On-Orbit Pixel Responsivity (Flat-field) Estimation

F. Masci, version 1.0 (3/26/2008)

1. Background

This document outlines the specifications for developing a module that takes on-orbit image data and creates a responsivity map (flat-field). The responsivity map is a calibration product that stores *relative* pixel-to-pixel responsivity variations for an array. This product will be created dynamically, i.e., when a desired number of (good quality) frames become available. It will then be applied to frames in the instrumental calibration pipeline. A suggested name for this module is *flatcal,* for "flat-field calibration". The developer can choose a new name if this doesn't appeal.

The method is essentially the same as that used on 2MASS where flat-fields were derived from the twilight sky. On WISE, we will use the zodiacal background. In a nutshell, the method entails measuring the *change* in intensity in a pixel in response to the variation in overall (e.g., median) intensity in a frame. The relative responsivity for a pixel is given by the slope of a *robust* linear fit to the pixel versus overall intensity data. This will use scans where the background is known *a priori* to increase monotonically. It is expected to vary by ~40-50% in bands 1 and 2, and ~30% in bands 3 and 4 from an ecliptic pole to equator (~ a quarter orbit). An important assumption for this method to work is that *every* pixel in an array must see the same intrinsic *change* in the background. A scenario where this may fail is described in section 4. We shall refer to this method as the *slope method*.

The slope method helps mitigate incomplete knowledge of a *static* absolute dark/bias bias level since this effectively cancels out. This is generally not true for the classic flat-field estimation method based on combining frames in a stack and then normalizing. In the slope method, short-term fluctuations will still contribute to the scatter in the intensity measurements, and hence uncertainties in the final responsivity estimates.

Crude estimates by the Project Office show that at least 15 orbits (i.e., 60 quarter orbit scans or ~thousands of frames) will need to be combined to obtain flats to better than 1% accuracy in bands 1 and 2. The reason is that the background signal in bands 1 and 2 will be low and many frames are needed to get good signal-to-noise. For bands 3 and 4, at least one orbit worth (i.e., 4 quarters) will give responsivity maps to the desired accuracy to meet sensitivity requirements. This is a detail that may be needed by the developer for memory allocation and resources, especially for bands 1 and 2.

2. Command-line Synopsis and I/O

Here's a baseline (suggested) synopsis – basically the on-screen tutorial if the module is executed with no command-line arguments. Some inputs and variables are further discussed in section 3. The developer is free to reword, reorder any of the below. I'm sure there'll be more as development proceeds.

```
Program flatcal vsn 1.0

Usage: flatcal
 -f1 <inp_img_list_fname>    (Required: list of input pre-calibrated frames in
                              FITS format)

 -f2 <inp_mask_list_fname>   (Optional: list of input bad-pixel masks in 32-bit
                              INT FITS format; only values 0 -> 2^31 are used)

 -f3 <inp_unc_list_fname>    (Optional: list of input uncertainty images in
                              FITS format)

 -lf <min_frame_signal>      (Optional: minimum median frame signal to retain;
                              units = native image units; Default = no limit)

 -hf <max_frame_signal>      (Optional: maximum median frame signal to retain;
                              units = native image units; Default = no limit)

 -lt <lower_threshold>       (Optional: lower-tail SNR threshold for in-frame
                              outlier trimming; Default = 5)

 -ut <upper_threshold>       (Optional: upper-tail SNR threshold for in-frame
                              outlier trimming; Default = 5)

 -m  <inp_mask_bits>         (Optional: mask template [decimal] specifying
                              bits to flag/omit from processing; Default=0)

 -r  <rescale_uncerts>       (Optional switch: if specified, inflate or deflate
                              input uncertainties to obtain reasonable
                              chi-squares and parameter uncertainties)

 -o1 <out_slope_img>         (Required: output flat-field [slope-fit]
                              image FITS filename)

 -o2 <out_slope_unc_img>     (Required: output flat-field [slope-fit]
                              uncertainty image FITS filename)

 -o3 <out_intercept_img>     (Optional: output intercept-fit image FITS
                              filename)

 -o4 <out_intercept_unc_img>(Optional: output intercept-fit uncertainty
                              image FITS filename)

 -o5 <out_co-std-dev_img>    (Optional: output co-standard deviation [between
                              slope and intercept] image FITS filename;
```

```
                              pixel value = sign[cov] sqrt[|cov|])

-o6 <out_mask_img>           (Optional: output 8-bit mask FITS filename
                              flagging pixels with unreliable/bad responsivity
                              estimates)

-d                           (Optional: switch to print debug statements
                              to stdout, ancillary QA to ascii files)

-v                           (Optional: switch to increase verbosity to stdout)
```

