

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^d, 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

^aSETI Institute/NASA Ames Research Center, M/S 244-30, Moffett Field, CA, USA 94305;
^bOrbital Sciences Corporation/NASA Ames Research Center, M/S 244-30, Moffett Field, CA, USA 94305;

^cNASA Ames Research Center, M/S 244-30, Moffett Field, CA, USA 94305;

^dLawrence Livermore National Laboratory, P.O. Box 808, L-478, Livermore, CA, USA 94551

ABSTRACT

The *Kepler* space telescope is designed to detect Earth-like planets around Sun-like stars using transit photometry by simultaneously observing more than 100,000 stellar targets nearly continuously over a three-and-a-half year period. The 96.4-megapixel focal plane consists of 42 Charge-Coupled Devices (CCD), each containing two 1024 x 1100 pixel arrays. Since cross-correlations between calibrated pixels are introduced by common calibrations performed on each CCD, downstream data processing requires access to the calibrated pixel covariance matrix to properly estimate uncertainties. However, 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 filter the raw uncertainty data as well as any data-dependent kernels. This combination of POU framework and SVD compression allows the downstream consumer access to the full covariance matrix of any subset of the calibrated pixels which is traceable to the pixel-level measurement uncertainties, all 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, propagation of uncertainties, 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 more than 100,000 stars in our galaxy over a three-and-one-half-year period. The field of view is static, 115.6 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 per million, the amount of dimming expected as an Earth-size planet passes in front of a Sun-like star. *Kepler* will observe these extra-solar transits using a 0.95-meter Schmidt telescope and a 96-megapixel focal plane consisting of forty-two 1024 x 2200 pixel Charge-Coupled Devices (CCDs). Each CCD consists of two separate outputs providing 84 separate 1024 x 1100 pixel array channels across the focal plane. The telescope is slightly defocused to reduce the photometric noise by spreading the light from any one particular star over typically about thirty pixels. To prevent saturation, the pixel data are

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

read out every 6.54 seconds. Two hundred seventy reads are accumulated and time-tagged to produce “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 29.4 minutes, and the short cadence interval is 58.9 seconds (1/30 long cadence) [1]. Communications bandwidth constraints, lack of an articulating high-gain antenna, and limited onboard data storage capacity combine to preclude downlinking the readout from the entire focal plane at the long and short cadence intervals.¹ Pixel data for only the selected 156,000 long cadence² and 512 short cadence target stars is saved onboard, transmitted to the ground once a month through the Deep Space Network (DSN), and stored at the Space Telescope Science Institute (STScI). This raw pixel data, which represents about 6% of the focal plane image, is then available for processing through the data analysis pipeline operated and maintained by the *Kepler* Science Operations Center (SOC) at the NASA Ames Research Center at Moffett Field, California.

2. THE SOC DATA ANALYSIS PIPELINE

The SOC pipeline ingests raw pixel data received from the STScI 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 Search (TPS), Data Validation (DV), and Photometer Data Quality (PDQ) [2]. 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 to eliminate edge effects inherent in segmenting the data on a monthly basis. PDQ is a standalone 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 twice a week based on a small set of reference pixels [3].

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

The pixel-level calibration performed by CAL 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 shutterless operation, which is mitigated by the smear correction; and a constant bias current inherent to CCDs, which is mitigated by the dark current correction [4]. Some of these calibrations are based on models determined from pre-launch data and some are based on collateral data (see sub-section 3.1) collected along with the photometric science data.

CAL also 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 tracked to

¹ A few full-frame images 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.

² The number of long cadence targets observed at the start of the mission is approximately 156,000. Since the bandwidth of the high-gain communications decreases as the spacecraft drifts away from Earth and since science data cannot be collected while data is being downlinked, the program may decide to reduce the number of targets observed over time to as few as 100,000 at the end of three and one half years to maintain the same observational duty cycle throughout the mission.

correctly estimate the uncertainties of the calibrated products. This means the pipeline must propagate the full covariance matrix through each of the calibration operations.

3.1 Collateral data calibration

Collateral data consists of masked smear, virtual smear, and trailing black pixels (Figure 1). 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. The collateral pixel processing steps in CAL 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.

