week3

August 22, 2023

Notebook file for Week 3, Diffusion 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
```

Diffusion 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.plottm(ax1,1)
```

[ ]:
```
## 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([0.25,0.35])

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(ax2,1)   # plot the 1st scan
ax2.set_ylim([-0.15,0.15])
ax2.set_xlim([0.25,0.35])
```

[ ]:
```
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.pproc['dispper']=0.2              # leave 20% space between each acquired␣
 ↪spectrum
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.
```

[ ]:
```
# Gradient-duration-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 starting gradient duration and increment, then set the correct x-axis
dx=a.pinc["inc"][0]
x0=a.p["p2"]
print('x0 =',x0,'s , dx =',dx,'s')          # x0 is the starting gradient␣
 ↪duration, and dx is the increment
```

```
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
big_delta=a.p.get('tau',0.1)                 # time interval between gradients

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

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

a.p1,a.p2=scipy.optimize.curve_fit(fun,x,y,p0=[y[0],0.001],maxfev=5000)       ␣
 ↪         # OPTIMIZE the initial conditions, especially the exponential index

print(round(a.p1[1],SF1),'±',round(np.linalg.eig(a.p2)[0][1]**0.5,SF2))       ␣
 ↪            # 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**2*(big_delta-x/3))
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 gradient strength/diffusion coefficient 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
```