PyUnfold: A Python package for iterative unfolding

James Bourbeau¹ and Zigfried Hampel-Arias¹, ²

¹ University of Wisconsin - Madison, Wisconsin, USA ² IIHE, Université Libre de Bruxelles, Bruxelles, Belgium

Summary

In an ideal world, experimentalists would have access to the perfect detector: an apparatus that makes no error in measuring a desired quantity. However, in real life scenarios, detectors have:

- Finite resolutions
- Characteristic biases that cannot be eliminated
- Less than 100% detection efficiencies
- Statistical uncertainties

By building a matrix that encodes a detector’s smearing of the desired true quantity into the measured observable(s), a deconvolution can be performed that provides an estimate of the true variable. This deconvolution process is known as unfolding.

PyUnfold is an extensible framework for the unfolding of discrete probability distributions via the iterative unfolding method described in (D’Agostini 1995). This method falls into the general class of inverse problems, and is especially powerful when no analytic form of the inverse function is explicitly available, instead requiring an estimate (e.g. via a finite amount of simulation) of the response function. Given that measured data comprise a finite sample, PyUnfold also implements the uncertainty contributions stated in (Adye 2011).

The unfolding method itself is data-agnostic, referring to the measurement process as the smearing of a set of true causes into a set of detectable effects. For example one could define as causes the true energy of a particle and the effects the measured energy of that particle in a detector. Another example might be a set of diseases (causes) and possible clinical symptoms (effects). So long as it is possible to encode estimable resolutions and biases connecting causes to effects in a binned response matrix, one can perform a deconvolution with PyUnfold.

The primary purpose of PyUnfold is to provide an unfolding toolkit for members of all scientific disciplines in an easy-to-use package. For example, unfolding methods are commonly used in the analysis pipeline of the high-energy physics (HEP) community. However, the deconvolution packages used in HEP maintain a strong dependence on the ROOT data analysis framework (ROOT 1997), which is almost exclusively used in HEP. Instead, PyUnfold is built on top of the Python scientific computing stack (i.e. NumPy, SciPy, and pandas), thus broadening its scope to a general scientific audience.

PyUnfold has been designed to be both easy to use for first-time users as well as flexible enough for fine-tuning an analysis and seamlessly testing the robustness of results. It provides support for the following:

- Custom, user defined initial prior probability distributions, the default being the uniform prior. The non-informative Jeffreys’ prior (Jeffreys 1946) is accessible as a utility function.

By building a matrix that encodes a detector’s smearing of the desired true quantity into the measured observable(s), a deconvolution can be performed that provides an estimate of the true variable. This deconvolution process is known as unfolding.

PyUnfold is an extensible framework for the unfolding of discrete probability distributions via the iterative unfolding method described in (D’Agostini 1995). This method falls into the general class of inverse problems, and is especially powerful when no analytic form of the inverse function is explicitly available, instead requiring an estimate (e.g. via a finite amount of simulation) of the response function. Given that measured data comprise a finite sample, PyUnfold also implements the uncertainty contributions stated in (Adye 2011).

The unfolding method itself is data-agnostic, referring to the measurement process as the smearing of a set of true causes into a set of detectable effects. For example one could define as causes the true energy of a particle and the effects the measured energy of that particle in a detector. Another example might be a set of diseases (causes) and possible clinical symptoms (effects). So long as it is possible to encode estimable resolutions and biases connecting causes to effects in a binned response matrix, one can perform a deconvolution with PyUnfold.

The primary purpose of PyUnfold is to provide an unfolding toolkit for members of all scientific disciplines in an easy-to-use package. For example, unfolding methods are commonly used in the analysis pipeline of the high-energy physics (HEP) community. However, the deconvolution packages used in HEP maintain a strong dependence on the ROOT data analysis framework (ROOT 1997), which is almost exclusively used in HEP. Instead, PyUnfold is built on top of the Python scientific computing stack (i.e. NumPy, SciPy, and pandas), thus broadening its scope to a general scientific audience.

PyUnfold has been designed to be both easy to use for first-time users as well as flexible enough for fine-tuning an analysis and seamlessly testing the robustness of results. It provides support for the following:

- Custom, user defined initial prior probability distributions, the default being the uniform prior. The non-informative Jeffreys’ prior (Jeffreys 1946) is accessible as a utility function.
• Unfolding stopping criteria based on test statistic calculations comparing unfolded distributions from one iteration to the next. These include Kolmogorov-Smirnov (Kolmogorov 1933)(Smirnov 1948), $\chi^2$, relative difference, and Bayes factor (BenZvi et al. 2011) tests.

• Tunable spline regularization as a means of ensuring that unfolded distributions do not suffer from growing fluctuations potentially arising from the finite binning of the response matrix.

