Removing Background Flares

Michael Nowak, MIT-CXC, October 23, 2007

This memo is a follow-on to my memo of June 25, 2007. I am again going with the assumption that the background is well-characterized, so the question becomes one of whether or not removing a background flare improves the signal-to-noise for a source. I will be using the following definitions throughout. We presume that we have a total observation duration, T_o, over which time *in a given source region* the detector collects a baseline of B background counts and S source counts. During the observation, a flare of duration T_f induces an *additional* B_f counts in the source region. The fraction of time without a flare is defined as $\mathcal{F} \equiv (T_o - T_f)/T_o$. The ratio of the background rate during the flare to the baseline rate is given by $\mathcal{R} \equiv 1 + (B_f/T_f)/(B/T_o)$.

As discussed in the previous memo, removing the flare increases the source signal-to-noise for

$$\mathcal{R} \geq 1 + \mathcal{F}^{-1}(S/B + 2) \ . \tag{1}$$

Thus, we immediately find that unless $\mathcal{R} \geq 1 + \mathcal{F}^{-1} \geq 2$, removing the flare never improves the signal-to-noise. Realistically, our criterion for flare removal will be even more stringent, and we should only exclude flares with $\mathcal{R} \gtrsim 5$ (perhaps even > 10).

The main quandary that we have is where to place this threshold. Flare removal may improve the signal-to-noise for fainter, more extended[1] sources, while it may decrease the signal-to-noise for brighter, more point-like sources. So long as the background rate increases by greater than a factor of two, above a certain threshold of source size and integration time, flare removal will improve the source signal-to-noise. Taking a fiducial background, B_0, that is detected within a source region of radius r_0 over a time T_0 (i.e., so that $B = B_0(r/r_0)^2(T/T_0)$), then background flare removal improves the source signal-to-noise if:

$$\left(\frac{r}{r_0}\right)\left(\frac{T}{T_0}\right)^{1/2} > \left(\frac{S/B_0}{2[\mathcal{F}(R-1)-1]}\right)^{1/2} \ . \tag{2}$$

In Fig. 1, I plot the change in S/N that occurs when excising background flares of given durations (10%, 20%, or 50% of the lightcurve) for 10 count and 100 count sources of given radii (ranging essentially from point sources up to a 1 arcmin radius). As references, I assume that the flare increases the background rate either by a factor of 5 or 10. In general, we see that the S/N slightly decreases for point sources, and only increases for sources with radii $\gtrsim$ 0.1–0.3 arcmin. Even in these latter cases, the improvement of S/N is mild.

Another way of looking at this is to try and determine what amplitude flare must be removed to go from a given signal-to-noise in the presence of the flare (let us call this $\mathcal{SN}_f$) to a given signal-to-noise for the flare removed (let us call this $\mathcal{SN}_{nf}$). We can again look at this for a fiducial background rate, with the total background counts again scaling as radius squared multiplied by time. Fixing the desired no-flare signal-to-noise is fixing the relationship between extraction radius and source counts, for a given fraction of time without flares, $\mathcal{F}$. Choosing the signal-to-noise in the presence of the background flare then fixes the ratio of flare background rate to the baseline level. Specifically, we have:

$$S = \frac{\mathcal{SN}_{nf}^2}{2\mathcal{F}} + \frac{1}{2}\sqrt{\frac{\mathcal{SN}_{nf}^4}{\mathcal{F}^2} + \frac{8\mathcal{SN}_{nf}^2}{\mathcal{F}}} \ , \tag{3}$$

and

$$\mathcal{R} = \left[2B\left(1-\mathcal{F}\right)\right]^{-1}\left[S\left(\frac{\mathcal{SN}_{nf}^2}{\mathcal{SN}_f^2\mathcal{F}} - 1\right) + 2B\left(\frac{\mathcal{SN}_{nf}^2}{\mathcal{SN}_f^2\mathcal{F}} - \mathcal{F}\right)\right] \ . \tag{4}$$

[1] We only care about the extent of the source on the detector. For purposes of flare removal, it doesn't matter whether the source extent is due to being resolved on the sky, or is due to being viewed far off axis.

