Plotting and Graphics

- Chapter 8: Windows, graphic keywords, plot text, titles and axis labels, data dimensionality, and saving figures
- In short, playing with pictures

Data Dimensionality

- An array is one-dimensional:
  - e.g. `x = [5,4,3,6,1,3,2]`
  - However, if you plot it on a graph, you get a two-dimensional plot (an X-axis and a Y-axis)
- An image is two-dimensional, but the normal display has a 3rd dimension: color

Clickers

Are the clicker questions...

A) Too hard

B) Too easy

C) Too frequent

D) Too infrequent

E) None of the above. Click click click.

What's up with this array?

```
 5.36200e+07   1.76657e-30   2.43073e-37 -9.00800e-35
-4.34209e+24   6.98694e-25   7.35844e-18 -2.36351e+30
```

A) It could be (part of) a spectrum

B) It could be a list of solar luminosities

C) It looks like it might have the wrong endianness

D) It looks like something generated by `randomu`

E) Nothing's up with it. It just is.

Plot Windows

- Any time you enter the '`plot`' command, a window shows up with the title "`IDL 0`"
  - This implies that there could be `IDL >0`. There is such a thing!
  - The `window` command will make a new one

window

- You can specify the size and the window number
  - `window,xsize=700,ysize=600`
  - `window,1,xsize=200,ysize=200`
  - `window,/free ; get next free window`

wset

- Every time you use the `window` command, it will reset the window to be blank and will move it around
- If you have multiple windows open, you can use `wset,window_number` to activate them
  - Dewey never used `wset` according to the text; I've used it often

Multiple Plots

- You can make multiple plots in the same window in a grid
- Uses the `!p.multi` system variable
  - `!p` is the generic “plot-keyword system variable”. It allows you to change the defaults for plotting

System Variables

- Anything preceded with an exclamation point `!` is a system variable
- We've seen `!pi` and `!p`; there will be more

!p.multi

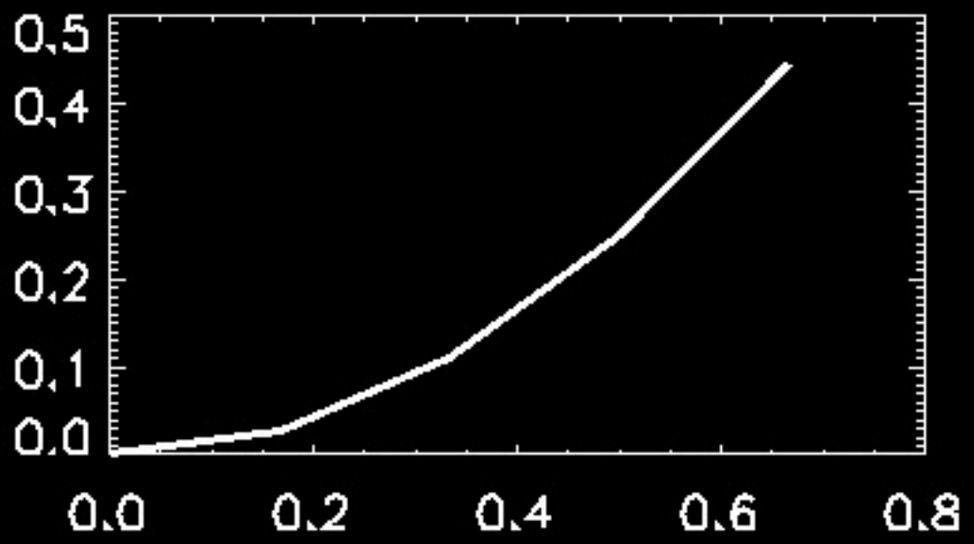
- 5-element integer array:
- [a,b,c,d,e]
  - a: How many free plots are left on the page?
  - b: Number of columns
  - c: Number of rows
  - d,e: ignore for now

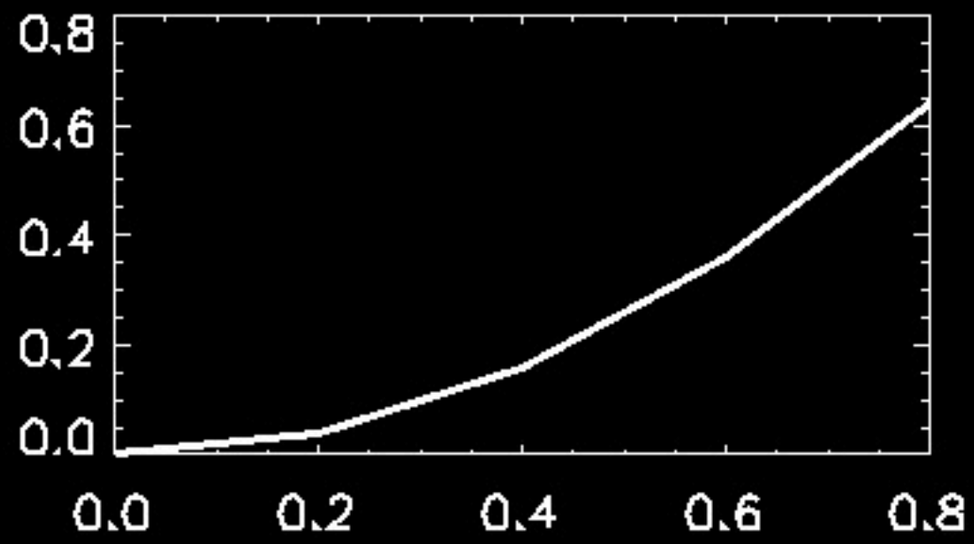
!p.multi

- Each call to `plot` after `!p.multi` is set will appear in a different part of the plot window
- If you want to go back to 1-plot-per-window, reset to the default:
  - `!p.multi = [0,1,1,0,0]`

The `plot` command

- We've already seen it, but haven't covered all its quirks
- `plot,x,y` - x,y are two arrays
  - By default, plot tries to pick nearby integers for the plot ticks
  - `plot` only draws straight lines - curvature is approximated with lots of little lines


0.5
0.4
0.3
0.2
0.1
0.0
0.0
0.2
0.4
0.6
0.8


0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

2.0
1.5
1.0
0.5
0.0
0.0
0.5
1.0
1.5

Defaults

- Changing defaults is an effective shortcut to save you some typing
- but it’s dangerous because you may end up in a different environment (the defaults may be different than you’re accustomed to)

`oplot`

- If you want to plot multiple lines on a single plot, you use `oplot`
- But, what happens if you plot something out of bounds?
  - It won't show up at all.

`oplot`

- If you want to be sure everything fits on your plot, plot the thing with the biggest X and Y range first
- If one line has a greater X extent but the other has a greater Y extend, you'll need to use the `xrange` and `yrange` keywords

Titles and Axis Labels

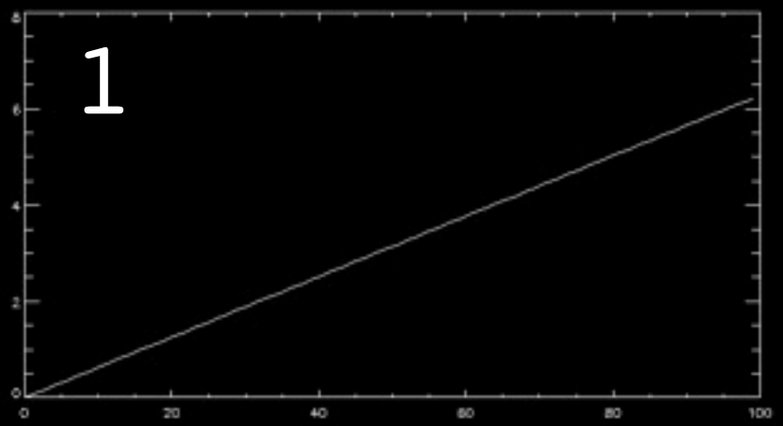
- You can (and should) add labels to your axes
  - plot, x, y,
    xtitle="wavelength (angstroms)",
    ytitle="flux (W/m^2)",
    title="A Spectrum!"

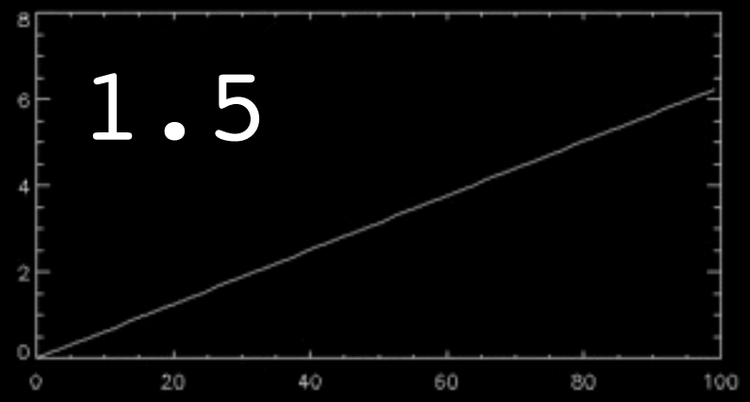
Titles and Axis Labels

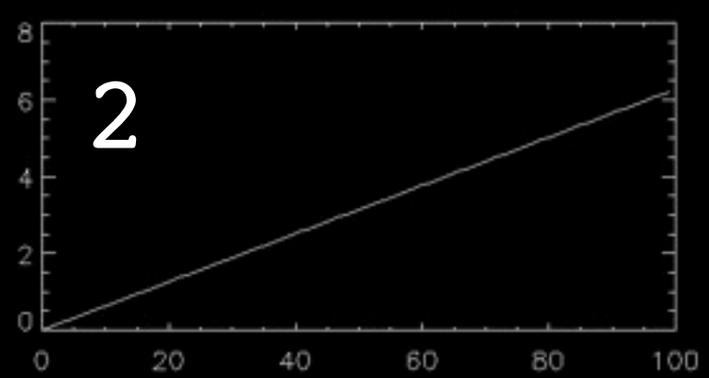
- You can't use the `title` keywords with oplot
- Also, the color keyword (next chapter) will affect the axis colors if passed to plot
- You can use `plot,x,y,/nodata` to set up the plot as if you were going to plot your data, but not actually make a line

{X|Y|Z}charsize

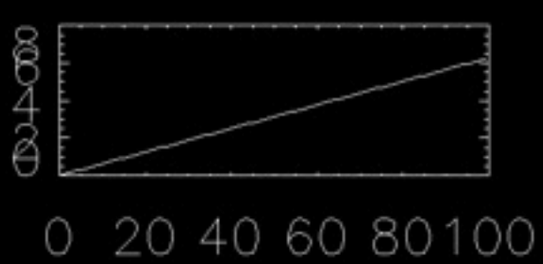
- Can change the character size...



