# Inverse Problems in Experimental Particle Physics

Sean Gilligan

**Abstract**

This report presents a survey of the common methods of solving ill-posed inverse problem in experimental high energy physics.

## 1 Introduction

A common problem faced in the quantitative sciences and their associated technologies is the introduction of errors during the data collection process. While the possible sources of these errors are as varied as the possible events which the data might describe, significant work has been done to develop methods the can help would-be analysts reconcile them. The requisite understanding of a scenario's underlying systematic and stochastic processes might not allow researchers to truly reverse entropy, but it can approximate it with a quantifiable degree of certainty. The applied mathematics that this involves falls within the general category of *inverse problems*, and there are a variety of labels used to refer to the procedures in its arsenal. Within the applications described here there is the colloquially vague *unsmearing*, but there are also names that reference specific applications and methods, such as those characterized in this report.

### 1.1 The Deconvolution

One way to characterize a basic example would be the following. Assume that data collected regarding $n$ statistical events represent the measurement of $n$ independent and identically distributed (i.i.d.) random variables $\boldsymbol{X} = \{X_1, X_2, \ldots, X_n\}$ from a distribution of possible values represented by the probability density function (PDF) $f_X(x)$, such that the probability of a random variable $X_i$ having a value between $x_a$ and $x_b$ is

$$P(\, x_a < X_i < x_b \,) = \int_{x_a}^{x_b} f(x)\, dx$$

and

$$\int_{\mathcal{X}} f(x)\, dx = 1,$$

where $\mathcal{X}$ represents the domain of $x$. The error introduced during the measurement process is similarly represented by a set of i.i.d. random variables $\boldsymbol{\varepsilon} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n\}$ with a PDF $f_\varepsilon(\varepsilon)$, where the sets $\boldsymbol{\varepsilon}$ and $\boldsymbol{X}$ are typically assumed to be independent of each other. The set of measured values $\boldsymbol{Y} = \{Y_1, Y_2, \ldots, Y_n\}$ then are also i.i.d. and can be defined in terms of the preceding sets of variables such that for event $i \in \{1, \ldots, n\}$,

$$\begin{aligned} Y_i &= g(X_i, \varepsilon_i) \\ &= X_i + \varepsilon_i. \end{aligned} \tag{1}$$

In light of this relationship, the corresponding PDF $f_Y(y)$ can be found explicitly through an operation on $f_X(x)$ and $f_\varepsilon(\varepsilon)$ using the mathematics of functional analysis. Stated in more general terms, the empirical density function $f_Y$ is formed from the *convolution* of the true density function $f_X$ and the error density

function $f_\varepsilon$, and is defined by [8]

$$f_Y \equiv f_X * f_\varepsilon \tag{2}$$

$$\begin{aligned} f_Y(y) &\equiv \int_{\mathcal{X}} f_X(x) f_\varepsilon(\varepsilon)\, dx \\ &= \int_{\mathcal{X}} f_X(x) f_\varepsilon\big(g_x^{-1}(y)\big) \left| J_{g_x^{-1}}(y) \right| dx \\ &= \int_{\mathcal{X}} f_X(x) f_\varepsilon(y-x)\, dx, \end{aligned} \tag{3}$$

where $J$ represents the Jacobian of the transformation involved in performing the change of basis on $f_\varepsilon$ from $\varepsilon$ to $x$, which is necessary for the evaluation of the integral for a given $y$. The magnitude of the Jacobian for transformation of $\varepsilon$ to $y - x$ through the manipulation of Equation (1) happens to be 1.

As the collection of measured values $\boldsymbol{Y}$ accumulates an estimate of empirical density $\hat{f}_Y$ can readily be formed. However, a major goal in an analysis of data like this is typically to develop an accurate estimate of the true density $\hat{f}_X$. Using the information contained in $\hat{f}_Y$ to accomplish this necessarily requires some attempt at finding an inverse process to the *convolution*, i.e. the *deconvolution*.

For cases in the form of this particular example there are a variety approaches, but they commonly involve the Fourier transform of the density functions $\{f_X, f_\varepsilon, f_Y\}$ into their corresponding characteristic functions $\{\phi_X, \phi_\varepsilon, \phi_Y\}$ [7][8]. The definition of the Fourier transform is subject to a number of conventions based on its application, but here it can be defined for some random variable $U$ with density function $f_U(u)$ as

$$\phi_U(t) = \int f_U(u)\, e^{itu}\, du, \quad \forall\, u \in \mathbb{R}. \tag{4}$$

When conditions permit the inverse Fourier transform can be found via

$$f_U(u) = \int \phi_U(t)\, e^{-itu}\, dt, \quad \forall\, t \in \mathbb{R}. \tag{5}$$

The Fourier transform is important in deconvolution methods because when you apply it to the convolution of two density functions the link between their respective characteristic functions becomes purely multiplicative, i.e.

$$f_Y = f_X * f_\varepsilon \implies \phi_Y = \phi_X \phi_\varepsilon.$$

## 1.2 Generalizing

The remainder of this paper is dedicated to characterizing the major methods adopted by the High Energy Physics (HEP) community toward solving their inverse problems. While most literature on deconvolution methods do use the word *convolution*, this operation is also referred to by the German word *faltung* [10]. The latter's English translation, *folding*, is featured prominently in the particle physics community, but refers to a more generalized process than what is described by Equation (3) [4][1][2]. In general, the terms *folding* and *unfolding* are used to describe two supersets of processes that respectively include *convolution* and *deconvolution*.

One way to arrive at the intended generalization is by considering conditional probability. The mechanisms governing the migration of $f_X$ to $f_Y$ can be understood as the conditional probability $f_{Y|X}(y|x)$.

$$\begin{aligned} f_Y(y) &= \int_{\mathcal{X}} f_{Y|X}(y|x) f_X(x)\, dx \\ &= \int_{\mathcal{X}} K(x,y) f_X(x)\, dx \end{aligned} \tag{6}$$

With that said, when considering the inverse process it cannot be assumed that all contributions to the measured values are coming from the same source as the signal described by $f_X$. A background source background density function $b_Y$ becomes a contributing

# 2 Unfolding methods in particle physics

## 2.1 Bin-by-bin

$$\begin{aligned}\hat{x}_i &= C_i(n_i - \beta_i) \\ &= C_i y_i\end{aligned} \tag{7}$$

$$C_i = \frac{x_i^{\mathrm{MC}}}{y_i^{\mathrm{MC}}} \tag{8}$$

$$\begin{aligned}E_i[\hat{x}_i] &= C_i E[n_i - \beta_i] \\ &= \frac{x_i^{\mathrm{MC}}}{y_i^{\mathrm{MC}}} y_i \qquad\qquad =\end{aligned}$$

