S1 Principles of Data Science Coursework Report

Thomas Breitburd

University of Cambridge

0.1 Section A

(a)

We have the continuous random variable $M \in [5; 5.6]$. Our model is the weighted sum of a background and signal such that:

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

where:

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

We want to show that:

$$I = \int_{-\infty}^{+\infty} p(M; f, \lambda, \mu, \sigma)\, dM = 1 \tag{2}$$

We have:

$$I = \int_{-\infty}^{+\infty} fs(M; \mu, \sigma) + (1 - f)b(M; \lambda), dM$$
$$I = \int_{-\infty}^{+\infty} f \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(M-\mu)^2}{2\sigma^2}}\, dM + \int_{0}^{+\infty} (1 - f)\lambda e^{-\lambda M}\, dM$$

Since the exponential decay distribution is only defined from 0 to $+\infty$. We can first evaluate the 2^{nd} integral:

$$\int_{0}^{+\infty} (1 - f)\lambda e^{-\lambda M}\, dM = (1 - f)[-e^{-\lambda M}]_0^{\infty} = (1 - f)[0 - (-1)] = (1 - f)$$

Then, we evaluate the integral of the signal part of the model, taking the f weight out. We use a change of variable so that $u = \frac{(M-\mu)}{\sigma} \iff du = \sigma dM$, thus:

$$J = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(M-\mu)^2}{2\sigma^2}}\, dM = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{(u)^2}{2}}\, du$$

We can square multiply this integral by itself, using dummy variables x and y:

$$J^2 = \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{(x)^2}{2}}\, dx \times \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{(y)^2}{2}}\, dy$$

$$J^2 = \tfrac{1}{2\pi} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} e^{-\frac{(x)^2}{2}} e^{-\frac{(y)^2}{2}} \, dx \, dy$$
$$J^2 = \tfrac{1}{2\pi} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} e^{-\frac{1}{2}(x^2+y^2)} \, dx \, dy$$

From here, we can switch to polar coordinates to be able to evaluate this. We have $x = r\cos(\theta) y = r\sin(\theta)$. This means we need to change the limits to polar equivalents. We will get $r \in [0, \infty]$ and $\theta \in [0, 2\pi]$. Further, $dxdy = rdrd\theta$:

$$J^2 = \tfrac{1}{2\pi} \int_{0}^{2\pi} \int_{0}^{+\infty} e^{-\frac{1}{2}r^2} r \, dr \, d\theta$$
$$J^2 = \tfrac{1}{2\pi} \int_{0}^{2\pi} \left[-e^{-\frac{1}{2}r^2}\right]_0^{\infty} d\theta$$
$$J^2 = \tfrac{1}{2\pi} \int_{0}^{2\pi} [-0 - (-1)] \, d\theta$$
$$J^2 = \frac{1}{2\pi} \int_{0}^{2\pi} 1 \, d\theta$$
$$J^2 = \frac{1}{2\pi} [\theta]_0^{2\pi}$$
$$J^2 = \frac{1}{2\pi} [2\pi - 0]$$
$$J^2 = 1$$

This then means that $J = \pm 1$ but since $s(M; \mu, \sigma) > 0 \forall M \in [-\infty, \infty]$, $J = 1$. Thus: $I = (1 - f) + f = 1$.

(b)

We want to find an expression of $p(M; \vec{\theta})$ such that it is normalised between α and β. Now because the signal fraction f is such that:

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

This means that we need to normalise the signal and background components separately. This gives:

$$\int_{\alpha}^{\beta} f N_s \times s(M; \mu, \sigma) + (1 - f) N_b \times b(M; \lambda) \, dM = 1 \tag{4}$$

with:

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

$$\frac{1}{N_b} = \int_{\alpha}^{\beta} b(M;\lambda)\, dM. \tag{6}$$

where N_s and N_b are a function of parameters $\vec{\theta}$. Now, from the definition of the Cumulative Distribution Function (c.d.f.): $F(X') = \int_{-\inf}^{X'} f(X)\, dX$, where X' is a specific value of the random variable X, and $f(X)$ is the p.d.f.(The lower limit is 0 for the exponential decay distribution). This means we can write (5) and (6) as:

$$\frac{1}{N_s} = \int_{\alpha}^{\beta} s(M;\mu,\sigma)\, dM = F_s(\beta) - F_s(\alpha) \tag{7}$$

$$\frac{1}{N_b} = \int_{\alpha}^{\beta} b(M;\lambda)\, dM = F_b(\beta) - F_b(\alpha) \tag{8}$$

