QTI+: MATLAB Implementation
User Guide

Contents

1 Introduction **2**

1.1 Software overview 2

2 Installation **4**

2.1 CVX 4

2.2 QTI+ 4

3 Processing the data **5**

3.1 Input: Data & B-tensor format 5

3.2 qtiplus_fit 5

3.2.1 choosing the pipeline 6

3.2.2 passing a mask 6

3.2.3 choosing the SDP solver 7

3.2.4 excluding measurements from the fit 7

3.2.5 Using the parallel computing toolbox 8

3.2.6 selecting the number of voxels to be fitted simultaneously 8

3.3 Output: model parameters and scalar invariants 9

3.4 Computational times 10

4 Example - synthetic data **11**

5 Example - experimental data **15**

1 Introduction

This document contains the User Guide for the QTI+ software as implemented in Matlab (`https://github.com/DenebBoito/qtiplus`). The document is structured in three sections. The first contains instructions on the necessary modules that need to be installed and how to install them. Then comes the main section describing how to process diffusion MRI data acquired with time-varying magnetic field gradients using this software. The final section contains two examples, one involving synthetic data and one involving experimental data. The example in which synthetic data are used shows how to process data using different options, while the example in which experimental data are used guides through the reproduction of selected results from [1].

This guide assumes that the reader is or will be familiar with the articles describing the QTI framework [2], linked here, and the methods belonging to the "+" version [1], linked here. The focus of this guide lies only on providing an overview of the software functionalities and how to use them. For the terminology, methods, and science, we refer the reader to the articles. Note that in this guide we adopted a "loose" terminology, meaning that we refer to routines with their name spelled with or without parenthesis interchangeably, i.e., SDPdc and SDP(dc) represent the same fitting routine.

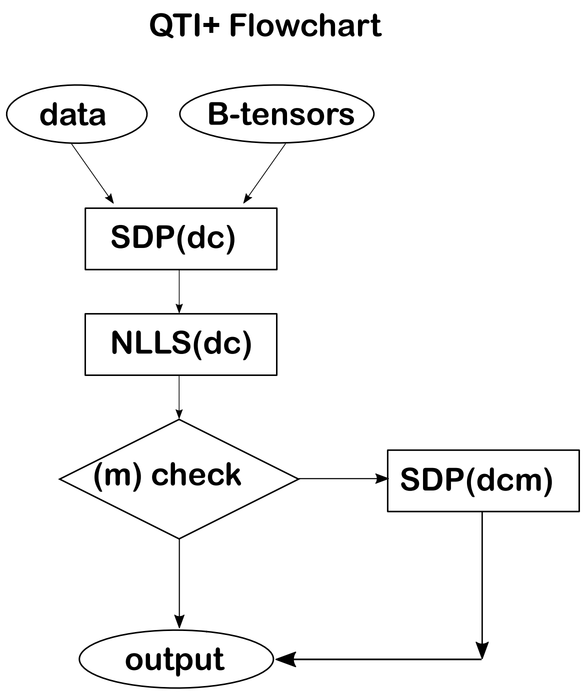
1.1 Software overview

The QTI+ framework and the nomenclature are displayed in Figure 1. The different blocks correspond to the different fitting routines as presented in [1], please refer to that paper for information regarding the various steps.

For the SDPdc, (m) check, and SDPdcm steps, one of the solvers interfaced by CVX is used. CVX is introduced in section 2.1, while information about the solvers are found in section 3.2.3. For solving the nonlinear problem, the Matlab routine *lsqcurvefit* with the *Levenberg-Marquardt* algorithm is used. Note that, since the solution from the SDPdc step is used as starting point for the nonlinear fit NLLSdc, if the residuals after the SDPdc step are smaller than those after the NLLSdc step, the solution produced by SDPdc is retained.

Note that it is not necessary to perform all the steps in the framework; satisfactory results can be obtained just by performing SDPdc. In the software, the user can decide which of the steps to perform by selecting predefined pipelines (By default, only SDPdc is performed). The main output consists of the estimated QTI model parameters. Additionally, the user can obtained the scalar maps derived from these estimates.

The software is really intended to be used through the main function called qtiplus_fit. This is the function that takes the data and the measurement B-tensors as inputs, and returns the estimated model parameters and the scalar invariants derived from these estimates. Details about the expected format for the data and B-tensors are given in section 3.1. Used in the following fashion:



Nomenclature:

SDP(dc): Semidefinite Programming formulation with (d) and (c) constraints

NLLS(dc): Non-linear least square formulation with (d) and (c) constraints

(m) check: Check on condition (m) via Semidefinite Programming

SDP(dcm): Semidefinite Programming formulation with (d), (c), and (m) constraints

Figure 1: QTI+ framework layout and nomenclature.

```
[model, invariants] = qtiplus_fit(data,btensors);
```

the software uses all the default settings, which is possibly what most users would like to do. Details about the different settings can be found in sections 3.2.1, 3.2.2, 3.2.3, 3.2.4, and 3.2.5

2 Installation

2.1 CVX

In order to use the routines in this library, CVX (`http://cvxr.com/cvx/`) needs to be installed [3]. CVX offers an environment for defining and solving convex problems. It is free to download and install, and comes with a few free solvers that can be readily used upon installation. If CVX is not already installed, please visit the download page (`http://cvxr.com/cvx/download/`) and download the option that suits you. Please consult the installation page (`http://cvxr.com/cvx/doc/install.html`) for instruction on how to install CVX on your machine.

