1 COMP0118 - Assignment 1

Student ID: 22197823 - 17th Feb 2023

Q1.1.1

code for this answer related to code lines number 1 to 137 Given the signal data and the Diffusion Tensor model as defined in the question, we want to estimate the parameters for the DT model for each voxel across image slice 72. We will do this using Weighted Linear Least Squares (WLLS) method on the *log* of the signal data (as suggested in the question). We use the WLLS method because we assume the noise in the measured signals are i.i.d Gaussian, but when we take the *log* of the signals they remain independent but stop being identical. The closed form solution of WLLS is, $\mathbf{x} = (Y^T W Y)^{-1} Y^T W A$. Where A is the *log* of the observed signals, Y is our design matrix as defined in the question, and W is approximated by a diagonal matrix with the $k-th$ diagonal entry equal to $k-th$ signal squared. See *Fig 1* to see estimated signals and observed signals at the voxel using the best fit parameter x.

Now we repeat the same process at every voxel in the image slice. Note, at each voxel we check if there are any zero signals, if there are we don't solve for x. Now we can calculate mean diffusivity and fractional anisotropy (FA) across the image. We also map a weighted FA with the diffusion tensor eigenvalues in that voxel. See *Fig 3* for results.

Q1.1.2

code for this answer related to code lines number 138 to 211

As suggested in the question, we try and fit our model on the single voxel using the non-linear fit method *scipy.optimize.minimize* in Python using the *BFGS* optimise algorithm. But using the given starting parameters ($x = [3300, 0.001, 0.45, 1, 1]$) the optimise function doesn't work, and it returns the starting parameter as the fitted parameter. We can see this is clearly wrong when visualising the fit with this parameter in *Fig 4*.

We can empirically check this as well. Assuming the standard deviation of the noise is Gaussian we have $\mathbb{E}(RESNORM) = N * \sigma^2$. This is because if we have a perfect fitting model the RESNORM is equal to the sum of the noise error squared, which has an expected value equal to the variance. So with $\sigma = 200$ we would expect RESNORM to be around $4.32e6$, and instead we have a RESNORM over ten times larger, $5.48e7$.

Because the optimising algorithm didn't work the fitted parameter didn't change. But we would expect our parameter to have the following constraints: $S(0,0) > 0$, $diff > 0$, $f \in [0,1]$, $\theta \in (0,\pi)$, $\phi \in (0, 2\pi)$

Q1.1.3

code for this answer related to code lines number 212 to 277

To constrain the fitted parameter to the domain we want it (eg. $S(,0,0) > 0$) we will use the transform method. To do this we use the same approach as in the previous question but the *scipy.optimize.minimize* Python function will be working in an unconstrained domain, and then the found fitted parameter $\tilde{x}$ will be transformed back into our constrained domain to give our wanted fitted parameter x.

For the Ball and Stick model we use the following transform, $x = [S(0,0), diff, f, \theta, \phi] = [\tilde{S}(0,0)^2, \tilde{diff}^2, logit(\tilde{f}), logit(\tilde{\theta}/\pi), logit(\tilde{\phi}/(2\pi))$ (where a tilde represents the unconstrained parameter, eg. $\tilde{\phi}$ is an unconstrained ϕ). Even though θ and ϕ can be any value on the Reals, we constrain $\theta \in (0,\pi)$ and $\phi \in (0, 2\pi)$ to ensure each point in our domain has a unique mapping for θ and ϕ, this makes it easy to check if we have found the same fitted parameter in later questions.

Using this method we find $x = [4257, 0.00114, 0.357, 2.161, 0.5794]$ which is within our forced constraints, the optimise function now works in Python using the transformation method. The RESNORM has significantly reduced to $5.87e6$ which is much more aligned with our expected RESNORM of $5.48e7$. The RESNORM has reduced significantly because the minimise function has the objective to reduce the RESNORM, so it's found a minimum (maybe global min) that does that. See the plotted results in *Fig 5*.

Q1.1.4

code for this answer related to code lines number 281 to 344

Ideally we want to find x at the global minimum of minimising SSD. To do this we repeat the previous process but with a different random starting point each iteration. We perturb the starting point by adding Gaussian Noise to our original starting point ($x = [3300, 0.001, 0.45, 1, 1]$) with each noise dimension using a standard deviation of the correct scale ($\sigma = x/5$). This is so we explore realistic starting points, there is no point starting where we know there is no solution (eg. $S(0,0) << 3000$).

Using this method for 100 iterations on the single voxel the smallest RENORM we find is the same as we found in the previous question ($5.87e6$), and we achieve this 82% of the time. We can't be sure that this is the minimum, it is possible the loss space has a large local minimum "bowl" and a very small "pointy" global minimum, making it exceptionally hard to find the global minimum. But considering we have explored the loss space 100 times with different starting positions and find the same minimum there is a good chance we have found the global minimum.

If we assume we have found the global minimum we can assume the $P(findglobalmininonerun) = p = 0.82$. Modelling each run as a Bernoulli event of either finding the global min or not, we can calculate the number of times we need to do a run to find the global min with a probability of 0.95, this is $log(0.05)/log(1-p)$. We also round this up to be an integer. For this voxel we calculate we only need 2 runs to have a 95% chance of finding the global min

Repeating this for a number of other voxels we find a range of N runs required, the highest being 10, therefore across the whole image slice we will use 10 runs. See *Table 1* for results.

