To: CSC Distribution

From: Frank Primini

Subject: Implementing *pymc3* in *CSC 2.1* Aperture Photometry

Version: 4.0

Date: April 6, 2022

---

1 Introduction

Aperture Photometry in *CSC 2.0* is computed for detections at the stack and observation level, and for sources at the master level, using the Bayesian formalism described in Primini & Kashyap (2014). Key to the approach at all levels is the use of Markov Chain Monte Carlo (*MCMC*) sampling to characterize the marginalized posterior probability distribution (*MPDF*) for source intensity. Detailed specifications are provided in Primini (2013) and Primini (2019). Although this approach is satisfactory in general, it fails for a small but significant number of detections and sources, resulting in NULL values for photometry columns in the catalog database. For example, $\sim 1\%$ of the master sources that include *ACIS* observations have NULL values for photflux_aper_b or photflux_aper_avg_b (failure rates for flux_aper_b and flux_aper_avg_b are slightly worse because they're compounded by known systematic errors in computing model-independent fluxes for low-count sources).

Rafael hypothesized that part of the problem was due to the particular *MCMC* routine used in *CSC 2.0*, here referred to as *sherpa/get_draws*, and suggested replacing it with a more sophisticated and efficient routine in the OTS python package *pymc3*. He presented some preliminary research indicating that *pymc3* was able to recover fluxes for some sources with NULL catalog values (Martinéz-Galarza , 2020).

In this memo I describe the steps required to implement *pymc3* in *CSC 2.1* Aperture Photometry and present the results of more extensive tests that support its use.

2 Implementing *pymc3* in *CSC 2.1*

2.1 Scope of Modifications

I should emphasize that the sole intent of the proposed modifications is to replace the existing *MCMC* sampling engine with a different one. That will, of course, necessitate some changes to the infrastructure that supports the *MCMC* engine, but the basic Aperture Photometry workflow remains the same. All Aperture Photometry quantities computed in *CSC 2.0* will be computed in *CSC 2.1*, and no new quantities will be computed. With one exception, described below, the definition of all aperture quantities remains unchanged.

2.2 Basic Algorithm

All input data needed for *pymc3* are contained in the obsid-level *prep3* files, which are unchanged from *CSC 2.0*. Specifically, these data are the array of raw counts in all apertures in the bundle, provided in the 'counts' column **C** of the source properties table extension, and the exposure-corrected PSF matrices **F**, provided in the PSF extensions:

$$\mathbf{C} = \begin{pmatrix} C_1 \\ C_2 \\ \vdots \\ B \end{pmatrix}; \quad \mathbf{F} = \begin{pmatrix} f_{11} & \cdots & f_{1n} & \Omega_1 \\ \vdots & \ddots & \vdots & \vdots \\ f_{n1} & \cdots & f_{nn} & \Omega_n \\ g_1 & \cdots & g_n & \Omega_b \end{pmatrix}. \tag{1}$$

Here, $C_1, C_2, ...B$ represent the counts in the source and background apertures. The quantity f_{ij} is the *PSF* fraction for source j in aperture i, modified by the exposure for that aperture, and Ω_j is the aperture area, also modified by exposure.

The different types of exposure correction determine the units of the resulting intensities. As in *CSC 2.0*, all will be used in *CSC 2.1*.

The organization and use of these data in *pymc3* at the obsid, stack, and master level are described in the following sections.

2.2.1 Obsid-Level Photometry

At the level of a single obsid, **C** and **F** are used directly in pymc3, as illustrated in the code snippet in Listing 1.

2.2.2 Stack-Level Photometry

The situation is more complicated when we need to combine data from multiple obsids in a stack. Here, we still use **C** and **F** for each contributing obsid, but combine them into a single, augmented **C** vector and **F** matrix. Thus, for a two-source bundle and two obsids, we have

$$\mathbf{C^a} = \begin{pmatrix} C_1^1 \\ C_2^1 \\ B^1 \\ C_1^2 \\ C_2^2 \\ B^2 \end{pmatrix}; \quad \mathbf{F^a} = \begin{pmatrix} f_{11}^1 & f_{12}^1 & \Omega_1^1 & 0 \\ f_{21}^1 & f_{22}^1 & \Omega_2^1 & 0 \\ g_1^1 & g_2^1 & \Omega_b^1 & 0 \\ f_{11}^2 & f_{12}^2 & 0 & \Omega_1^2 \\ f_{21}^2 & f_{22}^2 & 0 & \Omega_2^2 \\ g_1^2 & g_2^2 & 0 & \Omega_b^2 \end{pmatrix}, \tag{2}$$

where the superscripts represent the different obsids. Note, we require source intensities to be the same in all obsids, with a different background density in each obsid. Thus, the total number of parameters to fit, *i.e.* the number of columns in $\mathbf{F^a}$ is the sum of the number of sources in the bundle and the number of obsids. Similarly, the number of rows $\mathbf{F^a}$ is the sum of the number of apertures in the bundle for contributing obsids.

