KPO @AMES DESIGN NOTE

Design Note No.: KADN-26197

Title: Pre-Launch Model of LDE Undershoot

Author: Jeffrey Van Cleve

Signature: [Signature]

Instrument Scientist Approval: Doug Caldwell

Signature: [Signature]

SO Approval: Michael Haas

Signature: [Signature]

Distribution: M. Haas, J. Jenkins, D. Caldwell, J. Kolodziejczak, S. Bryson, V. Argabright (BATC)

Revision History:

<table>
<thead>
<tr>
<th>Rev. Letter</th>
<th>Revision Description</th>
<th>Date</th>
<th>Author/Initials</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial Unsigned quasi-Release</td>
<td>12/11/2008</td>
<td>JVC</td>
</tr>
<tr>
<td>A</td>
<td>Incorporate JMJ comments about inversion; formula indexing consistency in Section 3.1</td>
<td>9/3/2015</td>
<td>JVC</td>
</tr>
</tbody>
</table>
Overview:
Undershoot is the distortion of signals with large pixel-to-pixel variation. The undershoot is modeled by a three time constant model, though unmodeled fine structure of uncertain relevance is detectable in the long time-constant regime. The three time constant model of step function undershoot is numerically differentiated to give the δ-function response, which is the undershoot forward kernel. The elements of the kernel are shown to equal the filter parameters used in the undershoot correction algorithm. The stability of this correction is shown by representing the kernel as a matrix, and calculating its inverse. Errors in the finite kernel are calculated by comparing corrected data to a step function, and errors in the kernel elements are calculated by differencing the results obtained with two different data sets.

Photometric errors resulting from image motion and kernel truncation are on the order of 200 ppm/pixel. Hence, if Kepler achieves its pointing requirements, corrected overshoot will not add significantly to photometric noise.

Recommendations:

Reference Documents

2283391  Kepler GSE SW Detector Properties Pipeline Description
KADN-26176  Catalog, Conversion, and Format of Ball Test Data from the SO to the SOC

Applicable Documents

Open Items/Action Required

TBDs/TBRs

Paragraph  TBD Item
Table of Contents

1. Introduction .................................................................................................................................................. 3
2. Data ............................................................................................................................................................... 4
   2.1 Experimental Method and Data Set ........................................................................................................ 4
   2.2 Example Images ....................................................................................................................................... 4
   2.3 Calculation of Step Function Response ............................................................................................... 5
   2.4 Example Step Function Responses ....................................................................................................... 6
3. Representations of Undershoot ................................................................................................................... 8
   3.1 Discrete Filter Difference Equation ...................................................................................................... 8
   3.2 Lower Triangular Matrix Representation ............................................................................................ 9
   3.3 δ-function Kernel from Step Function Response ................................................................................ 10
   3.4 Three Time Constant Modeling ......................................................................................................... 10
   3.5 Forward Kernel from 3 Tau fit ............................................................................................................. 10
   3.6 Existence and Calculation of Reverse Kernel ..................................................................................... 10
4. Error Analysis ............................................................................................................................................... 11
   4.1 Total Flux Error from Filter Truncation ............................................................................................... 11
   4.2 Errors in Filter Parameters from Comparison of the Two Data Sets .............................................. 11
   4.3 Residuals in Corrected Data .............................................................................................................. 12
   4.4 Photometric Errors induced by Image Motion ................................................................................... 13
5. Algorithm Functional Description ............................................................................................................... 13
   5.1 Overview ................................................................................................................................................ 13
   5.2 Details of Each Program ...................................................................................................................... 14
      5.2.1 Pipeline ......................................................................................................................................... 14
      5.2.2 threetau_fit .................................................................................................................................. 14
      5.2.3 kernel_invert ............................................................................................................................... 15
      5.2.4 correct_undershoot ..................................................................................................................... 15
      5.2.5 undershoot_output_format ......................................................................................................... 15
6. Results .......................................................................................................................................................... 15
7. IDL Code .................................................................................................................................................... 16
   7.1 Pipeline ................................................................................................................................................ 16
   7.2 threetau_fit .......................................................................................................................................... 16
   7.3 kernel_invert ....................................................................................................................................... 18
   7.4 correct_undershoot ............................................................................................................................ 20
   7.5 undershoot_output_format .................................................................................................................. 21