• Option to choose between Poisson or multinomial forms of the covariance matrices for both the data and response contributions to the uncertainty calculation.

• Multivariate unfolding via definitions of subsets of causes, which are regularized in their respective blocks or groups.

Further mathematical details regarding the iterative unfolding procedure, including complete derivations of the statistical and systematic uncertainty propagation can be found in the online documentation.

PyUnfold has been applied successfully for the measurement of the cosmic-ray energy spectrum by the HAWC Observatory (Alfaro 2017) and is currently being used in an analysis by members of the IceCube Neutrino Observatory.

Acknowledgements

The authors acknowledge support from the Wisconsin IceCube Particle Astrophysics Center at the UW-Madison, and especially for the guidance of Professor Stefan Westerhoff. We also acknowledge the financial support provided by the National Science Foundation, the Belgian American Educational Foundational Fellowship, and Wallonie-Bruxelles International.

References

Adye, T. 2011. “Corrected Error Calculation for Iterative Bayesian Unfolding.” [http://hepunx.rl.ac.uk/~adye/software/unfold/bayes_errors.pdf].

Alfaro, R. et al. 2017. “All-Particle Cosmic Ray Energy Spectrum Measured by the HAWC Experiment from 10 to 500 Tev.” Phys. Rev. D 96 (12):122001. [https://doi.org/10.1103/PhysRevD.96.122001].

BenZvi, S. Y., B. M. Connolly, C. G. Pfendner, and S. Westerhoff. 2011. “A Bayesian Approach to Comparing Cosmic Ray Energy Spectra.” The Astrophysical Journal 738 (1):82. [https://doi.org/10.1088/0004-637X/738/1/82].

D’Agostini, G. 1995. “A multidimensional unfolding method based on Bayes’ theorem.” Nucl. Instrum. Meth. A 362:487–98. [https://doi.org/10.1016/0168-9002(95)00274-X].

Jeffreys, H. 1946. “An Invariant Form for the Prior Probability in Estimation Problems.” Proc. Of the Royal Society of London A: Mathematical, Physical and Engineering Sciences 186 (1007). London, England:453–61. [https://doi.org/10.1098/rspa.1946.0056].

Kolmogorov, A. N. 1933. “Sulla Determinazione Empirica Di Una Legge Di Distribuzione.” Giornale Dell’Istituto Italiano Degli Attuari 4:83–91.

ROOT. 1997. “An Object Oriented Data Analysis Framework.” Nucl. Instr. & Meth. In Phys. Rev. A 389:81–86. [https://doi.org/10.1016/S0168-9002(97)00048-X].

Bourbeau et al., (2018). PyUnfold: A Python package for iterative unfolding. Journal of Open Source Software, 3(26), 741. [https://doi.org/10.21105/joss.00741]
Smirnov, N. 1948. “Table for Estimating the Goodness of Fit of Empirical Distributions.” *Ann. Math. Statist.* 19 (2). The Institute of Mathematical Statistics: 279–81. https://doi.org/10.1214/aoms/1177730256.

Bourbeau et al., (2018). PyUnfold: A Python package for iterative unfolding. *Journal of Open Source Software*, 3(26), 741. https://doi.org/10.21105/joss.00741
TOWARDS UNDERSTANDING ITERATIVE UNFOLDING

MAY 16, 2018

REVISION 1.0
Abstract

This reference outlines the D’Agostini procedure used for iterative unfolding. We first motivate and outline the method in a manner suitable for users new to the technique. Next is shown the full propagation of errors due to the unfolding process, including the derivation of the final form of the covariance matrix.

Contents

1 Introduction 3
2 Method 4
3 Regularization 6
4 Unfolding Uncertainties 7
   4.1 $V_{Data}$ 7
   4.2 $V_{MC}$ 8
   4.3 Updated Unfolding Algorithm 9
   4.4 Expansion of Components of $V$ 9
      4.4.1 Some useful formulae 9
      4.4.2 Expansion of $V_{Data}$ 10
      4.4.3 Expansion of $V_{MC}$ 11
1 Introduction

The general class of unfolding methods is amongst the physicist’s toolbox as a powerful means to connect an experiment’s observable variables with true physical quantities. Typically a matrix can be built to encompass the effects of the measurement process on a simulated ‘true’ distribution and the manifestation of said distribution as an experimenter’s desired observable. With this response matrix, a distribution of the observable in an experiment can be unfolded, providing an estimate of the true parent distribution.

A variety of unfolding methods exist, each with its respective strengths and weaknesses. For example, the simplest method is the matrix inversion unfolding, which for a well populated, highly linear response matrix can be both efficient and precise. However, even with relatively small off-diagonal elements, this method can be unfavorable, as the matrix may be singular or may introduce wildly fluctuating results due to limited statistics. There exist methods to quell such issues, though these require the tuning of various parameters which typically have no physical connection to the experiment at hand.

