NPTFit: A code package for Non-Poissonian Template Fitting

Siddharth Mishra-Sharma,[1,*] Nicholas L. Rodd,[2,†] and Benjamin R. Safdi[2,‡]

[1] *Department of Physics, Princeton University, Princeton, NJ 08544*
[2] *Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA 02139*

(Dated: June 5, 2017)

We present NPTFit, an open-source code package, written in python and cython, for performing non-Poissonian template fits (NPTFs). The NPTF is a recently-developed statistical procedure for characterizing the contribution of unresolved point sources (PSs) to astrophysical data sets. The NPTF was first applied to *Fermi* gamma-ray data to give evidence that the excess of ∼GeV gamma-rays observed in the inner regions of the Milky Way likely arises from a population of sub-threshold point sources, and the NPTF has since found additional applications studying sub-threshold extra-galactic sources at high Galactic latitudes. The NPTF generalizes traditional astrophysical template fits to allow for the ability to search for populations of unresolved PSs that may follow a given spatial distribution. NPTFit builds upon the framework of the fluctuation analyses developed in X-ray astronomy, and thus likely has applications beyond those demonstrated with gamma-ray data. The NPTFit package utilizes novel computational methods to perform the NPTF efficiently. The code is available at https://github.com/bsafdi/NPTFit and up-to-date and extensive documentation may be found at http://nptfit.readthedocs.io.

I. INTRODUCTION

Astrophysical point sources (PSs), which are defined as sources with angular extent smaller than the resolution of the detector, play an important role in virtually every analysis utilizing images of the cosmos. It is useful to distinguish between resolved and unresolved PSs; the former may be detected individually at high significance, while members of the latter population are by definition too dim to be detected individually. However, unresolved PSs – due to their potentially large number density – can be a leading and sometimes pesky source of flux across wavelengths. Recently, a novel analysis technique called the non-Poissonian template fit (NPTF) has been developed for characterizing populations of unresolved PSs at fluxes below the detection threshold for finding individually-significant sources [1, 2]. The technique expands upon the traditional fluctuation analysis technique (see, for example, [3, 4]), which analyzes the aggregate photon-count statistics of a data set to characterize the contribution from unresolved PSs, by additionally incorporating spatial information both for the distribution of unresolved PSs and for the potential sources of non-PS emission. In this work, we present a code package called NPTFit for numerically implementing the NPTF in python and cython.

The most up-to-date version of the open-source package NPTFit may be found at

https://github.com/bsafdi/NPTFit

and the latest documentation at

http://nptfit.readthedocs.io.

In addition, the version used in this paper has been archived at

https://zenodo.org/record/380469#.WN_pSFPyvMV.

The NPTF generalizes traditional astrophysical template fits. Template fitting is useful for pixelated data sets consisting of some number of photon counts n_p in each pixel p, and it typically proceeds as follows. Given a set of model parameters $\boldsymbol{\theta}$, the mean number of predicted photon counts $\mu_p(\boldsymbol{\theta})$ in the pixel p may be computed. More specifically, $\mu_p(\boldsymbol{\theta}) = \sum_\ell T^{(S)}_{p,\ell}(\boldsymbol{\theta})$, where ℓ is an index of the set of templates $T^{(S)}_{p,\ell}$, whose normalizations and spatial morphologies may depend on the parameters $\boldsymbol{\theta}$. These templates may, for example, trace the gas-distribution or other extended structures that are expected to produce photon counts. Then, the probability to detect n_p photons in the pixel p is simply given by the Poisson distribution with mean $\mu_p(\boldsymbol{\theta})$. By taking a product of the probabilities over all pixels, it is straightforward to write down a likelihood function as a function of $\boldsymbol{\theta}$.

The NPTF modifies this procedure by allowing for non-Poissonian photon-count statistics in the individual pixels. That is, unresolved PS populations are allowed to be distributed according to spatial templates, but in the presence of unresolved PSs the photon-count statistics in individual pixels, as parameterized by $\boldsymbol{\theta}$, no longer follow Poisson distributions. This is heuristically because we now have to ask two questions in each pixel: first, what is the probability, given the model parameters $\boldsymbol{\theta}$ that now also characterize the intrinsic source-count distribution of the PS population, that there are PSs within the pixel p, then second, given that PS population, what is the probability to observe n_p photons?

It is important to distinguish between resolved and unresolved PSs. Once a PS is resolved – that is once its location and flux is known – that PS may be accounted for

* smsharma@princeton.edu
† nrodd@mit.edu
‡ bsafdi@mit.edu

by its own Poissonian template. Unresolved PSs are different because their locations and fluxes are not known. When we characterize unresolved PSs with the NPFT, we characterize the entire population of unresolved sources, following a given spatial distribution, based on how that population modifies the photon-count statistics.

The NPTF has played an important role recently in addressing various problems in gamma-ray astroparticle physics with data collected by the *Fermi*-LAT gamma-ray telescope.[1] The NPTF was developed to address the excess of gamma rays observed by *Fermi* at ∼GeV energies originating from the inner regions of the Milky Way [5–18]. The GeV excess, as it is commonly referred to, has received a significant amount of attention due to the possibility that the excess emission arises from dark matter (DM) annihilation. However, it is well known that unresolved PSs may complicate searches for annihilating DM in the Inner Galaxy region due to, for example, the expected population of dim pulsars [12, 19–26]. In [2] (see also [27]) it was shown, using the NPTF, that indeed the photon-count statistics of the data prefer a PS over a smooth DM interpretation of the GeV excess. The same conclusion was also reached by [28] using an unrelated method that analyzes the statistics of peaks in the wavelet transformation of the *Fermi* data.

In the case of the GeV excess, there are multiple PS populations that may contribute to the observed gamma-ray flux and complicate the search for DM annihilation. These include isotropically distributed PSs of extragalactic origin, PSs distributed along the disk of the Milky Way such as supernova remnants and pulsars, and a potential spherical population of PSs such as millisecond pulsars. Additionally, there are various identified PSs that contribute significantly to the flux as well as a variety of smooth emission mechanisms such as gas-correlated emission from pion decay and bremsstrahlung. The power of the NPTF is that these different source classes may be given separate degrees of freedom and constrained by incorporating the spatial morphology of their various contributions along with the difference in photon-count statistics between smooth emission and emission from unresolved PSs. Although the origin of the GeV excess is still not completely settled, as even if the excess arises from PSs as the NPTF suggests the source class of the PSs remains a mystery at present, the NPTF has emerged as a powerful tool for analyzing populations of dim PSs in complicated data sets with characteristic spatial morphology.

The NPTF and related techniques utilizing photon-count statistics have also been used recently to study the contribution of various source classes to the extragalactic gamma-ray background (EGB) [4, 29–32].[2] In these works it was shown that unresolved blazars would predominantly show up as PS populations under the NPTF, while other source classes such as star-forming galaxies would show up predominantly as smooth emission. For example, in [32] it was shown using the NPTF that blazars likely account for the majority of the EGB from ∼2 GeV to ∼2 TeV. These results set strong constraints on the flux from more diffuse sources, such as star-forming galaxies, which has significant implications for, among other problems, the interpretation of the high-energy astrophysical neutrinos observed by IceCube [34–37] (see, for example, [38, 39]). This is because certain sources that contribute gamma-ray flux at *Fermi* energies, such as star forming galaxies and various types of active galactic nuclei, may also contribute neutrino flux observable by IceCube.

The NPTF originates from the older fluctuation analysis technique, which is sometimes referred to as the $P(D)$ analysis. This technique has been used extensively to study the flux of unresolved X-ray sources [3, 40–43]. In these early works, the photon-count probability distribution function (PDF) was computed numerically for different PS source-count distributions using Monte Carlo (MC) techniques. The fluctuation analysis was first applied to gamma-ray data in [4],[3] and in that work the authors developed a semi-analytic technique utilizing probability generating functions for calculating the photon-count PDF. The code package `NPTFit` presented in this work uses this formalism for efficiently calculating the photon-count PDF. The specific form of the likelihood function for the NPTF, while reviewed in this work, was first presented in [2]. The works [2, 27, 32] utilized an early version of `NPTFit` to perform their numerical analyses.

The `NPTFit` code package has a `python` interface, though the likelihood evaluation is efficiently implemented in `cython` [46]. The user-friendly interface allows for an arbitrary number of PS and smooth templates. The PS templates are characterized by pixel-dependent source-count distributions $dN_p/dF = T_p^{(\mathrm{PS})} dN/dF$, where $T_p^{(\mathrm{PS})}$ is the spatial template tracking the distribution of point sources on the sky and dN/dF is the pixel-independent source-count distribution. The distribution dN_p/dF quantifies the number of sources dN_p that contributes flux between F and $F + dF$ in the pixel p. The dN/dF are parameterized as multiply broken power-laws, with an arbitrary number of breaks. The code is able to account for both an arbitrary exposure map (accounting for the pointing strategy of an instrument) as well as an arbitrary point spread function (PSF, accounting for the instrument's finite angular resolution) in translating between flux F and counts S.

`NPTFit` has a built-in interface with `MultiNest` [47, 48], which efficiently implements nested sampling of the pos-

[1] http://fermi.gsfc.nasa.gov/

[2] The complementary analysis strategy of probabilistic catalogues has also been applied to this problem [33].

[3] The fluctuation analysis has more recently been applied to both gamma-ray [44] and neutrino [45] datasets.

terior distribution and Bayesian evidence for the user-specified model, given the specified data and instrument response function, in the Bayesian framework [49–51]. The interface handles the Message Passing Interface (MPI), so that inference may be performed efficiently using parallel computing. A basic analysis package is provided in order to facilitate easy extraction of the most relevant data from the posterior distribution and quick plotting of the `MultiNest` output. The preferred format of the data for `NPTFit` is `HEALPix` [52] (a nested equal-area pixilation scheme of the sky), although the the code is also able to handle non-`HEALPix` data arrays. Note that the code package may also be used to simply extract the NPTF likelihood function so that `NPTFit` may be interfaced with any numerical package for Bayesian or frequentist inference.

A large set of example `Jupyter` [53] notebooks and `python` files are provided to illustrate the code. The examples utilize 413 weeks of processed *Fermi* Pass 8 data in the UltracleanVeto event class collected between August 4, 2008 and July 7, 2016 in the energy range from 2 to 20 GeV. We restrict this dataset to the top quartile as graded by PSF reconstruction and further apply the standard quality cuts `DATA_QUAL==1 && LAT_CONFIG==1`, as well as restricting the zenith angle to be less than 90°. This data is made available in the code release. Moreover, the example notebooks illustrate many of the main results in [2, 27, 32].

In addition to the above, the base `NPTFit` code makes use of the `python` packages `corner` [54], `matplotlib` [55], `mpmath` [56], `GSL` [57] and `numpy` [58].

