Notes on Binned Likelihood Calculation in the Fermi-LAT Science Tools.

Some folks

ABSTRACT

This is a collection of notes and equations about the binned log likelihood calculation in the Fermi-LAT Science Tools.

1. Introduction

The negative log-likelihood is:

$$-\ln \mathcal{L} = \sum_{i}^{\text{pix}} \sum_{k}^{\text{energy}} N_{ik} \ln M_{ik} - M_{ik}, \tag{1}$$

where N_{ik} and M_{ik} are respectively the number of observed and model counts for pixel i and energy bin k. The sum runs over all pixels and energy bins in the analysis.

In the case of a weighted analysis this becomes:

$$-\ln \tilde{\mathcal{L}} = \sum_{i}^{\text{pix}} \sum_{k}^{\text{energy}} w_{ik} \cdot (N_{ik} \ln M_{ik} - M_{ik}), \tag{2}$$

where w_{ik} is the weight assigned to pixel and energy bin ik.

We also want to calculate the derivatives of the likelihood with respect to the free parameters (x_α):

$$\frac{-\partial \ln \tilde{\mathcal{L}}}{\partial x_\alpha} = \sum_{i}^{\text{pix}} \sum_{k}^{\text{energy}} \frac{w_{ik} N_{ik}}{M_{ik}} \frac{\partial M_{ik}}{\partial x_\alpha} - w_{ik} \frac{\partial M_{ik}}{\partial x_\alpha}. \tag{3}$$

There are a number of complications in the way that these quantities are calculated.

1. The spatial binning can be either WCS or HEALPix based.
2. Some quantities are integral quantities, defined over the energy bins, and some are differential, defined at the bin edges. We have to take care to treat the two cases properly.

[11]W. W. Hansen Experimental Physics Laboratory, Kavli Institute for Particle Astrophysics and Cosmology, Department of Physics and SLAC National Accelerator Laboratory, Stanford University, Stanford, CA 94305, USA

3. We need need to treat both the weighted and unweighted cases.

4. We need to retain the ability to deal with energy dispersion.

5. For speed of evaluation we calculate the summation terms separately. This allows us to take certain short cuts in the two different cases.

6. For speed, we also precompute and cache many parts of the calculation.

In this note we are going to ignore the difference between WCS and HEALPix based maps. For our purposes this just means that the pixel index i runs over the flattened map in the WCS case.

In principle we should be working with the energy bin edges: E_k^-, E_k^+. However, we always use connected energy bins, so that the lower edge of one energy bin is the upper edge of the previous bin: $E_k \equiv E_k^- = E_{k-1}^+$.

2. Evaluation of the likelihood function

Most of the work in evaluation the likelihood function comes in computing the model counts (§2.1 and §2.2). Then, as stated above, we calculate the two terms in the likelihood expression separately: the first is the summation over $w_{ik} N_{ik} \ln M_{ik}$ (§2.5) and the second is the summation over $w_{ik} M_{ik}$ (§2.8).

2.1. Computing the model counts: M.

To compute the model counts we sum over the model counts, M_{ijk} of the sources in the model, indexed by j:

$$M_{ik} = \sum_j^{\rm src} M_{ijk}. \tag{4}$$

We also precompute the "spatial" part of the model counts m_{ijk} , i.e., the spatial model convolved with the point-spread function (PSF) and the effective area $A_{\rm eff}$. Those computations are discussed elsewhere.

We then use log-log quadrature to integrate the product of the spatial and spectral parts of the model over each energy bin. This gives us:

$$M_{ijk} = (m_{ijk} S_{jk} E_k + m_{ijk+1} S_{jk+1} E_{k+1}) \cdot \frac{1}{2} \ln \frac{E_{k+1}}{E_k}, \tag{5}$$

where m_{ijk} is the precomputed value of the "source map" and S_{jk} is the spectral model for source j at energy bin edge k.

