IDL Week 5: What we'll cover today

- Review exercise from Tuesday
- Poly_fit
- 3D plotting
Exercise #1

The first part of this exercise involved reading in multiple files and extracting their data into a single array:

```plaintext
sstfiles = file_search('sstdata/tmi.fusion.*.gz')
sstdata = fltarr(1440,720,n_elements(sstfiles)); x by y by time array
binarydata=bytarr(1440,720)
scale = 0.15
offset = -3.0
time = fltarr(132)
for i=0,n_elements(sstfiles)-1 do begin
  openr,lun,sstfiles[i],/get_lun,/compress
  readu,lun,binarydata
  free_lun, lun
  sstdata[*,*,i] = float(binarydata)*scale+offset
  ;get Julian time values for plotting PC1 later
  time[i] = JulDay(1,1,fix(strmid(sstfiles[i],19,4)))+fix(strmid(sstfiles[i],24,3))-1
endfor
print, n_elements(sstfiles)
sst_subset = sstdata[400:1119,280:439,*]; extract sub region
```
Exercise #2

The second step involves extracting the subset and reducing the resolution:

```plaintext
sst_subset = sstdata[400:1119,280:439,*] ;extract sub region

;rebin to 2-degree resolution - SVDC won't run with original resolution
nx = 90
ny = 20
sst_subset = rebin(sst_subset,nx,ny,132)
```
Exercise #3

Before calling SVDC, the data must be converted to anomalies and weighted by latitude:

```plaintext
;get monthly means
sst_subset = reform(sst_subset,nx,ny,12,11) ;convert to an x by y by month by year array
monthly_means = rebin(rebin(sst_subset,nx,ny,12),nx,ny,12,11)

;create anomaly time series & compact time into one dimension
sst_anom = reform(sst_subset-monthly_means,nx,ny,132)

;weight anomalies
lat = findgen(ny)-19.
weights = rebin(sqrt(cos(transpose(lat)*DTOR)),nx,ny,132)
sst_anom_wgt = sst_anom*weights

;compact spatial dimensions
sst_anom_wgt = reform(sst_anom_wgt,nx*ny,132)
```
Exercise #3

Once data is in the correct format, run **SVDC** and regress anomalies onto PCs to get EOFs:

```plaintext
;perform Singular Value Decomposition and get normalized PCs
SVDC, sst_anom_wgt, W, U, V, /DOUBLE
PC1 = REFORM(U(0,*) / STDDEV(U(0,*)))
PC2 = REFORM(U(1,*) / STDDEV(U(1,*)))
PC3 = REFORM(U(2,*) / STDDEV(U(2,*)))

EOF1 = fltarr(nx,ny)
FOR i = 0, nx-1 DO BEGIN
   FOR j=0, ny-1 do begin
      EOF1[i,j] = REGRESS(PC1, REFORM(sst_anom(i,j,*)))
   endfor
ENDFOR
```
Exercise #4

```plaintext
!p.multi=[0,1,2,0,0]
loadct,0
plot, time, PC1, background=255, color=0, chsize=2, title='Normalized PC1', xtickunits=['Years']

dmax = max(abs(eof1))
dmin = -dmax
pimage, EOF1, region=[-20,20,100,280], col_file='blueredanom.tbl', position=[0,0,1,0.5], noerase, vmin=dmin, vmax=dmax, title='EOF 1', lsize=2, tsize=2, coast
```
Exercise #5

The last step involved running the **FFT** procedure and converting the output into power (as a function of period). To test for significance, you would also need to do an autocorrelation analysis and compare to the red noise spectrum.

See `sst_eof.pro`
POLY_FIT

The **POLY_FIT** function performs a least-square polynomial fit with optional weighting and returns a vector of coefficients.

\[ a = \text{POLY}_\text{FIT}( X, Y, \text{Degree [}, \text{CHISQ=}\text{variable}], \text{MEASURE}_\text{ERRORS=}\text{vector}] \]

\( a \) is a vector of coefficients of length \( \text{degree+1} \), such that

\[ y = a[0] + a[1]x + a[2]x^2 + \ldots + a[n]x^n. \]

**CHISQR** returns the goodness-of-fit statistic

**MEASURE_ERRORS** is used to weight the data
In IDL, three-dimensional data can be plotted using the PLOT_3DBOX procedure:

```
PLOT_3DBOX, X, Y, Z, AX=degrees, AZ=degrees, /SOLID_WALLS, /XY_PLANE, /XZ_PLANE, /YZPLANE + graphics keywords
```

- **AX** sets the rotation about the X axis (zenith angle) towards the viewer in degrees.
- **AZ** sets the azimuth angle (rotation about the Z axis) of the viewer in degrees.
- **/SOLID_WALLS** fills the far boundaries with a solid color.
- **/[XYZ]PLANE** plots the data onto the corresponding axis plane.
Example: PLOT_3DBOX

plot_3DBOX, lon[0:nlon-1], lat[0:nlat-1],
elevation[0:nelev-1,0], background=255, color=0, AX=20, AZ=i, thick=2
3D Surface Plotting

A 3D surface can be plotted using the **SURFACE** (to display a wire mesh) or **SHADE_SURF** procedures:

**SURFACE**, $Z[X, Y][, AX=\text{degrees}][, AZ=\text{degrees}] +$ graphics keywords

**SHADE_SURF**, $Z[X, Y][, AX=\text{degrees}][, AZ=\text{degrees}]$, $[\text{SHADES}=\text{array}] +$ graphics keywords

Setting the **SHADES=** array keyword will color the 3D surface with the values of array (which should be the same size as $Z$), instead of light/shadow values.

The light source can be controlled with the **SET_SHADING** procedure:

**SET_SHADING** $[, /\text{GOURAUD}] [, LIGHT=[x, y, z]]$ $[, /\text{REJECT}] [, VALUES=[\text{darkest, brightest}]]$
Example: SHADE_SURF

SHADE_SURF, dem,
   x0+ds*findgen(nx),y0+ds*findgen(ny),AX=30,AZ=i,zrange=[min(dem),min(dem)
   +30.*nx],zstyle=1,background=255,color=0,/save