3.2 Photometric data calibration

Photometric data consists of pixels from the exposed silicon region. The pixels near stars of interest are called *target pixels* and are used to measure the light output from a particular target star. The pixels that fall in the space between stars are the *background pixels* and are used to estimate the amount of background light mixed in with each target. 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 [4], and the flat field correction, which is estimated from pre-launch test data. The CAL photometric pixel 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.

The two largest contributors to calibration-induced correlations in the photometric pixels are the smear level correction and the dynamic 1D black correction, both estimated from collateral data. The smear level correction correlates pixels that share a common column on the CCD; the dynamic 1D black correction correlates pixels that share a common row.

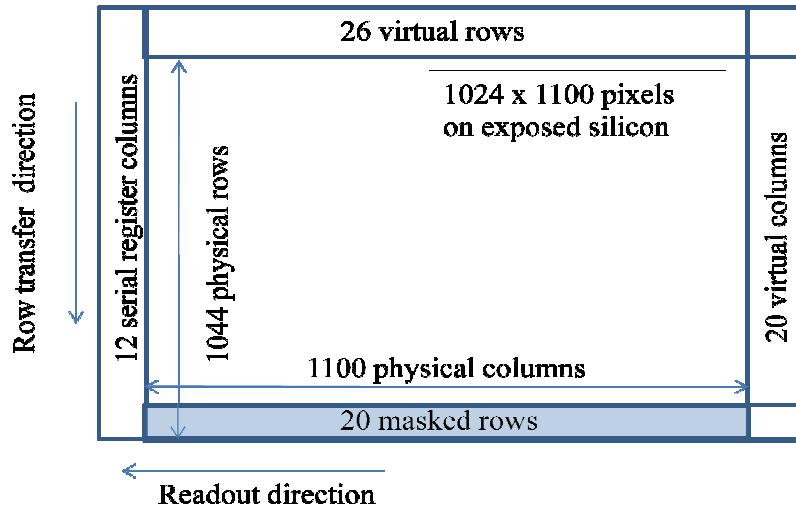


Figure 1. The row and column layout of one CCD channel is shown. During readout, the pixels are pulled down row by row and read out pixel by pixel to the left. Row and column dimensions of the readout include the 1024 x 1100 photometric pixel array on exposed silicon plus the collateral pixels – 20 rows 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 [5].

4. THE COVARIANCE PROBLEM AND A NOVEL SOLUTION

4.1 The covariance problem

The traditional method of propagating covariance is to transform the covariance matrix alongside the underlying data as each operation in the analysis sequence is performed. This conventional lock-step method of applying *Standard Propagation of Uncertainties* (see section 5) 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 is beyond the processing and storage capabilities of the pipeline.

For long cadence data, CAL processes up to 75,000 target and background pixels plus 3,400 collateral pixels for each of the 84 channels. Memory constraints mandate that the storage of any covariance information be limited to a small fraction of the memory needed for the underlying calibrated data. A double precision representation of the covariance matrix for just a single channel would require 22.5 gigabytes/cadence, 250 times the size of the calibrated data. Aside from the storage problem, it is not possible to hold a 75,000 x 75,000-element covariance matrix in RAM in order to operate on it. An elaborate segmenting scheme would be needed, which would dramatically increase the CAL run time. Ultimately, CAL is required to process three months (one quarter) of long cadence data at a time. Storage of the pixel covariance would require a whopping 100 terabytes/channel/quarter. For comparison, storage of the calibrated pixel data and the associated uncertainties for the entire focal plane for one quarter requires only about half a terabyte.

4.2 A novel solution

Rather than propagate the full covariance matrix at each step, the SOC applies standard Propagation of Uncertainties (POU) by employing a custom data structure which stores only the uncorrelated primitive data and the small metadata kernel of each transformation. Each calibration transformation and its corresponding Jacobian matrix³ are reconstructed as needed and the covariance matrix for any subset of the calibrated pixel products is produced by cascading the primitive covariance through the reconstructed chain of transformations.

The custom data structure solves part of the covariance problem; it reduces the amount of storage needed for propagation of covariance by no more than a factor of 78,000. 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 stored in the data structure turns out to be quite large, so a two-step data compression scheme is employed.

