An Improved Method for MCMC Sampling in CSC2 Aperture Photometry

Rafael Martínez-Galarza (CfA) Date: 01/24/2020

1 Motivation

This specification document describes a novel approach for performing MCMC sampling of the posterior probability density functions resulting from aperture photometry analysis in the Chandra Source Catalog. The general algorithm, as described in this memo and in this memo, remains unchanged. That is, for the purposes of this document, the Bayesian formalism to determine the marginalized posterior probability density functions (MPDFs) for the photon flux in each band is described by equations (1) to (3) in the first memo above.

During catalog processing, the sampling of the MPDFs was done using MCMC as implemented in *Sherpa*, which uses a MCMC algorithm called *pyBLoCXS*, which is described here. Although *pyBLoCXS* has been shown to be successful and flexible for fitting of Chandra spectra, it is not optimized for aperture photometry. In addition, it uses a rather outdated approach to efficient MCMC sampling, namely a Metropolis-Hastings jumping rule with a t-distribution as proposal distribution. It also requires the user to define important parameters such as the factor by which the covariance matrix (which approximates the shape of the MPDF near the optimum) is scaled in order to provide a step size for the jump. This can result in very inefficient sampling in multi-variate parameter spaces with many degrees of freedom, as optimal step sizes can be different in each dimension (and also, depending on the complexity of the joint PDF, a covariance matrix is not always avaliable).

Even if the step sizes were optimized more appropriately, the Metropolis-Hastings jumping rule does not profit from recent improvements in MCMC sampling, that can significantly enhance the convergence efficiency by one or two orders of magnitude. In particular, *pyBLoCXS* does not use gradient-based methods for sampling. Gradient-based methods have been broadly adopted for many applications that require the sampling of complex, multi-dimensional parameter spaces. Their power rests in the fact that given any smooth function that describes the PDF, these methods use the gradient information at each step in order to propose an informed step in the direction of the optimum, rather than a random step. Newly developed methods of automatic differentiation are used to provide the gradients even if the function does not have an analytical formulation. This allow for extremely fast optimization and sampling.

The flaws of plain MCMC smapling with Metropolis-Hastings steps resulted in a significant number of CSC2 sources for which the *Sherpa* MCMC did not converge in aperture photometry. This was specially problematic for sources with a low number of counts, for which reliable covariance matrices could not be obtained. CSC2 scientists had to establish empirical correlations between approximated covariance diagonals obtained using alternative methods (such as int-unc) and the actual width of the PDF. These correlations were later applied to the results before the MCMC algorithm could be run again. Even so, there were a number of sources for which the source flux PDFs did not converge, as can be demonstrated by finding the data products for these sources in CSC2.

Here I propose the use of gradient-based methods for the MCMC sampling of aperture photometry MPDFs, and demonstrate how using them significantly improves convergence and effiency for those problematic sources. Because these methods require little parameter tunning, they should help in the automathization of catalog processing in the future.

2 The NUTS sampler

One of the most popular gradient-based methods is the No-U-Turn Sampling (NUTS, Hoffman & Gelman, 2011), which is an extension of the Hamiltonian Monte Carlo (HMC) algorithm that requires very little tuning of the hyperparamters by the user. HMC uses intuition from Hamiltonian dynamics in order to sample the posterior function by solving a dynamical system using leapfrog steps. Each new proposed steps comes from the resulting trajectory of a particle with a random initial kinetic energy across the probability landscape. The user needs to set both the size of the step for the leapfrog simulation and the total

number of steps before another proposal is considered. The NUTS algorithm makes it unnecessary to tune the number of steps in the trajectory calculation by setting a stopping criterion for the trajectory at each step, based in the direction of motion of the particle (hence the No-U-Turn name). The algorithm is extremely efficient at sampling the posteriors that result from Poisson-like likelihoods we are dealing with in aperture photometry (as well as other aspects of MLE fitting).

