

Data Analysis in Geophysics

ESCI 7205

Class 11

Bob Smalley

Printing

OLNY SRMAT POELPE CAN RAED TIHS

I cdnuolt blveiee taht I cluod aulacly uesdnatnrd waht I was rdanieg. The phaonmneal pweor of the hmuancnid.

Aoccdrnig to rscheearch at Cmabrigde Uinervtisy, it deosn't mttær in waht oredr the ltteers in a wrod are, the olny iprmoatnt tihng is taht the frist and lsat ltteer be in the rghit pclae. The rset can be a taotl mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe. Amzanig huh? yaeh and I awlyas tghuhot slpelng was ipmorant!

Tihs deos not wrok for the cetupmor!

Manipulating & Printing Files

Basics of the UNIX/Linux Environment

Printing Commands

lpr: submit files for printing

```
% lpr -P3892_grad file.txt
```

Printing Commands

lpq: show printer queue status useful to find out if other jobs are before yours.

```
%lpq -P3892_grad  
3892_grad is ready and printing  
Rank      Owner      Job          File(s)        Total Size  
active    hdeshon   146          junk.pdf     108544 bytes
```

Identifies the job.

lprm: cancel print job (by number)

% lprm -P3892_grad 146

lpstat: printer status information

useful for finding out printer names on Macs,
which are not necessarily the same as on the SUN
system

% lpstat -a

_3876langston accepting requests since Wed Aug 27 13:11:36 2008

hp_color_LaserJet_4600 accepting requests since Mon Aug 4

11:50:47 2008

CERI Printers

Long Building (3892 Central)

3892_grad -- B & W printer in Mac Lab

3892_hpcolor -- Color printer in Mac Lab

3892_hpxlfp -- Poster printer in Mac Lab

3892_Mitch -- B & W printer in Mitch's office

3892_colorps -- B & W printer in

CERI Printers

House 3 (3876 Central)

3876_langston --

3876_hpcolor -- Color printer

3876_grad -

3876_bodin -

3876_powell -

CERI Printers (Continued)

House 2 (3890 Central)

3890_hpcolor ~ Color printer in copier room

3890_copy ~ B & W printer in copier room

3890_sheila ~ B & W printer in Michelle's office

House 0 (3918 Central)

3918_usgs ~

CERI Printers (Continued)

House 1 (3904 Central)

3904_tek -- Color printer

3904_tekdup -- Color printer

3904_hallway -- B & W printer

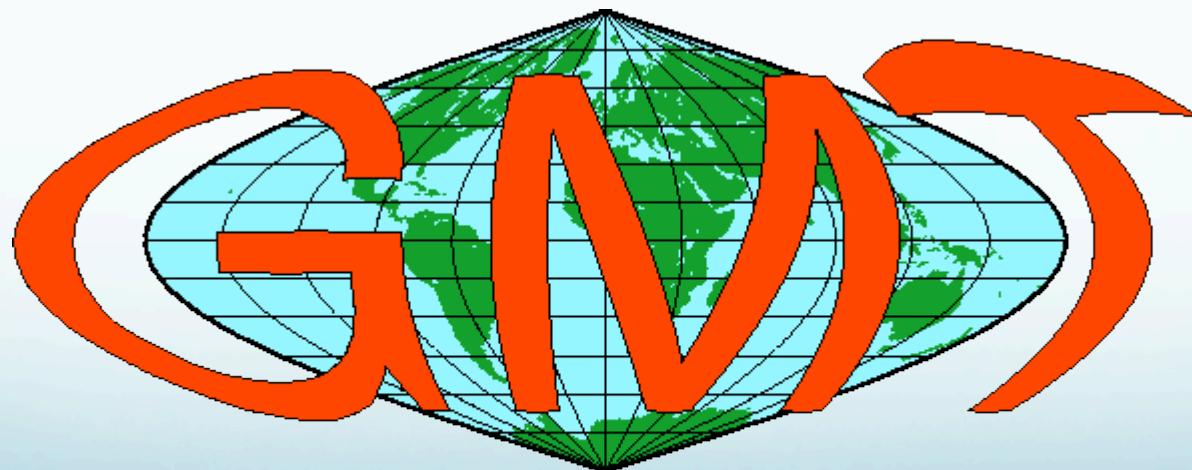
3904_brother --

Data Analysis in Geophysics

ESCI 7205

Class 11

Bob Smalley



Generic Mapping Tools Graphics

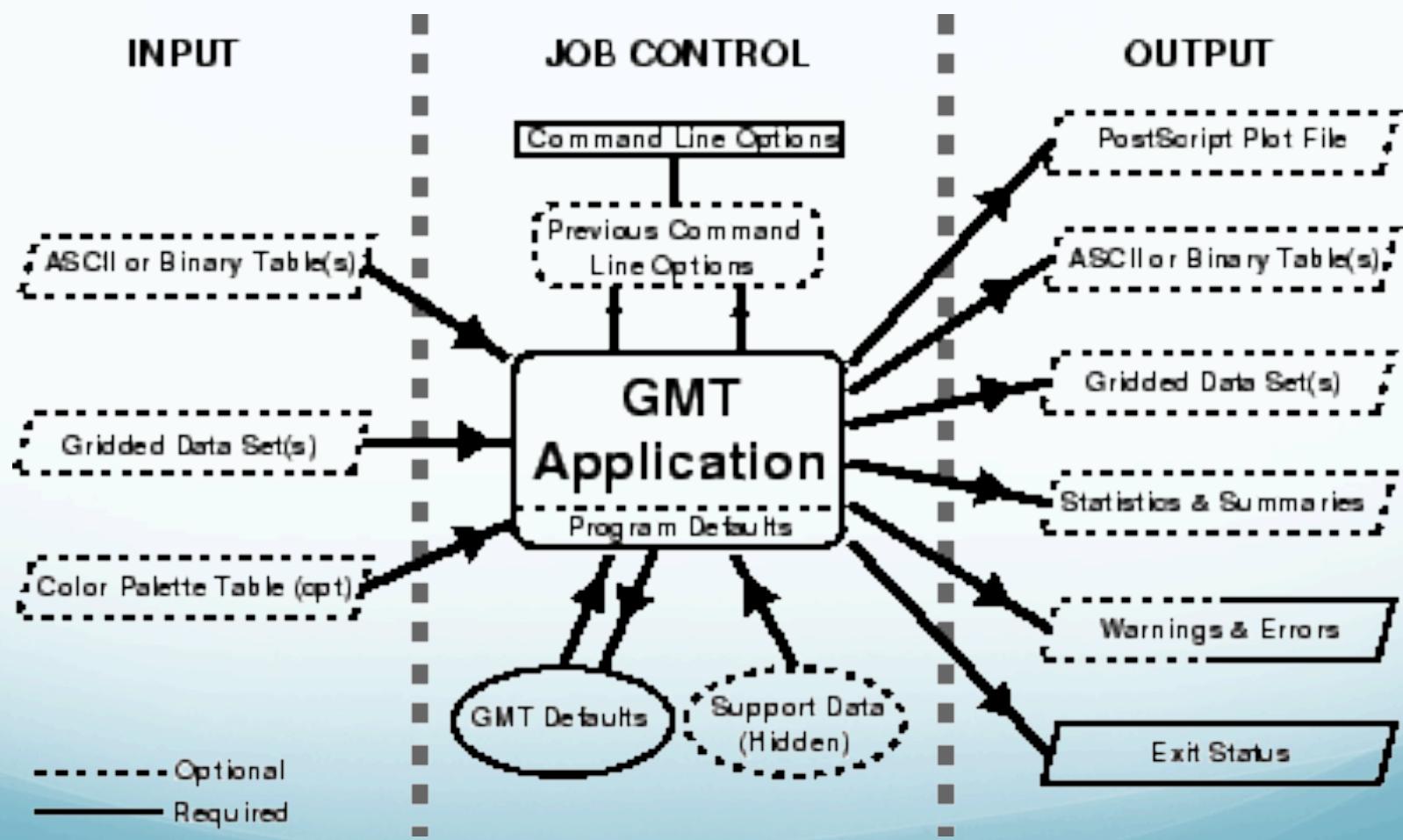
Easiest way to get started

- 1) Find system with GMT already set up
- 2) Get working program (shell script) from someone else and modify (hack) it.

Lots examples in

- Tutorial
- available on www
- available from your “friends”

What goes on in GMT



Sources of operational parameters/job control

- i) command line options/switches or program defaults
- ii) carried over from execution of previous commands
- iii) from your .gmtdefaults file

(looks first in working directory, then in your home directory, finally the system, program defaults)

Sources of operational parameters/job control

Why a defaults file?

- too many parameters to require setting all explicitly (powerful)
- customize – can have different defaults in different directories

Basic GMT use

Most GMT programs

read input from terminal (*stdin*) or files, and

write output to terminal (*stdout*) (a few write to
files)

– follow UNIX philosophy.

To write output to files one can use *UNIX*
redirection (else goes to screen – uselessly):

GMTprogram switches >> Outputfile

Most GMT programs will accept input-file names
and pipes in lieu of stdin

GMTprogram input-file switches > outfile

GMTprogram switches < input-file > outfile

Someprogram | GMTprogram1 | GMTprogram2 > outfile

Many GMT programs will also accept input redirection (in-line input) – reads whatever follows -- up to character string XXX -- as input.

```
GMTprogram switches << END > output-file
```

```
.1 .1
```

```
.2 .2
```

```
END
```

Can also do with “command substitution”:

```
GMTprogram switches << FIN > output-file  
`someprogram swithches < input-file...`  
FIN  
  
echo `someprogram swithches < input-file...` | GMTprogram switches  
> output-file
```

Some GMT programs require input-file names
(usually when need more than one input file, or
input usually so big that one would be forced to
pipe or redirect input all the time, or binary file,
etc.)

GMT and scripts

GMT commands act much like regular UNIX commands.

Generally, the commands are enacted within a shell script so that they may be combined with other UNIX commands such as awk.

bash and csh are the most commonly encountered shells in academia and passing down GMT scripts is how much of seismology gets illustrated

Use Comments!

Comments are very popular to forget but if you don't comment your script, 2 years later you may not remember what you were doing (especially if you write tight UNIX code that took 20 iterations to get "correct" [as compact as possible]).

Spaces and blank lines make your script readable. While it may take more paper if you print it, it only takes two bytes to make new line or a space.

Keeping track of your scripts

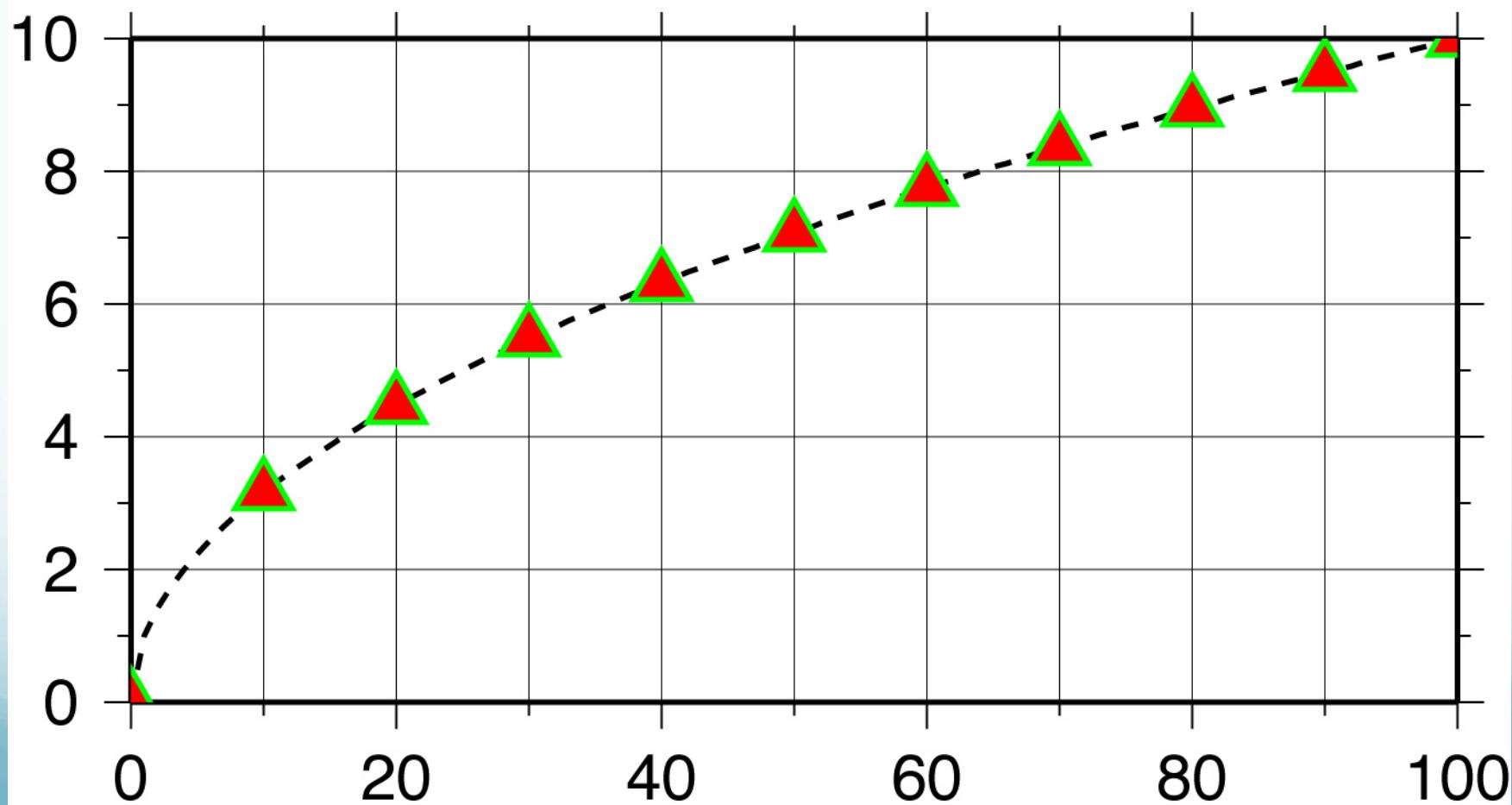
You will be glad (someday) if you set up directories and subdirectories to keep your maps, data, and scripts organized.