The rest of this paper is organized as follows. Section II outlines in more detail the framework of the NPTF. Section III highlights the key classes and features in the `NPTFit` code package and usage instructions. In Sec. IV we present an example of how to perform an NPTF scan using `NPTFit`, looking at the Galactic Center with *Fermi* data to reproduce aspects of the main results of [2]. We conclude in Sec. V. Appendices A, B, and C describe further details behind the mathematical framework of the NPTF.

II. THE NON-POISSONIAN TEMPLATE FIT

In this section we review the NPTF, which was first presented in [2] and described in more detail in [27, 32] (see also [1, 4, 29, 31]). The NPTF is used to fit a model $\mathcal{M}$ with parameters $\boldsymbol{\theta}$ to a data set d consisting of counts n_p in each pixel p. The likelihood function for the NPTF is then simply

$$p(d|\boldsymbol{\theta}, \mathcal{M}) = \prod_p p^{(p)}_{n_p}(\boldsymbol{\theta}) \,, \tag{1}$$

where $p^{(p)}_{n_p}(\boldsymbol{\theta})$ gives the probability of drawing n_p counts in the given pixel p, as a function of the parameters $\boldsymbol{\theta}$. The main computational challange, of course, is in computing these probabilities.

It is useful to divide the model parameters into two different categories: the first category describes smooth templates, while the second category describes PS templates. We describe each category in turn, starting with the smooth templates.

For most applications, the data has the interpretation of being a two-dimensional pixelated map consisting of an integer number of counts in each pixel. The smooth templates may be used to predict the mean number of counts $\mu_p(\boldsymbol{\theta})$ in each pixel p:

$$\mu_p(\boldsymbol{\theta}) = \sum_\ell \mu_{p,\ell}(\boldsymbol{\theta}) \,. \tag{2}$$

Above, ℓ is an index over templates and $\mu_{p,\ell}(\boldsymbol{\theta})$ denotes the mean contribution of the $\ell^{\rm th}$ template to pixel p for parameters $\boldsymbol{\theta}$. In principle, $\boldsymbol{\theta}$ may describe both the spatial morphology as well as the normalization of the templates. However, in the current implementation of the code, the Poissonian model parameters simply characterize the overall normalization of the templates: $\mu_{p,\ell}(\boldsymbol{\theta}) = A_\ell(\boldsymbol{\theta}) T^{(S)}_{p,\ell}$. Here, A_ℓ is the normalization parameter and $T^{(S)}_{p,\ell}$ is the $\ell^{\rm th}$ template, which takes values over all pixels p and is independent of the model parameters. The superscript (S) implies that the template is a counts templates, which is to be contrasted with a flux template, for which we use the symbol (F). The two are related by the exposure map of the instrument E_p: $T^{(S)}_p = E_p T^{(F)}_p$. In the case where we only have smooth, Poissonian templates, the probabilities are then given by the Poisson distribution:

$$p^{(p)}_{n_p}(\boldsymbol{\theta}) = \frac{\mu_p^{n_p}(\boldsymbol{\theta})}{n_p!} e^{-\mu_p(\boldsymbol{\theta})} \,. \tag{3}$$

In the presence of unresolved PS templates, the probabilities $p^{(p)}_{n_p}(\boldsymbol{\theta})$ are no longer Poissonian functions of the model parameters $\boldsymbol{\theta}$. Each PS template is characterized by a pixel-dependent source-count distribution dN_p/dF, which describes the differential number of sources per pixel per unit flux interval. In this work, we model the source-count distribution by a multiply broken power-law:

$$\frac{dN_p}{dF}(F;\boldsymbol{\theta}) = A(\boldsymbol{\theta})T_p^{(\mathrm{PS})} \begin{cases} \left(\frac{F}{F_{b,1}}\right)^{-n_1}, & F \geq F_{b,1} \\ \left(\frac{F}{F_{b,1}}\right)^{-n_2}, & F_{b,1} > F \geq F_{b,2} \\ \left(\frac{F_{b,2}}{F_{b,1}}\right)^{-n_2}\left(\frac{F}{F_{b,2}}\right)^{-n_3}, & F_{b,2} > F \geq F_{b,3} \\ \left(\frac{F_{b,2}}{F_{b,1}}\right)^{-n_2}\left(\frac{F_{b,3}}{F_{b,2}}\right)^{-n_3}\left(\frac{F}{F_{b,3}}\right)^{-n_4}, & F_{b,3} > F \geq F_{b,4} \\ \ldots & \ldots \\ \left[\prod_{i=1}^{k-1}\left(\frac{F_{b,i+1}}{F_{b,i}}\right)^{-n_{i+1}}\right]\left(\frac{F}{F_{b,k}}\right)^{-n_{k+1}}, & F_{b,k} > F \end{cases} . \tag{4}$$

Above, we have parameterized the source-count distribution with an arbitrary number of breaks k, denoted by $F_{b,i}$ with $i \in [1, 2, \ldots, k]$, and $k+1$ indices n_i with $i \in [1, 2, \ldots, k+1]$. The spatial dependence of the source-count distribution is accounted for by the overall factor $A(\boldsymbol{\theta})T_p^{(\mathrm{PS})}$, where $A(\boldsymbol{\theta})$ is the pixel-independent normalization, which is a function of the model parameters, and $T_p^{(\mathrm{PS})}$ is a template describing the spatial distribution of the PSs. More precisely, the number of sources $N_p^{\mathrm{PS}} = \int dF dN_p/dF$ (and the total PS flux $F_p^{\mathrm{PS}} = \int dF F dN_p/dF$) in pixel p, for a fixed set of model parameters $\boldsymbol{\theta}$, follows the template $T_p^{(\mathrm{PS})}$. On the other hand, the locations of the flux breaks and the indices are taken to be fixed between pixels.[4]

To summarize, a PS template described by a broken power-law with k breaks has $2(k+1)$ model parameters describing the locations of the breaks, the power-law indices, and the overall normalization. For example, if we take a single break then the PS model parameters may be denoted as $\{A, F_{b,1}, n_1, n_2\}$. Additionally, a spatial template $T^{(\mathrm{PS})}$ must be specified, which describes the distribution of the number of sources (and total flux) with pixel p.

Notice that when we discussed the Poissonian templates we used the counts templates $T^{(S)}$ and talked directly in terms of counts S, while so far in our discussion of the unresolved PS templates we have used the point source distribution template $T^{(\mathrm{PS})}$ and written the source-count distribution dN/dF in terms of flux F. Of course as the total flux from a distribution of point sources is also proportional to the template $T^{(\mathrm{PS})}$, it can be thought of as a flux template, however conceptually it is being used to track the distribution of the sources rather than the flux they produce. For this reason we have chosen to distinguish the two. Moreover, in the presence of a non-trivial PSF, $T^{(S)}$ should also be smoothed by the PSF to account for the instrument response function. That is, $T^{(S)}$ is a template for the observed counts taking into account the details of the instrument, while $T^{(\mathrm{PS})}$ ($T^{(F)}$) is a map of the physical point sources (flux), which is independent of the instrument. In photon-counting applications, the exposure map E_p often has units of $\mathrm{cm}^2\mathrm{s}$ and flux has units of counts $\mathrm{cm}^{-2}\mathrm{s}^{-1}$.

For the unresolved PS templates, we also need to convert the source-count distribution from flux to counts. This is done by a simple change of variables:

$$\frac{dN_p}{dS}(S;\boldsymbol{\theta}) = \frac{1}{E_p}\frac{dN_p}{dF}(F = S/E_p;\boldsymbol{\theta}) , \tag{5}$$

which implies that for a non-Poissonian template the spatial dependence of dN_p/dS is given by $T_p^{(\mathrm{PS})}/E_p$. This inverse exposure scaling may seem surprising, but it is straightforward to confirm that the mean number of counts in a given pixel, $\int dS S dN_p/dS$, is given by $E_p T_p^{(\mathrm{PS})}$, as expected, up to pixel independent factors.

As an important aside, the template $T^{(S)}$ used by the Poissonian models needs to be smoothed by the PSF. Incorporating the PSF into the unresolved PS models, on the other hand, is more complicated and is not accomplished simply by smoothing the spatial template. Indeed, $T_p^{(\mathrm{PS})}$ should remain un-smoothed by the PSF when used for non-Poissonian scans.

In the remainder of this section we briefly overview the mathematic framework behind the computation of the $p_{n_p}^{(p)}(\boldsymbol{\theta})$ with `NPTFit`; however, details of the algorithms used to calculate these probabilities in practice, along with more in-depth explanations, are given in Apps. A, B, and C. We use the probability generating function formalism, following [4], to calculate the probabilities. For a discrete probability distribution p_k, with $k = 0, 1, 2, \ldots$, the generating function is defined as:

$$P(t) \equiv \sum_{k=0}^{\infty} p_k t^k , \tag{6}$$

from which we can recover the probabilities:

$$p_k = \frac{1}{k!}\left.\frac{d^k P(t)}{dt^k}\right|_{t=0} . \tag{7}$$

[4] In principle, the breaks and indices could also vary between pixels. However, in the current version of `NPTFit`, only the number of sources (and, accordingly, the total flux) is allowed to vary between pixels.

The key feature of generating functions exploited here is that the generating function of a sum of two independent random variables is simply the product of the individual generating functions.

The probability generating function for the smooth templates, as a function of $\boldsymbol{\theta}$, is simply given by

$$P_{\rm P}(t;\boldsymbol{\theta}) = \prod_p \exp\left[\mu_p(\boldsymbol{\theta})(t-1)\right] . \tag{8}$$

The probability generating function for an unresolved PS template, on the other hand, takes a more complicated form:

$$P_{\rm NP}(t;\boldsymbol{\theta}) = \prod_p \exp\left[\sum_{m=1}^{\infty} x_{p,m}(\boldsymbol{\theta})(t^m-1)\right] , \tag{9}$$

where

$$x_{p,m}(\boldsymbol{\theta}) = \int_0^\infty dS \frac{dN_p}{dS}(S;\boldsymbol{\theta}) \int_0^1 df \rho(f) \frac{(fS)^m}{m!} e^{-fS} . \tag{10}$$

Above, $\rho(f)$ is a function that takes into account the PSF, which we describe in more detail in App. A. In the presence of a non-trivial PSF, the flux from a single source is smeared among pixels. The distribution of flux fractions among pixels is described by the function $\rho(f)$, where f is the flux fraction. By definition $\rho(f)df$ equals the number of pixels which, on average, contain between f and $f+df$ of the flux from a PS; the distribution is normalized such that $\int_0^1 df f\rho(f) = 1$. If the PSF is a δ-function, then $\rho(f) = \delta(f-1)$.

Putting aside the PSF correction for the moment, the $x_{p,m}$ have the interpretation of being the average number of m-count PSs within the pixel p, given the distribution $dN_p(S;\boldsymbol{\theta})/dS$. The generating function for x_m m-count sources is simply $e^{x_m(t^m-1)}$ (see [4] or App. A), which then leads directly to (9). The PSF correction, through the distribution $\rho(f)$, incorporates the fact that PSs only contribute some fraction of their flux within a given pixel.

