A framework for propagation of uncertainties in the Kepler data analysis pipeline

Bruce D. Clarke*[a], Christopher Allen[b], Stephen T. Bryson[c], Douglas A. Caldwell[a], Hema Chandrasekaran[a], Miles T. Cote[c], Forrest Girouard[b],Jon M. Jenkins[a], Todd C. Klaus[b], Jie Li[a], Chris Middour[b], Sean McCauliff[b], Elisa V. Quintana[a], Peter Tenenbaum[a], Joseph D. Twicken[a], Bill Wohler[b], Hayley Wu[a]

[a]SETI Institute/NASA Ames Research Center, M/S 244-30, Moffett Field, CA, USA 94305;
[b]Orbital Sciences Corporation/NASA Ames Research Center, M/S 244-30, Moffett Field, CA, USA 94305;
[c]NASA Ames Research Center, M/S 244-30, Moffett Field, CA, USA 94305

ABSTRACT

The Kepler space telescope is designed to detect Earth-like planets around Sun-like stars using transit photometry by simultaneously observing 100,000 stellar targets nearly continuously over a three and a half year period. The 96-megapixel focal plane consists of 42 charge-coupled devices (CCD) each containing two 1024 x 1100 pixel arrays. Cross-correlations between calibrated pixels are introduced by common calibrations performed on each CCD requiring downstream data products access to the calibrated pixel covariance matrix in order to properly estimate uncertainties. The prohibitively large covariance matrices corresponding to the ~75,000 calibrated pixels per CCD preclude calculating and storing the covariance in standard lock-step fashion. We present a novel framework used to implement standard propagation of uncertainties (POU) in the Kepler Science Operations Center (SOC) data processing pipeline. The POU framework captures the variance of the raw pixel data and the kernel of each subsequent calibration transformation allowing the full covariance matrix of any subset of calibrated pixels to be recalled on-the-fly at any step in the calibration process. Singular value decomposition (SVD) is used to compress and low-pass filter the raw uncertainty data as well as any data dependent kernels. The combination of POU framework and SVD compression provide downstream consumers of the calibrated pixel data access to the full covariance matrix of any subset of the calibrated pixels traceable to pixel level measurement uncertainties without having to store, retrieve and operate on prohibitively large covariance matrices. We describe the POU Framework and SVD compression scheme and its implementation in the Kepler SOC pipeline.

Keywords: Kepler, calibration, covariance, compression, framework, MATLAB.

1. INTRODUCTION

On March 6, 2009 at 10:50pm EST, the Kepler spacecraft was launched aboard a Delta II rocket from NASA's Kennedy Space Center at Cape Canaveral, Florida beginning an historic search for habitable planets orbiting distant stars. From an Earth-trailing orbit around the sun, Kepler will continuously observe approximately 160,000 stars in our galaxy over a three and one-half year period.[1] The field of view is 105-square degrees in the Cygnus and Lyra constellations. The science instrument is a photometer designed to detect variations in starlight as small as 100 parts-

* bruce.d.clarke@nasa.gov; kepler.nasa.gov

[1] There are enough consumables onboard to support observations for at least six years. Since the science instrument and spacecraft are currently performing to the design requirements, there is an extended mission proposal in process to fund an additional two years observing.

per-million; the amount of dimming expected as an Earth sized planet passes in front of a Sun-like star. Kepler will observe these extra-solar transits using a 0.95-meter Schmidt telescope which provides a slightly defocused image of the field of on a 95-megapixel focal plane consisting of forty-two 2200 x 1024 pixel charge-coupled devices (CCD). Each CCD consists of two separate outputs providing 84 separate 1100 x 1024 pixel array channels across the focal plane. The slight defocusing reduces the photometric noise by spreading the light from any one particular star over typically about one hundred pixels. In order to prevent saturation, the pixel data are read out every 6.539 seconds. Two-hundred and seventy reads are accumulated and time tagged producing the "long cadence" data, the primary science data for the transit search mission. Data at nine-read accumulations, called "short cadence" data, is also produced and made available for other studies such as asteroseismology. The long cadence sampling interval is 29.42 minutes while the short cadence interval is 58.85 seconds (1/30 long cadences).[1] Communications bandwidth constraints, lack of an articulating high gain antenna, limited onboard data storage capacity and spacecraft operations combine to preclude down linking the readout from the entire focal plane on the long and short cadence intervals.[2] The pixel data for only the selected 160,000 long cadence and 512 short cadence target stars is stored onboard and transmitted to the ground once a month through the Deep Space Network (DSN) and stored at NASA's Data Management Center (DMC). This raw pixel data, which represents about 6% of the focal plane image, is then available from the DMC for processing through the data analysis pipeline operated and maintained at the Kepler Science Operations Center (SOC) at the NASA Ames Research Center in Mountain View, California.

