S1 Coursework Report

Keying Song (ks2146)

18th December 2024

Word Count: 2,893 words

Abstract

This coursework has compared two model fitting techniques for mixed signal and background components: multi-dimensional Extended Maximum Likelihood (EML) fitting and the *sWeights* method. Firstly, the mathematical form of the normalisation constant for the Crystal Ball distribution is proved. Subsequently, the p.d.f.s. of a mixed model in different directions and components were defined and their normalisation was verified by numerical integration. Then, 100,000 events from this model were generated to perform EML fitting, where parameter estimates, uncertainties, and computational execution times were evaluated. Next, the parametric bootstrapping was employed to assess bias and uncertainty trends for the parameter λ across various sample sizes. After that, the *sWeights* method was applied to split the signal and background components using the discriminating variable X, followed by the estimation of λ from the weighted Y distribution, enabling comparison with EML results. Finally, the report concluded with a comparative discussion, highlighting the scenarios in which the two methods is more appropriate.

Contents

1) Normalisation Constant

Problem Statement

(a) Show by a mathematical proof that the inverse of the normalisation constant, N, can be written

$$N^{-1} = \sigma \left[\frac{m}{\beta(m-1)} e^{-\beta^2/2} + \sqrt{2\pi}\Phi(\beta) \right], \tag{1.1}$$

where $\Phi(x)$ is the cumulative density of the standard normal distribution.

1.1 Proof

Recalling the definition of the Crystal Ball probability density function:

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

where $Z = \frac{X-\mu}{\sigma}$, $\quad \beta > 0, \quad m > 1$.

At this point, the integral of the pdf in the whole real number region is 1:

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

Change the variable of integration from X to Z: $dX = \sigma\, dZ$, Thus,

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

Then, split the integral into two parts.

First Integral

Consider:

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

Perform the substitution:

$$u = \frac{m}{\beta} - \beta - Z \quad \Longrightarrow \quad dZ = -\,du$$

So that:

$$\int_{-\infty}^{-\beta} \quad \Longrightarrow \quad \int_{\infty}^{\frac{m}{\beta}}$$

Thus, the first integral turns to:

$$\int_{\infty}^{\frac{m}{\beta}} \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} (u)^{-m}\, (-du)$$

$$= \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \int_{m/\beta}^{\infty} u^{-m}\, du$$

$$= \left(\frac{m}{\beta}\right)^m e^{-\beta^2/2} \left[\frac{u^{-m+1}}{-m+1} \right]_{m/\beta}^{\infty}$$

Since $m > 1$, $-m+1 < 0$, ensuring that $u^{-m+1} \to 0$ as $u \to \infty$ïijŇ hence:

$$\left[\frac{u^{-m+1}}{-m+1}\right]_{m/\beta}^{\infty} = 0 - \frac{(m/\beta)^{-m+1}}{-m+1} = \frac{(m/\beta)^{1-m}}{m-1}$$

Thus, the first integral:

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

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

$$= \frac{m}{\beta(m-1)} e^{-\beta^2/2}$$

Second Integral

Consider:

$$\int_{-\beta}^{\infty} e^{-Z^2/2}\, dZ$$

Recall the cumulative distribution function of the standard normal distribution:

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

From this, it can be seen that:

$$\begin{cases} \Phi(-\beta) & = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-\beta} e^{-t^2/2}\, dt \\ 1 & = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-t^2/2}\, dt \end{cases}$$

Applying subtraction to the above two equations yields:

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

Using the symmetry property $\Phi(-x) = 1 - \Phi(x)$, we get:

$$1 - \Phi(-\beta) = \Phi(\beta)$$

Hence,

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

Putting Two Integrals Together

Add up the two integral terms and recall the normalisation condition

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

So that:

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

Now, the proof is complete.