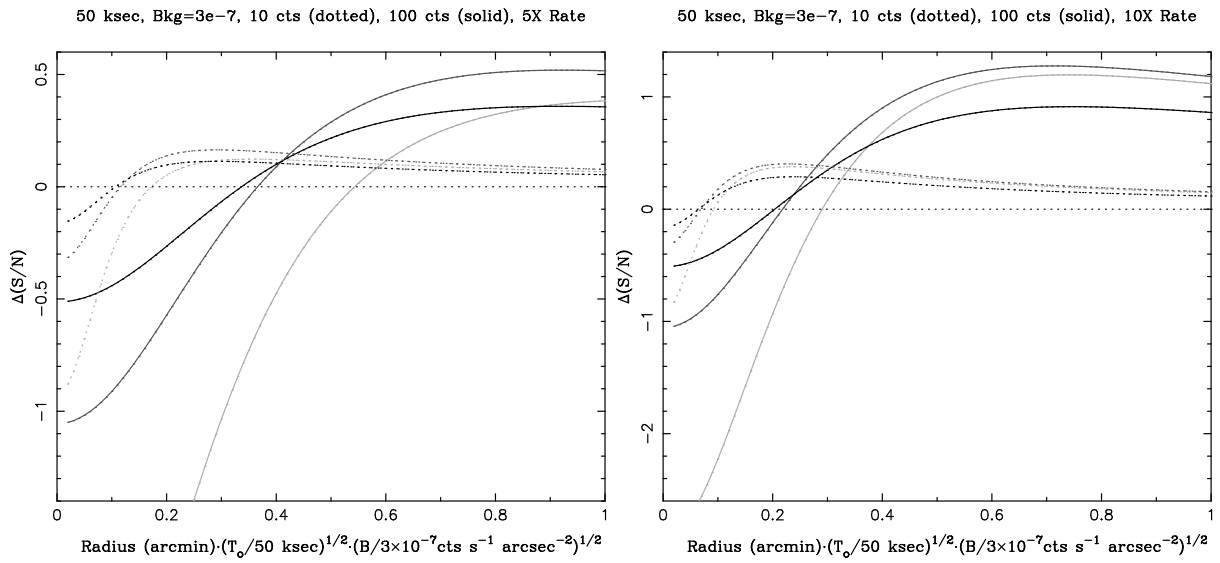

Figure 1: Change in S/N achieved vs. source size (assuming a given radius, a 50 ksec observation, and a baseline background rate of 3×10^{-7} cts/sq. pixel/sec, for excising a $5\times$ increase in background rate (left) or a $10\times$ increase in background rate (right). Black lines assume the flare covers 10% of the observation time, dark grey 20%, and light gray 50%. Dotted lines assume 10 counts in the source and solid lines assume 100 counts in the source.

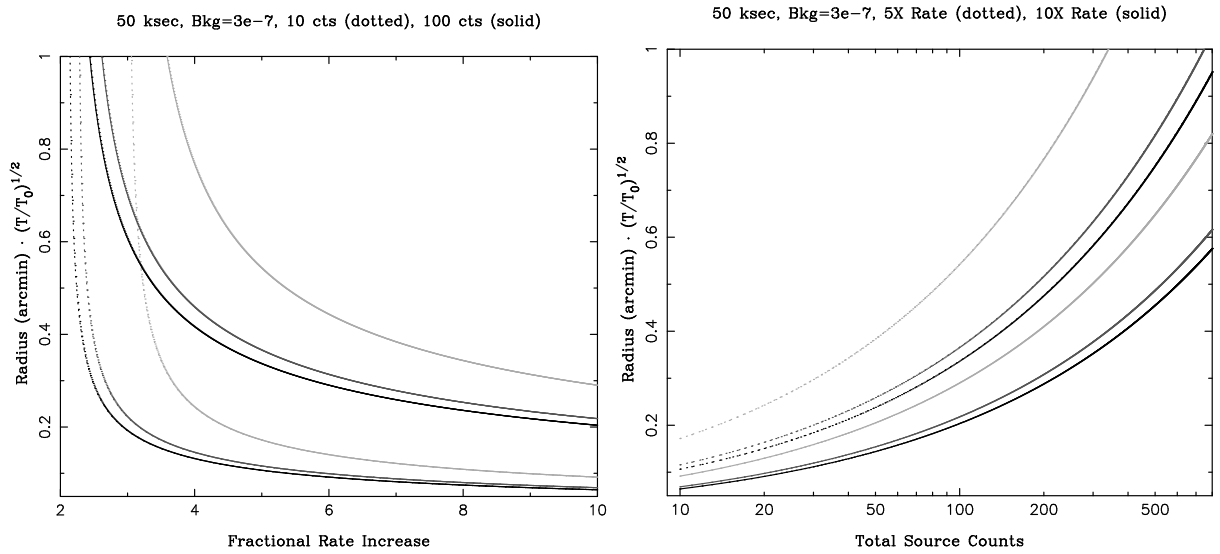

Figure 2: *Left:* The source radius above which removing the background flare increases the signal-to-noise vs. the background flare factor increase. Same assumptions about integration time and mean background rate as in Fig. 1, and same color scheme. *Right:* The source radius above which removing the background flares increases signal-to-noise vs. total source counts for background flares $5\times$ (dotted lines) and $10\times$ (solid lines) the baseline background level. Black lines assume the flare covers 10% of the observation time, dark grey 20%, and light gray 50%.

