CMBI - Coursework 1 - Report

Yoga Advaith Veturi
17056549
advaith.veturi.17@ucl.ac.uk

The main script for this report is `parametric_models.mlx`, although some functions are implemented in separate `.m` files. Specific lines of code that I refer to are within the main script, but when necessary, I refer to the specific scripts within the sub-questions. Relevant tables and figures are presented in the Figures document that is attached with the submission.

Question 1.1.1

To perform diffusion tensor estimation, I first implemented the least squares regression model on a single voxel `Avox = (:, 92, 65, 72)`. The code for this is in lines 12-19. This model returns a 7-dimensional vector containing S(0,0), the model signal with b=0, and the 6 elements of the diffusion tensor D. Next, I implemented the model on all voxels by performing a for loop on each voxel (lines 20-86). The parameters are stored in `dt_map72`, a (145, 174, 7) mapping. This map was used to compute the mean diffusivity and fractional anisotropy map, which are presented in Figure 1(a, b). A directionally encoded color map was also created by taking the principal eigenvector of the diffusion tensor D (the eigenvector with largest eigenvalue) and visualizing the values as the RGB channels of the image. Note this eigenvector was weighted by the fractional anisotropy values. The color map is visualized in Figure 1(c).

Question 1.1.2

The ball-and-stick model was implemented in this section, for which the sum of squared difference (SSD) function is in `BallStickSSD.m`. This produces a 5-dimensional parameter vector `parameter_hat` containing the model parameters S0, d, f, θ and ϕ.

First, I estimate the parameters on the same voxel from 1.1.1 (lines 87-105 of code listing). The parameter values and the `RESNORM` are as follows:

`parameter_hat = [3.52e+03 -3.99e-06 1.50e+02 8.856e-01 1.55e+00] RESNORM = 2.8654e+07`

To determine the quality of the model fitting for this voxel, I substituted these parameter estimates into the model to obtain predicted signal values `S` (lines 135-149 of code listing). This `S` is visualized with the true signal measurements `Avox` in Figure 2. The fitting is extremely poor for the b=0 values. Given that the standard deviation of the measurements is 200, we can estimate an upper bound on the RESNORM to be

$$RESNORM \leq N\sigma^2$$

where $\sigma = 200$ and $N = 108$. Thus, the expected RESNORM that should be seen is at max 4.32×10^6. The current model fitting produces a greater RESNORM, which is not expected. Furthermore, the parameter values are not sensible as they lie outside their respective ranges - S0 and d must be positive while f must be in range [0, 1]. These constraints are not met, hence this result can be improved.

Question 1.1.3

To constrain the parameters, the transformation method is applied, for which the SSD function is in `BallStickSSD_transformation.m`. The chosen constraints for the ball-and-stick model parameters are shown in Table 1. Note that θ and ϕ are not constrained. The reason for this is because when using ϕ and θ to compute the fibre direction, the sine and cosine functions are applied on these parameters and because they are periodic functions, the result does not change regardless of whether the parameter is constrained or not.

The transformed ball-and-stick model is implemented on a single voxel in lines 150-173 of the code listing. The results are as follows:

`parameter_hat = [4.25e+03 1.14e-03 3.57e-01 -9.81e-01 5.79e-01] RESNORM = 5.872e+06`

Firstly, it is noticeable that the `parameter_hat` values are now constrained within their respective ranges, compared to the `parameter_hat` from 1.1.2. Secondly, the `RESNORM` on this voxel is smaller compared to the previous model,

indicating the transformed model performs better. This is also visualized by comparing the predicted signals `S` with the true measurements `Avox` in Figure 3. This improved performance can be attributed to enforcing constraints on the parameters, making them realistic with reference to the task at hand.

Question 1.1.4

To assess if the global minimum was reached, I implemented the transformed ball-and-stick model for 100 runs, where on each run, I added noise to the starting point in order to optimize from different starting points. This is implemented in lines 223-293 of the code listing. The noise added to each parameter was Gaussian with a mean of 0 and standard deviation chosen based on the magnitude of the parameter. For example, if S(0,0)=4.25e+03, then the STD was chosen as 1e+03. The STDs for the other parameters are listed in Table 2.