3. Suggested Algorithm and Details

Below is an outline of the main processing steps. I encourage the developer to fill in any gaps.

- Input frames for flat-field creation will have already been pre-filtered according to quality metrics at the *ingest*-QA stage: e.g., frames will be flagged according to those affected by anneals, South Atlantic Anomaly passes, other predicted events, but also unforeseen events such as bright sources and persistence there from. The frames will have also been pre-calibrated, e.g., de-biased, droop-corrected, dark subtracted and linearized. The units of the pixel values in each are in native WISE "*scaled*-slope" units with *no offset*. More specifically, they will be in units of *scaled* DN/SUR, where DN = Data Number and SUR = Sample Up the Ramp.

- The input list will contain frames from as many orbits and scans as are necessary, in no particular order. Since the zodiacal background will increase from ecliptic pole to equator, then decrease towards the opposite pole, and then repeat again for the second half of an orbit, we make no distinction between the different quarter orbits. A plot of signal for the same pixel versus median signal over a frame should not matter where in the orbit the frame came from. In other words, we expect the ratio of pixel to median frame signal to only vary as a consequence of fluctuations in the background on sub-frame scales (from both instrumental transients and the real structure itself). Therefore all frames (regardless of orbit location) can be included in the one fit.

- After reading the input frames and accompanying mask/uncertainty images, prior-known bad pixels should be flagged according to the mask bits specified by the <–m> input option. These will be omitted from further processing. For pixels flagged as permanently bad in all frames, no responsivity estimates will be possible.

- It is not known at the time of writing exactly what filters will be needed to isolate the optimum (or useful) dynamic range in signals for this exercise. This may be driven by the caveat described in section 4, or, since there will be "crowding" at the poles and equator where there won't be much variation in background signal, some trimming of frames may be needed to avoid those regions dominating the scatter and deteriorating the fits. I.e., we want to retain sequences of frames where the change in background is *maximal*. One way to filter the input frame list is according to optimum ecliptic latitude ranges once we have a better idea of how the WISE sky looks like. Orbit-position filtering (if needed) will be done upstream, i.e., when creating the input lists, not in *flatcal*. Another filtering metric involves truncating at pre-determined low/high median frame signal values. This type of filtering can be done in *flatcal*. The input options <-lf> and <-hf> are suggested for this purpose. A method for computing the median frame signal is described next.

- To compute the median signal in each frame, I suggest using a method that doesn't significantly bias the result due to outliers (e.g., real sources in the upper tail of the pixel histogram). A good all-round, pseudo-outlier-resistant method is the MADTUT method[1] or some variation thereof. This method also provides a *robust* estimate of sigma of the distribution (appropriately scaled for consistency with asymptotic normality). This sigma is important because it can be used to detect pixels affected by outliers (including sources) in each frame. The input lower/upper signal-to-noise ratio thresholds: <-lt> and <-ut> may be used for this purpose. These outliers will wreck havoc in the fitting if not flagged. A scheme to detect and flag additional rogue outliers during the fitting stage is described below. The latter may not be needed if the bulk can be detected in the above step. Note: not all parameters to support the MADTUT method were included in the above synopsis. This is because it is relatively parameter insensitive, and any parameters can be included in a namelist. At this stage in processing, we expect the median signal *med*(*S*), and a list of outlier + prior-known bad pixels for each post-filtered input frame (i.e., satisfying -lf < *med*(S) < -hf) to have been stored in memory.