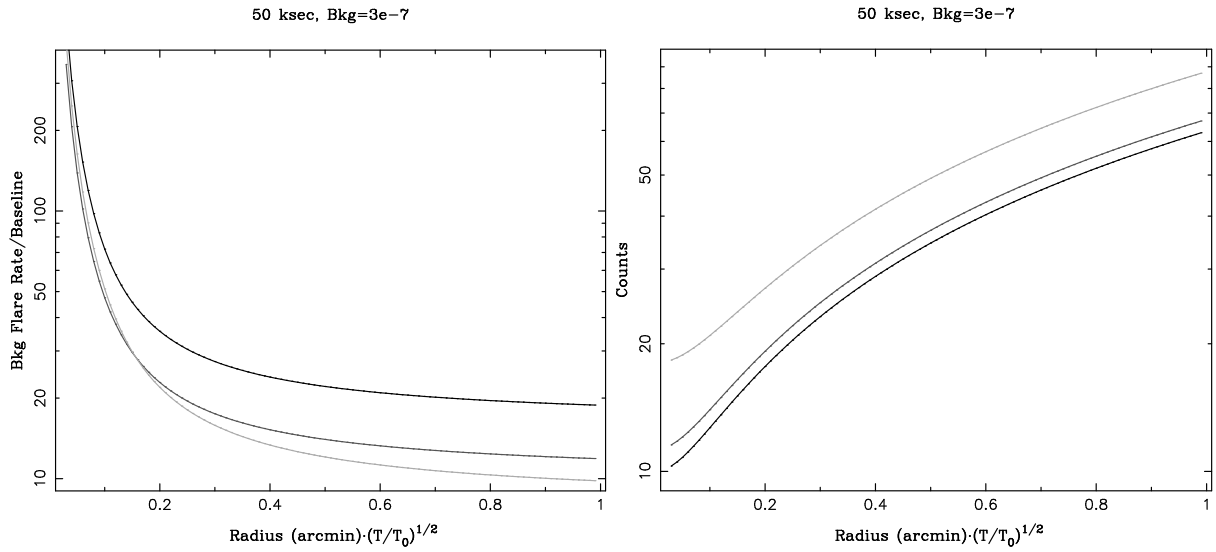

Figure 3: *Left:* Under the same assumptions of background rate and observation duration as before, the background rate factor increase for which removing the flare increases the signal-to-noise from 2 to 3. *Right:* The associated source counts for the figure on the left. Black lines assume the flare covers 10% of the observation time, dark grey 20%, and light gray 50%.

We see that in the limiting case of $B \gg S$ and $\mathcal{SN}_{nf} = \mathcal{SN}_f$ we again recover $\mathcal{R} = 1 + \mathcal{F}^{-1}$ (i.e., removing $\mathcal{R} \leq 2$ never leads to any improvement).

An interesting case arises when we ask what amplitude flare we need to remove to go from signal-to-noise of 2 to 3 (i.e., getting a source to meet something akin to the catalog criterion). Looking at Fig. 3, we see that for source radii $\lesssim 0.2$ arcmin, the required flare rate increase factor is enormously large. We see that flares really have to be greater than a factor of 10 or more.

Given the above caveats, there are cases where the background rate can increase by factors as large as 50 (namely ObsID 1712). Below are the specifications for background flare removal that I have tested. Note that the first steps follow the current concepts for removing point sources as part of creating the background map for wavdetect. It is probably reasonable to fold these procedures in with the background creation process. They could be run independently, however, as a precursor to the wavdetect background creation step.

1. Begin with the filtered events (i.e., bad pixels, afterglows, bad grades, etc., removed). Rotate the sky coordinates so that the readout streaks are aligned along one sky coordinate, and the chip readout edge is aligned along the other sky coordinate. Only consider sky coordinates that do not dither off of the chip edge. (I chose a simple cut of removing 30 pixels from both ends of the new sky x and y coordinates.)

2. For each chip, histogram the data to a one dimensional grid aligned with the chip readout edge. I chose 8 pixel bins along this direction.

