S1 Coursework Report

Zhimei Liu (zl474)

December 18, 2024

Word count: 2355

Declaration: I, Zhimei Liu, hereby declare that the work presented in this report and the GitLab repository is entirely my own. I have only used ChatGPT for some debugging to speed up the process.

Contents

0 Introduction

In this coursework, we aim to compare the statistical power of two fitting techniques: an extended maximum likelihood fit and a weighted fit utilizing *sWeights*. Our analysis is centered on a two-dimensional model where we first generate an ensemble of samples through parametric bootstrapping, also referred to as a "toy study". These samples are then fitted back to the model to determine the parameters of the underlying distribution.

We contrast the performance of this approach with *sWeights*, a method where only one dimension of the model is fitted, and the other dimension is projected through a weighted transformation. By comparing these techniques, we will assess their respective strengths and weaknesses in terms of fitting accuracy and computational efficiency, ultimately evaluating their statistical power for use in high-energy physics and other areas requiring multi-dimensional analysis.

The report will detail the methods, results, and discussion surrounding the two fitting approaches, with an emphasis on understanding their implications for real-world data analysis.

The main part of the code is run in the file named `S1_coursework.ipynb`. The probability density functions are defined in a separate file named `DistriFuncs.py`.

Sections of this report are divided corresponding to the questions listed.

1 Checking Normalization for the Crystall Ball Density Function

We want to normalise the following probability density function:

$$p(X;\mu,\sigma,\beta,m) = N \begin{cases} e^{-Z^2/2} & \text{for } Z > -\beta \\ \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2}\left(\frac{m}{\beta} - \beta - Z\right)^{-m} & \text{for } Z \leq -\beta \end{cases} \tag{1}$$

with $Z = \frac{X-\mu}{\sigma}$ and

$$\Phi(x') = \int_{-\infty}^{x'} \frac{1}{\sqrt{2\pi}} \exp\big(-\frac{x^2}{2}\big)dx = \int_{-x'}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\big(-\frac{x^2}{2}\big)dx \tag{2}$$

where the second equality is obtained by symmetry. Using this change of variable, we have $dX = \sigma dZ$ and

$$\begin{aligned} 1 &= \int_{-\infty}^{\infty} p(X;\mu,\sigma,\beta,m)dx \\ &= N\sigma\Big[\int_{-\infty}^{-\beta} \Big(\frac{m}{\beta}\Big)^m e^{-\beta^2/2}\big(\frac{m}{\beta} - \beta - Z\big)^{-m}dZ + \int_{-\beta}^{\infty} e^{-Z^2/2}dZ\Big] \\ &= N\sigma\left(\Big(\frac{m}{\beta}\Big)^m e^{-\beta^2/2}\Big[-\frac{1}{-m+1}\big(\frac{m}{\beta} - \beta - Z\big)^{-m+1}\Big]_{-\infty}^{-\beta} + \sqrt{2\pi}\Phi(\beta)\right) \\ &= N\sigma\Big[\Big(\frac{m}{\beta}\Big)^m e^{-\beta^2/2}\frac{1}{m-1}\Big(\frac{m}{\beta}\Big)^{-m+1} + \sqrt{2\pi}\Phi(\beta)\Big] \quad (\text{for } m > 1) \\ &= N\sigma\Big(\frac{m}{m-1}\frac{1}{\beta}e^{-\beta^2/2} + \sqrt{2\pi}\Phi(\beta)\Big). \end{aligned}$$

Therefore, we can derive the required value for N^{-1}:

$$N^{-1} = \sigma\Big(\frac{m}{m-1}\frac{1}{\beta}e^{-\beta^2/2} + \sqrt{2\pi}\Phi(\beta)\Big). \tag{3}$$

2 Checking Normalization for Various Functions

The true value for the eight parameters are:

```
initial_f = 0.6
initial_mu = 3
initial_sigma = 0.3
initial_beta = 1
initial_m = 1.4
initial_Lambda = 0.3
initial_mu_b = 0
initial_sigma_b = 2.5
```

I defined all the necessary probability density functions in the file named `DistriFuncs.py`, where the `scipy.stats` library was used.

To check the normalization of functions g_s, g_b, h_s and h_b, I employed the `quad` function from `scipy.integrate` library. The output for the nomalizations of the above functions is as follows:

```
Normalization of g_s(x): 1.000000, Error: 7.289797e-09
Normalization of g_b(x): 1.000000, Error: 1.110223e-14
Normalization of h_s(y): 1.000000, Error: 1.110223e-14
Normalization of h_b(y): 1.000000, Error: 4.839205e-12
```

which shows that the functions are correctly normalized, with the errors within acceptable ranges. To check the normalization of the joint distribution $F(X, Y)$ (to avoid confusion between the parameter f and the joint density function $f(X, Y)$, I renamed the joint density function to be $F(X, Y)$), I used *Monte Carlo integration*, with the function `check_normalisation_for_F` in the file `S1_coursework.ipynb`. The output is

```
Normalization of F: 1.000004, Error: 1.427056e-03
```

which shows that $F(X, Y)$ is also correctly normalized.

I also checked the normalizations of the above functions with different values for the eight parameters, and the results were shown to be correct.

3 Plots of Distributions

It would be nice to visualize the one-dimensional projections of the distributions in both X and Y variables. To plot the one-dimensional projections of the distributions in X, we integrate out the variable Y in the joint distribution

$$F(X, Y) = f g_s(X) h_s(Y) + (1 - f) g_b(X) h_b(Y). \tag{4}$$

Since the distribution $h_s(Y)$ is correctly normalized, then the one-dimensional projection of f in X is

$$F(X) = f g_s(X) + (1 - f) g_b(X). \tag{5}$$

Similarly, the one-dimensional projection of f in Y is

$$F(Y) = f h_s(Y) + (1 - f) h_b(Y). \tag{6}$$

The plots of one-dimensional projections of distributions in both X and Y variables are shown in Figure 1.

Next, I plotted the two-dimensional plot of the joint density $F(X, Y)$ in Figure 2. I also plotted a 3D surface plot of the joint density F using `Axes3D` from the `matplotlib` library, as displayed in Figure 3. The function that did the coding is called `plot_3d_for_F`.

With those plots in mind, I can compare my later results with the correct plots and check that my later results are correct.