Mitch suggests have something like this in ~ /GMT

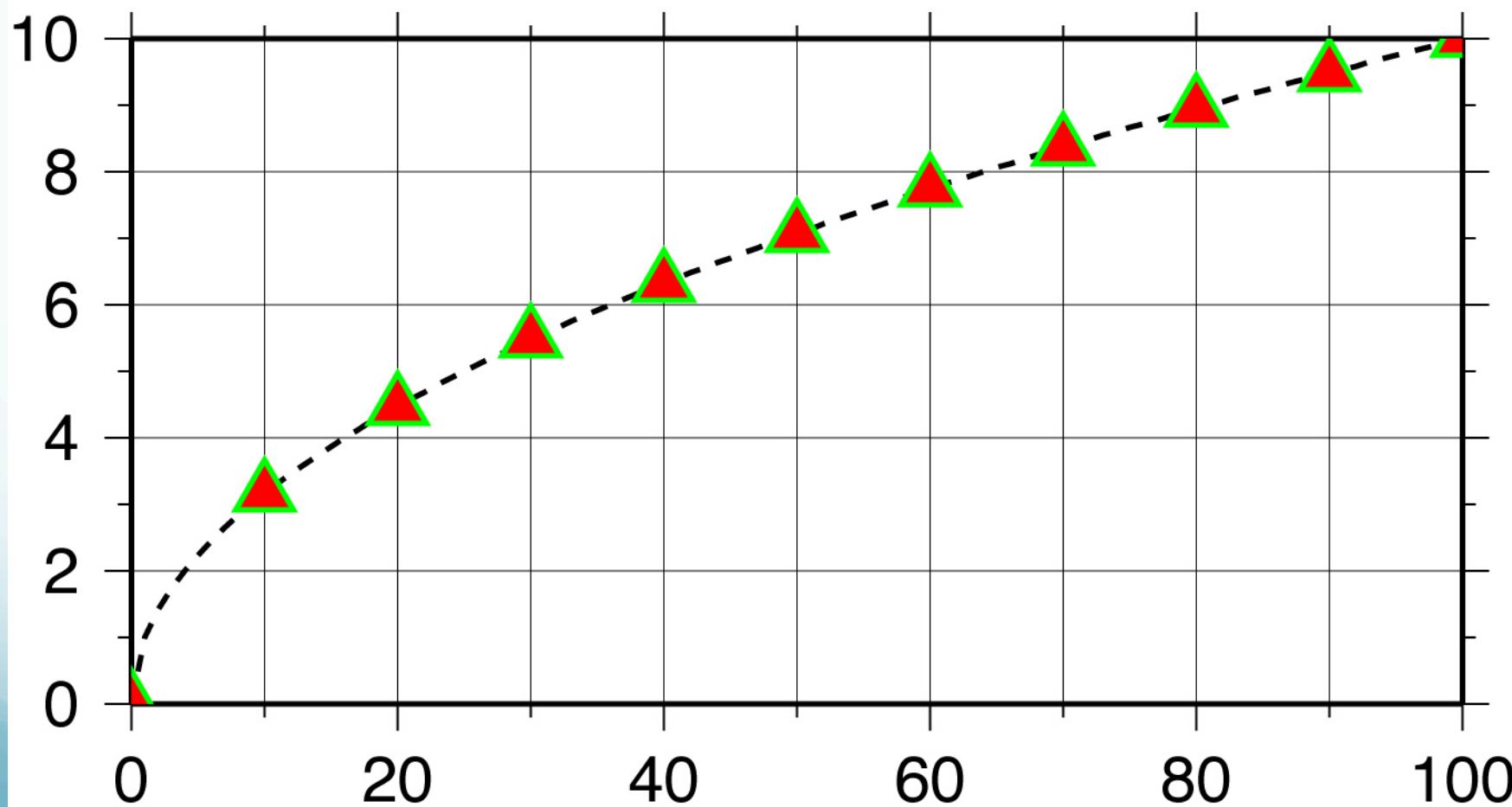
csh/ data/ ps/ scratch/ sh/

There is a directory for csh scripts, for sh scripts, for data files, for postscript files, and for scratch files.

OK lets look at some “simple” examples:



Plot $x^{1/2}$ from 0 to 100 as a dashed line, using red triangles with green borders at $x=n \cdot 10$.



- 1) We start by making the basemap frame for a linear x - y plot.
- 2) We want it to go from 0 to 100 in x , with ticks, grid and annotation every 10, and from 0 to 10 in y , with ticks, grid and annotation every 2.

3) The final plot should be 4 by 3 inches in size.

Note GMT does not make any helpful assumptions such as

- a) You want to plot the whole x and y range of the data and
- b) You want it to fit nicely on the page.

You have to specify EVERYTHING (comes under the excuse of being “powerful”)

Here's how we do it:

```
psbasemap -R0/100/0/10 -JX4i/3i -B10/1::."My first plot": -P \
> plot.ps
```

We will first look at how we specify to GMT how to make the map/figure.

This is done using the command line options/switches.

`psbasemap`

draws a map frame and sets up the map parameters

(so they don't have to be re-specified in later GMT program calls, although it is a good idea – variables make it easy).

```
psbasemap -R0/100/0/10 -JX4i/3i -B10/1::."My first plot": -P \
> plot.ps
```

Requirements 1 (projection) and 3 (axis sizes) are specified to GMT together

- 1) We start by making the basemap frame for a linear (the projection, or lack of one) x-y plot.
- 3) The final plot should be 4 by 3 inches in size.

psbasemap

The **-J** option selects the type of projection and the scale.

In this case we want a linear x-y plot, or no projection, which is specified by

x or X.

```
psbasemap -R0/100/0/10 -JX4i/3i -B10/1::."My first plot": -P \
> plot.ps
```

There are 25 projections available in GMT, each specified by one letter (case sensitive to set options).

There are no provisions for providing your own projection.

(short of using the open source to roll your own.)

Requirements 1 and 3 are specified to GMT together

The **-J** option also sets the axis scales (distance per unit, **x**) or (axis length, **x**)

Where the “unit” is specified in **.gmtdefaults** or explicitly – inches, **i**, or cm, **c**.

```
psbasemap -R0/100/0/10 -JX4i/3i -B10/1::."My first plot": -P \
>! plot.ps
```

2) We want it to go from 0 to 100 in x, with grid and annotation every 10, and from 0 to 10 in y, annotating every 1.

This is really two conditions

i) We want it to go from 0 to 100 in x, and from 0 to 10 in y.

Specified by the REGION (-R) option, which (in the usual form) is

-Rxmin/xmax/ymin/ymax

2) We want it to go from 0 to 100

-Rxmin/xmax/ymin/ymax

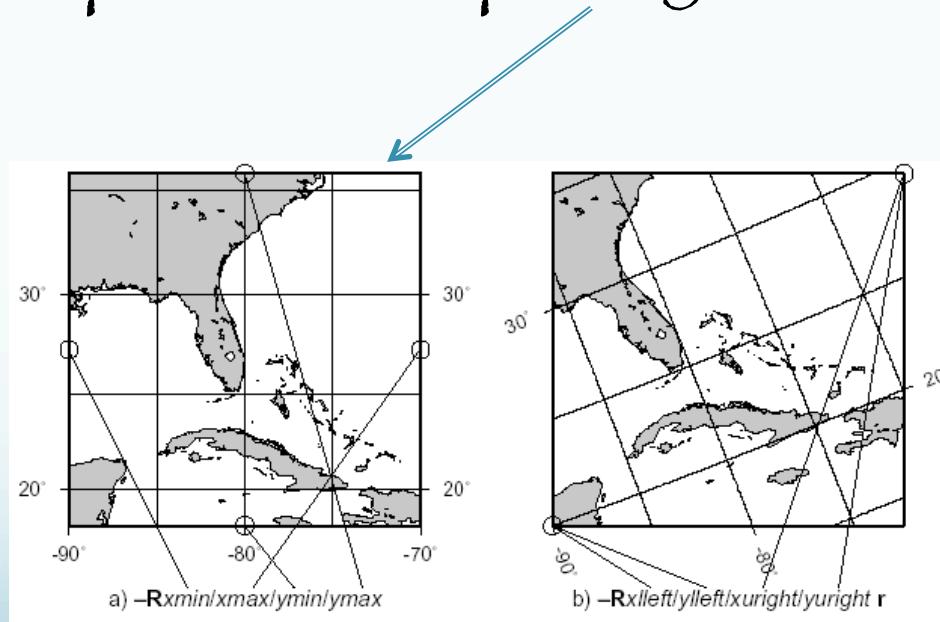
Notice that unlike MATLAB, GMT does not make any assumptions about what you want (such as the reasonable one that you just might want the region to show all the input data).

You have to specify every detail. (i.e. powerful)
(why should the writers of gmt work hard when they can convince the user that it is “better” if the users do!)

```
psbasemap -R10/100/10 -JX4i/3i -B10/1:"My first plot": -P \
>! plot.ps
```

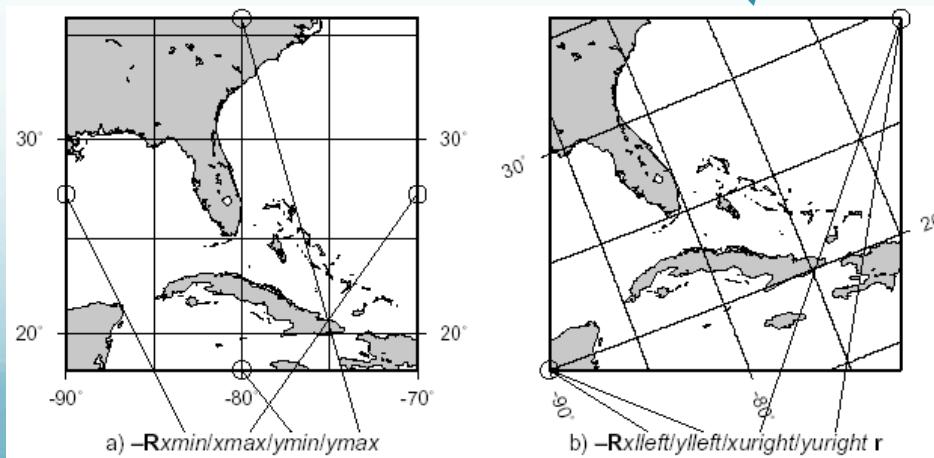
There are two forms for the **-R** option

- 1) For projections where the boundaries follow lines of latitude and longitude (“rectangle” on sphere) – specify sides.



There are two forms for the **-R** option

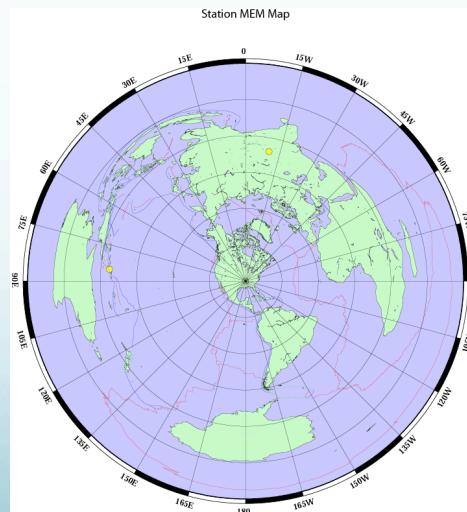
- 2) For regions where the sides do not follow lines of latitude and longitude (will make more sense when we do map projections) - specify corners by appending an “r” to end



The idea of “region” to plot specified this way
breaks down for azimuthal projections

(outside border of plot is a circle, you really want
to specify center and radius)

will see how to do this later.



2) We want ticks, grid and annotation in x every 10, and in y every 1.

This is specified by the **-B** option (Border?).

```
psbasemap -R0/100/0/10 -JX4i/3i -B10/1::."My first plot": -P \
>! plot.ps
```

This is the most complicated GMT option.

Ticks and annotation – every 10 for x (first one)
and every 1 for y (second one).

If you wanted the same ticks and annotation for x
and y you would only have to specify it once.

```
psbasemap -R0/100/0/10 -JX4i/3i -B10/1:"My first plot": -P \
>! plot.ps
```

Not in our specifications, but controlled by the **-B** option, the plot title.

This is a little more complicated.

Labels are between colons, with

“.” for plot title,
nothing for x axis label,
“,” for y axis label.

If label/title is more than one word, has to be in double quotes.

```
psbasemap -R0/100/0/10 -JX4i/3i -B10/1::."My first plot": -P \
>! plot.ps
```

If this sounds confusing you can look at the man page for `psbasemap` for the full explanation and more examples.

The man page for the `-B` option, however, is practically incomprehensible.

The BUGS section of the man page states

“The `-B` option is somewhat complicated to explain and comprehend. However, it is fairly simple for most applications (see examples).”

Remaining options/switches

-P

Sets the output to Portrait (long side vertical) mode.

“Default” is Landscape (long side horizontal) mode.

```
psbasemap -R10/70/-3/8 -JX4i/3i -B10/1::."My first plot": -P \
>! plot.ps
```

This option actually switches “states”.

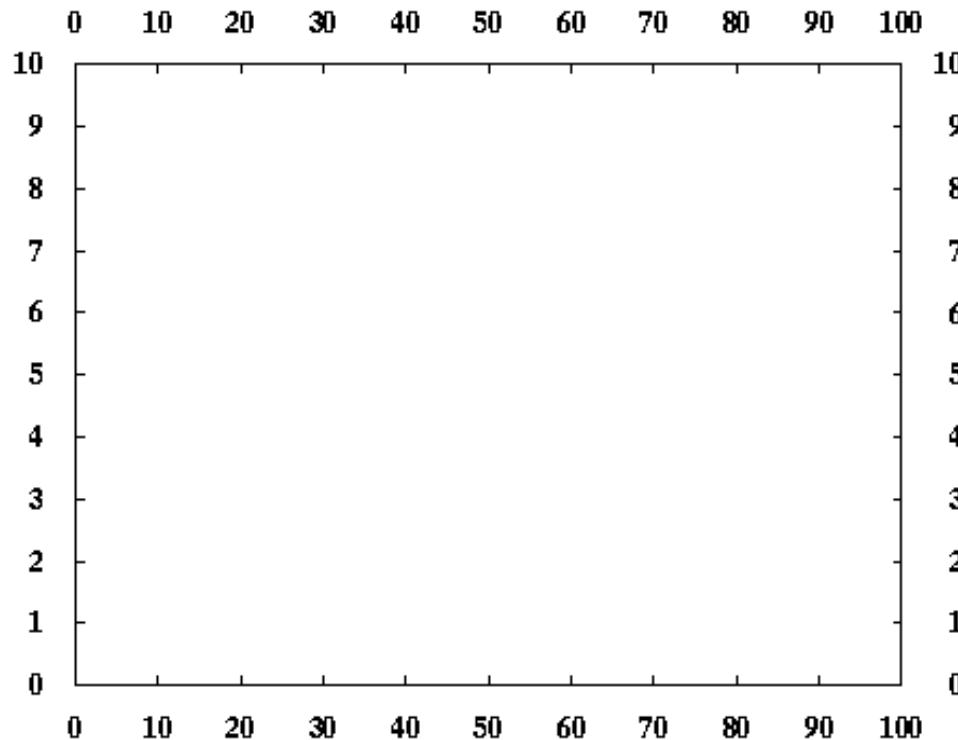
Remaining options/switches

If `.gmtdefaults` defines portrait mode as the default, then `-P` will send it to landscape.

