UNIVERSITY OF CAMBRIDGE

DiS MPHIL

Statistics for 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
S2 Statistics for Data Science

Supervision
Dr Christopher Moore

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

List of Figures

List of Tables

Contents

Total LaTeX Word Count: 2995/3000 (excluding Figures, Tables, Bibliography, Appendix)

1 Introduction

The *Lighthouse Problem* [1] is a classic example to illustrate Bayesian inference as discussed in [2]. We present inference techniques to tackle the original and an extended lighthouse problem. The setup is presented in Figure 1. The methods and results are presented in Sections 2 and 3 for the original and extended problems, respectively. Section 4 compares the results.

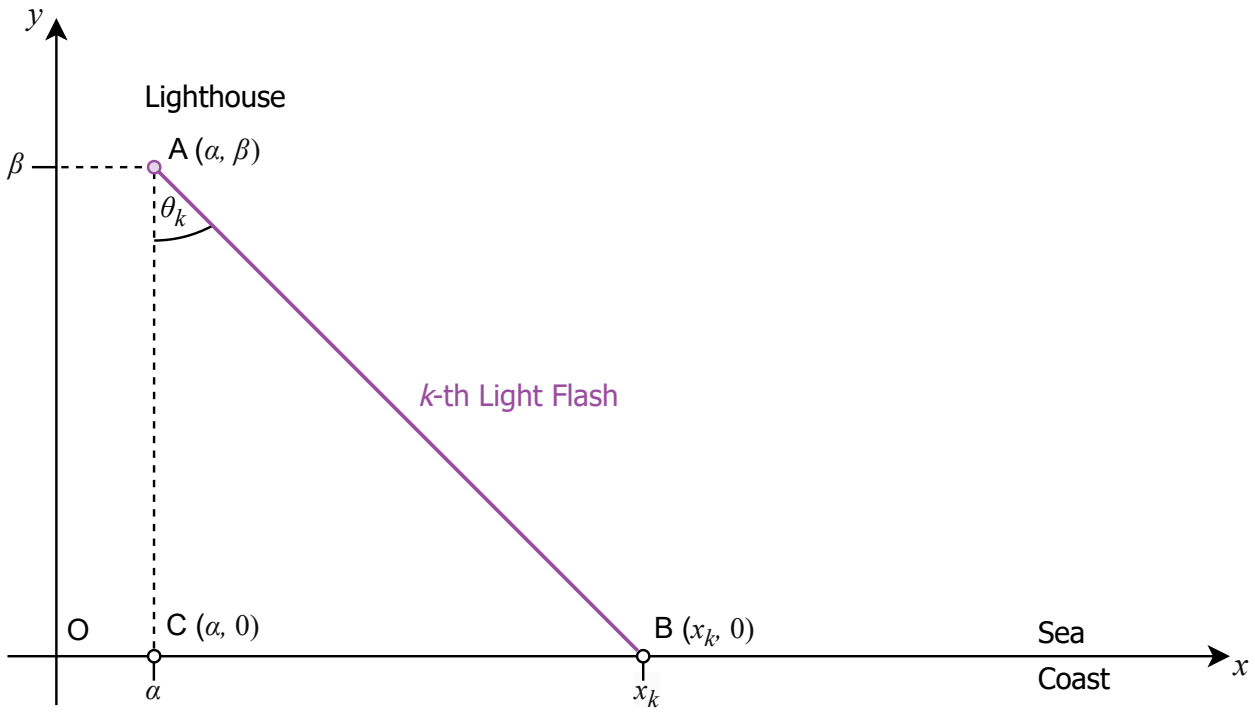

Figure 1: Diagram of the lighthouse problem setup for the k-th light flash. *Original Problem:* A lighthouse is located at a point (α, β) in the sea, where α is the position along a straight coastline and β is the distance from the coast out at sea. The lighthouse emits N flashes at random angles θ_k uniformly distributed in the range $\theta_k \in (-\frac{\pi}{2}, \frac{\pi}{2})$ hitting the coastline at points x_k recorded by an array of detectors for $k = 1, 2, \ldots, N$. Note that the detectors are only sensitive to x_k and not θ_k. *Extended Problem:* The array of detectors now also records the intensity of the flashes I_k for $k = 1, 2, \ldots, N$ with some known measurement uncertainty. The intensity measurements are independent and follow log-normal distributions with an uncertainty $\sigma_k = 1$. On average, the intensity of the observed flashes decreases with distance according to an inverse square law. The expectation of the log-intensity is $\mu_k = \log(I_0/d_k^2)$, where I_0 is the absolute intensity of the lighthouse and $d_k = \sqrt{\beta^2 + (x_k - \alpha)^2}$ is the distance between the lighthouse and point x_k. *Task:* Estimate the location of the lighthouse (α, β) (and its absolute intensity I_0) based on the $N = 20$ observations $\{x_k\}$ (and $\{I_k\}$) stored in the `lighthouse_flash_data.txt` file in the project repository S2-Assessment-sd2022.

2 Original Problem

2.1 Part (i)

Consider the right-angled triangle $\triangle ABC$ in Figure 1, with the angle θ_k at vertex A and the right angle at vertex C. The corresponding position vectors are $\overrightarrow{OA} = \langle \alpha, \beta \rangle$, $\overrightarrow{OB} = \langle x_k, 0 \rangle$ and $\overrightarrow{OC} = \langle \alpha, 0 \rangle$. The tangent of angle θ_k is given by the ratio of the opposite side to the adjacent side, i.e.

$$\tan\theta_k = \frac{\|\overrightarrow{OB} - \overrightarrow{OC}\|}{\|\overrightarrow{OA} - \overrightarrow{OC}\|} = \frac{\|\langle x_k, 0\rangle - \langle \alpha, 0\rangle\|}{\|\langle \alpha, \beta\rangle - \langle \alpha, 0\rangle\|} = \frac{\|\langle (x_k - \alpha), 0\rangle\|}{\|\langle 0, \beta\rangle\|} = \frac{x_k - \alpha}{\beta}, \tag{1}$$

and can be rearranged to give an expression of x_k in terms of α, β and θ_k given by

$$x_k = \beta \cdot \tan\theta_k + \alpha. \tag{2}$$

2.2 Part (ii)

The angle for a single flash θ_k follows the uniform distribution $\theta_k|\alpha, \beta \sim \mathcal{U}(-\frac{\pi}{2}, \frac{\pi}{2})$. The corresponding probability density function (PDF) is given by

$$p(\theta_k|\alpha, \beta) = \begin{cases} \frac{1}{\pi} & \text{for} -\frac{\pi}{2} \leq \theta_k \leq \frac{\pi}{2}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The detectors are only sensitive to the coastal positions x_k of the flashes rather than their direction θ_k. The transformation $x_k = \beta \cdot \tan\theta_k + \alpha$ is monotonically increasing over the interval $(-\pi/2, \pi/2)$. We apply a change of variables as described in Theorem 1 to find the PDF of $x_k|\alpha, \beta$:

$$p(x_k|\alpha, \beta) = p(\theta_k|\alpha, \beta) \left| \frac{d\theta_k}{dx_k} \right|. \tag{4}$$

Theorem 1. *Consider the continuous random variable X with a known probability distribution $f_X(x)$ and a new random variable $Y = g(X)$ obtained by a transformation of X using a function g. If the transformation function g is a monotonic and invertible, i.e. $X = g^{-1}(Y)$, the probability distribution $f_Y(y)$ of the new random variable Y is given by*

$$f_Y(y) = f_X(x) \left| \frac{dx}{dy} \right|.$$

The derivative $\frac{d\theta_k}{dx_k}$ is obtained from Equation 2. Considering the trigonometric identity $\sec^2\theta_k = 1 + \tan^2\theta_k$ and substituting $\tan\theta_k = \frac{x_k - \alpha}{\beta}$ from Equation 1 gives

$$\frac{dx_k}{d\theta_k} = \beta \cdot \sec^2\theta_k = \beta \cdot \left[1 + \left(\frac{x_k - \alpha}{\beta}\right)^2\right] = \frac{\beta^2 + (x_k - \alpha)^2}{\beta} \quad \Rightarrow \quad \frac{d\theta_k}{dx_k} = \frac{\beta}{\beta^2 + (x_k - \alpha)^2}. \tag{5}$$

Substituting Equations 3 and 5 into Equation 4 gives us the final expression for the PDF of $x_k|\alpha, \beta$, which corresponds to the *likelihood for the location of a single flash* given by

$$\mathcal{L}_x(x_k|\alpha, \beta) = p(x_k|\alpha, \beta) = \frac{1}{\pi} \cdot \frac{\beta}{\beta^2 + (x_k - \alpha)^2}. \tag{6}$$

2.3 Part (iii)

Equation 6 is a Cauchy distribution (see Definition 1) with location parameter α and scale parameter β, i.e. $x_k|\alpha,\beta \sim Cauchy(\alpha,\beta)$. The most likely location for any single flash $\hat{x}_k$ is found by maximising the PDF $p(x_k|\alpha,\beta)$. The PDF $p(x_k|\alpha,\beta)$ in Equation 6 is largest when the denominator is smallest, which occurs at the scale parameter α, i.e. the mode is $\hat{x}_k = \alpha$.

