Coursework 1: Parametric models

This coursework accompanies the first half of COMP0118: Computational Modelling for Biomedical Imaging. It divides into subsections that reflect the main sections of material in the lectures.

In each section there is some core material, which is a fairly structured task for you to complete using the data provided. Each section also contains some further tasks that are more open ended. Each of the three groups of core questions (1.1.1-1.1.5; 1.2.1-1.2.2; 1.3.1-1.3.3) are worth 20 points divided equally among the questions in each group (1.1.1 is worth 4 points, 1.2.1 is worth 10 points, etc). The further questions are all marked out of 10, but weighted to give diminishing returns for attempting multiple further questions; further question scores are ordered from highest scoring to lowest scoring and then weighted by (0.8, 0.8, 0.6, 0.6, 0.4, 0.4, 0.2, 0.2, 0, 0, …). Coursework submissions that fail to provide a good attempt at all core questions will be capped below 70% regardless of performance on the further questions. So: get the core questions right first! Then start the further questions and do as well as you can on a few rather than scattered attempts at all. To get close to full marks on a further question, I expect to see some innovation: show me an interesting result I have not explicitly asked for.

Your final submission should contain three components:

1. A short written report (**max 3 sides**), in pdf format, documenting what you have done;
2. A second pdf document containing figures presenting your results. You can include as many figures as you like, but they must all be numbered and referred to from the report. Each figure must have a short caption explaining what it is.
3. A code listing, which the report should also refer to enabling the marker to find which code corresponds to which experiment or result if necessary. However, don't expect me to look at the code! A well-written report should mean I don't have to and marks are lost every time I feel compelled to check the code.

Components 1 and 2 **must** be submitted as **pdf files,** component 3 as a **zip** file containing **only** your code, via the **Coursework 1 TurnItIn** on the CMBI moodle page.

1.1 Parameter estimation and mapping

This part of the coursework uses a high angular resolution diffusion imaging (HARDI) data set acquired as part of the Human Connectome Project (http://www.humanconnectome.org/). It has 18 b=0 images and 90 diffusion weighted images with different gradient directions and a b-value of 1000 s/mm^2. You can download the data and associated files from this dropbox link: https://www.dropbox.com/s/q6qeump8knlaav2/data_p1.zip?dl=0.

Once you have the data file, you can load it into matlab as follows:

```
load('data');
dwis=double(dwis);
dwis=permute(dwis,[4,1,2,3]);
```

The data array has shape 108x145x174x145, which reflects the image dimensions: each of the 108 image volumes has 145 slices with 145x174 voxels. The voxel dimensions are 1.25x1.25x1.25 mm^3.

Display a single slice to verify it is loaded in correctly:

```
% Middle slice of the 1st image volume, which has b=0
imshow(flipud(squeeze(dwis(1,:,:,72))'), []);

% Middle slice of the 2nd image volume, which has b=1000
imshow(flipud(squeeze(dwis(2,:,:,72))'), []);
```

Take the time to understand what the commands above are doing. We transpose and flip the image slice to view it using the 'radiological' convention: the right side of the brain appears on the left side of the screen.

You will also need the gradient directions stored in the file bvecs, which we can load into matlab like this:

```
qhat = load('bvecs');
```

and the b-value array, which we can construct like this:

```
bvals = 1000*sum(qhat.*qhat);
```

See CMBI Lecture 1 for corresponding code in Python.

Core questions (1.1.1 to 1.1.5; answer all of these)

Now to the real work… The task for this section is to map the parameters of the diffusion tensor model and the ball-and-stick model (see lecture slides) over an image of the human brain. We'll start with the (linear) diffusion tensor model then move on to the (non-linear) ball-and-stick model.

Q1.1.1. Implement linear diffusion tensor estimation (see lecture notes, DTI tutorial video, and below) and use it to map the mean diffusivity, fractional anisotropy, and colour-coded principal direction over one slice of the image.

The diffusion tensor model is

$S(b,\hat{\mathbf{q}}) = S(0,0)\exp\left(-b\hat{\mathbf{q}}^T D\hat{\mathbf{q}}\right)$, where

$$D = \begin{pmatrix} D_{xx} & D_{xy} & D_{xz} \\ D_{xy} & D_{yy} & D_{zy} \\ D_{xz} & D_{zy} & D_{zz} \end{pmatrix}$$

is the diffusion tensor, b is the diffusion weighting and $\hat{\mathbf{q}} = \left(q_x, q_y, q_z\right)^T$ is the gradient direction. Taking logs and expanding we can write the right hand side as **y**.**x**, where **y** = (1 $-bq_x^2$ $-2bq_xq_y$ $-2bq_xq_z$ $-bq_y^2$ $-2bq_yq_z$ $-bq_z^2$) contains all known quantities and **x** = (log $S(0,0)$ D_{xx} D_{xy} D_{xz} D_{yy} D_{yz} D_{zz}) contains unknown parameters that we want to estimate.