4 High-Statistics Sample Generation and Extended Maximum Likelihood Fit

4.1 Accept-Reject Algorithm for Sample Generation

To generate 100,000 events from the joint distribution $F(X, Y)$, I used the accept-reject algorithm. The code for this is displayed as follows:

```
def generate_samples(n_events, f, mu, sigma, beta, m, lambda_param, mu_b, sigma_b):
    sample_x = []
    sample_y = []
    batch_size = 1000000

    while len(sample_x) < n_events:
        # Generate random candidate points
        x = np.random.uniform(0, 5, batch_size)
        y = np.random.uniform(0, 10, batch_size)

        # Evaluate F(x, y)
        probs = DistriFuncs.F(x, y, f, mu, sigma, beta, m, lambda_param, mu_b,
            sigma_b)

```

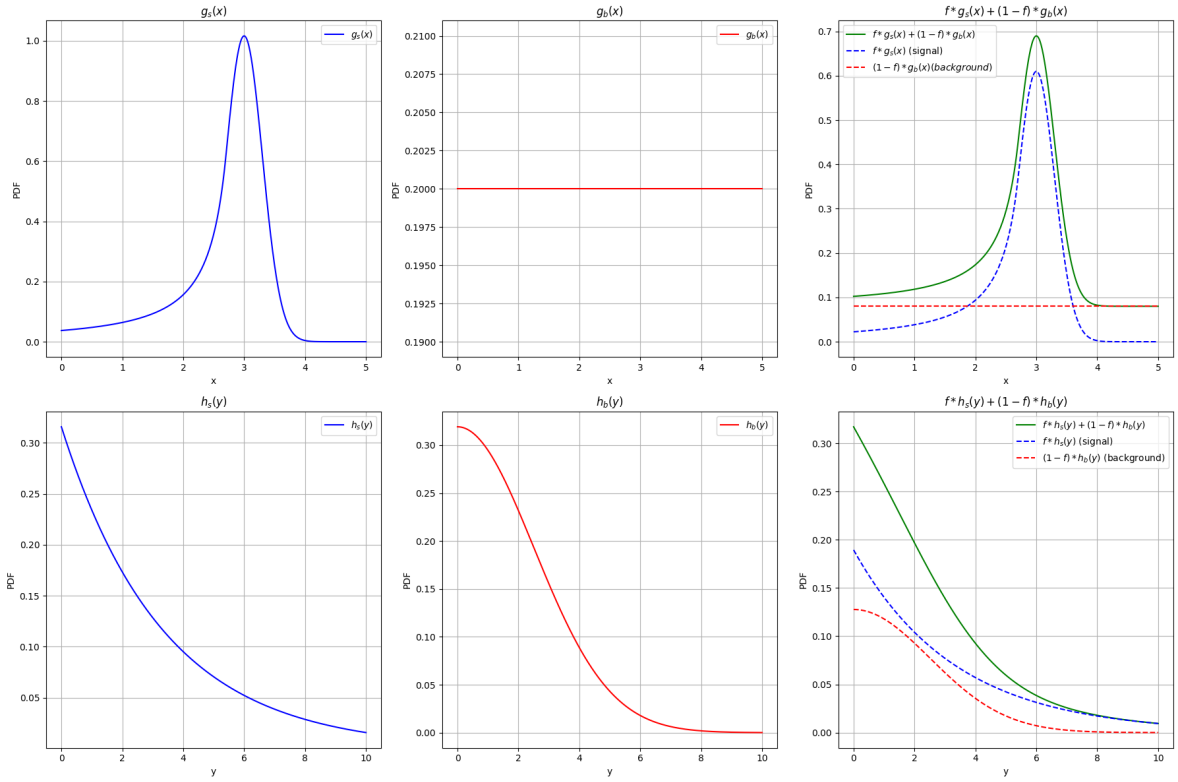

Figure 1: One-dimensional projections of the distributions in both the X and Y variables.

```
        # Perform accept-reject step
        accepts = np.random.uniform(0, 1, batch_size) < probs
        accepted_x = x[accepts]
        accepted_y = y[accepts]

        # Collect accepted samples
        sample_x.extend(accepted_x)
        sample_y.extend(accepted_y)

        # Stop once we have enough samples
        if len(sample_x) >= n_events:
            sample_x = sample_x[:n_events]
            sample_y = sample_y[:n_events]
            break

    return (np.array(sample_x), np.array(sample_y))
```

Let's visualise the generated samples with `plt.hist2d`. The two-dimensional histogram of 100,000 events is displayed in Figure 4.

Comparing between Figure 2 and Figure 4, we can see that the two figures coincide with high enough accuracy. Therefore, we can conclude that the accept-reject algorithm is correct.

4.2 Extended Maximum Likelihood Fit

In this subsection, I perform an extended maximum likelihood fit. Since the fit is extended, then there will be an extra parameter which is the number of events. Therefore, I will estimate a total of nine parameters. I used the `Minuit` package from the `iminuit` library. The function to perform the fit is called `extended_maximum_likelihood_fit`. The estimated values for the nine parameters with uncertainties are as follows:

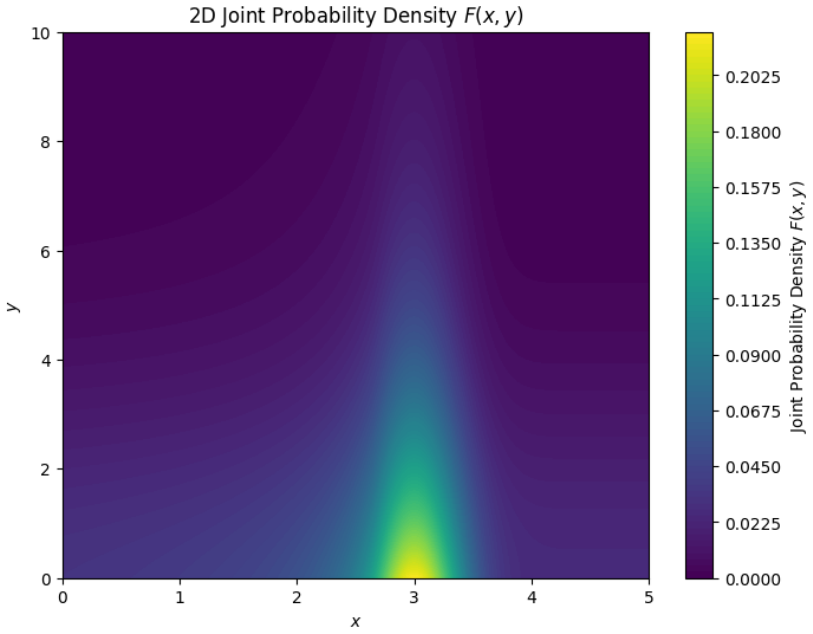

Figure 2: The two-dimensional plot of the joint probability density $F(X, Y)$ with $f = 0.6$.

```
N:              100.00e3 ± 0.32e3
f:              0.599    ± 0.004
mu:             2.9998   ± 0.0026
sigma:          0.3028   ± 0.0024
beta:           1.034    ± 0.023
m:              1.30     ± 0.06
lambda_param:   0.2981   ± 0.0021
mu_b:           -0.07    ± 0.08
sigma_b:        2.52     ± 0.04
```

We can see that the estimated values are close to the true values for the parameters. The fitted values are saved in the file named `fitted_parameters.npy`.

4.3 Evaluation of the Execution Time

We want to estimate the time it takes to generate 100,000 samples as well as to perform the fit to the sample to estimate the parameters. I used the `timeit` library to evaluate the execution time.

