Examples of *S-Lang* in Data Analysis[*]

David Huenemoerder (dph@space.mit.edu)
MIT/CXC

27 January 2003

Introduction

I present here some simple examples of fundamental analysis functions as implemented in intrinsic *S-Lang* and *S-Lang*-based packages which are important for data analysis:

- manipulating arrays with the "`where()`" function;
- making histograms, on arbitrary grids;
- extending the analysis system;
- regridding histograms; and
- smoothing arrays, using complex arithmetic.

These examples are primarily intended to convey the nature of analysis in a *S-Lang*-based system. I have used stand-alone ISIS as the platform, since some functions are not available elsewhere. But since it also exists as an importable module, it can be used to provide its specific capabilities to other analysis environments with a *S-Lang* interpreter.

[*] `http://space.mit.edu/CXC/docs.html#SLang`

1 The `where` Function (Your New Best Friend)

`where` is one function you cannot do without. Given boolean operations to find array elements meeting arbitrary criteria, `where` will tell you their indices. `where` is the "dmfilter" of *S-Lang*.

A Simple Example

```
 x = [ 5 : 10 : 1 ] ; % generate an array
 a = ( x <= 7 ) ;     % new array, value 1 if x <= 7, 0 otherwise.
 b = ( x >= 7 ) ;     % another, but equal 1 only if x >= 7
 print(a) ;           % let's look
1
1
1
0
0
0

 print( x[ where( a ) ] );    % print selected x elements for
                              % the first conditional
5
6
7

 l = where( a and b ); % define array elements for both conditionals true
 print( x[ l ] );      % print the value of x for both being true
7

 print( where(a) );     % Directly see what where does
0
1
2
```

A More Complicated Example

```
fevt1 = "acisf01451_000N002_evt1.fits";  % pick an event file
(x,y,s) = fits_read_col(fevt1, "x", "y", "status" ); % read some columns
x0 = 4060.86 ;   % set the source position (or read, compute, ...)
y0 = 4116.83 ;

% find events within a given distance from the source position:
l = where( hypot( x-x0, y-y0 ) < 30 ) ;

% define relative coordinates for the selected events:
dx = x[l] - x0 ;       dy = y[l] - y0 ;

% make scatterplot of points in the region:
p1=open_plot("/xwin"); resize(16,1.0); erase;
connect_points(0);   yrange; xrange;
label("dx [pix]", "dy [pix]", "Source events profile (red=afterglow)");
plot( dx, dy );

%%% Now find certain status bits, and overplot:

ag_bits = 0xf shl 16;         % 16:19 are for ``afterglow''
lg = where ( s[l] & ag_bits );

oplot( dx[lg], dy[lg], red );
```

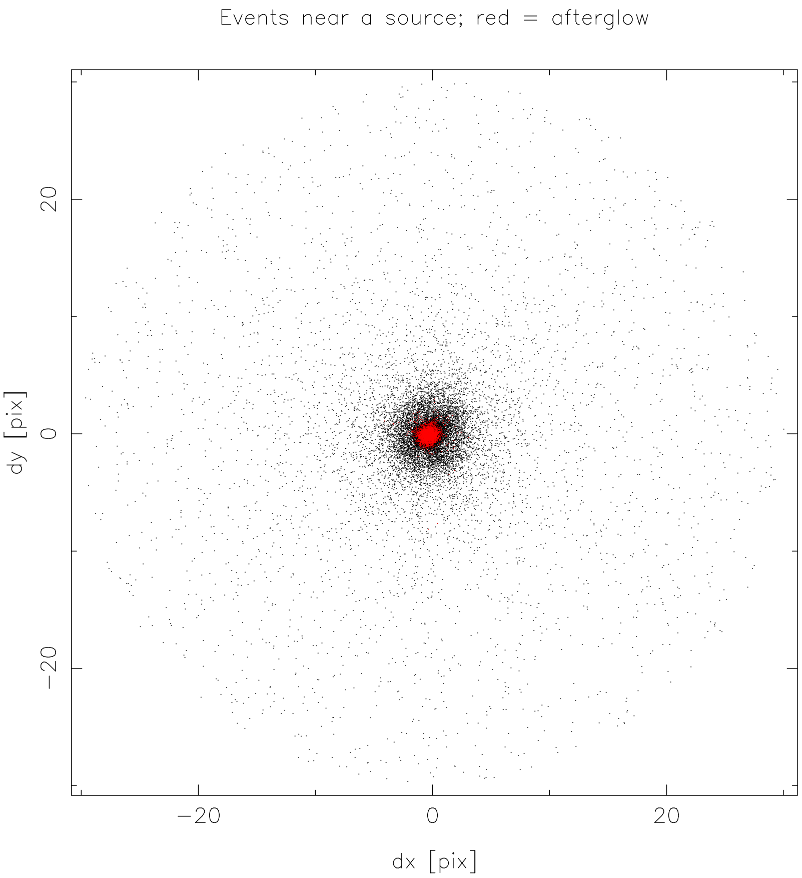

Figure 1: Events selected by radius from a given location, with red events selected among those via afterglow bits.

