# IDL Week 4:
# What we'll cover today

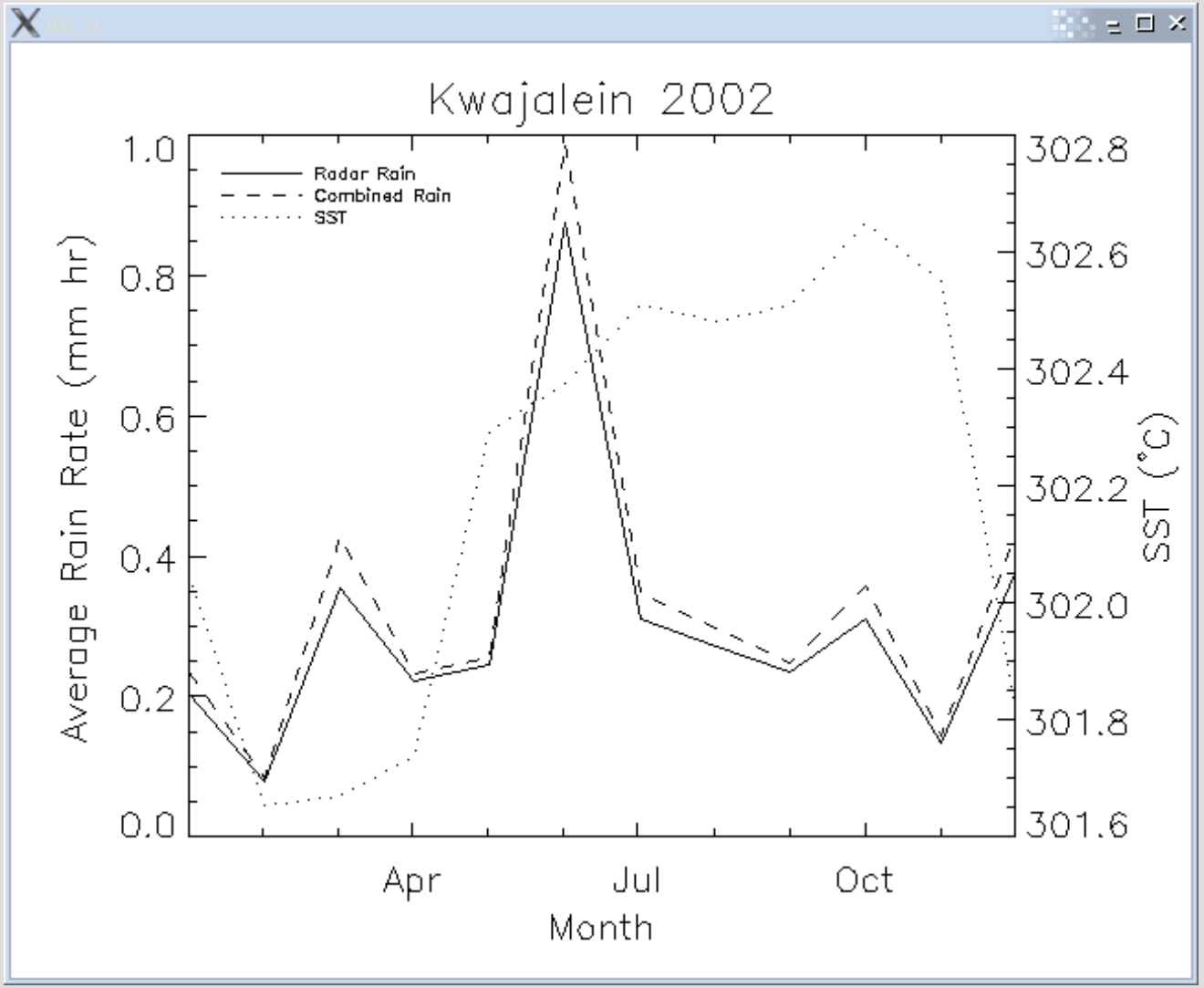
- Review exercises from last week
- Some canned plotting routines
  - pcontour
  - pimage
  - Pmap
- Statistics and Mathematics
  - Regression
  - Probability & Hypothesis Testing
  - Time Series Analysis
  - Histograms
  - EOFs
- Organizing Your Code

# Exercise #1

The intent of this exercise was to become more familiar with the **AXIS** procedure and graphics keywords.

```
IDL> plot, (month-1)*365./12., rain1,background=255,color=0,charsize=2,title='Kwajalein
   2002',xtitle='Month',ytitle='Average Rain Rate (mm
   hr)',xtickunits='month',xrange=[0,334],xstyle=1,ystyle=8,position=[0.15,0.15,0.85,0.9]
IDL> oplot, (month-1)*365/12.,rain2,linestyle=2,color=0
IDL> axis, /yaxis, yrange=[min(sst),max(sst)],/save,color=0,ytitle='SST ('+string("232B)
   +'C)',charsize=2
IDL> oplot, (month-1)*365/12.,sst,linestyle=1,color=0
IDL> legend,['Radar Rain','Combined Rain','SST'],linestyle=[0,2,1],color=0,textcolor=0
```