III. NPTFIT: ORIENTATION

`NPTFit` implements the NPTF, as described above, in `python`. In this section we give a brief orientation to the code package and its main classes. A more thorough description of the code and its uses is available in the online documentation.

class NPTFit.nptfit.NPTF

This is the main class used to set up and perform non-Poissonian and Poissonian template scans. It is initialized by

```
nptf = NPTF(tag='Untagged',work_dir=None)
```

with keywords

Argument	Default	Purpose	type
tag	'Untagged'	Label of scan	str
work_dir	None	Output directory	str

If no `work_dir` is specified, the code will default to the current directory. This is the directory where all output is stored. Specifying a `tag` will create an additional folder, with that name, within the `work_dir` for the output.

The data, exposure map, and templates are loaded into the `nptfit.NPTF` instance after initialization (see the example in Sec. IV). The data and exposure map are loaded by

```
nptf.load_data(data, exposure)
```

Here, `data` and `exposure` are 1-D `numpy` arrays. The recommended format for these arrays is the `HEALPix` format, so that all pixels are equal area, although the code is able to handle arbitrary data and exposure arrays so long as they are of the same length. The templates are added by

```
nptf.add_template(template, key,
    units='counts')
```

Here, `template` is a 1-D `numpy` array of the same length as the data and exposure map, `key` is a string that will be used to refer to the template later on, and `units` specifies whether the template is a counts template (keyword `'counts'`) or a flux template (keyword `'flux'`) in units counts $\mathrm{cm}^{-2}\mathrm{s}^{-1}$. The default, if unspecified, is `units = 'counts'`. The template should be pre-smoothed by the PSF if it is going to be used for a Poissonian model. If the template is going to be used for a non-Poissonian model, either choice for `units` is acceptable, though in the case of `'counts'` the template should simply be the product of the exposure map times the flux template and not smoothed by the PSF.

The user also has the option of loading in a mask that reduces the region of interest (ROI) to a subset of the pixels in the data, exposure, and template arrays. This is done through the command

```
nptf.load_mask(mask)
```

where `mask` is a boolean `numpy` array of the same length as the data and exposure arrays. Pixels in `mask` should be either `True` or `False`; by convention, pixels that are `True` will be masked, while those that are `False` will not be masked. Note if performing an analysis with non-Poissonian templates, regions where the exposure map is identically zero should be explicitly masked.

Afterwards, Poissonian and non-Poissonian models may be added to the instance using the available templates. An arbitrary number of Poissonian and non-Poissonian models may be added to the scan. Moreover, each non-Poissonian model may be specified in terms of a multiply broken power law with a user-specified number of breaks, as in (4).

Poissonian models are added sequentially using the syntax

```
nptf.add_poiss_model(template_name,
    model_tag, prior_range=[], log_prior=
    False, fixed=False, fixed_norm=1.0)
```

where the keywords are

Argument	Default	Purpose	type
template_name	-	key of template	str
model_tag	-	LaTeX-ready label	str
prior_range	[]	Prior [min, max]	[float, float]
log_prior	False	Log/linear-flat prior	bool
fixed	False	Is template fixed	bool
fixed_norm	1.0	Norm if fixed	float

Any of the model parameters may be fixed to a user specified value instead of floated in the scan. For those parameters that are floated in the scan, a prior range needs to be specified along with whether or not the prior is flat or log-flat. Note that if `log_prior = True`, then the prior range is set with respect to $\log_{10}$ of the linear prior range.[5] For example, if we want to scan the normalization of a template over the range from $[0.1, 10]$ with a log-flat prior, then we would set `log_prior = True` and `prior_range = [-1,1]`. In this case, it might make sense to label the model with `model_tag = '$\log_{10}A$'` to emphasize that the actual model parameter is the log of the normalization; this label will appear in various plots made using the provided analysis class for visualizing the posterior.

The non-Poissonian models are added with a similar syntax:

```
nptf.add_non_poiss_model(template_name,
    model_tag, prior_range=[], log_prior=
    False, dnds_model='specify_breaks',
    fixed_params=None, units='counts')
```

The `template_name` keyword is the same as for the Poissonian models. The rest of the keywords are

Argument	Default	Purpose	type
model_tag	-	LaTeX-ready label	[str, str, ...]
prior_range	[]	Prior [[min, max], ...]	[[float, float], ...]
log_prior	[False]	Log/linear-flat prior	[bool,bool, ...]
dnds_model	'specify_breaks'	How to specify multiple breaks	str
fixed_params	None	Fix certain parameters	[[int,float], ...]
units	'counts'	'flux' or 'counts' units for breaks	str

The syntax for adding non-Poissonian models is that the model parameters are specified by $[A, n_1, n_2, \ldots, n_{k+1}, S_{b,1}, S_{b,2}, \ldots, S_{b,k}]$ for a broken power-law with k breaks. As such, the `model_tag`, `prior_range`, and `log_prior` are now arrays where each entry refers to the respective model parameter. The code automatically determines the number of breaks by the length of the `model_tag` array. The arrays `prior_range` and `log_prior` should only include entries for model parameters that will be floated in the scan. Any model parameter may be fixed using the `fixed_params` array, with the syntax such that `fixed_params = [[i,c_i],[j, c_j]]` would fix the i^{th} model parameter to c_i and the j^{th} to c_j, where the parameter indexing starts from 0.

The `units` keyword determines whether the priors for the breaks in the source-count distribution (and also the fixed parameters, if any are given) will be specified in terms of `'flux'` or `'counts'`. The relation between flux and counts varies between pixels if the exposure map is non-trivial. For this reason, it is more appropriate to think of the breaks in the source-count distribution in terms of flux. The keyword `'counts'` still specifies the breaks in the source-count distribution in terms of flux, with the relation between counts and flux given through the mean of the exposure map mean(E): $F_{b,i} = S_{b,i}/\text{mean}(E)$.

The `dnds_model` keyword has the options `'specify_breaks'` and `'specify_relative_breaks'`. If `'specify_breaks'` is chosen, which is the default, then the breaks are the model parameters. If instead `'specify_relative_breaks'` is chosen, the full set of model parameters is given by $[A, n_1, n_2, \ldots, n_{k+1}, S_{b,1}, \lambda_2, \ldots, \lambda_k]$. Here, $S_{b,1}$ is the highest break and the lower breaks are determined by $S_{b,i} = \lambda_i S_{b,i-1}$. Note that the prior ranges for the λ's should be between 0 and 1 (for linear flat), since $S_{b,i} < S_{b,i-1}$.

After setting up a scan, the configuration is finished by executing the command

```
nptf.configure_for_scan(f_ary=[1.0],
    df_rho_div_f_ary=[1.0], nexp=1)
```

For a purely Poissonian scan, none of the keywords above need to be specified. For non-Poissonian scans, `f_ary` and `df_rho_div_f_ary` incorporate the PSF correction. In particular, `f_ary` is a discretized list of f values between 0 and 1, while `df_rho_div_f_ary` is a discretized list of $df\rho(f)/f$ at those f values. A class is provided for

[5] More complicated priors will be incorporated in future releases of `NPTFit`.

computing these lists; it is described later in this section. If no keywords are given for these two arrays they default to the case of a δ-function PSF.

The keyword `nexp`, which defaults to 1, is related to the exposure correction in the calculation of the source-count distribution dN_p/dS from dN_p/dF. In many applications, it is computationally too expensive to perform the mapping in (5) in each pixel. The overall pixel-dependent normalization factor $T_p^{(\rm PS)}/E_p$ factorizes from many of the internal computations, and as a result this contribution to the exposure correction is performed in every pixel. However, it is useful to perform the mapping from flux to counts, which should be performed uniquely in each pixel $F = S/E_p$, using the mean exposure within small sub-regions. Within a given sub-region, we map flux to counts using $F = S/\text{mean}(E)$, where the mean is taken over all pixels in the sub-region. The number of sub-regions is given by `nexp`, and all sub-regions have approximately the same area. As `nexp` approaches the number of pixels, the approximation becomes exact; however, for many applications the approximation converges for a relatively small number of exposure regions. We recommend verifying, in any application, that results are stable as `nexp` is increased.

After configuring the `NPTF` instance, the log-likelihood may be extracted, as a function of the model parameters, in addition to the prior range. The log-likelihood and prior range may then be used with any external package for performing Bayesian or frequentist inference. This is particularly useful if the user would like to combine likelihood functions between different energy bins or otherwise add to the default likelihood function, for example, incorporating nuisance parameters beyond those associated with individual templates. The package `MultiNest`, however, is already incorporated into the `NPTF` class and may be run immediately after configuring the `NPTF` instance. This is done simply by executing the command

```
nptf.perform_scan(run_tag=None,nlive=100)
```

where `nlive` is an integer that specifies the number of live points used in the sampling of the posterior distribution. `MultiNest` recommends an `nlive` $\sim$500-1000, though the parameter defaults to 100 if unspecified for quick test runs. Additional `MultiNest` arguments may be passed as a dictionary through the optional `pymultinest_options` keyword (see the online documentation for more details). The optional keyword `run_tag` is used to create a subfolder for the `MultiNest` output with that name.

After a scan has been run (or if a scan has been run previously and saved), the results may be loaded through the command

```
nptf.load_scan(run_tag=None)
```

The `MultiNest` chains, which give a discretized view of the posterior distribution, may then be accessed through, for example, `nptf.samples`. An instance of the `PyMultiNest` analyzer class may be accessed through `nptf.a`. A small analysis package, described later in this section, is also provided for performing a few common analyses.

`class NPTFit.psf_correction.PSFCorrection`

This is the class used to construct the arrays `f_ary` and `df_rho_div_f_ary` for the PSF correction. An instance of `PSFCorrection` is initialized through

```
pc_inst = PSFCorrection.PSFCorrection(
    psf_dir=None, num_f_bins=10, n_psf
    =50000, n_pts_per_psf=1000, f_trunc
    =0.01, nside=128, psf_sigma_deg=None,
    delay_compute=False)
```

with keywords

Argument	Default	Purpose	type
`psf_dir`	`None`	Where PSF arrays are stored	`str`
`num_f_bins`	10	Number of linear-spaced points in`f_ary`	`int`
`n_psf`	50000	Number of MC simulations for determining `df_rho_div_f_ary`	`int`
`n_pts_per_psf`	1000	Number of points drawn for each MC simulation	`int`
`f_trunc`	0.01	Minimum f value	`float`
`nside`	128	`HEALPix` parameter for size of map	`int`
`psf_sigma_deg`	`None`	Standard deviation σ of 2-D Gaussian PSF	`float`
`delay_compute`	`False`	If `True`, PSF not Gaussian and will be specified later	`bool`

Note that the arrays `f_ary` and `df_rho_div_f_ary` depend both on the PSF of the detector as well as the pixelation of the data; at present the `PSFCorrection` class requires the pixelation to be in the `HEALPix` pixelation.