Since logarithms are expensive, we precompute the term $r_k = \ln \frac{E_{k+1}}{E_k}$. Also, for each source we note the product $S_{jk} E_K \frac{r_k}{2}$ only depends on the energy bin, so we can precompute and cache two "spectral weights" ($\tilde{S}^{\pm}_{jk}$) for each energy bin:

$$\begin{aligned} \tilde{S}^{-}_{jk} &= S_{jk} E_k \frac{r_k}{2}, \\ \tilde{S}^{+}_{jk} &= S_{jk+1} E_{k+1} \frac{r_k}{2}. \end{aligned} \tag{6}$$

Then we can rewrite the model counts as:

$$M_{ijk} = m_{ijk} \tilde{S}^{-}_{jk} + m_{ijk+1} \tilde{S}^{+}_{jk}. \tag{7}$$

We have reduced the number of operations (and the complexity of those operations) in the innermost loop of the likelihood evalation.

To speed up the calculation we separately compute and cache the model counts for the all of the fixed sources:

$$M^{\text{fixed}}_{ik} = \sum_{j}^{\text{fixed}} M_{ijk}. \tag{8}$$

So that the total model weights are:

$$M_{ik} = M^{\text{fixed}}_{ik} + \sum_{j}^{\text{free}} M_{ijk}. \tag{9}$$

We can also save the fixed counts spectrum, C^{fixed}_{j}:

$$C^{\text{fixed}}_{k} = \sum_{i}^{\text{pix}} M^{\text{fixed}}_{ik}. \tag{10}$$

2.2. Applying the energy dispersion.

To apply the energy dispersion we first precompute the detector response matrix (DRM, D_{kl}) that gives the chances for a photon in true energy bin l to be observed in energy bin k.

We can then apply the energy dispersion in one of two different ways, depending on the configuration parameters.

2.3. Applying the energy dispersion with scaling

In this case we first compute the unconvolved energy spectrum for a source, given the current model by summing the model over all the pixels for that energy layer:

$$U_{jl} = \sum_{i}^{\text{pix}} M_{ijl}. \tag{11}$$

And use that to compute the convolved spectrum:

$$C_{jk} = \sum_{l}^{\text{energies}} D_{kl} U_{jl}. \tag{12}$$

For each source we take the ratio of the convolved to unconvolved spectra in each energy bin (i.e., for k=l) as a correction factor for that energy for that source:

$$\xi_{jk} = C_{jk} U_{jk}. \tag{13}$$

We use this to adjust the model energy spectrum from true energy to measured energy:

$$M_{ijk} = \xi_{jk} M_{ijk}^{\text{true}}. \tag{14}$$

So that the equation for the model becomes:

$$M_{ijk} = \xi_{jk} \cdot (m_{ijk} S_{jk} E_k + m_{ijk+1} S_{jk+1} E_{k+1}) \frac{r_k}{2}. \tag{15}$$

Then we can write the model in terms of the spectral weights as:

$$M_{ijk} = \xi_{jk} \cdot (m_{ijk} \tilde{S}_{jk}^{-} + m_{ijk+1} \tilde{S}_{jk}^{+}). \tag{16}$$

This method has the flaw that it does not properly account for the changing PSF as a function of energy. It merely rescales the counts in the spectral domain using the ξ_{jk} parameters.

2.4. Applying the energy dispersion with adjacent bins

To more correctly deal with energy dispersion in conjunction with the PSF, we should actually apply the summation over D_{kl} for every single pixel. This will greatly slow down the execution time. So as a compromise we only do the sum over the nearest energy bins, which are most likely to contribute:

$$M_{ijk} = \sum_{l}^{\mathrm{k\pm n}} D_{kl} M_{ijl}^{\mathrm{true}}. \tag{17}$$

Where the sum over true energy bins runs from $k - n$ to $k + n$ where n is user-configurable and typically 1 or 2.

Expanding this in term of the spectral weights we obtain:

$$M_{ijk} = \sum_{l}^{\mathrm{k\pm n}} D_{kl}(m_{ijl}\tilde{S}_{jl}^{-} + m_{ijl+1}\tilde{S}_{jl}^{+}). \tag{18}$$

2.5. The summation over $WN \ln M$.

For this term we note that only bins with observed counts contribute to the sum. Therefore, in general we will identify the "filled" pixels once and then only sum over those.