3. For each chip, determine the median of the histogram values, and then the median plus one sigma (i.e., the square root of the median bin). Save the histogram reverse indices (i.e., the indices of the events in each histogram bin) for the events in bins that are less than the median plus one sigma.

4. For each chip, find the event times and energies associated with the above reverse indices. For events with energies ≥ 0.2 keV and≤ 7 keV (i.e., the full band used in the source detect algorithm), run the event times through the Gregory-Loredo algorithm and create an output lightcurve.

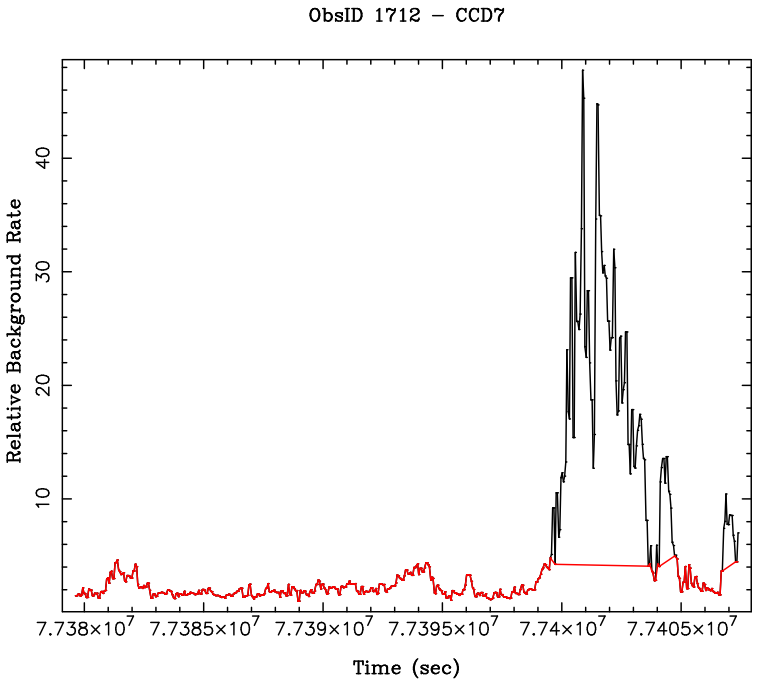

Figure 4: Results of the flare removal procedure for CCD 7 on ObsID 1712. The black line is the Gregory-Loredo lightcurve (presented as a ratio to the minimum value), while the red is the lightcurve that passes the filter threshold. For this case, the flare ratio threshold was 5, which is probably too low for catalog purposes.

• Note that I have tweaked the Gregory-Loredo defaults slightly to favor more subdivisions in the lightcurve, not fewer. (Specifically, we keep up to the maximum bin where the odds ratio is $\leq$ the peak odds divided by $\exp(1)$, rather than divided by $\exp(0.5)$. We can play with this level somewhat.) Furthermore, the output lightcurve is set to be evenly binned in time with 50 second bins – independent of how many subdivisions the Gregory-Loredo algorithm divides the lightcurve into. We can vary this 50 sec number a little, but the goal is to get an output lightcurve that when we later search for good time intervals based on its relative amplitude we don't create time intervals that are too coarsely defined.

5. For each chip, find the times when the ratio of the Gregory-Loredo lightcurve rate to its minimum value exceeds a given threshold, and exclude those times from further catalog analysis. Merge these new good time intervals with the existing GTI, and continue processing. (At this point is probably where one would begin to create the background maps for wavdetect.)

   • I am still debating what this threshold value should be. I would suggest we try runs on a set of ObsIDs with values of 10 and ∞ (i.e., don't run the algorithm) and see what differences result.