Here we discuss D’Agostini’s iterative unfolding technique presented in [1], a manifestly inferential method. Starting from Bayes’ theorem, an iterative unfolding procedure is developed, which then can be implemented without too much difficulty for the typical experimenter. This document has been adapted from Chapter 7 and Appendix B of [2].
2 Method
As discussed in the Section 1, the conceptually simplest way to connect true causes $C$ and observable effects $E$ is via their respective count distributions, $n$ and $\phi$, a reponse matrix, $R$, and it’s inverse $M^1$:

$$n(E) = R \phi(C),$$
$$\phi(C) = M n(E).$$

(1)

Due to the aforementioned potential difficulties in matrix inversion, we can take into consideration Bayes’ theorem,

$$P(C|E) = \frac{P(E|C) P(C)}{\sum_{\nu} P(E|C_{\nu}) P(C_{\nu})},$$

(2)

where $n_C$ is the number of possible causes. Equation 2 dictates that having observed the effect $E_j$, the probability that it’s origin is due to the cause $C$ is proportional to product of the probability of the cause and the probability of the cause to produce that effect. Hence, the elements $P(E_i|C)$ represent the probability that a given $C$ results in the effect $E_i$, and is the response matrix typically generated via modeling or simulation. Continuing with $P(C|E)$, we can then connect the measured observed effects to their causes by

$$\phi(C) = \sum_i^n P(C|E_i) n(E_i).$$

(3)

Stepping back to eq. 2 for a moment, one identifies $P(C)$ as the prior cause distribution, representing our current knowledge of the causes. The prior is a normalized distribution such that $\sum_{C} P(C) = 1$. This normalization requirement is not imposed on the response matrix efficiency $\epsilon$: $0 \leq \epsilon = \sum_{E} P(E_j|C) \leq 1$, i.e., a cause does not need to produce any effect. Taking this (in)-efficiency into account, we rewrite eq. 3 as

$$\phi(C) = \frac{1}{\epsilon} \sum_i^n P(C|E_i) n(E_i).$$

(4)

Identifying here the explicit form of $M$, the full matrix (Bayesian) inversion equation is then

$$\phi(C) = \sum_j M_{\mu j} n(E_j),$$

(5)

where

$$M_{\mu j} = \frac{P(E_i|C) P(C)}{\sum_{k} P(E_k|C) \sum_{\nu} P(E_i|C_{\nu}) P(C_{\nu})}.$$

(6)

The response matrix $P(E_i|C)$ is generated via simulation, and the $n(E)$ provided through measurement, apparently bestowing the freedom to choose the form of $P(C)$.

---

1 Except for C and E, all variables and subscripts related to causes are Greek letters, while Latin letters are used for effects. The only superscript is the iteration number, i.
represents the total of our prior knowledge of the parent distribution. Typically an experimenter refrains from introducing bias in the prior, and two such appropriate choices are the uniform and non-informative Jeffreys’ [3] priors:

\[ P_{\text{Uniform}}(C_\mu) = \frac{1}{n_C} \]

\[ P_{\text{Jeffrey}}(C_\mu) = \frac{1}{\log(C_{\text{max}}/C_{\text{min}}) C_\mu}, \]

keeping in mind that these priors place the causes on equal footing, not that all parent cause distributions are equally probable.

We now possess all the necessary machinery to perform an unfolding. Having started with a conservative initial prior, the unfolded result is a Bayesian best estimate of the true distribution. There is nothing stopping us from using this result as the best knowledge estimate of \( P(C_\mu) \) in eq. 6 for a subsequent unfolding. We can take this any number of steps further, making the process an iterative unfolding. Thus, after calculating \( \phi(C_\mu) \) via eq. 5, we recalculate \( M_{\mu j} \) per eq. 6, returning again to eq. 5 for an updated \( \phi(C_\mu)' \). Since \( P(C_\mu) = \phi_\mu/N_{\text{True}} = \phi_\mu' \), where \( N_{\text{True}} \) is the estimated true number of cause events, we can make the change \( \phi(C_\mu) \rightarrow \phi_\mu \) in eq. 6.

