Preliminary Assessment of Dither Corrections for L3 Lightcurves

Michael Nowak, MIT-CXC, August 31, 2007

The purpose of this document is to describe the ability of the Gregory-Loredo algorithm, in conjunction with the `dither_region` tool, to properly characterize lightcurve variability in the presence of dither across chip boundaries, bad pixels, etc. I will briefly touch upon backgrounds in this memo, but that will not be the focus. Additionally, I have not yet tested a working version of the `glvary` tool, but instead am running these tests via a `S-lang` script (attached) that implements my interpretation of the Gregory-Loredo algorithm.

- *Caveat 1:* The following tests were not run with the `glvary` code, but rather, they were run with a `S-lang` script. For no effective area/deadtime corrections, the algorithm should be identical to that used by `glvary`. The `S-lang` algorithm has been run on lightcurve files from ObsID 635 (provided to me by Arnold Rots, approximately two years ago), and yields the same list of variable sources, with very similar lightcurves, as to those found in Arnold's memo. Run times are comparable to those from `glvary` (I obtain < 0.5 sec per source; Arnold obtained ≈ 1 sec per source, albeit on a machine that undoubtedly was slower.)

The small differences in the lightcurves come predominantly from the fact that I have used a different sampling/binning for the final output lightcurve as compared to the default `glvary` output. This choice is not a fundamental one, and I note that the `glvary` tool allows different possibilities via an input parameter.

There is one fundamental difference between the lightcurves that I generate and the Gregory-Loredo paper (and I presume the `glvary` implementation– I haven't read the code quite carefully enough to be sure). Gregory & Loredo suggest that the "best" lightcurve be a weighted sum of the generated lightcurves with $m = 2$–m_{max} partitions. I instead create the weighted sum from lightcurves with $m = 1$–m_{max} partitions. For significantly variable lightcurves, this makes very little difference. Given, however, that we could include lightcurves from sources with fairly low Gregory-Loredo probabilities, and that we will likely calculate Gregory-Loredo lightcurves for background regions (whether or not they are independently variable), this change could make a difference down the line.

Other differences in my code arise in the way I handle the effective area/deadtime correction. First, I think there are some (non-fatal) flaws in the logic of the Gregory-Loredo paper, and second, I rather dislike the Gregory-Loredo description of their logic. (In their attempt to show how similar the algorithm is with effective area changes, they obfuscate the essential differences, burying, I think, the aforementioned logic flaw both to themselves and their readers. They also make the unfortunate choice of keeping all the variable names identical to the prior case, while fundamentally altering the definition of some, but not others.) Here, I outline some of my differences with the Gregory-Loredo derivation.

Given a lightcurve broken up into l narrow (i.e., ≤ 1 event per bin) time intervals of uniform width Δt, rate, r, and some prior information, I, with observed data, D, the probability of obtaining this data (for Poisson statistics) is given by:

$$P(D|r,I) = \Delta t^N \left(\prod_{i=1}^{N} r_i a_i \right) \exp\left(-\Delta t \sum_{k=1}^{l} r_k a_k \right) \ , \tag{1}$$

where a_i is a normalized effective area/deadtime correction in each bin. (Gregory & Loredo use different notation for that correction factor.)

Dividing the lightcurve into m bins (indexed by j), and taking a model of the rate being piecewise constant in each bin, we can define

$$r_j \equiv mAf_j \ , \ \text{ where } \ \sum_{j=1}^{m} f_j = 1 \ . \tag{2}$$

A is a mean rate over total elapsed time (gaps and all) T. We can further define n_j to be the number of events in bin j. (Gregory & Loredo for the effective area later redefine the f_j, yet retain the same symbol, while

ostensibly leaving all the other variables unchanged,. They implicitly redefine T, however, as I discuss further below.) We can then write the above probability as

$$P(D|r,I) = \Delta t^N (mA)^N \left(\prod_{i=1}^{N} a_i\right) \left(\prod_{j=1}^{m} f_j^{n_j}\right) \exp\left(-AT\sum_{j=1}^{m} f_j \langle a\rangle_j\right) \ , \tag{3}$$

where $\langle a\rangle_j$ is the average of the effective area/deadtime correction in bin j. The $\prod a_i$ term is missing from Gregory & Loredo, but this is a trivial error in that it is a term common to all calculated probabilities, and hence drops out in all of the odds ratios.