2) Model Definition and Validation

Problem Statement

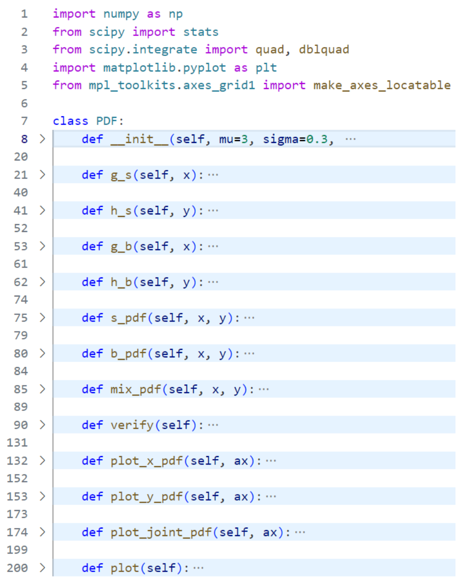
(b) Consider a statistical model of two independent random variables, $X \in [0,5], Y \in [0,10]$ which consists of a signal component and a background component.

$$
\begin{aligned}
f(X,Y) &= fs(X,Y) + (1-f)b(X,Y) && (2.1) \\
&= fg_s(X)h_s(Y) + (1-f)g_b(X)h_b(Y) && (2.2) \\
g_s(X) &= p_{crystalball}(X;\mu,\sigma,\beta,m) && (2.3) \\
h_s(Y) &= \lambda e^{-\lambda Y} && (2.4) \\
g_b(X) &= 1/5 && (2.5) \\
h_b(Y) &= \frac{1}{\sqrt{2\pi}\sigma_b} e^{\frac{(Y-\mu_b)^2}{2\sigma_b^2}} && (2.6)
\end{aligned}
$$

Write a small module in `Python` which defines these p.d.f.s and demonstrate by numeric integration that the definitions for $g_s(X), h_s(Y), g_b(X), h_b(Y), s(X,Y), b(X,Y)$ and $f(X,Y)$are properly normalised in the valid range. In order to check your definitions are correct it may be worth checking the normalisation with different values of the parameters.

2.1 Methodology

To solve this issue, I created a Python class named `PDF`, which is encapsulated in a module named `distribution`. There are totally 12 functions in this class, see figure(1). The first seven functions define all the probability density functions described in this context, followed by the function `verify` for verifying the normalisation of them while the last four functions implement the visualisation of them.



```python
import numpy as np
from scipy import stats
from scipy.integrate import quad, dblquad
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

class PDF:
    def __init__(self, mu=3, sigma=0.3, ···

    def g_s(self, x):···

    def h_s(self, y):···

    def g_b(self, x):···

    def h_b(self, y):···

    def s_pdf(self, x, y):···

    def b_pdf(self, x, y):···

    def mix_pdf(self, x, y):···

    def verify(self):···

    def plot_x_pdf(self, ax):···

    def plot_y_pdf(self, ax):···

    def plot_joint_pdf(self, ax):···

    def plot(self):···
```

Figure 1: The structure of `class PDF`

Using the library `scipy.stats`, I generated the *pdfs* of the Crystal Ball distribution and standard normal distribution, as well as the *cdfs* of them for calculating the truncated normalisation constant.

$$f(X) = N \cdot pdf(X), \quad X \in [min, max] \tag{2.7}$$

$$\implies \int_{min}^{maxy} N \cdot pdf(X) = N \cdot cdf(X)|_{min}^{max} = 1 \tag{2.8}$$

$$\implies N = 1/(cdf(X)|_{min}^{max}) \tag{2.9}$$

As for $h_s(Y)$ and $g_b(X)$, the probability density functions and the normalisation constants were defined manually:

$$h_s(Y) = N \cdot \lambda e^{-\lambda Y}, \quad Y \in [0, 10] \tag{2.10}$$

$$\implies N \cdot [\frac{e^{-\lambda Y}}{-\lambda}]_0^{10} = 1 \tag{2.11}$$

$$\implies N = 1/(1 - e^{-10\lambda}) \tag{2.12}$$

$$g_b(X) = N \cdot 1, \quad X \in [0, 5] \tag{2.13}$$

$$\implies N \cdot [X]_0^5 = 1 \tag{2.14}$$

$$\implies N = 1/5 \tag{2.15}$$

According to these independent p.d.f.s, the joint p.d.f.s of signal component, background component and the mixture of signal and background can be generated by performing the simple multiplication.