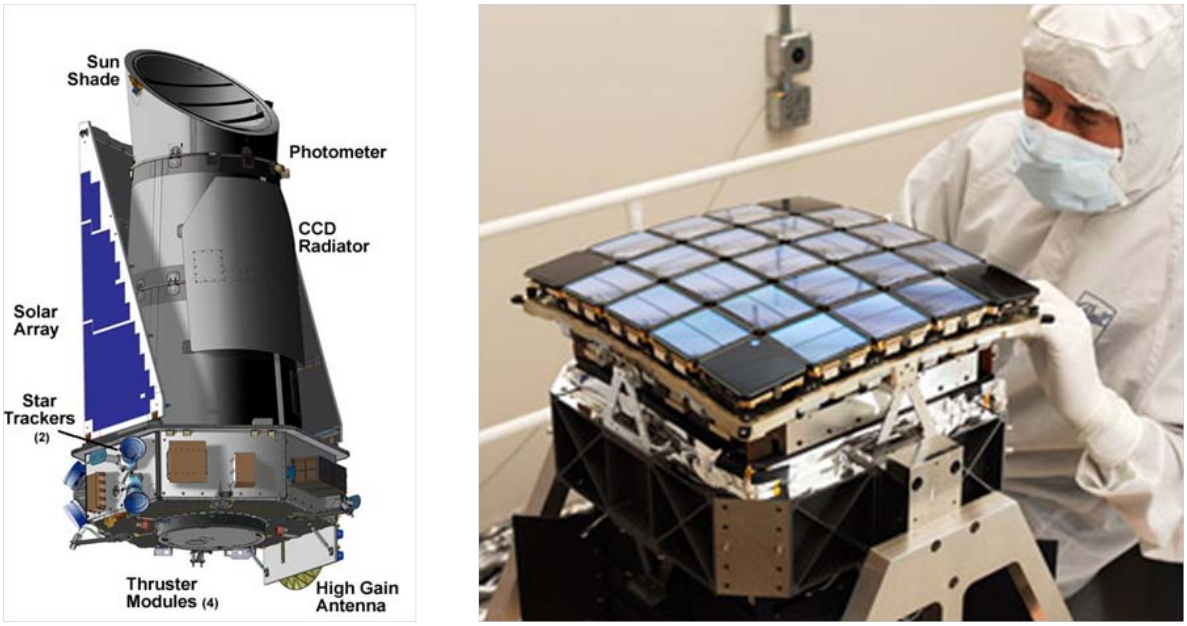

Figure 1– Left - Drawing of Kepler spacecraft some of the external components shown. Right – the focal plane assembly prior to integration into the photometer at Ball Aerospace in Boulder, Colorado. The focal plane is mounted inside the photometer at the location of the CCD radiator. [1]

2. THE SOC DATA ANALYSIS PIPELINE

The SOC pipeline ingests the raw pixel data received from the DMC and produces calibrated pixels, target flux (calibrated pixel aggregates), diagnostic metrics to assess photometer performance and a list of planetary candidates, each with an associated report. The analysis segment consists of six Computer Software Configuration Items (CSCIs) – Calibration (CAL), Photometric Analysis (PA), Pre-Search Data Conditioning (PDC), Transiting Planet

[2] A few full frame images (FFI) at long cadence accumulations are stored and down linked at the beginning of each month. These are used by the Kepler Science Office to verify target pixel assignments and to analyze image artifacts.

Search (TPS), Data Validation (DV) and Photometer Data Quality (PDQ).[ii] The monthly data flows through the CAL-PA-PDC-TPS-DV pipeline in a serial fashion, one channel at a time. CAL performs the pixel level calibration, PA uses the calibrated pixels to estimate and remove background flux and produce calibrated flux time series, PDC assesses and corrects artifacts from the calibrated flux time series, TPS flags suspected transit events in the corrected flux time series and generates a list of planet candidates and DV fits transit models to the flagged events and generates a report for each planet candidate. Every three months a quarterly pipeline run is performed on the three preceding months of data in order to eliminate edge effects inherent in segmenting the data on a monthly basis. PDQ is a stand-alone hybrid version of the CAL-PA segment with the added functionality of attitude determination. It operates on only a few select targets per channel but over the full focal plane at once and provides photometer performance metrics and a spacecraft pointing solution every few days based on a small set of reference pixels.[iii]

Data is passed between CSCIs using software written in JAVA. The algorithms internal to the CSCIs are implemented in compiled MATLAB.

3. PIXEL LEVEL CALIBRATION

Calibration applies a scale factor and removes known instrument and image artifacts converting raw pixel data in analog-to-digital-units (ADU) per cadence to photoelectrons (e-) per cadence. Known artifacts include; static and dynamic pixel biases which are mitigated by the 2D and dynamic 1D black corrections, nonlinear step response of the analog readout electronics which is mitigated by the undershoot correction, nonlinear static response of the readout amplifiers and analog-to-digital converters which is mitigated by the nonlinearity correction, variations in pixel sensitivity across the focal plane which are mitigated by the flat field correction, smearing of the image during readout due to shutter-less operation which is mitigated by the smear correction, and a constant bias current inherent to CCDs which is mitigated by the dark current correction.[iv] Some of these calibrations are based on models determined from pre-launch data and some are based on collateral data collected along with the photometric science data.

CAL estimates uncertainties for the calibrated pixels. The raw pixel variance is estimated as the quadrature sum of the modeled read noise, the calculated shot noise and the effective quantization noise. Each pixel read is assumed to be independent and the raw uncertainty is simply the square root of the variance. During the calibration process, the common corrections applied across large subsets of pixels introduce correlations. These correlations must be estimated in order to correctly estimate the uncertainties. This means the pipeline must propagate the full covariance matrix through the calibration operations.

Collateral data calibration

The object of collateral data calibration is to produce the dynamic 1D black, smear level and dark level corrections for use in the photometric pixel calibrations. Collateral data consists of masked smear, virtual smear and trailing black pixels (Figure 4). The CAL collateral pixel processing steps are; 1) 2D black correction - subtract static 2d black model from trailing black, masked and virtual smear, 2) estimate dynamic 1D black from trailing black columns, 3) dynamic 1D black correction - subtract dynamic 1D black estimate from black, masked and virtual smear, 4) undershoot correction - apply undershoot filter per row to the smear pixels, 5) nonlinearity correction - multiply smear pixels by nonlinearity model, 6) gain correction - multiply smear pixels by gain model and 7) estimate smear correction and dark current correction from the calibrated masked and virtual smear pixels.

