Principles of Data Science

Coursework Submission

A REPORT PRESENTED

BY

STEVEN DILLMANN

Departments
Department of Applied Mathematics and Theoretical Physics
Department of Physics (Cavendish Laboratory)
Institute of Astronomy

Degree
MPhil Data Intensive Science

Module
S1 Principles of Data Science

Supervision
Dr Matt Kenzie

ST JOHN'S COLLEGE
UNIVERSITY OF CAMBRIDGE
17TH DECEMBER 2023

List of Figures

List of Tables

List of Listings

Contents

Section A Word Count: 999
Section B Word Count: 1997
Total LaTeX Word Count: 2996/3000 (excluding Bibliography and Appendix)

1 Section A

1.1 Question 1 (a)

The statistical model below depends on the continuous random variable M, where M takes values within the range $M \in [5, 5.6]$. The model consists of a background (exponential decay distribution with rate parameter λ) and a signal (normal distribution with mean μ and standard deviation σ). The probability density functions (PDFs) of background and signal are given by Equations (1) and (2) respectively:

$$b(M; \lambda) = \begin{cases} \lambda e^{-\lambda M} & \text{for} \quad M \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$s(M; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(M-\mu)^2}{2\sigma^2}} \tag{2}$$

The total PDF is given by Equation (3) with the parameter f governing the signal fraction:

$$p(M; f, \lambda, \mu, \sigma) = (1 - f) \cdot b(M; \lambda) + f \cdot s(M; \mu, \sigma) \tag{3}$$

In the following proof, we show that the total PDF in Equation (3) is properly normalised in the range $M \in [-\infty, \infty]$ by demonstrating that the condition in **Definition 1** holds.

Definition 1. *A probability distribution function $f(x)$ is said to be normalised over the range $x \in [a, b]$ if the following condition holds:*

$$\int_a^b f(x)\, dx = 1$$

Proof.

Step 1: Show that** $b(M; \lambda)$ **is normalised
Demonstrate that the integral of $b(M; \lambda)$ over the range $M \in [-\infty, \infty]$ equals unity:

$$\begin{aligned} \int_{-\infty}^{\infty} b(M; \lambda)\, dM = \int_0^{\infty} b(M; \lambda)\, dM = \int_0^{\infty} \lambda e^{-\lambda M}\, dM = -\left[e^{-\lambda M}\right]_0^{\infty} \\ = -\left[\lim_{M\to\infty}\left(e^{-\lambda M}\right) - e^{-\lambda M}\Big|_{M=0}\right] = -(0-1) = 1 \end{aligned}$$

$$\therefore b(M; \lambda) \text{ is normalised}$$

Step 2: Show that** $s(M; \mu, \sigma)$ **is normalised
Demonstrate that the integral of $s(M; \lambda)$ over the range $M \in [-\infty, \infty]$ equals unity:

First, we show that the Gaussian integral $I = \int_{-\infty}^{\infty} e^{-x^2}\, dx$ equals to $\sqrt{\pi}$. Using a dummy variable switch from x to y, we can write I^2 as:

$$I^2 = \left(\int_{-\infty}^{\infty} e^{-x^2}\, dx\right)\left(\int_{-\infty}^{\infty} e^{-x^2}\, dx\right) = \left(\int_{-\infty}^{\infty} e^{-x^2}\, dx\right)\left(\int_{-\infty}^{\infty} e^{-y^2}\, dy\right) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\, dx\, dy$$

Using a polar coordinate transformation, i.e. $r^2 = x^2 + y^2$ with $x = r\cos\theta$ and $y = r\sin\theta$, gives $dx\,dy = \begin{vmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{vmatrix} dr\,d\theta = \begin{vmatrix} \cos\theta & -r\sin\theta \\ \sin\theta & r\cos\theta \end{vmatrix} dr\,d\theta = (r\cos^2\theta + r\sin^2\theta) dr\,d\theta = r\,dr\,d\theta$. We can now write I^2 as:

$$I^2 = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-(x^2+y^2)}\,dx\,dy = \int_0^{2\pi}\int_0^{\infty} e^{-r^2}\,r\,dr\,d\theta = 2\pi\int_0^{\infty} e^{-r^2}\,r\,dr$$

Using the substitution $u = r^2$ and $r\,dr = \frac{du}{2}$, we compute I^2 and hence I:

$$\begin{aligned} I^2 &= 2\pi\int_0^{\infty} e^{-r^2}\,r\,dr = \pi\int_0^{\infty} e^{-u}\,du = -\pi\Big[e^{-u}\Big]_0^{\infty} \\ &= -\pi\Big[\lim_{u\to\infty}\left(e^{-u}\right) - e^{-u}\big|_{u=0}\Big] = -\pi\,(0-1) = \pi \end{aligned}$$

$$\Rightarrow I = \sqrt{I^2} = \int_{-\infty}^{\infty} e^{-x^2}\,dx = \sqrt{\pi}$$

By substituting $x = \frac{M-\mu}{\sqrt{2}\sigma}$ and $dM = \sqrt{2}\sigma\,dx$, we show that $s(M;\lambda)$ is normalised:

$$\int_{-\infty}^{\infty} s(M;\mu,\sigma)\,dM = \int_{-\infty}^{\infty}\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(M-\mu)^2}{2\sigma^2}}\,dM = \frac{1}{\sqrt{\pi}}\underbrace{\int_{-\infty}^{\infty} e^{-x^2}\,dx}_{=\sqrt{\pi}} = \frac{1}{\sqrt{\pi}}\cdot\sqrt{\pi} = 1$$

$$\therefore s(M;\mu,\sigma) \text{ is normalised}$$

Step 3: Show that $p(M;f,\lambda,\mu,\sigma)$ ***is normalised***
We demonstrate that the integral of $p(M;f,\lambda,\mu,\sigma)$ over the range $M \in [-\infty,\infty]$ equals unity:

$$\begin{aligned} \int_{-\infty}^{\infty} p(M;f,\lambda,\mu,\sigma)\,dM &= \int_{-\infty}^{\infty}\Big[(1-f)\cdot b(M;\lambda) + f\cdot s(M;\mu,\sigma)\Big]\,dM \\ &= (1-f)\underbrace{\int_{-\infty}^{\infty} b(M;\lambda)\,dM}_{=1\text{ (from Step 1)}} + f\underbrace{\int_{-\infty}^{\infty} s(M;\mu,\sigma)\,dM}_{=1\text{ (from Step 2)}} \\ &= (1-f)\cdot 1 + f\cdot 1 = 1 - f + f = 1 \end{aligned}$$

$$\therefore p(M;f,\lambda,\mu,\sigma) \text{ is normalised}$$

$\square$

1.2 Question 1 (b)

As the random variable M is restricted by lower and upper limits (α,β), the PDF needs to be properly normalised over the range $M \in [\alpha, \beta]$. We define the normalised PDF of the background and signal as given by Equations (4) and (5) with the normalisation constants N_b and N_s respectively:

$$b_{\alpha\beta}(M; \lambda) = \frac{1}{N_b} \cdot b(M; \lambda) \tag{4}$$

$$s_{\alpha\beta}(M; \mu, \sigma) = \frac{1}{N_s} \cdot s(M; \mu, \sigma) \tag{5}$$

The total normalised PDF is then given by Equation (6):

$$p_{\alpha\beta}(M; f, \lambda, \mu, \sigma) = (1 - f) \cdot b_{\alpha\beta}(M; \lambda) + f \cdot s_{\alpha\beta}(M; \mu, \sigma) \tag{6}$$

In the following derivation, we make use of **Definition 2** to obtain an expression for the PDF in Equation (6) in terms of the parameters $\boldsymbol{\theta} = (f, \lambda, \mu, \sigma)$.