Note that in our implementation we favoured Mosek (MOSEK ApS, Denmark) as the solver to be used for solving the problem using Semidefinite Programming (SDP). Even though Mosek can be obtained together with CVX (depending on which bundle is downloaded), it requires a licence in order to be used. Such licence is free for academic purposes, and instructions on how to obtain the licence and where to place it in your machine can be found here `http://cvxr.com/cvx/doc/mosek.html`. If Mosek is not available, the QTI+ routines will use another solver that ships with CVX, SDPT3 [4]. There should be no difference in terms of results when choosing one or the other solver. In our experience, the only relevant difference is found in the computational times, with Mosek being the faster option. In section 3.4 you may find a comparison between Mosek and SDPT3 in terms of computational times.

2.2 QTI+

All the codes for QTI+ are written in Matlab, meaning that it is sufficient to either clone or download the files from the Github page, and add them to your Matlab path to start using it. The function qtiplus_setup may assist in adding all the folders to the path, and should be run before anything else to avoid error messages. This function is also called when running the script containing the example on how to use the software example_qtiplus_fit_synthetic_data.

3 Processing the data

In this section we detail inputs, outputs, and optional parameters that can be set in the software.

3.1 Input: Data & B-tensor format

The data is expected to be a 4D Matlab matrix with shape [nx, ny, nz, nd]. nx, ny, and nz are the number n of voxels for each dimension x, y, and z. The nd different diffusion measurements/volumes are stacked along the 4th dimension. The example dataset provided with the package in the example dataset folder has dimensions [64, 64, 6, 217]. This means, there are 6 xy-plane slices of size 64 × 64, and there are 217 diffusion measurements.

The B-tensors are expected to be shaped as a 2D Matlab matrix with shape [nd, 6], where the number of rows represents the number of collected diffusion volumes, and each row contains one B-tensor in Voigt format. For the example mentioned above, the matrix containing all the B-tensors has size [217, 6]. The following convention for converting a B-tensor from a 3 × 3 matrix representation to its 1 × 6 Voigt representation is adopted:

$$\begin{pmatrix} B_{xx} & B_{xy} & B_{xz} \\ B_{xy} & B_{yy} & B_{yz} \\ B_{xz} & B_{yz} & B_{zz} \end{pmatrix} \longleftrightarrow \begin{bmatrix} B_{xx} & B_{yy} & B_{zz} & \sqrt{2}B_{xy} & \sqrt{2}B_{xz} & \sqrt{2}B_{yz} \end{bmatrix}$$

The functions named convert_* in the helper functions folder may assist the user in switching from one representation to the other.

Please note that the B-tensors are expected to have SI units, i.e. s/m^2!

3.2 qtiplus_fit

This is the main function of the package which is intended as interface between the user and the software. It controls the input, output, and behaviour of the fit via different key-value pairs. Concise information about input, output, and optional settings can be obtained by typing "help qtiplus_fit" in the Matlab command window or by looking at the comments in the script defining the function. In the next sections the available options for controlling the software behaviour via key-value pairs are detailed. For each controllable option, the keyword, the available options, and the default value are highlighted in a box. The accompanying text provides more insights on choosing between the different options.

3.2.1 choosing the pipeline

KEY: 'pipeline'
VALUE OPTIONS:
- 0: SDPdc
- 1: SDPdc $\rightarrow$ NLLSdc
- 2: SDPdc $\rightarrow$ NLLSdc $\rightarrow$ (m)-check $\rightarrow$ SDPdcm
- 3: SDPdc $\rightarrow$ NLLSdc $\rightarrow$ SDPdcm
- 4: SDPdc $\rightarrow$ (m)check $\rightarrow$ SDPdcm
- 5: SDPdc $\rightarrow$ SDPdcm

DEFAULT: 0

The keyword 'pipeline' allows the user to select which of the steps in the QTI+ framework to perform. As mentioned in the introduction, it is not necessary to perform all of the steps in the framework as satisfactory results can be obtained employing SDPdc only. This saves computational time and is therefore the default options. For a comparison between computational times using different options, the user is referred to [1] and to section 3.4.

The choice between different pipelines is made by passing an integer after the keyword. For example, to analyse the data using all the steps in the QTI+ framework, which corresponds to option 2 as shown in the box, the qtiplus_fit function can thus be called in the following way:

```
[model, invariants] = qtiplus_fit(data,btensors, 'pipeline', 2);
```

3.2.2 passing a mask

KEY: 'mask'
VALUE OPTION: 3D binary image volume
DEFAULT: if none is input, one is computed using the median_otsu method implemented similarly to that in Dipy [5].

The purpose of passing a mask is to avoid computations in parts of the image that do not belong to the object, or to perfom computations only on selected regions of interest. For example, in neuroimaging it is quite common to do computations only on voxels containing brain tissue. The user has here the choice to either pass a mask to the qtiplus_fit function, or let the software compute one using the median_otsu method implemented similarly to that in Dipy [5]. The following call shows how to pass a mask to the fitting function. The variable "mymask" contains the 3D matrix with binary (or logical) values, and has the same [nx, ny, nz] size as the input data.

```
[model, invariants] = qtiplus_fit(data,btensors, 'mask', mymask);
```

Note: if it is desired to compute the model parameters on all parts of the volume, it is recommended that the corresponding mask is input to the qtiplus_fit function. This would correspond to the following call:

```
mymask = ones(nx, ny, nz);
[model, invariants] = qtiplus_fit(data,btensors, 'mask', mymask);
```

Note: The implemented method for automatically computing a mask works fine on the datasets we have tried it on, but it is not guaranteed that it will work fine on all possible datasets. The user should therefore always check that the automatically computed mask is ok. This can be verified by checking the output of the simple_mask function located in the helper functions. For the data that will be analysed, the following call will return the mask that would be automatically computed if none is passed to the qtiplus_fit:

```
mymask = simple_mask(data);
```

3.2.3 choosing the SDP solver

KEY: 'solver'
VALUE OPTIONS:
    'mosek'
    'sdpt3'
DEFAULT: 'mosek' if available, otherwise revert to 'sdpt3'

This key-value pair determines which of the solvers to use for solving the SDP problem. If no option is input, the function qtiplus_fit checks whether Mosek is available, and if so, it uses it. If Mosek is not available, the software reverts to SDPT3, which is installed together with CVX. Mosek is generally recommended compared to other solvers for convex programming, and it is the solver that was used to produce the results in [1]. In terms of results, there should not be any difference between Mosek and SDPT3. In terms of speed, Mosek is in our experience the faster option. In section 3.4 there is a comparison between the computational times achievable with both Mosek and SDPT3 on the example dataset provided with this software package.