To perform the marginalization of the probability over rate in a similar manner to the case of no effective area/deadtime corrections, we now define:

$$AT\sum_{j=1}^{m} f_j \langle a\rangle_j \equiv AT_m \ , \tag{4}$$

which implies that

$$\sum_{j=1}^{m} f_j \langle a\rangle_j \frac{T}{T_m} \equiv \sum_{j=1}^{m} h_j \equiv 1 \ , \tag{5}$$

with

$$T_1 = \int dt\ a(t) \ , \ f_j = \frac{T_m}{T}\frac{h_j}{\langle a\rangle_j} \ . \tag{6}$$

In the above is where all the issues with the Gregory & Loredo derivation occur. First, in eq. (4), they treat it as an equality, rather than a definition, which it properly is. It cannot be shown from their given assumptions; it is an implicit definition that they make. Second, they implicitly take $T_m = T_1$ for all m (i.e., the aforementioned redefinition of T), which is not strictly true. By taking all the T_m to be the same, they drop out in the subsequent marginalized odds ratios. In general, $T_m \approx T_1$ for all m is a good approximation, *if* there is no correlation of the source variability with the variability of $a(t)$, the instrumental effective area/deadtime changes. In general, this is very, very likely to be true, which is why I refer to this problem as a "non-fatal" flaw. However, it is a caveat for the catalog.

- *Caveat 2:* For sources that have variability that fortuitously correlates with the instrumental time scales for effective area/deadtime changes (typically the dither time scales), the corrections employed in the Gregory & Loredo algorithm likely will fail.

I would label this as a catalog characterization issue, but not something for which we have to find a specific remedy.

Taking the new fundamental rate variables over which to marginalize to be the h_j (which in Gregory & Loredo are the redefined f_j), we now have

$$P(D|r,I) \approx \Delta t^N \left(\frac{mAT_1}{T}\right)^n \left(\prod_{i=1}^{N} a_i\right) \left(\prod_{j=1}^{m} \langle a\rangle_j^{-n_j}\right) \left(\prod_{j=1}^{n_j} {h_j}^{n_j}\right) \exp\left(-AT_1\right) \ . \tag{7}$$

The rest then proceeds following the derivation of Gregory & Loredo. Specifically, we get for the logairthm of the odds ratio

$$\log(O_{m1}) = \sum_{j=1}^{m}\left[n_j \log\left(\frac{\langle a\rangle}{\langle a\rangle_j}\right) + \log(n_j!)\right] + N\log(m) + \log[(m-1)!] - \log[(N+m-1)!] - \log(m_{max}-1) \ , \tag{8}$$

and for the rate and rate variance:

$$\left\langle \frac{r(t)}{AT_1/T} \right\rangle = \sum_{m=1}^{m_{max}} m\ \frac{O_{m1}}{\sum_{k=1}^{m_{max}} O_{k1}}\ \frac{n_{j'}+1}{(N+m)\langle a\rangle_{j'}} \ , \tag{9}$$

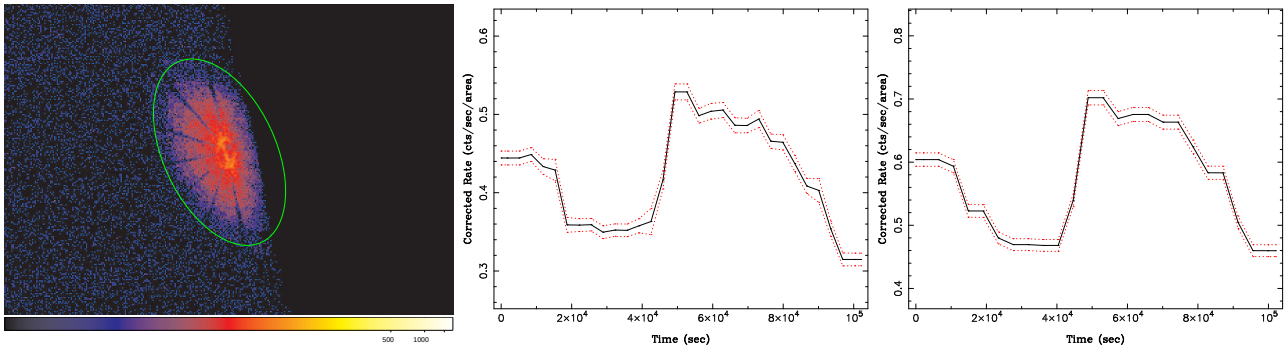

Figure 1: Source I from ObsID 635 and its extraction region (left). Lightcurves show the output of the Gregory-Loredo algorithm with no dither correction (middle) and with dither correction (right).