(make a figure and see how it comes out, if you don't like the orientation stick in a `-P`).

So, what did we get for all our effort?

My first plot



Good start – but usually we make plots to show
some sort of data
– so how do we do that?

Now let's look at a little more complicated example:

Lets call it “full_court_press.sh”

```
#!/bin/sh
#plot square root x
sampleId -I1 << END | awk '{print $1, sqrt($1)}' > ${0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P ${0}_1.dat -K > ${0}.ps
sampleId ${0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> ${0}.ps
```

This is more than “a little more” complicated.

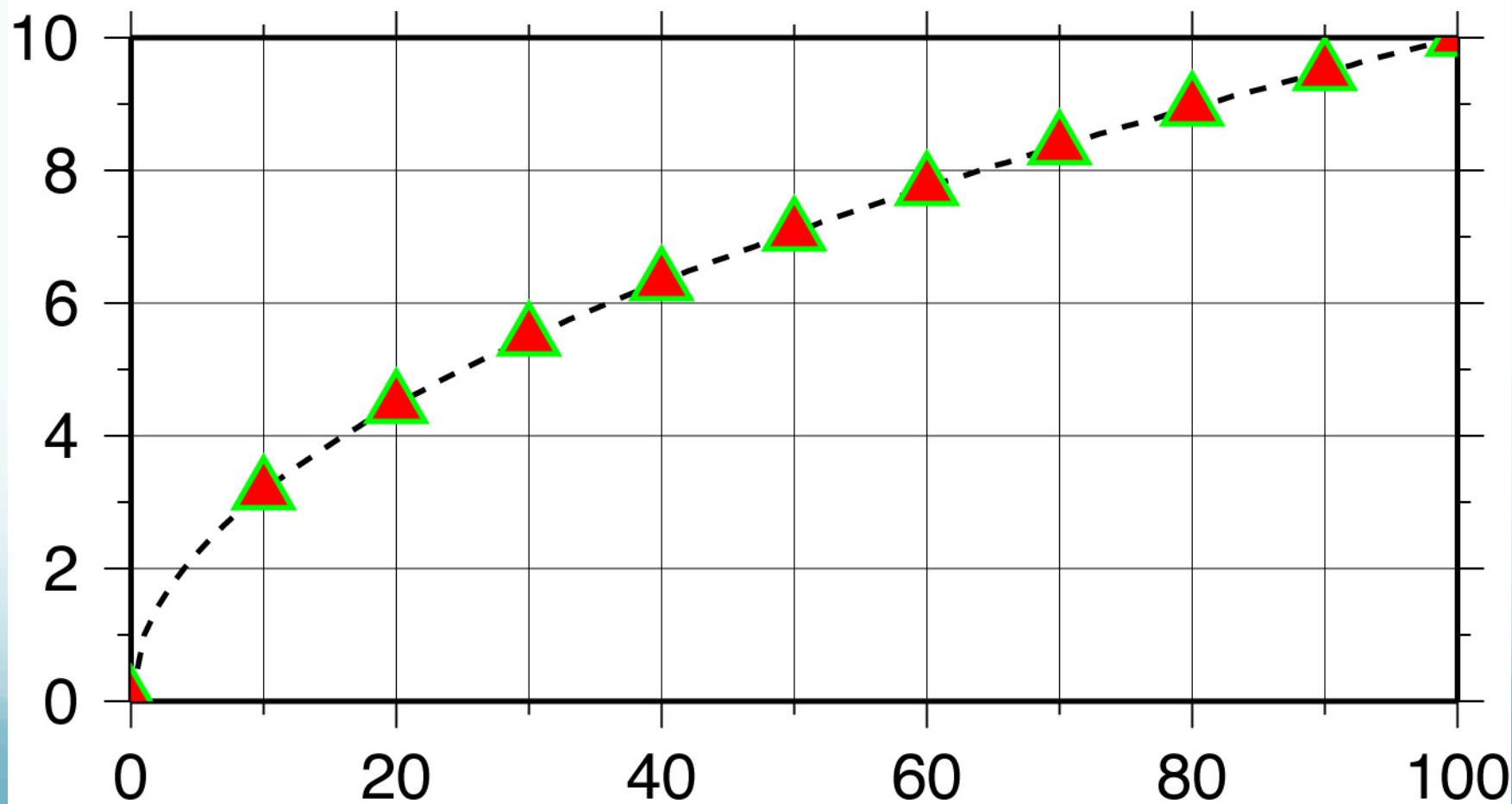
```
#!/bin/sh
#plot square root x
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P ${0}_1.dat -K > $0.ps
sample1d ${0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

But it follows the UNIX philosophy – a bunch of simple things stuck together to do something more complex.

Gives you the idea that most useful GMT produced figures are going to be a LOT of GMT calls

Here's what the output looks like

(actually the output is a ascii file containing a PostScript program, this is what it looks like after displaying with GhostScript or GhostView to the screen or printing to a PostScript printer).



Let's look at it piece, by simple piece.

Set shell

```
#!/bin/sh  
#plot square root x  
sample1d -I1 << END | awk '{print $1,  
...  
...
```

Set shell to Bourne Shell.

Could also have set it to bash or csh

(change first line to #!/usr/bin/csh -f, this works because this script does not contain anything that is specific to one shell script – such as variable name definition. Use -f, fast, option which stops it from running your .cshrc).

Next piece

Name the output file.

```
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -o >> $0.ps
```

Being lazy and disorganized - I don't want to have to type the output file name in lots of times nor keep track of which shell script made which postscript file in my directory.

Next piece

Name the output file.

```
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -o >> $0.ps
```

So I want to find a short and easy way to name the file, and I might want to associate the output file name with the name of the shell script that made it.

Enter UNIX argument passing to the rescue.

When you call a shell script, the system passes predefined, pre-named “arguments” to the shell script from the command line.

So if I enter

“myscript arg1 arg2”

UNIX automatically gives me (in this case 3 arguments)

\$0 the name of the shell script

\$1 the value of arg1 (character string)

\$2 the value of arg2

All I have to do to use these arguments in my shell script (within some constraints) is stick them in.

The Shell will expand them to their proper values.

So my output file will be named

“full_court_press.sh.ps”,

since \$0 will get expanded to

“full_court_press.sh”

(the name of the shell script)

Next piece.

Get (actually make) input data – part 1

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P ${$0}_1.dat -K > $0.ps
sample1d ${$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -o >> $0.ps
```

sample1d, resamples (here interpolates) the input, which in this case is redirected (<<) to being in-line from the shell script (from the end of this command line, which is somewhat far away, to END).

```
#!/bin/sh
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P ${0}_1.dat -K > $0.ps
sample1d ${0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

We have to specify the resampling step (-I1, which is steps of 1).

We will leave everything else at the default values.
(see man page if you want more info)

sample1d provides a list of numbers from 0 to 100 in steps of 1 to stnd out.

Next piece. Get (make) input data – part 2
We want x and \sqrt{x}

```
#!/bin/sh
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P ${0}_1.dat -K > ${0}.ps
sample1d ${0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -o >> ${0}.ps
```

Pipe the re-sampled data into the program `nawk`.

`nawk` is a great tool for preprocessing data for
GMT.

Next piece: Generate input data – part 2

Using awk, one does not have to write programs to make intermediate files in GMT input format, but can go right to the source data file, read it, modify each line into GMT input format and pipe this directly into the GMT program.

```
sample1d -I1 << END | awk '{print $1, sqrt($1)}' | \
psxy -R0/100/0/10 -JX4/2 -Ba20f10g10/a2g2WSNe -W5t15_15:0 \
-Y2 -P -K > $0.ps
```

0 0

100 0

END

Next piece. Generate input data – part 2

```
sampleId -I1 << END | awk '{print $1, sqrt($1)}' | psxy -  
R0/100/0/10 \  
-JX4/2 -Ba20f10g10/a2g2WSne -W5t15_15:0 -Y2 -P -K > $0.ps  
0 0  
100 0  
END
```

The awk command says to print the first column (\$1) and the square root of the first column (sqrt (\$1)) of every line.

We will (break the UNIX philosophy and) make an intermediate file as we will need it more than once.

Next piece.

Plot it

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2wSne -W5t15_15:0 \
-Y2 -P ${$0}_1.dat -K > ${$0}.ps
sample1d ${$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> ${$0}.ps
```

Finally we get to the graphics part of GMT

psxy is the GMT program that plots points and lines.

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSNe -W5t15_15:0 \
-Y2 -P ${0}_1.dat -K > ${0}.ps
sample1d ${0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> ${0}.ps
```

psxy accepts the “standard”/”global” options of the GMT filters that produce PostScript output.

(one can put all the plot/map specifications into the first GMT call rather than use psbasemap, everything we said before holds, so we don't have to re-say it.)

We already know what **-R**, **-J**, **-B** and **-P** do, although the **-B** option here is a bit more complicated looking.

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P ${0}_1.dat -K > $0.ps
sample1d ${0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Output file `$0.ps` (is first instance so use `>`,
append in second instance so use `>>` – this takes
care of UNIX part)

Use \ to continue command on next line

Next piece.

```
...  
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \  
-Y2 -P {$0}_1.dat -K > $0.ps  
...
```

So, what's all that extra stuff on the **-B**?

Each of the letters controls a different feature/
aspect of the plotting of the axis

```
...  
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \  
-Y2 -P {$0}_1.dat -K > $0.ps  
...
```

a is for annotation spacing

g is for grid spacing

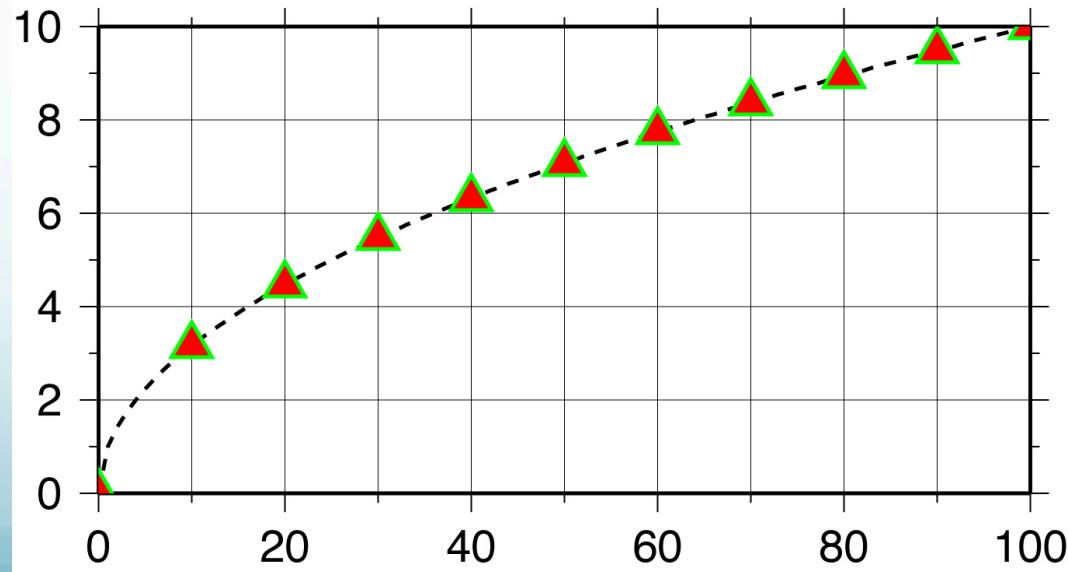
WSNe says to plot the annotation and ticks on the
West and South sides and ticks only on the North
and East sides.

(how would you put annotation without ticks?)

Next piece.

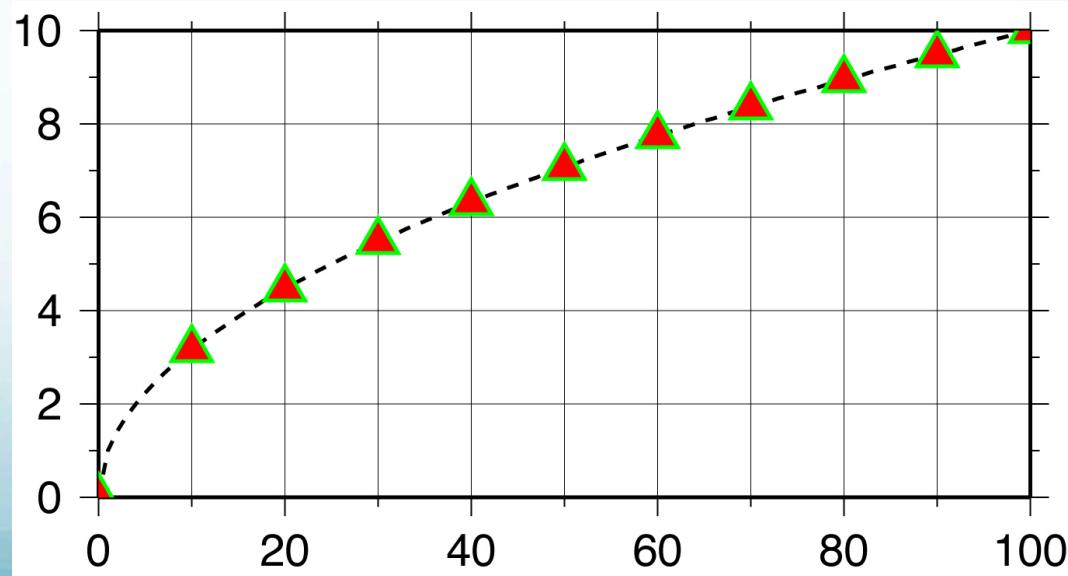
Draw a line **-W5t15_15:0**