We can merge the $W_{ik}N_{ik}$ into a "weighted counts map" with $\tilde{N_{ik}} = W_{ik}N_{ik}$. We do this because there are a number of reasons to carry around the weighed counts map, e.g., for debugging and diagnostic purposes, and because it is more efficient to use the weighted counts map in many computations.

$$\sum_{i}^{\mathrm{pix}} \sum_{k}^{\mathrm{energy}} \tilde{N_{ik}} \ln M_{ik}, \tag{19}$$

where in practice we will only sum over the "filled" pixels, where $\tilde{N_{ik}} \neq 0$. This also means that pixels with zero weight will be ignored in the summation over filled pixels.

2.6. The summation over M.

For this term we all the bins with observed counts contribute to the sum, but we note that we can use something similar to the model weights to speed things up. Specifically, the sum of the source maps over all the pixels in a energy layer does not change as we are fitting the sources. This sum is called "Npred" in the code, which is unfortunate, as it is not really a predicted number of counts, but rather something that you can used to get the predicted number of counts. Specifically we can define the sums:

$$P_{jk} = \sum_{i} m_{ijk}, \tag{20}$$

Then the unconvolved and convolved spectra for a free source j become:

$$
\begin{aligned}
U_{jk} &= (P_{jk}S_{jk}E_k + P_{jk+1}S_{jk+1}E_{k+1})\frac{r_k}{2}, \\
C_{jk} &= \xi_{jk}(P_{jk}S_{jk}E_k + P_{jk+1}S_{jk+1}E_{k+1})\frac{r_k}{2}, \\
C_{jk} &= \sum_{l}^{\mathrm{k\pm n}} D_{kl}(P_{jl}S_{jl}E_l + P_{jl+1}S_{jl+1}E_{l+1})\frac{r_l}{2}.
\end{aligned} \tag{21}
$$

Where we pick the second or third equation depending on how we are applying the energy dispersion.

We can rewrite this in terms of the spectra weights as:

$$
\begin{aligned}
U_{jk} &= P_{jk}\tilde{S}^{-}_{jk} + P_{jk+1}\tilde{S}^{+}_{jk}, \\
C_{jk} &= \xi_{jk}(P_{jk}\tilde{S}^{-}_{jk} + P_{jk+1}\tilde{S}^{+}_{jk}), \\
C_{jk} &= \sum_{l}^{\mathrm{k\pm n}} D_{kl}(P_{jl}\tilde{S}^{-}_{jl} + P_{jl+1}\tilde{S}^{+}_{jl}).
\end{aligned} \tag{22}
$$

2.7. Applying weights to the summation over M.

In this case we have to account for the weights when we sum over the pixels. Also, we want to be able to switch between the weighted and unweighted case easily, so we compute the weighted npreds on either side of the energy bin:

$$
\begin{aligned}
\tilde{P}^{-}_{jk} &= \sum_{i} w_{ik} m_{ijk} \\
\tilde{P}^{+}_{jk} &= \sum_{i} w_{ik} m_{ijk+1}.
\end{aligned} \tag{23}
$$

Then the weighted unconvolved and convolved spectrum for a free source j if we are using the rescaling method become:

$$
\begin{aligned}
\tilde{U}_{jk} &= (\tilde{P}^{-}_{jk}S_{jk}E_k + \tilde{P}^{+}_{jk}S_{jk+1}E_{k+1})\frac{r_k}{2}, \\
\tilde{C}_{jk} &= \xi_{jk}(\tilde{P}^{-}_{jk}S_{jk}E_k + \tilde{P}^{+}_{jk}S_{jk+1}E_{k+1})\frac{r_k}{2}.
\end{aligned} \tag{24}
$$

The only differences with respect to Eq. 22 are the $\tilde{P}_{jk}$ terms. Of course we can rewrite this in terms of the spectral weights:

$$\tilde{U}_{jk} = \tilde{P}^{-}_{jk}\tilde{S}^{-}_{jk} + \tilde{P}^{+}_{jk}\tilde{S}^{+}_{jk},$$
$$\tilde{C}_{jk} = \xi_{jk}(\tilde{P}^{-}_{jk}\tilde{S}^{-}_{jk} + \tilde{P}^{+}_{jk}\tilde{S}^{+}_{jk}). \tag{25}$$