Comments

This simple example could have been done with **dmcopy** and a circular region for the source events, then with another **dmcopy** to select the afterglow events. Each file could then be viewed with **ds9**, for example. The equivalent commands are:

```
dmcopy acisf01451_000N002_evt1.fits"[col x,y,status,grade]"\
"[(x,y)=circle(4060.86,4116.83,30)]" src.fits
dmcopy src.fits"[exclude status=xxxxxxxxxxxx0000xxxxxxxxxxxxxxxx]"\
       srcglow.fits
ds9 -log -tile -zoom 2 -cmap bb -geometry 640x640 src.fits srcglow.fits &
```

The *S-Lang* example takes about 4 seconds to execute (the evt1 file is about 190 MB), and produces no files. The two dmcopy's take about 7 seconds and produce 2 output files. The difference is not a significant factor, so analysis style and scripting capability can be considered instead.

For multiple filters, however, the *S-Lang* method is clearly more efficient, since the full file read only needs to be done once. Ten filters (for 10 sources in one event list, for example) takes about 14 seconds, but the ten dmcopy filters takes 90 seconds.

2 Histograms (Your Second-Best Friend)

Making histograms ("binning") is a fundamental operation of most data analysis, whether it is in one-dimension for spectra or radial profiles or two dimensions for spatial images, spatial-spectral, or other coordinates. The unix command-line binning programs are dmcopy and dmextract. In ISIS, we have the low-level functions, histogram and histogram2d. Along with these are utilities for handling grids. We can continue with the prior example to extract radial profiles for different event selection criteria.

```
r = hypot( dx, dy );                          % create a radial distance variable
(rlo, rhi ) = linear_grid( 0.0, 30.0, 60 ); % define a grid
a =  PI*(rhi^2-rlo^2)  ;                      % area of each annulus
rprof = histogram( r, rlo, rhi ) ;            % bin into a radial profile

% take a look, computing density "on-the-fly":
ylog;  yrange(0.1,1.e4);  xrange; connect_points(1);  % configure plot
label("r [pix]", "cts/area", "Source events profile (red=g7; green=g0)");
hplot(rlo, rhi, rprof/a );   % histogram plot, counts/area

% look at grade 7 only...  (this is a piled source)

g = fits_read_col(fevt1, "grade" );   % read some columns
lg7 = where (g[l] == 7) ;             % sub-select a list within the list...
r7 = histogram( r[lg7], rlo, rhi ) ; % ... and bin this selection

ohplot(rlo, rhi, r7/a, red );         % overplot histogram

% "on the fly" plot of histogram, for grade = 0...
ohplot(rlo, rhi,  histogram( r[where(g[l] == 0)], rlo, rhi ) / a, green  );
```