```
...
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P {$0}_1.dat -K > $0.ps
. . .
```



Make line 5 units thick (where units depends on the device and default settings)

-W5t15_15:0



Can also specify color

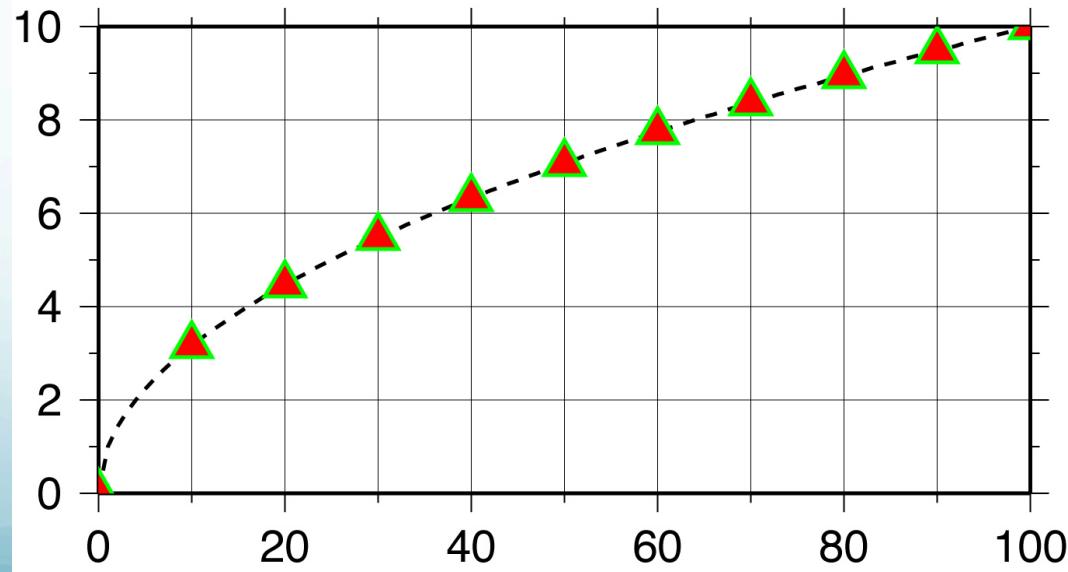
$\textcolor{black}{-W5/0t15_15:0}$ for black line or

$\textcolor{red}{-W5/255/0/0t15_15:0}$ for red line

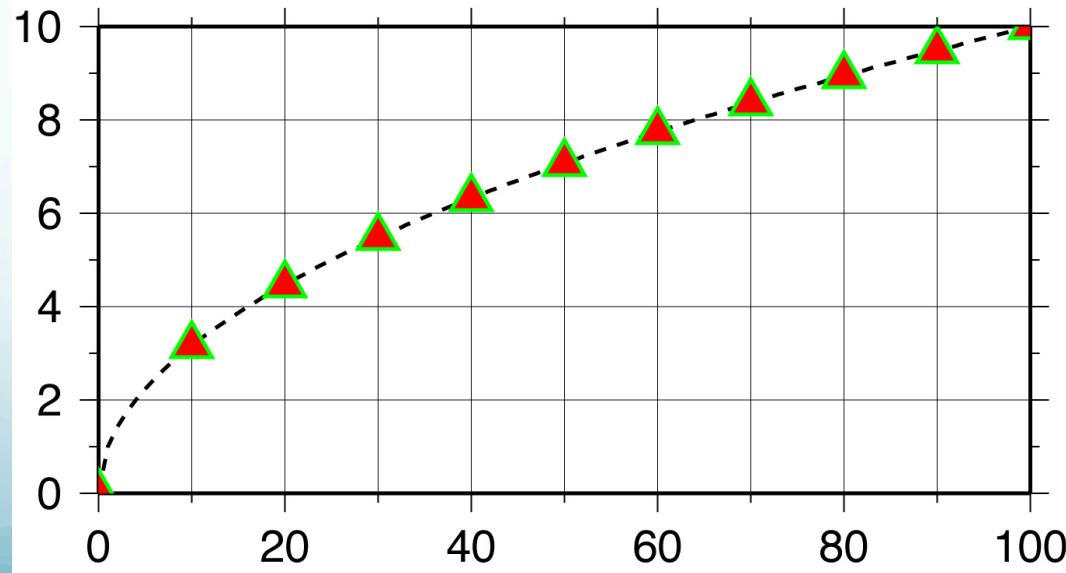
...

```
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne  $\textcolor{red}{-W5t15\_15:0}$  \  
-Y2 -P {$0}_1.dat -K > $0.ps
```

... .



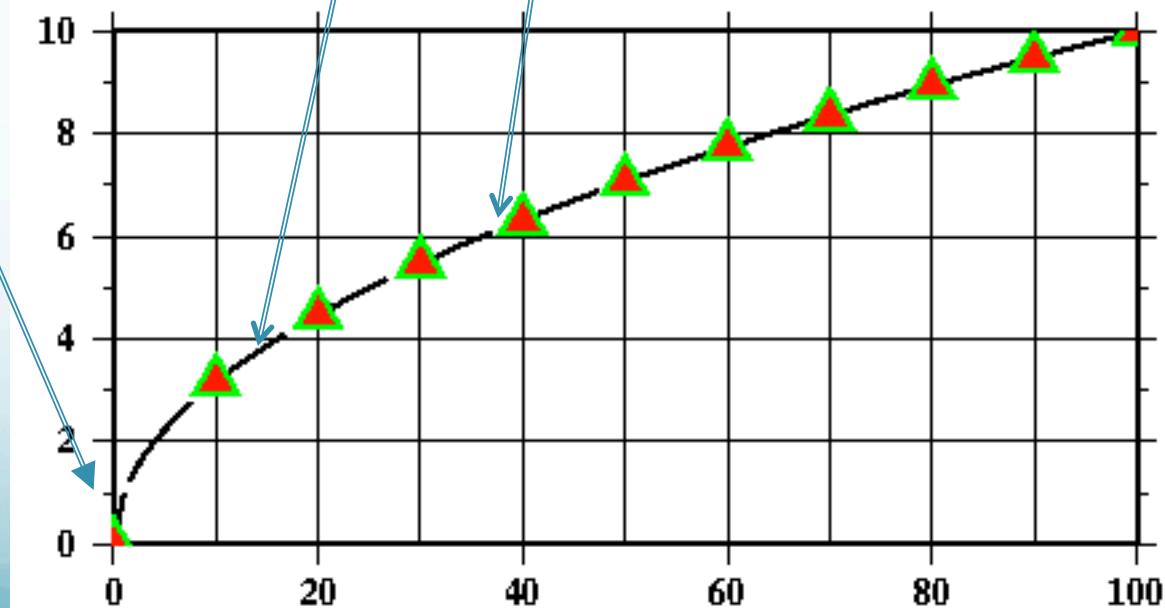
Make it dashed with dashes 15 units long followed by 15 unit long open spaces -`w5t15_15:0`, and a “phase offset” for the dashes of zero -
`w5t15_15:0`



What is “phase offset”

compare to -W5t120_15:60

(line segment 120 long, 15 blank, first line segment only 60 long [phase])



This format also works

`W[width],[color],[texture]`

`-W5,0,15_15:-`

`-W5,255/0/0,15_15:-`

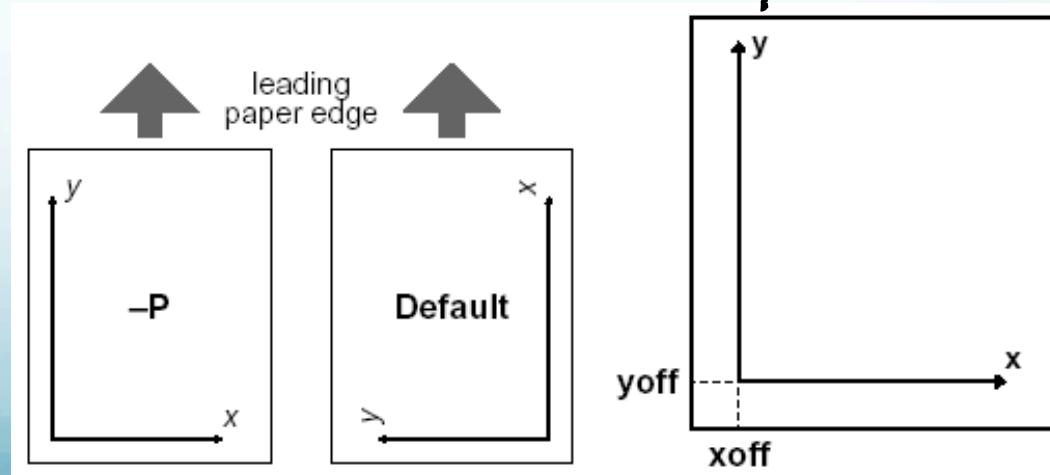
`-W5,red,15_15:-`

Gives the same plot.

Next piece. Misc. 1

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSNe -W5t15_15:0 \
-Y2 -P ${$0}_1.dat -K > $0.ps
sample1d ${$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -o >> $0.ps
```

-Y2 offset plot 2 units in the y direction (else x axis labels get cut off across bottom of plot)

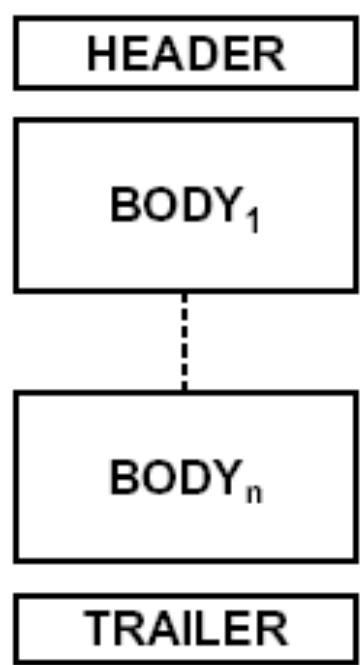


Misc. 2

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P ${0}_1.dat -K > $0.ps
sample1d ${0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

-K do not close PostScript file

-O do not initialize PostScript



-O ommits the header.

2nd trough n-1'th overlays
require both -O and -K.

-K do not close
PostScript file (don't output
"showpage") so more

PostScript can be
appended to the file

-o do not initialize
PostScript

(does not output PostScript header
stuff) so this can be

appended to existing
 file (that hopefully
 does not have a
 showpage at the end).

Misc. 3

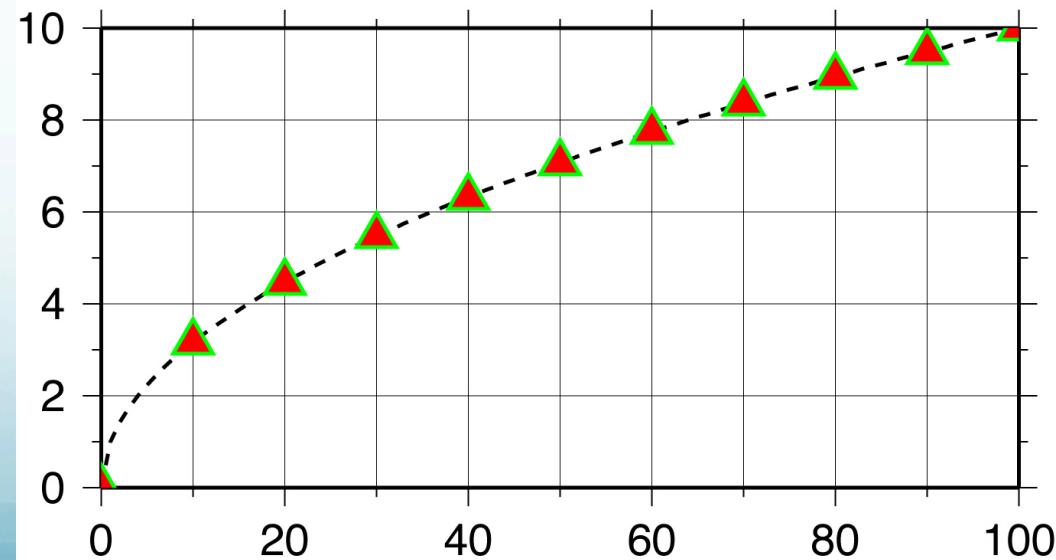
Several common “gotchas”

- no showpage (can see on screen, but does not print – actually prints a blank page) (error: have a `-K` in last GMT call)
- showpage in middle of file (error: forgot the `-K` somewhere) – only get part of file on screen or in final print or get ghostscript error message.
- Have header in middle of file (error: forgot `-o` somewhere), get ghostscript error message.

Next piece.
Draw symbols every 10th point

```
#!/bin/sh
#plot square root x
sample1d -I1 << END | awk '{print $1, sqrt($1)}' > ${$0}_1.dat
0
100
END
psxy -R0/100/0/10 -JX4/2 -Ba20g10/a2g2WSne -W5t15_15:0 \
-Y2 -P ${$0}_1.dat -K > $0.ps
sample1d ${$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \
-G255/0/0 -W5/0/255/0 -O >> $0.ps
```

Resample our temporary file – taking every 10th point (-I10). Pipe output to psxy

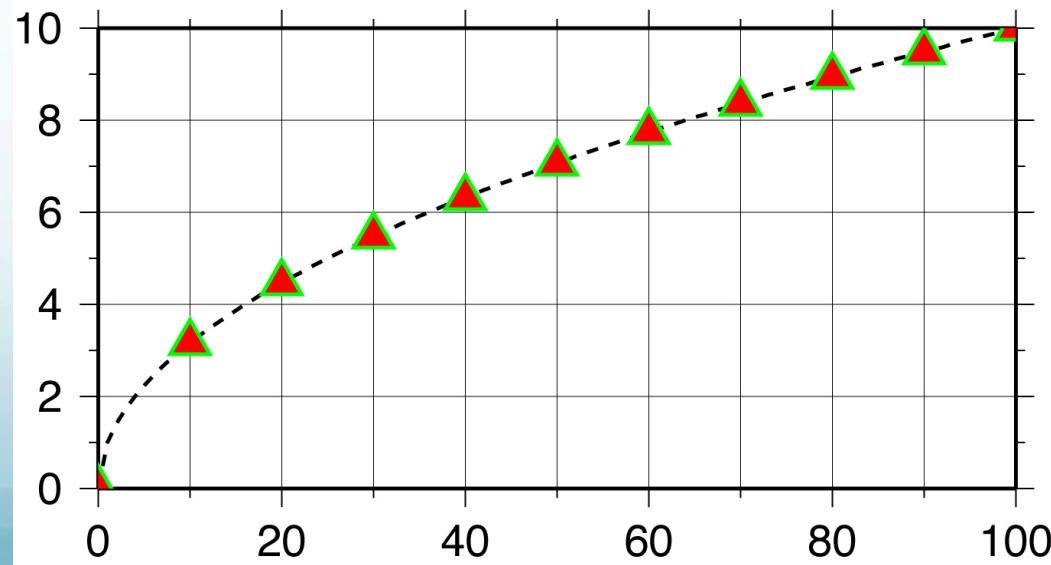


Next piece.
Draw symbols **-St0.2**

...

```
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \  
-G255/0/0 -W5/0/255/0 -o >> $0.ps
```