If we are using the adjacent bins energy dispersion treatment this becomes quite a bit more complicated. This is because we must account for the energy dispersion when computing the weighted npreds.

$$\begin{aligned}\tilde{P}^{-}_{jkl} &= \sum_i w_{ik} \sum_l^{\mathrm{k\pm n}} D_{kl} m_{ijl} \\ \tilde{P}^{+}_{jkl} &= \sum_i w_{ik} \sum_l^{\mathrm{k\pm n}} D_{kl} m_{ijl+1}.\end{aligned} \tag{26}$$

Where, once again the sum over true energy bins (l) runs from $k-n$ to $k+n$ where n is user-configurable and typically 1 or 2. Then the weighted, convolved spectra become:

$$\tilde{C}_{jk} = \sum_l^{\mathrm{k\pm n}} \tilde{P}^{-}_{jkl}\tilde{S}^{-}_{jl} + \tilde{P}^{+}_{jkl}\tilde{S}^{+}_{jl}. \tag{27}$$

2.8. Caching various fixed source spectra.

We note that we can save time by caching the convolved and unconvolved and weighed and unweighted version of the fixed counts spectra:

$$\begin{aligned}U^{\mathrm{fixed}}_k &= \sum_j^{\mathrm{fixed}} U_{jk}, \\ C^{\mathrm{fixed}}_k &= \sum_j^{\mathrm{fixed}} C_{jk}, \\ \tilde{U}^{\mathrm{fixed}}_k &= \sum_j^{\mathrm{fixed}} \tilde{U}_{jk}, \\ \tilde{C}^{\mathrm{fixed}}_k &= \sum_j^{\mathrm{fixed}} \tilde{C}_{jk}.\end{aligned} \tag{28}$$

3. Analytic derivatives

In general, we want the derivatives of the weighted log likelihood w.r.t., the free parameters:

$$\frac{-\partial \ln \tilde{\mathcal{L}}}{\partial x_\alpha} = \sum_i^{\text{pix}} \sum_k^{\text{energy}} \frac{w_{ik} N_{ik}}{M_{ik}} \frac{\partial M_{ik}}{\partial x_\alpha} - w_{ik} \frac{\partial M_{ik}}{\partial x_\alpha}. \tag{29}$$

As before, we can replace the observed counts with the weighed observed counts $\tilde{N_{ik}} = w_{ik} N_{ik}$ and split the evaluation into two sums, one over the filled pixels (§3.1), the second over the derivatives of the total models counts for each of the free sources (§3.2).

3.1. The derivatives $\frac{\partial M_{ijk}}{\partial x_\alpha}$

The first term of Eq. 29 is:

$$\sum_i^{\text{pix}} \sum_k^{\text{energy}} \frac{w_{ik} N_{ik}}{M_{ik}} \frac{\partial M_{ik}}{\partial x_\alpha} = \sum_i^{\text{pix}} \sum_k^{\text{energy}} \frac{\tilde{N}_{ik}}{M_{ik}} \sum_j^{\text{src}} \frac{\partial M_{ijk}}{\partial x_\alpha}. \tag{30}$$

We write it like this because we have already computed M_{ij} so we do not need to re-do the sums over sources there. However, we need to evaluate the derivatives $\frac{\partial M_{ijk}}{\partial x_\alpha}$. We can ignore the fixed sources, as they won't contribute to the partial derivative. Also, in general we are only varying spectral parameters, so only the spectral terms (the S_{jk}) are relevant. (This neglects the effect of the change in the energy dispersion corrections: ξ_{jk}, but that is a higher order effect.) Finally, in the first term this is multiplied by $\tilde{N_{ik}}$ so we only need to sum over the filled pixels. If we are using the rescaling method of handling the energy dispersion this yields:

$$\frac{\partial M_{ijk}}{\partial x_\alpha} = \xi_{jk} \cdot (m_{ijk} \frac{\partial S_{jk}}{\partial x_\alpha} E_k + m_{ijk+1} \frac{\partial S_{jk+1}}{\partial x_\alpha} E_{k+1}) \frac{r_k}{2}. \tag{31}$$