Below is my `S-lang` code for implementing the above procedures. A copy can be found at: `/pool14/mnowak/flar`
An example of the results are shown in Fig. 4. Note that the Gregory-Loredo code below is nearly identical to my previous version, with the exception that one bug has been fixed (one that only showed up for weakly varying sources) and that it's output lightcurve now has a fixed number of bins, regardless of what the variability algorithm finds.

```
%  Background Flare Removal
%  Mike Nowak, mnowak@space.mit.edu, October 21, 2007
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%

%%  Gregory-Loredo Part %%

%%%%%%%%%%%%%%%%%%%%%%%%%

require("gsl");

% The GSL lngamma function is pretty fast, but tabulating lngamma
% is slightly faster, especially for multiple runs

#ifnexists fst_lngamma
   private variable log_sum = Double_Type[320001];
   log_sum[[0,1]] = [0.,0.];

   variable i;
   for(i=2; i<=320000; i++)
   {
      log_sum[i] = log_sum[i-1] + log(i);
   }

   public define fst_lngamma(i)
   {
      return log_sum[int(i-1)];
   }
#endif

% t = event times; tmin & tmax =  max & min time of lightcurve,
% mmax = maximum partition number in trial lightcurves
% ta, adt = effective area/deadtime times and values
% dodither = Toggle for applying dither correction to lightcurve
% thresh = helps determine truncation of partitioned lightcurve
% mbin = number of bins in output lightcurve (differs from
%    previous version in that this is now a *fixed* number,
%    rather than a multiplier of the GL output lightcurve)

define log_odds(t,tmin,tmax,mmax,thresh,dodither,ta,adt,mbin)
{
   if(dodither !=0)
   {
      variable inz = where(adt>0);
      variable adt_use = [inz[0]:inz[length(inz)-1]];

      ta = ta[adt_use];
      adt = adt[adt_use];

      variable na = length(adt);
```

```
   if(tmin < ta[0]){ tmin = ta[0]; }
   if(tmax > ta[na-1]){ tmax = ta[na-1]; }
}

% Only look at times within tmin & tmax

t = t[where(t>=tmin and t<tmax)];

variable j;
variable n=length(t);
variable dt_int = tmax - tmin;
variable a_avg=0.;

if(dodither !=0)
{
   % Define the spline of the effective area curve

   variable spline_adt = interp_akima_init(ta,adt);
   a_avg = log(interp_eval_integ(spline_adt,tmin,tmax)/dt_int);

   % Do integration of spline in neighboring points, then sum

   variable iadt = Double_Type[na];
   _for j (1,na-1,1)
   {
      iadt[j] = interp_eval_integ(spline_adt,ta[j-1],ta[j]);
   }
   iadt=cumsum(iadt);

   % Define the spline of the integrated effective area curve

   variable spline_iadt = interp_akima_init(ta,iadt);
}

variable nj = Array_Type[int(mmax)-1];
variable aj = @nj;
variable m=[2:int(mmax):1];
variable lods=Double_Type[int(mmax)-1];

variable lo,hi,im;

_for im (2,int(mmax),1)
{
   % Create grid for partitioning lightcurve into im bins

    lo = [0:im-1];
    hi = [1:im];
    lo = __tmp(lo)*(dt_int/im)+tmin;
```

```
    hi = __tmp(hi)*(dt_int/im)+tmin;

   if(dodither != 0 )
   {
      % For each partitioning, create average deadtime/effective
      % area per bin.  Dead bins are set equal to the average
      % effective area, so as not to contribute to the sum

      aj[im-2] = (interp_eval(spline_iadt,hi)
                  -interp_eval(spline_iadt,lo))*(im/dt_int);

      aj[im-2][where(aj[im-2]==0.)] = exp(a_avg);
   }
   else
   {
      aj[im-2] = Double_Type[im]+1;
   }

   % For each lightcurve partition, the arrays of counts per bin

   nj[im-2] = histogram(t,lo,hi);

   % The odds ratio for each partitioning.  The nj[im-2]*()
   % term is removed if effective area variation is unimportant

   if(dodither != 0 )
   {
      lods[im-2] = sum( nj[im-2]*(a_avg-log(aj[im-2])) +
                        fst_lngamma(nj[im-2]+1) ) +
                   n*log(im) + fst_lngamma(im) - fst_lngamma(n+im);
