Resolving Histogram Binning Dilemmas with Binless and Binfull Algorithms

Abram Krislock[1]
Nathan Krislock[2]

May 17, 2014

[1] Department of Physics, Stockholm University – Oskar Klein Centre, AlbaNova, SE-106 91, Stockholm, Sweden
[2] Department of Mathematical Sciences, Northern Illinois University, DeKalb, Illinois 60115, USA

Abstract

The histogram is an analysis tool in widespread use within many sciences, with high energy physics as a prime example. However, there exists an inherent bias in the choice of binning for the histogram, with different choices potentially leading to different interpretations. This paper aims to eliminate this bias using two "debinning" algorithms. Both algorithms generate an observed cumulative distribution function from the data, and use it to construct a representation of the underlying probability distribution function. The strengths and weaknesses of these two algorithms are compared and contrasted. The applicability and future prospects of these algorithms is also discussed.

1 Introduction

High energy physics (HEP) research makes common use use of the histogram as a data analysis tool[1]. A great deal of particle physics data, both experimental [1] and phenomenological [2], is analyzed with histograms.

[1] Many other areas of scientific research involving statistical analysis also commonly use the histogram. The techniques described in this paper apply to any such similar analysis. For concreteness, in this paper we will restrict our discussion to that of HEP research.

The histogram is a way of representing the data in such a way as to have it look like and be comparable to the *underlying* probability distribution function (UPDF) of the particle physics reality or model prediction. We will refer to the histogram, or any other representation, of a given data set as an *observed* probability distribution function (OPDF). The UPDF depends on some set of parameters, and so a set of bins for the OPDF histogram are chosen in terms of those parameters.

However, the choice of these bins involves an inherent bias [3]. The analysis of histogrammed data can be highly dependent upon the set of bins chosen. This is particularly true for the case of performing a fit of a histogram with a theoretical or parameterized UPDF; the results obtained from different choices of bin sets may differ more than the reported uncertainty in the fit.

We demonstrate this problem with a simple example from HEP phenomenology: Early searches for supersymmetry at the Large Hadron Collider were expecting to find kinks or endpoints within kinematical distributions such as invariant masses [4], or the m_{T2} variable [5,6]. We show such a possible signal in Fig. 1.1, which shows two histograms of the same data but with different bin sets. The two histograms are then fit with the following function:

$$y = \begin{cases} m_1(x - x_{\text{kink}}) + b, & \text{if } x < x_{\text{kink}}, \\ m_2(x - x_{\text{kink}}) + b, & \text{if } x \geq x_{\text{kink}}. \end{cases} \tag{1.1}$$

This equation describes a bent line with a kink at $x = x_{\text{kink}}$. The slopes of the lines on either side of the kink are m_1 and m_2, and b is the value of the function at the kink. The location of the kink is the only parameter of interest. Fig. 1.1 shows the two histograms fit with this function, as well as the fit result and uncertainty for the parameter x_{kink}. As the figure shows, the locations of x_{kink} from the two fits do not agree. If we treat the fit results from the figure as normal distributions, the probability for a sample x_{kink2} to be greater than a sample x_{kink1} is 0.2%. Thus, they disagree at the 99.8% level.

Thus, the choice of bin set can certainly affect the outcome of an analysis. Bin sets are typically chosen by eye to be the smallest width bins such that there are "enough" statistics in the bins of greatest interest. Ofttimes the bin set is chosen under the constraint of aesthetic rather than scientific reasons. Authors choose uniform bin sizes, usually using bin widths of two, five, or ten times their measurement unit. The bin edges are also chosen to line up with notable x-axis landmarks, such as the origin. The consequences of these

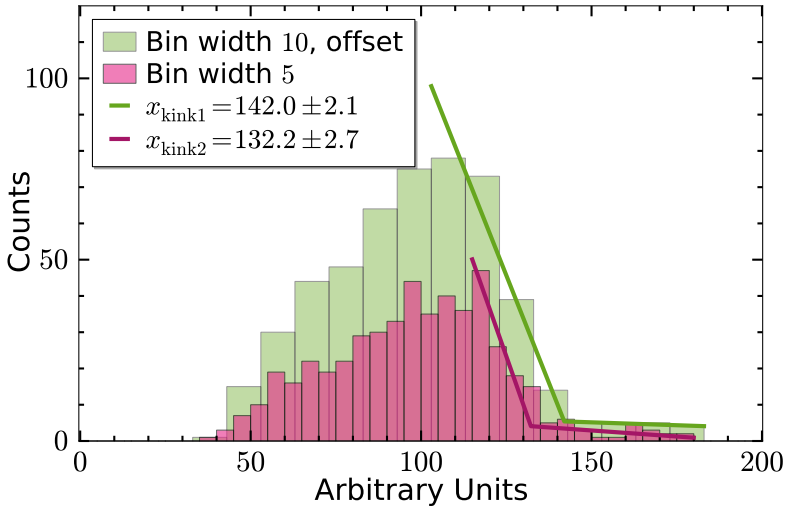

Figure 1.1: The same data is used to fill two histograms with different bin sets. Eq. (1.1) is used to fit each of these from their peak bin to their last non-zero bin. The green (light gray) histogram has offset bins, with the first bin edge beginning at $x = 3$ and a bin width of 10. The purple (gray) histogram has a bin width of 5, beginning at the origin. The fit results for these two histograms are shown as green (gray) and purple (dark gray) lines, respectively. The locations of the kinks, which are shown in the figure legend, disagree at the 99.8% level.

choices on scientific results is generally not discussed.