Adding the iteration superscript and shortening the notation, this equates to

\[ M_{\mu j} = \frac{P_{\mu j} \phi^i_\mu}{\epsilon_\mu \sum_\rho P_{\rho j} \phi^i_\rho} \]

\[ \phi^{i+1}_\mu = \sum_j M_{\mu j} n_j. \]

The unfolding proceeds until a desired stopping criterion is satisfied, say by comparing subsequent iterations with a test statistic such as a \( \chi^2 \). The algorithm below outlines the basics to the iterative unfolding scheme:

**Algorithm 1 Unfolding Algorithm**

\[ \phi^0 \leftarrow \text{Prior} \]
\[ \text{testStatistic} \leftarrow \text{Pass} \]

\[ \textbf{while} \ ( \text{testStatistic} = \text{Pass} ) \ \textbf{do} \]
\[ \quad M \leftarrow M(P(E|C), \phi^i) \]
\[ \quad \phi^{i+1} \leftarrow M \times n \]
\[ \quad \text{testStatistic} \leftarrow \text{TS}(\phi^i, \phi^{i+1}) \]

\[ \textbf{end while} \]
3 Regularization

After each iteration, the resulting posterior distribution, \( P(C_\mu) \), is our new best guess of the (normalized) parent distribution. Using this best estimate as the prior for the next iteration, one can induce large fluctuations in neighboring \( C_\mu \) bins. It is here the equivalence of matrix inversion techniques and iterative unfolding is seen. After many iterations, wild fluctuations can appear, indicating the granularity in the MC derived \( P_\mu j \). Furthermore, in using the posterior as the subsequent prior, one is ‘telling’ the unfolding that physical distributions of that nature are allowable priors. Instead, as pointed out in [1] (section 6.3), for an experimenter interested in a particular model’s parameters, fitting all but the last posterior is equivalent to performing a maximum likelihood fit to the data.

As physical measurements are expected to be smooth (a safe assumption for energy spectra for example), one can regularize the \( \phi^i_\mu \). In principle one can choose any smoothing function. For the cosmic-ray energy spectrum for example, \( \phi^i_\mu \) can be simply fit to a power law or a spline as was done in [4], using the fitted function as the input prior for the next iteration. While this could be seen as a loss of information, it is important to remember that any improved prior distribution will enhance our estimation method, along with the prior expectation that our distribution is smooth.

The other possibility is to avoid regularization altogether and instead ensure that \( P_\mu j \) is smooth enough. The granularity of the cause and effect bins will dictate the degree of smoothness required to ensure non-fluctuating \( \phi^i \) solutions. The more widely used techniques for smoothing \( P_\mu j \) include kernel density estimation and penalized spline fitting routines.
4 Unfolding Uncertainties

To begin the excursion into the calculation of uncertainties, we first shorten the notation in accordance with footnote 1:

\[ P(E_i|C_\mu) = P_{\mu i} \quad \phi(C_\mu) = \phi_\mu \quad n(E_j) = n_j. \]

As outlined in [1] (section 4), the covariance matrix \( V = V(\phi, \phi') \) from statistical contributions has two components: \( V^{\text{Data}} \) from the counted measured effects distribution, and \( V^{\text{MC}} \) due to the limited MC statistics in \( P_{\mu j} \). This can be seen from considering the uncertainties from \( n_j \) and \( M_{\mu j} \) in eq. 5. Since \( \phi = M \times n = M(P(E|C)) \times n \), we can identify respectively the aforementioned error contributions as

\[ V^{\text{Total}} = V^{\text{Data}} + V^{\text{MC}} \]
\[ = \frac{\partial \phi}{\partial n} \text{Cov}(n, n') \frac{\partial \phi'}{\partial n} + \frac{\partial \phi}{\partial P} \text{Cov}(P, P') \frac{\partial \phi'}{\partial P}. \]

4.1 \( V^{\text{Data}} \)

D’Agostini argues that since the data sample \( n_j \) is a realization of a multinomial distribution, then

\[ V^{\text{Data}} = M \text{Cov}(n, n') M \]

where the \( \text{Cov}(n, n') \) is the covariance matrix of the measurements with respect to the estimated true number of events \( \sum_{\mu} \phi_\mu = N_{\text{true}} \):

\[ \text{Cov}(n_k, n_j) = \begin{cases} n_j(1 - \frac{n_j}{N_{\text{true}}}) & \text{if } k = j \\ -\frac{n_k n_j}{N_{\text{true}}} & \text{if } k \neq j \end{cases}. \] (8)