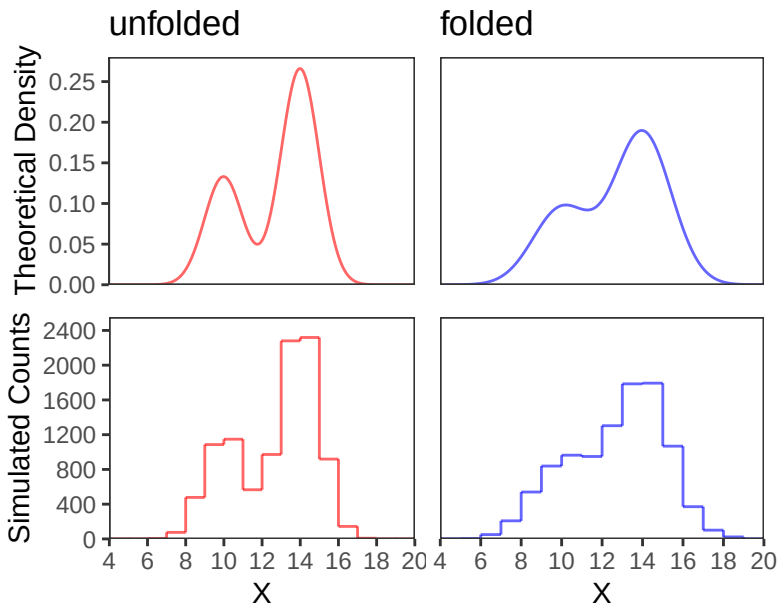


Figure 1: *The unfolded continuous distribution is of the form* $X_i^{unfolded} = Z_i Y_{2,i} + (1 - Z_i) Y_{1,i}$ *where* $Y_{1,i} \sim N(10, 1)$, $Y_{2,i} \sim N(14, 1)$, *and* $Z_i \sim$ *Bernoulli*$(1/3)$. *The folded continuous distribution is attained with the addition of simple Gaussian noise, such that* $X_i^{folded} = X_i^{unfolded} + e_i$, *where* $e_i \sim N(0, 1)$. *The binned distributions consist of 10,000 simulated events from these distributions.*

While many data collection methods record or reconstruct variables of interest regarding the events they observe that have continuous values, discretization of this data into bins is routinely used to form estimates of these variables' distributions, with bin widths being chosen based on the needs of the researchers as well as the needs of the statistics. When combined with folding processes, binning can readily obscure otherwise prominent features in an underlying true distribution. An example of this coarse-grained look at reality afforded to us by the imperfect nature of our data collection methods can be seen in Figure [1], in which 10,000 random samples are drawn from a simple theoretical distribution and experience folding via a comparably variant Gaussian noise process.

The Bayesian methods described here were first proposed in the context of high energy physics in 1994 [4] by Giulio D'Agostini, an Italian physicist and strong proponent of Bayesian reasoning in statistics. Later improvements to this method by him in 2010 [5] have lead to this iterative Bayesian approach becoming, along with methods based off Tikhonov regularization [6][9], one of the major unfolding methods used in high energy physics today. To ease any potential reviewing of the reference material I will be adopting much of D'Agostini's notation and conventions, while also attempting to provide a more concise and clear description of his described methods.

# 3 Algorithm Construction

Unfolding is ultimately concerned with finding a reliable inverse to the process by which an event occurring in some true bin $i$ maps to an observed bin $j$. In generalizing this a bit we can think in terms of *causes*, $C_i$ $(i = 1, \ldots, n_C)$ and *effects*, $E_j$ $(j = 1, \ldots, n_E)$, representing the true and observed bins respectively. In regard to a single event we are then interested in conditional probabilistic view for causation, $P(C_i|E_j, I) \equiv \theta_{ij}$, the probability that we can attribute some cause $C_i$ to an observed effect $E_j$. Using Bayes' theorem we can define this in terms of other probabilities that can be estimated more directly,

$$\begin{aligned} P(C_i|E_j, I) &= \frac{P(E_j|C_i, I) \cdot P(C_i|I)}{\sum_{i=1}^{n_C} P(E_j|C_i, I) \cdot P(C_i|I)} \\ \theta_{ij} &= \frac{\lambda_{ji} \cdot P(C_i|I)}{\sum_{i=1}^{n_C} \lambda_{ji} \cdot P(C_i|I)}, \end{aligned} \tag{9}$$

where the conditional probability regarding inference (effect), $P(E_j|C_i, I) \equiv \lambda_{ji}$, is the probability that some effect $E_j$ will result with some cause $C_i$ and $P_o(C_i|I)$ is the true probability of an event occurring from cause $C_i$. The term $I$ represents any implicit conditional information regarding the analysis, such as the choice of prior, and is usually apparent when the probabilities are written out as density functions.

At the analysis level we care less about individual events and more about mapping the total number events per effect, $\boldsymbol{x}_E = \{x(E_1), \ldots, x(E_{n_E})\}$, to the total number of events per cause, $\boldsymbol{x}_C = \{x(C_1), \ldots, x(C_{n_C})\}$. However, so far this regards only observed events as categorized into the $n_E$ effects, as we cannot expect to observe or select for all effects resulting from some arbitrary cause $C_i$. In light of this, while we can add causes to account for any independent background sources to assume the normalization of $P_o(C_i|E_j, I)$ and $P_o(C_i|I)$, such that $\sum_{i=1}^{n_C} P_o(C_i|E_j, I) = 1$ and $\sum_{i=1}^{n_C} P(C_i|I) = 1$, we cannot say the same for $P(E_j|C_i, I)$. A necessarily imperfect effect selection capability results in

$$0 \leq \sum_{j=1}^{n_E} P(E_j|C_i, I) = \sum_{j=1}^{n_E} \lambda_{ji} \equiv \epsilon_i \leq 1,$$

the exact value of which provides for us a definition for $\epsilon_i$, the *efficiency* at which we detect cause $C_i$ from all accounted for observed effects, being also defined and useable as the ratio of observed events resulting from cause $C_i$ to the true number of events resulting from $C_i$, $x^{obs}(C_i)/x(C_i)$.

A visualization of these lost events can be seen in Figure [2], where some collection of undocumented effects resulting from our collection of causes are lumped into a composite effect $E_{n_E+1}$, which should relate to our efficiency regarding cause $C_i$ by $P(E_{n_E+1}|C_i, I) = \lambda_{n_E+1,i} = 1 - \epsilon_i$. Including this new effect with the others results in $\sum_{j=1}^{n_E} \lambda_{ji} = 1$, creating normalized basis vectors to define the columns of a *smearing matrix* $\Lambda$,

$$\begin{aligned} \Lambda &= \begin{pmatrix} P(E_1|C_1, I) & P(E_1|C_2, I) & \ldots & P(E_1|C_{n_C}, I) \\ P(E_2|C_1, I) & P(E_2|C_2, I) & \cdots & P(E_2|C_{n_C}, I) \\ \vdots & \vdots & \ddots & \vdots \\ P(E_{n_E}|C_1, I) & P(E_{n_E}|C_2, I) & \ldots & P(E_{n_E}|C_{n_C}, I) \\ P(E_{n_E+1}|C_1, I) & P(E_{n_E+1}|C_2, I) & \ldots & P(E_{n_E+1}|C_{n_C}, I) \end{pmatrix} \\ &= (\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots, \boldsymbol{\lambda}_{n_c}), \end{aligned} \tag{10}$$

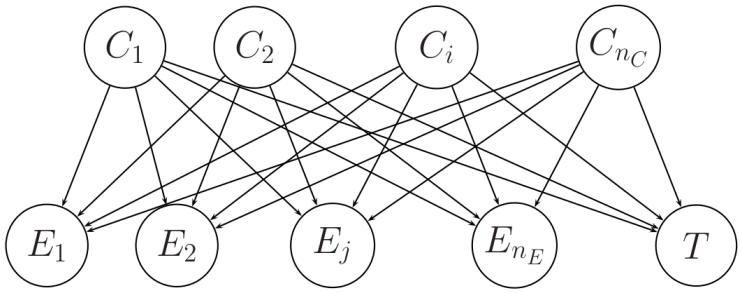


Figure 2: *Thinking of causes and effects as distinct subsets within one or more dimensional cause and effect phase spaces, this figure shows how events are probabilistically mapped from the subsets used to define our causes to the subsets used to define our effects. The node indicated by $T$ ('trash') represents the event selection inefficiency, and can be thought of as an additional effect $E_{n_E+1}$ that contains an unobserved number of events. Unlike the possibility of alloting independent sources of background to different causes, $E_{n_E+1}$ $(T)$ can consist of any number of potentially distinguishable effects depending on how the lost events are distributed across the complement of $\cup(E_1, E_2, \ldots, E_{n_E})$ in our effect phase space. Figure source:* [5].

where $\boldsymbol{\lambda}_i$ refers to the $i$-th column consisting of $\{\lambda_{1,i}, \lambda_{2,i}, \ldots, \lambda_{n_E+1,i}\}$.