Definition 2. *The cumulative density function, defined as* $F(X) = \int_{-\infty}^{X} f(X')\, dX'$, *for the exponential decay and normal distribution is given by* $F_b(X)$ *and* $F_s(X)$ *respectively:*

$$F_b(X) = \int_{-\infty}^{X} b(X'; \lambda)\, dX' = 1 - e^{-\lambda X}$$

$$F_s(X) = \int_{-\infty}^{X} s(X'; \mu, \sigma)\, dX' = \Phi\left(\frac{X - \mu}{\sigma}\right) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{X - \mu}{\sigma\sqrt{2}}\right)\right]$$

Derivation.

Step 1: Ensure that $p_{\alpha\beta}(M; f, \lambda, \mu, \sigma)$ ***is normalised and rearrange for*** N_b ***and*** N_s
To ensure that $p_{\alpha\beta}(M; f, \lambda, \mu, \sigma)$ is normalised, we equate the integral over $M \in [\alpha, \beta]$ to 1:

$$\begin{aligned}\int_{\alpha}^{\beta} p_{\alpha\beta}(M; f, \lambda, \mu, \sigma)\, dM &= \int_{\alpha}^{\beta} \Big[(1 - f) \cdot b_{\alpha\beta}(M; \lambda) + f \cdot s_{\alpha\beta}(M; \mu, \sigma)\Big]\, dM \\ &= (1 - f) \int_{\alpha}^{\beta} b_{\alpha\beta}(M; \lambda)\, dM + f \int_{\alpha}^{\beta} s_{\alpha\beta}(M; \mu, \sigma)\, dM \\ &= 1\end{aligned}$$

We see that $p_{\alpha\beta}(M; f, \lambda, \mu, \sigma)$ is normalised if $b_{\alpha\beta}(M; \lambda)$ and $s_{\alpha\beta}(M; \mu, \sigma)$ are normalised, hence we equate the integrals over $M \in [\alpha, \beta]$ to 1 and rearrange for N_b and N_s:

$$\int_{\alpha}^{\beta} b_{\alpha\beta}(M; \lambda)\, dM = \int_{\alpha}^{\beta} \frac{1}{N_b} \cdot b(M; \lambda)\, dM = 1 \quad \Rightarrow \quad \underline{\underline{N_b = \int_{\alpha}^{\beta} b(M; \lambda)\, dM}}$$

$$\int_{\alpha}^{\beta} s_{\alpha\beta}(M; \mu, \sigma)\, dM = \int_{\alpha}^{\beta} \frac{1}{N_s} \cdot s(M; \mu, \sigma)\, dM = 1 \quad \Rightarrow \quad \underline{\underline{N_s = \int_{\alpha}^{\beta} s(M; \mu, \sigma)\, dM}}$$

Step 2: Find N_b ***and*** N_s

We evaluate the integrals of $b_{\alpha\beta}(M;\lambda)$ and $s_{\alpha\beta}(M;\mu,\sigma)$ over $M \in [\alpha,\beta]$ to find N_b and N_s in terms of $\boldsymbol{\theta} = (f,\lambda,\mu,\sigma)$:

$$N_b = \int_\alpha^\beta b(M;\lambda)\,dM = \underbrace{\int_{-\infty}^\beta b(M;\lambda)\,dM}_{=F_b(\beta)} - \underbrace{\int_{-\infty}^\alpha b(M;\lambda)\,dM}_{=F_b(\alpha)} = F_b(\beta) - F_b(\alpha)$$

$$= \left[1 - e^{-\lambda\beta}\right] - \left[1 - e^{-\lambda\beta}\right] = \underline{\underline{e^{-\lambda\alpha} - e^{-\lambda\beta}}}$$

$$N_s = \int_\alpha^\beta s(M;\mu,\sigma)\,dM = \underbrace{\int_{-\infty}^\beta s(M;\mu,\sigma)\,dM}_{=F_s(\beta)} - \underbrace{\int_{-\infty}^\alpha s(M;\mu,\sigma)\,dM}_{=F_s(\alpha)} = F_s(\beta) - F_s(\alpha)$$

$$= \frac{1}{2}\left[1 + \text{erf}\left(\frac{\beta-\mu}{\sigma\sqrt{2}}\right)\right] - \frac{1}{2}\left[1 + \text{erf}\left(\frac{\alpha-\mu}{\sigma\sqrt{2}}\right)\right] = \underline{\underline{\frac{1}{2}\left[\text{erf}\left(\frac{\beta-\mu}{\sigma\sqrt{2}}\right) - \text{erf}\left(\frac{\alpha-\mu}{\sigma\sqrt{2}}\right)\right]}}$$

Step 3: Plug in N_b ***and*** N_s ***into*** $p_{\alpha\beta}(M;f,\lambda,\mu,\sigma)$

First, we plug in N_b and N_s to find expressions for $b_{\alpha\beta}(M;\lambda)$ and $s_{\alpha\beta}(M;\mu,\sigma)$ in terms of $\boldsymbol{\theta} = (f,\lambda,\mu,\sigma)$:

$$b_{\alpha\beta}(M;\lambda) = \frac{1}{N_b}\cdot b(M;\lambda) = \frac{1}{e^{-\lambda\alpha} - e^{-\lambda\beta}}\cdot b(M;\lambda)$$

$$s_{\alpha\beta}(M;\mu,\sigma) = \frac{1}{N_s}\cdot s(M;\mu,\sigma) = \frac{2}{\text{erf}\left(\frac{\beta-\mu}{\sigma\sqrt{2}}\right) - \text{erf}\left(\frac{\alpha-\mu}{\sigma\sqrt{2}}\right)}\cdot s(M;\mu,\sigma)$$

Then, we plug in $b_{\alpha\beta}(M;\lambda)$ and $s_{\alpha\beta}(M;\mu,\sigma)$ to find an expression for $p_{\alpha\beta}(M;f,\lambda,\mu,\sigma)$ in terms of $\boldsymbol{\theta} = (f,\lambda,\mu,\sigma)$:

$$\begin{aligned} p_{\alpha\beta}(M;f,\lambda,\mu,\sigma) &= (1-f)\cdot b_{\alpha\beta}(M;\lambda) + f\cdot s_{\alpha\beta}(M;\mu,\sigma) \\ &= \underline{\underline{\frac{(1-f)}{e^{-\lambda\alpha} - e^{-\lambda\beta}}\cdot b(M;\lambda) + \frac{2\cdot f}{\text{erf}\left(\frac{\beta-\mu}{\sigma\sqrt{2}}\right) - \text{erf}\left(\frac{\alpha-\mu}{\sigma\sqrt{2}}\right)}\cdot s(M;\mu,\sigma)}} \end{aligned}$$

1.3 Question 1 (c)

We code the derived expressions for $b_{\alpha\beta}(M;\lambda)$, $s_{\alpha\beta}(M;\mu,\sigma)$ and $p_{\alpha\beta}(M;f,\lambda,\mu,\sigma)$ in Python [1] with the `numpy` library [2] and `scipy.stats` module [3]. Testing different values for $\boldsymbol{\theta} = (f,\lambda,\mu,\sigma)$ and using a numerical integration method in the `scipy.integrate` package, we can show that the total PDF integrates to 1 over the range $M \in [\alpha,\beta]$ with $\alpha = 5.0$ and $\beta = 5.6$.

Numerical integration involves finite precision arithmetic leading to small discrepancies between the numerical and exact integral. We thus introduce a tolerance level of 1×10^{-6} to determine whether a test passed or failed. The test results are shown in Table 1. All tests pass.