The *PyMC3* python package provides a very straightforward and modular way to define a probabilistic model, optimize it, and sample the resulting posterior PDFs. *PyMC3* uses a *theano* backend[1] in order to perform automatic differentiation, and allows for straightforward sampling using the NUTS algorithm. All the user needs to care about is defining the priors, the likelihood, and provide the data to be fitted.

Implementation

The implementation of the prototype is very straightforward and it requires minimum changes to the overall structure of the pipeline. Starting from the *prep3* files that contain the information of the counts and the F matrices that use the PSF franctions and exposure maps in oder to convert the counts to photon and energy fluxes, the algorithm builds a *PyMC3* model that defines uniform uninformative priors for the net source counts of each source in the bundle and the common background. It then defines the expected value for the total counts in each aperture (equation 3 in the orginal memo) as the sum of the contribution from each source in the bundle and the common background. Finally, it defines a Poisson likelihood, which can be expressed in terms of a Cash statistic (see code below). The model can be easily expanded to include contributions from more than one obsid to a given source.

The resulting model is then sampled using NUTS. All that needs to be specified is the number of samples in the chain. If more than one processor is available in the system where the code is being run, *PyMC3* will automatically start multiple MCMC chains, each with the number of samples requested. The tests performed here indicate that for a typical bundle with a handful of sources, single observation, it takes only 1000 samples to have all sources in the bundle converged, and about 5s of computing time in a typical desktop machine, to reach acceptable values of $\hat{r}$. The computing times does not increase dramatically when the joint model for multiple observations is computed. In particular, the convergence behavior is satisfactory for cases where there are no counts in the aperture. The model is able to interpret this as an upper limit and produces samples for the limiting PDF.

2.1 Inputs

The inputs required are the *prep3* files containing the number of total counts for each source in the bundle as well as the F matrices that convert the contributing counts in each aperture from each source to photon and energy fluxes, taking into account the PSF fractions and exposure maps. The *prep3* files have a column corresponding to the source aperture, and one column corresponding to the ecf90 area. This input is the same as for the *Sherpa* implementation.

2.2 Outputs

The outputs are the converged MCMC traces for each source in the bundle. Processing is done in a per-bundle basis, which means that the output for a particular run has as many traces as sources in the bundle. The code also returns a statistical summary for each trace, including the mean, the standard deviation, the MC error, 2.5% and 97.5% percentiles, and the $\hat{r}$ value to assess convergence. Plotting of the traces is modular and straightforward.

3 Tests

In order to test the performance of this implementation and its ability to reduce fine-tuning of hyper-parameters for corner cases in future releases of the catalog, I have applied this formulation to a sub-sample of the corner cases that initially did not converge during processing of CSC2, and for which a

[1] PyMC4 will switch to a TensorFlow backend, as support for *theano* is being discontinued

source	mean	sd	mc-error	$hpd_2.5$	$hpd_9 7.5$	$\hat{r}$
s_0	7.889459e-17	7.817299e-17	5.896827e-19	9.813239e-21	2.349844e-16	1.000025
s_1	1.919984e-15	6.037024e-16	4.591080e-18	8.214130e-16	3.127957e-15	1.000002
b	1.107955e-18	7.916104e-20	5.792091e-22	9.549496e-19	1.264129e-18	1.000327

Table 1: Summary of trace statistics for acisf$00972_0$02N020_b0511_s.

fine-tuning of the MCMC step size was needed. I describe the results below. Most of the cases, but not all of them, are in the very-low count regime, with some of the sources having 0 total counts in the aperture. *PyMC3* is able to understand that this is compatible with an upper limit, and produces a well converged trace for these cases. Using the different versions of the F matrices ('`PSF_UNEX`', '`PSF_FLUX`'), we obtain predictions for the number of counts and the energy fluxes, respectively.