We pre-compute and cache the various $\delta_{jk\alpha} = \frac{\partial S_{jk}}{\partial x_\alpha}$ so we have:

$$\frac{\partial M_{ijk}}{\partial x_\alpha} = \xi_{jk} \cdot (m_{ijk} \delta_{jk\alpha} E_k + m_{ijk+1} \delta_{jk+1\alpha} E_{k+1}) \frac{r_k}{2}. \tag{32}$$

All of these terms except for the $\delta_{jk\alpha}$ were already computed in evaluating the likelihood. We do need to make check if any of the parameter values have changed since the evaluation of the likelihood, and possibly updated the cached M_{ik} in the prefactor. We can further speed up the evaluation by pre-computing the terms that depend only on the energy bin:

$$\tilde{\delta}^{-}_{jk\alpha} = \delta_{jk\alpha} E_k \frac{r_k}{2},$$
$$\tilde{\delta}^{+}_{jk\alpha} = \delta_{jk+1\alpha} E_{k+1} \frac{r_k}{2}. \tag{33}$$

So that we are evaluating:

$$\frac{\partial M_{ijk}}{\partial x_\alpha} = \xi_{jk} \cdot (m_{ijk} \tilde{\delta}^{-}_{jk\alpha} + m_{ijk+1} \tilde{\delta}^{+}_{jk\alpha}) \tag{34}$$

If, on the other hand, we are using the adjacent bins method of handling the energy dispersion we have to sum over the nearby true energy bins:

$$\frac{\partial M_{ijk}}{\partial x_\alpha} = \sum_{l}^{\mathrm{k\pm n}} D_{kl} (m_{ijl} \frac{\partial S_{jl}}{\partial x_\alpha} E_l + m_{ijl+1} \frac{\partial S_{jl+1}}{\partial x_\alpha} E_{l+1}) \frac{r_l}{2}. \tag{35}$$

Or, in terms of the precomputed quantities:

$$\frac{\partial M_{ijk}}{\partial x_\alpha} = \sum_{l}^{\mathrm{k\pm n}} D_{kl} (m_{ijl} \tilde{\delta}^{-}_{jl\alpha} + m_{ijl+1} \tilde{\delta}^{+}_{jl\alpha}). \tag{36}$$

3.2. The derivative $\frac{\partial M}{\partial x_\alpha}$

The second term of Eq. 29 is:

$$\sum_{i}^{\mathrm{pix}} \sum_{k}^{\mathrm{energy}} w_{ik} \frac{\partial M_{ik}}{\partial x_\alpha} = \sum_{k}^{\mathrm{energy}} \sum_{j}^{\mathrm{src}} \sum_{i}^{\mathrm{pix}} w_{ik} \frac{\partial M_{ijk}}{\partial x_\alpha}, \tag{37}$$

where we need to sum over all the pixels, but we only need to evaluate this for the free sources. Since we are summing over all the pixels, we can express the derivatives $\frac{\partial M_{ijk}}{\partial x_\alpha}$ in terms of $\tilde{U}_{jk}$ and $\tilde{C}_{jk}$ which we computed in the likelihood evaluation:

$$\sum_{i}^{\mathrm{pix}} \frac{\partial M^{\mathrm{true}}_{ijk}}{\partial x_\alpha} = \frac{\partial \tilde{U}_{jk}}{\partial x_\alpha},$$
$$\sum_{i}^{\mathrm{pix}} \frac{\partial M^{\mathrm{meas}}_{ijk}}{\partial x_\alpha} = \frac{\partial \tilde{C}_{jk}}{\partial x_\alpha}. \tag{38}$$