Table 1: Test parameters and results for the numerical integration of $p_{\alpha\beta}(M;f,\lambda,\mu,\sigma)$ over the range $M \in [5.0, 5.6]$ with a tolerance level of 1×10^{-6} between the numerical and exact integral.

f	λ	μ	σ	Integral	Test Result
0.1	0.2	5.1	0.1	1.0	Passed
0.3	0.6	5.2	0.3	1.0	Passed
0.5	1.0	5.3	0.5	1.0	Passed
0.7	1.4	5.4	0.7	1.0	Passed
0.9	1.8	5.5	0.9	1.0	Passed

1.4 Question 1 (d)

Given the true values of the parameters are $f_{true} = 0.1$, $\lambda_{true} = 0.5$, $\mu_{true} = 5.28$, $\sigma_{true} = 0.018$, we plot the true signal, background and total PDFs with the `matplotlib` library [4]. Figure 1 shows $p_{\alpha\beta}(M; f_{true}, \lambda_{true}, \mu_{true}, \sigma_{true})$ overlaid by its background and signal components, $(1 - f_{true}) \cdot b_{\alpha\beta}(M; \lambda_{true})$ and $f_{true} \cdot s_{\alpha\beta}(M; \mu_{true}, \sigma_{true})$. Figure 2 shows $p_{\alpha\beta}(M; f_{true}, \lambda_{true}, \mu_{true}, \sigma_{true})$ overlaid by the full PDFs for the background and signal, $b_{\alpha\beta}(M; \lambda_{true})$ and $s_{\alpha\beta}(M; \mu_{true}, \sigma_{true})$.

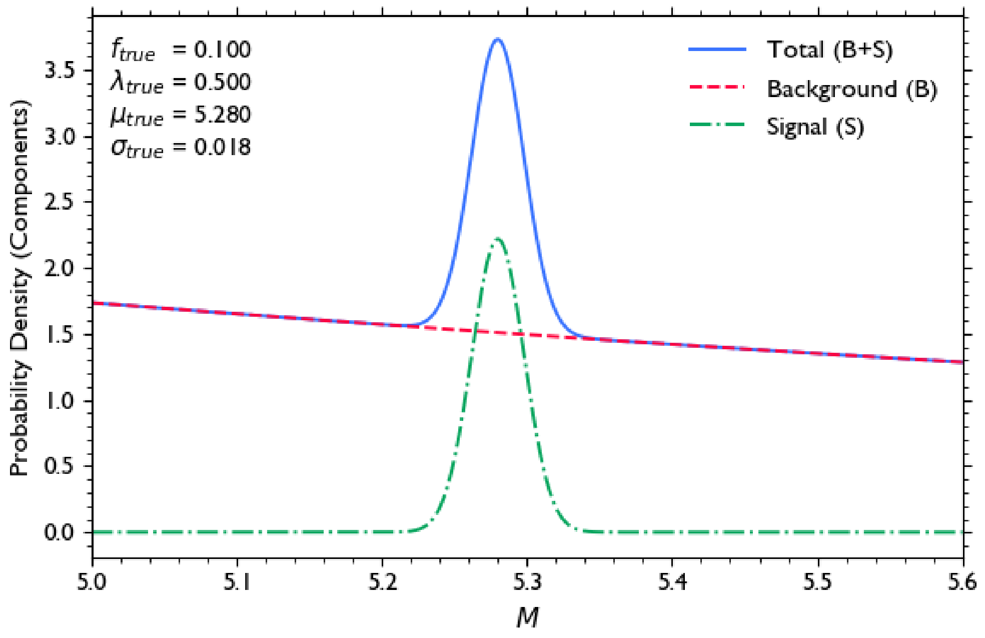

Figure 1: True total PDF overlaid by its background and signal components with parameters $f_{true} = 0.1$, $\lambda_{true} = 0.5$, $\mu_{true} = 5.28$ and $\sigma_{true} = 0.018$.

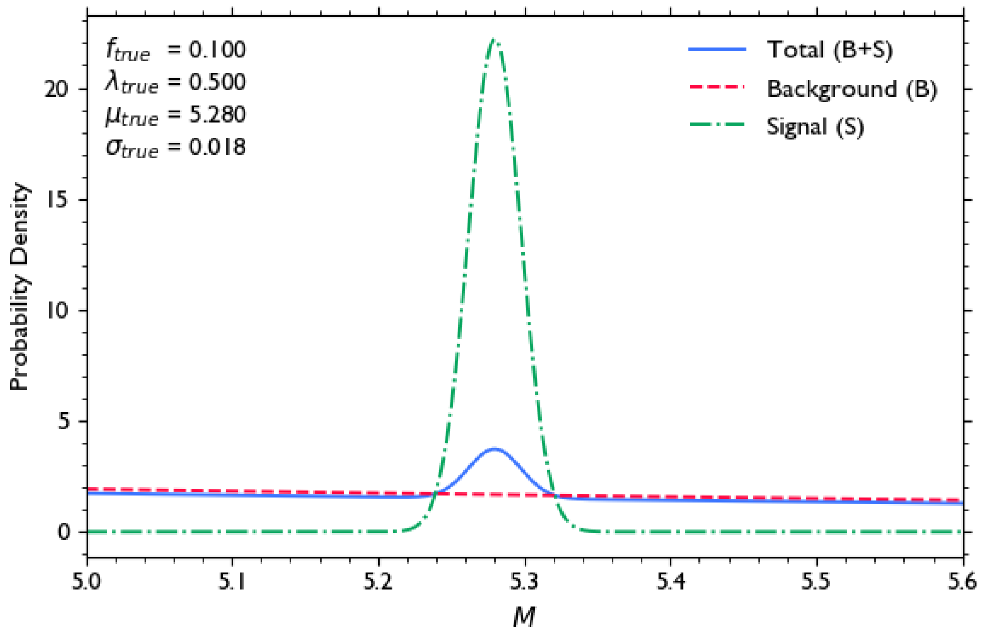

Figure 2: True signal, background and total PDF with parameters $f_{true} = 0.1$, $\lambda_{true} = 0.5$, $\mu_{true} = 5.28$ and $\sigma_{true} = 0.018$.

1.5 Question 1 (e)

We can now generate a high-statistics sample from the total PDF and use an estimation method to obtain model parameter estimates.

Sample Generation

We generate a high-statistics sample containing 100,000 events using a numerical version of the inverse cumulative distribution function (inverse CDF) method, here also called the percentage point function (PPF) method. We use this method due to the following reasons:

- The analytical PPF is not readily available for truncated exponential and normal distributions in the `scipy.stats` or `numba_stats` modules [5].
- The numerical PPF method achieves maximum sampling efficiency, unlike the accept-reject method where some samples might be rejected.
- The numerical PPF method is simpler to implement than the accept-reject method, as the PPF can be approximated with a linear interpolation using the `numpy.interp` function in only one line of code.
- The numerical PPF method is more efficient than the accept-reject method. The numerical PPF method took around 0.02 seconds, while the accept-reject method took 5.39 seconds.

The Python code implementation is shown in Listing 1.

```python
# Define the numerical inverse CDF (PPF) method to generate samples
def generate_samples_ppf(n_samples, cdf_function, *args, xrange, seed=seed_dob):
    # Set seed for reproducibility
    if seed is not None:
        np.random.seed(seed)
    # Generate n_samples uniform random numbers in [0, 1]
    uniform_vals = np.random.rand(n_samples)
    # Generate n_samples values in the range of the random variable [alpha, beta]
    x_vals = np.linspace(*xrange, n_samples)
    # Calculate the CDF values for the range of samples
    cdf_vals = cdf_function(*args)
    # Use linear interpolation to approximate the inverse CDF (PPF) and generate samples
    samples_ppf = np.interp(uniform_vals, cdf_vals, x_vals)
    return samples_ppf
```

Listing 1: Python code implementation of the numerical point percentage function method for sample generation.

Parameter Estimation

To estimate the parameters from the generated sample, we use the `iminuit` library [6] as a minimiser. We choose the negative log-likelihood $NLL(\boldsymbol{\theta}) = -\ln L(\boldsymbol{\theta})$ as a cost function with the `cost.UnbinnedNLL` class. Using the `Minuit` class, we construct the minimiser with initial parameter guesses of $f_{init} = 0.30$, $\lambda_{init} = 0.60$, $\mu_{init} = 5.30$ and $\sigma_{init} = 0.03$. We set parameter limits for $f \in [0, 1]$ and $\mu \in [5.0, 5.6]$ and perform the parameter and uncertainty estimation with `migrad()` and `hesse()`.