In constructing $\mathbf{F^a}$, we need to make sure that data associated with a particular source or background in the contributing obsids are all assigned the same column index. If the number of sources in the bundle is the same in all obsids (as it usually is, since bundle membership is determined at the stack level) that's easily accomplished by pasting a column of $0's$ in the appropriate background column location in the individual **F** before appending to $\mathbf{F^a}$, as indicated in Equation 2 above. The Python function in Listing 2 illustrates how this can be accomplished.

```
import pymc3 as pm
basic_model = pm.Model()
with basic_model:
    # Define source and background priors
    # The array C is used to compute the number of sources in the bundle.
    number_of_sources=len(C)-1
    s=[]
    for nsrc in range(number_of_sources):
        s.append(pm.Uniform('s_{}'.format(nsrc), lower=0.0,upper=1.0e-8))
    b = pm.Uniform('b',lower=0.0,upper=1.0e-12)
    # Define modelled stochastic variable mu
    mu=0
    # The transpose of the PSF matrix F is used to compute mu.
    theta=F.T
    for i in range(number_of_sources):
        mu += s[i]*theta[i]
    mu += b*theta[-1]
    # Define likelihood of observations, given counts C
    likelihood=pm.Poisson('C_obs',mu=mu,observed=C)
    # Perform MCMC sampling
    # Define pymc3 output data structure
    db = pm.backends.ndarray.NDArray()
    # Do MCMC sampling
    trace = pm.sample(1000,tune=1000,target_accept=0.95, init='advi+adapt_diag', trace
    =db,progressbar=False)
    # Get arrays of draws. Each element in the draws_data list
    # is an array of draws, corresponding to the source or
    # background named in the draws_name list.
    draws_data=[]
    draws_name=[]
    for nsrc in range(number_of_sources):
        src='s_{}'.format(nsrc)
        draws_data.append(trace.get_values(src))
        draws_name.append(src)
    draws_data.append(trace.get_values('b'))
    draws_name.append('b')
```

Listing 1: Example pymc3 code for a single obsid

```
def augment_F(F1,F2):
    #Appends F2 matrix rows to bottom of F1 matrix, inserting a dummy column
    # of zeros between the last source column and the background column,
    # F1 and F2 do not need to have the same number of rows or columns.
    #
    # add a column of zeros to F1 for F2 bkg
    newcol_1=np.zeros([len(F1),1])
    F1=np.hstack((F1,newcol_1))
    # add a submatrix of zeros to F2 so it has same number of columns as F11
    newcol_2=np.zeros([len(F2),len(F1[0])-len(F2[0])])
    F2_trans=np.hstack((F2,newcol_2)).T
    # Now swap b row and last zero row of F2 transpose
    lastndx=len(F2)-1
    F2_trans[[lastndx,-1]]=F2_trans[[-1,lastndx]]
    return np.vstack((F1,F2_trans.T))
```

Listing 2: Function to combine two F matrices

However, if the detection is near the edge of the field-of-view of the stack, it's possible that one

or more sources in the bundle may be outside the valid stack boundaries in one or more obsids. Consider again the example in Equation 2, but now assume that in obsid 1, source 1 is outside the valid stack. We have

$$\mathbf{F_1} = \begin{pmatrix} f_{22}^1 & \Omega_2^1 \\ g_2^1 & \Omega_b^1 \end{pmatrix}; \quad \mathbf{F_2} = \begin{pmatrix} f_{11}^2 & f_{12}^2 & \Omega_1^2 \\ f_{21}^2 & f_{22}^2 & \Omega_2^2 \\ g_1^2 & g_2^2 & \Omega_b^2 \end{pmatrix}; \quad \mathbf{C^a} = \begin{pmatrix} C_2^1 \\ B^1 \\ C_1^2 \\ C_2^2 \\ B^2 \end{pmatrix}; \quad \mathbf{F^a} = \begin{pmatrix} 0 & f_{22}^1 & \Omega_2^1 & 0 \\ 0 & g_2^1 & \Omega_b^1 & 0 \\ f_{11}^2 & f_{12}^2 & 0 & \Omega_1^2 \\ f_{21}^2 & f_{22}^2 & 0 & \Omega_2^2 \\ g_1^2 & g_2^2 & 0 & \Omega_b^2 \end{pmatrix}. \tag{3}$$

In constructing $\mathbf{F^a}$, the order of the contributing obsids is arbitrary (it's analagous to rearranging the rows in a set of simultaneous linear equations), as long as the same order is followed in constructing $\mathbf{C^a}$.

Once $\mathbf{F^a}$ and $\mathbf{C^a}$ have been constructed, MCMC sampling proceeds as in the single-obsid case (see Listing 3).

```
import pymc3 as pm
# Set the pymc3 model
# Assume F = augmented F matrix, C=augmented C vector
# number_of_sources = total number of source columns in F
# The number of background parameters is the number of columns in the augmented # PSF
    matrix minus the number of sources.
number_of_backgrounds=len(F[0])-number_of_sources
basic_model = pm.Model()
with basic_model:

    # Priors for the net source and background energy fluxes. These are now
    # arrays of priors

    s =pm.Uniform('s', lower=0.0, upper=1.0e-8, shape=(number_of_sources,))
    b =pm.Uniform('b', lower=0.0, upper=1.0e-12,shape=(number_of_backgrounds,))

    # mu will be the modeled stochastic variable, i.e., the expected
    # values for flux plus background flux each aperture

    # The transpose of the PSF matrix F is used to compute mu.
    theta=F.T

    # Expected value of outcome
    mu = 0
    for i in range(number_of_sources):
        mu += s[i]*theta[i]
    for j in range(number_of_sources,number_of_sources+number_of_backgrounds):
        mu += b[j-number_of_sources]*theta[j]

    likelihood = pm.Poisson('C_obs',mu=mu,observed=C)

    # Now do the MCMC sampling
    db = pm.backends.ndarray.NDArray()
    trace = pm.sample(1000,tune=1000,target_accept=0.95, trace=db,progressbar=False,
    init='advi+adapt_diag')
    # sdraws and bdraws are now arrays of draws arrays
    sdraws=trace.get_values('s')
    bdraws=trace.get_values('b')
```

Listing 3: Example pymc3 code for multiple obsids

2.2.3 Master Source Photometry

The situation becomes yet more complicated in computing photometry for master averages or Bayesian Blocks. $\mathbf{F^a}$ and $\mathbf{C^a}$ are still computed using the approach in Section 2.2.2, but the contributing obsids may now come from different stacks, in which the source labelling and bundle membership are different. The number of sources should now include all the sources in all the contributing bundles, and all stack component IDs corresponding to a given master source should be assigned the same source column in $\mathbf{F^a}$. The mst3 files provide the association of different stack component IDs with a particular master source.