1. Introduction

Undershoot is the distortion of signals with large pixel-to-pixel variation, in the direction of the change. Pixels following a steeply decreasing signal go below the undistorted signal for those pixels, and vice versa for steeply increasing signals. For example, when a step function input – such as Charge Injection – goes to zero relative to the local black level of the CCD, the observed signal goes below zero and remains detectably below zero for the longest delays analyzed (20 pixels). The undershoot is modeled by a three time constant model, though unmodeled fine structure of uncertain relevance is detectable in the long time-constant regime. The short time constant term is nonzero in only a few channels, and has a positive sign. The
medium time constant term varies in amplitude by an order of magnitude across the FPAA, from -2.0% to -0.2%, with time constants between 0.3 and 1.0 pixels. The long time constant amplitude, however, varies by only +/-10% across the FPAA.

The three time constant model of step function undershoot is then numerically differentiated to give the $\delta$-function response; that is, the undershoot forward kernel. The elements of the kernel are shown to equal the filter parameters used in the undershoot correction algorithm. The stability of this correction is shown by representing the kernel as a matrix, and calculating its inverse. The reverse kernel is extracted from this inverse matrix and used to remove undershoot from the observed data. Errors in the finite kernel are calculated by comparing corrected data to a step function, and errors in the kernel elements themselves are calculated by differencing the results obtained with two different data sets.

2. Data
2.1 Experimental Method and Data Set

The data used were 275 coadd FFIs, forward-clocked, cold CCD with timestamp 200803140517 and 200803140837, collected during Photometer Thermal Vacuum testing, Cold Acceptance. The results from these two data sets were averaged together to give the delivered parameters. Charge Injection (CI) was enabled, which puts a step function signal into rows 1059:1062, ending at column 1111. The CI amplitude is typically ~4000 DN per frame.

2.2 Example Images

Figure 1 shows the undershoot for channel 11.3, which along with 15.3 showed the largest cold overshoot. The linear scaling on the image corresponds to a single-frame equivalent range of [656.3, 660.0] DN. Note that the undershoot is visible to the edge of the array, or a distance of 20 columns after the end of the CI step function.
2.3 Calculation of Step Function Response

The Charge Injection step function response is calculated from images using this method:

1. Subtract 2\textsuperscript{nd} order polynomial fits to trailing black and smear and remove from image.
2. Take the mean of the good pixels for rows 1059:1061 for each column. Row 1062 is omitted because of FGS clock crosstalk.
3. Calculate mean of the good pixels in adjacent smear (non-CI) row pairs for each column, rows 1057:1058 and 1063:1064
4. Subtract item 3 from item 2.
5. Normalize item 4 to 1 at column 1111.
2.4 Example Step Function Responses

Figure 2 shows example responses to the charge injection step function. The largest, least and median overshoots are shown, as well as an example of an anomalous overshoot in which the first pixel (column 1112) is greater than the 2nd (column 1113). Figure 3 shows the same data, but scaled to better show the long time constant region. It is interesting that while the undershoot amplitude between 1 and 3 pixels after the step function varies considerably from channel to channel, the undershoot 4 or more pixels after the step function is about the same for all channels.

Figure 2. Example undershoots. The charge injection step function has amplitude unity and ends at column 1111. The legend identifies the channels.
Figure 3: Same data as Figure 2, but scale expanded to show long time-constant region.

Figure 3 shows oscillations around the long time constant decay, for columns > 1115. Closer examination of the images, in Figure 4, shows that these oscillations are different from row to row. In the interests of bounding the complexity of the undershoot correction algorithm, no attempt has been made to model these oscillations. However, I will include them in my error analysis in Section 4.3.
3. Representations of Undershoot

3.1 Discrete Filter Difference Equation

The undershoot correction model is a set of filter coefficients. The forward model transforms the “true” signal $X(x_1, x_2, \ldots, x_n, \ldots x_{N_{\text{sam}}})$ into the observed signal $Y(y_1, y_2, \ldots y_n, \ldots y_{N_{\text{sam}}})$ which has the undershoot. The reverse model transforms the observed data into the pre-undershoot signal.