Subsequently, the verification function was created by integrating all the p.d.f.s over the domain of definition, one-dimensional integrals for one-variable function, two-dimensional integrals for two-variable functions.

2.2 Results

Importing this class, the normalisation of these p.d.f.s can be verified through the following simple code.

```python
from fit_methods import PDF

# verify with default parameters
print("Verify normalisation with default parameters:")
results = PDF().verify()

# verify with different parameters
test_params = [ {'mu': 2.5, 'beta': 2.0, 'f': 0.4, 'mu_b': 0.5},
                {'lamda': 0.5, 'sigma': 0.8, 'm':2.5, 'lamda': 0.5, 'mu_b': 1.3, 'sigma_b': 2.1}]

for i, params in enumerate(test_params, 1):
    print(f"\nVerify normalisation with parameters group {i}:")
    results = PDF(**params).verify()
```

I tried the default parameters and other two different groups of parameters to test the normalisation, the testing groups are shown in table(1). The experimental results demonstrate that under the three groups of test parameters,

Table 1: The label distribution in training set

	μ	σ	β	m	f	λ	μ_b	σ_b	result
group 0	3.0	0.3	1.0	1.4	0.6	0.3	0.0	2.5	pass
group 1	2.5	0.3	2.0	1.4	0.4	0.3	0.5	2.5	pass
group 2	3.0	0.8	1.0	2.5	0.6	0.5	1.3	2.1	pass

the integrals of the seven probability density functions within their valid domains all equal one, thus confirming the success of normalisation.

3) Distribution Plots

Problem Statement

(c) Plot the one dimensional projections of these distributions in both the variables X and Y. Ensure that your plots contain the total pdf as well as a breakdown of the signal and background components. You should also make a two-dimensional plot of the joint probability density.

3.1 Methodology

To observe the signal and background components within the X, Y and joint distributions, I utilised the visualisation features of the PDF module. Having imported the packages, the code used here is very simple:

```python
from fit_methods import PDF
import matplotlib.pyplot as plt

fig = PDF().plot()
plt.show()
```

By invoking the `plot` function, a plot comprising three subplots illustrating the probability density distributions was generated, as shown in figure (2).

3.2 Results

When no parameters are passed to the `PDF()` object, the model parameters are set to their default values, which correspond to those of group 0 in the table(2).

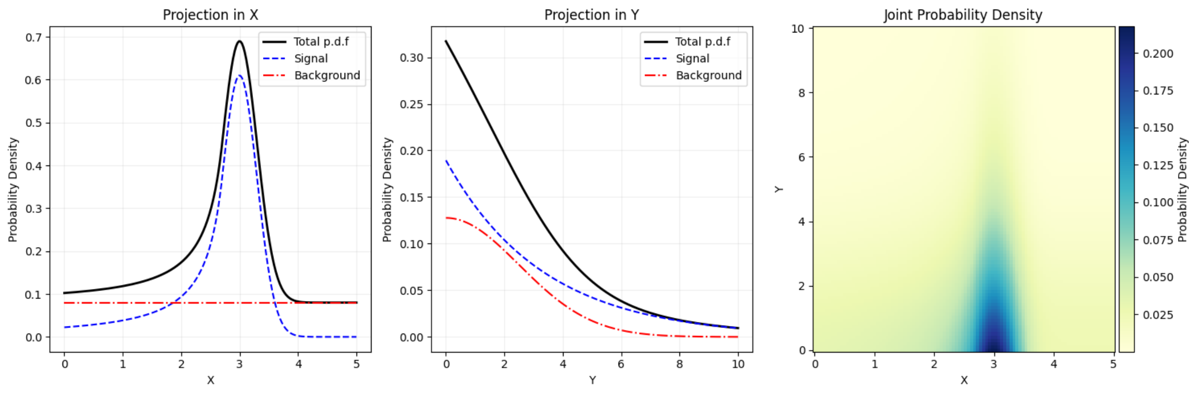


Figure 2: Probability density function projections in X , Y (centre) and joint density (right).