   }
   else
   {
      lods[im-2] = sum( fst_lngamma(nj[im-2]+1) ) +
                   n*log(im) + fst_lngamma(im) - fst_lngamma(n+im);
   }
}

% This bit uses the return pieces from above to find what the
% maximum odds ratio is, and temporarily takes that out (lomax -
% a replacement for Arnold's bias parameter), and then like
% Arnold's code, truncates the number of lightcurve binnings kept

variable iw = where(lods==max(lods));
variable lomax = lods[min(iw)];

lods = exp(lods-lomax);
variable csum = cumsum(lods)/(m-1);
```

```
% Truncate the number of partitionings of the lightcurve

if(thresh < 0) thresh==0.;
variable imax = max(where(csum >= max(csum)/exp(thresh)));
iw = where(lods[[0:imax]]==max(lods[[0:imax]]));

% Return the probability (p), the odds ratio for each partioning
% (oratio), the log of the summed odds (lodds_sum), the # of
% partitions for the peak odds ratio (mpeak), the # of partitions
% for each (m), the histogrammed counts for each partitioning (nj),
% the integrated and averaged effective area for each partioning
%(aj, a_avg), and the used min/max times (tmin/tmax)

variable gl_struct=
  struct{p, oratio, lodds_sum, mpeak, mmax, m, nj,
         aj, a_avg, tmin, tmax, tlc, rate, erate};

% The # of partitions of the lightcurve with the highest odds ratio.

gl_struct.mpeak = min(iw)+2;

% Calculate the total probability of variability (p) and the odds
% ratio for each partitioning of the lightcurve (oratio).

variable msum = csum[imax];
variable lsum = log(msum) + lomax;
gl_struct.p = 1/(exp(-lsum)+1);
gl_struct.oratio = __tmp(lods)[[0:imax]]/
                         ((imax+1)*(msum+exp(-lomax)));

% The rest of the return values

gl_struct.lodds_sum = lsum;
gl_struct.mmax = imax+2;
gl_struct.m = __tmp(m);
gl_struct.nj = __tmp(nj);
gl_struct.aj = __tmp(aj);
gl_struct.a_avg = exp(a_avg);
gl_struct.tmin = tmin;
gl_struct.tmax = tmax;

% This is the bit that differs slightly from the previous
% version.  It has a fixed number of output lightcurve
% bins, rather than a multiplier X mmax.

mbin = int(mbin);
if(mbin !=0)
```

```
   {
      variable tfrac = [0:mbin-1]/(mbin*1.);
      variable rate = Double_Type[mbin];
      variable erate=@rate;

      % We might not have used all the data ...

      variable ntot=sum(gl_struct.nj[0]);

      variable nj_ii_k, oratio_ii, aj_ii_k, drate;

      % Best estimate of the lightcurve is calculated here.
      % Fractional area/deadtime correction is included.

      variable ii;
      _for ii (0,imax,1)
      {
         variable mloop=ii+2;
         variable k = int( tfrac*mloop );

         nj_ii_k=gl_struct.nj[ii][k];
         oratio_ii=gl_struct.oratio[ii];
         aj_ii_k = gl_struct.aj[ii][k];
         drate = (mloop/(ntot+mloop))*
                 (__tmp(oratio_ii)*(nj_ii_k+1)/aj_ii_k);
         rate = __tmp(rate) + drate;
         erate = __tmp(erate) + (mloop/(ntot+mloop+1))*
                 (__tmp(drate)*(__tmp(nj_ii_k)+2)/__tmp(aj_ii_k));
      }

      % Here I differ from Gregory&Loredo and Arnold. G-L only include
      % the variable part of the lightcurve (i.e., partitionings with
      % >=2 bins), reasonable for p~1. L3 goes down to p~0.6, therefore
      % the estimated constant lightcurve part should be included.

      rate = __tmp(rate) + (1.-gl_struct.p)/gl_struct.a_avg;
      erate = __tmp(erate) + (1.-gl_struct.p)/gl_struct.a_avg^2;
      erate = sqrt(__tmp(erate)-rate^2);

      gl_struct.tlc = __tmp(tfrac)*(tmax-tmin)+tmin;
      gl_struct.rate =
         __tmp(rate)*(ntot/(tmax-tmin));
      gl_struct.erate =
         __tmp(erate)*(ntot/(tmax-tmin));
   }

   return gl_struct;
}
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%  Background Flare Part %%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%