$$\left\langle \left(\frac{r(t)}{AT_1/T} \right)^2 \right\rangle = \sum_{m=1}^{m_{max}} m^2 \, \frac{O_{m1}}{\sum_{k=1}^{m_{max}} O_{k1}} \, \frac{(n_{j'}+2)(n_{j'}+1)}{(N+m+1)(N+m)\langle a \rangle_{j'}^2} \quad , \tag{10}$$

where $j'(t,m)$ is the index for the bin that contains time t in the lightcurve with m partitions, and the O_{m1} are the odds ratios for a lightcurve with m partitions relative to a single partition. Again, I take the sum starting at $m = 1$, since I believe that is proper in all situations.

Fundamentally, I don't think this differs too much from the implementation of effective area correction in the `glvary` tool. Essentially, the needed extra pieces are the $\langle a \rangle_j$ terms for each bin within each lightcurve partitioning. In principle, this factor should include both effective area changes as well as deadtime corrections. In what follows, I have not included deadtime corrections, and only include area variations as calculated by the `dither_region` tool.

The way I have chosen to implement the above is as follows:

1. For a source extraction region, obtain $a(t)$ using the `dither_region` tool. *My estimate is that on my relatively fast machine, this will take 10–30 sec per source.* This is the rate limiting step. I'll have better estimates of the requied time, once `dither_region` makes it into CIAOX locally at MIT

2. The $\langle a \rangle_j$ terms are integrals of $a(t)$. I have implemented this by first creating a spline of $a(t)$, saving the spline information, and then for each time bin within each partition calculating the integral. *This is the second most significant rate limiting step, which I estimate to add about 5 sec per source.*

   Proper integrals via splines are chosen since the $a(t)$ curves are defined on non-uniformly sampled grids. I found that simple grid interpolations were less numerically well-behaved, and not much faster.

   The splines were implemented via the `GNU Scientific Library S-lang` module. GSL was fast and very accurately reproduced the `dither_region` curve for many different binnings.

   The main reason that this is an approximately constant time hit on the algorithm is that, as opposed to histogramming the data where the histogram run-time scales with the number of events, $a(t)$ is defined predominantly by the length of the observation, regardless of whether there is data in that time bin or not. We cannot throw out times without data, so the spline integral needs to be calculated $\approx 2^6$ times no matter what. There are probably some ways of being a little more clever here; however, the big hit is still application of `dither_region` (which seems reasonably fast, given the job that it is performing).

3. Include the effective area changes in the calculation of the odds ratio and the optimal lightcurve.

To test out how well the algorithm works on real data, I used a number of sources that fall into dithered regions from ObsID 635. This is the same dataset for which Arnold previously generated 118 ASCII lightcurve files (and for which the bare algorithm, in both `glvary` and the `S-lang` script, works well). It is also one of