However, Adye ([5] section 5) demonstrates that this error estimation is only valid for the first iteration, as subsequent \( \phi' \) are not independent of \( n_j \). Indeed, we should re-write eq. 7 appropriately as

\[ V^{\text{Data}} = \frac{\partial \phi^{i+1}}{\partial n} \times \text{Cov}(n, n') \times \frac{\partial \phi^{i+1}}{\partial n}, \] (9)

with

\[ \frac{\partial \phi^{i+1}}{\partial n_j} = M_{\mu j} + \frac{\phi^{i+1}}{\phi_\mu} \frac{\partial \phi_\mu}{\partial n_j} - \sum_{\sigma, k} \frac{\epsilon_\sigma}{\phi_\sigma} M_{\mu k} M_{\sigma k} \frac{\partial \phi^{i}_\sigma}{\partial n_j} \]

where again the superscripts \( i \) and \( i + 1 \) refer to the iteration number. The full derivation of \( \frac{\partial \phi^{i+1}}{\partial n} \) (eq. 20) is found in section 4.4.2 below. In some cases it is safe to use the Poisson form of \( \text{Cov}(n, n') \):

\[ \text{Cov}(n_k, n_j) = n_k \delta_{kj}. \] (10)
4.2 $V^{MC}$

The contribution from $V^{MC}$, while well outlined in [1] and below, is quite a monster. If one simply implements the equation verbatim into code, the expected time for calculating all elements $\sim (\text{number of bins})^7$. Thus, here we present the form of $V^{MC}$, while in section 4.4.3 we show the explicit expansion and further contraction of indices towards a more reasonable, practical calculation.

D’Agostini identifies $V^{MC}$ via $\partial \over \partial M$ giving

$$V^{MC} = n \times \text{Cov}(M, M') \times n'.$$

(11)

Further expansion reveals

$$\text{Cov}(M_{\mu k}, M_{\lambda j}) = \sum_{\{\sigma r\},\{\sigma s\}} \frac{\partial M_{\mu k}}{\partial P_{\sigma r}} \frac{\partial M_{\lambda j}}{\partial P_{\sigma s}} \text{Cov}(P_{\sigma r}, P_{\sigma s}),$$

(12)

$$\frac{\partial M_{\mu k}}{\partial P_{\sigma r}} = M_{\mu k} \left[ \frac{\delta_{\mu r} \delta_{jk}}{\epsilon_{\sigma r}} - \frac{\delta_{\mu r} \delta_{jk} M_{\sigma k} \epsilon_{\sigma}}{P_{\sigma k}} \right],$$

(13)

$$\text{Cov}(P_{\sigma r}, P_{\sigma s}) = \begin{cases} \frac{1}{n_{\sigma r}} \ P_{\sigma r} \ (1 - P_{\sigma r}) & \text{if } r = s \\ -\frac{1}{n_{\sigma r}} \ P_{\sigma r} \ P_{\sigma s} & \text{if } r \neq s \end{cases}.$$ (14)

In the final expression, $\tilde{n}_\mu$ represents the number of simulated events which fell into the true cause bin $\mu$. If our simulation is weighted, we identify $\tilde{n}$ with the effective number of events $\tilde{n}_\mu = \left( \sum_j w_{\mu j} \right)^2$ for all $j$ events in bin $\mu$.

Once again, Adye ([6]) shows this is a first order estimate, only valid for the first iteration. Re-writing 11 with $\partial \over \partial P$,

$$V^{MC} = \frac{\partial \phi^{i+1}}{\partial P} \times \text{Cov}(P, P') \times \frac{\partial \phi^{i+1}}{\partial P},$$

(15)

we identify $\frac{\partial \phi^{i+1}}{\partial P}$ as

$$\frac{\partial \phi^{i+1}}{\partial P_{\lambda k}} = \frac{\delta_{\lambda k}}{\epsilon_{\mu}} \left( \frac{n_k \phi^{i+1}_{\mu}}{f_k} - \phi^{i+1}_{\lambda} \right) - \frac{n_k \phi^{i}_{\lambda}}{f_k} M_{\mu k}$$

$$+ \frac{\phi^{i+1}_{\mu}}{\phi^{i}_{\mu}} \frac{\partial \phi^{i}}{\partial P_{\lambda k}} - \sum_{\rho,j} n_j \frac{\epsilon_{\rho}}{\phi^{i}_{\rho}} M_{\rho j} M_{\rho k} \frac{\partial \phi^{i}_{\rho}}{\partial P_{\lambda k}},$$

whose derivation (eq. 21) is found in section 4.4.3 below. Of course, D’Agostini’s form of $\text{Cov}(P, P')$ remains valid for use with the new construction of the partials. One may also use a Poisson covariance if justified appropriately:

$$\text{Cov}(P_{\sigma r}, P_{\sigma s}) = \sigma_{pr} \sigma_{\lambda \rho} \delta_{\rho s},$$