Thus to estimate the diffusion tensor, we must solve the matrix equation **A** = Y **x**, where **A** is the vector of log measurements, Y is the design matrix with rows corresponding to **y** above (one row for each measurement), and **x** is the list of unknown parameters as above.

Try this in one voxel first. Extract the set of 108 measurements from one single voxel; we'll take one near the centre of the image volume:

```
Avox = dwis(:,92,65,72);
```

Construct the matrix Y and use it to estimate **x** from the log measurements **A** (= `log(Avox)` above). Then repeat the process for each voxel in slice 72.

Derive a map of mean diffusivity (D_{xx} + D_{yy} + D_{zz})/3 and **fractional anisotropy**. To compute the fractional anisotropy, you will need to construct the diffusion tensor matrix in each voxel

```
D = [[x(2) x(3) x(4)]; [x(3) x(5) x(6)]; [x(4) x(6)
x(7)]];
```

You can then either extract the eigenvalues $\lambda_1, \lambda_2, \lambda_3$ and use the formula

$$\text{FA} = \sqrt{\frac{3}{2}\frac{\sum(\lambda_i - \bar{\lambda})^2}{\sum \lambda_i^2}}, \text{ where } \bar{\lambda} = (\lambda_1 + \lambda_2 + \lambda_3)/3,$$

or compute it more directly via

$$\text{FA} = \sqrt{\frac{1}{2}\left(3 - \frac{1}{\text{trace}(R^2)}\right)}, \text{ where } R = D/\text{trace}(D).$$

Use the FA map to **generate a directionally encoded colour map**.

Q1.1.2. Now we have a feel for basic parameter mapping, let's move on to a more interesting model. We will use the ball-and-stick model (see slides L3_ParameterEstimation) to measure, indirectly, the density of axon fibres in each voxel and thus produce a map of that parameter over the brain.

The steps below help you get started by showing you code that fits the model, in a fairly naïve way, using non-linear least squares in one single voxel. Your task in the rest of this section is to make that fitting procedure more robust and expand it to mapping a whole brain slice.

Before we can run the fitting algorithm we need to implement the ball-and-stick model and an objective function to minimize for finding the best parameter settings. The following function computes the sum of square differences between the measurements `Avox` and the model predictions with parameter settings `x`.

```
function sumRes = BallStickSSD(x, Avox, bvals, qhat)

% Extract the parameters
S0 = x(1);
diff = x(2);
f = x(3);
theta = x(4);
phi = x(5);

% Synthesize the signals according to the model
fibdir = [cos(phi)*sin(theta) sin(phi)*sin(theta)
cos(theta)];

fibdotgrad = sum(qhat.*repmat(fibdir, [length(qhat)
1])');

S = S0*(f*exp(-bvals*diff.*(fibdotgrad.^2)) + (1-f)*exp(-
bvals*diff));

% Compute the sum of square differences
sumRes = sum((Avox - S').^2);
```

Here is code for fitting the ball and stick model in one voxel using the objective function above with comments to explain each step. We'll use the same single voxel we started with in Q1.1.1:

```
Avox = dwis(:,92,65,72);

% Define a starting point for the non-linear fit
startx = [3.5e+00   3e-03   2.5e-01 0 0];

% Define various options for the non-linear fitting
```

```
% algorithm.
h=optimset('MaxFunEvals',20000,...
    'Algorithm','quasi-newton',...
    'TolX',1e-10,...
    'TolFun',1e-10);

% Now run the fitting
[parameter_hat,RESNORM,EXITFLAG,OUTPUT]=fminunc('BallStic
kSSD',startx,h,Avox,bvals,qhat);
```

Run the code above to check that everything is working correctly. After the fitting routine, `fminunc`, completes to see the fitted parameter values, enter:

```
format short e
parameter_hat
```

I find that the following fitted parameter values in the variable `parameter_hat`:

```
parameter_hat =

   3.5202e+03  -4.9785e-06   1.2006e+02   8.8690e-01
1.5567e+00
```

in the order S0, d, f, theta, phi. The variable `RESNORM` contains the value of the objective function for the fitted parameters:

```
RESNORM =

 2.8750e+07
```

However, results may vary among versions of matlab and operating system (I used matlabR2017a on a mac).