Results

The results give parameter estimates of $\hat{f} = 0.1001 \pm 0.0016$, $\hat{\lambda} = 0.5045 \pm 0.0193$, $\hat{\mu} = 5.2801 \pm 0.0003$ and $\hat{\sigma} = 0.0181 \pm 0.0003$. The generated sample and estimates of the signal, background and total PDFs are shown in Figure 3. We bin the sample data using 100 bins with counts N_i for $i = 1, 2, ..., 100$. As the bin counts obey Poisson statistics, the uncertainty associated with the counts is $\sigma_{N_i} = \sqrt{N_i}$. We normalise counts and uncertainties appropriately giving a probability density histogram of the generated sample, allowing comparison to the total PDF estimate. To assess the quality of the fit, we first compute the residuals by taking the difference between the fitted and observed values at the bin centers M_i, i.e. $residual_i = p_{\alpha\beta}(M_i; \hat{f}, \hat{\lambda}, \hat{\mu}, \hat{\sigma}) - N_i$. We then normalise the residuals by the uncertainties giving the pull values $pull_i = residual_i / \sigma_{N_i}$. The pull distribution follows a Gaussian-like distribution with a mean of $\mu_{pull} \approx 0$ and standard deviation of $\sigma_{pull} \approx 1$. This suggests that the model fits the data well and that the uncertainties are correctly estimated.

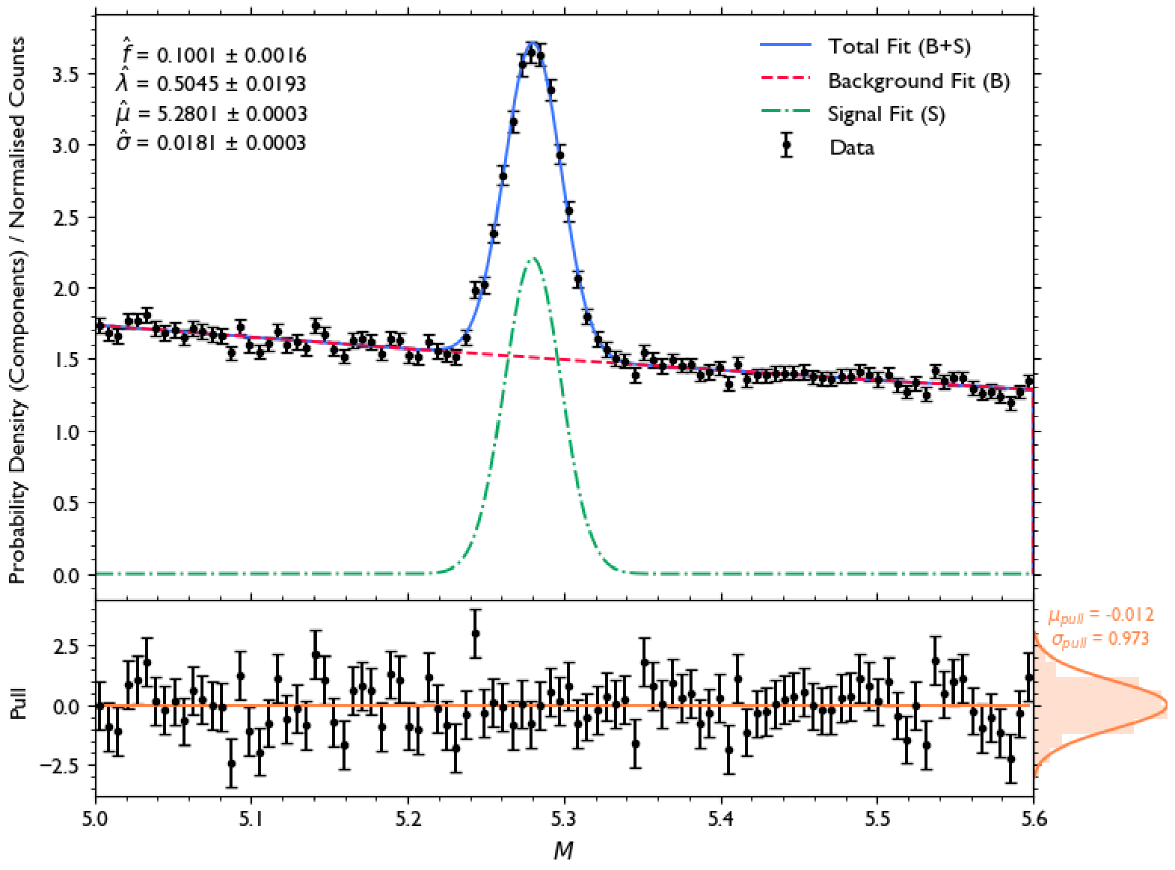

Figure 3: Top: Generated sample distribution binned with 100 bins overlaid by the estimated total PDF and its background and signal components with parameter estimates $\hat{f} = 0.1001$, $\hat{\lambda} = 0.5045$, $\hat{\mu} = 5.2801$ and $\hat{\sigma} = 0.0181$. Bottom: Pull values between the fitted and observed values and their distribution following a Gaussian-like distribution with a mean of $\mu_{pull} = -0.012$ and standard deviation of $\sigma_{pull} = 0.973$.

2 Section B

2.1 Question 1 (f)

2.1.1 Introduction

With the given true parameter values, we can estimate how much data one would need to collect to discover the signal at least 90% of the time with the simulation study presented in this section. The threshold for discovery is a p-value of less than 2.9×10^{-7}.

2.1.2 Methodology

The study involves important statistical concepts like hypothesis testing, bootstrapping and parameter estimation.

Hypothesis Testing

To find a p-value for a pretend dataset of size n generated from the total PDF, we perform hypothesis testing. We define the background only model, $p_{\alpha\beta}(M; f, \lambda, \mu, \sigma)$ with $f = 0$, as the null hypothesis H_0 and the background + signal model, $p_{\alpha\beta}(M; f, \lambda, \mu, \sigma)$ with $f \neq 0$, as the alternate hypothesis H_1. We use the negative log-likelihood ratio given by Equation (7) as the test statistic:

$$T = -2\ln\left(\frac{L(M|H_0)}{L(M|H_1)}\right) = -2\ln\left(\frac{L_0}{L_1}\right) = -2\,\Delta\ln L \tag{7}$$

We use this test statistic as it is the most powerful test, i.e. it maximises the probability of rejecting H_0 when H_1 is true, according to the Neyman-Pearson Lemma [7]. The hypothesis test is performed using the `iminuit` library as a minimiser, as shown in Listing 3. The set up of the minimiser is similar to the one explained for Question 1 (e), see in Section 1.5. The minimum value of the log-likelihood cost function for each hypothesis after the minimisation is accessed using `fval`. Taking the difference gives the observed test statistic value T_0 based on the pretend dataset.

```python
# Define the p-value calculator function for Question f)
def pval_calculator_f(dataset, chi2_dof):
    # Construct the iminuit minimiser (Unbinned NLL)
    nll = cost.UnbinnedNLL(dataset, total_model_pdf)
    minu = Minuit(nll, f=f_init, lb=lb_init, mu=mu_init, sigma=sigma_init)
    # Run fit for the null hypothesis (H0 : background only, i.e. f = 0)
    minu.values['f'] = 0
    minu.fixed['f'] = True
    minu.fixed["mu"] = True
    minu.fixed["sigma"] = True
    minu.migrad()
    minu.hesse()
    h0_min = minu.fval # minimum value of the cost function for H0
    # Run fit for the alternative hypothesis (H1 : signal + background, i.e. f != 0)
    minu.values['f'] = f_init
    minu.limits['f'] = (0,1)
    minu.fixed['f'] = False
    minu.fixed["mu"] = False
    minu.fixed["sigma"] = False
    minu.limits['mu'] = (alpha,beta)
    minu.migrad()
    minu.hesse()
    h1_min = minu.fval # minimum value of the cost function for H1
    # Calculate the test statistic and p-value
    T = h0_min - h1_min
    p_value = chi2.sf(T,chi2_dof) # 1 - chi2.cdf(T,chi2_dof)
    return p_value
```

Listing 2: Function to calculate the p-value from hypothesis testing in Question 1 (f).