Definition 1. *The probability density function of the Cauchy distribution for a random variable X denoted as $X \sim Cauchy(a,b)$ is given by*

$$f(x|a,b) = \frac{1}{\pi} \cdot \frac{b}{b^2 + (x-a)^2},$$

where a is the location parameter and b is the scale parameter of the distribution.

The normal distribution is unimodal and symmetric about the mean μ, which coincides with the mode and median. Using Maximum Likelihood Estimation (MLE) [3], it can be shown that the best estimate of μ is given by the arithmetic mean of the data [4]. The Cauchy distribution is also unimodal and symmetric about the location parameter α. Thus, one could suggest using the sample mean $\bar{x}_k = (1/N)\sum_k x_k$ as an estimator for the parameter α. In the following, we compute the MLE for α, denoted as $\hat{\alpha}_{MLE}$, and show that it does not correspond to $\bar{x}_k$.

Maximum Likelihood Estimation

MLE maximises the likelihood function to find parameters that make the observed data most probable. Because the flash angles θ_k are independent, the *likelihood for a set of flash locations* is given by

$$\mathcal{L}_x(\{x_k\}|\alpha,\beta) = \prod_{k=1}^{N} \mathcal{L}_x(x_k|\alpha,\beta). \tag{7}$$

The goal is to find the value of α that maximises the likelihood, i.e. $\hat{\alpha}_{MLE}$. For easier manipulation, differentiation and optimisation, we first find the log-likelihood given by

$$\begin{aligned}\ln \mathcal{L}_x(\{x_k\}|\alpha,\beta) &= \sum_{k=1}^{N} \ln \mathcal{L}_x(x_k|\alpha,\beta) = \sum_{k=1}^{N} \ln\left(\frac{\beta}{\pi} \cdot \frac{1}{\beta^2 + (x_k-\alpha)^2}\right) \\ &= N\ln(\beta) - N\ln(\pi) - \sum_{k=1}^{N} \ln(\beta^2 + (x_k-\alpha)^2).\end{aligned} \tag{8}$$

Next, we differentiate the log-likelihood with respect to α and set the derivative to zero:

$$\frac{\partial}{\partial\alpha} \ln \mathcal{L}_x(\{x_k\}|\alpha,\beta) = -2\sum_{k=1}^{N} \frac{(x_k-\alpha)}{\beta^2 + (x_k-\alpha)^2} = 0. \tag{9}$$

Finally, we obtain $\hat{\alpha}_{MLE}$ by solving Equation 9 for α. Although a closed-form analytical solution cannot be found for this problem, we could implement numerical methods to find the MLE. It is clear that the sample mean $\bar{x}_k$ is not a solution to Equation 9, i.e. it is not equivalent to the MLE.

The MLE $\hat{\alpha}_{MLE}$ is a good estimator for the location parameter α because:

- It is robust to the heavy tails of the distribution and its proneness to produce outliers.
- It does not require finite moments. The Cauchy distribution has an undefined expectation and variance.
- It directly maximises the likelihood function and has desirable properties relating to its consistency and efficiency.

The sample mean $\bar{x}_k$ is a poor estimator for α because:

- It is highly sensitive to outliers. Extreme values are common in the Cauchy distribution due to its heavy tails and can significantly affect the sample mean.
- It provides a measure of central tendency. However, the expectation and variance of the Cauchy distribution are undefined, hence the concept of "mean" does not apply.
- It has undesirable estimation properties when applied to the Cauchy distribution, i.e. it is not consistent and can be highly biased due to outliers.

Given that the sample mean $\bar{x}_k$ is not equivalent to the MLE and the reasons outlined above, we conclude that the sample mean is not a good estimator for α.

Note on the Central Limit Theorem

According to the Central Limit Theorem (CLT) in Theorem 2, the sample mean drawn from practically any distribution with an expected value μ tends to a normal distribution centered around the true mean μ as the sample size approaches infinity. As mentioned before, the best estimate of the mean μ of a normal distribution is given by the sample mean. Thus, one could further suggest increasing the sample size N towards the limit would make the sample mean a good estimator.

Theorem 2. *Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed (i.i.d.) random variables with a common distribution and finite expectation $\mathrm{E}[X_i] = \mu$ and finite variance $\mathrm{Var}[X_i] = \sigma^2$. The CLT states that as the sample size n tends to infinity, the distribution of the sample mean converges to a normal distribution regardless of the original distribution of the variables:*

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \to \mathcal{N}\left(\mu, \frac{\sigma^2}{n}\right) \qquad \text{as} \qquad n \to \infty.$$

However, the Cauchy distribution violates one of the underlying assumptions of the CLT, i.e. the distributions expectation and variance are not finite due to its heavy tails. This means that increasing the number of observations does no result in the sample mean converging to a normal distribution. Thus, the sample mean is still a poor estimator for α even in the case of large N.

2.4 Part (iv)

We use Bayesian inference to estimate α and β. We choose a non-informative uniform prior for α as it is a *location parameter* about which we have no prior information. For the bounds of the uniform distribution, we choose $\alpha_l = -4$ and $\alpha_u = 4$[1]. The uniform prior $\alpha \sim \mathcal{U}(-4, 4)$ is given by

$$P(\alpha) = \begin{cases} \frac{1}{8} & \text{for} \quad -4 \leq \alpha \leq 4, \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

Similarly, we choose a uniform prior for β with the lower bound of $\beta_l = 0$ because we know that the lighthouse is out at sea. The upper bound is chosen to be $\beta_u = 8$[1]. The uniform prior $\beta \sim \mathcal{U}(0, 8)$ is given by

$$P(\beta) = \begin{cases} \frac{1}{8} & \text{for} \quad 0 \leq \beta \leq 8, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

Assuming α and β are independent, the joint prior probability is given by

$$P(\alpha, \beta) = P(\alpha)P(\beta). \tag{12}$$

[1]Initially, broader ranges were selected for α_l, α_u, and β_u in the MCMC analysis in Part (v). However, observing that the posterior distribution consistently fell within much narrower limits, we iteratively adjusted these ranges to more accurately reflect the algorithm's exploration of the parameter space.

2.5 Part (v)

2.5.1 Posterior Distribution

Bayes' Theorem

To infer the lighthouse location, we use Bayes' Theorem [5] to compute the posterior probability of our parameters (α, β) given the data $\{x_k\}$. It is denoted as $P(\alpha, \beta|\{x_k\})$ and given by

$$P(\alpha, \beta|\{x_k\}) = \frac{\mathcal{L}_x(\{x_k\}|\alpha, \beta)P(\alpha, \beta)}{P(\{x_k\})} = \frac{\mathcal{L}_x(\{x_k\}|\alpha, \beta)P(\alpha)P(\beta)}{P(\{x_k\})}. \tag{13}$$

As the evidence $P(\{x_k\})$ serves as a normalisation constant, we can ignore it[2] and instead focus on the proportionality aspect by writing Bayes' Theorem as

$$P(\alpha, \beta|\{x_k\}) \propto \mathcal{L}_x(\{x_k\}|\alpha, \beta)P(\alpha)P(\beta). \tag{14}$$

We write Bayes' Theorem in terms of natural logarithms and express the log-posterior as

$$\ln P(\alpha, \beta|\{x_k\}) \propto \ln \mathcal{L}_x(\{x_k\}|\alpha, \beta) + \ln P(\alpha) + \ln P(\beta) \tag{15}$$

to improve computational efficiency and stability. This expression avoids issues involving numerical underflow and simplifies multiplications into additions [6].

2.5.2 Markov Chain Monte Carlo

Using the data $\{x_k\}$ in the first column of the file `lighthouse_flash_data.txt`, we draw stochastic samples from the log-posterior $\ln P(\alpha, \beta|\{x_k\})$. Markov Chain Monte Carlo (MCMC) methods combine the principles of Markov chains [7] and Monte Carlo simulations [8] to allow stochastic sampling from complex target distributions and are well suited for exploring high-dimensional parameter spaces. MCMC methods generate samples by constructing a Markov chain (see Definition 2) that has the desired target distribution as its stationary distribution [9]. MCMC methods rely on the principle that the Markov chain will, given enough number of iterations, converge to a *stationary distribution* that is independent of the initial state [10]. MCMC algorithms construct a Markov chain that satisfies the *detailed balance condition* and thus has the target distribution from which we wish to sample from as its stationary distribution [11, 12]. Fore more details on Markov chains and their role in MCMC, please refer to [13, 14].

