BQ-Minded Summer School: Python & model-fitting

Non-linear fitting (e.g. least-squares) of a model to a series of (noise corrupted) data points using `Python`. Each group of students receives a different data set (in separate `group_`n`.csv` files, with $n = 1, 2, 3, \ldots$) that contains relaxometry data. The model should be fitted to this data to obtain estimates of quantitative parameters.

Exercise outline:

Let us recall here the signal equation of the gradient-echo inversion recovery (GRE-IR) sequence $\{\theta_1 - \mathrm{TI} - \theta_2 - (\mathrm{TR} - \mathrm{TI})\}$, where θ_1 and θ_2 are the RF pulses, typically prescribed as 180° and 90°, respectively. TI is the inversion time; and TR is the repetition time. GRE-IR is regarded as the gold standard sequence for quantitative T1-mapping. If we assume instantaneous pulses, perfect spoiling of M_{xy} after θ_1, and no off-resonance effects, then the magnetization for the GRE-IR is given by [1]:

$$M_z = M_0 \frac{1 - \cos(\theta_1) e^{-\frac{\mathrm{TR}}{T_1}} - [1 - \cos(\theta_1)]\, e^{-\frac{\mathrm{TI}}{T_1}}}{1 - \cos(\theta_1)\cos(\theta_2) e^{-\frac{\mathrm{TR}}{T_1}}} \tag{1}$$

1. You are given magnitude (!!) GRE-IR data of a single voxel sampled at different inversion times TI_n. Try to read this data from file, and analyze the data graphically. Check whether your data is indeed magnitude data. How can you see this?

2. Simplify the model in (1) using the following assumptions:

   - Assume $\mathrm{TR} \gg \mathrm{T}_1$
   - Assume ideal flipping angles of $\theta_1 = 180°$ (inversion flip angle) and $\theta_2 = 90°$ (angle that flips magnetization vector in transverse plane).

   How do you incorporate in the model the fact that we consider magnitude data only? How many model parameters (i.e. unknown parameters to be estimated) do you have?

3. Write the corresponding model function.

4. In this step we will perform the Least-Squares optimization. For this we need to:

   - Write the objective function (a.k.a. cost function) that should be minimized.
   - Declare some starting point for the optimizer.
   - Declare the optimizer algorithm/method/... (Levenberg-Marquardt?)
   - Declare some convergence criteria.
   - ...

5. Visualize the results of the model fit by overlaying them with the original data set (i.e. for chosen finite stepsize generate a time series $[0, \mathrm{TI}_{\max}]$ from the estimated model parameters).

6. In the original simulation of the single-voxel data, the (literature) ground truth T_1 value for white matter was equal to 838ms while the M_0 parameter was chosen to be 0.77 [2]. How do your own estimation results compare to these ground truth values? Can you improve on these results by changing some of the optimizer settings (tolerance criteria, etc.)?

References:

[1] Barral JK, et al. A robust methodology for in vivo T1 mapping. *Magn. Reson. Med.*, 2010. **64**(4):1057–1067.

[2] Bojorquez JZ, Bricq S, Acquitter C, Brunotte F, Walker PM, Lalande A. What are normal relaxation times of tissues at 3 T? *Magn. Reson. Imaging*, 2017. **35**:69–80.

Additional info/answers:

1. Visualizing the data should result in a graph similar to figure 1.

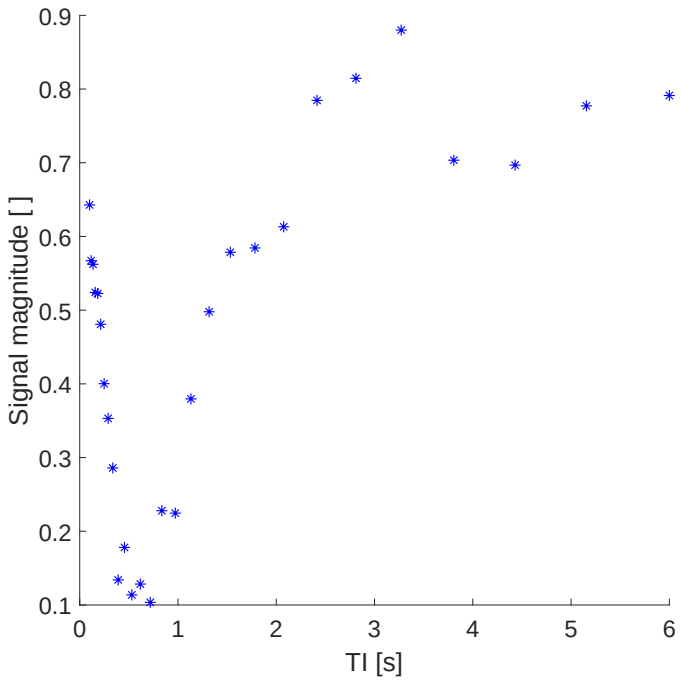

Figure 1: Raw magnitude single-voxel GRE-IR data.