Now that Eq. (2) accounts for lost events we can begin to construct a conditional probability for $\boldsymbol{x}_C$ similar to that for Eq. (1) using Bayes' theorem,

$$P(\boldsymbol{x}_C|\boldsymbol{x}_E, \Lambda, I) \propto P(\boldsymbol{x}_E|\boldsymbol{x}_C, \Lambda, I) \cdot P(\boldsymbol{x}_C|I), \tag{11}$$

and account for uncertainties in $\Lambda$ with

$$P(\boldsymbol{x}_C|\boldsymbol{x}_E, I) = \int P(\boldsymbol{x}_C|\boldsymbol{x}_E, \Lambda, I)\ f(\Lambda|I)\ \mathrm{d}\Lambda.$$

At this point one should recognize in Eq. (3) $P(\boldsymbol{x}_E|\boldsymbol{x}_C, \Lambda, I)$ as the likelihood and $P(\boldsymbol{x}_C|I)$ as the prior in the formal calculation of a posterior. Ignoring the prior for now and focusing on the likelihood, for a given cause $C_i$ we can model this expression as a multinomial distribution such that

$$P(\boldsymbol{x}_E|x(C_i), \Lambda, I) \quad = \quad \frac{x(C_i)!}{\prod_j^{n_E+1} x(E_j)!} \prod_j^{n_E+1} \lambda_{ji}^{x(E_j)}, \tag{12}$$

which leads us finally to asking how we should estimate our $\lambda_{ji}$'s. Once again, using Bayes' theorem we can see that for a fixed $i$,

$$f(\boldsymbol{\lambda}_i|\boldsymbol{x}_E, x(C_i), I) \quad \propto \quad P(\boldsymbol{x}_E|x(C_i), \boldsymbol{\lambda}_i, I) \cdot f(\boldsymbol{\lambda}_i|I). \tag{13}$$

Previously mentioned properties about $\boldsymbol{\lambda}_i$ make a Dirichlet($\boldsymbol{\alpha}_{prior_i}$) prior appropriate. A flat prior in which $\boldsymbol{\alpha}_{prior_i} = \{1, \ldots, 1\}$ is often chosen, and is done so here. Regardless of the choice for $\boldsymbol{\alpha}_{prior_i}$, multiplying by a multinomial distribution results in the posterior

$$\begin{aligned} f(\boldsymbol{\lambda}_i|\boldsymbol{x}_E, x(C_i), I) \quad &\propto \quad \left[\frac{x(C_i)!}{\prod_j^{n_E+1} x(E_j)!} \prod_j^{n_E+1} \lambda_{ji}^{x(E_j)}\right] \cdot \left[\frac{1}{\mathrm{B}(\boldsymbol{\alpha}_{prior_i})} \prod_{j=1}^{n_E+1} \lambda_{ji}^{\alpha_{prior_{ji}}-1}\right] \\ &\propto \quad \prod_j^{n_E+1} \lambda_{ji}^{\left(\alpha_{prior_{ji}}+x(E_j)\right)-1} \\ &= \quad \mathrm{Dirichlet}(\boldsymbol{\alpha}_{prior_i} + \boldsymbol{x}_E). \end{aligned} \tag{14}$$

Samples from this distribution informs much of the the uncertainty resulting from the unfolding process, creating distributions for objects fully or partially calculated from them, such as the smearing matrix, efficiency, and inverse probabilities $\theta_{ij}$ once a prior for $P(C_i|I)$ is made. These will be necessary, as $P(\boldsymbol{x}_C|\boldsymbol{x}_E, I)$

becomes the sum of independent multinomial distributions, which does not have a closed solution that we can analytically maximize the likelihood of. We have to make the rest of our progress starting from Eq. (1) where the choice around a prior for $P(C_i|I)$ is due. The choice of $P(C_i|I) = constant$ is considered here, which D'Agostini acknowledges is a strong prior that produces biases that will require iterations to be resolved.

Defining $\boldsymbol{\theta}_j = \{\theta_{1,j}, \theta_{2,j}, \ldots, \theta_{n_C,j}\}$, we can model how the $x(E_j)$ observed events with the effect $E_j$ are likely distributed from potential causes by the multinomial distribution

$$\boldsymbol{x}_C|_{x(E_j)} \sim \text{Mult}(x(E_j), \boldsymbol{\theta}_j).$$

Before summing over the effects to get the total observed causes we should acknowledge that each $x(E_j)$ is the result of a Poisson process with an unknown rate parameter $\mu_j$. Using the conjugate prior $\mu_j \sim \text{Gamma}(c_j, r_j)$, with $c_j = 1$ and very small $r_j$ to create a flat prior, we arrive at

$$\mu_j|_{x(E_j)} \sim \text{Gamma}(c_j + x(E_j), r_j + 1),$$

which tells us not to use $x(E_j)$, but $\mu_j$. In dealing with fractional values of $\mu_j$ D'Agostini suggests:

1. Rounding $\mu_j$ to its nearest positive integer $m_j$,
2. Sampling from $\boldsymbol{x}_C|_{m_j} \sim \text{Mult}(m_j, \boldsymbol{\theta}_j)$,
3. Rescaling by $\boldsymbol{x}_C|_{\mu_j} = \frac{\mu_j}{m_j}\boldsymbol{x}_C|_{m_j}$,
4. Summing over each effect with $\boldsymbol{x}_C|_{\boldsymbol{x}_E} = \sum_{j=1}^{n_E} \boldsymbol{x}_C|_{\mu_j}$,
5. And applying the inefficiency correction with $\boldsymbol{x}_C = \frac{\boldsymbol{x}_C|_{\boldsymbol{x}_E}}{\epsilon_i}$.

The drawing of multiple samples from the posteriors is used to form an ensemble of values of $\boldsymbol{x}_C$ and estimate credible intervals. The performance of second and later iterations is accomplished by using the previous iteration's posteriors as the new iteration's priors.

# 4 A Basic Example

In the following example 100,000 random samples are drawn from a Cauchy distribution and then subject to some processes that disperse, bias, and reduce event selection efficiency. The results of these simulations are shown in Figure [3]. The migration matrix does a decent job of demonstrating how the true data was smeared. For unaffected data one would see just a diagonal line from the bottom left to the top right. It was with this in mind that instead of choosing the earlier mentioned flat prior for $\boldsymbol{\alpha}_{prior_i}$ I decided to go with

$$\alpha_{ij} = e^{-|x_{truth} - x_{smeared}|},$$

the values of which are represented in Figure [4].

# 5 Discussion of Results and Conclusion

The results of my unfolding are shown below in Figure [5]. The iterations get off to an okay start, but instead of converging they begin to behave erratically. The tails blowing up clearly comes from using random samples from a posterior distribution as basis for new priors, embedding events in places where there were zero before. I attempted to remedy this with a prior that disfavored events occurring far from the diagonal, as mentioned previously in the context of Figure [4]. It kept these tails from blowing up for at least the first iteration. In the future I will look more into options relating to this issue.

There is almost certainly an error in the code governing the 3rd iteration, resulting in a vastly larger confidence band. Instead of trying to fix this issue I think it would be better use of my time to study some similar working examples [3].