Consider, for example, a master source $M1$ with two contributing stacks, $S1$ and $S2$, each with a single obsid, $S1.O1$, and $S2.O2$. Let's further assume that $S1.O1$ is a two-source bundle, with region-ids $S1.O1.r1$ and $S1.O1.r2$, and $S2.O2$ is a one-source bundle with region-id $S2.O2.r3$. Finally, let's assume the order of components in $S1.O1$ is $[S1.O1.r1, S1.O1.r2]$ (so that component $S1.O1.r1$ corresponds to the left-most column in $\mathbf{F}$). The F matrices for the two stacks are then

$$\mathbf{F^{S1.O1}} = \begin{pmatrix} f_{11}^{S1.O1} & f_{12}^{S1.O1} & \Omega_1^{S1.O1} \\ f_{21}^{S1.O1} & f_{22}^{S1.O1} & \Omega_2^{S1.O1} \\ g_1^{S1.O1} & g_2^{S1.O1} & \Omega_b^{S1.O1} \end{pmatrix}; \quad \mathbf{F^{S2.O2}} = \begin{pmatrix} f_{11}^{S2.O2} & \Omega_1^{S2.O2} \\ g_1^{S2.O2} & \Omega_b^{S2.O2} \end{pmatrix}. \tag{4}$$

If the components $S1.O1.r1$ and $S2.O2.r3$ comprise $M1$ (and component $S1.O1.r2$ is associated with a different source, say $M2$), then the augmented matrix for $M1$ is

$$\mathbf{F^a} = \begin{pmatrix} f_{11}^{S1.O1} & f_{12}^{S1,O1} & \Omega_1^{S1.O1} & 0 \\ f_{21}^{S1.O1} & f_{22}^{S1.O1} & \Omega_2^{S1.O1} & 0 \\ g_1^{S1.O1} & g_2^{S1.O1} & \Omega_b^{S1.O1} & 0 \\ f_{11}^{S2.O2} & 0 & 0 & \Omega_1^{S2.O2} \\ g_1^{S2.O2} & 0 & 0 & \Omega_b^{S2.O2} \end{pmatrix}. \tag{5}$$

If, however, components $S1.O1.r2$ and $S2.O2.r3$ comprise $M1$, $\mathbf{F^a}$ is given by

$$\mathbf{F^a} = \begin{pmatrix} f_{11}^{S1.O1} & f_{12}^{S1,O1} & \Omega_1^{S1.O1} & 0 \\ f_{21}^{S1.O1} & f_{22}^{S1.O1} & \Omega_2^{S1.O1} & 0 \\ g_1^{S1.O1} & g_2^{S1.O1} & \Omega_b^{S1.O1} & 0 \\ 0 & f_{11}^{S2.O2} & 0 & \Omega_1^{S2.O2} \\ 0 & g_1^{S2.O2} & 0 & \Omega_b^{S2.O2} \end{pmatrix}. \tag{6}$$

In either case, the augmented $\mathbf{C}$ vector is given by

$$\mathbf{C^a} = \begin{pmatrix} C_2^1 \\ B^1 \\ C_1^2 \\ C_2^2 \\ B^2 \end{pmatrix}. \tag{7}$$

2.3 Determining the Range of Uniform Priors

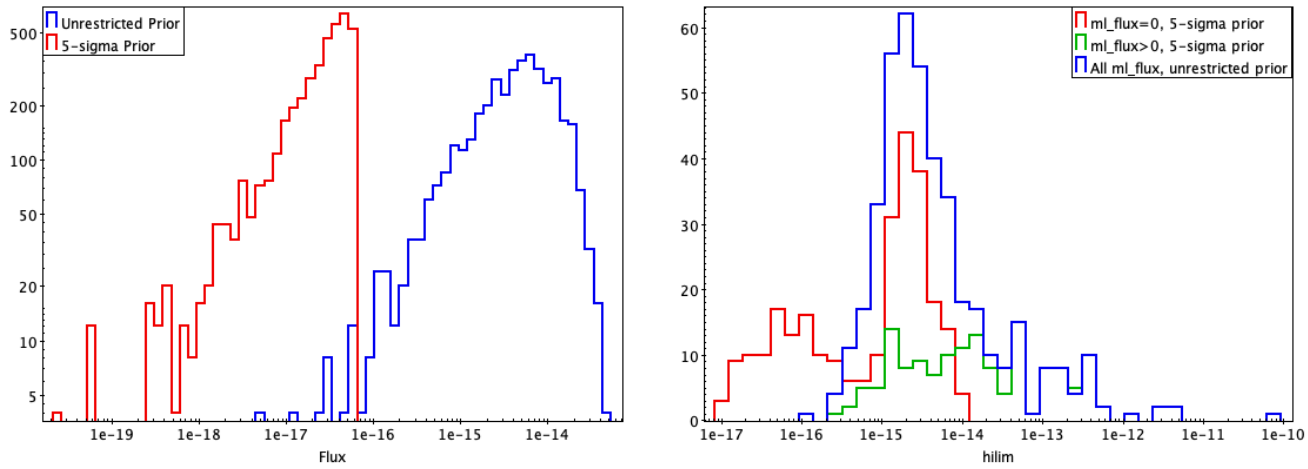
In *CSC 2.0*, uniform priors were used for both source and background. The range of the priors was determined from $s_{ML} \pm 5 \times \sigma_{ML}$, where s_{ML}, σ_{ML} are the Maximum-Likelihood values and

uncertainties that are the solution to the matrix equation

$$\mathbf{F} \times \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ b \end{pmatrix} = \mathbf{C}, \tag{8}$$

where **F** and **C** are as defined in Equation 1 (see *e.g.* equations 8 & 9 in Primini & Kashyap (2014)). When $s_{ML} - 5\sigma_{ML} < 0$, the lower bound of the prior was set to a very small positive number.

