UNIVERSITY OF CAMBRIDGE

S1 Statistical Methods Coursework

xl628

Department of Physics, University of Cambridge

December 18, 2024

Document Statistics:
Words in text: 1590
Words in headers: 79
Words outside text (captions, etc.): 178
Number of headers: 25
Number of floats/tables/figures: 8
Number of math inlines: 116
Number of math displayed: 20

1 Crystal Ball Normalization

The Crystal Ball probability distribution is defined as:

$$p(X;\mu,\sigma,\beta,m) = N \cdot \begin{cases} e^{-Z^2/2}, & \text{for } Z > -\beta \\ \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \left(\frac{m}{\beta} - \beta - Z\right)^{-m}, & \text{for } Z \leq -\beta \end{cases}$$

where $Z = \frac{X-\mu}{\sigma}$ is the standard normal transformation for a random variable X at location, μ, with scale, σ. The distribution is only valid when $\beta > 0$ and $m > 1$.

Proof By definition, the normalization requires:

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

Rewrite the normalization in terms of Z. Since $dX = \sigma\, dZ$:

$$1 = \int_{-\infty}^{\infty} N \cdot p_z(Z)\sigma\, dZ,$$

This gives:

$$1 = N\sigma \left[\int_{-\infty}^{-\beta} \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \left(\frac{m}{\beta} - \beta - Z\right)^{-m} dZ \; + \; \int_{-\beta}^{\infty} e^{-Z^2/2} dZ \right] = N\sigma(I_1 + I_2). \quad (1)$$

Consider the integral:

$$I_1 = \int_{-\infty}^{-\beta} \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \left(\frac{m}{\beta} - \beta - Z\right)^{-m} dZ.$$

Factor out the constants:

$$I_1 = \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \int_{-\infty}^{-\beta} \left(\frac{m}{\beta} - \beta - Z\right)^{-m} dZ. \tag{2}$$

Use the substitution:

$$t = \frac{m}{\beta} - \beta - Z \implies dZ = -dt.$$

When $Z = -\infty$, we have $t \to \infty$. When $Z = -\beta$, we get $t = \frac{m}{\beta} - \beta - (-\beta) = \frac{m}{\beta}$.
So the integral limits transform as $Z : -\infty \to -\beta$ to $t : \infty \to \frac{m}{\beta}$:

$$I_1 = \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \int_{\infty}^{m/\beta} t^{-m}(-dt) = \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \int_{m/\beta}^{\infty} t^{-m} dt.$$

The integral $\int t^{-m} dt = \frac{t^{-m+1}}{-m+1}$ for $m > 1$. Thus:

$$I_1 = \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \left(\frac{(m/\beta)^{-m+1}}{m-1}\right) = e^{-\beta^2/2} \frac{m}{\beta(m-1)}.$$

Now consider the second integral:

$$I_2 = \int_{-\beta}^{\infty} e^{-Z^2/2} dZ. \tag{3}$$

This integral can be expressed in terms of the standard normal CDF $\Phi(z)$, where

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{-t^2/2} dt.$$

We note:

$$\int_{-\beta}^{\infty} e^{-Z^2/2} dZ = \sqrt{2\pi}\big[\Phi(\infty) - \Phi(-\beta)\big] = \sqrt{2\pi}\,[1 - \Phi(-\beta)]\,.$$

Therefore:

$$I_2 = \sqrt{2\pi}\,[1 - \Phi(-\beta)] = \sqrt{2\pi}\Phi(\beta).$$

Substituting I_1 (Eq. 2) and I_2 (Eq. 3) back into Eq. 1, we have:

$$1 = N\sigma[I_1 + I_2] = N\sigma \left[\frac{m}{\beta(m-1)} e^{-\beta^2/2} + \sqrt{2\pi}\,\Phi(\beta)\right].$$

Thus, we have proven that:

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

2 Module Development

Several classes implementing the required distributions are provided in `src/pdfs_numba_stats.py`.