- Unless the developer is feeling ambitious, uncertainties in frame median estimates *need not* be computed and used in the fitting (i.e., for the abscissae). This is because they are expected to be about a thousand (i.e., $\sim \sqrt{[1024^2]}$) times smaller than the actual pixel-signal uncertainties (the ordinate measurements). Note: uncertainties in the abscissae will make the fitting procedure (via χ^2 minimization) non-linear.

- Now to the actual fitting procedure. I suggest using the standard χ^2 minimization method[2] because input prior uncertainties are available, and I'm optimistic that the outlier detection step above will be reliable enough to avoid biasing parameter estimates. This obviates the need to implement some non-parametric, outlier-resistant regression method. We are interested in fitting a straight-line: $y = m_{ij}\, x + c_{ij}$ to a data set $\{x_k, y_{ijk}\}$, where here x_k corresponds to the median signal in some frame k [$med(S_k)$], and y_{ijk} to the signal in a specific pixel i, j of frame k [S_{ijk}]. Estimates for the slope m_{ij} (relative responsivity), intercept c_{ij}, and their (co)variances when this model is fit to a set of frames $k = 1 \ldots N$ can be written analytically:

[1] See: http://web.ipac.caltech.edu/staff/fmasci/home/wise/MADTUT_method.pdf
[2] E.g., see: http://web.ipac.caltech.edu/staff/fmasci/home/statistics_refs/Chi-square-min.pdf and http://web.ipac.caltech.edu/staff/fmasci/home/statistics_refs/Chi2andLinearFits.pdf

$$m_{ij} = \frac{KK_{xy} - K_x K_y}{KK_{xx} - K_x^2}, \qquad c_{ij} = \frac{K_{xx} K_y - K_x K_{xy}}{KK_{xx} - K_x^2},$$

$$\sigma^2_{m_{ij}} = \frac{K}{KK_{xx} - K_x^2}, \qquad \sigma^2_{c_{ij}} = \frac{K_{xx}}{KK_{xx} - K_x^2},$$

$$\mathrm{cov}(m_{ij}, c_{ij}) = \frac{-K_x}{KK_{xx} - K_x^2},$$

where :

$$K = \sum_{k=1}^{N} \frac{1}{\sigma^2_{y_{ijk}}}; \quad K_x = \sum_{k=1}^{N} \frac{x_k}{\sigma^2_{y_{ijk}}}; \quad K_y = \sum_{k=1}^{N} \frac{y_{ijk}}{\sigma^2_{y_{ijk}}}; \quad K_{xx} = \sum_{k=1}^{N} \frac{x_k^2}{\sigma^2_{y_{ijk}}}; \quad K_{xy} = \sum_{k=1}^{N} \frac{x_k y_{ijk}}{\sigma^2_{y_{ijk}}}$$

Note, if pixel i, j in any frame k was flagged as an outlier, or as bad from an input mask, then that pixel (or more specifically, the data-pair $\{x_k, y_{ijk}\}$) should be omitted from the fit. There will be a fit for every *good* pixel, i.e., with sufficient data from all N frames. This obviously excludes pixels flagged as (permanently) bad in all frames of the stack.

- The chi-square function for pixel i, j is defined by:

$$\chi^2(m_{ij}, c_{ij}) = \sum_{k=1}^{N} \frac{\left[y_{ijk} - m_{ij} x_k - c_{ij} \right]^2}{\sigma^2_{y_{ijk}}}$$

If the model and pixel-signal uncertainties are trustworthy, we expect $\chi^2 \approx D_F = N - 2$, i.e., the number of degrees of freedom. Actually, a χ^2 within a few standard deviations of this value is considered "good", i.e., within $D_F \pm 3\sqrt{(2D_F)}$.