We tested the code for the following corner case bundles, all of which had at least one source that failed to converge during the initial processing of CSC2:

```
acisf00972_000N020_b0252_b  acisf00972_002N020_b0511_s
acisf04698_000N021_b0103_b  acisf04698_000N021_b0103_m
acisf04699_000N021_b0103_b  acisf04700_000N021_b0103_b
acisf04701_001N021_b0103_b  acisf04702_000N021_b0103_b
acisf04703_000N021_b0103_b  acisf04704_001N021_b0103_b
acisf04705_000N021_b0103_b  acisf04708_000N021_b0103_b
acisf06403_001N020_b0511_s  acisf06420_000N020_b0252_b
acisf06420_000N020_b0511_s  acisf06421_000N020_b0252_b
acisf06421_000N020_b0511_s  acisf07188_000N021_b0030_h
acisf08460_000N020_b0252_b  acisf08460_000N020_b0511_s
acisf08461_000N020_b0252_b  acisf08461_000N020_b0511_s
acisf09555_000N021_b0103_b
```

In some cases (e.g., `acisf00972_002N020_b0511_s`, `acisf04698_000N021_b0103_m`), the pipeline MCMC algorithm failed to converge even after the parameters were tuned in reprocessing. As a result, these sources, as currently available in the catalog, have invalid MPDFs, with the values bein either zeroes or 'nans'.

3.1 Single obsid case

In figures 1 and 2, and in the tables below, I show some of the converged chains for these corner cases, starting with those for which convergence failed even after convergence fixes were attempted. For these shown results I used 5000 samples for each of the traces (and for each of the individual MCMC walkers), but the tests show that convergence is already achieved after about 1000 samples.

In order to make sure that we are consistent with CSC2 pipeline results that converged, in the following plots I compare some results for CSC2 converged sources vs. the *PyMC3* results. The goal here is to corroborate the the converged MPDFs look similar in both cases. We have done this for a few of the converged sources. Figures 3 and 4 show the results of this comparison.

Bundle `acisf08461_000N020_b0252_b` converged for all of the three sources it contains during CSC2 reprocessing, after the MCMC algorithm was tuned to fix convergence issues. Figure 3 and 4 show a comparison of the PDFs between the archive data and this new implementation. Both approaches agree. However, the pymc3 approach did not require any fine tuning of the step size or initial conditions in order to converge. It also converged much faster than the algorithm currently in the pipeline. In order to have an idea of the speed, consider that the pymc3 approach fits a bundle at a time, fitting all sources in a bundle simultaneously. The test data we have used is composed of 253 bundles, each of them having on average 3 sources, plus the background. It takes less than 40 seconds in a HEA desktop workstation to go over each of these bundles, using 5000 samples in each MCMC chain.