4) EML Fitting in Two Dimensions

Problem Statement

Generate a high-statistics sample from the joint distribution, containing a total of 100,000 events, and then perform an extended maximum likelihood fit to estimate the nine parameters. You should also determine an estimate of the uncertainty on these estimates. Please provide an evaluation of the execution time, using the `timeit` library, averaged over 100 calls for the following processes:
(i) Calling np.random.normal (size =100000) (this simply sets a standard benchmark for your machine);
(ii) The call which generates your sample of 100,000 events;
(iii) The call which performs the fit to the sample to estimate the parameters.
Please present the last two numbers relative to the first.

4.1 Methodology

For this question, a generator should be built firstly. I created a class named `Generator` inside the module `generation`, which generates the required number of samples based on the model parameters passed in, and returns the distribution of the samples in both X and Y. We can plot their histograms and compare them to the real probability density distribution presented in figure(2), in order to check the correctness of generated samples.

Having the samples generated, we can then perform the extended maximum likelihood fit to estimate the nine parameters, including the added parameter, number of events, N. A class `Fitter` inside the module `fitting` was designed specifically for this task. When passing the generated data and a group of parameters (optional) in it, the main function `fit` began with the nine-parameter set as the initial guess for the optimisation. Following this, the optimisation process that depends on the `iminuit` package was implemented, which minimised the negative log likelihood function with a extended poisson term.

After the fitting, the nine estimated parameters and their uncertainties were returned. It is worth noting that the uncertainties were calculated by `Minuit` in module `iminuit`. I chose the Hesse method, one of the two ways for parameter uncertainty in `Minuit`, which returns the parameter parabolic errors using a property called *Minuit.errors*, as discussed in the 'Parameter uncertainties, covariance, and confidence intervals/regions' section of the iminuit Basics tutorial (Dembinski et al., 2024).

Subsequently, the evaluation of the execution time of fitting was performed. By executing the processes 100 times each, and using the `timeit` library to find the average times respectively for the three operations, the relation between the sample generation time and the fitting time of our model to the generation time of normal distribution samples can be easily calculated by the ratios.

4.2 Result

Firstly I generated 100,000 samples with default parameters, then plotted their density histograms in X, Y and joint cases, shown in figure(3).

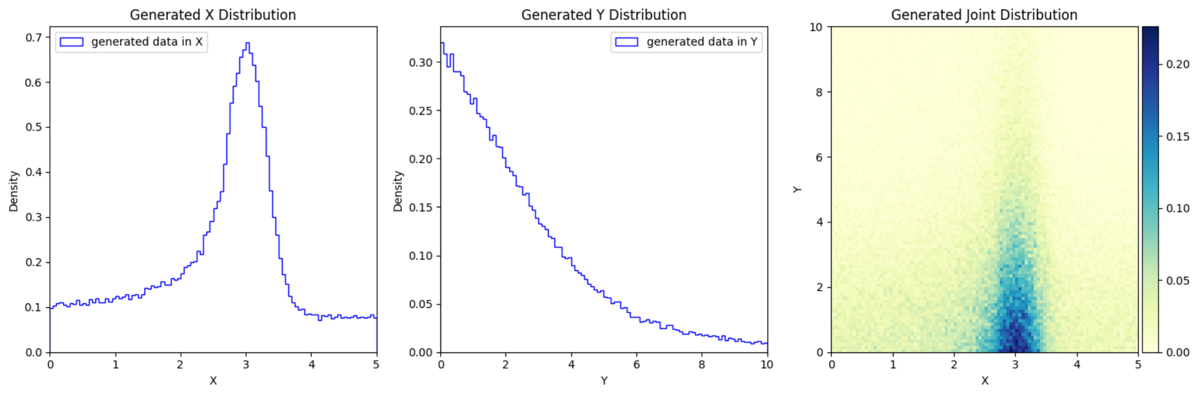


Figure 3: Probability density function projections in X (left) , Y (centre) and joint density (right).