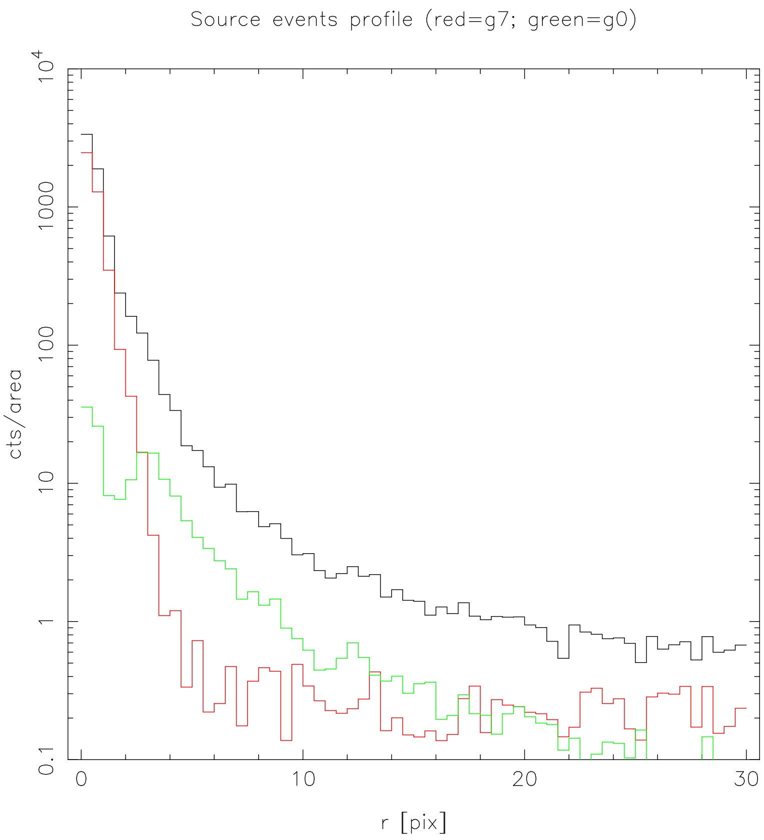

Figure 2: Radial profiles for source events. Top curve is all grades, status values; green is grade=0, and red is grade=7.

Comments

This example can also be done with the unix-level programs, on the file `src.fits` made above.

```
dmextract src.fits"[bin sky=annulus(4060.86,4116.83,0:30:0.5)]" srcprof.fits
dmextract src.fits"[filter grade=7][bin sky=annulus(4060.86,4116.83,0:30:0.5)]"\
 srcprof_7.fits
dmextract src.fits"[filter grade=0][bin sky=annulus(4060.86,4116.83,0:30:0.5)]"\
 srcprof_0.fits
```

Which method one chooses depends upon the nature of the analysis. The command-line versions have been designed with more options, such as input background, and to output more information, such as regions and net counts.

But suppose you want something slightly different, like a logarithmic radial grid. You could probably do this via `dmtcalc` computations to add the proper information to a table before `dmextract` (Exercise for the reader: maybe there is an option on the binning syntax). Alternatively, you could make slight modifications to the *S-Lang* procedure:

```
(lrlo,lrhi) = linear_grid(log10(0.1), log10(30), 60);
lrlo=10.^lrlo;   lrhi=10.^lrhi;
la =  PI*(lrhi^2-lrlo^2)  ;                   % area of each annulus
lrprof = histogram( r, lrlo, lrhi ) ;         % bin into a radial profile
```

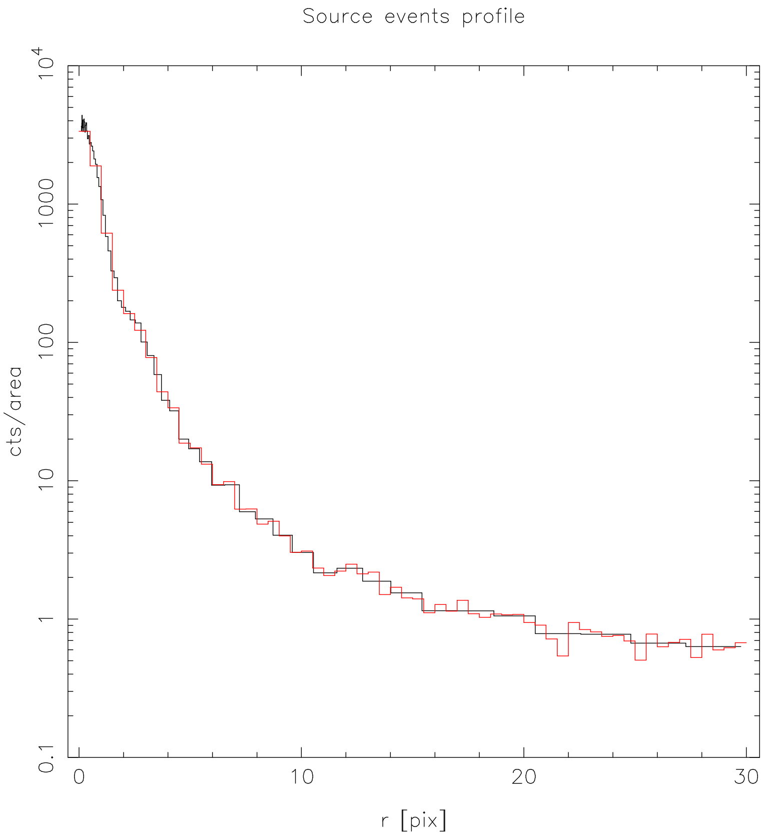

Figure 3: Radial profile for source events on a logarithmic radial grid (black) compared to the prior linearly gridded profile (red)

3 Extending the System

Suppose again: you want to do something related to *S-Lang* functions available, but which doesn't yet exist. For example, you might want a cumulative distribution.