The keyword `psf_dir` points to the directory where the `f_ary` and `df_rho_div_f_ary` will be stored; if unspecified, they will be stored to the current directory. The `f_ary` consists of `num_f_bins` entries linear spaced between 0 and 1. The PSF correction involves placing many (`n_psf`) PSFs at random positions on the `HEALPix` map, drawing `n_pts_per_psf` points from each PSF, and then looking at the distribution of points among pixels. The larger `n_psf` and `n_pts_per_psf`, the more accurate the computation of `df_rho_div_f_ary` will be. However, the computation time of the PSF arrays also increases as these parameters are increased.

By default the `PSFCorrection` class assumes that the PSF is a 2-D Gaussian distribution:

$$\mathrm{PSF}(r) = \frac{1}{2\pi\sigma^2} \exp\left[-\frac{r^2}{2\sigma^2}\right] . \quad (11)$$

Here, $\mathrm{PSF}(r)$ describes the spread of arriving counts with angular distance r away from the arrival direction. The parameter `psf_sigma_deg` denotes σ in degrees. Upon initializing `PSFCorrection` with `psf_sigma_deg` specified, the class automatically computes the array `df_rho_div_f_ary` and stores it in the `psf_dir` with a unique name related to the keywords. If such a file already exists in the `psf_dir`, then the code will simply load this file instead of recomputing it. After initialization, the relevant arrays may be accessed by `pc_inst.f_ary` and `pc_inst.df_rho_div_f_ary`.

The `PSFCorrection` class can also handle arbitrary PSF functions. In this case, the class should be initialized with `delay_compute = True`. Then, the user should manually set the function `pc_inst.psf_r_func` to the desired function $\mathrm{PSF}(r)$. This function will be discretized with `pc_inst.psf_samples` points out to `pc_inst.sample_psf_max` degrees from $r = 0$. These two quantities also need to be manually specified. The user also needs to set `pc_inst.psf_tag` to a string that will be used for saving the PSF arrays. After these four attributes have been set manually by the user, the PSF arrays are computed and stored by executing `pc_inst.make_or_load_psf_corr()`.

def NPTFit.create_mask.make_mask_total

This function is used to make masks that can then be used to reduce the data and templates to a smaller ROI when performing the scan. While these masks can always be made by hand, this function provides a simple masking interface for maps in the `HEALPix` format. The `make_mask_total` function can mask pixels by latitude, longitude, and radius from any point on the sphere. See the online documentation for more specific examples.

class NPTFit.dnds_analysis.Analysis

The analysis class may be used to extract useful information from the results of an NPTF performed using `MultiNest`. The class also has built-in plotting features for making many of the most common types of visualizations for the parameter posterior distribution. An instance of the analysis class can be instantiated by

```
an = Analysis(nptf, mask=None, pixarea
    =0.)
```

where `nptf` is itself an instance of the `NPTF` class that already has the results of a scan loaded. The keyword arguments `mask` and `pixarea` are optional. The user should specify a `mask` if the desired ROI for the analysis is different that that used in the scan. The user should specify a `pixarea` if the data is not in the `HEALPix` format. The code will still assume the pixels are equal area with area `pixarea`, which should be specified in sr.

After initialization, the intensities of Poissonian and non-Poissonian templates, respectively, may be extracted from the analysis class by the commands

```
an.return_intensity_arrays_poiss(comp)
```

and

```
an.return_intensity_arrays_non_poiss(
    comp)
```

Here, `comp` refers to the template key used by the Poissonian or non-Poissonian model. The arrays returned give the mean intensities of that model in the ROI in units of counts $\mathrm{cm}^{-2}\mathrm{s}^{-1}$, assuming the exposure map was in units of $\mathrm{cm}^2\mathrm{s}$. The arrays computed over the full set of entries in the discretized posterior distribution output by `MultiNest`. Thus, these intensity arrays may be interpreted as the 1-D posteriors for the intensities. For additional keywords that may be used to customize the computation of the intensity arrays, see the online documentation.

The source-count distributions may also be accessed from the analysis class. Executing

```
an.return_dndf_arrays(comp, flux)
```

will return the discretized 1-D posterior distribution for $\mathrm{mean}_{\mathrm{ROI}} dN_p(F)/dF$ at flux F for the PS model with template key `comp`. Note that the mean is computed over pixels p in the ROI.

The 1-D posterior distributions for the individual model parameters may be accessed by

```
A_poiss_post = an.
    return_poiss_parameter_posteriors(
    comp)
```

for Poissonian models, and

```
A_non_poiss_post, n_non_poiss_post,
    Sb_non_poiss_post = an.
    return_non_poiss_parameter_posteriors(
    comp)
```

for non-Poissonian models. Here `A_poiss_post` is a 1-D array of the discretized posterior distribution for the Poissonian template normalization parameter. Similarly, `A_non_poiss_post` is the posterior array for the non-Poissonian normalization parameter. The arrays `n_non_poiss_post` and `Sb_non_poiss_post` are 2-D, where – for example – `n_non_poiss_post = [n_1_array, n_2_array, ...]` and `n_1_array` is a 1-D array for the posterior for n_1.

Another useful piece of information that may be extracted from the scan is the Bayesian evidence:

```
l_be, l_be_err = an.get_log_evidence()
```

returns the log of the Bayesian evidence along with the uncertainty on this estimate based on the resolution of the MCMC.

For information on the plotting capabilities in the analysis class, see the online documentation or the example in the following section.

IV. NPTFIT: AN EXAMPLE

In this section we give an example for how to perform an NPTF using `NPTFit`. Many more examples are available in the online documentation. This particular example reproduces aspects of the main results of [2], which found evidence for a spherical population of unresolved gamma-ray PSs around the Galactic Center. The example uses the processed, public *Fermi* data made available with the release of the `NPTFit` package. The data set consists of 413 weeks of *Fermi* Pass 8 data in the UltracleanVeto event class (top quartile of events as ranked by PSF) from 2 to 20 GeV. The map is binned in `HEALPix` with `nside` = 128. The data, along with the exposure map and background templates, may be downloaded from

http://hdl.handle.net/1721.1/105492.

In the example we will perform an NPTF on the subregion where we mask the Galactic plane at latitude $|b| < 2^\circ$ and mask pixels with angular distance greater than 30° from the Galactic Center. We also mask identified PSs in the 3FGL PS catalog [59] at 95% containment using the provided PS mask, which is added to the geometric mask. We include smooth templates for diffuse gamma-ray emission in the Milky Way (using the *Fermi* `p6v11` diffuse model), isotropic emission (which can also absorb instrumental backgrounds), and emission following the *Fermi* bubbles, which are taken to be uniform in flux following the spatial template in [60]. We also include a dark matter template, which traces the line of sight integral of the square of a canonical NFW density profile.

We additionally include point source (non-Poissonian) models for the DM template, as well as for a disk template which corresponds to a doubly exponential thin-disk source distribution with scale height 0.3 kpc and radius 5 kpc. The source-count distributions for these are parameterized by singly-broken power laws, each described by four parameters $\{A, F_{b,1}, n_1, n_2\}$.

A. Setting up the scan

We begin the example by loading in the relevant modules, described in the previous section, that we will need to setup, perform, and analyze the scan.

```
import numpy as np
# module for performing scan
from NPTFit import nptfit
# module for creating the mask
from NPTFit import create_mask as cm
# module for determining the PSF
    correction
from NPTFit import psf_correction as pc
# module for analyzing the output
from NPTFit import dnds_analysis
```

Next, we create an instance of the `NPTF` class, which is used to configure and perform a scan.

```
n = nptfit.NPTF(tag='GCE_Example')
```

We assume here that the supplementary *Fermi* data has been downloaded to a directory `'fermi_data'`. Then, we may load in the data and exposure maps by

```
fermi_data = np.load('fermi_data/
    fermidata_counts.npy').astype(int)
fermi_exposure = np.load('fermi_data/
    fermidata_exposure.npy')
n.load_data(fermi_data, fermi_exposure)
```

Importantly, note that the exposure map has units of $\mathrm{cm}^2\mathrm{s}$. Next, we use the `create_mask` class to generate our ROI mask, which consists of both the geometric mask and the PS mask loaded in from the `'fermi_data'` directory:

```
pscmask=np.array(np.load('fermi_data/
    fermidata_pscmask.npy'), dtype=bool)
mask = cm.make_mask_total(band_mask =
    True, band_mask_range = 2, mask_ring =
     True, inner = 0, outer = 30,
    custom_mask = pscmask)
n.load_mask(mask)
```

The templates may also be loaded in from this directory,

```
dif = np.load('fermi_data/template_dif.
    npy')
iso = np.load('fermi_data/template_iso.
    npy')
bub = np.load('fermi_data/template_bub.
    npy')
gce = np.load('fermi_data/template_gce.
    npy')
dsk = np.load('fermi_data/template_dsk.
    npy')
```

These templates are counts map (i.e. flux maps times the exposure map) that have been pre-smoothed by the PSF (except for the disk-correlated template labeled `dsk`). We then add them to our `NPTF` instance with appropriately chosen keywords:

```
n.add_template(dif, 'dif')
n.add_template(iso, 'iso')
n.add_template(bub, 'bub')
n.add_template(gce, 'gce')
n.add_template(dsk, 'dsk')

# remove the exposure correction for PS
    templates
rescale = fermi_exposure/np.mean(
    fermi_exposure)
n.add_template(gce/rescale, 'gce_np',
    units='PS')
n.add_template(dsk/rescale, 'dsk_np',
    units='PS')
```

Note that templates `'gce_np'` and `'dsk_np'` intended to be used in non-Poissonian models should trace the underlying PS distribution, without exposure correction, and are added with the keyword `units='PS'`.

B. Adding models

Now that we have loaded in all of the external data and templates, we can add models to our NPTF instance. First, we add in the Poissonian models,

```
n.add_poiss_model('dif', '$A_\mathrm{dif}
    $', False, fixed=True, fixed_norm
    =14.67)
n.add_poiss_model('iso', '$A_\mathrm{iso}
    $', [0,2], False)
n.add_poiss_model('gce', '$A_\mathrm{gce}
    $', [0,2], False)
n.add_poiss_model('bub', '$A_\mathrm{bub}
    $', [0,2], False)
```

All Poissonian models are taken to have linear priors, with prior ranges for the normalizations between 0 and 2. However, the normalization of the diffuse background has been fixed to the value 14.67, which is approximately the correct normalization in these units for this template, in order to provide an example of this syntax. Next, we add in the two non-Poissonian models:

```
n.add_non_poiss_model('gce_np', ['$A_\
    mathrm{gce}^\mathrm{ps}$','$n_1^\
    mathrm{gce}$','$n_2^\mathrm{gce}$','
    $S_b^{(1), \mathrm{gce}}$'],
    [[-6,1],[2.05,30],[-2,1.95],[0.05,40]],
     [True,False,False,False])
n.add_non_poiss_model('dsk_np', ['$A_\
    mathrm{dsk}^\mathrm{ps}$','$n_1^\
    mathrm{dsk}$','$n_2^\mathrm{dsk}$','
    $S_b^{(1), \mathrm{dsk}}$'],
    [[-6,1],[2.05,30],[-2,1.95],[0.05,40]],
     [True,False,False,False])
```

We have added in the models for disk-correlated and NFW-correlated (line of sight integral of the the NFW distribution squared) unresolved PS templates. Each of these models takes singly-broken power-law source-count distributions. In each case, the normalization parameter is taken to have a log-flat prior while the indices and breaks are taken to have linear priors. The units of the breaks are specified in terms of counts.

