth; otherwise it is a Cartesian angle. For **M**, **T**, and all the lower-case symbol codes you may optionally append specific pens (with **-W***pen*) and fills (with **-G***pen*). These settings will override the pens and fills you may have specified on the command line. Passing **-G**- or **-W**- means no fill or outline, respectively.

24.2.4 Text substitution

Normally, the **l** macro code will place a hard-wired text string. However, you can also obtain the entire string from your input file via a single symbol variable that must be declared with type **s** (string). The string read from your input file must be a single word, so if you need spaces you must use the octal \040 code. Similarly, to place the dollar sign $ you must use octal \044 so as to not confuse the parser with a symbol variable. The string itself, if obtained from the symbol definition file, may contain special codes that will be expanded given the current record. You can embed %X or %Y to add the current longitude (or x) and latitude (or y) in your label string. You may also use $n to embed a numerical symbol variable as text. It will be formatted according to *FORMAT_FLOAT_MAP*, unless you append the modifiers **+X** (longitude via *FORMAT_GEO_MAP*), **+Y** (latitude via *FORMAT_GEO_MAP*), or **+T** (calendar time via *FORMAT_DATE_MAP* and *FORMAT_CLOCK_MAP*.

24.2.5 Text alignment and font

Like the **Sl** symbol in `psxy`, you can change the current font by appending to **l** the modifier **+f***font* and the text justification by appending the modifier **+j***justify*.

24.2.6 Conditional statements

There are two types of conditional statements in the macro language: A simple condition preceding a single command, or a more elaborate if-then-elseif-else construct. In any test you may use one (and only one) of many logical operators, as listed in Table *custop*.

Operator	Purpose
<	Is *var* less than *constant*?
<=	Is *var* less than or equal to *constant*?
==	Is *var* equal to *constant*?
!=	Is *var* not equal to *constant*?
>=	Is *var* greater than or equal to *constant*?
>	Is *var* greater than *constant*?
%	Does *var* have a remainder with *constant*?
!%	Is *var* an exact multiple of *constant*?
<>	Is *var* within the exclusive range of *constant*?
[]	Is *var* within the inclusive range of *constant*?
<]	Is *var* within the in/ex-clusive range of *constant*?
[>	Is *var* within the ex/in-clusive range of *constant*?

Simple conditional test

The simple if-test uses a one-line format, defined as

if *var OP constant* **then** *command*

where *var* must be one of the symbol parameters, specified as $1, $2, $3, etc. You must document what these additional parameters control. For example, to plot a small cyan circle at (0.2, 0.3) with diameter 0.4 only if $2 exceeds 45 you would write

```
if $2 > 45 then 0.2 0.3 0.4 c -Gcyan
```

Note that this form of the conditional test has no mechanism for an **else** branch, but this can be accomplished by repeating the test but reversing the logic for the second copy, e.g.,

```
if $1 > 10 then 0 0 0.5 c -Gred
if $1 <= 10 then 0 0 0.5 c -Gblue
```

or you may instead consider the complete conditional construct below.

Complete conditional test

The complete conditional test uses a multi-line format, such as

if *var OP constant* **then** {
    <one or more lines with commands>
} **elseif** *var OP constant* **then** {
    <one or more lines with commands>
} **else** {
    <one or more lines with commands>
}

The **elseif** (one or more) and **else** branches are optional. Note that the syntax is strictly enforced, meaning the opening brace must appear after **then** with nothing following it, and the closing brace must appear by itself with no other text, and that the **elseif** and **else** statements must have both closing and opening braces on the same line (and nothing else). You may nest tests as well (up to 10 levels deep), e.g.,

```
if $1 > 45 then {
        if $2 [> 0:10 then 0 0 0.5 c -Gred
} elseif $1 < 15 then {
        if $2 [> 0:10 then 0 0 0.5 c -Ggreen
} else {
        if $2 [> 10:20 then {
                0 0 M -W1p,blue
                0.3 0.3 D
                S
                0.3 0.3 0.3 c -Gcyan
        }
}
```

CHAPTER 25

N. Annotation of Contours and "Quoted Lines"

The GMT programs `grdcontour` (for data given as 2-dimensional grids) and `pscontour` (for *x,y,z* tables) allow for contouring of data sets, while `psxy` and `psxyz` can plot lines based on *x,y*- and *x,y,z*-tables, respectively. In both cases it may be necessary to attach labels to these lines. Clever or optimal placements of labels is a very difficult topic, and GMT provides several algorithms for this placement as well as complete freedom in specifying the attributes of the labels. Because of the richness of these choices we present this Appendix which summarizes the various options and gives several examples of their use.

25.1 Label Placement

While the previous GMT versions 1–3 allowed for a single algorithm that determined where labels would be placed, GMT 4 allows for five different algorithms. Furthermore, a new "symbol" option (**-Sq** for "quoted line") has been added to `psxy` and `psxyz` and hence the new label placement mechanisms apply to those programs as well. The contouring programs expect the algorithm to be specified as arguments to **-G** while the line plotting programs expect the same arguments to follow **-Sq**. The information appended to these options is the same in both cases and is of the form [**code**]*info*. The five algorithms correspond to the five codes below (some codes will appear in both upper and lower case; they share the same algorithm but differ in some other ways). In what follows, the phrase "line segment" is taken to mean either a contour or a line to be labeled. The codes are:

d: Full syntax is **d***dist*[**c|i|p**][/*frac*]. Place labels according to the distance measured along the projected line on the map. Append the unit you want to measure distances in [Default is taken from *PROJ_LENGTH_UNIT*]. Starting at the beginning of a line, place labels every *dist* increment of distance along the line. To ensure that closed lines whose total length is less than *dist* get annotated, we may append *frac* which will place the first label at the distance $d = dist \times frac$ from the start of a closed line (and every *dist* thereafter). If not given, *frac* defaults to 0.25.