Definition 2. *A Markov chain is an ordered sequence of random points $y_0, y_1, y_2, \ldots$ in the sample space $y_i \in \mathcal{Y}$ that satisfies the Markov property*

$$P(y_{i+1}|y_0, y_1, \ldots, y_i) = P(y_{i+1}|y_i),$$

where $P(y_{i+1}|y_i)$ are the transition probabilities.

The most important and widely used MCMC method is the Metropolis-Hastings (MH) algorithm [15, 16]. We use MH rather than Gibbs sampling [17] or Hamiltonian Monte Carlo (HMC) [18] due to its applicability to a wide variety of problems, simplicity and ease of implementation[3].

[2] We are interested in the shape of the posterior distribution of the parameters. The evidence $P(\{x_k\})$ involves integrating over all possible parameter values and is often computationally expensive to calculate.

[3] Gibbs requires sampling from the full conditional distributions of each parameter, given all others. HMC is highly efficient for high-dimensional problems, but its complex implementation makes its use less justified than MH for the straightforward lighthouse problem.

Metropolis-Hastings

MH is presented in Algorithm 1. After initialising the first sample y_0 with an initial value y_{init} (line 2), MH performs the following steps for n_{iter} iterations (line 3):

1. At each iteration i, a new candidate y' for the next point is sampled from a proposal distribution Q based on the current state y_i (line 4).
2. An acceptance ratio A is calculated taking into account Q and the target distribution P. It represents the probability of moving from the current state of the chain y_i to a proposed state y' (line 5) and ensures that the the detailed balance condition is satisfied [19].
3. A uniform random number u between 0 and 1 is drawn (line 6). If u is less than A, the proposed sample y' is accepted as the next sample (line 8). If not, then the proposed state y' is rejected and the algorithm remains at the current state (line 10).

This results in a Markov chain $\{y_i\}$ for $i = 0, 1, \ldots, n_{iter} - 1$.

Algorithm 1 Metropolis-Hastings Algorithm

1: **procedure** METROPOLISHASTINGS(n_{iter}, y_{init}, Q, P)
2: $y_0 \leftarrow y_{init}$ ▷ Initialise the chain with y_{init}
3: **for** $i = 0, 1, \ldots, n_{iter} - 1$ **do** ▷ Iterate $i = 0, 1, \ldots, n_{iter} - 1$
4: $y' \sim Q(y'|y_i)$ ▷ Sample from the proposal distribution Q
5: $A \leftarrow \min(1, \frac{P(y')Q(y_i|y')}{P(y_i)Q(y'|y_i)})$ ▷ Acceptance ratio using the target and proposal P and Q
6: $u \sim \mathcal{U}(0, 1)$ ▷ Draw from a uniform distribution
7: **if** $u < A$ **then**
8: $y_{i+1} \leftarrow y'$ ▷ Accept the proposed sample
9: **else**
10: $y_{i+1} \leftarrow y_i$ ▷ Reject and keep the old sample
11: **end if**
12: **end for**
13: **end procedure**

The proposal distribution Q directly affects the exploration of the target distribution P and the algorithm's efficiency and effectiveness[4]. Therefore, choosing a good proposal Q is crucial to balance exploration and exploitation. Optimal acceptance ratios are often quoted to be in the range $0.2 < A < 0.5$ [20, 21] depending on the dimensionality of the problem. For lower-dimensional problems, aiming towards the higher end of this range can be beneficial.

Hyperparameter Selection

We choose a multivariate Gaussian proposal distribution Q centered at the current state, i.e.

$$(\alpha', \beta') \sim \mathcal{N}((\alpha_i, \beta_i), \Sigma) \quad \text{with} \quad \Sigma = \begin{pmatrix} \sigma_\alpha^2 & 0 \\ 0 & \sigma_\beta^2 \end{pmatrix}, \tag{16}$$

where Σ is the covariance matrix with proposal widths $\sigma = \sigma_\alpha = \sigma_\beta$ because both parameters have similar scales. As there are only 2 parameters, we aim for a relatively high acceptance ratio of $A \approx 0.4$. Iteratively refining the proposal width, we eventually choose $\sigma = 0.8$ achieving an acceptance ratio of $A = 0.42$. We initialise the chain using the midpoints of our parameter ranges defined earlier, hence $(\alpha_0, \beta_0) = (0, 4)$. The target distribution P is the log-posterior $\ln P(\alpha, \beta|\{x_k\})$. We run the algorithm with a relatively high number of iterations $n_{iter} = 100000$ for better convergence, coverage and effective sample size without encountering run-time issues.

[4] A broader Q encourages a wider exploration of the parameter space allowing the algorithm to escape local modes but at the risk of lower acceptance rates. A narrower Q focuses on the exploitation of the current region, increasing the acceptance rate but potentially trapping the chain in local modes and slowing exploration [14].

2.5.3 Convergence Diagnostics and Chain Refinement

Trace Plots, Running Statistics and Burn-in Removal

As a first convergence diagnostics tool, we investigate the trace plots of the chain shown in Figure 2. We also show the running mean and standard deviation of the chain, efficiently computed using Welford's method [22]. After the so-called *burn-in* period [23], the chain oscillates in a stationary manner indicating good mixing, an effective parameter space exploration and convergence to equilibrium. The stable running mean and standard deviation also indicate chain convergence. We define the end of the burn-in as the point at which both the running mean and standard deviation of the chain start to deviate less than 5% from the overall final values for all parameters. This gives us a burn-in length of $b_\alpha = 3762$ for α and $b_\beta = 1933$ for β. We use the maximum of these values as the overall burn-in length, giving us $b = 3762$ samples. During the burn-in period, the chain will generate samples that do not follow the target distribution, especially if the starting point of the chain is far from the equilibrium. Therefore, we discard the burn-in period resulting in a new chain length of $n_b = n_{iter} - b = 100000 - 3762 = 96238$.

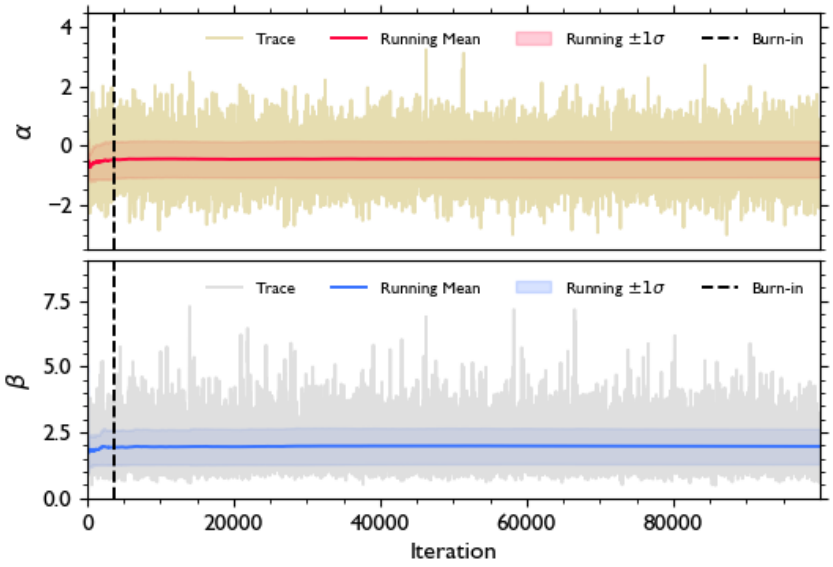

Figure 2: Trace plots for α and β including the running mean and standard deviation of the chains. The trace plots show good mixing, an effective exploration of the parameter space and convergence. The running statistics stabilise and converge at higher iterations. The burn-in period is 3762 samples long.

