UNIVERSITY OF CAMBRIDGE

S1: Statistical Methods Coursework

Yuanzhen Zhao (yz929)

Department of Physics, University of Cambridge

December 16, 2024

Contents

Word Counts: 1xxx

1 Introduction

This coursework aims to compare the capability of multi-dimensional likelihood fit and weighted fit using sWeight by constructing and fitting two-dimensional data to the signal and background.

2 Mathematical Derivations

Proof. The definite integral of probability density function (p.d.f.) along the entire axis should be unity, i.e.

$$\int_{-\infty}^{\infty} p(X;\mu,\sigma,\beta,m)\,\mathrm{d}X = 1$$

Now I change the variable from X to Z by normalisation, i.e. $Z = (X-\mu)/\sigma$, where the equation becomes

$$\sigma\int_{-\infty}^{\infty} p(Z;\mu,\sigma,\beta,m)\,\mathrm{d}Z = 1$$

Plugging in the p.d.f. gives

$$\sigma N\left[\int_{-\beta}^{\infty} e^{-Z^2/2}\,\mathrm{d}Z + \int_{-\infty}^{-\beta}\left(\frac{m}{\beta}\right)^m e^{-\beta^2/2}\left(\frac{m}{\beta}-\beta-Z\right)^{-m}\mathrm{d}Z\right] = 1$$

For the first integral, I notice that $\mathrm{erf}(z) = \frac{2}{\sqrt{\pi}}\int_0^z e^{-t^2}\,\mathrm{d}t$ could be useful, and its parity is odd.

$$\begin{aligned}
\mathrm{erf}\left(\frac{z}{\sqrt{2}}\right) &= \frac{2}{\sqrt{\pi}}\int_0^{z/\sqrt{2}} e^{-t^2}\,\mathrm{d}t \\
&= \frac{2}{\sqrt{2\pi}}\int_0^{z} e^{-t'^2/2}\,\mathrm{d}t' \quad t' = \sqrt{2}t \quad \text{(integration by substitution)} \\
&= \frac{2}{\sqrt{2\pi}}\int_0^{z} e^{-t^2/2}\,\mathrm{d}t \quad t' \to t \quad \text{(dummy indices)}
\end{aligned}$$

$$\begin{aligned}
\mathrm{erf}(-z) &= \frac{2}{\sqrt{\pi}}\int_0^{-z} e^{-t^2}\,\mathrm{d}t \\
&= -\frac{2}{\sqrt{\pi}}\int_{-z}^{0} e^{-t^2}\,\mathrm{d}t \quad \text{(swapping the limits)} \\
&= -\frac{2}{\sqrt{\pi}}\int_0^{z} e^{-t^2}\,\mathrm{d}t \quad \text{(even function)} \\
&= -\mathrm{erf}(z)
\end{aligned}$$

Now substituting $z = -\beta$ gives

$$\mathrm{erf}\left(\frac{-\beta}{\sqrt{2}}\right) = \sqrt{\frac{2}{\pi}}\int_0^{-\beta} e^{-t^2/2}\,\mathrm{d}t$$

Based on the identity $\int_{-\infty}^{\infty} e^{-ax^2}\,\mathrm{d}x = \sqrt{\pi/a}$, as well as the integrand is an even function, I could write

$$\int_0^{\infty} e^{-x^2/2}\,\mathrm{d}x = \sqrt{\frac{\pi}{2}}$$

Hence the first integral could evaluated via

$$\begin{aligned}\int_{-\beta}^{\infty} e^{-Z^2/2}\,\mathrm{d}Z &= \int_0^{\infty} e^{-Z^2/2}\,\mathrm{d}Z - \int_0^{-\beta} e^{-Z^2/2}\,\mathrm{d}Z \\ &= \sqrt{\frac{\pi}{2}} - \sqrt{\frac{\pi}{2}}\mathrm{erf}\left(\frac{-\beta}{\sqrt{2}}\right) \\ &= \sqrt{\frac{\pi}{2}}\left[1 + \mathrm{erf}\left(\frac{\beta}{\sqrt{2}}\right)\right]\end{aligned}$$