(16)

with $\sigma_{pr}$ being the error estimates on $P_{\sigma r}$ estimated when filling $P$ with Monte Carlo.
Algorithm 2 Unfolding Algorithm - Including Errors

$$
\phi^0 \leftarrow Prior
\text{testStatistic} \leftarrow Pass
\text{while} \ (\text{testStatistic} = Pass) \ \text{do}
\quad M \leftarrow M(P(E|C), \phi^i)
\quad \phi^{i+1} \leftarrow M \times n
\quad \frac{\partial \phi^{i+1}}{\partial n} \leftarrow \text{eq. } 20
\quad \frac{\partial \phi^{i+1}}{\partial P} \leftarrow \text{eq. } 21
\quad \text{testStatistic} \leftarrow TS(\phi^i, \phi^{i+1})
\text{end while}
\quad V_{\text{Total}} \leftarrow V_{\text{Data}} \left( \frac{\partial \phi^{i+1}}{\partial n} \right) + V_{\text{MC}} \left( \frac{\partial \phi^{i+1}}{\partial P} \right)
\quad \sigma^2_i \approx \text{diag}(V_{\text{Total}})
$$

4.3 Updated Unfolding Algorithm

The afore-outlined unfolding algorithm must be modified to include the propagation of systematic errors. At each iteration we have $$\phi^{i+1}$$, so both $$\frac{\partial \phi^{i+1}}{\partial n}$$ and $$\frac{\partial \phi^{i+1}}{\partial P}$$ can be calculated. The results are propagated and saved until the full covariance matrix is required for error estimates on the final $$\phi$$.

4.4 Expansion of Components of $$V$$

4.4.1 Some useful formulae

Recalling the unfolding formulae from before,

$$
\phi^{i+1}_\mu = \sum_k M_{\mu k} n_k \quad M_{\mu j} = \frac{P_{\mu j}}{\epsilon_\mu f_j},
$$

where the efficiency, $$\epsilon$$, and normalization, $$f$$, of $$M$$ are

$$
\epsilon_\mu = \sum_j P_{\mu j} \quad f_j = \sum_\mu P_{\mu j} \phi^i_\mu.
$$

Of note is the presence of $$\phi^i$$, i.e., the unfolded cause distribution from the previous iteration, or the prior in the case $$i = 0$$.

We will be taking derivatives of these objects with respect to $$n_k$$ and $$P_{\lambda k}$$, to wit,

$$
\frac{\partial P_{\mu j}}{\partial n_k} = 0 \quad \frac{\partial \epsilon_\mu}{\partial n_k} = 0 \quad \frac{\partial f_j}{\partial n_k} = \sum_\mu P_{\mu j} \frac{\partial \phi^i_\mu}{\partial n_k} \quad (17)
\quad \frac{\partial P_{\mu j}}{\partial P_{\lambda k}} = \delta_{\mu \lambda} \delta_{jk} \quad \frac{\partial \epsilon_\mu}{\partial P_{\lambda k}} = \delta_{\lambda \mu} \quad \frac{\partial f_j}{\partial P_{\lambda k}} = \delta_{jk} \phi^i_\lambda + \sum_\mu P_{\mu j} \frac{\partial \phi^i_\mu}{\partial P_{\lambda k}} \quad (18)
$$

The explicit forms of $$\frac{\partial \phi^i_\mu}{\partial n_k}$$ and $$\frac{\partial \phi^i_\mu}{\partial P_{\lambda k}}$$ will be shown below, but only for $$i = 0$$ do

$$
\frac{\partial \phi^0_\mu}{\partial n_k} = 0, \quad \frac{\partial \phi^0_\mu}{\partial P_{\lambda k}} = 0, \quad (19)
$$
as no unfolding has been performed. This will clearly not be the case for subsequent iterations when \( \phi^i \) becomes dependent on \( n_k \) and \( P_{\lambda k} \).

### 4.4.2 Expansion of \( V^{Data} \)

Making the appropriate substitutions, the index representation of eq. 9 is

\[
V(\phi_{\mu}^{i+1}, \phi_{\nu}^{i+1})^{Data} = \sum_{jk} \frac{\partial \phi_{\mu}^{i+1}}{\partial n_j} \text{Cov}(n_j, n_k) \frac{\partial \phi_{\nu}^{i+1}}{\partial n_k},
\]

with

\[
\frac{\partial \phi_{\mu}^{i+1}}{\partial n_j} = \frac{\partial}{\partial n_j} \sum_k M_{\mu k} n_k
\]

\[
= \sum_k (M_{\mu k} \frac{\partial n_k}{\partial n_j} + n_k \frac{\partial M_{\mu k}}{\partial n_j})
\]

\[
= \sum_k (M_{\mu k} \delta_{jk} + n_k \frac{\partial M_{\mu k}}{\partial n_j})
\]

\[
= M_{\mu j} + \sum_k n_k \frac{\partial M_{\mu k}}{\partial n_j}
\]

\[
\frac{\partial M_{\mu k}}{\partial n_j} = \frac{\partial}{\partial n_j} \frac{P_{\mu k} \phi_{\mu}^i}{\epsilon_{\mu f_k}}
\]