Throughout the project, the computational efficiency of probability density functions (PDFs) and cumulative distribution functions (CDFs) has been a critical factor, significantly influencing the speed of sample generation and optimization. Various measures have been implemented to enhance this efficiency:

Library Choice We employ `numba-stats` [4] instead of `scipy.stats` due to its substantial performance improvements. It enables faster computations, conveniently supports direct operations on NumPy arrays, and eliminates the need for manual function vectorization.

Class-Based Programming Class-based programming is utilized throughout the development process. Parameters and methods specific to each distribution, such as `pdf`, `cdf`, and their truncated versions, are encapsulated within the classes.

Class initialization allows for one-time parameter passing and pre-computation of values, such as normalization constants for truncated functions. This approach reduces redundant calculations during method calls, leading to significant performance improvements.

2.1 Normalization Test

To ensure the accuracy of the defined distributions, we verified their normalization numerically in `notebooks/b_c_norm_visualization.ipynb`.

The signal-background model is defined as follows:

$$f(X,Y) = fg_s(X)h_s(Y) + (1-f)g_b(X)h_b(Y).$$

Using `integrate.quad` and `integrate.dblquad`, we validated the normalization of the following components:

- $g_s(X)$ over $X \in [0,5]$,
- $g_b(X)$ over $X \in [0,5]$,
- $h_s(Y)$ over $Y \in [0,10]$,
- $h_b(Y)$ over $Y \in [0,10]$,
- $s(X,Y) = g_s(X)h_s(Y)$ over $X \in [0,5]$ and $Y \in [0,10]$,
- $b(X,Y) = g_b(X)h_b(Y)$ over $X \in [0,5]$ and $Y \in [0,10]$,
- $f(X,Y) = fg_s(X)h_s(Y) + (1-f)g_b(X)h_b(Y)$ over $X \in [0,5]$ and $Y \in [0,10]$,

The parameters used are:

```python
signal_params = {
   'mu': 3,
   'sigma': 0.3,
   'beta': 1,
   'm': 1.4,
   'decay_rate': 0.3
}
background_params = {
   'mu_bg': 0,
   'sigma_bg': 2.5
}

f = 0.6 # Signal fraction
x_min, x_max = 0, 5
y_min, y_max = 0, 10
```

The following output confirms proper normalization of the distributions:

```
1D Normalizations:
- Signal X distribution 1.000
- Background X distribution 1.000
- Signal Y distribution 1.000
- Background Y distribution 1.000

2D Normalizations:
- Total distribution    1.000
- Signal XY distribution 1.000
- Background XY distribution 1.000
```

3 Distribution Visualization

For the signal-background model:

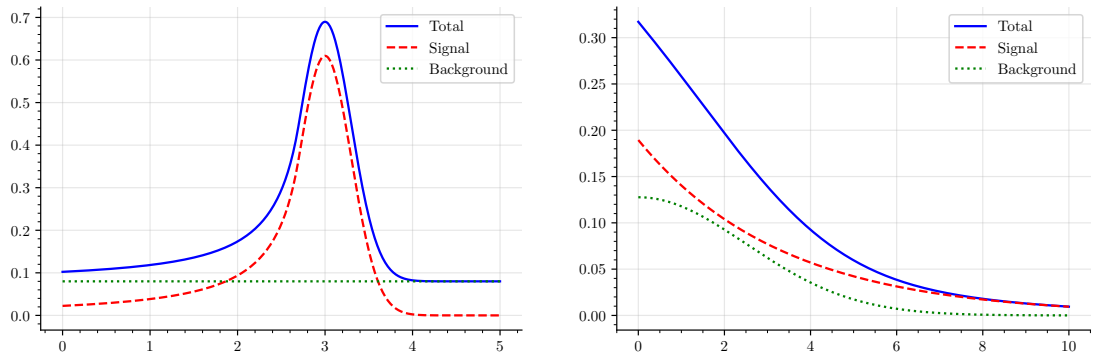
$$f(X,Y) = fg_s(X)h_s(Y) + (1-f)g_b(X)h_b(Y),$$

we can integrate out X or Y to obtain marginal distributions. Given the separability of X and Y and the rectangular integration range, the one-dimensional projections are:

$$g(X) = \int f(X,Y)dY = fg_s(X) + (1-f)g_b(X),$$

$$h(Y) = \int f(X,Y)dX = fh_s(Y) + (1-f)h_b(Y).$$

Thus, we can plot the one-dimensional projections of these distributions in both the variables X and Y in Fig. 1 and the joint distribution in Fig. 2.