The coefficients will be formatted for use with MATLAB's built-in "filter" function, which can use $N_B$ feed-forward coefficients $B$ and $N_A$ feedback coefficients $A$. Since $X$ is an ordered time series of sampled CCD signal transformed by the LDE electronics, causality suggests that $A_1=1$ and all other $A_j=0$. All subsequent discussion is based on this assumption.

The MATLAB usage of the forward model to show the effect of undershoot with coefficients $B$ on the “true” signal $X$ is then

$$Y = \text{filter}(B,1,X),$$

where the MATLAB filter function in this simplified case implements

$$y_n = \sum_{i=1}^{\min(n,N_B)} B_i x_{n-i+1}.$$
MATLAB documentation may refer to a coefficient vector $B$ which has $N_B + 1$ elements. Users not using MATLAB may correct the undershoot by implementing this recursive formula:

$$x_n = \frac{1}{B_1} \left[ y_n - \sum_{i=2}^{\min(n,N_B)} B_i x_{n-i+1} \right],$$

which is numerically faster than representing the forward model as a lower-triangular matrix (Section 3.2) $Y = BX$ and inverting it to do the correction: $X = B^{-1}Y$. In either case, the inverse exists and has been shown to be numerically robust by the IDL routine kernel_invert (Section 7.3).

The correction will have to be done row-by-row, as the undershoot effect occurs as a row of pixels is read out through the electronics.

### 3.2 Lower Triangular Matrix Representation

Eq. 1 with the causality assumption can be written in matrix form as

$$Y = BX$$

where $B$ is a lower triangular (LT) matrix, an example of which is shown below for a forward filter with 4 parameters:

$$\begin{pmatrix}
    b_1 & 0 & \ldots & \ldots & 0 \\
    b_2 & b_1 & 0 & & \\
    b_3 & b_2 & b_1 & 0 & \\
    b_4 & b_3 & b_2 & b_1 & 0 \\
    0 & b_4 & b_3 & b_2 & b_1 & 0 \\
    : & 0 & b_4 & b_3 & b_2 & b_1 & 0 \\
    : & 0 & 0 & & \\
    : & 0 & & & \\
    0 & \ldots & \ldots & \ldots & 0 & b_4 & b_3 & b_2 & b_1
\end{pmatrix}$$

Eq. 2

The general expression for the matrix elements is

$$B_{rc} = \begin{cases} 
0 & c > r \\
b_{r-c+1} & r - c + 1 \leq N_B \\
0 & r - c + 1 > N_B
\end{cases}$$
The notation is that of MatLab: indices are in order row, column starting from 1, and row 1 is at the top of the pictorial representation of the matrix.

3.3 δ-function Kernel from Step Function Response

Undershoot correction is tractable only if we assume undershoot is linear; that is, doubling the signal doubles the undershoot without changing its shape. Fortunately, the preponderance of experimental data supports this assumption, with the exceptions possibly due to large-area saturation effects which are not relevant for science observations. Then, since the input signal (end of CI) is step function, and the derivative of a step function is a δ-Function, the δ-Function undershoot response (forward kernel) for discretely sampled data is its one-pixel shift derivative. Then, the forward kernel equals the B-parameter array of MatLab filter function (Eq. 1, 2):

\[ X = \delta(n - n_0) = (0,0,0,0,1,0,0,0) \]
\[ Y = BX = (0,0,0,0,b_1,b_2,...,b_{N_y},0,0,0) \]

where the starting index of the nonzero Y’s is \( n_0 \). Therefore, the filter parameters are determined by inspection of the forward kernel.

3.4 Three Time Constant Modeling

Three undershoot features need to be accounted for by any model: short and long decays, and anomalous overshoots in which the 1\textsuperscript{st} pixel after the CI step function is greater than the 2\textsuperscript{nd}.

Hence, a function of the form

\[ f(t) = a_0e^{a_1t} + a_2e^{a_2t} + a_3e^{-10t} \quad \text{Eq. 3} \]

was used, where \( t = (n - 1112) \) so that \( t = 0 \) is the first pixel of the undershoot. I will call Eq. 3 the 3t fit. The last term represents the anomalous undershoot by adjusting the value of the r=0 pixel, but in a convenient functional form. The data were fit with the nonlinear least squares IDL routine CURVEFIT. The short time constant is fixed at 0.1 pixel, the medium time constant is \( 1/a_1 \), and the long time constant is \( 1/a_3 \).

