UNIVERSITY OF CAMBRIDGE

S1: Statistical Methods Coursework Report

Yuanzhen Zhao (yz929)

Department of Physics, University of Cambridge

December 18, 2024

Contents

Word Counts: 2262

1 Introduction

This coursework aims to evaluate and compare statistical methods for estimating model parameters in multi-dimensional datasets, especially focusing on parametric bootstrapping and the sWeights method. The analysis involved a two-dimensional model comprising signal and background distributions with explicit probability density functions (p.d.f.s) for each component. The comparison centres on the performance of these methods in generating artificial datasets using different sampling approaches and fitting them with extended unbinned models.

2 Mathematical Derivations

Proof. The definite integral of 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

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

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

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

Recalling the Gaussian c.d.f. $\Phi(z) = \frac{1}{2}\left[1 + \operatorname{erf}\left(\frac{z}{\sqrt{2}}\right)\right]$, 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

These functions were defined by SCIPY modules at first, 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 been 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

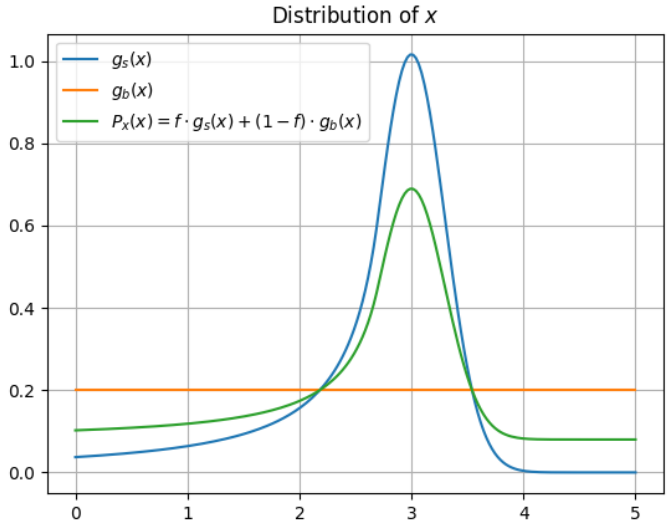

Figure 1: One-dimensional projection in variable X, where g_s is signal model of X, g_b is background model for X and P_x is the X-axis projection of total p.d.f.

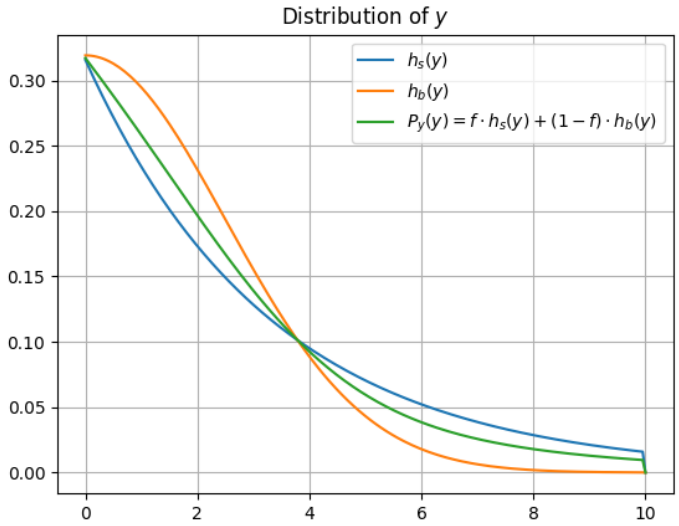

Figure 2: One-dimensional projection in variable Y, where h_s is signal model of Y, h_b is background model for Y and P_y is the Y-axis projection of total p.d.f. **Notably h_s truncates a bit earlier than 10, arguably arising from the floating-point error from** NUMBA-STATS.