3.2.4 excluding measurements from the fit

KEY: 'ind'
VALUE OPTION: binary (or logical) array of size [nd,1] indicating which measurements to consider for the fitting.
DEFAULT: all measurement points are used

By using this option the user can select to exclude some of the collected data from being part of the fitting. For example, in diffusion MRI, when applying a model to some data, it is quite common to exclude the measurements acquired without diffusion encoding gradients applied. Considering the example dataset provided with the software package, the first 13 measurements were acquired without diffusion weighting. Therefore we can choose to exclude them from the fitting. This is done in the following way: first, we define an array with zeros on the indices of the measurements that we want to exclude, and ones on the indices of the measurements that we want to keep. In our example, this corresponds to an array with zeros in the first 13 positions, and ones everywhere else:

```
indx = ones(217,1);
indx(1:13) = 0;
```

Then, we pass this array to the qtiplus_fit function with the keyword 'ind'.

```
[model, invariants] = qtiplus_fit(data, btensors, 'ind', indx);
```

3.2.5 Using the parallel computing toolbox

KEY: 'parallel'
VALUE OPTIONS:
  0: single worker computation
  1: computation distributed over multiple workers
DEFAULT: 1

The QTI+ fitting routines can take advantage of computations being independent between voxels, thus meaning that they can be performed in parallel. In Matlab, distributing the operations over multiple workers requires the *Parallel Computing Toolbox*. If the option for performing parallel computations is selected, the software first check whether this toolbox is available, and if so exploits as many workers as per Matlab's settings. If the user prefer instead to not perform computations in parallel, this needs to be enforced in the following way:

```
[model, invariants] = qtiplus_fit(data, btensors, 'parallel', 0);
```

3.2.6 selecting the number of voxels to be fitted simultaneously

KEY: 'nvox'
VALUE OPTIONS: integer indicating how many voxels to be fitted simultaneously.
DEFAULT: 50

As there is no dependence between the fits performed in each voxel, several can be fitted independently simultaneously. This parameter thus represents the number of voxels that get fitted at the same time. This applies to SDPdc, (m)-check, and SDPdcm. The step involving the nonlinear fit, NLLSdc, can only be applied on one voxel at a time.

Note the difference between this and the performing parallel computations: performing the computations on multiple workers means that each worker gets to solve one problem. With the *nvox* parameter we decide instead over how many voxels one problem is defined. Thus, for example, if 4 workers are available, and we select $nvox = 50$, 200 voxels are being fitted simultaneously.

Performing the fit with nvox > 1 provides a great speed up in computations, since the major bottleneck for SDPdc, (m) check, and SDPdcm is the repeated preconditioning done by the solvers before solving the given problem. By constructing a problem over several voxels, we avoid having to do the preconditioning for each of them. Note however that there is a trade-off between fitting many voxels at a time, and speeding up the computations. Trying to fit too many will eventually slow the computations, as larger problems take more time to built within the solvers. In our experience, we find that 50 voxels is about the right amount, but this number could exhibit some dependence on the specs of the computer on which computations are performed.

3.3 Output: model parameters and scalar invariants

The following call to the software produces the two outputs “model" and “invariants"":

```
[model, invariants] = qtiplus_fit(data, btensors);
```

The “model" variable constitutes the main output of the software, and contains the estimated QTI model parameters. These are returned in a 4D Matlab matrix of size [nx, ny, nz, 28] where the arrangement is as follows:

model[x, y, z, 1] = S_0, the estimated non diffusion-weighted signal, in arbitrary units

model[x, y, z, 2:7] = the 6 unique elements of the mean diffusion tensor, in SI units $[m^2/s]$

model[x, y, z, 8:28] = the 21 unique elements of the covariance tensor, in SI units $[m^4/s^2]$

with x, y, and z being the indices of a generic voxel in the volume. The functions in the helper functions folder can help in switching between different tensor representations.
The second output, “invariants", consists of a structure containing the scalar maps derived from the estimated model parameters. For details about the invariants and their nomenclature please refer to [2]. Figure 2 shows the content of the “invariants" structure. There, S_0 is returned in arbitrary units, while the Mean Diffusivity MD is returned in $[\mu m^2/ms]$.

When the selected pipeline includes multiple steps of the QTI+ framework, i.e., pipelines 2, 3, 4, and 5, the qtiplus_fit function can also return the model parameters computed at intermediate stages of the framework. For example, if pipeline 4, which performs SDPdc, NLLSdc, and SDPdcm, is selected, the function's first output will be the model parameters computed with SDPdcm, and the second output will be the scalar invariants computed on these model parameters. The model parameters computed with SDPdc and NLLSdc can also be obtained by adding an additional output variables to the function call:

```
[model, invariants, model_SDPdc, model_NLLsdc] = qtiplus_fit(data, ...
   btens, 'pipeline',3)
```

The function compute_invariants can then be used to computed the scalar invariants on model parameters obtained in intermediate steps. For example, following the call in the previous snippet, the invariants for the SDPdc fit can be computed in the following way:

```
invariants_SDPdc = compute_invariants(model_SDPdc);
```

3.4 Computational times

An extensive review of the computational times for the different steps of the software and different datasets is provided in [1]. In table 1 we provide exemplary computational times based on the example dataset provided with the software package. The size of the dataset is [64, 64, 6], for a total of 24576 voxels. The machine on which the times were recorded features a 6-core Intel Xeon W-2133 CPU and Matlab R2020b. Computations were performed over 6 workers using the Parallel Computing Toolbox. *nvox* was set to 50.