Comparing the true p.d.f distributions in Figure(2), it can be seen that these generated data represent the distribution of this model very well because of their high similarity.

Exploiting the generated data, the `fit` function returned the parameter estimates and uncertainties, presented in table(2).

Parameter	**Estimate**	**Uncertainty**	**True Value**
μ	3.0018	$\pm$ 0.0037	3.0000
σ	0.2980	$\pm$ 0.0035	0.3000
β	0.9924	$\pm$ 0.0311	1.0000
m	1.4200	$\pm$ 0.0886	1.4000
f	0.5962	$\pm$ 0.0050	0.6000
λ	0.2994	$\pm$ 0.0029	0.3000
μ_b	-0.0079	$\pm$ 0.1113	0.0000
σ_b	2.5130	$\pm$ 0.0515	2.5000
N	100000.0800	$\pm$ 447.2136	100000.0000

Table 2: Parameter estimates, uncertainties, and true values.

When it comes to the evaluation of the execution times, the output is listed in table (3). It can be found that the generation and fitting processes in 2-dimension are time-consuming.

Task	**Execution Time (s)**	**Relative to First Task**
Normal distribution	0.0015	1.00x
Sample generation	0.0190	12.65x
Parameter fitting	5.8407	3893.89x

Table 3: Execution times for different tasks and their relative comparison.

5) Parametric Bootstrapping

Problem Statement

(e) Now run a simulation study using parametric bootstrapping (with an ensemble of at least 250 samples) from the true probability distribution. You should trial sample sizes of 500, 1000, 2500, 5000 and 10000, including a Poisson variation on the sample size. Determine whether you see any bias on the λ parameter as a function of the sample size. Also determine the expected uncertainty on λ as a function of the sample size.

5.1 Methodology

In order to perform the study with parametric bootstrapping, I developed a `Bootstrap` class in a module named `bootstrap`, which encapsulated two functions, `toy-study` and `uncertainties`, to respectively run the fitting using parametric bootstrapping and to calculate the 5 standard deviations of λ that arise when iterating the 250 sub-sets in the 5 ensembles. Using the standard errors to represent uncertainties of λ, a plot of uncertainty as a function of sample size can be drawn.

The main workflow of `Bootstrap` class is broken down in the following steps:

1. The true parameters were input into the generator (with `Generator` imported) inside the `generation` module, so that it was ready to generate the sample sets from our model;
2. An ensemble size of 250 was selected, which means the number of sample sets in the ensemble is 250;
3. The first size of sample set was set to 500, and the corresponding ensemble was constructed by randomly generating 500 samples 250 times;
4. An extended maximum likelihood fit supported by `fitting` module was performed for each sample set within the ensemble, and the outputs were recorded;
5. The above process was iterated at different sample sizes, i.e., 500, 1000, 2500, 5000, and 10000.
6. The outputs recorded were visualised by function `uncertainties`.

At a certain sample set size, I used the average of 250 estimated λ to calculate the bias while the standard error (SE, the standard deviation of the ensemble) of λ was utilised to represent the uncertainty at that size of sample set:

$$Bias(ss) = \overline{\lambda}(ss) - \lambda_{true} \tag{5.1}$$

$$SE(ss) = [\frac{1}{250}\sum_{i}^{250}(\lambda_i(ss) - \overline{\lambda}(ss))^2]^{1/2} \tag{5.2}$$

, where

$$\overline{\lambda}(ss) = \frac{1}{250}[\lambda_1(ss) + \lambda_2(ss) + ... + \lambda_{250}(ss)] \tag{5.3}$$

$$ss = [500, 1000, 2500, 5000, 10000] \tag{5.4}$$

Consequently, plots of the SE and Bias with the size of the sample set (ss) can be drawn and we can observe whether the trend of change satisfies the theoretical relationship (from S1 lecture note, chapter 2, page 40):

$$SE(ss) = \frac{SD}{\sqrt{ss}} \tag{5.5}$$

, where SD is the standard deviation of λ when sampling size equals to one.