Familiarize yourself with the list of options available for `fminunc` via the `h=optimset(…)` command, which the matlab help page for `fminunc` lists. The `'Display'` option is useful. Play around with the options e.g. `'MaxFunEvals','MaxIter','TolX', 'TolFun'` and observe their effect.

Think about the quality of the fit you obtain above. First, visualise the fit to the data by comparing the predictions from the model with your fitted parameters to the actual measurements, as in the lectures. Then, is the final value of RESNORM above the kind of value for the sum of square differences we expect? The standard deviation of the noise on each measurement of this data set is around 200. Given that, what would you expect a typical value of RESNORM to be? Are the parameter values sensible given the quantities they represent?

The python function equivalent to matlab's `fminunc` is called `scipy.optimize.minimize` and it works similarly: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html. If using Python, for the purposes of the exercise above, use

```
startx = np.array([3500, -5e-6, 120, 0, 0])
```

Alternatively, use the matlab values for `startx` and add `method='Nelder-Mead'` in Python's `minimize` function.

Q1.1.3. Adapt the implementation in Q1.1.2 to use the transformation method to allow only physically realistic settings of the parameters: S0 must be positive, d must be positive, and f must be in [0, 1]. For example, to ensure that S0 is always positive, we can write it as `x(1)^2` rather than `x(1)`. Do we need constraints for theta and phi?

Rerun the fitting above to confirm that the fitted parameters are realistic. Notice that you need to use the inverse transformation to maintain the same starting point. Also `parameter_hat` no longer contains the parameters S0, d, f, theta, phi directly: you need to use the same transformation to get the model parameters. How do the model parameters compare to those from the original implementation? How does the new implementation affect `RESNORM`? Why?

Q1.1.4. We need to assess whether we have really found the global minimum of our objective function and thus the best estimate of the model parameters. To do this, repeat the fitting procedure above lots of times from different starting points. To get each starting point, add normally distributed random numbers to the one we used above; the scale of each random number should reflect the scale of each individual parameter to ensure appropriate perturbation sizes.

In what proportion of trials do we find the solution with the smallest value of `RESNORM`? Do you think this is the global minimum? If so, how many runs do we need to be 95% sure we find the global min?

Try some other voxels. How consistent are your answers with the voxel above?

Q1.1.5. Now create parameter maps over one slice of the image volume, say the middle slice `dwis(:,:,:,72)`, by using an appropriate set of constraints and number of runs in each voxel of the slice. Produce maps of S0, d, f and the residual error `RESNORM`. Then produce a map of the fibre direction **n** from the parameters theta and phi. Look up Matlab's `quiver` function; there is a similar function in Python's `matplotlib.pyplot`. Weight **n** by the volume fraction f before plotting and keep in mind that directions **n** and **-n** are equivalent.

Further questions (only answer some of these)

Q1.1.6. This question explores various ways to tweak the non-linear fitting procedure to find the global minimum more quickly and reliably. You don't need to try everything – a solid exploration of some of these ideas (or others) will get the marks. In each case, how do the tweaks affect the likelihood of finding the global minimum in each run and how does that affect overall computation time for mapping parameters over a whole slice or volume (accounting for the fact that the number of runs per voxel to ensure finding the global min may change)? Compare with Q1.1.3 and Q1.1.5.

First, informed starting points can increase the likelihood of finding the global minimum. We can obtain good guesses for the parameters of the ball and stick model from the diffusion tensor model, which we can estimate from a linear fit (much faster). Use the diffusion tensor estimate from Q1.1.1 to define a starting point for the ball and stick fitting routine. Repeat steps 1.1.2 to 1.1.5 above and calculate the difference in computation time required to generate the parameter map. Try a few potential mappings from diffusion tensor estimate to ball and stick starting point. Which maximizes the likelihood of finding the global min?

An alternative way to constrain your parameter estimates to realistic ranges, as opposed to the transformation method in Q1.1.2, is to use constrained optimization via the `fmincon` function. Look up the help page and implement the constraints this way. Python's `minimize` function also has options to add constraints

By default both `fminunc` and `minimize` calculate derivatives of the objective function numerically. You can calculate them analytically instead to add accuracy and stability. Add the derivative of the objective function with respect to the model parameters to your version of `BallStickSSD.m`. Or in Python use the `'jac'` option in `minimize`. Does it help?