Table 1: Run times for Mosek and SDPT3

Solver	Pipeline	Run Times
Mosek	SDP(dc)	5 min
Mosek	SDPdc and NLLSdc	11 min
Mosek	SDPdc, NLLSdc, (m) check, and SDPdcm	14 min
Mosek	SDPdc, NLLSdc, and SDPdcm	20 min
Mosek	SDPdc, (m) check, and SDPdcm	8 min
Mosek	SDPdc and SDPdcm	13 min
SDPT3	SDPdc	18 min
SDPT3	SDPdc and NLLSdc	23 min
SDPT3	SDPdc, NLLSdc, (m) check and SDPdcm	26 min
SDPT3	SDPdc, NLLSdc, and SDPdcm	41 min
SDPT3	SDPdc, (m) check, and SDPdcm	20 min
SDPT3	SDPdc and SDPdcm	35 min

4 Example - synthetic data

In this section we step-by-step go through an example based on synthetic data to show how to use the software and the different functionalities. The content of this section is also available in the script example_qtiplus_fit_synthetic_data.

We start by loading the provided dataset and experimental parameters. The dataset here has size [64, 64, 6, 217].

```
data = niftiread('example_dataset.nii');
load('bten');
```

This dataset is constructed as follows: for the B-tensors contained in the variable "bten", a diffusion MR signal is generated using a non-central Wishart distribution [6] with known mean and covariance. As in [1], two signals are generated, one for a diffusion tensor distribution (DTD) with isotropic mean diffusion tensor, and one for a DTD with anisotropic mean diffusion tensor. Rician distributed signal is then obtained by adding Gaussian noise to the real and imaginary parts of the noiseless signal, creating 4096 signal values at SNR $= 1/\sigma = 30$ and SNR$= 15$ for both cases, where σ is the standard deviation of the underlying Gaussian distribution. See [1] for details.

The data are then organized in a [64, 64, 6, 217] matrix, where each of the 6 slices represents the following:

1. Noiseless signals for a non-central Wishart distribution with isotropic mean diffusion tensor
2. Signals from 1. with SNR $= 30$
3. Signals from 1. with SNR $= 15$
4. Noiseless signals for a non-central Wishart distribution with anisotropic mean diffusion tensor
5. Signals from 4. with SNR $= 30$
6. Signals from 4. with SNR $= 15$

Note that while this dataset is hardwired, we provide a function called create_new_example_dataset in the example dataset, with which the user can select the noise level to be applied to slice pairs 2,5 and 3,6. For example, we could create a new dataset where the signals in slices 2 and 5 have SNR $= 40$, and signals from slices 3 and 6 have SNR $= 10$. This is done with the following call:

```
data = create_new_example_dataset(bten, 40, 10);
```

Next, we create a mask to be passed to the qtiplus_fit function. While a mask in a real-life case scenario would have the purpose of limiting the computations to the areas of interest, here we use it with the sole purpose of reducing the number of voxels over which computations are performed thus speeding up the fitting on the example dataset. First, we define a variable having the same size as one of the diffusion volumes in our dataset (which in this case is [64, 64, 6]):

```
mask = zeros(64, 64, 6);
```

Then, we select an inner region of size [11, 11, 6] over which computations are to be performed:

```
range = 5;
mask(end/2-range:end/2+range,end/2-range:end/2+range,:) = 1;
```

Next, we decide to exclude some of the collected data from the fitting. This could be useful, for example, for excluding outliers present in experimental data. In our case, we want to exclude the first 13 measurements in the dataset as these were acquired without diffusion weighting. To do so, we define a binary array indicating which of the 217 measurements in this dataset to keep, and which to discard:

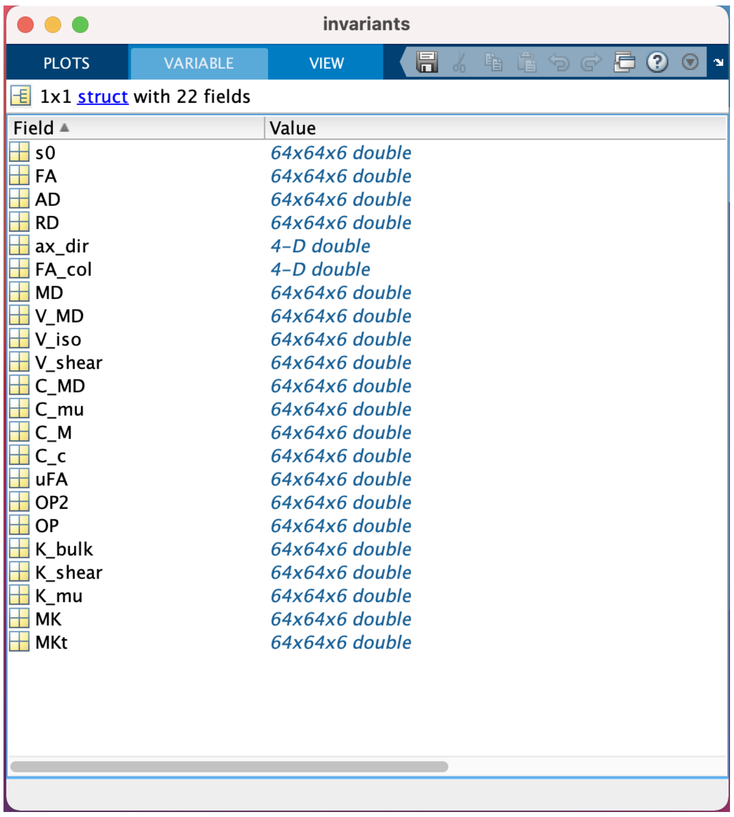
```
indx = ones(217, 1);
indx(1:13) = 0;
```