D: Full syntax is **D***dist*[**d|m|s|e|f|k|M|n**][/*frac*]. This option is similar to **d** except the original data must be referred to geographic coordinates (and a map projection must have been chosen) and actual Earth [1] surface distances along the lines are considered. Append the unit you want to measure distances in; choose among arc **d**egree, **m**inute, and **s**econd, or m**e**ter [Default], **f**eet, **k**ilometer, statute **M**iles, or **n**autical miles. Other aspects are similar to code **d**.

f: Full syntax is **f***fix.txt*[/*slop*[**c|i|p**]]. Here, an ASCII file *fix.txt* is given which must contain records whose first two columns hold the coordinates of points along the lines at which locations the labels should be placed. Labels will only be placed if the coordinates match the line coordinates

[1] or whatever planet we are dealing with.

to within a distance of *slop* (append unit or we use *PROJ_LENGTH_UNIT*). The default *slop* is zero, meaning only exact coordinate matches will do.

l: Full syntax is **l***line1*[,*line2*[, ...]]. One or more straight line segments are specified separated by commas, and labels will be placed at the intersections between these lines and our line segments. Each *line* specification implies a *start* and *stop* point, each corresponding to a coordinate pair. These pairs can be regular coordinate pairs (i.e., longitude/latitude separated by a slash), or they can be two-character codes that refer to predetermined points relative to the map region. These codes are taken from the `pstext` justification keys [**L**|**C**|**R**][**B**|**M**|**T**] so that the first character determines the *x*-coordinate and the second determines the *y*-coordinate. In `grdcontour`, you can also use the two codes **Z+** and **Z-** as shorthands for the location of the grid's global maximum and minimum, respectively. For example, the *line* **LT/RB** is a diagonal from the upper left to the lower right map corner, while **Z-**/135W/15S is a line from the grid minimum to the point (135W, 15S).

L: Same as **l** except we will treat the lines given as great circle start/stop coordinates and fill in the points between before looking for intersections.

n: Full syntax is **n***number*[/*minlength*[**c**|**i**|**p**]]. Place *number* of labels along each line regardless of total line length. The line is divided into *number* segments and the labels are placed at the centers of these segments. Optionally, you may give a *minlength* distance to ensure that no labels are placed closer than this distance to its neighbors.

N: Full syntax is **N***number*[/*minlength*[**c**|**i**|**p**]]. Similar to code **n** but here labels are placed at the ends of each segment (for *number* >= 2). A special case arises for *number = 1* when a single label will be placed according to the sign of *number*: -1 places one label justified at the start of the line, while +1 places one label justified at the end of the line.

x: Full syntax is **x***cross.d*. Here, an ASCII file *cross.d* is a multi-segment file whose lines will intersect our segment lines; labels will be placed at these intersections.

X: Same as **x** except we treat the lines given as great circle start/stop coordinates and fill in the points between before looking for intersections.

Only one algorithm can be specified at any given time.

25.2 Label Attributes

Determining where to place labels is half the battle. The other half is to specify exactly what are the attributes of the labels. It turns out that there are quite a few possible attributes that we may want to control, hence understanding how to specify these attributes becomes important. In the contouring programs, one or more attributes may be appended to the **-A** option using the format +*code*[*args*] for each attribute, whereas for the line plotting programs these attributes are appended to the **-Sq** option following a colon (:) that separates the label codes from the placement algorithm. Several of the attributes do not apply to contours so we start off with listing those that apply universally. These codes are:

+a: Controls the angle of the label relative to the angle of the line. Append **n** for normal to the line, give a fixed *angle* measured counter-clockwise relative to the horizontal. or append **p** for parallel to the line [Default]. If using `grdcontour` the latter option you may further append **u** or **d** to get annotations whose upper edge always face the next higher or lower contour line.

+c: Surrounding each label is an imaginary label "textbox" which defines a region in which no segment lines should be visible. The initial box provides an exact fit to the enclosed text but clearance may be extended in both the horizontal and vertical directions (relative to the label baseline) by the given amounts. If these should be different amounts please separate them by a slash; otherwise the single value applies to both directions. Append the distance units of your choice (**c|i|m|p**), or give % to indicate that the clearance should be this fixed percentage of the label font size in use. The default is 15%.

+d: Debug mode. This is useful when testing contour placement as it will draw the normally invisible helper lines and points in the label placement algorithms above.

+d: Delayed mode, to delay the plotting of the text as text clipping is set instead.

+f: Specifies the desired label font, including size or color. See `pstext` for font names or numbers. The default font is given by *FONT_ANNOT_PRIMARY*.

+g: Selects opaque rather than the default transparent text boxes. You may optionally append the color you want to fill the label boxes; the default is the same as *PS_PAGE_COLOR*.

+j: Selects the justification of the label relative to the placement points determined above. Normally this is center/mid justified (**CM** in `pstext` justification parlance) and this is indeed the default setting. Override by using this option and append another justification key code from [**L|C|R**][**B|M|T**]. Note for curved text (**+v**) only vertical justification will be affected.

+o: Request a rounded, rectangular label box shape; the default is rectangular. This is only manifested if the box is filled or outlined, neither of which is implied by this option alone (see **+g** and **+p**). As this option only applies to straight text, it is ignored if **+v** is given.

+p: Selects the drawing of the label box outline; append your preferred *pen* unless you want the default GMT pen [0.25p,black].

+r: Do not place labels at points along the line whose local radius of curvature falls below the given threshold value. Append the radius unit of your choice (**c|i|p**) [Default is 0].

+u: Append the chosen *unit* to the label. Normally a space will separate the label and the unit. If you want to close this gap, append a *unit* that begins with a hyphen (-). If you are contouring with `grdcontour` and you specify this option without appending a unit, the unit will be taken from the *z*-unit attribute of the grid header.

+v: Place curved labels that follow the wiggles of the line segments. This is especially useful if the labels are long relative to the length-scale of the wiggles. The default places labels on an invisible straight line at the angle determined.

+w: The angle of the line at the point of straight label placement is calculated by a least-squares fit to the *width* closest points. If not specified, *width* defaults to 10.

+=: Similar in most regards to **+u** but applies instead to a label *prefix* which you must append.

For contours, the label will be the value of the contour (possibly modified by **+u** or **+=**). However, for quoted lines other options apply:

+l: Append a fixed *label* that will be placed at all label locations. If the label contains spaces you must place it inside matching quotes.