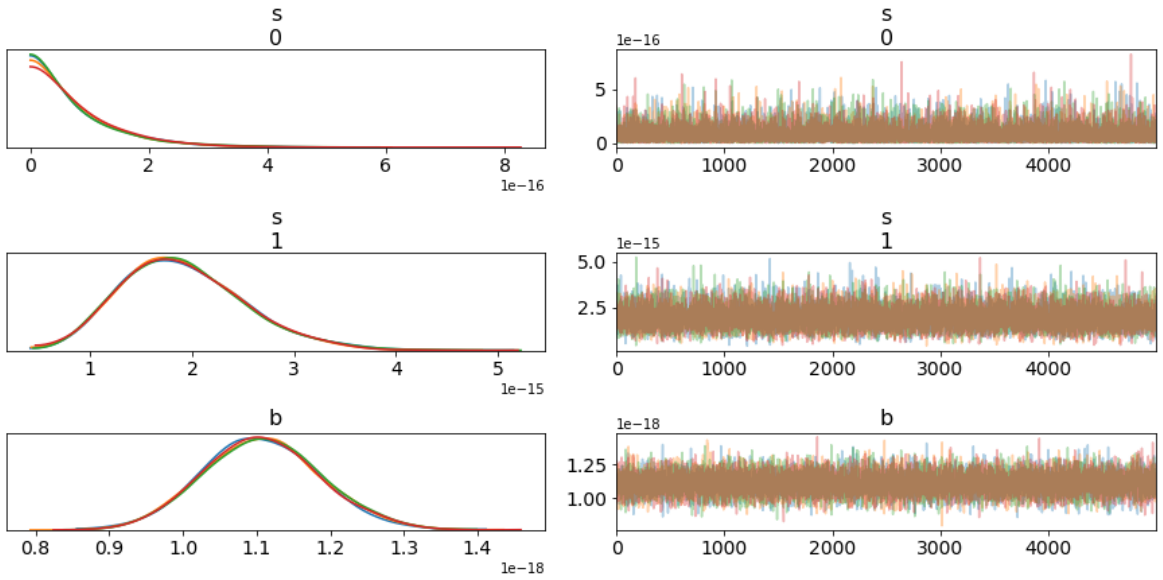


Figure 1: *PyMC3* results for bundle acisf00972$_0$02N020$_b$0511$_s$. The flux MPDFs are shown to the left, whereas the traces are shown to the right. Units are ergs^{-1}cm^{-2}. The first of the three sources shown had not converged in the CSC2 implementation.

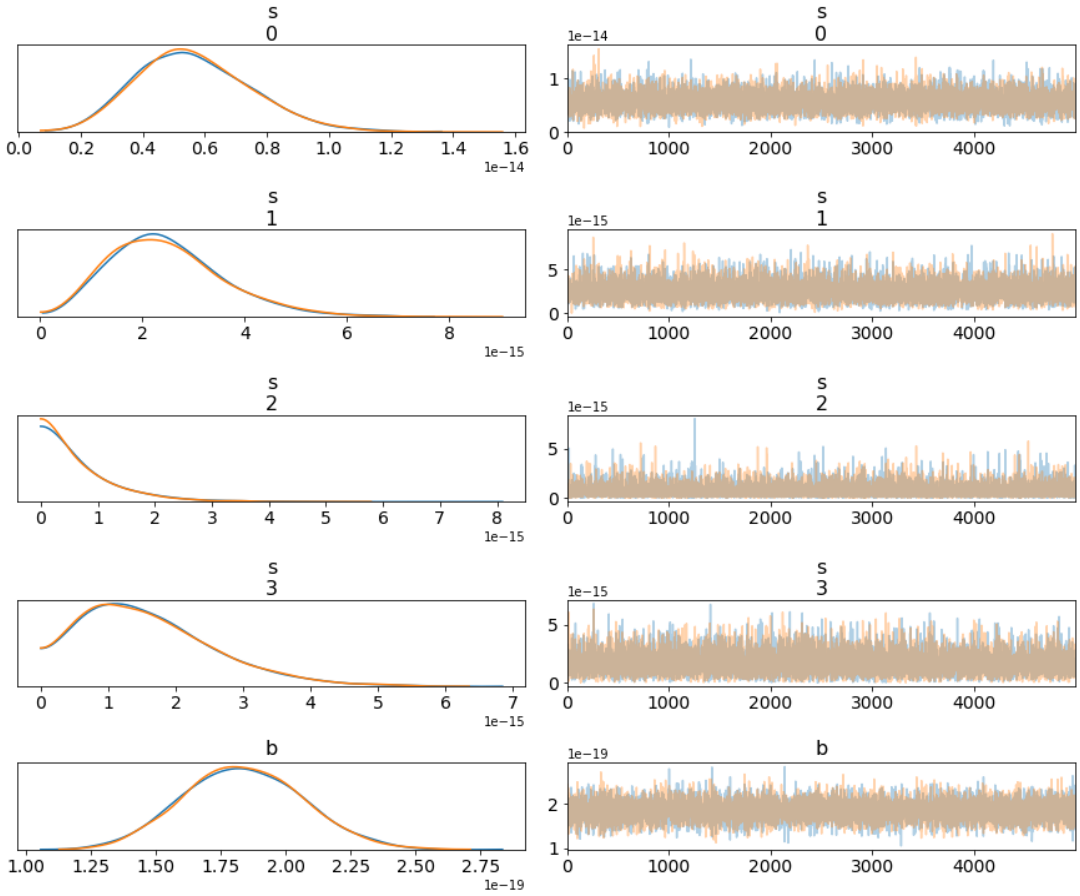


Figure 2: *PyMC3* results for bundle acisf04698$_0$00N021$_b$0103$_m$. The flux MPDFs are shown to the left, whereas the traces are shown to the right. Units are ergs^{-1}cm^{-2}. The third of the three sources shown had not converged in the CSC2 implementation.

