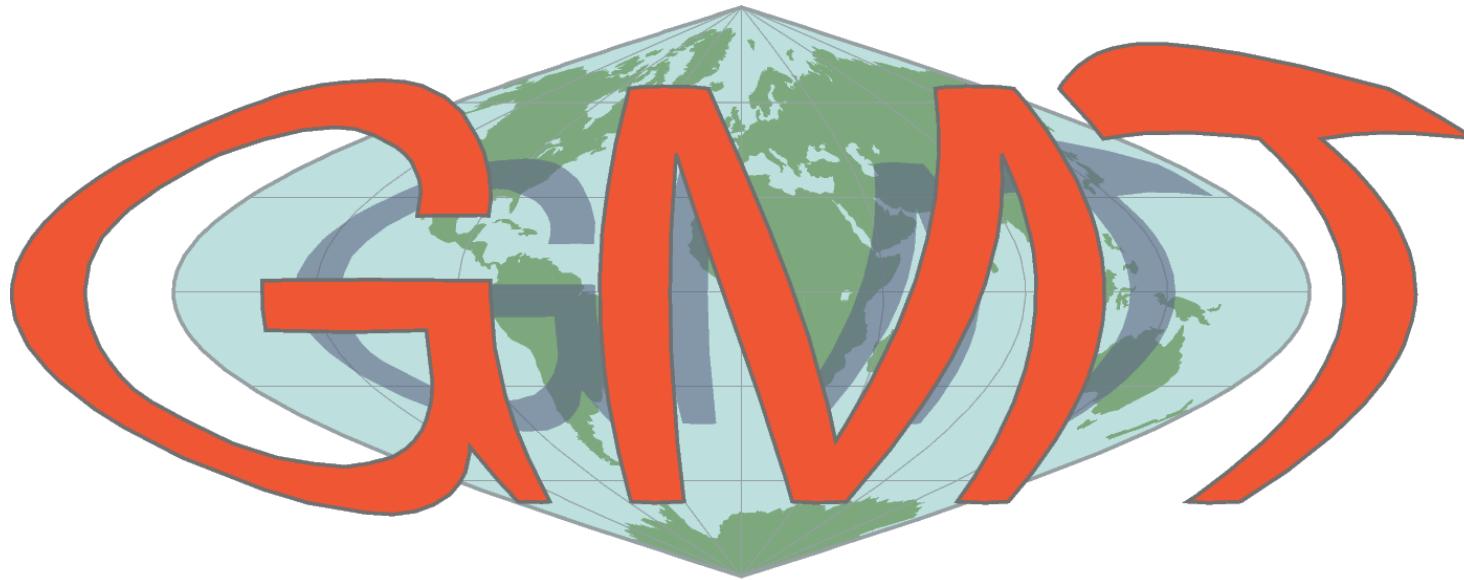


GEO 242: Numerical methods and modeling in the geosciences



THE GENERIC MAPPING TOOLS

Lecture 4: An introduction to GMT

Outline

The Generic Mapping Tools

Some basic commands – psbasemap, pscoast, psxy

General tips and strategies for GMT scripts

Examples

GMT

The Generic Mapping Tools were developed by Paul Wessel and Walter Smith in the late Eighties

They are a series of command-line utilities that generate PostScript-format plots

Although originally developed with plotting marine geophysical data in mind, the tools were made sufficiently generic as to allow numerous other applications

Development of GMT was managed by Wessel and Smith, and supported by grants from the NSF, for decades. It has now been handed over to a steering committee, with the code open-sourced. Ongoing funding for development will be through the National Geophysical Facility, operated by EarthScope.