3.5 Forward Kernel from 3 Tau fit

The modeled forward kernel is the derivative of Eq. 3 with respect to \( t \). Numerically, the kernel is the shift derivative of the best fit to Eq. 3.

3.6 Existence and Calculation of Reverse Kernel

Though the MatLab FILTER function requires no further inputs, it is worth considering how to show that the undershoot transformation is invertible. The determinant of a triangular matrix is the product of its diagonal elements, so the determinant of B (Eq. 2) is \( b_1^{N_y} \). If a matrix has a nonzero determinant, its inverse formally exists. Since the b\textsuperscript{1}s for each of the 84 channels are within a few percent of unity, all undershoot matrices are invertible.
To show the inverse is numerically meaningful, the numerical calculation of the inverse is done by the IDL INVERT routine, which returns 0 for successful inversion, 1 for a Singular array (which indicates that the inversion is invalid), and 2 to warn that a small pivot element was used and that significant accuracy was probably lost. The elements of the reverse kernel can then be extracted from any row > 20 of the inverse matrix, since these rows operate on pixels for which all 20 elements of the forward and reverse kernels can be applied. Since the inverse of an LT matrix is also LT, small but nonzero values to the right of the diagonal in the inverse are numerical errors. Empirically, all inversions returned 0, hence an inverse exists and is robust.

I will make the unproven assertion that the existence of an inverse for the 128x128 representation of Eq. 2 is equivalent to the existence of an inverse for the 1132x1132 representation, which would represent correcting an entire Kepler CCD line for undershoot. This assertion dramatically reduces computation time for the 84 matrices which must be inverted.

There is probably a proof that matrix B has a numerically robust inverse if the off-diagonal elements are small compared to the diagonal elements, and $N_B << N$, but I thought it more economical to do the calculation numerically. There is probably also a proof the number of nonzero elements in the inverse δ-function kernel, as determined by matrix inversion, is equal to the number of nonzero elements in the forward δ-function kernel.

### 4. Error Analysis

#### 4.1 Total Flux Error from Filter Truncation

As you can see in Figure 1, the undershoot is still visible at the edge of the array, a full 20 pixels from the step function. Hence, the kernels will be incomplete; that is, the sum of the forward kernel terms is greater than 1, and reverse kernel terms are less than 1, so applying the correction (reverse kernel) does not recover all of the flux. Not surprisingly, since the long time-constant term is more or less the same across the FPAA, the filter truncation error is more or less the same across the array, about 110 +/- 30 ppm. The truncation error manifests itself as a constant offset in the corrected data, as shown in Figure 6.

#### 4.2 Errors in Filter Parameters from Comparison of the Two Data Sets

To estimate the error in the forward kernel values, I differenced the results for data sets 200803140517 and 200803140837. From Figure 5, I concluded that the errors for the first 5 terms could be estimated as the standard deviation of the difference in the first 5 terms, and so on with the $6^{th}$-$10^{th}$ and $11^{th}$-$20^{th}$ terms.
Figure 5: Difference between forward kernel parameters, showing 3 distinct regimes for random errors.

4.3 Residuals in Corrected Data
The observed data can be operated on with the reverse kernel to see whether a precise step function is obtained, and to quantify deviations between the corrected data and 0 for the pixels following the step function (residuals). Figure 6 shows an example. In all cases, the mean residual was roughly equal to the truncation error as expected. While a cosmetically more appealing result can be obtained by renormalizing the kernel, the mean residual has no effect on photometric precision, and I have not renormalized the delivered files.
Figure 6: Example residuals from Application of Reverse Kernel to Data

4.4 Photometric Errors induced by Image Motion
Photometric errors are introduced by the results in Figure 6 if there is image motion; that is, the photometric noise is the shift derivative of Figure 6. I calculated the shift derivative of the residuals for each of the 84 channels, and found that the largest absolute value was 160 ppm/pixel, on channel 4.2 (labeled “worst” in the Figure). Since the 1-σ pointing of Kepler is required to be 0.009 arc second (3 sigma) per axis, maximum, for any period of 15 minutes or more while the photometer is taking data, this source of error will be negligible. I’ve assumed that image motion can be detected, and an interpolated sub-pixel kernel is applied, if the aperture is smaller than the kernel (20 pixels, which is unrealistically large for Long Cadence apertures). If not, the errors will be larger, a subject beyond the scope of this KADN.