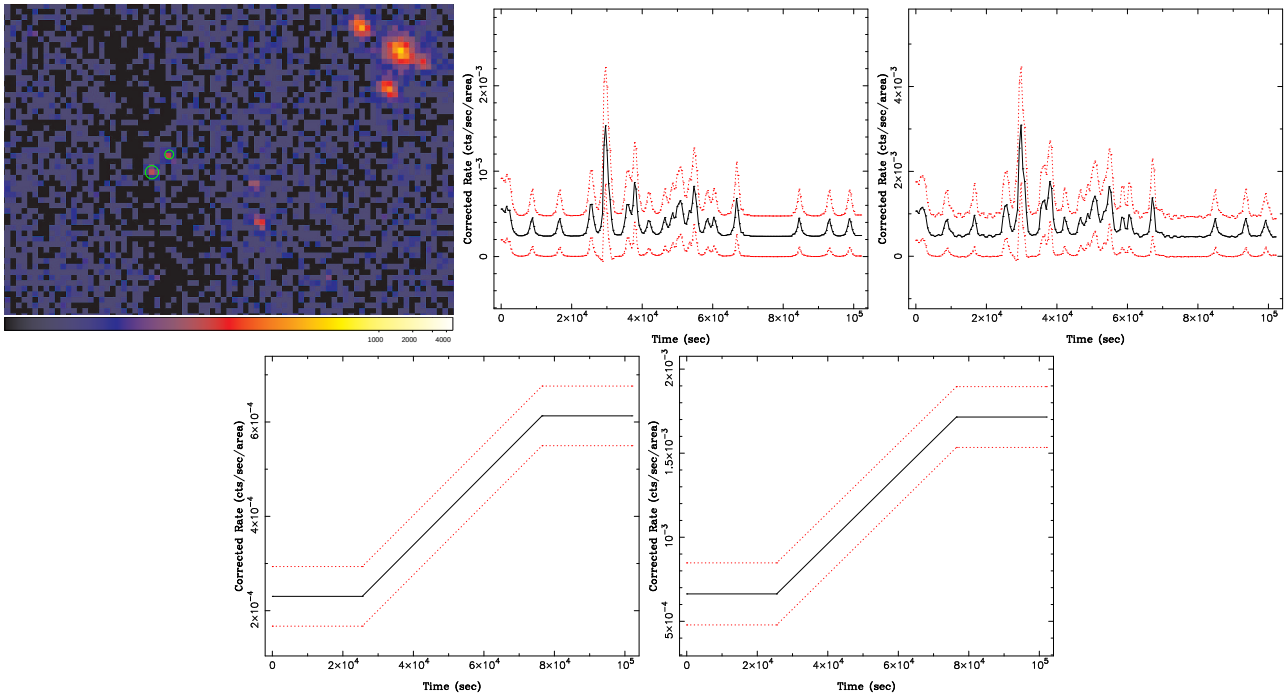

Figure 2: Source II & IV from ObsID 635 and their extraction regions (top left - source II is the one on the right). Lightcurves show the output of the Gregory-Loredo algorithm with no dither correction (top middle, source II; bottom left, source IV) and with dither correction (top right, source II; bottom right, source IV).

the datasets that John Houck has been using for source extent measurements. For the dither correction tests, I concentrate on five sources (whose numbers are *not* the same as used in Arnold's memo on the Gregory-Loredo algorithm, although all these sources were included in those prior tests).

Source I, shown in Fig. 1, is bright, fairly far off axis and hence very spatially extended. It has in the range of 25–35% of its area dithered on and off the chip. Given the fairly mild $\pm 7\%$ (relative) changes in area due to dither, especially compared to the long-term, large scale changes, the dither correction does not fundamentally alter the optimal lightcurve. One sees, however, that dither correction smooths out some of the short time scale structure in the lightcurve, and I would judge this an improvement.

Sources II and IV, shown in Fig. 2, are faint, and closer on axis. Neither is strongly dominated by dither, however. Again, we see that dither correction tends to smooth out slightly the lightcurve of Source II. However, for that curve, the spikiness is really dominated by a brief flare at 30,000 seconds. This is one of the artifacts of the Gregory & Loredo method. Since the lightcurve is sampled via successive, even binnings, in order to capture a significant, narrow flare, the very finely sampled lightcurve is considered to be overall significant and strongly contributes to the final lightcurve. This is true even for regions outside of the flare, and hence regions near isolated events show up with spikes as narrow as the main flare.

Sources III & V, shown in Figs. 3 and 4, are both reasonably bright sources, one is fairly close to being on axis, and both are strongly affected by dither. Here we see that the dither correction makes dramatic improvements. It essentially removes all of the variability solely due to dither, and all that remains is the real, longer time scale behavior. Both of these sources argue that `dither_region` applied via the Gregory & Loredo algorithm can be used to characterize the effects of dither on variability measurements.