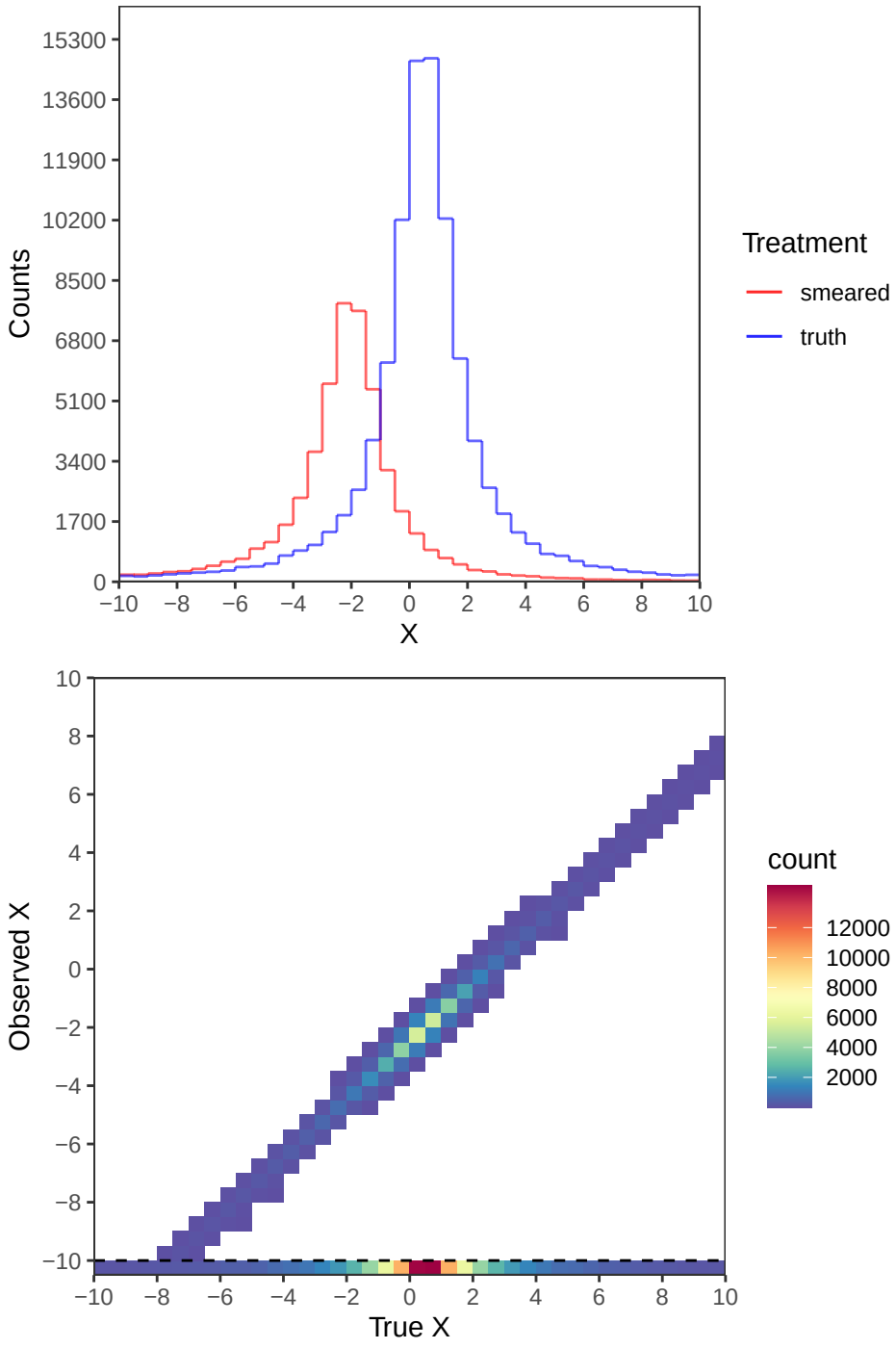


Figure 3: *TOP: The true and smeared distribution of our toy model. BOTTOM: When considering count data, the smearing matrix is often referred to as the response or migration matrix. The bottom row corresponds to the events that were not assigned to an effect.*

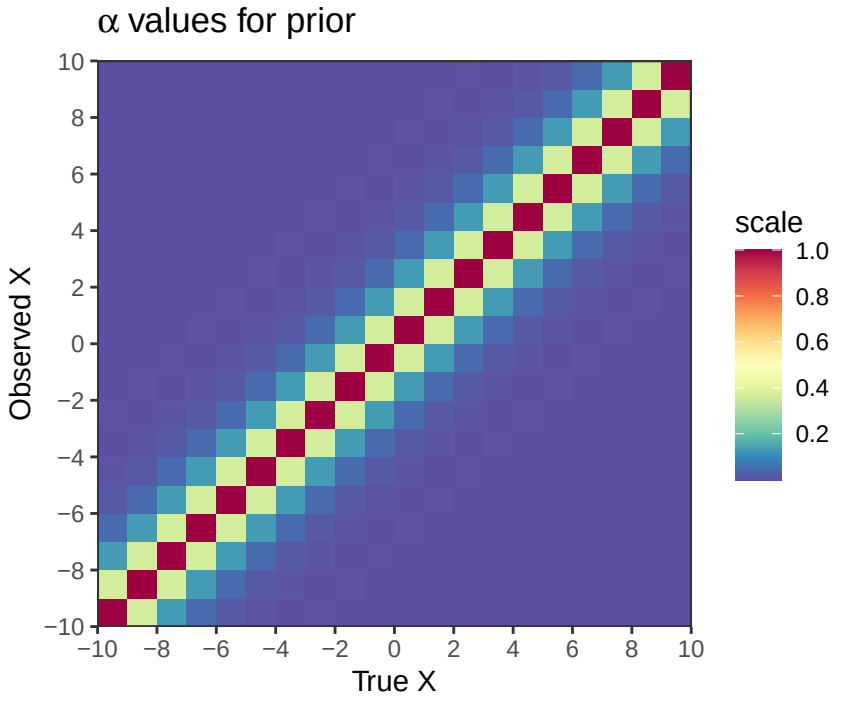


Figure 4: *Instead of choosing a flat prior Dirichlet($\boldsymbol{\alpha}_{prior_i}$) in which $\boldsymbol{\alpha}_{prior_i} = \{1, \ldots, 1\}$, I opted for one that assumed a dominant diagonal signal, corresponding to a max value of alpha along the diagonal that decays exponentially the further you get from the diagonal.*

# 6 Code

The libraries I used are shown here

```
library(tidyverse)
library(gridExtra)
library(knitr)
library(RColorBrewer)
library(gtools)
```

## 6.1 Code used for Figure [1]

```
# Assume 10000 events
nsim <- 10000

# Two possible types of processes with different means but equal variances
mu1 <- 10
mu2 <- 14
sd <- 1

# Event 1 is 1/3 as likely to occur as event 2
p <- 1/3

y1 <- rnorm(nsim, mean = mu1, sd = sd)
y2 <- rnorm(nsim, mean = mu2, sd = sd)
```