Autocorrelation, Effective Sample Size and Thinning

As a second convergence diagnostics tool, we analyse the autocorrelation of our MCMC chain. Figure 3 shows the autocorrelation function (ACF) and partial autocorrelation function (PACF) at different lags computed with `statsmodels` [24]. The ACF plot shows the MCMC chain's self-correlation at various lags, and the PACF plot shows correlations at different lags while eliminating the impact of shorter lag correlations. The ACF plots show a relatively rapid decay to zero for both α and β indicating a well-mixed chain and efficient sampling from the target distribution. We can further quantify the efficiency and autocorrelation of the chain using the integrated autocorrelation time (IAT). The IAT is denoted by τ and given by

$$\tau = 1 + 2\sum_i \rho(i), \tag{17}$$

where $\rho(i)$ is the autocorrelation at lag i. The IAT essentially measures the time it takes for how many steps in the chain are necessary to generate independent samples from the target distribution. The total number of independent samples n_{eff}, also called the effective sample size (ESS), is directly related and given by

$$n_{eff} = \frac{n_b}{\tau}. \tag{18}$$

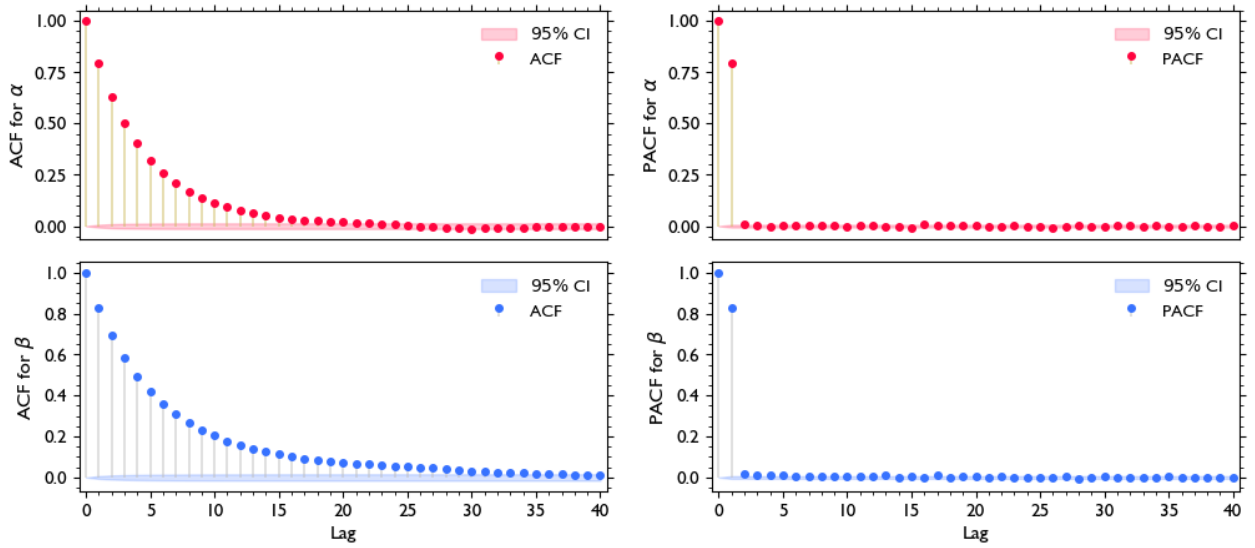

Figure 3: ACF and PACF for α and β. Both ACFs decay relatively rapidly indicating good mixing and efficient sampling from the target distribution. The PCFs visualise that the Markov chains satisfy the Markov property, i.e. each state only depends on the previous state.

A low IAT and high ESS indicate a high efficiency of the MCMC chain. We can also use these metrics to assess the convergence of the chains by monitoring them over time as the chain evolves. Figure 4 shows the running IAT and ESS, calculated with `emcee` [25] and using Equation 18. We can see the IAT of the chain stabilising at higher iterations indicating good convergence.

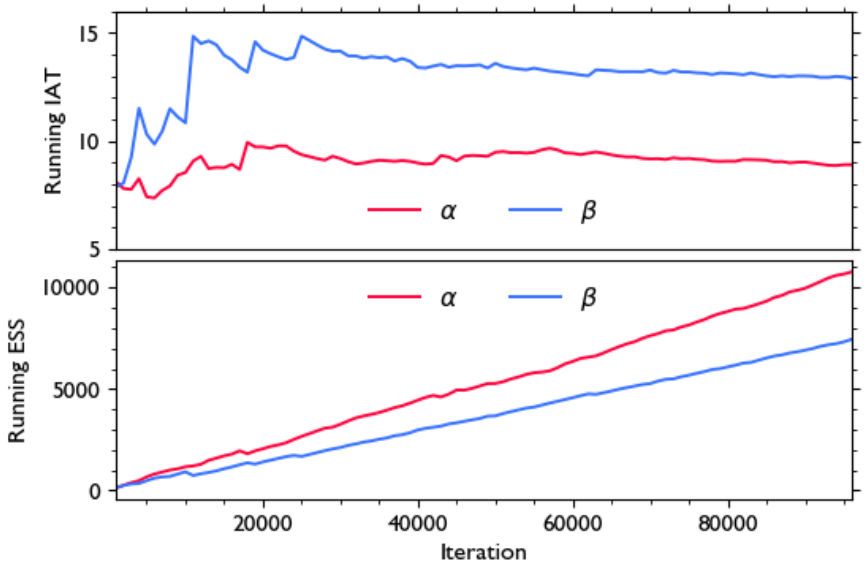

Figure 4: Running IAT and ESS for α and β. The IAT stabilises for both parameters at higher iterations.

The final IAT is $\tau_\alpha = 8.94$ for α and $\tau_\beta = 12.88$ for β. By taking the maximum of these values and rounding to the nearest integer, we obtain an overall IAT of $\tau = 13$. This IAT is relatively low suggesting high efficiency. The process of only choosing every τ-th sample to reduce autocorrelation, i.e. only taking independent samples from the chain, is called *thinning* [26]. This can help in achieving a better approximation of the target distribution and gives us a new chain length of $n_{eff} = \lfloor n_b/\tau \rfloor = \lfloor 96238/13 \rfloor = 7403$.

2.5.4 Results

We use the refined MCMC chain to estimate the posterior distribution $\ln P(\alpha, \beta|\{x_k\})$. Figure 5 shows the joint and marginalised posteriors on α, β and I_0. All parameters are well-constrained. The mean, standard deviation, median and credible intervals (CIs) for our parameter estimates are summarised in Table 1. Our parameter estimates in the form of the mean and standard deviation are $\hat{\alpha} = -0.45 \pm 0.60$ and $\hat{\beta} = 1.96 \pm 0.67$. However, the posterior distribution on β is slightly skewed. Therefore, we also quote our estimate for β in the form of the median and 68% CI (analogous to ± 1 standard deviation), which is $\hat{\beta}_{med} = 1.85^{+0.71}_{-0.49}$.