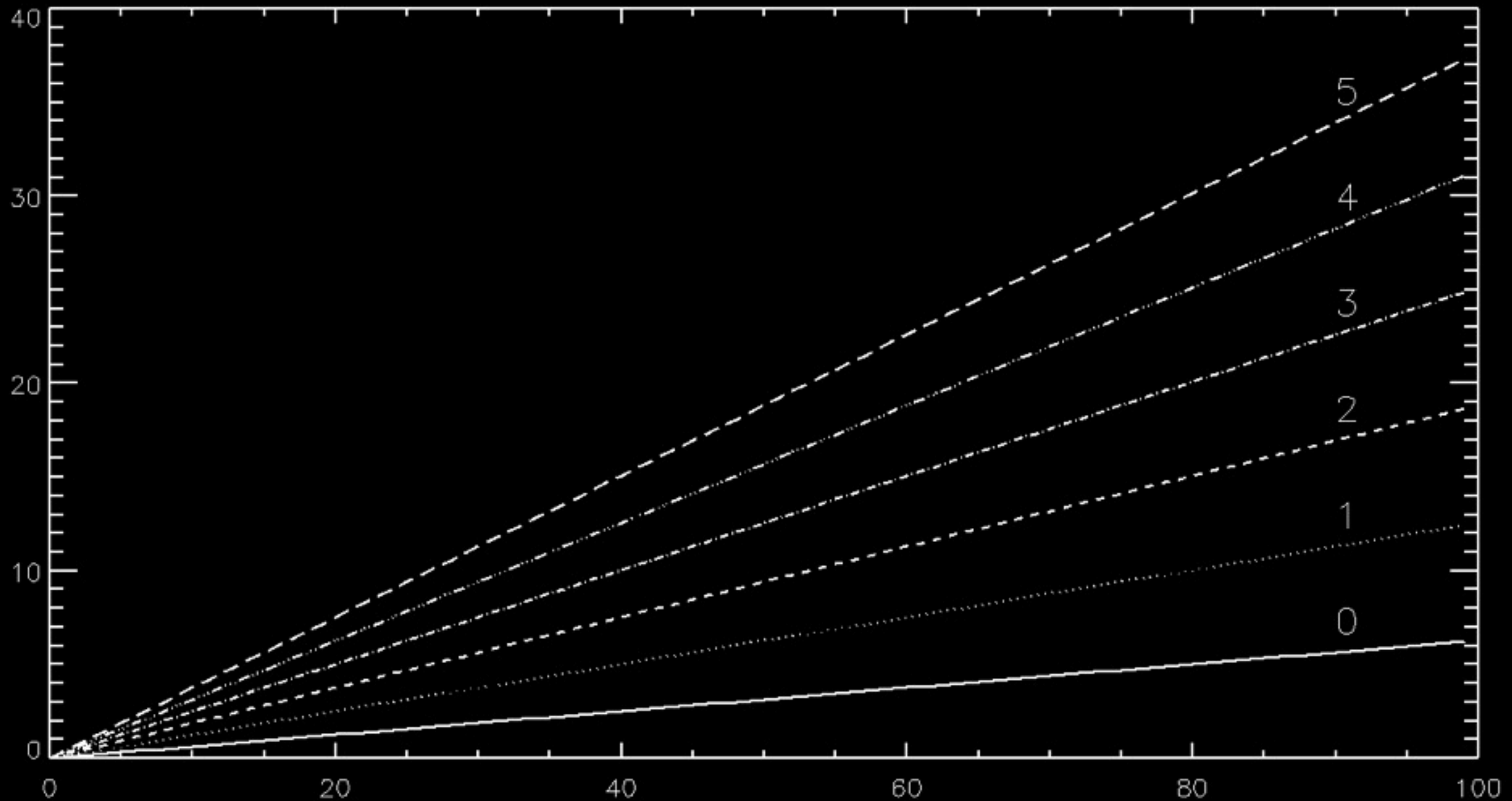




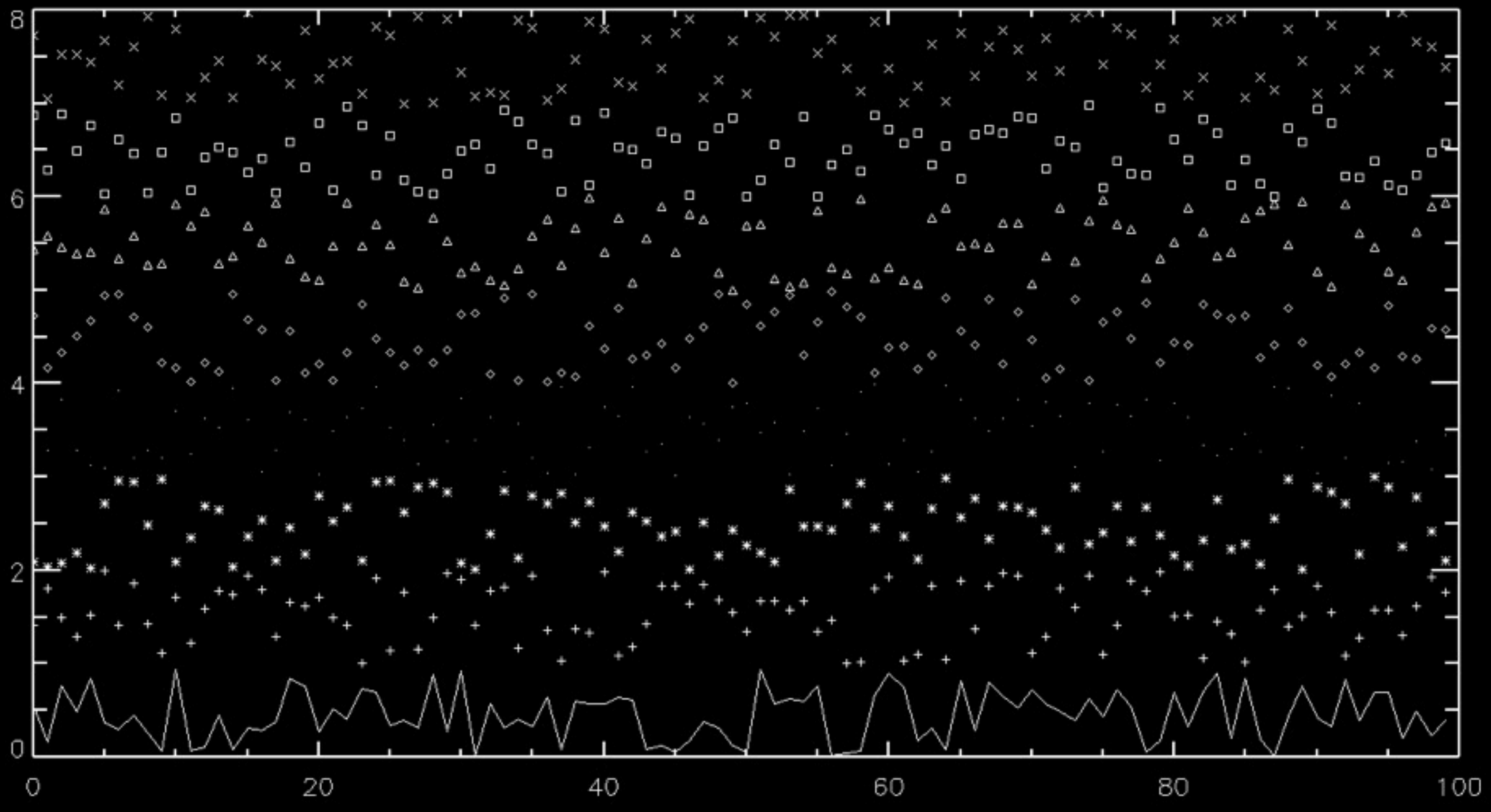
4



linestyle

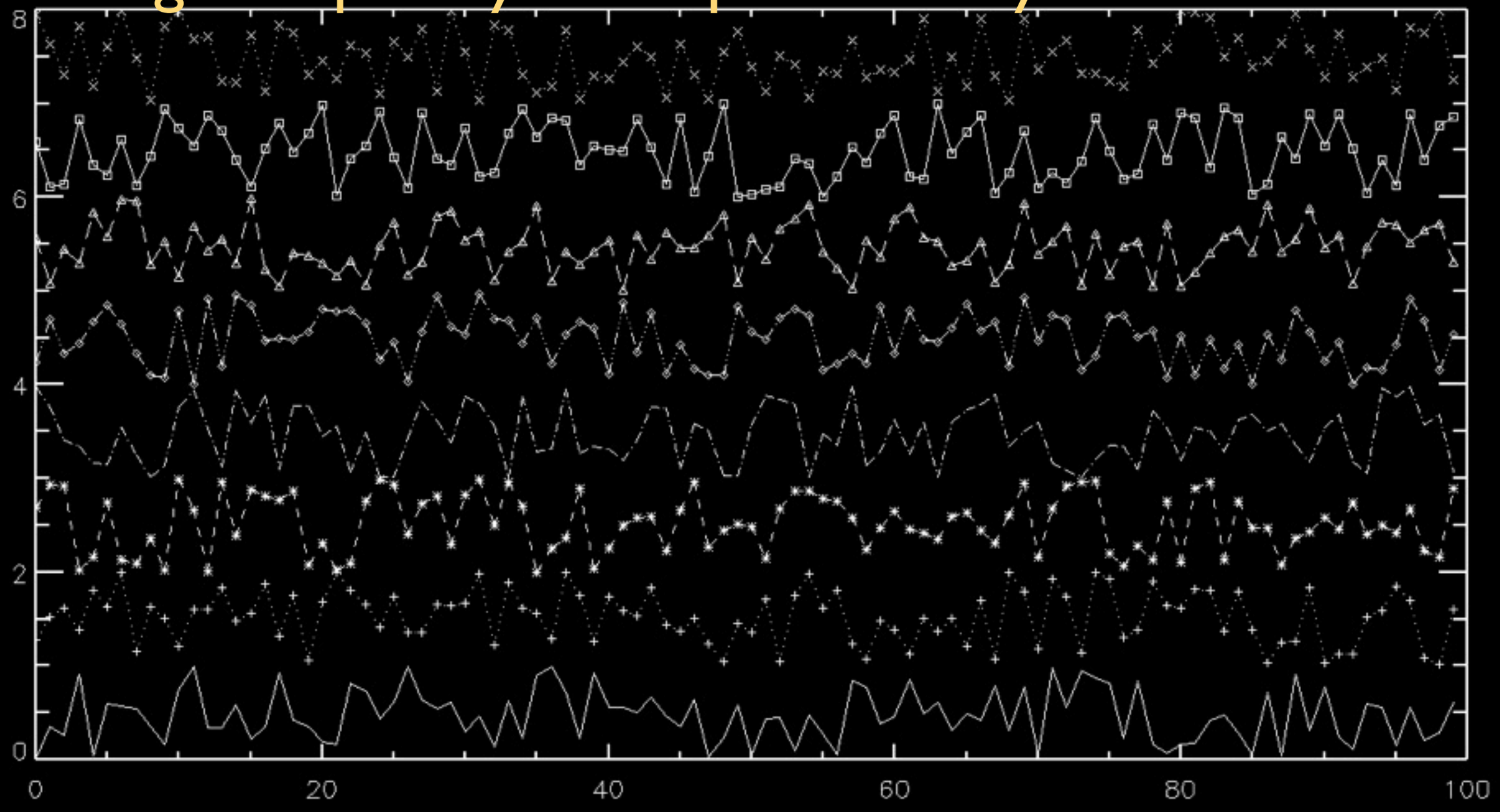


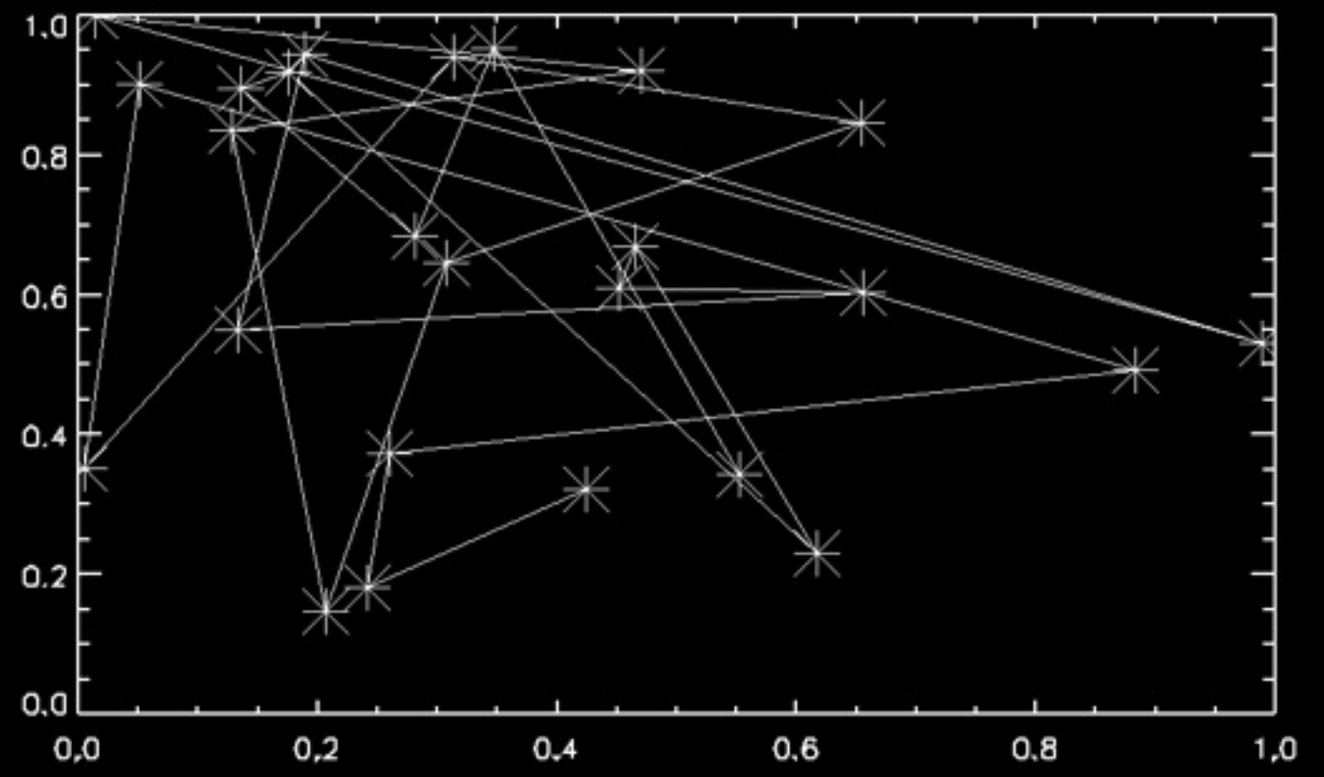
Plot Symbols (`psym`)



Plot Symbols (psym)

Negative plot symbols plot both symbols & line





Which command might have been used to make this plot?

A
```
IDL> plot,x,y,psym=-2,thick=0.5,xthick=2,ythick=2,symsize=5
```
B
```
IDL> plot,x,y,psym=2,thick=0.5,xthick=2,ythick=2,symsize=5
```
C
```
IDL> plot,x,y,thick=0.5,xthick=2,ythick=2,symsize=5
```
D
```
IDL> plot,x,y
```
E