source	mean	sd	mc-error	$hpd_2.5$	$hpd_9 7.5$	$\hat{r}$
s_0	5.696561e-15	1.840816e-15	1.695993e-17	2.316890e-15	9.296070e-15	0.999933
s_1	2.497005e-15	1.124688e-15	1.098092e-17	6.490148e-16	4.833312e-15	0.999968
s_2	7.042024e-16	7.056787e-16	6.467169e-18	2.311603e-20	2.080785e-15	1.000008
s_3	1.622625e-15	1.005664e-15	8.347785e-18	3.515415e-17	3.546822e-15	0.999914
b	1.839223e-19	2.277386e-20	2.361439e-22	1.391440e-19	2.274110e-19	0.999958

Table 2: Summary of trace statistics for acisf04698$_0$00N021$_\mathrm{b}$0103$_\mathrm{m}$.

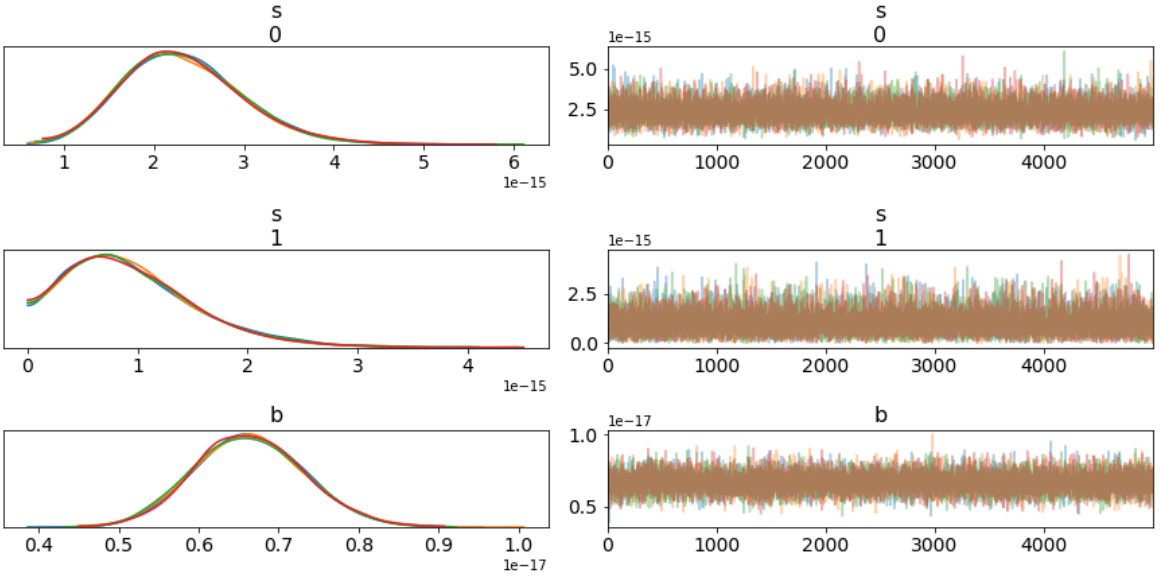


Figure 3: *PyMC3* results for bundle acisf08461$_0$00N020$_\mathrm{b}$0252$_\mathrm{b}$. The flux MPDFs are shown to the left, whereas the traces are shown to the right. Units are ergs^{-1}cm^{-2}. This is a case for which the two sources and the background converged in CSC2 reprocessing.