I found that in 96% of the runs, a minimum of 5.872×10^6 was seen. Given this frequency, it can be assumed that the global minimum is indeed the same. Using this, I estimate how many runs are required to be 95% sure the global minimum has been reached. To do so, I first define that the proportion of global minima is essentially the probability that a global minimum is found. Now, given that I performed N runs, I next define the probability that atleast 1 global minimum is found within those N runs. This can be represented using the binomial theorem formula:

$$P(\text{atleast 1 global minimum found in N runs}) = 1 - P(\text{No global minimum found in N runs})$$

$$P(\text{atleast 1 global minimum found in N runs}) = 1 - \binom{N}{0} p^0 (1-p)^N$$

where p is the proportion of global minima. As I want to be 95% sure that there is at least one global minimum in the N runs, I set $P(\text{atleast 1 global minimum found in N runs}) = 0.95$. Then the equation can be rearranged to find N. Using this method, I find N=0.93. I rounded this to 1 run.

It must be noted that the frequency of global minima and the required number of runs N can vary with different voxels. Hence, I also performed the above computations with other test voxels. The results for these voxels are shown in Figure 4 and Table 3. Most voxels required only 1 run to while some require 8-9 runs.

Question 1.1.5

Using the constraints defined in Table 1, I now run the ball-and-stick model on all voxels within the 72nd slice of the image volume (lines 317-352 of code listing). I first iterate over each voxel and then, I run the parameter estimation for multiple runs per voxel. Ideally, the number of runs to perform should be that for the voxel that requires the highest number of runs, hence I selected N=10 as a worst-case. The lowest RESNORM within the 10 runs was assumed to be the global minimum and the corresponding parameters were selected as the final parameters for that voxel.

The maps of S0, d, f and RESNORM are visualized in Figures 5(a, b, c, d) respectively. The parameters θ and ϕ which are related to the fibre direction $\mathbf{n}$ are visualized as a fibre direction map in Figure 5(e). Note that this fibre direction was weighted by the volume fraction f.

Question 1.2.1

The code for parametric bootstrapping is in `PBootstrap.m`. I implement this algorithm on a single voxel `Avox = (:, 92, 65, 72)` for 1000 iterations in order to define a posterior distribution for the S(0,0), d and f parameters (lines 353-395 of code listing). Within each iteration of the parameteric bootstrap procedure, 10 runs of optimization are performed in order to ensure that a global minimum is found on at least one of the runs. The posteriors for S(0,0), d and f are presented in Figures 6. The 2-sigma range and 95% range for each posterior is presented in Table 4. For comparison, boostrapping was also performed for 4 other voxels in the same image slice, for which results are presented in Figure 7 and Tables 5-8. From these results, it can be seen that for the d parameter, the values are very similar in most voxels, however for S(0,0) and f, the range of values varies between 3000 to 5000 for S0 and 1.5e-01 to 2.5e-01 for f.

Question 1.2.2

The code for the Markov Chain Monte-Carlo (MCMC) implementation is in `MCMC.m`. This algorithm was implemented on a single voxel `Avox = (:, 92, 65, 72)` for 1000 iterations (lines 396-401 in code listing). I used the pseudocode provided in the lecture notes to implement this algorithm, although it must be noted that when calculating the α likelihood ratio, rather than using the equations provided, in order to avoid numerical underflow issues, I instead compute the $\log(\alpha)$ term and then exponentiate. Applying the log function on α gives the following:

$$\log(\alpha(x_c, x_{t-1})) = \log(\sin(\theta_c)) - \log(\sin(\theta_{t-1})) + \frac{1}{\sigma^2}\left[\sum_{k=1}^{K}(A_k - S_k(x_{t-1}))^2 - \sum_{k=1}^{K}(A_k - S_k(x_c))^2\right] \quad (1)$$

The parameter settings I used for MCMC are provided in Table 9. The posteriors for S0, d and f are presented in Figures 8. Comparing these results with those from bootstrapping, it is noticeable that the S0 and f posteriors have a somewhat bell-curve distribution and similar ranges of values in both sampling methods. However, the posterior for d appears skewed in the MCMC case. Furthermore, the range of values on the histogram does not match that of the bootstrapping result, which could be due to an inappropriate choice of the size of the Gaussian perturbations on the parameter estimates or choice of the proposal distribution itself.