Outliers that escaped detection in the frame median step above will inflate the overall χ^2 beyond the desirable $D_F + 3\sqrt{(2D_F)}$ maximum. This first step should have detected most of the outliers, and we foresee no problem if this was performed a little on the aggressive side (e.g., clipping at $\pm 3\sigma$) since we expect a good number of pixels (N) to have survived across frames to enable a reliable measure of the slope. The following *2nd pass* outlier detection step is therefore *optional*, and it may be included in a future upgrade of *flatcal* in case the initial outlier detection step does not perform as expected. This special processing should only be performed if a command-line switch is set (not shown in above synopsis). If set, then processing should proceed as follows:

1. If $\chi^2 > D_F + n\sqrt{[2D_F]}$, where n is some input parameter (e.g., default value 3), then either there could be an outlier, or, the σ_{y_ijk} are under-estimated. Our hypothesis here is that the larger than expected χ^2 is due to outliers alone. If this condition is not satisfied, stop here, otherwise go to step 2.
2. Find the largest residual in absolute value: $r_{max} = | y_{ijk} - m_{ij} x_k - c_{ij} |$;
3. Reject the measurements (x_k, y_{ijk}) corresponding to r_{max} from the data-set, and estimate new values for the slope, intercept and chi-square: m_{ij}, c_{ij} and $\chi^2_{new} = \chi^2(m_{ij}, c_{ij})$.

4. If $\chi^2_{new} \le D_F + n\sqrt{[2D_F]}$, stop here. If this condition is not satisfied, go back to 1 and repeat the process by omitting the next largest residual. Continue iterating until some pre-specified fraction *f* of the *N* initial measurements (x_k, y_{ijk}) have been rejected. A suggested default for *f* is 0.5.

If there is no convergence to give $\chi^2_{new} \le D_F + n\sqrt{[2D_F]}$ after $N*f$ measurements have been discarded, this is a strong indication that the input uncertainties σ_{y_ijk} have been *under*-estimated. Note, we are ignoring here the (unlikely) possibility that the initial assumption of a linear model was bad. However if initially $\chi^2 < D_F - n\sqrt{[2D_F]}$, this is an indication that the σ_{y_ijk} have been *over*-estimated. This leads to the next step: uncertainty rescaling.

- Rescaling of the input uncertainties should only be considered if the input switch <-r> was specified. It should only be performed if the final value of χ^2 (after any additional outlier rejection) satisfies: $|\chi^2 - D_F| > n\sqrt{[2D_F]}$, where *n* is some input parameter (same as above). This rescaling is needed in order to obtain reasonable estimates for *uncertainties* in the slope (responsivity) and intercept. As an aside, if all the σ_{y_ijk} were equal to the same value, *m* and *c* would be independent of the σ_{y_ijk} but σ_m and σ_c would depend on them explicitly (see equations above). This means that for the *m* and *c* estimates, σ_{y_ijk} merely contribute as (inverse-variance) weighting factors, but for σ_m and σ_c the magnitude of the σ_{y_ijk} are crucial. The input σ_{y_ijk} can be scaled by the operation:

$$\sigma_{y_{ijk}} \rightarrow \sigma^{new}_{y_{ijk}} = \sigma_{y_{ijk}} \sqrt{\frac{\chi^2}{D_F}},$$

where D_F is the number of degrees of freedom, $N - 2$, and *N* is the number of data-points used in the final fit. Following this rescaling, *m* and *c* need not be re-estimated. This will only affect σ_m, σ_c and cov(*m*, *c*).

- The output product files specified by <-o1>, <-o2>, <-o3>, <-o4>, and <-o5> are FITS images of the slope (responsivity), slope 1-σ uncertainty, intercept, intercept 1-σ uncertainty, and signed co-standard deviation respectively. An example header is below. The angle brackets indicate information whose literal content depends on actual execution or generation circumstances.

