week2b

June 15, 2023

Notebook file for Week 2, T2 Relaxation experiment

Import libraries. The nmrbase folder needs to be located within the folder containing this Jupyter notebook

[ ]:
```
import numpy as np
import scipy
import matplotlib.pyplot as plt
import nmrbase.expbase as expbase
import nmrbase.expdta as expdta
```

T2 Relaxation NOTE: To analyze and report multiple datasets, simply copy the necessary blocks of code.

[ ]:
```
filename = r"../DIRECTORY/FILENAME"    #Defines the path to the data file. Here␣
↪the path is relative to the current folder.
```

[ ]:
```
a = expbase.expbase()
a.load(filename)                        # load data

f1=plt.figure()
ax1=f1.subplots()
a.split()                               # splitting raw T2 data for later␣
↪processes
a.plottm(ax1,0)
```

[ ]:
```
## TASKS:
## fine tune with set_xlim and set_ylim parameters to zoom in on the echo
ax1.set_ylim([-0.15,0.15])
ax1.set_xlim([0.15,0.25])

f1
```

[ ]:
```
a.pproc['digfmin']=1500     # set appropriate digital filter parameters
a.pproc['digfmax']=3500
a.digfilt()                 # perform digital filter operation
```

[ ]:
```
## TASKS:
## fine tune with set_xlim and set_ylim parameters to zoom in on the echo
```

```
f2=plt.figure()
ax2=f2.subplots()

a.plottm(ax2,0)   # plot the entire scan
ax2.set_ylim([-0.15,0.15])
ax2.set_xlim([0.15,0.25])   #zoom in on the 1st echo, which is the strongest one
```

[ ]:
```
f3=plt.figure()
ax3 = f3.subplots()

a.pproc['ftmin']=0                   # time interval for Fourier transform (0 =␣
 ↪all data)
a.pproc['ftmax']=0
a.pproc['ffmin']=0                   # frequency interval for spectrum display␣
 ↪(0 = all data)
a.pproc['ffmax']=0
a.proc()                             # calculate Fourier transform of the data␣
 ↪in a
## this will use the digitally filtered data from before. Instead, the original␣
 ↪data can be processed by loading it again.

a.plotfrq(ax3,0)                     # plot the frequency domain data of the␣
 ↪first scan from the data set

## TASKS:
## change parameter "ftmin" and "ftmax" to select the time interval␣
 ↪corresponding to the echoes for the Fourier transform
## change parameter "ffmin" and "ffmax" to select the frequency range of the␣
 ↪NMR signal
## include statements to label axes
## use statements to change appearance such as font size, etc.
```

[ ]:
```
# Time-resolved integration of peaks on NMR spectra
f4=plt.figure()
ax4=f4.subplots()

a.pproc['intmin']=0      # set correct frequency range for integration
a.pproc['intmax']=0

a.integrate()              # perform integration

#find the waiting time tau, then set the correct x-axis
dx=a.pproc["soff"]
x0=a.pproc["slen"]
print('x0 =',x0,'s , dx =',dx,'s')          # x0 is the time of 1st echoes, and␣
 ↪dx is the interval between echoes
```

```
a.idt.x0=x0
a.idt.dx=dx

a.idt.plot(ax4,disp=[0])                    # disp=0 plots only the NMR signal␣
 ↪in trace 0

ax4.set_xlabel("seconds [s]")
ax4.set_ylabel("Y LABEL")

## TASKS:
## change the intmin and intmax parameters to select the correct frequency␣
 ↪range for integration
## use set_xlabel and set_ylabel to set the labels to get a publication quality␣
 ↪figure
```

Fitting of the integral to time axis. NOTE: For some results to fit, optimize the initial conditions

[ ]:
```
# Fitting of the integral

ax5 = plt.figure().subplots()
a.idt.plot(ax5,disp=[0])                    # to be overlayed by fitted curve,␣
 ↪disp=0 plots only the NMR signal in trace 0

def fun(t,a,b,c):
    return FORMULA                           # INPUT the formula used for␣
 ↪fitting. Use "np.exp()" for exponential, and t for time-axis.

y = a.idt.dta[0]
x = np.linspace(a.idt.x0,a.idt.x0+a.idt.dx*round(len(y)-1),len(y))

a.p1,a.p2=scipy.optimize.curve_fit(fun,x,y,p0=[y[0],0.1,0],maxfev=5000)          ␣
 ↪    # OPTIMIZE the initial conditions

print('T\u2082 = ',round(1/a.p1[1],SF1),'±',round(np.linalg.eig(a.p2)[0][1]**0.
 ↪5,SF2),'s')              # REPLACE "SF1" and "SF2" with positive integrers to␣
 ↪report the fitting results with correct significant figures

x=np.linspace(x[0],x[-1],1000)                           # use 1000 points to␣
 ↪generate a smooth curve
a.fit_points = a.p1[0]*np.exp(-a.p1[1]*x)+a.p1[2]
pl,=ax5.plot(x,a.fit_points,'r-')

ax5.set_xlabel("seconds [s]")                         # SET the labels to get a␣
 ↪publication quality figure
ax5.set_ylabel("Y LABEL")
pl.figure.set_tight_layout('pad')
pl.figure.canvas.draw()
```

```
## TASKS:
## change the initial fitting parameters to obtain proper fitting
## report the fitted relaxation constants with proper significant figures
## use set_xlabel and set_ylabel to set the labels to get a publication quality figure
## adjust appearance of figure as needed
```