# Exercise #1

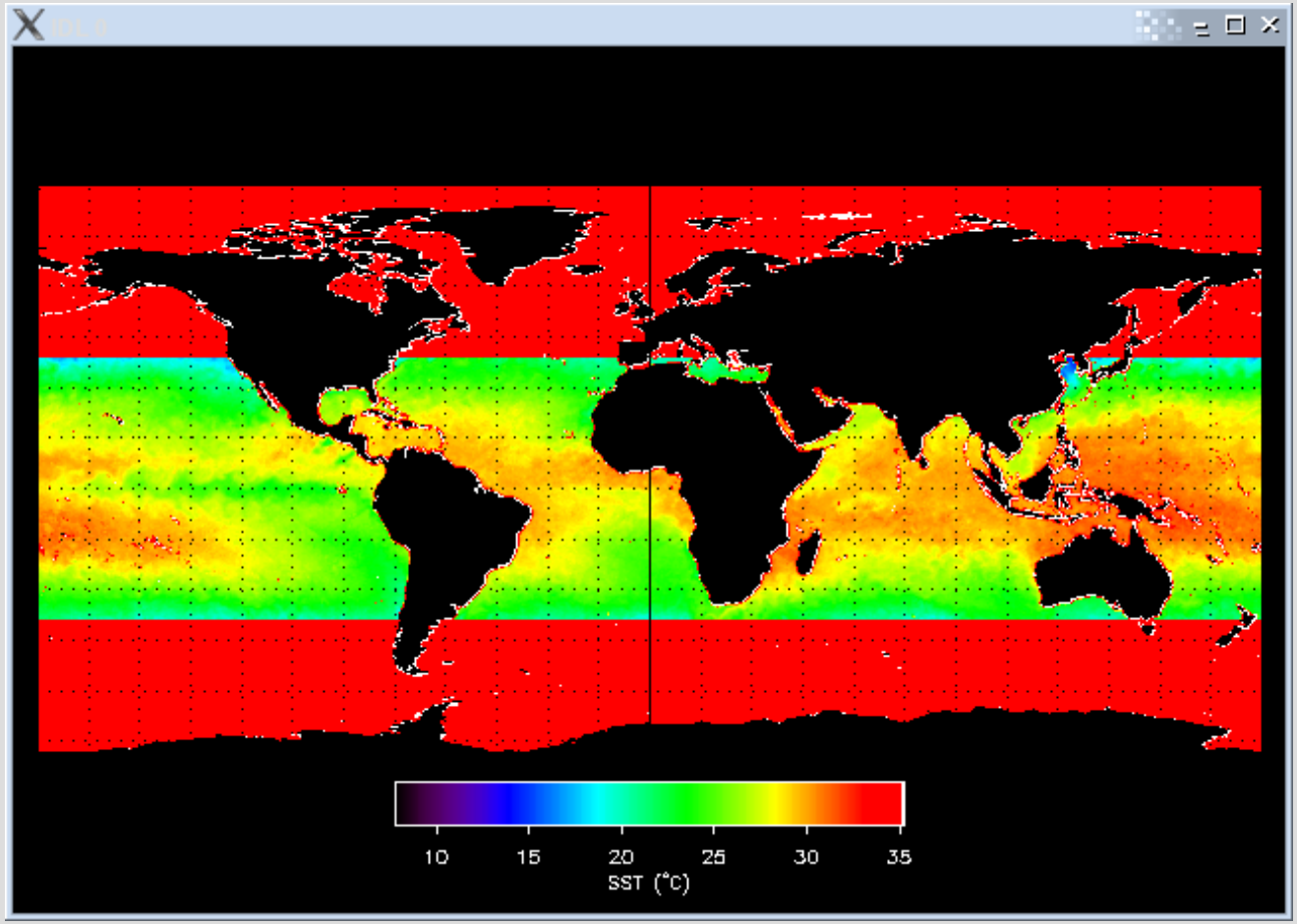

Kwajalein 2002
Radar Rain
Combined Rain
SST
Average Rain Rate (mm hr)
SST (°C)
1.0
0.8
0.6
0.4
0.2
0.0
302.8
302.6
302.4
302.2
302.0
301.8
301.6
Apr
Jul
Oct
Month

# Exercise #2

- The intent of this exercise was to learn the **MAP_IMAGE** procedure and when it should be used instead of **CONTOUR**.
- The **contour** procedure produces a PS file size of 52.3 MB!
- The **map_image** file size is only 215 KB.

```
map_set, 0.,0.,/cylindrical,/isotropic
result= map_image(sst, startx, starty,xs,ys,/bilinear, compress=1,
   latmin=lat[0],latmax=lat[719], lonmin = lon[0], lonmax=lon[1439]+360.)
tvscl, result,startx,starty,xsize=xs,ysize=ys
map_continents,color=0,fill_continents=1
map_grid,color=0,latdel=15.,londel=15.
colorbar, vmin=min(sst),vmax=max(sst),cmin=0,cmax=255,title='SST ('+string("232B)
   +'C)',position=[0.3,0.1,0.7,0.15]
```

# Exercise #2



# Pcontour

The procedure **pcontour** is intended to display or plot a contour map of regularly spaced data in a 2D array and automatically includes a color bar. In addition to the standard contour procedure keywords and graphics keywords, **pcontour** takes the following:

***UNITS***=*string*: units label for colorbar

***PROJECT***=[cyl,mer,mol,sat,polN,polS,xy]: Map projection (xy for Cartesian)

***PS_FILE***=*filename*: name for output PS (do not need to set display)

***REGION***=[*minlat*,*maxlat*,*minlon*,*maxlon*]: Set map bounds and data coordinates

***MIS_VAL***=*missing_value*, ***MSK_VAL***=*mask_value*: mask missing data

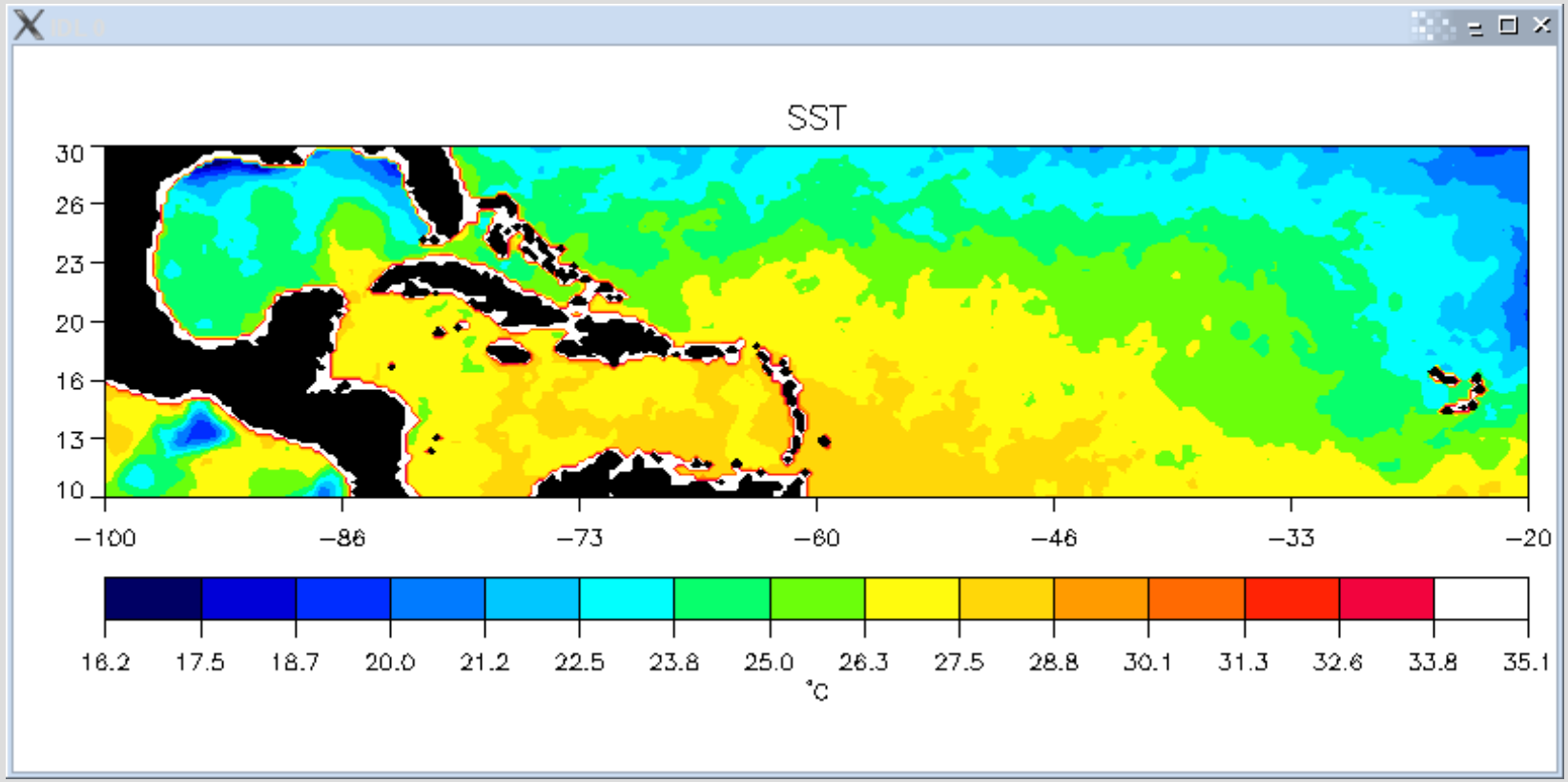
See pcontour.pro or simply run 'pcontour' without arguments to see more.