Some of the metadata is compressible across cadences in a lossless fashion. Additionally, the primitive data and metadata derived from the primitive data are compressible across cadences using Singular Value Decomposition (SVD) [6]. Since we expect the primitive data and metadata to be slowly varying functions 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 also provides low-pass filtering. Over-fitting is prevented by applying the Akaike Information Criteria (AIC) on a per-pixel basis to determine the appropriate number of SVD components to use in reconstructing the data [7].

This combination of the novel POU framework and SVD data compression has proven a viable solution to the covariance problem. Storage of the compressed data structure for a single channel requires only 40 kilobytes/cadence, about 4% of the storage required for the underlying calibrated data. Testing shows that covariance elements computed from decompressed data agree with those computed from uncompressed data to better than 0.01%, which is within SOC requirements for the propagation of uncertainties.

The POU framework enables the SOC pipeline to provide downstream consumers of the calibrated pixels full correlation information through the calibrated pixel covariance, traceable to the raw pixel uncertainties.

5. STANDARD PROPAGATION OF UNCERTAINTIES

The POU framework is an implementation of standard propagation of uncertainties. We review the standard POU formalism here.

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. 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.

5.1 Standard POU formalism

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 domain using standard POU in which the Jacobian of each transformation at $x = x_0$ is approximated as the first-order Taylor expansion in $x - x_0$ [7]. The following matrix formulation illustrates the chain of multidimensional transformations on the underlying data and the corresponding chain of transformations on the covariance using the Jacobians.

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

$$\begin{aligned} \mathbf{x}_0 &= [x_1 \ x_2 \ x_3 \ \dots \ x_n]^T \\ \mathbf{y}_0 &= [y_1 \ y_2 \ y_3 \ \dots \ y_m]^T \end{aligned} \quad \mathbf{y}_0 = f(\mathbf{x}_0) = \begin{bmatrix} f_1(x_0) \\ f_2(x_0) \\ \vdots \\ f_m(x_0) \end{bmatrix} \quad (1)$$

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

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

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

$$\mathbf{J}(\mathbf{x}_0) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \quad (3)$$

Such that the underlying data and corresponding covariance transform like:

$$\mathbf{y}_0 = \mathbf{T}(\mathbf{x}_0) \cdot \mathbf{x}_0 \quad \mathbf{C}_{y_0} = \mathbf{J}(\mathbf{x}_0) \cdot \mathbf{C}_{x_0} \cdot \mathbf{J}(\mathbf{x}_0)^T \quad (4)$$

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:

$$\mathbf{z}_0 = \mathbf{T}_j(\mathbf{x}_j) \cdot \dots \cdot \mathbf{T}_2(\mathbf{x}_2) \cdot \mathbf{T}_1(\mathbf{x}_1) \cdot \mathbf{T}_0(\mathbf{x}_0) \cdot \mathbf{x}_0 \quad (5)$$

With the propagated covariance is given by:

$$\mathbf{C}_{z_0} = \mathbf{J}_j(\mathbf{x}_j) \cdot \dots \cdot \mathbf{J}_2(\mathbf{x}_2) \cdot \mathbf{J}_1(\mathbf{x}_1) \cdot \mathbf{J}_0(\mathbf{x}_0) \cdot \mathbf{C}_{x_0} \cdot \mathbf{J}_0(\mathbf{x}_0)^T \cdot \mathbf{J}_1(\mathbf{x}_1)^T \cdot \mathbf{J}_2(\mathbf{x}_2)^T \cdot \dots \cdot \mathbf{J}_j(\mathbf{x}_j)^T \quad (6)$$

5.2 Implementing standard POU in CAL

The operations in CAL are completely represented by a small set of simple transform types, each with a small kernel. The Jacobian matrix (J) follows directly from the transformation matrix (T) by applying equations (2) and (3). We present several of these transform types and their associated transformation and Jacobian matrices in detail below [7]. The MATLAB equivalents to the matrix multiplications are also shown. Table 1 lists the supported transform types.