+L: Append a code *flag* that will determine the label. Available codes are:

+Lh: Take the label from the current multi-segment header (hence it is assumed that the input line segments are given in the multi-segment file format; if not we pick the single label from the file's header record). We first scan the header for an embedded **-L***label* option; if none is found we instead use the first word following the segment marker [>].

+Ld: Take the Cartesian plot distances along the line as the label; append **c|i|p** as the unit [Default is *PROJ_LENGTH_UNIT*]. The label will be formatted according to the *FORMAT_FLOAT_OUT* string, *unless* label placement was determined from map distances along the segment lines, in which case we determine the appropriate format from the distance value itself.

+LD: Calculate actual Earth surface distances and use the distance at the label placement point as the label; append **d|e|f|k|m|M|n|s** to specify the unit [If not given we default to **d**egrees, *unless* label placement was determined from map distances along the segment lines, in which case we use the same unit specified for that algorithm]. Requires a map projection to be used.

+Lf: Use all text after the 2nd column in the fixed label location file *fix.txt* as labels. This choice obviously requires the fixed label location algorithm (code **f**) to be in effect.

+Ln: Use the running number of the current multi-segment as label.

+LN: Use a slash-separated combination of the current file number and the current multi-segment number as label.

+Lx: As **h** but use the multi-segment headers in the *cross.d* file instead. This choice obviously requires the crossing segments location algorithm (code **x|X**) to be in effect.

25.3 Examples of Contour Label Placement

We will demonstrate the use of these options with a few simple examples. First, we will contour a subset of the global geoid data used in GMT Example 01; the region selected encompasses the world's strongest "geoid dipole": the Indian Low and the New Guinea High.

25.3.1 Equidistant labels

Our first example uses the default placement algorithm. Because of the size of the map we request contour labels every 1.5 inches along the lines:

As seen in Figure *Contour label 1*, the contours are placed rather arbitrary. The string of contours for -40 to 60 align well but that is a fortuitous consequence of reaching the 1.5 inch distance from the start at the bottom of the map.

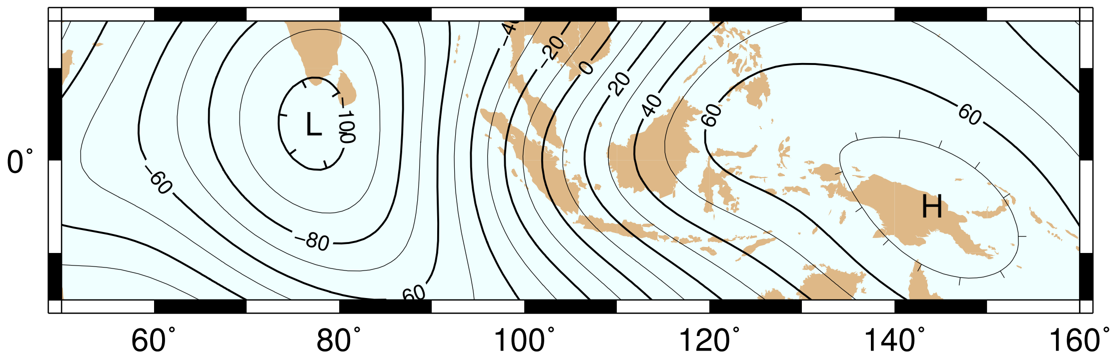


Figure 25.1: Equidistant contour label placement with **-Gd**, the only algorithm available in previous GMT versions.

25.3.2 Fixed number of labels

We now exercise the option for specifying exactly how many labels each contour line should have:

By selecting only one label per contour and requiring that labels only be placed on contour lines whose length exceed 1 inch, we achieve the effect shown in Figure *Contour label 2*.

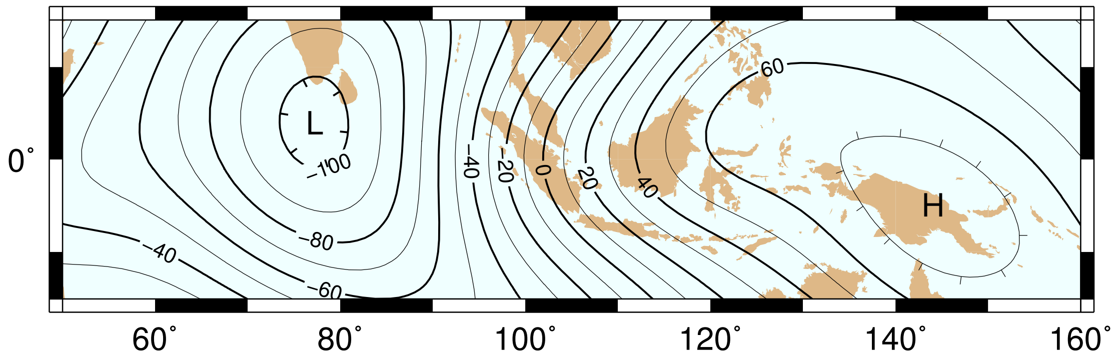


Figure 25.2: Placing one label per contour that exceed 1 inch in length, centered on the segment with **-Gn**.

25.3.3 Prescribed label placements

Here, we specify four points where we would like contour labels to be placed. Our points are not exactly on the contour lines so we give a nonzero "slop" to be used in the distance calculations: The point on the contour closest to our fixed points and within the given maximum distance will host the label.

The angle of the label is evaluated from the contour line geometry, and the final result is shown in Figure *Contour label 3*. To aid in understanding the algorithm we chose to specify "debug" mode (**+d**) which placed a small circle at each of the fixed points.