Now we can call the qtiplus_fit function to perform the fit, and pass the mask and the indices using the respective keywords. Additionally, we specify that we want to perform all the steps in the QTI+ framework (done through the keyword *pipeline*), and that we want the fit to be performed over 70 voxels at a time (achieved through the keyword *nvox*). Note that since we are not specifying whether we want to use parallel computations, the software will use the default behaviour as explained in section 3.2.5. Moreover, we are also not specifying which solver to use for the SDP problems, in which case the software will proceed with Mosek if available, and revert to SDPT3 in case it is not. The call to the function will therefore be as follow:

```
[model, invariants] = qtiplus_fit(data, bten, ...
                                  'mask', mask, ...
                                  'ind',  ind, ...
                                  'pipeline', 2,...
                                  'nvox', 70);
```

where we assigned the outputs to the variables "model" and "invariants". In Figure 2 the content of the invariants structure is shown. It contains the scalar maps derived from the model parameters.

Using the code provided within the example_qtiplus_fit_synthetic_data, we can plot one of the scalar maps (the microscopic fractional anisotropy μFA), and look at its value distributions for the different SNRs. This is shown in Figure 3.



1x1 struct with 22 fields

Field	Value
s0	64x64x6 double
FA	64x64x6 double
AD	64x64x6 double
RD	64x64x6 double
ax_dir	4-D double
FA_col	4-D double
MD	64x64x6 double
V_MD	64x64x6 double
V_iso	64x64x6 double
V_shear	64x64x6 double
C_MD	64x64x6 double
C_mu	64x64x6 double
C_M	64x64x6 double
C_c	64x64x6 double
uFA	64x64x6 double
OP2	64x64x6 double
OP	64x64x6 double
K_bulk	64x64x6 double
K_shear	64x64x6 double
K_mu	64x64x6 double
MK	64x64x6 double
MKt	64x64x6 double

Figure 2: Content of the "invariants" structure as produced by the qtiplus_fit function.For the nomenclature please consult [2].

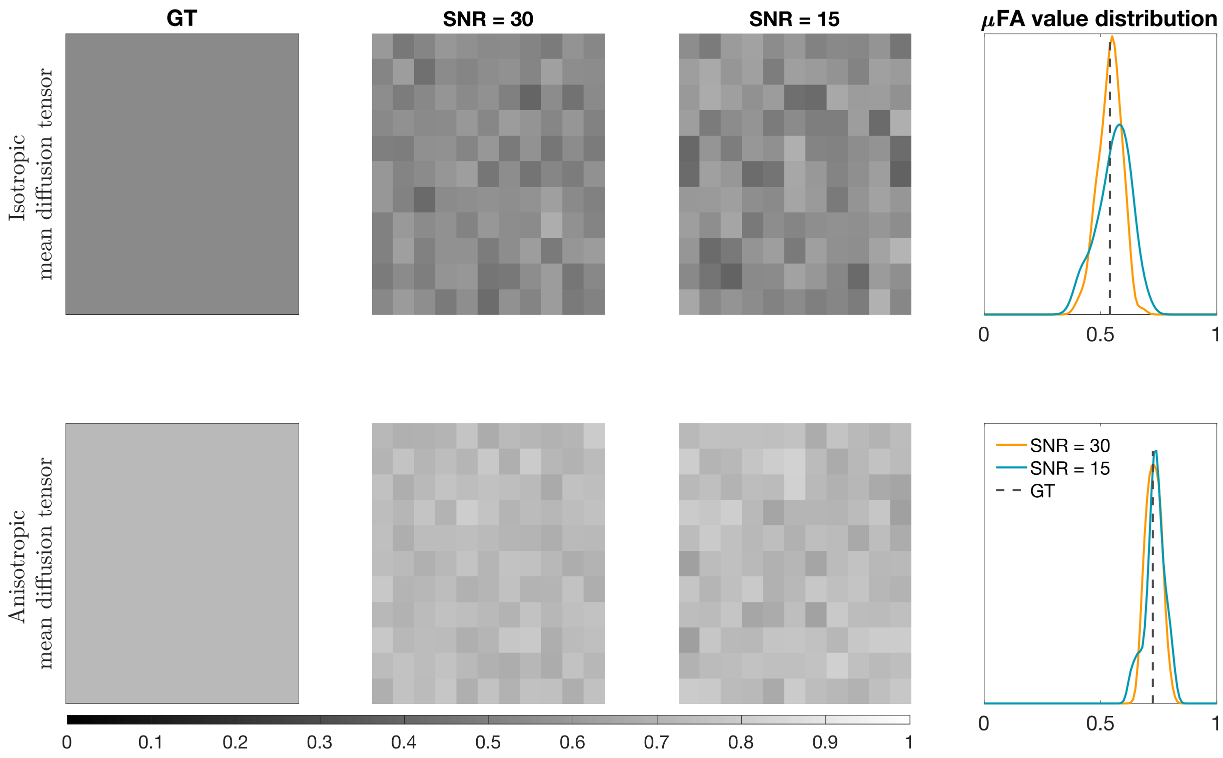


Figure 3: Example maps (shown here is the μFA) obtainable with the software. In the first column, the model is fit to the noiseless signal (slices 1 and 4 in the *example_dataset*), thus the produced values can be considered as Ground Truth (GT). The two middle columns shows the μFA values produced when fitting the model to noisy data at SNR = 30 and SNR = 15, respectively. The rightmost column shows the μFA values distribution for the SNR = 30 and SNR = 15 cases with respect to the ground truth.