Write a function! Here is one:

```
define cumdist()
{
  variable h = NULL;

  if (_NARGS != 1)
  {
    message("Usage: h_cum = cumdist( h ); ");
    message("   Compute the cumulative histogram given histogram, h.");
    return h ;
  }

  variable h_in = () ;
  variable i,  len = length( h_in );

  h = @h_in;

  for (i=1; i<len; i++)
    h[i] += h[i-1] ;

  return h ;
}
```

It is now a simple matter to compute the integrated radial profile from the previously defined variables:

```
evalfile("cumdist.sl");             % ``compile'' the source
crprof = cumdist( rprof );          % compute the cumulative distribution

% Look at it...
ylin; yrange(0); label("r[pix]", "Integrated counts/area", "");
hplot(rlo, rhi, crprof );
ohplot(rlo, rhi, cumdist( r7 ), red );

%%% now pick only good grades and status=0
good = where( (g[l] < 7) and (g[l] != 1) and (g[l] != 5) and (not s[l]) );
crgood = cumdist( histogram( r[good], rlo, rhi ) );
ohplot(rlo, rhi, crgood, green );
```

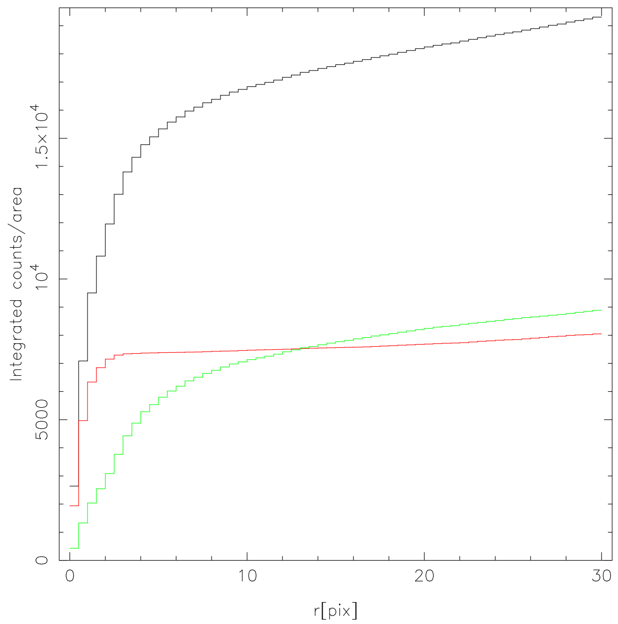

Figure 4: Integrated radial profile for all source events (black) compared to that for grade=7 (red). Filtering on good grades and status produces the green curve.

4 Regridding

Sometimes we wish to combine existing histograms, but which are on different grids. We could re-compute from the original data and specify commensurate grids. Or we can simply take existing histograms and rebin them (while accounting for density). This is an example of summing HEG and MEG spectra, after putting the HEG onto the same grid. (This could be done in fairly low-level *S-Lang*, but I use ISIS since it has much infrastructure for manipulating data sets).

```
fpha = "acisf01451N002_pha2.fits";        % a binned spectrum file
() = load_data(fpha, [3,4,9,10]);         % load only +-1st orders' rows

d = get_data_counts(1);                   % copy to variable
xhlo = d.bin_lo;                          % define convenient variables
xhhi = d.bin_hi;
yh = d.value + get_data_counts(2).value;  % sum +-1st HEG (indices 1,2)

d = get_data_counts(3);                   % repeat for MEG (indices 3,4)
xmlo = d.bin_lo;  xmhi = d.bin_hi;
ym = d.value + get_data_counts(4).value;

yy = ym + rebin( xmlo, xmhi, xhlo, xhhi, yh ); % rebin and add HEG to MEG counts

label("Wavelength", "Counts/bin", ""); yrange(0,60);  xrange(1.5,5.5);
multiplot([1,1]);resize(6*2.54, 11./8.5); erase;    % two panes, same size

shplot(xmlo,xmhi,yy, 0.01);           % smoothed plot, summed HEG+MEG, on MEG grid
oshplot(xmlo,xmhi, ym, 0.01, 2);      % MEG +-1st order sum
oshplot(xhlo,xhhi, yh, 0.01, 3);      % HEG +-1st order sum, on HEG grid.

hplot(xmlo, xmhi, yy, 4);             % unsmoothed summed HEG+MEG
oshplot(xmlo, xmhi, yy, 0.01, 2 );    % smoothed
```