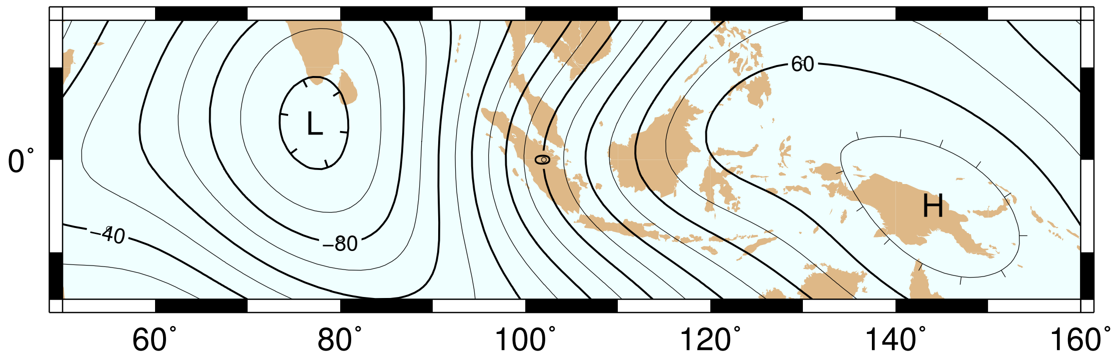


Figure 25.3: Four labels are positioned on the points along the contours that are closest to the locations given in the file fix.txt in the **-Gf** option.

25.3.4 Label placement at simple line intersections

Often, it will suffice to place contours at the imaginary intersections between the contour lines and a well-placed straight line segment. The **-Gl** or **-GL** algorithms work well in those cases:

The obvious choice in this example is to specify a great circle between the high and the low, thus placing all labels between these extrema.

The thin debug line in Figure *Contour label 4* shows the great circle and the intersections where labels are plotted. Note that any number of such lines could be specified; here we are content with just one.

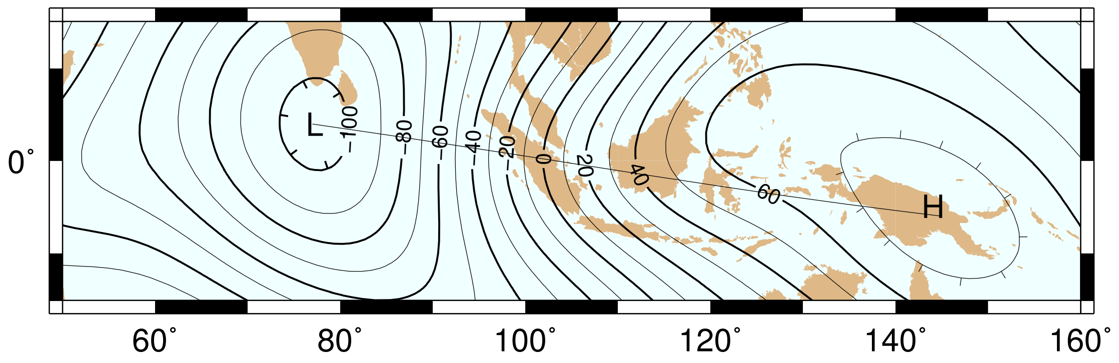


Figure 25.4: Labels are placed at the intersections between contours and the great circle specified in the **-GL** option.

25.3.5 Label placement at general line intersections

If (1) the number of intersecting straight line segments needed to pick the desired label positions becomes too large to be given conveniently on the command line, or (2) we have another data set or lines whose intersections we wish to use, the general crossing algorithm makes more sense:

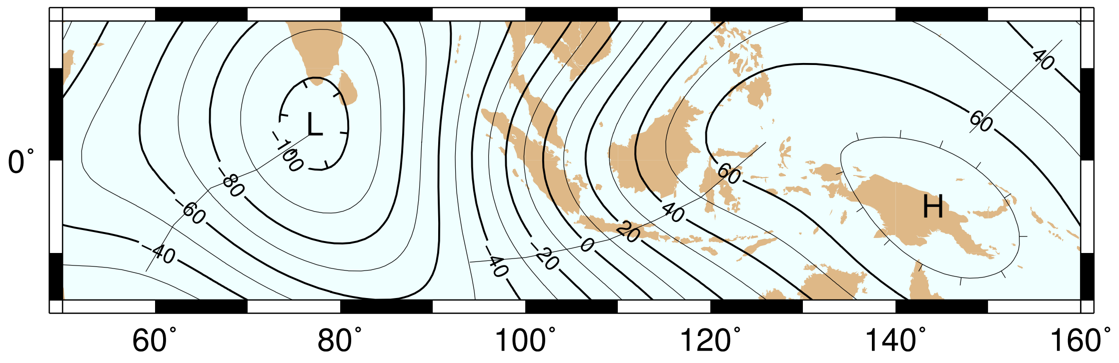


Figure 25.5: Labels are placed at the intersections between contours and the multi-segment lines specified in the **-GX** option.

In this case, we have created three strands of lines whose intersections with the contours define the label placements, presented in Figure *Contour label 5*.

25.4 Examples of Label Attributes

We will now demonstrate some of the ways to play with the label attributes. To do so we will use `psxy` on a great-circle line connecting the geoid extrema, along which we have sampled the ETOPO5 relief data set. The file thus contains *lon, lat, dist, geoid, relief*, with distances in km.

25.4.1 Label placement by along-track distances, 1

This example will change the orientation of labels from along-track to across-track, and surrounds the labels with an opaque, outlined text box so that the label is more readable. We choose the place the labels every 1000 km along the line and use that distance as the label. The labels are placed normal to the line:

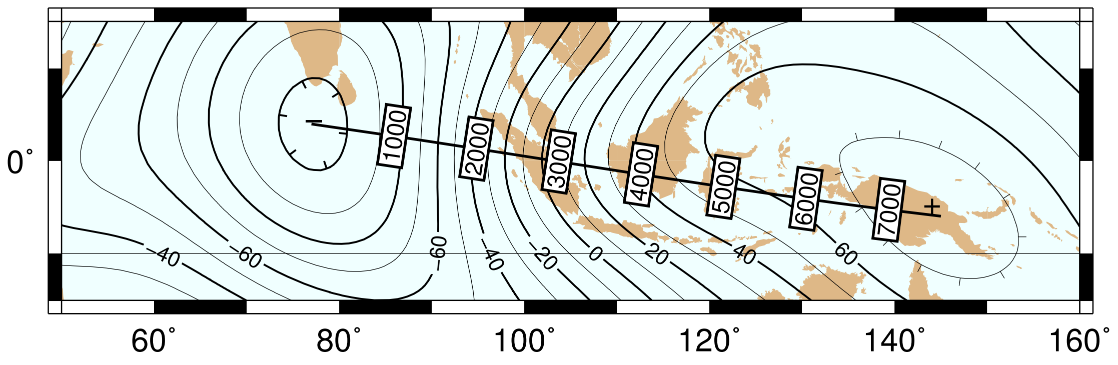


Figure 25.6: Labels attributes are controlled with the arguments to the **-Sq** option.