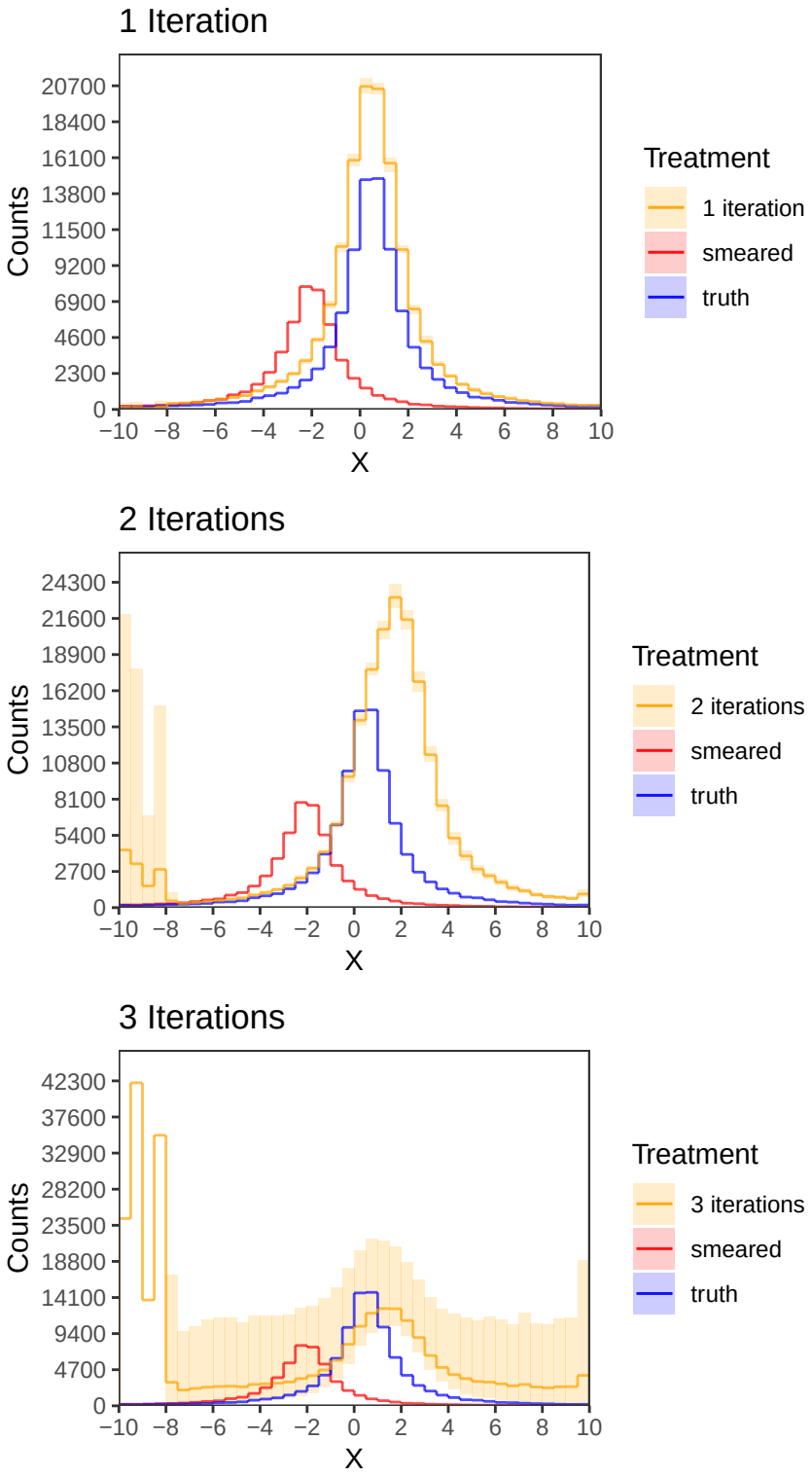


Figure 5: *The first iteration looks like itd heading the right direction. It is my belief that I have an error somehwere in my work.*

```
z <- rbernoulli(nsim, p)
x <- z*y1 + (1-z)*y2

# Add additional noise
xf <- x + rnorm(nsim, mean = 0, sd = sd)

xmin <- (floor(min(c(x,xf))) %/% 2)*2
xmax <- ceiling(max(c(x,xf)))
xmax <- (xmax/2 - xmax %/% 2)*2 + xmax

step_hist_unfolded <- hist(x, breaks = seq(xmin,xmax,1), plot = F)
step_hist_folded <- hist(xf, breaks = seq(xmin,xmax,1), plot = F)

ymax <- 1.075*max(c(step_hist_folded$density,step_hist_unfolded$density))
ymax_count <- 1.1*max(c(step_hist_folded$count,step_hist_unfolded$count))

step_hist <-
  tibble(binLow = c(step_hist_folded$breaks[-length(step_hist_folded$breaks)],
                    step_hist_unfolded$breaks[-length(step_hist_unfolded$breaks)]),
         binHigh = c(step_hist_folded$breaks[-1],step_hist_unfolded$breaks[-1]),
         CountsL = c(c(0,step_hist_folded$counts[-length(step_hist_folded$counts)]),
                     c(0,step_hist_unfolded$counts[-length(step_hist_unfolded$counts)])),
         Counts = c(step_hist_folded$counts,step_hist_unfolded$counts),
         DensityL = c(c(0,step_hist_folded$density[-length(step_hist_folded$density)]),
                      c(0,step_hist_unfolded$density[-length(step_hist_unfolded$density)])),
         Density = c(step_hist_folded$density,step_hist_unfolded$density),
         Treatment = c(rep("folded", length(step_hist_folded$density)),
                       rep("unfolded", length(step_hist_unfolded$density))))


# Continuous representation
X <- seq(xmin,xmax,0.01)
nx <- length(X)

unfold_fig_data <- tibble(x = X,
                          y_unfolded = (1/3)*dnorm(X, mean=mu1, sd=sd) +
                            (2/3)*dnorm(X, mean=mu2, sd=sd),
                          y_folded = (1/3)*dnorm(X, mean=mu1, sd=sqrt(sd+sd)) +
                            (2/3)*dnorm(X, mean=mu2, sd=sqrt(sd+sd)))

unfold_fig_data <- tibble(X = rep(seq(xmin,xmax,0.01),2),
                          Density = c(unfold_fig_data$y_unfolded,unfold_fig_data$y_folded),
                          Treatment = rep(c("unfolded","folded"), each = nx))

ymax = max(ymax,ceiling(105*max(unfold_fig_data$Density))/100)

# Plot binned
p_step_unfolded <- ggplot(data = filter(step_hist,Treatment=="unfolded")) +
  theme_bw() +
  geom_segment(mapping = aes(x = binLow,
                             y = Counts,
                          xend = binHigh,
                          yend = Counts),
```

```
              color="red", alpha=0.6) +
  geom_segment(mapping = aes(x = binLow,
                             y = CountsL,
                          xend = binLow,
                          yend = Counts),
              color="red", alpha=0.6) +
  scale_x_continuous(breaks = seq(xmin,xmax,2),
                     limits = c(xmin,xmax),
                     expand = c(0,0)) +
  scale_y_continuous("Simulated Counts", limits = c(0,ymax_count), expand = c(0,0),
                     breaks = seq(0,ceiling(ymax_count), floor(ymax_count/1000)*200)) +
  labs(x="X", y="", title=element_blank()) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        plot.margin = margin(0, 0.16, 0.12, 0, "cm"))

p_step_folded <- ggplot(data = filter(step_hist,Treatment=="folded")) +
  theme_bw() +
  geom_segment(mapping = aes(x = binLow,
                             y = Counts,
                          xend = binHigh,
                          yend = Counts),
              color="blue", alpha=0.6) +
  geom_segment(mapping = aes(x = binLow,
                             y = CountsL,
                          xend = binLow,
                          yend = Counts),
              color="blue", alpha=0.6) +
  scale_x_continuous(breaks = seq(xmin,xmax,2),
                     limits = c(xmin,xmax),
                     expand = c(0,0)) +
  scale_y_continuous("Simulated Counts",
                     limits = c(0,ymax_count), expand = c(0,0),
                     breaks = seq(0,ceiling(ymax_count), floor(ymax_count/1000)*200)) +
  labs(x="X", y="", title=element_blank()) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        axis.text.y = element_blank(), axis.ticks.y = element_blank(),
        axis.title.y = element_blank(), plot.margin = margin(0, 0.16, 0.12, 0.08, "cm"))

# Plot continuous
p_smooth_unfolded <- ggplot(filter(unfold_fig_data,Treatment=="unfolded"), aes(X, Density)) +
  geom_line(alpha=0.6, color="red") + theme_bw() + ggtitle("unfolded") +
  scale_x_continuous(limits = c(xmin,xmax), expand = c(0, 0)) +
  scale_y_continuous("Theoretical Density",
                     seq(0,ymax,0.05), limits = c(0,ymax), expand = c(0, 0)) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        axis.text.x = element_blank(), axis.ticks.x = element_blank(),
        axis.title.x = element_blank(), plot.margin = margin(0.08, 0.16, 0.04, 0.08, "cm"))

p_smooth_folded <- ggplot(filter(unfold_fig_data,Treatment=="folded"), aes(X, Density)) +
  geom_line(alpha=0.6, color="blue") + theme_bw() + ggtitle("folded") +
  scale_x_continuous(limits = c(xmin,xmax), expand = c(0, 0)) +
  scale_y_continuous("Theoretical Density",
                     seq(0,ymax,0.05), limits = c(0,ymax), expand = c(0, 0)) +
```

```
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank(),
        axis.text.x = element_blank(), axis.ticks.x = element_blank(),
        axis.title.x = element_blank(),
        axis.text.y = element_blank(), axis.ticks.y = element_blank(),
        axis.title.y = element_blank(), plot.margin = margin(0.08, 0.16, 0.04, 0.08, "cm"))

grid.arrange(p_smooth_unfolded,p_smooth_folded,
             p_step_unfolded,p_step_folded,
             ncol=2, widths=c(12,10), heights=c(9,10))
```