The two largest contributors to calibration induced correlations are the smear correction and 1D black correction which are both estimated from the collateral data. The smear correction correlates pixels which share a common column on the CCD, the dynamic 1D black correction ties together pixels which share a common row.

Photometric data calibration

Photometric data consists of pixels from the exposed silicon region. The pixels near stars of interest are called target pixels (used to measure the light output from a particular target star) and the pixels that fall in the space between stars are the background pixels (used to estimate the amount of background light). The calibrations applied to the target and background pixels include those applied to the collateral data plus smear and dark level corrections which are estimated from the collateral data[v] and the flat field corrections which is estimated from pre-launch test data. The CAL photometric pixels processing steps are; 1) 2D black correction - subtract static 2d black model from target and

background pixels, 2) dynamic 1D black correction – subtract dynamic 1D black estimate, 3) undershoot correction - apply undershoot filter per row, 4) nonlinearity correction- multiply pixels by nonlinearity model, 5) gain correction - multiply pixels by gain model, 6) smear level correction – subtract smear level, 7) dark level correction – subtract dark level and 8) flat field correction – divide pixels by the flat field model.

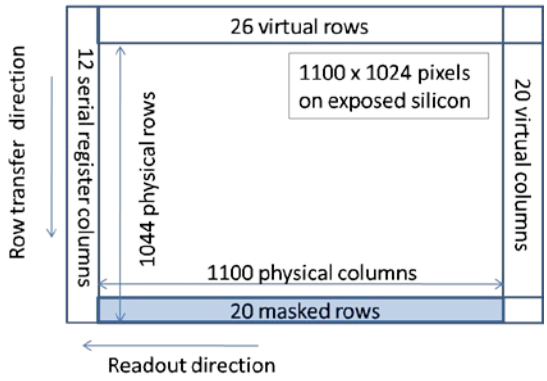

Figure 2 – Row and column layout one CCD channel. The pixels are pulled down row by row and read out pixel by pixel to the left. Row and column dimensions of the readout include the 1100 x 1024 photometric pixel array on exposed silicon plus the collateral pixels which are 20 rows masked from exposure to light by a thin layer of aluminum, 26 virtual (over-clocked) rows which do not exist as physical silicon, 20 virtual columns at the end of each row and 12 serial register columns (also virtual columns) at the beginning of each row. The effective array size is 1070 rows x 1132 columns. Masked and virtual rows are used to measure the smear of the image and are referred to as "masked" and "virtual" smear pixels. Virtual columns at the end of the readout are used to estimate pixel biases per cadence. They are referred to as the "trailing black" pixels. The serial register columns at the start of the row readout could also be used to estimate the black pixel value and are called the "leading black" pixels.

4. THE COVARIANCE PROBLEM AND A NOVEL SOLUTION

The covariance problem

The traditional method of propagating covariance is to transform the covariance matrix alongside the underlying data. This conventional lock-step method is not viable in the SOC pipeline since the memory required for both operating on and storing the covariance matrix for the full set of calibrated pixels are beyond the processing and storage capabilities of the pipeline.

CAL processes up to 75000 target and background pixels plus 3400 collateral pixels for each of the 84 channels. A double precision representation of the associated covariance matrix would require 22.5 gigabytes/cadence for storage. Furthermore, it is not possible to hold a 75000 x 75000 element covariance matrix in RAM in order to operate on it without using a segmenting scheme which would dramatically increase the CAL run time. Finally, CAL is required to process three months (one quarter) of long cadence data at a time which would require a whopping 100 terabytes/channel/quarter, or 8400 terabytes/quarter for the entire focal plane for storage of the pixel covariance alone. For comparison, storage of the calibrated pixel data and the associated uncertainties for the entire focal plane for one quarter requires about half a terabyte.

A novel solution

Rather than propagate the full covariance matrix at each step, the SOC employs a custom data structure storing only the uncorrelated primitive data and the small metadata kernel of each transformation. Each calibration transformation and its corresponding Jacobian matrix are reconstructed as needed.[3] The covariance matrix of the

[3] The Jacobian alone is sufficient to propagate the covariance matrix however; some of the transformations involved in calibration are functions of the underlying data, that is, they are nonlinear, and so are the Jacobians. In order to minimize the amount of metadata stored, the underlying transformations must also be reconstructed in order to provide the underlying data at any point in the transformation chain.

calibrated pixel products is produced by cascading the primitive covariance through the reconstructed chain of transformations.

The custom data structure reduces the amount of memory needed to propagate covariance by no more than a factor of 78000. But CAL is required to process up to three months of a single channel of long cadence data at a time. For that unit of work, even the relatively small set of primitive data plus metadata needed turns out to be quite large so a data compression scheme is employed. Some of the metadata is compressible across cadences in a lossless fashion. The primitive data and metadata derived from the primitive data are compressible across cadences using singular value decomposition (SVD)[vi]. Since we expect the primitive data and metadata to be slowly varying function of time, selecting only the highest power components of the SVD eliminates outliers which occur at spurious high frequencies. Truncating the SVD in this way not only dramatically reduces the number of bytes to store but provides low pass filtering. The cost of the SVD compression is a tolerable loss of fidelity in covariance elements of less than a part in 10^4. The Akaike Information Criteria is used on a per pixel basis to determine the appropriate number of SVD components to use without over-fitting.[vii]

The combination of this novel Propagation of Uncertainties (POU) framework and SVD data compression allows the SOC pipeline to provide full covariance information of the calibrated products, traceable to the raw pixel uncertainties.