Firstly, I evaluated the execution time when calling `np.random.normal(size=100000)` to set a standard benchmark for my machine. The times for 100 calls and averaged over 100 calls are listed in the following, respectively:

```
Benchmark time (standard normal) for 100 calls: 0.251 s
Benchmark time (standard normal) averaged over 100 calls: 0.00251 s
```

Averaged over 100 calls, it takes 0.0026 seconds to complete a single process.

Next, I evaluated the execution time of generating 100,000 samples using the function `generate_samples`. The time for 100 calls and averaged over 100 calls are listed in the following, respectively, where the time inside the bracket shows the relative time of generating sample of 100,000 events compared to the previous process:

```
Sample generation time for 100 calls: 163.76 s (652.47x relative)
Sample generation time averaged over 100 calls: 1.638 s
```

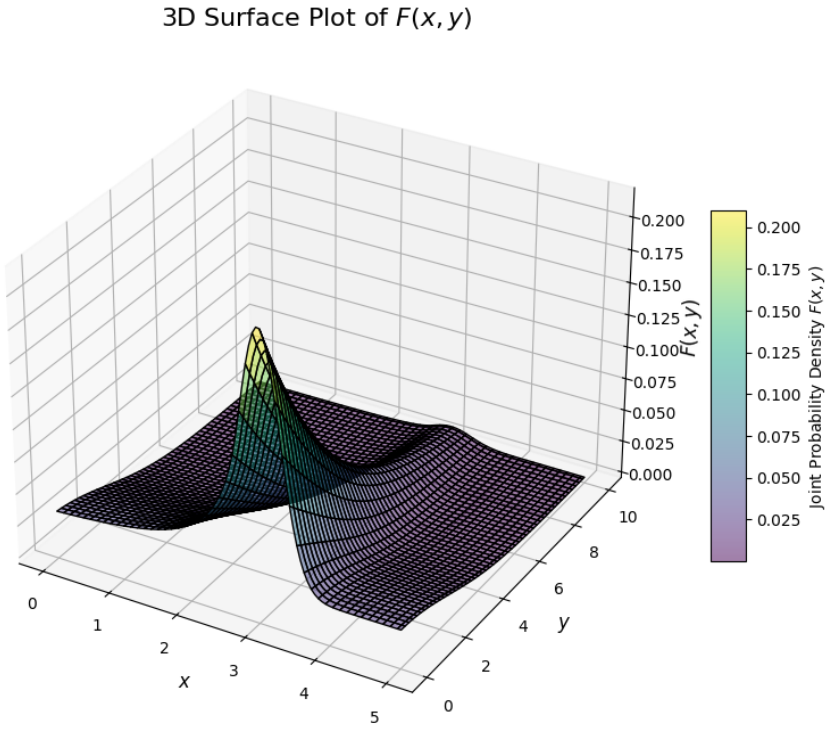

Figure 3: The three-dimensional surface plot of the joint probability density $F(X, Y)$ with $f = 0.6$.

Finally, I evaluated the execution time of performing the fit to the 100,000 samples to estimate the parameters. The results are listed below:

```
Extended maximum likelihood fit time for 100 calls: 1080.24 s (4303.95x relative)
Extended maximum likelihood fit time averaged over 100 calls: 10.80 s
```

5 Parametric Bootstrapping

In this section, I would run a simulation study using parametric bootstrapping with an ensemble of 250 sample (so-called "toys"). The trial sample sizes are 500, 1000, 2500, 5000 and 10000, respectively, with a Poisson variation of the sample sizes.

5.1 A Summary of the "Toy" Study

In the file `S1_coursework.ipynb`, the function `plot` is used to plot the distributions of the estimates of the nine parameters and the pulls of the estimates; the function `generate_250_toys` is used to generate 250 toys from the Poisson variation of each of the sample size and the fitted values from the previous extended maximum likelihood fit; and finally the function `show_final_plots` is used to show the plots for the nine parameters altogether.

From Figure 5 to Figure 9, they are the outputs from the function `show_final_plots(10000)` with a trial sample size of 500, 1000, 2500, 5000 and 10000 respectively. Specifically, I look at the distribution of estimated $\hat{\lambda}$ values and the distribution of its pulls for various sample sizes. These are reasonable plots of distributions because:

- For the distributions of estimated value, the means all are close to the original value with small enough uncertainties on the means;

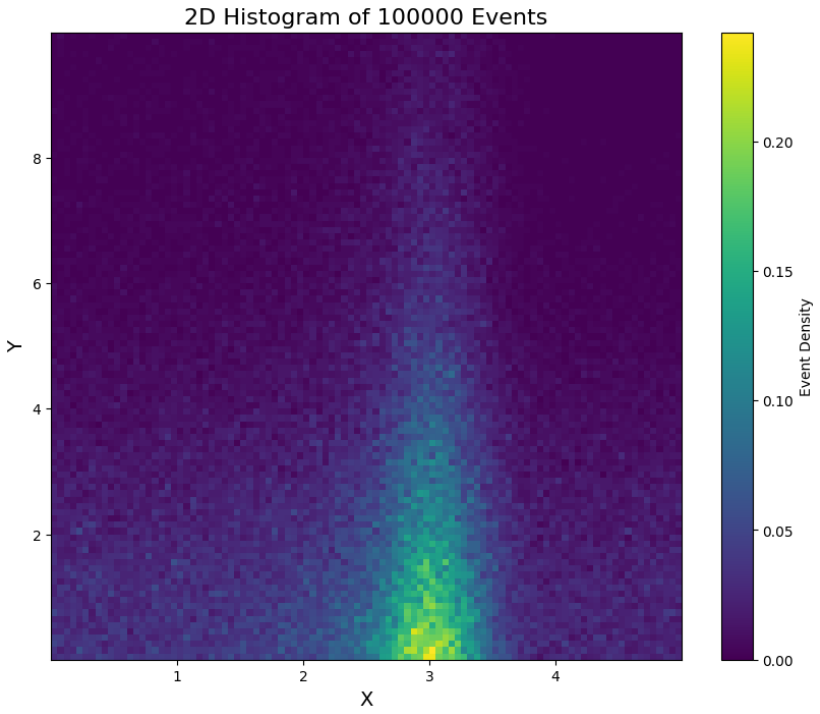

Figure 4: 2D histogram of 100,000 events generated from the accept-reject algorithm.

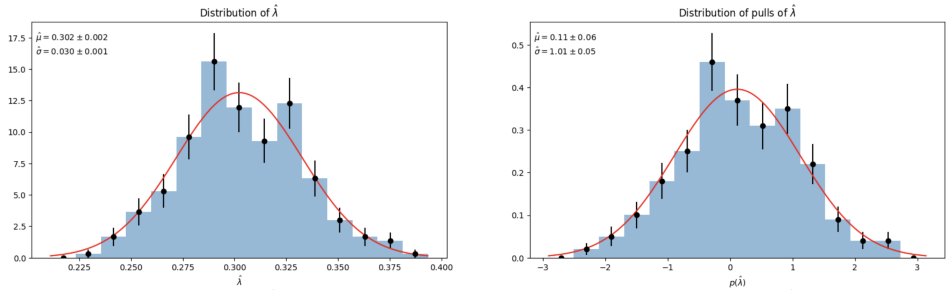

Figure 5: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=500 from parametric bootstrapping.