C. Configure scan with PSF correction

In this energy range and with this data set, the PSF may be modeled by a 2-D Gaussian distribution with $\sigma = 0.1812^\circ$. From this, we are able to construct the PSF-correction arrays:[6]

```
pc_inst = pc.PSFCorrection(psf_sigma_deg
    =0.1812)
f_ary, df_rho_div_f_ary = pc_inst.f_ary,
    pc_inst.df_rho_div_f_ary
```

[6] For an example of how to construct these arrays with a more complicated, non-Gaussian PSF function, see the online documentation.

These arrays are then passed into the NPTF instance when we configure the scan:

```
n.configure_for_scan(f_ary,
    df_rho_div_f_ary, nexp=1)
```

Note that since our ROI is relatively small and the exposure map does not change significantly over the region, we have a single exposure region with nexp=1.

D. Performing the scan with MultiNest

We perform the scan using MultiNest with nlive=100 as an example to demonstrate the basic features and conclusions of this analysis while being able to perform the scan in a reasonable amount of time on a single processor, although ideally nlive should be set to a higher value for more reliable results:

```
n.perform_scan(nlive=100)
```

E. Analyzing the results

Now, we are ready to analyze the results of the scan. First we load in relevant modules:

```
import corner
import matplotlib.pyplot as plt
```

and then we load in the results of the scan (configured as above),

```
n.load_scan()
```

The chains, giving a discretized view of the posterior distribution, may be accessed simply through the attribute n.samples. However, we will analyze the results by using the analysis class provided with NPTFit. We make an instance of this class simply by

```
an = dnds_analysis.Analysis(n)
```

1. *Make triangle plots*

Triangle plots are a simple and quick way of visualizing correlations in the posterior distribution. Such plots may be generated through the command

```
an.make_triangle()
```

which leads to the plot in Fig. 1.

2. *Plot source-count distributions*

The source-count distributions for NFW- and disk-correlated point source models may be plotted with

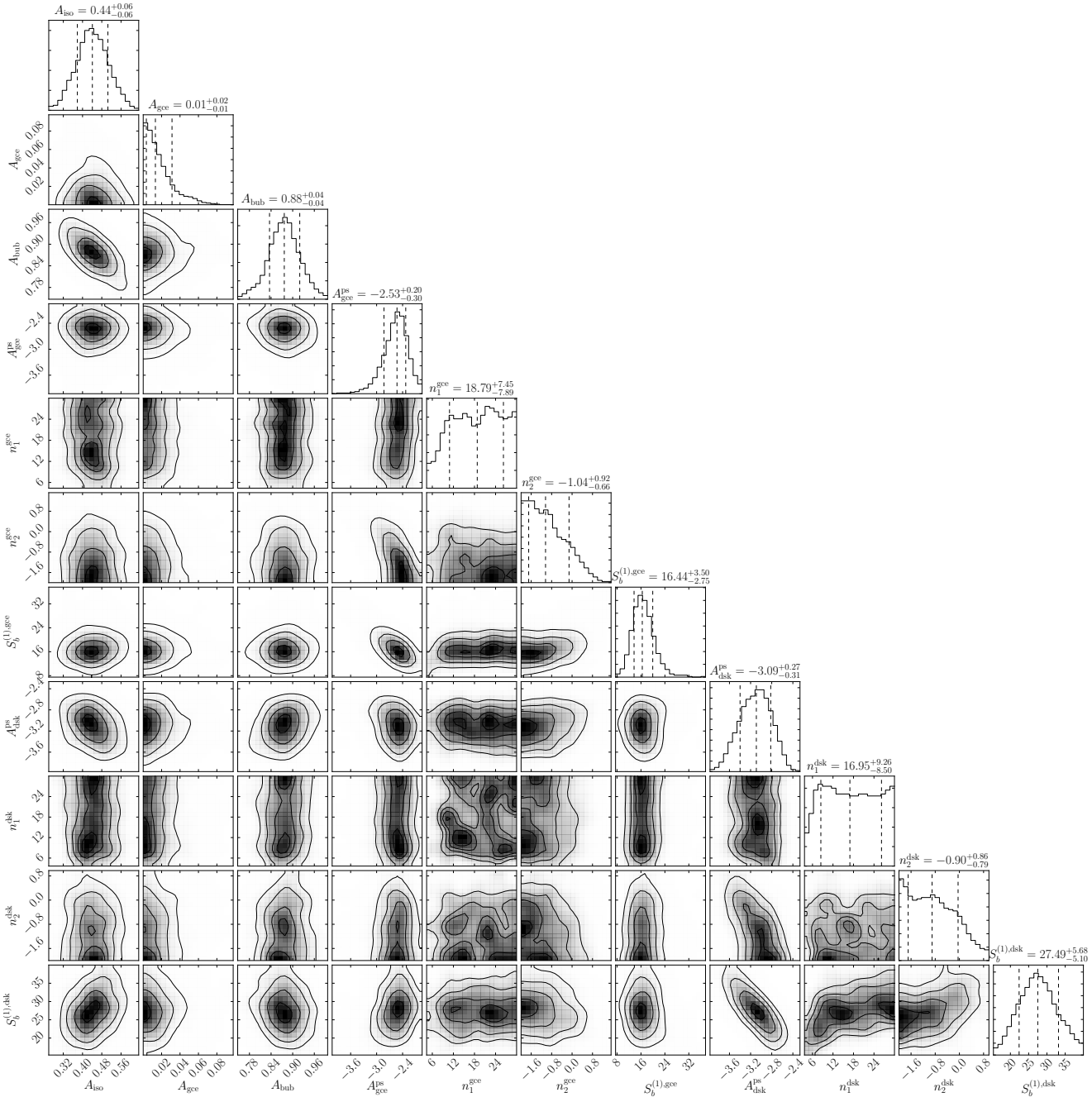

FIG. 1. The triangle plot obtained by analyzing the results of an NPTF in the Galactic Center, showing the one and two dimensional posteriors of the 11 parameters floated in the fit corresponding to three Poissonian and two non-Poissonian templates. For this analysis 3FGL point sources have been masked at 95% containment. See text for details.

```
an.plot_source_count_median('dsk',smin
    =0.01,smax=1000,nsteps=1000,color='
    cornflowerblue',spow=2,label='Disk')
an.plot_source_count_band('dsk',smin
    =0.01,smax=1000,nsteps=1000,qs
    =[0.16,0.5,0.84],color='cornflowerblue
    ',alpha=0.3,spow=2)
an.plot_source_count_median('gce',smin
    =0.01,smax=1000,nsteps=1000,color='
    forestgreen',spow=2,label='GCE')
an.plot_source_count_band('gce',smin
    =0.01,smax=1000,nsteps=1000,qs
    =[0.16,0.5,0.84],color='forestgreen',
    alpha=0.3,spow=2)
```

along with the following `matplotlib` plotting options.

```
plt.yscale('log')
plt.xscale('log')
plt.xlim([5e-11,5e-9])
```

```
plt.ylim([2e-13,1e-10])
plt.tick_params(axis='x', length=5, width
    =2, labelsize=18)
plt.tick_params(axis='y', length=5, width
    =2, labelsize=18)
plt.ylabel('$F^2 dN/dF$ [counts/cm$^2$/s/
    deg$^2$]', fontsize=18)
plt.xlabel('$F$  [counts/cm$^2$/s]',
    fontsize=18)
plt.title('Galactic Center NPTF', y=1.02)
plt.legend(fancybox=True)
plt.tight_layout()
```

This is shown in Fig. 2. Contribution from both NFW- and disk-correlated PSs may be seen, with NFW-correlated sources contributing dominantly at lower flux values. In that figure, we also show a histogram of the detected 3FGL sources within the relevant energy range and region, with vertical error bars indicating the 68% confidence interval from Poisson counting uncertainties only.[7] Since we have explicitly masked all 3FGL sources, we see that the disk- and NFW-correlated PS templates contribute at fluxes near and below the 3FGL PS detection threshold, which is $\sim 5 \times 10^{-10}$ counts cm^{-2} s^{-1} in this case.

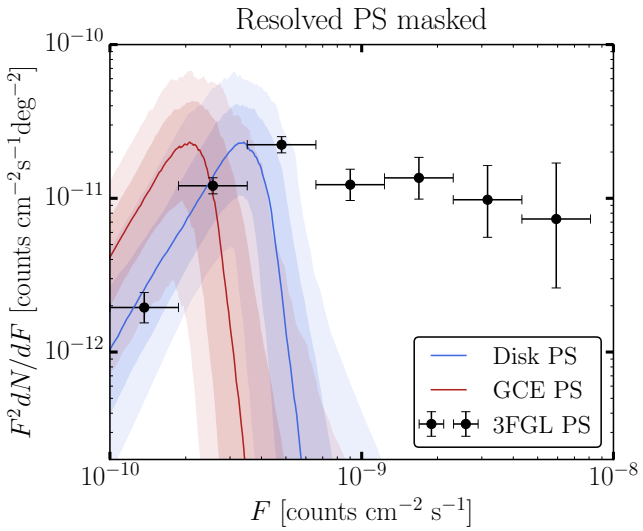

FIG. 2. The source-count distribution as constructed from the analysis class, for the example NPTF described in the main text. This scan looks for disk-correlated PSs along with PSs correlated with the expected DM template (GCE PSs). Since all resolved PSs are masked in this analysis, the source-count distributions are seen to contribute dominantly below the 3FGL detection threshold. A histogram of resolved 3FGL sources is also shown.

[7] The data for plotting these points is available in the online documentation.

3. *Plot intensity fractions*

The intensity fractions for the smooth and PS NFW-correlated models may be plotted with

```
an.plot_intensity_fraction_non_poiss('gce
    ', bins=800, color='cornflowerblue',
    label='GCE PS')
an.plot_intensity_fraction_poiss('gce',
    bins=800, color='lightsalmon', label='
    GCE DM')
plt.xlabel('Flux fraction (%)')
plt.legend(fancybox = True)
plt.xlim(0,6)
```

This is shown in Fig. 3. We immediately see a preference for NFW-correlated point sources over the smooth NFW component.

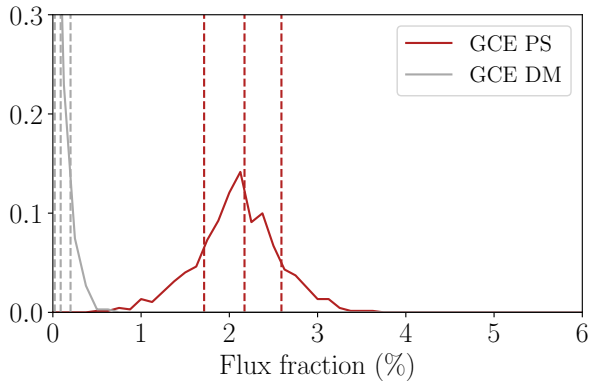

FIG. 3. Intensity fractions for the smooth (green) and point source (red) templates correlating with the DM template, obtained by analyzing the results of an NPTF in the Galactic Center with 3FGL point sources masked at 95% containment.