Transform type ‘scale’ – scale an input vector by a scalar constant (k)

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

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

Transform type ‘scaleV’ – scale an input vector by a vector constant (v)

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

$$z = T \cdot x = v.*x \quad C_z = J \cdot C_x \cdot J^T = \text{scalecol}(v, \text{scalerow}(v, C_x))$$

Transform type ‘addV’ or ‘diffV’ – add or subtract two input vectors element by element

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

$$z = T \cdot \begin{bmatrix} x \\ y \end{bmatrix} = x \pm y \quad C_z = J \cdot \begin{bmatrix} C_x & 0 \\ 0 & C_y \end{bmatrix} \cdot J^T = C_x + C_y$$

Transform type ‘multV’ – multiply two input vectors element by element

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

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

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

Transform type ‘divV’ – divide two input vectors element by element

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

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

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

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

Table 1. Complete list of transform types supported by the POU framework

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

6. THE POU FRAMEWORK

The POU framework is implemented in CAL as a MATLAB structure. The transformation chain is stored by capturing first 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 a few associated functions used to store transformations and retrieve transformed data.

6.1 Creating the POU structure

A 2D array (variables x cadences) of empty POU structures (called errorPropStruct in CAL) is initialized using the top-level constructor (empty_errorPropStruct). Column-wise there is one array element for each CAL product plus several intermediate data products required for two-input transformations, as shown in Figure 2. Each array element in a column corresponds to a unique variable name.