5. Algorithm Functional Description

5.1 Overview
1. Run the modified Pipeline (2283391) on a complete set of FFI data (all 21 modules, named so that they are listed in ascending module order). The output, along with standard Pipeline output, is a pipe_out_ci*.txt file with one header line, one column for each channel (84 total), and one row for each of the 32 trailing pixels. The algorithm used is described in Section 2.3.

2. Find best parameters of 3\( \tau \) fit, and output the parameters and the shift derivative of the best-fit model results (IDL routine threetau_fit)

3. Average together 3\( \tau \) results from different data sets (spreadsheet undershoot_kernel_yyyymmdd.xls)

4. Difference results from different data sets and estimate errors (spreadsheet undershoot_kernel_yyyymmdd.xls)

5. Invert the forward kernel using IDL routine kernel_invert to obtain the reverse kernel for each channel, and prove numerically that a robust inverse exists.

6. Calculate residuals with IDL routine correct_undershoot

7. Write filter parameters = forward kernel values in format specified in KADN-26176 with IDL routine

5.2 Details of Each Program

5.2.1 Pipeline

See Section 2.3 and 2283391

5.2.2 threetau_fit

Function fit:
\[ f(x) = a[0]\exp(a[1]\times x) + a[2]\exp(a[3]\times x) + a[4]\exp(-10\times x) \]

1. Standard I/O setup

2. The data show by inspection show that 10/a_1 and a_1/a_2 are > 3 so for robust convergence I calculate initial values from the undershoot:

\[
\begin{align*}
a[0] &= \text{observed[1112]} \\
a[1] &= \text{alog(}\text{observed[1113]}/\text{observed[1112]}\text{)} \\
a[2] &= \text{observed[1122]}\times \exp(1) \\
a[3] &= 0.1\times \log(\text{observed[1125]}/\text{observed[1115]})
\end{align*}
\]

where the observed indices refer to the 0-based column index.

Anomalous undershoots are detected by the comparing the ratio of column 1112 to column 1113. Values less than 1.65 are suspicious, since they are outliers in the distribution of this ratio. So the last initial parameter is set by this logic

\[
\text{if (a[1] le -0.5) then begin}
\text{if normal, set short time amplitude to 0}
\]
a[4] = 0
endif else begin
; if anomalous, set medium time constant to median time constant of	normal channels and choose short time constant amplitude to give
amplitude 0 at pixel 0
a[1] = -1.3
a[4] = -1*observed[1112]
endelse

3. Calculate the best fit using IDL CURVEFIT, with uniform weighting. CURVEFIT returns the
parameters, their errors, and the fit evaluated at each point.
4. Plot results scaled for short and long time scales.
5. Calculate shift derivative of fit to get model forward kernel.
6. Write parameter file to disk. For each channel, there are 5 parameters, followed by 5 errors
in parameters, and chi^2.
7. Write forward kernel file to disk. For each channel, there are 20 forward kernel values, since
there are 20 pixels between the end of the step function CI and the edge of the array.

5.2.3 kernel_invert

1. Read in forward kernel values.
2. Populate a matrix as shown in Section 3.2.
3. Invert the matrix using the IDL INVERT routine.
4. Select the 20 reverse kernel values from the middle of the inverted matrix.
5. Write the INVERT status and the 20 reverse kernel values to disk.

5.2.4 correct_undershoot

1. Read in reverse kernel and observed data (pipe_out_ci*.txt)
2. Copy the observed data into the middle of a 128 element array. Pad the beginning with 1’s
and the end with 0’s to eliminate edge effects.
3. Apply Eq. 3 with X understood to be the padded data and Y the corrected data.
4. Write the corrected data to disk. The residual is the corrected values for columns 1112-
1131, since these should be zero after the step function if the correction is perfect.

5.2.5 undershoot_output_format

Just formatting here, see code.