Recalling the Gaussian c.d.f. $\Phi(z) = \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{z}{\sqrt{2}}\right)\right]$, I could transform the first integral into

$$\int_{-\beta}^{\infty} e^{-Z^2/2}\,\mathrm{d}Z = \sqrt{2\pi}\Phi(\beta)$$

The second integral could be integrated ordinarily.

$$\begin{aligned}\int_{-\infty}^{-\beta} \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2}\left(\frac{m}{\beta} - \beta - Z\right)^{-m}\,\mathrm{d}Z &= \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2}\left[\frac{1}{m-1}\left(\frac{m}{\beta} - \beta - Z\right)^{-m+1}\right]_{Z=-\infty}^{Z=-\beta} \\ &= \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2}\frac{1}{m-1}\left(\frac{m}{\beta}\right)^{-m+1} - 0 \\ &= \frac{m}{\beta(m-1)}e^{-\beta^2/2}\end{aligned}$$

Hence, the inverse of the normalisation constant equals

$$N^{-1} = \sigma\left[\sqrt{2\pi}\Phi(\beta) + \frac{m}{\beta(m-1)}e^{-\beta^2/2}\right]$$

■

3 Implementation of Probability Distributions

I started with defining these functions using SCIPY modules, but I changed them in NUMBA-STATS lately and imported these modules `crystalball`, `truncexpon`, `uniform`, `truncnorm`, for significantly better performances in time. These functions are defined with complete form in the module `dists.py`.

- $g_s(X)$: As there is no ready-made 'truncated crystal ball' function, I had to truncate all the inputs between 0 to 5 first using `numpy.clip` function. The p.d.f. could be evaluated by calling `crystallball.pdf` times the normalisation constant, which is the reciprocal of the

difference `crystallball.cdf` between $x = 0$ and $x = 5$. All the functions above share the same parameters of **beta** setting to β, **m** setting to m, **loc** setting to μ and **scale** setting to σ.

- $h_s(Y)$: Using `truncexpon.pdf` directly with declaring $x_{\min} = 0$ and $x_{\max} = 10$, and **loc** setting to zero, **scale** setting to λ^{-1}. The function `truncexpon.pdf` itself has built in the normalisation process, so the λ prefactor would not need to be specified.
- $g_b(X)$: `uniform.pdf` with declaring $a = 0$ and $w = 5$ would do.
- $h_b(Y)$: `truncnorm.pdf` with declaring $x_{\min} = 0$ and $x_{\max} = 10$, and **loc** setting to μ_b, **scale** setting to σ_b.
- $s(X, Y)$: product of $g_s(X)$ and $h_s(Y)$
- $b(X, Y)$: product of $g_b(X)$ and $h_b(Y)$
- $f(X, Y)$: here I calculated the expression of $fg_s(X)h_s(Y) + (1 - f)g_b(X)h_b(Y)$ whereas substituting with $s(X, Y)$ and $b(X, Y)$ should be equivalent.

The normalisation of the above functions is also double checked by numerically estimating the area under the p.d.f curve within each valid domain, whereas some of them are not precisely equal to unity due to float-point errors. It is still credible to say these functions are normalised within $x \in [0, 5]$ and $y \in [0, 10]$.

4 Visualisation

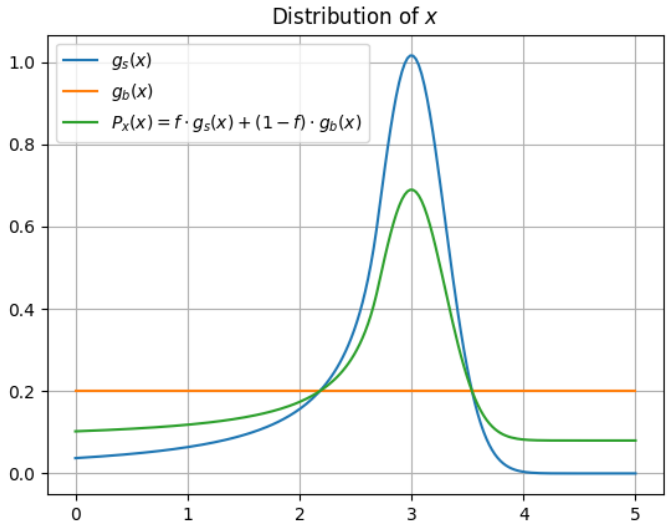

Figure 1: One-dimensional projection in variable X, where g_s is signal model of X, g_b is background model for X and P_x is the X-axis projection of total p.d.f.