The one concern that needs to be addressed is whether or not this correction is worth the extra time in processing (let us estimate about 30 seconds for the time being)? As the aspect histogram is otherwise being calculated for spectral purposes, one could first check whether or not the source region crosses a chip boundary. If not, then perhaps the dither correction could be skipped. (Although bad pixels still might be an issue.) In general, we could consider setting a binary flag: one bit for an aspect histogram that crosses a node,

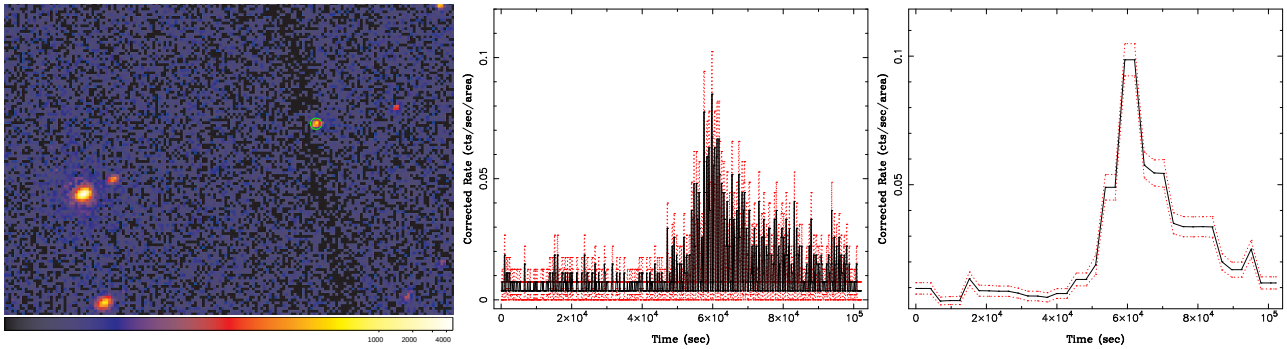

Figure 3: Source III from ObsID 635 and its extraction region (left). Lightcurves show the output of the Gregory-Loredo algorithm with no dither correction (middle) and with dither correction (right).

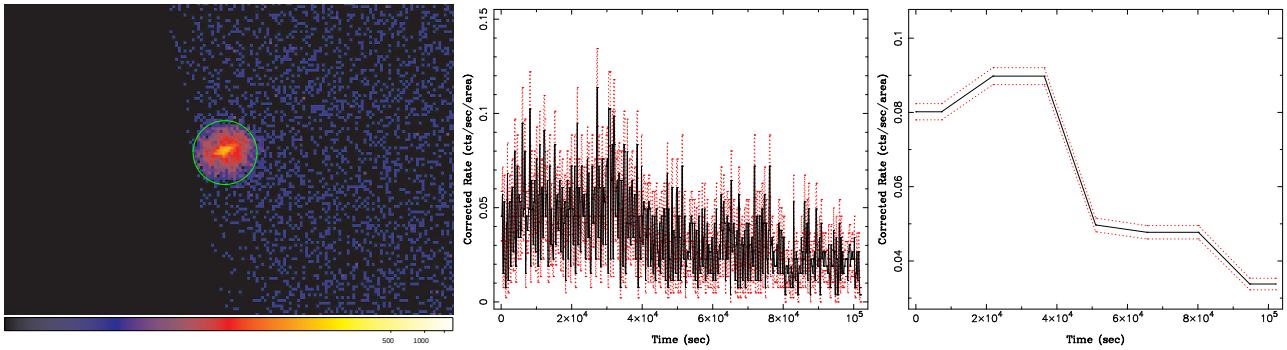

Figure 4: Source V from ObsID 635 and its extraction region (left). Lightcurves show the output of the Gregory-Loredo algorithm with no dither correction (middle) and with dither correction (right).

another for crossing a chip edge, another for being across 2 or more chips.

The next concern is what to do about background. I have more work to go on that aspect, but here are some brief preliminary thoughts. Background variability won't be an issue for the majority of sources. It is also difficult to come up with a scheme that can be employed within the context of the Gregory & Loredo algorithm. In terms of eq. (3), background would be a correction to the f_j factors, appearing in both the product before and the sum within the exponential. One could kind of "fudge it" via the effective area, if one first characterized the mean of the lightcurve, and then characterized the variability of the background, and had the effective area fluctuate with a term proportional to the normalized, varying background relative to the lightcurve mean. That approach is kind of ugly, and is really only very approximate, but it might reduce sensitivity to background variations in some cases.

Instead, one could run the Gregory & Loredo variability algorithm on the background lightcurve, but force the optimal background lightcurve to be evaluated on the bins of the optimal source lightcurve. One could then scale the background area and rate to that of the source, and place limits on the fraction of source variability attributable to the background. To do this properly, `dither_region` would also have to be run on the background region, invoking another 20–30 sec hit per source. Is this trade off worth it? I think that issue needs more study before I can come up with an answer.