Note: pcontour requires hdr_intf.pro, extract.pro, load_ctbl.pro, imgsize.pro, map_axis.pro, gen_labels.pro, and cont_cbar.pro.

You will also need to modify the color table path in load_ctbl.pro.

# Example: pcontour

pcontour, sst[1040:1359,400:479],
region=[10.,30.,-100.,-20.],ncolors=32,col_file='rain.tbl',title='SST',UNITS=string("232B)+'C',nlevels=15,/coast,mis_val=35.25,msk_val=35.10

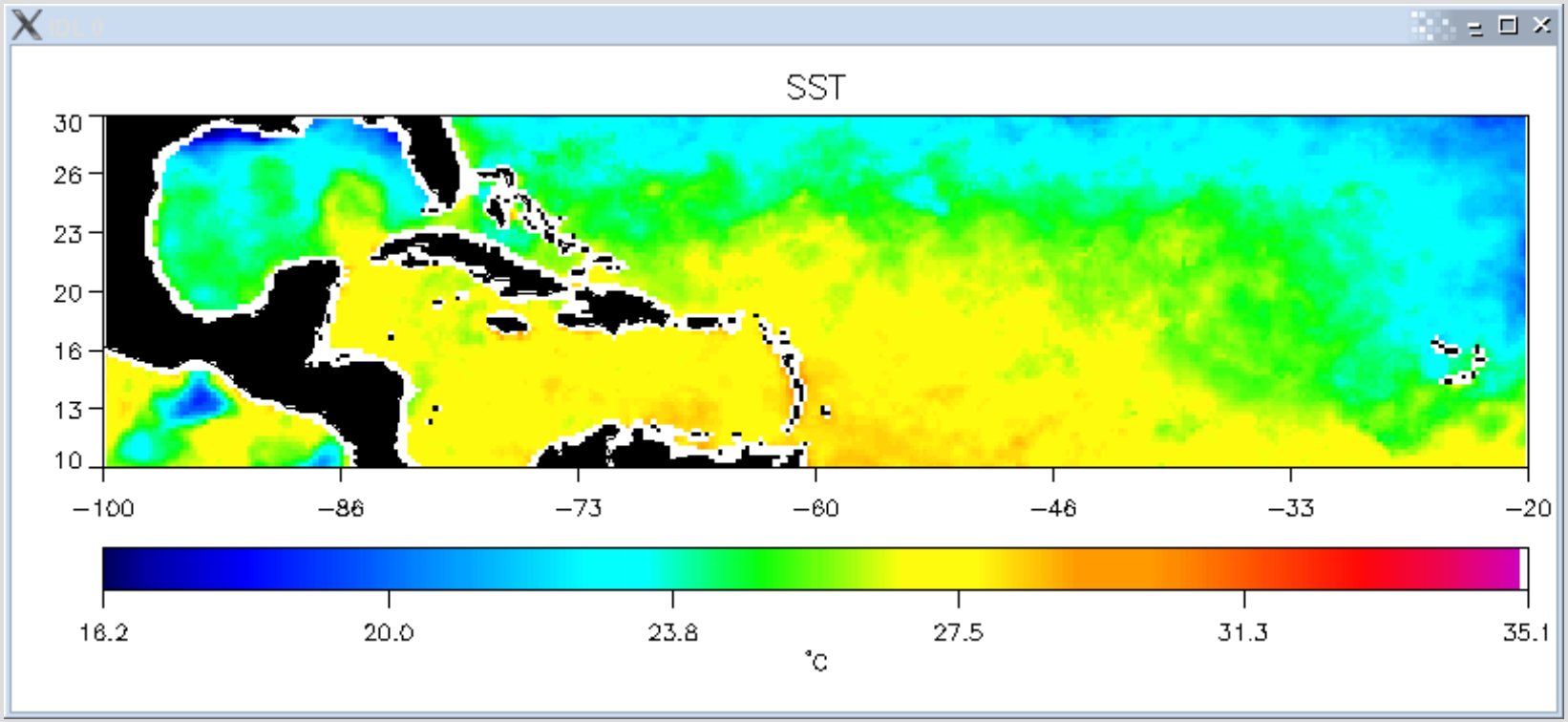


# Pimage

The **pimage** procedure is similar to **pcontour**, except that it uses map_image internally. pimage.pro requires the same files as pcontour.pro, and in addition, expscl.pro.