Make triangles, t, that are 0.2 units big -
St0.2



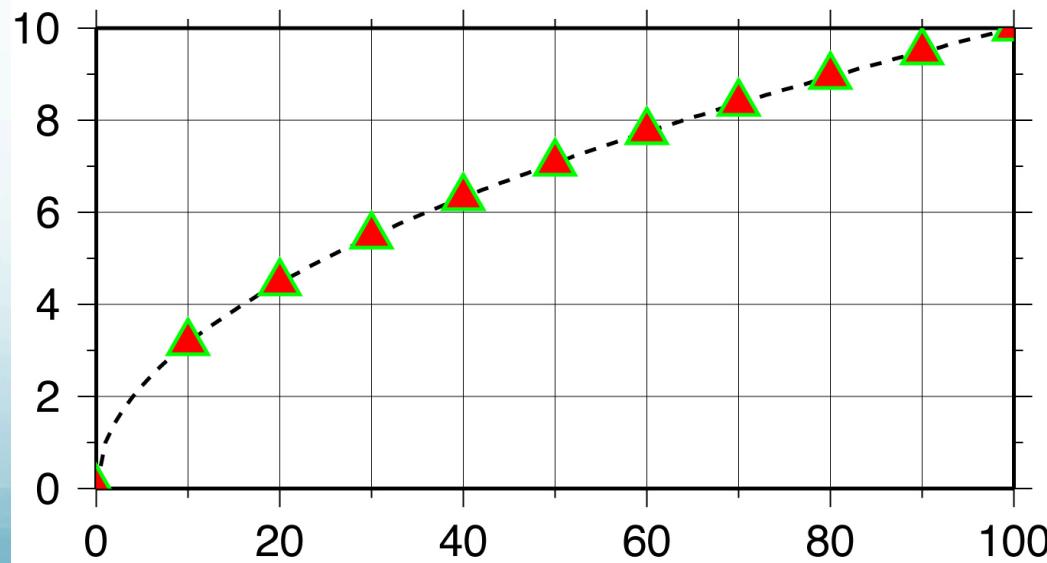
Next piece. Draw symbols

```
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \  
-G255/0/0 -W5/0/255/0 -o >> $0.ps
```

Make the line outlining/drawing the symbols, -W flag, that is 5 units wide, and draw outline in green (R/G/B) -W5/0/255/0 or W5/green

Colors specified in
R/G/B format

(intensity of Red, Green and Blue
color guns – primary colors for
additive system).

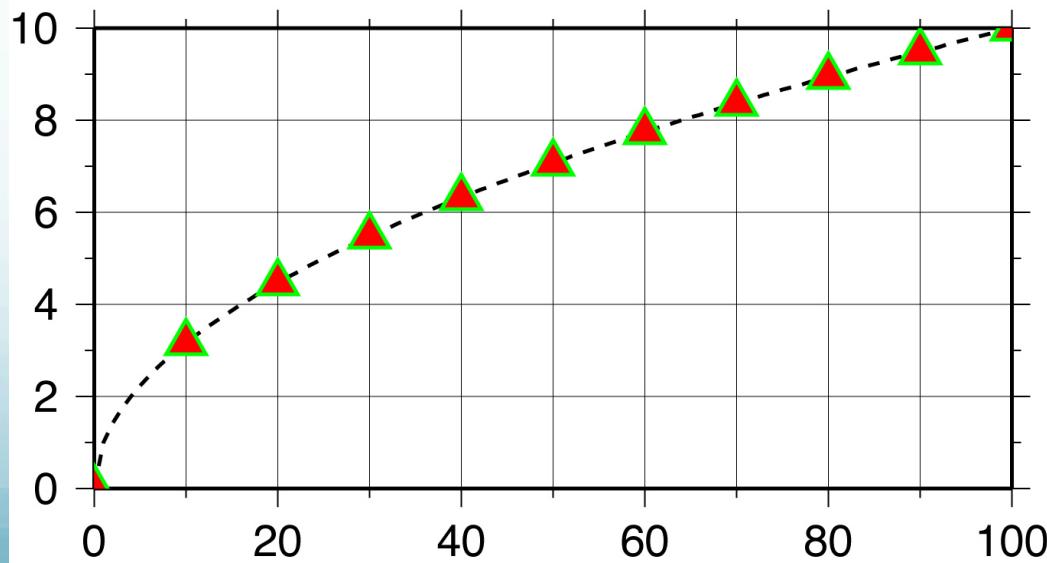


Next piece. Draw symbols

```
sample1d {$0}_1.dat -I10 | psxy -R -JX4/2 -St0.2 \  
-G255/0/0 -W5/0/255/0 -o >> $0.ps
```

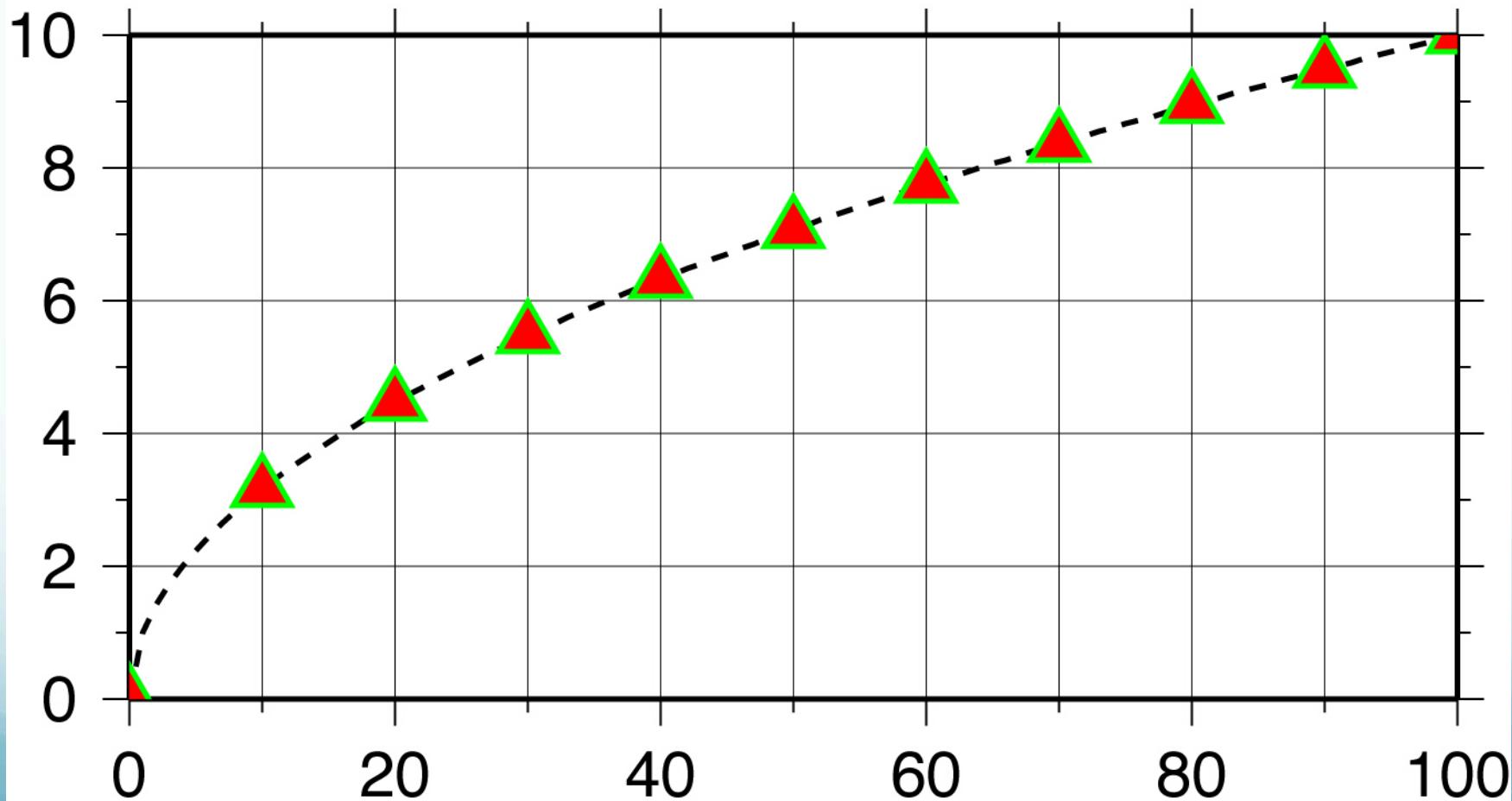
Fill the symbol, -G flag, with color red, -G255/0/0, or -Gred

Can also use color names (red, green, blue. There are over 700 X-11 color names).



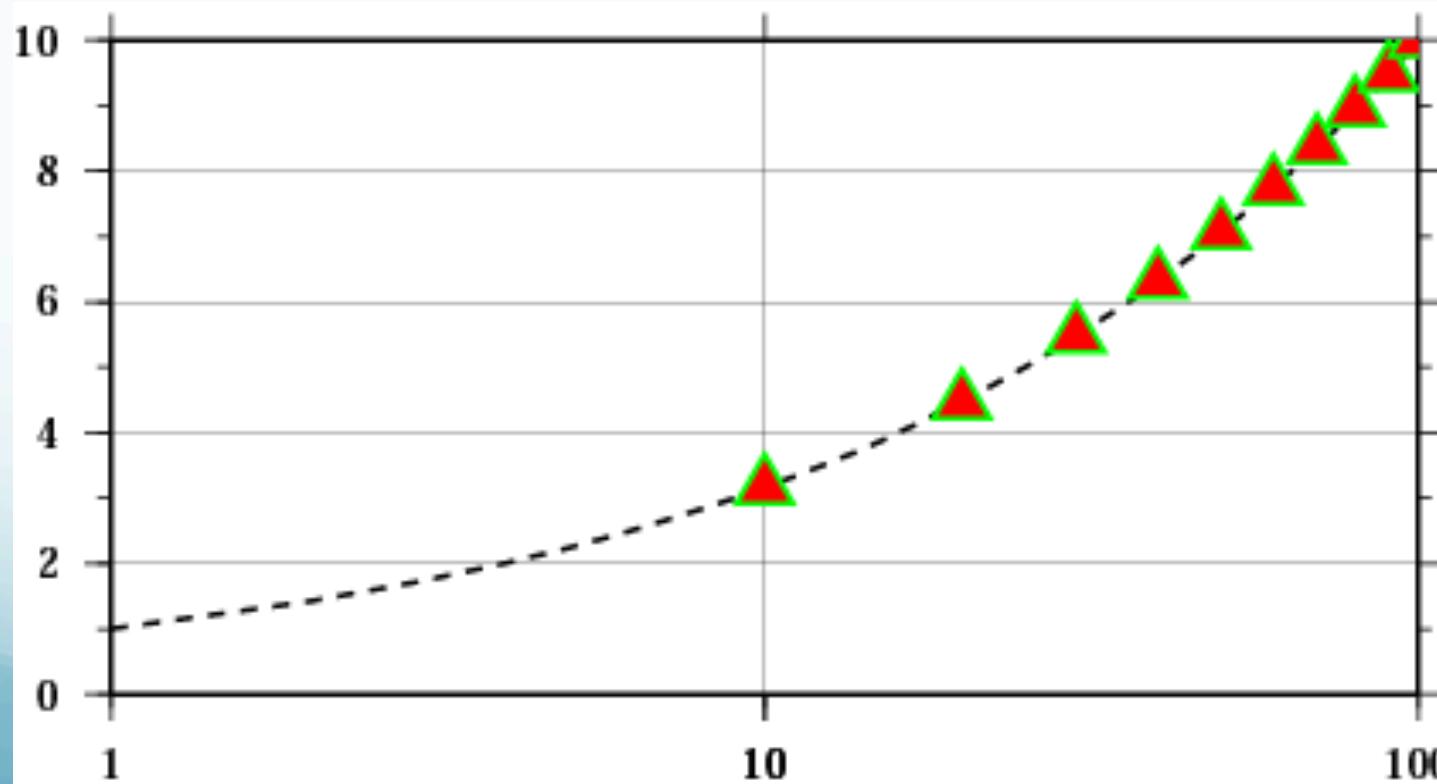
We're done!

That wasn't so bad now, was it?



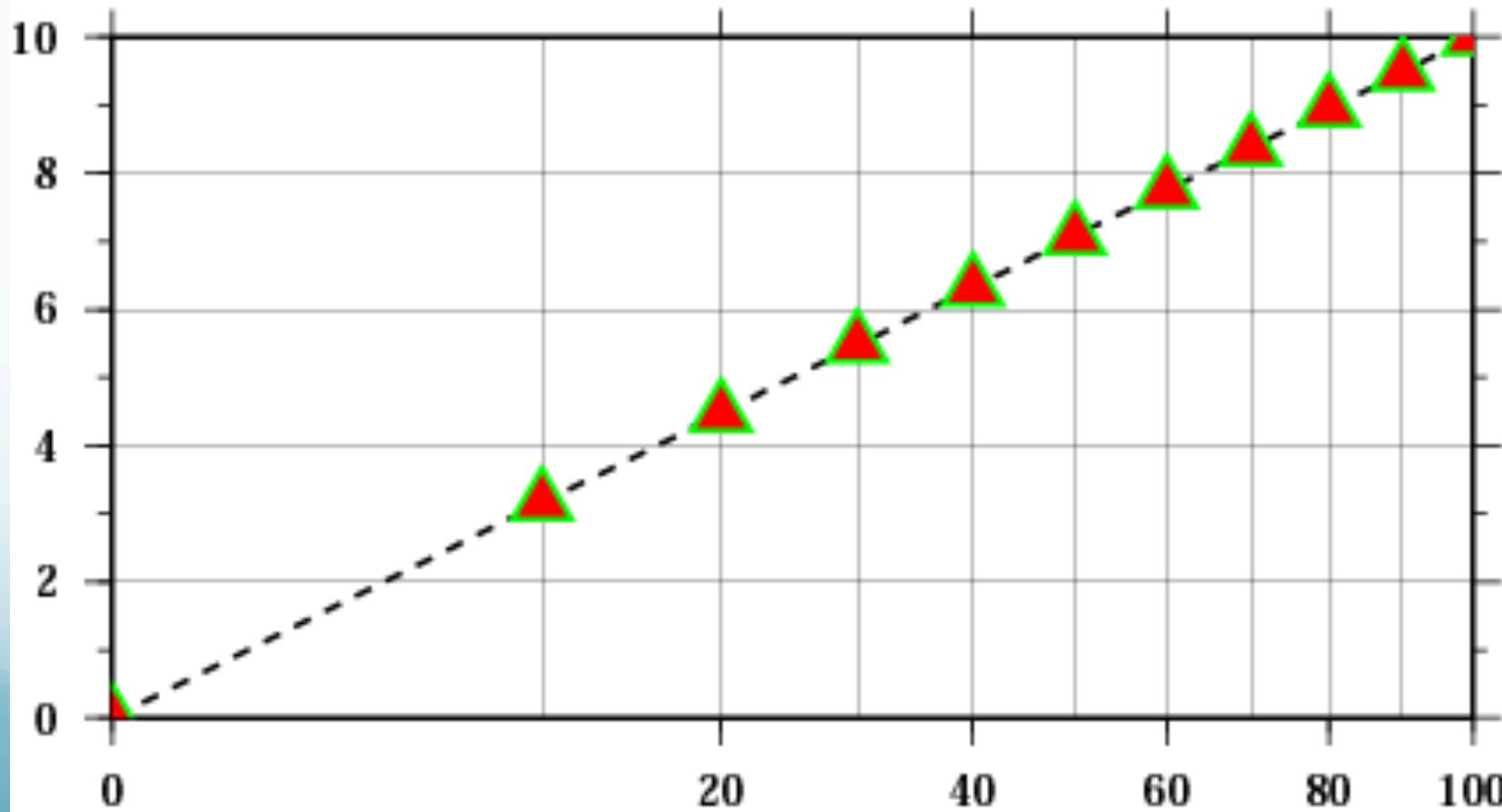
There are two other non-map-projected forms