- For the distributions of the pulls of the parameter, they are all close to the standard normal distribution (recall that the pulls of parameters refers to the differences between the estimated values of parameters from the original data and the values of parameters generated fro simulated data, i.e., the bootstrapped samples).

For more plots with other parameters, please check the outputs in the file `S1_coursework.ipynb`.

5.2 Analysis of the Decay Constant λ

Let's run some analysis of the decay constant λ. I stored the λ parameter values and its pulls with various sample sizes in lists named `estimated_lambda_values_list_for_part_e` and `pull_values_list_for_part_e` respectively. Then, using the function `plot_error_bar_for_pulls_of_lambda_part_e`, I plotted Figure 10, which is a summary of the pulls of λ with error bars for the "toy" study described above. We can see that as sample size increases, the mean of pulls gets closer to 0, showing that the estimated λ values agrees better with the true fitted λ value from the extended maximum likelihood fit from Section 4.2.

Next, from the function `plot_error_bar_for_lambda_part_e`, I plotted the mean of the estimated

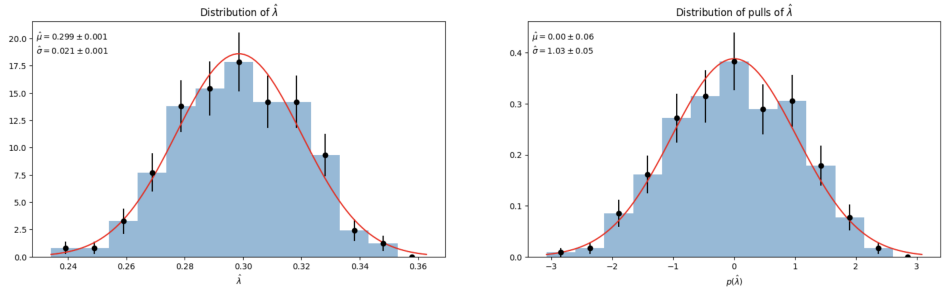

Figure 6: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=1000 from parametric bootstrapping.

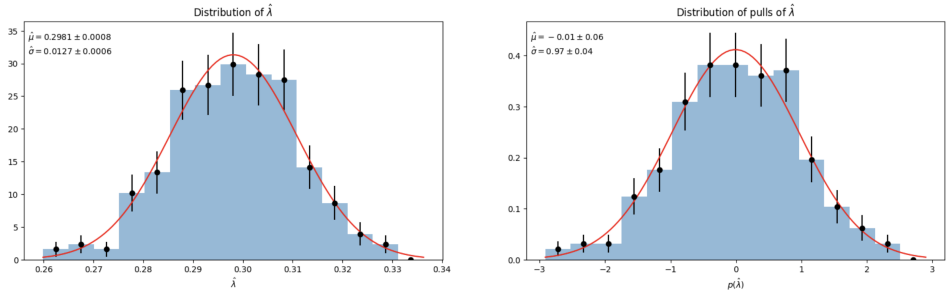

Figure 7: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=2500 from parametric bootstrapping.

λ values as a function of the sample sizes to examine the bias on λ, as demonstrated in Figure 11. We can see that for the mean plot, the mean of λ gets closer to the fitted λ value, as expected, with a decreasing width of the error bar. The bias, calculated by the difference between the mean of λ and the fitted λ value, has a decreasing trend in its absolute value as well.

Lastly, using the function `plot_uncertainty_on_lambda_part_e`, I plotted the uncertainty on λ as a function of the sample size, as shown in Figure 12. The uncertainty on λ is defined as the standard deviation of the estimated λ values

$$\hat{\sigma} = \frac{1}{N-1}\sum_i (x_i - \hat{\mu})^2, \quad \text{where} \quad \hat{\mu} = \frac{1}{N}\sum_i x_i, \tag{7}$$

while the width of error bars is calculated as the error on the standard deviation estimate

$$\sigma_{\hat{\sigma}} = \frac{\hat{\sigma}}{\sqrt{2(N-1)}}. \tag{8}$$

It is clear that the uncertainty generally has a small value for various sample sizes, and as the sample size decreases, the uncertainty also decreases consistently. This shows that my parametric bootstrapping model is correctly constructed.

6 sWeights

After successfully running parametric bootstrapping, let's run a different method called *sWeights* and compare between the two methods. First, I perform an extended maximum likelihood fit in just the X variable, and then use it to produce *sWeights* which project out the signal density in Y.

6.1 Extended Maximum Likelihood Fit in X

The code that performs an extended maximum likelihood fit in just the X variable is the function called `extended_maximum_likelihood_fit_on_x`, where it used `ExtendedUnbinnedNLL(x_sample, DistriFuncs.total_density_x)` with the total density function `total_density_x` defined in `DistriFuncs.py`

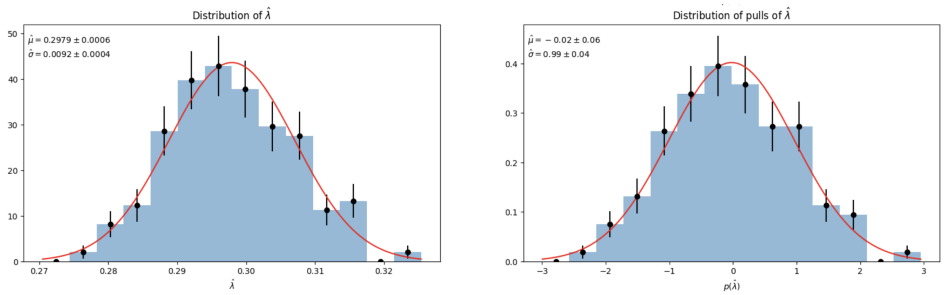

Figure 8: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=5000 from parametric bootstrapping.

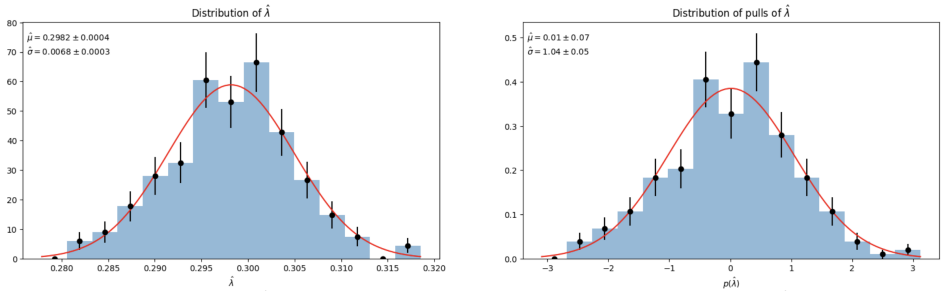

Figure 9: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=10000 from parametric bootstrapping.

by integrating variable Y out. Since we only focus on the projection in X, there are only six parameters we need to consider: N (number of events), f, μ, σ, β and m. The estimated parameter values with uncertainties are

```
N:      100.00e3  ±  0.32e3
f:      0.601     ±  0.004
mu:     2.9996    ±  0.0026
sigma:  0.3029    ±  0.0026
beta:   1.041     ±  0.024
m:      1.27      ±  0.06
```

6.2 Producing sWeights

By defining the function called `sweight_plot`, I plotted the distributions of $\hat{\lambda}$ as well as its pulls for sample sizes of 500, 1000, 2500, 5000, 10000, as shown from Figure 13 to Figure 17.