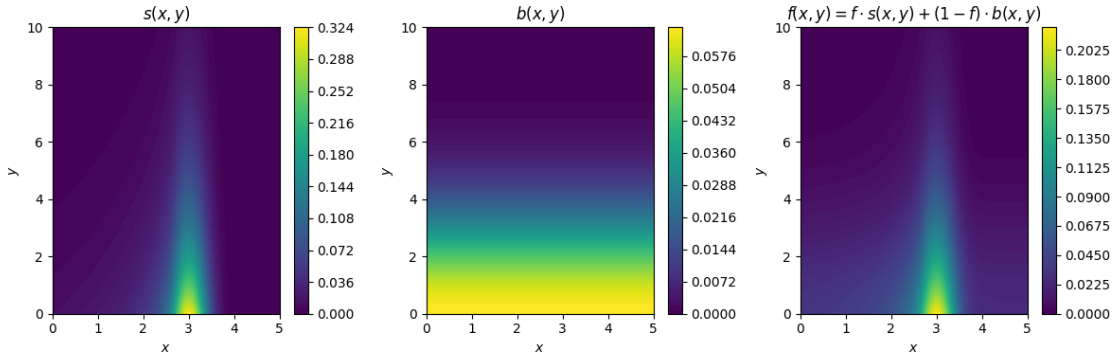

Figure 3: Total p.d.f. with signal and background components

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 a 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` have built `rvs` function in, 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, followed 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.

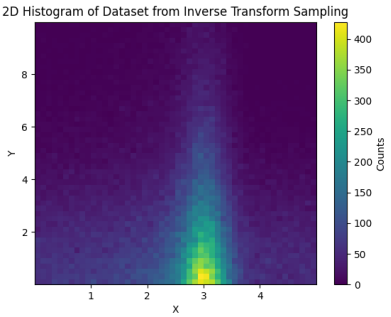

Figure 4: 2D distributions of dataset sampling with the inverse transform method

5.2 Accept-Reject Method

Accept-Reject method, however, is a more intuitive and computationally demanding algorithm. At first, I defined a function for 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.

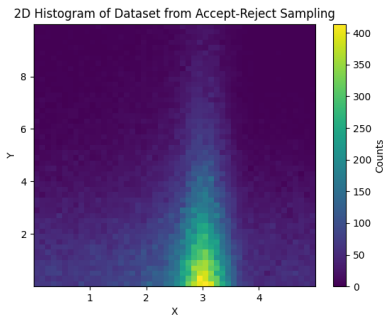

Figure 5: 2D distributions of dataset sampling with the accept-reject method

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 for 142 milliseconds, approximately 105 times the benchmark time. The difference arises because inverse transform sampling simply generates 100,000 data points, whereas the accept-reject method has to produce 20 times more samples to ensure 100,000 valid points for a dataset.

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.

Metric	Scipy Time (s)	Scipy Ratio	Numba-stats Time (s)	Numba-stats Ratio
Benchmark Time	0.001326	-	0.001399	-
Inverse Transform Sampling	0.018001	13.58	0.010626	7.60
Accept-Reject Sampling	0.585360	441.59	0.163106	116.59
Fitting (Inverse Transform)	10.398362	7844.42	3.803853	2718.99
Fitting (Accept-Reject)	11.869181	8953.99	3.347707	2392.94

Table 1: Comparison of SCIPY and NUMBA-STATS

A major improvement in computational efficiency was achieved by using NUMBA-STATS instead of SCIPY. The sampling time using SCIPY takes approximately 14 times the benchmark for the inverse transform method and 441 times the benchmark for the accept-reject method. For the fitting times, it takes SCIPY to fit both datasets about 12 seconds, which is approximately 9000 times the benchmark. So there is a 4-times improvement on average across all the operations. These functions are rewritten with faster implementations even without introducing auto-parallelisation in this coursework.

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 refitted into the extended model, using the prior parameter estimates as initial guesses. This process was repeated 500 times for each sample size.

For every synthetic dataset, I re-estimated all 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 `numpy.nanmean` of the λ values and stored the results in a separate list.

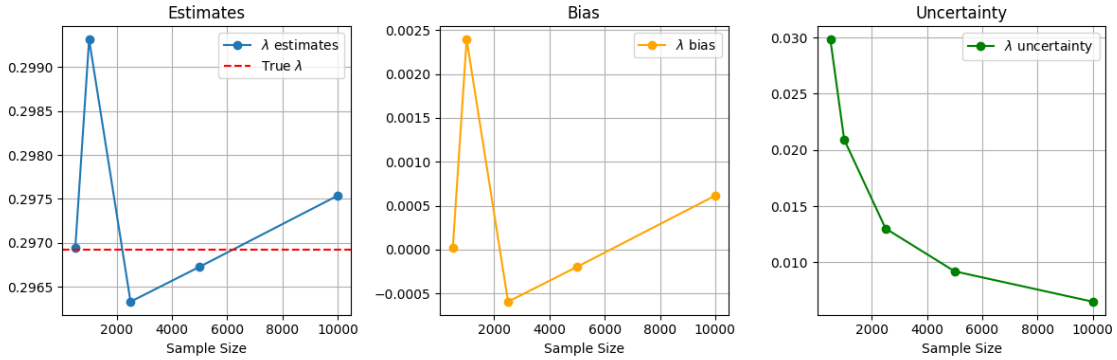

Figure 6: Estimates, Bias, and Uncertainty of λ using Parametric Bootstrapping

The results show that the estimates of λ are highly accurate with true value. As the sample size increases, the estimates demonstrate convergence toward the true value, with bias fluctuating near zero on the scale of 10^{-4}. Notably, the estimates decrease abruptly when the sample size rises from 1000 to 2500, after which they gradually converge.