Thus, our goal for this paper is to avoid such bias by constructing OPDF representations other than regular histograms. Our new representations must avoid the above mentioned binning dilemmas. In this paper, we will present two such "debinning" algorithms, both of which involve the relations between the OPDF and the observed Cumulative Distribution Function (OCDF). These algorithms were inspired by the method used in [3]. The "binless" algorithm never bins the data at all, instead determining the OPDF as the smoothed, numerical derivative of the OCDF. In contrast, the "binfull" algorithm uses the OCDF as a Monte Carlo generator for the OPDF. A smoothing function is applied during the generation of a very large number of points. These points are then used to create a histogram which is full of very small bins. Each of these two methods has its strengths and weaknesses, which we will also describe.

Currently, both of these algorithms assume an intimate understanding of the backgrounds inherent in a given HEP study. Our algorithms both consider the comparison between the debinned background-only OPDF and the background UPDF. We run each algorithm multiple times over the background data, scanning to find the best-fit value of the smoothing parameter. We then use the resulting smoothing parameter to generate the ultimate debinned full-data OPDF.

The remainder of this paper is organized as follows. In Sec. 2, we describe the binless algorithm by deriving its equations, describing their implementation, and presenting example plots. The binfull algorithm is described in Sec. 3, where we describe the Monte Carlo generator, smoothing function, and implementation, as well as presenting more example plots. We conclude, compare these two methods, and comment on future work in Sec. 4. Lastly, we have included Appx. A to describe the generation of the data used for testing these debinning algorithms. Additional plots, animations, and all of our code can be found on our debinning webpage [7].

2 The Binless Algorithm

As stated in the introduction, the binless algorithm first forms the OCDF from the data. This is followed by taking the smoothed numerical derivative of it to obtain the binless OPDF.

The method of constructing an OCDF is as follows [3]. First, the data is sorted by value into an array. Thus we have, say, N data values, $x_1, \cdots, x_N$, where $x_i > x_j$ if $i > j$. Then, the OCDF, $z(x)$ is constructed from this data simply as,

$$z(x) = \begin{cases} 0, & \text{if } x < x_1, \\ i, & \text{if } x_i \leq x < x_{i+1}, \text{ for } i \in [1, N-1], \\ N, & \text{if } x_N \leq x. \end{cases} \tag{2.1}$$

This function z thus looks like a staircase with stairs of non-uniform length.

Next, we take the derivative of z to arrive at our binless OPDF. The reason it must be a smoothed numerical derivative is that taking a naive numerical derivative of z results in a large amount of noise in the resulting binless OPDF [8]. Thus, to obtain a binless OPDF which is even remotely useful, we must have a more sophisticated numerical differentiation algorithm.

This algorithm must be able to smooth the noise introduced by numerical differentiation.

2.1 Numerical Differentiation with Noise Smoothing

We use the method of numerical differentiation presented in [8], which is a least squares minimization problem including a Total Variation (TV) penalization term [9]. The problem is defined as follows: We are working in $\Re$ over some domain $\Omega = [0, L]$, using the inner-product $\langle u, v\rangle = \int_\Omega u(x)v(x)\,\mathrm{d}x$. We are trying to find the function, $u(x)$, which describes the derivative of some data, $z(x)$, while ensuring that $u(x)$ does not vary too much due to the noise in $z(x)$. The plan is to minimize the following functional[2]:

$$f[u] = \frac{1}{2}\|Au - z\|_2^2 + \alpha\|u'\|_1. \tag{2.2}$$

Note that the 2-norm is defined to be $\|v\|_2 = \sqrt{\langle v, v\rangle} = \sqrt{\int_\Omega [v(x)]^2\,\mathrm{d}x}$ and the 1-norm is defined as $\|v\|_1 = \int_\Omega |v(x)|\,\mathrm{d}x$.

The functional in Eq. (2.2) contains two terms. The first term, the least squares term, penalizes the difference between the anti-derivative of $u(x)$, where $[Au](x) \equiv \int_0^x u(w)\,\mathrm{d}w$, and $z(x)$. The second term is the TV term which minimizes the total amount of variation of $u(x)$. The parameter α controls the relative influence of the TV term compared to the least squares term.

In order to find the u that minimizes the functional $f[u]$ in Eq. (2.2), we first need to determine the gradient, or functional derivative, of f. However, since the TV term in $f[u]$ is not differentiable when $\|u'\|_1 = 0$, we will instead minimize

$$f_\beta[u] = \frac{1}{2}\|Au - z\|_2^2 + \alpha\int_\Omega \sqrt{(u'(x))^2 + \beta^2}\,\mathrm{d}x, \tag{2.3}$$

where β is a small positive parameter.