We know that the normal and exponential decay distributions have:

$$Normal : F(X) = \frac{1}{2}[1 + erf(\frac{X-\mu}{\sigma\sqrt{2}})] \tag{9}$$

$$Exponential\ Decay : F(X) = 1 - e^{-\lambda X} \tag{10}$$

From equation (7) and (8), we can write the normalisation factors as:

$$\frac{1}{N_s} = \frac{1}{2}[1 + erf(\frac{\beta-\mu}{\sigma\sqrt{2}})] - \frac{1}{2}[1 + erf(\frac{\alpha-\mu}{\sigma\sqrt{2}})] = \frac{1}{2}[erf(\frac{\beta-\mu}{\sigma\sqrt{2}}) - erf(\frac{\alpha-\mu}{\sigma\sqrt{2}})] \tag{11}$$

$$\frac{1}{N_b} = (1 - e^{-\lambda\beta}) - (1 - e^{-\lambda\alpha}) = e^{-\lambda\alpha} - e^{-\lambda\beta} \tag{12}$$

Finally, this gives the total p.d.f. of the model as:

$$p(M;\vec{\theta}) = \frac{f}{2}[erf(\frac{\beta-\mu}{\sigma\sqrt{2}}) - erf(\frac{\alpha-\mu}{\sigma\sqrt{2}})] \times s(M;\mu,\sigma) + (1-f)[e^{-\lambda\alpha} - e^{-\lambda\beta}] \times b(M;\lambda) \tag{13}$$

(c)

In this question, we check that the integral of the p.d.f. between α and β does equal unity. For this specific case, the `scipy.integrate` library is used to numerically integrate the component-wise normalised p.d.f. as described in equation (4). Though N_s and N_b, are computed using the `scipy.integrate` library, like in (5) and (6). The components are computed using `scipy.stats` library's `norm.pdf` and `expon.pdf` methods (cf. `funcs.py` file, `pdf_norm` function). Then, the weighted sum of the two is computed. Finally, that p.d.f. is integrated from α to β, with randomly generated $\vec{\theta}$ parameters, using `random.uniform`. The results are shown in the table below:

μ	σ	λ	f	$Integral$
5.2357	0.0190	0.3643	0.5044	1.0
5.5436	0.0105	0.3823	0.6081	1.0
5.3184	0.0214	0.6424	0.6882	1.0
5.3621	0.0221	0.3695	0.3625	1.0
5.3640	0.0231	0.5015	0.2442	1.0

Table 1: Results for different values of $\vec{\theta}$ parameters. (cf. `solve_part_c.py` file)

(d)

For this question, the true values of the parameters were set and used to plot the p.d.f. of the model, along with the signal and background component overlayed. The parameters are assumed to be:

$$\mu = 5.28;\ \sigma = 0.018;\ \lambda = 0.5;\ f = 0.1$$

The `signal_norm`, `background_norm`, and `pdf_norm` functions, described in (c) were used to compute the p.d.f. of the model, signal, and background, normalised for $M \in [5, 5.6]$. The results are shown in the figure below:

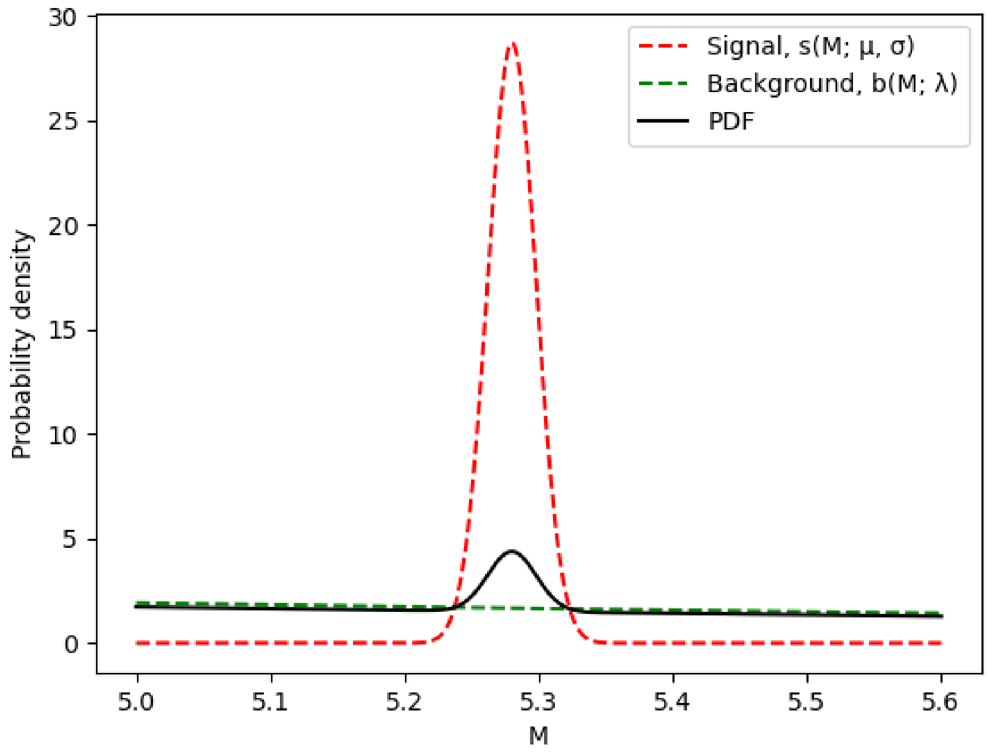

Figure 1: Plot of the total p.d.f. of the model, signal, and background, normalised for $M \in [5, 5.6]$.

This shows the impact of the f weight, which considerably reduces the size of the signal component in the total p.d.f..

(e)

The "true" p.d.f., signal and background component are now known and have been visualised. The context is now to generate a dataset from this "true" p.d.f., and to assume this is an observed sample obtained in an experiment. From this sample, the goal is now to try and estimate these "true" parameters, using a chosen estimation method.

To generate the sample, an accept-reject method was used. The accept-reject method requires us to know the p.d.f. function over the M domain, and the maximum value of the p.d.f. in the M domain, y_{max}. Uniformly random values of M between α and β are generated, using `random.uniform`. Then, for each of these values, many uniform random values of y are generated between 0 and y_{max}. If these generated y's are such that $y \leq p(M;\vec{\theta})$ for the generated M then the **M value** is accepted and added to the sample. This is repeated until the sample size is reached (cf. `funcs.py` file, `accept_reject` function).

The chosen estimation method is the Maximum Likelihood Estimation (MLE) method. This method relies on the likelihood function, which stands for the probability of observing the data given a set of parameters and is defined as:

$$\mathcal{L}(\vec{\theta}) = \prod_{i=1}^{N} p(M_i;\vec{\theta}) \tag{14}$$

It is a function of the parameters $\vec{\theta}$ only. The goal of the MLE method is to find estimates of the parameters $\vec{\theta}$ that maximise the likelihood function. In other words, finding $\hat{\vec{\theta}}$, such that:

$$\frac{\delta\mathcal{L}(\hat{\vec{\theta}})}{\delta\theta} = 0 \tag{15}$$

This result is because maximising the probability of observing M given a certain set $\hat{\vec{\theta}}$ means we have found the set of parameters $\hat{\vec{\theta}}$ that are the most likely to be the ones the data was observed under. Now because summing small numbers is less of a problem as multiplying small numbers in computations, we take the logarithm of the likelihood function, to turn the product into a sum. And taking twice the negative of that gives a negative log-likelihood function:

$$-2\ln\mathcal{L}(\vec{\theta}) = -2\sum_{i=1}^{N}\ln p(M_i;\vec{\theta}) \tag{16}$$

And this is the function to now **minimise**. To do so we use a python package named `iminuit`, which possesses a very useful set of minimising tools. The `iminuit` package requires a function to minimise, called a cost function, and a set of initial guesses for the parameters. Luckily, `iminuit` possesses a series of standard cost functions. The one used here is the `BinnedNLL`, Binned Negative Log-Likelihood, which takes in the c.d.f. of a **normalised** probability density and the observed **binned** data we want to fit that density to. With the cost function defined, `iminuit`'s main function `Minuit`, a function minimizer and error computer, takes that cost function in along with initial guesses for the parameters. These initial guesses are usually educated guesses at what the true parameters may be. Then we call the `migrad` method, which is a gradient descent method, to conduct the minimization.

An important point is that `iminuit.Minuit` reads the function given in the cost function definition to read the parameters it needs to fit for. In other words, using the `pdf_norm` function from (d), `Minuit.migrad()` would minimise the cost function for the $\vec{\theta}$ parameters as well as α and β.