(a) (b)

Figure 1: (a) Probability density function $g(X) = fg_s(X)+(1-f)g_b(X)$, signal component $fg_s(X)$, and background component $(1-f)g_b(X)$ (marginal distributions in X). (b) Probability density function $h(Y) = fh_s(Y) + (1-f)h_b(Y)$, signal component $fh_s(Y)$, and background component $(1-f)h_b(Y)$ (marginal distributions in Y).

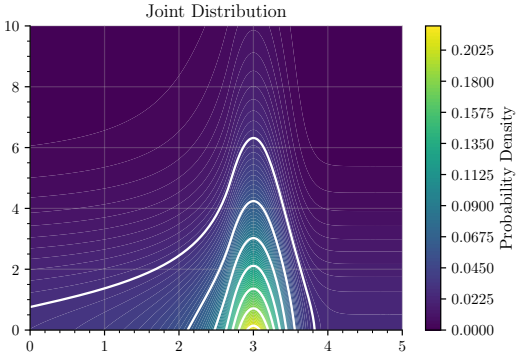


Figure 2: Two-dimensional plot of the joint probability density.

4 Sampling and Fitting

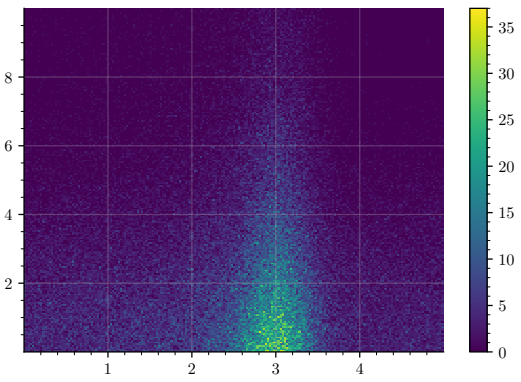


Figure 3: Data generated using the Accept-Reject Method.

4.1 Accept-Reject Method

Since the percentage point function (PPF) is not easily obtainable in this case, we implement the accept-reject method to generate samples from the model.

The maximum of the probability density function (PDF) can sometimes be derived analytically. Because X and Y are separable, and with respect to X, both the signal and background (uniform) components achieve their maximum at the same X value. When $\mu_b = 0$, the signal and background components with respect to Y reach their maximum at $Y = 0$.

For the given parameters where $\mu_b = 0$, the analytical maximum occurs at $(3.0, 0.0)$.

The implementation is detailed in the function `accept_reject_sampling_2d` from `notebooks/d_gen_sample_estimate.ipynb`. To accelerate the process, candidate points are generated in batches, leveraging vectorized computations for efficiency.

The generated data is stored in `notebooks/samples.npy`, and the corresponding visualization is shown in Fig. 3.

4.2 Extended Maximum Likelihood Estimate (EMLE)

We implemented both unbinned and binned fits in `notebooks/d_gen_sample_estimate.ipynb` using the `iminuit` library[1].

For consistency, the same starting point and 29 bins were used for the binned fit. The parameter estimates and their estimated uncertainties (Hesse errors) from one run are presented in Table 1.

Table 1: Comparison of unbinned and binned EMLE with estimated values and Hesse errors.