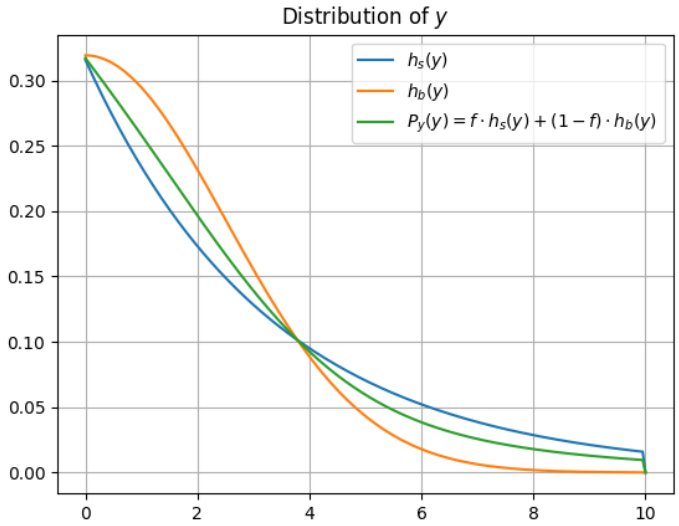

Figure 2: One-dimensional projection in variable Y, where h_s is signal model of Y, h_b is background model for Y and P_y is the Y-axis projection of total p.d.f. **It is noticeable that h_s is truncated a bit earlier than 10**

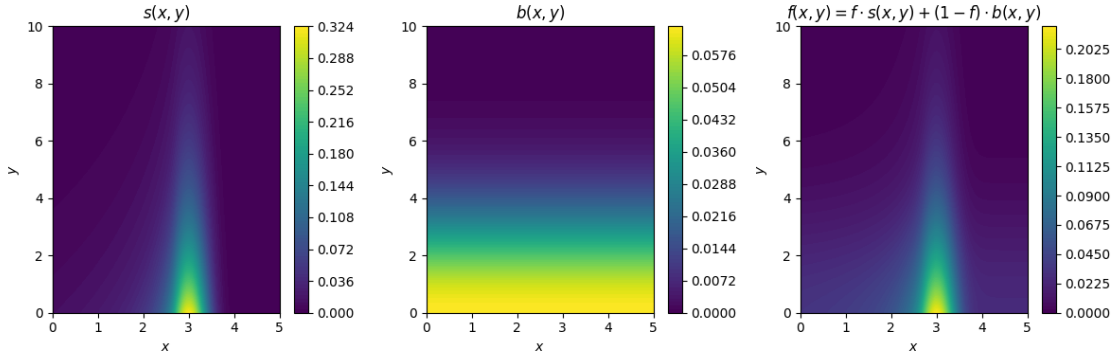

Figure 3: Total p.d.f. with signal and background components

5 Sample Generation and Fitting

There are two methods of generating legitimate samples: **inverse transform sampling** and **accept-reject method**.

5.1 Inverse Transform Sampling

Inverse transform sampling is a particularly useful and time-saving algorithm if the c.d.f. of a distribution is known. The number of signal events and background events could be determined by the ratio f and total number N.

- X_s: As `crystalball` from NUMBA-STATS has no built-in `ppf` function, I had to import `scipy.crystalball` module as substitute. Then uniformly sampling numbers within the c.d.f. of 0 and 5 could technically return a set of c.d.f. within the required domain. Evaluating `ppf`, which is inverse c.d.f., of those values could generate a set of X_s easily for any length.

- X_b: The `uniform` module could not set the limit of generated random variables directly, so I followed the same methods as above.