- 1) Logarithmic - add l (lower case letter L) after scale of axis you want logarithmic - JX41/2



2) Power/exponential – add p and exponent after scale of axis you want exponential (can scale axes individually)

$$-JX4p0.5/2$$



Common command options on first, and possibly subsequent, calls

Need on all calls

- R Define region for plot – will need on first call and at least “-R” on subsequent
- J define projection for plot – will need this on all calls if need to define region

Common command options on first, and possibly subsequent, calls

(Generally) Need on first call only

- B Borders -- annotation, frame, grid. Only need on first (or a single) call.
- P Switch between landscape and portrait modes
 - X Shift X axis
 - Y Shift Y axis

Common command options on first, and possibly subsequent, calls.

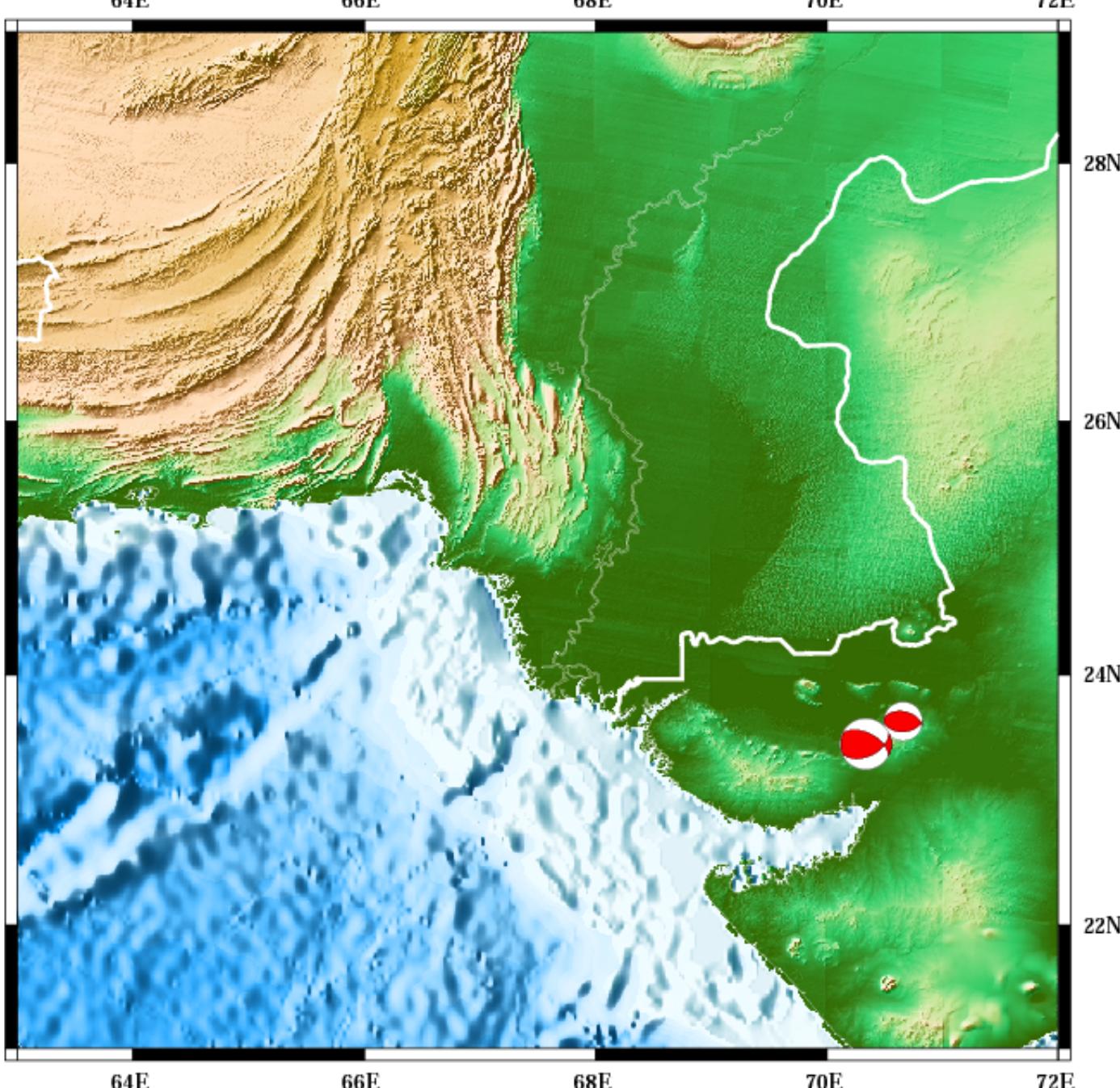
Need when needed.

- K Don't close PostScript (showpage), use when more will follow
 - need on all but last GMT call
- o Don't initialize PostScript, use when appending to pre-existing file
 - need on all but first GMT call
 - use both -K and -o when putting a large number of GMT call outputs together

Common command options on first, and possibly subsequent, calls.

Need when needed.

- v Verbose (prints out stuff to standard error for user).
- H Header records (tells GMT to skip first H lines of ascii input file)



How about
making
pretty
MAPS?
(this was
made by
the shell
script I put
in Mitch's
GMT -ToT
web page.)

Map projections available in GMT

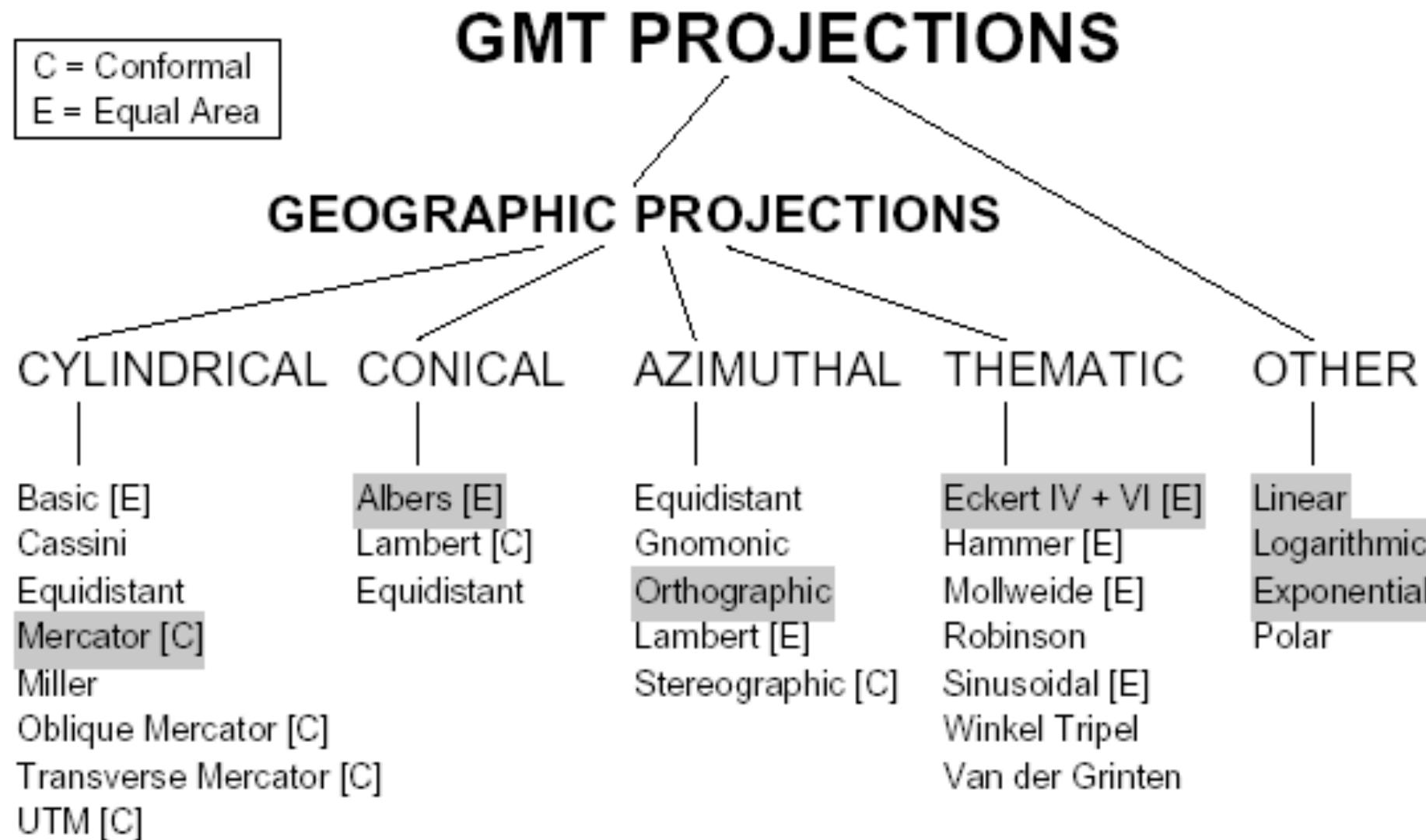


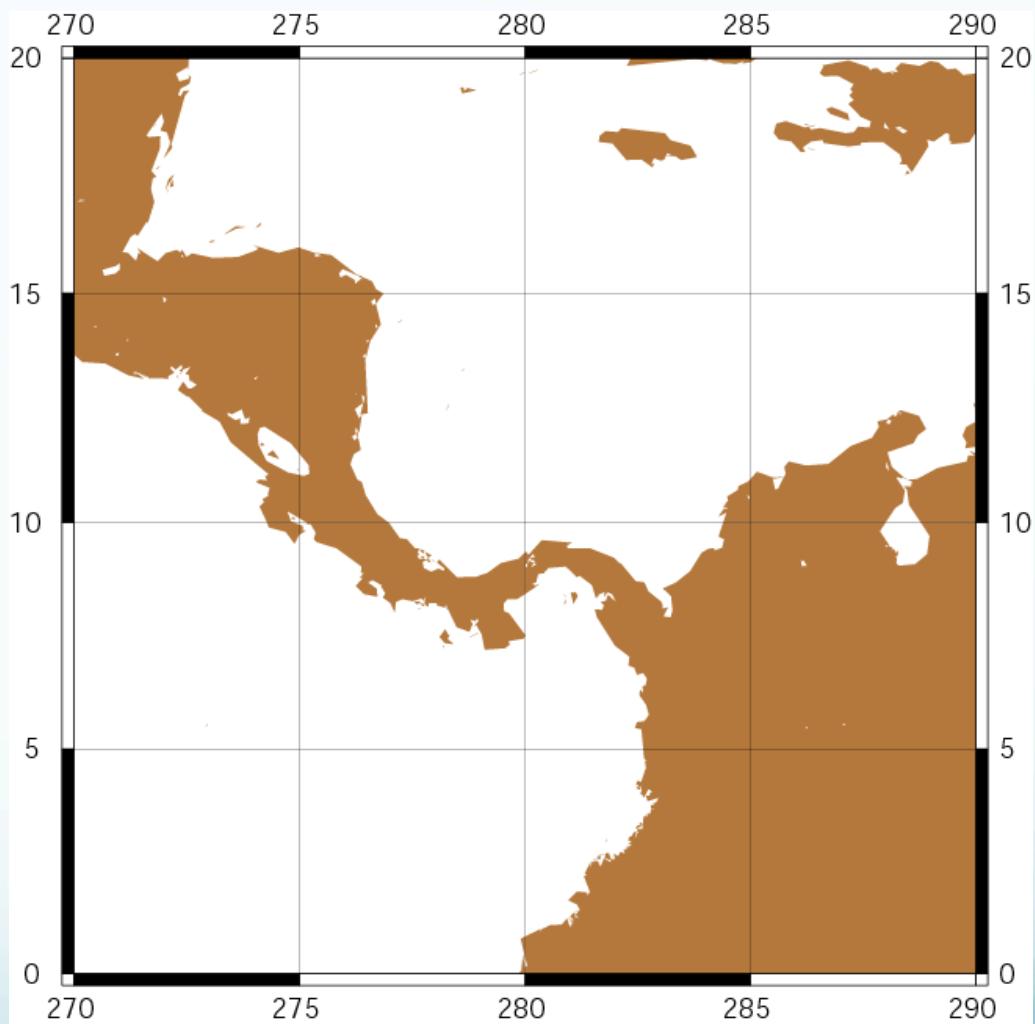
Figure 1.9: The 25 projections available in **GMT**.