6.3 Analysis of the Decay Constant λ

To analyse the behaviour of the decay constant λ, I stored the estimated λ values and its corresponding pull values in the lists `estimated_lambda_values_list_for_part_f` and `pull_values_list_for_part_f`, respectively, as I did for the method of parametric bootstrapping.

Using the function `plot_error_bar_for_pulls_of_lambda_part_f`, the first plot shown in Figure 18 illustrates the how the mean value of the pulls of λ as the sample size changes. The figure proves that the distribution of pulls largely follows a standard normal distribution with mean of 0 and standard deviation of 1.

The second plot shown in Figure 19 tells us the behaviour of the mean values of λ and its bias as a function of the sample sizes. This was plotted by the function `plot_error_bar_for_lambda_part_f` We can see that as the sample size increases, the mean of λ tends towards to the fitted value of λ from the extended maximum likelihood fit from Section 4.2. Each of the error bar traps the fitted value inside its region. The bottom plot of Figure 19 gives the bias on λ as a function of the sample sizes. It

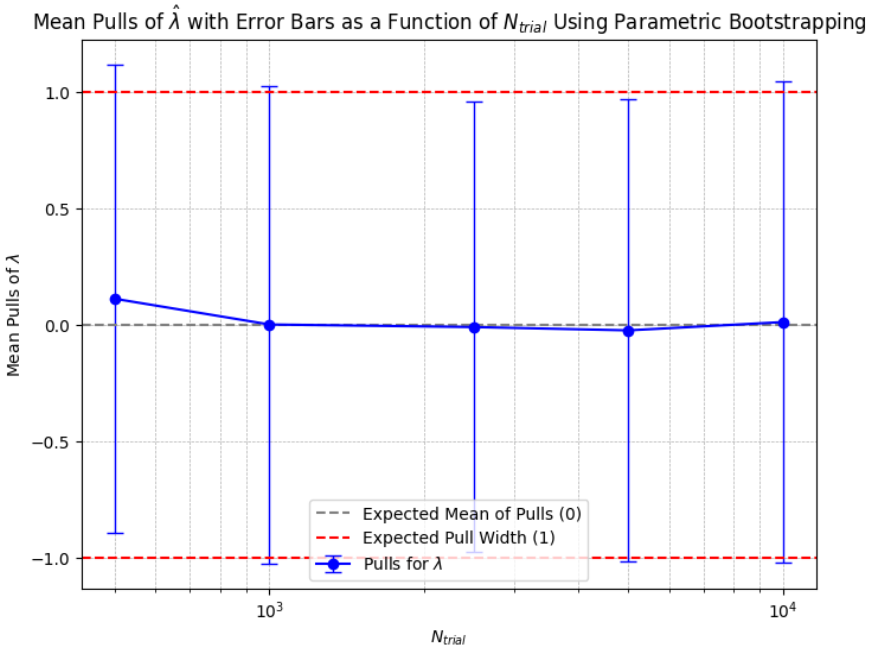

Figure 10: A summary of the pulls of the parameter λ with error bars for the "toy" study described above.

is clear that the bias gets closer and closer to 0 for larger sample sizes, proving that the performance of the method is increasing accordingly.

Last plot shown in Figure 20 gives the uncertainty on λ. This was plotted by the function `plot_uncertainty_on_lambda_part_f`. The small values of the uncertainty demonstrate the good performance of the method. As the sample size increases, the uncertainty decreases consistently to the value of 0.00675 for a sample size of 10000.

7 Comparison Between the Two Methods

In this last section, let's compare the performance of the two models.

7.1 Comparison of Parametric Bootstrapping and sWeights

The function `comparing_lambda_and_bias` would produce Figure 21 for us, which basically combines Figure 11 and Figure 19 in the same plot. In terms of accuracy and bias, we have the following conclusion:

- **Parametric Bootstrapping:** The mean of λ closely approximates the fitted value across all sample sizes, and the bias is consistently small with little variation.
- **sWeights:** Similarly, the sWeights method produces an accurate mean of λ, matching the fitted value well. However, the error bars for smaller sample sizes (e.g., $N_{\text{trial}} = 500$) are larger compared to parametric bootstrapping, suggesting higher variability in small-sample regimes.
- **Overall Observation:** Both methods are unbiased overall, but parametric bootstrapping appears slightly more stable for small sizes, as indicated by the tighter error bars.

Figure 22, produced by function `comparing_uncertainty`, combines Figure 12 and Figure 20 in the same plot. The following conclusions can be made:

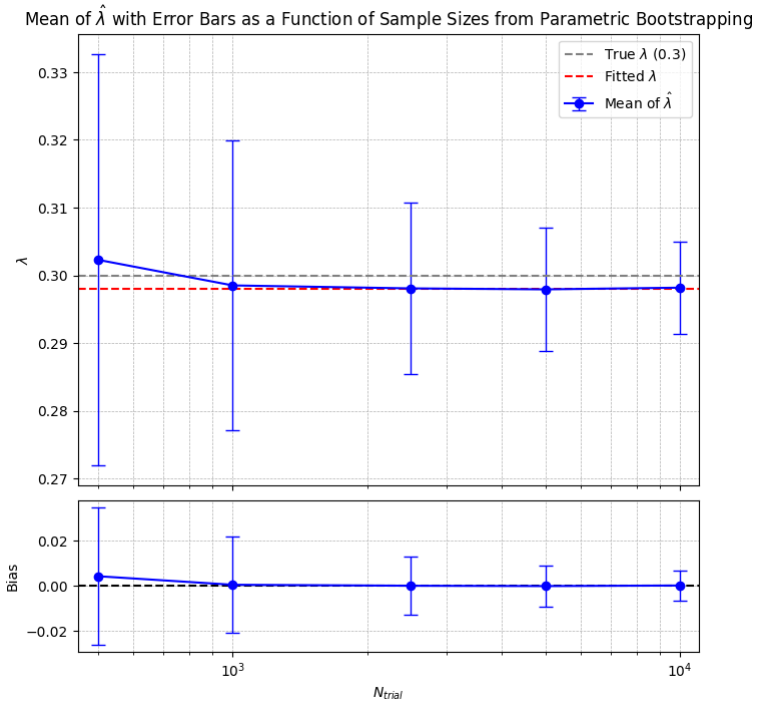

Figure 11: The top plot shows the mean of parameter λ using parametric bootstrapping as a function of the sample size, with error bar representing the standard deviation of λ. The bottom plot shows the bias on λ.