- Y_s & Y_b: `truncexpon` and `truncnorm` has built-in `rvs` function, which could pick random variables for any declared length with the same parameters as defined in `dists.py`.

Using `numpy.column_stack` could combine X_s and Y_s for signal events and X_b and Y_b for background events, following with `numpy.concatenate` could stack them into one big dataset. As the fitting method would take all the data at once, the sequence of signal events and background events does not matter at all.

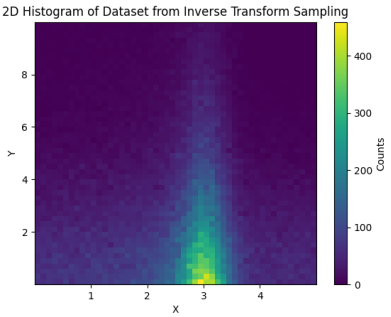

Figure 4: 2D distributions of dataset sampling with inverse transform method

5.2 Accept-Reject Method

Accept-Reject method, however, is a more intuitive and computationally demanding algorithm. At first, I defined a function of finding the maximum of a two-argument function, by finding an additive inverse of the minimum of the opposite function using `brute` function, which could be used to find $f_{\max}$

Then I generated samples uniformly within domains of each axis, for example, $x \in [0, 5]$, $y \in [0, 10]$ and $f \in [0, f_{\max}]$. It is important to calculate the true values of $f(X, Y)$ for all data points. The validity of all data points could be examined by comparing the sample values and true values.

Finally, the subset of valid samples of the first required length could be used as high-statistics samples.

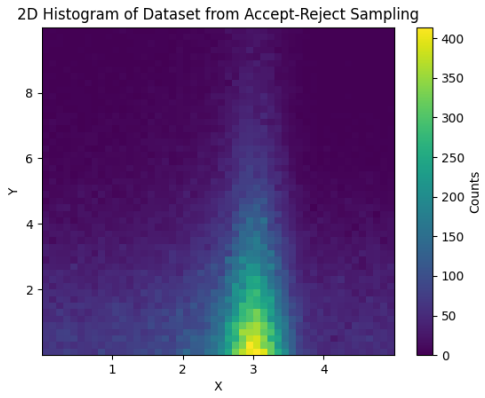

Figure 5: 2D distributions of dataset sampling with accept-reject method

5.3 Extended Maximum Likelihood

To fit these artificial datasets onto our model, I used `ExtendedUnbinnedNLL` in IMINUIT.COST to perform an Extended Maximum Likelihood (EML) fit.

In real experimental scenarios, for example, in particle physics, the total number of observed events often varies due to statistical fluctuations or systematic effects. EML methods account for the variability in the total event count with a Poisson variation. In contrast, a non-extended maximum likelihood assumes a fixed number of events and focuses solely on the data distribution. This assumption can lead to biased parameter estimates, particularly when the event count itself carries significant physical meaning.

For an EML fit, it is crucial to rewrite the p.d.f. to explicitly account for the dependence on the total number of samples, as the fit includes this information. Since `ExtendedUnbinnedNLL` only accepts one argument as input for data, each data point must be unpacked into its respective x- and y- values. The initial guesses of parameters for fit should closely reflect the true values, as these are the most reliable starting points. However, the initial guess for the total event counts N should include a Poisson variation to better represent the inherent statistical fluctuations in real-world data.

5.4 Performance Comparison

The computational efficiency of different operations was evaluated in terms of execution times relative to a benchmark. The benchmark time of 1.35 milliseconds served as a standard for comparison. The inverse transform sampling method demonstrated a moderate computational overhead, requiring 9.01 milliseconds, approximately 6.68 times the benchmark time. In contrast, the accept-reject method was significantly slower, taking 142 milliseconds, approximately 105 times the benchmark time.

Furthermore, the dataset-fitting process exhibited substantial time costs. Fitting the inverse transform dataset took 2.72 seconds, which is about 2014 times the benchmark. Similarly, the fitting time for the accept-reject dataset was slightly faster at 2.58 seconds, about 1916 times the benchmark.