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 points to another array element, allowing transformation chains to merge. Output of the constructors for the top level, second level, and third level is 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: [] % order of weighted poly
polyOrder: [] % vector to evaluate poly
polyXvector: []
binSizes: []
FCmodelCall: []
userM: [] % user defined matrix
xIndices: [] % select indices
size: [1x1 struct] % used for compression scheme
originalShape: [1x1 struct] % used for compression scheme

```

CAL DATA PROCESSING

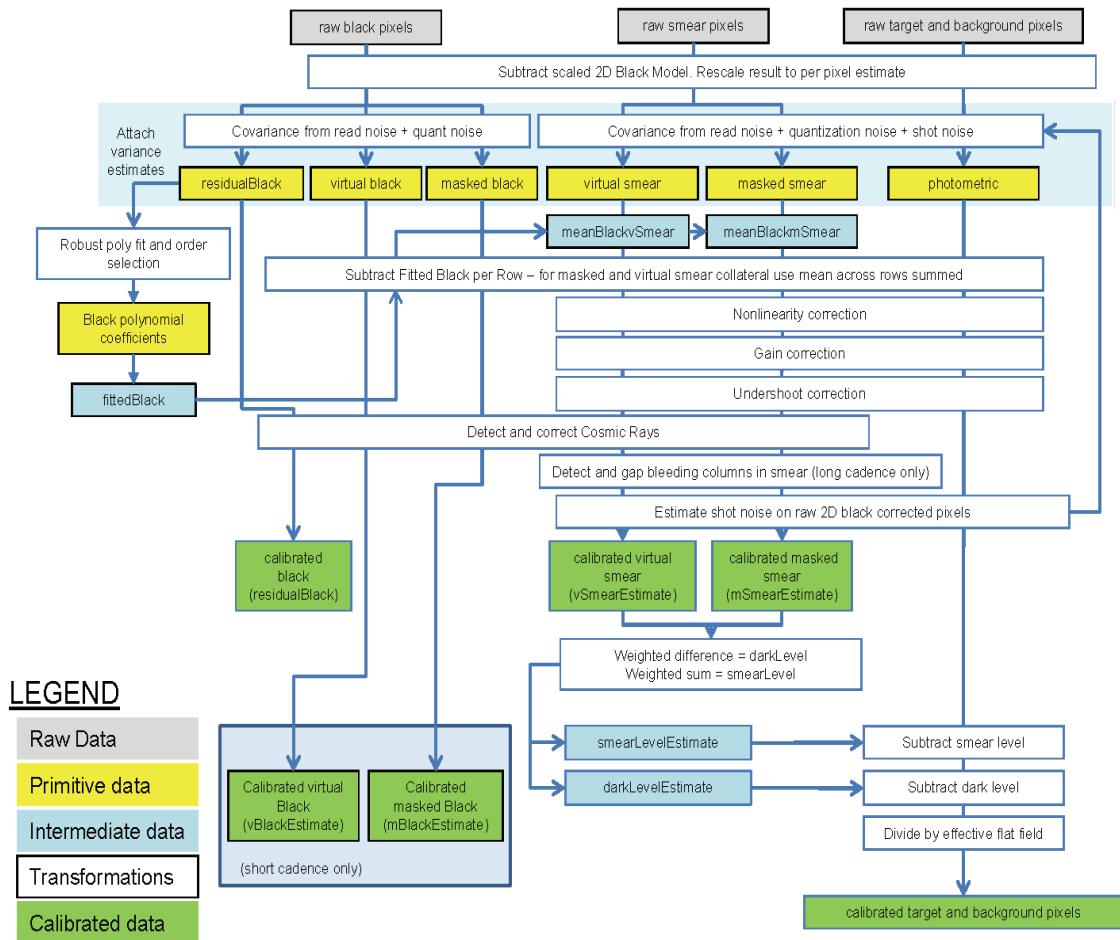


Figure 2. Data flow diagram showing operations performed in CAL. The variable names used in the POU structure are indicated on the diagram under ‘calibrated data’ and ‘intermediate data’. The variable names are typically chosen to reflect the end product of the transformation chain. For example, ‘vSmearEstimate’ is the variable name for the transformation chain which begins with raw virtual smear pixels and ends with calibrated virtual smear.

6.2 Append transformations

A small set of functions provides for populating the POU data structure. 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’ transform type. Subsequent transformations using combinations of other transform types which must be consistent with the supplied metadata. A ‘disableLevel’ parameter allows selective enabling of the base transformation and, separately, 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 [4].

6.3 Cascade transformations

The transformation chain associated with any variable name may be executed given a column of POU array elements, 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 underlying 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. To minimize the load on the database, the array of POU structure elements is compressed in a two-step process and stored in two arrays; one is a 1D version of the original 2D POU array and the other is an array of structures containing SVD basis vectors and singular values of the original primitive data and associated metadata.

7.1 Compression step 1

The 2D array of POU structures (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 arrays is 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 so the original structure can be restored 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 if a memory savings can be realized.

7.2 Compression step 2

The resulting 2D primitive data arrays and selected transformation metadata, which closely resemble the primitive data, are decomposed using Singular Value Decomposition. A user-defined number of the highest-power components are selected, and the resulting row and column basis vectors and associated singular values for only these components are saved in a data structure. The original data in the 1D array of POU structures is replaced with a pointer to the compressed data. The optimal model order as determined by the Akaike Information Criteria and the chi-square of the fit residual (residual power) are 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.