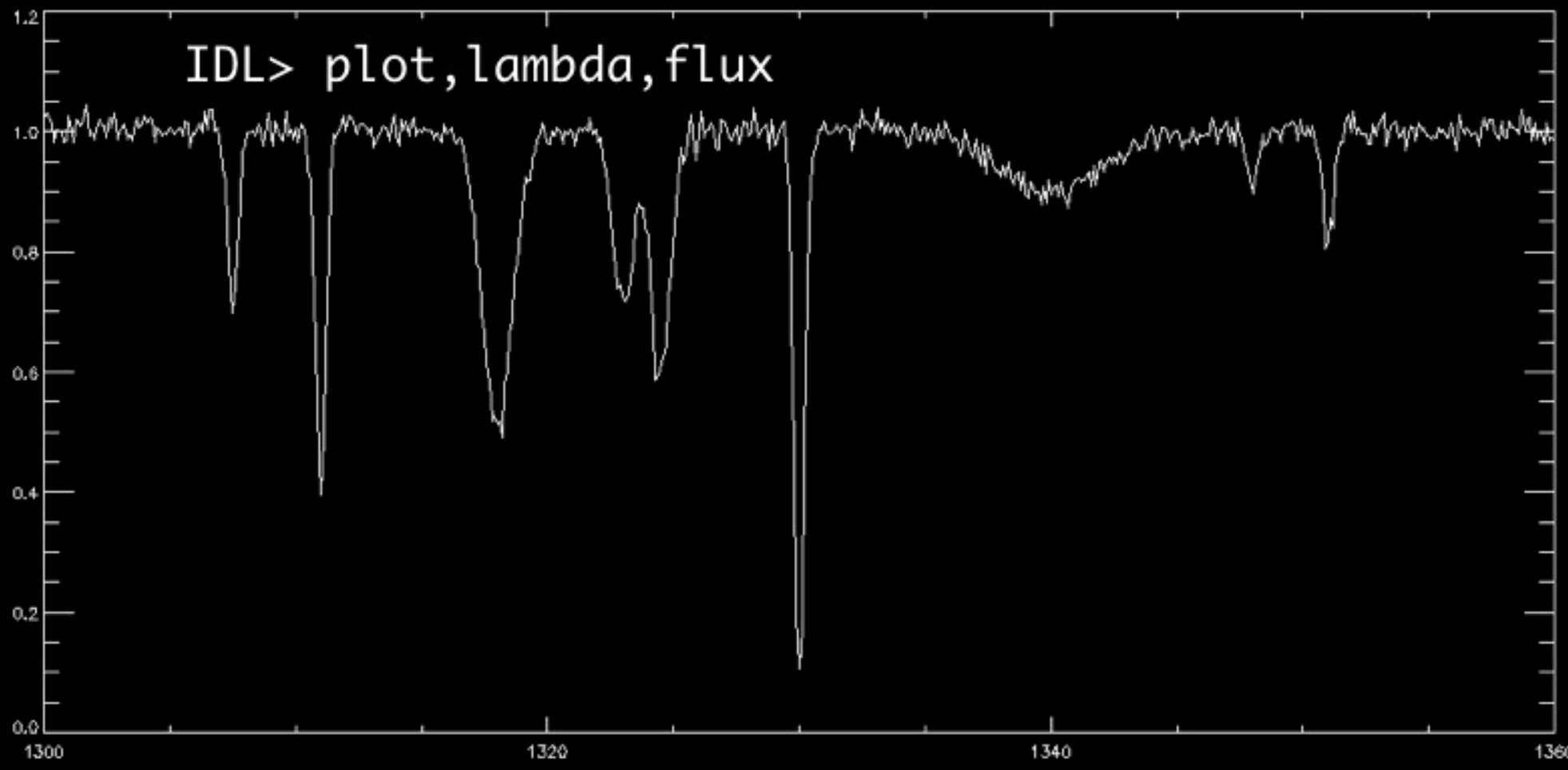
None of the above

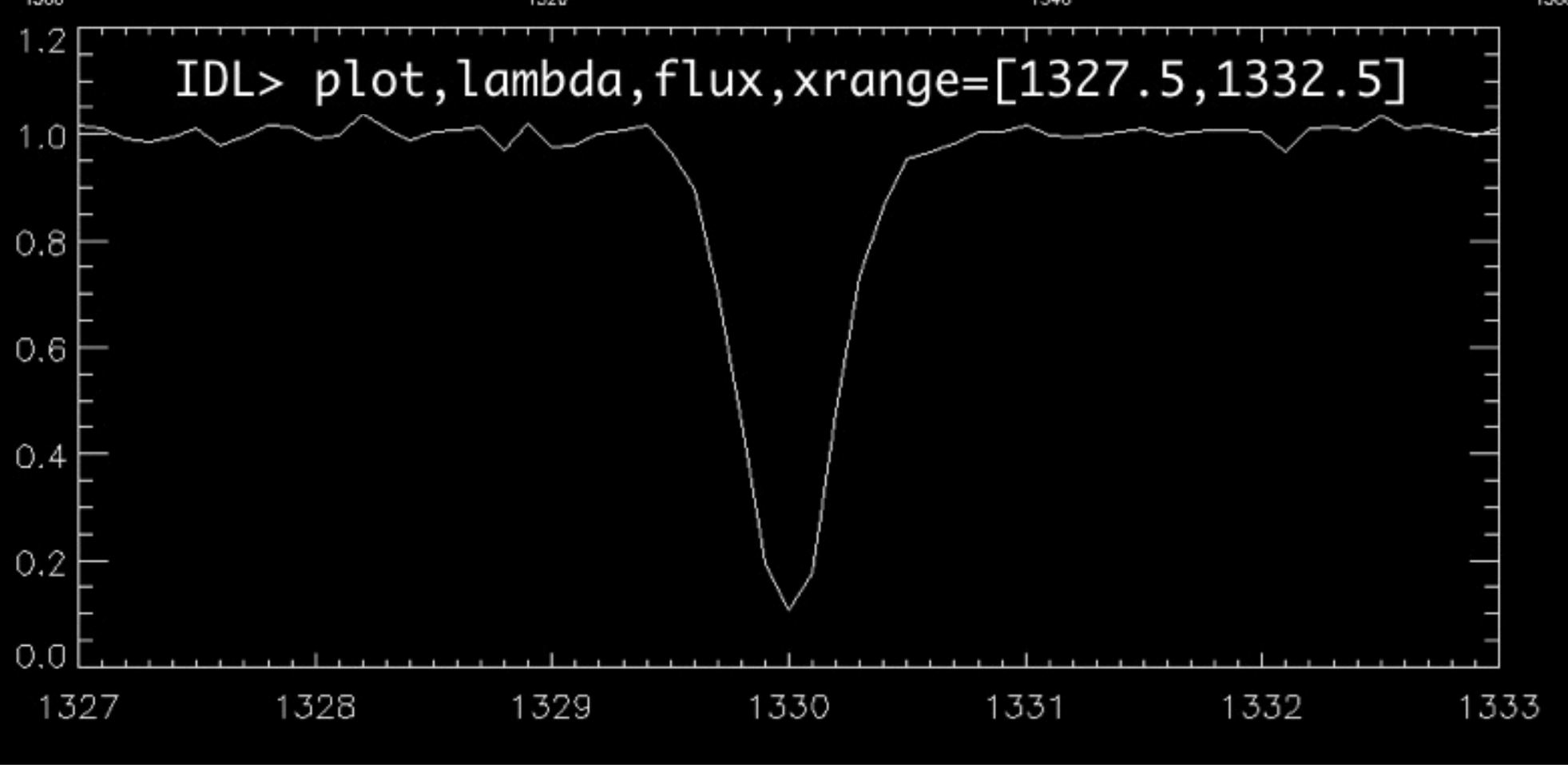
plot,/nodata

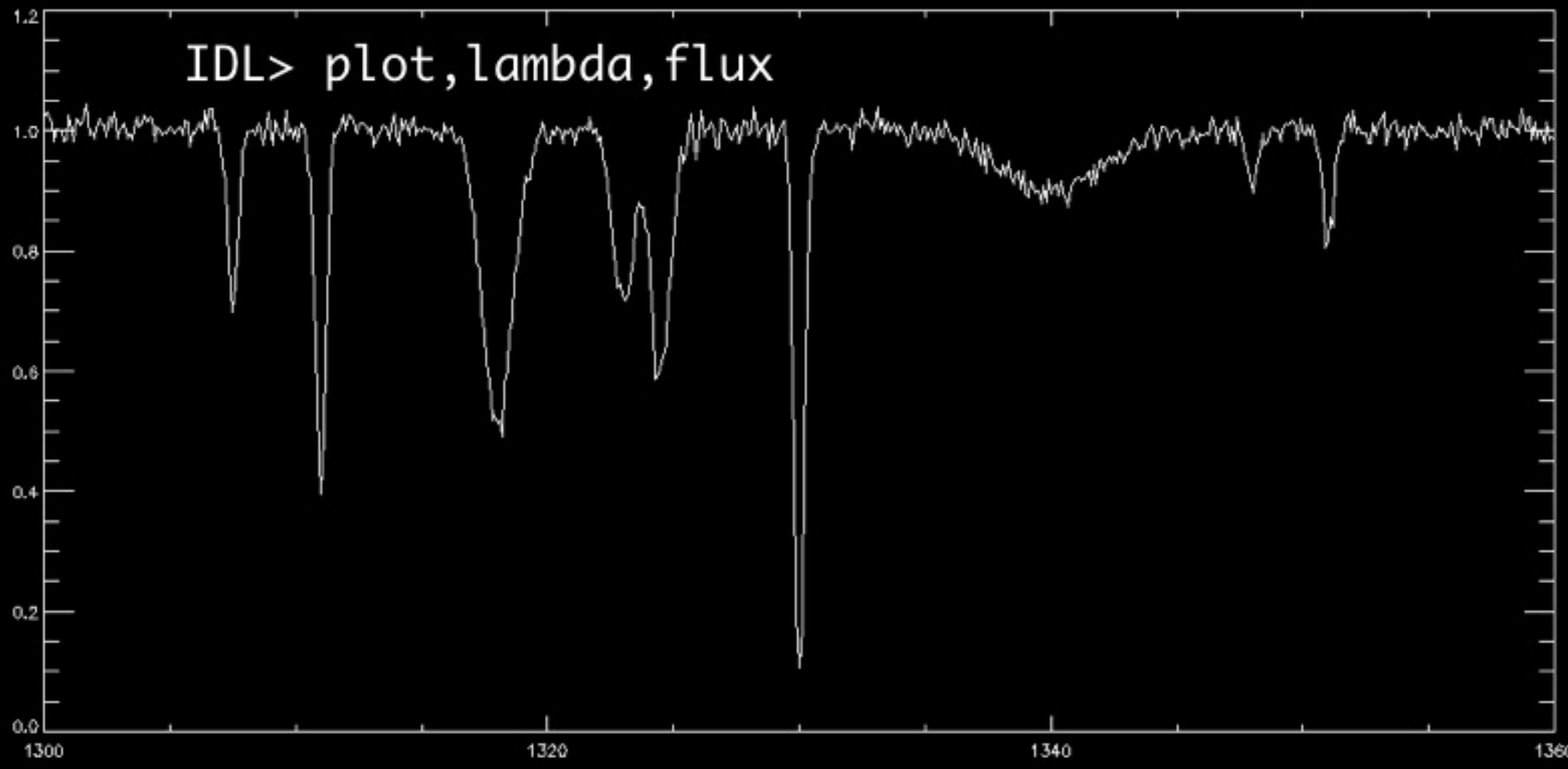
- You can set up a plot window to have the correct range beforehand
- `plot,[xmin,xmax],[ymin,ymax],/nodata` will create a window with boundaries specified
- Then, you plot the data using `oplot`
  - Good for colored lines on white axes!

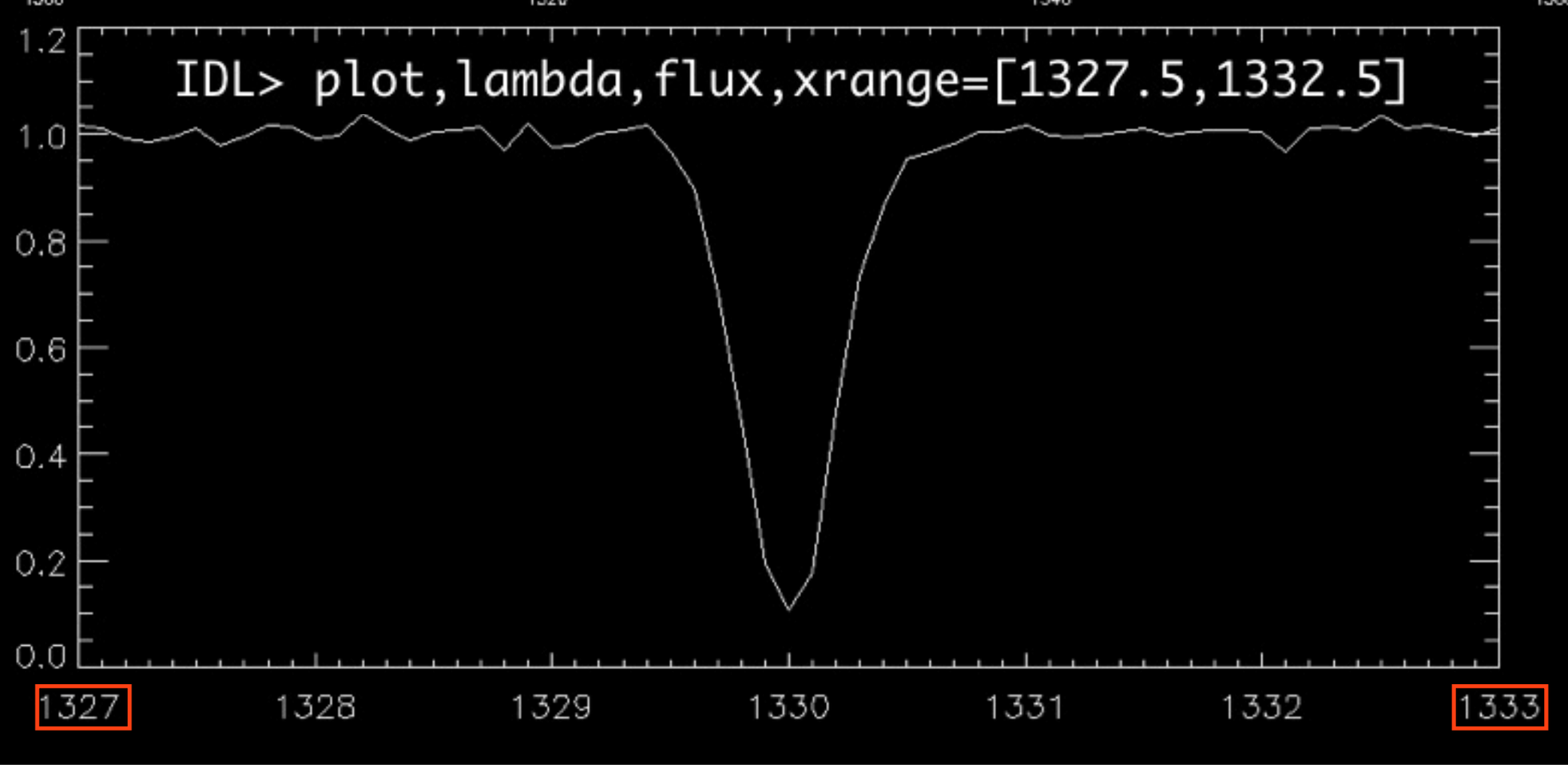
`range, style`

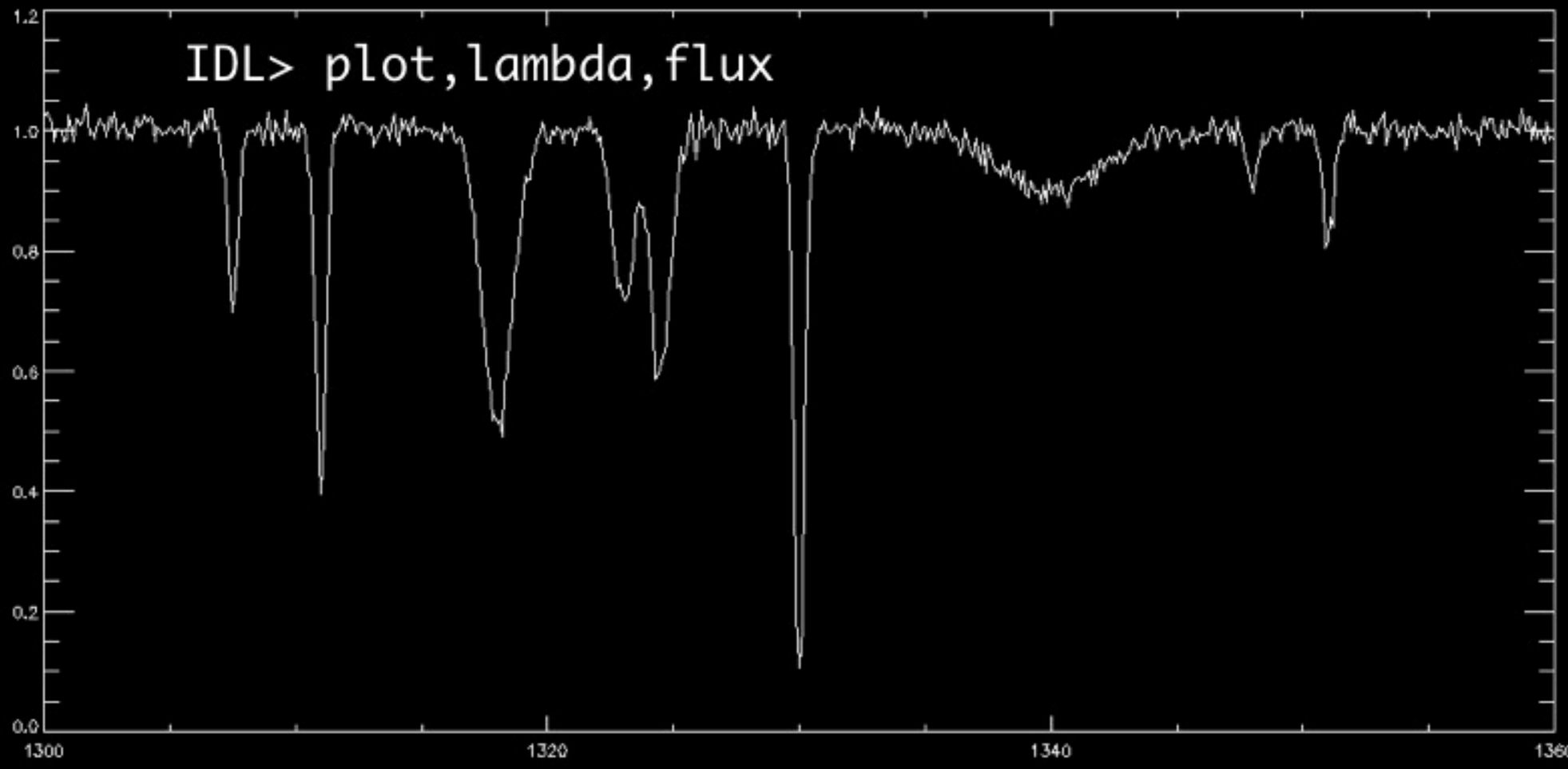
- You can directly control the range plotted (both x and y)
- IDL doesn't respect these exactly unless you specify the `style` keyword