### 6.2 Code used for simulations shown in Figure [3]

```
nsim <- 100000

true <- rcauchy(nsim, 0.5, 1)
eff <-  runif(nsim) > 0.2 + (1.0-0.5)/20*(true+10.0)

predicted <- (true + rnorm(nsim, -2.5, 0.2))[eff]
step_hist_unfolded <- hist(true[which(true>-10 & true<10)],
                           breaks = seq(-10,10,0.5), plot = F)
step_hist_folded <- hist(predicted[which(predicted>-10 & predicted<10)],
                         breaks = seq(-10,10,0.5), plot = F)


ymax <- 1.075*max(c(step_hist_folded$density,step_hist_unfolded$density))
ymax_count <- 1.1*max(c(step_hist_folded$count,step_hist_unfolded$count))

step_hist <-
  tibble(binLow = c(step_hist_folded$breaks[-length(step_hist_folded$breaks)],
                    step_hist_unfolded$breaks[-length(step_hist_unfolded$breaks)]),
         binHigh = c(step_hist_folded$breaks[-1],step_hist_unfolded$breaks[-1]),
         CountsL = c(c(0,step_hist_folded$counts[-length(step_hist_folded$counts)]),
                     c(0,step_hist_unfolded$counts[-length(step_hist_unfolded$counts)])),
         Counts = c(step_hist_folded$counts,step_hist_unfolded$counts),
         DensityL = c(c(0,step_hist_folded$density[-length(step_hist_folded$density)]),
                      c(0,step_hist_unfolded$density[-length(step_hist_unfolded$density)])),
         Density = c(step_hist_folded$density,step_hist_unfolded$density),
         Treatment = c(rep("smeared", length(step_hist_folded$density)),
                       rep("truth", length(step_hist_unfolded$density))))

dists <- ggplot(data = step_hist) +
  theme_bw() +
  geom_segment(mapping = aes(x = binLow,
                             y = Counts,
                          xend = binHigh,
                          yend = Counts,
                          color = Treatment),
               alpha=0.6) +
  geom_segment(mapping = aes(x = binLow,
                             y = CountsL,
                          xend = binLow,
                          yend = Counts,
```

```
                          color = Treatment),
               alpha=0.6) +
  scale_x_continuous(breaks = seq(-10,10,2),
                     limits = c(-10,10),
                     expand = c(0,0)) +
  scale_y_continuous("Simulated Counts", limits = c(0,ymax_count), expand = c(0,0),
                     breaks = seq(0,ceiling(ymax_count), ceiling(ymax_count/1000)*100)) +
  labs(x="X", y="", title=element_blank()) +
  scale_color_manual(values=c("red","blue")) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())

rf <- colorRampPalette(rev(brewer.pal(11,'Spectral')))
r <- rf(32)

smear <- tibble(`True X` = c(true[eff],true[-eff]),
                `Observed X` = c(predicted,rep(-10.249999999,length(true[-eff]))))

smear_m <- ggplot(filter(smear, `True X` <= 10 & `Observed X` <= 10, `Observed X` >= -10 &
                           (`Observed X` >= -10 | `Observed X` == -10.249999999))) +
  scale_x_continuous(breaks = seq(-10,10,2),
                     limits = c(-10,10),
                     expand = c(0,0)) +
  scale_y_continuous(breaks = seq(-12,10,2),
                     limits = c(-10.5,10),
                     expand = c(0,0)) +
  theme_bw() + theme(panel.grid.major = element_blank(),
                     panel.grid.minor = element_blank()) +
  scale_fill_gradientn(colours=r, breaks=seq(0,12000,2000)) +
  stat_bin2d(mapping=aes(x=`True X`, y=`Observed X`),
             breaks = list(x=seq(-10,10,0.5),
                           y=seq(-10.5,10,0.5))) +
  geom_abline(slope = 0,intercept = -10,lty=2)

grid.arrange(dists, smear_m, nrow=1)
```

### 6.3 Code used for Figure [4]

```
test <- tibble(x = rep(seq(-9.5,9.5,1),20),
               y = rep(seq(-9.5,9.5,1),each=20))
test$scale <- exp(-abs(test$x-test$y))

rf <- colorRampPalette(rev(brewer.pal(11,'Spectral')))
r <- rf(40)
ggplot(test) +
  scale_x_continuous(breaks = seq(-10,10,2),
                     limits = c(-10,10),
                     expand = c(0,0)) +
  scale_y_continuous(breaks = seq(-10,10,2),
                     limits = c(-10,10),
                     expand = c(0,0)) +
  theme_bw() + theme(panel.grid.major = element_blank(),
                     panel.grid.minor = element_blank()) +
```

```
  labs(x="True X", y="Observed X", title=expression(alpha ~"values for prior")) +
  scale_fill_gradientn(colours=r, breaks=seq(0,1,0.2)) +
  geom_tile(mapping=aes(x=x, y=y, fill=scale))
```