Example:

```
pimage, sst[1040:1359,400:479],  region=[10.,30.,-100.,-20.], col_file='rain.tbl',
   title='SST',UNITS=string("232B)+'C',/coast,mis_val=35.25,msk_val=35.10
```



# Pmap

**Pmap** is similar to **pcontour** and **pimage**, except that it creates an image out of individual dots. This offers the advantage of being able to use irregularly positioned data, like **contour**, with the size advantage of **map_image**.

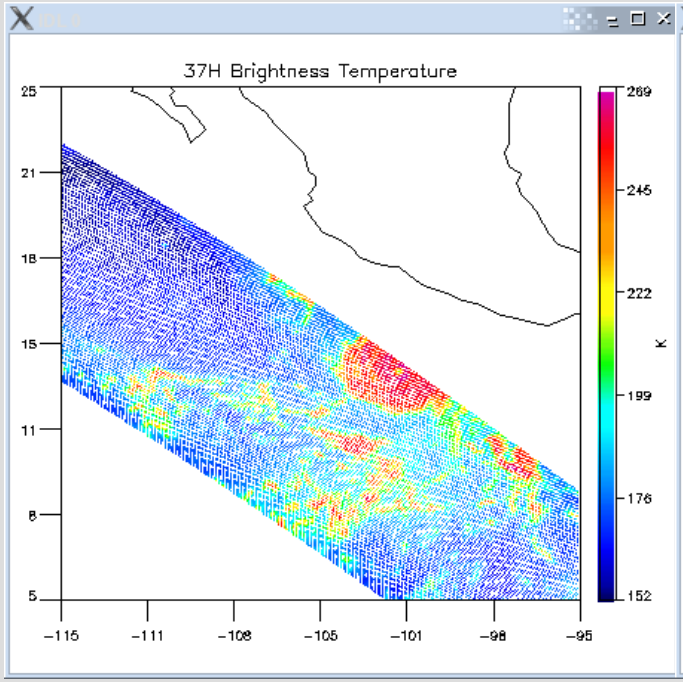
**Pmap** takes the same keywords as **pcontour** and **pimage**, and requires expscl_limits.pro in addition to the previous files.

Syntax: **Pmap**, *data*, *lat*, *lon*[,***keywords***]

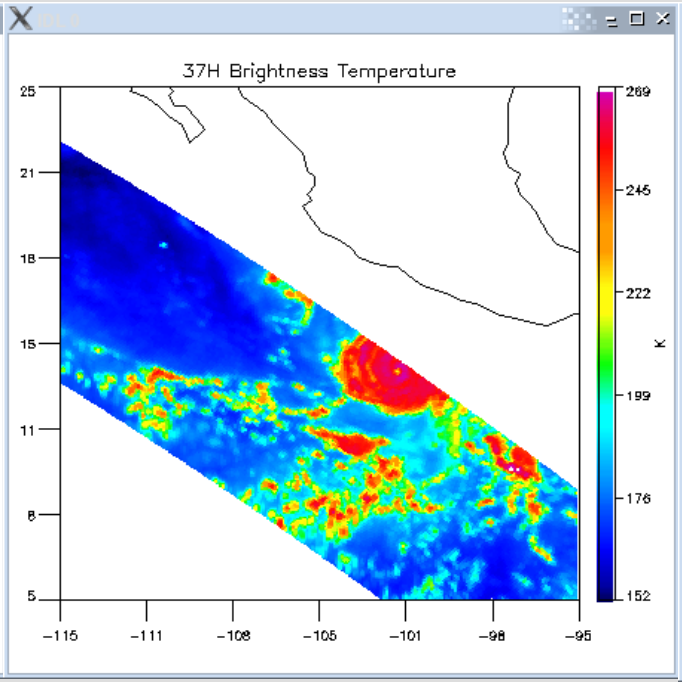
# Pmap tips

The ***psize*** keyword is used to set the dot size (default=1.0). Try to use a dot size that doesn't leave data gaps, but doesn't obscure other dots either:

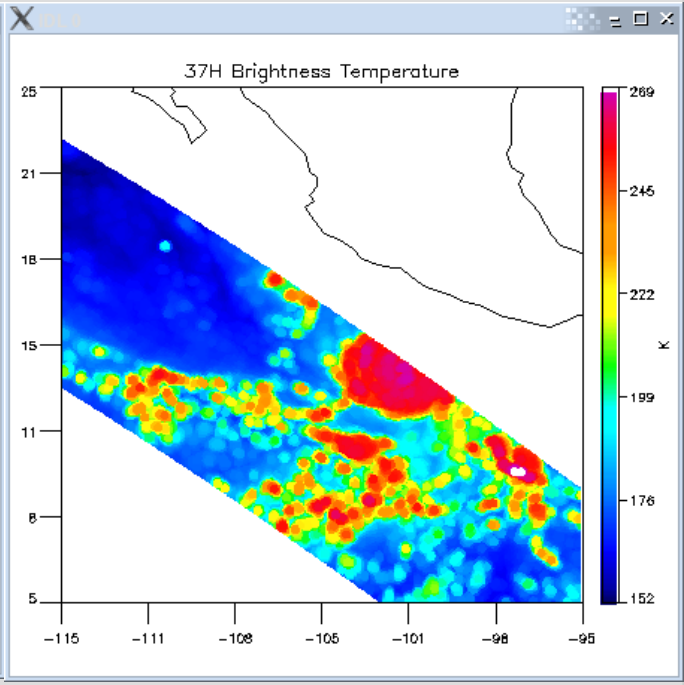
Psize=1.0




Psize=2.2




Psize=5.0




Example code: pmap_tmi.pro

# Manipulating Data: rebin

The **rebin** function shrinks or enlarges an array by an integer factor:

*result* = **rebin**(*array*,*new_dim1*,*new_dim2*,…,***/sample***)

The ***/sample*** keyword instructs **rebin** to use nearest neighbor sampling when expanding an array:

```
IDL> print, rebin(findgen(3),6,/sample)
     0.00000      0.00000      1.00000      1.00000      2.00000      2.00000
```

Otherwise, the function will interpolate (but not extrapolate) values:

```
IDL> print, rebin(findgen(3),6)
     0.00000     0.500000      1.00000      1.50000      2.00000      2.00000
```

When shrinking an array, rebin uses neighborhood averaging.

# rebin tips and tricks

You can use **rebin** to create multi-dimensional index arrays that only increase along one dimension (useful for later contouring a subset using ***/irregular***):

```
IDL> print, rebin(indgen(5),5,3)
      0       1       2       3       4
      0       1       2       3       4
      0       1       2       3       4
IDL> print, rebin(reform(indgen(5),1,5),3,5)
      0       0       0
      1       1       1
      2       2       2
      3       3       3
      4       4       4
```

Another common use of **rebin** is to remove the seasonal cycle from a regularly-sampled time series.

