Project Summary

Model Fitting using Microstructure Imaging of Crossing (MIX): DIPY

Google Summer of Code: Python Software Foundation

Mentors: Eleftherios Garyfallidis, Maryam Afzali, Mauro Zucchelli, Rafael Neto Henriques

Student: Shreyas Fadnavis

August, 2018
Indiana University Bloomington

Title
Model Fitting using Microstructure Imaging of Crossing (MIX): DIPY

Abstract

Diffusion MRI measures water diffusion in biological tissue, which can be used to probe its microstructure. The most common model for water diffusion in tissue is the diffusion tensor (DT), which assumes a Gaussian distribution. This assumption of Gaussian diffusion oversimplifies the diffusive behavior of water in complex media, and is known experimentally to break down for relatively large b-values. DT derived indices, such as mean diffusivity or fractional anisotropy, can correlate with major tissue damage, but lack sensitivity and specificity to subtle pathological changes.

Microstructure Imaging of Crossing (MIX) is versatile and thus suitable to a broad range of generic multicompartment models, in particular for brain areas where axonal pathways cross.

Multicompartment models (assess the variability of diffusion in sub-voxel regions) enable the estimation of more specific indices, such as axon diameter, density, orientation, and permeability, and so potentially give much greater insight into tissue architecture and sensitivity to pathology.

Goal of Model Fitting: Identify which model compartments are essential to explain the data and parameters that are potentially estimable from a particular experiment.

As a part of GSoC, I worked a lot with Model Fitting using the Neurite Orientation Dispersion and Density Imaging (NODDI) model with its implementation using the MIX framework.

Achievements and Benchmarks

The MIX framework for Microstructure Imaging is a very novel and advanced technique, however required a higher fitting time. As reported by the MIX paper the MATLAB implementation took 191.8 seconds to fit, whereas, the DIPY implementation that I worked on is significantly faster, taking only 10-14 seconds to fit.

This basically means that the current implementation is ~ 20x faster than the state-of-the-art.
Challenging aspects of the work

My project was particularly challenging as it was geared more towards research based on Mathematical Optimization and Model Fitting which I had not worked on before.

I feel that working with DIPY under PSF was one of the most amazing experiences as I really learnt how to overcome the following challenges and contribute quality code to DIPY:

1. Optimization in Scientific Computing using Approximations and not losing out on model accuracy at the same time.
3. Fitting the Model Parameters in Less time so that it can be used from an analytical standpoint.
4. Understanding and implementing cython modules to remove the Python Bottlenecks.
5. Line-by-line profiling to understand and remove bottlenecks from the code.

As a part of this project, I have worked on:

➢ Implementing the NODDIx model with 2 fiber orientation crossings
➢ Simulating the signal from the NODDIx model
➢ Fitting the Simulated Signal
➢ Speeding up the Model Fitting process and Mathematical Optimizations
➢ Legendre Polynomials and Numerical Methods in Cython for Speedups
➢ Creating Error Functions for the model and rigorous testing for edge cases
Implementing the NODDI model for fitting 2 Fiber Crossings using MIX

The NODDI model consists of normalized signals from intracellular and extracellular compartments. The estimated dMRI signal $\hat{S}$ comprises of the normalized signals from the following three compartments:

$$\hat{S} = (1 - v_{iso})v_{ic} S_{ic}(OD, \theta, \varphi) + (1 - v_{ic}) S_{ec}(d_{\perp}, \theta, \varphi) + v_{iso} S_{iso}$$

where $S_{ic}$ and $S_{ec}$ are the normalized signals from intracellular and extracellular compartments respectively.

Parameters to be estimated: Six $\Rightarrow (v_{ic}, v_{iso}, OD, d_{\perp}, \theta, \varphi)$

Noise: Rician noise needs is added to signal for each substrate for different noise realizations.

To further expand the above model to 2 fiber orientation crossings, the following formulation has been implemented:

$$\hat{S}_{Noddi} = \sum_{n=1}^{2} v_{icn} S_{icn}(OD_n, \theta_n, \varphi_n) + v_{ecn} S_{ecn}(d_{\perp n}, \theta_n, \varphi_n) + v_{iso} S_{dot}$$

Parameters to be estimated: 13 $\Rightarrow$ but for implementation, we use only 11 since some of the parameters are reused.

On simulating the signal and visualizing it with the Shore-3D model of DIPY, we can visualize the fiber orientations as follows:
Implementation of NODDIx using MIX framework of Optimization

The Microstructure Imaging of Crossings is a novel and robust method using a 3 step optimization process. It enables to fit existing biophysical models with improved accuracy by utilizing the Variable Separation Method (VSM) to distinguish parameters that enter in both, linear and non-linear manner, in the model (Methods). The estimation of non-linear parameters is a non-convex problem and is handled first. This is done by using Differential Evolution since it is more effective in approximating exponential time series models.

[Note: the steps to implement the above simulations and visualize the signals have been explained in detail below]
The task to estimate linear parameters amounts to a convex problem and can be solved using standard least squares techniques. These parameter estimates provide a starting point for a Trust Region method in search for a refined solution.