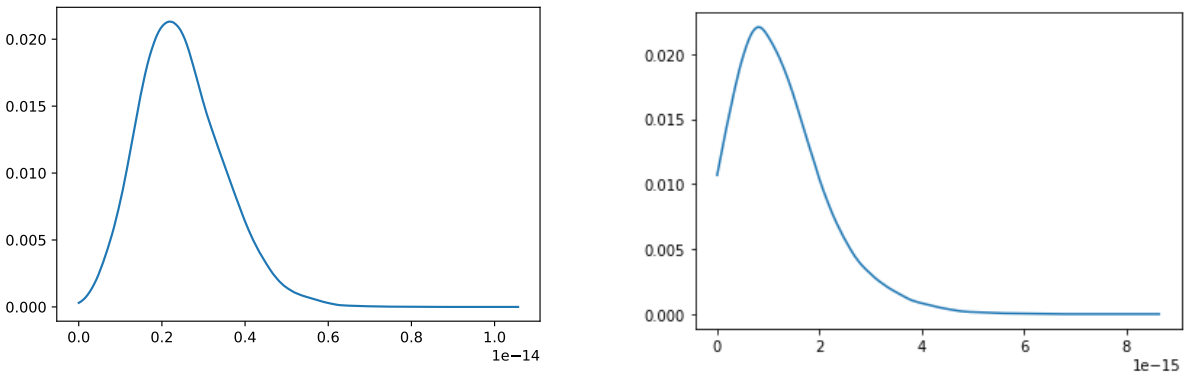


Figure 4: The archive MPDFs for the same two sources as in figure 3. Notice the agreement between the two methods.

For those detections with available MCMC draws in the catalog -i.e., those that converged-, we compare the mean value of the flux from the catalog database with our results from the present pymc3 prototype. We show the comparison in Fig. 5. We note that the average values are in reasonable

agreement with the current values in the catalog. The comparison of the PDFs show that also the errors are similar. Pymc3, however, offers the advantage of faster convergence, and fixes those cases that failed to converge in processing.

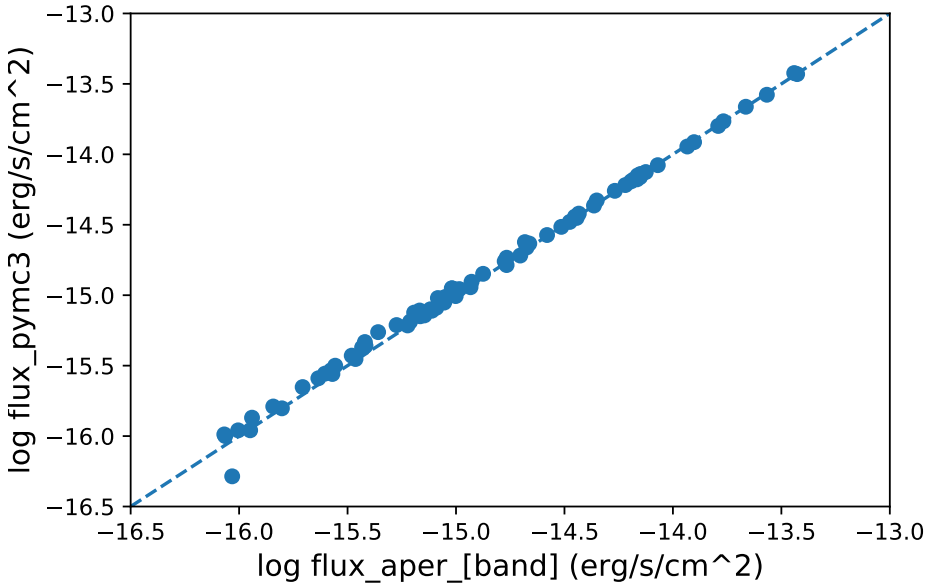


Figure 5: A comparison of the fluxes extracted from the CSC2 database ($\mathrm{flux_aper_{[band]}}$) and the mean fluxes extracted from the pymc3 MCMC traces for the same detections. We used detections that did have converged MCMC traces in the catalog.