The current major version, and the one we will learn, is version 6 (specifically, version 6.6)

GMT references

Wessel, P. and W. H. F. Smith, Free software helps map and display data, *EOS Trans. AGU*, 72, 441, 1991) [**version 2.4.1**]

Wessel, P., and W. H. F. Smith (1995), New version of the generic mapping tools, *Eos Trans. AGU*, 76(33), 329–329 [**version 3**]

Wessel, P. and W. H. F. Smith (1998) New, improved version of the Generic Mapping Tools released, *EOS Trans. AGU*, 79, 579 [**version 3.1**]

Wessel, P., W. H. F. Smith, R. Scharroo, J. Luis and F. Wobbe (2013), Generic Mapping Tools: Improved Version Released, *Eos Trans. AGU*, 94(45), 409. [**version 5**]

Wessel, P., Luis, J. F., Uieda, L., Scharroo, R., Wobbe, F., Smith, W. H. F., & Tian, D. (2019). The Generic Mapping Tools version 6. *Geochemistry, Geophysics, Geosystems*, 20, 5556–5564. <https://doi.org/10.1029/2019GC008515> [**version 6**]

Wessel, P. (2024). The origins of the generic mapping tools: From table tennis to geoscience. *Perspectives of Earth and Space Scientists*, 5, e2023CN000231.
<https://doi.org/10.1029/2023CN000231>

Using GMT

Although in some cases the command line can be used to make a plot, in most cases you are going to want to write a script

Advantages of writing a script:

- repeatability
- complexity
- automation
- adaptability

Using GMT

Many GMT commands, especially the ones dealing with ASCII text,
can accept input from files or from standard input

Equally, the same GMT commands write their results to standard
output

This means that you can pipe output from `awk` directly into GMT,
and not need to keep making intermediate files for plotting

GMT 4 vs GMT 5 (and 6)

Major changes to the syntax of GMT commands was introduced with version 5 in 2013.

One example: in GMT 4 you could call a GMT command directly from the shell, e.g.

```
psxy points.xy -Jm -R -Sc0.3 -Gred -O -K >>  
$outfile
```

After version 5, each GMT command is executed as an option of a GMT ‘wrapper’ command, e.g.

```
gmt psxy points.xy -J -R -Sc0.3 -Gred -O -K >>  
$outfile
```

GMT 6

GMT 6 broke even more with tradition and introduced 'modern mode', which included a simplified syntax, more in keeping with the Python version, PyGMT. (I have not yet adopted it...)

GMT 5 syntax, which many of us took several years to adjust to, is preserved as 'classic mode' in GMT 6

Another major addition was GDAL compatibility, meaning that many more geospatial data formats were now available to be plotted natively with GMT functions

Help with GMT

GMT has comprehensive online help – both typing the commands without arguments and the man pages give comprehensive information on syntax

The man pages are also available through the GMT website (<https://www.generic-mapping-tools.org/>), where there is also a tutorial, manual and ‘cookbook’ (a set of examples of different plot styles and map projections) as well as a forum

Various snarky comments in the docs suggest that for the 'classic mode' cookbook, you should look at the version 5.4 documentation... here: https://docs.generic-mapping-tools.org/5.4/GMT_Docs.html

GMT and data types

GMT handles point data in ASCII format and, before version 6, used its own raster (image) data format, ‘.grd’ format (a netCDF-compatible format).

Now, with GDAL support, it can handle most standard raster formats

In the most part, the same kinds of operations possible with ASCII data are possible with raster data, although the commands will usually have different names – at least, in classic mode (ASCII commands usually start with ‘ps’ and raster commands with ‘grd’)

[I hear that modern mode makes no distinction between ASCII and raster commands...]

What is PostScript and how do I view it?

PostScript is an interpreted programming language, originally developed by Adobe in the early 1980s to tell printers where to put things on a page. Apple licensed it for their LaserWriter printer, and it took off!

In subsequent versions, it became used as a format for images, particularly those with vector components.

It is ASCII readable, in theory (plots with raster components have sections that are not especially readable). But reading the ASCII is not going to show you what you want to see!

What is PostScript and how do I view it?