6. Results

The results are in a file of the format specified in KADN-26176. The supporting analysis is in
spreadsheet undershoot_kernel_20080728.xls. The tabs are described below:
forward: forward (input --> observed) kernels for data sets 200803140517 and 200803140837, means and differences of these two kernels for each channel, list of errors estimated from differences.
reverse: reverse (observed → corrected) kernel for mean kernel shown on tab forward.
corrected: Observed data from 200803140517 corrected with reverse kernel, and the shift derivative of residuals (for error estimation)

7. IDL Code
7.1 Pipeline
The Pipeline code which calculates the CI end undershoot response is not shown. See Section 2.3 and 2283391.

7.2 threetau_fit

pro threetau_fit
; NAME:  threetau_fit
; PURPOSE:  calculate three time-constant fits to Kepler undershoot, and numerical differences (forward kernel)
; for each of the Kepler output channels
; REFERENCE DOCUMENT:  KADN-26197
;
; INPUTS:
; 1.  Channel list table for mappint 0,1,2...83 onto 2.1,2.2,2.3,...,24,4
; 2.  Experimental Pipeline Charge injection output table which has 1 header line, 32 lines (for each of
;   the 32 trailing columns num_channels columns.
; 3.  Definition of function threetau
;
; Optional Keyword Parameters
; None
; OUTPUTS:
; 1.  Table of fit parameters
; 2.  Table of forward kernel values
; OUTPUT PARAMETERS:
;
; COMMON BLOCKS:
; None.
; SIDE EFFECTS:
; None.
; MODIFICATION HISTORY:
; Name, Org, Date, Description
;
; Jeffrey Van Cleve  BATC/KSO  7/15/2008  Created
; 7/25/2008 Calculate finite-difference forward kernel

;default input file
ver = 'Three Time Constant Undershoot Fit 1.2'

in_dir = DIALOG_PICKFILE(/DIRECTORY,TITLE='Select Directory for Input Files ')
```
cd, in_dir
if (strlen(in_dir) eq 0) then return
input_file = DIALOG_PICKFILE(/MUST_EXIST, TITLE='Select *.txt file listing full-path file names ')
basename = FILE_BASENAME(input_file,'.tbl')

image_structure = read_ascii(input_file, data_start = 1)
if (keyword_set(debug) eq 0) then begin
  s = size(image_structure.field01)
  num_channels = s[1]
  num_measurements = s[2]
end else begin
  s = size(image_structure.field1)
  num_channels = 1
  num_measurements = s[1]
endelse
print,num_channels,num_measurements
outpars = make_array(num_channels,11,/float,value=0.0)
outkern = make_array(num_channels,32,/float,value=0.0)
a = make_array(5,/double,value=0.0)
readcol,'C:\Kepler\KSO\rec\undershoot\channel_list.txt',channel,module,output
openw,lunw,'C:\Kepler\KSO\rec\undershoot\' + basename + '_three_tau_out.txt',/get_lun
openw,lunw2,'C:\Kepler\KSO\rec\undershoot\' + basename + '_three_tau_kernel.txt',/get_lun
print,lunw,'Best Fit Parameters'
printf,lunw,ver
printf,lunw,systime()
printf,lunw, 'Jeffrey Van Cleve, BATC'
printf,lunw2,ver
printf,lunw2,systime()
printf,lunw2, 'Jeffrey Van Cleve, BATC'

color_table = 39
device,decomposed=0
loadct, color_table

for channel_index = 0, num_channels - 1 do begin
  if (keyword_set(debug) eq 0) then timeseries =
      image_structure.field01[channel_index,12:31]
  if keyword_set(debug) then timeseries = image_structure.field1[12:31]
  ;calculate initial guesses for fit parameters
  a[0] = timeseries[0]
  a[1] = alog(timeseries[1]/timeseries[0])
  a[2] = timeseries[10]*exp(1)
  a[3] = 0.1*alog(timeseries[13]/timeseries[3])
  if (a[1] le -0.5) then begin
    ;if normal, set initial delta-function term to 0
    a[4] = 0
  endif else begin
    ;if anomalous, set short time constant to median of normal channels and choose delta-function term to give amplitude 0 at pixel 0
    a[1] = -1.3
```
a[4] = -1*timeseries[0]
endelse