This behaviour aligns with the Central Limit Theorem, as these events are independent and identically distributed (i.i.d.). Consequently, their variance of decay constant decreases as the sample size grows. Therefore, the uncertainties, which are equal to the square root of variances, should be theoretically inversely proportional to the square root of the total number of events.

$$\sigma \propto \frac{1}{\sqrt{N}}$$

To statistically validate this relationship, it is common practice to take the logarithms of σ (uncertainty) and N (sample size). It would preserve the information while linearising the relationship. By applying `scipy.stats.linregress`, it is feasible to perform a linear regression between $\log(\sigma)$ and $\log(N)$.

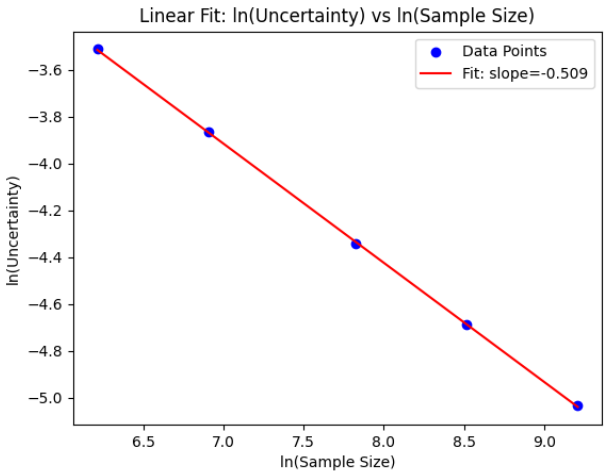

Figure 7: Linear Regression of log(σ) against log(N) for Parametric Bootstrapping

The fitted slope from the linear regression is -0.509, which is very close to -0.5, validating the inverse square root dependence. The resulting regression function is

$$\sigma = 0.702 \cdot N^{-0.509}$$

7 sWeights

sWeights is a specialised weighted method frequently used in high-energy physics to separate signal and background distributions multi-dimensionally[1]. This method allows for fitting data in one dimension and projecting the signal component onto the other dimension by estimating individual weights. By applying weights, it becomes possible to separate signal contributions from overall measurements in an easier dimension, thus avoiding computationally expensive multi-dimensional regressions.

For each sample size, I generated a synthetic dataset following the same procedure as parametric bootstrapping except for no repetition. I performed an EML fit on the X-values with the knowledge of p.d.f. on X. Using the fitted signal and background components, I applied the `get_weight` method on a `SWeight` object to estimate weights for each point, determining whether it is signal or background.

Next, I calculated the signal weights of Y and the bin edges using histograms with 200 bins, as the binned fit is unbiased when the bin content is generated from p.d.f. instead of taking the centre density as average[2]. The bin centres were approximated as the average of adjacent bin edges.

Using the binned likelihood method, I fit the decay constant λ in c.d.f. on Y, as binned methods from IMINUIT require the c.d.f. as an argument. The binned fit proved highly efficient, taking only 0.6 seconds to produce estimates for all sample sizes.

The resulting λ values and their associated uncertainties are shown in the following plots.