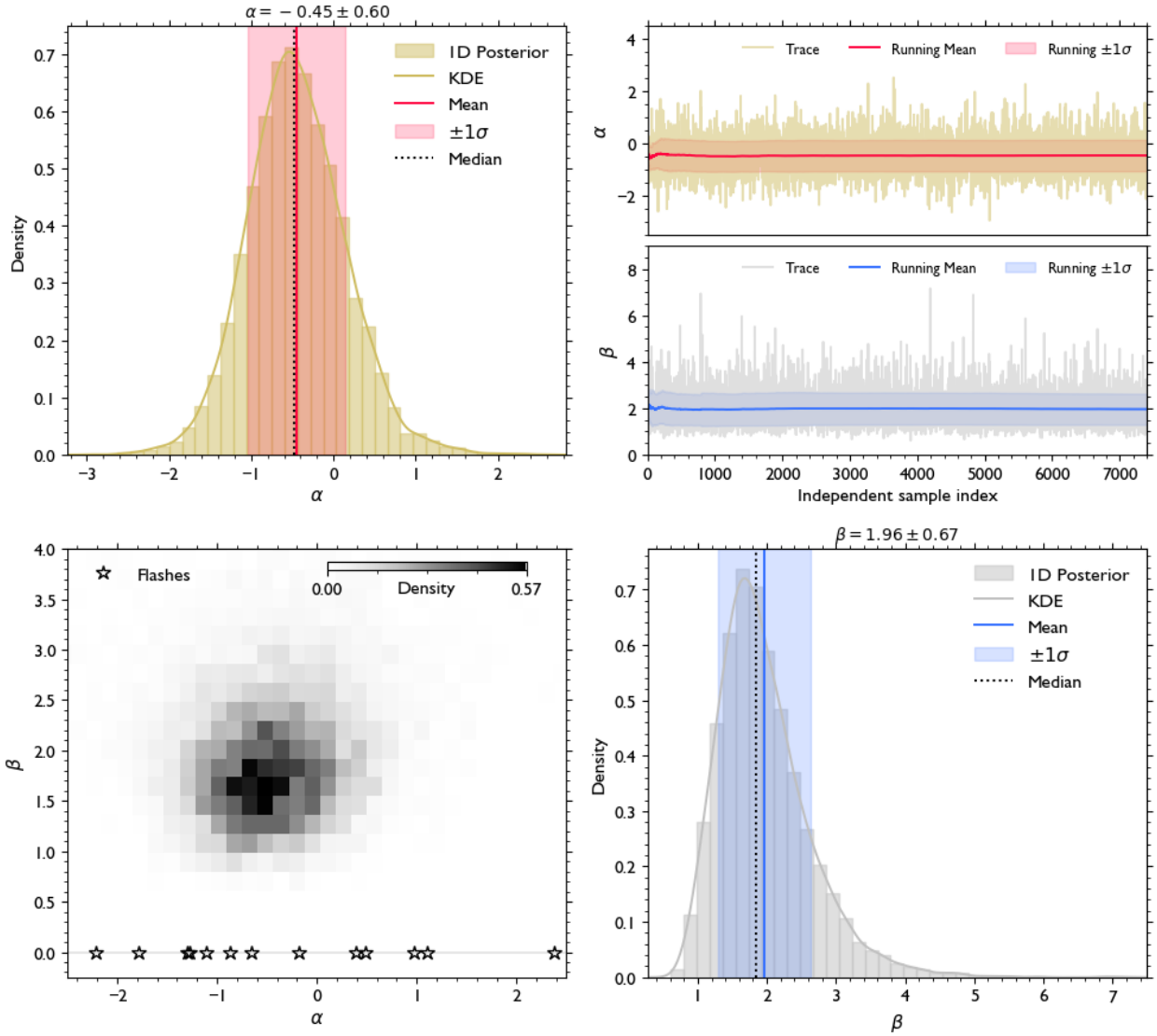

Figure 5: Corner plot of our posterior distribution $\ln P(\alpha, \beta|\{x_k\})$ estimated using Bayesian techniques and MCMC. *Bottom left:* Two-dimensional histogram showing the joint posterior on α and β. The flash locations x_k falling within the interval [-2.5, 2.5] are also visualised. *Top left:* One-dimensional histograms of the marginalised posteriors on α. *Bottom right:* One-dimensional histograms of the marginalised posteriors on α. *Top right:* Trace plots and running statistics of the independent samples from the posterior (refined chain).

Table 1: Summary statistics of the Bayesian posterior distribution on α and β.

Parameter	Mean	Standard Deviation	Median	68% CI	95% CI
α	-0.45	$\pm$0.60	-0.47	[-1.02, 0.13]	[-1.57, 0.76]
β	1.96	$\pm$0.67	1.85	[1.36, 2.56]	[0.99, 3.60]

3 Extended Problem Solutions

3.1 Part (vi)

We use a non-informative and scale-invariant log-uniform prior (see Definition 3), also called Jeffreys prior [27], for the unknown parameter I_0 in the extended problem as it is a *scale parameter*.

Definition 3. *The probability density function of the log-uniform distribution for a random variable X on the interval $x \in [a, b]$ denoted as $X \sim Log\mathcal{U}(a, b)$ is given by*

$$f(x|a,b) = \begin{cases} \frac{1}{x \cdot \ln(b/a)} & \text{for} \quad a \leq x \leq b, \\ 0 & \text{otherwise.} \end{cases}$$

where a is the lower bound and b is the upper bound of the distribution.

The key feature of this prior is its invariance under change of scale or units. This makes it particularly suitable for the absolute intensity I_0, where the inherent scale of the parameter is not precisely known. For the lower and upper bound of the log-uniform distribution, we choose $I_{0_l} = 0.1$ and $I_{0_u} = 12.1$[1]. The log-uniform prior $I_0 \sim Log\mathcal{U}(0.1, 12.1)$ is given by

$$P(I_0) = \begin{cases} \frac{1}{I_0 \cdot \ln(12.1/0.1)} & \text{for} \quad 0.1 \leq x \leq 12.1, \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

Assuming α, β and $_0$ are independent, the joint prior probability is given by

$$P(\alpha, \beta, I_0) = P(\alpha)P(\beta)P(I_0). \tag{20}$$

3.2 Part (vii)

This part is focused on the stochastic sampling results and analysis for the extended problem. For further details, please refer to Part (v) in which most relevant methodological justifications and theoretical underpinnings of this part have already been covered.

3.2.1 Posterior Distribution

The *likelihood for the intensity of a single flash* is

$$\mathcal{L}_I(\log I_k|\alpha, \beta, I_0) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(\log I_k - \mu_k)^2}{2\sigma_k^2}\right), \tag{21}$$

where the uncertainty is $\sigma_k = 1$ and the expectation of the log-intensity is $\mu_k = \ln(I_0/d_k^2)$ with the lighthouse-detector distance being $d_k = \sqrt{\beta^2 + (x_k - \alpha)^2}$. The location and intensity measurements are independent, so the *likelihood for the combined location and intensity measurement* is

$$\mathcal{L}(x_k, I_k|\alpha, \beta, I_0) = \mathcal{L}_x(x_k|\alpha, \beta)\mathcal{L}_I(\log I_k|\alpha, \beta, I_0). \tag{22}$$

Because the individual flashes are independent, we write the *likelihood for a set of combined flash locations and intensities* as

$$\mathcal{L}(\{x_k\}, \{I_k\}|\alpha, \beta, I_0) = \prod_{k=1}^{N} \mathcal{L}(x_k, I_k|\alpha, \beta, I_0). \tag{23}$$

[1] I_{0_l} is chosen to be slightly above 0 because the log-uniform prior is undefined for non-positive values. I_{0_u} is based on iterative experimental runs, similar to Part (iv).

Bayes' Theorem

Using Bayes' Theorem, we write the three-dimensional posterior $P(\alpha, \beta, I_0|\{x_k\}, \{I_k\})$ as

$$P(\alpha, \beta, I_0|\{x_k\}, \{I_k\}) \propto \mathcal{L}(\{x_k\}, \{I_k\}|\alpha, \beta, I_0)P(\alpha)P(\beta)P(I_0) \tag{24}$$

and further write it in terms of natural logarithms, i.e.

$$\ln P(\alpha, \beta, I_0|\{x_k\}, \{I_k\}) \propto \ln \mathcal{L}(\{x_k\}, \{I_k\}|\alpha, \beta, I_0) + \ln P(\alpha) + \ln P(\beta) + \ln P(I_0). \tag{25}$$

3.2.2 Markov Chain Monte Carlo

Metropolis-Hastings

We draw stochastic samples from the posterior distribution using the MH algorithm due to the reasons outlined in Part (v) and for easier comparison between both inference results in Part (viii).

Hyperparameter Selection

We choose a multivariate Gaussian proposal distribution Q centered at the current state, i.e.

$$(\alpha', \beta', I_0') \sim \mathcal{N}((\alpha_i, \beta_i, I_{0_i}), \Sigma) \quad \text{with} \quad \Sigma = \begin{pmatrix} \sigma_\alpha^2 & 0 & 0 \\ 0 & \sigma_\beta^2 & 0 \\ 0 & 0 & \sigma_{I_0}^2 \end{pmatrix}, \tag{26}$$

where Σ is the covariance matrix with proposal widths $\sigma_\alpha = \sigma_\beta$ because both location parameters have similar scales. As the dimensionality of the extended problem is higher than before, we aim for a lower acceptance ratio of $A \approx 0.3$. We choose $\sigma_\alpha = \sigma_\beta = 0.4$ and $\sigma_{I_0} = 1.2$ as these hyperparameters of the proposal result in good mixing and convergence achieving an acceptance ratio of $A = 0.31$. We initialise the chain with $(\alpha_0, \beta_0, I_{0_0}) = (0, 4, 6)$. The target distribution P is the log-posterior $\ln P(\alpha, \beta, I_0|\{x_k\}, \{I_k\})$. We run the algorithm with $n_{iter} = 100000$ iterations for the same reasons outlined in Part (v).

3.2.3 Convergence Diagnostics and Chain Refinement

Trace Plots, Running Statistics and Burn-in Removal

The trace plots and chain running statistics in Figure 6 show an effective exploration of the parameter space and good mixing without any signs of convergence issues beyond the burn-in. We find burn-in lengths of $b_\alpha = 23$, $b_\beta = 5969$ an $b_{I_0} = 1434$ based on the same criteria as in Part (v). Taking the maximum gives us an overall burn-in length of $b = 5969$ samples. Removing the burn-in results in a new chain length of $n_b = n_{iter} - b = 100000 - 5969 = 94031$.

Autocorrelation, Effective Sample Size and Thinning

While the ACF plots in Figure 7 show a relatively rapid decay for α, the chains for β and I_0 show higher autocorrelation and less efficient sampling from the target distribution. The difference in efficiency for the different parameters is also reflected in the lower IAT of $\tau_\alpha = 14.27$ for α as compared to $\tau_\beta = 25.90$ for β and $\tau_{I_0} = 29.54$. Taking the maximum gives us an overall IAT of $\tau = 30$, which is approximately twice as high as in Part (v). However, the running IAT in Figure 8 stabilises for all parameters at higher iterations suggesting good convergence of our chain. We again thin the chain by keeping every τ-th sample for the reasons outlined in Part (v) giving $n_{eff} = \lfloor n_b/\tau \rfloor = \lfloor 94031/30 \rfloor = 3135$ independent samples.