3.2 Average flux case

Given a source detected in several obsids, we can obtain a more reliable measure of the source flux by combining the results from the individual obsids. In the catalog this is done by running the Bayesian algorithm using a model that has more as many flux parameters as we have obsids, using the measured counts in each observation as the target values to be fitted. In the catalog, the resulting posterior distributions for the average fluxes are returned as part of the *phot3* files. A similar implementation is very straightforward in pymc3. Such implementation is obtained by passing the measured counts in each obsid as an input, and then forcing the model to fit a single flux to all of them.

I now compare the results of the two approaches. In Fig. 6 I show the PDFs for the average flux corresponding to components 646 and 651 in obsid 8461, together with the PDF for the background (bottom panel). Fig. 7 shows the corresponding PDFs as obtained from the catalog. We note that the pymc3 approach gives narrower (by a factor of 2 or so) PDFs, which translates in more accurate average flux measurements. The mean values, however, are very similar between the two approaches.

4 Concluding remarks

I have demonstrated the power of the NUTS algorithm in obtaining optimization and MCMC samples of Bayesian posterior probability density functions for estimating aperture photometry of X-ray detections. The pymc3 approach requires very little fine-tuning of parameters, converges in practically all relevant cases, including upper limits and those detections that failed to converge during CSC2 processing, and coverges comparatively quickly with respect to the pyBLoCXS implementation. By comparing the results for a selection of CSC2 detections, I have demonstrated that the results from the new approach are in good agreement with the results in the catalog. In the case sources detected in more than one observations,

Pymc3 produces more accurate results than the current pipeline approach (error bars are a factor of 2 smaller). Pymc3 both optimizes (finds an optimum) and samples the posterior MPDFs in a seamless way. A similar approach could in principle be implemented for the MLE algorithm.

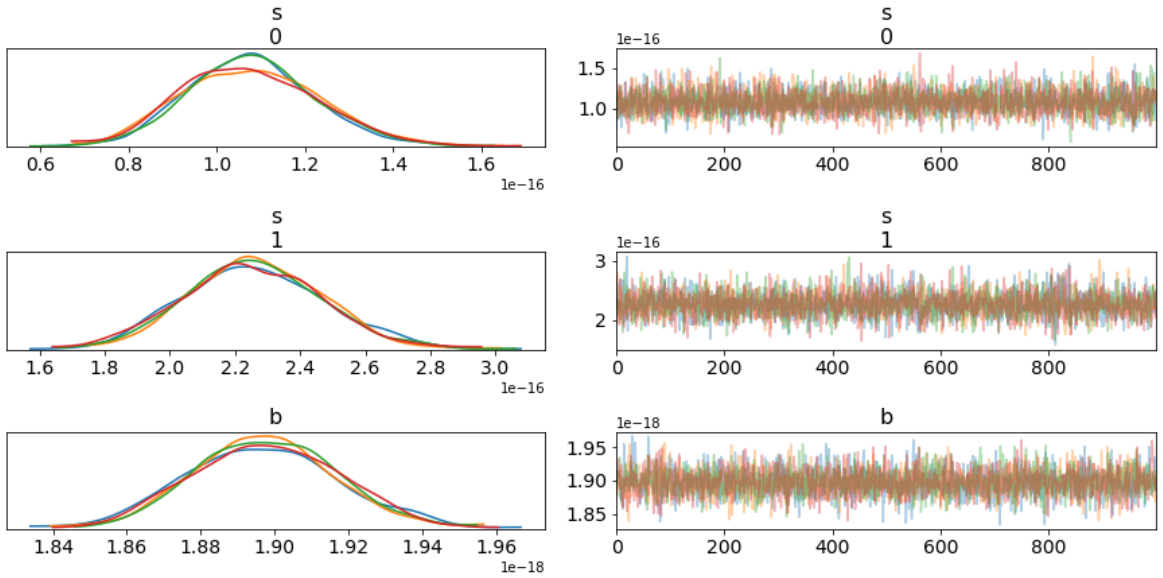
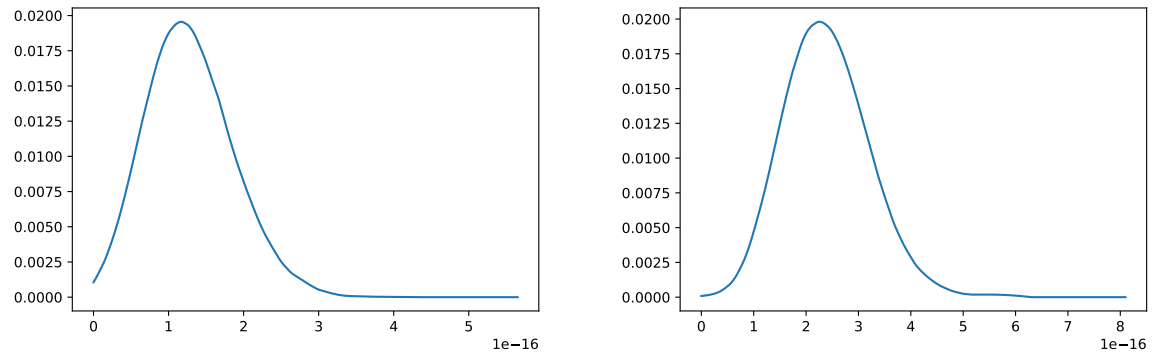