To view your plots, you will need a PostScript interpreter

- GhostScript (`gs`) is the classic (i.e. ageing) program that was probably installed on your system if you installed texlive
- Preview on the Mac is a very versatile, built-in image viewer than can read PostScript
- On Linux, my current favorite viewer is evince, which can be installed with aptitude (`sudo apt install evince`)

You can also convert your figure to a pdf using `ps2pdf`, also installed with texlive, and use a pdf viewer

Some basic GMT commands

`psbasemap` – set the plot area

`pscoast` – plot coastlines, rivers, lakes, boundaries

`psscale` – plot a color scale

`pstext` – plot text

`psxy` – plot x,y data (symbols or lines/polygons)

`makecpt` – make a color palette

`info` – report maximum and minimum values

Common command options

- J<proj><scale> – sets the map projection/plot type. ‘proj’ is a letter code specifying a projection, ‘scale’ is a number setting the plot scale (if ‘proj’ is a capital letter, this is the size of the plot, if it is lower case, this is the scale per unit)
- R<west>/<east>/<south>/<north> – sets the plot extents (‘R’ for ‘region’)

Once you have specified these two options, you do not have to repeat them in full unless you want to change the plot; simply state -J and -R

Common command options

- B<axes> – ‘axes’ is a combination of upper and/or lower case letters, e.g. ‘WeSn’, where the capitals reflect annotated axes and lower case unannotated axes; if a letter is omitted, so is the corresponding axis
- B [x | y] a<annot_ival>f<frame_ival> – sets the axis annotation interval and frame tick interval for the x or y axis; you can have one separate option for each
- O – tells GMT to overlay the current plot on a pre-existing plot file
- K – tells GMT that more PostScript output (i.e. more GMT output) will be added after this command

Common command options

- X<distance> -Y<distance> – these options shift the position of the plot origin (default is 2.5 cm for both, from the bottom left corner of the plot)
- P – plot paper will be in portrait orientation
- N – will plot symbols or text outside of the specified plot area (useful if you have points right on the boundary)

Specifying units in GMT

GMT can handle both geographic (i.e. lat-long) and xy data.

Where plotting is concerned, units are implicit (e.g. where you are using a geographical map projection, the plot units are assumed to be degrees; in an xy plot, no units are assumed)

Where scales/plot dimension are concerned, you have the option of using a range of different units. Most commonly, cm or inches can be used. To specify a size in cm, append a 'c' to the value (e.g. '5c'); for inches an 'i' is appended.

Specifying color in GMT

In most cases, color is specified using the RGB (red, green, blue) color system.

A number between 0 (zero intensity) and 255 (full intensity) is given for the three components, separated by a slash