The derivative of the weighted unconvolved and convolved spectrum for a free source j:

$$\begin{aligned}
\frac{\partial \tilde{U}_{jk}}{\partial x_\alpha} &= (P^-_{jk}\delta_{jk\alpha}E_k + P^+_{jk+1}\delta_{jk+1\alpha}E_{k+1})\frac{r_k}{2}, \\
\frac{\partial \tilde{C}_{jk}}{\partial x_\alpha} &= \xi_{jk}(\tilde{P}^-_{jk}\delta_{jk\alpha}E_k + \tilde{P}^+_{jk}\delta_{jk+1\alpha}E_{k+1})\frac{r_k}{2}, \\
\frac{\partial \tilde{C}_{jk}}{\partial x_\alpha} &= \sum_l^{\mathrm{k\pm n}} D_{kl}(\tilde{P}^-_{jkl}\delta_{jl\alpha}E_l + \tilde{P}^+_{jkl}\delta_{jl+1\alpha}E_{l+1})\frac{r_l}{2}.
\end{aligned} \tag{39}$$

Where the second and third form are for the rescaling and adjacent bins versions of the energy dispersion, respectively. We can re-write those in terms of the spectral derivative weights, $\tilde{\delta}^{\pm}_{jk\alpha}$ as:

$$\begin{aligned}
\frac{\partial \tilde{U}_{jk}}{\partial x_\alpha} &= P^-_{jk}\tilde{\delta}^-_{jk\alpha} + P^+_{jk}\tilde{\delta}^+_{jk\alpha}, \\
\frac{\partial \tilde{C}_{jk}}{\partial x_\alpha} &= \xi_{jk}(P^-_{jk}\tilde{\delta}^-_{jk\alpha} + P^+_{jk}\tilde{\delta}^+_{jk\alpha}), \\
\frac{\partial \tilde{C}_{jk}}{\partial x_\alpha} &= \sum_l^{\mathrm{k\pm n}} D_{kl}(P^-_{jl}\tilde{\delta}^-_{jl\alpha} + P^+_{jl}\tilde{\delta}^+_{jl\alpha})
\end{aligned} \tag{40}$$

As with the first term, all of these terms except for the $\delta_{jk\alpha}$ were already computed in evaluating the likelihood.

Thanks.

A. Access to various quantities.

These are the access functions used to retrieve various quantities. These all assume that the user has an object of class `BinnedLikelihood` stored in the variable `blike`. Some of them refer to quantities associated with a particular source with `name`.

- N_{ik}: `blike.countsMap()`. Returns a pointer to an object of class `CountsMapBase`.
- $\tilde{N_{ik}}$: `blike.dataCache().weightedCounts()`. Returns a pointer to an object of class `CountsMapBase` that may be null if no weights have been supplied.
- E_k: `blike.energies()`. Returns a vector of doubles, whose length is equal to the number of energy bin edges.
- r_k: `blike.dataCache().log_energy_ratios()`. Returns a vector of doubles, whose length is equal to the number of energy bins.

- w_{ik}: `blike.weightMap()`. Returns a pointer to an object of class `WeightMap` that may be null if no weights have been supplied.
- M_{ik}: `blike.computeModelMap(modelMap)`. Fills `modelMap` with the M_{ik} values and returns void.
- M_{ijk}: `blike.computeModelMap(name, modelMap)`. Fills `modelMap` with the M_{ijk} values for the source in question and returns void.
- $M_{ik}^{\rm fixed}$: `blike.fixedModelCounts()`. Returns a vector of doubles, for only the filled pixels.
- $U_{k}^{\rm fixed}$: `blike.fixedModelSpectrum()`. Returns a vector of doubles, whose length is equal to the number of energy bins.
- $\tilde{U}_{k}^{\rm fixed}$: `blike.fixedModelSpectrum_wt()`. Returns a vector of doubles, whose length is equal to the number of energy bins.
- $C_{k}^{\rm fixed}$: `blike.fixedModelSpectrum_edisp()`. Returns a vector of doubles, whose length is equal to the number of energy bins.
- $\tilde{C}_{k}^{\rm fixed}$: `blike.fixedModelSpectrum_edisp_wt()`. Returns a vector of doubles, whose length is equal to the number of energy bins.
- m_{ijk}: `blike.sourceMap(name).cached_model()`. Returns a vector of floats, whose length is equal to the number of energy bin edges times the number of pixels.
- S_{jk}: `blike.sourceMap(name).cached_specVals()`. Returns a vector of doubles, whose length is equal to the number of energy bin edges.
- $\tilde{S}_{jk}^{\pm}$: `blike.sourceMap(name).cached_specWts()`. Returns a vector of pairs of doubles, whose length is equal to the number of energy bins.
- P_{jk}: `blike.sourceMap(name).cached_npreds()`. Returns a vector of doubles, whose length is equal to the number of energy bin edges.
- $P_{jkl}^{\pm}$: `blike.sourceMap(name).cached_weighted_npreds()`. Returns a vector of vector of pairs of doubles, whose length is equal to the number of energy bin edges. Each sub-vector has length equal to the number of bins being used in the energy dispersion calculation.
- U_{jk}: `blike.sourceMap(name).cached_drm_cache()->true_counts()`. Returns a vector of doubles, whose length is equal to the number of energy bins.
- C_{jk}: `blike.sourceMap(name).cached_drm_cache()->meas_counts()`. Returns a vector of doubles, whose length is equal to the number of energy bins.
- ξ_{jk}: `blike.sourceMap(name).cached_drm_cache()->xi()`. Returns a vector of doubles, whose length is equal to the number of energy bins.