5. STANDARD PROPAGATION OF UNCERTAINTIES

The pixel level calibrations are a series of transformations on multidimensional data converting an input vector of raw pixels to an output vector of calibrated pixels. We propagate the covariance in the spatial using "standard propagation of uncertainties" (POU) in which the Jacobian of each transformation at $x = x_0$ is approximated as the first order Taylor expansion in $x - x_0$.[viii] The following matrix formulation illustrates the chain of multidimensional transformations on the underlying data and the corresponding transformation of the covariance using the Jacobians.

Standard POU formalism

Given a functional relationship (f) between an input vector (x_0) and an output vector (y_0):

$$\boldsymbol{x_0} = [x_1\ x_2\ x_3\ \dots\ x_n\ ]^{\mathrm{T}} \qquad \boldsymbol{y_0} = [y_1\ \ y_2\ y_3\ \dots y_m\ ]^{\mathrm{T}}$$

$$\boldsymbol{y_0} = f(\boldsymbol{x_0}) = \begin{bmatrix} f_1(x_0) \\ f_2(x_0) \\ \vdots \\ f_m(x_0) \end{bmatrix} \tag{1}$$

The transformation, $T(x_0)$, may be written as:

$$\mathbf{T}(x_0) = \begin{bmatrix} \mathrm{T_{11}} & \mathrm{T_{12}} & \cdots & \mathrm{T_{1n}} \\ \mathrm{T_{21}} & \mathrm{T_{22}} & \cdots & \mathrm{T_{2n}} \\ \vdots & \vdots & \cdots & \vdots \\ \mathrm{T_{m1}} & \mathrm{T_{m2}} & \cdots & \mathrm{T_{mn}} \end{bmatrix} \quad \text{where: } f_i(x_0) = \sum_{k=1}^{n} \mathrm{T}_{ik} \cdot x_{0_k} \tag{2}$$

And the Jacobian of the transformation, $J(x_0)$, may be written as:

$$\mathbf{J}(\boldsymbol{x_0}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{3}$$

Such that the underlying data and corresponding covariance transform like:

$$\mathbf{y_0} = \mathbf{T}(\boldsymbol{x_0}) \cdot \boldsymbol{x_0} \qquad \mathbf{C}_{y_0} = \mathbf{J}(\boldsymbol{x_0}) \cdot \mathbf{C}_{x_0} \cdot \mathbf{J}(\boldsymbol{x_0})^{\mathrm{T}} \tag{4}$$

Since the output of one calibration operation is the input to the next, the result of a chain of j calibration operations taking x_0 to z_0 can be written:

$$\boldsymbol{z_0} = \mathbf{T_j}(\boldsymbol{x_j}) \cdot \cdots \mathbf{T_2}(\boldsymbol{x_2}) \cdot \mathbf{T_1}(\boldsymbol{x_1}) \cdot \mathbf{T_0}(\boldsymbol{x_0}) \cdot \boldsymbol{x_0} \tag{5}$$

With the propagated covariance is given by:

$$\mathbf{C}_{z_0} = \mathbf{J_j}(\boldsymbol{x_j}) \cdot \cdots \cdot \mathbf{J_2}(\boldsymbol{x_2}) \cdot \mathbf{J_1}(\boldsymbol{x_1}) \cdot \mathbf{J_0}(\boldsymbol{x_0}) \cdot \mathbf{C}_{x_0} \cdot \mathbf{J_0}(\boldsymbol{x_0})^{\mathrm{T}} \cdot \mathbf{J_1}(\boldsymbol{x_1})^{\mathrm{T}} \cdot \mathbf{J_2}(\boldsymbol{x_2})^{\mathrm{T}} \cdot \cdots \cdot \mathbf{J_j}(\boldsymbol{x_j})^{\mathrm{T}} \tag{8}$$

Implementing standard POU in CAL

The operations in CAL are represented by a small set of simple transformations, each with small kernels. The Jacobian matrix (J) follows directly from the transformation matrix (T) by applying equations (2) and (3). We present several of these simple transformations and associated Jacobian matrices below.[ix] Table 1 lists the supported types. The MATLAB equivalents to the matrix multiplications are also shown.

Scale an input vector by a scalar constant (k) – transformType = 'scale'

$$T = \begin{bmatrix} k & 0 & \cdots & 0 \\ 0 & k & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & k \end{bmatrix} \qquad J = \begin{bmatrix} k & 0 & \cdots & 0 \\ 0 & k & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & k \end{bmatrix}$$

$$z = T \cdot x = k .* x$$

$$C_z = J \cdot C_x \cdot J^T = k^2 .* C_x$$

Scale an input vector by a vector constant (v) – transformType = 'scaleV'

$$T = \begin{bmatrix} v_1 & 0 & \cdots & 0 \\ 0 & v_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & v_n \end{bmatrix} \qquad J = \begin{bmatrix} v_1 & 0 & \cdots & 0 \\ 0 & v_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & v_n \end{bmatrix}$$

$$z = T \cdot x = v .* x$$

$$C_z = J \cdot C_x \cdot J^T = scalecol\left(v, scalerow\left(v, C_x\right)\right)$$

Add or subtract two input vectors element by element - transformType = 'addV' or 'diffV'

