The Generic Mapping Tools

Version 3.4.6

Technical Reference and Cookbook

by

Pål (Paul) Wessel

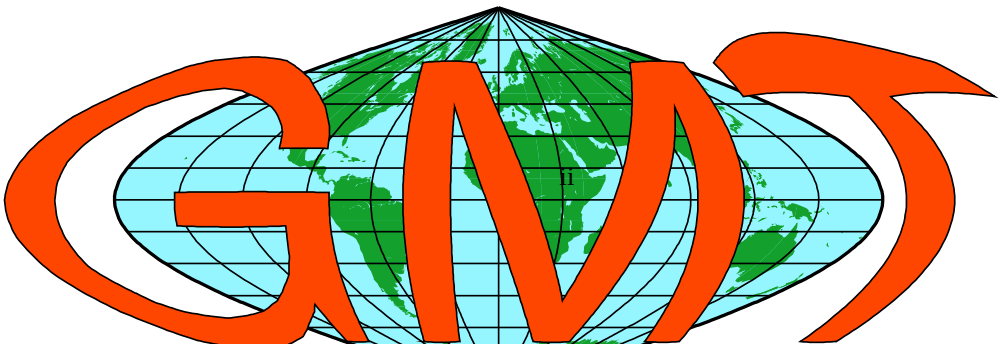
School of Ocean and Earth Science and Technology
University of Hawai'i at Mānoa

and

Walter H. F. Smith

Laboratory for Satellite Altimetry
NOAA/NESDIS/NODC

September 2005



Contents

List of Tables

List of Figures

Acknowledgments

The Generic Mapping Tools (GMT) could not have been designed without the generous support of several people. We gratefully acknowledge A. B. Watts and 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 of our questions over e-mail. The subroutine `gaussj` was written and supplied by Bill Menke, L-DEO. Further development of versions 2.0 and 2.1 at SOEST would not have been possible without the support from the Hawaii Institute of Geophysics and School of Ocean and Earth Science and Technology 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 versions 3.0–3.4.5 owe their existence to grants EAR-93-02272, OCE-95-29431, and OCE-0082552 from the National Science Foundation, which we gratefully acknowledge.

We would like to acknowledge the feedback we have received from many of the users of earlier versions. Many of these suggestions have been implemented, and the bug reports have been useful in providing more robust programs. Specifically, we would like to thank William Weibel, Ameet Raval, Manfred Brands, Angel Li, Andrew Macrae, John Lillibridge, Richard Signell, Michael Barck, Alex Madon, Ben Horner-Johnson, Georg Schwarz, Lloyd Parkes, David Townsend, and many others for advice on how to make GMT portable to DEC, SGI, HP, IBM, Apple, and NEXT workstations. John Lillibridge provided example 11. William Yip helped translate GMT to POSIX ANSI C and incorporate netCDF 3, Allen Cogbill provided OS/2 patches for EMX, and Hanno von Lom helped resolve problems with DLL libraries for Win32. The SOEST RCF staff (Ross Ishida, Pat Townsend, and Sharon Stahl) provided valuable help on Linux, web, and cgi script issues.

Honolulu, HI and Silver Spring, MD, October 2004

Dr. Pål (Paul) Wessel
Professor
Department of Geology and Geophysics
School of Ocean and Earth Science and Technology
University of Hawaii at Manoa

Dr. Walter H. F. Smith
Geophysicist
Laboratory for Satellite Altimetry
National Oceanographic Data Center
National Oceanic and Atmospheric Administration

The GMT Documentation Project

Starting with GMT version 3.2, all GMT documentation was converted from Microsoft ***Word*** to LaTeX files. This step was taken for a number of reasons:

1. Having all the documentation source available in ASCII format makes it easier to access by several GMT developers working on different platforms in different countries.
2. GMT scripts can now be included directly into the text so that the documentation is automatically up-to-date when scripts are modified.
3. All figures are generated on the fly and included as GMT EPS files which thus are always up-to-date.
4. It is easy to convert the LaTeX files to other formats, such as HTML, SGML, *PostScript*, and PDF.
5. The whole task of assembling the pieces, be it generating figures or extracting text portions from the master archive under CVS control, is automated by a simple cshell script.
6. Only free software are used to maintain the GMT Documentation.

Because this transition was undertaken by a complete LaTeX novice it is likely that the layout will change with time. It is also likely that errors have crept into the document. Please send email to the GMT help list if you find any.

A Reminder

If you feel it is appropriate, you may consider paying us back by citing our *EOS* articles on GMT (and perhaps also our Geophysics article on the GMT program **surface**) when you publish papers containing results or illustrations obtained using GMT. The EOS articles on GMT are

- Wessel, P., and W. H. F. Smith, New, improved version of Generic Mapping Tools released, *EOS Trans. Amer. Geophys. U.*, vol. 79 (47), pp. 579, 1998.
- Wessel, P., and W. H. F. Smith, New version of the Generic Mapping Tools released, *EOS Trans. Amer. Geophys. U.*, vol. 76 (33), pp. 329, 1995.
- Wessel, P., and W. H. F. Smith, New version of the Generic Mapping Tools released, *EOS Trans. Amer. Geophys. U. electronic supplement*, http://www.agu.org/eos_elec/95154e.html, 1995.
- Wessel, P., and W. H. F. Smith, Free software helps map and display data, *EOS Trans. Amer. Geophys. U.*, vol. 72 (41), pp. 441, 445-446, 1991.

The article in *Geophysics* on surface is

- Smith, W. H. F., and P. Wessel, Gridding with continuous curvature splines in tension, *Geophysics*, vol. 55 (3), pp. 293-305, 1990.

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 (to jrs@cs.cmu.edu) 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.

Copyright and Caveat Emptor!

[1] http://www.gnu.org/copyleft/gpl.html

Typographic conventions

In reading this documentation, the following provides a summary of the typographic conventions used in this document.

1. User input and GMT or *UNIX* commands are indicated by using the `typewriter` type style, e.g., `chmod +x job03.csh`.
2. The names of GMT programs are indicated by the **bold, sans serif** type style, e.g., we plot text with **pstext**.
3. The names of other programs are indicated by the ***bold, slanted*** type style, e.g., ***grep***.
4. File names are indicated by the underline type style, e.g., gmt.h.

1. 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 Earth and Ocean scientists. Development has at times been rapid, and numerous releases have seen the light of day since the early versions. For a history of the changes from release to release, see the online Release Announcements and the file CHANGES in the main GMT directory.

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 were driven by 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.

[1] when version 1.0 was released at Lamont-Doherty Earth Observatory

2. Introduction

Most scientists are familiar with the sequence: *raw data → processing → 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; 1995a,b; 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, 25 map 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 25 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 → processing → final illustration* flow is broken 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

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

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

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; *PostScript* files 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 avaliable for web browsing and may be installed at the users 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 GMT plot 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 "grdfile" 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 gridded files. One such program, **surface**, includes new modifications to the gridding algorithm developed by *Smith and Wessel* [1990] using continuous splines in tension.

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. 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. 2-D binary (netCDF or user-defined) "grdfiles" – Programs that grid ASCII (x,y,z) data or operate on existing grdfiles produce this type of output.

3. *PostScript* – The plotting programs all use the *PostScript* page 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.

4. 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, read the relevant information on the GMT home page gmt.soest.hawaii.edu, or email listproc@hawaii.edu a note containing the single message

[3] One public-domain RIP is ***ghostscript***, available from www.gnu.org.
[4] Programs now also allow for fast, binary multicolumn file i/o.
[5] While the netCDF format is the default, other formats are also possible, including user-defined formats.

information gmt-group

The listserver will mail you back a shell-script that you may run to obtain all necessary programs, libraries, and support data. After you obtain the GMT archive, you will find that it contains information on how to install GMT on your hardware platform and how to obtain additional files that you may need or want. The archive also contains a license agreement and registration file. We also maintain two electronic mailing lists you may subscribe to in order to stay informed about bug fixes and upgrades (See Chapter 7).

For those without net-access that need to obtain GMT: Geoware (http://www.geoware-online.com) makes and distributes CD-Rs with the GMT package, compatible supplements, and several Gb of useful 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.

References

1. Kernighan, B. W., and D. M. Ritchie, *The C programming language*, 2nd edition, p. 272, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

2. Adobe Systems Inc., *PostScript Language Reference Manual*, 2nd edition, p. 764, Addison-Wesley, Reading, Massachusetts, 1990.

3. Lewine, D., POSIX programmer's guide, 1st edition, p. 607, O'Reilly & Associates, Sebastopol, California, 1991.

4. Treinish, L. A., and M. L. Gough, A software package for the data-independent management of multidimensional data, *EOS trans. AGU, 68,* 633–635, 1987.

5. Smith, W. H. F., and P. Wessel, Gridding with continuous curvature splines in tension, *Geophysics, 55,* 293–305, 1990.

6. Wessel, P., and W. H. F. Smith, New, improved version of Generic Mapping Tools released, *EOS trans. AGU,* 79, 579, 1998.

7. Wessel, P., and W. H. F. Smith, New version of the Generic Mapping Tools released, *EOS trans. AGU,* 76, 329, 1995a.

8. Wessel, P., and W. H. F. Smith, New version of the Generic Mapping Tools released, *EOS electronic supplement,* http://www.agu.org/eos_elec/95154e.html, 1995b.

9. Wessel, P., and W. H. F. Smith, Free software helps map and display data, *EOS trans. AGU,* 72, 441 & 445–446, 1991.

3. GMT overview and quick reference

3.1 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 3.2.

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
gmtconvert	Convert data tables from one format to another
gmtdefaults	List the current default settings
gmtmath	Mathematical operations on table data
gmtselect	Select subsets of table data based on multiple spatial criteria
gmtset	Change selected parameters in current .gmtdefaults file
grd2cpt	Generate a color palette table from a gridded file
grd2xyz	Conversion from 2-D gridded file to table data
grdclip	Limit the *z*-range in gridded data sets
grdcontour	Contouring of 2-D gridded data sets
grdcut	Cut a sub-region from a gridded file
grdedit	Modify header information in a 2-D gridded file
grdfft	Perform operations on gridded files in the frequency domain
grdfilter	Filter 2-D gridded data sets in the space domain
grdgradient	Compute directional gradient from gridded files
grdhisteq	Histogram equalization for gridded files
grdimage	Produce images from 2-D gridded data sets
grdinfo	Get information about gridded files
grdlandmask	Create masking gridded files from shoreline data base
grdmask	Reset grid nodes in/outside a clip path to constants
grdmath	Mathematical operations on gridded files
grdpaste	Paste together gridded files along a common edge
grdproject	Project gridded data sets onto a new coordinate system
grdreformat	Converts gridded 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 along 1-D track
grdtrend	Fits polynomial trends to gridded 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
makecpt	Make color palette tables
mapproject	Transformation of coordinate systems for table data
minmax	Report extreme values in table data files
nearneighbor	Nearest-neighbor gridding scheme
project	Project table data onto lines or great circles
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 rasterfiles on a map

psmask	Create overlay to mask out regions on maps
psrose	Plot sector or rose diagrams
psscale	Plot grayscale or colorscale on maps
pstext	Plot textstrings 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
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 gridded file

3.2 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 25 map projections available in GMT.

FILTERING OF 1-D AND 2-D DATA	
blockmean	L_2 estimate (x,y,z) data filters/decimators
blockmedian	L_1 estimate (x,y,z) data filters/decimators
blockmode	Mode estimate (x,y,z) data filters/decimators
filter1d	Filter 1-D data (time series)
grdfilter	Filter 2-D data in space domain
PLOTTING OF 1-D and 2-D DATA	
grdcontour	Contouring of 2-D gridded data
grdimage	Produce images from 2-D gridded data
grdvector	Plot vector fields from 2-D gridded data
grdview	3-D perspective imaging of 2-D gridded data
psbasemap	Create a basemap frame
psclip	Use polygon files as clipping paths
pscoast	Plot coastlines, filled continents, rivers, and political borders
pscontour	Direct contouring or imaging of *xyz* data by triangulation
pshistogram	Plot a histogram
psimage	Plot Sun rasterfiles on a map
psmask	Create overlay to mask specified regions of a map
psrose	Plot sector or rose diagrams
psscale	Plot grayscale or colorscale
pstext	Plot textstrings
pswiggle	Draw anomalies along track
psxy	Plot symbols, polygons, and lines in 2-D
psxyz	Plot symbols, polygons, and lines in 3-D

GRIDDING OF (X,Y,Z) TABLE DATA	
nearneighbor	Nearest-neighbor gridding scheme
surface	Continuous curvature gridding algorithm
triangulate	Perform optimal Delauney triangulation on *xyz* data
SAMPLING OF 1-D AND 2-D DATA	
grdsample	Resample a 2-D gridded data onto new grid
grdtrack	Sampling of 2-D data along 1-D track
sample1d	Resampling of 1-D data
PROJECTION AND MAP-TRANSFORMATION	
grdproject	Transform gridded data to a new coordinate system
mapproject	Transform table data to a new coordinate system
project	Project data onto lines or great circles
INFORMATION	
gmtdefaults	List the current default settings
gmtset	Command-line editing of parameters in the .gmtdefaults file
grdinfo	Get information about the content of gridded files
minmax	Report extreme values in table data files
MISCELLANEOUS	
gmtmath	Reverse Polish Notation (RPN) calculator for table data
makecpt	Create GMT color palette tables
spectrum1d	Compute spectral estimates from time-series
triangulate	Perform optimal Delauney triangulation on xyz data
CONVERT OR EXTRACT SUBSETS OF DATA	
gmtconvert	Convert table data from one format to another
gmtselect	Select table data subsets based on multiple spatial criteria
grd2xyz	Convert 2-D gridded data to table data
grdcut	Cut a sub-region from a gridded file
grdpaste	Paste together gridded files along common edge
grdreformat	Convert from one grid format to another
splitxyz	Split (x,y,z) table data into several segments
xyz2grd	Convert table data to 2-D gridded file
DETERMINE TRENDS IN 1-D AND 2-D DATA	
fitcircle	Finds best-fitting great or small circles
grdtrend	Fits polynomial trends to gridded files ($z = f(x,y)$)
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 gridded file
grdclip	Limit the *z*–range in gridded data sets
grdedit	Modify grid header information
grdfft	Operate on gridded files in frequency domain
grdgradient	Compute directional gradients from gridded files
grdhisteq	Histogram equalization for gridded files
grdlandmask	Creates mask gridded file from coastline database
grdmask	Set grid nodes in/outside a clip path to constants
grdmath	Reverse Polish Notation (RPN) calculator for gridded files
grdvolume	Calculate volume under a surface within a contour

STANDARDIZED COMMAND LINE OPTIONS	
–B*xinfo*[*/yinfo*[*/zinfo*]][*WESNZwesnz+*][*:.title:*] Tickmarks. Each *info* is [**a**]*tick*[**m**|**c**][**f***tick*[**m**|**c**]][**g***tick*[**m**|**c**]][**l**|**p**][:"label":][:,"unit":]	
–H[*n_headers*]	ASCII tables have header record[s]
–J (upper case for width, lower case for scale)	Map projection (see below)
–JA*lon*$_0$/*lat*$_0$/*width*	Lambert azimuthal equal area
–JB*lon*$_0$/*lat*$_0$/*lat*$_1$/*lat*$_2$/*width*	Albers conic equal area
–JC*lon*$_0$/*lat*$_0$/*width*	Cassini cylindrical
–JD*lon*$_0$/*lat*$_0$/*lat*$_1$/*lat*$_2$/*width*	Equidistant conic
–JE*lon*$_0$/*lat*$_0$/*width*	Azimuthal equidistant
–JF*lon*$_0$/*lat*$_0$/*horizon*/*width*	Azimuthal Gnomonic
–JG*lon*$_0$/*lat*$_0$/*width*	Azimuthal orthographic
–JH*lon*$_0$/*width*	Hammer equal area
–JI*lon*$_0$/*width*	Sinusoidal equal area
–JJ*lon*$_0$/*width*	Miller cylindrical
–JKf*lon*$_0$/*width*	Eckert IV equal area
–JKs*lon*$_0$/*width*	Eckert VI equal area
–JL*lon*$_0$/*lat*$_0$/*lat*$_1$/*lat*$_2$/*width*	Lambert conic conformal
–JM*width* or **–JM***lon*$_0$/*lat*$_0$/*width*	Mercator cylindrical
–JN*lon*$_0$/*width*	Robinson
–JOa*lon*$_0$/*lat*$_0$/*az*/*width*	Oblique Mercator, 1: origin and azimuth
–JOb*lon*$_0$/*lat*$_0$/*lon*$_1$/*lat*$_1$/*width*	Oblique Mercator, 2: two points
–JOc*lon*$_0$/*lat*$_0$/*lon*$_p$/*lat*$_p$/*width*	Oblique Mercator, 3: origin and pole
–JP[**a***width*[/*origin*]	Polar [azimuthal] (θ, r) (or cylindrical)
–JQ*lon*$_0$/*width*	Equidistant cylindrical (Plate Carrée)
–JR*lon*$_0$/*width*	Winkel Tripel
–JS*lon*$_0$/*lat*$_0$/*width*	General stereographic
–JT*lon*$_0$/*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**][/*height*[**l**|**p**]][**d**]	Linear, $\log_{10}$, and x^a–y^b (exponential)
–JY*lon*$_0$/*lat*$_s$/*width*	General cylindrical equal area
–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[*/dx/dy/*][*label*]	Plot time-stamp on plot
–V	Run in verbose mode
–X[**a**]*off*	Shift plot origin in *x*-direction
–Y[**a**]*off*	Shift plot origin in *y*-direction
–c*copies*	Set number of plot copies [1]
–:	Expect *y*/*x* input rather than *x*/*y*

4. General features

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

4.1 GMT Units

GMT programs can accept dimensional quantities in **c**m, **i**nch, **m**eter, or **p**oint. There are two ways to ensure that GMT understands which unit you intend to use.

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 4 cm.
2. Set the parameter MEASURE_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.

4.2 GMT defaults

There are more than 50 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 GMT defaults[1], then tries to open the file .gmtdefaults in the current directory. If not found, it will look for that file in your home directory. 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 **gmtdefaults** manual pages). You may create your own .gmtdefaults 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 .gmtdefaults 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. Default values may also be changed from the command line with the utility **gmtset**.

4.3 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:

```
pscoast -R0/20/0/20 -G200 -JM6i -W0.25p -B5 -V > map.ps
```

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

Option	*Meaning*
–B	Defines tickmarks, annotations, and labels for basemaps and axes
–H	Specifies that input tables have header record(s)
–J	Selects a map projection or one of several non-map projections
–K	Allows more plot code to be appended to this plot later
–O	Allows this plot code to be appended to an existing plot
–P	Selects Portrait plot orientation [Default is landscape]
–R	Defines the min. and max. coordinates of the map/plot region
–U	Plots a time-stamp, by default in the lower left corner of page
–V	Verbose operation
–X	Sets the *x*-coordinate for the plot origin on the page
–Y	Sets the *y*-coordinate for the plot origin on the page
–c	Specifies the number of plot copies
–:	Input geographic data are (*lat*,*lon*) rather than (*lon*,*lat*)

Table 4.1: Standardized **GMT** command line switches.

4.4 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 13 switches in Table 4.1 have the same meaning in all the programs (Some programs may not use all of them). These options are described in more detail in the manual pages.

4.5 Command Line History

GMT programs "remember" the standardized command line options (See previous section) 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

```
-Joc190/25.5/327/56/1:500000
```

then a subsequent **psxy** command to plot symbols only needs to state **–J**o in order to activate the same projection. Previous commands are maintained in the file .gmtcommands, 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 last entry in the .gmtcommands file and, if execution is successful, will replace this entry as the previous option argument in the .gmtcommands file. If you call several GMT modules piped together then GMT cannot guarantee that the .gmtcommands file is processed in the intended order from left to right. The only guarantee is that the file will not be clobbered since GMT now 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.

4.6 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.

4.7 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, you must use the **–H** option. The number of header records is one of the many parameters in the .gmtdefaults file, but can be overridden by **–H***n_header_recs*. ASCII files may in many cases also contain sub-headers separating data segments; see Appendix B for complete documentation. For binary table data no headers are allowed.

4.8 Verbose Operation

Most of the programs take an optional **–V** argument which will run the program in the "verbose" mode. 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.

4.9 Output

Most programs write their results, including *PostScript* plots, to standard output. The exceptions are those which may create binary netCDF grd-files such as **surface** (due to the design of netCDF a filename must be provided; however, alternative output formats allowing piping are available). With *UNIX* you can redirect standard output or pipe it into another process. Error messages, usage messages, and verbose comments are written to standard error in all cases. You can use *UNIX* to redirect standard error as well, if you want to create a log file of what you are doing.

4.10 *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). All GMT programs create *PostScript* code by calling the **pslib** plot library (The user may call these functions from his/her own C or FORTRAN plot programs. See the manual pages for **pslib** 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 ***IslandDraw*** and Adobe ***Illustrator*** for further embellishment, change the PAPER_MEDIA setting in the .gmtdefaults file. See Appendix C and the **gmtdefaults** man page for more details.

4.11 Landscape and Portrait Orientations

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). All the programs in GMT have the same default orientation, which is Landscape. Use **–P** to change to Portrait (Note that PAPER_MEDIA is a user-definable parameter, by default set to Letter (or A4). For other paper dimensions you must change this value accordingly).

4.12 Overlay and Continue Modes

A typical *PostScript* file has a beginning, a middle, and an end. The beginning defines certain features (e.g., macros, origin, orientation, scale, etc.) which will be needed to create the plot. The middle has the commands which actually do the plotting. The end tells the graphics device to put out the plot (`showpage` in *PostScript*) and reset the graphics state. Many of the illustrations in this cookbook are built up by appending *PostScript* files together. If you do this, the first file needs a "beginning" and no "end", the last an "end" but no "beginning", and the middle files need only a "middle". You accomplish this automatically with the Overlay (**–O**) and Continue (**–K**) options. Overlay indicates that this plot will be laid on top of an earlier one; therefore the "beginning" is not included in the output. The default is always no overlay, i.e. write out the "beginning". Continue indicates that another plot will follow this one later; therefore the "end" is not included in the output. The default is to output the "end". If you run only one plot program, ignore both the **–O** and **–K** options; they are only used when stacking plots.

4.13 Specifying pen attributes

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

–W*width*[/*color*][[**t***texture*][**p**]

→ *Width* is normally measured in units of the current device resolution (i.e., the value assigned to the parameter DOTS_PR_INCH in your .gmtdefaults file). Thus, if the dpi is set to 300 this unit is 1/300th of an inch. Append **p** to specify pen width in points (1/72 of an inch)[2]. Note that a pen thickness of 5 will be of different physical width depending on your dpi setting, whereas a thickness of 5**p** will always be 5/72 of an inch. Minimum-thickness pens can be achieved by giving zero width, but the result is device-dependent.

→ The *color* can be specified as a *gray* shade in the range 0–255 (linearly going from black to white) or using the RGB system where you specify *r*/*g*/*b*, each ranging from 0–255. Here 0/0/0 is black and 255/255/255 is white.

→ The *texture* attribute controls the appearance of the line. To get a dotted line, simply append "**t**o" after the width and color arguments; a dashed pen is requested with "**t**a". For exact specifications you may append **t***string*:*offset*, where *string* is a series of integers separated by underscores. These numbers represent a pattern by indicating the length of line segments and the gap between segments. The *offset* shifts the pattern along the line. For example, if you want a yellow line of width 2 that alternates between long dashes (20 units), a 10 unit gap, then a 5 unit dash, then another 10 unit gap, with pattern offset by 10 units from the origin, specify **–W**2/255/255/0**t**20_10_5_10:10. Here, the texture units can be specified in dpi units or points (see above).

[2] *PostScript* definition. In the typesetting industry a slighly different definition of point (1/72.27 inch) is used.

4.14 Specifying area fill attributes

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

–G*fill*
–Gp*dpi/pattern*[:**B***r/g/b*[**F***r/g/b*]]

In the first case we may specify a *gray* shade (0–255) or a color (*r*/*g*/*b* in the 0–255 range), similar to the pen color settings. 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 E. The latter allows the user to create customized, repeating images using standard Sun rasterfiles. 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 .gmtdefaults. 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 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***r/g/b*[**F***r/g/b*]) 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 rasterimages 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 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 grayshade fill instead of the patterns. Installing more memory in your printer *may or may not* solve the problem!

4.15 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. The colors may be specified either in the RGB (red, green, blue) system or in the HSV system (hue, saturation, value), and the parameter COLOR_MODEL in the .gmtdefaults file must be set accordingly. Using the RGB system, the format of the cpt-file is:

z_0	R_{min}	G_{min}	B_{min}	z_1	R_{max}	G_{max}	B_{max}	[**A**]
...								
z_{n-2}	R_{min}	G_{min}	B_{min}	z_{n-1}	R_{max}	G_{max}	B_{max}	[**A**]

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 $R_{min} = R_{max}$, $G_{min} = G_{max}$, and $B_{min} = B_{max}$) or a continuous, linear function of *z*. The optional flag **A** is used to indicate annotation of the colorscale when plotted using **psscale**. **A** may be **L**, **U**, or **B** to select annotation of the lower, upper, or both limits of the particular *z*-slice. However, the standard **–B** option can be used by **psscale** to affect annotation and ticking of colorscales. 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 .gmtdefaults file, but can be overridden by the statements

B	R_{back}	G_{back}	B_{back}
F	R_{fore}	G_{fore}	B_{fore}
N	R_{nan}	G_{nan}	B_{nan}

which can be inserted into the beginning or end of the cpt-file. If you prefer the HSV system, set the .gmtdefaults parameter accordingly and replace red, green, blue with hue, saturation, value. Color palette

tables that contain grayshades only may replace the r-g-b triplets with a single grayshade in the 0–255 range.

A few programs (i.e., those that plot polygons such as **grdview**, **psscale**, and **psxy**) can accept pattern fills instead of grayshades. You must give the pattern as in the previous section (no leading **–G** of course), and only the first (low *z*) is used (we cannot interpolate between patterns). Finally, some programs let you skip features whose *z*-slice in the cptfile has grayshades set to –. As an example, consider

```
30    p200/16   80    –
80    –         100   –
100   255       0     0   200   255   255   0
```

where slice $30 < z < 80$ is painted with pattern # 16 at 200 dpi, slice $80 < z < 100$ is skipped, while slice $100 < z < 200$ is painted in a range of red to yellow, 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 I for more details). The illumination is decoupled from the data grd-file in that a separate grdfile holding intensities in the <-1,+1> range must be provided. Such intensity files can be derived from the data grdfile 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.

4.16 Character escape sequences

For annotation labels or textstrings 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 4.2.

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

Table 4.2: **GMT** text escape sequences.

Shorthand notation for a few special Scandinavian characters has also been added (Table 4.3):

Code	*Effect*
@E	Æ
@e	æ
@O	Ø
@o	ø
@A	Å
@a	å

Table 4.3: Shortcuts for Scandinavian characters.

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 (See Appendix F). Also see the definition of (and reason for) WANT_EURO_FONT in the **gmtdefaults** man page. Basically, WANT_EURO_FONT must be set to TRUE for the special characters to be available. Many characters that are 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:

`2@~p@~r@+2@+h@-0@- E\363tv\363s`	=	$2\pi r^2 h_0$ Eötvös
`10@+-3 @Angstr@om`	=	10^{-3} Ångstrøm
`Se\227or Gar\215on`	=	Señor Garçon
`M@!\305anoa stra\373e`	=	Manoa straße
`A@\#cceleration@\# (ms@+-2@+)`	=	ACCELERATION (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 F.

4.17 Embedded grdfile format specifications

GMT has the ability to read more than one grdfile format. As distributed, GMT now recognizes 12 predefined file formats. These are

0. GMT netCDF 4-byte float format [Default]

1. Native binary single precision floats in scanlines with leading grd header

2. Native binary short integers in scanlines with leading grd header

3. 8-bit standard Sun rasterfile (colormap ignored)

4. Native binary unsigned char in scanlines with leading grd header

5. Native binary bits in scanlines with leading grd header

6. Native binary "surfer" grid files

7. netCDF 1-byte byte format

8. netCDF 1-byte char format

9. netCDF 2-byte int format

10. netCDF 4-byte int format

11. netCDF 8-byte double format

In addition, users with some C-programming experience may add their own read/write kernels 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.

Because of these new formats it is sometimes necessary to provide more than simply the name of the file on the command line. For instance, a 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 grdfiles will decode the given filename as follows:

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

where everything in brackets is optional. If you only use the default netCDF file format then no options are needed: just continue to pass the name of the grdfile. However, if you use another format you must append the *=id* string, where *id* is the format id number listed above. In addition, should you want to multiply the data by a scale factor, then add a constant offset you may 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 may append the */nan* suffix to the scaling string (it cannot go by itself; note the nesting of the brackets!).

Some of the grd formats allow writing to standard output and reading from standard input which means you can connect GMT programs that operate on grdfiles with pipes, thereby speeding up execution and eliminating the need for large, intermediate grdfiles. You specify standard input/output by leaving out the filename entirely. That means the "filename" will begin with "*=id*" since the GMT default netCDF format does not allow piping (due to the design of netCDF).

Everything looks more obvious after a few examples:

1. To write a binary float grd file, specify the name as my_file.grd=1.
2. To read a short integer grd 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.grd=2/10/32000/32767.
3. To read a 8-bit standard Sun rasterfile (with values in the 0–255 range) and convert it to a ± 1 range, give the name as rasterfile=3/7.84313725e-3/-1 (i.e., 1/127.5).
4. To write a short integer grd file to standard output after subtracting 32000 and dividing its values by 10, give filename as =2/0.1/-3200.