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                <Num> / length of data axis 1
NAXIS2  =                <Num> / length of data axis 2
BAND    =                <Num> / WISE band number (1, 2, 3 or 4)
NUMINP  =                <Num> / Number of input frames used
UTCSBGN =                <Num> / Earliest UTCS in frame stack [sec]
UTCSEND =                <Num> / Latest UTCS in frame stack [sec]
FRMIDSEQ=          <'Num..Num'> / Range of frameIDs used
COMMENT <type> for WISE flat calibration, created <YYYY-MM-DD>
COMMENT generated by flatcal, v.1.0 on <YYYY-MM-DD> at <HH:MM:SS>
```

Notes:

  - BITPIX = -32 refers to single precision floating point.

- For bands 1, 2 and 3, the input frames will have NAXIS1 = NAXIS2 = 1016. For band 4, NAXIS1 = NAXIS2 = 508.
- The UTCSBGN, UTCSEND keywords specify the start/end observation time of frames in the input ensemble. These time tags will be available in the input frame headers [e.g., UTCS_OBS]
- The FRMIDSEQ specifies the range of input frame IDs: a string composed of the *min* and *max* ID delimited by two dots, e.g., '31412..31505'. The frame IDs will be present in the input frame headers [keyword unknown at time of writing].
- The second-last COMMENT field indicates the product type where <type> is any one of the above five product names.

- If a *slope* estimate was unreliable or could not be computed for any pixel: e.g., flagged due to a permanently bad pixel in the stack, too many outliers (hence not enough data), or too noisy (with signal-to-noise ratio $m/\sigma_m < 2$, a suggested default), then a bit should be set in the output mask <-o6> if provided on input. It is left to the developer to decide if each of these conditions will merit their own bit in the mask.

4. Potential Caveat

It was mentioned above that for the *slope method* to be reliable, *every* pixel in an array must see the same intrinsic *change* in the background on average. An example where this may fail is if the background intensity exhibits some curvature along a scan. This implies that there will be a gradient that changes from frame-to-frame. The signals in pixels (*relative* to the frame median background) located at the extreme edges of a frame along the *in-scan* direction will change in a systematic manner along the scan. This will bias their slope (hence relative responsivity) estimates towards either the low or high side depending on where the pixels are located. A schematic of this is shown in Figure 1.