Table 1 - Complete list of transformations supported by the POU framework as of SOC release 6.1

	transformType	Definition	MATLAB notation
1	scale	Scale variable vector by a constant	z = k.* x
2	scaleV	Scale variable vector by a constant vector	z = v.* x
3	addV	Add two variable vectors	z = x + y
4	diffV	Subtract two variable vectors	z = x – y
5	multV	Multiply two variable vectors	z = x .* y
6	divV	Divide two variable vectors	z = x ./ y
7	wSum	Weighted sum of elements in a variable vector	z = sum(w .* x)
8	wMean	Weighted mean of elements in a variable vector	z = mean(w .* x)
9	bin	Bin the elements of a variable vector	z(i)z = sum(x(j:k))
10	wPoly	Apply a weighted polynomial design matrix	z = scalecol(w, M) * x
11	filter	Filter the input data	z = filter(b, a, x)
12	userM	User defined matrix transformation	z = M * x
13	selectIndex	Select a subset of input data based on index	z = x(index, :)
14	concatRows	Concatenate data row-wise	z = [x; y]
15	fillRows	Fill rows in x at index with rows of y	z = x; x(index) = y
16	eye	Identity transformation – Start of transform chain	z = I * x

$$T = \begin{bmatrix} 1 & 0 & \cdots & 0 & \pm 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \pm 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & \pm 1 \end{bmatrix} \qquad J = \begin{bmatrix} 1 & 0 & \cdots & 0 & \pm 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \pm 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & \pm 1 \end{bmatrix}$$

$$z = T \cdot \begin{bmatrix} x \\ y \end{bmatrix} = x \pm y$$

$$C_z = J \cdot \begin{bmatrix} C_x & 0 \\ 0 & C_y \end{bmatrix} \cdot J^T = C_x + C_y$$

Multiply two input vectors element by element - transformType = ‘multV’

$$T = \frac{1}{2}\begin{bmatrix} y_1 & 0 & \cdots & 0 & x_1 & 0 & \cdots & 0 \\ 0 & y_2 & \cdots & 0 & 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & y_n & 0 & 0 & \cdots & x_n \end{bmatrix} \qquad J = \begin{bmatrix} y_1 & 0 & \cdots & 0 & x_1 & 0 & \cdots & 0 \\ 0 & y_2 & \cdots & 0 & 0 & x_2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & y_n & 0 & 0 & \cdots & x_n \end{bmatrix}$$

$$z = T \cdot \begin{bmatrix} x \\ y \end{bmatrix} = x.*y$$

$$C_z = J \cdot \begin{bmatrix} C_x & 0 \\ 0 & C_y \end{bmatrix} \cdot J^T = scalecol(y, scalerow(y, C_x)) + scalecol(x, scalerow(x, C_y))$$

Divide two input vectors element by element - transformType = ‘divV’

$$T = \frac{1}{2}\begin{bmatrix} 1/y_1 & 0 & \cdots & 0 & x_1/{y_1}^2 & 0 & \cdots & 0 \\ 0 & 1/y_2 & \cdots & 0 & 0 & x_2/{y_2}^2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1/y_n & 0 & 0 & \cdots & x_n/{y_n}^2 \end{bmatrix}$$

$$J = \begin{bmatrix} 1/y_1 & 0 & \cdots & 0 & -x_1/{y_1}^2 & 0 & \cdots & 0 \\ 0 & 1/y_2 & \cdots & 0 & 0 & -x_2/{y_2}^2 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1/y_n & 0 & 0 & \cdots & -x_n/{y_n}^2 \end{bmatrix}$$

$$z = T \cdot \begin{bmatrix} x \\ y \end{bmatrix} = x./y$$

$$C_z = J \cdot \begin{bmatrix} C_x & 0 \\ 0 & C_y \end{bmatrix} \cdot J^T = scalecol(1./y, scalerow(1./y, C_x)) + scalecol(-x./y.\hat{}2, scalerow(-x./y.\hat{}2, C_y))$$

6. THE POU FRAMEWORK

The propagation of uncertainties (POU) framework is implemented in CAL as a MATLAB structure. The transformation chain is stored by first capturing the raw pixel data and its variance then the kernel of each transformation in turn. The framework expands as the chain grows to accommodate new transformations. We describe the data structure and associated functions to store transformations and retrieve transformed data.

Creating the POU structure

A 2D array (variables x cadences) of empty POU structures is initialized (called errorPropStruct in CAL) using the top level constructor. Column wise there is one array element for each CAL product plus several intermediate data products required for two input transformations – see Figure 5. The primitive data, its covariance, row and column index, indices gaps and cadence gaps are stored at the top level. Transformations are stored in the second level in the order they are applied. Metadata is stored in the third level. For the transformation types which take input of two vectors, the field yDataInputName is a pointer to another array element, allowing transformation chains to merge. Output of the constructors for the top level, second level and third level are shown below

```
empty_errorPropStruct() = …                 % top level POU array element
                      variableName: []                    % name of variable
                        xPrimitive: []                    % raw data
                       CxPrimitive: []                    % variance of raw data
                               row: []                    % ccd row indices if needed
                               col: []                    % ccd row indices if needed
                           gapList: []                    % index into xPrimitive
                     cadenceGapped: 0                     % boolean gap indicator
                 cadenceGapFilled: 0                      % Boolean fill indicator
                             size: [1x1 struct]           % used for compression scheme
                    originalShape: [1x1 struct]           % used for compression scheme
              transformStructArray: [1x1 struct]          % transformation array (chain)


empty_tStruct() = …                         % transformation array element
                      transformType: []                   % transform type
                       disableLevel: 0                    % apply transformation or not
                    yDataInputName: []                    % variable name of y-data
                          yIndices: []                    % indices of y-data
                             size: [1x1 struct]           % used for compression scheme
                    originalShape: [1x1 struct]           % used for compression scheme
              transformParamStruct: [1x1 struct]          % metadata for transform

empty_tParamStruct() = …                    % metadata for transform
                       scaleORweight: []                  % scalar or nx1 vector
                     filterCoeffs_b: []                   % filter coefficients
                     filterCoeffs_a: []
                          polyOrder: []                   % order of weighted poly
                        polyXvector: []                   % vector to evaluate poly
                           binSizes: []
                        FCmodelCall: []
                              userM: []                   % user defined matrix
                           xIndices: []                   % select indices
                             size: [1x1 struct]           % used for compression scheme
                    originalShape: [1x1 struct]           % used for compression scheme
```