Programs that both read and/or write more than one grdfile may specify different formats and/or scaling for the files involved. The only restriction with the embedded grd specification mechanism is that no grdfiles 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 home directory and define any number of custom formats. The following is an example of a .gmt_io file:

```
# GMT i/o shorthand file
# It can have any number of comment lines like this one anywhere
# suffix format_id scale offset NaN    Comments
grd     0         -     -      -       Default format
b       1         -     -      -       Native binary floats
i2      2         -     -      32767   2-byte integers
ras     3         -     -      -       Sun rasterfiles
byte    4         -     -      255     1-byte grids
bit     5         -     -      -       0 or 1 grids
mask    5         -     -      0       1 or NaN masks
faa     2         0.1   -      32767   Gravity in 0.1 mGal
```

These suffices can be anything that make sense to the user. To activate this mechanism, set parameter GRIDFILE_SHORTHAND to TRUE in your .gmtdefaults file. Then, using the filename stuff.i2 is equivalent to saying stuff.i2=2/1/0/32767, and the filename wet.mask means wet.mask=5/1/0/0. For a file intended for masking, i.e., the nodes are either 1 or NaN, the bit or mask format file may be down to 1/32 the size of the corresponding grd format file.

4.18 Binary table i/o

All GMT programs that accept table data input may read ASCII or binary data. When using 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**[**s**]*n* option, where *n* is the actual number of data columns (**s** indicates single rather than double precision). 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*. If $n < m$ an error is generated. For more information, see Appendix B.

5. GMT Projections

GMT programs that read position data will need to know how to convert the input coordinates to positions on the map. This is achieved by selecting one of several projections. The purpose of this section is to summarize the properties of map projections available in GMT, what parameters define them, and demonstrate how they are used to create simple basemaps. We will mostly be using the **pscoast** command and occasionally **psxy**. (Our illustrations may differ from yours because of different settings in our .gmtdefaults file.) Note that while we will specify dimensions in inches (by appending **i**), you may want to use cm (**c**), meters (**m**), or points (**p**) as unit instead (see **gmtdefaults** man page).

5.1 Non-map Projections

The linear projection comes in three flavors: linear, $\log_{10}$, and power (or exponential). The projection for the *y*-axis can be set independently from the *x*-axis. We will show examples of all three by first creating dummy data sets using **gmtmath**, a "Reverse Polish Notation" (RPN) calculator that operates on or creates table data:

```
#!/bin/sh
#
#    $Id: GMT_dummydata.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#
#    This script makes the dummy data sets needed in Section 5.1

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

5.1.1 Cartesian Linear Projection (–Jx –JX)

Selection of this transformation will result in a linear scaling 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 5.1.

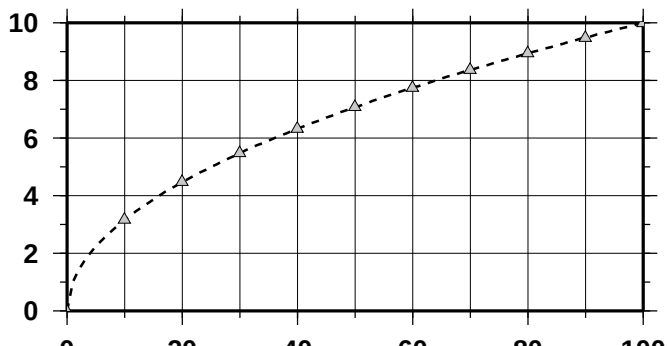
The complete commands given to produce this plot were

```
#!/bin/sh
#    $Id: GMT_linear.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

psxy -R0/100/0/10 -JX3i/1.5i -Ba20f10g10/a2f1g2WSne -W1t3_3:0p -P -K sqrt.d > GMT_linear.ps
psxy -R -JX -St0.075i -G200 -W -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).

5.1.2 Logarithmic projection

The $\log_{10}$ transformation 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), try


10
8
6
4
2
0

```
#!/bin/sh
#    $Id: GMT_log.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

psxy -R1/100/0/10 -Jx1.5il/0.15i -B2g3/a2f1g2WSne -W1t2_2:0p -P -K -H sqrt.d > GMT_log.ps
psxy -R -Jx -Ss0.075i -G0 -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.

5.1.3 Power projection

This projection allows us to display x^a versus y^b. While a and b can be any values, we will select $a = 0.5$ and $b = 1$ which means we will plot y 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 $b = 1$ we do not need to specify **p**1 since it is identical to the linear scaling. Thus our command becomes

```
#!/bin/sh
#    $Id: GMT_pow.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

psxy -R0/100/0/10 -Jx0.3ip0.5/0.15i -Ba1p/a2f1WSne -W1p -P -K sqrt.d > GMT_pow.ps
psxy -R -Jx -Sc0.075i -G255 -W -O sqrt.d10 >> GMT_pow.ps
```

5.1.4 Geographical linear projection

While these linear projections are primarily designed for generic *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 data although a linear projection has been chosen. We do so by appending a **d** (for degrees) to the end of the **–Jx** (or **–JX**) option. As an example, we want to plot a crude world map centered on 125°E. Our command will be

```
#!/bin/sh
#    $Id: GMT_linear_d.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset GRID_CROSS_SIZE 0.1i BASEMAP_TYPE FANCY DEGREE_FORMAT 3
pscoast -R-55/305/-90/90 -Jx0.014id -B60g30f15/30g30f15WSen -Dc -A1000 -G200 -W0.25p -P \
    > GMT_linear_d.ps
gmtset GRID_CROSS_SIZE 0
```

with the result reproduced in Figure 5.4.

5.1.5 Linear Projection with Polar (θ, r) Coordinates (–Jp –JP)

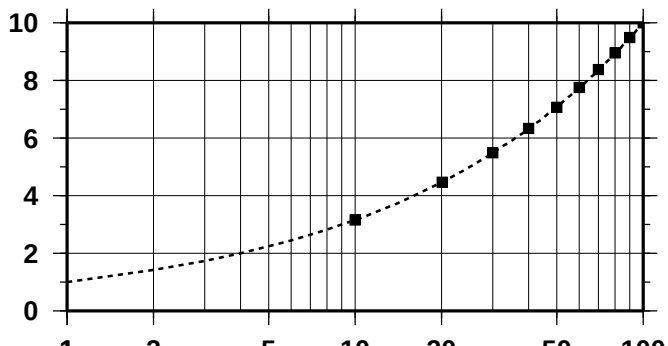
In many applications the data is better described in polar or cylindrical (θ, r) coordinates rather than the usual Cartesian coordinates (x, y). The relationship between the Cartesian and polar coordinates are described by $x = r \cdot \cos\theta, y = r \cdot \sin\theta$. The polar transformation is simply defined by providing

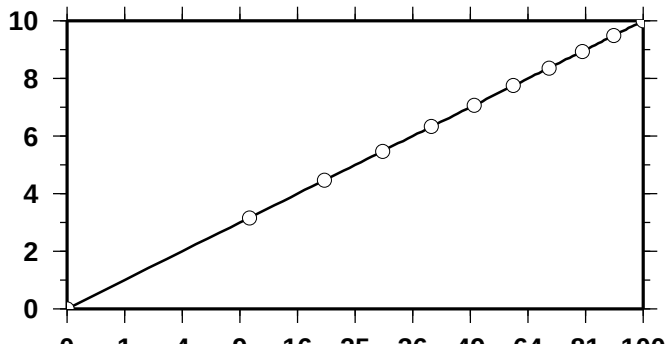
- scale in inches/unit (**–Jp**) or full width of plot in inches (**–JP**)
- Optionally, insert **a** after **p**|**P** to indicate CW azimuths rather than CCW directions
- Optionally, append /*origin* in degrees to indicate an angular offset [0]

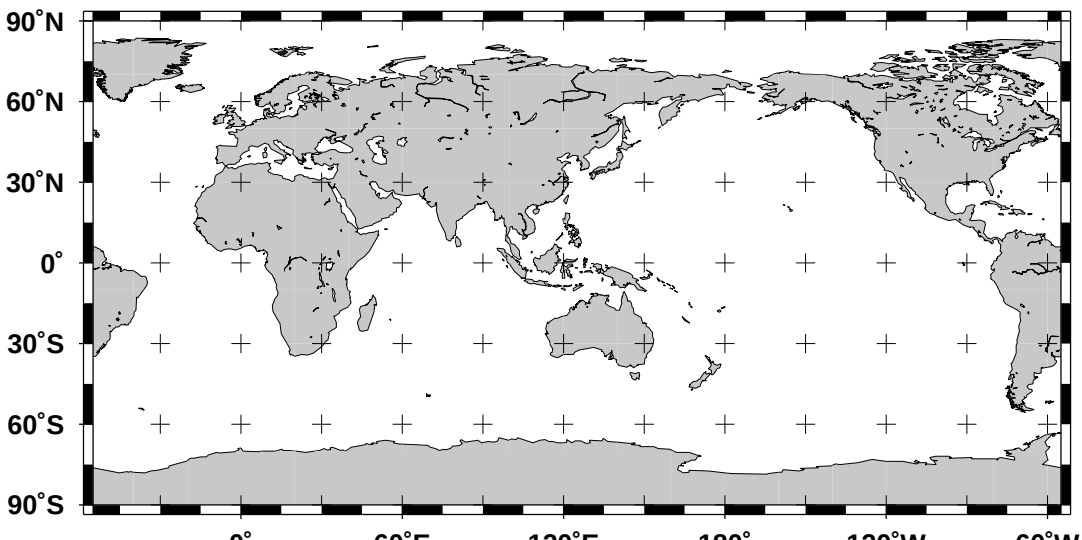
As an example of this projection we will create a gridded data set in polar coordinates $z(\theta, r) = r^2 \cdot \cos 4\theta$ using **grdmath**, a RPN calculator that operates on or creates grdfiles.

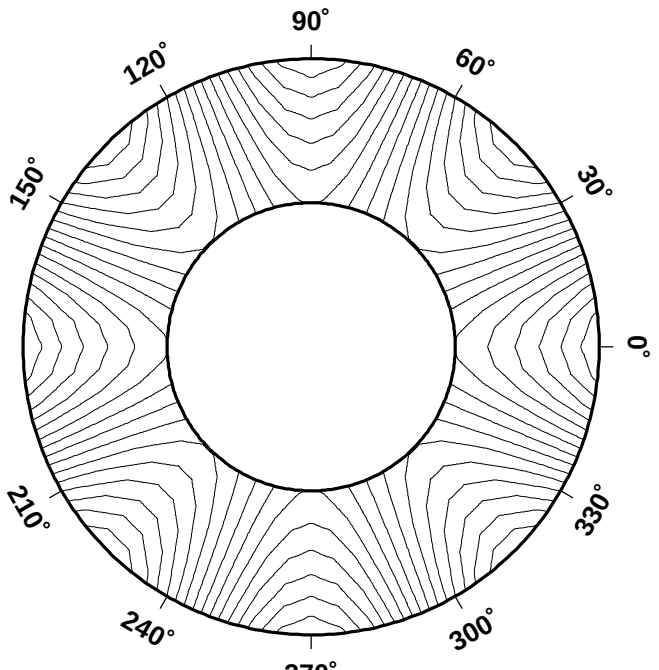
```
#!/bin/sh
#    $Id: GMT_polar.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

grdmath -R0/360/2/4 -I6/0.1 X 4 MUL PI MUL 180 DIV COS Y 2 POW MUL = test.grd
grdcontour test.grd -JP3i -B30Ns -P -C2 -S4 > 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 5.5.


10
8
6
4
2
0


10
8
6
4
2
0


90˚N
60˚N
30˚N
0˚
30˚S
60˚S
90˚S


90°
60°
30°
0°
330°
300°
270°
240°
210°
180°
150°
120°

5.2 Conic Projections

5.2.1 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, 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:

```
#!/bin/sh
#    $Id: GMT_albers.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset GRID_CROSS_SIZE 0
pscoast -R110/140/20/35 -JB125/20/25/45/5i -B10g5 -Dl -G200 -W0.25p -A250 -P > GMT_albers.ps
```

5.2.2 Lambert Conic Conformal Projection (–Jl –JL)

This conic projection was designed by 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 33°N and 45°N. 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

```
#!/bin/sh
#    $Id: GMT_lambert_conic.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset BASEMAP_TYPE FANCY DEGREE_FORMAT 3 GRID_CROSS_SIZE 0.05i
pscoast -R-130/-70/24/52 -Jl-100/35/33/45/1:50000000 -B10g5 -Dl -N1/1p -N2/0.5p -A500 -G200 \
   -W0.25p -P > GMT_lambert_conic.ps
gmtset GRID_CROSS_SIZE 0
```

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

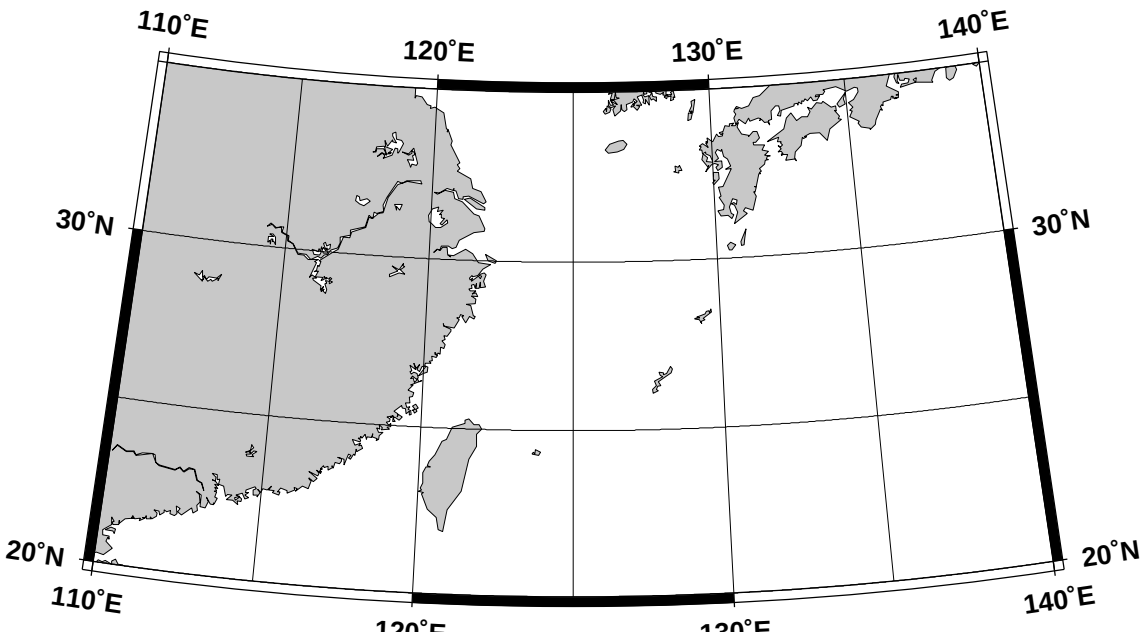
5.2.3 Equidistant Conic Projection (–Jd –JD)

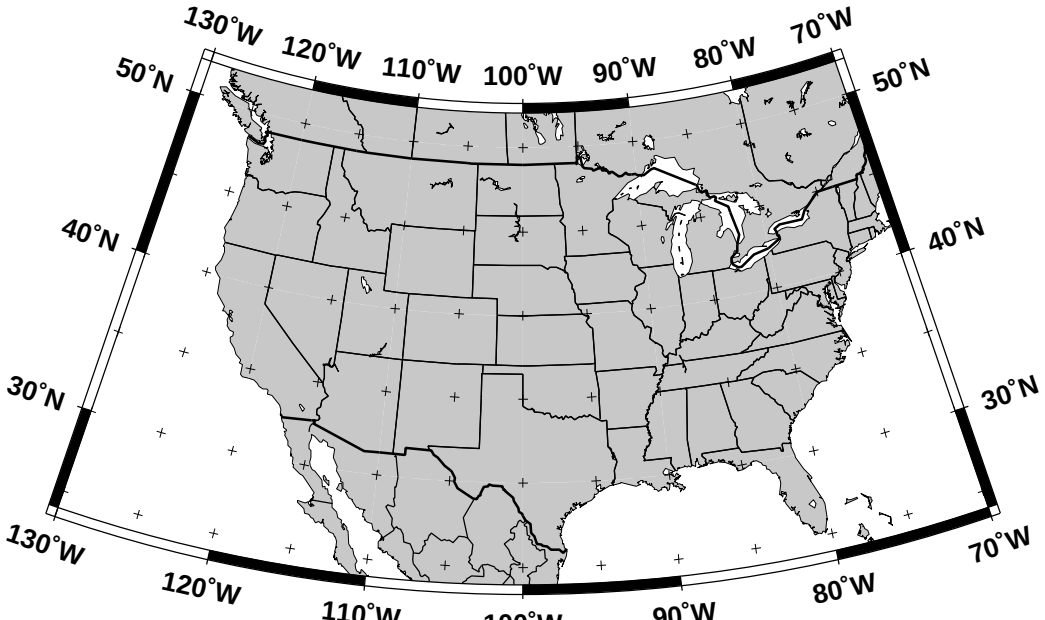
The equidistant conic projection was described by the Greek philosopher Claudius Ptolemy about A.D. 150. It is neither conformal or equal-area, but serves as a compromise between them. The scale is true along all meridians and the standard parallels. To select this projection in GMT you must provide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

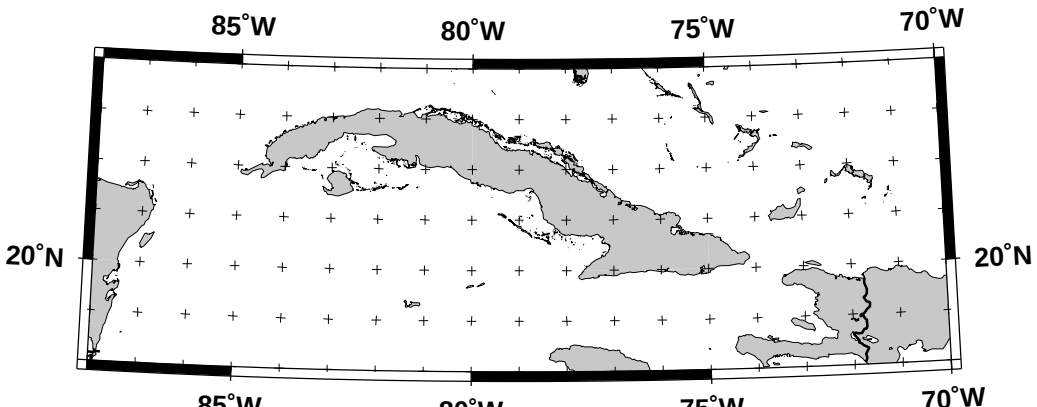
- Longitude and latitude of the projection center
- Two standard parallels
- Map scale in inch/degree or 1:xxxxx notation (**–Jd**), or map width (**–JD**)

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

```
#!/bin/sh
#    $Id: GMT_equidistant_conic.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset DEGREE_FORMAT 3 GRID_CROSS_SIZE 0.05i
pscoast -R-88/-70/18/24 -JD-79/21/19/23/4.5i -B5g1 -Di -N1/1p -G200 \
   -W0.25p -P > GMT_equidistant_conic.ps
gmtset GRID_CROSS_SIZE 0
```


110˚E
120˚E
130˚E
140˚E
30˚N
30˚N
20˚N
20˚N
110˚E
140˚E


130˚W
120˚W
110˚W
100˚W
90˚W
80˚W
70˚W
50˚N
40˚N
30˚N


85°W
80°W
75°W
70°W
20°N
20°N
85°W
80°W
75°W
70°W

5.3 Azimuthal Projections

5.3.1 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
- Scale as 1:xxxxx or as radius/latitude where radius is distance on map in inches from projection center to an oblique latitude (**–Ja**), or map width in inches (**–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 0°E/40°S (lower left) and 60°E/10°S (upper right) as our corners we try

```
#!/bin/sh
#    $Id: GMT_lambert_az_rect.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset DEGREE_FORMAT 0 GRID_CROSS_SIZE 0
pscoast -R0/-40/60/-10r -JA30/-30/4.5i -B30g30/15g15 -Dl -A500 -G200 -W0.25p -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 (**–R**0/360/-90/90). E. g., to obtain a hemisphere view that shows the Americas, try

```
#!/bin/sh
#    $Id: GMT_lambert_az_hemi.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-90/90 -JA280/30/3.5i -B30g30/15g15 -Dc -A1000 -G0 -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* (Wulff) stereonet can be obtained by using the stereographic projection (discussed later). The stereonets produced by these two projections appear below.

5.3.2 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.
- 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**).

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

```
#!/bin/sh
#    $Id: GMT_stereographic_polar.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset DEGREE_FORMAT 1
pscoast -R-30/30/60/72 -Js0/90/4.5i/60 -Ba10g5/5g5 -Dl -A250 -G0 -P > GMT_stereographic_polar.ps
```

Rectangular Stereographic Map

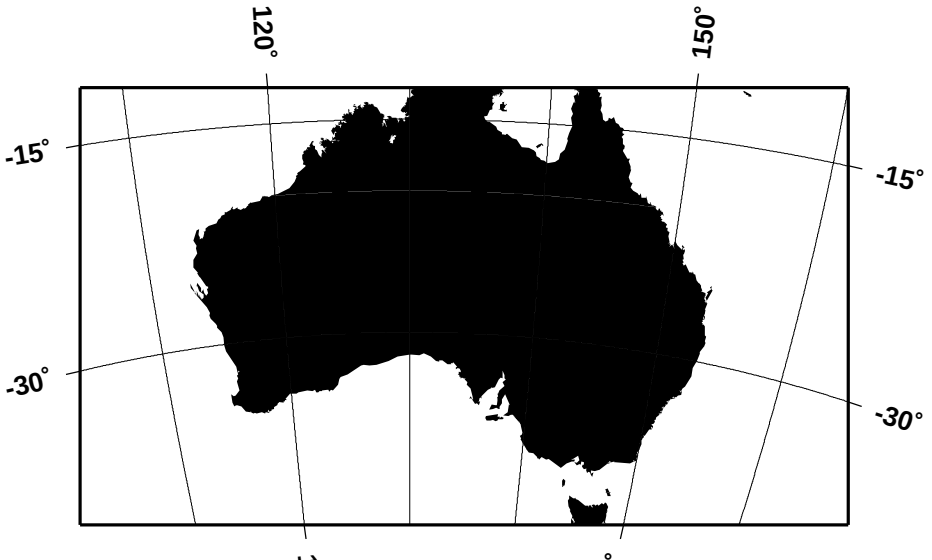
As with Lambert's azimuthal equal-area projection we have the option to use rectangular boundaries rather than the wedge-shape typically associated with polar projections. This choice is defined by selecting two points as corners in the rectangle and appending an "r" to the **–R** option. This command produces a map as presented in Figure 5.13:

```
#!/bin/sh
#    $Id: GMT_stereographic_rect.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset DEGREE_FORMAT 1 OBLIQUE_ANOTATION 30
pscoast -R-25/59/70/72r -JS10/90/11c -B30g10/5g5 -Dl -A250 -G200 -W.25p -P > \
    GMT_stereographic_rect.ps
```

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/30°S. The command used was

```
#!/bin/sh
#    $Id: GMT_stereographic_general.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#
gmtset DEGREE_FORMAT 1 OBLIQUE_ANOTATION 0
pscoast -R100/-40/160/-10r -JS130/-30/4i -B30g10/15g15 -Dl -A500 -G0 -P \
    > GMT_stereographic_general.ps
```


120°
150°
-15°
-15°
-30°
-30°

By choosing 0°/0°as the pole, we obtain the conformal stereonet presented next to its equal-area cousin in the Section 5.3.1 on the Lambert azimuthal equal-area projection (Figure 5.11).

5.3.3 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 space. As with Lambert's equal-areal 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 introducted 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 pictoral views at a small scale, only the spherical form is necessary.

To specify the orthographic projection you must supply

- Longitude and latitude of the projection center.
- 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 75°W/40°N can therefore be generated by the following **pscoast** command:

```
#!/bin/sh
#    $Id: GMT_orthographic.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-90/90 -JG-75/40/4.5i -B15g15 -Dc -A5000 -G0 -P > GMT_orthographic.ps
```

5.3.4 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).

To specify the azimuthal equidistant projection you must supply:

- Longitude and latitude of the projection center.
- 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 100°W/40°N 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:

```
#!/bin/sh
#    $Id: GMT_az_equidistant.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-90/90 -JE-100/40/4.5i -B15g15 -Dc -A10000 -G200 -W0.25p -P > GMT_az_equidistant.ps
```

5.3.5 Gnomonic Projection (–Jf –JF)

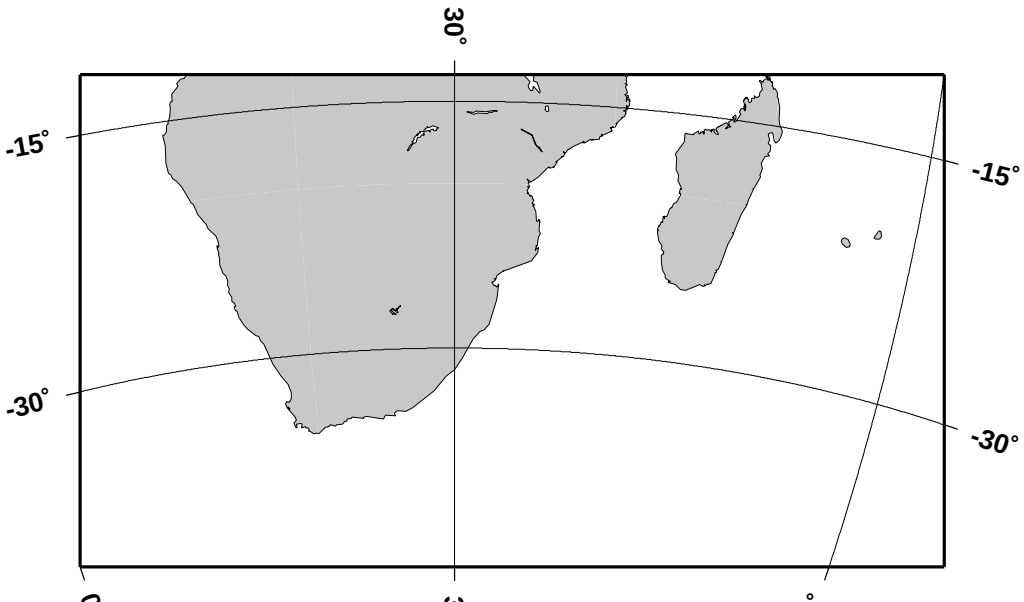
The Gnomonic azimuthal projection is a perspective projection from the center onto a plane tangent to the surface. Its origin goes back to the old Greeks who used it for star maps almost 2500 years ago. The projection is neither equal-area nor conformal, and much distortion is introducted near the edge 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 pictoral views at a small scale, only the spherical form is necessary.

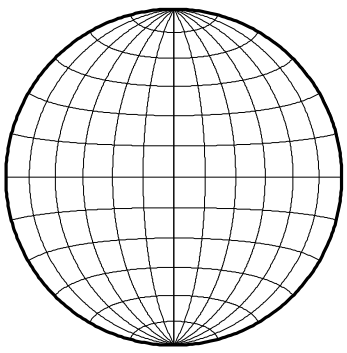
To specify the Gnomonic projection you must supply:

- Longitude and latitude of the projection center.
- The horizon, i.e., the number of degrees from the center to the edge. This must be $< 90^\circ$.
- 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**).