\[
= \frac{P_{\mu k}}{\epsilon_{\mu f_k}} \frac{\partial \phi_{\mu}^i}{\partial n_j} - \frac{M_{\mu k}}{\epsilon_{\mu f_k}} \frac{f_k}{n_{\mu k}} \sum_{\sigma} P_{\sigma k} \frac{\partial \phi_{\sigma}^i}{\partial n_j}
\]

\[
= \frac{M_{\mu k}}{\phi_{\mu}^i} \frac{\partial \phi_{\mu}^i}{\partial n_j} - M_{\mu k} \sum_{\sigma} \frac{\epsilon_{\sigma f_k}}{\epsilon_{\mu f_k}} \frac{n_{\mu k}}{n_{\sigma}} \frac{\partial \phi_{\sigma}^i}{\partial n_j}
\]

\[
= \frac{M_{\mu k}}{\phi_{\mu}^i} \frac{\partial \phi_{\mu}^i}{\partial n_j} - \sum_{\sigma} \epsilon_{\sigma} \phi_{\sigma}^i M_{\mu k} M_{\sigma k} \frac{\partial \phi_{\sigma}^i}{\partial n_j}
\]

\[
\frac{\partial \phi_{\mu}^{i+1}}{\partial n_j} = M_{\mu j} + \frac{1}{\phi_{\mu}^i} \frac{\partial \phi_{\mu}^i}{\partial n_j} \sum_k M_{\mu k} n_k - \sum_{\sigma, k} \frac{\epsilon_{\sigma}}{\phi_{\sigma}^i} M_{\mu k} M_{\sigma k} \frac{\partial \phi_{\sigma}^i}{\partial n_j}
\]

\[
\frac{\partial \phi_{\mu}^{i+1}}{\partial n_j} = M_{\mu j} + \phi_{\mu}^{i+1} \frac{\partial \phi_{\mu}^i}{\partial n_j} \sum_{\sigma, k} \frac{\epsilon_{\sigma}}{\phi_{\sigma}^i} M_{\mu k} M_{\sigma k} \frac{\partial \phi_{\sigma}^i}{\partial n_j}
\]

(20)
Recalling eq. 19, \( \frac{\partial \phi_0^i}{\partial n_j} = 0 \) for the first iteration, eliminating the last two terms of eq. 20 and recovering \( \frac{\partial \phi_1^i}{\partial n_j} = M_{i j} \) as per [1]. In practice, one need only calculate \( \frac{\partial \phi_{i+1}^i}{\partial n_j} \) for each iteration, saving the result until the full calculation of \( V^{\text{Data}} \) is required.

### 4.4.3 Expansion of \( V^{MC} \)

Similar to \( V(\phi_{i+1}^i, \phi_{i+1}^j)^{\text{Data}} \), we identify the contributions to \( V \) from the Monte Carlo:

\[
V(\phi_{i+1}^i, \phi_{i+1}^j)^{MC} = \sum_{\lambda j} \sum_{pk} \frac{\partial \phi_{i+1}^i}{\partial P_{\lambda j}} \text{Cov}(P_{\lambda j}, P_{pk}) \frac{\partial \phi_{i+1}^j}{\partial P_{pk}}.
\]

Proceeding forward,

\[
\frac{\partial M_{i j}}{\partial P_{\lambda k}} = \frac{\partial}{\partial P_{\lambda k}} \sum_{j} M_{i j} n_j = \sum_{j} n_j \frac{\partial M_{i j}}{\partial P_{\lambda k}}.
\]

\[
\frac{\partial M_{i j}}{\partial P_{\lambda k}} = \frac{\partial}{\partial P_{\lambda k}} \left( \frac{P_{i j}}{\epsilon_{\mu} f_j} \frac{\phi_{i j}^i}{\phi_{i j}^i} \right)
= \frac{\phi_{i j}^i}{\epsilon_{\mu} f_j} \delta_{\lambda k} \delta_{\mu} + \frac{P_{i j}}{\phi_{i j}^i} \frac{\partial M_{i j}}{\partial P_{\lambda k}} - \frac{1}{\epsilon_{\mu} f_j} \frac{P_{i j}}{\phi_{i j}^i} \left( f_j \frac{\partial \epsilon_{\lambda}}{\partial P_{\lambda k}} + \epsilon_{\lambda} \frac{\partial f_j}{\partial P_{\lambda k}} \right)
= \frac{\phi_{i j}^i}{\epsilon_{\mu} f_j} \delta_{\lambda k} \delta_{\mu} + \frac{M_{i j}}{\phi_{i j}^i} \frac{\partial M_{i j}}{\partial P_{\lambda k}} - \frac{1}{\epsilon_{\mu} f_j} M_{i j} \left( f_j \frac{\partial \epsilon_{\lambda}}{\partial P_{\lambda k}} + \epsilon_{\lambda} \frac{\partial f_j}{\partial P_{\lambda k}} \right) - \sum_{\rho} \frac{P_{i j}}{\epsilon_{\rho} f_j} \frac{\partial \phi_{i j}^i}{\partial P_{\lambda k}}
= \frac{\phi_{i j}^i}{\epsilon_{\mu} f_j} \delta_{\lambda k} \delta_{\mu} + \frac{M_{i j}}{\phi_{i j}^i} \frac{\partial M_{i j}}{\partial P_{\lambda k}} - \frac{1}{\epsilon_{\mu} f_j} M_{i j} \left( f_j \frac{\partial \epsilon_{\lambda}}{\partial P_{\lambda k}} + \epsilon_{\lambda} \frac{\partial f_j}{\partial P_{\lambda k}} \right) - \sum_{\rho} \frac{M_{i j}}{\phi_{i j}^i} \frac{\partial \phi_{i j}^i}{\partial P_{\lambda k}}
\]