channel_string = strcompress(string(round(module[channel_index])),/remove_all) + '.'
+
strcompress(string(round(output[channel_index])),/remove_all)
print,'Input parameters for channel ',round(channel_index),' ',channel_string
print,format='(5(E14.4))',a
fit = CURVEFIT(findgen(20), timeseries, Weights, a, Sigma, CHISQ=chisq, /DOUBLE, $ 
FUNCTION_NAME='threetau')
outpars[channel_index,0:4] = transpose(a)
outpars[channel_index,5:9] = transpose(Sigma)
outpars[channel_index,10] = chisq
temp = make_array(32,/float,value=1.0)
temp[12:31] = fit
outkern[channel_index,*] = shift(temp,1) - temp
print,'Output parameters'
print,format='(5(E14.4))',a

window,0,xsize=500,ysize=320
plot,findgen(20)+1112,timeseries,ba=color,co=0,yrange=[fit[0] < fit[1],0],TITLE='Undershoot, short time constant scale, channel ' + 
channel_string,XTITLE='column',YTITLE='undershoot', psym = 5
oplot,findgen(20)+1112,fit,color=128
window,1,xsize=500,ysize=320
plot,findgen(20)+1112,timeseries,ba=color,co=0,yrange=[-1e-03,0],TITLE='Undershoot, long time constant scale, channel ' + 
channel_string,XTITLE='column',YTITLE='undershoot', psym = 5
oplot,findgen(20)+1112,fit,color=128
endfor

out_format = '(' + strcompress(string(num_channels),/remove_all) + '(E14.4))'
printf,lunw,format=out_format,outpars

out_format2 = '(' + strcompress(string(num_channels),/remove_all) + '(E15.5))'
printf,lunw2,format=out_format2,outkern

close,/all
print,'Done'
end

7.3 kernel invert

pro kernel_invert,B,B_inv
; NAME:
;
; PURPOSE: invert undershoot kernel
; REFERENCE DOCUMENT: KADN-26197
;
; CALLING SEQUENCE:
; INPUTS:
;   1. Forward kernel values, either output of threetau_fit or Excel-->text file of same format
;   2. Channel list table for mapping 0,1,2...83 onto 2.1,2.2,2.3,...24,4
;
; Optional Keyword Parameters
;
; OUTPUTS:
;   1. Forward and Reverse transformation matrices are returned as arrays
;   2. The reverse coefficients are written to the _reverse.txt output file
;
; MODIFICATION HISTORY:
;   Name, Org, Date
;
; Jeffrey Van Cleve, BATC 7/24/1008

dim = 128
midrow = floor(dim/2)
cd,'C:\Kepler\KSO\rec\undershoot'
input_file = DIALOG_PICKFILE(/MUST_EXIST, TITLE='Select *.txt file containing forward kernel and errors ','FILTER='*kern*.txt')
basename = FILE_BASENAME(input_file,'.txt')
outname = basename + '_reverse.txt'
openw,lunw,outname,/get_lun
image_structure = read_ascii(input_file, data_start = 1)
readcol,'C:\Kepler\KSO\rec\undershoot\channel_list.txt',channel,module,output
s = size(image_structure.field01)
num_channels = s[1]
outarr = make_array(num_channels,21,/float,value=0.0)
for i = 0, num_channels - 1 do begin
   B = make_array(dim,dim,/float,value = 0.0)
   kernel = image_structure.field01[i,0:19]
   for row = 0, dim - 1 do begin
      B[0 > (row - 19):row,row] = reverse(kernel[0:row < 19])
   endfor
   B_inv = INVERT(B,status,/double)
   outarr[i,0] = status
   reverse_kernel = reverse(B_inv[midrow-19:midrow,midrow])
   outarr[i,1:20] = transpose(reverse_kernel)
endfor
out_format = '(' + strcompress(string(num_channels),/remove_all) + '(E14.4))'
printf,lunw,format=out_format,outarr
close,/all
7.4 correct undershoot

pro correct_undershoot

; NAME:
;
; PURPOSE: Use reverse kernel parameters to correct observed undershoot and show residuals
;
; REFERENCE DOCUMENT: KADN-26197
;
; CALLING SEQUENCE:
;
; INPUTS:
; 1. Reverse Kernel file, output of kernel_invert, one header line, 21 lines (for each of
; the 20 nonzero kernel values + leading value for matrix inversion status)
num_channels columns.
; 2. Experimental Pipeline Charge injection output table which has 1 header line, 32 lines (for each of
; the 32 trailing columns num_channels columns.
; 3. Channel list table for mapping 0,1,2...83 onto 2.1,2.2,2.3,...24,4
;
; Optional Keyword Parameters
;
; OUTPUTS:
; 1. Run time plots
; 2. Residual file _corrected.txt
;
; MODIFICATION HISTORY:
;   Name, Org, Date, Description
;Jeffrey Van Cleve   BATC/KSO    7/27/08    Created