Q1.1.7. Above we assume a Gaussian noise model on our measurements by using the sum of squared differences objective function. However, a more accurate model for the noise on MRI voxel intensities is the Rician distribution. Implement an alternative objective function for the Rician noise model. How much does it affect the parameter estimates and maps and their computation time? How about simple approximations to it that you can find in (Jones 2004; Alexander 2009)?

Q1.1.8. An alternative way to fit models to data is via modern machine learning techniques. We can construct a training set from which to learn a regression function by synthesizing sets of measurements from the model for a wide range of plausible parameter combinations and learn the inverse mapping from measurements to parameters. The regression function might be represented as a neural network, a random forest, or even a simple look-up table. See for example Nedjati-Gilani 2017, who use a random forest this way, Nilsson 2010 who use a look-up table, or Golkov 2016, who use CNNs. Try an approach like this for fitting the ball-and-stick model. How does such it compare to the non-linear

optimization approaches above in terms of computation time and residual errors? How much does the range and density of sampling in the training set, as well as the particular choice of regression algorithm, affect the performance? Gyori et al 2021 provide some pointers to investigating the last question.

1.2 Uncertainty estimation

This part continues with the same data as the previous subsection, but extends from estimating a single maximum likelihood estimate to estimating uncertainty and confidence intervals on each parameter.

Core questions (1.2.1 to 1.2.2; answer all of these)

Q1.2.1. Use the classical bootstrap procedure to estimate the 2-sigma range and 95% range for the parameters S0, d and f in the ball and stick model in the single voxel you used in Q1.1.2. Make sure to heed your findings from section 1.1 to make sure you the global minimum for each bootstrap sample. Try some other voxels and compare the estimates.

Q1.2.2. Use MCMC to provide another estimate of the 2-sigma range and the 95% range for the same parameters as in Q1.2.1. How does it compare with the bootstrap output?

Further questions (only answer some of these)

Q1.2.3. Try Laplace's method (look at the help pages of `fminunc` or `fmincon` or `minimize` to see how to return the objective function Hessian), parametric bootstrap, and the various flavours of non-parametric bootstrap to get parameter confidence intervals. Compare the parameter confidence intervals you estimate with those from Q1.2.1 and Q1.2.2. Comment on any differences you observe.

Q1.2.4. Create visualizations of the uncertainty in the various scalar parameter estimates, and the fibre orientation, mapped over a whole brain slice. What might be an equivalent for the orientation parameter to the 95% range on the scalar parameters S0, d and f? Uncertainty in scalar parameters can be visualized in various ways, from simple maps of 95% ranges, to videos show random instances of the image drawn from the posterior distribution at each voxel. For some ideas on how to visualize uncertainty in the orientation parameter, see (Jones 2003).

1.3 Model Selection

We'll switch to a different data set for this part. You can download the files isbi2015_data.txt and isbi2015_protocol.txt from the Moodle website. These

files have been used for an international scientific challenge on white matter modelling at the ISBI 2015 conference; results and discussion were eventually published in (Ferizi 2017), which also describes the data and challenge.

Here is some code that loads the data and independent variables describing the device settings for each measurement into matlab and computes the gradient directions and b-values, etc., that you'll need to fit models in the tasks that follow.

```
% Load the diffusion signal
fid = fopen('isbi2015_data_normalised.txt', 'r', 'b');
fgetl(fid); % Read in the header
D = fscanf(fid, '%f', [6, inf])'; % Read in the data
fclose(fid);

% Select the first of the 6 voxels
meas = D(:,1);

% Load the protocol
fid = fopen('isbi2015_protocol.txt', 'r', 'b');
fgetl(fid);
A = fscanf(fid, '%f', [7, inf]);
fclose(fid);

% Create the protocol
grad_dirs = A(1:3,:);
G = A(4,:);
delta = A(5,:);
smalldel = A(6,:);
TE = A(7,:);

GAMMA = 2.675987E8;
bvals = ((GAMMA*smalldel.*G).^2).*(delta-smalldel/3);
% convert bvals units from s/m^2 to s/mm^2
bvals = bvals/10^6
```

Note that this data set is not an image, but a set of measurements from a single voxel. The total number of measurements per voxel is much larger than in the data set we used for sections 1.1 and 1.2: 3612 rather than 108. That allows us to fit some more interesting models.