The compressed data array element below (compressedData(3)) is an example for 1132 virtual smear pixels over 476 cadences. The CxPrimitive and transformData fields are not expanded to the second level, but also contain an SVD decomposition set of basis vectors (U, V), singular values (S), and the associated optimal model order and residual power vector seen in the xPrimitive field. The top-level corresponding element in the original POU structure array is shown after compression (errorPropStruct(3)). The 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) [9]. In this way, consumers of the calibrated pixels have access to the propagated covariance. One consumer is the Photometric Analysis (PA) CSCI, which estimates and removes the background photometric signal and aggregates the target pixels to form flux time series for each target [10]. Using the functions within the POU framework, the pixel covariance matrix for any subset of photometric pixels is retrieved for any given cadence, row and column indices, and corresponding POU structure array. The need for providing the covariance matrix as a measure of the calibrated pixel correlations is evident through examination of the covariance images, as cross-correlations of up to several percent are noted. Figure 3 shows the location of 525 target and background pixels covering a 35 row x 160 column patch on one CCD and Figure 4 shows the calibrated pixel covariance 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 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.

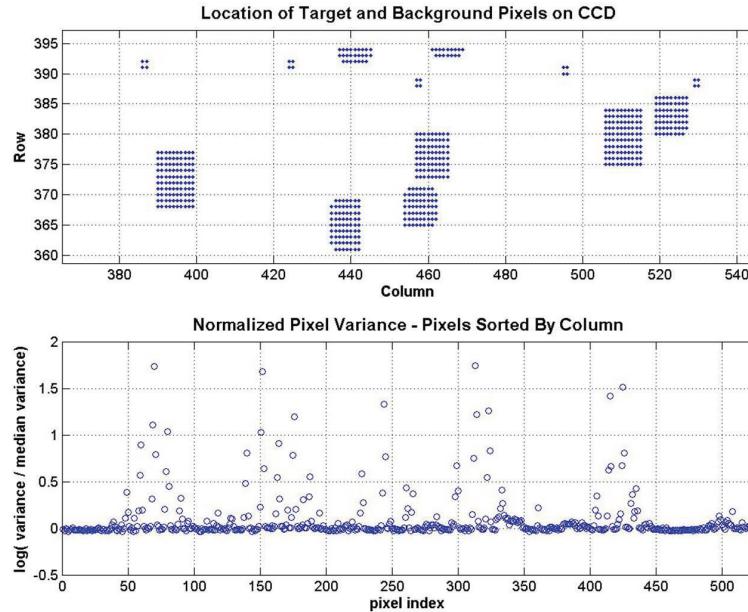


Figure 3. Top – Location of pixels for eight targets and five groups of background pixels from quarter 1 long cadence flight data. Bottom – variance of these pixels normalized to the median.

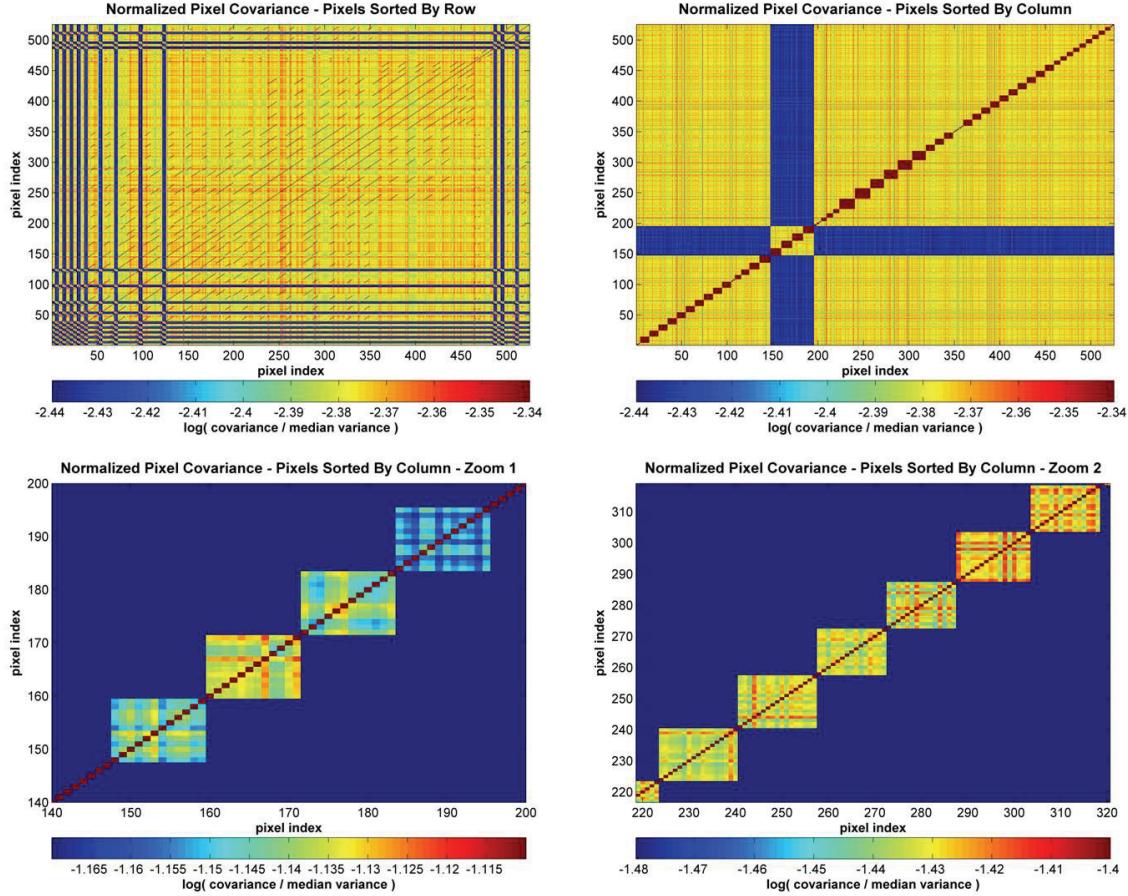


Figure 4. Top left – row-sorted covariance normalized to median variance. Top right – column-sorted covariance normalized to median variance. Bottom left and right – column sorted covariance normalized to median variance for two regions of interest in the top right image. The color axis limits have been adjusted on all images to enhance detail.

9. CONCLUSIONS

We note calibration-induced correlations between target and background pixels of up to several percent of the median variance, large enough to be non-negligible in downstream analysis. The SOC implementation of standard propagation of uncertainties using the POU framework provides full information about calibration-induced pixel correlations to downstream data analysis processes through the propagated covariance of the calibrated pixels. The SVD compression scheme makes propagation of the full covariance a tractable problem by reducing the memory needed for storage to just 4% of that needed for the underlying calibrated data. It also acts as a low-pass filter on the primitive data, removing spurious high frequency outliers. The fidelity of the propagated covariance elements is preserved to better than 0.01%, which is within the *Kepler* requirements.