We remind the reader that the functional derivative can be defined as follows:

$$\frac{\partial f[u]}{\partial u}(x) = \lim_{h\to 0}\frac{f[u + h\delta(\cdot - x)] - f[u]}{h}. \tag{2.4}$$

[2]In this article, whenever its meaning is unambiguous, an apostrophe or "prime" in any equation will be understood to be a regular derivative, such that $u'(x) \equiv \frac{\mathrm{d}u}{\mathrm{d}x}$.

Here, δ is the Dirac delta function [10]. For our purposes, the relevant properties of the Dirac delta function are

$$\begin{aligned}\int_\Omega v(x)\delta(x-y)\,\mathrm{d}x &= v(y)\\ \int_\Omega v(x)\delta'(x-y)\,\mathrm{d}x &= -v'(y)\\ \int_\Omega v(x)(\textstyle\int_0^x \delta(w-y)\,\mathrm{d}w)\,\mathrm{d}x &= \displaystyle\int_y^L v(x)\ \mathrm{d}x\end{aligned} \tag{2.5}$$

for any function v which is defined over Ω.

Computing the gradient of Eq. (2.3), using Eq. (2.5), is left as an exercise for the reader. The result is

$$g[u] \equiv \frac{\partial f_\beta[u]}{\partial u} = \left[A^\ddagger A + \alpha L(u)\right] u - A^\ddagger z, \tag{2.6}$$

where the L operator is given by

$$L(u)v \equiv -\left(\frac{v'}{\sqrt{(u')^2+\beta^2}}\right)', \tag{2.7}$$

and $A^\ddagger$ is the adjoint of the A anti-derivative operator, and is given by $[A^\ddagger v](x) \equiv \int_x^L v(w)\,\mathrm{d}w$. Note that $A^\ddagger$ satisfies $\langle Au, v\rangle = \langle u, A^\ddagger v\rangle$.

We define our algorithm by recognizing that the minimum of the least squares functional is found by setting the gradient to zero, and iterating the resulting equation, which is

$$\left[A^\ddagger A + \alpha L(u^{(k)})\right] u^{(k+1)} = A^\ddagger z. \tag{2.8}$$

In this equation, $u^{(k)}$ is the result of the kth iteration, starting from some initial guess, $u^{(0)}$. We can define the step of our algorithm to be $s^{(k)} \equiv u^{(k+1)} - u^{(k)}$. Then the algorithm equation becomes

$$H^{(k)}s^{(k)} = -g[u^{(k)}], \qquad u^{(k+1)} = u^{(k)} + s^{(k)}, \tag{2.9}$$

where $H^{(k)} \equiv \left[A^\ddagger A + \alpha L(u^{(k)})\right]$. Since $H^{(k)}$ is the Hessian, or second derivative, of f_β, this is Newton's method for minimizing f_β.

2.2 Implementation for a General Derivative

Unlike [8], our x values are not uniformly spaced for our data $z(x)$. Thus, some care has to be taken in defining the various operators which go into the algorithm equation, Eq. (2.9). Since our data $z(x)$ is a "staircase" function, we treat it as a column vector, z, associated with a column vector, x. Both of these vectors have length N, the number of data points. The operators we need are then defined as square matrices which act on these vectors.

The derivative matrix is based upon the discretization of a simple derivative:

$$\frac{\mathrm{d}z}{\mathrm{d}x}(x) = \lim_{h\to 0} \frac{z(x) - z(x-h)}{h} \quad \Rightarrow \quad \left[\frac{\mathrm{d}z}{\mathrm{d}x}\right]_{\mathrm{i}} \approx \frac{z_{\mathrm{i}} - z_{\mathrm{i}-1}}{x_{\mathrm{i}} - x_{\mathrm{i}-1}}. \tag{2.10}$$

The derivative is then defined in terms of a forward difference matrix, D, where

$$D = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & -1 \\ -1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & -1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 1 \end{pmatrix}. \tag{2.11}$$

(The -1 in the upper right corner serves to define a derivative with periodic boundary conditions.) The derivative matrix, ∇_x, is then given as

$$\nabla_x = (\Delta_x)^{-1} D, \tag{2.12}$$

where Δ_x is a diagonal matrix, with ith entry $[\Delta_x]_{\mathrm{ii}} = x_{\mathrm{i}} - x_{\mathrm{i}-1}$ (where $x_0 \equiv 0$), and $(\Delta_x)^{-1}$ is also diagonal, with ith entry $[(\Delta_x)^{-1}]_{\mathrm{ii}} = ([\Delta_x]_{\mathrm{ii}})^{-1}$.