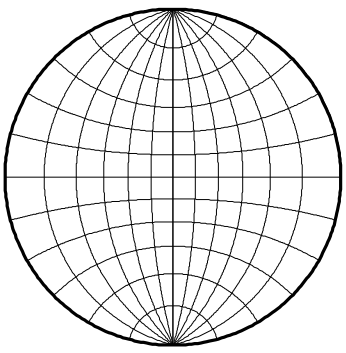
Using a horizon of 60°, our example of this projection centered on 120°W/35°N can therefore be generated by the following **pscoast** command:

```
#!/bin/sh
#    $Id: GMT_gnomonic.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-90/90 -JF-120/35/60/4.5i -Bg15 -Dc -A10000 -G200 -W0.25p -P > GMT_gnomonic.ps
```

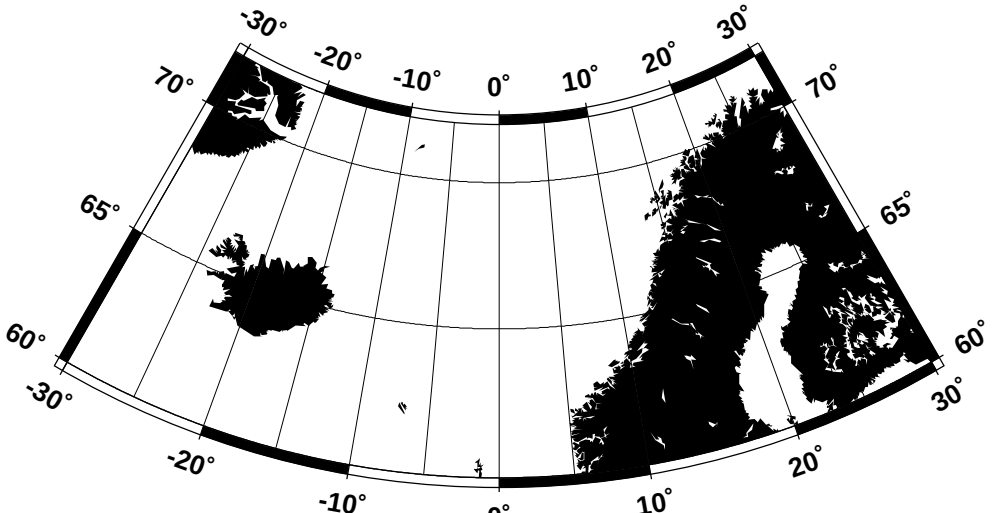

30°
-15°
-15°
-30°
-30°

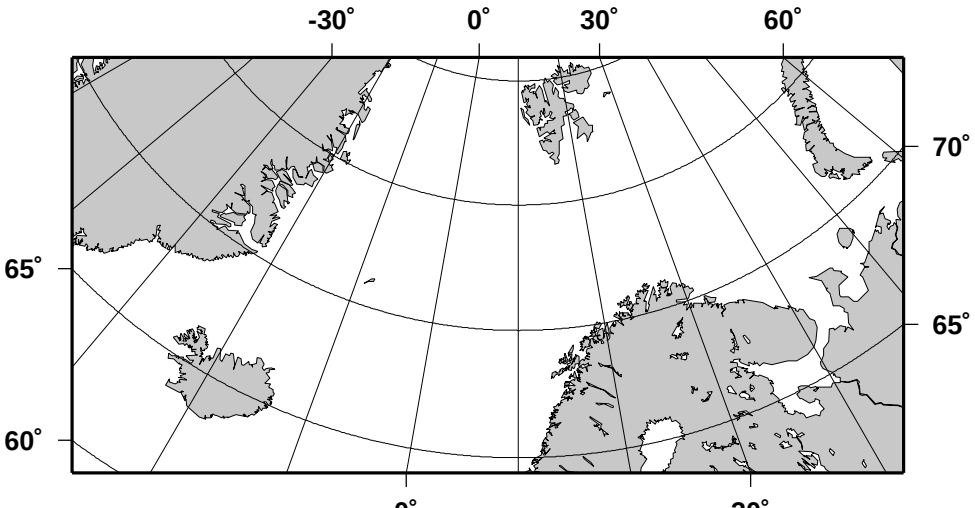


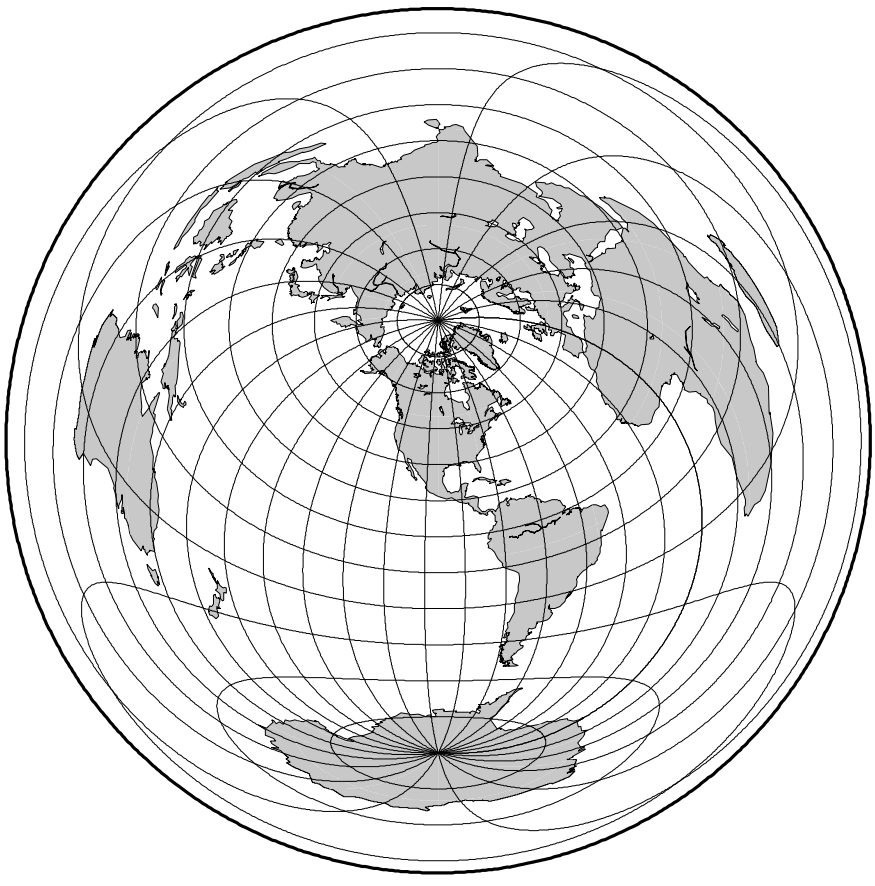
SCHMIDT



WULFF


-30°
-20°
-10°
0°
10°
20°
30°
70°
65°
60°


-30˚
0˚
30˚
60˚
70˚
65˚
65˚
60˚



5.4 Cylindrical Projections

5.4.1 Mercator Projection (–Jm –JM)

Probably the most famous of the various map projections, the Mercator projection takes its name from Mercator who presented it in 1569. It is a cylindrical, conformal projection 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[1]. 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.

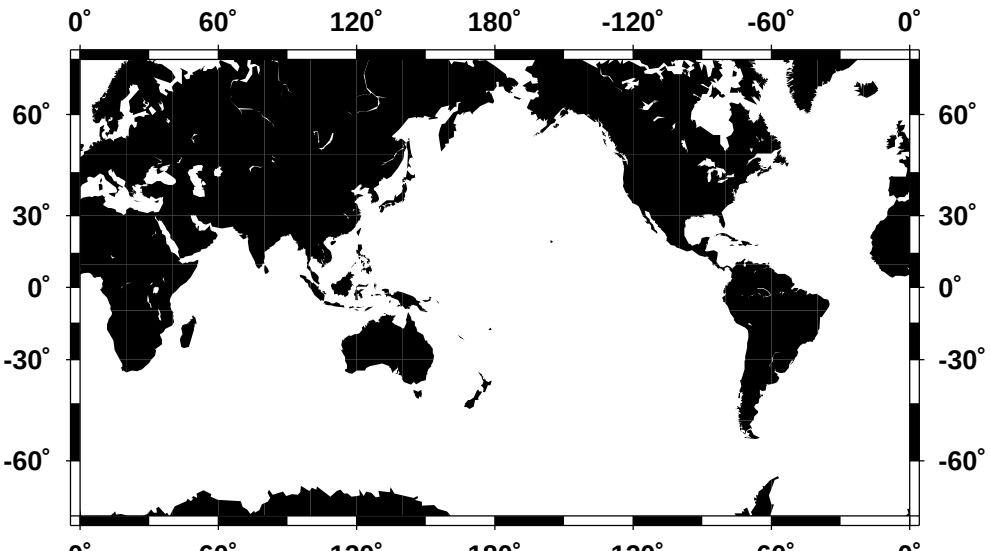
The regular Mercator projection requires a minimum of parameters. To use it in GMT programs you supply this information (the first two items are optional and have defaults):

- Central meridian [Middle of your map]
- Standard parallel for true scale [Equator]
- 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:

```
#!/bin/sh
#    $Id: GMT_mercator.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset DEGREE_FORMAT 1 BASEMAP_TYPE FANCY
pscoast -R0/360/-70/70 -Jm1.2e-2i -Ba60f30/a30f15 -Dc -A5000 -G0 -P > GMT_mercator.ps
```

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


0˚
60˚
120˚
180˚
-120˚
-60˚
0˚
60˚
30˚
0˚
-30˚
-60˚

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.

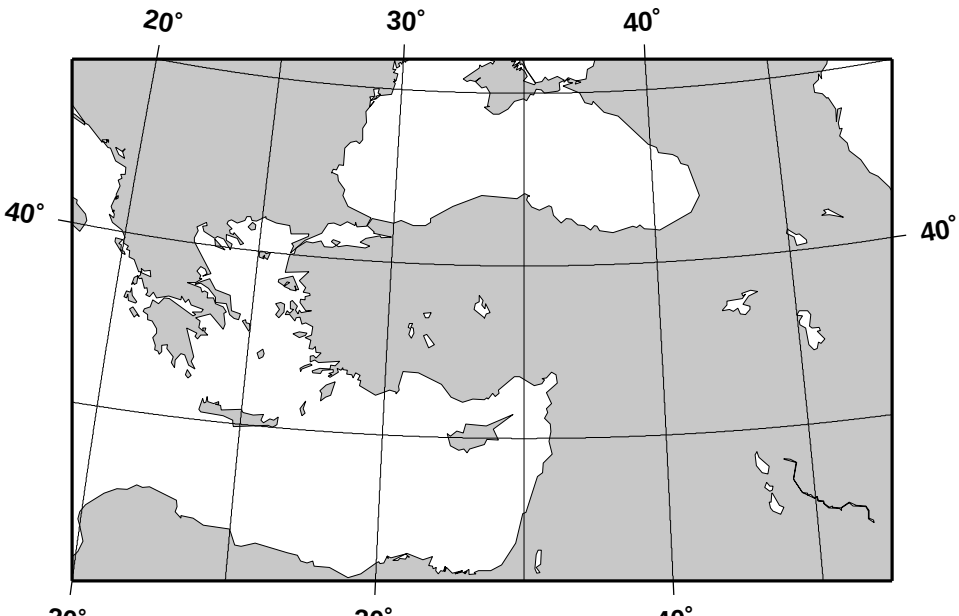
5.4.2 Transverse Mercator (–Jt –JT)

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
- The latitude of origin
- 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. Our example shows a transverse Mercator map of south-east Europe and the Middle East with 35°E as the central meridian:

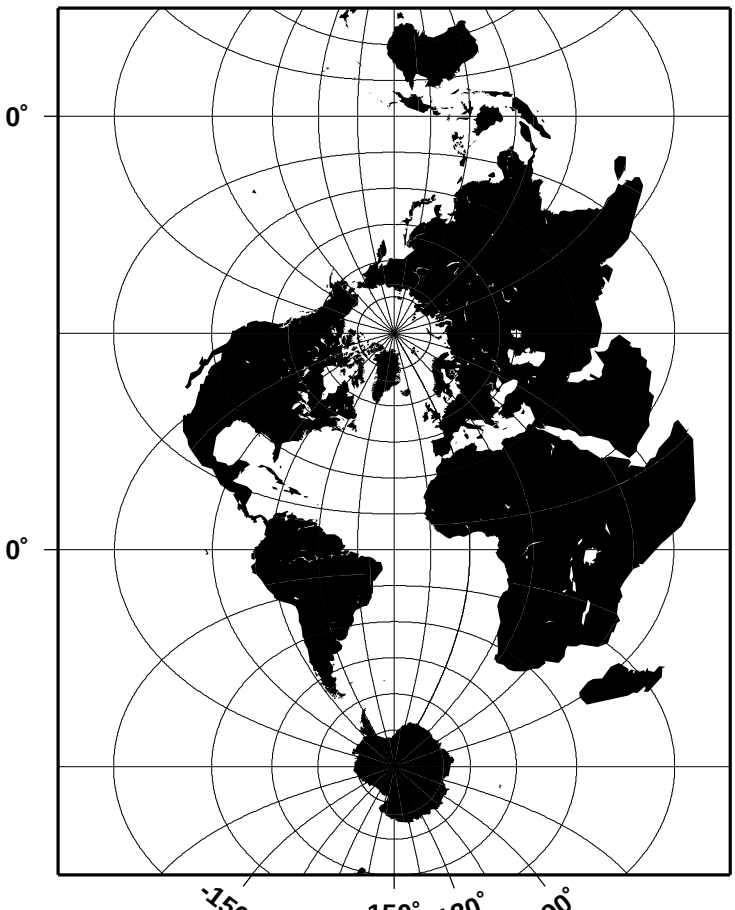
```
#!/bin/sh
#    $Id: GMT_transverse_merc.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R20/30/50/45r -Jt35/0.18i -B10g5 -Dl -A250 -G200 -W0.25p -P > GMT_transverse_merc.ps
```


20°
30°
40°
40°
40°

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

```
#!/bin/sh
#    $Id: GMT_TM.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-80/80 -JT330/-45/3.5i -B30g15/15g15WSne -Dc -A2000 -G0 -P > GMT_TM.ps
```

we made the map illustrated in Figure 5.20. 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 degrees.


0°
0°

5.4.3 Universal Transverse Mercator UTM (–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 84°S and 84°N, most of which are 6° wide. Each of these UTM zones have their unique central meridian. GMT implements both the transverse Mercator and the UTM projection. When selecting UTM you must specify:

- UTM zone (1–60). Use negative value for zones in the southern hemisphere
- 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. 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.

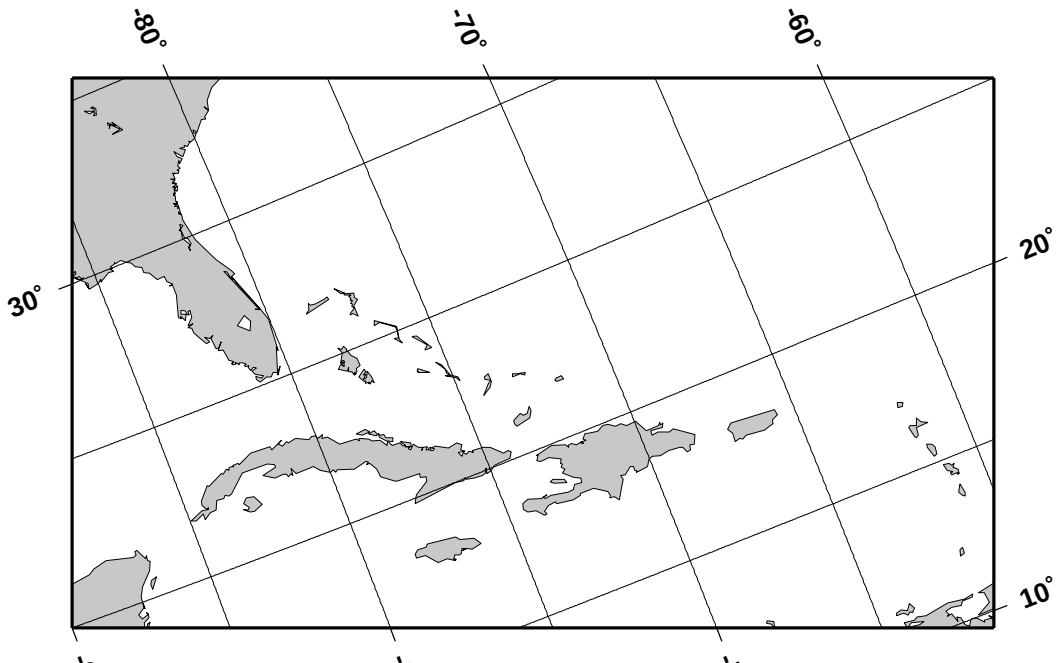
5.4.4 Oblique Mercator (–Jo –JO)

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

```
#!/bin/sh
#    $Id: GMT_oblique_merc.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R270/20/305/25r -JOc280/25.5/22/69/4.8i -B10g5 -Dl -A250 -G200 -W0.25p -P \
    > GMT_oblique_merc.ps
```


-80°
-70°
-60°
30°
20°
10°

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.

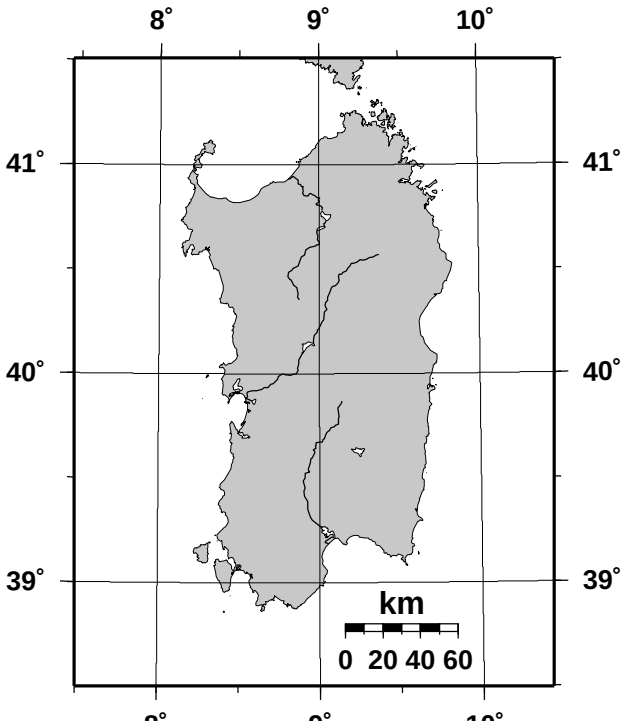
5.4.5 Cassini Cylindrical Projection (–Jc –JC)

This cylindrical projection was developed in 1745 by C. F. Cassini 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 8°45'E meridian using the Cassini projection can be obtained by running the command:

```
#!/bin/sh
#    $Id: GMT_cassini.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

gmtset LABEL_FONT_SIZE 12
pscoast -R7:30/38:30/10:30/41:30r -JC8.75/40/2.5i -B1g1f30m -Lf9.5/38.8/40/60 -Dh -G200 -W0.25p \
   -Ia/0.5p -P > GMT_cassini.ps
```


8°
9°
10°
41°
40°
39°
km
0 20 40 60

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

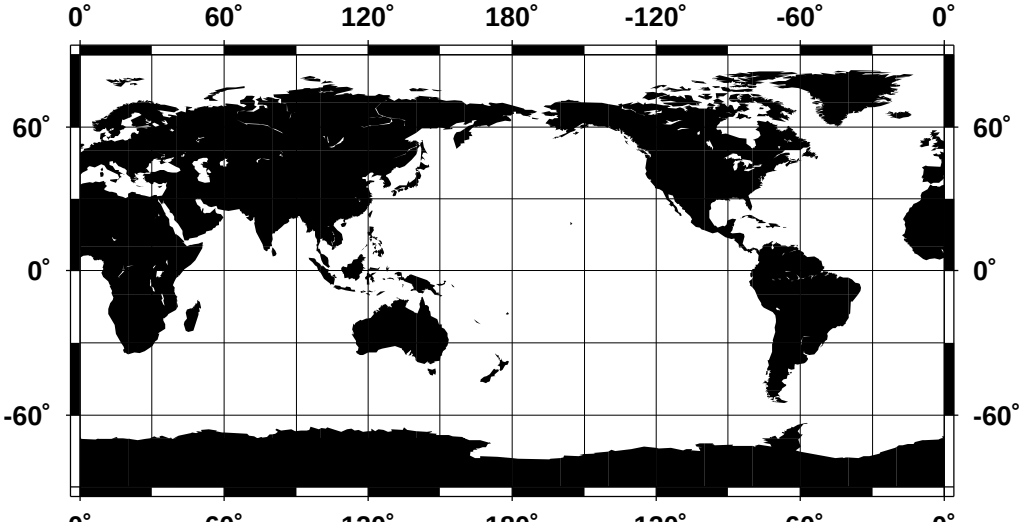
5.4.6 Cylindrical Equidistant Projection (–Jq –JQ)

This simple cylindrical projection is really a linear scaling of longitudes and latitudes (if you desire a different scaling for one of the axes you must choose the linear projection **–Jx** and append **d** for degrees; see Section 5.1.4.) It is also known as the Plate Carrée projection. All meridians and parallels are straight lines. The requirements to define this projection are:

- The central meridian
- Scale in inch/degree or as 1:xxxxx (**–Jq**), or map width (**–JQ**)

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

```
#!/bin/sh
#    $Id: GMT_equi_cyl.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-90/90 -JQ180/4.5i -B60f30g30 -Dc -A5000 -G0 -P > GMT_equi_cyl.ps
```


0˚
60˚
120˚
180˚
-120˚
-60˚
0˚
60˚
0˚
-60˚
60˚
0˚
-60˚

5.4.7 General Cylindrical 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**)

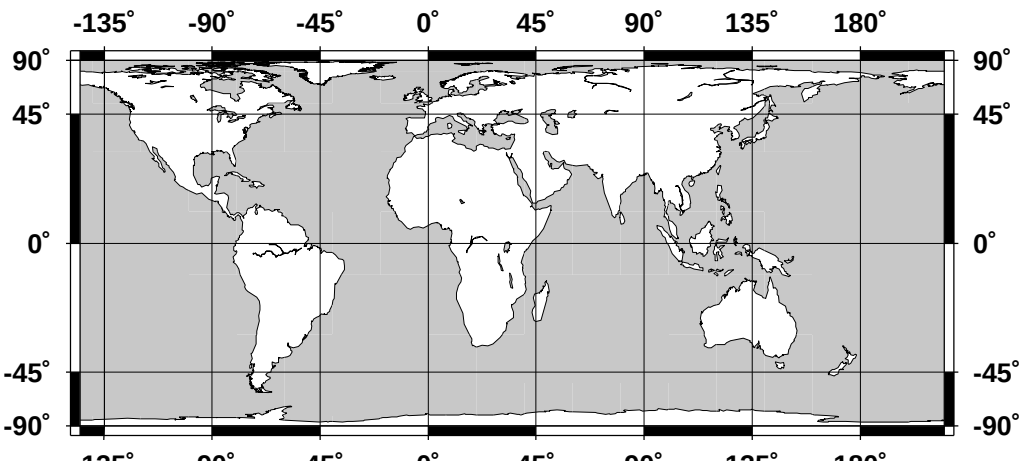
While you may choose any value for the standard parallel and obtain your own personal projection, there are four choices of standard parallels that result in known (or named) projections. These are listed in Table 5.1.

Projection name	*Standard parallel*
Lambert	0°
Behrman	30°
Trystan-Edwards	37°24’ (= 37.4°)
Peters (Gall)	45°

Table 5.1: Standard parallels for some cylindrical projections.

For instance, a world map centered on the 35°E meridian using the Behrman projection can be obtained by running the command:

```
#!/bin/sh
#    $Id: GMT_general_cyl.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R-145/215/-90/90 -JY35/30/4.5i -B45g45 -Dc -A10000 -S200 -W0.25p -P > GMT_general_cyl.ps
```


-135°
-90°
-45°
0°
45°
90°
135°
180°
90°
45°
0°
-45°
-90°

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

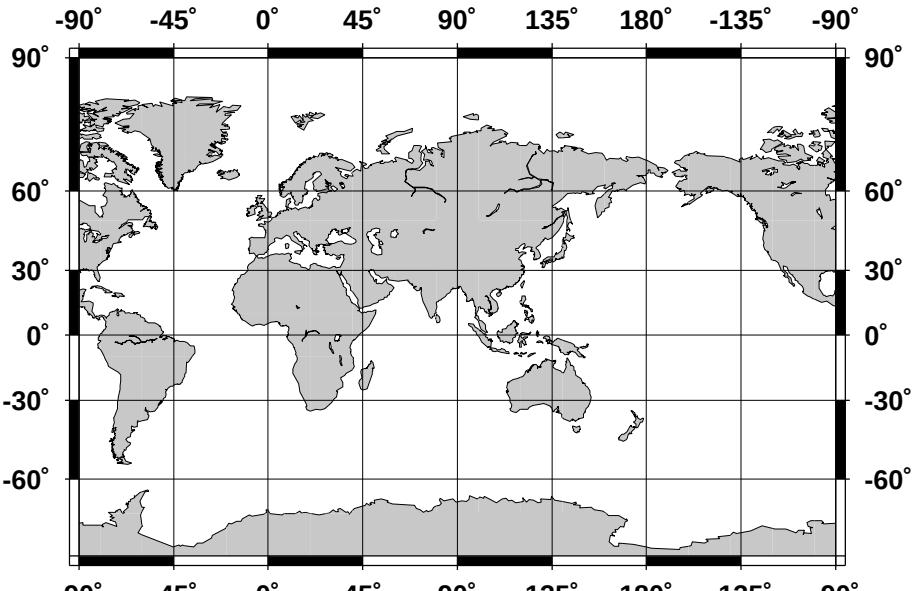
5.4.8 Miller Cylindrical Projections (–Jj –JJ)

This cylindrical projection, presented by O. M. 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 Mercators 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. The requirements to define this projection are:

- The central meridian
- The standard parallel
- Scale in inch/degree or as 1:xxxxx (**–Jj**), or map width (**–JJ**)

For instance, a world map centered on the 90°E meridian at a map scale of 1: 400,000,000 can be obtained as follows:

```
#!/bin/sh
#    $Id: GMT_miller.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R-90/270/-80/90 -Jj90/1:400000000 -B45g45/30g30 -Dc -A10000 -G200 -W0.25p -P \
    > GMT_miller.ps
```


-90˚
-45˚
0˚
45˚
90˚
135˚
180˚
-135˚
-90˚
90˚
60˚
30˚
0˚
-30˚
-60˚

5.5 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.

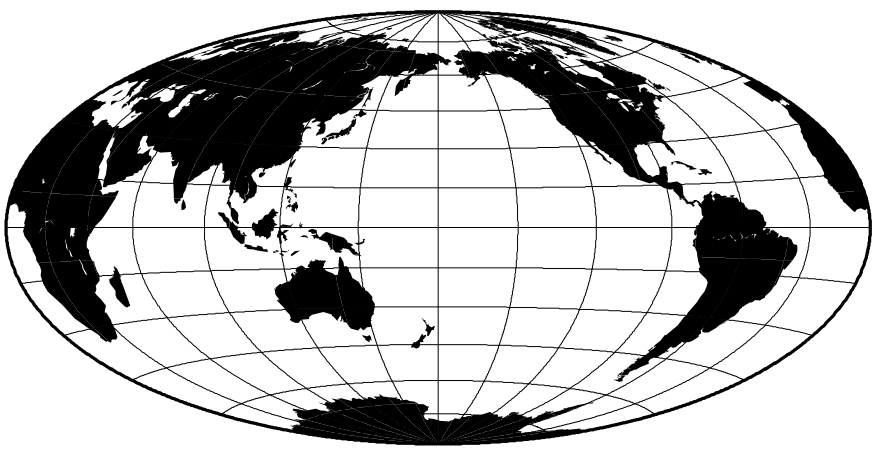
5.5.1 Hammer Projection (–Jh –JH)

The equal-area Hammer projection, first presented by 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
- 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

```
#!/bin/sh
#    $Id: GMT_hammer.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-90/90 -JH180/4.5i -Bg30/g15 -Dc -A10000 -G0 -P > GMT_hammer.ps
```



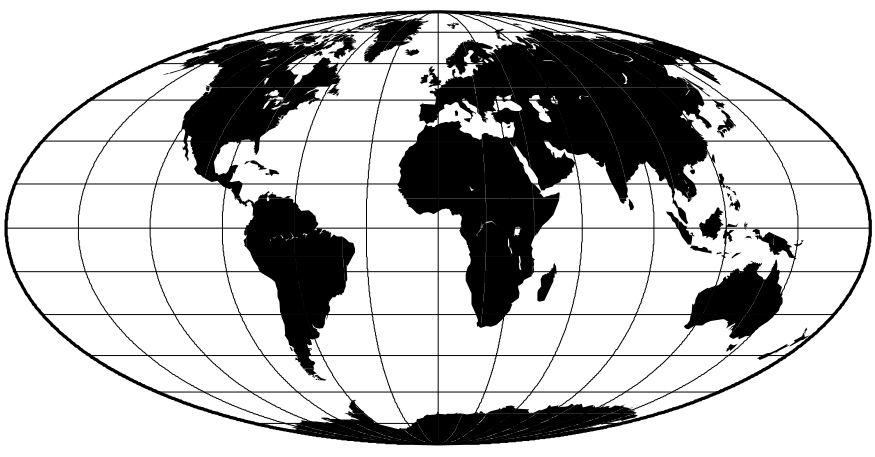
5.5.2 Mollweide Projection (–Jw –JW)

This pseudo-cylindrical, equal-area projection was developed by Mollweide in 1805. Parallels are unequally spaced straight lines with the meridians being equally spaced elliptical arcs. The scale is only true along latitudes 40°44' 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 only two parameters to completely define the mapping of longitudes and latitudes into rectangular *x*/*y* coordinates:

- The central meridian
- Scale along equator in inch/degree or 1:xxxxx (**–Jw**), or map width (**–JW**)

An example centered on Greenwich can be generated thus:

```
#!/bin/sh
#    $Id: GMT_mollweide.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R-180/180/-90/90 -JW0/4.5i -Bg30/g15 -Dc -A10000 -G0 -P > GMT_mollweide.ps
```



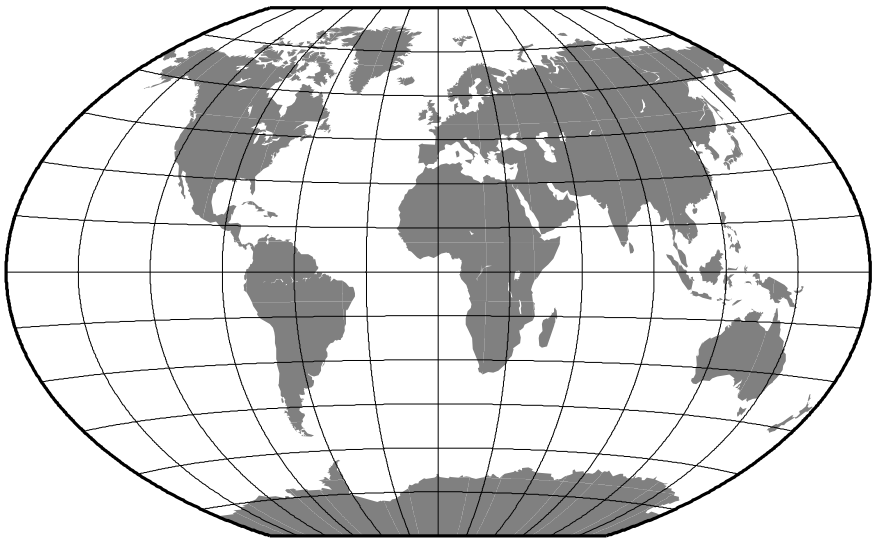
5.5.3 Winkel Tripel Projection (–Jr –JR)

The Winkel Tripel projection, presented by Oswald Winkel in 1921, is a modified azimuthal projection that 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
- Scale along equator in inch/degree or 1:xxxxx (**–Jr**), or map width (**–JR**)

Centered on Greenwich, the example in Figure 5.28 was created by this command:

```
#!/bin/sh
#    $Id: GMT_winkel.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R-180/180/-90/90 -JR0/4.5i -Bg30/g15 -Dc -A10000 -G128 -P > GMT_winkel.ps
```



5.5.4 Robinson Projection (–Jn –JN)

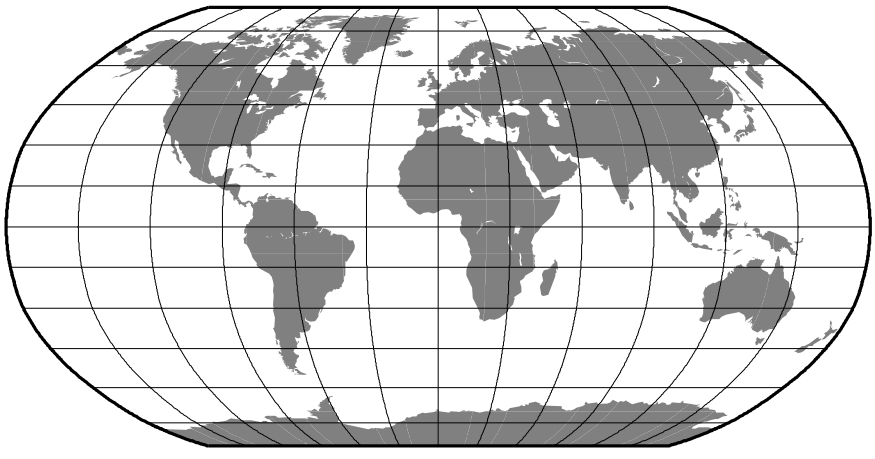
The Robinson projection, presented by Arthur H. Robinson in 1963, is a modified cylindrical projection that is neither conformal nor equal-area. Central meridian and all parallels are straight lines; other meridians are curved. It uses lookup tables rather than analytic expressions to make the world map "look" right[2]. The scale is true along latitudes $\pm$ 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
- 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:

```
#!/bin/sh
#    $Id: GMT_robinson.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R-180/180/-90/90 -JN0/4.5i -Bg30/g15 -Dc -A10000 -G128 -P > GMT_robinson.ps
```

[2]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 INTERPOLANT in the .gmtdefaults file.