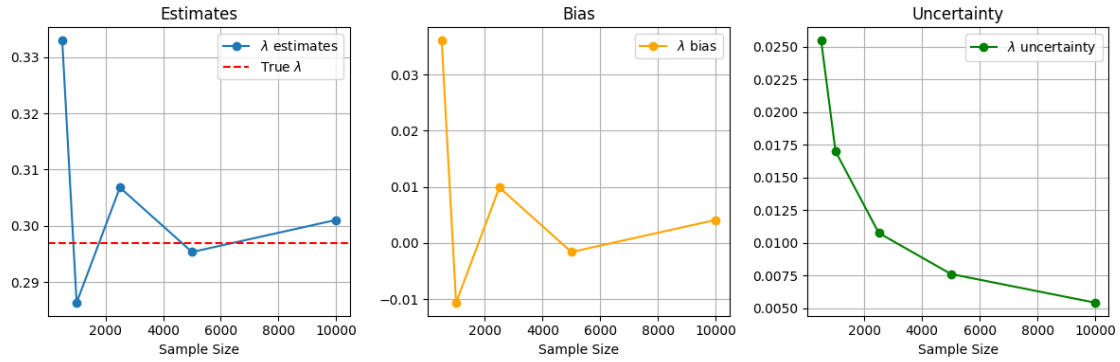

Figure 8: Estimates, Bias, and Uncertainty of λ using sWeights

Binning some events in the fit may lead to a loss of information, as the binning process averages event densities within each bin. Despite that, this issue could be mitigated by reducing the bin width sufficiently[2].

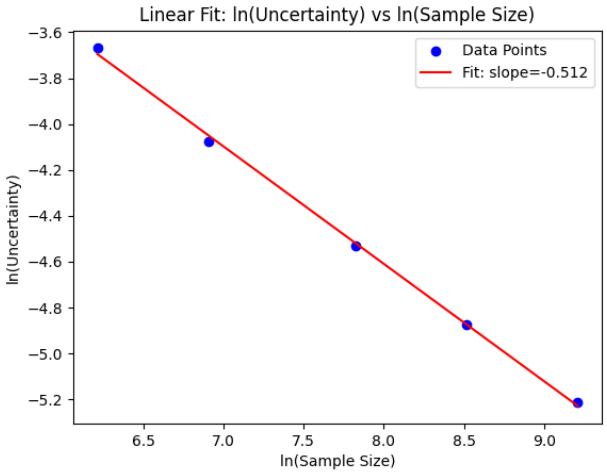

Figure 9: Linear Regression of $\log(\sigma)$ against log(N) for sWeights

Applying the same linear fit to the logarithms of uncertainties and sample size yields the relationship:

$$\sigma = 0.599 \cdot N^{-0.512}$$

which is consistent with the expected inverse square root dependence as well.

When comparing the bias and uncertainty of the above two methods, **parametric bootstrapping** exhibits smaller bias but larger uncertainty, indicating high accuracy with low precision, whereas **sWeights** shows larger bias but smaller uncertainty, reflecting high precision with low accuracy.

8 Conclusion

In this analysis, I compared parametric bootstrapping and the sWeights method in terms of their performance and efficiency. Both methods have distinct advantages and drawbacks depending on the scenario.

Parametric bootstrapping provides highly accurate results by reducing both bias and uncer-

tainty through repeated pseudo-experiments. Nonetheless, it is computationally expensive and time-consuming, making it less practical for limited computational resources. It is best suited for situations where precision and accuracy are critical, and sufficient computational ability is available.

The **sWeights** method is computationally efficient, producing results much faster compared to bootstrapping. It tends to offer smaller uncertainties with larger systematic biases. Reducing the bin width could mitigate uncertainties, but there is no straightforward way to eliminate the bias effectively. This method works particularly well when an explicit one-dimensional p.d.f. is available and the variables are uncorrelated.

In this case, the **sWeights** method is a more appropriate choice, since p.d.f. on X is clearly well-defined and there is no correlation between X and Y. Its computational efficiency and simplicity provide an influential advantage over parametric bootstrapping.

To summarise, when computational resources are limited and a one-dimensional p.d.f. is easy to model, the **sWeights** method is preferred; when higher accuracy and reduced bias are prior with no constraint on computational power, **parametric bootstrapping** is favoured.

Reference

[1] Hans Dembinski et al. "Custom Orthogonal Weight functions (COWs) for event classification". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 1040 (2022), p. 167270. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2022.167270. URL: https://www.sciencedirect.com/science/article/pii/S0168900222006076.

[2] *Binned vs. unbinned fits — iminuit 2.30.2 compiled with ROOT-v6-25-02-9213-g754d22635f documentation*. Scikit-hep.org, 2019. URL: https://scikit-hep.org/iminuit/notebooks/binned_vs_unbinned.html.

A Appendix

A.1 Use of Auto-Generation Tools

I used ChatGPT to assist with the proofreading process and to improve the formatting of my report written in LaTeX.

Proofreading: reviewing grammar and ensuring clarity in sentences.

LaTeX formatting: aligning multiple equations, adding texts within equations and asking how to generate a table to organise my timings for performance comparison.