Q1.1.5

code for this answer related to code lines number 344 to 434

Using the Transform method above on each voxel in the image slice $N = 10$ times we get the results shown in *Fig 6*. Note in the RESNORM mapping bright spots show where our fitted model doesn't fit the data very well. In particular there is a region in the bottom right of the image which isn't fitting well compared to the rest of the brain. To get around this we could increase the number of times we run the optimise algorithm in just this region.

Q1.2.1

code for this answer related to code lines number 434 to 520

We now use the classical bootstrap to find fitted parameters. We do this by sampling with replacement from our original voxel signal data to create a sampled dataset the same size, and then fit using our new sampled dataset with the transform constrain optimise method (with 95% confidence level). Results of this can be seen for our original voxel in *Table 2*. Bootstrap distributions of each parameter for a number of voxels can be seen in *Fig 8*. In the figure we can see the distributions are clearly not normal because the 95% grey region doesn't not match the 2σ range. Also the distribution for f is not right. Bootstrap is fitting f to be close to 1 which is not correct, and due to the restriction of $f \in (0, 1)$ the distribution is very skewed. We got similar results for the other voxels we tried.

Q1.2.2

code for this answer related to code lines number 520 to 659

Now we use MCMC method. We do this by peterbing our parameter x with gaussian noise to get y. We decide whether to keep y using the likelihood ratio α. Note that because we perturb our parameters with Gaussian noise we aren't sampling the parameter space of $\mathbf{n}$ due to it being a unit sphere. Therefore we have to weight $\alpha(y, x)$ with $sin(\theta_y)/\sin(\theta_x)$ to compensate. I changed the perturbing noise until I had about a 45% acceptance rate. This gives us a sequence that is eventually sampled from $P(x|A)$. We use a burn in of 2000 to make sure our sequence has settled into the main mass of the distribution $P(x|A)$, and we only keep every 5-th sample to make sure our points are independently sampled. On the left of *Fig 9* we see the kept

sampled sequence has converged onto a distribution, and on the right we see the 2σ range matches with our grey 95% region well. The MCMC mean parameter for the voxel is in *Table 3*.

Plotting the fitted mean parameter results from MCMC vs Bootstrap in *Fig 10* we see that MCMC preforms much better. It looks like due to sampling the data in Bootstrap it meant it fitted to the bulk of the signal below 2500, whereas MCMC fits to the whole data set.

Q1.3.1

code for this answer related to code lines number 659 to 778

Using the new data set we use the transform constrain optimise method used in Q1.1.3. We now have 6 voxels. We use the starting parameter $x = [1, 0.005, 0.8, 1, 1]$. I found this starting position by altering the starting point until all 6 voxels had to be run under 100 times to have a 95% confidence of finding the global min. The transform constraint and optimise method is the same as before in Q1.1.3. See *Table 4* for the results on all voxels. We would expect a RESNORM of $N * \sigma^2 = 3612 * 0.04^2 = 5.78$. In all of our voxels we are about 3 to 4 times larger than this, indicating our model isn't fitting perfectly but is of the right order. You can visualise the fit in *Fig 11*.

Q1.3.2

code for this answer related to code lines number 778 to 1055

We repeat the above method for 4 more models and adapt the starting parameter accordingly and constrain accordingly. The visual results, with the RESNORM for the DT model, Zeppelin model and the Zeppelin with Turtuosity model are in *Figures 12, 13, 14* respectively.

Q1.3.3

code for this answer related to code lines number 1055 to 1144

We now use AIC and BIC to rank the four models for each voxel separately. When calculating AIC we can estimate the noise standard deviation, but because we have been given the noise std is 0.04 we don't need to do this, therefore our degrees of freedom for the Ball, DT, Zeppelin, and Zepplin with Turtuosity models are 5, 7, 6 and 5 respectively. To find the starting position for the DT model I used WLLS one one voxel to find a solution, and used that as the starting position for all voxels.

The rankings for each voxel using AIC or BIC was the same for my results. BIC penalises complex models more than AIC, but in this case it didn't change the rankings. Zeppelin & Stick was top ranked model for all 6 voxels. The balance between complexity and accuracy for the Zeppelin & Stick model is good because the RESNORM is lowest with this model, but the degrees of freedom is not as high as some of the other models, like the DT model.

I am not confident I have found the global minimum for all voxels with each model. There were some voxels which were finding the found minimum very infrequently making me think my starting position and optimise strategy is not optimal.

Q1.3.4

code for this answer related to code lines number 1044 to 1236

I have implemented the Ball and 2 Stick model. The results can be seen in *Fig 15*. I constrained the model in a similar way to the previous models. The only notable difference is how to constrain $f1$ and $f2$. We now have the constraint that $f_1 + f_2 <= 1$, and $0 <= f_2 <= f_1 <= 1$. To do this the model takes $f2$, and r, and we set $f_1 = f_2 + (1 - f_2) * r$. Then we have $f_2, r \in (0, 1)$ and we can use the same constrain techniques we used for the previous models. I used the starting point $x = (S(0,0), d, f_2, r, \theta_1, \phi_1, \theta_2, \phi_2) = (1, 1e{-}5, 0.3, 0.7, 1, 2, 1, 1)$. But the fitting has not worked well as can be seen in the figure.