5 Example - experimental data

In this section we provide instructions for reproducing part of the results presented in Herberthson et al. [1]. The data for this example, described in [7], have to be autonomously downloaded by the user, and can be found at the following link `https://github.com/filip-szczepankiewicz/Szczepankiewicz_DIB_2019/tree/master/DATA/brain/MD-dMRI`. The content of this section is also available in the script example_qtiplus_fit_experimental_data. What is needed for this example are the data contained in the MD-dMRi.zip file, and the experimental parameters contained in the BRAIN_FWF_MERGED_mc_xps.mat files. Download and unzip these files, and add them to the Matlab path. It might be the case that the native unzipping software available in your machine will not be able extract the files. In this case, please try another tool, the data are accessible. A successful unzipping might still produce two different outputs, one in which the nifti files are compressed (*BRAIN_FWF_MERGED_mc.nii.gz*), and one in which the nifti files have been uncompressed (*BRAIN_FWF_MERGED_mc.nii*). Please choose the option that suits your unzipped files in the snippet below and in the matlab script.

Then, if not already, add the qtiplus folder to your Matlab path (to do this it is also possible to use the script qtiplus_setup located inside the qtiplus folder).

The data and experimental parameters can then be loaded into Matlab:

```
data = niftiread('BRAIN_FWF_MERGED_mc.nii.gz');
% data = niftiread('BRAIN_FWF_MERGED_mc.nii');
load('BRAIN_FWF_MERGED_mc_xps.mat')
```

The B-tensors are then available and can be extracted from the *xps* structure.

```
btens = xps.bt;
```

In Herberthson et al. [1], 4 datasets were created by removing volumes from the full 'BRAIN_FWF_MERGED_mc' dataset. These were indicated using p217, p81, p56, and *p39* where 217, 81, 56, and 39 volumes out of the total 377 were retained, respectively. In the example dataset folder there is a folder named indices where mat files containing the indices of the volumes to be kept for each dataset are stored. These can be used to tell the qtiplus_fit function which volumes to consider for the model fitting. In the same folder, there is also a mat file containing indices to sort the data according to b-values, such that volumes acquired with linear, planar, and spherical tensor encoding are organized by b-shell.

```
% first we order the data according to b-value and tensor-encoding shape
% for each b-shell, the data will be in the order LTE, PTE, STE.
load('indx_order')
data = data(:,:,:,indx_order);
btens = btens(indx_order,:);
```

```
% load the indices
% p217
load('indx_p217')
% p81
load('indx_p81')
% p56
load('indx_p56')
% p39
load('indx_p39')
```

At this point a mask can be created, and later passed to the qtiplus_fit function. To keep the computational time reasonable, in this example we use the mask to also limit the computation to only one axial slice (on the workstation specified in section 3.4, using parallel computation and fitting 50 voxels simultaneously, this example script takes approximately 15 minutes to complete if Mosek is used, and about 50 minutes if SDPT3 is used).

```
mask = simple_mask(data);
% fit only slice 13.
mask(:,:,1:12) = 0;
mask(:,:,14:end) = 0;
```

To reproduce some of the results presented in [1], we fit the model to the data and specify which volumes to consider for the fitting, and which steps of the QTI+ framework to perform. In the following code we show how to compute the model parameters and invariants on the p217, p81, p56, and p39 datasets using the full QTI+ framework (SDPdc → NLLSdc → m-check → SDPdcm). For the p56 case, we also let the function qtiplus_fit output the model parameters computed at intermediate steps (SDPdc and NLLSdc), to compare the scalar invariants obtained with increased fitting refinement. We use the keyword *pipeline* to specify the QTI+ steps to perform, and the keyword *ind* to specify which measurements to consider for the fitting. The pipeline to perform all the steps of the QTI+ framework is number 2. The indices contained in the indx_p217, indx_p81, indx_p56, and indx_p39 variables can be used to perform the fit only on selected measurements recreating the reduced datasets used in [1].

```
[model_p217, invariants_p217] = qtiplus_fit(data, btens,...
                                             'ind', indx_p217,...
                                             'mask', mask, ...
                                             'pipeline', 2);

[model_p81, invariants_p81] = qtiplus_fit(data, btens,...
                                           'ind', indx_p81,...
                                           'mask', mask, ...
                                           'pipeline', 2);

[model_p56, invariants_p56, model_SDPdc_p56, model_NLLSdc_p56] = ...
                          qtiplus_fit(data, btens, 'ind', indx_p56,...
                                                   'mask', mask, ...
                                                   'pipeline', 2);
```