e.g. 0/0/255 is bright blue

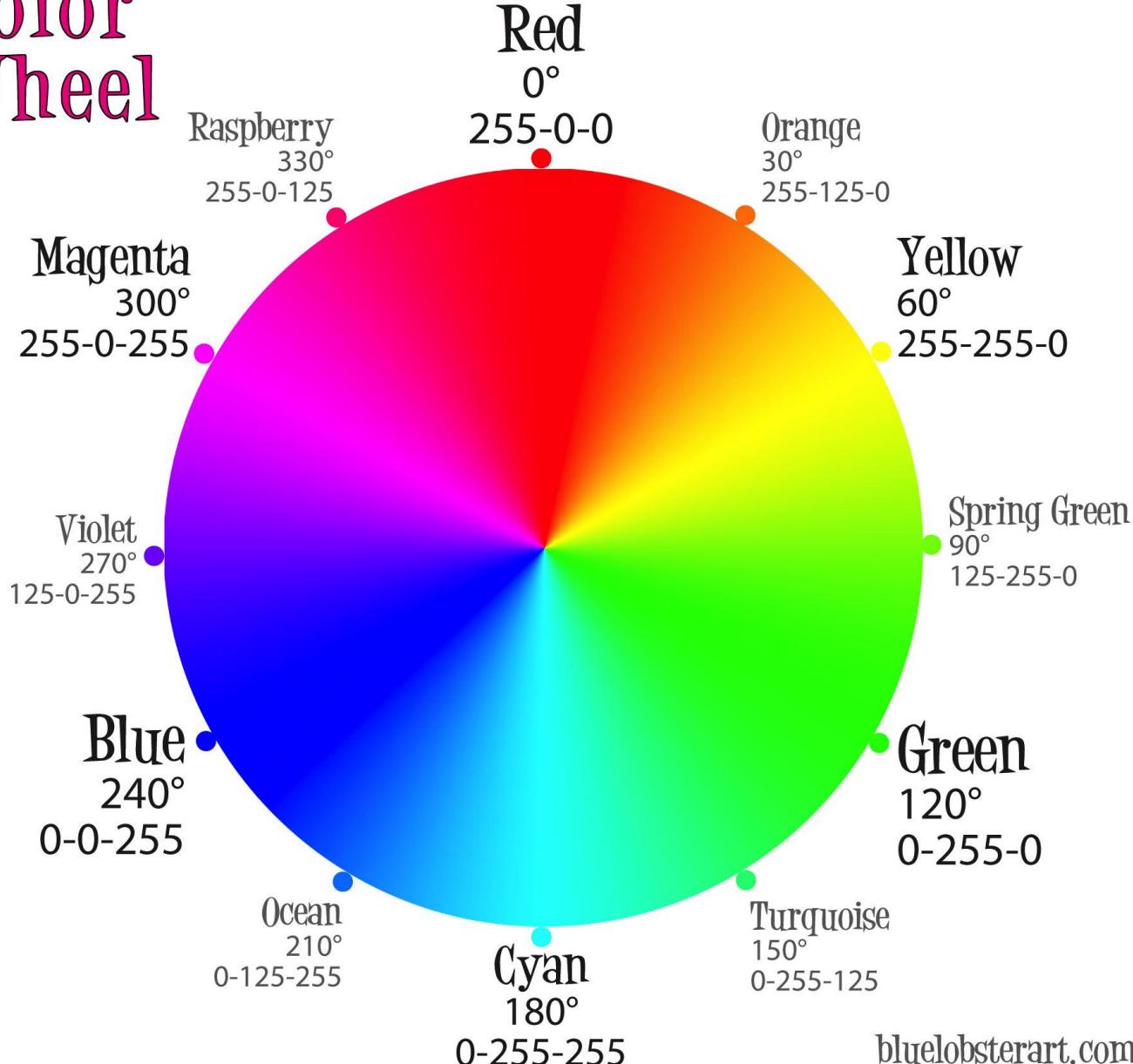
128/0/0 is a darkish red

255/0/255 is magenta

For grayscale a single number between 0 (black) and 255 (white) can be used.

Since version 4, you can also use words to describe colors (e.g. darkred, lightgreen, etc), or the HSV system.

RGB Color Wheel



psbasemap

define map projection and plot area

```
gmt psbasemap -J<options> -R<extents> -B<options> -K >  
<outfile>
```

will create a map frame in which to place the rest of your plot.

```
gmt psbasemap -J<options> -R<extents> -B<options>  
-L<xp>/<yp>/<scl_lat>/<length> -K > <outfile>
```

will add a map scale of length ‘length’ (this is in plot units), estimated at latitude ‘scl_lat’, to the frame at the position <xp>,<yp> (there is no other way of doing this)

psbasemap

psbasemap could be regarded as a redundant step, as if you use the same `-J`, `-R` and `-B` options in several other commands (e.g. psxy or pscoast), GMT will create a basemap frame by default.

However, it may be useful to separate out the basemap frame drawing from the other steps, in order to keep a script simple.

Also, for additional help with obscure map projections, the psbasemap man page is the place to look...

Map projections

In most cases, you will want to plot maps in one of two projections – Mercator or xy.

The Mercator projection ($-Jm$ or $-JM$) is the most common rectilinear projection for lat-long data, essentially trying to represent spherical coordinates on a cylinder. This is great for plotting low-latitude data, but is highly distorted at high latitudes.