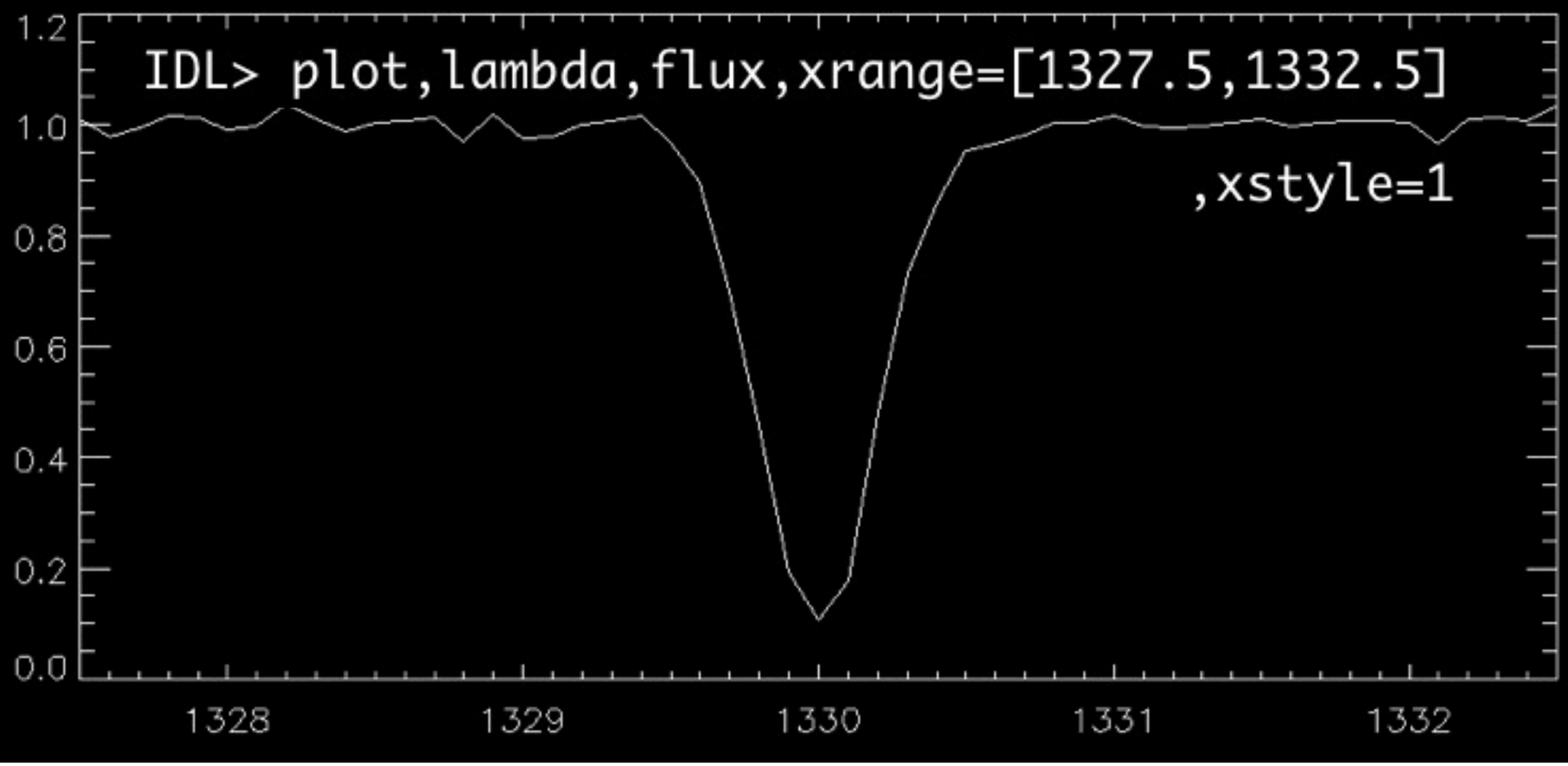

IDL> plot,lambda,flux
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1300
1320
1340


IDL> plot,lambda,flux,xrange=[1327.5,1332.5]
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1327
1328
1329
1330
1331
1332
1333


IDL> plot,lambda,flux
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1300
1320
1340


IDL> plot,lambda,flux,xrange=[1327.5,1332.5]
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1327
1328
1329
1330
1331
1332
1333


IDL> plot,lambda,flux
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1300
1320
1340

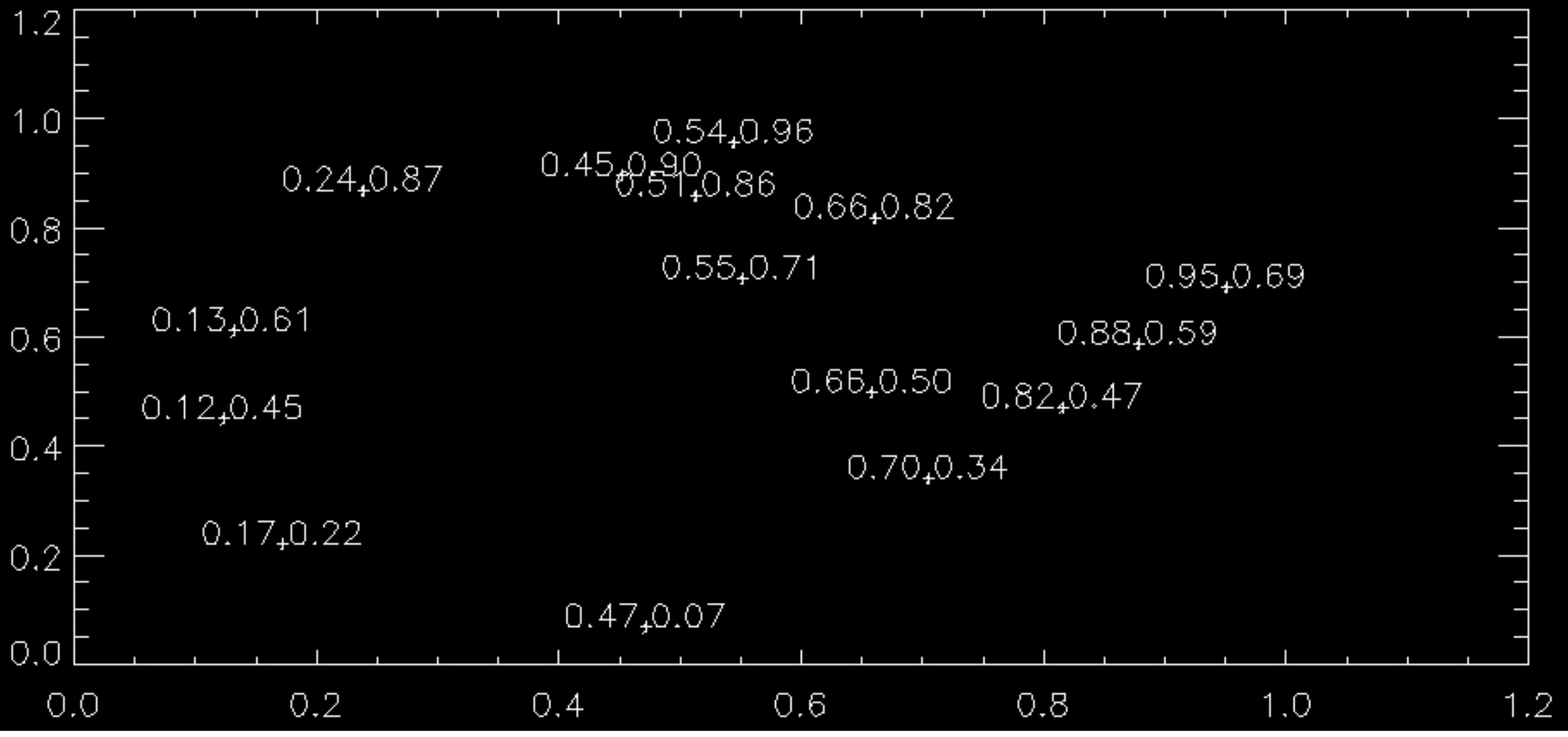

IDL> plot,lambda,flux,xrange=[1327.5,1332.5]
,xstyle=1
1.2
1.0
0.8
0.6
0.4
0.2
0.0
1328
1329
1330
1331
1332

log plots

- Pass the `/xlog` and `/ylog` keyword flags

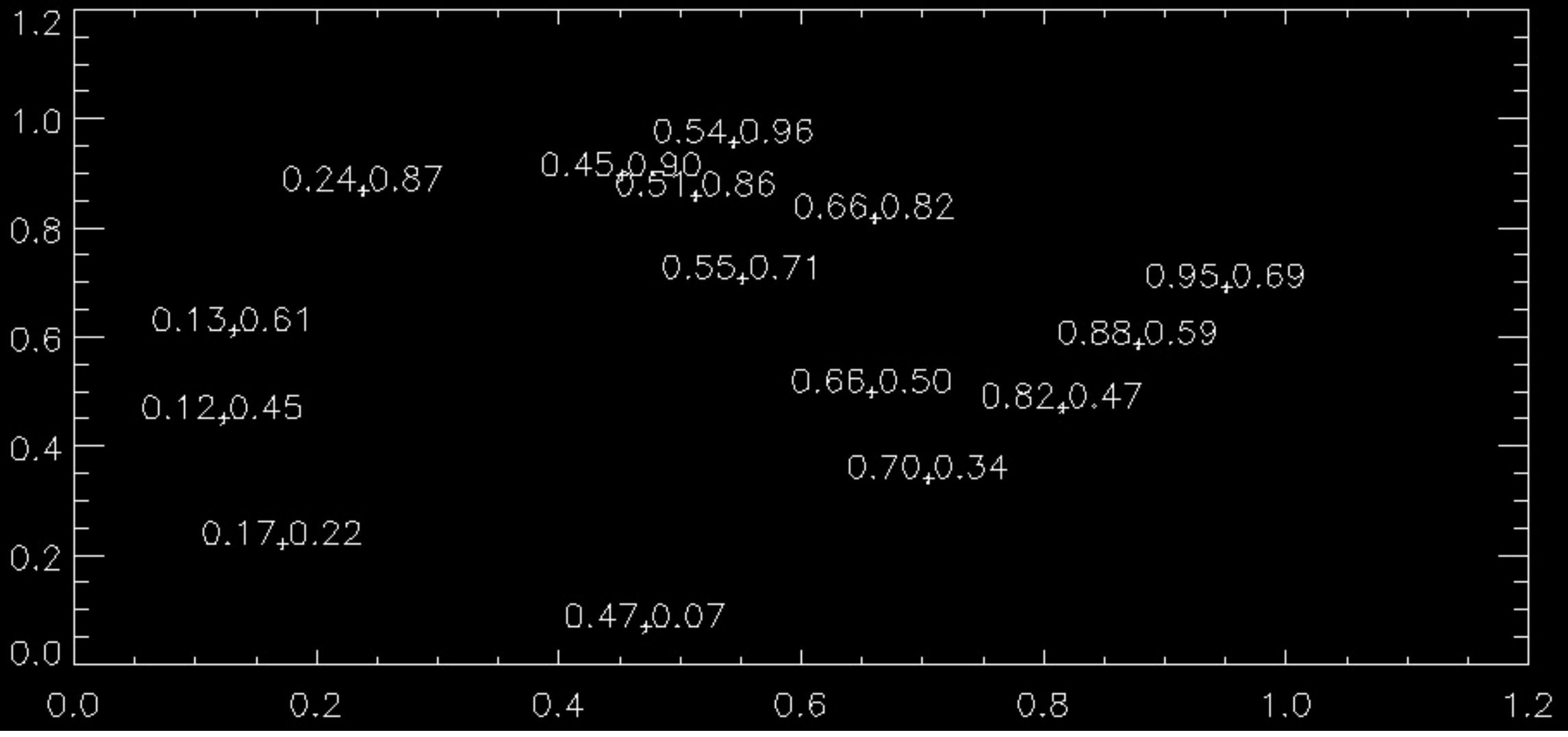
Text in the plot window: `xyouts`

- `xyouts,x,y,string`

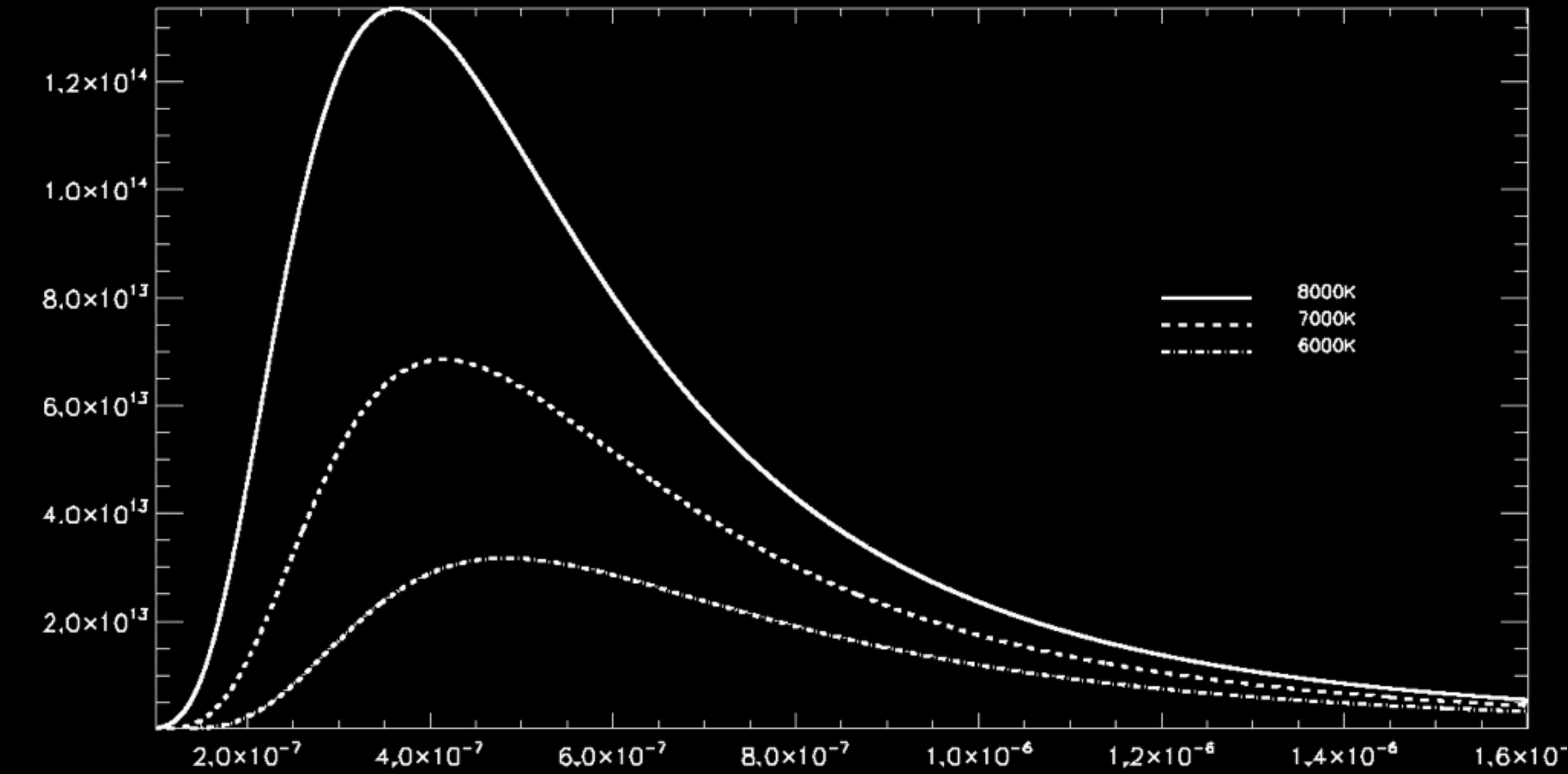


xyouts example: snapshot of code the way I edit it in vim

```
x = randomu(seed,15)
y = randomu(seed2,15)
plot,x*1.1,y*1.1,/nodata,charsize=2
for ii=0,14 do xyouts,x[ii],y[ii],string(x[ii],y[ii],format="(F0.2,',',F0.2)"),charsize=2,alignment=0.5
oplot,x,y,psym=1
```



Legends with `xyouts`



Are you...