The p-value is defined as the probability of obtaining a test result as large or larger than T_0, assuming that the null hypothesis H_0 is true. To calculate the p-value from the test statistic value T_0 on the pretend dataset, we employ Wilks' theorem [8] in **Theorem 1**, which states that as the sample size n tends to infinity, the test statistic T follows a chi-squared distribution χ^2 under the null hypothesis H_0.

Theorem 1. *As the sample size n tends to infinity, the negative log-likelihood ratio between a reduced model (null hypothesis H_0 with parameters $\hat{\boldsymbol{\theta}}_0$) and a full model (alternate hypothesis H_1 with parameters $\hat{\boldsymbol{\theta}}_1$) asymptotically approaches a chi-squared distribution χ^2 under the null hypothesis H_0, where the degrees of freedom k is equal to the difference between the number of free parameters of the alternate hypothesis dof_{H_1} and the null hypothesis dof_{H_0}:*

$$-2\ln\left(\frac{L(\hat{\boldsymbol{\theta}}_0)}{L(\hat{\boldsymbol{\theta}}_1)}\right) = -2\,\Delta\ln L(\hat{\boldsymbol{\theta}}) \;\sim\; \chi^2(k) \quad \text{with} \quad k = dof_{H_1} - dof_{H_0} \qquad \text{as} \qquad n \to \infty$$

We can thus use the cumulative density function F_{χ^2} or survival function S_{χ^2} of the χ^2 distribution evaluated at T_0 in Equation (8) to obtain the p-value:

$$p-\text{value} = P(T \geq T_0|H_0) = \int_{T_0}^{\infty} P(T|H_0)\,dT = 1 - F_{\chi^2}(T_0;k) = S_{\chi^2}(T_0;k) \tag{8}$$

The null hypothesis H_0 has one free parameter λ, while the alternate hypothesis H_0 has four free parameters f, λ, μ and σ. Therefore, the degrees of freedom of the χ^2 distribution is $k = 4 - 1 = 3$ according to Wilks' theorem. However, the theorem relies on the assumption that the parameters being estimated are independent. The signal fraction f and standard deviation of the normal distribution σ, i.e. the height and width of the signal, are correlated with each other. This impacts the actual degrees of freedom k. Before running a simulation based on the hypothesis testing strategy, we need to correct for the correlation between the parameters and find an estimate of k.

χ^2 Parameter Estimation

In order to find a degrees of freedom estimate $\hat{k}$ of the χ^2 distribution, we use the boostrap resampling method [9] with the `resample` module to produce the test statistic distribution under the null hypothesis H_0 and then fit it to the χ^2 distribution. We use bootstrapping to build up the test statistic because it doesn't assume any specific distribution or parameter values a priori. First, we generate a high-statistics dataset with 100,000 data events from the true background only model with the numerical PPF method. We then create 1,000 resampled datasets of the same size as the generated dataset by drawing from it with replacement. Using a similar procedure as for the hypothesis testing strategy, we obtain the negative log-likelihood ratio for each bootstrap sample and construct the distribution of T. Due to `imniuit` convergence issues, the lower limit of f was set to a small non-zero value, i.e. $f \in [0.01, 1]$, when running the fit for H_1. A degrees of freedom estimate $\hat{k}$ is then obtained by fitting it to a χ^2 distribution using `iminuit`. We can now use this estimate to calculate the p-value from the test statistic value T_0 for a given pretend dataset by implementing Equation (8) with `scipy.stats` as part of the hypothesis testing strategy.

Simulation Study

Having established the strategy to calculate the p-value for a given pretend dataset of size n, we run a simulation study to estimate the required dataset size $n_{discovery}$ to achieve a signal discovery rate of $r_{discovery} \geq 0.9$, where the threshold for discovery is a p-value of $p_{discovery} = 2.9 \times 10^{-7}$. We generate 1,000 datasets of the same size n, determine the p-value for each of them with the hypothesis testing strategy described before and count a successful discovery if $p-$value $< p_{discovery}$. The different datasets are generated using 1,000 different seeds in the numerical PPF method. We choose a relatively large number of 1,000 samples to achieve a more statistically significant and reliable estimate of $n_{discovery}$. Summing up the number of successful discoveries gives us the total number of discoveries $m_{discovery}$ for a given dataset size n. The signal discovery rate is thus $r_{discovery} = \frac{m_{discovery}}{1000}$ for a given dataset size n. Finally, we iterate through multiple dataset sizes in the range $n \in [400, 900]$ with a step size of 8 and determine the signal discovery rate $r_{discovery}$ for each n. This range is chosen based on a trial-and-error study to determine an approximate interval in which $n_{discovery}$ is likely to be found in by obtaining $r_{discovery}$ for different orders of n. The first dataset size for which $r_{discovery} \geq 0.9$ gives our final estimate for the required dataset size $n_{discovery}$.

```python
# Define the n_discovery estimator function for Question f)
def ndiscovery_calculator_f(N_list, seeds, p_value_threshold, discovery_threshold):
    # Calculate the discovery rate for each n value and seed
    discovery_rate_list = []
    for N in N_list:
        M_gen = np.linspace(*Mrange, N)
        p_value_list = []
        for seed in seeds:
            # Generate pseudo-data
            dataset = generate_samples_ppf(N, total_cdf, M_gen, f_true, lb_true, mu_true,
                                        sigma_true, Mrange, xrange = Mrange, seed=seed)
            # Calculate the p-value
            p_value = pval_calculator_f(dataset, chi2_dof_est)
            p_value_list.append(p_value)
        # Calculate the discovery rate for each n value
        p_value_array = np.array(p_value_list)
        discovery_rate = np.sum(p_value_array < p_value_threshold) / len(p_value_array)
    discovery_rate_list.append(discovery_rate)
    # Find the n value at which the discovery rate is greater than 90%
    N_discovery_index =  np.where(
        np.array(discovery_rate_list) > discovery_threshold)[0][0]
    N_discovery = N_list[N_discovery_index]
    discovery_rate_result = discovery_rate_list[N_discovery_index]
    return N_discovery, discovery_rate_result, discovery_rate_list
```

Listing 3: Function to calculate $n_{discovery}$ from the simulation study in Question 1 (f).

2.1.3 Analysis

The test statistic distribution under H_0 and the fitted χ^2 distribution in comparison to the χ^2 distribution from Wilks' theorem are shown in Figure 4. The estimated degrees of freedom from the χ^2 fit is $\hat{k} = 2.69 \pm 0.06$, which is within the expected range of k, i.e. it is higher than the number of independent parameters but lower than the value obtained from Wilks' theorem.