Furthermore, in the interest of performance, the `numbastats` library was used to speed up any computation that uses our new p.d.f. function. This library provides `numba`-accelerated statistical function for common probability distributions. So the c.d.f. of the model was defined to only take the $\vec{\theta}$ parameters. It then uses the `numba_stats.truncnorm.cdf` and `.truncexpon.cdf` functions. Running the minimisation, the following parameter estimates and their uncertainties were obtained:

$$\hat{\mu} = 5.27939 \pm 0.00033;$$
$$\hat{\sigma} = 0.01820 \pm 0.0032;$$
$$\hat{\lambda} = 0.490 \pm 0.019;$$
$$\hat{f} = 0.1008 \pm 0.0017$$

We can see that the minimisation performed well, with close estimates and relatively small uncertainties. And the following plot of the data and the fitted model was obtained:

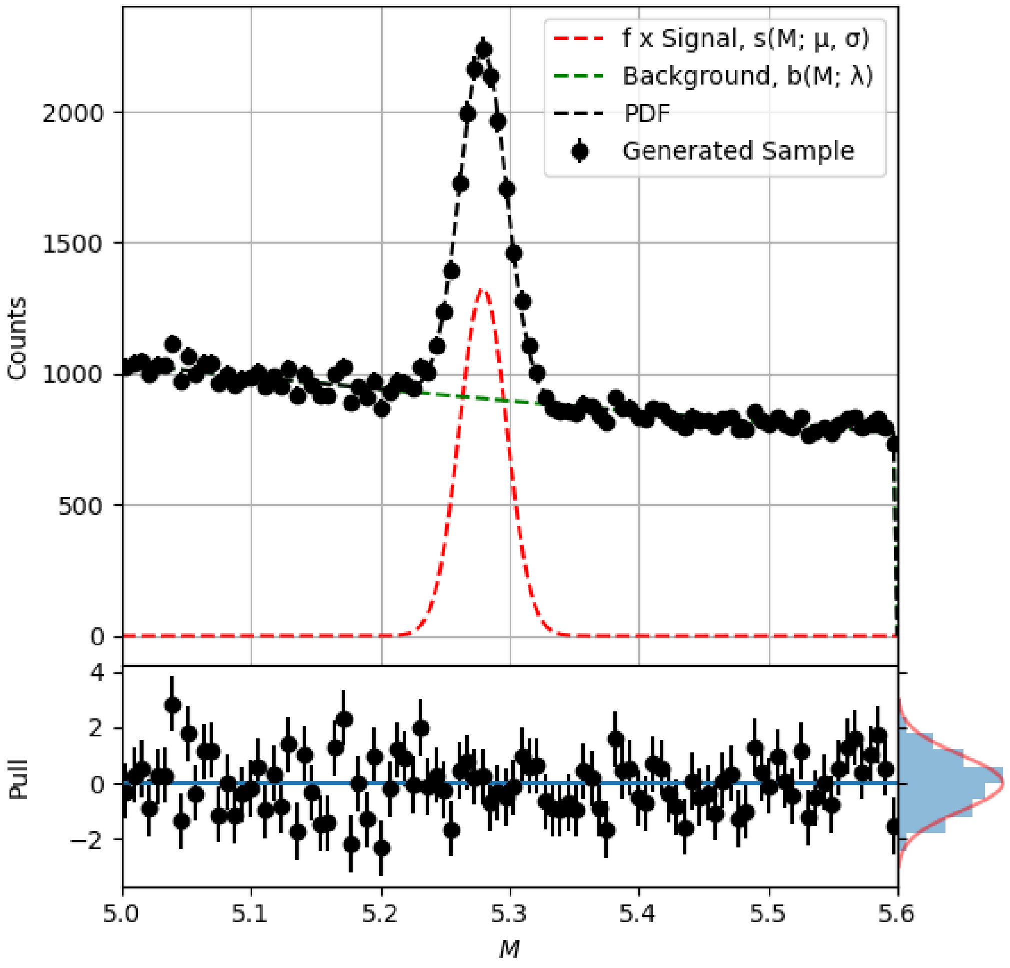

Figure 2: Plot of the data and the fitted model.

The lower plots are plots of the pull: $pull = \frac{obs-pred}{\sqrt{N}}$, where N is the bin counts. The pull plots are a good way to check that the uncertainties are sensible. If there were a visible trend in the pull v M plot, or if the pull distribution plot on the right was not normally distributed, then the uncertainties would be too small or too large.

0.2 Section B

blabla