### 6.4 Code used for unfolding, corresponding to results in Figure [5]

```
# Number of "causes" and "effects"
nc <- ne <-  40

# Get migration counts from stored ggplot and create smearing matrix
smear_data <- ggplot_build(smear_m)$data[[1]]
smearing_matrix <- matrix(rep(0,nc*(ne+1)), ncol = nc)
for( i in 1:nrow(smear_data)){
  smearing_matrix[cbind(smear_data$xbin, (ne+2)-smear_data$ybin)[i,2],
                  cbind(smear_data$xbin, (ne+2)-smear_data$ybin)[i,1]] <- smear_data$value[i]
}
smearing_matrix_normcols <- smearing_matrix /
  matrix(rep(apply(smearing_matrix, MARGIN = 2, sum),ne+1), nrow=ne+1, byrow = T)

# Get observed number of events per effect
xE1 <- smearing_matrix %*% rep(1,nc)

# Number of samples drawn from each posterior distribution
nsamp <- 10
Lamb1 <- theta1 <- invMig1 <- list(NA)
xC1 <- efficiency1 <- mu1 <- intmu1 <- matrix(rep(NA,nsamp*nc), ncol=nsamp)
for(i in 1:nsamp){
  # New smearing matrices sampled from Dirichlet posterior
  Lamb1[[i]] <-
    matrix(unlist(map(1:nc, ~ t(rdirichlet(1, exp(-abs(.-1:(ne+1)))+
                                             smearing_matrix[,.])))), nrow=ne+1)
  # Efficiencies calculated for each
  efficiency1[,i] <- apply(Lamb1[[i]][-(ne+1),], MARGIN = 2, sum)
  # Apply Bayes' theorem to get P(C|E)
  theta1[[i]] <- t(Lamb1[[i]][-(ne+1),]/(matrix(rep(apply(Lamb1[[i]][-(ne+1),],
                                                          MARGIN = 1, sum), nc), ncol=nc)))
  mu1[,i] <- map_dbl(xE1[-(ne+1)], ~ rgamma(1,1+.,1))
  intmu1[,i] <- round(mu1[,i])
  intmu1[,i][which(intmu1[,i]==0)] <- 1
  invMig1[[i]] <- matrix(unlist(map(1:ne, ~ rmultinom(n=1,size=intmu1[.,i],
                                                      prob=theta1[[i]][,.]))),nrow=ne)
  xC1[,i] <- (apply(invMig1[[i]],MARGIN=1, sum)/
                efficiency1[,i])*(mu1[,i]/intmu1[,i])
}

step_hist$barlow <- rep(0,nc+ne)
step_hist$barhigh <- rep(0,nc+ne)

step <- filter(step_hist, Treatment == "truth")
step$Counts <- apply(xC1, MARGIN=1, mean)
step$CountsL[-1] <- apply(xC1, MARGIN=1, mean)[-nc]
step$Treatment <- "1 iteration"
```

```
step$Density <- apply(xC1, MARGIN=1, mean)/sum(apply(xC1, MARGIN=1, mean))
step$DensityL[-1] <- step$Density[-nc]
step$barlow <- apply(xC1, MARGIN=1, FUN=quantile, probs=0.025)
step$barhigh <- apply(xC1, MARGIN=1, FUN=quantile, probs=0.975)
step_hist_iter1 <- rbind(step_hist,step)

ymax_count <- 1.1*max(step_hist_iter1$Counts)

it1 <- ggplot(data = step_hist_iter1) +
  theme_bw() +
  geom_rect(mapping = aes(xmin = binLow,
                          ymin = barlow,
                          xmax = binHigh,
                          ymax = barhigh,
                          fill = Treatment),
            alpha=0.2) +
  geom_segment(mapping = aes(x = binLow,
                             y = Counts,
                          xend = binHigh,
                          yend = Counts,
                          color = Treatment),
               alpha=0.7) +
  geom_segment(mapping = aes(x = binLow,
                             y = CountsL,
                          xend = binLow,
                          yend = Counts,
                          color = Treatment),
               alpha=0.7) +
  scale_x_continuous(breaks = seq(-10,10,2),
                     limits = c(-10,10),
                     expand = c(0,0)) +
  scale_y_continuous("Counts", limits = c(0,ymax_count), expand = c(0,0),
                     breaks = seq(0,ceiling(ymax_count), ceiling(ymax_count/1000)*100)) +
  labs(x="X", y="", title="1 Iteration") +
  scale_color_manual(values=c("orange","red","blue")) +
  scale_fill_manual(values=c("orange","red","blue")) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())
```

```
# Get observed number of events per effect
xE2 <- xC1

# Number of samples drawn from posterior distribution
Lamb2 <- efficiency2 <- theta2 <- invMig2 <- list(NA)
xC2 <- efficiency2 <- mu2 <- intmu2 <- matrix(rep(NA,nsamp*nsamp*nc), ncol=nsamp*nsamp)
for(i in 1:nsamp){
  for(j in 1:nsamp){
    # New smearing matrices sampled from Dirichlet posterior
    Lamb2[[(i-1)*nsamp+j]] <-
      matrix(unlist(map(1:nc, ~ t(rdirichlet(1,exp(-abs(.-1:(ne+1)))+
                                                 smearing_matrix[,.]+c(invMig1[[i]][.,],0))))),
             nrow=ne+1)
    # Efficiencies calculated for each
    efficiency2[,(i-1)*nsamp+j] <- apply(Lamb2[[(i-1)*nsamp+j]][-(ne+1),],MARGIN=2, sum)
```

```
    # Apply Bayes' theorem to get P(C|E)
    theta2[[(i-1)*nsamp+j]] <-
      t((Lamb2[[(i-1)*nsamp+j]][-(ne+1),]*matrix(rep(xC1[,i], nc),byrow=T,ncol=nc))/
          (matrix(rep(Lamb2[[(i-1)*nsamp+j]][-(ne+1),] %*%
                        xC1[,i], nc), ncol=nc)))

    mu2[,(i-1)*nsamp+j] <- map_dbl(1:ne, ~ rgamma(1,1+xE1[.]+xE2[.,j],2))
    intmu2[,(i-1)*nsamp+j] <- round(mu2[,(i-1)*nsamp+j])
    intmu2[,(i-1)*nsamp+j][which(intmu2[,(i-1)*nsamp+j]==0)] <- 1
    invMig2[[(i-1)*nsamp+j]] <-
      matrix(unlist(map(1:ne,~rmultinom(n=1,size=intmu2[.,(i-1)*nsamp+j],
                                        prob=theta2[[(i-1)*nsamp+j]][,.]))),nrow=ne)
    xC2[,(i-1)*nsamp+j] <- (apply(invMig2[[(i-1)*nsamp+j]],MARGIN=1,sum)/efficiency2[,(i-1)*nsamp+j])*
      (mu2[,(i-1)*nsamp+j]/intmu2[,(i-1)*nsamp+j])
  }
}

step <- filter(step_hist, Treatment == "truth")
step$Counts <- apply(xC2, MARGIN=1, mean)
step$CountsL[-1] <- apply(xC2, MARGIN=1, mean)[-nc]
step$Treatment <- "2 iterations"
step$Density <- apply(xC2, MARGIN=1, mean)/sum(apply(xC2, MARGIN=1, mean))
step$DensityL[-1] <- step$Density[-nc]
step$barlow <- apply(xC2, MARGIN=1, FUN=quantile, probs=0.025)
step$barhigh <- apply(xC2, MARGIN=1, FUN=quantile, probs=0.975)
step_hist_iter2 <- rbind(step_hist,step)

ymax_count <- 1.1*max(step_hist_iter2$Counts,step_hist_iter2$barhigh)

it2 <- ggplot(data = step_hist_iter2) +
  theme_bw() +
  geom_rect(mapping = aes(xmin = binLow,
                          ymin = barlow,
                          xmax = binHigh,
                          ymax = barhigh,
                          fill = Treatment),
            alpha=0.2) +
  geom_segment(mapping = aes(x = binLow,
                             y = Counts,
                          xend = binHigh,
                          yend = Counts,
                          color = Treatment),
               alpha=0.7) +
  geom_segment(mapping = aes(x = binLow,
                             y = CountsL,
                          xend = binLow,
                          yend = Counts,
                          color = Treatment),
               alpha=0.7) +
  scale_x_continuous(breaks = seq(-10,10,2),
                     limits = c(-10,10),
                     expand = c(0,0)) +
  scale_y_continuous("Counts", limits = c(0,ymax_count), expand = c(0,0),
```

```
                     breaks = seq(0,ceiling(ymax_count), ceiling(ymax_count/1000)*100)) +
  labs(x="X", y="", title="2 Iterations") +
  scale_color_manual(values=c("orange","red","blue")) +
  scale_fill_manual(values=c("orange","red","blue")) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())
```

```
# Get observed number of events per effect
xE3 <- xC2

# Number of samples drawn from posterior distribution
Lamb3 <- efficiency3 <- theta3 <- invMig3 <- list(NA)
xC3 <- efficiency3 <- mu3 <- intmu3 <- matrix(rep(NA,nc*nsamp^3), ncol=nsamp^3)
for(i in 1:nsamp){
  for(j in 1:nsamp){
    for(k in 1:nsamp){
      # New smearing matrices sampled from Dirichlet posterior
      Lamb3[[(i-1)*nsamp^2+(j-1)*nsamp+k]] <-
        matrix(unlist(map(1:nc, ~ t(rdirichlet(1,exp(-abs(.-1:(ne+1)))+
                                                 smearing_matrix[,.]+
                                                 c(invMig1[[i]][.,],0)+
                                                 c(invMig2[[(j-1)*nsamp+k]][.,],0))))),
               nrow=ne+1)
      # Efficiencies calculated for each
      efficiency3[,(i-1)*nsamp^2+(j-1)*nsamp+k] <-
        apply(Lamb3[[(i-1)*nsamp^2+(j-1)*nsamp+k]][-(ne+1),],MARGIN=2, sum)
      # Apply Bayes' theorem to get P(C|E)
      theta3[[(i-1)*nsamp^2+(j-1)*nsamp+k]] <-
        t((Lamb3[[(i-1)*nsamp^2+(j-1)*nsamp+k]][-(ne+1),]*
             matrix(rep(xC3[,(j-1)*nsamp+k], nc),byrow=T,ncol=nc))/
            (matrix(rep(Lamb3[[(i-1)*nsamp^2+(j-1)*nsamp+k]][-(ne+1),] %*%
                          xC3[,(j-1)*nsamp+k], nc), ncol=nc)))

      mu3[,(i-1)*nsamp^2+(j-1)*nsamp+k] <-
        map_dbl(1:ne, ~ rgamma(1,1+xE1[.]+xE2[.,k]+xE3[(j-1)*nsamp+k],3))
      intmu3[,(i-1)*nsamp^2+(j-1)*nsamp+k] <-
        round(mu3[,(i-1)*nsamp^2+(j-1)*nsamp+k])
      intmu3[,(i-1)*nsamp^2+(j-1)*nsamp+k][which(intmu3[,(i-1)*nsamp^2+(j-1)*nsamp+k]==0)] <- 1
      invMig3[[(i-1)*nsamp^2+(j-1)*nsamp+k]] <-
        matrix(unlist(map(1:ne,~rmultinom(n=1,size=intmu3[.,(i-1)*nsamp^2+(j-1)*nsamp+k],
                                          prob=theta2[[(i-1)*nsamp+j]][,.]))),nrow=ne)
      xC3[,(i-1)*nsamp^2+(j-1)*nsamp+k] <-
        (apply(invMig3[[(i-1)*nsamp^2+(j-1)*nsamp+k]],
               MARGIN=1,sum)/efficiency3[,(i-1)*nsamp^2+(j-1)*nsamp+k])*
        (mu3[,(i-1)*nsamp^2+(j-1)*nsamp+k]/intmu3[,(i-1)*nsamp^2+(j-1)*nsamp+k])
    }
  }
}

step <- filter(step_hist_iter1, Treatment == "truth")
step$Counts <- apply(xC3, MARGIN=1, mean)
step$CountsL[-1] <- apply(xC3, MARGIN=1, mean)[-nc]
step$Treatment <- "3 iterations"
step$Density <- apply(xC3, MARGIN=1, mean)/sum(apply(xC3, MARGIN=1, mean))
```

```
step$DensityL[-1] <- step$Density[-nc]
step$barlow <- apply(xC3, MARGIN=1, FUN=quantile, probs=0.025)
step$barhigh <- apply(xC3, MARGIN=1, FUN=quantile, probs=0.975)
step_hist_iter3 <- rbind(step_hist,step)

ymax_count <- 1.1*max(step_hist_iter3$Counts,step_hist_iter2$barhigh)

it3 <- ggplot(data = step_hist_iter3) +
  theme_bw() +
  geom_rect(mapping = aes(xmin = binLow,
                          ymin = barlow,
                          xmax = binHigh,
                          ymax = barhigh,
                          fill = Treatment),
            alpha=0.2) +
  geom_segment(mapping = aes(x = binLow,
                             y = Counts,
                          xend = binHigh,
                          yend = Counts,
                          color = Treatment),
               alpha=0.7) +
  geom_segment(mapping = aes(x = binLow,
                             y = CountsL,
                          xend = binLow,
                          yend = Counts,
                          color = Treatment),
               alpha=0.7) +
  scale_x_continuous(breaks = seq(-10,10,2),
                     limits = c(-10,10),
                     expand = c(0,0)) +
  scale_y_continuous("Counts", limits = c(0,ymax_count), expand = c(0,0),
                     breaks = seq(0,ceiling(ymax_count), ceiling(ymax_count/1000)*100)) +
  labs(x="X", y="", title="3 Iterations") +
  scale_color_manual(values=c("orange","red","blue")) +
  scale_fill_manual(values=c("orange","red","blue")) +
  theme(panel.grid.major = element_blank(), panel.grid.minor = element_blank())
```