- Both methods show a decreasing trend in uncertainty (standard deviation of λ) as the sample size increases. This is expected because larger sample sizes reduce statistical fluctuations.
- For **large sample sizes**, both methods converge, producing nearly identical uncertainties.
- For **small sample sizes**, parametric bootstrapping shows slightly higher uncertainty (blue points), but sWeights exhibits more variability (green points), possibly due to its reliance on weights.

7.2 Potential Drawbacks and Disadvantages

Let's first analyse the advantages and disadvantages of the method of **parametric bootstrapping**:

1. **Advantages:**
   - Makes full use of the joint distribution, leading to robust results.
   - Performs well even when data distributions are relatively complex, as it resamples from a fitted model.
2. **Disadvantages:**
   - Computationally expensive, especially with large datasets or complex models, as it requires repeated resampling and fitting.
   - Its accuracy depends on the correctness of the underlying parametric model. If the model is misspecified, results can be biased.

As for the method of **sWeights**, there are the following advantages and drawbacks:

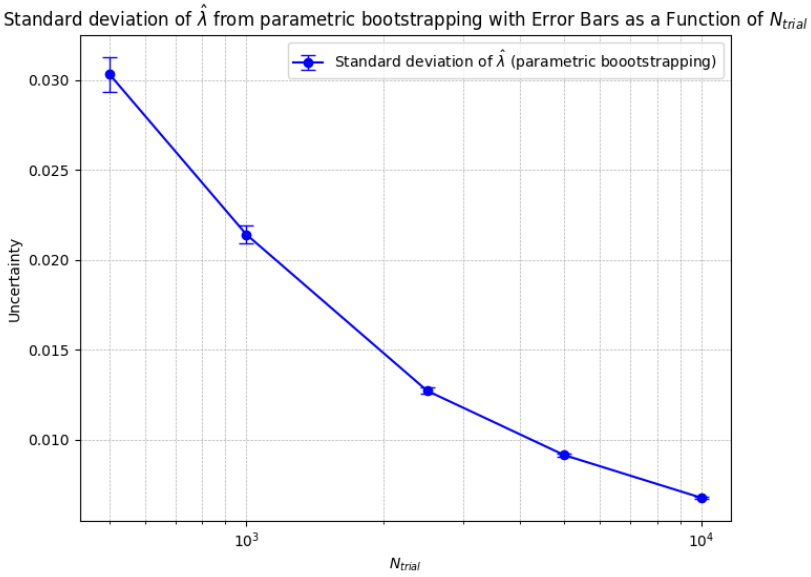

Figure 12: The expected uncertainty (standard deviation of λ) as a function of the sample size using parametric bootstrapping.

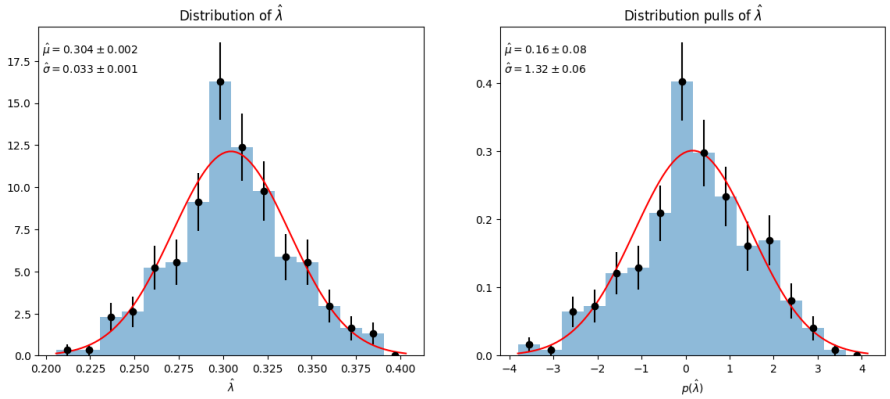

Figure 13: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=500 from `sWeights`.

1. **Advantages:**
   - Computationally efficient compared to parametric bootstrapping because it avoids repeated resampling.
   - Works directly with weighted datasets, making it faster for large data sizes.
   - Particularly useful in scenarios where one dimension of the distribution (e.g., background distribution) is explicitly known.
2. **Disadvantages:**
   - Sensitive to statistical fluctuations, especially for small sample sizes.
   - The weights introduce additional variance and might be unstable for low statistics or poorly constrained models.
   - Assumes a correct decomposition of signal and background components, which may not always be true.

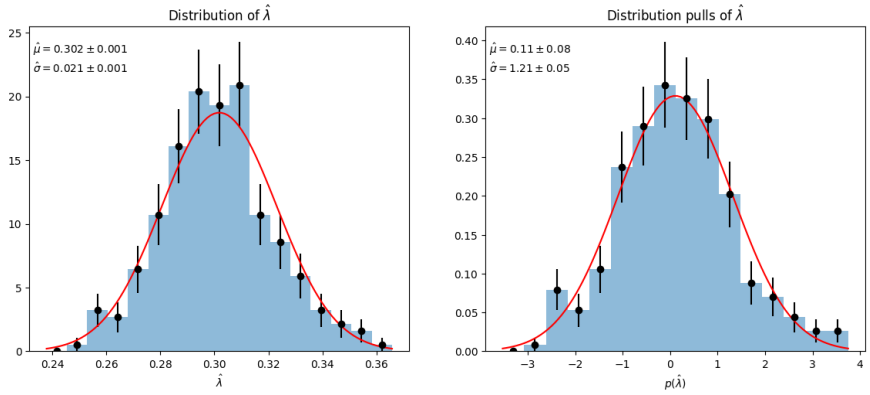

Figure 14: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=1000 from `sWeights`.

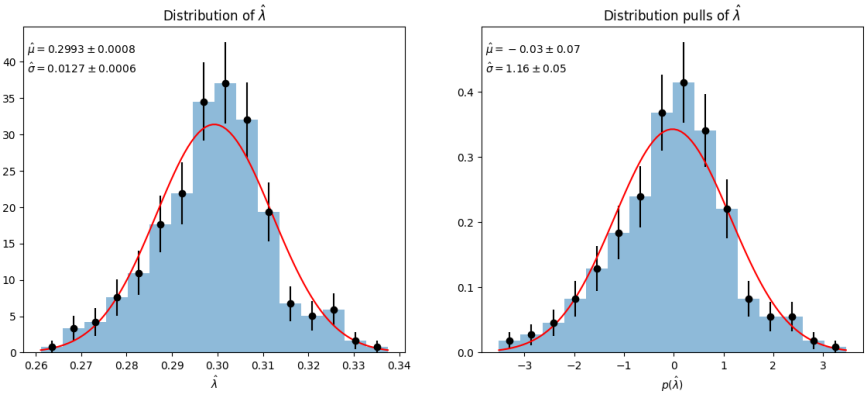

Figure 15: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=2500 from `sWeights`.

7.3 Most Appropriate Method and Preferred Scenarios

Based on our previous analysis on the two methods, **parametric bootstrapping** is generally more appropriate when:

- We have sufficient computational resources.
- The model's joint distribution is well-understood and parametric assumptions are valid.
- Accuracy and robustness are critical, especially for smaller sample sizes.
- Applications involve validating model fits or estimating uncertainties in complex systems.