5.5.5 Eckert IV and VI Projection (–Jk –JK)

The Eckert IV and VI projections, presented by Max Eckert in 1906, are pseudocylindrical equal-area projections. Central meridian and all parallels are straight lines; other meridians are equally spaced elliptical arcs (IV) or sinusoids (VI). The scale is true along latitudes $\pm$ 40°30' (IV) and $\pm$ 49°16' (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
- 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:

```
#!/bin/sh
#    $Id: GMT_eckert4.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-90/90 -JKf180/4.5i -Bg30/g15 -Dc -A10000 -W0.25p -G255 -S200 -P > GMT_eckert4.ps
```

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

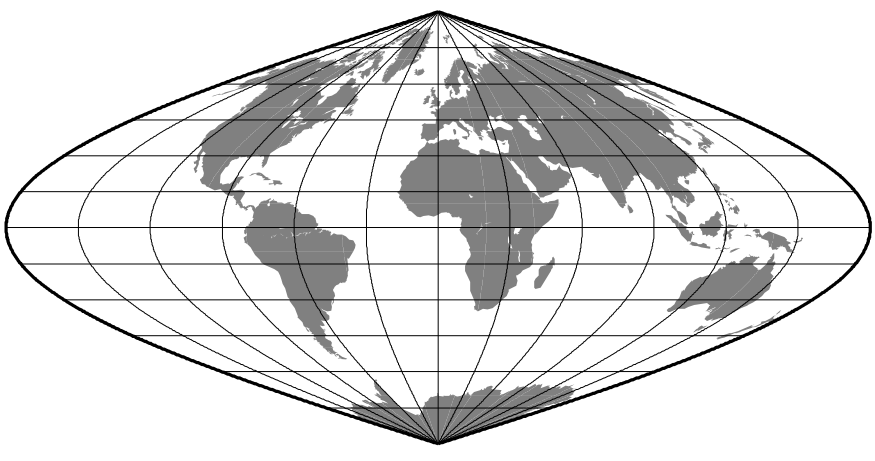
5.5.6 Sinusoidal Projection (–Ji –JI)

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

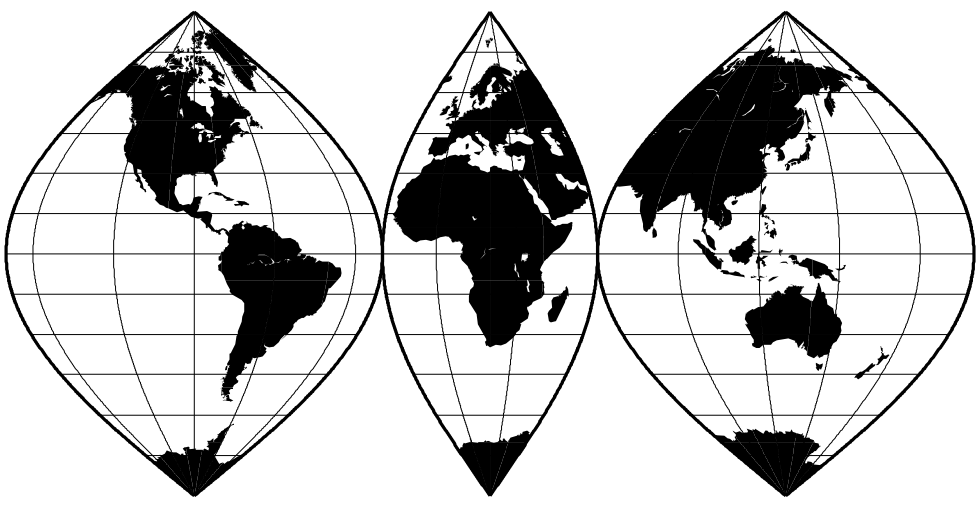
A simple world map using the sinusoidal projection is therefore obtained by

```
#!/bin/sh
#    $Id: GMT_sinusoidal.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#
pscoast -R-180/180/-90/90 -JI0/4.5i -Bg30/g15 -Dc -A10000 -G128 -P > GMT_sinusoidal.ps
```



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 160°W, 20°W, and 60°E. 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):

```
#!/bin/sh
#    $Id: GMT_sinusoidal_int.sh,v 1.1.4.1 2004/01/03 03:45:37 pwessel Exp $
#

pscoast -R200/340/-90/90 -Ji270/0.014i -Bg30/g15 -A10000 -Dc -G0 -K -P > GMT_sinusoidal_int.ps
pscoast -R-20/60/-90/90 -Ji20/0.014i -Bg30/g15 -Dc -A10000 -G0 -X1.96i -O -K >> GMT_sinusoidal_int.ps
pscoast -R60/200/-90/90 -Ji130/0.014i -Bg30/g15 -Dc -A10000 -G0 -X1.12i -O >> GMT_sinusoidal_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.

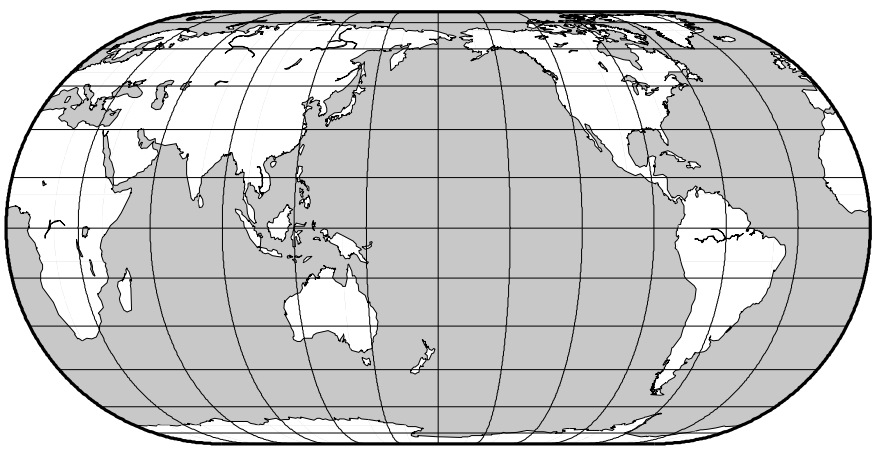
5.5.7 Van der Grinten Projection (–Jv –JV)

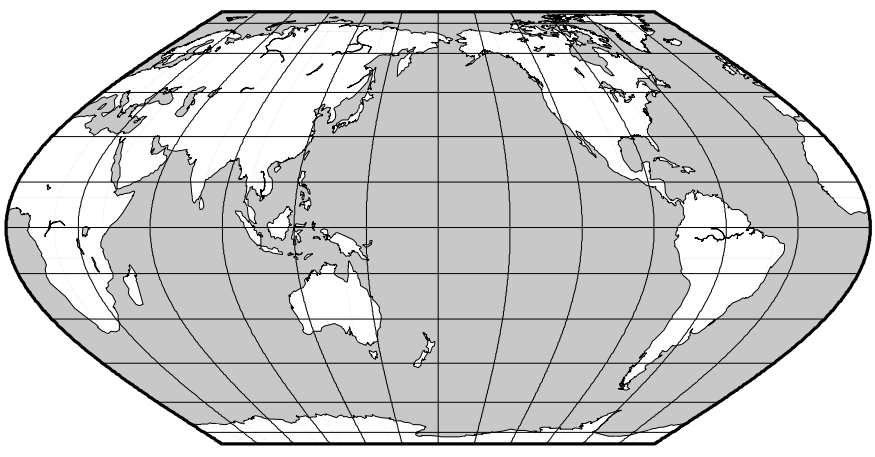
The Van der Grinten projection, presented by Alphons J. van der Grinten in 1904, is neither equal-area nor conformal. Central meridian and Equator are straight lines; other meridians are arcs of circles. The scale is true along the Equator only. Its main use is to show the entire world enclosed in a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

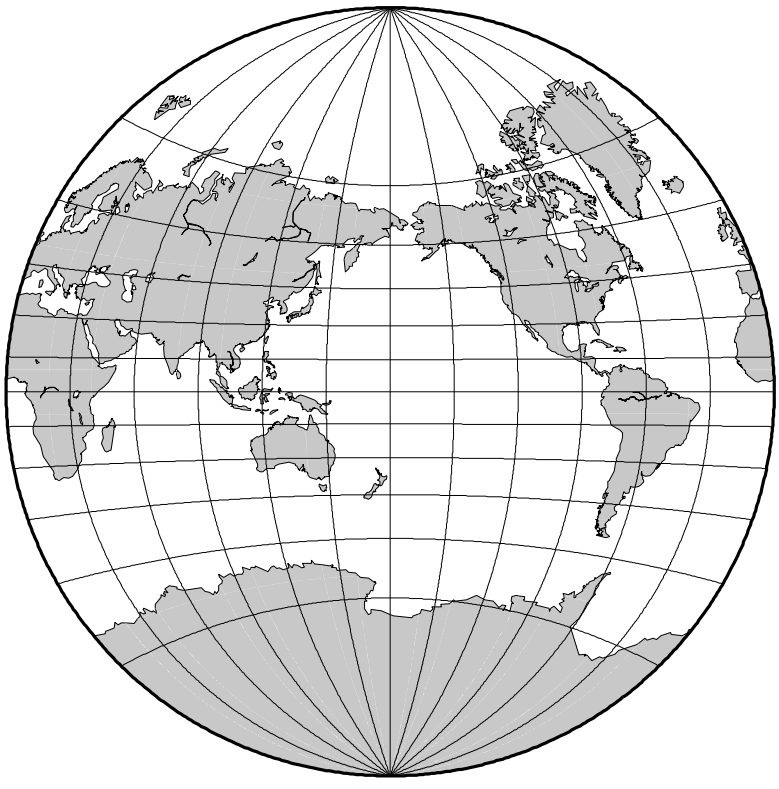
- The central meridian
- Scale along equator in inch/degree or 1:xxxxx (**–Jv**), or map width (**–JV**)

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

```
#!/bin/sh
#    $Id: GMT_grinten.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#

pscoast -R0/360/-90/90 -JV180/4i -Bg30/g15 -Dc -G200 -A10000 -W0.25p -P > GMT_grinten.ps
```







6. Cook-book

In this section we will be giving several 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 a cshell 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 D if you would like an electronic version of all the shell-scripts (both ***csh*** and ***bash*** scripts are available; only the ***csh***-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 ***awk*** is broken as designed on some systems, and ***nawk*** is not available on others we refer to ***$AWK*** in the scripts below; the ***do_examples*** scripts will set this when running all examples.

6.1 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.grd 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 gray in the background. Finally, we want a rectangular frame surrounding the two maps. This is how it is done:

```
gmtset GRID_CROSS_SIZE 0 ANOT_FONT_SIZE 10
psbasemap -R0/6.5/0/9 -Jx1i -B0 -P -K -U"Example 1 in Cookbook" >! example_01.ps
pscoast -R-180/180/-90/90 -JH0/6i -X0.25i -Y0.5i -O -K -Bg30 -Dc -G200 >> example_01.ps
grdcontour -R osu91a1f_16.grd -JH -C10 -A50f7 -G4i -L-1000/-1 -Wc0.25pta -Wa0.75pt2_2:0 -O -K \
   -T0.1i/0.02i >> example_01.ps
grdcontour -R osu91a1f_16.grd -JH -C10 -A50f7 -G4i -L-1/1000 -O -K -T0.1i/0.02i >> example_01.ps
pscoast -R0/360/-90/90 -JH180/6i -Y4i -O -K -Bg30:."Low Order Geoid": -Dc -G200 >> example_01.ps
grdcontour osu91a1f_16.grd -JH -C10 -A50f7 -G4i -L-1000/-1 -Wc0.25pta -Wa0.75pt2_2:0 -O -K \
   -T0.1i/0.02i:-+ >> example_01.ps
grdcontour osu91a1f_16.grd -JH -C10 -A50f7 -G4i -L-1/1000 -O -T0.1i/0.02i:-+ >> example_01.ps
\rm -f .gmtcommands
```

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 tickmarks 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 Figure 6.1.

6.2 Image presentations

As our second example we will demonstrate how to make color images from gridded data sets (again, we will deferr the actual making of gridded files to later examples). We will use the supplemental program **grdraster** to extract 2-D grdfiles 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 (68°W, 69°N). 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:

```
gmtset HEADER_FONT_SIZE 30 OBLIQUE_ANOTATION 0 DEGREE_FORMAT 0
makecpt -Crainbow -T-2/14/2 >! g.cpt
grdimage HI_geoid2.grd -R160/20/220/30r -JOc190/25.5/292/69/4.5i -E50 -K -P -B10 -Cg.cpt \
   -U/-1.25i/-1i/"Example 2 in Cookbook" -X1.5i -Y1.25i >! example_02.ps
psscale -Cg.cpt -D5.1i/1.35i/2.88i/0.4i -O -K -L -B2:GEOID:/:m: -E >> example_02.ps
grd2cpt HI_topo2.grd -Crelief -Z >! t.cpt
grdgradient HI_topo2.grd -A0 -Nt -GHI_topo2_int.grd
grdimage HI_topo2.grd -IHI_topo2_int.grd -R -JO -E50 -B10:."H@#awaiian@# T@#opo and @#G@#eoid:" -O -K \
   -Ct.cpt -Y4.5i >> example_02.ps
psscale -Ct.cpt -D5.1i/1.35i/2.88i/0.4i -O -K -I0.3 -B2:TOPO:/:km: >> example_02.ps
cat << EOF | pstext -R0/8.5/0/11 -Jx1i -O -N -Y-4.5i >> example_02.ps
-0.4 7.5 30 0.0 1 2 a)
-0.4 3.0 30 0.0 1 2 b)
EOF
\rm -f .gmtcommands HI_topo2_int.grd ?.cpt
```

The first step extracts the 2-D data sets from the local data base using **grdraster**, which is a supplemental utility program (see Appendix A) 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 grdfile format mechanism we saved the topography using kilometers as the data unit. We now have two grdfiles 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 scale to the right of the image with **psscale**. We also annotate the color scales with **psscale**. Similarly, we run **grdimage** but specify **–Y**4.5 to plot above the previous image. Adding scale and label the two plots a) and b) completes the illustration (Figure 6.2).

6.3 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 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 **minmax** 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_3[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:

```
fitcircle sat.xyg -L2 >! report
set cpos = ‘grep "L2 Average Position" report‘
set ppos = ‘grep "L2 N Hemisphere" report‘
project  sat.xyg -C$cpos[1]/$cpos[2] -T$ppos[1]/$ppos[2] -S -Fpz -Q >! sat.pg
project ship.xyg -C$cpos[1]/$cpos[2] -T$ppos[1]/$ppos[2] -S -Fpz -Q >! ship.pg
set plotr = ‘cat sat.pg ship.pg | minmax -I100/25 -C‘
gmtset MEASURE_UNIT INCH
psxy -R$plotr[1]/$plotr[2]/$plotr[3]/$plotr[4] -U/-1.75i/-1.25i/"Example 3a in Cookbook" \
   -JX8i/5i -X2i -Y1.5i -K -W1p sat.pg \
   -Ba500f100:"Distance along great circle":/a100f25:"Gravity anomaly (mGal)":WeSn >! example_03a.ps
psxy -R -JX -O -Sp0.03i ship.pg >> example_03a.ps
$AWK ’{ if (NR > 1) print $1 - last1; last1 = $1; }’ ship.pg | pshistogram  -W0.1 -G0 -JX3i -K \
   -X2i -Y1.5i -B:."Ship": -U/-1.75i/-1.25i/"Example 3b in Cookbook" >! example_03b.ps
$AWK ’{ if (NR > 1) print $1 - last1; last1 = $1; }’ sat.pg  | pshistogram  -W0.1 -G0 -JX3i -O \
   -X5i -B:."Sat": >> example_03b.ps
head -1 ship.pg >! ship.pg.extr
head -1 sat.pg >! sat.pg.extr
paste ship.pg.extr sat.pg.extr | $AWK ’{ if ($1 > $3) print int($1); else print int($3); }’ \
   >! sampr1
tail -1 ship.pg >! ship.pg.extr
tail -1 sat.pg >! sat.pg.extr
paste ship.pg.extr sat.pg.extr | $AWK ’{ if ($1 < $3) print int($1); else print int($3); }’ \
   >! sampr2
set sampr = ‘paste sampr1 sampr2‘
$AWK ’BEGIN { for (i = ’$sampr[1]’; i <= ’$sampr[2]’; i++) print i }’ /dev/null >! samp.x
sample1d sat.pg -Nsamp.x >! samp_sat.pg
filter1d ship.pg -Fm1 -T$sampr[1]/$sampr[2]/1 -E | sample1d -Nsamp.x >! samp_ship.pg
psxy -R$plotr[1]/$plotr[2]/$plotr[3]/$plotr[4] -JX8i/5i -X2i -Y1.5i -K -W1p samp_sat.pg \
   -Ba500f100:"Distance along great circle":/a100f25:"Gravity anomaly (mGal)":WeSn  \
   -U/-1.75i/-1.25i/"Example 3c in Cookbook" >! example_03c.ps
psxy -R -JX -O -Sp0.03i samp_ship.pg >> example_03c.ps
paste samp_ship.pg samp_sat.pg | cut -f2,4 | spectrum1d -S256 -D1 -W -C >& /dev/null
psxy spectrum.coh -Ba1f3p:"Wavelength (km)":/a0.25f0.05:"Coherency@+2@+":WeSn -JX-4il/3.75i \
   -R1/1000/0/1 -U/-2.25i/-1.25i/"Example 3d in Cookbook" -P -K -X2.5i -Sc0.07i -G0 \
   -Ey/2 -Y1.5i >! example_03.ps
echo "3.85 3.6 18 0.0 1 11 Coherency@+2@+" | pstext -R0/4/0/3.75 -Jx1i -O -K >> example_03.ps
cat << END >! box.d
2.375    3.75
2.375    3.25
4    3.25
END
psxy -R -Jx -O -K -W1.5p box.d >> example_03.ps
psxy -Ba1f3p/a1f3p:"Power (mGal@+2@+km)"::."Ship and Satellite Gravity":WeSn spectrum.xpower \
   -St0.07i -O -R1/1000/0.1/10000 -JX-4il/3.75il -Y4.2i -K -Ey/2 >> example_03.ps
psxy spectrum.ypower -R -JX -O -K -G0 -Sc0.07i -Ey/2 >> example_03.ps
echo "3.9 3.6 18 0.0 1 11 Input Power" | pstext -R0/4/0/3.75 -Jx -O -K >> example_03.ps
psxy -R -Jx -O -K -W1.5p box.d >> example_03.ps
psxy -R -Jx -O -K -G240 -L -W1.5p << END >> example_03.ps
0.25    0.25
1.4    0.25
1.4    0.9
0.25    0.9
END
echo "0.4 0.7" | psxy -R -Jx -O -K -St0.07i -G0 >> example_03.ps
echo "0.5 0.7 14 0.0 1 5 Ship" | pstext -R -Jx -O -K >> example_03.ps
echo "0.4 0.4" | psxy -R -Jx -O -K -Sc0.07i -G0 >> example_03.ps
echo "0.5 0.4 14 0.0 1 5 Satellite" | pstext -R -Jx -O >> example_03.ps
trend1d -Fxw -N2r samp_ship.pg >! samp_ship.xw
psxy -R$plotr[1]/$plotr[2]/$plotr[3]/$plotr[4] -JX8i/4i -X2i -Y1.5i -K -Sp0.03i \
   -Ba500f100:"Distance along great circle":/a100f25:"Gravity anomaly (mGal)":WeSn \
   -U/-1.75i/-1.25i/"Example 3e in Cookbook" samp_ship.pg >! example_03d.ps
psxy -R$plotr[1]/$plotr[2]/0/1.1 -JX8i/1.1i -O -Y4.25i -Bf100/a0.5f0.1:"Weight":Wesn -Sp0.03i \
   samp_ship.xw >> example_03d.ps
trend1d -Fxrw -N2r samp_ship.pg | $AWK ’{ if ($3 > 0.6) print $1, $2 }’ | sample1d -Nsamp.x >! \
   samp2_ship.pg
trend1d -Fxrw -N2r samp_sat.pg  | $AWK ’{ if ($3 > 0.6) print $1, $2 }’ | sample1d -Nsamp.x >! \
   samp2_sat.pg
set plotr = ‘cat samp2_sat.pg samp2_ship.pg | minmax -I100/25 -C‘
psxy -R$plotr[1]/$plotr[2]/$plotr[3]/$plotr[4] -JX8i/5i -X2i -Y1.5i -K -W1p \
   -Ba500f100:"Distance along great circle":/a50f25:"Gravity anomaly (mGal)":WeSn \
   -U/-1.75i/-1.25i/"Example 3f in Cookbook" samp2_sat.pg >! example_03e.ps
psxy -R -JX -O -Sp0.03i samp2_ship.pg >> example_03e.ps
paste samp2_ship.pg samp2_sat.pg | cut -f2,4 | spectrum1d -S256 -D1 -W -C >& /dev/null
```

```
psxy spectrum.coh -Ba1f3p:"Wavelength (km)":/a0.25f0.05:"Coherency@+2@+":WeSn -JX-4il/3.75i \
   -R1/1000/0/1 -U/-2.25i/-1.25i/"Example 3g in Cookbook" -P -K -X2.5i -Sc0.07i -G0 \
   -Ey/2 -Y1.5i >! example_03f.ps
echo "3.85 3.6 18 0.0 1 11 Coherency@+2@+" | pstext -R0/4/0/3.75 -Jx -O -K >> example_03f.ps
cat << END >! box.d
2.375    3.75
2.375    3.25
4    3.25
END
psxy -R -Jx -O -K -W1.5p box.d >> example_03f.ps
psxy -Ba1f3p/a1f3p:"Power (mGal@+2@+km)"::."Ship and Satellite Gravity":WeSn spectrum.xpower \
   -St0.07i -O -R1/1000/0.1/10000 -JX-4il/3.75il -Y4.2i -K -Ey/2 >> example_03f.ps
psxy spectrum.ypower -R -JX -O -K -G0 -Sc0.07i -Ey/2 >> example_03f.ps
echo "3.9 3.6 18 0.0 1 11 Input Power" | pstext -R0/4/0/3.75 -Jx -O -K >> example_03f.ps
psxy -R -Jx -O -K -W1.5p box.d >> example_03f.ps
psxy -R -Jx -O -K -G240 -L -W1.5p << END >> example_03f.ps
0.25    0.25
1.4    0.25
1.4    0.9
0.25    0.9
END
echo "0.4 0.7" | psxy -R -Jx -O -K -St0.07i -G0 >> example_03f.ps
echo "0.5 0.7 14 0.0 1 5 Ship" | pstext -R -Jx -O -K >> example_03f.ps
echo "0.4 0.4" | psxy -R -Jx -O -K -Sc0.07i -G0 >> example_03f.ps
echo "0.5 0.4 14 0.0 1 5 Satellite" | pstext -R -Jx -O >> example_03f.ps
\rm -f box.d report samp* *.pg *.extr spectrum.* .gmtcommands
```

The final illustration (Figure 6.3) 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.

6.4 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. This is how to do it:

```
echo '-10     255     0       255' >! zero.cpt
echo '0       100     10      100' >> zero.cpt
grdcontour HI_geoid4.grd -Jm0.45i -E60/30 -R195/210/18/25 -C1 -A5 -G4i -K -P -X1.5i -Y1.5i \
   -U/-1.25i/-1.25i/"Example 4 in Cookbook" >! example_04.ps
pscoast -Jm -E60/30 -R -B2/2NEsw -G0 -O -K >> example_04.ps
echo '205 26 0 0 1.1' | psxyz -Jm -E60/30 -R -SV0.2i/0.5i/0.4ii -W1p -O -K -N >> example_04.ps
echo '205 29.2 36 -90 1 5 N' | pstext -Jm -E60/30 -R -O -K -N >> example_04.ps
grdview HI_topo4.grd -Jm -Jz0.34i -Czero.cpt -E60/30 -R195/210/18/25/-6/4 -N-6/200/200/200 -Qsm -O -K \
   -B2/2/2:"Topo (km)":neswZ -Y2.2i >> example_04.ps
echo '3.25 5.75 60 0.0 33 2 H@#awaiian@# R@#idge' | pstext -R0/10/0/10 -Jx1i -O >> example_04.ps
\rm -f zero.cpt
csh -f job4c.csh
```

The purpose of the color palette file zero.cpt is to have the positive topography mesh painted light gray (the remainder is white). Figure 6.4 shows the complete illustration.

A color version of this figure was used in our first article in EOS Trans. AGU (Oct. 8th, 1991). It was created along similar lines, but instead of a mesh plot we chose a color-coded surface with artificial illumination from a light-source due north. We choose to use the **–Qi** 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). We used **grdgradient** to provide the intensity files. The following script creates the color *PostScript* file. Note that the size of the resulting output file is directly dependent on the square of the dpi chosen for the scanline conversion. A higher value for dpi in **–Qi** would have resulted in a much larger output file. The cpt files were taken from Example 2.

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

```
grdgradient HI_geoid4.grd -A0 -Gg_intens.grd -Nt0.75 -M
grdgradient HI_topo4.grd -A0 -Gt_intens.grd -Nt0.75 -M
grdview HI_geoid4.grd -Ig_intens.grd -JM6.75i -E60/30 -R195/210/18/25 -Cgeoid.cpt -Qi100 -K -X1.5i \
   -Y1.25i -P -U/-1.25i/-1i/"Example 4c in Cookbook" >! example_4c.ps
pscoast -JM -E60/30 -R -B2/2NEsw -G0 -O -K >> example_4c.ps
echo '205 26 0 0 1.1' | psxyz -JM -E60/30 -R -SV0.2i/0.5i/0.4ii -W1p -G255/0/0 -O -K -N \
   >> example_4c.ps
echo '205 29.2 36 -90 1 5 N' | pstext -JM -E60/30 -R -O -K -N >> example_4c.ps
grdview HI_topo4.grd -It_intens.grd -JM -JZ3.4i -Ctopo.cpt -E60/30 -R195/210/18/25/-6/4 \
   -N-6/200/200/200 -Qi100 -O -K -Y2.2i >> example_4c.ps
psbasemap -JM -JZ3.4i -E60/30 -R -Z-6 -O -K -B2/2/2:"Topo (km)":neZ >> example_4c.ps
echo '3.25 5.75 60 0.0 33 2 H@#awaiian@# R@#idge' | pstext -R0/10/0/10 -Jx1i -O >> example_4c.ps
\rm -f *_intens.grd .gmtcommands
```

6.5 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 grdfile 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 grdfiles 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:

```
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.grd
echo '-5    128    5    128' >! gray.cpt
grdgradient sombrero.grd -A225 -Gintensity.grd -Nt0.75
grdview sombrero.grd -JX6i -JZ2i -B5/5/0.5SEwnZ -N-1/255/255/255 -Qs -Iintensity.grd -X1.5i -K \
   -Cgray.cpt -R-15/15/-15/15/-1/1 -E120/30 -U/-1.25i/-0.75i/"Example 5 in Cookbook" >! example_05.ps
echo "4.1 5.5 50 0 33 2 z(r) = cos (2@~p@~r/8) * e@+-r/10@+" | pstext -R0/11/0/8.5 -Jx1i -O \
   >> example_05.ps
\rm -f gray.cpt sombrero.grd intensity.grd .gmtcommands
```

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 6.5.

6.6 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 windrose 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 6.6) was made running this script:

```
psrose fractures.d -A10r -S1.8in -U/-2.25i/-0.75i/"Example 6 in Cookbook" -P -G0 -R0/1/0/360 -X2.5i \
   -K -B0.2g0.2/30g30 >! example_06.ps
pshistogram -Ba2000f1000:"Topography (m)":/a10f5:"Frequency"::,%::."Two types of histograms":WSne \
v3206.t -R-6000/0/0/30 -JX4.8i/2.4i -G200 -O -Y5.5i -X-0.5i -L0.5p -Z1 -W250 >> example_06.ps
\rm -f .gmtcommands
```

6.7 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[2]. We create the final location map (Figure 6.7) with the following script:

```
pscoast -R-50/0/-10/20 -JM9i -K -GP300/26 -Dl -W0.25p -B10 -U"Example 7 in Cookbook" >! example_07.ps
psxy -R -JM -O -K -M fz.xy -W0.5pta >> example_07.ps
$AWK '{print $1-360.0, $2, $3*0.01}' quakes.xym | psxy -R -JM -O -K -H1 -Sci -G255 -W0.25p \
   >> example_07.ps
psxy -R -JM -O -K -M isochron.xy -W0.75p >> example_07.ps
psxy -R -JM -O -K -M ridge.xy -W1.75p >> example_07.ps
psxy -R -JM -O -K -G255 -W1p -A << END >> example_07.ps
-14.5    15.2
-2    15.2
-2    17.8
-14.5    17.8
END
psxy -R -JM -O -K -G255 -W0.5p -A << END >> example_07.ps
-14.35    15.35
-2.15    15.35
-2.15    17.65
-14.35    17.65
END
echo "-13.5 16.5" | psxy -R -JM -O -K -Sc0.08i -G255 -W0.5p >> example_07.ps
echo "-12.5 16.5 18 0 6 5 ISC Earthquakes" | pstext -R -JM -O -K >> example_07.ps
pstext -R -JM -O -S0.75p -G255 << END >> example_07.ps
-43 -5 30 0 1 6 SOUTH
-43 -8 30 0 1 6 AMERICA
-7 11 30 0 1 6 AFRICA
END
\rm -f .gmtcommands
```

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.

6.8 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 topo.grd 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 illustration by running this command:

```
grd2xyz guinea_bay.grd >! $$
psxyz $$ -B1/1/1000:"Topography (m)"::.ETOPO5:WSneZ+ -R-0.1/5.1/-0.1/5.1/-5000/0 \
   -P -JM5i -JZ6i -E200/30 -So0.0833333ub-5000 -U"Example 8 in Cookbook" -W0.25p -G240 -K >! \
   example_08.ps
echo '0.1 4.9 24 0 1 9 This is the surface of cube' | pstext -R -JM -JZ -Z0 -E200/30 -O \
   >> example_08.ps
\rm -f $$ .gmtcommands
```

The output can be viewed in Figure 6.8.

6.9 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

[2]These data are available on CD-ROM from NGDC (www.ngdc.noaa.gov).

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 files *.xys, the ridge in ridge.xy, and all the fracture zones are stored in the multiple segment file fz.xy. We extract the profile id (which is the first part of the file name for each profile) and the last point in each of the track files to construct an input file for **pstext** that will label each profile with the track number. We know the profiles trend approximately N40°E 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, paste this file with the tracks.lis file, and use ***$AWK*** to create the format needed for **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:

```
pswiggle track_*.xys -R185/250/-68/-42 -U"Example 9 in Cookbook" -K -Jm0.13i -Ba10f5 -G0 -Z2000 \
   -W0.25p -S240/-67/500/@~m@~rad >! example_09.ps
psxy -R -Jm -O -K ridge.xy -W1.25p >> example_09.ps
psxy -R -Jm -O -K -M fz.xy -W0.5pta >> example_09.ps
if (-e tmp) then
    \rm -f tmp
endif
foreach file (track_*.xys)    # Make label file
    tail -1 $file >>! tmp
end
ls track_*.xys | $AWK -F. '{print $2}' >! tracks.lis
paste tmp tracks.lis | $AWK '{print $1, $2, 10, 50, 1, 7, $4}' | pstext -R -Jm -D-0.05i/-0.05i -O \
   >> example_09.ps
\rm -f tmp tracks.lis .gmtcommands
```

The output shows the sea-surface slopes along 42 descending Geosat tracks in the Eltanin and Udintsev fracture zone region in a Mercator projection (Figure 6.9).

6.10 A geographical bar graph plot

Our next and perhaps silliest example presents a three-dimensional bargraph plot showing the geographic distribution of the membership in the American Geophysical Union (AGU). The input data was taken from the 1991 AGU member directory and added up to give total members per continent. We decide to plot a 3-D column centered on each continent with a height that is proportional to the logarithm of the membership. A $\log_{10}$-scale is used since the memberships vary by almost 3 orders of magnitude. We choose a plain linear projection for the basemap and add the columns and text on top. Our script reads:

```
pscoast -R-180/180/-90/90 -JX8i/5id -Dc -G0 -E200/40 -K -U"Example 10 in Cookbook" >! example_10.ps
psxyz agu.d -R-180/180/-90/90/1/100000 -JX -JZ2.5il -So0.3ib1 -G140 -W0.5p -O -K -E200/40 \
   -B60g60/30g30/a1p:Memberships:WSneZ >> example_10.ps
$AWK '{print $1-10, $2, 20, 0, 0, 7, $3}' agu.d | pstext -R-180/180/-90/90 -JX -O -K -E200/40 -G255 \
   -S0.5p >> example_10.ps
echo "4.5 6 30 0 5 2 AGU 1991 Membership Distribution" | pstext -R0/11/0/8.5 -Jx1i -O \
   >> example_10.ps
\rm -f .gmtcommands
```

with the result presented in Figure 6.10.

6.11 Making a 3-D RGB color cube

In this example we generate a series of 6 color images, arranged in the shape of a cross, that 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 method for generating the 6 color faces utilizes ***$AWK*** in two steps. First, a *z*-grid is composed which is 256 by 256 with *z*-values increasing in a planar fashion from 0 to 65535. This *z*-grid is common to all six faces. The color variations are generated by creating a different color palette for each face using

the supplied ***$AWK*** script rgb_cube.awk. This script generates a "cpt" file appropriate for each face using arguments for each of the three color components. The arguments specify if that component (r,g,b) is to be held fixed at 0 or 255, is to vary in x, or is to vary in y. If the color is to increase in x or y, a lower case x or y is specified; if the color is to decrease in x or y, an upper case X or Y is used. Here is the shell script and accompanying ***$AWK*** script to generate the RGB cube:

```
grdmath -I1 -R0/255/0/255 Y 256 MUL X ADD = rgb_cube.grd
gmtset TICK_LENGTH 0 COLOR_MODEL rgb
pstext -R0/8/0/11 -Jx1i < /dev/null -P -U"Example 11 in Cookbook" -K >! example_11.ps
$AWK -f rgb_cube.awk r=x g=y b=255 < /dev/null >! rgb_cube.cpt
grdimage rgb_cube.grd -Crgb_cube.cpt -JX2.5i/2.5i -R0/255/0/255 -K -O -X2i -Y4.5i -B256wesn \
   >> example_11.ps
$AWK -f rgb_cube.awk r=255 g=y b=X < /dev/null >! rgb_cube.cpt
grdimage rgb_cube.grd -Crgb_cube.cpt -JX -K -O -X2.5i -B256wesn >> example_11.ps
$AWK -f rgb_cube.awk r=x g=255 b=Y < /dev/null >! rgb_cube.cpt
grdimage rgb_cube.grd -Crgb_cube.cpt -JX -K -O -X-2.5i -Y2.5i -B256wesn >> example_11.ps
psxy -W0.25pto -JX -R -K -O -X2.5i << END >> example_11.ps
0 0
20 20
20 235
0 255
END
psxy -W0.25pto -JX -R -K -O -X-2.5i -Y2.5i << END >> example_11.ps
0 0
20 20
235 20
255 0
END
psxy -W0.25pto -JX -R -K -O -X-2.5i -Y-2.5i << END >> example_11.ps
255 0
235 20
235 235
255 255
END
$AWK -f rgb_cube.awk r=0 g=y b=x < /dev/null >! rgb_cube.cpt
grdimage rgb_cube.grd -Crgb_cube.cpt -JX -K -O -Y-2.5i -B256wesn >> example_11.ps
$AWK -f rgb_cube.awk r=x g=0 b=y < /dev/null >! rgb_cube.cpt
grdimage rgb_cube.grd -Crgb_cube.cpt -JX -K -O -X2.5i -Y-2.5i -B256wesn >> example_11.ps
pstext -JX -R -G255 -K -O << END >> example_11.ps
10 10 14 0 Times-BoldItalic 1 GMT 3
END
psxy -W0.25pto -JX -R -K -O -X2.5i << END >> example_11.ps
0 0
20 20
20 235
0 255
END
psxy -W0.25pto -JX -R -K -O -X-5i << END >> example_11.ps
255 0
235 20
235 235
255 255
END
$AWK -f rgb_cube.awk r=x g=Y b=0 < /dev/null >! rgb_cube.cpt
grdimage rgb_cube.grd -Crgb_cube.cpt -JX -K -O -X2.5i -Y-2.5i -B256wesn >> example_11.ps
psxy -W0.25pto -JX -R -K -O -X2.5i << END >> example_11.ps
0 0
20 20
20 235
0 255
END
psxy -W0.25pto -JX -R -O -X-5i << END >> example_11.ps
255 0
235 20
235 235
255 255
END
\rm -f rgb_cube.cpt rgb_cube.grd .gmtcommands
```

The ***$AWK*** script rgb_cube.awk is as follows:

```
END{
  z=-.5;
  if(r=="X" || g=="X" || b=="X"){
    xl=255; xr=0; xd=-255;
  }else{
```

```
    xl=0; xr=255; xd=255;
  }
  if(r=="Y" || g=="Y" || b=="Y"){
    yb=255; yt=-1; yd=-1;
  }else{
    yb=0; yt=256; yd=1;
  }
  for(y=yb; y!=yt; y+=yd){
    x=xl;
    if(r=="x" || r=="X"){
      if(g=="y" || g=="Y"){
    printf("%7.1f %3d %3d %3d " ,z,x,y,b);
    x+=xd; z+=256;
    printf("%7.1f %3d %3d %3d\n",z,x,y,b);
      }else{
    printf("%7.1f %3d %3d %3d " ,z,x,g,y);
    x+=xd; z+=256;
    printf("%7.1f %3d %3d %3d\n",z,x,g,y);
      }
    }else if(g=="x" || g=="X"){
      if(r=="y" || r=="Y"){
    printf("%7.1f %3d %3d %3d " ,z,y,x,b);
    x+=xd; z+=256;
    printf("%7.1f %3d %3d %3d\n",z,y,x,b);
      }else{
    printf("%7.1f %3d %3d %3d " ,z,r,x,y);
    x+=xd; z+=256;
    printf("%7.1f %3d %3d %3d\n",z,r,x,y);
      }
    }else{
      if(r=="y" || r=="Y"){
    printf("%7.1f %3d %3d %3d " ,z,y,g,x);
    x+=xd; z+=256;
    printf("%7.1f %3d %3d %3d\n",z,y,g,x);
      }else{
    printf("%7.1f %3d %3d %3d " ,z,r,y,x);
    x+=xd; z+=256;
    printf("%7.1f %3d %3d %3d\n",z,r,y,x);
      }
    }
  }
  exit;
}
```

The cube can be viewed in Figure 6.11

6.12 Optimal triangulation of data

Our next example (Figure 6.12) 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 gridded files but still are able to contour and image the data. We use a color palette table topo.cpt (supplied with the script data separately). The script becomes:

```
triangulate table_5.11 -M >! net.xy
psxy -R0/6.5/-0.2/6.5 -JX3.06i/3.15i -B2f1WSNe -M net.xy -W0.5p -P -K -Y4.65i >! example_12.ps
psxy table_5.11 -R -JX -O -K -Sc0.12i -G255 -W0.25p >> example_12.ps
$AWK '{print $1, $2, 6, 0, 0, 6, NR-1}' table_5.11 | \
   pstext -R -JX -O -K >> example_12.ps
psxy -R -JX -B2f1eSNw -M net.xy -W0.5p -O -K -X3.25i >> example_12.ps
psxy -R -JX -O -K table_5.11 -Sc0.03i -G0 >> example_12.ps
$AWK '{printf "%g %s 6 0 0 5 %g\n", $1, $2, $3}' table_5.11 | pstext -R -JX -O -K -W255o \
   -C0.01i/0.01i -D0.08i/0i -N >> example_12.ps
set z = `minmax table_5.11 -C -I25`
makecpt -Cjet -T$z[5]/$z[6]/25 >! topo.cpt
pscontour -R -JX table_5.11 -B2f1WSne -W0.75p -Ctopo.cpt -L0.25pta -G1i -X-3.25i -Y-3.65i -O -K \
   -U"Example 12 in Cookbook" >> example_12.ps
pscontour -R -JX table_5.11 -B2f1eSnw -Ctopo.cpt -I -X3.25i -O -K >> example_12.ps
echo "3.16 8 30 0 1 2 Delaunay Triangulation" | pstext -R0/8/0/11 -Jx1i -O -X-3.25i >> example_12.ps
\rm net.xy topo.cpt .gmtcommands
```

6.13 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 grdfiles. 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:

```
grdmath -R-2/2/-2/2 -I0.1 X Y R2 NEG EXP X MUL = z.grd
grdmath z.grd DDX = dzdx.grd
grdmath z.grd DDY = dzdy.grd
grdcontour dzdx.grd -JX3i -B1/1WSne -C0.1 -A0.5 -K -P -G2i/10 -S4 -T0.1i/0.03i \
   -U"Example 13 in Cookbook" >! example_13.ps
grdcontour dzdy.grd -JX -B1/1WSne -C0.05 -A0.2 -O -K -G2i/10 -S4 -T0.1i/0.03i -X3.45i >> example_13.ps
grdcontour z.grd -JX -B1/1WSne -C0.05 -A0.1 -O -K -G2i/10 -S4 -T0.1i/0.03i -X-3.45i -Y3.45i \
   >> example_13.ps
grdcontour z.grd -JX -B1/1WSne -C0.05 -O -K -G2i/10 -S4 -X3.45i >> example_13.ps
grdvector dzdx.grd dzdy.grd -I0.2 -JX -O -K -Q0.03i/0.1i/0.09in0.25i -G0 -S5i >> example_13.ps
echo "3.2 3.6 40 0 6 2 z(x,y) = x * exp(-x@+2@+-y@+2@+)" | pstext -R0/6/0/4.5 -Jx1i -O -X-3.45i \
   >> example_13.ps
\rm -f z.grd dzdx.grd dzdy.grd .gmtcommands
```

A **pstext** call to place a header finishes the plot (Figure 6.13.

6.14 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 (table 5.11 from example 12). 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 gridded 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 6.14):

```
gmtset GRID_PEN 0.25pta
psxy table_5.11 -R0/7/0/7 -JX3.06i/3.15i -B2f1WSNe -Sc0.05i -G0 -P -K -Y6.45i >! example_14.ps
$AWK '{printf "%g %s 6 0 0 5 %g\n", $1+0.08, $2, $3}' table_5.11 | pstext -R -JX -O -K -N \
   >> example_14.ps
blockmean table_5.11 -R0/7/0/7 -I1 >! mean.xyz
psbasemap -R0.5/7.5/0.5/7.5 -JX -O -K -B0g1 -X3.25i >> example_14.ps
psxy -R0/7/0/7 -JX -B2f1eSNw mean.xyz -Ss0.05i -G0 -O -K >> example_14.ps
$AWK '{printf "%g %s 6 0 0 5 %g\n", $1+0.1, $2, $3}' mean.xyz | pstext -R -JX -O -K -W255o \
   -C0.01i/0.01i -N >> example_14.ps
surface mean.xyz -R -I1 -Gdata.grd
grdcontour data.grd -JX -B2f1WSne -C25 -A50 -G3i/10 -S4 -O -K -X-3.25i -Y-3.55i >> example_14.ps
psxy -R -JX mean.xyz -Ss0.05i -G0 -O -K >> example_14.ps
grdtrend data.grd -N10 -Ttrend.grd
grdcontour trend.grd -JX -B2f1wSne -C25 -A50 -G3i/10 -S4 -O -K -X3.25i >> example_14.ps
project -C0/0 -E7/7 -G0.1 >! track
psxy -R -JX track -W1pto -O -K >> example_14.ps
grdtrack track -Gdata.grd | cut -f3,4 >! data.d
grdtrack track -Gtrend.grd | cut -f3,4 >! trend.d
psxy 'minmax data.d trend.d -I0.5/25' -JX6.3i/1.4i data.d -W1p -O -K -X-3.25i -Y-1.9i -B1/50WSne \
   >> example_14.ps
psxy -R -JX trend.d -W0.5pta -O -U"Example 14 in Cookbook" >> example_14.ps
\rm mean.xyz track *.grd *.d .gmt*
```

6.15 Gridding, contouring, and masking of unconstrained areas

This example (Figure 6.15) 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 allthough 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 landmasses 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 gridded files using **grdinfo**.

```
gmtconvert ship.xyz -bo >! ship.b
set region = ‘minmax ship.b -I1 -bi3‘
nearneighbor $region -I10m -S40k -Gship.grd ship.b -bi3
set info = ‘grdinfo -C -M ship.grd‘
grdcontour ship.grd -JM3i -P -B2WSne -C250 -A1000 -G2i -K -U"Example 15 in Cookbook" >! example_15.ps
blockmedian $region -I10m ship.b -bi3 -bo >! ship_10m.b
surface $region -I10m ship_10m.b -Gship.grd -bi3
psmask $region -I10m ship.b -JM -O -K -T -G220 -bi3 -X3.6i >> example_15.ps
grdcontour ship.grd -JM -B2WSne -C250 -L-8000/0 -A1000 -G2i -O -K >> example_15.ps
psmask $region -I10m ship_10m.b -bi3 -JM -B2WSne -O -K -X-3.6i -Y3.75i >> example_15.ps
grdcontour ship.grd -JM -C250 -A1000 -L-8000/0 -G2i -O -K >> example_15.ps
psmask -C -O -K >> example_15.ps
grdclip ship.grd -Sa-1/NaN -Gship_clipped.grd
grdcontour ship_clipped.grd -JM -B2WSne -C250 -A1000 -L-8000/0 -G2i -O -K -X3.6i >> example_15.ps
pscoast $region -JM -O -K -G150 -W0.25p >> example_15.ps
echo $info[12] $info[13] | psxy -R -JM -O -K -Sa0.15i -W1p >> example_15.ps
echo "-0.3 3.6 24 0 1 CB Gridding with missing data" | pstext -R0/3/0/4 -Jx1i -O -N >> example_15.ps
\rm -f ship.b ship_10m.b ship.grd ship_clipped.grd
```

6.16 Gridding of data, continued

pscontour (for contouring) and **triangulate** (for gridding) use the simplest method of interpolating data: a Delaunay triangulation (see Example 12) 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 Example 15).

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 6.16). 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, patterns, and a "skip slice" request for $700 < z < 725$.

```
gmtset MEASURE_UNIT INCH ANOT_FONT_SIZE 9
pscontour -R0/6.5/-0.2/6.5 -Jx0.45i -P -K -Y5.5i -Ba2f1WSne table_5.11 -Cex16.cpt -I \
   >! example_16.ps
echo "3.25 7 18 0 4 CB pscontour (triangulate)" | pstext -R -Jx -O -K -N >> example_16.ps
surface table_5.11 -R -I0.1 -Graws0.grd
grdview raws0.grd -R -Jx -Ba2f1WSne -Cex16.cpt -Qs -O -K -X3.5i >> example_16.ps
echo "3.25 7 18 0 4 CB surface (tension = 0)" | pstext -R -Jx -O -K -N >> example_16.ps
surface table_5.11 -R -I0.1 -Graws5.grd -T0.5
grdview raws5.grd -R -Jx -Ba2f1WSne -Cex16.cpt -Qs -O -K -Y-3.75i -X-3.5i >> example_16.ps
echo "3.25 7 18 0 4 CB surface (tension = 0.5)" | pstext -R -Jx -O -K -N >> example_16.ps
triangulate table_5.11 -Grawt.grd -R -I0.1 > /dev/null
grdfilter rawt.grd -Gfiltered.grd -D0 -Fc1
grdview filtered.grd -R -Jx -Ba2f1WSne -Cex16.cpt -Qs -O -K -X3.5i >> example_16.ps
echo "3.25 7 18 0 4 CB triangulate @~\256@~ grdfilter" | pstext -R -Jx -O -K -N >> example_16.ps
echo "3.2125 7.5 32 0 4 CB Gridding of Data" | pstext -R0/10/0/10 -Jx1i -O -K -N -X-3.5i >> example_16.ps
psscale -D3.21/0.35/5/0.25h -Cex16.cpt -O -U"Example 16 in Cookbook" -Y-0.75i >> example_16.ps
\rm -f *.grd .gmtcommands
```

6.17 Images clipped by coastlines

This example demonstrates how **pscoast** can be used to set up clippaths 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 clippaths, 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 6.17):

```
grd2cpt india_geoid.grd -Crainbow >! geoid.cpt
grdgradient india_geoid.grd -Nt1 -A45 -Gindia_geoid_i.grd
grdimage india_geoid.grd -Iindia_geoid_i.grd -JM6.5i -Cgeoid.cpt -P -K -U"Example 17 in Cookbook" \
   >! example_17.ps
pscoast -R60/90/-10/25 -JM -O -K -Dl -Gc >> example_17.ps
echo "-10000 150 10000 150" >! gray.cpt
grdgradient india_topo.grd -Nt1 -A45 -Gindia_topo_i.grd
grdimage india_topo.grd -Iindia_topo_i.grd -JM -Cgray.cpt -O -K >> example_17.ps
pscoast -R -JM -O -K -Q -B10f5:."Clipping of Images": >> example_17.ps
pstext -R -JM -O -M -W255O0.5p -D-0.1i/0.1i << EOF >> example_17.ps
> 90 -10 12 0 4 RB 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.
EOF
\rm -f geoid.cpt gray.cpt *_i.grd .gmt*
```

6.18 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[3] 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.

```
gmtset ELLIPSOID Sphere
echo "-142.65    56.25" >! pratt.d
makecpt -Crainbow -T-60/60/10 -Z >! grav.cpt
```

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

```
grdgradient AK_gulf_grav.grd -Nt1 -A45 -GAK_gulf_grav_i.grd
grdimage AK_gulf_grav.grd -IAK_gulf_grav_i.grd -JM5.5i -Cgrav.cpt -B2f1 -P -K -X1.5i -Y5.85i >! example_18.ps
pscoast -R-149/-135/52.5/58 -JM -O -K -Di -G160 -W0.25p >> example_18.ps
psscale -D2.75i/-0.4i/4i/0.15ih -Cgrav.cpt -B20f10/:mGal: -O -K >> example_18.ps
$AWK '{print $1, $2, 12, 0, 1, "LB", "Pratt"}' pratt.d | pstext -R -JM -O -K -D0.1i/0.1i \
   >> example_18.ps
$AWK '{print $1, $2, 0, 200, 200}' pratt.d | psxy -R -JM -O -K -SE -W0.25p >> example_18.ps
grdcontour AK_gulf_grav.grd -JM -C20 -B2f1WSEn -O -K -Y-4.85i -U/-1.25i/-0.75i/"Example 18 in Cookbook" \
   >> example_18.ps
grdcontour AK_gulf_grav.grd -JM -C10 -L49/51 -O -K -Dsm -Wc0.75p/0/255/0 >> example_18.ps
pscoast -R -JM -O -K -Di -G160 -W0.25p >> example_18.ps
$AWK '{print $1, $2, 0, 200, 200}' pratt.d | psxy -R -JM -O -K -SE -W0.25p >> example_18.ps
\rm -f sm_*[0-9].xyz    # Only consider closed contours
cat << EOF >! center.awk
BEGIN {
    x = 0
    y = 0
    n = 0
}
{
    x += \$1
    y += \$2
    n++
}
END {
    print x/n, y/n
}
EOF
\rm -f centers.d
foreach file (sm_*.xyz)
    $AWK -f center.awk $file >>! centers.d
end
gmtselect -R -JM -C200/pratt.d centers.d >! $$
psxy $$ -R -JM -O -K -SC0.04i -G255/0/0 -W0.25p >> example_18.ps
psxy -R -JM -O -K -ST0.1i -G255/255/0 -W0.25p pratt.d >> example_18.ps
grdmath -R -I2m -F -142.65 56.25 GDIST = mask.grd
grdclip mask.grd -Sa200/NaN -Sb200/1 -Gmask.grd
grdmath AK_gulf_grav.grd mask.grd MUL = tmp.grd
set info = 'grdvolume tmp.grd -C50 -Sk'
psxy -R -JM -A -O -K -L -W0.75p -G255 << EOF >> example_18.ps
-148.5 52.75
-140.5    52.75
-140.5    53.75
-148.5    53.75
EOF
pstext -R -JM -O << EOF >> example_18.ps
-148 53.08 14 0 1 LM Areas: $info[2] km@+2@+
-148 53.42 14 0 1 LM Volumes: $info[3] mGal\264km@+2@+
EOF
\rm -f $$ grav.cpt sm_*.xyz *_i.grd tmp.grd mask.grd pratt.d center* .gmt*
```

Our illustration is presented in Figure 6.18.

6.19 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** clippaths 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. The final panel repeats the top panel except that the land and sea images have changed places (Figure **??**).

```
gmtset COLOR_MODEL rgb
grdmath -R-180/180/-90/90 -I1 -F Y COSD 2 POW = lat.grd
grdmath -R-180/180/-90/90 -I1 -F X = lon.grd
echo "0 255 255 255 1 0 0 255" >! lat.cpt
makecpt -Crainbow -T-180/180/60 -Z >! lon.cpt
grdimage lat.grd -JI0/6.5i -Clat.cpt -P -K -Y7.5i -B0 >! example_19.ps
pscoast -R -JI -O -K -Dc -A5000 -Gc >> example_19.ps
grdimage lon.grd -JI -Clon.cpt -O -K >> example_19.ps
pscoast -R -JI -O -K -Q >> example_19.ps
pscoast -R -JI -O -K -Dc -A5000 -W0.25p >> example_19.ps
```

```
echo "0 20 32 0 1 CM FIRST INTERNATIONAL" | pstext -R -JI -O -K -G255/0/0 -S0.5p >> example_19.ps
echo "0 -10 32 0 1 CM GMT CONFERENCE" | pstext -R -JI -O -K -G255/0/0 -S0.5p >> example_19.ps
echo "0 -30 18 0 1 CM Honolulu, Hawaii, April 1, 2000" | pstext -R -JI -O -K -G0/255/50 -S0.25p \
   >> example_19.ps
pscoast -R -JI -O -K -Dc -A5000 -Gp100/86:F255/0/0B255/255/0 -Sp100/7:F255/0/0B0/0/0 -B0 -Y-3.25i \
   >> example_19.ps
echo "0 15 32 0 1 CM SILLY USES OF" | pstext -R -JI -O -K -G50/255/50 -S0.5p >> example_19.ps
echo "0 -15 32 0 1 CM GMT COLOR PATTERNS" | pstext -R -JI -O -K -G255/0/255 -S0.5p >> example_19.ps
grdimage lon.grd -JI -Clon.cpt -O -K -Y-3.25i -B0 -U"Example 19 in Cookbook" >> example_19.ps
pscoast -R -JI -O -K -Dc -A5000 -Gc >> example_19.ps
grdimage lat.grd -JI -Clat.cpt -O -K >> example_19.ps
pscoast -R -JI -O -K -Q >> example_19.ps
pscoast -R -JI -O -K -Dc -A5000 -W0.25p >> example_19.ps
echo "0 20 32 0 1 CM FIRST INTERNATIONAL" | pstext -R -JI -O -K -G255/0/0 -S0.5p >> example_19.ps
echo "0 -10 32 0 1 CM GMT CONFERENCE" | pstext -R -JI -O -K -G255/0/0 -S0.5p >> example_19.ps
echo "0 -30 18 0 1 CM Honolulu, Hawaii, April 1, 2000" | pstext -R -JI -O -G0/255/50 -S0.25p \
   >> example_19.ps
\rm -f l*.grd l*.cpt .gmt*
```

6.20 Custom map 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 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 and arcs of circles. This is how your symbol will look when a size of 1 inch is chosen. Figure 6.20 illustrates a new symbol we will call volcano.

(2) After designing the symbol we will encode it using a simple set of rules. In our case we describe our volcano using no more than 4 possible constructs:

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
x_0 y_0 r $\Delta\alpha$ **C** [**–G***fill*] [**–W***pen*]	Draw single circle of radius r around x_0, y_0 in steps of $\Delta\alpha^\circ$
x_0 y_0 r α_1 α_2 $\Delta\alpha$ **A**	Draw arc segment of radius r from angle α_1 to α_2 every $\Delta\alpha^\circ$

The optional **–G** and **–W** can be used to hardwire the color fill and pen for segments. By default the segments are painted based on the values of the command line settings.

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