Bottom line for this memo: The dither correction via `dither_region` applied in the Gregory & Loredo algorithm appears to work well. We need to iterate the existing `glvary` code with the results from the `S-lang` code, to a) make sure that we agree on the basics of the algorithm, and b) see if there are improvements that can be made to the former.

```
% We will use the GSL module for interpolating and
% integrating the effective area/deadtime function

require("gsl");

% Although the GSL lngamma function is pretty fast,
% we are only working with integer counts. Tabulating
% the lngamma function is slightly (5%) faster.  There's
% no benefit in tabulating the log(integer) function.

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);
}

define ln_gamma(i)
{
   return log_sum[int(i-1)];
}

% The meat of the routine.  t is the event times array,
% tmin & tmax are scalars for the max and min of the lightcurve,
% mmax is the maximum number of partitions to try,
% ta are the times at which the integrated normalized effective
% area/deadtime curve is defined.
% adt is the array for the effective area/deadtime curve.

define log_odds(t,tmin,tmax,mmax,ta,adt)
{
   % Effective area/deadtime curve needs to be defined for the
   % whole range of the lightcurve, so truncate lightcurve if
   % it isn't

   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 bounds

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

   variable lo,hi,tj,im,j;
   variable n=length(t);

   % Define the spline of the effective area curve
```

```
variable spline_adt = interp_akima_init(ta,adt);

% Get log of average effective area for the whole curve

variable a_avg, dt_int = tmax - tmin;
a_avg = log(interp_eval_integ(spline_adt,tmin,tmax)/dt_int);

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

foreach im (m)
{
   % Create grid for partitioning lightcurve into im bins

   (lo,hi) = linear_grid(tmin,tmax,im);

   variable aavg = Double_Type[im];

   % For this partitioning of the lightcurve, average
   % deadtime/effective area in each bin.  For most sources,
   % this is the bottleneck in run time.  If you don't need
   % this piece, skipping it will really speed things up.
   % Note also that for completely dead bins, we set equal
   % to the average effective area, so it doesn't contribute
   % to the sum (since we should have 0*-infty => 0 there).

   foreach j ([0:im-1])
   {
      aavg[j]=interp_eval_integ(spline_adt,lo[j],hi[j])*im/dt_int;
      if(aavg[j]==0.){ aavg[j]=a_avg; }
   }

   % For each lightcurve partition, the arrays of average
   % deadtime/effective area in each bin.

   aj[im-2] = aavg;

   % 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

   lods[im-2] = sum( nj[im-2]*(a_avg-log(aj[im-2])) +
                     ln_gamma(nj[im-2]+1)            )
                + n*log(im) + ln_gamma(im) - ln_gamma(n+im);
```

```
   }

   % Return log of Odds ratio, *without* the 1/nu = 1/(m_max-1)
   % normalization (i.e., uniform prior for the model with m
   % divisions), the number of partitions in each curve, the counts
   % per bin arrays for each partitioning, the min and max times
   % actually used, and the average effective area per bin arrays for
   % each partitioning. aj, tmin, tmax are used in the lightcurve,
   % the other bits are used in odds calculations

   return lods,m,nj,tmin,tmax,exp(a_avg),aj;
}

#iffalse
variable nfile = 118;
#endif
#iftrue
variable nfile = 1;
#endif

variable file,fp,t_event,lods,lomax;
variable nj,aj,lm,m,mmax,p,csum,imax,lt,iw,mpeak,msum,lsum;
variable tlo,thi,tfrac,k,ii,mloop,ntot,oratio,rate,erate;

fp = fopen("gl_results_all","w");
() = fprintf(fp,"Source  N_event  mpeak   mmax  Log(Odds)    P \n");
() = fclose(fp);

variable tmin = 7.2039524e7, tmax = 7.2141507e7;
variable tmin_used, tmax_used;
tmax -= tmin;
mmax = int(min([tmax/50,3000]));
lm = mmax-1;

variable i=1;
loop(nfile)
{
   file = sprintf("%04d",i);

#iftrue