On the other hand, **sWeights** is more preferred when:

- Computational efficiency is a priority.
- We need rapid results, especially with large datasets.
- A clear decomposition into weighted components (e.g., signal and background) exists and is well-constrained.
- Sample sizes are sufficiently large to minimize statistical fluctuations.

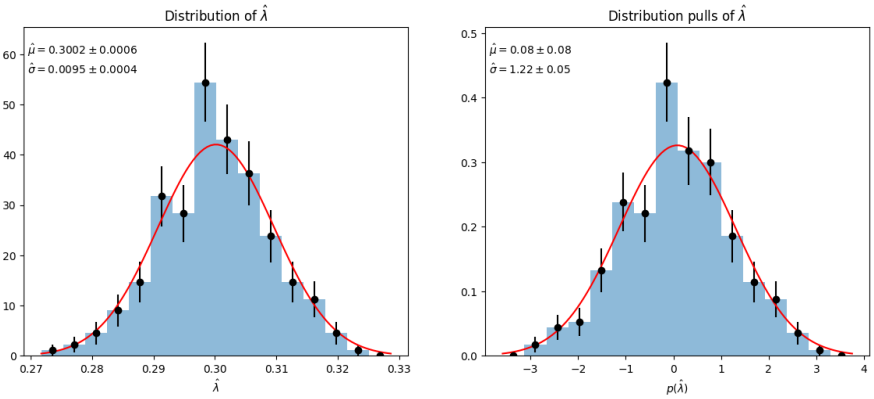

Figure 16: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=5000 from `sWeights`.

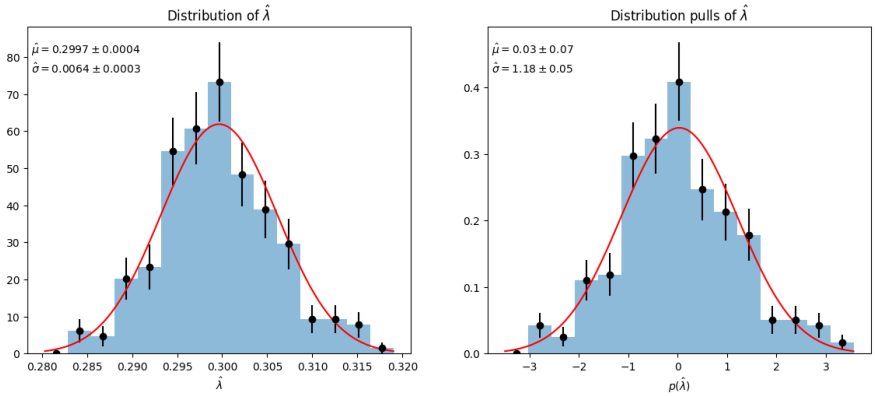

Figure 17: Distribution of $\hat{\lambda}$ and pulls of $\hat{\lambda}$ for sample size=10000 from `sWeights`.

8 Conclusion

Both methods produce unbiased results, but **parametric bootstrapping** shows slightly greater stability and precision for smaller sample sizes due to its rigorous use of the joint distribution. **sWeights**, while computationally faster, introduces higher variability in small-sample scenarios. For applications requiring precise uncertainty estimation and robustness, parametric bootstrapping is generally the more appropriate choice. For large datasets where speed is critical, sWeights may be preferred.

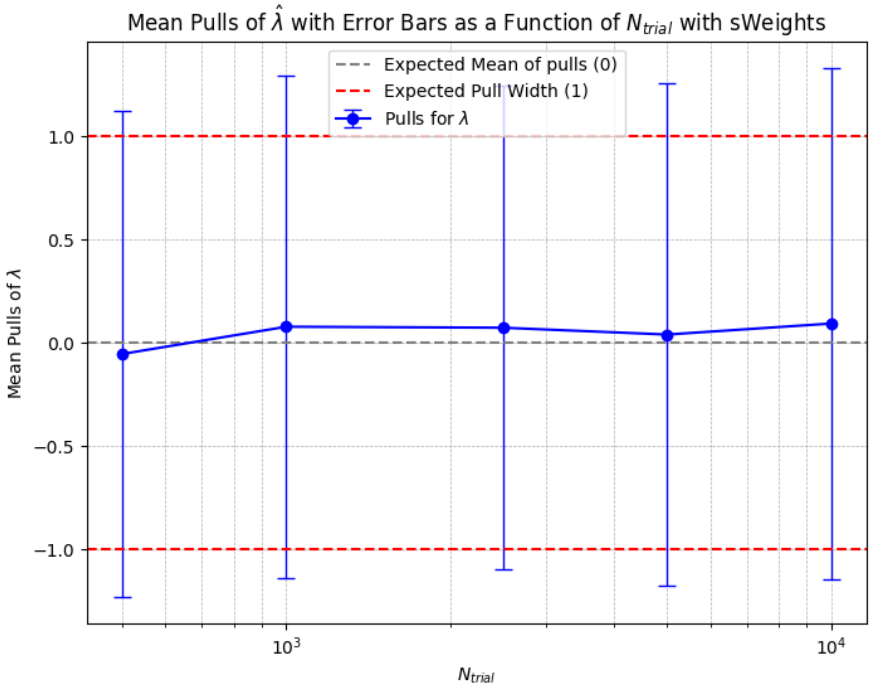

Figure 18: A summary of the pulls of parameter λ with error bars for the "toy" study described before.

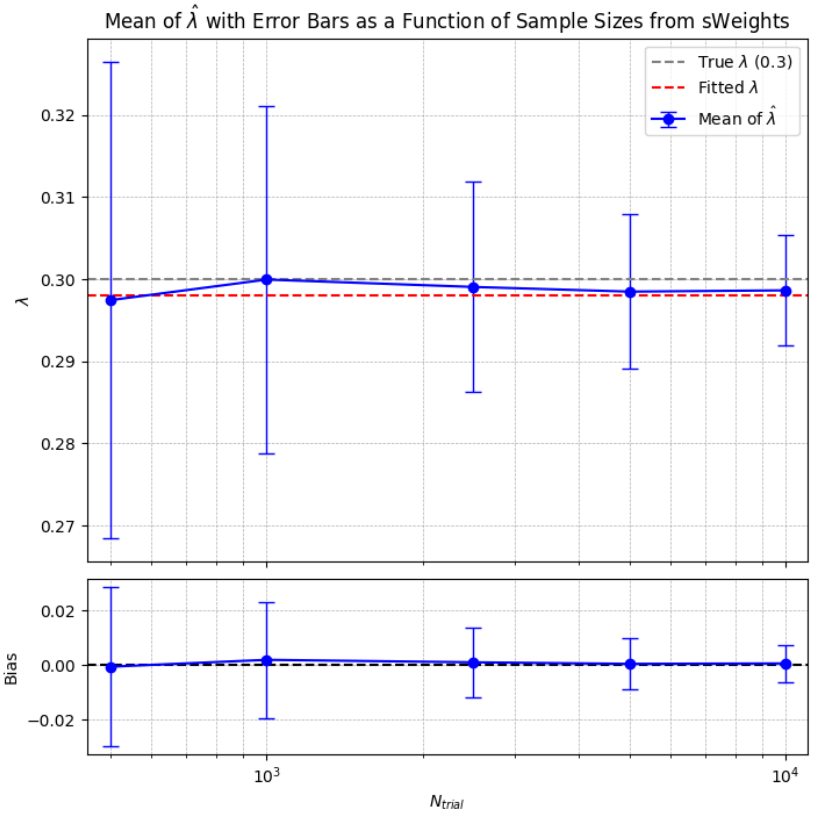

Figure 19: The top plot shows the mean of parameter λ using sWeights as a function of the sample size, with error bar representing the standard deviation of λ. The bottom plot shows the bias on λ.