```
-0.5    -0.5    M
-0.2    0    D
-0.1    0.173205081    0.2    240    300    2    A
0    0    D
0.3    -0.5    D
-0.05    0.15    0.1    5    C
0.15    0.3    0.075    5    C
0.325    0.4    0.05    10    C
0.45    0.45    0.025    10    C
```

The values refer to positions and dimensions illustrated in Figure 6.20 above. (3) Given a proper definition file we use the supplied ***$AWK***-script make_symbol to create a custom ***$AWK***-script that will read locations and sizes of the desired symbols and replace them with line-drawings like the one in the figure but translated and scaled accordingly. The output is a multi-segment file (see Appendix B) which may be plotted 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 ***$AWK***-script volcano.awk and **psxy**. Note that you must first transform the coordinates using **mapproject** prior to running the ***$AWK***-script. Without further discussion we also make a definition for a multi- colored bulls-eye symbol:

0.50
0.25
0.00
-0.25
-0.50

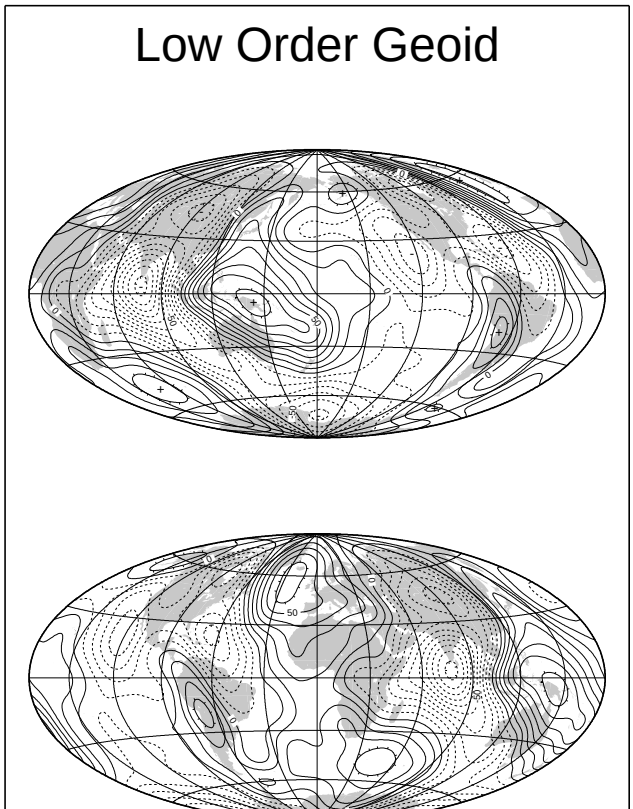
```
0    -0.7    M    -W0.5p/255/0/0p
0    0.7    D
-0.7    0    M    -W0.5p/255/0/0p
0.7    0    D
0    0    0.45    5    C    -Gp0/12
0    0    0.45    5    C    -W0.25p
0    0    0.35    5    C    -G255/255/0 -W0.25p
0    0    0.25    5    C    -Gp0/9
0    0    0.25    5    C    -W0.25p
0    0    0.15    5    C    -G255/255/0 -W0.25p
0    0    0.05    5    C    -G255 -W0.25p
```

Here is our final map script:

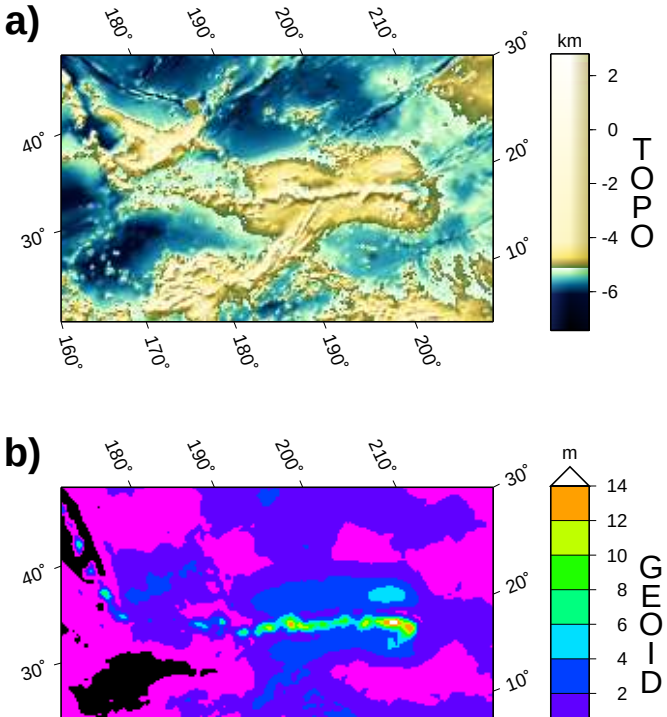
```
$AWK -f make_symbol < volcano.def >! volcano.awk
$AWK -f make_symbol < bullseye.def >! bullseye.awk
cat << EOF >! hotspots.d
55.5    -21.0    0.25
63.0    -49.0    0.25
-12.0    -37.0    0.25
-28.5    29.34    0.25
48.4    -53.4    0.25
155.5    -40.4    0.25
-155.5    19.6    0.5
-138.1    -50.9    0.25
-153.5    -21.0    0.25
-116.7    -26.3    0.25
-16.5    64.4    0.25
EOF
pscoast -R0/360/-90/90 -JR180/9i -B60/30:."Hotspot Islands and Cities": -G0/150/0 -S200/200/255 \
   -Dc -A5000 -K -U"Example 20 in Cookbook" >! example_20.ps
mapproject -R0/360/-90/90 -JR -Di hotspots.d | $AWK -f volcano.awk | psxy -R0/9/0/9 -Jx1i -O -K \
   -M -W0.25p -G255/0/0 -L >> example_20.ps
cat << EOF >! cities.d
286    40.45    0.8
31.15    30.03    0.8
115.49    -31.58    0.8
EOF
mapproject -R0/360/-90/90 -JR -Di cities.d | $AWK -f bullseye.awk | psxy -R0/9/0/9 -Jx -O \
   -M >> example_20.ps
\rm -f volcano.awk bullseye.awk hotspots.d cities.d
```

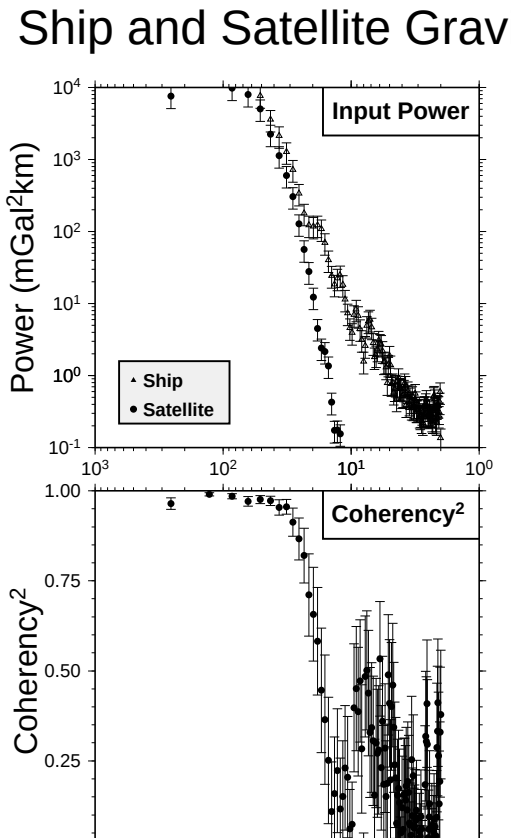
which produces the plot in Figure **??**.

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, unlike the built-in symbols in GMT, can be used with the built-in patterns (Appendix E). Other approaches are also possible, of course.

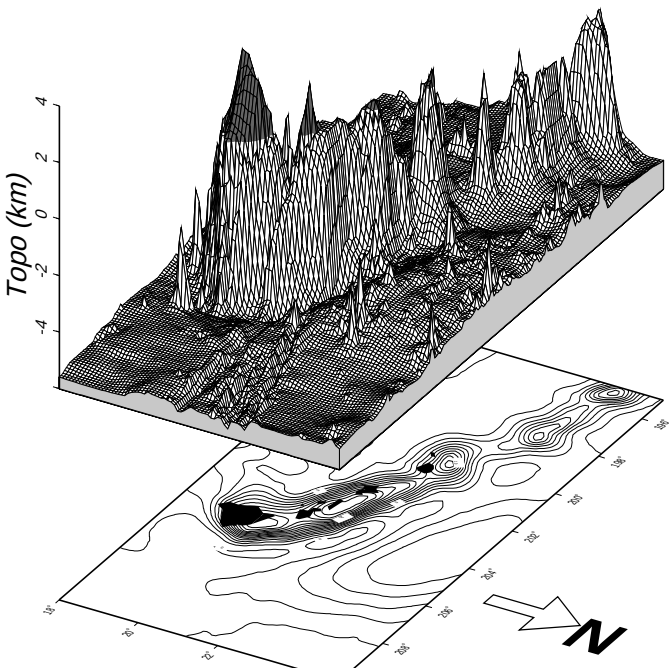

Low Order Geoid

HAWAIIAN TOPO AND GEOID

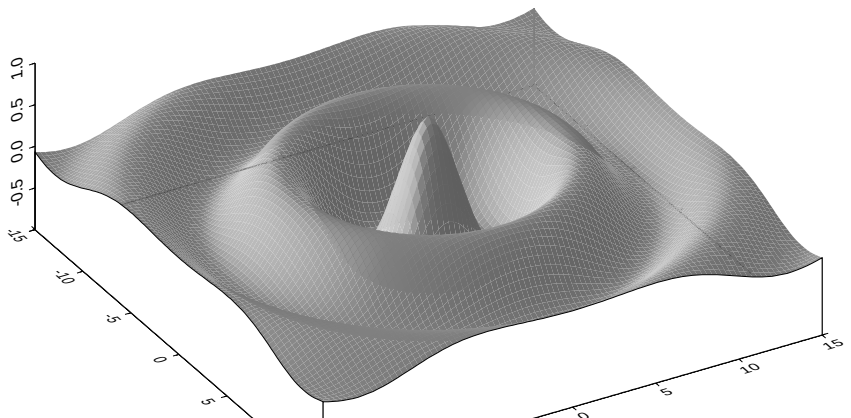


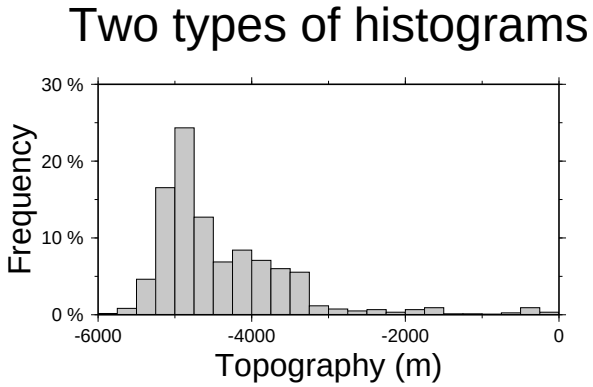

Ship and Satellite Gravity
Input Power
Power (mGal²km)
Ship
Satellite
Coherency²
Coherency²

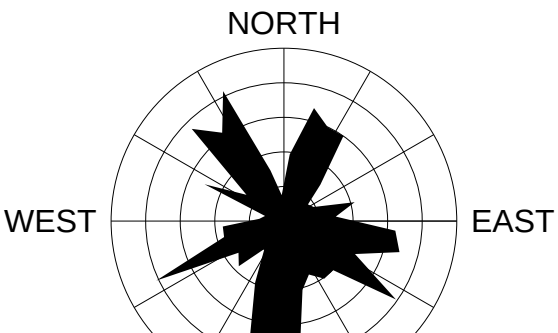
HAWAIIAN RIDGE

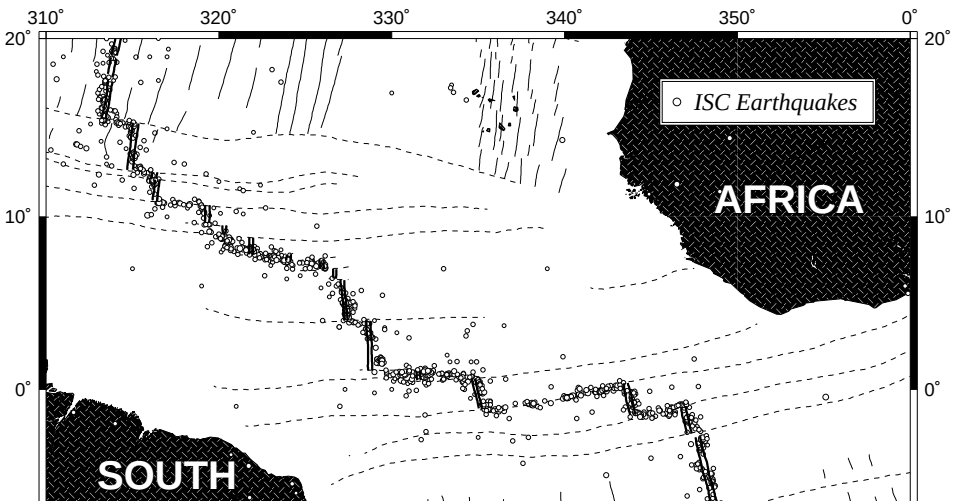


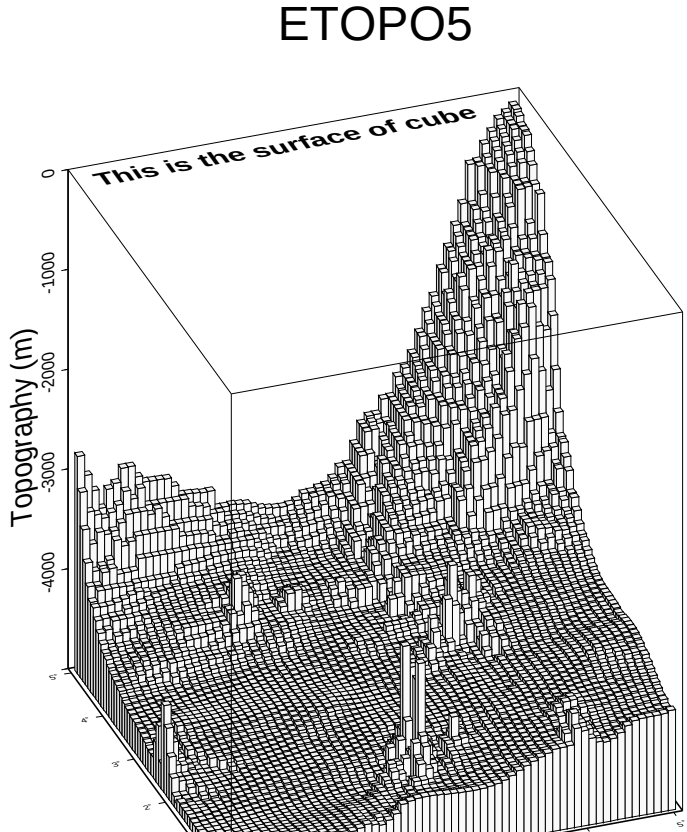
$$z(r) = \cos(2\pi r/8) * e^{-r/10}$$

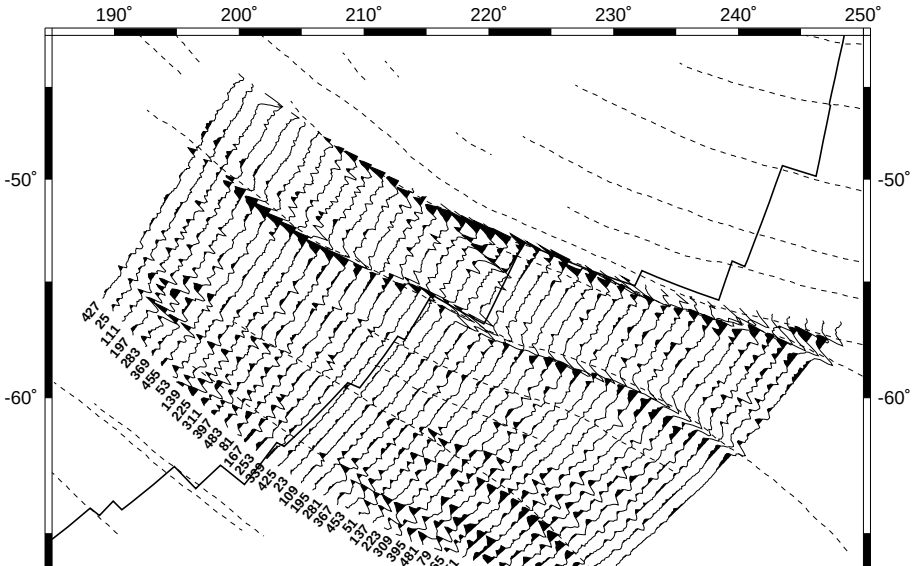



Two types of histograms
30 %
20 %
10 %
0 %
Frequency
-6000
-4000
-2000
0
Topography (m)

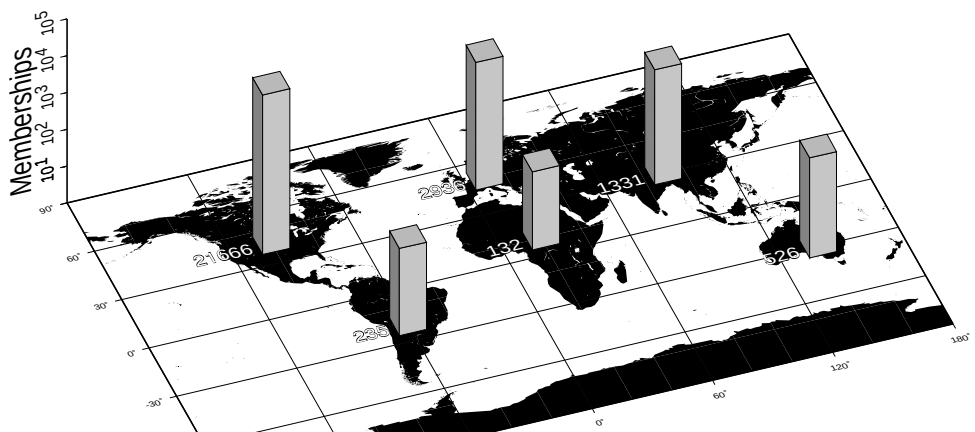

NORTH
WEST
EAST

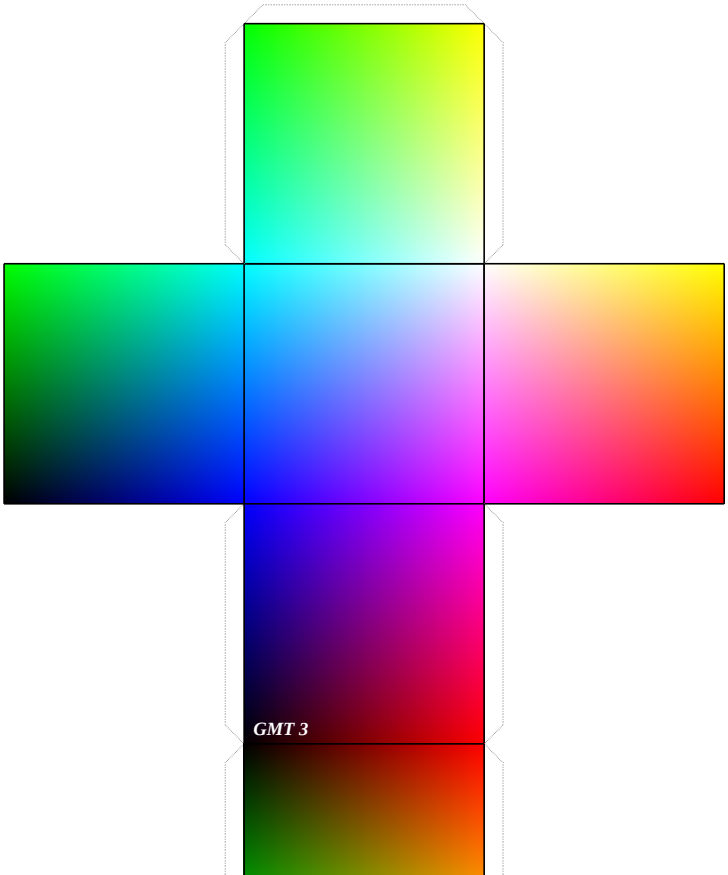

310˚
320˚
330˚
340˚
350˚
0˚
20˚
10˚
0˚
ISC Earthquakes
AFRICA
SOUTH

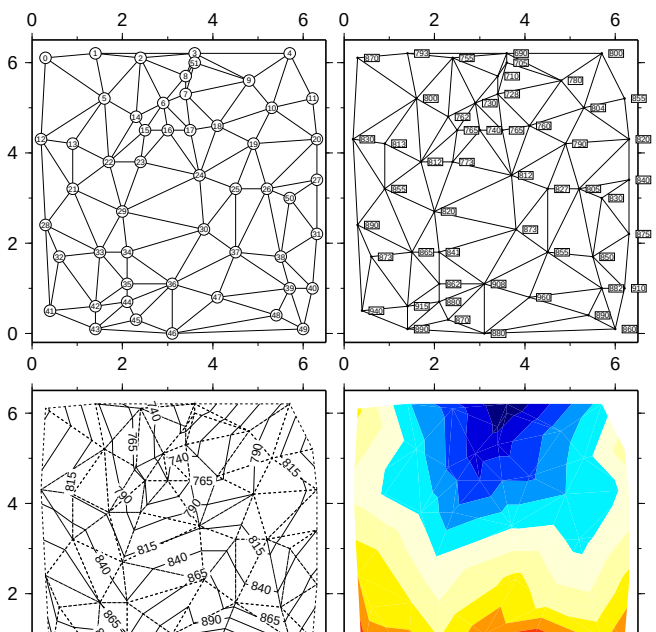

ETOPO5
This is the surface of cube
Topography (m)
0
-1000
-2000
-3000
-4000


190˚
200˚
210˚
220˚
230˚
240˚
250˚
-50˚
-60˚

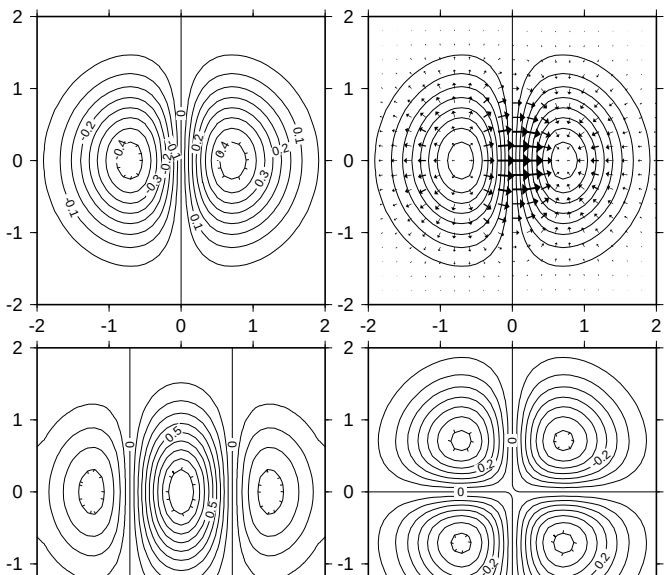
AGU 1991 Membership Distribution

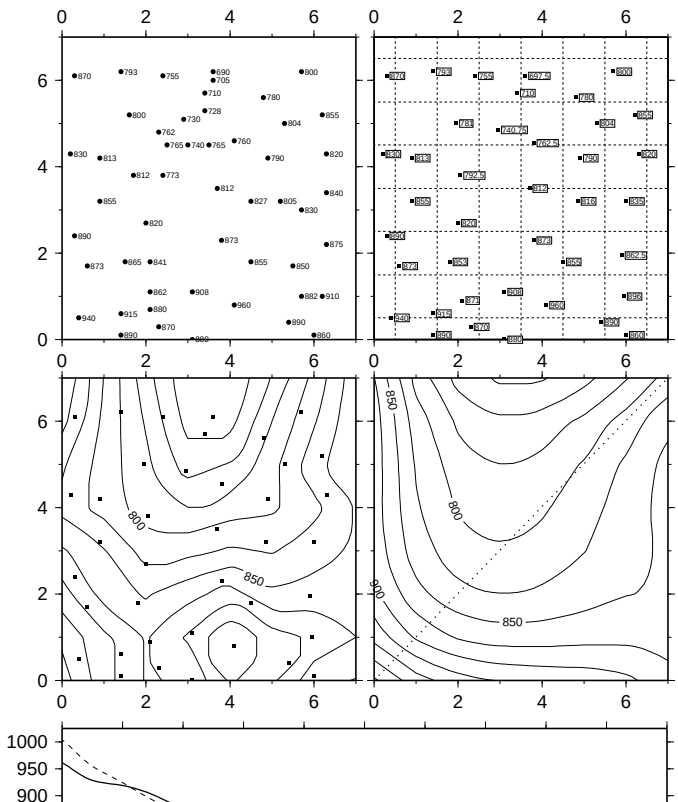


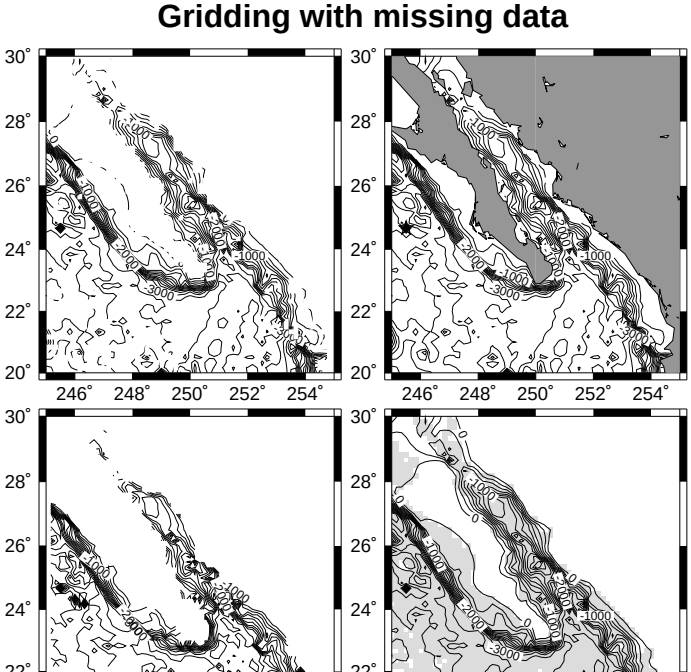

GMT 3

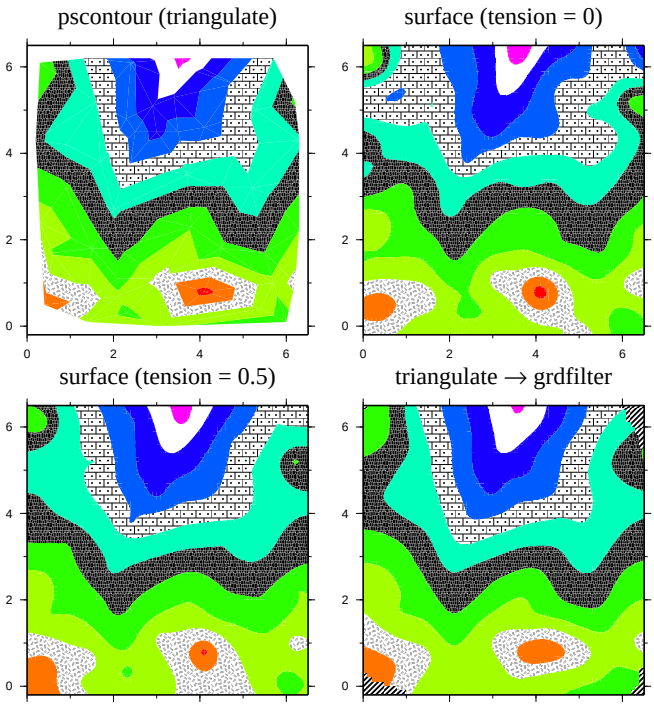

Delaunay Triangulation

$$z(x,y) = x * exp(-x^2-y^2)$$

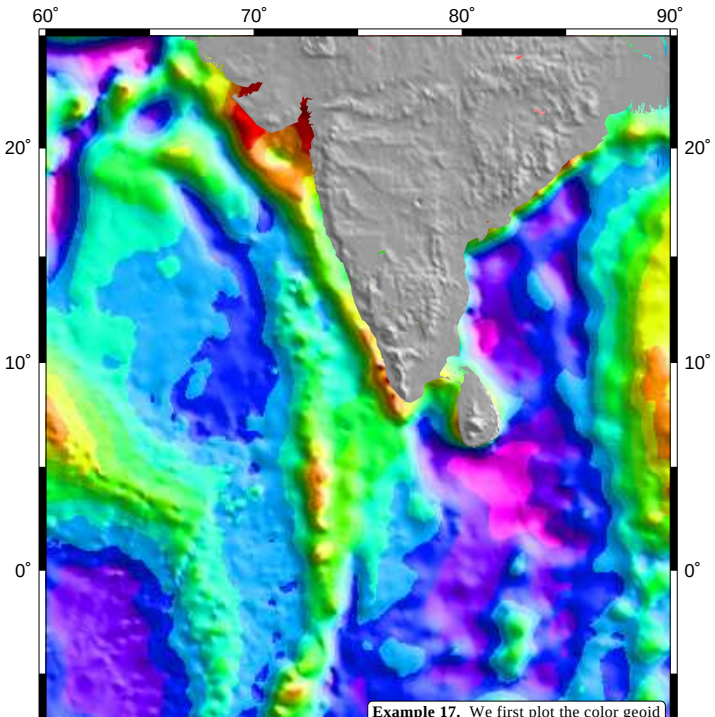



0
2
4
6
800
850
900
1000
950
900

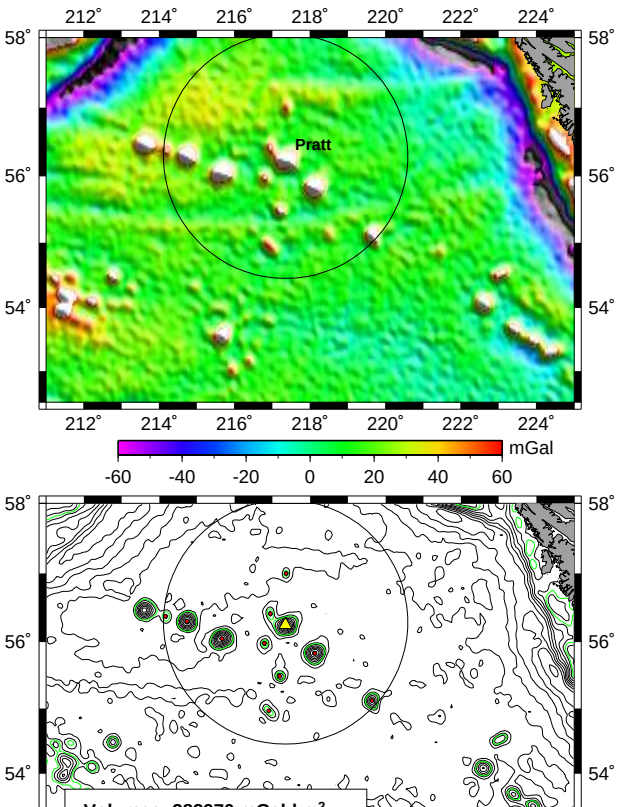

Gridding with missing data
30˚
28˚
26˚
24˚
22˚
20˚
246˚ 248˚ 250˚ 252˚ 254˚
-1000
-2000
-3000
0


Gridding of Data
pscontour (triangulate)
surface (tension = 0)
surface (tension = 0.5)
triangulate → grdfilter
0
2
4
6

Clipping of Images



Example 17. We first plot the color geoid


212˚
214˚
216˚
218˚
220˚
222˚
224˚
58˚
56˚
54˚
Pratt
mGal
-60
-40
-20
0
20
40
60

7. Mailing lists, updates, and bug reports

Most public-domain (and even commercial) software comes with bugs, and the speed with which such bugs are detected and removed depends to a large degree on the willingness of the user community to report these to us in a useful manner. When your car breaks down, simply telling the mechanic that it doesn't work will hardly speed up the repair or cut back costs! Therefore, we ask that if you detect a bug, first make sure that it in fact is a bug and not a user error. Then, send us email about the problem. Be sure to include all the information necessary for us to recreate the situation in which the bug occurred. This will include the full command line used and, if possible, the data file used by the program. Send the bug-reports to gmt-bugs@hawaii.edu. We will try to fix bugs as soon as our schedules permit and inform users about the bug and availability of updated code (See Appendix D).

Two electronic mailing lists are available to which users may subscribe. gmt-group@hawaii.edu and is primarily a way for us to notify the users when bugs have been fixed or when new updates have been installed in the ftp directory (See Appendix D). We also maintain another list (gmt-help@hawaii.edu) which interested users may subscribe to. It basically provides a forum for GMT users to exchange ideas and ask questions about GMT usage, installation and portability, etc. Please use this utility rather than sending questions directly to us personally. We hope you appreciate that we simply do not have time to be everybody's personal GMT tutor.

The electronic mailing lists are maintained automatically by a program. To subscribe to one or both of the lists, send a message to listproc@hawaii.edu containing the command(s):

subscribe gmt-group <your full name, not email address>

subscribe gmt-help <your full name, not email address>

(Do not type the angular brackets <>). You may also register electronically via the GMT home web page[1]. For information on what commands you may send, send a message containing the word help. You must interact with the listserver to be added to or removed from the mailing lists! We strongly recommend that you at least subscribe to gmt-group since this is how we can notify you of future updates and bug-fixes. Most new users will also benefit from having the other forum (gmt-help) as they struggle to realign their sense of logic with that of GMT. While anybody may post messages to gmt-help, access to gmt-group is restricted to minimize net traffic. Any message sent to gmt-group will be intercepted by the GMT manager who will determine if the message is important enough to cause thousands of mailtools to go BEEP. Communication with other GMT users should go via gmt-help. Finally, all GMT information is provided online at the main GMT home page in Hawaii, i.e., gmt.soest.hawaii.edu. Changes to GMT will also be posted on this page. The main GMT page has links to the official GMT ftp sites around the world.

[1] gmt.soest.hawaii.edu

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); see Appendix D). 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 necessarilly 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:

A.1 dbase: gridded data extractor

This package contains **grdraster** which you can use to extract data from global gridded data sets such as those available from NGDC. We have used it to prepare some of the grids in the examples (Chapter 6). You can also customize it to read your own data sets. The package is maintained by the GMT developers.

A.2 gshhs: GSHHS data extractor

This package contains ***gshhs*** which you can use to extract shoreline polygons from the Global Self-consistent Hierarchical High-resolution Shorelines (GSHHS) available separately from NGDC[1] or the GSHHS home page[2] (GSHHS is the polygon data base from which the GMT coastlines derive). It also contains ***gshhs_dp*** for cleverly decimating a shoreline, and ***gshhstograss*** to convert shoreline segments to the GRASS database format. The package is maintained by Paul Wessel.

A.3 imgsrc: gridded altimetry extractor

This package consists of the program **img2mercgrd** to extract subsets of the global gravity and predicted topography solutions derived from satellite altimetry[3]. The package is maintained by Walter Smith[4].

A.4 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 is maintained by Kurt Feigl[5] and Genevieve Patau[6].

A.5 mex: Matlab–GMT interface

Here you will find the mex files ***grdinfo***, ***grdread***, and ***grdwrite***, which can be used in Matlab to read and write GMT grdfiles. The package originated with David Sandwell, UCSD, and was subsequently modified by Paul Wessel and Phil Sharfstein, UCSB. It is now mainained by Paul Wessel.

[1] http://www.ngdc.noaa.gov/mgg/shorelines/gshhs.html
[2] http://www.soest.hawaii.edu/wessel/gshhs/gshhs.html
[3] For data bases, see http://topex.ucsd.edu/marine_grav/mar_grav.html.
[4] walter@raptor.grdl.noaa.gov
[5] Kurt.Feigl@cnes.fr
[6] patau@ipgp.jussieu.fr

A.6 mgg: MGD77 extractor and plotting tools

This package holds the programs **binlegs**, **dat2gmt**, **gmt2dat**, **gmtinfo**, **gmtlegs**, **gmtlist**, **gmtpath**, **gmttrack**, and **mgd77togmt**, which can be used to maintain, access, extract data from, and plot marine geophysical data files (MGD-77[7]). The package is maintained by the GMT developers.

A.7 misc: posters, patterns, and digitizing

At the moment, this package contains the programs **psmegaplot** which you can use to make large posters using a simple laserwriter, **makepattern** which generates raster patterns from GMT 3.0 icon files, and **gmtdigitize** which provides a GMT interface to a digitizing tablet via a serial port. The package is maintained by Paul Wessel.

A.8 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. The package is maintained by Tim Henstock[8].

A.9 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, **hotspotter** which generates CVA grids based on seamount locations and a set of absolute plate motion stage poles, and **originator**, which associates seamounts with the most likely hotspot origins. The package is maintained by Paul Wessel.

A.10 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. It is a new generation of tools intended to replace the old "X_SYSTEM" crossover tools (below). The package is maintained by Paul Wessel.

A.11 x_system: Track crossover error estimation

This package contains the tools **x_edit**, **x_init**, **x_list**, **x_over**, **x_remove**, **x_report**, **x_setup**, **x_solve_dc_drift**, and **x_update**. Collectively, they make up the old "XSYSTEM" crossover tools. This package with remain in the GMT supplemental archive until ***x2sys*** is complete. The package is maintained by Paul Wessel.

A.12 xgrid: visual editor for grdfiles

The package contains an X11 editor (***xgridedit***) for visual editing of grdfiles. It was originally developed by Hugh Fisher, CRES, in March 1992 but is now maintained by Lloyd Parkes[9].

[7] These data are available on CD-ROM from NGDC (www.ngdc.noaa.gov).
[8] Timothy.J.Henstock@soc.soton.ac.uk
[9] lloyd@must-have-coffee.gen.nz

B. GMT file formats

B.1 Table data

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

B.1.1 ASCII tables

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

B.1.2 Binary tables

GMT programs also support binary tables to speed up input-output for i/o-intensive tasks like gridding and preprocessing. Files may have no header (hence the **–H** option cannot be used) and all data must either be single or double precision (no mixing allowed). Multiple segment files are allowed (**–M**) and the segment headers are assumed to be records where all the fields equal NaN. Flags appended to **–M** are ignored. The format and number of fields are specified with the **–b** option. Thus, for input you may set **–bi**[**s**][*n*], where **s** designates single precision (default is double) and *n* is the number of fields. For output, use **–bo**[**s**] (the programs know how many columns to write, unless you use **–M** in which case we need to know the number of output columns up front).

B.2 2-D grdfiles

B.2.1 File contents

The default 2-D binary netCDF grdfile in GMT has several attributes. The **grdedit** utility program will allow you to edit parts of the header of an existing grdfile. The attributes listed in Table B.1 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).

GMT version 3 also allows other formats to be read. In addition to the default netCDF format it can use binary floating points, short integers, bytes, and bits, as well as 8-bit Sun rasterfiles (colormap ignored). Additional formats may be used by supplying read/write functions and linking these with the GMT libraries. The source file gmt_customio.c has the information that programmers will need to augment

Parameter	*Description*
int *nx*	Number of nodes in the *x*-dimension
int *ny*	Number of nodes in the y-dimension
int *node_offset*	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 grdfile
char *remark*[160]	Any additional comments
float *z*[nx*ny]	1-D array with *z*-values in scanline format

Table B.1: **GMT** gridded file header record.

GMT to read custom grdfiles. We anticipate that the number of pre-programmed formats will increase as enterprising users implement what they need.

B.2.2 Grid line and Pixel 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 *node_offset* signals how the nodes are laid out. The grid is always defined as the intersections of all x ($x = x_{min}, x_{min} + x_{inc}, x_{min} + 2 \cdot x_{inc}, \ldots, x_{max}$) and y ($y = y_{min}, y_{min} + y_{inc}, y_{min} + 2 \cdot y_{inc}, \ldots, y_{max}$) lines. The two scenarios differ in which area each data point represents. The default registration in GMT is grid line registration. Most programs can handle both types, and for some programs like **grdimage** a pixel registered file makes more sense. Utility programs like **grdsample** and **grdproject** will allow you to convert from one format to the other.

Grid line registration

In this registration, the nodes are centered on the grid line intersections and the data points represent the average value in a cell of dimensions ($x_{inc} \cdot y_{inc}$) centered on the nodes (Figure B.1). In the case of grid line registration the number of nodes are related to region and grid spacing by

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

which for the example in Figure B.1 yields $nx = ny = 4$.

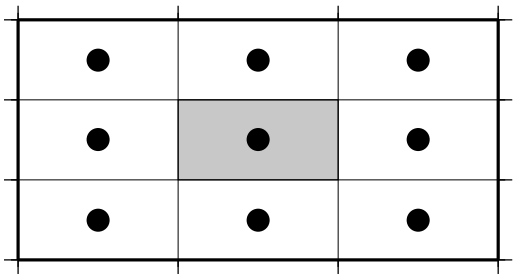
Pixel registration

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 (Figure B.2. In the case of pixel registration the number of nodes are

related to region and grid spacing by

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

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



B.2.3 Boundary Conditions for operations on grids

GMT has the option to specify boundary conditions in some programs that operate on grids (**grdsample –L**; **grdgradient –L**; **grdtrack –L**; **nearneighbor –L**; **grdview –L**). The boundary conditions come into play when interpolating or computing derivatives near the limits of the region covered by the grid. The *default* boundary conditions used are those which are "natural" for the boundary of a minimum curvature interpolating surface. If the user knows that the data are periodic in *x* (and/or *y*), or that the data cover a sphere with *x*,*y* representing *longitude*,*latitude*, then there are better choices for the boundary conditions. Periodic conditions on *x* (and/or *y*) are chosen by specifying *x* (and/or *y*) as the boundary condition flags; global spherical cases are specified using the *g* (geographical) flag. Behavior of these conditions is as follows:

Periodic conditions on *x* indicate that the data are periodic in the distance ($x_{max} - x_{min}$) and thus repeat values after every $N = (x_{max} - x_{min})/x_{inc}$. Note that this implies that in a grid-registered file the values in the first and last columns are equal, since these are located at $x = x_{min}$ and $x = x_{max}$, and there are $N + 1$ columns in the file. This is not the case in a pixel-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.

B.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 B.2.

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

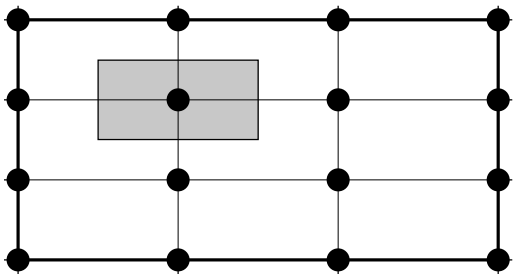
Table B.2: Structure of a Sun rasterfile.

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 B.3.

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[])

Table B.3: Sun macro definitions relevant to rasterfiles.

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



C. Making GMT Encapsulated *PostScript* Files

GMT can produce both freeform *PostScript* files and the more restricted Encapsulated *PostScript* files (EPS). The former is intended to be sent to a printer or *PostScript* previewer, while the latter is indended to be included in another document (but should also be able to print and preview). You control what kind of *PostScript* that GMT produces by manipulating the PAPER_MEDIA parameter (see the **gmtdefaults** man page for how this is accomplished). 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***) do not understand these valid instructions and may fail to image the file. If this is your situation you should choose another viewer (we recommend ***ghostview***) or select EPS output instead.

However, there is much confusion over what an EPS file is and if other programs can read it. Much of this has to do with the claim by some software manufacturers that their programs can read and edit EPS files. We used to get much mail from people asking us to let GMT produce EPS files that can be read, e.g., by Adobe ***Illustrator***. This was a limitation of early versions of Adobe ***Illustrator*** and similar programs, not GMT! Since then, Adobe ***Illustrator*** and other programs have improved their abilities to parse freeform *PostScript* such as that produced by GMT.

An EPS file that is to be placed into another application (such as a text document) need 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 GMT makes the *PostScript* files on the fly, often with several overlays, it is not possible to do so accurately. However, GMT does make an effort to ensure that the boundingbox is large enough to contain the entire composite plot[1]. Therefore, if you need 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 yourplot.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.

If you do not want to modify the illustration but just include it in a text document: Many word processors (such as Microsoft ***Word*** and Corel ***WordPerfect***) will let you include a *PostScript* file that you may place but not edit. You won't be able to view the figure on-screen, but it will print correctly. All illustrations in this GMT documentation were GMT-produced *PostScript* files that were converted to EPS files using ***ps2epsi*** and then included into a LaTeX document.

These examples do not constitute endorsements of the products mentioned above; they only represent our limited experience with the problem. For other solutions and further help, please post messages to gmt-help@hawaii.edu.

[1] In contrast, regular GMT *PostScript* files simply have a %%BoundingBox that equal the size of the chosen paper.

D. Availability of GMT and related code

All the source code, support data, *PostScript* and HTML versions of all documentation, and *UNIX* (including HTML) manual pages can be obtained by anonymous ftp from several mirror sites. We also maintain a GMT page on the World Wide Web (http://gmt.soest.hawaii.edu); See this page for installation directions which allow for a simplified, automatic install procedure (for a CD-R solution, see http://www.geowareonline.com.)

The GMT tar archives are available both in ***gzip*** and ***bzip2*** format. If neither of these utilities are installed on your system, you should know that the former program is available from GNU[1] while the latter can be obtained from Cygnus[2]. ***bzip2*** compresses much better than ***gzip***: for example, the full resolution coastline database is only ∼29 Mb in ***bzip2*** format compared to a hefty ∼44 Mb in ***gzip***. These files have the .bz2 suffix.

The GMT archives are as follows:

GMT_progs.tar.{**gz,bz2**} Contains all GMT source code, cpt files, and *PostScript* patterns.

GMT_share.tar.{**gz,bz2**} Contains the intermediate, low, and crude resolutions of the coastline database. Required with _progs.tar for minimal setup needed to run GMT.

GMT_doc.tar.{**gz,bz2**} Contains all GMT documentation (man pages, PostScript versions of both the Cookbook and Technical Reference and the tutorial, and the short course material).

GMT_web.tar.{**gz,bz2**} Contains all HTML versions of all GMT documentation (man pages, Cookbook and Technical Reference, and tutorial).

GMT_full.tar.{**gz,bz2**} Contains the optional full-resolution coastline database.

GMT_high.tar.{**gz,bz2**} Contains the optional high-resolution coastline database.

GMT_scripts.tar.{**gz,bz2**} Contains all the shellscripts and support data used in the Cookbook section.

GMT_suppl.tar.{**gz,bz2**} Contains several programs written by us and GMT users elsewhere. (See Appendix A for more details).

triangle.tar.{**gz,bz2**} Contains John Shewchuck's fast Delaunay triangulation routine which may be installed with GMT. (See the copyright information first if you are a commercial user).

All of the above archives are also available as Windows ZIP archives, e.g., **GMT_progs.zip**. For Windows users who do not want to compile themselves, there are two zip files with Win32 executables:

GMT_exe.zip ZIP archive with all main GMT executables.

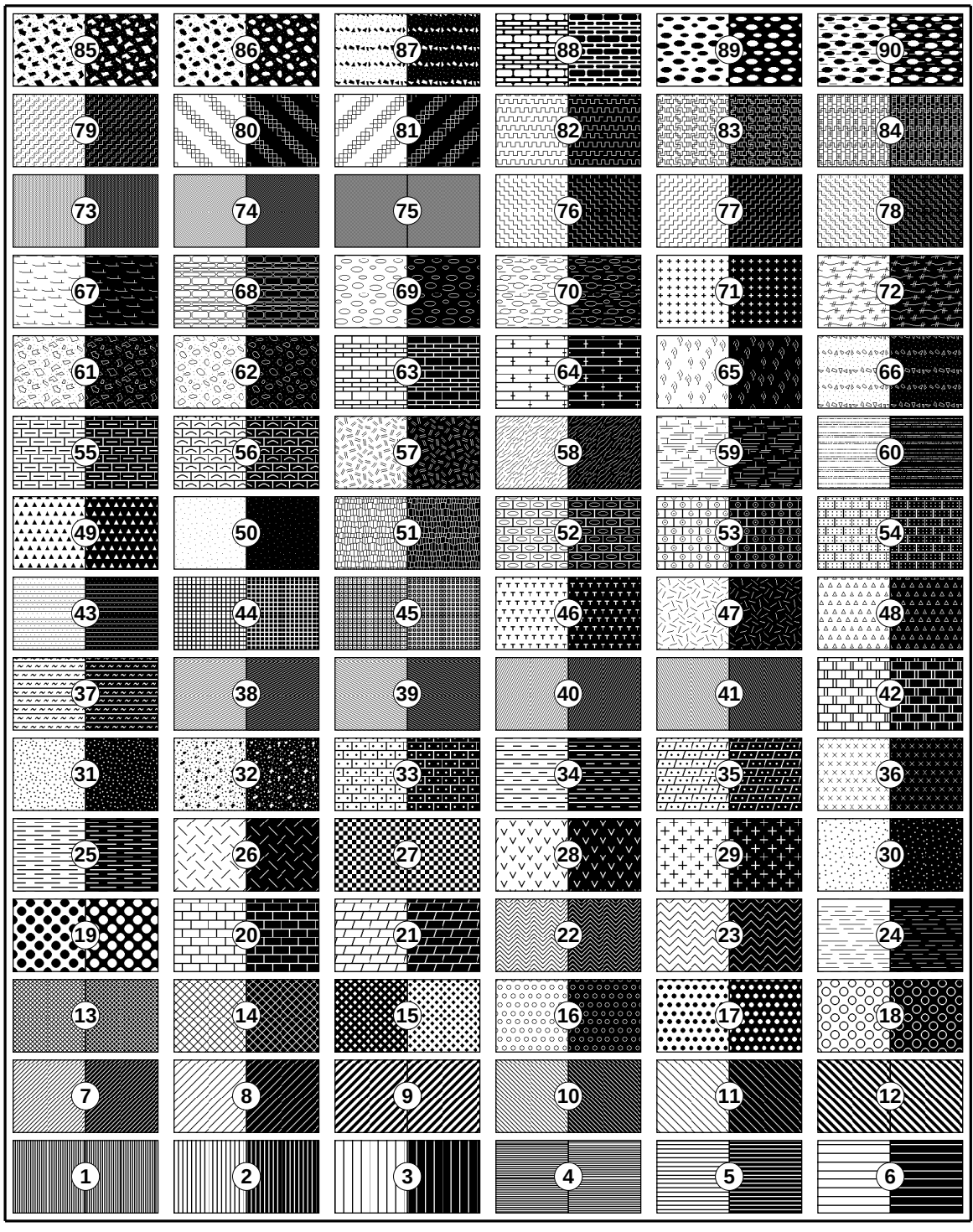
GMT_suppl_exe.zip ZIP archive with all supplemental GMT executables.

The netCDF library that makes up the backbone of the grdfile I/O operations can be obtained from Unidata. A compressed tar file can be accessed (in binary mode) from the file netcdf.tar.Z in the anonymous FTP directory of unidata.ucar.edu. The software distribution includes a *PostScript* file of the netCDF User's Guide, and there is also online documentation from their web site. [netcdfgroup@unidata.ucar.edu is available for discussion of the netCDF interface and announcements about netCDF bugs, fixes, and enhancements. To subscribe, send a request to netcdfgroup-adm@unidata.ucar.edu]. Precompiled libraries for WIN32 are also available[3].

[1] www.gnu.org
[2] http://sourceware.cygnus.com/bzip2/index.html
[3] Required with GMT_exe.zip

E. Predefined bit and hachure patterns in GMT

GMT provides 90 different bit and hachure patterns that can be selected with the **–Gp** option in most plotting programs. These patterns are reproduced below at 300 dpi.


85
86
87
88
89
90
79
80
81
82
83
84
73
74
75
76
77
78
67
68
69
70
71
72
61
62
63
64
65
66
55
56
57
58
59
60
49
50
51
52
53
54
43
44
45
46
47
48
37
38
39
40
41
42
31
32
33
34
35
36
25
26
27
28
29
30
19
20
21
22
23
24
13
14
15
16
17
18
7
8
9
10
11
12
1
2
3
4
5
6

F. Chart of octal codes for characters