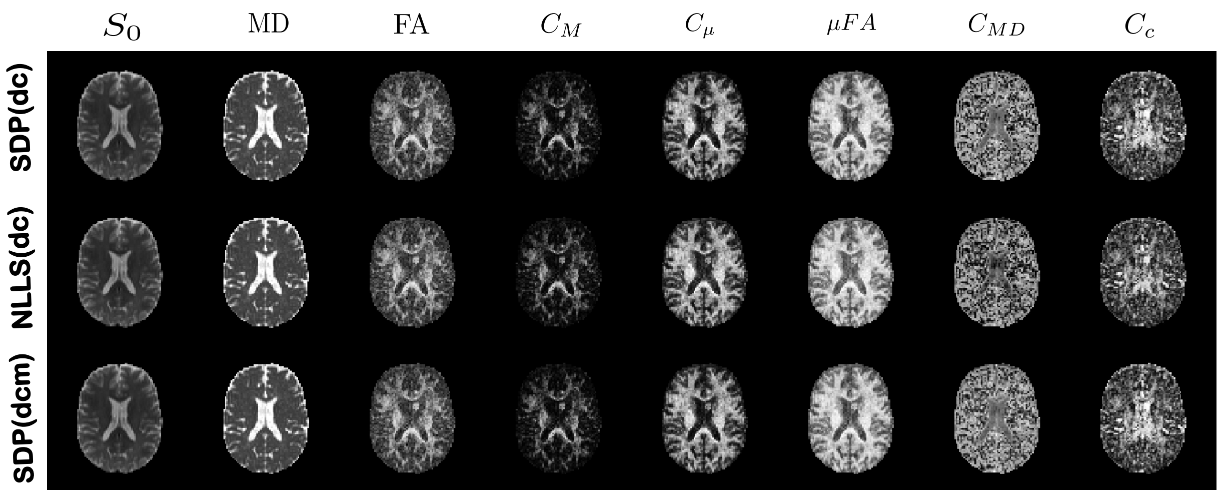


Figure 4: Reproduction of the last three rows of Figure 6 in [1].

```
[model_p39, invariants_p39] = qtiplus_fit(data, btens, ...
                                           'ind', indx_p39,...
                                           'mask', mask, ...
                                           'pipeline', 2);
```

With these results computed, parts of some figures in [1] can be mimicked. For example, Figure 6 in [1] shows a comparison between different routines on p56. In Figure 4 we show the results obtained with the QTI+ routines, which correspond to the last three rows of said Figure 6 in [1]. Note that we added the labels editing the figure outside Matlab, so these will not appear in the figures produced with the provided script.

```
% figure 6, last three rows
invariants_p56_SDPdc = compute_invariants(model_SDPdc_p56);
invariants_p56_NLLSdc = compute_invariants(model_NLLSdc_p56);
invs_fig6{1} = invariants_p56_SDPdc;
invs_fig6{2} = invariants_p56_NLLSdc;
invs_fig6{3} = invariants_p56;
plot_qti_invariants(invs_fig6);
```

With the computed results we can also reproduce the first four rows of Figure 7b in [1], where a comparison between the scalar invariants derived from the model parameters computed on the different reduced datasets is shown. This is shown here in Figure 5.

```
% figure 7b, first 4 rows
invs_fig7b{1} = invariants_p217;
invs_fig7b{2} = invariants_p81;
invs_fig7b{3} = invariants_p56;
invs_fig7b{4} = invariants_p39;
plot_qti_invariants(invs_fig7b);
```

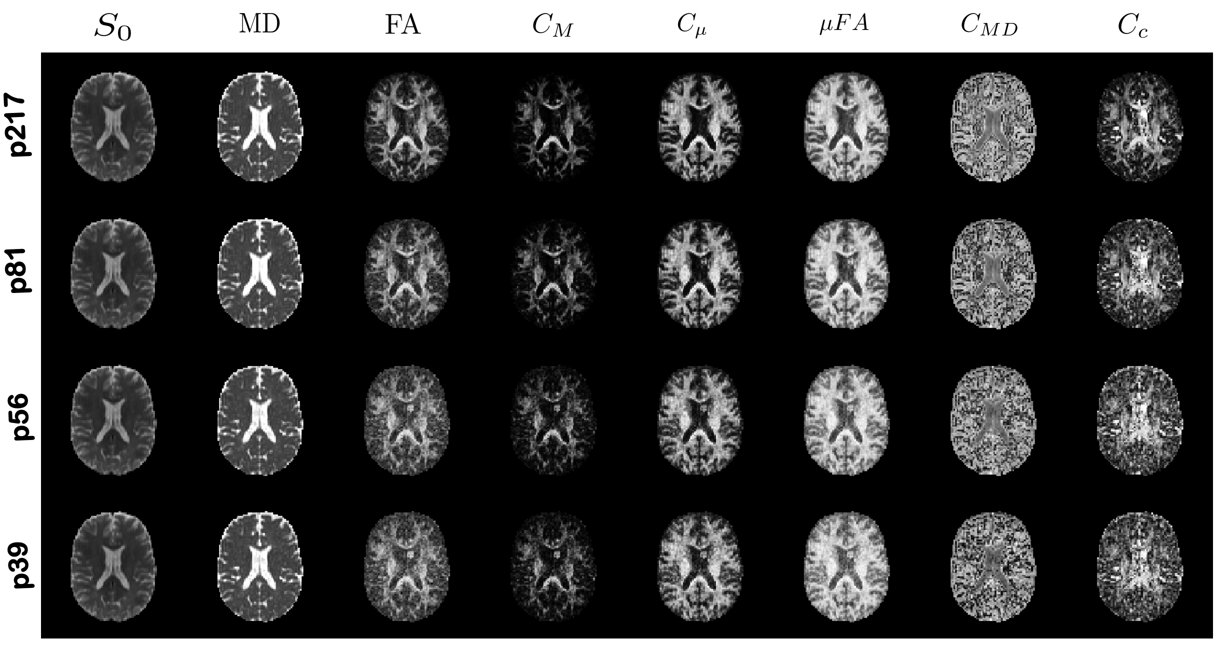


Figure 5: Reproduction of the first three rows of Figure 7b in [1].