These numbers illustrate the higher efficiency of the inverse transform method during sampling but also highlight the substantial costs during the fitting stage.

6 Parametric bootstrapping

Parametric bootstrapping is a statistical method used to assess the robustness of a model by generating pseudo-experiments (or 'toys') from a fitted model based on its parameters. The process involves fitting the data to a nominal model, generating a large ensemble of pseudo-datasets from the fitted parameters with Poisson variation for extended fits and re-fitting these datasets to observe behaviours of parameters of interest. Through this procedure, the bias and uncertainty of the estimation methods could be assessed.

This method is particularly advantageous when the data are known to follow a specified probabilistic model, while non-parametric bootstrapping relies on resampling directly from the observed data.

For each sample size, I generated synthetic datasets of the specified lengths using the previously fitted parameter estimates, obtained via inverse transform sampling. These datasets were then

fitted into the extended model with the previously fitted parameters serving as initial guesses. This process was repeated 400 times for each sample size.

For every synthetic dataset, I refitted all the model parameters and extracted the fitted values of λ and their associated errors. To handle potential invalid results (e.g., when $\beta = 0$ causes invalid `NaN` values), I computed the `nanmean` of the λ values and stored the results in a separate list.

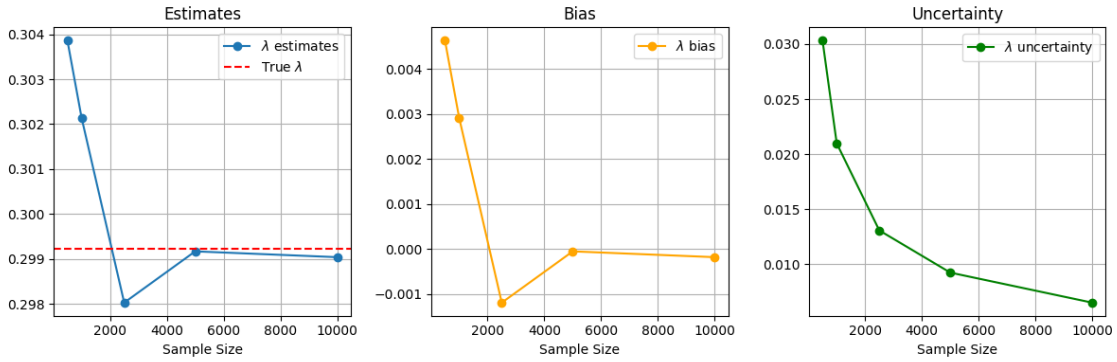

Figure 6: Estimates, Bias, and Uncertainty of λ using Parametric Bootstrapping

7 sWeights

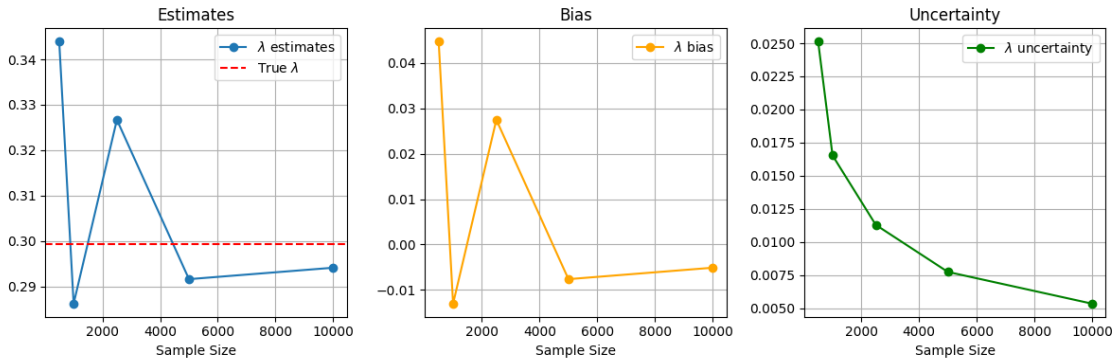

Figure 7: Estimates, Bias, and Uncertainty of λ using sWeights