The composite illustration in Figure *Contour label 6* shows the new effects. Note that the line connecting the extrema does not end exactly at the '-' and '+' symbols. This is because the placements of those symbols are based on the mean coordinates of the contour and not the locations of the (local or global) extrema.

25.4.2 Label placement by along-track distances, 2

A small variation on this theme is to place the labels parallel to the line, use spherical degrees for placement, append the degree symbol as a unit for the labels, choose a rounded rectangular text box, and inverse-video the label:

The output is presented as Figure *Contour label 7*.

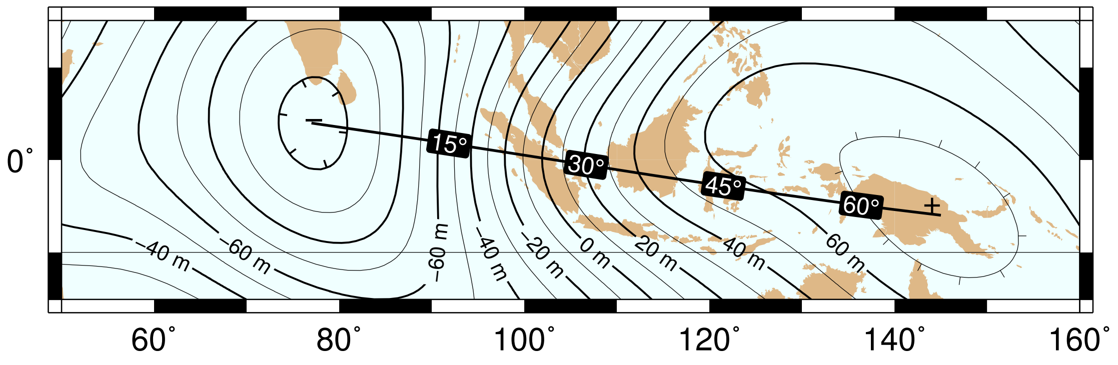


Figure 25.7: Another label attribute example.

25.4.3 Using a different data set for labels

In the next example we will use the bathymetry values along the transect as our label, with placement determined by the distance along track. We choose to place labels every 1500 km. To do this we need to pull out those records whose distances are multiples of 1500 km and create a "fixed points" file that can be used to place labels and specify the labels. This is done with **awk**.

The output is presented as Figure *Contour label 8*.

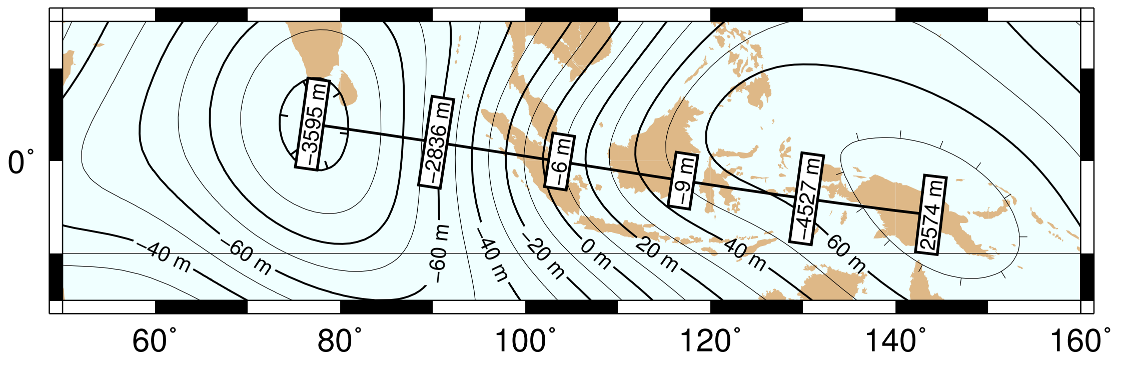


Figure 25.8: Labels based on another data set (here bathymetry) while the placement is based on distances.

25.5 Putting it all together

Finally, we will make a more complex composite illustration that uses several of the label placement and label attribute settings discussed in the previous sections. We make a map showing the tsunami travel times (in hours) from a hypothetical catastrophic landslide in the Canary Islands [2]. We lay down a color map based on the travel times and the shape of the seafloor, and travel time contours with curved labels as well as a few quoted lines. The final script is

with the complete illustration presented as Figure *Contour label 9*.