4 Steps involved in Implementing MIX:

**Step 1 - Variable Separation:** The objective function has a separable structure which can be exploited to separate the variables by **Variable Separation (VarPro)** method. We can rewrite our objective function as a projection using the Moore-Penrose Inverse (Pseudoinverse) and get the variable projection functional.

This is a rather advanced and mathematically well formed method which makes use of variable projections to transform the complicated computations between the variables of the NODDIx model into a space where they can be fit individually.

Taking advantage of this special structure of the model, the method of variable projections eliminates the linear variables obtaining a somewhat more complicated function that involves only the nonlinear parameters.

This procedure not only reduces the dimension of the parameter space but also results in a better-conditioned problem. The same optimization method applied to the original and reduced problems will always converge faster for the latter.

Further literature for this method can be found here:


**Step 2 - Stochastic search for non-linear parameters ‘x’:** The objective function is non-convex, particularly of non-linear least-square form. Any gradient based method employed to estimate the parameters will have critical dependence on a good starting point, which is unknown. Alternative approach can be regular grid search, which is time consuming and adds computational burden. This particular type of problem therefore points towards considering stochastic search methods like Differential Evolution (DE). In case of time series analysis, DE can be used efficiently for sum of exponentials functions. DE parameters can be varied for each selected biophysical model and time complexity may change with each choice.
I have written a different blog post for implementation of DE with a detailed explanation of its working and its nature for NODDIx.


**Step 3** - Constrained search for linear parameters ‘f’: After estimating the parameters ‘x’, estimation of linear parameters ‘f’ is a constrained linear least-squares estimation problem. I have made use of the cvxpy optimizer to perform the constrained search to find the f’s of the model compartments.

**Step 4** - Non-Linear Least Squares Estimation (NLLS) using Trust Region Method: Step 2 and step 3 give a reliable initial guess of both ‘x’ and ‘f’ as initial value for Trust Region method. This method has been implemented using the Levenberg Marquardt method to perform (NLLS) fitting from SciPy: Optimize module.

**Testing and Running my code for NODDIx model in DIPY**

Here is the link to the code for NODDIx implemented in DIPY: https://github.com/ShreyasFadnavis/dipy/tree/noddix_speed [1]

Steps to install DIPY:
1. Install Anaconda: https://anaconda.org. Its free: you just need to create an account and you should be good to go!
2. In the search of your Windows, search for: Anaconda Prompt.
   - This should open up a terminal where you will need to operate from. (try avoiding the normal windows command prompt as Anaconda uses a virtual environment to do so.)
3. Install DIPY:
   - conda install vtk
   - You may need to install cvxpy (a convex optimization library) using:
     - conda install -c cvxgrp cvxpy on Windows

Following are the steps you need to take to run the code:
I am not sure how well versed you are with Git, but all you need to do is:
Click on the above link in (1).
Clone the repository:
  ○ A green button will appear on the top right corner of the GitHub web-page, which will give you a link. Copy that link.
  ○ Open the command prompt and run the following 2 commands:
    ■ `git clone <link copied from the above step>`
  ○ This should create a folder in your local file-system which we need to compile using the following command:
    ■ `cd path/to/dipy`
    ■ Once inside the main dipy folder, run the following command to install the repository:
      ■ `pip install --user -e`.

Dabbling with the NODDIx code:

- Anaconda will have already installed an IDE (editor) called Spyder. You can access it from the default Windows search.
- Once inside spyder, click on open file and navigate to: ./dipy/dipy/reconst
- There inside reconst is the file which contains the NODDIx code, namely:
  ○ NODDIx.py (has some code in cython: noddi_speed.pyx)

The sim_noddix.py file is what contains the simulation and fitting of the signal using the model written in NODDIx.py.

All you need to do is hit the **RUN** button in the navbar of the Spyder IDE and it will return:

- Actual Parameters (10)
- Fitted Parameters (10)
- List of Errors in the Estimation (for each param) (10)
- Sum of all errors

To run the simulations and visualize the fiber orientation crossings, please navigate to: ./dipy/dipy/sims

Within sims, you should find a file named **sim_noddix.py** which will simulate the signal and visualize with shore. The code contains the error functions and functions to visualize the data. They are as follows:

- `show_with_shore(gtab, reconst_signal)` -> to visualize the simulated signal
• sim_voxel_fit(reconst_signal) -> to fit the simulation of 1 voxel

Fitting the Real Data With NODDIx Model

Now it’s time to see how the model works on real data. To do so, I am working on an Example using a subject from the HCP (Human Connectome Project) data which is publicly available online.

The whole brain takes really long to fit, however, I have made use of a mask to fit only the region of the brain with the corpus callosum to see how the model works.

Here is the link you can use to test the model with your own data by just replacing the file paths: https://github.com/ShreyasFadnavis/dipy/blob/noddix_speed/doc/examples/example_noddi_hcp.py

Next Steps:

Approximately ~ 3000 lines of code have been written so far for the NODDIx model and still has some work left. It includes the following:

• Creating an Example for the Gallery of DIPY to help new users use the MIX module.
• Extend the 2 crossing model to 1 and 3 crossings to fit neuro-data.