Remark: Data was corrupted with random Rician noise. The noise sigma was defined as the ratio of the mean signal intensity of the whole series, and the SNR value. The latter was chosen equal to 8. Possible, more realistic choices for the noise contributions go beyond the scope of these exercises.

2. The simplified model:

$$M_z = M_0 \left| 1 - 2\exp\left(-\frac{\mathrm{TI}}{T_1}\right) \right| \tag{2}$$

where the absolute value $|\cdot|$ operator is necessary to account for magnitude images. Two model parameters, M_0 and T_1 are to be estimated. The inversion times TI_n are known from the acquisition. M_0 is proportional to the proton density.

3. Example of a single-voxel model function in Matlab:

Code snippet 1: single-voxel T1 model function for magnitude data

```
function [ Mz ] = ModelFunction(TI,PD,T1)
%Magnitude 2 parameter (PD,T1) T1-model:
%
%INPUT:
%    TI = inversion times (in seconds, vector)
%    PD = proton density value
%    T1 = T1 relaxation time value
%
%MAGNETIZATION SIGNAL/OUTPUT:
%    Mz(TI) = abs( PD*[1 - 2*exp(-TI/T1)] )
%-------------------------------------------

Mz = zeros(size(TI));
for i = 1:length(TI)
  Mz(i) = abs( PD*(1-2*exp(-TI(i)/T1)) );
end

end
```

4. When performing the least-squares optimization with `Matlab`'s `lsqnonlin` function, a possible objective function could be:

Code snippet 2: Objective function

```
function [ F ] = residual_T1_2par_abs(theta, TI, data)
%Cost function used in lsqnonlin Levenberg-Marquardt voxel-wise fit.
%MAGNITUDE DATA, no derivatives included.
%
%INPUT:
%    theta = parameter vector
%    TI = inversion times (in seconds, vector)
%    data = data to be model-fitted
%
%OUTPUT:
%    F = vector containing the residual values
%-----------------------------------------------

PD = theta(1);
T1 = theta(2);

F = ModelFunction(TI,PD,T1) - data(:);

end
```

Calling `lsqnonlin` to perform the optimization:

Code snippet 3: Optimization with `lsqnonlin`

```
%MATLAB --> cfr. lsqnonlin
%
%SYNTAX:
%    [params, fval] = lsqnonlin(@(x) CostFunction(x, TI, data),x0,lb,ub,options);
%-----------------------------------------------

[params, fval] = lsqnonlin(@(x) residual_T1_2par_abs(x, TI, data),[0.5 1.0],[],[],options);
```

Remark: It is sufficient to calculate the residual values for the use of `lsqnonlin` in `Matlab`. This might be different in `Python`!

In addition, `Python` might allow choices for the use of different optimizer algorithms, convergence tolerances, use of starting points to help speed up convergence, etc.

5. Visualize the results of the model fit by overlaying them with the original data set. The result should look similar to this:

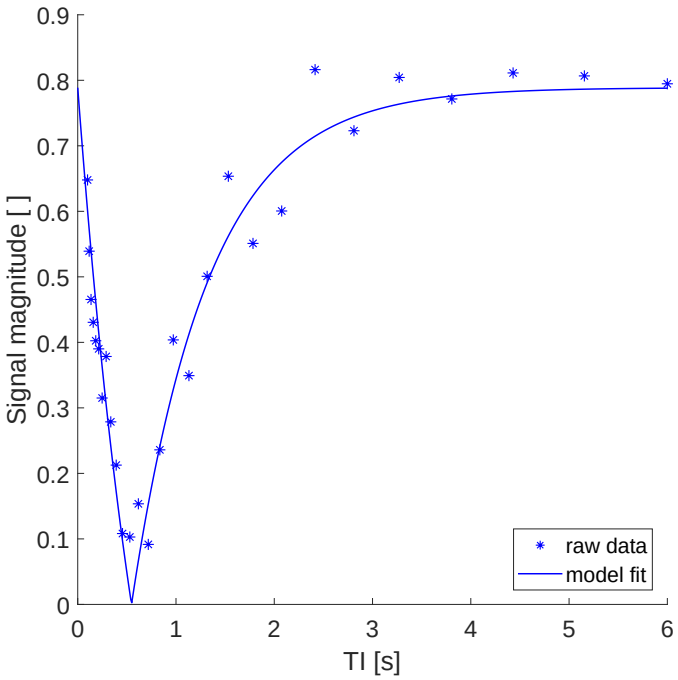

Figure 2: Results of model-fitting

6. In the scope of this exercise it might be sufficient for the students to study the impact of convergence criteria of the optimizer, i.e. it might be worthwhile to set stricter tolerances to let the estimates correspond better with the (local) optimum of the cost function.