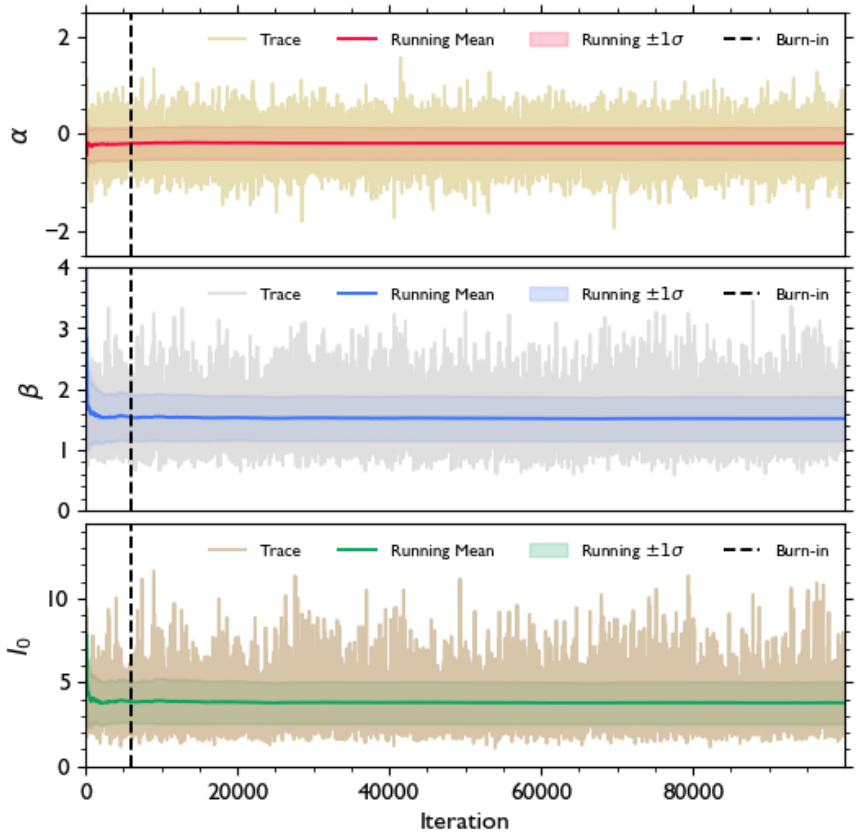

Figure 6: Trace plots for α, β and I_0 including the running mean and standard deviation of the chains. The trace plots show good mixing, an effective exploration of the parameter space and convergence. The running statistics stabilise and converge at higher iterations. The burn-in period is 5969 samples long.

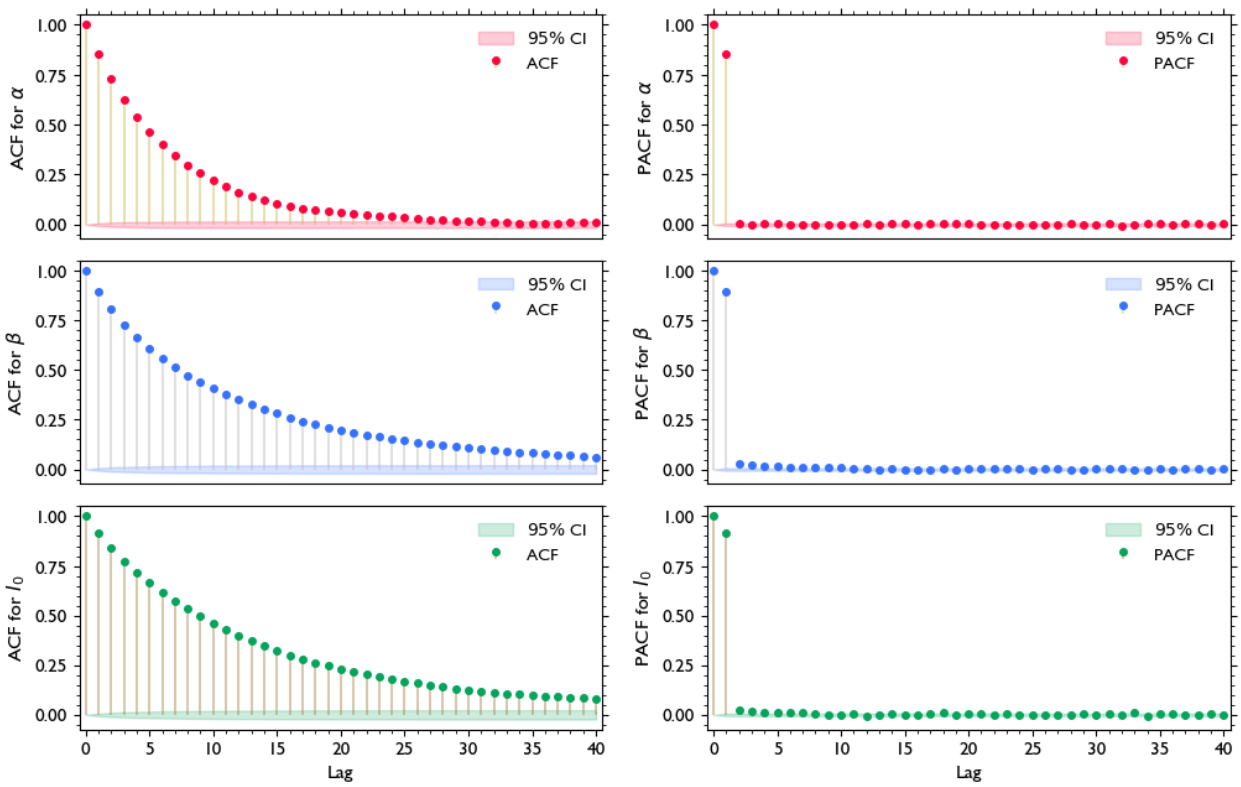

Figure 7: ACF and PACF for α, β and I_0. The ACF for α decays relatively rapidly indicating good mixing and efficient sampling from the target distribution. The ACFs for β and I_0 decay less rapidly indicating worse mixing. The PCFs visualise that the Markov chains satisfy the Markov property, i.e. each state only depends on the previous state.

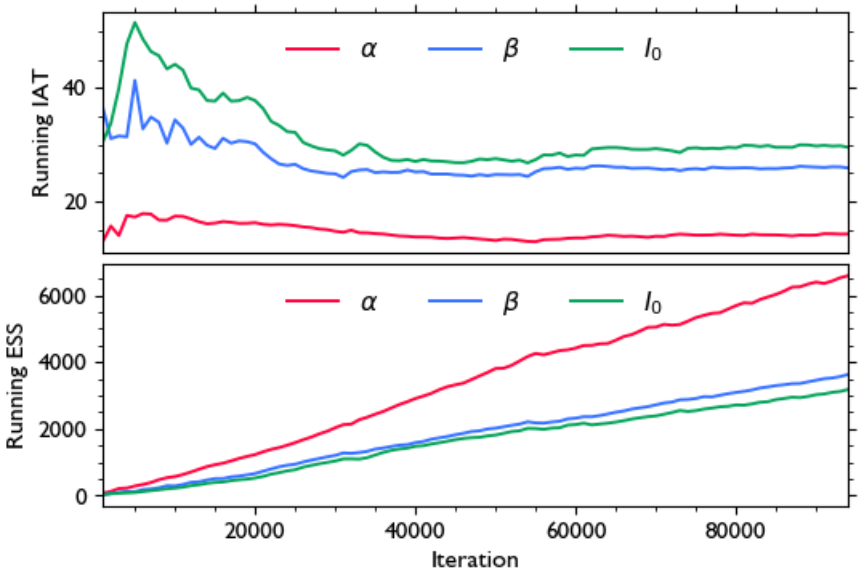

Figure 8: Running IAT and ESS for α, β and I_0. The IAT stabilises for all parameters at higher iterations.

3.2.4 Results

We use the refined MCMC chain to estimate the posterior distribution $\ln P(\alpha, \beta, I_0 | \{x_k\}, \{I_k\})$. Figure 9 shows the joint and marginalised posteriors on α, β and I_0. All parameters are well constrained. The summary statistics for our parameter estimates are shown in Table 2. Our parameter estimates in the form of the mean and standard deviation are $\hat{\alpha} = -0.19 \pm 0.33$, $\hat{\beta} = 1.51 \pm 0.36$ and $\hat{I_0} = 3.80 \pm 1.26$. The posterior distributions on β and I_0 are slightly skewed, hence we also quote the median and 68% CI, i.e. $\hat{\beta}_{med} = 1.47^{+0.39}_{-0.30}$ and $\hat{I_0}_{med} = 3.59^{+1.36}_{-0.97}$.