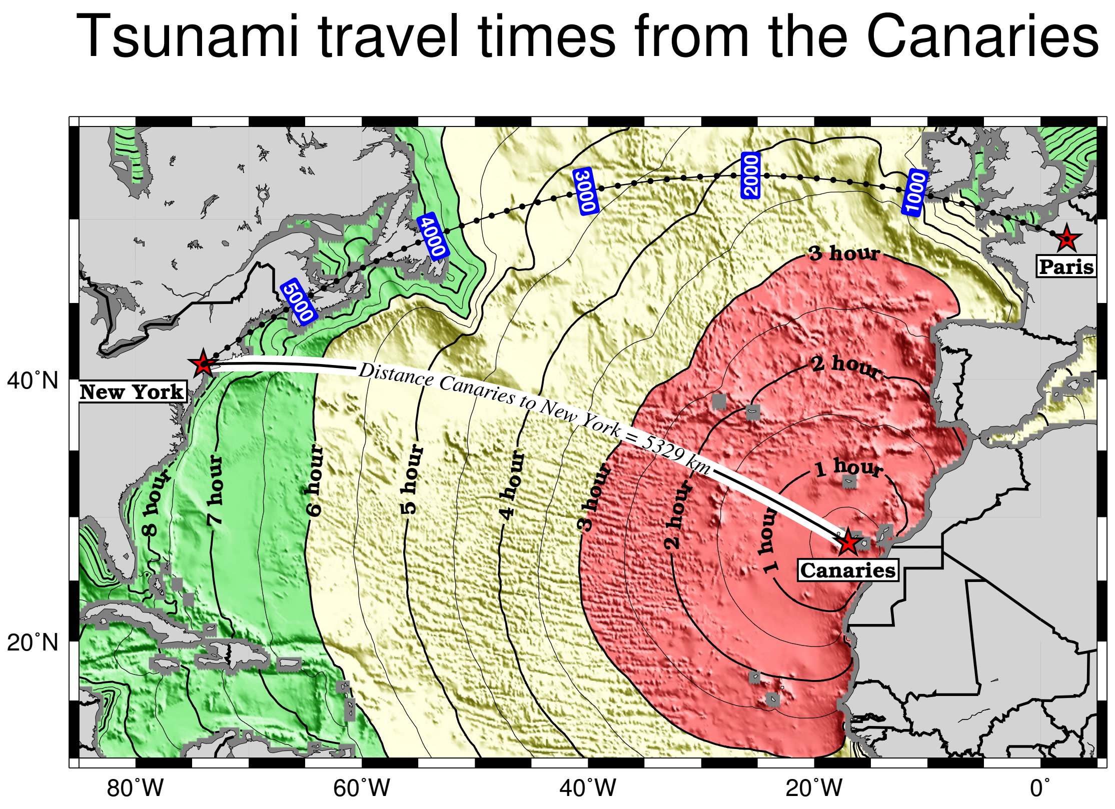


Figure 25.9: Tsunami travel times from the Canary Islands to places in the Atlantic, in particular New York. Should a catastrophic landslide occur it is possible that New York will experience a large tsunami about 8 hours after the event.

[2] Travel times were calculated using Geoware's travel time calculator, **ttt**; see http://www.geoware-online.com/.

CHAPTER 26

O. Special Operations

26.1 Running GMT in *isolation mode*

In Chapter General features it is described how GMT creates several (temporary) files to communicate between the different commands that make up the script that finally creates a plot. Among those files are:

gmt.conf This file covers about 100 different settings that influence the layout of your plot, from font sizes to tick lengths and date formats (See Section GMT defaults). Those settings can be altered by editing the file, or by running the `gmtset` command. A problem may arise when those settings are changed half-way through the script: the next time you run the script it will start with the modified settings and hence might alter your scripts results. It is therefore often necessary to revert to the original `gmt.conf` file. *Isolation mode* avoids that issue.

gmt.history This file is created to communicate the command line history from one command to the next (Section Command line history) so that shorthands like **-R** or **-J** can be used once it has been set in a previous GMT command. The existence of this file makes if impossible to run two GMT scripts simultaneously in the same directory, since those `gmt.history` files may clash (contain different histories) and adversely affect the results of both scripts.

A cure to all these woes is the *isolation mode* introduced in GMT version 4.2.2. This mode allows you to run a GMT script without leaving any traces other than the resulting PostScript or data files, and not altering the `gmt.conf` or `gmt.history` files. Those files will be placed in a temporary directory instead. And if properly set up, this temporary directory will only be used by a single script, even if another GMT script is running simultaneously. This also provides the opportunity to create any other temporary files that the script might create in the same directory.

The example below shows how *isolation mode* works.

The files `gmt.conf` and `gmt.history` are automatically created in the temporary directory `$GMT_TMPDIR`. The script is also adjusted such that the temporary grid file `lat.nc` and colormap `lat.cpt` are created in that directory as well. To make things even more easy, GMT now provides a set of handy shell functions in `gmt_shell_functions.sh`: simply include that file in the script and the creation and the removal of the temporary directory is reduced to a single command.

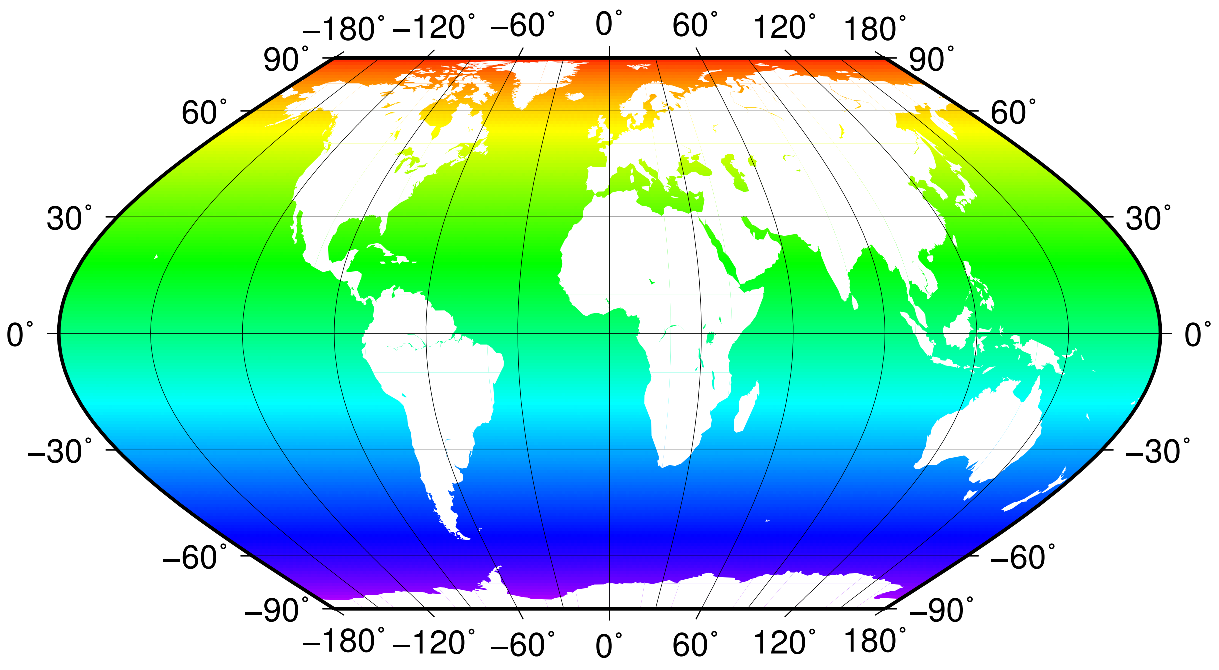


Figure 26.1: Example created in isolation mode

P. The GMT Vector Data Format for OGR Compatibility

27.1 Background

The National Institute for Water and Atmospheric Research (NIWA) in New Zealand has funded the implementation of a GMT driver (read and write) for the OGR package. OGR is an Open Source toolkit for accessing or reformatting vector (spatial) data stored in a variety of formats and is part of the. The intention was to enable the easy rendering (using GMT) of spatial data held in non-GMT formats, and the export of vector data (e.g., contours) created by GMT for use with other GIS and mapping packages. While **ogr2ogr** has had the capability to write this new format since 2009, GMT 4 did not have the capability to use the extra information.