Name	Value	Hesse Error
Unbinned EMLE		
N	100.00×10^3	0.32×10^3
μ	2.9986	0.0028
σ	0.3022	0.0026
β	0.969	0.025
m	1.45	0.07
λ	0.3010	0.0021
μ_{bg}	0.04	0.08
σ_{bg}	2.46	0.04
f	0.601	0.004
Binned EMLE		
N	100.00×10^3	0.32×10^3
μ	2.9974	0.0028
σ	0.3031	0.0027
β	0.981	0.025
m	1.42	0.07
λ	0.3012	0.0021
μ_{bg}	0.05	0.08
σ_{bg}	2.46	0.04
f	0.602	0.004

A visualization of the unbinned EMLE estimate for parameter N is shown in Fig. 4.

The binned fit is significantly faster than the unbinned fit while maintaining comparable accuracy. Additionally, the execution time for sampling and binned fitting was evaluated using the `%timeit` command:

```
%timeit -n 100 np.random.normal(size=100000) # benchmark
%timeit -n 100 accept_reject_sampling_2d(model.pdf, 100000, x_min, x_max, y_min,
    y_max, f_max_3)
%timeit -n 100 optimize_binned()
```

The measured execution times were as follows:

```
2.91 ms ± 770 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
79.1 ms ± 1.43 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
54.6 ms ± 1.23 ms per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

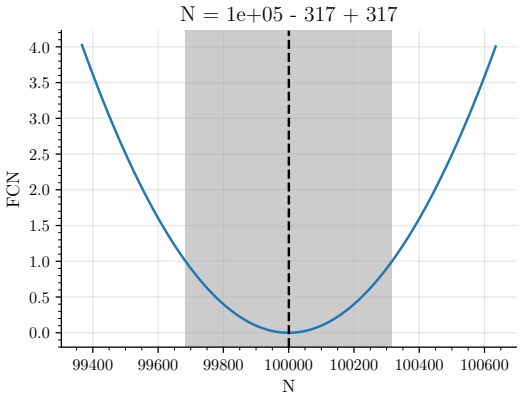


Figure 4: Estimate of N and its uncertainty.

5 Multi-Dimensional Fit and Parametric Bootstrap

In this section, we employ a multi-dimensional likelihood estimate to fit the signal-background model and implement parametric bootstrapping to assess the robustness of this procedure.

Generating Real Data We continue using the accept-reject sampling method to generate data, but introducing a Poisson variation in the sample size. This is implemented in the function `accept_reject_sampling_2d_with_Poisson` in `notebooks/e_bootstrapping.ipynb`.

Fitting the Data The data is fitted using the unbinned Extended Maximum Likelihood Estimation (EMLE) method to determine the "true" parameters for bootstrapping. Given that the sample sizes in this task are relatively small, the unbinned method is preferred over the binned method.

Bootstrap Procedure We use the `bootstrap` function from the `resample` package[2] to generate an ensemble of 250 samples for the bootstrapping process.