variable t,ccd,nd,cx,en, sx, sy;

define rotate(x,y,t)
{
   variable x1, y1;
   x1 = cos(t)*x - sin(t)*y;
   y1 = sin(t)*x + cos(t)*y;
   return (x1,y1);
}


% Read in sky coordinates (& roll angle, so we can rotate to
% be along the readout direction), ccd ID, times, & energy.
% We will be doing GTI separate for each CCD, and be doing
% and energy cut.

variable file = "../1712/primary/acisf01712N003_evt2.fits.gz";

%variable file = "../869/primary/acisf00869N003_evt2.fits.gz";

%variable file = "../635/primary/acisf00635N003_evt2.fits.gz";

(t,ccd,en,sx,sy) =
   fits_read_col(file,
                 "TIME","CCD_ID","ENERGY","X","Y");

variable theta = fits_read_key(file,"roll_nom");

variable ccd_on = Integer_Type[0];
variable iccd = Array_Type[10];
variable t_ccd = Array_Type[10];
variable en_ccd = Array_Type[10];
variable sx_ccd = Array_Type[10];
variable sy_ccd = Array_Type[10];
variable rx_ccd = Array_Type[10];
variable ry_ccd = Array_Type[10];
variable nx_ccd = Array_Type[10];
variable mn = Double_Type[10];
variable mm = Double_Type[10];
variable md = Double_Type[10];
variable rev_ccd;
```

```
variable nx_filt;
variable sig_crit=1.;
variable gl = Struct_Type[10];
variable ecut, elow=200.,ehigh=7000.;
variable tbin=50.;
variable gti_start_ccd = Array_Type[10];
variable gti_stop_ccd = Array_Type[10];
variable ngti_start_ccd = Array_Type[10];
variable ngti_stop_ccd = Array_Type[10];
variable rthresh=5.;
variable iw = Array_Type[10];

% Find out which CCD's are on, and get a baseline GTI

variable i;
foreach i ([0:9])
{
   iccd[i] = where(ccd==i);
   if(length(iccd[i])>0)
   {
      ccd_on = [ccd_on,i];
      (gti_start_ccd[i],gti_stop_ccd[i]) =
         fits_read_col(file+"[GTI"+string(i)+"]","START","STOP");
   }
}

variable ic,iy,in;

% Only keep x-coordinates that don't dither off the chips,
% and bin by a factor of 8

variable lox = [30:1014:8];
variable hix = lox+8;

foreach ic (ccd_on)
{
   sx_ccd[ic] = sx[iccd[ic]];
   sy_ccd[ic] = sy[iccd[ic]];

   % Rotate the sky coordinates to be along the readout

   if(ic>3)
   {
      (rx_ccd[ic],ry_ccd[ic]) =
         rotate(sx_ccd[ic],sy_ccd[ic],theta*PI/180);
      rx_ccd[ic] = rx_ccd[ic]- min(rx_ccd[ic]);
      ry_ccd[ic] = ry_ccd[ic]- min(ry_ccd[ic]);
   }
```

```
else if(ic == 0 or ic ==2)
{
   (rx_ccd[ic],ry_ccd[ic]) =
      rotate(sx_ccd[ic],sy_ccd[ic],theta*PI/180+PI/2);
   rx_ccd[ic] = rx_ccd[ic]- min(rx_ccd[ic]);
   ry_ccd[ic] = ry_ccd[ic]- min(ry_ccd[ic]);
}
else if(ic == 1 or ic==3)
{
   (rx_ccd[ic],ry_ccd[ic]) =
      rotate(sx_ccd[ic],sy_ccd[ic],theta*PI/180-PI/2);
   rx_ccd[ic] = rx_ccd[ic]- min(rx_ccd[ic]);
   ry_ccd[ic] = ry_ccd[ic]- min(ry_ccd[ic]);
}

% Throw out y-coordinates that dither off the chips

iy = where(ry_ccd[ic] >=30 and ry_ccd[ic] <= max(ry_ccd[ic])-30);

t_ccd[ic] = t[iccd[ic]][iy];
en_ccd[ic] = en[iccd[ic]][iy];
sx_ccd[ic] = sx_ccd[ic][iy];
sy_ccd[ic] = sy_ccd[ic][iy];
rx_ccd[ic] = rx_ccd[ic][iy];
ry_ccd[ic] = ry_ccd[ic][iy];

% Histogram perpendicular to the readout direction, and
% save the reverse indices of what went into the bins.

nx_ccd[ic] = histogram(rx_ccd[ic],lox,hix,&rev_ccd);

% At first I wasn't sure whether we'd be basing the cut
% on the minimum, the mean, or the median.  Median seems
% to be the best, but we can change that.

mm[ic] = min(nx_ccd[ic]);
mn[ic] = mean(nx_ccd[ic]);
md[ic] = median(nx_ccd[ic]);

% Only keep columns that are within sig_crit of the median.
% Since we aren't using this for background modeling in
% wavdetect, and are only going to go with GTI times, we
% can be a little more fussy with this criterion and
% (for now) throw out more data by taking sig_crit = 1 sigma

nx_filt = where(nx_ccd[ic] < md[ic]+sig_crit*sqrt(md[ic]));

% These are all the "source free" events.
```

```
variable bkg_inx = Long_Type[0];