# Data Analysis: Regression

The regress function performs a multiple linear regression fit and returns a column vector of coefficients. Regress fits the function:

*y*[i] = *const* + *a*[0]*x*[0,i]+*a*[1]*x*[1,i] + ... + *a*[*Nterms*-1]*x*[*Nterms*-1,i]

a = **REGRESS**( *X*, *Y*, [, ***CHISQ***=*variable*] [, ***CONST***=*variable*] [, ***CORRELATION***=*variable*] [, ***/DOUBLE***] [, ***FTEST***=*variable*] [, ***MCORRELATION***=*variable*] [, ***MEASURE_ERRORS***=*vector*] [, ***SIGMA***=*variable*] [, ***STATUS***=*variable*] [, ***YFIT***=*variable*] )

If *X* is one dimensional, then a simple linear regression will be performed.

# Regression Keywords

***CONST***=*c*: Set this to the constant term in the linear equation.
***/DOUBLE***: Return values as doubles (if input is not double)
***CHISQ***=*chi*: value of the unreduced chi-square goodness-of-fit statistic
***FTEST***=*f*: F-value for the goodness-of-fit test
***CORRELATION*** = *r*: vector of linear correlation coefficients
***MCORRELATION***=*mr*: multiple linear correlation coefficient
***MEASURE_ERRORS***=*Sy*: set standard error for each term in *Y*. This can be used to weight some data points more than others (weights are inverse of error)

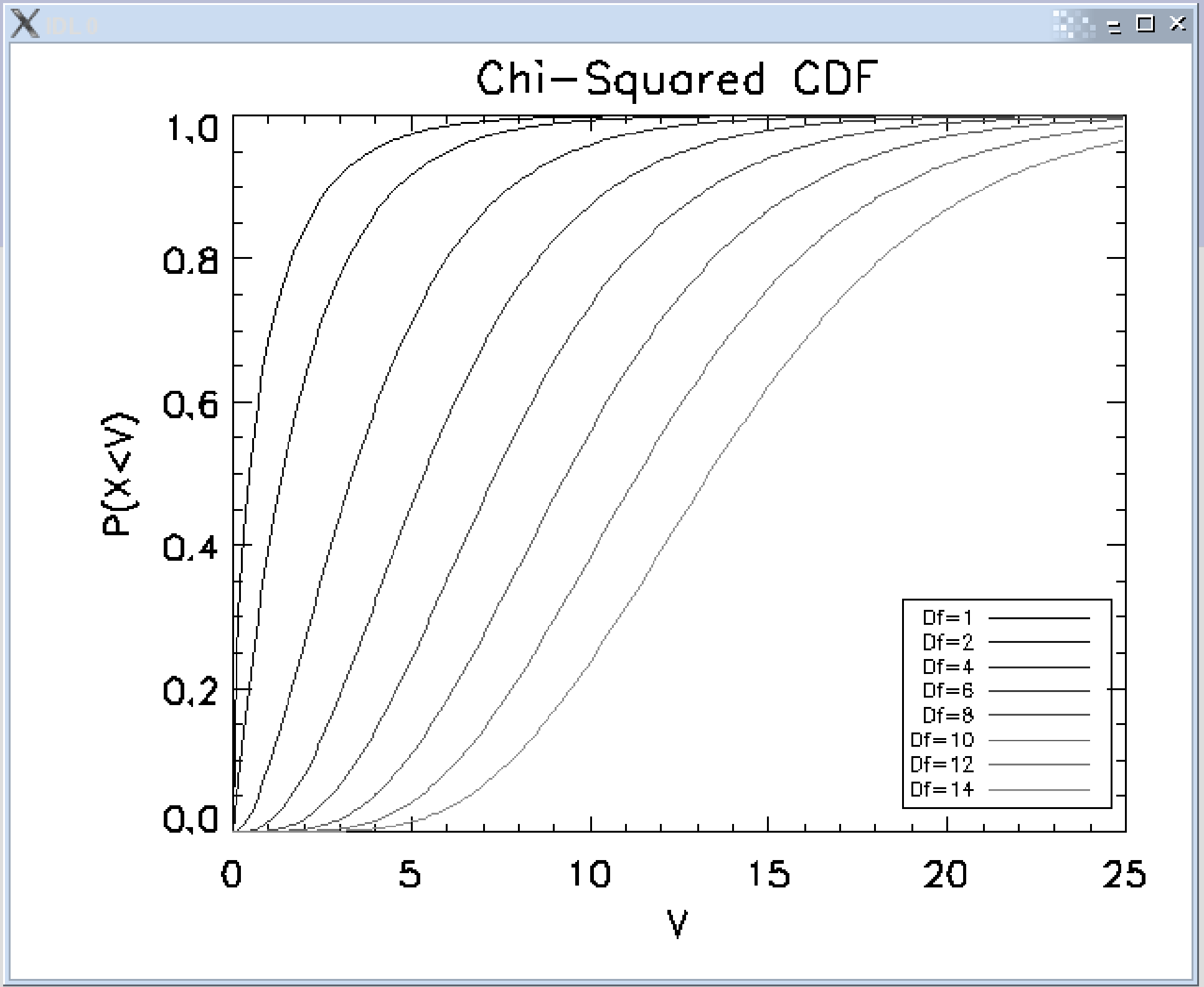
If you are just looking for the correlation coefficient, the *r*=**correlate**(*x*,*y*) function will provide that statistic.