```
boot_samples = np.array( [ b for b in bootstrap.resample(samples, size=250) ] )
```

Fitting the Bootstrap Ensemble To check for any bias in the estimation method, we fit the generated samples back into the model and evaluate their deviation from the "true" value. Specifically, we analyze the parameter λ.

For real data generated with $N = 1000$ (with actual Poisson-adjusted sample size of 988), we fit the pseudo-experiments and get the expectation of the estimate and the uncertainty of the estimate, the expectation of the estimated uncertainty (Hesse error) and uncertainty of it in Fig. 5.

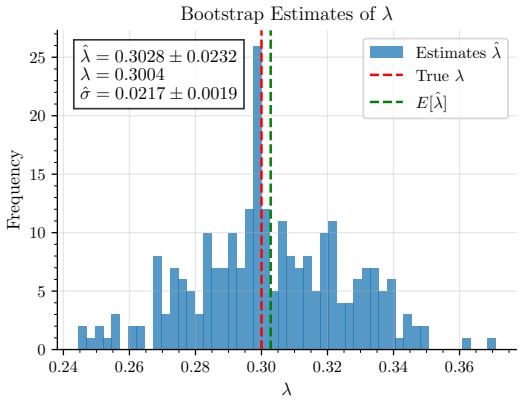


Figure 5: Bootstrap estimates of λ for a sample size of 1000.

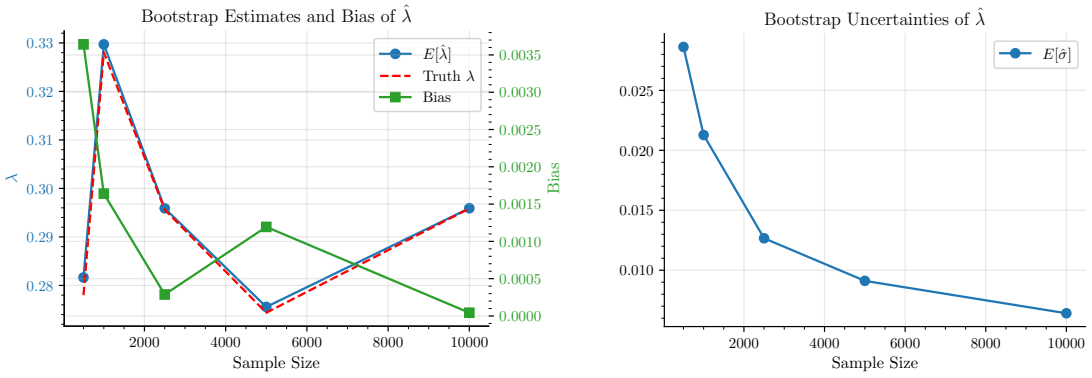
Impact of Sample Size To analyze the effect of sample size on the bias and uncertainty of λ, we repeat the entire procedure (data generation, fitting, and bootstrapping) for multiple sample sizes.

The double-loop process over sample sizes and bootstrap samples takes approximately 5 minutes on my laptop. The results are saved in `notebooks/bootstrap_results.csv` and summarized in Table 2.

Table 2: Comparison of sample size with estimate, uncertainty, and "true" value.

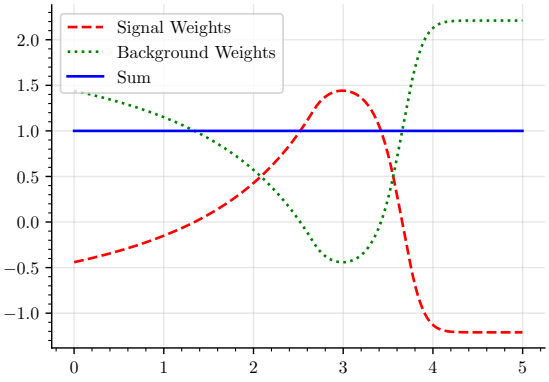
Sample Size	$E[\hat{\lambda}]$	**Std**	$E[\hat{\sigma}]$	**Std**	**Truth**
500	0.281632	0.026388	0.028621	0.002450	0.277990
1000	0.329699	0.023122	0.021276	0.001943	0.328058
2500	0.295903	0.012576	0.012663	0.000448	0.295615
5000	0.275563	0.011053	0.009112	0.000235	0.274368
10000	0.295914	0.006200	0.006409	0.000127	0.295871

Visualization of Results The results are visualized in Fig. 6. As the sample size increases, the (estimated) bias tends to 0 and the uncertainty decreases. This demonstrates that the unbinned EMLE method is an unbiased estimator.

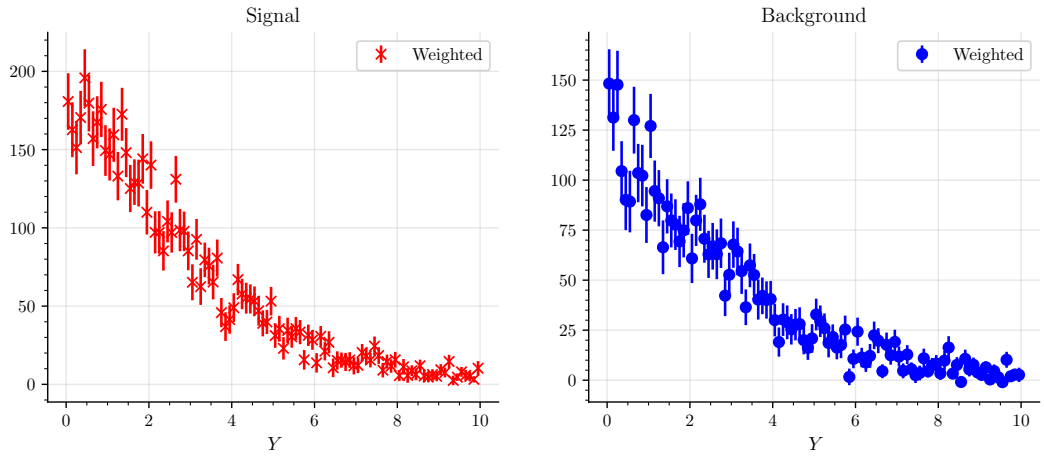


(a) (b)

Figure 6: (a) Bootstrap estimates of $\hat{\lambda}$ as a function of sample size. The plot compares the mean bootstrap estimate $E[\hat{\lambda}]$ (blue solid line with circles) with the true value of λ (red dashed line) and displays the corresponding bias (green solid line with squares). Bias decreases with increasing sample size. (b) Bootstrap uncertainties $E[\hat{\sigma}]$ of $\hat{\lambda}$ as a function of sample size, showing a steady decline in uncertainty as the sample size grows.



(a)



(b)

Figure 7: (a) Signal and background weights. (b) Projected distributions with error bars.

6 sWeights Projecting and Fit