A) Awake

B) Asleep

C) Zombie

D) Coma

E) None of the above

`erase` command

- Clears the plot window, but leaves the axis scaling intact (i.e., if you plot a line, it will show up in exactly the same place, but without axes)
- might be nice for drawing non-scientific “plots” like snakes

Interactive Use

- What if you want the user (maybe you, maybe someone else) to be able to interact using the mouse instead of the command prompt?
- `cursor,x,y` gets input from the graphics window, storing it in variables `x,y`

!mouse

- System variable `!mouse` will store information about the last click:

```
IDL> help,!mouse
** Structure !MOUSE, 4 tags, length=16, data length=16:
   X               LONG                 0
   Y               LONG                 0
   BUTTON          LONG                 0
   TIME            LONG                 0
```

cursor use

- Great for zooming in on things
- could be used for drawing (though it’s a pretty limited drawing system)

Two-Dimensional Arrays

- AKA images
- Not as easy to generate

```
IDL> print,indgen(3,3)
       0       1       2
       3       4       5
       6       7       8
```

Generating 2D Arrays

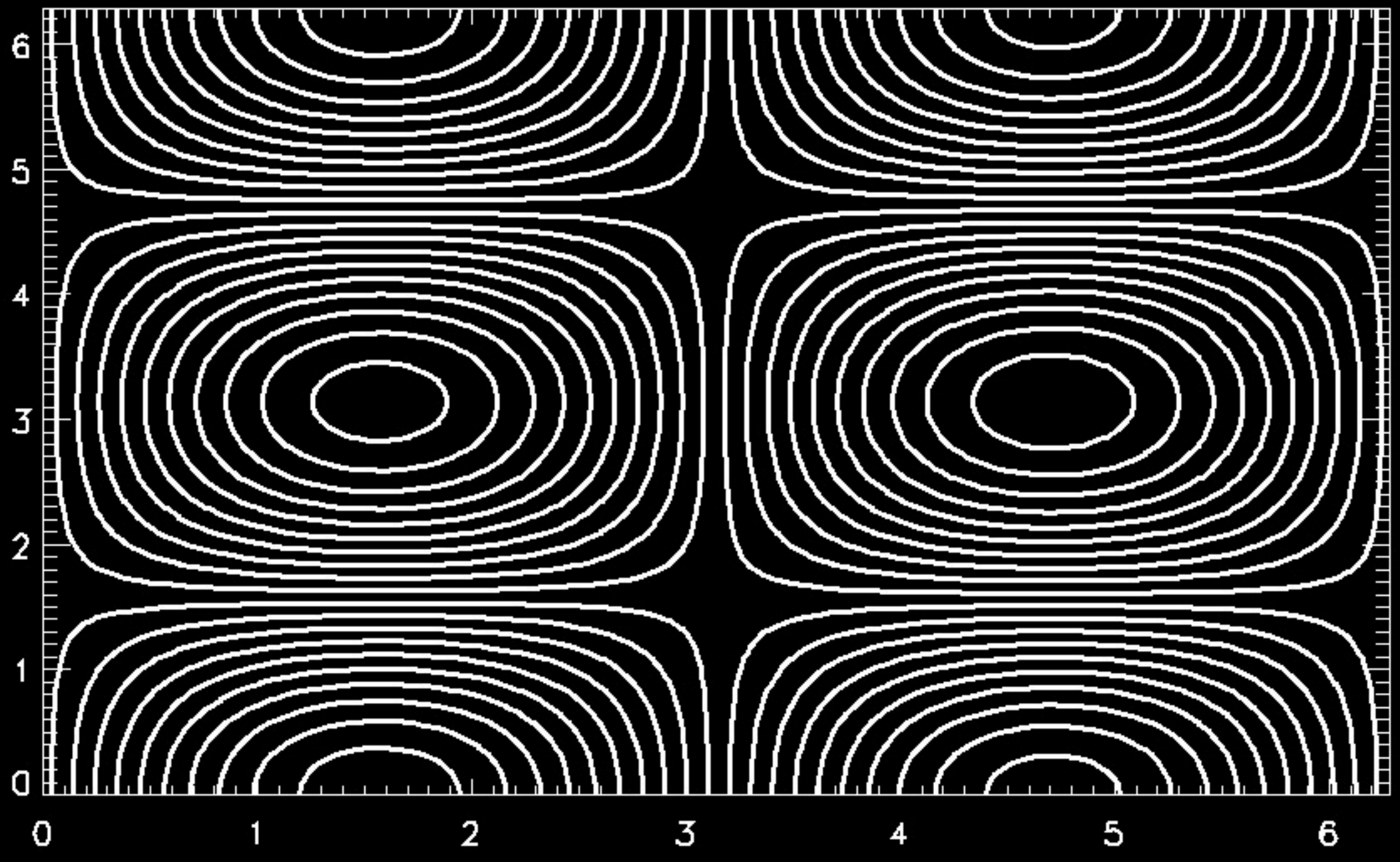
```
IDL> x = indgen(3)
IDL> xx = rebin(x,[3,3])
IDL> y = indgen(1,3)
IDL> yy = rebin(y,[3,3])
IDL> print,xx,yy
       0       1       2       0       0       0
       0       1       2       1       1       1
       0       1       2       2       2       2
```

- `rebin` replicates data along whichever axis is too small

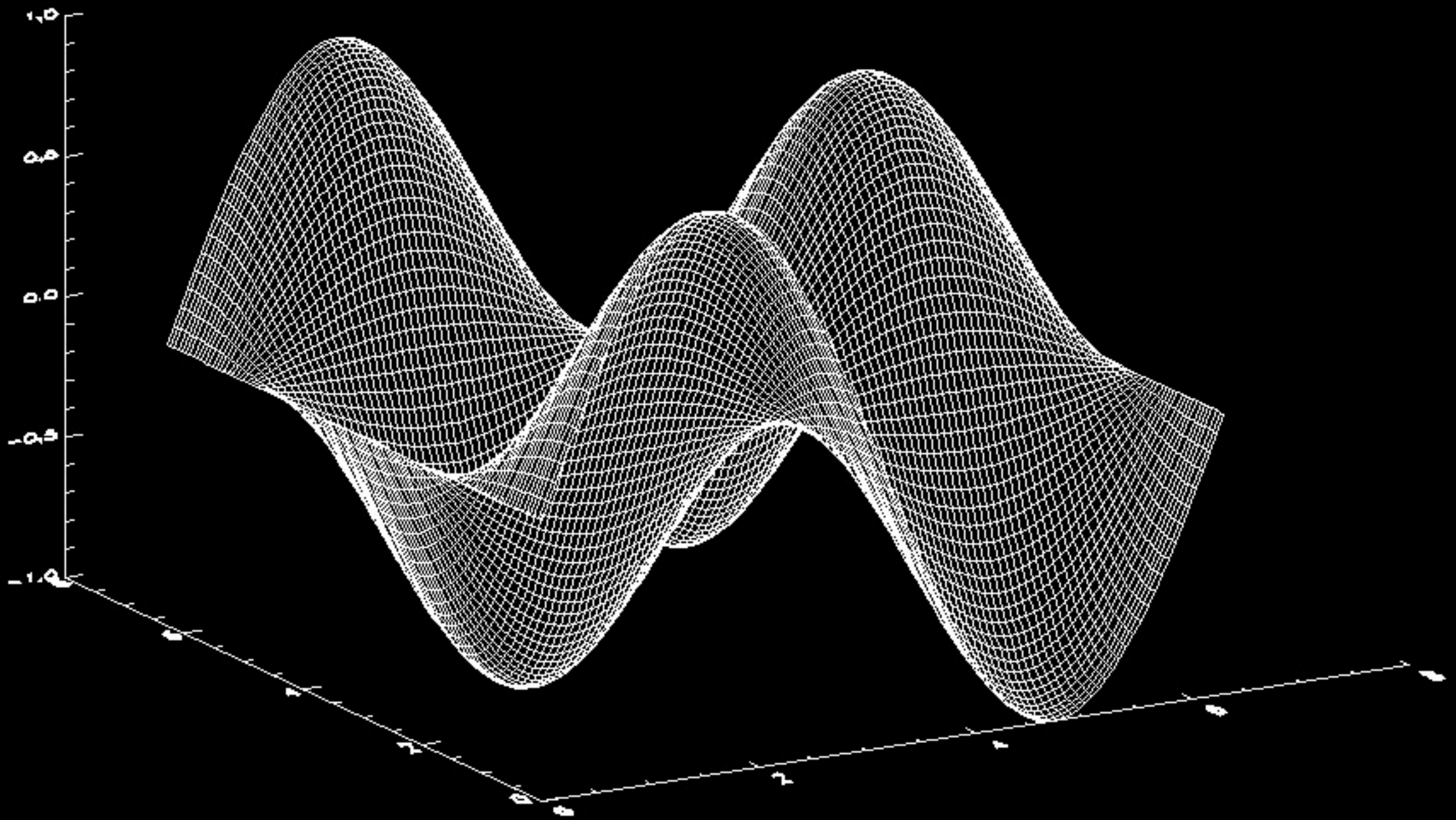
Contour Plots

- Say we generate our 2D xx and yy arrays, then we can make something like: `zz = sin(xx)*cos(yy)`