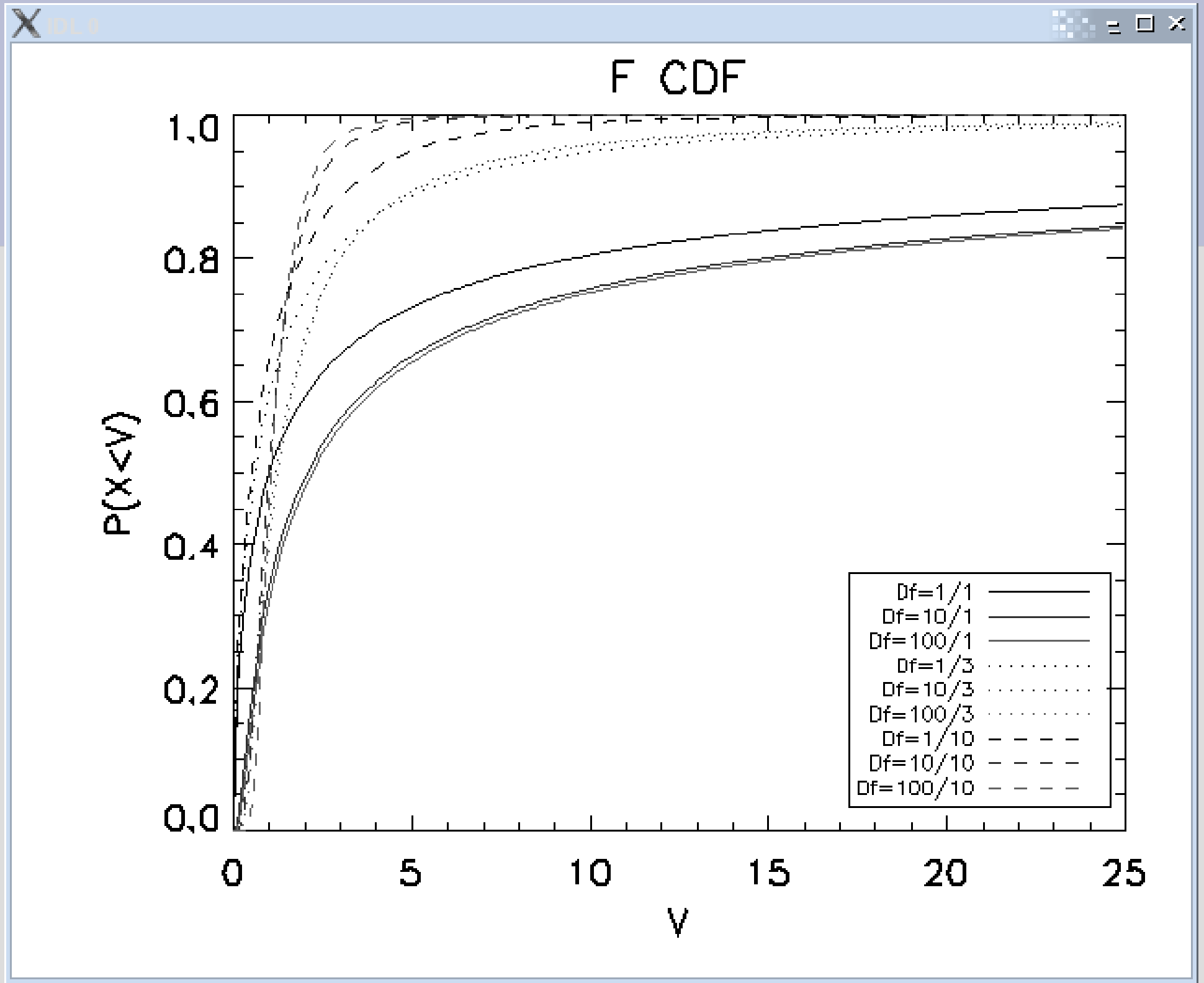
# Probability and Hypothesis Testing

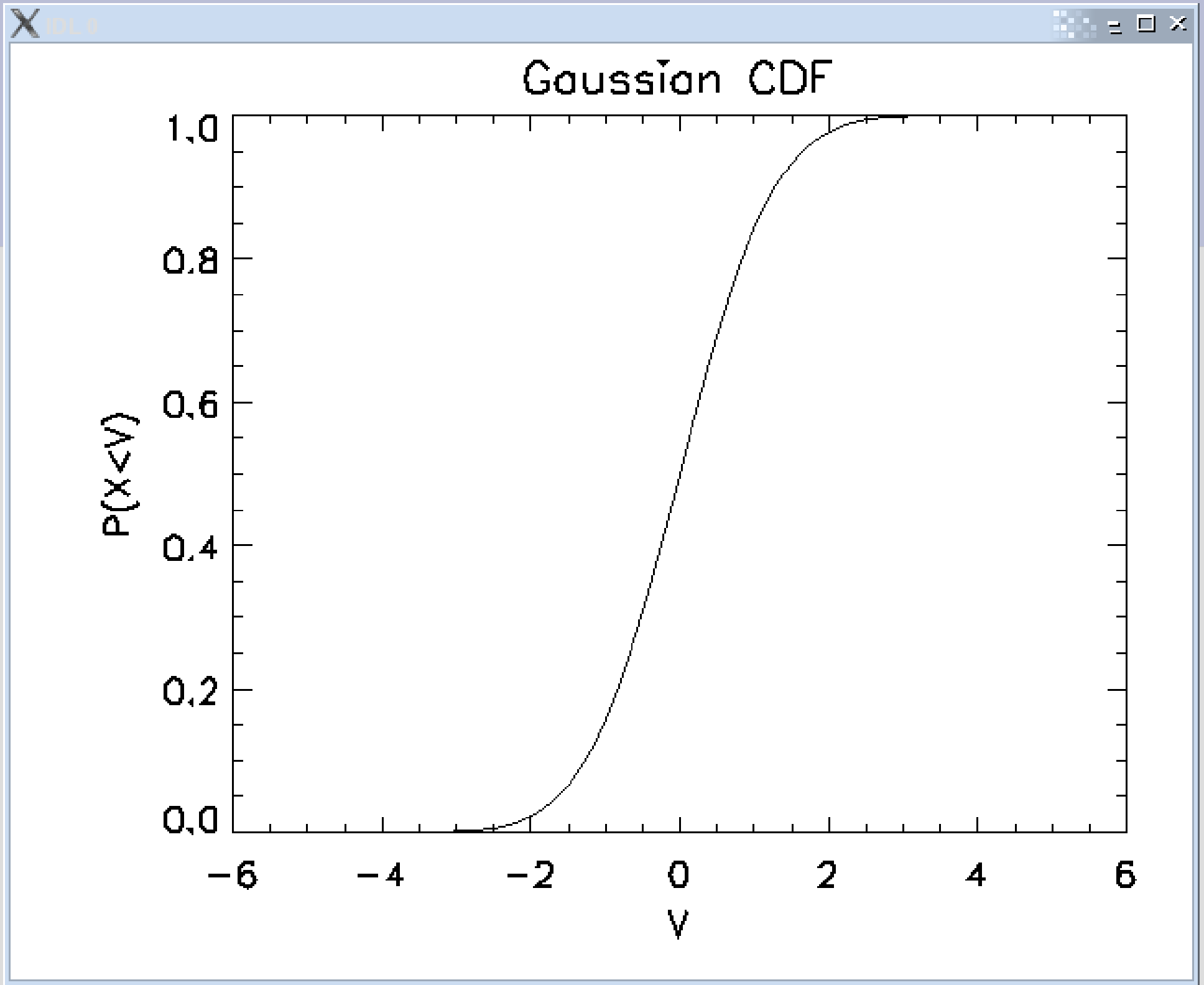
Often, we obtain a statistic from a regression, correlation, or other test, and want to determine probability that the result is not due to random chance. Depending on the type of distribution of that statistic, IDL has several look-up functions to calculate these probabilities:

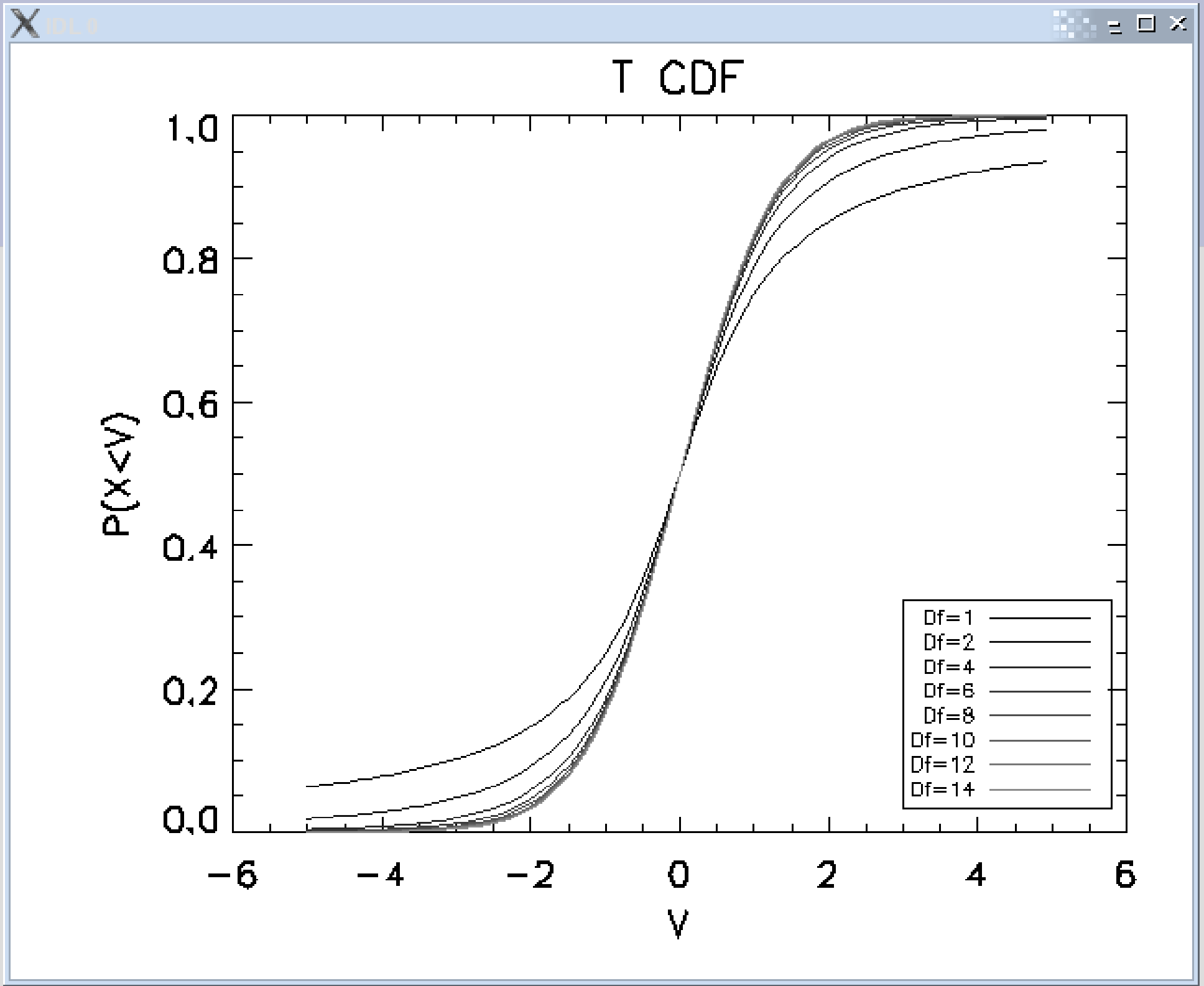
*V*=**CHISQR_CVF**(*P*,*Df*): Cutoff values for chi-squared distribution.
*P*=**CHISQR_PDF**(*V*,*Df*): pdf of chi-squared distribution.
*V*=**F_CVF**(*P*,*Dfn*,*Dfd*): Cutoff values for an F distribution.
*P*=**F_PDF**(*V*,*Dfn*,*Dfd*): pdf of the F-distribution.
*V*=**GAUSS_CVF**(*P*): Cutoff values for a Gaussian distribution.
*P*=**GAUSS_PDF**(*V*): pdf of the Gaussian distribution.
*V*=**T_CVF**(*P*,*Df*): Cutoff values for a T distribution.
*P*=**T_PDF**(*V*,*Df*): pdf of the T-distribution.

For each of these functions, *V* is the cutoff value of the appropriate statistic such that the probability that a random value *X* is less than *V* equals *P*. The degrees of freedom is given by *Df* in the chi-squared and T distributions, and for the F-distribution, the degrees of freedom in the numerator (denominator) is given by *Dfn* (*Dfd*).