The characters and their octal codes in the reencoded standard fonts are shown in Figure F.1. The chart for the Symbol (GMT font number 12) character sets are presented in Figure F.2 below. Gray areas signify codes reserved for control characters. The octal code is obtained by appending the column value to the \?? value, e.g., ∂ is \266 in the Symbol font. In order to use all the extended characters you need to set **WANT_EURO_FONT** to true in your .gmtdefaults file[1].

The Pifont ZapfDingbats is available as GMT font number 34 and can be used for special symbols not listed above. The various symbols are illustrated in Figure F.3.

[1] By default, this is true if you chose SI units and false if you chose US units during the installation.

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	{	|	}	~	
\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		¡	¢	£	⁄	¥	*f*	§
\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

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	′	≤	⁄	∞	*f*	♣
\25x	♦	♥	♠	↔	←	↑	→	↓
\26x	°	±	″	≥	×	∝	∂	•
\27x	÷	≠	≡	≈	…	⏐	—	↵
\30x	ℵ	ℑ	ℜ	℘	⊗	⊕	∅	∩
\31x	∪	⊃	⊇	⊄	⊂	⊆	∈	∉
\32x	∠	∇	®	©	™	∏	√	·
\33x	¬	∧	∨	⇔	⇐	⇑	⇒	⇓

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	①	②	③	④	⑤	⑥	⑦	⑧
\31x	⑨	⑩	❶	❷	❸	❹	❺	❻
\32x	❼	❽	❾	❿	➔	→	↔	↕
\33x	➘	➔	➚	→	➔	→	➞	➟

G. *PostScript* fonts used by GMT

GMT uses the standard 35 fonts that come with most *PostScript* laserwriters, as well as 4 Japanese fonts, for a total of 39. 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:

For the special fonts Symbol (12) and ZapfDingbats (34), see the octal charts in Appendix 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 **gmtdefaults**. For direct plotting of text-strings, see the man page for **pstext**.

#	Font Name	#	Font Name
0	Helvetica	19	Bookman-Light
1	**Helvetica-Bold**	*20*	*Bookman-LightItalic*
2	*Helvetica-Oblique*	21	Helvetica-Narrow
3	***Helvetica-BoldOblique***	**22**	**Helvetica-Narrow-Bold**
4	Times-Roman	*23*	*Helvetica-Narrow-Oblique*
5	**Times-Bold**	***24***	***Helvetica-Narrow-BoldOblique***
6	*Times-Italic*	25	NewCenturySchlbk-Roman
7	***Times-BoldItalic***	*26*	*NewCenturySchlbk-Italic*
8	Courier	**27**	**NewCenturySchlbk-Bold**
9	**Courier-Bold**	***28***	***NewCenturySchlbk-BoldItalic***
10	*Courier-Oblique*	29	Palatino-Roman
11	***Courier-BoldOblique***	*30*	*Palatino-Italic*
12	Σψμβολ (Symbol)	**31**	**Palatino-Bold**
13	AvantGarde-Book	***32***	***Palatino-BoldItalic***
14	*AvantGarde-BookOblique*	*33*	*ZapfChancery-MediumItalic*
15	**AvantGarde-Demi**	34	✺❁❒❄✿❇■✵✲❁▼▲ (ZapfDingbats)
16	***AvantGarde-DemiOblique***	35	Ryumin-Light-EUC-H (JPN)
17	**Bookman-Demi**	36	Ryumin-Light-EUC-V (JPN)
18	***Bookman-DemiItalic***	37	GothicBBB-Medium-EUC-H (JPN)
		38	GothicBBB-Medium-EUC-V (JPN)

H. Problems with display of GMT *PostScript*

GMT creates valid (so far as we know) Adobe *PostScript* Level 1. It does not use operators specific to Level 2 and should therefore produce output that will print on old as well as new *PostScript* printers. Sometimes unexpected things happen when GMT output is sent to certain printers or displays. This section lists some things we have learned from experience, and some work-arounds.

H.1 *PostScript* driver bugs

When you try to display a *PostScript* file on a device, such as a printer or your screen, then a program called a *PostScript* device driver has to compute which device pixels should receive which colors (black or white in the case of a simple laser printer) in order to display the file. At this stage, certain device-dependent things may happen. These are not limitations of GMT or *PostScript*, but of the particular display device. The following bugs are known to us based on our experiences:

1. Early versions of the Sun SPARCprinter software caused linewidth-dependent path displacement. We reported this bug and it has been fixed in newer versions of the software. Try using **psxy** to draw $y = f(x)$ twice, once with a thin pen (**–W**1) and once with a fat pen (**–W**10); if they do not plot on top of each other, you have this kind of bug and need new software. The problem may also show up when you plot a mixture of solid and dashed (or dotted) lines of various pen thickness

2. The first version of the HP Laserjet 4M (prior to Aug–93) had bugs in the driver program. The old one was *PostScript* SIMM, part number C2080-60001; the new one is called *PostScript* SIMM, part number C2080-60002. You need to get this one plugged into your printer if you have an HP LaserJet 4M.

3. Apple Laserwriters with the older versions of Apple's *PostScript* driver will give the error "limitcheck" and fail to plot when they encounter a path exceeding about 1000–1500 points. Try to get a newer driver from Apple, but if you can't do that, set the parameter MAX_L1_PATH to 1000–1500 or even smaller in the file src/pslib_inc.h and recompile GMT. The number of points in a *PostScript* path can be arbitrarily large, in principle; GMT will only create paths longer than MAX_L1_PATH if the path represents a filled polygon or clipping path. Line-drawings (no fill) will be split so that no segment exceeds MAX_L1_PATH. This means **psxy –G** will issue a warning when you plot a polygon with more than MAX_L1_PATH points in it. It is then your responsibility to split the large polygon into several smaller segments. If **pscoast** gives such warnings and the file fails to plot you may have to select one of the lower resolution databases The path limitation exemplified by these Apple printers is what makes the higher-resolution coastlines for **pscoast** non-trivial: such coastlines have to be organized so that fill operations do not generate excessively large paths. Some HP *PostScript* cartridges for the Laserjet III also have trouble with paths exceeding 1500 points; they may successfully print the file, but it can take all night!

4. 8-bit color screen displays (and programs which use only 8-bits, even on 24-bit monitors, such as Sun's ***pageview*** under OpenWindows) may not dither cleverly, and so the color they show you may not resemble the color your *PostScript* file is asking for. Therefore, if you choose colors you like on the screen, you may be surprised to find that your plot looks different on the hardcopy printer or film writer. The only thing you can do is be aware of this, and make some test cases on your hardcopy devices and compare them with the screen, until you get used to this effect. (Each hardcopy device is also a little different, and so you will eventually find that you want to tune your color choices for each device.) The rgb color cube in example 11 may help.

5. Some versions of Sun's OpenWindows program ***pageview*** have only a limited number of colors available; the number can be increased somewhat by starting ***openwin*** with the option "`openwin -cubesize large`".

6. Finally, ***pageview*** seem to have problems understanding the `setpagedevice` operator. We recommend you only use ***pageview*** on EPS files or use ***ghostview*** instead.

7. Many color hardcopy devices use CMYK color systems. GMT *PostScript* uses RGB (even if your cpt files are using HSV). The three coordinates of RGB space can be mapped into three coordinates in CMY space, and in theory K (black) is superfluous. But it is hard to get CMY inks to mix into a good black or gray, so these printers supply a black ink as well, hence CMYK. The *PostScript* driver for a CMYK printer should be smart enough to compute what portion of CMY can be drawn in K, and use K for this and remove it from CMY; however, some of them aren't.

8. In early releases of GMT we always used the *PostScript* command `r g b setrgbcolor` to specify colors, even if the color happened to be a shade of gray ($r = g = b$) or black ($r = g = b = 0$). One of our users found that black came out muddy brown when he used ***FreedomOfPress*** to make a Versatec plot of a GMT map. He found that if he used the *PostScript* command `g setgray` (where g is a graylevel) then the problem went away. Apparently, his installation of ***FreedomOfPress*** uses only CMY with the command `setrgbcolor`, and so `0 0 0 setrgbcolor` tries to make black out of CMY instead of K. To fix this, in release 2.1 of GMT we changed some routines in pslib.c to check if ($r = g$ and $r = b$), in which case `g setgray` is used instead of `r g b setrgbcolor`.

9. Recent experience with some Tektronix Phaser printers and with commercial printing shops has shown that this substitution creates problems precisely opposite of the problems our Versatec user has. The Tektronix and commercial (we think it was a Scitex) machines do not use K when you say `0 setgray` but they do when you say `0 0 0 setrgbcolor`. We believe that these problems are likely to disappear as the various software developers make their codes more robust. Note that this is not a fault with GMT: $r = g = b = 0$ means black and should plot that way. Thus, the GMT source code as shipped to you checks whether $r = g$ and $r = b$, in which case it uses `setgray`, else `setrgbcolor`. If your gray tones are not being drawn with K, you have two work-around options: (1) edit the source for pslib.c or (2) edit your *PostScript* file and try using `setrgbcolor` in all cases. The simplest way to do so is to redefine the `setgray` operator to use `setrgbcolor`. Insert the line

   ```
   /setgray {dup dup setrgbcolor} def
   ```

   immediately following the first line in the file (starts with %!PS.)

10. Some color film writers are very sensitive to the brand of film. If black doesn't look black on your color slides, try a different film.

H.2 Resolution and dots per inch

The parameter DOTS_PR_INCH can be set by the user through the .gmtdefaults file or **gmtset**. By default it is equal to the value in the gmt_defaults.h file, which is supplied with 300 when you get GMT from us. This seems a good size for most applications, but should ideally reflect the resolution of your hardcopy device (most laserwriters have at least 300 dpi, hence our default value). GMT computes what the plot should look like in double precision floating point coordinates, and then converts these to integer coordinates at DOTS_PR_INCH resolution. This helps us find out that certain points in a path lie on top of other points, and we can remove these, making smaller paths. Small paths are important for the laserwriter bugs above, and also to make fill operations compute faster. Some users have set their DOTS_PR_INCH to very large numbers. This only makes the *PostScript* output bigger without affecting the appearence of the plot. However, if you want to make a plot which fits on a page at first, and then later magnify this same *PostScript* file to a huge size, the higher DPI is important. Your data may not have the higher resolution but on certain devices the edges of fonts will not look crisp if they are not drawn with an effective resolution of 300 dpi or so. Beware of making an excessively large path. Note that if you change dpi the linewidths produced by your **–W** options will change, unless you have appended **p** for linewidth in points.

H.3 European characters

Note for users of ***pageview*** in Sun OpenWindows: GMT now offers some octal escape sequences to load European alphabet characters in text strings (see Section 4.16). When this feature is enabled, the header on GMT *PostScript* output includes a section defining special fonts. The definition is added to the header whether or not your plot actually uses the fonts.

Users who view their GMT *PostScript* output using ***pageview*** in OpenWindows on Sun computers or user older laserwriters may have difficulties with the European font definition. If your installation of OpenWindows followed a space-saving suggestion of Sun, you may have excluded the European fonts, in which case ***pageview*** will fail to render your plot.

Ask your system administrator about this, or run this simple test: (1) View a GMT *PostScript* file with ***pageview***. If it comes up OK, you will be fine. If it comes up blank, open the "Edit PostScript" button and examine the lower window for error messages. (The European font problem generates lots of error messages in this window). (2) Verify that the *PostScript* file is OK, by sending it to a laserwriter and making sure it comes out. (3) If the *PostScript* file is OK but it chokes ***pageview***, then edit the *PostScript* file, cutting out everything between the lines:

%%%%% START OF EUROPEAN FONT DEFINITION %%%%%
<bunch of definitions>
%%%%% END OF EUROPEAN FONT DEFINITION %%%%%

Now try ***pageview*** on the edited version. If it now comes up, you have a limited subset of OpenWindows installed. If you discover that these fonts cause you trouble, then you can edit your .gmtdefaults file to set WANT_EURO_FONT = FALSE, which will suppress the printing of this definition in the GMT *PostScript* header. With this set to FALSE, you can make output which will be viewable in ***pageview*** without any editing. However, you would have to reset this to TRUE before attempting to use European fonts, and then the output will become un-***pageview***-able again. If you try to concatenate segments of GMT *PostScript* made with and without the European fonts enabled, then you may find that you have problems, either with the definition, or because you ask for something not defined.

H.4 Hints

When making images and perspective views of large amounts of data, the GMT programs can take some time to run, the resulting *PostScript* files can be very large, and the time to display the plot can be long. Fine tuning a plot script can take lots of trial and error. We recommend using **grdsample** to make a low resolution version of the data files you are plotting, and practice with that, so it is faster; when the script is perfect, use the full-resolution data files. We often begin building a script using only **psbasemap** or **pscoast** to get the various plots oriented correctly on the page; once this works we replace the **psbasemap** calls with the actually desired GMT programs.

If you want to make color shaded relief images and you haven't had much experience with it, here is a good first cut at the problem: Set your COLOR_MODEL to HSV using **gmtset**. Use **makecpt** or **grd2cpt** to make a continuous color palette spanning the range of your data. Use the **–Nt** option on **grdgradient**. Try the result, and then play with the tuning of the .gmtdefaults, the cpt file, and the gradient file.

I. Color Space — The final frontier

Beginning with GMT version 2.1.4, "Example 11" was included in the cookbook. The example makes an RGB color cube by a simple ***awk*** script. We wrote a program to compute HSV grids for each face of this cube, and include a version of the cube with HSV contours on it as file contoured_cube.ps in /share.

In this appendix, we are going to try to explain the relationship between the RGB and HSV color systems so as to (hopefully) make them more intuitive. GMT allows users to specify colors in cpt files in either system (colors on command lines, such as pen colors in **–W** option, are always in RGB). GMT uses the HSV system to achieve artificial illumination of colored images (e.g. **–I** option in **grdimage**) by changing the *s* and *v* coordinates of the color. When the intensity is zero, the data are colored according to the cpt file. If the intensity is non-zero, the data are given a starting color from the cptfile but this color (after conversion to HSV if necessary) is then changed by moving (*s*,*v*) toward HSV_MIN_SATURATION, HSV_MIN_VALUE if the intensity is negative, or toward HSV_MAX_SATURATION, HSV_MAX_VALUE if positive. These are defined in the .gmtdefaults 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.

We are going to try to give you a geometric picture of color mixing in HSV from a tour of the RGB cube. The geometric picture is helpful, we think, since HSV are not orthogonal coordinates and not found from RGB by an algebraic transformation. But before we begin traveling on the RGB cube, let us give two formulae, since an equation is often worth a thousand words.

$$\begin{aligned} v &= max(r,g,b) \\ s &= (max(r,g,b) - min(r,g,b))/max(r,g,b) \end{aligned}$$

Note that when $r = g = b = 0$ (black), the expression for *s* gives 0/0; black is a singular point for *s*. The expression for *h* is not easily given without lots of "if" tests, but has a simple geometric explanation. 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 *(r, g, b)* 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 *(r, g, b)* as you look at the cube.

The RGB color cube has six faces. On three of these one of *(r, g, b)* is equal to 0. These three faces meet at the black corner, where $r = g = b = 0$. On these three faces saturation, the S in HSV, has its maximum value; *s* = 1 on these faces. (Accept this definition and ignore the *s* singularity at black for now). Therefore *h* and *v* are contoured on these faces; *h* in gray solid lines and *v* in white dashed lines (*v* ranges from 0 to 1 and is contoured in steps of 0.1).

On the other three faces one of *(r, g, b)* is equal to the maximum value. These three faces meet at the white corner, where $r = g = b = 255$. On these three faces value, the V in HSV, has its maximum value; *v* = 1 on these faces. Therefore *h* and *s* are contoured on these faces; *h* in gray solid lines and *s* in black dashed lines (*s* ranges from 0 to 1 with contours every 0.1).

The three faces where *v* = 1 meet the three faces where $s = 1$ in six edges where both $s = v = 1$ (and at least one of *(r, g, b)* = 0 and at least one of *(r, g, b)* = 255). Trace your finger around these edges, starting at the red point and moving to the yellow point, then on around. You will visit six of the eight corners of the cube, in this order: red ($h = 0$); yellow ($h = 60$); green ($h = 120$); cyan ($h = 180$); blue ($h = 240$); magenta ($h = 300$). 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 (color monitors usually use RGB). The only cube corners you did not visit on this path are the black and white corners. Imagine an axis running through the black and white corners. If you project the RYGCBM edge path onto a plane perpendicular to the black-white axis, the path will look like a hexagon, with RYGCBM at the vertices, every 60° apart. Now we can make a geometric definition of hue: Take a vector from the origin (black point) to any point in the cube; project this vector onto the plane with the RYGCBM hexagon; then hue is the angle this projected vector makes with the R direction on the hexagon. Thus hue is an angle describing

rotation around the black-white axis. Note that by this definition, if a point is on the black-white axis, its *(r, g, b)* vector will project as a point at the center of the hexagon, so its hue is undefined. Points on the black-white axis have $r = g = b$, and they are shades of gray; we will call the black-white axis the gray axis.

Let us call the points where $s = v = 1$ (the points on the RYGCBM path of cube edges) 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 *(r, g, b)* 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) *(r, g, b)* space, but so far we have only looked at (s, v) on the cube faces, not inside it. At interior points, none of *(r, g, b)* 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 *(r, g, b)* 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, because it is the intersection of three planes on which $s = 1$, but it is on a line where $s = 0$; it is a singular point, and we get "0/0" in the above formula. We see also that saturation is a measure of "purity" or "vividness" of the color.

It remains to define value, and the formula above is really the best definition. But if you like our geometric constructions, 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 equation is hue = 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 *(r, g, b)* while keeping the same hue. The effect this has on a color monitor is to make the color shine more brightly, but "brightness" has other meanings in color geometry, so let us say that if you can move in this way, you can make your hue "stronger"; if you are already on a plane where at least one of *(r, g, b)* = 255, then you cannot get a stronger version of the same hue. Thus, v measures 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 $\sqrt{r^2 + g^2 + b^2}/255$.

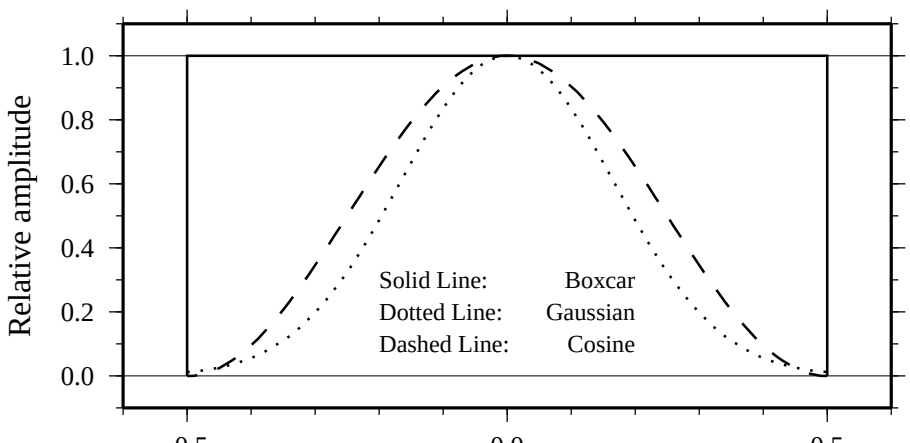
The RGB system is understandable because it is cartesian, and we all learned cartesian coordinates in school. But it doesn't help us create a tint or shade of a color; we cannot say, "We want orange, and a lighter shade of orange, or a less vivid orange". With HSV we can do this, by saying, "Orange must 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, 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.

Well, there you have it, folks. We've been doing GMT for 10 years and all we know about color can be written in about 2 pages. We hope we haven't told you any lies. For more details, you should consult a book about color systems. But as example 11 shows, a lot can be learned by experimenting with GMT tools. Our thanks to John Lillibridge for Example 11.

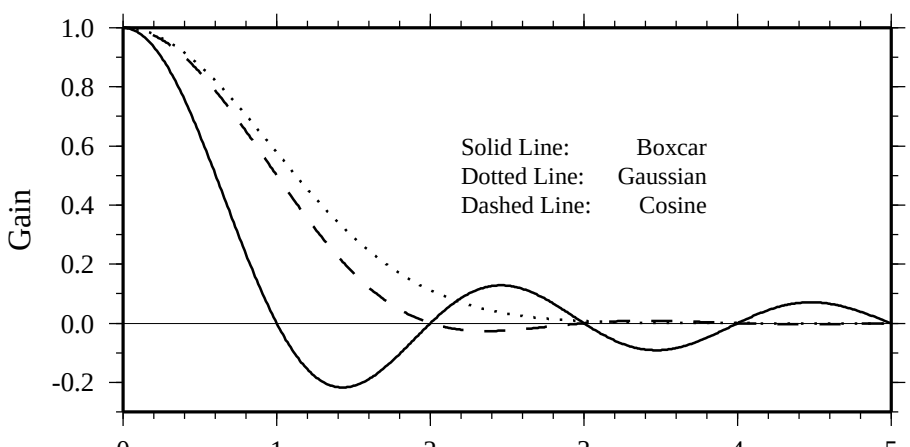
J. Filtering of data in GMT

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 or mode 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 J.1).

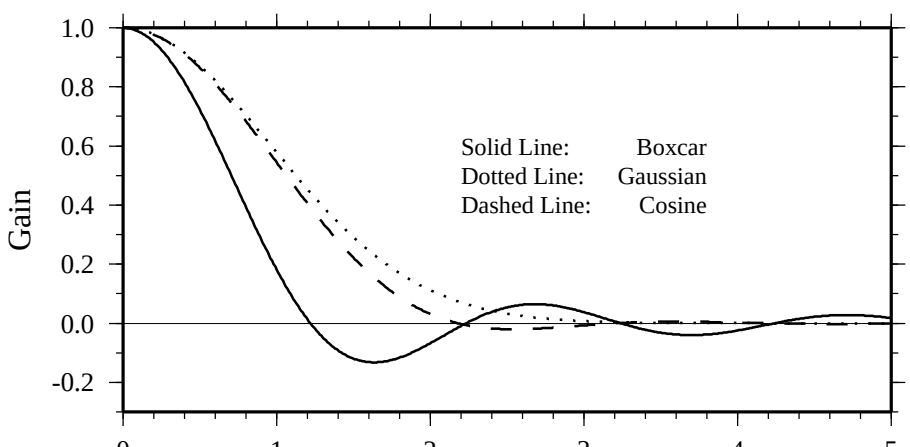

Relative amplitude
1.0
0.8
0.6
0.4
0.2
0.0
Solid Line: Boxcar
Dotted Line: Gaussian
Dashed Line: Cosine

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 J.2 and J.3, 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).


1.0
0.8
0.6
0.4
0.2
0.0
-0.2
Gain
Solid Line: Boxcar
Dotted Line: Gaussian
Dashed Line: Cosine

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.

[1] R. Bracewell, *The Fourier Transform and its Applications*, McGraw-Hill, London, 444p., 1965.


1.0
0.8
0.6
0.4
0.2
0.0
-0.2
Gain
Solid Line: Boxcar
Dotted Line: Gaussian
Dashed Line: Cosine

K. 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.

K.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].

K.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).

K.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. For instance, the reasonable requirement that no coastline should cross another coastline becomes a complicated processing step.

1. 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.

2. 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 ($\sim$25% of WVS).

[1] www.ngdc.noaa.gov

3. 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.

4. 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.

5. 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.

6. 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].

7. 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:

[2]Dougles, 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 2.1.4.

[4]If you need complete polygons in a simpler format, see the article on GSHHS (Wessel, P., and W. H. F. Smith, 1996, A Global, self-consistent, hierarchical, high-resolution shoreline database, *J. Geophys. Res. 101*, 8741–8743).

- 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.

K.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 -2000 km to +2000 km in the longitudinal direction and -1500 km to +1500 km in the latitudinal direction. However, GMT programs expects degrees in the **–R** option that specifies the desired region. Given the chosen map projection we can automate this process by using a simple cshell script that we call ***getbox***:

```
range=‘(echo $2 $4; echo $3 $5) | mapproject $1 -R0/360/-90/90 -I -Fk -C‘
echo $range | awk ’{printf "-R%f/%f/%f/%fr\n",$1,$2,$3,$4}’
```

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 need the geographical coordinates of the four corners of the region rectangle. Again, we automate this task by adding the simple script ***getrect***:

```
(echo $2 $4; echo $3 $4; echo $3 $5; echo $2 $5) | mapproject $1 -R0/360/-90/90 -I -Fk -C
```

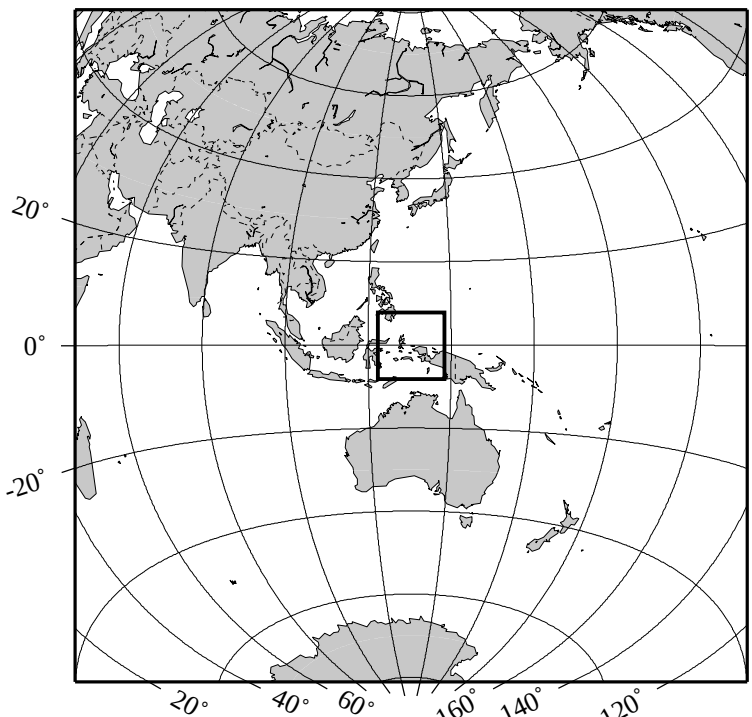
K.4.1 The crude resolution (–Dc)

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 is only 286 Kbytes. The plot is produced by the command (the box indicates the outline of the next map):

```
#!/bin/sh
#    $Id: GMT_App_K_1.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#
gmtset GRID_CROSS_SIZE 0 OBLIQUE_ANOTATION 0 WANT_EURO_FONT FALSE
pscoast ‘./getbox -JE130.35/-0.2/1i -9000 9000 -9000 9000‘ -JE130.35/-0.2/3.5i -P -Dc \
   -A500 -G200 -W0.25p -N1/0.25tap -B20g20WSne -K \
   | egrep -v ’\(80\\312|\(100\\312|\(120\\312|\(140\\312|\(160\\312|\(180\\312’ > GMT_App_K_1.ps
./getrect -JE130.35/-0.2/1i -2000 2000 -2000 2000 | psxy -R -JE130.35/-0.2/3.5i -O -W1.5p -L -A \
   >> GMT_App_K_1.ps
```

Here, the ***egrep*** command was used to remove some longitudinal annotations near the poles that otherwise would overprint[5].

[5] If this is confusing, remove the ***egrep*** command and view the result.


20°
0°
-20°
20°
40°
60°
160°
140°
120°

K.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 876 Kbytes; it is the default resolution in GMT. The plot is generated by the command:

```
#!/bin/sh
#    $Id: GMT_App_K_2.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#
pscoast './getbox -JE130.35/-0.2/1i -2000 2000 -2000 2000' -JE130.35/-0.2/3.5i -P -Dl -A100 -G200 \
   -W0.25p -N1/0.25tap -B10g5WSne -K > GMT_App_K_2.ps
./getrect -JE130.35/-0.2/1i -500 500 -500 500 | psxy -R -JE130.35/-0.2/3.5i -O -W1.5p -L -A \
   >> GMT_App_K_2.ps
```

K.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 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 3.28 Mbytes. The plot is generated by the commands:

```
#!/bin/sh
#    $Id: GMT_App_K_3.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#
pscoast './getbox -JE130.35/-0.2/1i -500 500 -500 500' -JE130.35/-0.2/3.5i -P -Di -A20 -G200 \
   -W0.25p -N1/0.25tap -B2g1WSne -K > GMT_App_K_3.ps
./getrect -JE130.35/-0.2/1i -100 100 -100 100 | psxy -R -JE130.35/-0.2/3.5i -O -W1.5p -L -A \
   >> GMT_App_K_3.ps
```

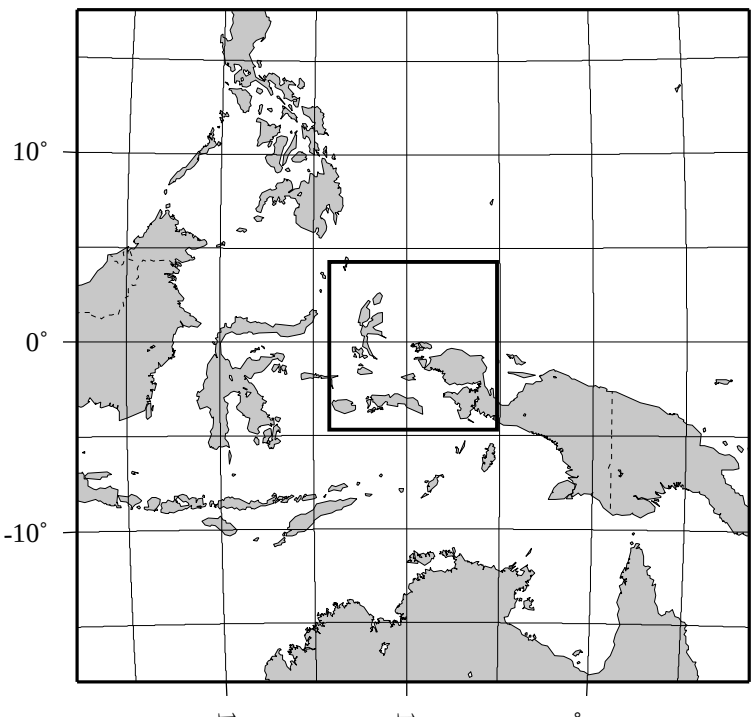
K.4.4 The high resolution (–Dh)

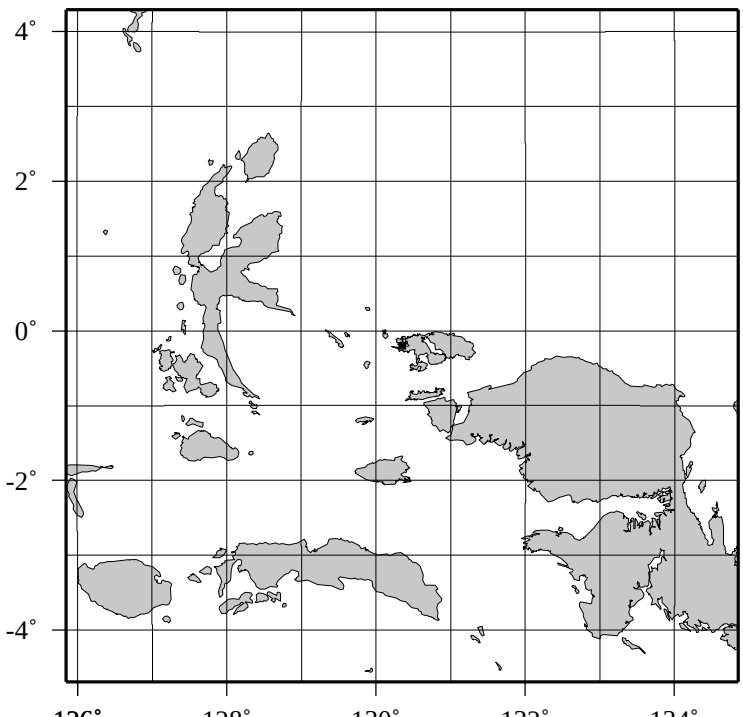
The relentless zooming continues! Now, the edges of the map are all 100 km true distance from the projection center. We step up to the high resolution data set as it is needed to accurately portray the detailed geographic features within the region. Because of the small scale we only ignore features 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 12.2 Mbytes. The map and the final outline box are generated by these commands:

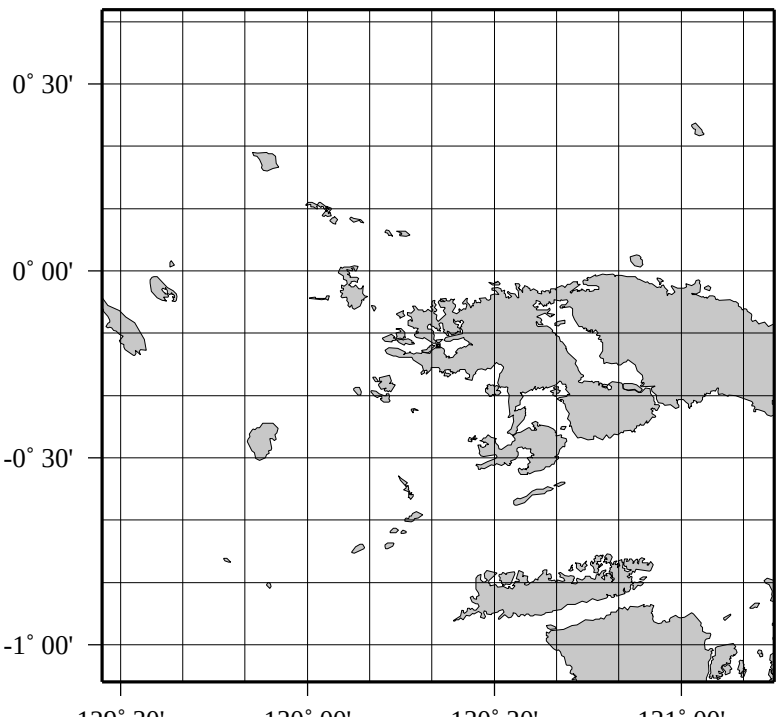
```
#!/bin/sh
#    $Id: GMT_App_K_4.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#
pscoast './getbox -JE130.35/-0.2/1i -100 100 -100 100' -JE130.35/-0.2/3.5i -P -Dh -A1 -G200 \
   -W0.25p -N1/0.25tap -B30mg10mWSne -K > GMT_App_K_4.ps
./getrect -JE130.35/-0.2/1i -20 20 -20 20 | psxy -R -JE130.35/-0.2/3.5i -O -W1.5p -L -A \
   >> GMT_App_K_4.ps
```

K.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 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 hefty 55.7 Mbytes. Our final map is reproduced by the single command:

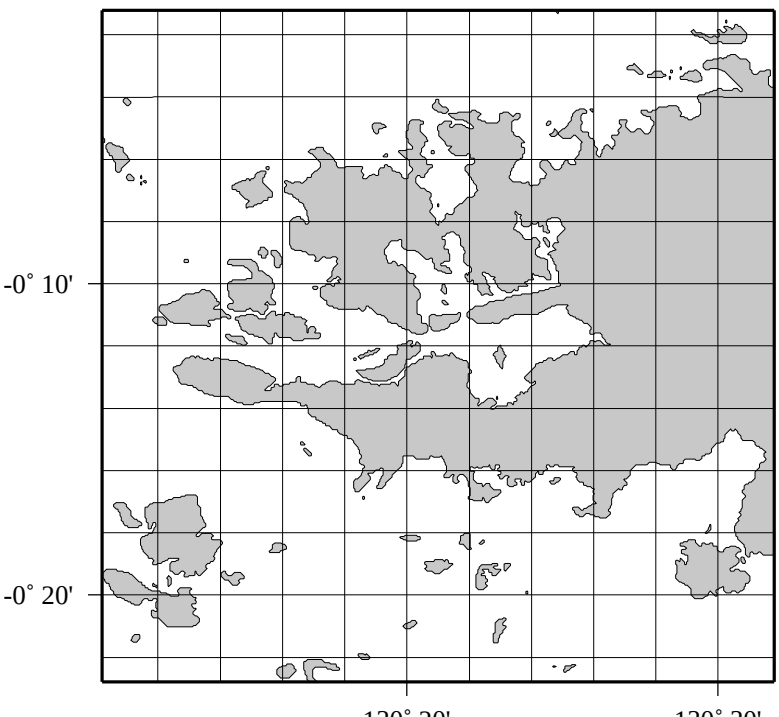

10˚
0˚
-10˚


4˚
2˚
0˚
-2˚
-4˚


0˚ 30'
0˚ 00'
-0˚ 30'
-1˚ 00'

```
#!/bin/sh
#    $Id: GMT_App_K_5.sh,v 1.1 2001/03/21 04:10:21 pwessel Exp $
#
pscoast ‘./getbox -JE130.35/-0.2/1i -20 20 -20 20‘ -JE130.35/-0.2/3.5i -P -Df -G200 -W0.25p \
   -N1/0.25tap -B10mg2mWSne > GMT_App_K_5.ps
```

We hope you will study these examples to enable you to make efficent and wise use of this vast data set.


-0˚ 10'
-0˚ 20'

L. GMT on non-*UNIX* platforms

L.1 Introduction

While GMT can be ported to non-*UNIX* systems such as Windows and MacOS, 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* PC environment such as Linux (PC) or MkLinux (Mac). There are also commercial products for PCs (e.g., Interix [formerly OpenNT][1]) and Macs (e.g., MachTen[2]) that will provide a POSIX environment without rebooting into *UNIX*. Installation of GMT under Interix or Machten is no different than under other POSIX *UNIX*systems.

However, if you own a PC and need a public domain, no-cost solution other than Linux you have a few additional options. At the time of this writing they include

1. Install GMT under Cygwin (A GNU port to Windows).
2. Install GMT under DJGPP (another GNU port to Windows/DOS).
3. Install GMT directly using Microsoft C/C++ or other compilers.

Unlike the first two, the latter will not provide you with any *UNIX* tools so you will be limited to what you can do with DOS batch files.

L.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.).

Follow the instructions on the Cygwin page[3] 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. For details see the general README file.

L.3 DJGPP and GMT

DJGPP[4] is similar to Cygwin in that it provides precompiled *UNIX* tools for DOS/WIN32, including the ***bash*** shell. At the time of this writing we have not been successful in compiling netCDF in this environment. This is fully due to our limited understanding of the innards of the netCDF installation whose configure script did not work for us. As soon as this problem is overcome we expect a smooth install similar to that of Cygwin.

L.4 WIN32 and GMT

GMT will compile and install using the Microsoft Visual C/C++ compiler. We expect other WIN32 C compilers to give similar results. Since ***configure*** cannot be run you must manually rename gmt_notposix.h.in to gmt_notposix.h. The netCDF home page gives full information on how to compile and install netCDF; precompiled libraries are also available. At present we simply have a lame gmtinstall.bat file that compiles

[1] www.interix.com
[2] www.tenon.com
[3] sources.redhat.com/cygwin
[4] See www.gnu.org for details.

the entire GMT package, and gmtsuppl.bat which compiles most of the supplemental programs. If you just need to run GMT and do not want to mess with compilations, get the precompiled binaries from the GMT ftp sites.

L.5 OS/2 and GMT

GMT has been ported to OS/2 by Allen Cogbill[5], Los Alamos National Laboratory. One must have EMX[6] installed in order to use the executables. All features that are present in the *UNIX* version of GMT are available in the OS/2 version. All executables may be obtained using links in the following document[7], which provides more detail on the port.

L.6 MacOS and GMT

GMT has not been ported to the classical Macintosh platform (i.e. MacOS 9.x or earlier). For that OS your only option is MachTen. However, GMT will install directly under MacOS X.

[5] mailto:ahc@lanl.gov
[6] ftp://ftp.geophysics.lanl.gov/pub/EES3/pub/gmt/emxrt.zip
[7] ftp://ees.lanl.gov/pub/EES3/pub/gmt/gmt4os2.html

Index

Y