- $\tilde{U}_{jk}$: `blike.sourceMap(name).cached_drm_cache()->true_counts_wt()`. Returns a vector of doubles, whose length is equal to the number of energy bins.
- $\tilde{C}_{jk}$: `blike.sourceMap(name).cached_drm_cache()->meas_counts_wt()`.. Returns a vector of doubles, whose length is equal to the number of energy bins.
- $\delta_{jk\alpha}$: `blike.sourceMap(name).cached_specDerivs()`. Returns a vector of vector of doubles, whose dimensions are the number of free parameters (for that source) and the number of energy bin edges.

B. Functions and caching

These are the higher level functions used to calculate and cache various quantities. Some of the functions assume that the user has an object of class `SourceMap` stored in the variable `srcMap`.

- `blike.value()`: returns the negative log likelihood value, $-\ln\mathcal{L}$.
- `blike.getFreeDerivs(derivs)`: fills derivs with the derivatives of the negative log likelihood, $\frac{\partial \ln \tilde{\mathcal{L}}}{\partial x_\alpha}$.
- `blike.computeModelMap_internal()`: computes and caches the M_{ik}, $U_k^{\rm fixed}$, $C_k^{\rm fixed}$, $\tilde{U}_k^{\rm fixed}$, $\tilde{C}_k^{\rm fixed}$ and everything needed to compute those.
- `blike.buildFixedModelWts()`: computes and caches the $M_{ik}^{\rm fixed}$, $U_k^{\rm fixed}$, $C_k^{\rm fixed}$, $\tilde{U}_k^{\rm fixed}$, $\tilde{C}_k^{\rm fixed}$ and everything needed to compute those.
- `srcMap.setSpectralValues(energies)`: computes and caches the S_{jk} and $\tilde{S}_{jk}^{\pm}$.
- `srcMap.setSpectralDerivs(energies, paramNames)`: computes and caches the $\delta_{jk\alpha}$.
- `srcMap.computeNpredArray()`: computes and caches the U_{jk}, C_{jk}, ξ_{jk} $\tilde{U}_{jk}$, and $\tilde{C}_{jk}$.
- `srcMap.update_drm_cache(drm)`: computes and caches the U_{jk}, C_{jk}, ξ_{jk} $\tilde{U}_{jk}$, and $\tilde{C}_{jk}$.

These are the lower level functions used to calculate specific quantities. These are generally static functions in the `FitUtils` namespace.

- `FitUtils::get_edisp_range(const BinnedCountsCache& dataCache, int edisp_val, size_t k, size_t& kmin, size_t& kmax)`: finds the range of energy bins to sum over to apply the energy dispersion for a given bin k, and fills kmin and kmax with the ends of that range.