Finally, we can show comparisons between specific scalar maps computed from the model parameters fitted on the different reduced datasets. In Figures 6 and 7 we show the fractional anisotropy (FA) and microscopic fractional anisotropy (μFA) for p217, p81, p56, and p39. These two figures correspond to the second rows of Figures 8a and 9a in [1].

```
% figure 8a, second row
plot_qti_invariant(invs_fig7b, "FA", [0 1])

% figure 9a, second row
plot_qti_invariant(invs_fig7b, "uFA", [0 1])
```

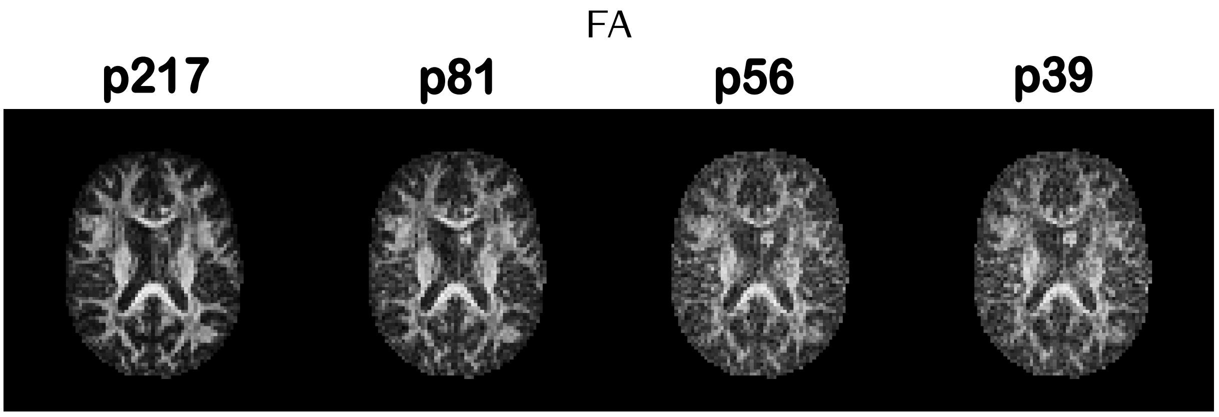


Figure 6: Reproduction of the second row of Figure 8a in [1].

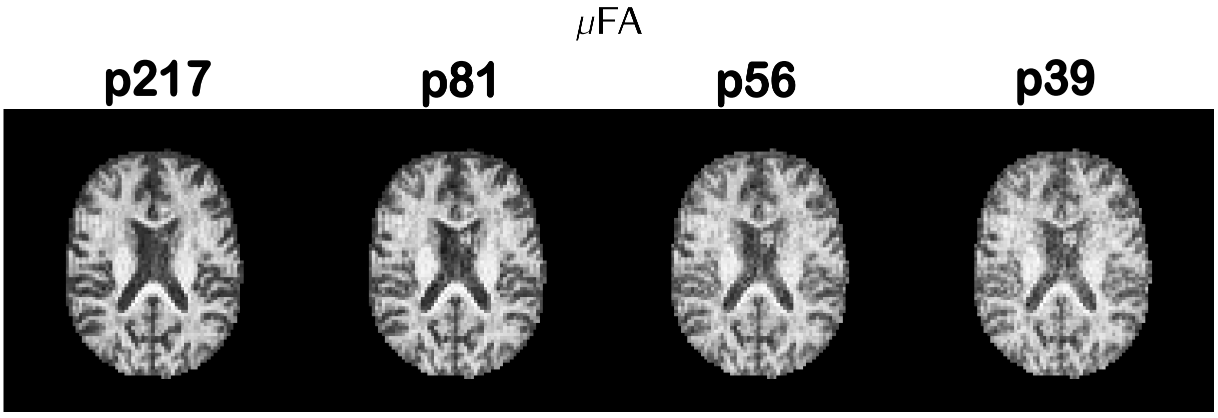


Figure 7: Reproduction of the second row of Figure 9a in [1].

References

[1] Magnus Herberthson, Deneb Boito, Tom Dela Haije, Aasa Feragen, Carl-Fredrik Westin, and Evren Özarslan. Q-space trajectory imaging with positivity constraints (QTI+). *NeuroImage*, 238:118198, 2021.

[2] Carl Fredrik Westin, Hans Knutsson, Ofer Pasternak, Filip Szczepankiewicz, Evren Özarslan, Danielle van Westen, Cecilia Mattisson, Mats Bogren, Lauren J O'Donnell, Marek Kubicki, Daniel Topgaard, and Markus Nilsson. Q-space trajectory imaging for multidimensional diffusion MRI of the human brain. *NeuroImage*, 135:345–62, Jul 2016.

[3] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1. `http://cvxr.com/cvx`, mar 2014.

[4] K. C. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3 — a Matlab software package for semidefinite programming, version 1.3. *Optimization Methods and Software*, 11(1-4):545–581, 1999.

[5] Eleftherios Garyfallidis, Matthew Brett, Bagrat Amirbekian, Ariel Rokem, Stefan Van Der Walt, Maxime Descoteaux, and Ian Nimmo-Smith. Dipy, a library for the analysis of diffusion mri data. *Frontiers in Neuroinformatics*, 8:8, 2014.

[6] S Shakya, N Batool, E Özarslan, and H Knutsson. Multi-fiber reconstruction using probabilistic mixture models for diffusion MRI examinations of the brain. In T Schultz, E Özarslan, and I Hotz, editors, *Modeling, Analysis, and Visualization of Anisotropy*, pages 283–308. Springer International Publishing, Cham, 2017.

[7] Filip Szczepankiewicz, Scott Hoge, and Carl-Fredrik Westin. Linear, planar and spherical tensor-valued diffusion MRI data by free waveform encoding in healthy brain, water, oil and liquid crystals. *Data Brief*, 25:104208, Aug 2019.