Append transformations

The POU array elements are populated with primitive data and augmented with subsequent transformation metadata in lock-step with the calibrations in CAL. Primitive data is inserted using the 'eye' type. Subsequent transformations must be consistent with the supplied metadata. A 'disableLevel' parameter allows selective enabling of the base transformation and the Jacobian. This feature is useful in the case of the undershoot correction in which a filter transformation is applied to the underlying data but not to the propagated covariance.[x]

Cascade transformations

Given a column of POU array elements the transformation chain associated with any variable name may be executed drawing transformations from the structure as the chain progresses. Other array elements are propagated in recursive fashion as they are referenced as inputs to any transformations in the chain. The transformed primitive data and the propagated covariance matrix are returned.

7. COMPRESSION

The size of the fully populated POU structure for a typical channel is 25 gigabytes per quarter in the MATLAB workspace which compresses to about 11 gigabytes when stored. This is still about twice the size of the stored calibrated pixels and uncertainties. In order to minimize the load on the database, the array of POU structure elements is compressed in a two step process and stored in an array of compressed data structure elements.

Compression step 1

The 2D array of POU structure (variables x cadences) is minimized through a process of lossless concatenation and conversion to sparse representation. The 2D array of elements containing fields with 1D vectors are collapsed into a 1D array of elements containing fields with 2D arrays by first padding the 1D field arrays for consistency then concatenating across cadences. The original shape of the 1D array is saved in order to restore the original structure when the data is reconstituted. Common elements of these 2D field arrays are further condensed into 1D arrays where possible. Field arrays are converted to the MATLAB sparse representation where a savings is realized.

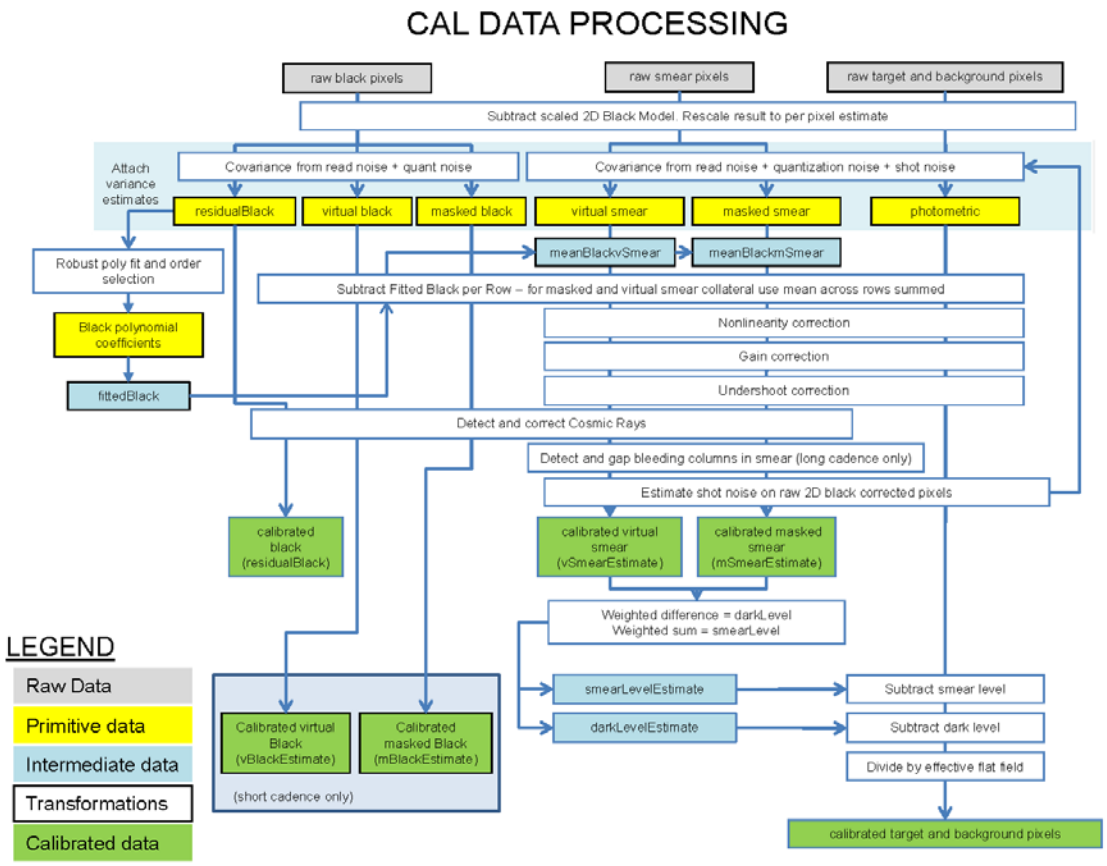

Figure 3 – Data flow diagram of operations performed in CAL. The variable names used in the POU structure are indicated on the diagram under 'calibrated data' and 'intermediate data'. The variable names are typically chosen to reflect the end product of the transformation chain. For example, 'vSmearEstimate' is the variable name for the transformation chain which begins with raw virtual smear pixels.

Compression step 2

The resulting 2D primitive data arrays and selected transformation metadata which closely resembles the primitive data are decomposed using singular value decomposition (SVD). A user defined number of the highest power components are selected and the resulting row and column basis vectors and the associated singular values for only these components are saved in a data structure. The original data is replaced with a pointer to the compressed data. The optimal model order as determined by the Akaike Information Criteria[xi] and the chi-square of the fit residual (residual power) is computed for each row. The model order selection prevents over fitting and the chi-square provides a metric on the fidelity of the reconstructed data.