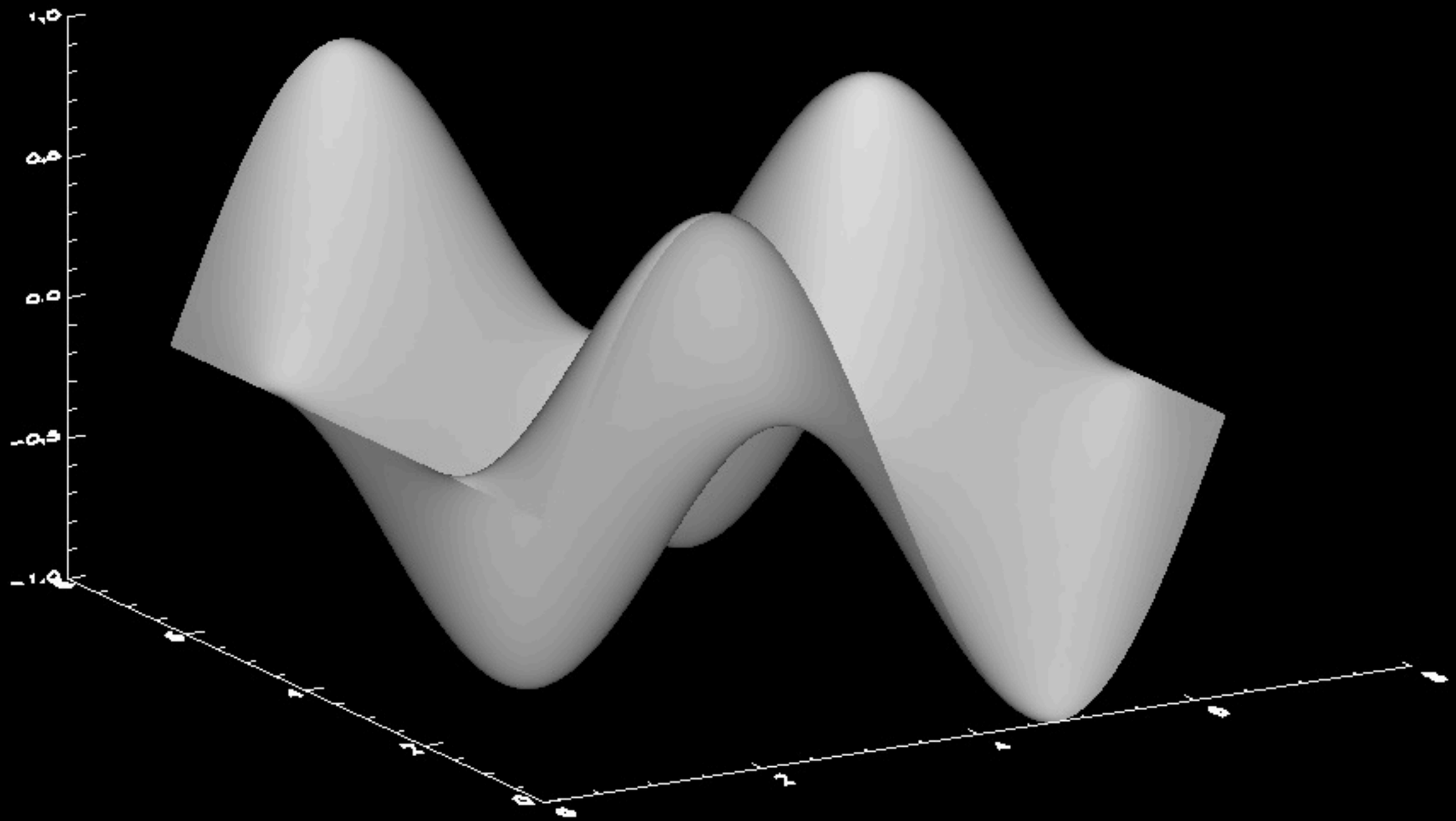
2D displays: `contour`



2D displays: `surface`

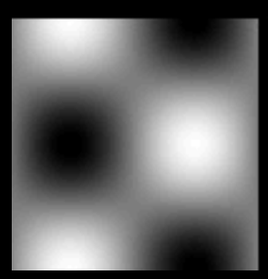


2D displays: `shade_surf`



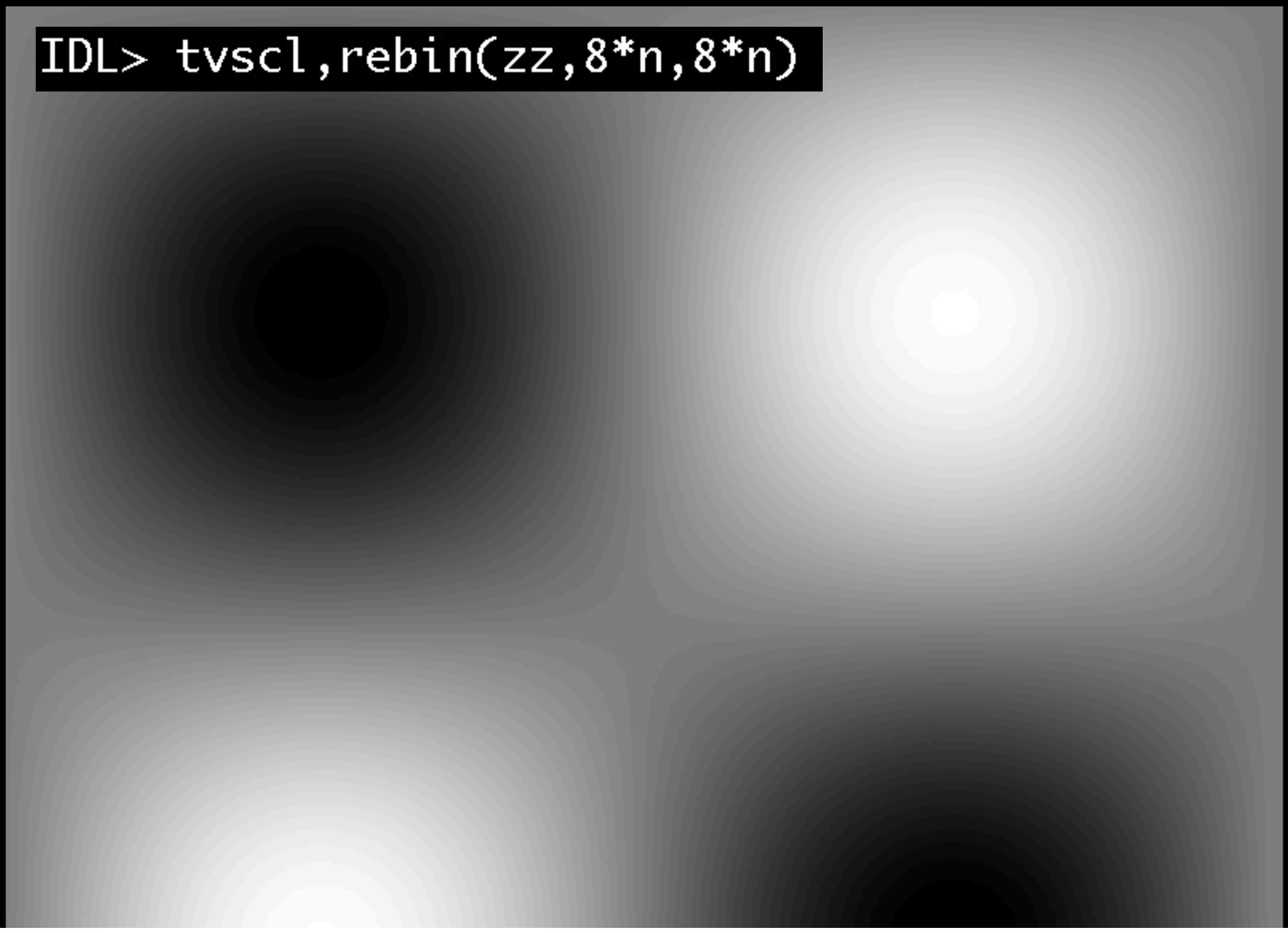
2D displays: `tvscl`

Displays in native resolution... which is not often what you want



2D displays: `tvscl`

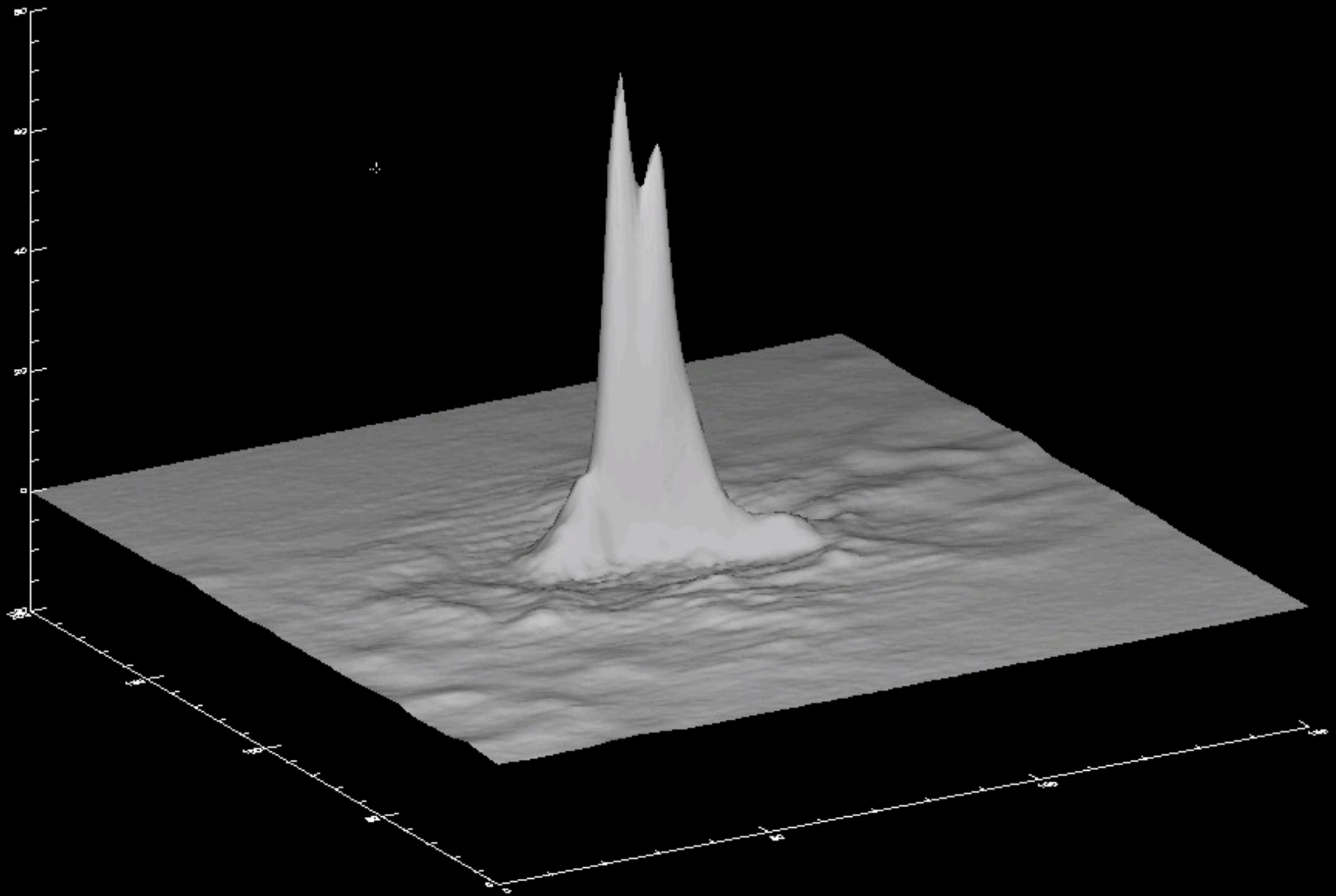
```
IDL> tvscl,rebin(zz,8*n,8*n)
```



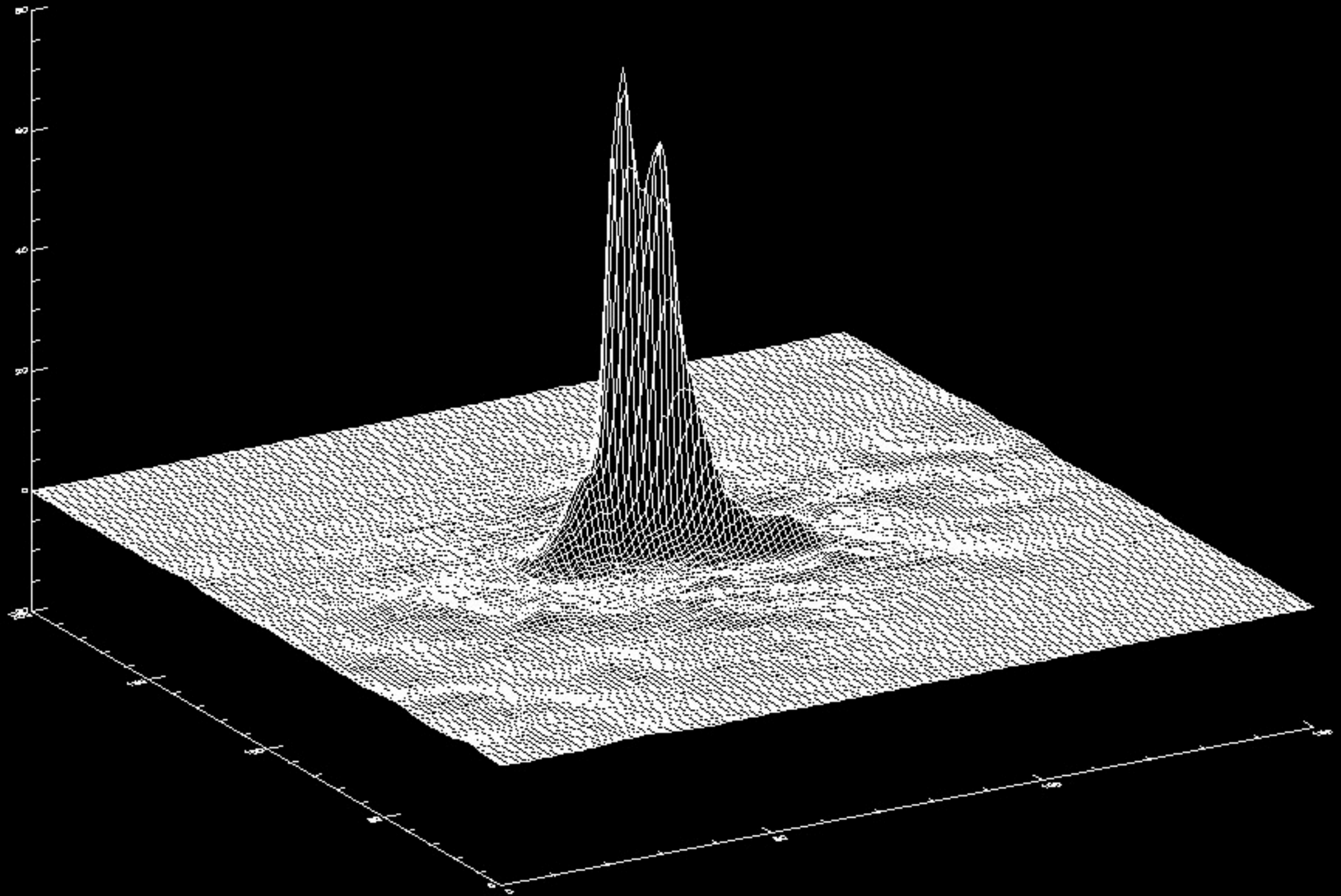
2D+ plots revisited

- `shade_surf`
- `surface`
- `contour`
- `tvscl`
- Next: Examples of Sgr B2 looking like a mountain

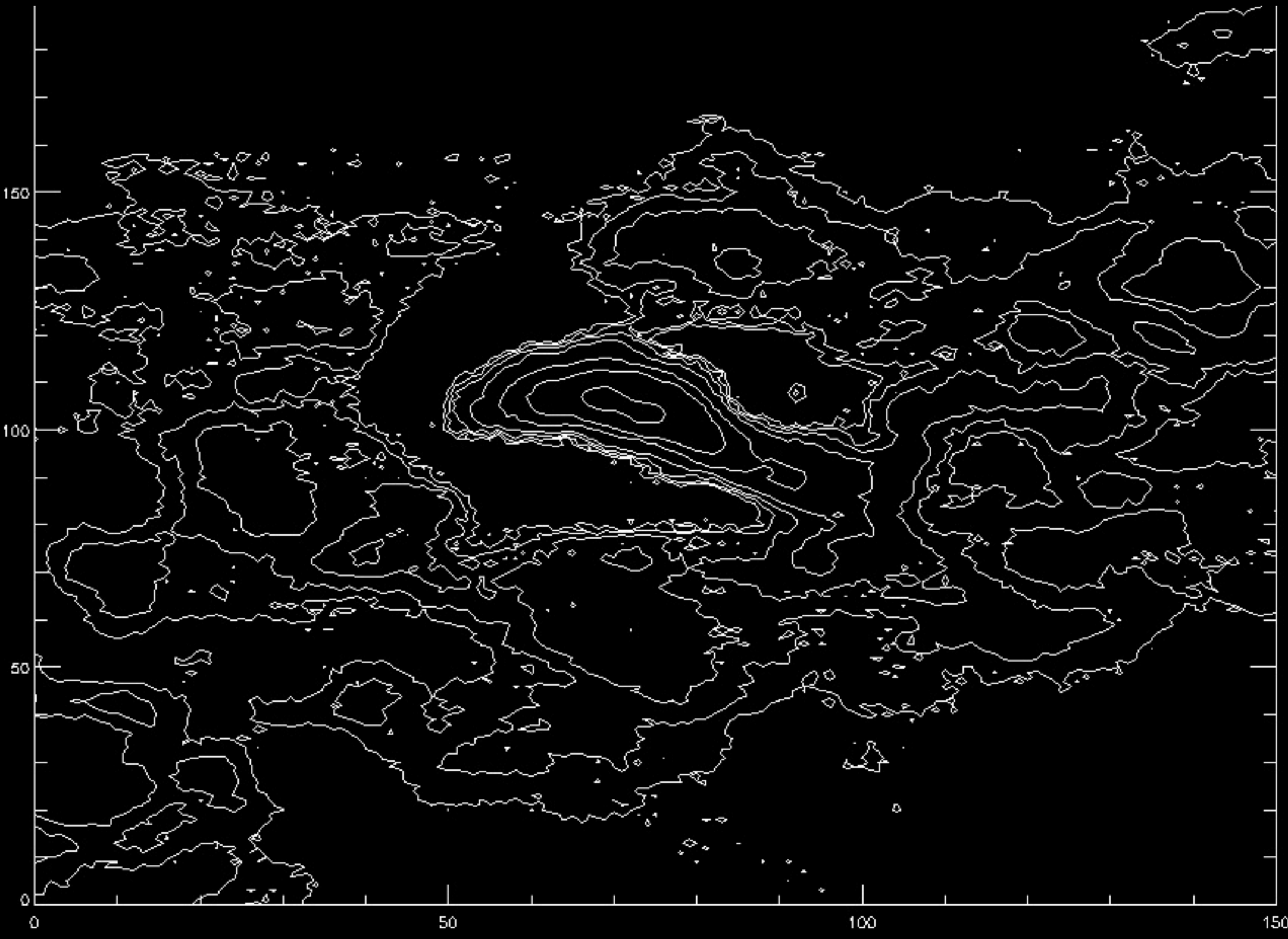
```
shade_surf,(img[2400:2550,700:900])
```



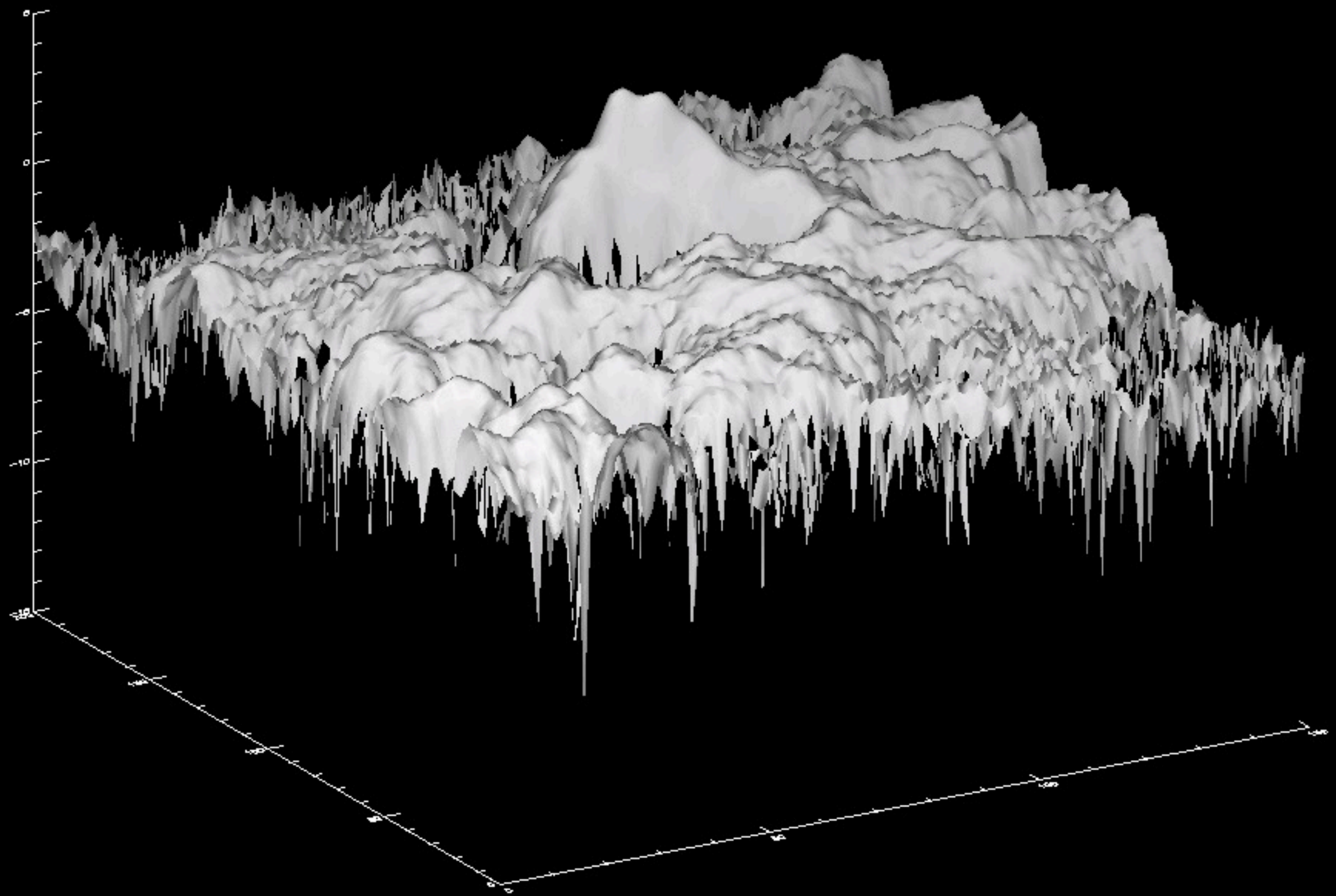
```
surface,img[2400:2550,700:900]
```



```
contour,img[2400:2550,700:900],levels=[0.2,0.5,1,
2,5,10,50]
```



```
shade_surf,alog(img[2400:2550,700:900])
```