# References


[1] Tim Adye. "Unfolding algorithms and tests using RooUnfold". In: *PHYSTAT 2011*. Geneva: CERN, 2011, pp. 313–318. DOI: 10.5170/CERN-2011-006.313. arXiv: 1105.1160 [physics.data-an].

[2] Volker Blobel. "Unfolding". In: *Data analysis in high energy physics: A practical guide to statistical methods*. Ed. by Olaf Behnke et al. Weinheim, Germany: Wiley-VCH, 2013. Chap. 6, pp. 187–226.

[3] Carsten Daniel Burgard. *RooUnfold*. https://gitlab.cern.ch/RooUnfold/RooUnfold. 2021.

[4] G. D'Agostini. "A Multidimensional unfolding method based on Bayes' theorem". In: *Nucl. Instrum. Meth. A* 362 (1995), pp. 487–498. DOI: 10.1016/0168-9002(95)00274-X.

[5] G. D'Agostini. "Improved iterative Bayesian unfolding". In: *Alliance Workshop on Unfolding and Data Correction*. Oct. 2010. arXiv: 1010.0632 [physics.data-an].

[6] Andreas Hocker and Vakhtang Kartvelishvili. "SVD approach to data unfolding". In: *Nucl. Instrum. Meth. A* 372 (1996), pp. 469–481. DOI: 10.1016/0168-9002(95)01478-0. arXiv: hep-ph/9509307.

[7] Alexander Meister. *Deconvolution Problems in Nonparametric Statistics*. Vol. Lecture Notes in Statistics. 193. Springer, Berlin, Heidelberg, 2009, pp. VI, 210. ISBN: 978-3-540-87556-7. DOI: 10.1007/978-3-540-87557-4.

[8] Victor M. Panaretos. "A Statistician's View on Deconvolution and Unfolding". In: *PHYSTAT 2011*. Geneva: CERN, 2011, pp. 252–259. DOI: 10.5170/CERN-2011-006.252. eprint: 1105.1160.

[9] Stefan Schmitt. "TUnfold: an algorithm for correcting migration effects in high energy physics". In: *JINST* 7 (2012), T10003. DOI: 10.1088/1748-0221/7/10/T10003. arXiv: 1205.6201 [physics.data-an].

[10] Eric W. Weisstein. *Convolution. From MathWorld—A Wolfram Web Resource*. Last visited on 15/2/2022. URL: https://mathworld.wolfram.com/Convolution.html.