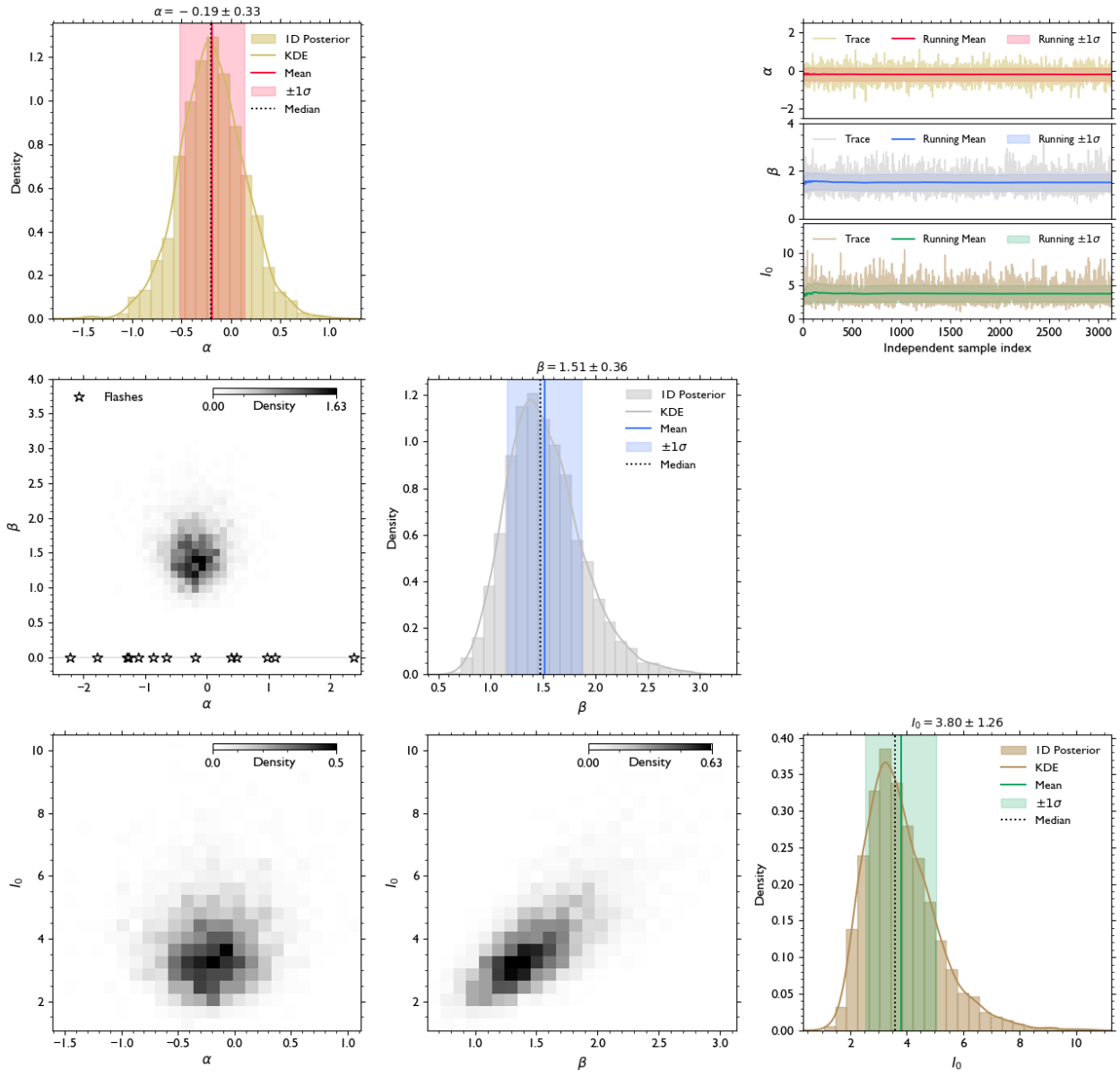

Figure 9: Corner plot of our posterior distribution $\ln P(\alpha, \beta, I_0 | \{x_k\}, \{I_k\})$, trace plots and running statistics of the independent samples from the posterior (refined chain).

Table 2: Summary statistics of the Bayesian posterior distribution on α, β and I_0.

Parameter	Mean	Standard Deviation	Median	68% CI	95% CI
α	-0.19	$\pm$0.33	-0.20	[-0.50, 0.13]	[-0.86, 0.47]
β	1.51	$\pm$0.36	1.47	[1.17, 1.86]	[0.93, 2.32]
I_0	3.80	$\pm$1.26	3.59	[2.62, 4.95]	[1.98, 6.79]

4 Comparison

4.1 Part (viii)

Original vs Extended Lighthouse Problem

We compare the inference results from the original and extended lighthouse problem in Part (vii) in Figure 10 and Table 3. The two-dimensional kernel density estimate (KDE) [28] shows that the location of the lighthouse is more well-constrained in the extended problem results. The marginal posterior distribution on α is also more peaked. Specifically, the uncertainty (standard deviation) of the parameter estimate $\hat{\alpha}$ is almost halved, and the mean and median move closer to 0. The 68% and 95% CIs for α are also significantly narrower.

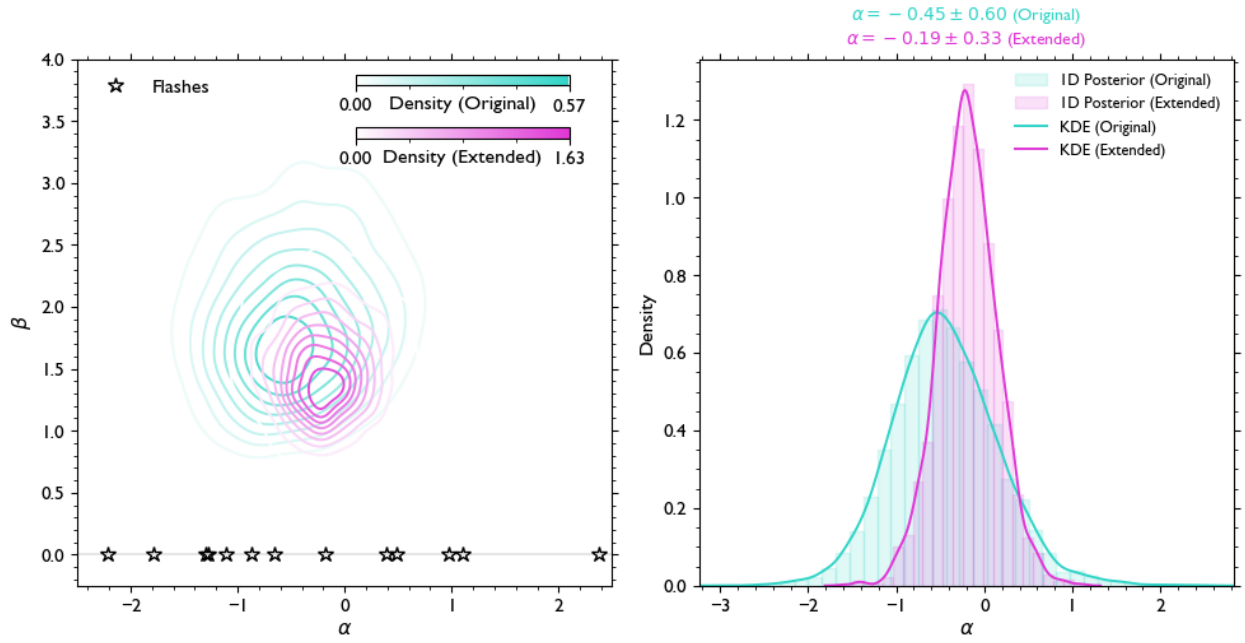

Figure 10: Left: Two-dimensional KDEs of the joint posterior distributions on α and β from the original and extended lighthouse problem. *Right:* One-dimensional histograms and KDEs of the marginalised posteriors on α from the original and extended lighthouse problem.

Table 3: Summary statistics of the posterior distribution on α for the original and extended problem.

Parameter	Mean	Standard Deviation	Median	68% CI	95% CI
α (Extended Problem)	-0.45	$\pm$0.60	-0.47	[-1.02, 0.13]	[-1.57, 0.76]
α (Original Problem)	-0.19	$\pm$0.33	-0.20	[-0.50, 0.13]	[-0.86, 0.47]

Including the intensity data $\{I_k\}$ in addition to the flash locations $\{x_k\}$ improves the quality of our parameter estimate of α by reducing the parameter estimate uncertainty and making its posterior distribution more well-constrained. This is due to the following reasons:

1. **Richer data structure**: Adding another data dimension provides more information about the parameter space, improving parameter estimation precision.
2. **Capturing relationships:** It reveals correlations, improving parameter estimation accuracy.
3. **Improved model robustness:** It makes the Bayesian inference more resilient to noise and outliers, improving parameter estimation reliability.

Bibliography

[1] Stephen F Gull. Bayesian inductive inference and maximum entropy. In Maximum-entropy and bayesian methods in science and engineering: foundations, pages 53–74. Springer, 1988.

[2] Devinderjit Sivia and John Skilling. Data analysis: a Bayesian tutorial. OUP Oxford, 2006.

[3] Ronald A Fisher. On the mathematical foundations of theoretical statistics. Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character, 222(594-604):309–368, 1922.