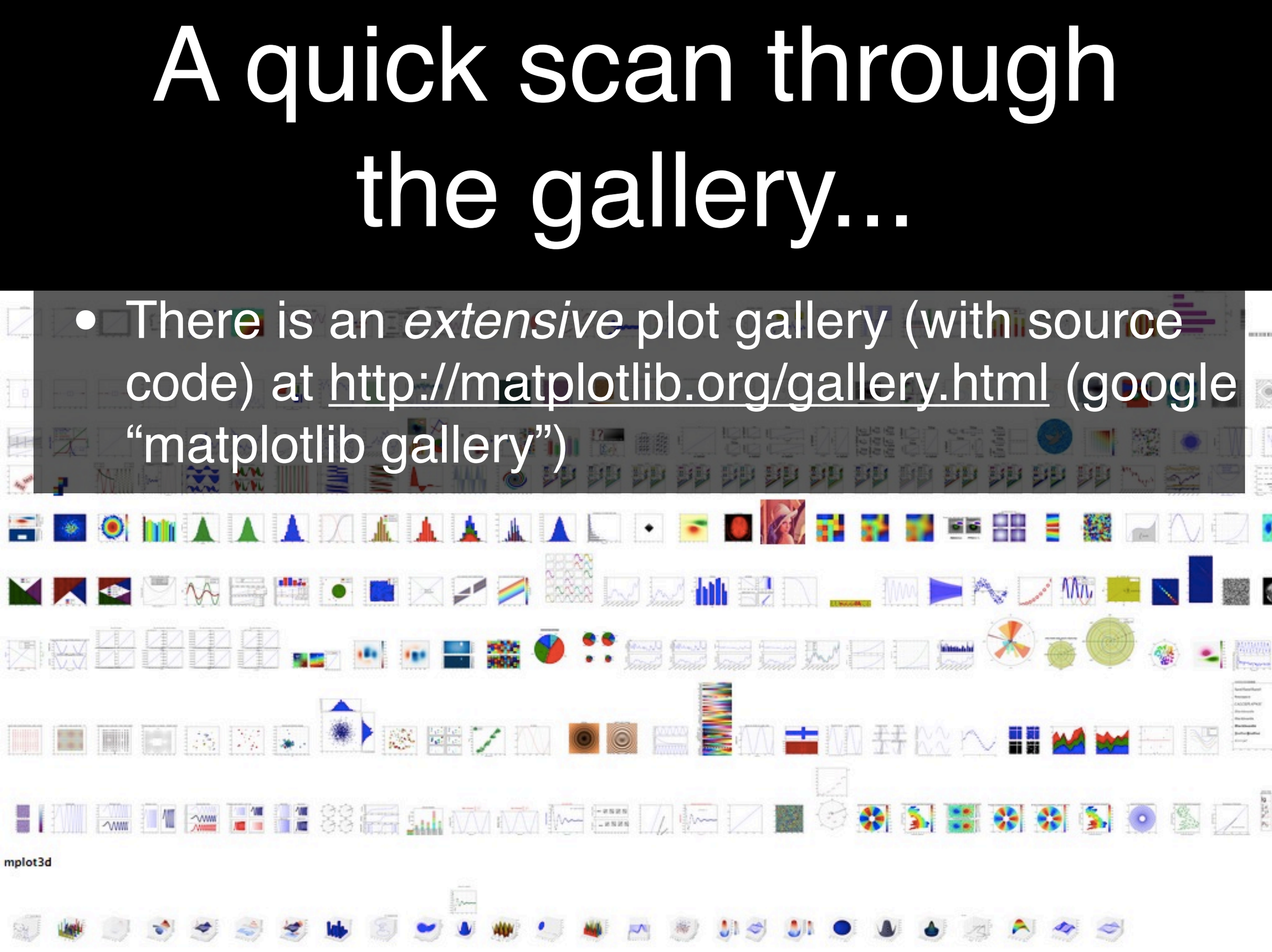
Now for something completely... pythonic

All the same in python

- `figure(1)` makes a new figure with number 1, or “activates” figure 1 if it’s open
- `clf()` clears a figure

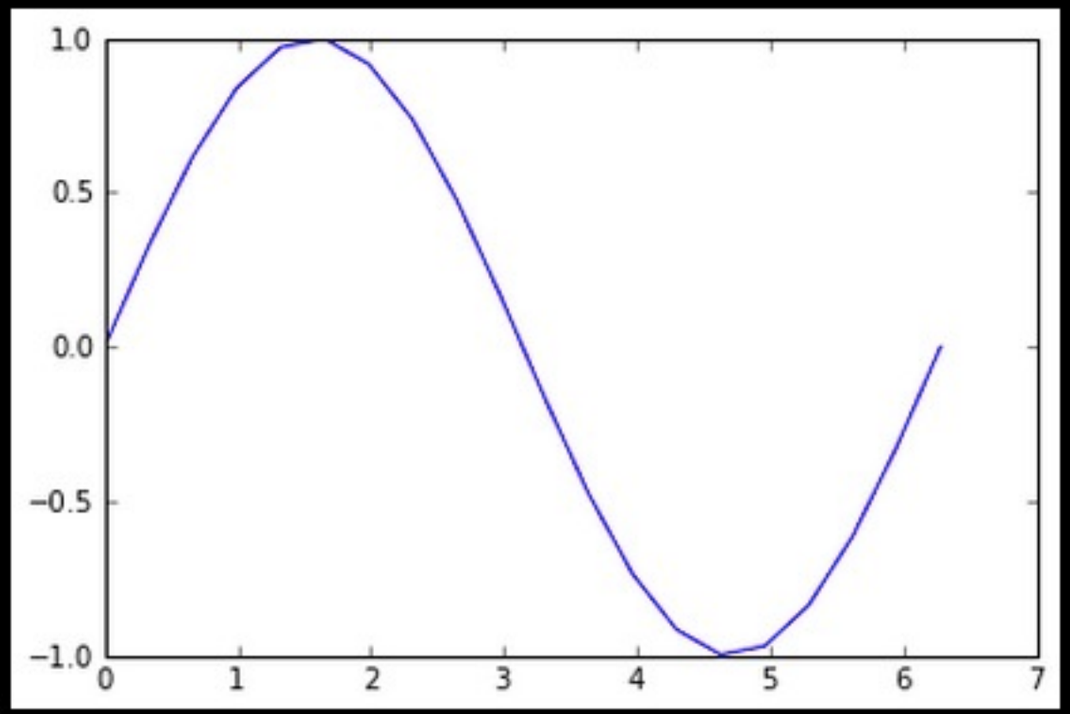
A quick scan through the gallery...

- There is an *extensive* plot gallery (with source code) at http://matplotlib.org/gallery.html (google "matplotlib gallery")



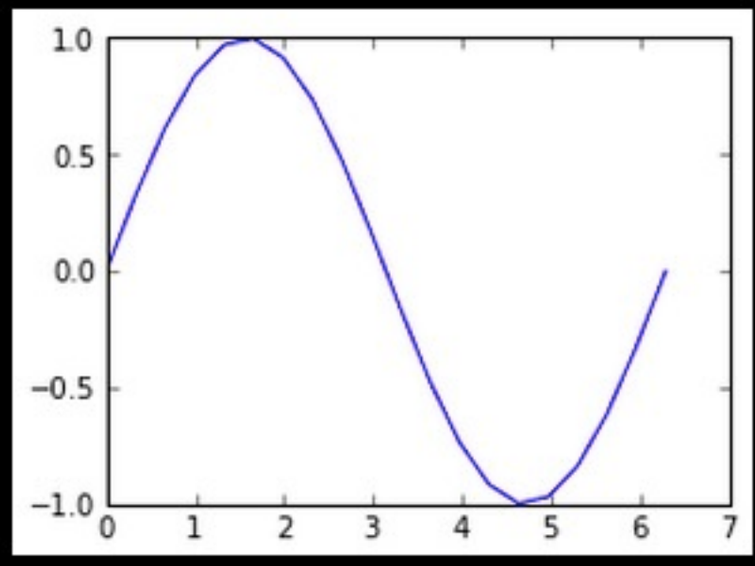
In [1]:
```
# plot something simple
x = linspace(0,2*pi,20)
plot(x,sin(x))
```

Out[1]: [<matplotlib.lines.Line2D at 0x105905650>]



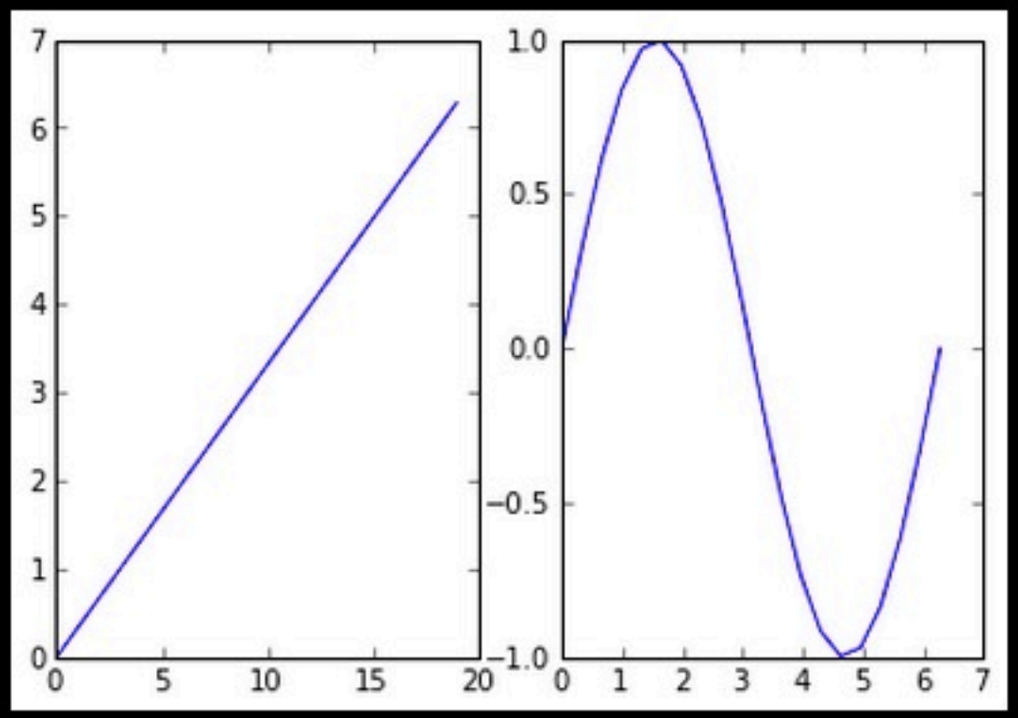
In [2]:
```
# specify figure number, figure size
figure(2,figsize=(4,3))
plot(x,sin(x))
```

Out[2]: [<matplotlib.lines.Line2D at 0x105d42890>]



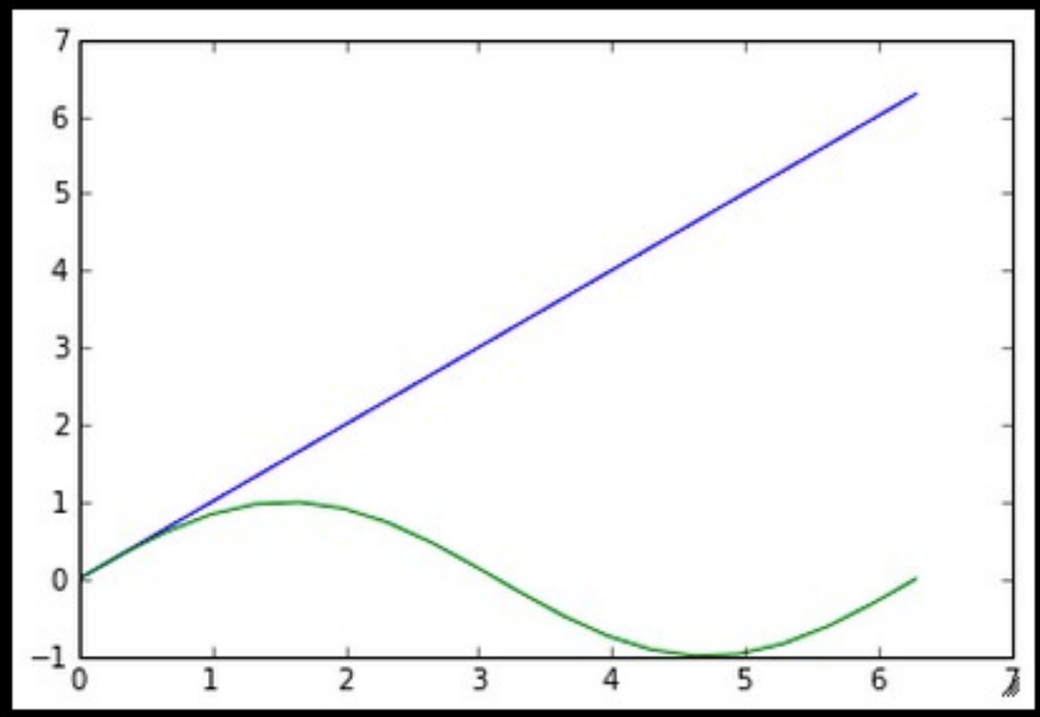
In [4]:
```
# multiple subplots
subplot(121)
plot(x)
subplot(122)
plot(x,sin(x))
```

Out[4]: [<matplotlib.lines.Line2D at 0x105f2d2d0>]