List of
“standard”
command line
options.
The **-J** option
sets the
“projection”
One has to
look at the man
page for each
one as
“different
things vary”

STANDARDIZED COMMAND LINE OPTIONS	
-Bxinfo[yinfo/zinfo][WESNZwesnz+][.title:]	Tickmarks. Each info is [a]tck[m]e [f]tck[m]e [g]tck[m]e][l]p [l]label [.][,][u]nit]
-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
-JK flon 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
-JO alon ₀ /lat ₀ /az/width	Oblique Mercator, 1: origin and azimuth
-JO blon ₀ /lat ₀ /lon ₁ /lat ₁ /width	Oblique Mercator, 2: two points
-JO clon ₀ /lat ₀ /lon _p /lat _p /width	Oblique Mercator, 3: origin and pole
-JP [awidth[.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[lp][height[lp]][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

```
pscoast -R-90/-70/0/20 -JM6i -P -B5g5 -G180/120/60 > map1.ps
```

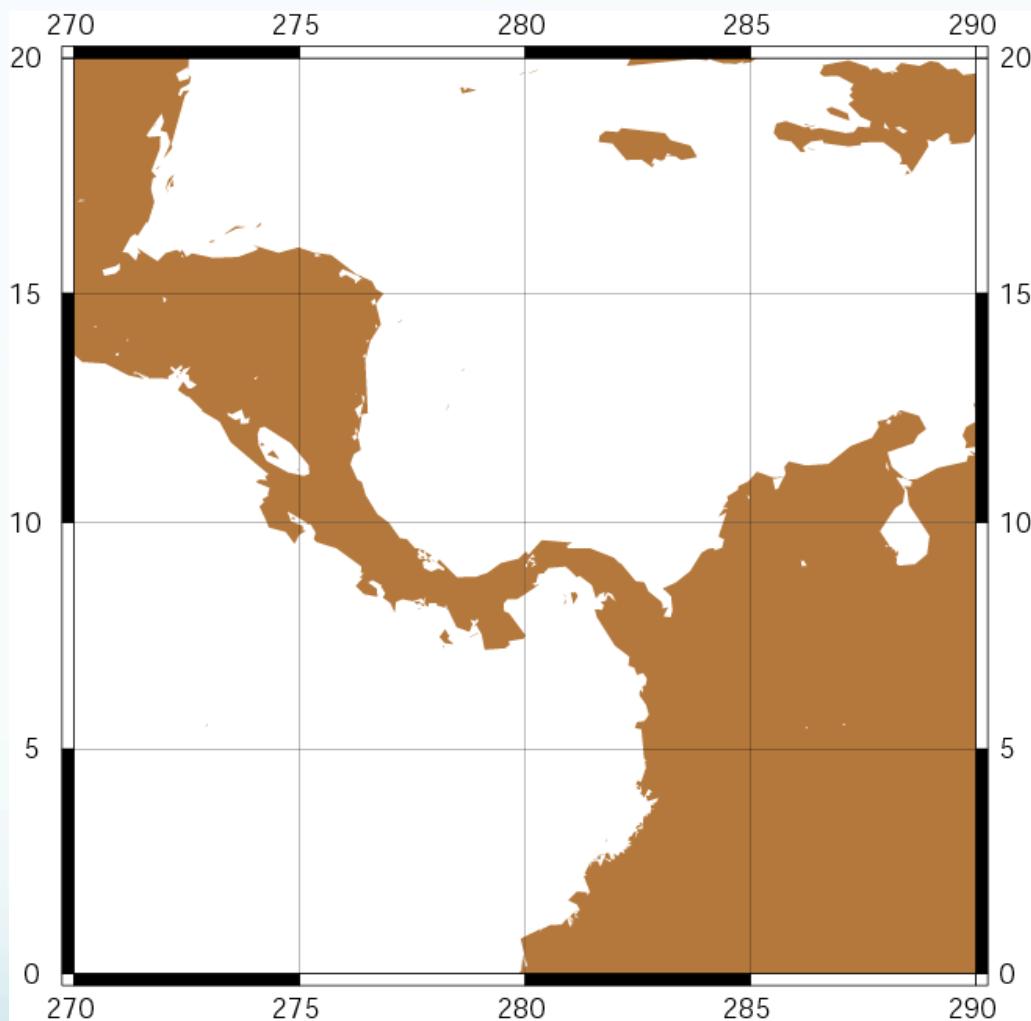
“All” gmt programs
plot “maps” through
the projection
command line option
or switch (even the x-y plot).



```
pscoast -R-90/-70/0/20 -JM6i -P -B5g5 -G180/120/60 > map1.ps
```

All projections give
you two selections for
specifying the scale

(note GMT takes the mapmakers attitude that a map has to have a predetermined/known scale – assuming you want the map to nicely fill the page does not cut it – a map without an explicitly known or specified scale is inconceivable.)



```
pscoast -R-90/-70/0/20 -JM6i -P -B5g5 -G180/120/60 > map1.ps
```

-Jmparameters

(Mercator).

Specify one of:

-Jmscale or **-JMwidth**

Give scale along
equator

(1:xxxx or UNIT/degree).



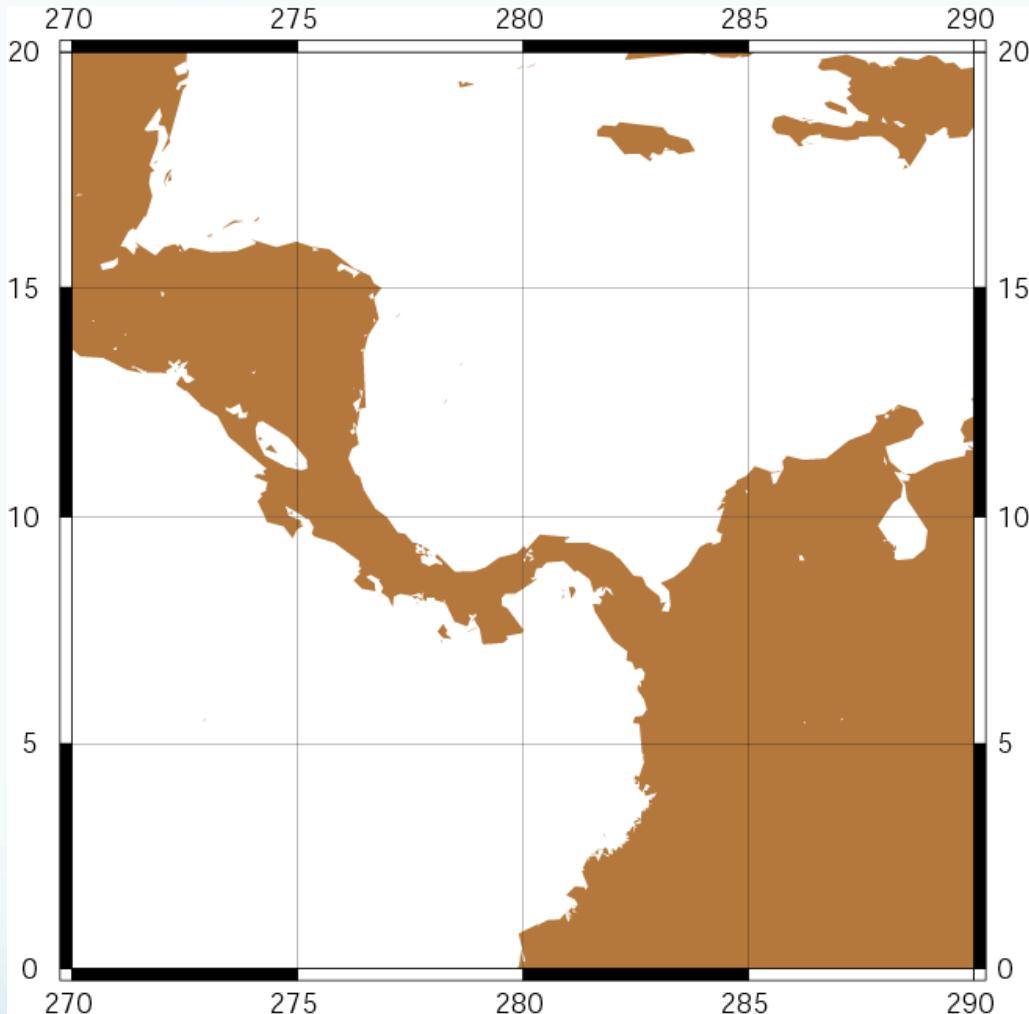
```
pscoast -R-90/-70/0/20 -JM6i -P -B5g5 -G180/120/60 > map1.ps
```

-Jmlon0/lat0/scale or

-JMlon0/lat0/width

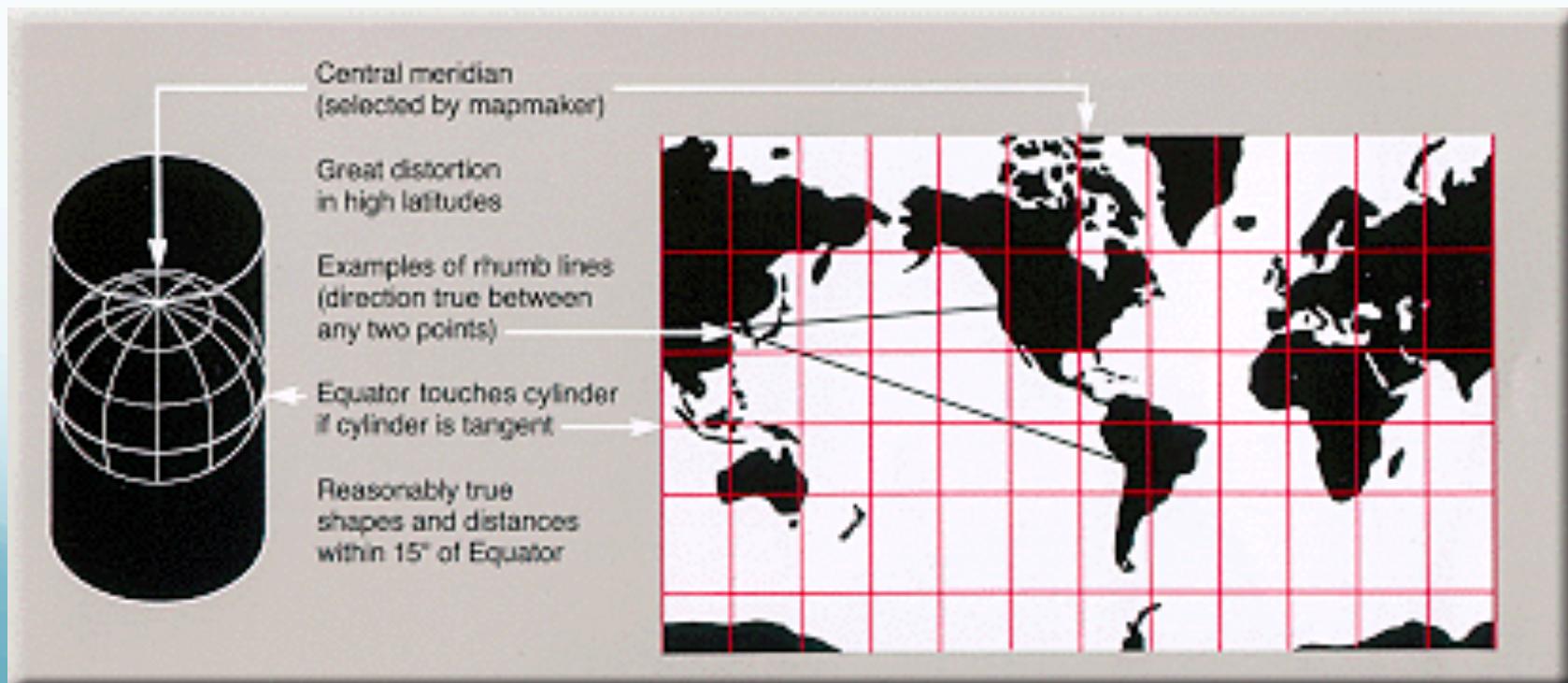
Give central
meridian, standard
latitude and scale
along parallel

(1:xxxx or UNIT/degree, UNIT
= number inches or cms).



Map Projection:
address plotting sphere on a plane

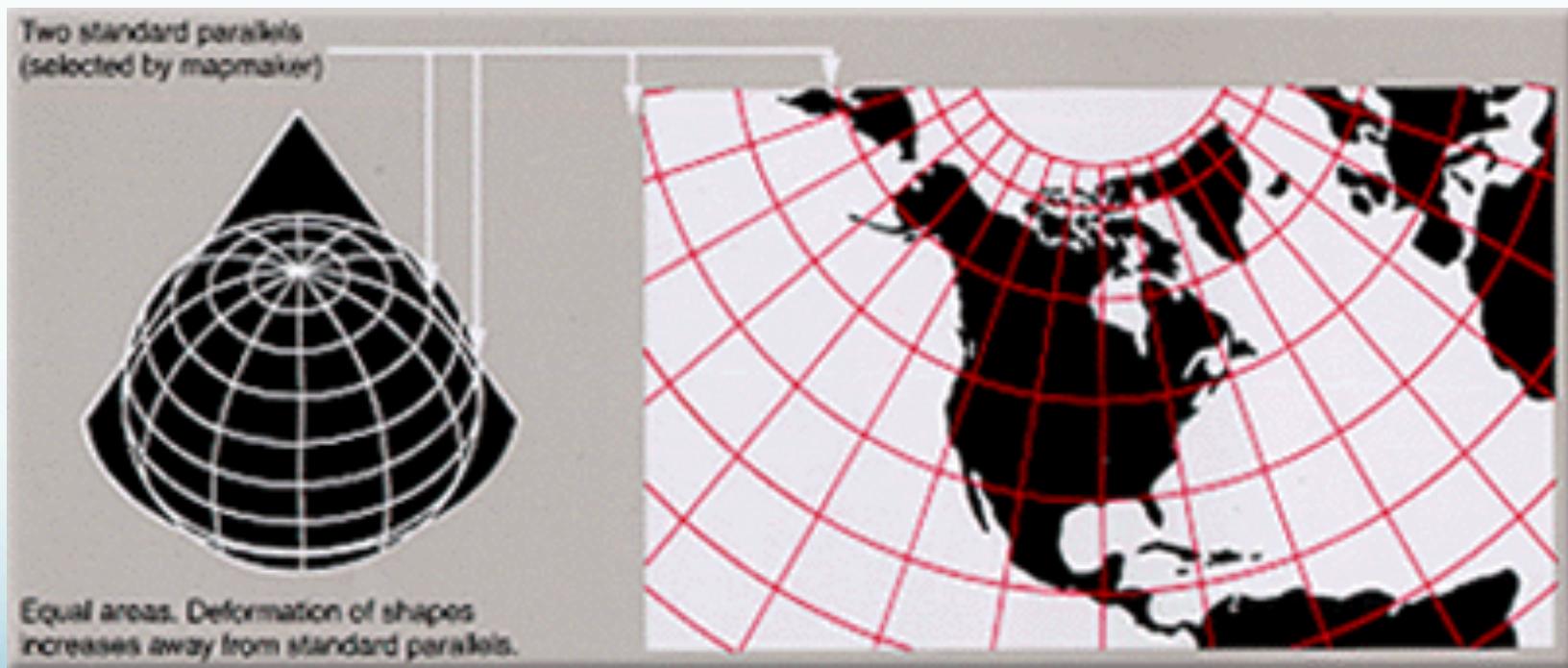
Mercator Projection:
One way to address plotting sphere on a plane
(which is whole 'nother subject)
Conformal (maintains shapes but not relative sizes)
Cylindrical projection