This approach appeared to work well for the *pymc3* tests described below, for samples that were not dominated by sources or detections with NULL *CSC 2.0* fluxes. However, for the NULL samples (samples 1&2), *pymc3* often returned modes and bounds which, although not NULL, were unphysically small. This is illustrated in Figure 1 (a), where I compare the distribution of *pymc3* draws for an observation with a NULL b band *CSC 2.0* flux, using both priors set as in *CSC 2.0* ("5-σ") and a less restrictive prior with a range of $0 - 10^{-8}$, comparable to the range of fluxes in *CSC 2.0* ("unrestricted"). The 5-σ draws are two to three orders of magnitude lower than the unrestricted draws, and fall sharply at the high end of the distribution, indicating that *pymc3* did not properly sample the parameter space.



(a) (b)

Figure 1: *(a) Distribution of draws for region_id 80 in obsid 78; (b) Distribution of pymc3 upper bounds for 500 observations with NULL b band fluxes, computed using both 5-σ and unrestricted priors.*

I hypothesize that, in such cases, the Maximum-Likelihood solution was in fact negative and reset to 0 in *CSC 2.0*, and the Maximum-Likelihood uncertainty underestimated the actual uncertainty. This limited the range of the prior to a very small section of the parameter space. This hypothesis is supported by the data in Figure 1 (b), where I compare distributions of *pymc3* upper bounds for a sample of ~ 500 observations with NULL b band fluxes, computed using both types of priors. The distribution computed with 5-σ priors for observations with non-zero Maximum-Likelihood values (ml_flux) is consistent with the distribution of bounds computed using unrestricted priors. However, for observations with $ml\_flux = 0$, the distribution for bounds computed with 5-σ priors is bi-modal, with a significant number of observations reporting very small upper bounds.

To avoid both the difficulties in determining which observations with $ml\_flux = 0$ would succeed with 5-σ priors and the added complexity of using differently defined priors for different

observations, I propose adopting unrestricted priors in all cases. After several rounds of experimentation, I've determined prior ranges that yielded successful results for both NULL and non-NULL samples. These are listed in Table 1, and were used in all the tests below. An example of their use is shown in lines 9 & 10 in Listing 1.

Table 1: *pymc3* Uniform Prior Bounds

Flux Units	Parameter	P_{lower}	P_{upper}
Energy Flux	Source	0.0	1.0×10^{-8}
Energy Flux	Background	0.0	1.0×10^{-12}
Energy Flux	HRC Background	0.0	1.0×10^{-13}
Photon Flux	Source	0.0	1.0×10^{-2}
Photon Flux	Background	0.0	1.0×10^{-6}
Count Rate	Source	0.0	10.0
Count Rate	Background	0.0	1.0×10^{-3}
Net Counts	Source	0.0	1.0×10^{6}
Net Counts	Background	0.0	10.0

2.4 Required Number of Sampling Chains and Samples per Chain

In *CSC 2.0*, a single sampling chain was used, with 5000 samples after a "burn-in" period of 100 samples. In *pymc3*, multiple chains are supported, and in the tests discussed below ranged from $2-4$, depending on the machine. Multiple chains are desirable because they enable a better determination of the *MCMC* diagnostic $\hat{r}$. Also, in the tests discussed below, the number of samples per chain was 1000 is a "burn-in" period of 1000, yielding a total number of draws of $2000-4000$. This proved more than adequate to characterize the *MPDFs*.

3 Verification Tests

The goal of these tests was to demonstrate that *pymc3* both reduced the failure rate for *CSC 2.0* sources and detections with NULL fluxes, and produced results comparable to *Sherpa/get_draws* for successful *CSC 2.0* detections and sources. The detailed strategy was described in the test plan circulated in February, 2020 (Primini , 2020). In the end, I was unable to achieve the sample sizes specified in the plan for all tests, but I believe the samples were sufficient to prove the point.

3.1 Overall Procedure

For each sample, a list of obsids, stacks, or master sources, plus the corresponding source region identifiers were generated. These data were used to identify the appropriate prep3 files for each sample member (stack detections and master sources could have multiple prep3 files), and the prep3 data were used as input to the pymc3 *MCMC* engine.

For each detection or source, $2-4$ sets of 1000 *MCMC* draws (*i.e. MCMC*-sampled values of source or background intensity) were generated, excluding burn-out draws, and the distribution of the combined set of draws was smoothed with an Epanechnikov Kernel Density Estimator, scaled to the rms about the mean. Uniformly-spaced samples from this smoothed distribution