Compressed data array element compressedData(3) is an example for 1132 virtual smear pixels over 476 cadences. The CxPrimitive and transformData fields also contain an SVD decomposition set of basis vectors (U,V), singular values (S) and the associated optimal model order and residual power vector. The top level corresponding element in the original POU structure array after compression is errorPropStruct(3). Primitive data has been replaced in this structure by the string 'SVD' indicating the data is stored in compressed form in a separate structure.

```
compressedData(3) = …
            variableName: 'vSmearEstimate'
              xPrimitive: [1x1 struct]              % primitive raw data
                       U: [476x10 double]          % SVD decomposition
```

```
                                S: [10x1 double]
                                V: [1132x10 double]
                          convFlag: 0                          % non-convergence flag
                  minimumAicOrder: [1132x1 double]             % optimal model order
                    residualPower: [1132x1 double]             % chi-squared

              CxPrimitive: [1x1 struct]                  % primitive variance
            transformData: [1x2 struct]                  % transform metadata

errorPropStruct(3) = …
             variableName: 'vSmearEstimate'
               xPrimitive: 'SVD'
              CxPrimitive: []
                      row: []
                      col: []
                  gapList: [1 2 3 4 5 6 7 8 9 10 11 12 1113 1114 1115 1116 1117 1118 1119 1120
1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132]
            cadenceGapped: [476x1 logical]
          cadenceGapFilled: [476x1 logical]
                      size: [1x1 struct]
            originalShape: [1x1 struct]
     transformStructArray: [7x1 struct]
```

8. RETRIEVING CALIBRATED PIXEL COVARIANCE

The minimized and compressed POU structure is passed between CSCIs in the pipeline and stored in the database as a Binary Large OBject (BLOB)[xii] allowing consumers of the calibrated pixels access to the covariance. One such consumer is the Photometric Analysis CSCI (PA) which estimates and removes the background photometric signal and aggregates the target pixels to form flux time series for each target.[xiii] The pixel covariance matrix for any subset of photometric pixels is retrieved for any given cadence given row and column indices, the corresponding POU structure array and a chunk size parameter used to throttle function performance.

The need for providing the covariance matrix as a measure of the pixel correlations is evident through examination of the covariance images as correlations up to several percent are noted. Figure 6 shows the location of 525 target and background pixels covering a 130 row x 160 column patch on one CCD and Figure 7 shows the calibrated pixel covariance for this set of pixels, retrieved from the CAL POU BLOB. The covariance is normalized to the median of the diagonal elements providing an upper limit on the correlation coefficients for the off-diagonal elements. Most of these off-diagonal elements are clustered around 0.42% with variations of 0.0003%. The banding in the row-sorted image identifies groups of pixels correlated at the 0.36% level. The column sorted equivalent image shows this banding to be localized over columns 439-442 and that the cross correlation of pixels within this band but not sharing a common column is close to the nominal 0.42%. The zoomed versions of the column-sorted images show the detail of the blocking along the diagonal indicating correlations as high as 3.6% with variations to 0.3% for most

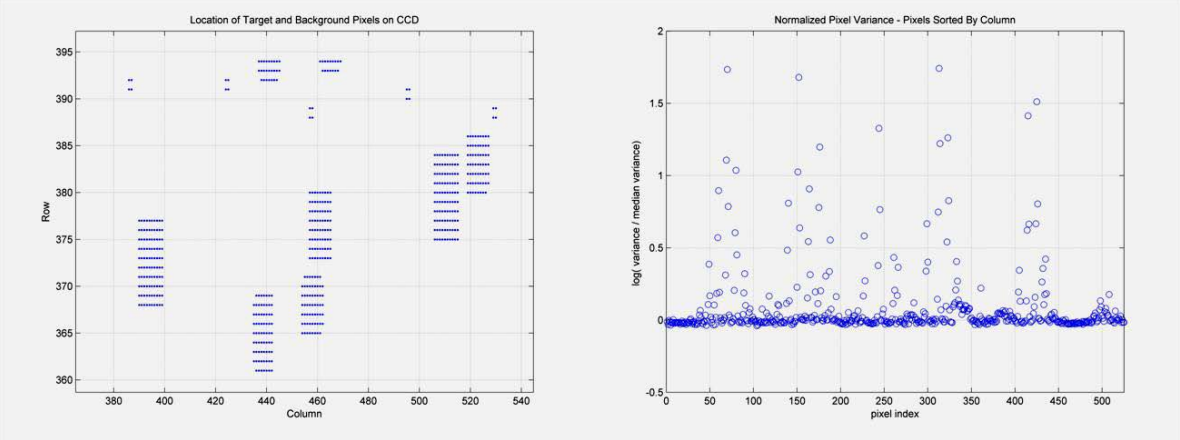

Figure 4 - Left – Location of pixels for eight targets and five groups of background pixels from quarter 1 long cadence flight data. Right – variance of these pixels normalized to the median.