The xy ‘projection’ ($-Jx$ or $-JX$) allows simple rectilinear xy plots. There can be separate scales for the x and y axes if desired. This also works well, where the axes scales are equal, for plotting of local (e.g. UTM) coordinates

psbasemap examples

```
gmt psbasemap -JM20c -R-130/-110/30/40 -Bxya5f1 -BWeSn  
-L-128/31/35/100k > test.ps
```

This will plot a basemap frame with longitude extents -130°E and -110°E , and latitude extents 30°N and 40°N . It is a Mercator projection plot 20 cm wide. It is annotated every 5° on the West and South axes, with 1° frame ticks. A map scale 100 km long (with distances estimated at 35°N) is plotted in the lower left corner of the plot.

In this case, specifying the projection as $-\text{Jm1c}$ would give the same result (the plot is 20° wide)

psbasemap examples

```
gmt psbasemap -JX16c -R0/500/0/300 -Bxya100f50 -BWesN >  
test.ps
```

This will plot a basemap frame with x extents 0 and 500, and y extents 0 and 300. It is a xy plot 16 cm wide and 16 cm tall (i.e. there are different scales for the x and y axes). It is annotated every 100 units on the West and North axes, with frame ticks every 50 units.

To get equally scaled axes, you would need to use:

```
gmt psbasemap -JX16c/9.6c -R0/500/0/300 -Bxya100f50 -BWesN  
> test.ps or
```

```
gmt psbasemap -Jx0.024c -R0/500/0/300 -Bxya100f50 -BWesN >  
test.ps
```

-K and -O

GMT in classic mode has a few foibles that modern mode was designed to avoid. The most prominent of these are the handling of "closing" the PostScript files generated by your GMT commands.

-K in a command indicates that "more PostScript will be added"

- typically you use this on the first plotting line, along with a single redirect (>) to write a fresh PostScript file
- you would also use it on every other line of your script except the last line (because no more PostScript is going to follow). All of those additional lines should use the double redirect (>>) to append more PostScript and not overwrite your file.
- of course, if you only have one plotting line, then you do not need to include it...

-K and -O

But you can't just add more PostScript willy-nilly! You have to tell GMT what to do with it...

- O in a plotting command indicates that you are plotting an overlay
 - whether or not what you are plotting is actually overlaying anything else is besides the point, if you don't include -O, your PostScript interpreter will claim it doesn't know what to do with your plot!

Practically, this means that if you are building a plot with lots of layers, you are going to want to include -O -K >> on every line from the second line onwards!

And to close your PostScript file...

If you are worried about adding `-K` to the end of every line and not closing your PostScript file properly, you should add the commands

```
echo showpage >> <outfile>
```

```
echo end >> <outfile>
```

to the end of your script. This has the same effect as not adding `-K` to the last GMT command, but means you can add more GMT commands easily, since you no longer have to keep track of which line is the last one

pscoast

Plot coastlines, rivers, lakes and political boundaries

```
gmt pscoast -J<options> -R<region> -B<axes> -W<thickness> >  
test.ps
```

By default this plots coastlines and lakes at default (low) resolution, with a line of specified thickness (GMT thicknesses seem to be multiples of a quarter of a point)

```
gmt pscoast -J<options> -R<region> -B<axes> -W<thickness>  
-Df > test.ps
```

This plots coastlines and lakes at ‘full’ resolution (other $-D$ options: h – high, i – intermediate, l – low, c – crude).

pscoast

```
gmt pscoast -J<options> -R<region> -B<axes> -W<thickness>  
-I<code> > test.ps
```

This draws rivers, ‘code’ specifying which classes of rivers get drawn, taken from a list on the man page (e.g. -I1 – permanent major rivers).

```
gmt pscoast -J<options> -R<region> -B<axes> -W<thickness>  
-N<code> > test.ps
```

This draws national and other political boundaries, again, ‘code’ specifying which classes of boundaries get drawn (e.g. -N1 – national boundaries; -N2 – state boundaries)

pscoast

```
gmt pscoast -J<options> -R<region> -B<axes> -W<thickness>  
-S<color> > test.ps
```