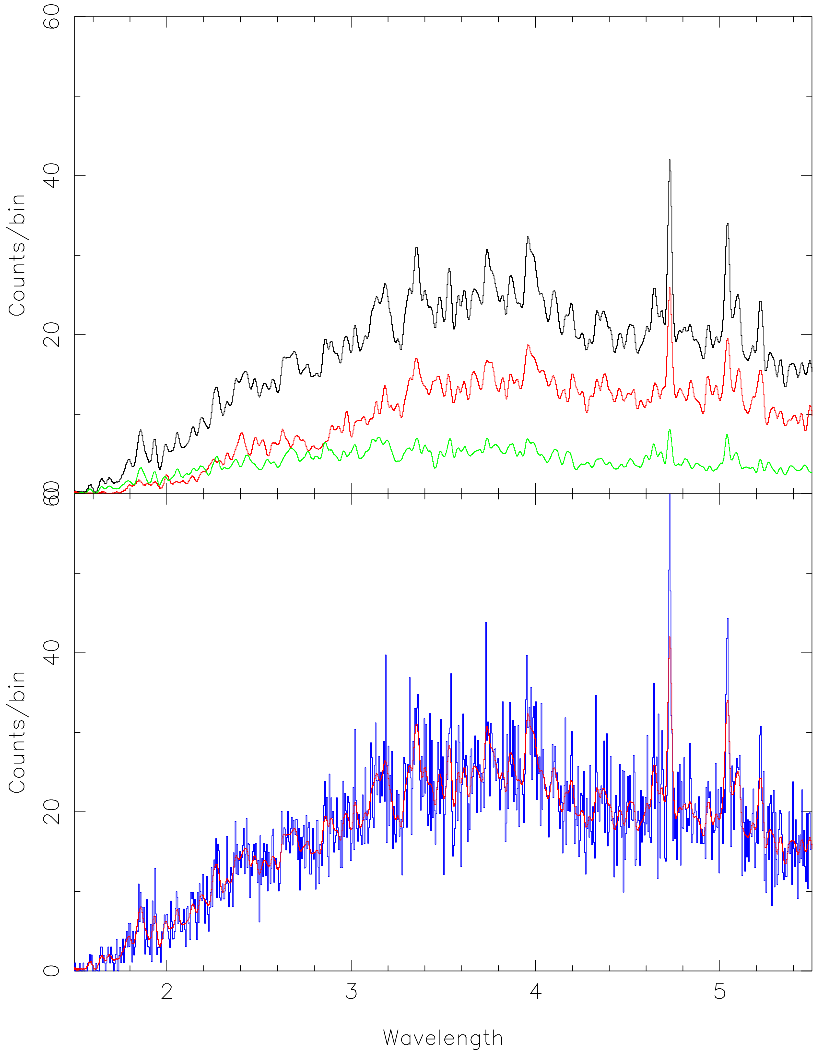

Figure 5: The top shows smoothed plots of an HEG spectrum, MEG, and the sum. To form the sum, the HEG was regridded to MEG resolution. The bottom shows the unsmoothed summed spectrum, with the smoothed version overplotted.

5 Smoothing

The smoothing of the previous spectra was accomplished by taking advantage of a low-level fast-fourier-transform function in ISIS and intrinsic complex arithmetic in *S-Lang*.

The ISIS function is `fft1d`. The user-supplied function is `gsmooth`, which was used to define new versions of `hplot, ohplot, plot_data_counts,` and oplot_data_counts.