foreach in (nx_filt)
{
   bkg_inx = [bkg_inx,rev_ccd[in]];
}

% Get filtered times, energies.  Coordinates are just
% for checking purposes/

t_ccd[ic] = t_ccd[ic][bkg_inx];
en_ccd[ic] = en_ccd[ic][bkg_inx];
sx_ccd[ic] = sx_ccd[ic][bkg_inx];
sy_ccd[ic] = sy_ccd[ic][bkg_inx];
rx_ccd[ic] = rx_ccd[ic][bkg_inx];
ry_ccd[ic] = ry_ccd[ic][bkg_inx];

% For the Gregory-Loredo lightcurve, put in the GTI
% limits as the bounds of the lightcurve

variable min_t = min(gti_start_ccd[ic]);
variable max_t = max(gti_stop_ccd[ic]);

% We only want to worry about flares that affect our
% science, so I am choosing the 0.2-7 keV detection
% band.  In principle, we could do this for each energy
% band of interest, and then take intersections.  I don't
% want to get that fussy at this point.

ecut = where(en_ccd[ic] >= elow and en_ccd[ic] <= ehigh);

% Calculate a Gregory-Loredo lightcurve with bins of length
% tbin.  Note also the 5th input parameter is 1 (rather than
% the usual 0.5 default A. Rots has been using) - it's to get
% a slightly more finely varying lightcuve (i.e., less blocky).
% That's a parameter to play with some more for these purposes.

% If there are no gaps in the GTI, do the faster Gregory-Loredo
% without an effective area correction.

if(length(gti_start_ccd[ic]) < 2)
{
   gl[ic] = log_odds(t_ccd[ic][ecut],min_t,max_t,
            (max_t-min_t)/tbin,1,0,,,(max_t-min_t)/tbin);
}
else
{
```

```
   % If there are gaps in the GTI, put that into a
   % normalized effective area curve, and feed that
   % to the Gregory-Loredo algorithm

   variable tadd, tadt = Double_Type[0];
   variable adt = Double_Type[0];
   i=0;
   foreach i ([0:length(gti_start_ccd[ic])-2])
   {
      tadd=[gti_start_ccd[ic][i]:
            gti_stop_ccd[ic][i]-0.1:1.];
      tadt = [tadt,tadd,gti_stop_ccd[ic][i],
                        gti_start_ccd[ic][i+1]-0.1];
      adt = [adt,ones(length(tadd)),0.,0.];
   }
   i++;
   tadd=[gti_start_ccd[ic][i]:gti_stop_ccd[ic][i]:1.];
   tadt = [tadt,tadd];
   adt = [adt,ones(length(tadd))];

   gl[ic] = log_odds(t_ccd[ic][ecut],min_t,max_t,
            (max_t-min_t)/tbin,1,1,tadt,adt,(max_t-min_t)/tbin);
}

% Flag times that exceed the threshold, rthresh.  This number
% *must* be greater than 2, and probably should be placed in the
% 4-5 range.  That's something that we should probably play
% with during the testing and characterization stage.

variable igti = where(gl[ic].rate/min(gl[ic].rate) > rthresh);
variable ligti = length(igti);


% Create the GTI's for the times that we are keeping.

if(ligti > 0)
{
   variable gdiff = where( igti - shift(igti,-1) > 1 or
                           igti - shift(igti,-1) <= 0 );
   variable lgdiff = length(gdiff);

   ngti_start_ccd[ic] = [gti_start_ccd[ic][0]];
   ngti_stop_ccd[ic] = [gl[ic].tlc[igti[gdiff[0]]]];

   i=1;
   loop(lgdiff-1)
   {
      ngti_start_ccd[ic] = [ngti_start_ccd[ic],
```

```
                              gl[ic].tlc[igti[gdiff[i]-1]+1]];
         ngti_stop_ccd[ic] = [ngti_stop_ccd[ic],
                              gl[ic].tlc[igti[gdiff[i]]]];
         i++;
      }

      if( igti[ligti-1] < length(gl[ic].tlc)-1 )
      {
         ngti_start_ccd[ic] = [ngti_start_ccd[ic],
                               gl[ic].tlc[igti[ligti-1]+1]];
         ngti_stop_ccd[ic] = [ngti_stop_ccd[ic],
                              max(gti_stop_ccd[ic])];
      }
   }
   else
   {
      ngti_start_ccd[ic] = [min(gti_start_ccd[ic])];
      ngti_stop_ccd[ic] = [min(gti_stop_ccd[ic])];
   }

   %  These are the indices that we've kept - for testing purposes.

   iw[ic] = Long_Type[0];
   foreach i ([0:length(ngti_stop_ccd[ic])-1])
   {
      iw[ic] = [iw[ic],where(gl[ic].tlc >= ngti_start_ccd[ic][i] and
                             gl[ic].tlc < ngti_stop_ccd[ic][i])];
   }
}
```