In this case, our objective is to extract the properties of the signal in the Y dimension using a one-dimensional fit. In this scenario, the background distribution in the Y dimension, referred to as the control variable dimension, is unknown. To address this, we utilize sWeights to project onto this dimension.

Given the limited sample sizes, we employ an unbinned extended maximum likelihood estimation to first fit the data in the X variable, which serves as the discriminant variable.

Then, we use sWeights to extract the pure $h_s(Y)$ component. This is achieved through a weight function $w_s(X)$ such that:

Then, we use sWeights to project out pure $h_s(Y)$ component, which use a weight function $w_s(X)$ to make:

$$\int w_s(X) f(X,Y) = f h_s(Y).$$

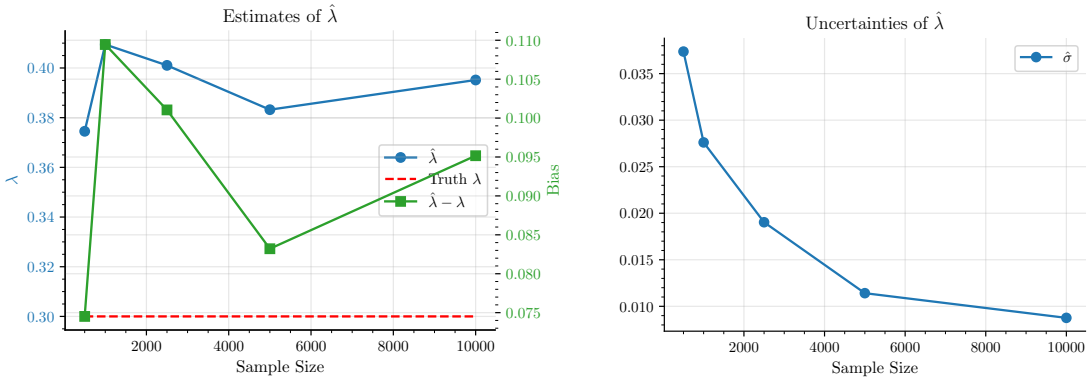
This process has been implemented by `sweights` package [3].

In a sample size of 10000, we present the weight functions for the signal and background, as well as the projected signal and background distributions in the Y dimension. These results are depicted in Figure 7.

By fitting these projected distributions, we can extract the desired signal properties in the Y dimension.

Change Sample Sizes and Fit In `notebooks/f_sWeights.ipynb`, we use the same samples as in Section 5, stored in `notebooks/all_samples.npy`.

We repeat the process of fitting in the X dimension, calculating the weights, and projecting the data across different sample sizes. The results are summarized in Table 3, where the "truth"



(a) (b)

Figure 8: (a) Estimates of $\hat{\lambda}$ for varying sample sizes. (b) Corresponding uncertainties ($\hat{\sigma}$) as a function of sample size.

value is $\lambda = 0.3$. For clarity, these results are also visualized in Figure 8.

Compared to findings from the previous part, the value of $\hat{\lambda} - \lambda$ is not neglectable for all sample sizes, based on which we cannot determine the estimates are unbiased. Furthermore, The true bias can be estimated using bootstrapping, although this has not been implemented here.

On the other hand, even though slightly larger than the previous part, the estimated uncertainties steadily decreases with the sample sizes.

Table 3: Comparison of parameter estimates, uncertainties, and true values across sample sizes.

Sample Size	$\hat{\lambda}$	$\hat{\sigma}$	**Truth** λ
500	0.374516	0.037371	0.3
1000	0.409456	0.027608	0.3
2500	0.401050	0.019033	0.3
5000	0.383212	0.011413	0.3
10000	0.395165	0.008759	0.3

7 Comparison of Two Methods

Multi-Dimensional Likelihood Fit Theoretically, the multi-dimensional likelihood fit is more computationally expensive because the search space grows exponentially, and the likelihood surface can be more complex. Furthermore, it requires more apriori knowledge, such as the distribution in the dimensions of interest.