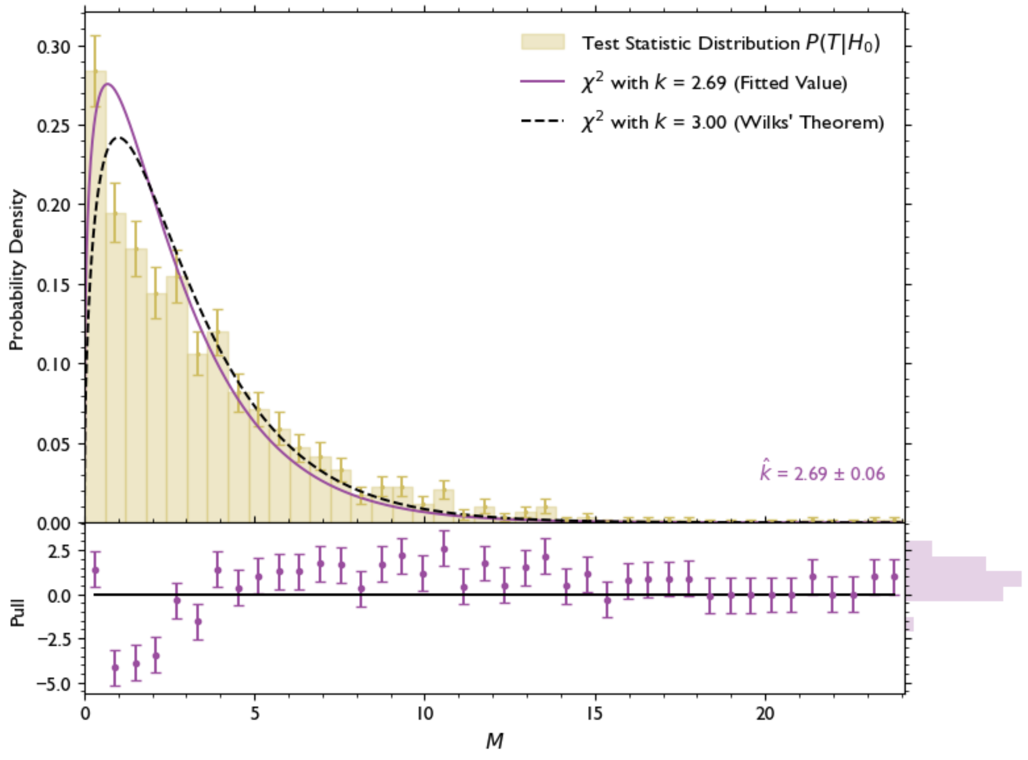

Figure 4: Test statistic distribution under the null hypothesis H_0 produced with bootstrap samples overlaid by χ^2 distributions with degrees of freedom $k = 2.69$ (fitted value) and $k = 3.00$ (Wilks' theorem).

Figure 5 illustrates the simulation study and its results. The estimate for the required dataset size to achieve a signal discovery rate of $r_{discovery} \geq 0.9$ is $n_{discovery} = 720$.

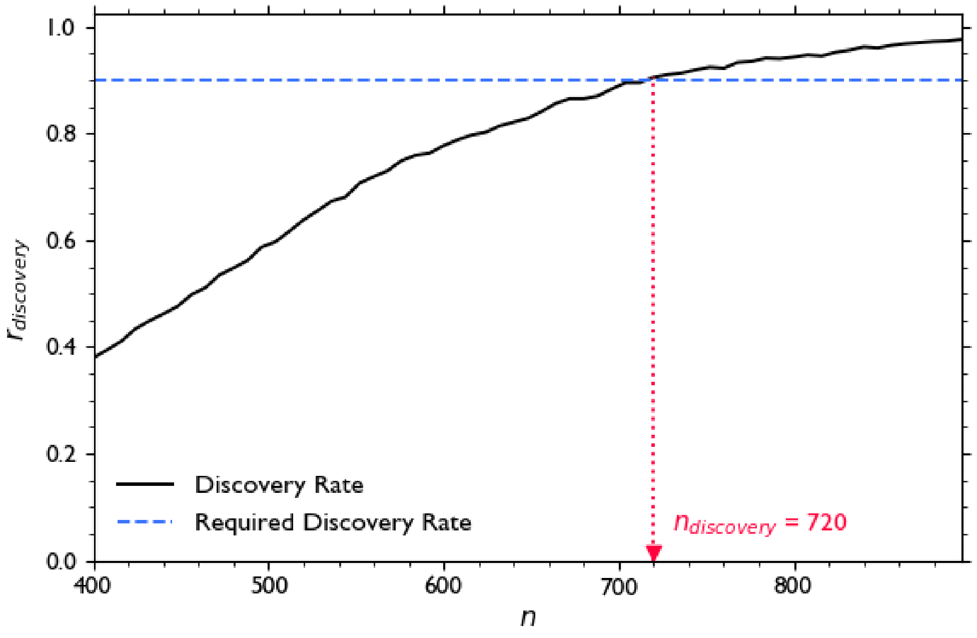

Figure 5: Discovery rate $r_{discovery}$ for different dataset sizes n determined by generating 1,000 samples for each n. The simulation study result is $n_{discovery} = 720$.

2.1.4 Discussion

Wilk's theorem relies on the assumption that parameters are independent. We mitigate against the issue of correlated parameters by fitting the test statistic distribution under H_0 to a χ^2 distribution. Despite the pull distribution of the fit not following a standard Gaussian, the resulting estimate for the degrees of freedom $\hat{k}$ is slightly lower than the value obtained through Wilks' theorem, which is in line with our expectations owing to parameter correlations.

Our simulation study's accuracy is highly dependent on the sample quantity per data size n. With only 100 rather than 1,000 samples per n, the $n_{discovery}$ estimate drops and the stability of the estimates decreases. Choosing 1,000 samples reduces statistical fluctuations. A higher amount of samples, e.g. 10,000 or 100,000, would increase the statistical significance and stability of our estimate even further. However, the execution time would be considerably higher.

To generate the 1,000 samples for each n in our simulation, we decide against bootstrapping. Our trial-and-error study revealed that $n_{discovery}$ is of order 10^2. With this small sample size, bootstrapping may yield unreliable estimates. If our original sample lacks variation, the bootstrap samples might not capture the full range of potential variation within the underlying data.

The simulation study could be further improved by decreasing the step size to get a finer estimate of $n_{discovery}$. This would against be at the expense of a higher execution time.

2.2 Question 1 (g)

2.2.1 Introduction

Assuming that there is a second signal with the same probability density function as the first signal but with a shifted mean, we write the new total probability density function as given by Equation (9) with the parameters f_1 and f_2 being the fractions of each signal, and μ_1 and μ_2 being the mean values of each signal:

$$p_{new}(M; f_1, f_2, \lambda, \mu_1, \mu_2, \sigma) = (1 - f_1 - f_2) \cdot b(M; \lambda) + f_1 \cdot s_1(M; \mu_1, \sigma) + f_2 \cdot s_2(M; \mu_2, \sigma) \quad (9)$$

The true parameter values are $f_{1_{true}} = 0.1$, $f_{2_{true}} = 0.05$, $\lambda_{true} = 0.5$, $\mu_{1_{true}} = 5.28$, $\mu_{2_{true}} = 5.35$ and $\sigma_{true} = 0.018$. Figure 6 shows $p_{new_{\alpha\beta}}(M; f_{1_{true}}, f_{2_{true}}, \lambda_{true}, \mu_{1_{true}}, \mu_{2_{true}}, \sigma_{true})$ overlaid by its background and double signal components.

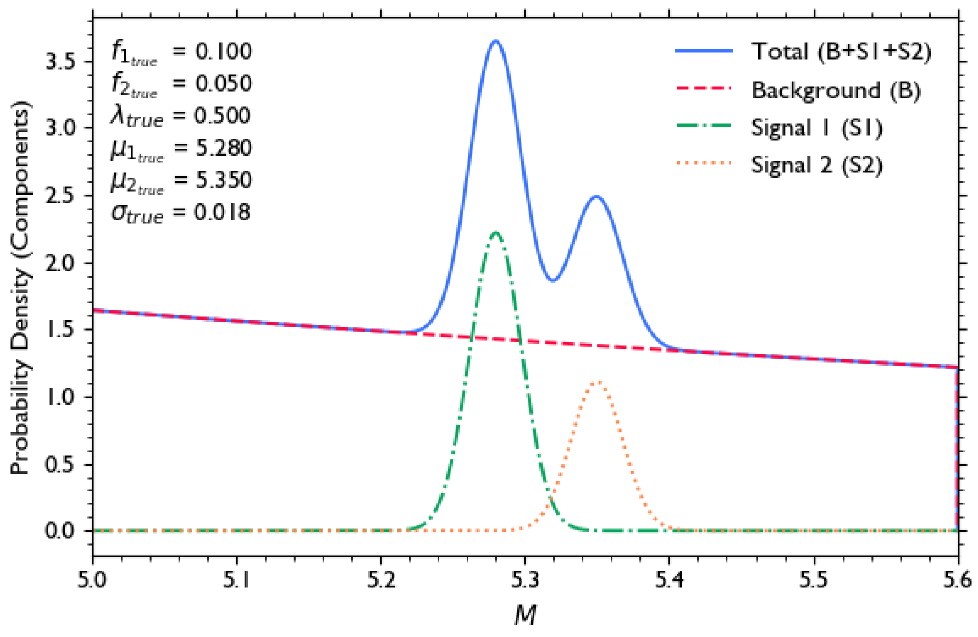

Figure 6: True new total probability distribution overlaid by its background and double signal components with parameters $f_{1_{true}} = 0.1$, $f_{2_{true}} = 0.05$, $\lambda_{true} = 0.5$, $\mu_{1_{true}} = 5.28$, $\mu_{2_{true}} = 5.35$ and $\sigma_{true} = 0.018$.