The A and $A^{\ddagger}$ operators are similarly based upon the discretization of a simple integral. We discretize our integral using trapezoidal areas:

$$[Az]_{\mathrm{i}} = \int_0^{x_i} z(w)\,\mathrm{d}w \approx \sum_{j=1}^{i} \frac{1}{2}(x_{\mathrm{j}} - x_{\mathrm{j}-1})(z_{\mathrm{j}} + z_{\mathrm{j}-1}). \tag{2.13}$$

Here, it is again understood that $x_0 \equiv 0$ and $z_0 \equiv 0$. Thus, A is defined as

$$A = \frac{1}{2} \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 1 & 0 \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{pmatrix} \Delta_x \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}. \tag{2.14}$$

Since

$$\left[A^{\ddagger}z\right]_{\mathrm{i}} = \int_{x_i}^{L} z(w)\,\mathrm{d}w \approx \sum_{j=i+1}^{L} \frac{1}{2}(x_{\mathrm{j}} - x_{\mathrm{j-1}})(z_{\mathrm{j}} + z_{\mathrm{j-1}}), \tag{2.15}$$

we find that $A^{\ddagger}$ is similarly defined as

$$A^{\ddagger} = \frac{1}{2}\begin{pmatrix} 0 & 1 & 1 & \cdots & 1 & 1 \\ 0 & 0 & 1 & \cdots & 1 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & \cdots & 0 & 0 \end{pmatrix} \Delta_x \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & 0 & \cdots & 1 & 1 \end{pmatrix}. \tag{2.16}$$

Finally, care also must be taken in constructing the $L(u^{(k)})$ operator. Since it depends on the iteration guess $u^{(k)}$, it must be re-initialized for each step of the algorithm. Considering Eq. (2.7), we see that derivatives appear in both the numerator and denominator within L. We can approximate this "derivative ratio" with a "ratio of forward difference matrix operations". Thus, the discretization of L is

$$L(u^{(k)}) = -\nabla_x \left(\widetilde{\Delta}_{u^{(k)}}\right)^{-1} D \tag{2.17}$$

where $\widetilde{\Delta}_{u^{(k)}}$ is a diagonal matrix, such that $\left[\widetilde{\Delta}_{u^{(k)}}\right]_{\mathrm{ii}} = \sqrt{[Du^{(k)}]_{\mathrm{i}}^2 + \beta^2}$.

2.3 Binless Algorithm Python Script

The main work in the binless algorithm is solving the linear system in Eq. (2.9) each iteration. Since we would like to apply the binless algorithm to very large data sets, simply forming the matrix $H^{(k)}$ and storing it will require vast amounts of time and memory. Therefore, we need to solve this linear system using an iterative matrix-free method. Such methods do not require the full matrix $H^{(k)}$, but instead just need a method for applying $H^{(k)}$ to a vector (i.e., given a vector v, the method returns $H^{(k)}v$).

Our `binless` Python module is a collection of simple functions which, given a data set x, provide the discretized calculus operators described in Sec. 2.2 as either sparse matrices or matrix-free methods. The derivative (and related) operators are very sparse matrices, so the functions which provide them simply return them as NumPy [11] linked-list sparse matrices. On

the other hand, the anti-derivative operators are not sparse, but we can represent them using efficient matrix-free methods, such as using cummulative summation for the left-most matrix in the description of A in Eq. (2.14).

We found the iterative solver LGMRES (loose generalized minimum residual algorithm) [12] to be particularly effective for solving the linear system in Eq. (2.9). However, we also found that it was necessary to use a well-chosen preconditioner P. A good preconditioner P will approximate the matrix $H^{(k)}$ but will also be easy to invert. Instead of solving the linear system $H^{(k)}s^{(k)} = -g[u^{(k)}]$, we solve the equivalent system:

$$\left(H^{(k)}P^{-1}\right) y = -g[u^{(k)}], \qquad s^{(k)} = P^{-1}y.$$

Recall that $H^{(k)} = A^{\ddagger}A + \alpha L(u^{(k)})$. We found that the following preconditioner was very effective:

$$P = \mathrm{Diag}(\mathrm{diag}(A^{\ddagger}A)) + \alpha L(u^{(k)}),$$

where $\mathrm{diag}(M)$ returns the diagonal of a square matrix M and $\mathrm{Diag}(v)$ returns a diagonal matrix with the vector v along its diagonal. Using LGMRES with this preconditioner makes the binless algorithm very fast. This allows us to run it multiple times using the background-only data to find the best-fit smoothing parameter α within a short amount of time.

The default run of our `runbinless.py` script generates multiple binless OPDFs as follows:

1. Loop over the sample signal plus background UPDFs defined in `utilities/sampleFunctions.py`.

2. Generate (or load) $N = 1000$ data points for each. Background-only data is also generated.

3. Iteratively run the binless algorithm on the background data and compare each result to the background-only UPDF to find the best-fit smoothing parameter, α.