GMT 5 now allows for more advanced vector data, including donut polygons (polygons with holes) and aspatial attribute data. At the same time, the spatial data implementation will not disrupt older GMT 4 programs since all the new information are written via comments.

The identification of spatial feature types in GMT files generally follows the technical description, (which is largely consistent with the OGC SFS specification). This specification provides for non-topological point, line and polygon (area) features, as well as multipoint, multiline and multipolygon features, and was written by Brent Wood based on input from Paul Wessel and others on the GMT list.

27.2 The OGR/GMT format

Several key properties of the OGR/GMT format is summarized below:

- All new data fields are stored as comment lines, i.e., in lines starting with a "#". OGR/GMT files are therefore compatible with GMT 4 binaries, which will simply ignore this new information.
- To be consistent with current practice in GMT, data fields are represented as whitespace-separated strings within the comments, each identified by the "@" character as a prefix, followed by a single character identifying the content of the field. To avoid confusion between words and strings, the word (field) separator within strings will be the "|" (pipe or vertical bar) character.
- Standard UNIX "\" escaping is used, such as \n for newline in a string.
- All new data are stored before the spatial data (coordinates) in the file, so when any GMT 5 program is processing the coordinate data for a feature, it will already have parsed any non-spatial information for each feature, which may impact on how the spatial data is treated (e.g., utilizing the aspatial attribute data for a feature to control symbology).

- The first comment line must specify the version of the OGR/GMT data format, to allow for future changes or enhancements to be supported by future GMT programs. This document describes v1.0.
- For consistency with other GIS formats (such as shapefiles) the OGR/GMT format explicitly contains a field specifying whether the features are points, linestrings or polygons, or the "multi" versions of these. (Other shapefile feature types will not be supported at this stage). At present, GMT programs are informed of this via command line parameters. This will now be explicit in the data file, but does not preclude command line switches setting symbologies for plotting polygons as lines (perimeters) or with fills, as is currently the practice.
- Note that what is currently called a "multiline" (multi-segment) file in GMT parlance is generally a set of "lines" in shapefile/OGR usage. A multiline in this context is a single feature comprising multiple lines. For example, all the transects from a particular survey may be stored as lines, each with it's own attribute set, such as transect number, date/time, etc. They may also be stored as a single multiline feature with one attribute set, such as trip ID. This difference is explicitly stored in the data in OGR/shapefiles, but currently specified only on the command line in GMT. This applies also to points and polygons. The GMT equivalent to {multipoint, multiline, multipolygon} datatypes is multiple GMT files, each comprising a single {multipoint, multiline, multipolygon} feature.
- The new GMT vector data files includes a header comment specifying the type of spatial features it contains, as well as the description of the aspatial attribute data to be associated with each feature. Unlike the shapefile format, which stores the spatial and aspatial attribute data in separate files, the GMT format will store all data in a single file.
- All the features in a GMT file must be of the same type.

27.3 OGR/GMT Metadata

Several pieces of metadata information must be present in the header of the OGR/GMT file, followed by both spatial and aspatial data. In this section we look at the metadata.

27.3.1 Format version

The comment header line will include a version identifier providing for possible different versions in future. It is indicated by the **@V** sequence.

Code	Argument	Description
V	GMT1.0	Data in this file is stored using v1.0 of the OGR/GMT data format

An OGR/GMT file must therefore begin with the line

```
# @VGMT1.0
```

Parsing of the OGR/GMT format is only activated if the version code-sequence has been found.

27.3.2 Geometry types

The words and characters used to specify the geometry type (preceded by the **@G** code sequence on the header comment line), are listed in Table *geometries*.

Code	Geometry	Description
G	POINT	File with point features (Each point will have it's own attribute/header line preceding the point coordinates)
G	MULTIPOINT	File with a single multipoint feature (All the point features are a single multipoint, with the same attribute/header information)
G	LINESTRING	File with features comprising multiple single lines (Effectively the current GMT multiline file, each line feature will have it's own attribute and header data)
G	MULTI-LINESTRING	File with features comprising a multiline (All the line features in the file are a single multiline feature, only one attribute and header which applies to all the lines)
G	POLYGON	File with one or more polygons (Similar to a line file, except the features are closed polygons)
G	MULTIPOLYGON	File with a single multipolygon (Similar to a GMT multiline file, except the feature is a closed multipolygon)

An example GMT polygon file header using this specification (in format 1.0) is

```
# @VGMT1.0 @GPOLYGON
```

27.3.3 Domain and map projections

The new format will also support region and projection information. The region will be stored in GMT **-R** format (i.e., **-R***W/E/S/N*, where the *W/E/S/N* values represent the extent of features); the **@R** code sequence marks the domain information. A sample region header is:

```
# @R150/190/-45/-54
```

Projection information will be represented as four optional strings, prefixed by **@J** (J being the GMT character for projection information. The **@J** code will be followed by a character identifying the format, as shown in Table *projectspec*.