With the given true parameter values, we can estimate how much data would be needed in order to determine that there are two separate signals (not a single overlapping signal) at least 90% of the time with the simulation study presented in this section. The threshold for double signal discovery is again a p-value of less than 2.9×10^{-7}.

2.2.2 Methodology

The methods employed for the simulation study are similar to the one explained in Section 2.1.

Hypothesis Testing

To find a p-value for a generated pretend dataset of size n, we again perform hypothesis testing. We define the background + single signal model, $p_{new_{\alpha\beta}}(M; f_1, f_2, \lambda, \mu_1, \mu_2, \sigma)$ with $f_2 = 0$, as the null hypothesis H_0 and the background + double signal model, $p_{new_{\alpha\beta}}(M; f_1, f_2, \lambda, \mu_1, \mu_2, \sigma)$ with $f_2 \neq 0$, as the alternate hypothesis H_1. The rest of the hypothesis testing strategy is similar to the one explained for Question 1 (f), see in Section 2.1. However, we use the binned negative log-likelihood as a cost function with the `cost.BinnedNLL` class in `iminuit` with 500 bins. This is to speed-up the code, as we found that using `cost.UninnedNLL` takes a significantly larger amount of time to execute.

χ^2 Parameter Estimation

The null hypothesis H_0 has four free parameters f_1, λ, μ_1 and σ, while the alternate hypothesis H_0 has six free parameters f_1, f_2, λ, μ_1, μ_2 and σ. Therefore, the degrees of freedom of the χ^2 distribution is $k = 6 - 4 = 2$ according to Wilks' theorem. Again, the signal fractions f_1 and f_2 are correlated with the standard deviation σ. In order to find a better degrees of freedom estimate $\hat{k}$ of the χ^2 distribution, we use the same bootstraping procedure and χ^2 fit as in Section 2.1.

Simulation Study

We can now run a similar simulation study as in Section 2.1, for which we iterate through multiple dataset sizes in the range $n \in [3400, 3900]$ with a step size of 8 and determine the signal discovery rate $r_{discovery}$ for each n with the hypothesis testing strategy described before. The search range is again determined based on a trial-and-error study.

2.2.3 Analysis

The test statistic distribution under H_0 and the fitted χ^2 distribution in comparison to the χ^2 distribution from Wilks' theorem are shown in Figure 7. The estimated degrees of freedom from the χ^2 fit is $\hat{k} = 1.71 \pm 0.05$, which is within the expected range of k, i.e. it is higher than the number of independent parameters but lower than the value obtained from Wilks' theorem.

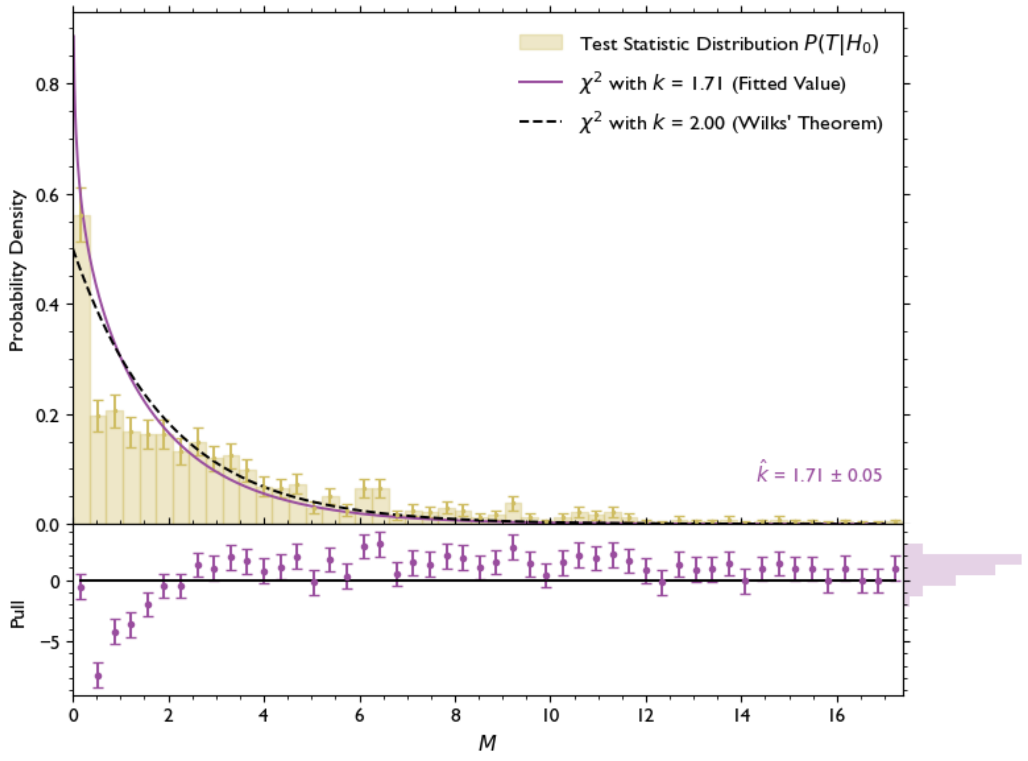

Figure 7: Test statistic distribution under the null hypothesis H_0 produced with bootstrap samples overlaid by χ^2 distributions with degrees of freedom $k = 1.71$ (fitted value) and $k = 2.00$ (Wilks' theorem).

Figure 8 illustrates the results of the simulation study. The first dataset size to achieve a signal discovery rate of $r_{discovery} \geq 0.9$ in this problem is $n_{discovery} = 3552$.

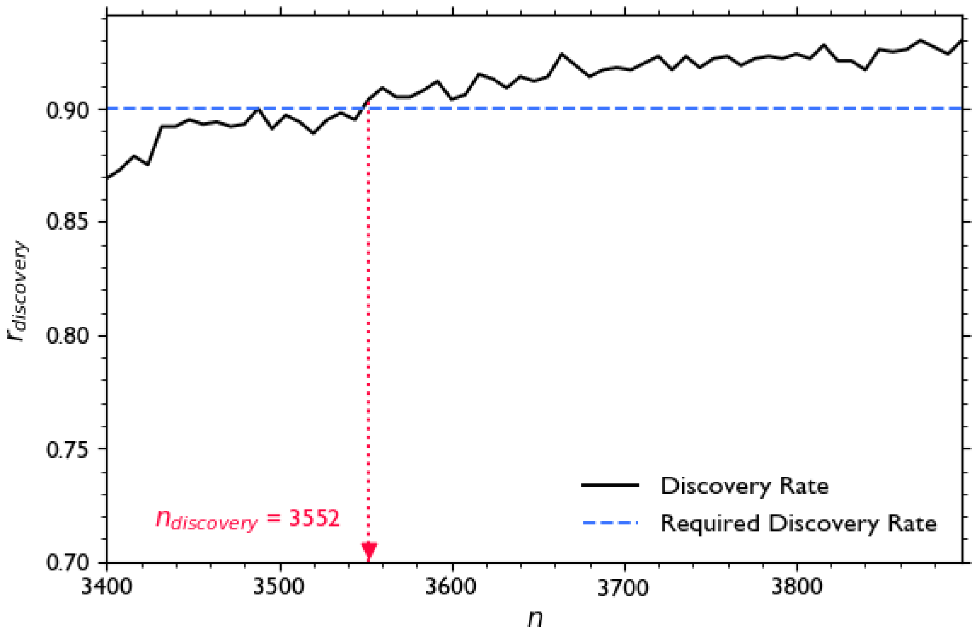

Figure 8: Discovery rate $r_{discovery}$ for different dataset sizes n determined by generating 1,000 samples for each n. The simulation study result is $n_{discovery} = 3552$.