4. Using the best-fit α, run the binless algorithm on the full signal plus background data.

5. Save (if necessary) the UPDF data and create the binless plots.

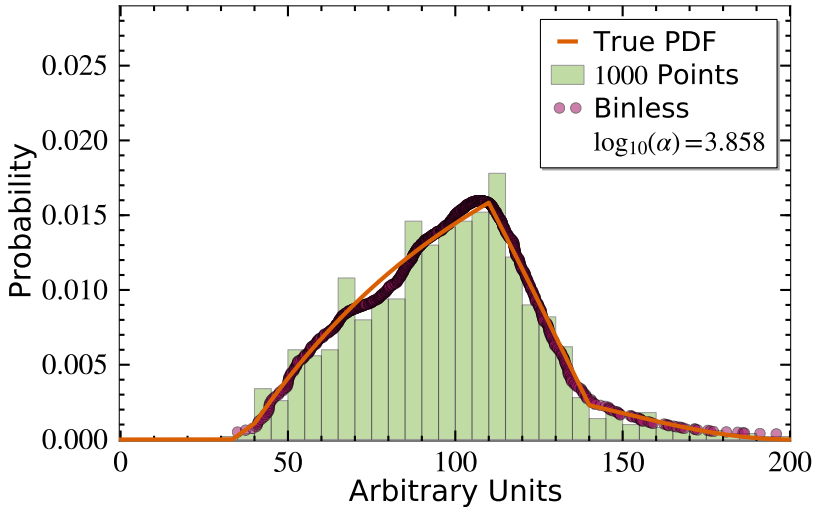

Figure 2.1: A comparison between the OPDFs and UPDF for the "easy endpoint" (Appx. A.1). The easy endpoint UPDF is shown as an orange (gray) line. The regular histgoram is filled with 1000 data points, and is shown with green (light gray) filled bars. The binless OPDF is shown as 1000 very densly packed circular points. The best-fit smoothing parameter α is given within the legend.

We present the results of this process for two sample functions here. The results for the UPDFs described in Appx. A.1 and Appx. A.2 are shown in Fig. 2.1 and Fig. 2.2, respectively. The former shows an excellent agreement between the binless result and the UPDF. The latter shows the limitation of using periodic boundary conditions within the binless algorithm.

3 The Binfull Algorithm

The binfull algorithm utilizes a Monte Carlo generator which is based upon the OCDF and can effectively regenerate the original data. To start, we construct the OCDF as $z(x)$ from Eq. (2.1). To regenerate a data value, a number, r, between zero and one is pulled from a pseudo-random number generator. We multiply by the total number of data values, N, and then find the smallest $z(x_i)$ value of the OCDF which is greater than or equal to rN. The result is x_i.

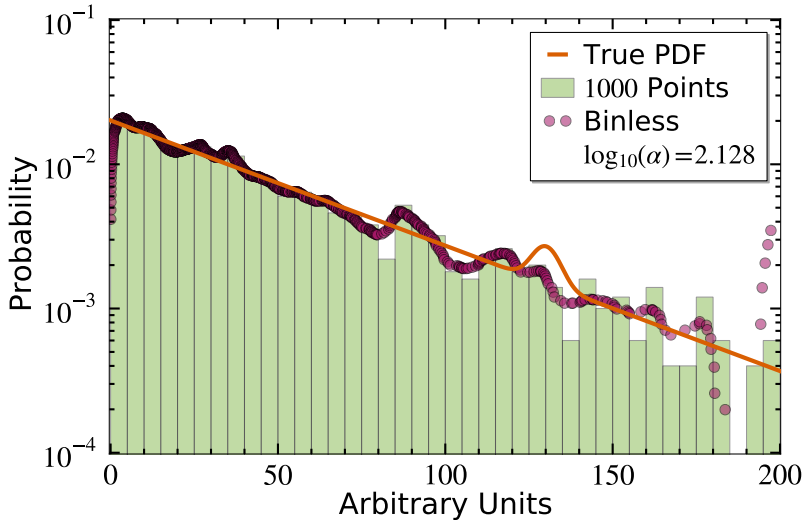

Figure 2.2: A comparison between the OPDFs and UPDF for "the line" (Appx. A.2).The line UPDF is shown as an orange (gray) line. The regular histgoram is filled with 1000 data points, and is shown with green (light gray) filled bars. The binless OPDF is shown as 1000 very densly packed circular points. The points near $x = 0$ and $x = 200$ look particularly bad due to the periodic boundary conditioned derivative operators. The best-fit smoothing parameter α is given within the legend.

Like the naive numerical derivative of the OCDF, there is not much use to this Monte Carlo method without data smoothing. A simple example of applying a smoothing function is to add a random deviate, $\epsilon(\sigma)$, which is pulled from a Gaussian distribution of width σ. Thus, instead of filling our binfull histogram with x_i, we fill it with $x_i + \epsilon(\sigma)$. We can run this smoothed Monte Carlo generator an arbitrarily large number of times. The more points it generates, the smaller we can make the bins in our binfull histogram.

3.1 Binfull Algorithm Python Script

The `binfull` module contains utility functions, classes representing different smoothing functions, and a class to contain the binfull histogram resulting from this algorithm. Like a regular histogram, the binfull histogram stores data as a set of bins and bin contents. (Storing all of the raw binfull data turned out to be a memory disaster.)

The default run for `runbinless.py` is as follows:

1. Loop over the sample signal plus background UPDFs defined in `utilities/sampleFunctions.py`.

2. Generate (or load) $N = 1000$ data points for each. Background-only data is also generated.

3. If binfull data already exists, load it and skip the next two steps.

4. Iteratively run the binfull algorithm on the background data and compare each result to the background-only UPDF to find the best-fit smoothing parameter, σ, for each smoothing function.