However, this computational cost comes with a trade-off: the method is more accurate in practice.

In scenarios where the distributions of components in the dimensions of interest are well understood and the search space is not excessively large, the multi-dimensional likelihood fit is both a straightforward and effective choice.

sWeights Projecting and One-Dimensional Fit One advantage of the sWeights method is its ability to handle situations where the properties in the control variable dimension are unknown. Theoretically, sWeights can perfectly extract the distribution in that dimension.

Moreover, because the method involves fitting only the projection in a single dimension, it is computationally more efficient. If the extraction is performed perfectly, it is as accurate as using same fit method in multi-dimension.

However, in this case, the extraction is not so perfect, and introduces errors in the estimate.

In scenarios where the distributions of components in the dimensions of interest are not well understood, sWeights provides a crucial alternative.

A Use of AI Tools

The following describes how AI tools were utilized in the preparation of this report:

- **ChatGPT 4o**
  - **Drafting** - Used for drafting certain sections of the report, including the frist section and Appendix. Also used for drafting captions of figures and tables.
  - **Proofreading** - Reviewed grammatical correctness, improved sentence structure, and suggested alternative wordings for clarity across the report.
  - **LaTeX Assistance** - Helped resolve issues encountered during the LaTeX formatting process, such as debugging compilation errors, optimizing figure placement, and improving overall document layout.

B Template Information

This report was formatted using the LaTeX template from the given Example Coursework (by Michal Dorko). Some modifications were made.

References

[1] Hans Dembinski and Piti Ongmongkolkul et al. "scikit-hep/iminuit". In: (Dec. 2020). DOI: 10.5281/zenodo.3949207. URL: https://doi.org/10.5281/zenodo.3949207.

[2] Hans Dembinski and Daniel Saxton. *resample: Randomization-based inference in Python*. Version 1.10.1. BSD-3-Clause License. Dec. 16, 2024. URL: https://github.com/scikit-hep/resample.

[3] 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.

[4] Hans Dembinski et al. *HDembinski/numba-stats: v1.8.0*. Version v1.8.0. Aug. 2024. DOI: 10.5281/zenodo.13236518. URL: https://doi.org/10.5281/zenodo.13236518.