2.2.4 Discussion

The insights discussed earlier for Question 1 (f) in Section 2.1 remain applicable here as well. Additionally, we discuss the effect of using `cost.BinnedNLL` rather than `cost.UnbinnedNLL` during the hypothesis testing strategy. Using the binned negative log-likelihood is computationally more efficient, but comes with a loss of detailed information potentially affecting the quality of our hypothesis tests. The trade-off between the computational efficiency and the need for detailed information should be investigated in more detail to improve our simulation study.

Bibliography

[1] Guido Van Rossum and Fred L. Drake. Python 3 Reference Manual. CreateSpace, Scotts Valley, CA, 2009.

[2] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. Nature, 585(7825):357–362, September 2020.

[3] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods, 17:261–272, 2020.

[4] J. D. Hunter. Matplotlib: A 2d graphics environment. Computing in Science & Engineering, 9(3):90–95, 2007.

[5] Hans Dembinski. numba-stats, version 1.4.1. https://github.com/HDembinski/numba-stats, 2023. Accessed: December 2, 2023.

[6] Hans Dembinski. iminuit: A python library for function minimization, version 2.24.0. https://github.com/scikit-hep/iminuit, 2023. Accessed: December 1, 2023.

[7] Jerzy Neyman and Egon Sharpe Pearson. Ix. on the problem of the most efficient tests of statistical hypotheses. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character, 231(694-706):289–337, 1933.

[8] Samuel S Wilks. The large-sample distribution of the likelihood ratio for testing composite hypotheses. The annals of mathematical statistics, 9(1):60–62, 1938.

[9] Bradley Efron. Bootstrap methods: another look at the jackknife. In Breakthroughs in statistics: Methodology and distribution, pages 569–593. Springer, 1992.

A README.md file

A copy of the `README.md` file is attached to this document and can be found on the next pages.

S1 Principles of Data Science Coursework Submission (sd2022)

License MIT

Description

This project is associated with the submission of the coursework for the S1 Principles of Data Science Module as part of the MPhil in Data Intensive Science at the University of Cambridge. The Coursework Instructions can be found under Instructions.md and the problems to be answered can be found under DIS_MPhil_S1_Coursework_v2.2.pdf. The associated project report can be found under S1 Coursework Report.

Table of Contents

Installation and Usage

To get started with the code associated with the coursework submission, follow these steps:

Requirements

- Python 3.9 or higher installed on your system.
- Conda installed (for managing the Python environment).
- Docker (if using containerisation for deployment).

Steps

You can either run the code locally using a `conda` environment or with a container using Docker. The code is presented in the s1_sd2022_answers.ipynb Jupyter Notebook (located in the `sd2022/src` directory). The notebook will run faster locally on a high-spec computer (recommended).

Local Setup (Using Conda) [RECOMMENDED]

1. **Clone the Repository:**

   Clone the repository to your local machine with the following command:

   ```
   $ git clone https://gitlab.developers.cam.ac.uk/phy/data-intensive-scie
   ```

   or simply download it from S1 Principles of Data Science Coursework (sd2022).

2. **Navigate to the Project Directory:**

   On your local machine, navigate to the project directory with the following command:

   ```
   $ cd /full/path/to/sd2022
   ```

   and replace `/full/path/to/` with the directory on your local machine where the repository lives in.

3. **Setting up the Environment:**

   Set up and activate the `conda` environment with the following command:

   ```
   $ conda env create -f environment.yml
   $ conda activate sd2022_s1_env
   ```

4. **Install ipykernel:**

   To run the notebook cells with `sd2022_s1_env`, install the ipykernel package with the following command:

   ```
   python -m ipykernel install --user --name sd2022_s1_env --display-name
   ```

5. **Open and Run the Notebook:**

   Open the `sd2022` directory with an integrated development environment (IDE), e.g. VSCode or PyCharm, select the kernel associated with the `sd2022_s1_env` environment and run the s1_sd2022_answers.ipynb Jupyter Notebook (located in the `sd2022/src` directory).

Containerised Setup (Using Docker)

1. **Clone the Repository:**

   Clone the repository to your local machine with the following command:

   ```
   $ git clone https://gitlab.developers.cam.ac.uk/phy/data-intensive-scie
   ```

   or simply download it from S1 Principles of Data Science Coursework (sd2022).

2. **Navigate to the Project Directory:**

   On your local machine, navigate to the project directory with the following command:

   ```
   $ cd /full/path/to/sd2022
   ```

   and replace `/full/path/to/` with the directory on your local machine where the repository lives in.

3. **Install and Run Docker:**

   You can install Docker from the official webpage under Docker Download. Once installed, make sure to run the Docker application.

4. **Build the Docker Image:**

   You can build a Docker image with the following command:

   ```
   $ docker build -t [image] .
   ```

   and replace `[image]` with the name of the image you want to build.

5. **Run a Container from the Image:**

   Once the image is built, you can run a container based on this image:

   ```
   $ docker run -p 8888:8888 [image]
   ```

   This command starts a container from the `[image]` image and maps port `8888` of the container to port `8888` on your local machine. The Jupyter Notebook server within the container will be accessible on JupyterLab at http://localhost:8888.

6. **Access and Run the Notebook:**

   After running the container, you'll see logs in the terminal containing a URL with a token. It will look similar to this:

   ```
   http://127.0.0.1:8888/lab?token=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   ```

   Navigate to http://localhost:8888 and enter the token `XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX`. Once you accessed JupyterLab, run the s1_sd2022_answers.ipynb Jupyter Notebook (located in the `sd2022/src` directory) with an `ipykernel` (Python 3).

   Note: Make sure that not other Jupter Notebook Servers are running. Otherwise, you might encounter 'Invalid credentials' issues when entering the token. Close any running Jupter Notebook Servers. To stop a running server, use `Ctrl + C` in the terminal where you launched JupyterLab. Also make sure port `8888` is not occupied.

Support

For any questions, feedback, or assistance, please feel free to reach out via email at sd2022@cam.ac.uk.

License

This project is licensed under the MIT License - see the LICENSE file for details.

Project Status

The project is in a state ready for submission. All essential features have been implemented, and the codebase is stable. Future updates may focus on minor improvements, bug fixes, or optimisations.

Note on the Use of auto-generation tools

GitHub Co-Pilot assisted the author in producing all function docstrings present in the project repository. No specific commands have been given, instead auto-completion suggestions have occasionally been accepted.

Authors and Acknowledgment

This project is maintained by Steven Dillmann at the University of Cambridge.

17th December 2023

B Execution Time and Specifications on Machine

Table 2 provides details on the execution time of the different notebook sections. The code was run on a MacBook Pro 2023, Chip: Apple M2 Pro, Memory: 16 GB with the `sd2022_s1_env` environment and kernel (see `README.md`).

Table 2: Execution times for the different code sections in the answer Jupyter notebook.

Question	**Execution Time [s]**
Question 1 (c)	0.0
Question 1 (d)	1.0
Question 1 (e)	1.2
Question 1 (f)	1444.1
Question 1 (g)	3276.5
Total	4722.8 (1 h 19 min)