Table 2: *pymc3* Test Samples

Sample	Level	Description
1	Obsid	Randomly selected sample of $\sim$ 1000 obsid-level detections with NULL flux_aper_u values
2	Obsid	Sample provided by a *CSC* user, of $\sim$ 2500 obsid-level detections with NULL fluxes in either h, m, or s bands
3	Obsid	Randomly selected sample of $\sim$ 1000 obsid-level detections with valid (*i.e.* not NULL, not upper limits) flux values in various *ACIS* energy bands in *CSC 2.0*
4	Obsid	Randomly selected sample of $\sim$ 1000 obsid-level detections with valid s-band upper limits in *CSC 2.0*
5	Obsid	Randomly selected sample of $\sim$ 400 HRC obsid-level detections
6	Stack	Randomly selected sample of $\sim$ 1000 stack-level detections with valid stack-average fluxes in *CSC 2.0*
7	Stack	Randomly selected sample of $\sim$ 5000 stack-level detections with valid stack-average fluxes in *CSC 2.0*
8	Master	Sample of $\sim$ 3000 s band master source averages from single-source bundles with five contributing obsids from multiple stacks
9	Master	Sample of $\sim$ 3000 s band master source averages from single-source bundles with five contributing obsids from multiple stacks
10	Master	Sample of $\sim$ 1000 u band master source averages from single-source bundles with five contributing obsids from multiple stacks

were then used to estimate the mode and 68% bounds. These values were then compared to the corresponding *flux_aper*, *flux_aper_lolim*, and *flux_aper_hilim* values.

For quantitative comparison of *pymc3* amd *CSC 2.0* fluxes, I defined the quantity

$$f = \frac{2 \times |flux\_aper - mode|}{flux\_aper\_hilim - flux\_aper\_lolim}, \tag{9}$$

where $flux\_aper, flux\_aper\_lolim, flux\_aper\_hilim$ refer to the appropriate *CSC 2.0* flux, photflux, src_rate, src_cnts, and bounds, and $mode, lolim, hilim$ refer to the corresponding *pymc3* mode and bounds. Because I didn't force *pymc3* modes to be 0 for upper limits, as we did in *CSC 2.0*, I used an alternate definition

$$f = \frac{2 \times |flux\_aper\_hilim - hilim|}{flux\_aper\_hilim - flux\_aper\_lolim}, \tag{10}$$

for upper limits (*i.e.* when $flux\_aper = 0$).

3.2 Test Results

A description of the test samples used is given in Table 2, and a summary of results is given in Table 3. Some samples were used multiple times to test different flux units or energy bands.

Table 3: $pymc3$ Test Results

Sample	Flux Units	Energy Band	$N_{success}/N_{total}$	$\overline{f}$	f_{median}	f_{99}	% with $f \leq 1.0$
1	Energy Flux	u	866/867	N/A	N/A	N/A	N/A
2	Energy Flux	h	830/846	N/A	N/A	N/A	N/A
2	Energy Flux	m	1425/1522	N/A	N/A	N/A	N/A
2	Energy Flux	s	2296/2367	N/A	N/A	N/A	N/A
3	Energy Flux	b,h,m,s,u	822/912	0.1	0.1	0.3	99.6%
4	Energy Flux	s	825/833	0.5[1]	0.5	1.4[1]	92.1%
5	Energy Flux	w	427/431	16.0	0.6	1.4	86.7%
5	Photon Flux	w	431/431	1.0	0.6	1.6	88.6%
6	Photon Flux	b,h,m,s,u	956/981	0.8	0.1	6.0	98.4%
6	Count Rate	b,h,m,s,u	956/981	0.4	0.1	1.0	98.8%
6	Net Counts	b,h,m,s,u	956/981	6.0	0.1	47.8	96.9%
7	Energy Flux	b,h,m,s,u	4700/5000	143.0	0.1	1.2	99.0%
8	Photon Flux	b	797/798	0.1	0.1	0.4	98.1%
9	Photon Flux	s	3191/3193	0.2	0.1	1.1	94.0%
10	Photon Flux	u	1095/1096	0.4	0.2	2.0	81.4%

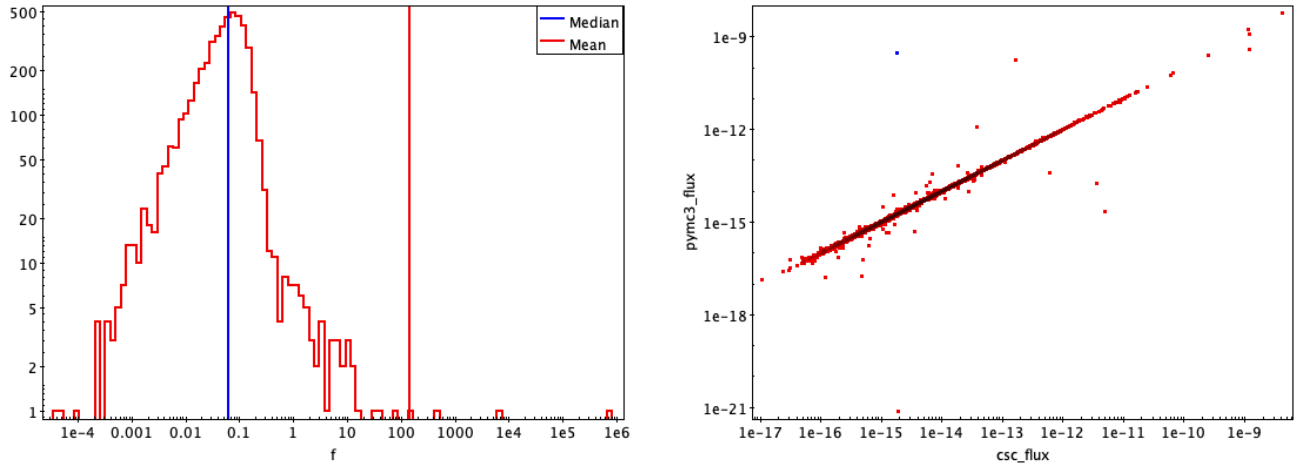
1. There's an apparent offset between pymc3 hilim and flux_aper_hilim_s; $pymc3\_hilim \sim 1.2 \times flux\_aper\_hilim\_s$.