and going back to \( \frac{\partial \phi_{i+1}^i}{\partial P_{\lambda k}} \) to include the sum over \( j \),

\[
\frac{\partial \phi_{i+1}^i}{\partial P_{\lambda k}} =
\]
\[
\sum_j n_j \left[ \frac{\phi^i_\mu}{\epsilon_\mu f_j} \delta_{\lambda\mu} \delta_{jk} + \frac{M_{\mu j}}{\phi^i_\mu} \partial P_{\lambda k} \right] - \frac{M_{\mu j}}{\epsilon_\mu} \partial_\mu \delta_{\lambda j} - \sum_\rho \frac{M_{\mu j} M_{\mu j}}{\phi^i_\mu} \partial P_{\lambda k}^{-1} \delta_{\lambda j} = \frac{n_k \phi^i_\mu}{\epsilon_\mu f_k} \delta_{\lambda\mu} + \frac{1}{\phi^i_\mu} \partial P_{\lambda k} \sum_j M_{\mu j} n_j - \frac{\delta_{\lambda\mu}}{\epsilon_\mu} \sum_j M_{\mu j} n_j - \frac{n_k M_{\mu k} \phi^i_\lambda}{f_k} \\
- \sum_j \sum_\rho n_j \epsilon_\rho \phi^i_\rho M_{\rho j} \partial_\rho \phi^i_\mu \partial P_{\lambda k},
\]

with final form

\[
\frac{\partial \phi^i_{\mu+1}}{\partial P_{\lambda k}} = \frac{\delta_{\lambda\mu}}{\epsilon_\mu} \left( \frac{n_k \phi^i_\mu}{f_k} - \phi^i_{\mu+1} \right) - \frac{n_k \phi^i_\lambda}{f_k} M_{\mu k} + \frac{\phi^i_{\mu+1}}{\phi^i_\mu} \frac{\partial \phi^i_\mu}{\partial P_{\lambda k}} - \sum_\rho n_j \epsilon_\rho \phi^i_\rho M_{\rho j} M_{\mu j} \frac{\partial \phi^i_\rho}{\partial P_{\lambda k}}. \tag{21}
\]

Again for the first iteration \(\frac{\partial \phi^0_\mu}{\partial P_{\lambda k}} = 0\), eliminating the last two terms of eq. 21, and recovering D'Agostini’s version. Again, upon implementation one need only calculate \(\frac{\partial \phi^i_{\mu+1}}{\partial P_{\lambda k}}\) at each iteration, saving it until \(V^{MC}\) is needed for error estimation.
References

[1] G. D’Agostini, “A Multidimensional unfolding method based on Bayes’ theorem”, Nucl. Instrum. Meth. A 362 (1995) 487

[2] Z. Hampel-Arias, “Cosmic Ray Observations at the TeV Scale with the HAWC Observatory”, Ph.D. thesis, University of Wisconsin – Madison (2017)

[3] Jeffreys, H., “An Invariant Form for the Prior Probability in Estimation Problems”. Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences. 186 (1007): 453461

[4] Alfaro, R., et al. “All-particle cosmic ray energy spectrum measured by the HAWC experiment from 10 to 500 TeV”, Phys. Rev. D 96, (2017) 122001, doi:10.1103/PhysRevD.96.122001

[5] T. Adye, “Unfolding algorithms and tests using RooUnfold”, Proceedings of the PHYSTAT 2011 Workshop, CERN, Geneva, Switzerland, January 2011, CERN-2011-006, pp 313-318, arXiv:1105.1160

[6] T. Adye, “Corrected error calculation for iterative Bayesian unfolding”, Personal Website