Question 1.3.1

In this section I implement the ball-and-stick model for a single voxel in the new dataset (lines 482-564 in the code listing). For this, I chose to use the `fminunc` optimization algorithm, and the same constraints are encoded via the transformation method. The starting point used is `[3.5e+00 3e-03 2.5e-01 0 0]`. In order to ensure that the global minimum is reached,, I implement the model for multiple runs where on every run, Gaussian noise with mean 0 and STD sigma (see Table 2) is used. The best fit parameter for the voxel is

```
parameter_hat = [1.0049e+00 3.7843e-02 7.4398e-01 -1.5447e+00 -8.2863e-02]
```

When performing 100 runs, the minimum RESNORM was found to be 15.6. This frequency of this error occuring is 0.13946. Given that the standard deviation of the noise in the data is 0.04, an upper bound can be found for the RESNORM, which is $N\sigma^2 = 3612 \times 0.04^2 = 5.7792$. From this, it can be stated that the calculated RESNORM from my model fitting is not expected.

Question 1.3.2

I next implemented the diffusion tensor model, the Zeppelin-Stick and the Zeppelin-Stick-and-Tortuosity models. The Zeppelin stick model is in the script `ZeppelinStickSSD.m` and the Zeppelin stick tortuosity model is in `ZeppelinStickTortuositySSD.m`.

For the diffusion tensor model, simple least squares regression is used, hence the model is just implemented once (lines 565-576 in code lisiting) unlike the other non-linear models which required multiple runs. The RESNORM for this model is found to be 1.0507e+04.

For the Zeppelin-stick model, there are two parameters: λ_1 and λ_2. λ_1 is associated with the fibre direction for the zeppelin; however the diffusivity d in the stick compartment is the same as the zeppelin along the fibre direction, hence I assume that $\lambda_1 = d$. Totally, there are 6 parameters, S0, d, f, λ_2, θ and ϕ. For λ_2, the constraints are that $0 < \lambda_2 \leq \lambda_1$. To enforce this, I define `d = x(1)^2*x(2)^2` while `lambda_2 = x(2)^2`. This transformation will ensure that the λ_1 will always be greater than λ_2, and λ_2 will always be positive. For the optimization, I used the start point `[3.5e+00 3e-03 2.5e-01 3e-03 0 0]` where an extra 3e-03 is inserted in the 4th index, corresponding to the start point for λ_2. Note that before optimizing I also perform the inverse transform $\sqrt{\lambda_2/2}$.

For this model, I obtained RESNORM=10.817 and the fitted parameters:

```
parameter_hat = [9.8303e-01, 1.6108e-03 4.3538e-01 9.3519e-04 -1.5438e+00 -9.2742e-02]
```

For the zepplin-stick and tortuosity model, since λ_2 is a function of λ_1 and $\lambda_1 = d$, this model will have only 5 parameters, similar to the ball-and-stick model. The RESNORM obtained for this model is 11.605 and the best fit parameters are:

```
parameter_hat = [9.9133e-01 1.53006e-03 5.1717e+01 -1.5442e+00 -8.7052e-02]
```

Question 1.3.3

For this section, I computed the AIC and BIC score for the Ball-and-Stick model, the Zeppelin-Stick and Zeppelin-Stick-and-Tortuosity model. The values are presented below:

Model	AIC	BIC
Ball and Stick	-1.9771e+04	-9.4831e+03
Zeppelin Stick	-2.0975e+04	-1.0690e+04
Zeppelin Stick and Tortuosity	-2.0723e+04	-1.0444e+04

Table 1: AIC and BIC scores for the three parameter estimation models.

From these results, it is noticable that the rankings of the models are consistent. The Zepplin Stick model has the lowest AIC score (-2.0975e+04) and lowest BIC score (-1.0690e+04), from which I conclude that the Zeppelin stick model is the best model of all three.