We will not find out if this effect is significant until we're in orbit. It may be that the dynamic range (i.e., ~40-50% background variation from an ecliptic pole to equator, irrespective of longitude) is too small for curvature effects to dominate over the background fluctuations. If so, we will need an additional pre-filtering step to ensure the input frames don't exhibit gradients that *depend systematically* on the total signal, e.g., only use frames from the (linear) portion of a scan, assuming such exists. In other words, we want any *gradients* in frames along a scan to be more-or-less random and independent of signal. This filtering may be adequately handled by the functionality outlined in the 4th bullet point of section 3.

To check if global curvature effects are biasing the slope (hence responsivity), one can test either of the following hypotheses for the slope (m) and intercept (c) for pixels located respectively at the maximum/minimum extremities of a frame along the in-scan direction:

$$c > 0 \;\&\; m < 1 \qquad \textit{and} \qquad c < 0 \;\&\; m > 1,$$

where "maximum" here is defined as that frame edge *closest* to the ecliptic equator. If these conditions are *significant*, i.e., unlikely to have occurred by chance given the errors in c and m, then nature isn't being kind. As a quick check, one can also examine the m and c images directly for patterns consistent with the above. It also doesn't hurt to check that the median background does indeed exhibit *curvature* in the right direction (i.e., is *concave-down* like in Fig. 1 where $d^2 med(S)/dt^2 < 0$, and t is time measured for a frame sequence going from pole to equator). If these conditions are satisfied, then further pre-processing of the

input data is needed. These checks are beyond the scope of the *flatcal* module. This caveat is described here for future reference in case it becomes a problem. We are optimistic that nature will be kind.

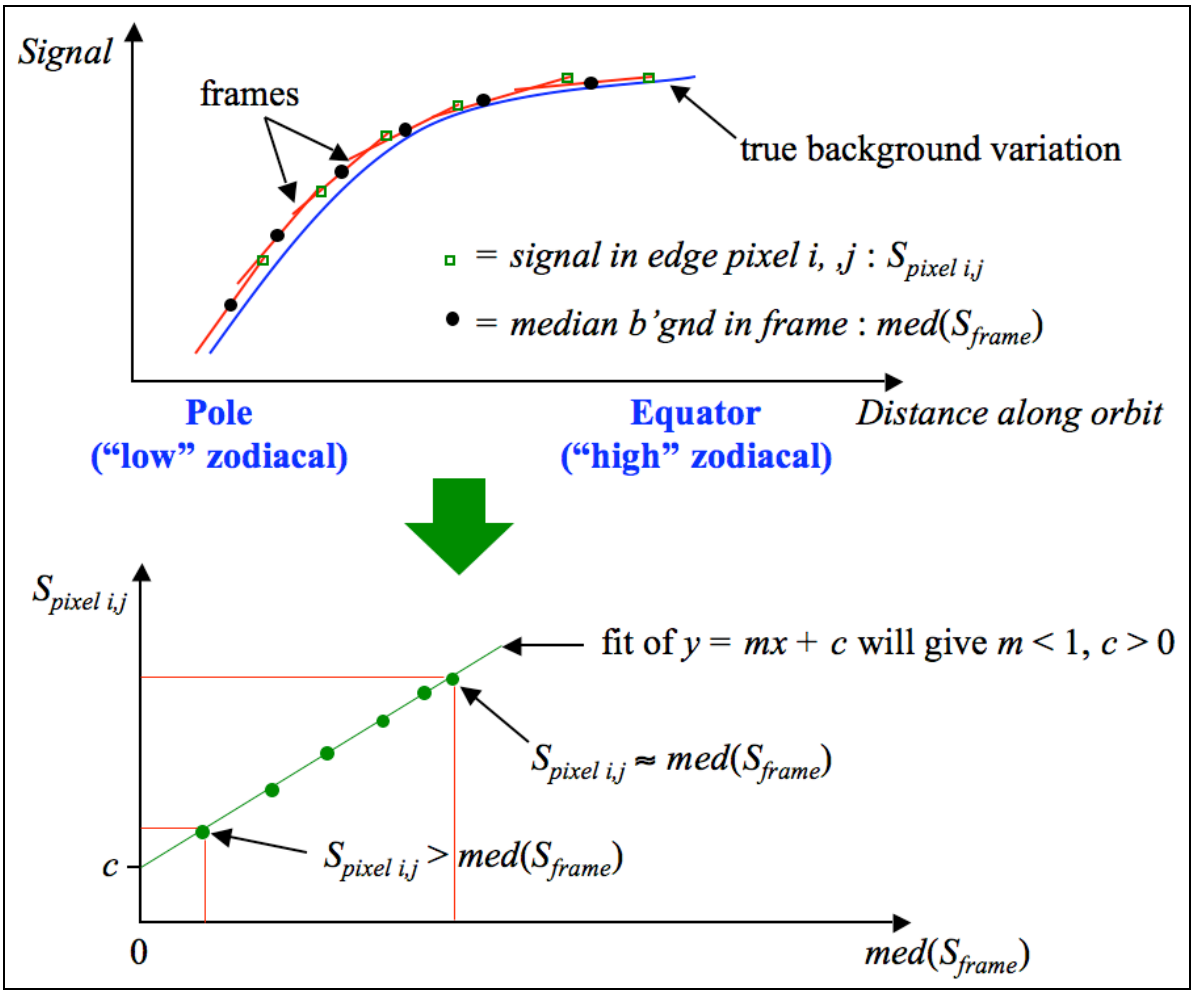

Figure 1: schematic showing exaggerated curvature in the background and its impact