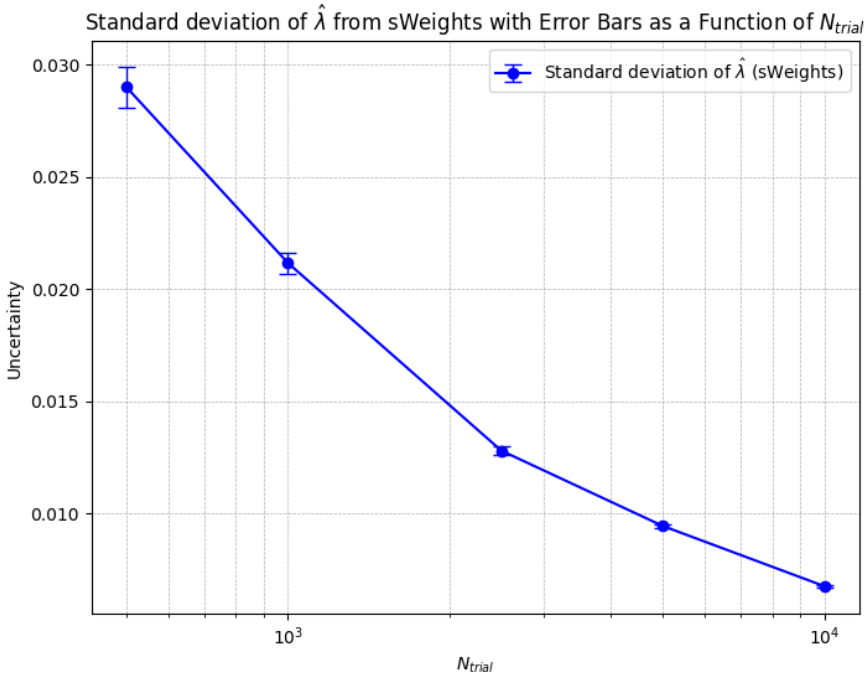

Figure 20: The expected uncertainty (standard deviation of λ) as a function of the sample size using sWeights.

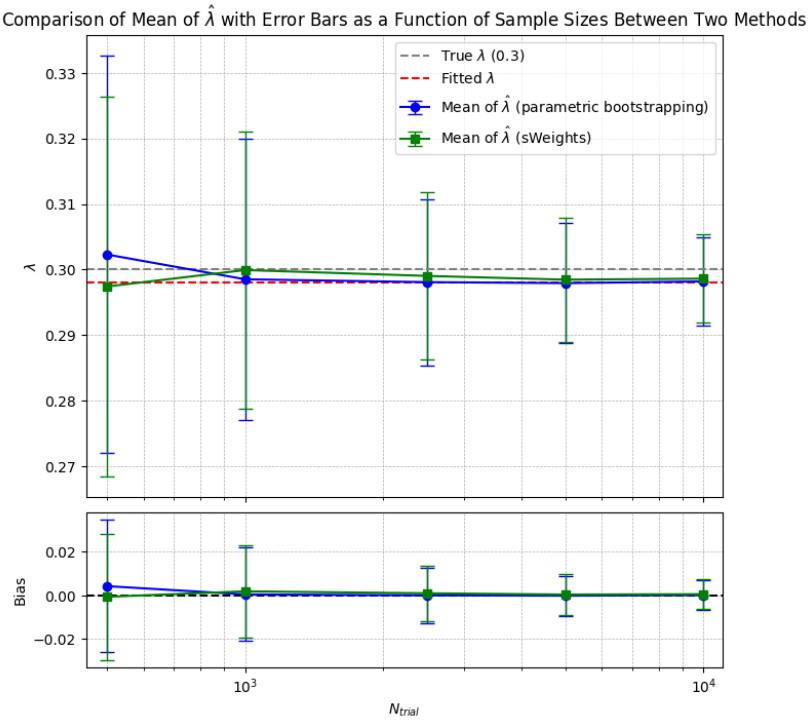

Figure 21: Comparison of the mean and bias on parameter λ as a function of sample sizes between the method of parametric bootstrapping and sWeights.

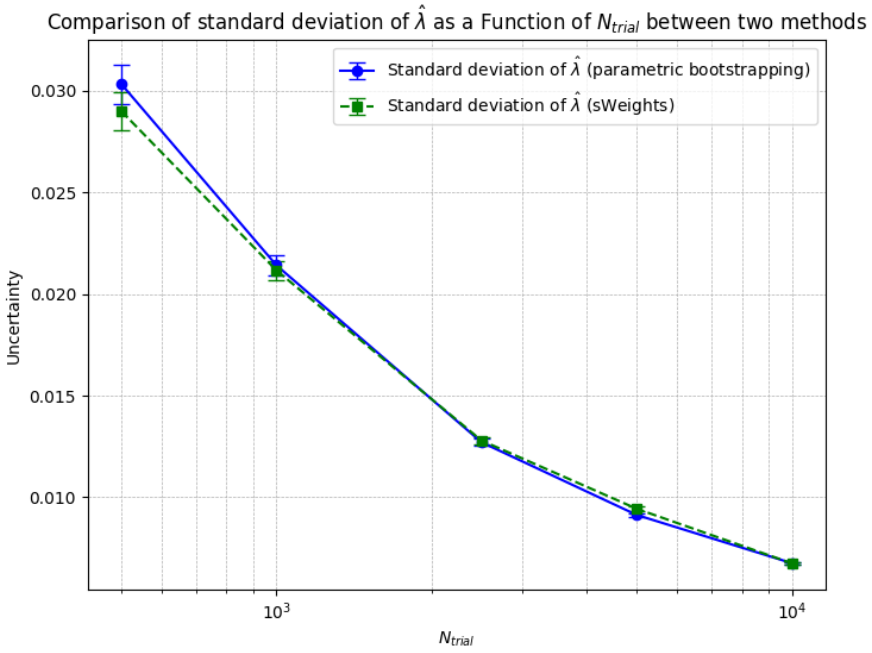

Figure 22: Comparison of the uncertainty of parameter λ as a function of sample sizes between the method of parametric bootstrapping and sWeights.