4. *Further analyses*

The example above may easily be pushed further in many directions, many of which are outline in [2]. For example, a natural method for performing model comparison in the Bayesian framework is to compute the Bayes factor between two models. Here, for example, we may compute the Bayes factor between the model with and without NFW-correlated PSs. This involves repeating the scan described above but only adding in disk-correlated PSs. Then, by comparing the global Bayesian evidence between the two scans (see Sec. III for the syntax on how to extract the Bayesian evidence), we find a Bayes factor $\sim 10^3$ in preference for the model with spherical PSs.

Another straightforward generalization of the example described above is simply to leave out the PS mask, so that the NFW- and disk-correlated PS templates must account for both the resolved and unresolved PSs. The likelihood evaluations take longer, in this case, since there

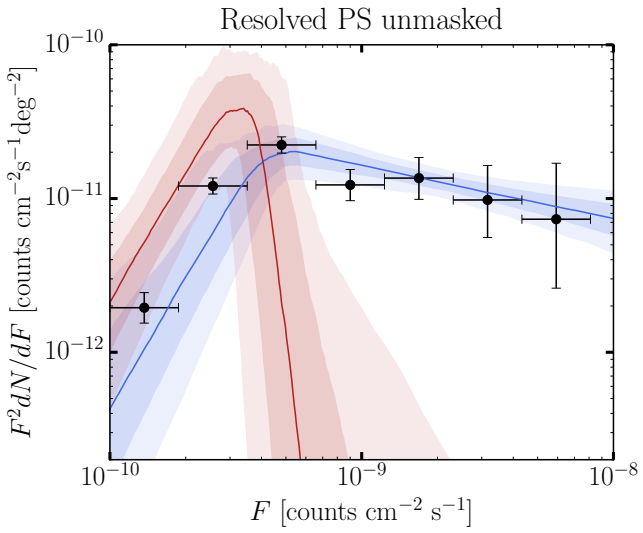

FIG. 4. As in Fig. 2, but in this case the resolved 3FGL sources were not masked. The disk-correlated template accounts for the majority of the resolved PS emission.

are pixels with higher photon counts compared to the 3FGL-masked scan. The result for the source-count distribution from this analysis is shown in Fig. 4. In this case, the disk-correlated PS template accounts for the resolved 3FGL sources, while the NFW-correlated PS template contributes at roughly the same flux range as in the 3FGL masked case. The Bayes factor in preference for the model with NFW-correlated PSs over that without – as described above – is found to be $\sim 10^{10}$ in this case.

V. CONCLUSION

We have presented an open-source code package for performing non-Poissonian template fits. We strongly recommend referring to the online documentation – which will be kept up-to-date – in addition to this paper accompanying the initial release. There are many way in which `NPTFit` can be improved in the future. For one, the `NPTFit` package only handles a single energy bin at a time. In a later version of the code we plan to incorporate the ability to scan over multiple energy bins simultaneously. Additionally, there are a few areas – such as the evaluation of the incomplete gamma functions – where the `cython` code may still be sped up. Such improvements to the computational cost are relevant for analyses of large data sets with many model parameters. Of course, we welcome additional suggestions for how we may improve the code and better adapt it to applications beyond the gamma-ray applications it has been used for so far.

ACKNOWLEDGEMENTS

Foremost, we thank Samuel Lee for contributing significantly not just to the original conception of the NPTF but also to an early version of the `NPTFit` package. We also thank Lina Necib for contributing to `NPTFit`. Additionally, we thank our collaborators Tim Linden, Mariangela Lisanti, Tracy Slatyer, and Wei Xue, who worked with us on projects utilizing an early version of `NPTFit`. We thank Douglas Finkbeiner, Dan Hooper, Christoph Weniger, and Hannes Zechlin for discussions related to the NPTF and `NPTFit`. NLR is supported in part by the American Australian Association's ConocoPhillips Fellowship. BRS is supported by a Pappalardo Fellowship in Physics at MIT. The work of BRS was performed in part at the Aspen Center for Physics, which is supported by National Science Foundation grant PHY-1066293. This work is supported by the U.S. Department of Energy (DOE) under cooperative research agreement DE-SC-0012567 and DE-SC-0013999.

Appendix A: Mathematical foundations of `NPTFit`

In this section we present the mathematical foundation of the NPTF and the evaluation of the non-Poissonian likelihood in more detail that what was shown in Sec. II. Note that many of the details presented in this section have appeared in the earlier works of [1, 2, 4], however we have reproduced these here in order to have a single clear picture of the method.

The remainder of this section is divided as follows. Firstly we outline how to determine the generating functions for the Poissonian and non-Poissonian case. We then describe how we account for finite PSF corrections.

1. The (non-)Poissonian generating function

There are two reasons why the evaluation of the Poissonian likelihood for traditional template fitting can be evaluated rapidly. The first of these is that the functional form of the Poissonian likelihood is simple. Secondly, and more importantly, is the fact that if we have two discrete random variables X and Y that follow Poisson distributions with means μ_1 and μ_2, then the random variable $Z = X+Y$ again follows a Poisson distribution with mean $\mu_1+\mu_2$. This generalizes to combining an arbitrary number of random Poisson distributed variables and is why we were able to write $\mu_{p,\ell}(\boldsymbol{\theta}) = A_\ell(\boldsymbol{\theta}) T^{(S)}_{p,\ell}$ in Sec. II. This fact is not true when combining arbitrary random variables, and in particular if we add in a template following non-Poissonian statistics.

An elegant solution to this problem was introduced in [4], using the method of generating functions. As we are always dealing with pixelized maps containing discrete counts (of photons or otherwise), for any model of interest there will always be a discrete probability dis-

tribution p_k, the probability of observing $k = 0, 1, 2, \ldots$ counts. In terms of these, we then define the probability generating function as in (6). The property of probability generating functions that make them so useful in the present context is as follows. Consider two random processes X and Y, with generating functions $P_X(t)$ and $P_Y(t)$, that follow arbitrary and potentially different statistical distributions. Then the generating function of $Z = X + Y$ is simply given by the product $P_X(t) \cdot P_Y(t)$. In this subsection we will derive the appropriate form of $P(t)$ for Poissonian and non-Poissonian statistics.

To begin with, consider the purely Poissonian case. Here and throughout this section we consider only the likelihood in a single pixel; the likelihood over a full map is obtained from the product of the pixel-based likelihoods. Then for a Poisson distribution with an expected number of counts μ_p in a pixel p:

$$p_k = \frac{\mu_p^k e^{-\mu_p}}{k!} \,. \tag{A1}$$

Note that the variation of the μ_p across the full map will be a function of the model parameters, such that $\mu_p = \mu_p(\boldsymbol{\theta})$. In order to simplify the notation in this section however, we leave the $\boldsymbol{\theta}$ dependence implicit. Given the p_k values, we then have:

$$\begin{aligned} P_{\rm P}(t) &= \sum_{k=0}^{\infty} \frac{\mu_p^k e^{-\mu_p}}{k!} t^k \\ &= e^{-\mu} \sum_{k=0}^{\infty} \frac{(\mu_p t)^k}{k!} \\ &= \exp\left[\mu_p (t-1)\right] \,. \end{aligned} \tag{A2}$$

From this form, it is clear that if we have two Poisson distributions with means $\mu_p^{(1)}$ and $\mu_p^{(2)}$, the product of their generating functions will again describe a Poisson distribution, but with mean $\mu_p^{(1)} + \mu_p^{(2)}$.

Next we work towards the generating function in the non-Poissonian case. At the outset, we let $x_{p,m}$ denote the average number of sources in a pixel p that emit exactly m counts. In terms of this, the probability of finding n_m m-count sources in this pixel is just a draw from a Poisson distribution with mean $x_{p,m}$, i.e.

$$p_{n_m} = \frac{x_{p,m}^{n_m} e^{-x_{p,m}}}{n_m!} \,. \tag{A3}$$

Given this, the probability to find k counts from a population of m-count sources is

$$p_k^{(m)} = \begin{cases} p_{n_m}, & \text{if } k = m \cdot n_m \text{ for some } n_m, \\ 0, & \text{otherwise} \end{cases} \,. \tag{A4}$$

We can then use this to derive the non-Poissonian m-count generating function as follows:

$$\begin{aligned} P_{\rm NP}^{(m)}(t) &= \sum_k p_k t^k \\ &= \sum_{n_m} t^{m \cdot n_m} \frac{x_{p,m}^{n_m} e^{-x_{p,m}}}{n_m!} \\ &= \exp\left[x_{p,m}(t^m - 1)\right] \,. \end{aligned} \tag{A5}$$

However this is just the generating function for m-count sources, to get the full non-Poissonian generating function we need to multiply this over all values of m. Doing so we arrive at

$$\begin{aligned} P_{\rm NP}(t) &= \prod_{m=1}^{\infty} \exp\left[x_{p,m}(t^m - 1)\right] \\ &= \exp\left[\sum_{m=1}^{\infty} x_{p,m}(t^m - 1)\right] \,, \end{aligned} \tag{A6}$$

justifying the form given in Sec. II. Again recall for the full likelihood we can just multiply the pixel based likelihoods and that $x_{p,m} = x_{p,m}(\boldsymbol{\theta})$.

So far we have said nothing of how to determine $x_{p,m}$, the average number of m-count source in pixel p. This value depends on the source-count distribution dN_p/dS, which specifies the distribution of sources as a function of their expected number of counts, S. Of course the physical object is dN/dF, where F is the flux. This distinction was discussed in Sec. II, and can be implemented in `NPTFit` to arbitrary precision. Nevertheless dN_p/dS does not fully determine $x_{p,m}$ – we need to account for the fact that a source that is expected to give S photons could Poisson fluctuate to give m. As such any source can in principle contribute to $x_{p,m}$, and so integrating over the full distribution we arrive at:

$$x_{p,m} = \int_0^{\infty} dS \frac{dN_p}{dS}(S) \frac{S^m e^{-S}}{m!} \,. \tag{A7}$$

An important part of implementing the NPTF in a rapid manner, which is a central feature of `NPTFit`, is the analytic evaluation of the integral in this equation. In order to do this, we need to have a specific form of the source-count distribution. For this purpose, we allow the source count distribution to be a multiply broken power-law and evaluate the integral for any number of breaks. The details of this calculation are presented in App. C.