Core questions (1.3.1 to 1.3.3; answer all of these)

Q1.3.1. Adapt your code from Q1.1.3 to fit the ball and stick model to this new data. Bear in mind that the best starting point, constraint encoding, and choice of optimization algorithm may be quite different to those you found for the other data set. Report your fitting strategy, the best-fit parameter values, frequency of identifying the global min, and minimum RESNORM. The standard deviation of the noise in this data set is about 0.04; is RESNORM what you expect?

Q1.3.2. Adapt the code further to fit various different models listed below as well as the diffusion tensor model. Each model, of course, will need a different

encoding to impose appropriate constraints, different algorithms to determine a suitable starting point, and will require different numbers of random starting points to ensure finding the global minimum fit. Show the fit to the data and quote the best RESNORM for each.

Two-compartment models all have the general form

$$S(b,\hat{\mathbf{q}}) = S(0,0)\left(fS_I(b,\hat{\mathbf{q}}) + (1-f)S_E(b,\hat{\mathbf{q}})\right),$$

where S_I is the signal from water inside cells (intra-cellular signal) and S_E is the signal from water outside cells (extra-cellular signal). For now, we'll only consider one model for the intra-cellular signal, which is the stick model where

$$S_I(b,\hat{\mathbf{q}}) = \exp\left(-bd(\hat{\mathbf{q}}.\mathbf{n})^2\right),$$

d is the diffusivity and the unit vector **n** is the fibre orientation.

The new models below have models for S_E that differ from the ball and stick model.

Ball and stick model revisited: in the ball and stick model, the extra-cellular signal uses the ball model:

$$S_E(b,\hat{\mathbf{q}}) = \exp\left(-bd\right)$$

in which the signal is independent of orientation.

The *zeppelin and stick model* uses the zeppelin model of the extra-cellular signal. The zeppelin model is a diffusion tensor with rotational symmetry around the principal axis, i.e. it has two equal eigenvalues. We will assume for now that the two smaller eigenvalues are equal, the largest eigenvalue is associated with the fibre direction **n**, and that the diffusivity in the stick compartment is the same as that in the zeppelin along the fibre direction. Thus,

$$S_E(b,\hat{\mathbf{q}}) = \exp\left(-b(\lambda_2 + (\lambda_1 - \lambda_2)(\hat{\mathbf{q}}.\mathbf{n})^2)\right)$$

where λ_1 (=d) >=λ_2>0 are the two unique eigenvalues.

The *zeppelin and stick with tortuosity model* is similar to the zeppelin and stick model above, but assumes that $\lambda_2 = (1-f)\lambda_1$.

Q1.3.3. Use the AIC and BIC to rank the four models. Are the rankings from each criterion consistent? What do you conclude from the ranking(s)?

Further questions (only answer some of these)

Q1.3.4. Extend the range of models to include some more exotic beasts. The set of models in (Panagiotaki 2012) or later work by (Ferizi 2014, 2015) certainly

contains some that should do better than any of those listed in Q1.3.2. Replacing the stick model with a cylinder is one option, although coding the cylinder model is a little fiddly. Another possibility is to include multiple stick compartments to relax the assumption that all the axon fibres in the voxel that produced the data are perfectly parallel. For example the ball-and-two-sticks model is

$$S(b,\hat{\mathbf{q}}) = S(0,0)\left(f_1 S_I(b,\hat{\mathbf{q}};\mathbf{n}_1) + f_2 S_I(b,\hat{\mathbf{q}};\mathbf{n}_2) + (1-f_1-f_2)S_E(b,\hat{\mathbf{q}})\right)$$

where S_E is the ball model and each S_I term has its own fibre orientation. See also the fibre dispersion models in (Zhang 2012), (Sotiropoulos 2012), and (Ferizi 2013, 2014, 2015, 2017).

Prize question! A small prize will be awarded for the person that can construct the model with the smallest AIC on this data. To be eligible, you must do better than my best attempt, which is the ball-and-two-sticks model above. To enter, send me your code for fitting the model, the best set of parameters that you have found, and what RESNORM you get; I'll compute the AIC myself and compare it with that for my implementations of all the models I've discussed. Don't feel you have to be limited to compartment models; any general kind of regression model might also do the job.

Q1.3.5. Is the best model you identify in Q1.3.3 or Q1.3.4 consistent across the other five voxels in the challenge data set?