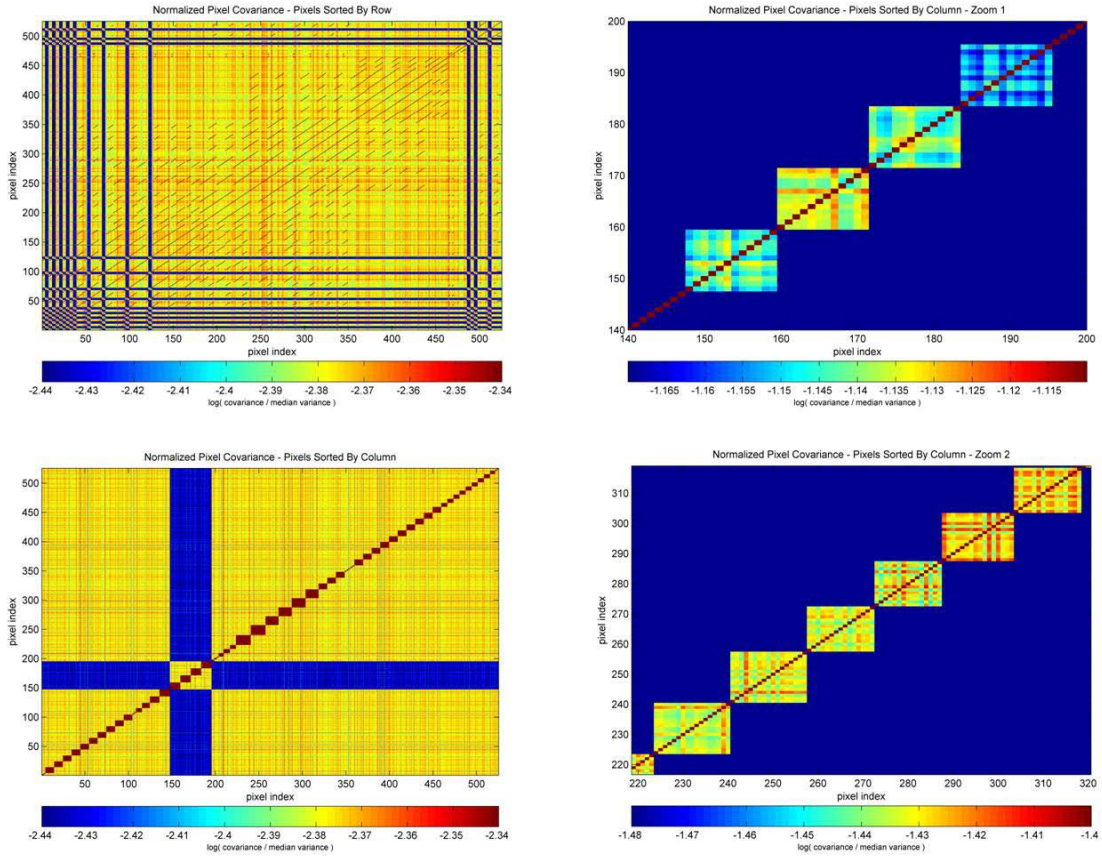

Figure 5 – Top left – row-sorted covariance normalized to median variance. Bottom left – column-sorted covariance normalized to median variance. Top left and bottom left – column sorted covariance normalized to median variance for two regions of interest in the bottom left image. The color axis limits have been adjusted on all images to enhance detail.

pixels sharing a common column and up to 7.2% with variations to 1% for those within the banded region. Speculation as to the source of the banding is beyond the scope of this paper.

9. CONCLUSIONS AND FORWARD WORK

The SOC implementation of the POU framework provides information about calibration induced pixel correlations traceable to the raw pixel uncertainties. The SVD compression scheme provides useful low pass filtering and makes propagation of the full covariance a tractable problem. The fidelity of the propagated uncertainties is preserved to better than 0.01%. Forward work includes expanding the POU framework to accept inputs from the background and target flux transformations in PA in order to evaluate target correlations based the propagated covariance. The structure should also be extended to accept a full primitive covariance matrix rather than simply a variance vector. Additionally, the POU framework should be implemented as a formal MATLAB class, for consistency with the rest of the pipeline code.

ACKNOWLEDGEMENTS

Funding for this work has been provided by the NASA Science Mission Directorate (SMD).

REFERENCES

[1] Koch, D.G., et al., "Kepler Mission design, realized photometric performance, and early science," Astrophysical Journal Letters, 713, L79-L86 (2010).

[2] Jenkins, J.M., et al., "Overview of the Kepler science processing pipeline," Astrophysical Journal Letters, 713, L87-L91 (2010).

[3] Haas, M.R., et al., "Kepler Science Operations," Astrophysical Journal Letters, 713, LXXX-LYYY (2010).

---

[i] [SPIE instrument paper, Doug Caldwell]

[ii] Middour, C., et al., "Kepler Science Operations Center architecture," Proc. SPIE, this volume (2010).

[iii] Klaus, T.C., et al., "The Kepler Science Operations Center pipeline framework," Proc. SPIE, this volume (2010).

[iv] Quintana, E.V., et al., "Pixel-level calibration in the Kepler Science Operations Center pipeline," Proc. SPIE, this volume (2010).

[v] Quintana, E.V., et al., "Pixel-level calibration in the Kepler Science Operations Center pipeline," Proc. SPIE, this volume (2010).

[vi] L.N. Trefethen and D. Bau, III, *Numerical Linear Algebra*, SIAM, 25-37, (1997).

[vii] Akaike, H., "A New Look at the Statistical Model Identification," IEEE Transactions on Automatic Control, AC-19 (6), 716-723 (1974).

[viii] [KADN-26185, POU in PDQ]

[ix] [KADN-26185, POU in PDQ]

[x] Quintana, E.V., et al., "Pixel-level calibration in the Kepler Science Operations Center pipeline," Proc. SPIE, this volume (2010).

[xi] Akaike, H., "A New Look at the Statistical Model Identification," IEEE Transactions on Automatic Control, AC-19 (6), 716-723 (1974).

[xii] [SPIE database/file store paper, Sean McCaulliff]

[xiii] Twicken, J.D., et al., "Photometric analysis in the Kepler Science Operations Center pipeline", Proc. SPIE, this volume (2010).