5.2 Results

For easy observation, I used logarithmic axes to visualise the Bias and uncertainty as shown in figure (4).

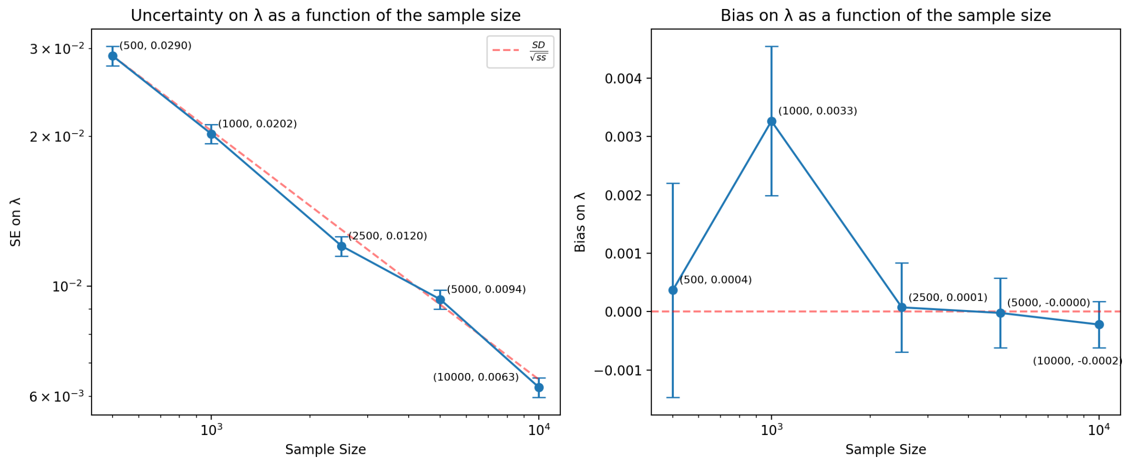


Figure 4: The bias (left) and expected uncertainty (right) on λ as the functions of the sample size

The right panel illustrates the bias on λ over different sampling sizes with the zero value reference (red dashed line), and it can be seen that it fluctuates randomly at 10^{-3} level.

The left panel shows that the trend of uncertainty with sampling size (blue solid line) fits well with the theoretical relationship (red dashed line), so we can calculate the uncertainty of λ using the bootstrapping method at other sample sizes from this functional relationship:

$$Uncertainty(ss) = \frac{SD}{\sqrt{ss}} \simeq \frac{\sqrt{500} \times 0.0290}{\sqrt{ss}} = \frac{0.6485}{\sqrt{ss}} \tag{5.6}$$

When the sample size is 100,000:

$$Uncertainty \simeq 0.0021 \tag{5.7}$$

Compare it to the result of the single fit in (2), it can be seen that the resampling process reduced the uncertainty of the estimation.

6) sWeights Fitting

Problem Statement

(f) Using the samples produced above, perform an extended maximum likelihood fit in just the X variable and use it to produce *sWeights* which project out the signal density in Y. Then use an estimation method of your choice to determine the λ based on the weighted sample in Y. Compare the bias and the uncertainty to your findings from the previous part.

6.1 Methodology

The module I wrote manually for this part was named `sweight`, in which the class `Sweightor` consisted of four functions:

Function `EMLx_fit` performed an extended maximum likelihood fit in just the X variable, supported by `Minuit` class and `ExtendedUnbinnedNLL` class in `iminuit` module and `iminuit.cost` module. The output of it showed the estimated parameters for reconstructing the signal-background models in the X direction.