IDL 0
Chi-Squared CDF
1.0
0.8
0.6
0.4
0.2
0.0
P(X<V)
0
5
10
15
20
25
V
Df=1
Df=2
Df=4
Df=6
Df=8
Df=10
Df=12
Df=14


IDL 0
F CDF
1.0
0.8
0.6
0.4
0.2
0.0
P(X<V)
0
5
10
15
20
25
V
Df=1/1
Df=10/1
Df=100/1
Df=1/3
Df=10/3
Df=100/3
Df=1/10
Df=10/10
Df=100/10


IDL 0
Gaussion CDF
1.0
0.8
0.6
0.4
0.2
0.0
P(X<V)
−6
−4
−2
0
2
4
6
V


IDL 0
T CDF
1.0
0.8
0.6
0.4
0.2
0.0
P(X<V)
−6
−4
−2
0
2
4
6
V
Df=1
Df=2
Df=4
Df=6
Df=8
Df=10
Df=12
Df=14

# Histograms

The **histogram** function computes the density distribution of an array:

*result* = **HISTOGRAM**(*array*[,***binsize***=*value*]
[,***locations***=*array*][,***max***=*value*][,***min***=*value*][,***/NAN***]
[,***nbins***=*value*][,***omax***=*value*][,***omin***=*value*]

result contains the number of elements in each bin

Keywords:

***binsize***=*value*: specify bin width

***locations***=*array*: outputs the minimum value of each bin

***max***/***min***: specify max and min cutoffs

***/NAN***: ignore NaN or Inf values

***nbins***=*value*: set the number of bins

***omin***/***omax***: output the min and max if automatically calculated

# 2D Histograms

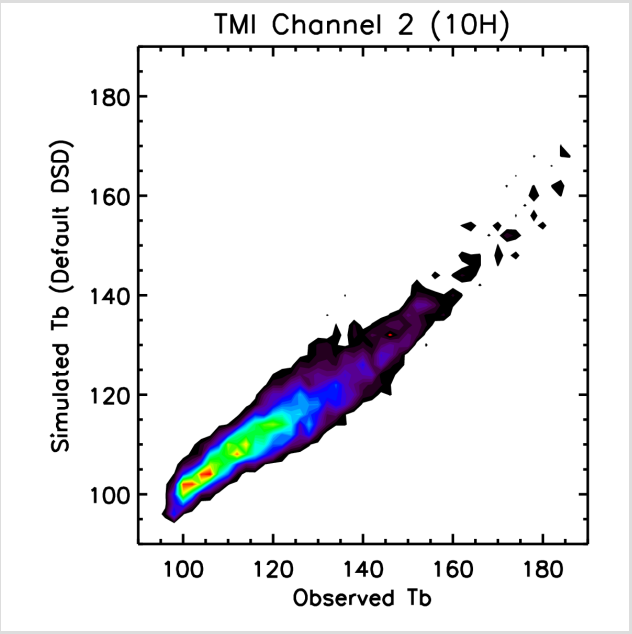
An alternative to scatter plots, especially when the data are extremely dense, is a 2D histogram. Instead of nested **for** loops, use the **HIST_2D** function:

*result* = **HIST_2D**(*array1*,*array2*[,***bin1***=*value*][,***bin2***=*value*][,***min1***=*value*][,***min2***=*value*][,***max1***=*value*][,***max2***=*value*]

*array1* and *array2* should be the same length (IDL will ignore extra data). If *bin1* and *bin2* are not specified, a value of 1 will be assumed.

Example:

```
contour, hist_2d(tmitb_obs[a],  tmitb_dflt[a],
   bin1=2., bin2=2., min1=90., min2=90.,
   max1=190., max2=190.), 90.+2.*findgen(51),
   90.+2.*findgen(51), /fill, nlevels=32,
   xrange=[90,190], yrange=[90,190], xstyle=1,
   ystyle=1, xthick=4, ythick=4, charthick=4,
   xtitle='Observed Tb', ytitle='Simulated Tb
   (Default DSD)', title='TMI Channel 2 (10H)'
```



# Time Series Analysis: Autocorrelation and Cross Correlation

The autocorrelation of a 1D vector (assumed to be regularly sampled) can be found with the **A_CORRELATE** function:

*result*=**A_CORRELATE**(*series*,*lag*[,***/covariance***][,***/double***])

Likewise, the cross correlation of one time series with another can be found with **C_CORRELATE**:

*result* = **C_CORRELATE**(*series1*,sereis2,*lag*[,***/covariance***][,***/double***])

If *lag* is a vector, *result* will be a vector of the same length containing the autocorrelation at the given lag intervals.

If ***/covariance*** is specified, the autocovariance will be given instead of the auto correlation.

***/double*** ensures that calculations and output are given in double precision, even if single precision input is given.

# Time Series Analysis: Power Spectra

The power spectrum of a time series can be obtained with the Fast Fourier Transform (**FFT**) function. However, some manipulation is required.

*result*=**FFT**(*series*[,***/INVERSE***])

Use the ***/INVERSE*** keyword to compute the inverse Fourier Transform.

*result* is a complex array of the same length as the input time series. The *n*th element of result contains the Fourier coefficients corresponding to the frequency *n*/(NΔT).

To get the power spectrum, simply compute the magnitude at each frequency:

$C^2 = A^2+B^2$

# EOF Analysis

1. Decompose matrix A such that $A = U\Sigma V^T$

- A - space x time array, seasonal cycle removed, weighted by area (cos(latitude))
- Columns of V are the EOFs
- Columns of U are the PCs
- Do this using eigenanalysis, or using the SVD routine in IDL

2. Standardize the PC time series

3. Regress the original data onto the PC time series (PC time series is the independent variable)

# EOF analysis

The SVDC procedure computes the Singular Value Decomposition (SVD) of a square (n x n) or non-square (n x m) array as the product of orthogonal and diagonal arrays.

**SVDC**, *A*, *W*, *U*, *V* [, ***/COLUMN***] [, ***/DOUBLE***]
*A*: space x time array of area-weighted anomaly values
*W*: contains eigenvalues
*U*: PCs
*V*: EOFs

See eof_example.pro

# Organizing Your Code

At this point, you've probably accumulated some routines that you will want to use over and over again. Instead of copying these to a new directory each time, create a common IDL directory and place those files in it. Then, modify the IDL search path using the following code:

```
!path = !path + ':' + expand_path('+/home/jmunchak/idl' )
```

You will probably not want to do this every time you run IDL. So, create an IDL startup file (e.g., idlstartup.pro) that contains any commands you wish to run each time you start IDL (such as setting graphics output, color tables, and the search directory). Then, tell IDL to run this file every time you start IDL. In Linux/Unix, this is done by modifying your .bashrc or .cshrc files:

```
C shell: setenv IDL_STARTUP /path/to/startupfile
Bash: IDL_STARTUP=”/path/to/startupfile” && export IDL_STARTUP
Windows and Mac: set in the IDL Preferences Dialog under the Window menu.
```

# Next Class

This Friday is your chance to answer any questions that you may still have, or cover additional topics of your choice. The only planned material I have is to review today's exercises, so please email me your requests (preferably by the end of the day tomorrow!)