In all applicable samples, $f_{median} < 1$, with most at or near the value of 0.1 specified in the Test Plan (Primini , 2020). The values of $\overline{f}$ are somewhat larger. Although most are still less than 1, three samples have $\overline{f} > 1$, the largest being Sample 7 with $\overline{f} = 143$. Plots of the distribution of f and a comparison of the *pymc3* and *CSC 2.0* fluxes for this sample are shown in Figure 2. In this case, the large value of f is primarily due to two extreme outliers with f 7300 and f 6.6×10^5. It should be noted that I did not remove outliers from any samples prior to computing statistics on f. In general, a few, very large outliers can severely bias $\overline{f}$ with little effect on f_{median}.

I did not compute f for the NULL samples 1&2, because I deemed the Maximum-Likelihood values and uncertainties needed to compute f for these detections to be unreliable (see Section 2.3).

4 Timing Tests

To estimate the processing burden of *pymc3* I used the *python* **time.perf_counter()** function to measure the time spent in each *pymc3* rountine, as illustrated in Listing 4, and compiled the cumulative time in all *pymc3* routines for a subset of observations in the samples in Table 2. I used ~ 1000 single-obsid and multi-obsid stacks and ran tests on three different machines: the HEAD-LAN compute server han-v, and two dedicated desktops, devel1 and dagny. The results are shown in Table 4.



(a) (b)

Figure 2: *(a) Distribution of f for Sample 7; (b) Comparison of pymc3 and CSC 2.0 fluxes or upper bounds for Sample 7. The data point in blue is an extreme outlier with* $f \sim 6.6 \times 10^5$.

```
        t0=time.perf_counter()
        s = pm.Uniform('s', lower=0.0, upper=1.0e-8, shape=(number_of_sources,))
        b = pm.Uniform('b', lower=0.0, upper=1.0e-12, shape=(number_of_backgrounds,))
        t1=time.perf_counter()
        dt_prior=t1-t0
```

Listing 4: Computing time spent generating priors

Table 4: Timing Test Results

Machine	Type of Stack	Number of Runs	Time per Run (sec.)
han-v	multi-obsid	955	91
dagny	multi-obsid	965	22
dagny	single-obsid	958	12
devel1	multi-obsid	956	13

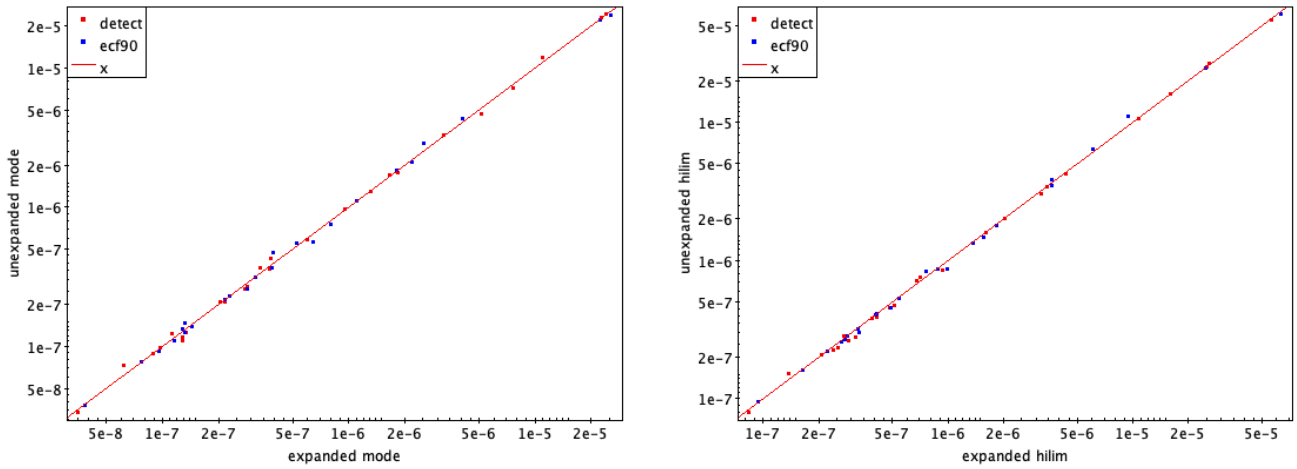
It should be noted that the **time.perf_counter()** function includes both system and process time, which may account for the large difference between the performance on the han-v server and the dedicated desktops. Also note that these results apply to a single band (out of 5 typically) and a single aperture type (detect or ecf90). Actual times per source may be a factor of ~ 10 larger.

5 Background Expansion Tests

In *CSC 2.0*, background apertures with too few counts were expanded to include more counts. This was deemed necessary to prevent aperture photometry from failing. The background expansion was done independently in each band, resulting in different area_aperbkg values for different bands, in violation of the requirement of constant aperture areas across all bands. To address this, once the archive was populated, a migration step was run to set the background apertures in all

bands to the b band value, and to scale the background aperture counts in other bands by the ratio of the b band aperture to that band's aperture. I estimated that $\sim 30\%$ of the *CSC 2.0* observations were affected.

The improved performance of *pymc3* over *sherpa/get_draws()* for *CSC 2.0* sources/detections with NULL fluxes suggests the background expansion can be avoided in *CSC 2.1*. To investigate this, I ran *pymc3* on a set of observations in the CAT5.0 integration test set, with aperture quantities from prep3 files generated with and without background expansion. The (admittedly small) sample included both detect and ECF90 apertures. Because expanded background cases typically involve few counts, I used photon fluxes, to avoid the known systematic errors in model-inependent energy flux for such cases.