Albers

Also conformal (maintains/conserves shape)

Conical projection

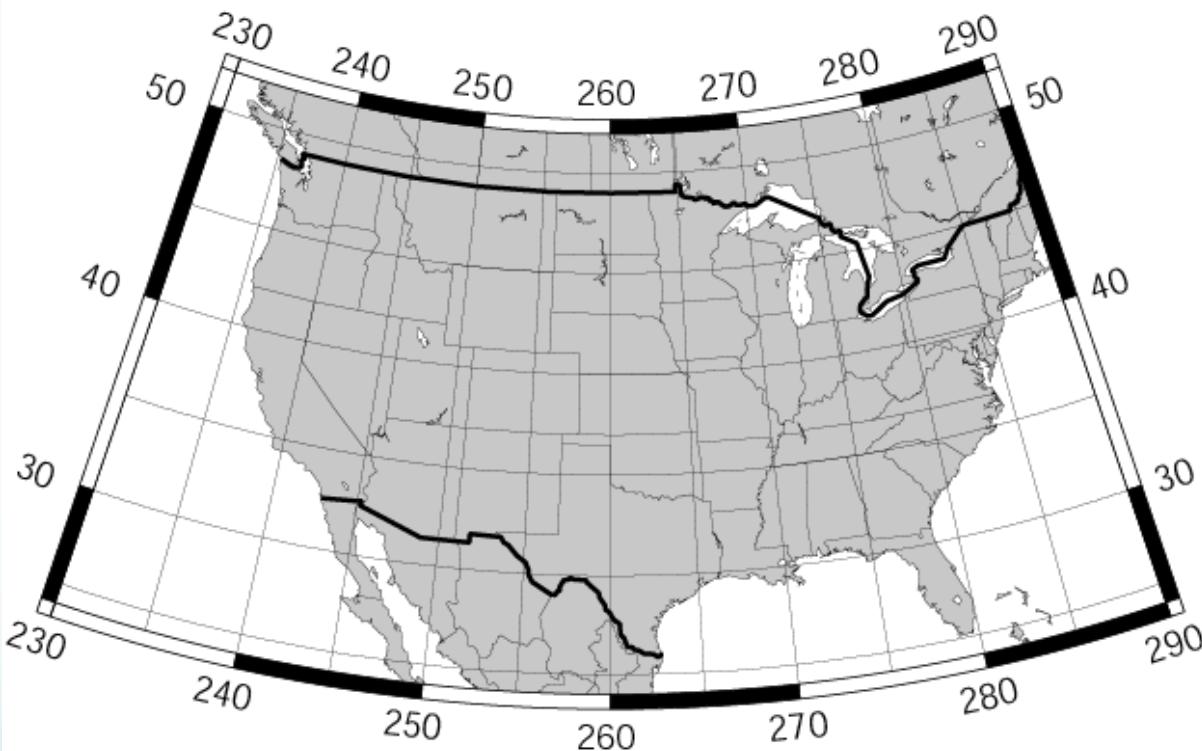


```
pscoast -R-130/-70/24/52 -JB-100/35/33/45/6i -B10g5:. "Conic\ Projection":  
-N1/2p -N2/0.25p -A500 -G200 -W0.25p -P >! map.ps
```

Region is
“rectangle” on
the spherical
earth.

-N for
boundaries
(international, US/Canadian/
Mexian state boundaries
“built in”), rivers.

Conic Projection

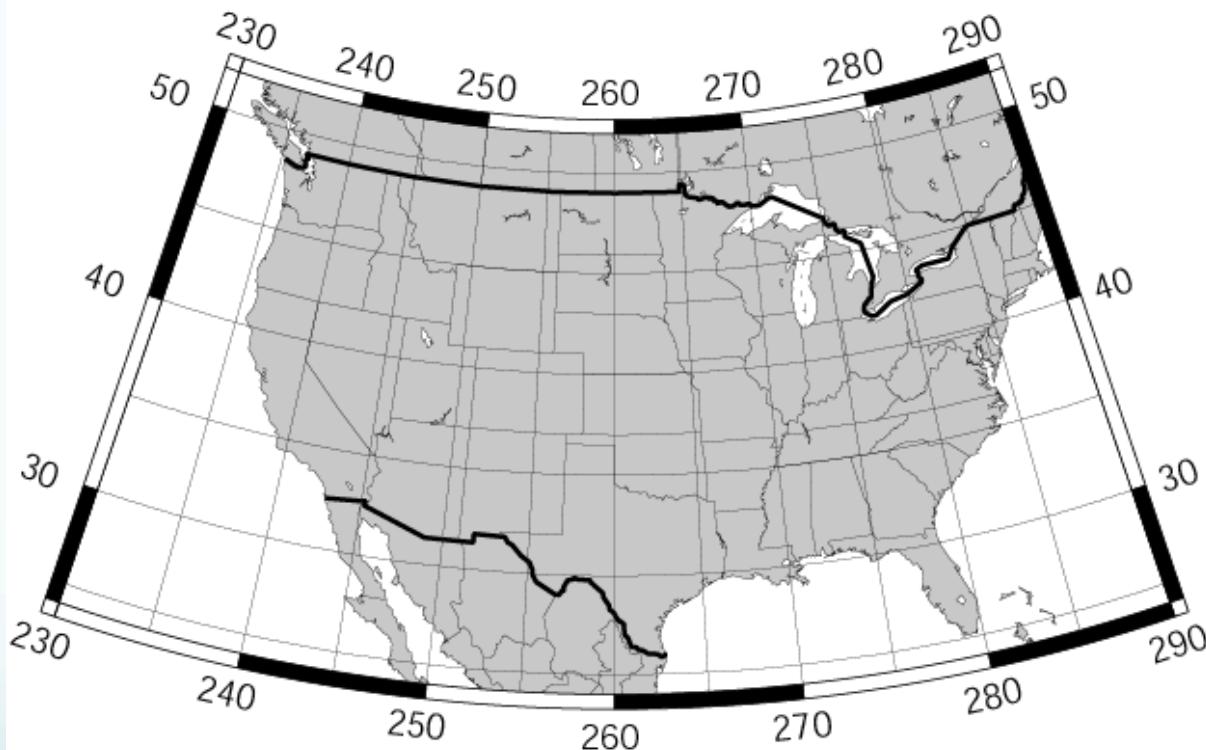


```
pscoast -R-130/-70/24/52 -JB-100/35/33/45/6i -B10g5:. "Conic\ Projection":  
-N1/2p -N2/0.25p -A500 -G200 -W0.25p -P >! map.ps
```

-A to get rid of
small water/
island features

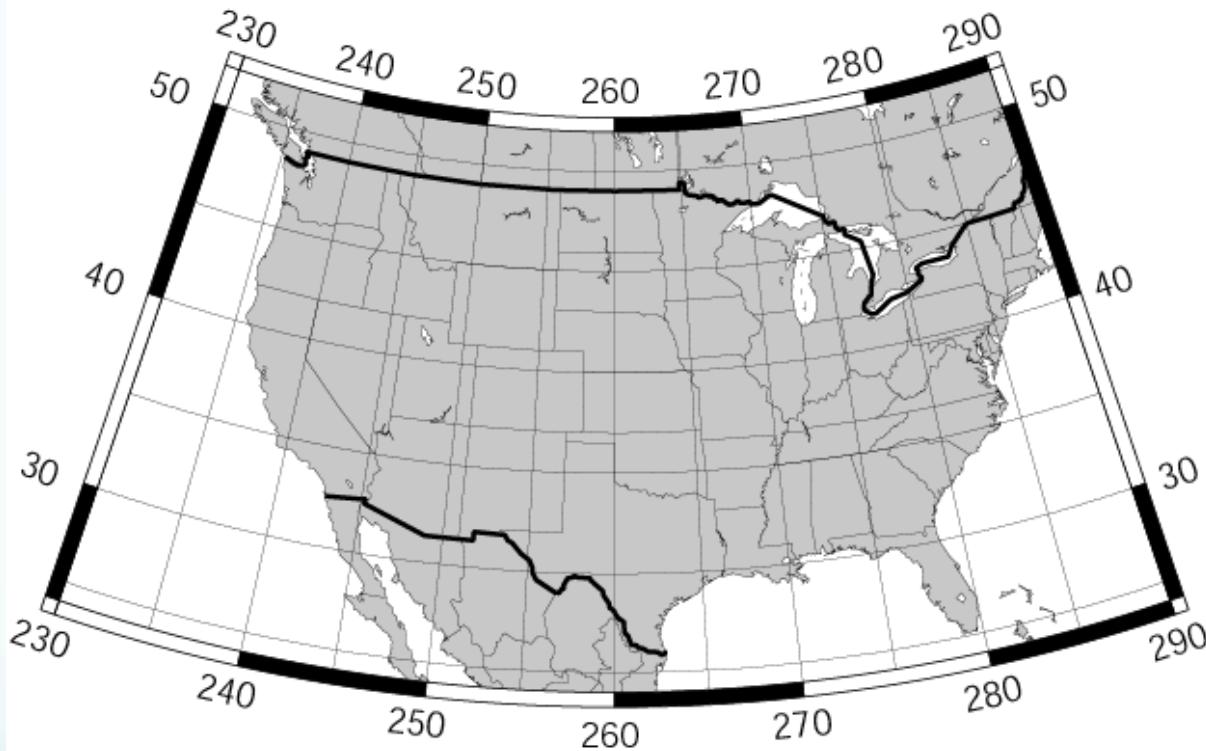
Projection (b/
B) – need to
know
something
(center and
standard
parallels).

Conic Projection



```
pscoast -R-130/-70/24/52 -JB-100/35/33/45/6i -B10g5:"Conic\ Projection":  
\ -N1/2p -N2/0.25p -A500 -G200 -W0.25p -P >! map.ps
```

Conic Projection



-Jblon0/lat0/lat1/lat2/scale or -JBlon0/lat0/lat1/lat2/width
(Albers [E]). Give projection center, two
standard parallels, and scale (1:xxxx or UNIT/degree).

```
pscoast -R0/360/-90/90 -JG280/30/6i -Bg30/g15 -Dc -A5000 \
-G255/255/255 -S150/50/150 -P >! map.ps
```

Other projections –

azimuthal orthographic
(projection mimics
looking at earth from
infinite distance).



```
pscoast -R0/360/-90/90 -JG280/30/6i -Bg30/g15 -Dc -A5000 \  
-G255/255/255 -S150/50/150 -P >! map.ps
```

New option

-Dc

Controls resolution of
coastline

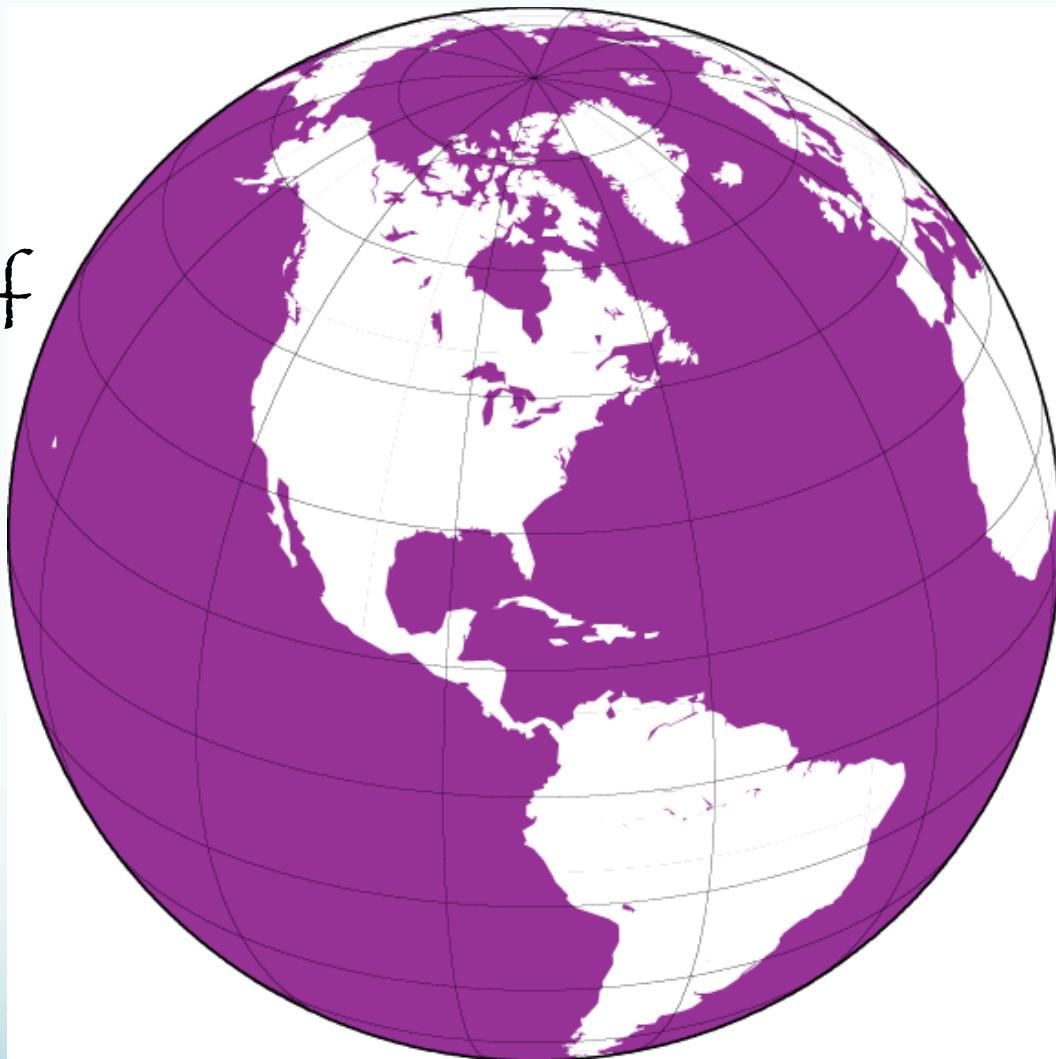
f full

h high

l low

c crude

Helps manage file sizes.



Some useful
maps.

The world
centered on
Memphis.

Use to get
back azimuth
and distance to
earthquakes at
a glance.

Station MEM Map