[4] Roger J Barlow. Statistics: a guide to the use of statistical methods in the physical sciences, volume 29. John Wiley & Sons, 1993.

[5] Thomas Bayes. Lii. an essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfr s. Philosophical transactions of the Royal Society of London, (53):370–418, 1763.

[6] Trevor Hastie et al. The elements of statistical learning: data mining, inference, and prediction, volume 2. Springer, 2009.

[7] Andrey Andreyevich Markov. Extension of the law of large numbers to dependent quantities. Izv. Fiz.-Matem. Obsch. Kazan Univ.(2nd Ser), 15(1):135–156, 1906.

[8] Nicholas Metropolis and Stanislaw Ulam. The monte carlo method. Journal of the American statistical association, 44(247):335–341, 1949.

[9] Christian P Robert, George Casella, and George Casella. Monte Carlo statistical methods, volume 2. Springer, 1999.

[10] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. Handbook of markov chain monte carlo. CRC press, 2011.

[11] Dani Gamerman and Hedibert F Lopes. Markov chain Monte Carlo: stochastic simulation for Bayesian inference. Chapman and Hall/CRC, 2006.

[12] Radford M Neal. Probabilistic inference using markov chain monte carlo methods. 1993.

[13] James R Norris. Markov chains. Number 2. Cambridge university press, 1998.

[14] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. The American Statistician, 49(4):327–335, 1995.

[15] Nicholas Metropolis et al. Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6):1087–1092, 1953.

[16] W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. Biometrika, 57(1):97—-109, 1970.

[17] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. IEEE Transactions on pattern analysis and machine intelligence, (6):721–741, 1984.

[18] Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. Physics letters B, 195(2):216–222, 1987.

[19] Luke Tierney. Markov chains for exploring posterior distributions. the Annals of Statistics, pages 1701–1728, 1994.

[20] Andrew Gelman, Walter R Gilks, and Gareth O Roberts. Weak convergence and optimal scaling of random walk metropolis algorithms. The annals of applied probability, 7(1):110–120, 1997.

[21] Andrew Gelman, Gareth O Roberts, and Walter R Gilks. Efficient metropolis jumping rules. Bayesian statistics 5, 5:599–608, 1996.

[22] BP Welford. Note on a method for calculating corrected sums of squares and products. Technometrics, 4(3):419–420, 1962.

[23] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. Markov chain Monte Carlo in practice. CRC press, 1995.

[24] Skipper Seabold and Josef Perktold. statsmodels: Econometric and statistical modeling with python. In 9th Python in Science Conference, 2010.

[25] Daniel Foreman-Mackey, David W Hogg, Dustin Lang, and Jonathan Goodman. emcee: the mcmc hammer. Publications of the Astronomical Society of the Pacific, 125(925):306, 2013.

[26] William A Link and Mitchell J Eaton. On thinning of chains in mcmc. Methods in ecology and evolution, 3(1):112–115, 2012.

[27] Harold Jeffreys. An invariant form for the prior probability in estimation problems. Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences, 186(1007):453–461, 1946.

[28] David W Scott. Multivariate density estimation: theory, practice, and visualization. John Wiley & Sons, 2015.

A Use of auto-generation tools

A.1 Co-Pilot

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

A.2 ChatGPT

A.2.1 Prompt 1

Prompt:

What is a fast way of tracking the running mean and standard deviation in Python?

Output:

A fast way to track the running (or moving) mean and standard deviation in Python, particularly in situations where you have streaming data or if you're processing a large dataset in chunks, is to use an algorithm that updates these statistics incrementally. One common approach for this is Welford's algorithm, which is designed for numerical stability and efficiency.

Here's how you can implement Welford's algorithm in Python to calculate the running mean and standard deviation:

```python
def update_running_stats(count, mean, M2, new_value):
    count += 1
    delta = new_value - mean
    mean += delta / count
    delta2 = new_value - mean
    M2 += delta * delta2
    return count, mean, M2

def finalize_stats(count, M2):
    variance = M2 / count if count > 1 else float('nan')
    stddev = variance ** 0.5
    return variance, stddev
```

Modification:

The output was modified by combining the suggested into one function and calculating the running mean without the logic of the algorithm, but the running standard deviation with the logic of the algorithm. Moreover, comments and the use of `numpy` were introduced:

```python
def welfords_method(samples):
    """
    Calculate the running mean and standard deviation of a set of samples using
    Welford's method.

    References
    ----------
    Welford, B.P. (1962) Note on a method for calculating corrected
    sums of squares and products. Technometrics, 4(3), 419-420.
    """
    # Calculate running mean
    running_mean = np.cumsum(samples)/(np.arange(len(samples)) + 1)
    # Initialise variables
    n = 0
    mean = 0
    M2 = 0
    running_std = np.zeros(len(samples))
    # Loop through the samples
    for i, x in enumerate(samples):
        n += 1
        # Calculate mean and M2
        delta = x - mean
        mean += delta / n
        delta2 = x - mean
```

```
        M2 += delta * delta2
        # Calculate running standard deviation
        if n < 2:
            running_std[i] = float('nan')
        else:
            variance = M2 / (n - 1)
            running_std[i] = np.sqrt(variance)
    return running_mean, running_std
```

A.2.2 Prompt 2

Prompt:

How do you write a curvy L in Latex?

Output:

In LaTeX, to write a curvy L, often used to denote the set of all continuous functions or the L-operator in certain mathematical contexts, you can use the `\mathcal{L}` command within math mode. This command produces a calligraphic or curvy "L".

Modification:

The `\mathcal{L}` command was used to generate curvy symbols in the report.

A.2.3 Prompt 3

Prompt:

What is the original paper on Markov chains?

Output:

The foundational work on Markov chains was introduced by Andrey Andreyevich Markov, a Russian mathematician, in a paper originally written in Russian. The seminal paper that laid the groundwork for what would become known as Markov Chains is titled "Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga" which translates to "Extension of the Law of Large Numbers to Dependent Quantities." This work was presented in 1906.

However, Markov's further exploration and significant contributions to the field, particularly his analysis applying his theories to the poem "Eugene Onegin" by Alexander Pushkin, were detailed in his later work published in 1913. This analysis showcased the practical application of what he termed "Markov Chains" in studying the sequence of vowels and consonants, illustrating the dependency of one event on its predecessor.

These contributions are foundational to the theory of stochastic processes, influencing a wide range of disciplines from mathematics and physics to economics and beyond. The original works, deeply mathematical and theoretical at their core, detailed the conditions under which sequences of dependent random variables converge to a particular distribution, the groundwork for what is now a vast field of study in both theory and application.

Modification:

The paper name was entered to Google Scholar to retrieve the citation [7].

B README.md

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

S2 Statistics for Data Science Coursework Submission (sd2022)

Description

This project is associated with the submission of the coursework for the S2 Statistics for Data Science Module as part of the MPhil in Data Intensive Science at the University of Cambridge. The Coursework Instructions can be found under lighthouse.pdf. The associated project report can be found under S2 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 main.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 S2 Statistics for 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 s2_sd2022_env
   ```

4. **Install ipykernel:**

   To run the notebook cells with `s2_sd2022_env`, install the ipykernel package with the following command:

   ```
   python -m ipykernel install --user --name s2_sd2022_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 `s2_sd2022_env`

environment and run the main.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 S2 Statistics for 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=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
   ```

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

   Note: Make sure that no 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.

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. Other uses of auto-generation tools:

ChatGPT Prompt 1: What is a fast way of tracking the running mean and standard deviation in Python?

- ChatGPT output:

```
def update_running_stats(count, mean, M2, new_value):
    count += 1
    delta = new_value - mean
    mean += delta / count
    delta2 = new_value - mean
    M2 += delta * delta2
    return count, mean, M2

def finalize_stats(count, M2):
    variance = M2 / count if count > 1 else float('nan')
    stddev = variance ** 0.5
    return variance, stddev
```

- Modification of the output:

```
def welfords_method(samples):
    """
    Calculate the running mean and standard deviation of a set of samples u
    Welford's method.

    References
    ----------
    Welford, B.P. (1962) Note on a method for calculating corrected
    sums of squares and products. Technometrics, 4(3), 419-420.
    """
    # Calculate running mean
    running_mean = np.cumsum(samples)/(np.arange(len(samples)) + 1)
    # Initialise variables
    n = 0
    mean = 0
    M2 = 0
    running_std = np.zeros(len(samples))
    # Loop through the samples
    for i, x in enumerate(samples):
        n += 1
        # Calculate mean and M2
        delta = x - mean
        mean += delta / n
        delta2 = x - mean
        M2 += delta * delta2
        # Calculate running standard deviation
        if n < 2:
            running_std[i] = float('nan')
        else:
            variance = M2 / (n - 1)
            running_std[i] = np.sqrt(variance)
    return running_mean, running_std
```

Authors and Acknowledgment

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

28th March 2024