(a) (b)

Figure 3: *Comparison of modes (a) and upper bounds (b) computed with and without expanded background apertures.*

Results are shown in Figure 3 and indicate good agreement between the "unexpanded" and "expanded" values.

I also compared the photon fluxes from the unexpanded backgrounds to their corresponding *CSC 2.0* photflux_aper values. These results are shown in Figure 4 and also indicate good agreement, except for detect and ecf90 aperture values for two detections, region_id 13 and 23 in obsid 1047.

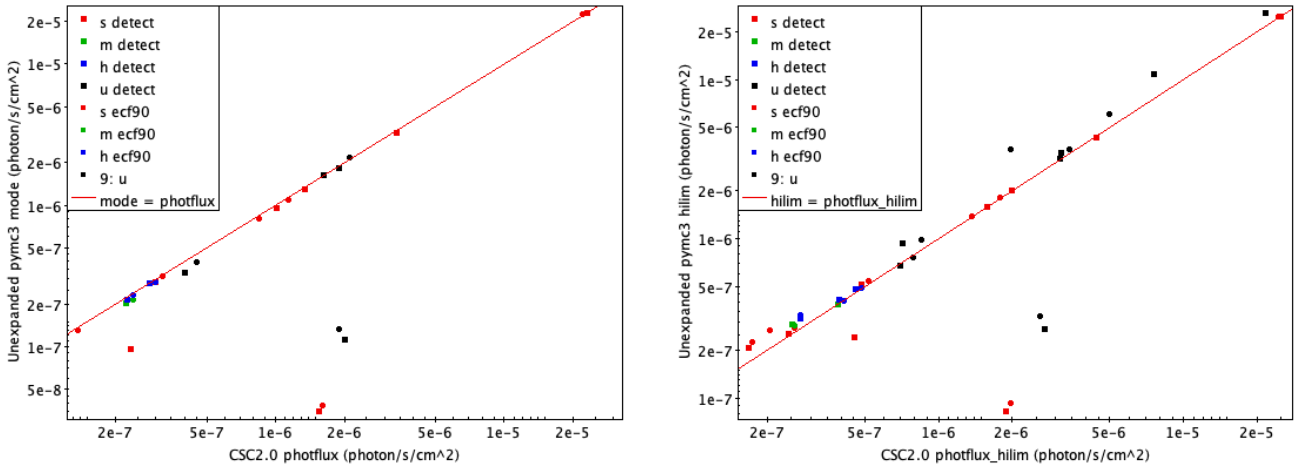
A larger test dataset is desirable here, but on the basis of this limited testing, I conclude that we can avoid background expansion when using *pymc3*.

6 Addressing Failures

I anticipate two failure modes for *pymc3*, and offer some recommendations for their mitigation.

6.1 When *pymc3* Fails

In *CSC 2.0*, if *sherpa/get_draws()* failed, we attempted to re-run after freezing background parameters at values determined by corresponding *bkgmaps*. In principle, We can address hard *pymc3* failures in the same way. We can estimate the background density from counts and psf matrix



(a) (b)

Figure 4: *Comparison of unexpanded pymc3 modes (a) and upper bounds (b) to corresponding CSC2.0 photflux and photflux_hilim values.*

data in the prep3 files, as $b_{frozen} \simeq B/\Omega_b$, where B and Ω_b are as defined in Equation 1. this is illustrated in the code block shown in Listing 5.

```
    with basic_model:
        # Priors for the net source and background energy fluxes
        s=[]
        for nsrc in range(number_of_sources):
            s.append(pm.Uniform('s_{}'.format(nsrc), lower=0.0,upper=prior_bound))
        #
        # Approximate background density by total background aperture counts divided
        # by exposure-corrected aperture area
        #
        b = C[-1]/theta[-1,-1]

        # mu will be the modeled stochastic variable
        # i.e., the expected values for theta
        # (source flux plus background flux in each aperture)
        mu = 0

        # Compute the expected flux theta_i in each aperture spec
        # as the sum of contributions from all sources plus background.
        for i in range(number_of_sources):
            mu += s[i]*theta[i]
        mu += b*theta[-1]
```

Listing 5: Freezing the Background Parameter

To demonstrate the validity of this approach, I've rerun the data in Sample 4, with backgrounds frozen as in Listing 5. The results are shown in Figure 5 and indicate good agreement between the source flux upper limits computed with and without frozen background parameters (this is the appropriate quantity to compare since this is a sample of *CSC 2.0* upper limits). It should also be noted that while 825 of 833 sample members were run successfully when fitting the background parameter, all 833 succeeded when the background was frozen, suggesting that this approach can recover *pymc3* failures.

An alternative approach to addressing *pymc3* failures is to adjust to bounds of the priors. Based

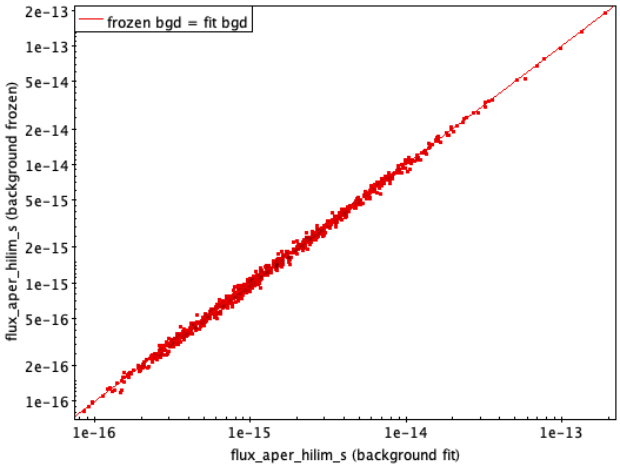


Figure 5: *Comparison of Sample 4 upper limits computed with and without frozen background parameters.*

on my experiences in testing samples and experimenting with different priors, I find that *pymc3* failures are most likely to occur if P_{upper} is set too high, and that reducing it by a factor of ~ 10 may eliminate the error. However, this is an *ad hoc* approach with no guarantee of success on the first iteration, and is not recommended over freezing background parameters.