Yet, reconstructing the model in Y direction still required additional operations, so that the function `est_lambda` was developed to estimate the Y-model parameters by minimising the one-dimensional negative-log-likelihood function with the weights, while function `do_sWeight` using the fitting results from `EMLx_fit` to performed the *sWeights* analysis and produced signal weights which project out the signal density in Y. The minimising process in the former exploited the `iminuit` again and the weighting process in the latter took use of the `SWeight` class from `sweights` module.

Finally, the function `plot_results` not only plotted the 'sweight+bweight' check image to evaluate the weight functions, but also estimated results in Y, as well as the true distribution of Y for comparison. It is worth mentioning that a uniform grid was newly created when performing weights validation for clear observation. Rather than using the actual data points, which may cluster in certain regions and result in visual congestion.

The usage of this module for fitting is quit straightforward:

```
from fit_methods import Generator, Sweightor

true_params = [3, 0.3, 1, 1.4, 0.6, 0.3, 0, 2.5]
x, y = Generator(true_params).generate_sample(100000)

result = Sweightor(x, y).do_sWeight(true_params[5])
```

In the code above, I finished the fitting process in 4 lines with the modules imported: The true model parameters were passed into the generator to generate 100000 true samples, of which the X data was put into the `Sweightor` to fit the X-model and produce the weights, while Y data was only used with the produced weights to reconstruct the Y-model. Since all the optimisation processes dealt with the one-dimensional case, the processing was very fast, about 3 seconds on average. In addition, the true λ value was passed into `do_sWeight` function for bias calculation. The default output is shown below:

```
    PDF normalisations:
     0 1.0000000001631473
     1 1.0000000000000002
    Integral of w*pdf matrix (should be close to the
                identity):
    [[ 1.00010971e+00 -1.85952565e-04]
     [-1.13097265e-04  1.00019457e+00]]
    Check of weight sums (should match yields):
    Component   | sWeightSum |   Yield    |   Diff     |
    ---------------------------------------------------
      0         | 60334.9993 | 60334.9993 |    -0.00% |
      1         | 39665.1382 | 39665.1382 |     0.00% |
```

6.2 Results

After passing the results from `do_sWeight` as well as the true parameters, the function `plot_results` gave the visualisation result as in Figure(5).

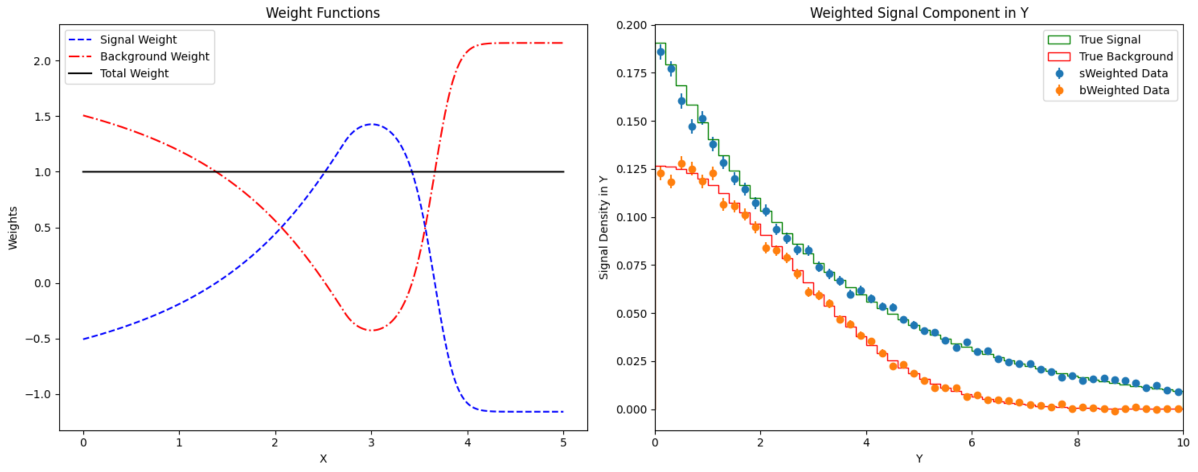


Figure 5: sWeights generation: (Left) Check the sum of sweights and bweights. (Right) Weighted signal component in Y.