cd,'C:\Kepler\KSO\rec\undershoot'
kern_length=0.625

kernel_file = DIALOG_PICKFILE(/MUST_EXIST, TITLE='Select *.txt file containing reverse
kernel and errors ',FILTER='*kern*.txt')
input_file = DIALOG_PICKFILE(/MUST_EXIST, TITLE='Select *.txt file containing observed
data ',FILTER='*ci*.txt')
basename = FILE_BASENAME(input_file,'.txt')
outname = basename + '_corrected.txt'

openw,lunw,outname,/get_lun
printf,lunw,systime()
printf,lunw,'Jeffrey Van Cleve, BATC/KSO 7/27/08 Created

observed_data = read_ascii(input_file, data_start = 1)
kernel = read_ascii(kernel_file, data_start = 1)

loadct, color_table
s = size(observed_data.field01)
num_channels = s[1]
outarr = make_array(num_channels,24,/float,value=0.0)

for i = 0, 83 do begin
  channel_string = strcompress(string(round(module[i])),/remove_all) + '.' + strcompress(string(round(output[i])),/remove_all)
padded_observed_data = make_array(128,/float,value = 1.0)
padded_observed_data[50:81] = observed_data.field01[i,*]
padded_observed_data[82:127] = 0
window,0,xsize=500,ysize=320
filter,padded_observed_data,kernel.field01[i,1:20],corrected_data
plot, corrected_data,ba=!color,co=0,xrange=[50,90],yrange=[-0.0002,0.0002],TITLE=channel_string
outarr[i,*] = corrected_data[60:83]
wait, 0.5
endfor

out_format = '(' + strcompress(string(num_channels),/remove_all) + '(E14.4))'
printf,lunw,format=out_format,outarr
close,/all
end

pro filter,observed,kernel,corrected
num_input = n_elements(observed)
num_filter = n_elements(kernel)
corrected = make_array(num_input,/float,value = 0.0)
;loop over
for j = 1, num_input do begin
  k_max = num_filter < (j - 1)
  for k = 1, k_max do begin
    corrected[j-1] = corrected[j-1] + kernel[k-1]*observed[j-k]
  endfor
endfor
end

7.5 undershoot_output_format

pro undershoot_output_format
mjd = strcompress(string(systime(/julian,/utc)-2400000),/remove_all)
cd,'C:\Kepler\KSO\rec\undershoot'
input_file = DIALOG_PICKFILE(/MUST_EXIST, TITLE='Select *.tbl file containing forward kernel and errors',FILTER='*kern*.tbl')
basename = FILE_BASENAME(input_file,'.tbl')
outname = 'undershoot.txt'
openw,lunw,outname,/get_lun
image_structure = read_ascii(input_file, data_start = 1)
readcol,'C:\Kepler\KSO\rec\undershoot\channel_list.txt',channel,module,output
s = size(image_structure.field01)
num_channels = s[1]
num_measurements = s[2]
for i = 0, 83 do begin
  if (module[i] ge 10) then begin
    module_string = strcompress(string(round(module[i])),/remove_all)
  end else begin
    module_string = '0'+ strcompress(string(round(module[i])),/remove_all)
  endelse
  output_string = strcompress(string(round(output[i])),/remove_all)
  line_string = mjd + '|' + module_string + '|' + output_string + '|1|1|0|20'
  start_string_length = strcompress(strlen(line_string),/remove_all)
  print,format='($,(A' + start_string_length + '))',line_string
  printf,lunw,format='($,(A' + start_string_length + '))',line_string
endfor

for j = 0, 39 do begin
  print,format='($,(A1),(E12.4))','|',image_structure.field01[i,j]
  printf,lunw,format='($,(A1),(E12.4))','|',image_structure.field01[i,j]
endfor

print
printf,lunw

close,/all
end