   t_event = fits_read_col("dither_sourceV.fits","time");
#endif

variable na = 59990, ta=[0:na-1]*(tmax/(na-1)), a0;
variable adt = ones(na);

#iftrue
   (ta,adt) =  fits_read_col("dither_sourceV.frac","time","fracarea");
   ta = ta-tmin;
#endif
```

```
t_event -= tmin;
lt = length(t_event);

% Get the log of the odds ratio (sans normalization factor)

(lods, m, nj, tmin_used, tmax_used, a0, aj) =
   log_odds(t_event,0,tmax,mmax,ta,adt);

% This bit uses the return pieces from above to find what the
% maximum odds ratio is, and temporarily takes that out (lomax),
% and then also truncates the number of lightcurve binnings kept
% beyond the max of the odds ratio.  The former is a replacement
% for Arnold's bias parameter - but should never fail or need to
% be changed beyond the logic given.  The latter is also done by
% Arnold's code, but I think I might be doing it with less
% recalculation, and thus perhaps a little faster.

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

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

% Truncating the number of partitionings of the lightcurve

imax = max(where(csum > max(csum)/exp(0.5)));
iw = where(lods[[0:imax]]==max(lods[[0:imax]]));

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

mpeak = min(iw)+2;

% Calculate the total probability of variability (p) and the odds
% ratio for each partitioning of the lightcurve (oratio).  Again,
% as written, I don't expect any numerical issues.  Any
% potentially large numbers (lsum, lomax) appear in exp(-#), and
% hence should go to zero for anything worrisome.

msum = sum(lods[[0:imax]]);
lsum = log(msum) + lomax - log(imax+1);
p = 1/(exp(-lsum)+1);
oratio = lods[[0:imax]]/(msum+(imax+1)*exp(-lomax));

% For p>0.9, plot and save the info.  (Note - I haven't added
% the additional logic by Arnold to calculate 3 sigma deviations
% of the lightcurve, as an additional variability check.)

if(lsum > log(9))
{
```

```
fp = fopen("gl_results_all","a");
() = fprintf(fp," "+file+"   %7i   %4i   %4i  %9.2f   %4.2f\n",
             lt,mpeak,imax+2,lsum,p);
() = fclose(fp);

(tlo,thi) = linear_grid(0,tmax,imax+2);
tfrac = (tlo+thi)/2./tmax;
rate = Double_Type[imax+2];
rate[*]=0.;
erate=@rate;

ntot=sum(nj[0]); % We might not have used all the data ...

variable nj_ii_k, oratio_ii, aj_ii_k, drate;

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

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

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

% Note that here I differ from Gregory-Loredo and Arnold.
% G-L, for reasons unbeknownst to me, only include the
% variable part of the lightcurve (i.e., partitionings
% with 2 or more bins).  I think that's because they
% presume that p~1, and therefore why bother?  Given that
% we could potentially go down to p~0.6, realistically
% the constant part of the lightcurve estimate should be
% included.  Hence the next two lines.

rate += (1-p)/a0;
erate += (1-p)/a0^2;
erate = sqrt(erate-rate^2);
rate = rate*ntot/(tmax_used-tmin_used);
erate = erate*ntot/(tmax_used-tmin_used);

%  Print and plot stuff.

()=printf("%4i    %4i    %4i    %6i    %9.2f\n",
          i,mpeak,imax+2,int(sum(nj[0])),lsum);
```

```
      xrange;
      yrange(min([rate-2*erate,0.8*rate]),max([rate+2*erate,1.2*rate]));
      charsize(1.12);
      xlabel("\\frTime (sec)");
      ylabel("\\frCorrected Rate (cts/sec/area)");
      set_frame_line_width(3);
      set_line_width(2);
      linestyle(4);

      plot([tlo[0],(tlo+thi)/2.,thi[length(thi)-1]],
           [rate[0]+erate[0],rate+erate,
            rate[length(rate)-1]+erate[length(rate)-1]],2);
      oplot([tlo[0],(tlo+thi)/2.,thi[length(thi)-1]],
            [rate[0]-erate[0],rate-erate,
             rate[length(rate)-1]-erate[length(rate)-1]],2);

      linestyle(1);

      oplot([tlo[0],(tlo+thi)/2.,thi[length(thi)-1]],
            [rate[0],rate,rate[length(rate)-1]],1);
   }
   i++;
}

xrange(); yrange();
```