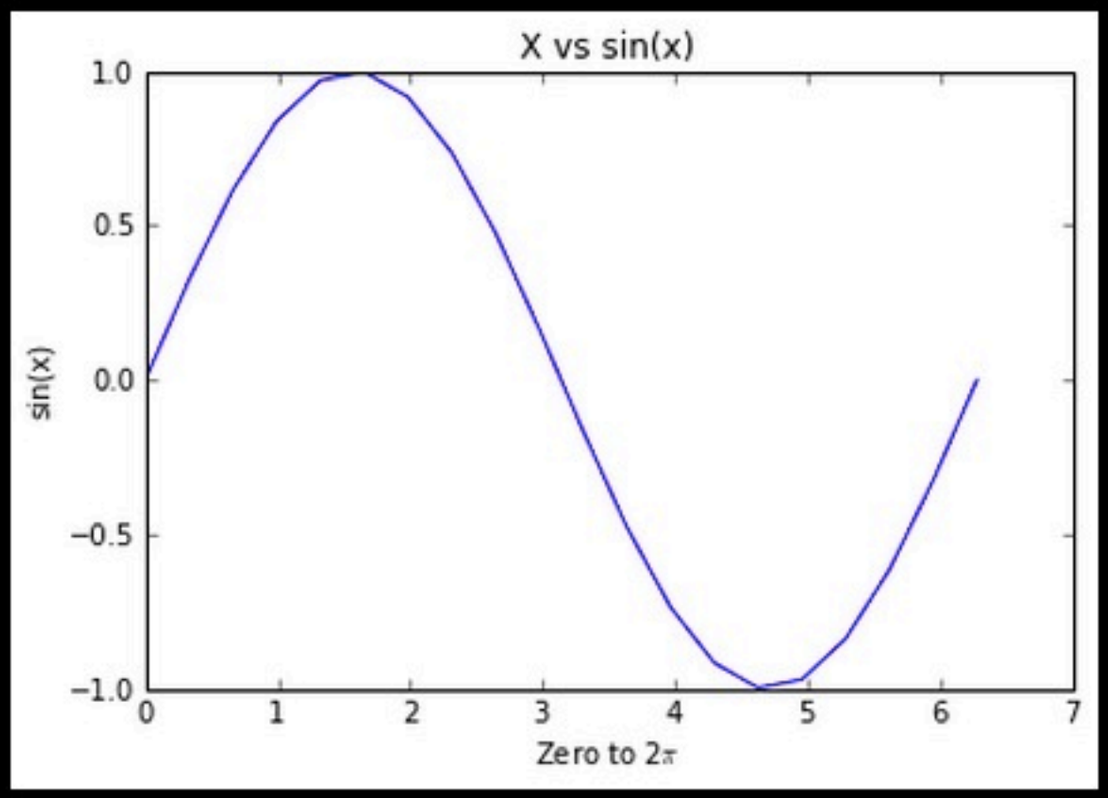
In [5]:
```
# overplot
plot(x,x)
plot(x,sin(x))
```

Out[5]: [<matplotlib.lines.Line2D at 0x105f5f590>]



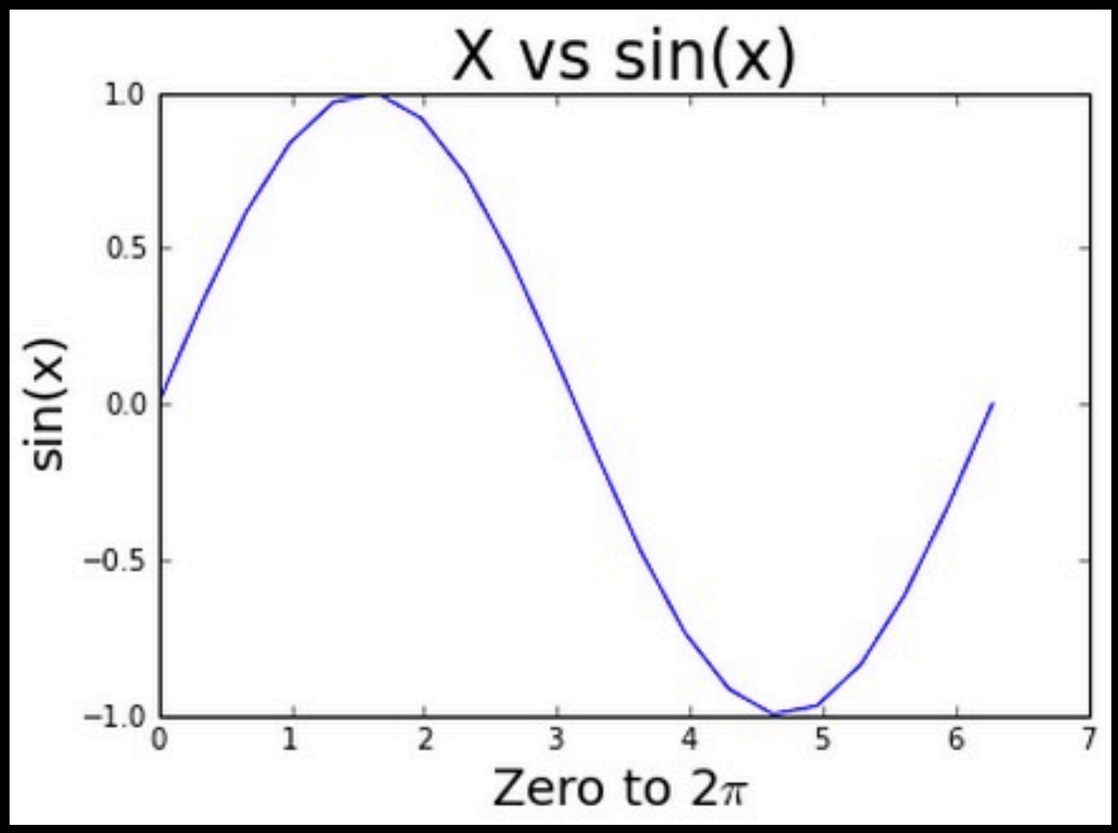
In [6]:
```
# axis labels, etc
plot(x,sin(x))
title("X vs sin(x)")
xlabel("Zero to 2$\pi$")
ylabel("sin(x)")
```

Out[6]: <matplotlib.text.Text at 0x106327d50>



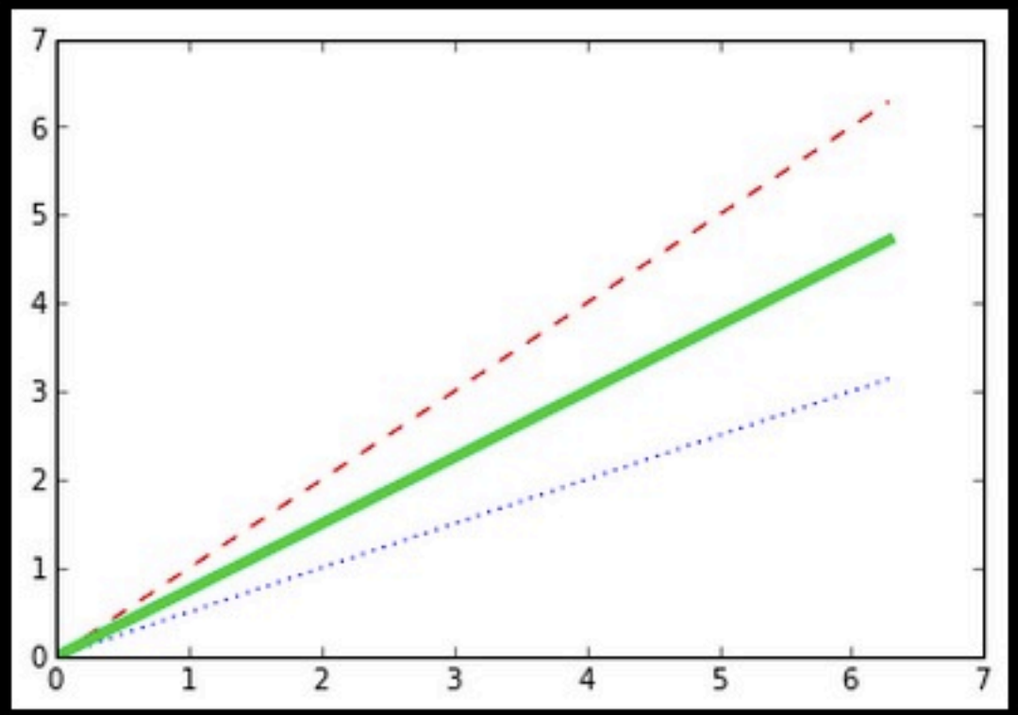
In [7]:
```
# font size
plot(x,sin(x))
title("X vs sin(x)",fontsize=24)
xlabel("Zero to 2$\pi$",fontsize=18)
ylabel("sin(x)",fontsize=18)
```

Out[7]: <matplotlib.text.Text at 0x1063be8d0>



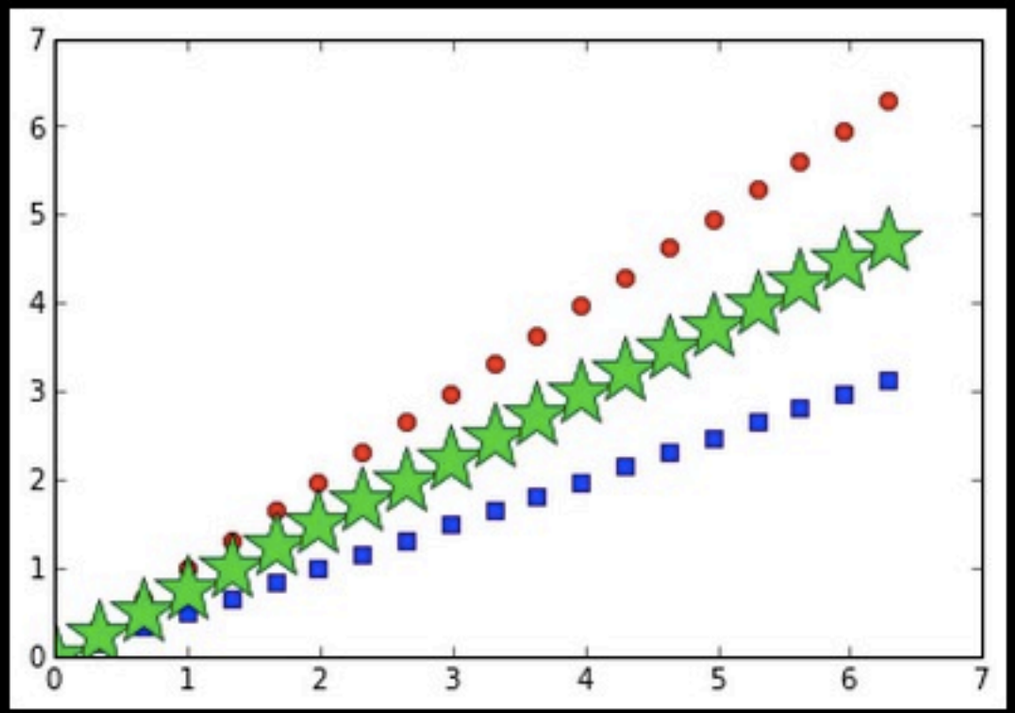
In [8]:
```
# linestyles
plot(x,x,color='r',linestyle='--') # red dashed
plot(x,x/2,color='b',linestyle=':') # blue dotted
# thick green solid
plot(x,x*3/4., color=(0.3,0.8,0.3), linestyle='-', linewidth=4)
```

Out[8]: [<matplotlib.lines.Line2D at 0x106722290>]



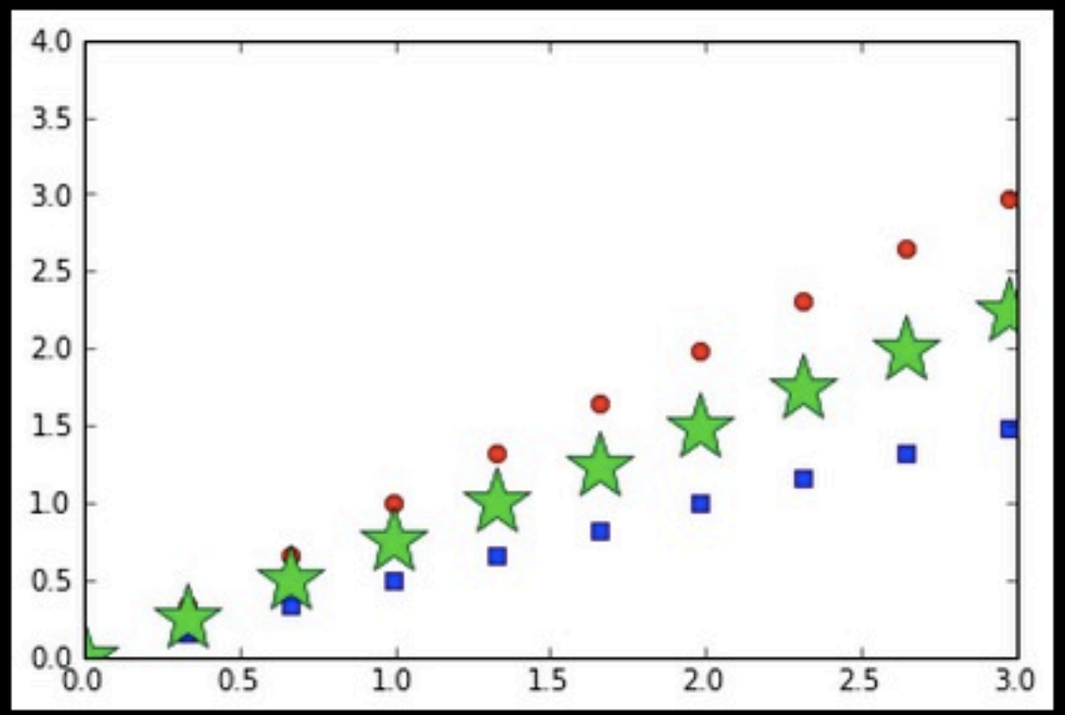
In [9]:
```
# plot symbols
plot(x,x,color='r',linestyle='none', marker='o') # red circles
plot(x,x/2,color='b',linestyle='none', marker='s') # blue squares
# green stars
plot(x,x*3/4., color=(0.3,0.8,0.3), linestyle='none', marker="*", markersize=25)
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x106976590>]
```



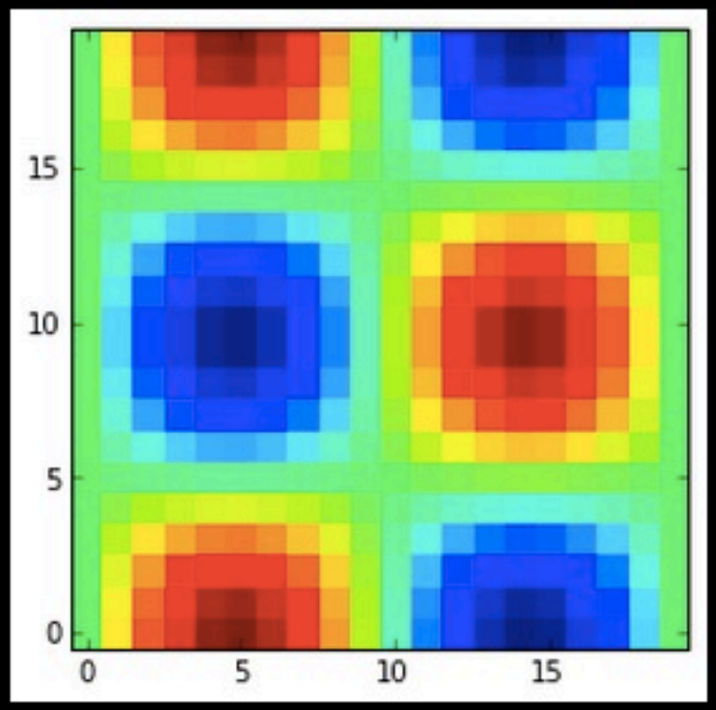
In [10]:
```
# plot symbols
plot(x,x,color='r',linestyle='none', marker='o') # red circles
plot(x,x/2,color='b',linestyle='none', marker='s') # blue squares
# green stars
plot(x,x*3/4., color=(0.3,0.8,0.3), linestyle='none', marker="*", markersize=25)
# set axis limits
axis([0,3,0,4])
```

Out[10]: [0, 3, 0, 4]



In [11]:
```
# image display
xx,yy = np.meshgrid(x,x)
zz = sin(xx)*cos(yy)
imshow(zz)
```

Out[11]: <matplotlib.image.AxesImage at 0x106a9e110>



You'll only get this with practice...

- To the lab