- `FitUtils::get_edisp_constants(SourceMap& srcMap, const BinnedCountsCache& dataCache, int edisp_val, size_t k, size_t& kmin, size_t& kmax, std::vector⟨double⟩& edisp_col)`: depending on the energy dispersion implementation, fills the edisp_col variable with the values of either D_{kl} or ξ_{jk}. Note that the first value in edisp_col correspond to the value for kmin.

- `FitUtils::get_spectral_weights(const std::vector⟨double⟩& spec, const BinnedCountsCache& dataCache, std::vector⟨std::pair⟨double, double⟩⟩&spec_wts)`: combines the spectral values (spec, s_{jk}) with the energy binning (dataCache, E_k and r_k) and computes the spectral weights ($\tilde{S}^{\pm}_{jk}$) and fills the spec_wts vector of doubles with those weights.

- `FitUtils::model_counts_contribution(SourceMap& srcMap, std::vector⟨std::pair⟨double, double⟩⟩& spec_wts, const double& xi, size_t npix, size_t kref, size_t ipix)`: finds and returns the model counts contribution m_{ijk} for a specific pixel and energy bin. This implementation is used when energy dispersion is turned off, or when the rescaling method is used for energy dispersion.

- `FitUtils::model_counts_edisp(SourceMap& srcMap, std::vector⟨std::pair⟨double, double⟩⟩& spec_wts, std::vector⟨double⟩& edisp_col, size_t ipix, size_t npix, size_t kmin, size_t kmax)`: finds and returns m_{ijk} for a specific pixel and energy bin. This implementation is used when energy dispersion adjacent bins method is used for energy dispersion.

- `FitUtils::npred_contribution(std::vector⟨double⟩& npred_vals, std::pair⟨double, double⟩& npred_weights, std::vector⟨std::pair⟨double, double⟩⟩& spec_wts, const double& xi, size_t kref, double& counts, double& counts_wt)`: finds both the weighted and unweighted counts for a specific energy bin and fill the count and counts_wt variables with them. This implementation is used when energy dispersion is turned off, or when the rescaling method is used for energy dispersion.

- `FitUtils::npred_edisp(std::vector⟨double⟩& npred_vals, std::vector⟨std::pair⟨double, double⟩⟩& npred_weights, std::vector⟨std::pair⟨double, double⟩⟩& spec_wts, std::vector⟨double⟩& edisp_col, size_t kmin, size_t kmax, double& counts, double& counts_wt)`: finds both the weighted and unweighted counts for a specific energy bin and fill the count and counts_wt variables with them. This implementation is used when energy dispersion adjacent bins method is used for energy dispersion..

- `FitUtils::addSourceCounts(std::vector⟨double⟩& modelCounts, SouceMap& srcMap, const BinnedCountsCache& dataCache, int edisp_val, bool subtract)`: adds (or subtracts if subtract is true) the model counts for a particular source to the modelCounts vector.

- `FitUtils::addFixedNpreds(std::vector⟨double⟩& fixed_counts_spec, std::vector⟨double⟩& fixed_counts_spec_wt, std::vector⟨double⟩& fixed_counts_spec_edisp, std::vector⟨double⟩& fixed_counts_spec_edisp_wt, SouceMap& srcMap, const BinnedCountsCache& dataCache, int edisp_val, bool subtract)`: adds (or subtracts if subtract is true) the various convolved and weighted versions of the npreds from a fixed source on the relevant vectors.

- `FitUtils::addFreeDerivs(std::vector⟨Kahan_Accumulator⟩& posDerivs, std::vector⟨Kahan_Accumulator⟩& negDerivs, long freeIndex, SourceMap& srcMap, const std::vector⟨double⟩& data_over_model, const BinnedCountsCache& dataCache, int edisp_val, size_t kmin, size_t kmax)`: adds the contributions to the derivatives of the log-likelihood for a single source.

- `FitUtils::updateModelMap(std::vector⟨float⟩& modelMap, SourceMap& srcMap, const BinnedCountsCache& dataCache, bool use_mask, int edisp_val)`: computes the model counts for a given source and fills the modelMap vector with them. If use_mask is true it will only fill the un-masked pixels.