5. Using the best-fit σ for each smoothing function, run the binless algorithm on the full signal plus background data.

6. Save (if necessary) the UPDF and binfull data, and create the binfull plots.

We now show the binfull results for the same data sets as we used in Sec. 2.3. They are shown in Fig. 3.1 and Fig. 3.2, respectively. For these plots, we use a Gaussian smoothing function whose width grows linearly every time the same x_i is generated by the Monte Carlo. Thus, as each point x_i is generated, we fill our binfull histogram with $x_i + \epsilon\left(\frac{\sigma \times N_i}{N_{\text{binfull}}}\right)$, where N_i is the number of x_is given by the Monte Carlo thus far, and $N_{\text{binfull}} = 10^5$ is the default number of points generated by the binfull algorithm.

4 Comparison, Conclusion, and Continuation

In this paper we have examined two algorithms for generating representations of OPDFs other than histograms. The binless algorithm determines an OPDF function as the smoothed derivative of the OCDF. The binfull algorithm creates an OPDF histogram which is so full of small bins that it may as well not have bins. Each of these methods has its own strengths and weaknesses.

The binless algorithm is incredibly fast and well suited for larger data sets. The memory requirements of the binless algorithm are small, since for an input data array of N data points, only arrays of length N and $N \times N$

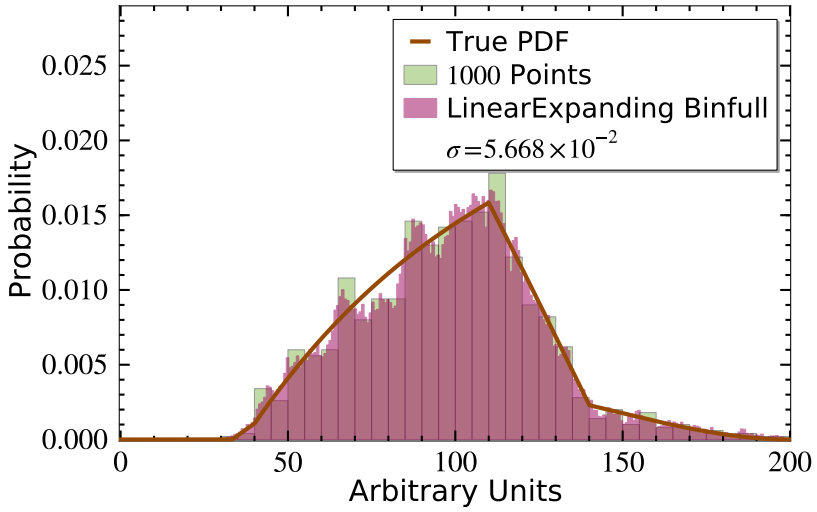

Figure 3.1: A comparison between the OPDFs and UPDF for the "easy endpoint" (Appx. A.1). The easy endpoint UPDF is shown as an brown (dark gray) line. The regular histgoram is filled with 1000 data points, and is shown with green (light gray) filled bars. The binless histogram is shown as a purple (gray) filled region. The best-fit smoothing parameter σ is given within the legend.

matrices are involved. These matrices are either sparse, or represented as matrix-free methods, each of which returns the operation of the matrix on an array. The result of the binless algorithm is also just an array of length N, which is no more memory demanding than the input. Thus, the entire result can be stored for future use.

However, the current implementation of the binless algorithm is only useful for OPDFs which tend towards zero at either end of the domain $\Omega = [0, L]$. This is due to the periodic boundary conditions which we built into the derivative matrices. Additionally, the binless algorithm suffers from a smoothing parameter which is not easy to interpret. It is not at all intuitive as to how α affects the minimization algorithm used to determine the binless OPDF.

The weaknesses of the binless algorithm tend to be the strengths of the binfull algorithm and vice versa. For instance, the smoothing of the binfull algorithm is highly customizable, since one may program their own smoothing function. Because of this, the smoothing parameter used for the binfull

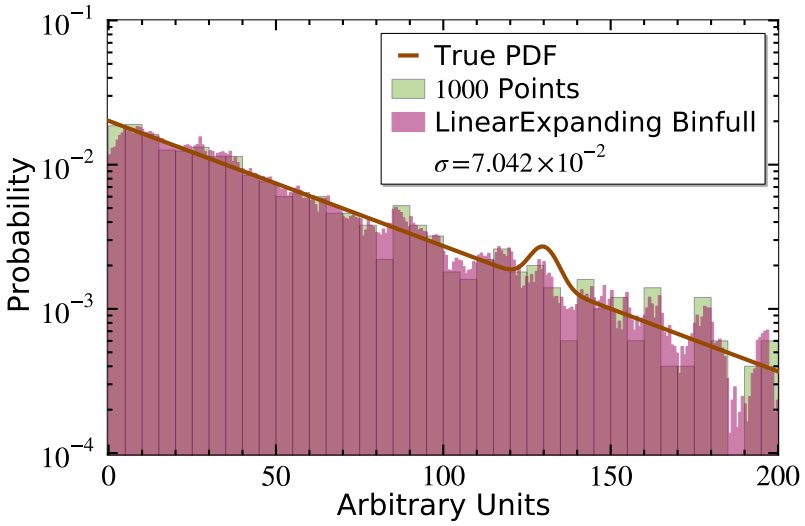

Figure 3.2: A comparison between the OPDFs and UPDF for "the line" (Appx. A.2). The easy endpoint UPDF is shown as an brown (dark gray) line. The regular histgoram is filled with 1000 data points, and is shown with green (light gray) filled bars. The binless histogram is shown as a purple (gray) filled region. The best-fit smoothing parameter σ is given within the legend.

algorithm is very intuitive. Also, the binfull algorithm can easily handle OPDFs of any shape, including OPDFs which tend to large values at only one end of the domain.