10. FORWARD WORK

The POU framework currently includes only captures the CAL transformations. To properly assess any processing-induced correlations in the target flux time series, the framework should be expanded to accept inputs from PA. Including the background and target flux transformations in the chain will allow consumers of the PA output to assess target cross-correlations based on covariance, traceable to pixel measurement uncertainties.

The POU framework currently accepts only a simple variance vector as primitive input under the assumption that the raw pixel measurements are uncorrelated. If this assumption breaks down and cross-correlations between the raw pixels

become important it may be necessary to include the raw pixel covariance in the analysis. The framework should be expanded to accept a full primitive covariance matrix.

The POU framework is currently implemented as a MATLAB structure with a set of associated functions. To be consistent with the rest of the pipeline code, it should be implemented as a formal MATLAB class.

ACKNOWLEDGEMENTS

The authors would like to thank all the members of the *Kepler* team for their hard work and commitment to the success of the mission. Without them these results would not have been possible. Funding for the *Kepler Mission* has been provided by the NASA Science Mission Directorate.

REFERENCES

- [1] Caldwell, D., *et al.*, “Kepler Instrument Performance: an In-flight Update,” *Proc. SPIE 7740*, in press (2010).
- [2] Middour, C., *et al.*, “Kepler Science Operations Center Architecture,” *Proc. SPIE 7740*, in press (2010).
- [3] Klaus, T., *et al.*, “The Kepler Science Operations Center pipeline framework,” *Proc. SPIE 7740*, in press (2010).
- [4] Quintana, E. V., *et al.*, “Pixel-Level Calibration in the Kepler Science Operations Center Pipeline,” *Proc. SPIE 7740*, in press (2010).
- [5] Van Cleve, J., & Caldwell, D., *Kepler Instrument Handbook, KSCI 19033-001*, NASA Ames Research Center, Moffett Field, CA (2009).
- [6] Trefethen, L. N., and Bau, D., *Numerical Linear Algebra*, SIAM, 25-37 (1997).
- [7] Akaike, H., “A New Look at the Statistical Model Identification,” *IEEE Transactions on Automatic Control AC-19 (6)*, 716-723 (1974).
- [8] Chandrasekaran, H., *Propagation of Uncertainties in the PDQ Pipeline, KADN-26185*. NASA Ames Research Center, Moffett Field, CA (2009).
- [9] McCauliff, S., *et al.*, “The Kepler DB, a database management system for arrays, sparse arrays and binary data,” *Proc. SPIE 7740*, in press (2010).
- [10] Twicken, J. D., *et al.*, “Photometric analysis in the Kepler Science Operations Center pipeline,” *Proc. SPIE 7740*, in press (2010).