6.2 When *pymc3* Values Are Unrealistic

This failure mode is mode difficult to identify, since the *pymc3* routines run to completion without error. However, a close examination would indicate unreasonable values for modes or bounds.

For example, the modes for either source or background could appear "pegged" at or near the upper bound of the corresponding prior. This could occur if we encounter a very bright or flaring source, whose intensity exceeds the limits on source priors in Table 1, which were based on *CSC 2.0* sources. Alternatively, the source could be in or near a very bright extended background region, and the background intensity exceeds the limit on background priors. I've examined $\sim$ 7500 draws distributions from Samples 2&7 in Table 2, and found one case in which the background prior was pegged. That distribution is shown in Figure 6, for Obsid 16190, region_id 716, from stack acisfJ0332281m274818_001 in Sample 7.

We can use a simple "mode+3σ" test to identify such distributions. If $mode, hilim, lolim$ are the mode and upper and lower confidence bounds of the KDE-smoothed draws distribution, then we can identify a distribution as pegged if

$$mode + \frac{3}{2}(hilim - lolim) > P_{upper}. \tag{11}$$

Another example of unrealistic values occurs in estimating energy fluxes. This could occur if there are too few counts in either source or background apertures, due to the known systematic errors in estimating model-independent energy fluxes in such cases. Alternatively, as with the NULL sources discussed in Section 2.3, the *MCMC* sampling may be restricted to a very narrow range of parameter space near 0 flux. It should be relatively easy to identiy such cases by requiring that modes and upper bounds to fluxes correspond to reasonable net counts. For example, for a power law spectrum with typical N_H and slope, ~ 3 net counts in a $\sim 10 Msec$ *ACIS* observation in 2020 would correspond to a b band flux of $\sim 5 \times 10^{-18} ergs - cm^{-2} - s^{-1}$. We could thus require

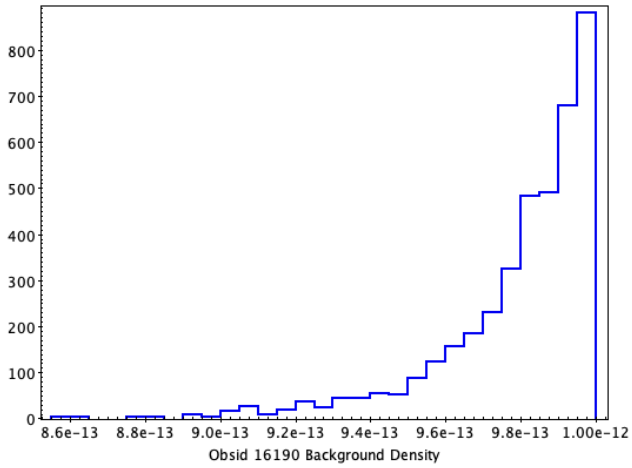


Figure 6: *Histogram of background draws distribution that exceeds P_{upper} of the background prior.*

that *CSC 2.1* fluxes and bounds less than this value be set to NULL. This could be done as fluxes are computed or in a simple migration step once *CSC 2.1* is complete.

7 Recommendations

I recommend that we adopt *pymc3* for *CSC 2.1* Aperture Photometry, following the procedures outlined in Listings $1-3$. Specifically,

1. All Aperture Photometry quantities computed in *CSC 2.0* will be computed in *CSC 2.1*;
2. All aperture definitions will remain the same with the exception that no background expansion will be done;
3. Scaling covariance matrices using *int_unc()* should be skipped; ad hoc seeding of the *MCMC* sampler is not needed in *pymc3*;
4. At least two independent *MCMC* chains should be computed for each *MPDF*, with 1000 draws and a comparable number of "burn-in" draws (see parameter "tune" in line 24 of Listing 1);
5. The distribution of draws from the combined chains will be examined to identify cases where source or background intensities are "pegged" at the upper range of their priors, using the prescription given in Equation 11; in such cases, P_{upper} for the offending prior should be increased by a factor of 10 and pymc3 rerun; if the draws distribution remains pegged: for photometry from single obsids, the photometry values should be set to NULL and the case noted; for photometry from multiple obsids, the offending obsid should be excluded and photometry recalculated;
6. The combined draws from all chains will be used to compute modes and bounds (including for upper limits) using the same procedures used in *CSC 2.0*;
7. Cases in which *pymc3* fail should be re-run with frozen background parameters; if they still fail, they should be re-run with background priors reduced by a factor of 10; if they still fail, the photometry should be set to NULL and the case noted;

8. For cases not identified as upper limits, computed energy fluxes and bounds should be greater than $\sim 5 \times 10^{-18} ergs - cm^{-2} - s^{-1}$ (TBR); if not, they should be set to NULL.

References

Martinéz-Galarza, R. 2020, An Improved Method for MCMC Sampling in CSC2 Aperture Photometry

Primini, F. 2013, Aperture Photometry Specifications for CSC R2

Primini, F. A., & Kashyap, V. L. 2014, ApJ, 796, 24

Primini, F. 2019, Specifications for Combining Data from Multiple ObsIDs in Aperture Photometry

Primini, F. 2020, Test Plan for PYMC3 Upgrade to Aperture Photometry