This fills the ‘wet’ areas (lakes/rivers/seas) with a specified grayscale (single number, between 0 and 255), RGB color (three numbers, between 0 and 255, separated with slashes), or color name.

```
gmt pscoast -J<options> -R<region> -B<axes> -W<thickness>  
-G<color> > test.ps
```

This instead fills the ‘dry’ areas (i.e. land), using the same color code system

pscoast example

```
gmt pscoast -JM20c -R-130/-110/30/40 -Bxya5f1 -BWeSn -Di  
-G225 -Wthin -N1,thick -N2,faint > test.ps
```

This will plot the same basemap frame as in the psbasemap example before, except without the map scale. Additionally, it will plot coastlines at intermediate resolution with a thin line width, shade the land areas pale gray, plot national boundaries with thick lines and state boundaries with faint lines.

psxy

Plot xy points, lines or polygons

```
gmt psxy <infile> -J<options> -R<region> -B<axes>  
-S<symbol><scale> > test.ps
```

This will plot x,y data from ‘infile’, representing each point with a chosen symbol, at a specified scale (defining the diameter of a circumcircle within which the symbol would fit).

Example ‘symbol’ codes: -Sa – star, -Sc – circle, -Si – inverted triangle, -Ss – square, -St – triangle, -Sx – cross.

psxy

```
gmt psxy <infile> -J<options> -R<region> -B<axes>  
-S<symbol><scale> -G<color> > test.ps
```

This will plot x,y data from ‘infile’, representing each point with a chosen symbol, filled with the shade ‘color’ at a specified scale

```
gmt psxy <infile> -J<options> -R<region> -B<axes>  
-S<symbol><scale> -G<color> -W<thickness> > test.ps
```

This will plot x,y data from ‘infile’, representing each point with a chosen ‘symbol’, filled with the ‘color’, and outlined with a line of specified thickness at a specified scale

psxy

```
gmt psxy <infile> -J<options> -R<region> -B<axes>  
-W<thickness> > test.ps
```

This will plot x,y data from ‘infile’, as the vertices of an unfilled, unclosed polygon (polyline?)

```
gmt psxy <infile> -J<options> -R<region> -B<axes>  
-W<thickness> -G<color> > test.ps
```

This will plot x,y data from ‘infile’, as the vertices of a filled (with shade ‘color’), unclosed polygon

```
gmt psxy <infile> -J<options> -R<region> -B<axes>  
-W<thickness> -G<color> -L > test.ps
```

This will plot x,y data from ‘infile’, as the vertices of a filled (with shade ‘color’), closed polygon

psxy

```
gmt psxy <infile> -J<options> -R<region> -B<axes>  
-W<thickness> > test.ps
```

This will also plot multiple polylines from ‘infile’, assuming that the vertices of each polyline are separated by a line containing only a ‘>’ symbol in ‘infile’ – this also works for polygons

```
gmt psxy <infile> -J<options> -R<region> -B<axes>  
-W<thickness> -h1 > test.ps
```

This will ignore the first line of ‘infile’ when plotting

psxy example

```
gmt psxy locs.dat -JM20c -R-130/-110/30/40 -Bxya5f1 -BWeSn  
-Sc0.2c -G255/0/0 > test.ps
```

This will plot the same basemap frame as in the `psbasemap` example before, except without the map scale. Additionally, it will plot 2 mm-diameter red circles at the locations given in 'locs.dat'.

Tips for GMT scripting

Specify variables for the input and output files (as necessary) at the start of the script

You might also want to set variables containing the extents

Try to do only one thing per GMT command – there is no premium on script length

Add comments for every line – ditto

My first map

- 1) Make a global map, using an appropriate map projection, plotting the land a tasteful shade of green, and filling a sheet of letter paper (with appropriate margins).

My first map

2) Find (on the Internet!), and then plot, plate boundaries on your map.

My second map

3) Plot a regional map of southern California, with borders and major water bodies plotted at appropriate levels of detail, and plot the largest daily earthquakes from September 2010 on it, color-coded by depth and scaled by magnitude