Figure 6: Average flux *PyMC3* results for bundle 511 in obsid 8461. The flux MPDFs are shown to the left, whereas the traces are shown to the right. Units are ergs^{-1}cm^{-2}. The top two panels correspond respectively to regions 646 and 651. The bottom panel corresponds to the background.



(a) 1b (b) 1a

Figure 7: The archive CSC2 MPDFs for the same two sources as in Fig. 6. Notice the agreement between the two methods in terms of the mean value, but the better accuracy obtained with the Pymc3, as shown by the narrower PDFs.

5 Code

Here is the full implementation for one particular bundle when it contains many observations.

```
import numpy as np
import matplotlib.pyplot as plt
import pymc3 as pm
```

```
from astropy.io import fits
import glob

theta = []
C = []
for file in glob.glob('*0103_b*prep3*'):
    print(file)
    hdul = fits.open(file)
    theta_comp = []
    for value in hdul[5].data['psf_matrix'].T:
        theta_comp.append([value[0],value[1],value[2],value[3],value[4]])  # Here you need to append
    C_comp = hdul[2].data['counts']
    theta_comp = np.array(theta_comp)
    theta.append(theta_comp)
    C.append(C_comp)

# Now we implement the PyMC3 model

basic_model = pm.Model()

with basic_model:

    # Priors for unknown model parameters
    s = pm.Uniform('s', lower=0.0, upper=1E-12, shape=(4,))
    b = pm.Uniform('b', lower=0.0, upper=1E-15)

# Expected value of outcome
    mu = []

    for i in range(len(theta)):
        mu.append(s[0]*theta[i][0] + s[1]*theta[i][1] + s[2]*theta[i][2] +
                  s[3]*theta[i][3] + b*theta[i][4]) # This also has to match the number of sources (
# Cash statistic
    def cash(observed_counts):
        logpm = np.array(mu - observed_counts*np.log(mu))
        return -2.0*np.sum(logpm)

# Likelihood (sampling distribution) of observations
    #C_obs = pm.Poisson('C_obs', mu=mu, observed=C)
    likelihood = pm.DensityDist("C_obs", cash, observed=C)

# Now we do MCMC sampling
with basic_model:
    trace = pm.sample(1000,tune=1000,target_accept=0.9)

# Plot traces
trarr = pm.traceplot(trace)
fig = plt.gcf()

# Print MCMC summary
summary = pm.stats.summary(trace)
summary
```