The left panel verifies that the sum of signal (blue dashed line) and background (red dot-dashed line) weights equals unity at each point (black solid line), while the right panel illustrates the separated signal (blue) and background (orange) densities in fitted Y-model, as well as their true distributions (green and red histograms with 'step' type). The consistency between the weighted data and their respective true distributions successfully verified that the *sweights* technique can extract the correct shape of each component without prior knowledge of the distributions in Y.

6.3 Bias and Uncertainty Comparisons

Using the obtained weights and the `est_lambda` function to fit the lambda, the fitting results were printed in table (4). Compare it to the uncertainty and bias in table (2), it can be seen that the 0.0024 uncertainty is slightly better than 0.0029 from the 2-dimentional EML fit, whereas the -0.0007 bias is very close to -0.0006 from directly fit. In addition, the bias and uncertainty from `sWeights` method were both slightly larger than those from EML fit with 250 resamplings (in figure (4) and equation (5.7)) .

Parameter	**Value**
True λ	0.3000
Estimated λ	0.2993
Uncertainty	0.0024
Bias	-0.0007

Table 4: sWeights analysis results for λ.

7) Comparison of Fitting Methods

Problem Statement

(g) Compare and contrast these two methods. Explain the potential drawbacks and disadvantages of one over the other. State which you think is most appropriate and why. Explain in which scenarios one method might be preferred over the other.

7.1 Comparison

Until now, I have applied two main methods of model parameters fitting: the multi-dimensional extended maximum likelihood (EML) fit and the weighted fit method with *sWeights*. They have several similarities and differences:

Similarities

- Both methods aim to estimate parameters from data containing mixed signal and background components;
- Both exploit maximum likelihood estimation for parameter optimisation;
- Both methods can handle large datasets, such as sample sizes on the 100,000 scale.

Differences

- The EML method directly performs a mult-dimensional fit to all variables, while the sWeights method separates the process into two steps and only apply the optimisation in one dimension;
- The EML method needs explicit modelling of all component distributions, whereas sWeights method can rebuild the target distribution without assuming the functional form;
- The computational complexity of: the former becomes more intensive with additional dimensions, while the latter maintains relatively stable;

- Uncertainty estimation is direct in the former through the Hessian matrix, but requires additional weight uncertainties in the latter.

7.2 Potential Drawbacks

Multi-dimensional EML fit

- Computational complexity increases significantly with model dimensions and dataset size;
- Highly rely on a correct description of the model in all dimensions and components, which can affect all parameter estimates.

***sWeights* fit**

- The different variables need to be independent;
- The performance is highly dependent on the quality of signal-background separation of the discriminant variable (X in this case);
- It may lead to larger uncertainties due to the two-step nature of the method (but this was not the case in this time).

7.3 Selections in Different Scenarios

For the distribution model in this case, I would choose the sWeights method for the parameters estimation. Firstly, the signal and background components separate well in the discriminating variable X, as well as the independency between the two variables, which are crucial for effective sWeights calculation.

In addition, I think that the advantage of the efficiency of this method outweigh the disadvantage of the relatively large uncertainties and biases than EML fit. Comparing the results in tables (2) and (4), it can be seen that this method gives very close results but within shorter running time compared to single EML fit without the resampling.

However, it is worth noting that in scenarios where the variables are correlated with each other, or where both their signal and background components are difficult to distinguish, the direct EML fit may be more appropriate despite its higher computational cost.

References

Hans Dembinski, Piti Ongmongkolkul, et al. *Basics iminuit 2.30.2 documentation*, 2024. URL `https://scikit-hep.org/iminuit/notebooks/basic.html`. Accessed: 2024-12-15.

Appendix

Declaration: No auto-generation tools were used in this report except for generation of BibTeX references.