However, the binfull algorithm is much slower and requires much more memory than the binless algorithm. The default `runBinless.py` script is roughly ten times faster than `runBinfull.py`. The binfull algorithm may be sped up by keeping each and every point of data which it generates within the speedy NumPy arrays. Unfortunately, it then becomes a memory disaster, potentially freezing up the user's computer. Thus, by default, binfull histograms retain only information about their bins and bin contents.

Ultimately, these two methods are quite complementary, each one making up for the weaknesses of the other. Together, they certainly overcome the binning bias inherent in regular histograms. The smoothing parameter they each use is determined blindly, chosen as the value which best reproduces the well-understood background UPDFs.

We even view the shortcomings of these algorithms instead as opportuni-

ties for further study. For instance, the determination of smoothing parameters is currently dependent upon the assumption that the background UPDF is known. A data driven method to determine the smoothing parameter or smoothing function would be more ideal. Also, it would be nice to reformulate the problem or the code in order to overcome some of the individual weaknesses of each algorithm, such as flexibility, speed, or memory issues.

Lastly, it is very important to understand the proper way to statisticlly interpret the results of these algorithms. This is well understood for regular histograms, and crucial for understanding the physics of the UPDF. We are eager to pursue all of these goals in future studies.

Acknowledgements

Abram Krislock would like to thank Teruki Kamon and Jan Conrad for useful discussions and follow-up ideas, and Maria Teresa Reynolds for ongoing support and motivation.

References

[1] J. Beringer *et al.* (Particle Data Group), "Review of Particle Physics," Phys. Rev. D **86** (2012) 010001.

[2] Due to the breadth and variety of high energy particle physics phenomenology, a comprehensive citation list is too long to reference here. The particular subset of papers which inspired this work is [4–6, 13–27].

[3] B. A. Berg, "Display of probability densities for data from a continuous distribution," Phys. Procedia **15** (2011) 17, `arXiv:1105.0696`.

[4] I. Hinchliffe, F. Paige, M. Shapiro, J. Soderqvist, and W. Yao, "Precision SUSY measurements at CERN LHC," Phys. Rev. D **55** (1997) 5520, `hep-ph/9610544`.

[5] C. Lester and D. Summers, "Measuring masses of semiinvisibly decaying particles pair produced at hadron colliders," Phys. Lett. B **463** (1999) 99, `hep-ph/9906349`.

[6] W. Cho, K. Choi, Y. Kim, and C. Park, "Measuring superparticle masses at hadron collider using the transverse mass kink," JHEP **0802** (2008) 035, arXiv:0711.4526.

[7] A. Krislock and N. Krislock (2014–), debinning, https://debinning.hepforge.org.

[8] R. Chartrand, "Numerical Differentiation of Noisy, Nonsmooth Data," ISRN Appl. Math. **2011** (2011) 164564.

[9] C. Vogel and M. Oman, "Iterative Methods for Total Variation Denoising," SIAM J. Sci. Comput. **17**, 1 (1996) 227.

[10] E. Weisstein, "Delta Function," From MathWorld — A Wolfram Web Resource, http://mathworld.wolfram.com/DeltaFunction.html.

[11] E. Jones, T. Oliphant, P. Peterson, *et al.*, "SciPy: Open source scientific tools for Python," (2001–), http://www.scipy.org/.

[12] A. H. Baker, E. R. Jessup, and T. Manteuffel, "A Technique for Accelerating the Convergence of Restarted GMRES," SIAM Journal on Matrix Analysis and Applications **26**, 4 (2005) 962.

[13] D. Curtin, "Mixing It Up With MT2: Unbiased Mass Measurements at Hadron Colliders," Phys. Rev. D **85** (2012) 075004, arXiv:1112.1095.

[14] R. Arnowitt, *et al.*, "Determining the Dark Matter Relic Density in the mSUGRA (˜X0(1))-˜tau Co-Annhiliation Region at the LHC," Phys. Rev. Lett. **100** (2008) 231802, arXiv:0802.2968.

[15] B. Dutta, *et al.*, "Supersymmetry Signals of Supercritical String Cosmology at the Large Hadron Collider," Phys. Rev. D **79** (2009) 055002, arXiv:0808.1372.

[16] B. Dutta, T. Kamon, A. Krislock, N. Kolev, and Y. Oh, "Determination of Non-Universal Supergravity Models at the Large Hadron Collider," Phys. Rev. D **82** (2010) 115009, arXiv:1008.3380.

[17] B. Dutta, T. Kamon, N. Kolev, and A. Krislock, "Bi-Event Subtraction Technique at Hadron Colliders," Phys. Lett. B **703** (2011) 475, arXiv:1104.2508.