Putting the evaluation of the integral aside for the moment then, we have arrived at the full non-Poissonian generating function:

$$\begin{aligned} P_{\rm NP}(t) &= \exp\left[\sum_{m=1}^{\infty} x_{p,m}(t^m - 1)\right] \,, \\ x_{p,m} &= \int_0^{\infty} dS \frac{dN_p}{dS}(S) \frac{S^m e^{-S}}{m!} \,. \end{aligned} \tag{A8}$$

Contrasting this with Eq. (A2), we see that whilst the Poissonian likelihood is specified by a single number μ_p,

the non-Poissonian likelihood is instead specified by a distribution dN_p/dS.

In the case of multiple PS templates, we should multiply the independent probability generating functions. However, this is equivalent to summing the $x_{p,m}$ parameters. This is how multiple PS templates are incorporated into the `NPTFit` code:

$$x_{p,m} \to x_{p,m}^{\rm total} = \sum_{\ell=1}^{N_{\rm NPT}} x_{p,m}^{\ell} \,, \tag{A9}$$

where the sum over ℓ is over the contributions from individual PS templates.

2. Correcting for a finite point spread function

The next factor to account for is the fact that in any realistic dataset there will be a non-zero PSF. Here, we closely follow the discussion in [4]. The PSF arises due to the inability of an instrument to perfectly reconstruct the original direction of the photon, neutrino, or quantity making up the counts. In practice, a finite PSF means that a source in one pixel can contribute counts to nearby pixels as well. To implement this correction, we modify the calculation of $x_{p,m}$ given in Eq. (A8), which accounts for the distribution of sources as a function of S and the fact that each one could Poisson fluctuate to give us m counts. The finite PSF means that in addition to this, we also need to draw from the distribution $\rho(f)$, that determines the probability that a given source contributes a fraction of its flux f in a given pixel. Once we know $\rho(f)$, this modifies our calculation of $x_{p,m}$ in Eq. (A8) – now a source that is expected to contribute S counts, will instead contribute fS, where f is drawn from $\rho(f)$. As such we arrive at the result in (10).

In `NPTFit` we determine $\rho(f)$ using Monte Carlo. To do this we place a number of PSs appropriately smeared by the PSF at random positions on a pixelized sphere. Then integrating over all pixels we can determine the fraction of the flux in each pixel f_p, $p = 1, \ldots, N_{\rm pix}$, defined such that $f_1 + f_2 + \ldots = 1$. Note in practice one can truncate this sum at some minimal value of f without impacting the argument below. From the set $\{f_p\}$, we then denote by $\Delta n(f)$ the number of fractions for n point sources that fall within some range Δf. From these quantities, we may determine $\rho(f)$ as

$$\rho(f) = \lim_{\substack{\Delta f \to 0 \\ n \to \infty}} \frac{\Delta n(f)}{n \Delta f} \,, \tag{A10}$$

which is normalized such that $\int df\ f\rho(f) = 1$. From this definition we see that the case of a vanishing PSF is just $\rho(f) = \delta(f-1)$ - i.e. the flux is always completely in the pixel with the PS.

Appendix B: `NPTFit`: algorithms

The generating-function formalism for calculating the probabilities $p_{n_p}^{(p)}(\boldsymbol{\theta})$ is described at the end of Sec. II and in more detail in App. A. In particular – given the generating function $P(t)$ – we are instructed to calculate the probabilities by taking n_p derivatives as in (7). However, taking derivatives is numerically costly, and so instead we have developed recursive algorithms for computing these probabilities. In the same spirit, we analytically evaluate the $x_{p,m}$ parameters defined in (10) for the multiply-broken source-count distribution in order to facilitate a fast evaluation of the NPTF likelihood function. In this section, we overview these methods that are essential to making `NPTFit` a practical software package.

In general we may write the full single pixel generating function for a model containing an arbitrary number of Poissonian and non-Poissonian templates as:

$$P(t) = e^{f(t)} \,, \tag{B1}$$

where we have defined

$$f(t) \equiv \mu_p (t-1) + \sum_{m=1}^{\infty} x_{p,m}(t^m - 1) \,. \tag{B2}$$

Above, $x_{p,m}$ represents the average number of m-count source in pixel p. The remaining task is to efficiently calculate the probabilities p_k, which are formally defined in terms of derivatives through (7). Nevertheless, derivatives are slow to implement numerically, so we instead use a recursion relation to determine p_k in terms of $p_{<k}$.

To begin with, note that

$$f^{(k)} \equiv \frac{d^k}{dt^k} f(t)\Big|_{t=0} = \begin{cases} -(\mu_p + \sum_{m=1}^{\infty} x_{p,m}), & k = 0 \,, \\ \mu_p + x_{p,1}, & k = 1 \,, \\ k! x_{p,k}, & k > 1 \,. \end{cases} \tag{B3}$$

For the rest of this discussion, we suppress the pixel index p, though one should keep in mind that this process must be performed independently in every pixel. From (B3), we can immediately write down

$$\begin{aligned} p_0 &= e^{f^{(0)}} \,, \\ p_1 &= f^{(1)} e^{f^{(0)}} \,. \end{aligned} \tag{B4}$$

Given p_0 and p_1, we may write our recursion relation for $k > 1$ as

$$p_k = \sum_{n=0}^{k-1} \frac{1}{k(k-n-1)!} f^{(k-n)} p_n \,, \tag{B5}$$

which as mentioned requires the knowledge of all $p_{<k}$.

To derive (B5), we first define

$$F^{(k)}(t) \equiv \frac{d^k}{dt^k} e^{f(t)} \,. \tag{B6}$$

Then, for example,

$$F^{(1)}(t) = f^{(1)}(t) e^{f^{(0)}(t)} \,. \tag{B7}$$

From here to determine $F^{(k)}(t)$ we simply need $k-1$ more derivatives. Using the generalized Leibniz rule, we have

$$\begin{aligned} F^{(k)}(t) &= \frac{d^{k-1}}{dt^{k-1}} \left(f^{(1)}(t) e^{f^{(0)}(t)} \right) \\ &= \sum_{n=0}^{k-1} \binom{k-1}{n} \frac{d^{k-1-n}}{dt^{k-1-n}} f^{(1)}(t) \frac{d^n}{dt^n} e^{f^{(0)}(t)} \\ &= \sum_{n=0}^{k-1} \binom{k-1}{n} f^{(k-n)}(t) F^{(n)}(t) \,. \end{aligned} \tag{B8}$$

Then setting $t = 0$ and recalling the definition of p_k, this yields

$$\begin{aligned} p_k &= \sum_{n=0}^{k-1} \frac{n!}{k!} \binom{k-1}{n} f^{(k-n)} p_n \\ &= \sum_{n=0}^{k-1} \frac{1}{k(k-n-1)!} f^{(k-n)} p_n \,, \end{aligned} \tag{B9}$$

as claimed.

To calculate the $f^{(k)}$ in a pixel p, we need to calculate the $x_{p,k}$ and the sum $\sum_{m=1}^{\infty} x_{p,m}$. We may calculate these expressions analytically using the general source-count distribution in (4). To calculate the sums, we make use of the relation

$$\begin{aligned} \sum_{m=1}^{\infty} x_{p,m} &= \int_0^{\infty} dS \frac{dN_p}{dS} e^{-S} \sum_{m=1}^{\infty} \frac{S^m}{m!} \\ &= \int_0^{\infty} dS \frac{dN_p}{dS} - \int_0^{\infty} dS \frac{dN_p}{dS} e^{-S} \\ &= \int_0^{\infty} dS \frac{dN_p}{dS} - x_{p,0} \,. \end{aligned} \tag{B10}$$

Finiteness of the total flux, and also the probabilities, requires $n_1 > 2$ and $n_{k+1} < 2$. However, both the integral and $x_{p,0}$, appearing in the last line above, may be divergent individually if $1 < n_{k+1} < 2$. In this case, we analytically continue in n_{k+1}, evaluate the contributions individually, and then sum the two expressions to get a result that is finite across the whole range of allowable parameter space. The expressions for the $x_{p,m}$ and the sums over these quantities are given in App. C in terms of incomplete gamma-functions.

Appendix C: Analytic expressions for $x_{p,m}$ and $\sum_{m=1}^{\infty} x_{p,m}$

In this appendix we derive analytic expressions for $x_{p,m}$ and $\sum_{m=1}^{\infty} x_{p,m}$, which go into (B3) and are needed to evaluate the non-Poissonian likelihood. This is done by a straightforward application of (A7) and (B10). Recall that $x_{p,m}$ represents the average number of m-count source in pixel p. We begin by working explicitly through the 1- and 2-break source-count distributions before discussing the general case.

1. 1 break

For a single break, the pixel-dependent source count distribution is given in terms of counts by

$$\frac{dN_p}{dS} = A \frac{T_p^{(\mathrm{PS})}}{E_p} \begin{cases} (S/S_b)^{-n_1} \,, & S \geq S_b \\ (S/S_b)^{-n_2} \,, & S < S_b \end{cases} \,. \tag{C1}$$

In the following, we will suppress the overall factor of $T_p^{(\mathrm{PS})}/E_p$, since it does not play an important role in this discussion and may always be restored by simply rescaling A. In the same spirit, we also suppress the pixel index p in $x_{p,m}$.

With this in mind, we may explicitly evaluate the expression for $x_{p,m}$ using (A7):

$$\begin{aligned} x_m =& \frac{A}{m!} \left[S_b^{n_1} \int_{S_b}^{\infty} dS \; S^{m-n_1} e^{-S} + S_b^{n_2} \int_0^{S_b} dS \; S^{m-n_2} e^{-S} \right] \\ =& \frac{A}{m!} \left[S_b^{n_1} \Gamma(1-n_1+m, S_b) + S_b^{n_2} \Gamma(1-n_2+m) \right. \\ & \left. - S_b^{n_2} \Gamma(1-n_2+m, S_b) \right] \,. \end{aligned} \tag{C2}$$

Now using our general result (B10) above, we have

$$\begin{aligned} \sum_{m=1}^{\infty} x_m =& A \left[S_b^{n_1} \int_{S_b}^{\infty} dS \; S^{-n_1} + S_b^{n_2} \int_0^{S_b} dS \; S^{-n_2} \right] - x_{p,0} \\ =& A S_b \left[\frac{1}{n_1 - 1} + \frac{1}{1 - n_2} \right] - x_0 \,. \end{aligned} \tag{C3}$$

This is useful because we already know x_0 from the general form of x_m above.