How about consistency with other model-selection techniques? Compare the results of model-selection via various cross-validation strategies to the information criteria used in Q1.3.3. Another alternative to compare is evaluating the posterior probability via Bayesian model selection to rank the models (see lecture notes and Mackay book referenced therein); what issues arise?

Q1.3.6. Try fitting the simple two compartment X-and-stick models in Q1.3.2 to the imaging data we used in sections 1.1 and 1.2. Use whichever model selection criterion you prefer to select the most appropriate model in each voxel of the image and create a map indicating which model is most appropriate at each location. What do you observe? Does the map show consistency between the left and right sides, which would suggest some anatomical meaning? Can you explain why it selects certain models in certain regions? Does it help to average parameter estimates from different models using the Akaike weights?

1.4 Experiment Design

This part uses with the same data as subsection 1.3 to illustrate the basic principles of experiment design.

Further questions (only answer some of these)

Q1.4.1. Write a function to compute the Fisher information matrix for the ball and stick model. Evaluate the function at the maximum likelihood estimate of the ball and stick parameters that you obtained in **Q1.3.1** for the imaging settings in `isbi2015_protocol.txt`

The imaging protocol divides into 36 "shells", each with 121 measurements. Each shell has different corresponding b-values arising from unique combinations of gradient strength G, pulse width (small delta) and pulse duration (big delta). A shell consists of all measurements with a certain b-value, in addition to all b=0 measurements with the same TE value as the diffusion-weighted measurements.

Evaluate the A-optimality criterion (trace of the inverse Fisher information matrix) for each individual shell to determine which single shell provides the most economical protocol for estimating ball-and-stick parameters. I suggest you ignore the orientation parameters, which simplifies evaluation of the A-optimality, although you can incorporate them if you are careful.

How do results compare if you use, say, T-optimality instead?

Q1.4.2. Repeat the evaluation in Q1.4.1 but now accounting for the loss of signal from T2 decay that differs among shells, which is an exponential function of the echo time TE:

$$S_0 = M_0 \exp\left(-\frac{TE}{T_2}\right).$$

How does accounting for T2 decay affect the optimal choice of shell?

Does parametric bootstrap of the ball-and-stick model support the choice of shell that A-optimality suggests?

Extend further to determine the advantage of using a pair of shells with different non-zero b-value rather than just one shell. For fixed number of measurements, does a combination of non-zero b-values offer any advantage?

References

Alexander (2009). Modelling, Fitting and Sampling in Diffusion MRI. In *Visualisation and processing of tensor fields*, Springer.
Ferizi et al (2013). The importance of being dispersed: A ranking of diffusion MRI models for fibre dispersion using in vivo human brain data. *Proc. MICCAI*.
Ferizi et al (2014). A ranking of diffusion MRI compartment models with in vivo human brain data. *Magnetic Resonance in Medicine*.
Ferizi et al (2015). White matter compartment models for in vivo diffusion MRI at 300mT/m. *Neuroimage*.

Ferizi et al (2017). Diffusion MRI microstructure models with in vivo human brain Connectome data: results from a multi-group comparison. *NMR in Biomedicine*.
Gyori et al (2021). Training data distribution significantly impacts the estimation of tissue microstructure with machine learning. *Magnetic Resonance in Medicine*.
Golkov et al (2016). q-Space Deep Learning: Twelve-Fold Shorter and Model-Free Diffusion MRI Scans. *IEEE Trans. Medical Imaging*.
Jones (2003). Determining and visualizing uncertainty in estimates of fiber orientation from diffusion tensor MRI. *Magnetic Resonance in Medicine*
Jones and Basser (2004). "Squashing peanuts and smashing pumpkins": How noise distorts diffusion-weighted MR data. *Magnetic Resonance in Medicine*
Nedjati-Gilani (2017). Machine learning based compartment models with permeability for white matter microstructure imaging. *Neuroimage*.
Nilsson (2010). Evaluating the accuracy and precision of a two-compartment Kärger model using Monte Carlo simulations. *Journal of magnetic resonance*.
Panagiotaki et al (2012). Compartment models of the diffusion MR signal in brain white matter: a taxonomy and comparison. *Neuroimage*.
Sotiropoulos, Behrens, and Jbabdi (2012). Ball and rackets: Inferring fiber fanning from diffusion-weighted MRI. *Neuroimage*.
Zhang et al (2012). NODDI: practical in vivo neurite orientation dispersion and density imaging of the human brain. *Neuroimage*.