[18] B. Dutta, T. Kamon, A. Krislock, K. Sinha, and K. Wang, "Diagnosis of Supersymmetry Breaking Mediation Schemes by Mass Reconstruction at the LHC," Phys. Rev. D **85** (2012) 115007, arXiv:1112.3966.

[19] R. Allahverdi, B. Dutta, T. Kamon, and A. Krislock, "Lepton Flavor Violation at the Large Hadron Collider," Phys. Rev. D **86** (2012) 015026, arXiv:1203.3276.

[20] G. Aad *et al.* (ATLAS Collaboration), "Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC," Phys. Lett. B **716**, 1 (2012) 1.

[21] S. Chatrchyan *et al.* (CMS Collaboration), "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC," Phys. Lett. B **716**, 1 (2012) 30.

[22] C. Weniger, "A Tentative Gamma-Ray Line from Dark Matter Annihilation at the Fermi Large Area Telescope," JCAP **1208** (2012) 007, arXiv:1204.2797.

[23] A. Rajaraman, T. Tait, and D. Whiteson, "Two Lines or Not Two Lines? That is the Question of Gamma Ray Spectra," JCAP **1209** (2012) 003, arXiv:1205.4723.

[24] D. Tovey, "Measuring the SUSY mass scale at the LHC," Phys. Lett. B **498** (2001) 1, hep-ph/0006276.

[25] K. Agashe, R. Franceschini, and D. Kim, "A simple, yet subtle 'invariance' of two-body decay kinematics," Phys. Rev. D **88** (2013) 057701, arXiv:1209.0772.

[26] K. Agashe, R. Franceschini, D. Kim, and K. Wardlow, "Using Energy Peaks to Count Dark Matter Particles in Decays," Phys. Dark Univ. **2** (2013) 72, arXiv:1212.5230.

[27] K. Agashe, R. Franceschini, and D. Kim, "Using Energy Peaks to Measure New Particle Masses," arXiv:1309.4776.

[28] E. Weisstein, "Heaviside Step Function," From MathWorld — A Wolfram Web Resource, http://mathworld.wolfram.com/HeavisideStepFunction.html.

A Data Generation

We generate our test data using a simple Monte Carlo method, much like the one used to generate the binfull OPDF in Sec. 3. For any UPDF, defined on any range of x values, the data generation is as follows. First, an array of x values spanning the range is generated, with a small (0.01) step between adjacent values. The UPDF is then evaluated for each of these x values to form an array of y values. Next, the CDF is constructed from the y values using the A operator given by Eq. (2.14). Thus, $z_{\rm CDF} = Ay$, where, within A, $\Delta_x = 10^{-4}$.

With the CDF generated in this way, we can use it as a Monte Carlo generator of data based upon the UPDF. The generator is the very same as the one described in Sec. 3, except that no smoothing function is applied. This generator can, with infinite statistics, reproduce the shape of the UPDF as given by the x and y arrays.

In this paper, we use the following two phenomenologically inspired UPDFs. For examples of other such UPDFs, please see [7].

A.1 Easy Endpoint

The "easy endpoint" UPDF is inspired by (highly optimistic) endpoint searches of Supersymmetry cascade decays within the context of the Large Hadron Collider or other collider experiment [4]. The easy endpoint UPDF is constructed as a cubic background piece,

$$y_{\rm BG} = p_0(x - p_1)(x - p_2)^2, \tag{A.1}$$

plus a triangular shaped signal piece,

$$y_{\rm signal} = \begin{cases} p_4 x - p_5, & \text{if } x < p_3 \\ m^\dagger(x - p_6), & \text{if } x \geq p_3 \end{cases}, \tag{A.2}$$

where p_i denote the seven parameters of the function, and $m^\dagger = -\frac{p_3 p_4 - p_5}{p_6 - p_3}$. These pieces are added together to form one function, but only if they are each positive. Thus, the overall easy endpoint UPDF can be written as

$$y = y_{\rm signal} H(y_{\rm signal}) + y_{\rm BG} H(y_{\rm BG}), \tag{A.3}$$

where $H(y)$ is the Heaviside step function [28]. The set of seven parameters we use for the easy endpoint are

$$p = (0.00006,\ 40.,\ 200.,\ 110.,\ 1.5,\ 50.,\ 140.). \tag{A.4}$$

We would like to emphasize that the parameter of interest is the location of the "endpoint," which is the maximum x-value where the signal piece meets the background. This occurs at $x = p_6 = 140$.

A.2 The Line

The second UPDF we use in this paper is "the line," which is inspired both by the discovery of the Higgs boson at the LHC [20, 21] and the recent apparent gamma ray line signal within the Fermi-LAT data [22]. The line UPDF is constructed as an exponential background plus a Gaussian peak signal:

$$y = p_0 \exp(-p_1 x) + p_2 \exp\left(\frac{-(x-p_3)^2}{2p_4^2}\right). \tag{A.5}$$

The set of parameters we use for the line are

$$p = (1000.,\ 0.02,\ 60.,\ 130.,\ 4.). \tag{A.6}$$

In this case, the parameter of interest is the peak location of the Gaussian, which is $x = p_3 = 130$.