2. 2 breaks

For 2 breaks, the source-count distribution is given in terms of counts by

$$\frac{dN_p}{dS} = A \frac{T_p^{(\mathrm{PS})}}{E_p} \begin{cases} \left(\frac{S}{S_{b,1}} \right)^{-n_1} \,, & S \geq S_{b,1} \\ \left(\frac{S}{S_{b,1}} \right)^{-n_2} \,, & S_{b,1} > S \geq S_{b,2} \\ \left(\frac{S_{b,2}}{S_{b,1}} \right)^{-n_2} \left(\frac{S}{S_{b,2}} \right)^{-n_3} \,, & S_{b,2} > S \end{cases} \,. \tag{C4}$$

Again suppressing the pixel-dependent pre-factors, an explicit evaluation gives

$$x_m = \frac{AS_{b,1}^{n_1}}{m!}\Big[\Gamma(1-n_1+m,S_{b,1}) + S_{b,1}^{n_2-n_1}\Gamma(1-n_2+m,S_{b,2}) - S_{b,1}^{n_2-n_1}\Gamma(1-n_2+m,S_{b,1}) + S_{b,1}^{n_2-n_1}S_{b,2}^{n_3-n_2}\Gamma(1-n_3+m) - S_{b,1}^{n_2-n_1}S_{b,2}^{n_3-n_2}\Gamma(1-n_3+m,S_{b,2})\Big]\,, \tag{C5}$$

$$\sum_{m=1}^{\infty} x_m = AS_{b,1}\left[\frac{1}{n_1-1} + \frac{1}{1-n_2}\left(1-\left(\frac{S_{b,2}}{S_{b,1}}\right)^{1-n_2}\right) + \frac{1}{1-n_3}\left(\frac{S_{b,2}}{S_{b,1}}\right)^{1-n_2}\right] - x_0\,. \tag{C6}$$

3. k breaks

The source-count distribution in the general k-break case is given in (4) in terms of flux. In terms of counts and again suppressing pixel-dependent prefactors the result for x_m and $\sum_{m=1}^{\infty} x_m$ is a simple generalization from the expressions for the 1- and 2-break cases:

$$x_m = \frac{AS_{b,1}^{n_1}}{m!}\left[\Gamma(1-n_1+m,S_{b,1}) + \sum_{i=1}^{k-1}\left[\prod_{j=1}^{i} S_{b,j}^{n_{j+1}-n_j}\right]\{\Gamma(1-n_{i+1}+m,S_{b,i+1}) - \Gamma(1-n_{i+1}+m,S_{b,i})\} + \left[\prod_{j=1}^{k} S_{b,j}^{n_{j+1}-n_j}\right]\{\Gamma(1-n_{k+1}+m) - \Gamma(1-n_{k+1}+m,S_{b,k})\}\right]\,, \tag{C7}$$

$$\sum_{m=1}^{\infty} x_m = AS_{b,1}\left[\frac{1}{n_1-1} + \frac{1}{1-n_2}\left(1-\left(\frac{S_{b,2}}{S_{b,1}}\right)^{1-n_2}\right) + \sum_{i=3}^{k}\frac{1}{1-n_i}\left[\prod_{j=1}^{i-2}\left(\frac{S_{b,j+1}}{S_{b,j}}\right)^{1-n_{j+1}}\right]\left(1-\left(\frac{S_{b,i}}{S_{b,i-1}}\right)^{1-n_i}\right) + \frac{1}{1-n_{k+1}}\left[\prod_{j=1}^{k-1}\left(\frac{S_{b,j+1}}{S_{b,j}}\right)^{1-n_{j+1}}\right]\right] - x_0\,. \tag{C8}$$

[1] S. K. Lee, M. Lisanti, and B. R. Safdi, JCAP **1505**, 056 (2015), 1412.6099.
[2] S. K. Lee, M. Lisanti, B. R. Safdi, T. R. Slatyer, and W. Xue, Phys. Rev. Lett. **116**, 051103 (2016), 1506.05124.
[3] T. Miyaji and R. E. Griffiths, Astrophys. J. **564**, L5 (2002), astro-ph/0111393.
[4] D. Malyshev and D. W. Hogg, Astrophys. J. **738**, 181 (2011), 1104.0010.
[5] L. Goodenough and D. Hooper (2009), 0910.2998.
[6] D. Hooper and L. Goodenough, Phys.Lett. **B697**, 412 (2011), 1010.2752.
[7] A. Boyarsky, D. Malyshev, and O. Ruchayskiy, Phys.Lett. **B705**, 165 (2011), 1012.5839.
[8] D. Hooper and T. Linden, Phys.Rev. **D84**, 123005 (2011), 1110.0006.
[9] K. N. Abazajian and M. Kaplinghat, Phys.Rev. **D86**, 083511 (2012), 1207.6047.
[10] D. Hooper and T. R. Slatyer, Phys.Dark Univ. **2**, 118 (2013), 1302.6589.
[11] C. Gordon and O. Macias, Phys.Rev. **D88**, 083521 (2013), 1306.5725.
[12] K. N. Abazajian, N. Canac, S. Horiuchi, and M. Kaplinghat, Phys.Rev. **D90**, 023526 (2014), 1402.4090.
[13] T. Daylan, D. P. Finkbeiner, D. Hooper, T. Linden, S. K. N. Portillo, N. L. Rodd, and T. R. Slatyer, Phys. Dark Univ. **12**, 1 (2016), 1402.6703.
[14] F. Calore, I. Cholis, and C. Weniger, JCAP **1503**, 038 (2015), 1409.0042.
[15] K. N. Abazajian, N. Canac, S. Horiuchi, M. Kaplinghat, and A. Kwa, JCAP **1507**, 013 (2015), 1410.6168.
[16] M. Ajello et al. (Fermi-LAT), Astrophys. J. **819**, 44 (2016), 1511.02938.
[17] O. Macias, C. Gordon, R. M. Crocker, B. Coleman, D. Paterson, S. Horiuchi, and M. Pohl (2016), 1611.06644.
[18] H. A. Clark, P. Scott, R. Trotta, and G. F. Lewis (2016), 1612.01539.
[19] K. N. Abazajian, JCAP **1103**, 010 (2011), 1011.4275.
[20] D. Hooper, I. Cholis, T. Linden, J. Siegal-Gaskins, and

T. R. Slatyer, Phys.Rev. **D88**, 083009 (2013), 1305.0830.
[21] F. Calore, M. Di Mauro, F. Donato, and F. Donato, Astrophys. J. **796**, 1 (2014), 1406.2706.
[22] I. Cholis, D. Hooper, and T. Linden, JCAP **1506**, 043 (2015), 1407.5625.
[23] J. Petrović, P. D. Serpico, and G. Zaharijas, JCAP **1502**, 023 (2015), 1411.2980.
[24] Q. Yuan and K. Ioka, Astrophys. J. **802**, 124 (2015), 1411.4363.
[25] R. M. O'Leary, M. D. Kistler, M. Kerr, and J. Dexter, 1504.02477 (2015).
[26] T. D. Brandt and B. Kocsis, Astrophys. J. **812**, 15 (2015), 1507.05616.
[27] T. Linden, N. L. Rodd, B. R. Safdi, and T. R. Slatyer (2016), 1604.01026.
[28] R. Bartels, S. Krishnamurthy, and C. Weniger, Phys. Rev. Lett. **116**, 051102 (2016), 1506.05104.
[29] H.-S. Zechlin, A. Cuoco, F. Donato, N. Fornengo, and A. Vittino, Astrophys. J. Suppl. **225**, 18 (2016), 1512.07190.
[30] M. Ackermann et al. (Fermi-LAT), Phys. Rev. Lett. **116**, 151105 (2016), 1511.00693.
[31] H.-S. Zechlin, A. Cuoco, F. Donato, N. Fornengo, and M. Regis, Astrophys. J. **826**, L31 (2016), 1605.04256.
[32] M. Lisanti, S. Mishra-Sharma, L. Necib, and B. R. Safdi (2016), 1606.04101.
[33] T. Daylan, S. K. N. Portillo, and D. P. Finkbeiner (2016), 1607.04637.
[34] M. G. Aartsen et al. (IceCube), Phys. Rev. Lett. **111**, 021103 (2013), 1304.5356.
[35] M. G. Aartsen et al. (IceCube), Science **342**, 1242856 (2013), 1311.5238.
[36] M. G. Aartsen et al. (IceCube), Astrophys. J. **809**, 98 (2015), 1507.03991.
[37] M. G. Aartsen et al. (IceCube), Phys. Rev. Lett. **115**, 081102 (2015), 1507.04005.
[38] K. Bechtol, M. Ahlers, M. Di Mauro, M. Ajello, and J. Vandenbroucke (2015), 1511.00688.
[39] K. Murase and E. Waxman, MNRAS (2016), 1607.01601.
[40] G. Hasinger, R. Burg, R. Giacconi, G. Hartner, M. Schmidt, J. Trumper, and G. Zamorani, A&A **275**, 1 (1993).
[41] I. Georgantopoulos, G. C. Stewart, T. Shanks, R. E. Griffiths, and B. J. Boyle, MNRAS **262**, 619 (1993).
[42] K. C. Gendreau, X. Barcons, and A. C. Fabian, Mon. Not. Roy. Astron. Soc. **297**, 41 (1998), astro-ph/9711083.
[43] M. Perri and P. Giommi, Astron. Astrophys. **362**, L57 (2000), astro-ph/0006298.
[44] M. R. Feyereisen, S. Ando, and S. K. Lee, JCAP **1509**, 027 (2015), 1506.05118.
[45] M. R. Feyereisen, I. Tamborra, and S. Ando (2016), 1610.01607.
[46] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith, Computing in Science Engineering **13**, 31 (2011), ISSN 1521-9615.
[47] F. Feroz, M. P. Hobson, and M. Bridges, Mon. Not. Roy. Astron. Soc. **398**, 1601 (2009), 0809.3437.
[48] J. Buchner, A. Georgakakis, K. Nandra, L. Hsu, C. Rangel, M. Brightman, A. Merloni, M. Salvato, J. Donley, and D. Kocevski, Astron. Astrophys. **564**, A125 (2014), 1402.0004.
[49] F. Feroz, M. P. Hobson, E. Cameron, and A. N. Pettitt (2013), 1306.2144.
[50] F. Feroz and M. P. Hobson, Mon. Not. Roy. Astron. Soc. **384**, 449 (2008), 0704.3704.
[51] J. Skilling, Bayesian Anal. **1**, 833 (2006), URL http://dx.doi.org/10.1214/06-BA127.
[52] K. M. Gorski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelman, Astrophys. J. **622**, 759 (2005), astro-ph/0409513.
[53] F. Pérez and B. E. Granger, Computing in Science and Engineering **9**, 21 (2007), ISSN 1521-9615, URL http://ipython.org.
[54] D. Foreman-Mackey, W. Vousden, A. Price-Whelan, M. Pitkin, V. Zabalza, G. Ryan, Emily, M. Smith, G. Ashton, K. Cruz, et al., *corner.py: corner.py v2.0.0* (2016), URL https://doi.org/10.5281/zenodo.53155.
[55] J. D. Hunter, Computing In Science & Engineering **9**, 90 (2007).
[56] F. Johansson et al., *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)* (2013), http://mpmath.org/.
[57] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, F. Rossi, and R. Ulerich, Library available online at http://www. gnu. org/software/gsl (2015).
[58] T. E. Oliphant, *A guide to NumPy*, vol. 1 (Trelgol Publishing USA, 2006).
[59] F. Acero et al. (Fermi-LAT) (2015), 1501.02003.
[60] M. Su, T. R. Slatyer, and D. P. Finkbeiner, Astrophys.J. **724**, 1044 (2010), 1005.5480.