While `gsmooth.sl` itself has overhead of applying a window function, forming the centered Gaussian convolution kernel, normalizing the result, the core use of the FFT and complex arithmetic can be seen in a few lines of *S-Lang*:

```
...
 (ry, iy) = fft1d(y, y*0., -1);            % forward fft of data
 (rk, ik) = fft1d(karray, karray*0., -1);  % forward fft of kernel

 cy = ry + iy * 1i;        % convert to Slang complex types.
 ck = rk + ik * 1i;
 c = cy * ck ;             % convolve: fft product.
...
 (rsy, isy) = fft1d(Real(c), Imag(c), 1);    % inverse fft;
...
```

The interface to `gsmooth` is:

```
isis> gsmooth;

USAGE: smoothed_y = gsmooth(x_lo, x_hi, y, sigma)

Apply gaussian smoothing filter of width sigma to histogram array y.
  x_lo, x_hi are histogram bin edge coordinate arrays.
  y is histogram values array (x_lo,x_hi,y must have same lengths
METHOD: Convolution is performed via FFT, after applying Hanning filter.
RESTRICTIONS: grid must be uniform.
```

The user-contributed splot_data.sl uses the ISIS plotting and data manipulation infrastructure to "wrap" `gsmooth()` around the intrinsic structures to smooth the data array before display.

6 Summary

I have attempted to give some illustrative examples of some basic *S-Lang* features. Some may seem trivial in terms of specific application, since there may be other more familiar or more terse options. However, the examples were meant to also lead to the impression that *S-Lang*-based approach provides for facile exploration and experimentation, prototype development, personal modifications, generalized extension of an analysis system, and possibly order of magnitude performance gain for repetitive tasks. In research, we always have *ad hoc* needs for which there is no explicit program to use. Conversely, if we find a general need, *S-Lang* implementations could become new general purpose tools, or lead to modifications of existing ones (such as a cumulative distribution option for `dmextract`, for example).

Useful low-level functions find their way into high-level applications. For example, I showed you how to regrid individual histograms. Some ISIS applications where this operation is done behind the scenes are:

`rebin_dataset` This function rebins a counts spectrum and its assigned instrument response matrix (RMF) so that the RMF maps onto the new instrument grid.

`group_data` The count data of each histogram is rebinned by summing the contents of the original input data bins and the associated bin uncertainties are recomputed assuming Poisson statistics.

`rebin_rmf` This function rebins an instrument response matrix (RMF) so that it maps onto a new instrument grid. The RMF normalization is preserved.

7 Appendix: The Isis Implementation of *S-Lang*

I have exclusively used the Isis implementation of the *S-Lang* interpreter and Isis analysis system for the examples here. It is not obvious to the novice which functions are *S-Lang* intrinsics, introduced by the interpreter (such as `print()`, or analysis functions defined as part of a package. While many of the features I have shown are available in ChiPS or sherpa, or are available by importing an Isis module, I prefer Isis because it is a *fully* S-Lang system: there is only one interpreter. In addition, Isis has many well tested and mature analysis functions, both of the generic type and higher-level for manipulation of n data sets and responses. (Ciao *S-Lang* development is ongoing; other presentations in these sessions will highlight Ciao *S-Lang*).

The Isis-specific functions I used in the examples are:

- All plotting functions.
- `hypot, linear_grid, histogram, rebin`
- `load_data, get_data_counts`

Of these, the "generic" Isis functions (those which can be used independently of data sets and responses) are

- `hypot, linear_grid, histogram, rebin`

The `fits_read_col` function is part of code used in common in both Ciao and Isis (i.e., it is not part of *S-Lang* itself).

The utility script, `~dph/libsl/mutils.sl`, can be used to import the Isis module into Ciao into a namespace and then redefine the desired functions into the current namespace. This way, they can be used as if part of Ciao without explicit reference to the namespace and without collision of duplicate function names. (The script contains a path to the module on the HEAD network.)

The Isis stand-alone program can be found on the HEAD network at `/home/ascds/houck/src/isis/bin/isis`.
Source and binaries are available from `http://space.mit.edu/CXC/ISIS/`.

8 Appendix: Exercises

1. Define the functions,

   ```
   is_equal( a, list )
   ```

   ```
   isnt_equal( a, list )
   ```

   Which allows the expressions like

   ```
   good = where( (g < 7) and (g != 1) and (g != 5) and (not s) );
   ```

   To be written more succinctly as:

   ```
   good = where( is_equal( g, [0,2,3,4,6] ) and  (not s) ) ;
   good = where( isnt_equal( g, [1,5,7] ) and  (not s) ) ;
   ```

2. Define the functions,

   ```
   between( x, lo, hi )
   ```

   which sets flags in an array where $lo < x <= hi$, and

   ```
   outside( x, lo, hi )
   ```

   which sets flags in an array where $x <= lo$ or $x > hi$.

   Points in a disjoint strip (think ACIS frame-shift streak, except for zero-order, in chip coordinates) can then be expressed as:

   ```
   where( between( x, x1, x2) and outside( y, y1, y2) )
   ```