Code	Projection Specification
@Je	EPSG code for the projection
@Jg	A string representing the projection parameters as used by GMT
@Jp	A string comprising the Proj.4 parameters representing the projection parameters
@Jw	A string comprising the OGR WKT (well known text) representation of the projection parameters

Sample projection strings are:

```
# @Je4326 @JgX @Jp"+proj=longlat +ellps=WGS84+datum=WGS84 +no_defs"
# @Jw"GEOGCS[\"WGS84\",DATUM[\"WGS_1984\",SPHEROID\"WGS84\",6378137,\
298.257223563,AUTHORITY[\"EPSG\",\"7030\"]],TOWGS84[0,0,0,0,0,0,0],
AUTHORITY[\"EPSG\",\"6326\"]],PRIMEM[\"Greenwich\",0,\
AUTHORITY[\"EPSG\",\"8901\"]],UNIT[\"degree\",0.01745329251994328,\
AUTHORITY[\"EPSG\",\"9122\"]],AUTHORITY[\"EPSG\",\"4326\"]]"
```

Note that an OGR-generated file will not have a **@Jg** string, as OGR does not have any knowledge of the GMT projection specification format. GMT supports at least one of the other formats to provide interoperability with other Open Source related GIS software packages. One relatively simple approach, (with some limitations), would be a lookup table matching EPSG codes to GMT strings.

27.3.4 Declaration of aspatial fields

The string describing the aspatial field names associated with the features is flagged by the **@N** prefix.

Code	Argument	Description
N	word|word|word	A "|" -separated string of names of the attribute field names

Any name containing a space must be quoted. The **@N** selection must be combined with a matching string specifying the data type for each of the named fields, using the **@T** prefix.

Code	Argument	Description
T	word|word|word	A "|" -separated string of the attribute field data types

Available datatypes should largely follow the shapefile (DB3) specification, including **string**, **integer**, **double**, **datetime**, and **logical** (boolean). In OGR/GMT vector files, they will be stored as appropriately formatted text strings.

An example header record containing all these is

```
# @VGMT1.0 @GPOLYGON @Nname|depth|id @Tstring|double|integer
```

27.4 OGR/GMT Data

All generic fields must be at the start of the file before any feature-specific content (feature attribute data follow the metadata, as do the feature coordinates, separated by a comment line comprising "# FEATURE_DATA". Provided each string is formatted as specified, and occurs on a line prefixed with "#" (i.e., is a comment), the format is free form, in that as many comment lines as desired may be used, with one or more parameter strings in any order in any line. E.g., one parameter per line, or all parameters on one line.

27.4.1 Embedding aspatial data

Following this header line is the data itself, both aspatial and spatial. For line and polygon (including multiline and multipolygon) data, features are separated using a predefined character, by default ">". For point (and multipoint) data, no such separator is required. The comment line containing the aspatial data for each feature will immediately precede the coordinate line(s). Thus in the case of lines and polygons, it will immediately follow the ">" line. The data line will be a comment flag ("#") followed by **@D**, followed by a string of "|"-separated words comprising the data fields defined in the header record.

To allow for names and values containing spaces, such string items among the **@N** or **@D** specifiers must be enclosed in double quotes. (Where double quotes or pipe characters are included in the string, they must be escaped using "\"). Where any data values are null, they will be represented as no characters between the field separator, (e.g., #@D|||). A Sample header and corresponding data line for points are

```
# @VGMT1.0 @GPOINT @Nname|depth|id @Tstring|double|integer
# @D"Point 1"|-34.5|1
```

while for a polygon it may look like

```
# @VGMT1.0 @GPOLYGON @Nname|depth|id @Tstring|double|integer
>
# @D"Area 1"|-34.5|1
```

27.4.2 Polygon topologies

New to GMT is the concept of polygon holes. Most other formats do support this structure, so that a polygon is specified as a sequence of point defining the perimeter, optionally followed by similar coordinate sequences defining any holes (the "donut" polygon concept).

To implement this in a way which is compatible with previous GMT versions, each polygon feature must be able to be identified as the outer perimeter, or an inner ring (hole). This is done using a **@P** or **@H** on the data comment preceding the polygon coordinates. The **@P** specifies a new feature boundary (perimeter), any following **@H** polygons are holes, and must be within the preceding **@P** polygon (as described in the shapefile specification). Any **@H** polygons will NOT have any **@D** values, as the aspatial attribute data pertain to the entire feature, the **@H** polygons are not new polygons, but are merely a continuation of the definition of the same feature.

27.5 Examples

Sample point, line and polygon files are (the new data structures are in lines starting with "#" in strings prefixed with "@"). Here is a typical point file:

```
# @VGMT1.0 @GPOINT @Nname|depth|id
# @Tstring|double|integer
# @R178.43/178.5/-57.98/-34.5
# @Je4326
# @Jp"+proj=longlat +ellps=WGS84 +datum=WGS84+no_defs"
# FEATURE_DATA
# @D"point 1"|-34.5|1
178.5 -45.7
# @D"Point 2"|-57.98|2
178.43 -46.8
...
```

Next is an example of a line file:

```
# @VGMT1.0 @GLINESTRING @Nname|depth|id
# @Tstring|double|integer
# @R178.1/178.6/-48.7/-45.6
# @Jp"+proj=longlat +ellps=WGS84 +datum=WGS84+no_defs"
# FEATURE_DATA
> -W0.25p
# @D"Line 1"|-50|1
178.5 -45.7
178.6 -48.2
178.4 -48.7
178.1 -45.6
> -W0.25p
# @D"Line 2"|-57.98|$
178.43 -46.8
...
```

Finally we show an example of a polygon file:

```
# @VGMT1.0 @GPOLYGON @N"Polygon name"|substrate|id @Tstring|string|integer
# @R178.1/178.6/-48.7/-45.6
# @Jj@Jp"+proj=longlat +ellps=WGS84 +datum=WGS84+no_defs"
# FEATURE_DATA
> -Gblue -W0.25p
```

```
# @P
# @D"Area 1"|finesand|1
178.1 -45.6
178.1 -48.2
178.5 -48.2
178.5 -45.6
178.1 -45.6
>
# @H
# First hole in the preceding perimeter, so is technically still
# part of the same geometry, despite the preceding > character.
# No attribute data is provided, as this is inherited.
178.2 -45.4
178.2 -46.5
178.4 -46.5
178.4 -45.4
178.2 -45.4
>
# @P
...
```