Python On Resonance (PyOR)

Everybody can simulate NMR

Author: Vineeth Thalakottoor

Email: vineethfrancis.physics@gmail.com

Tutorial 8: Supplementary - Spin Echo

Spin Echo is the main ingredient of INEPT. Let see how spin echo works. We have two spins (H1 and C13) with J coupling (Hetronuclear).

Referece book - "NMR: The Toolkit, How Pulse Sequences Work" by P.J Hore, J.A. Jones and S. Wimperis

Load Python packages and define path to the source file "PythonOnResonance.py"

In [1]:
```
pathSource = '/media/HD2/Vineeth/PostDoc_Simulations/Github/PyOR_G/Source'
```

In [2]:
```
from IPython.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
import sys
sys.path.append(pathSource)

import PythonOnResonance as PyOR

import time
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rc
%matplotlib notebook
import sympy as sp
from sympy import *
from IPython.display import display, Math, Latex
```

Generating Spin System

In [3]:
```
"""
Define Spin quantum numbers of your spins in "Slist1".
Slist1[0] is spin of first particle and Slist1[1] is spin of second particle.
""";

Slist1 = [1/2,1/2]
```

In [4]:
```
"""
Define Planck constant equals 1.
Because NMR spectroscopists are more interested to write Energy in frequency units.
if False then hbarEQ1 = hbar
""";
```

```
hbarEQ1 = True
```

In [5]:
```
"""
Generate Spin Operators
""";

System = PyOR.Numerical_MR(Slist1,hbarEQ1)

"""
Sx, Sy and Sz Operators
""";
Sx,Sy,Sz = System.SpinOperator()

"""
S+ and S- Operators
""";
Sp,Sm = System.PMoperators(Sx,Sy)
```

Zeeman Hamiltonian in Lab Frame

In [6]:
```
"""
Gyromagnetic Ratio
Gamma = [Gyromagnetic Ratio spin 1, Gyromagnetic Ratio spin 1, ...]
""";
Gamma = [System.gammaH1,System.gammaC13]

"""
Define the field of the spectromter, B0 in Tesla.
"""
B0 = 9.4

"""
Define the chemical Shift of individual spins
Offset = [chemical Shift spin 1, chemical Shift spin 1, ..]
"""
Offset = [10,0] # Offset frequency in Hz

"""
Function "LarmorF" give the list Larmor frequencies of individual spins in lab frame
"""
LarmorF = System.LarmorFrequency(Gamma,B0,Offset)

Hz = System.Zeeman(LarmorF,Sz)
```

```
Larmor Frequency in MHz:  [-400.22802765 -100.65886793]
```

Initialize Density Matrix

In [7]:
```
"""
We will generate Initial Density Matrix in two ways:
First we will generate a density matrix as we prefer say, Sz.
Second we will create density matrix at thermal equlibrium

First Case
""";

Thermal_DensMatrix = False

if Thermal_DensMatrix:
```

```
    Hz_EnUnit = System.Convert_FreqUnitsTOEnergy(Hz)
    HT_approx = False # High Temperature Approximation is False
    T = 300 # Temperature in Kelvin
    rho_in = System.EqulibriumDensityMatrix(Hz_EnUnit,T,HT_approx)
    rhoeq = rho_in.copy()
else:
    rho_in = np.sum(Sz,axis=0) # Initial Density Matrix
    rhoeq = np.sum(Sz,axis=0)  # Equlibrium Density Matrix
    print("Trace of density metrix = ", np.trace(rho_in))
```

```
Trace of density metrix =  0j
```

In [8]:

```
'''
Operator Basis
Option: 'Cartesian spin half' and 'PMZ spin half'
All the 16 operator basis are loaded in the matrix, 'B_car'
''';
Basis = 'Cartesian spin half'
B_car = System.TwoSpinOP(Sx,Sy,Sz,Sp,Sm,Basis)
```

Basis: $\frac{1}{2}E$,I_x,I_y,I_z,S_x,S_y,S_z,$2I_xS_z$,$2I_yS_z$,$2I_zS_x$,$2I_zS_y$,$2I_zS_z$,$2I_xS_x$,$2I_xS_y$,$2I_yS_x$,$2I_yS_y$

In [9]:

```
'''
B_car[0] = B0 = 1/2 E,
B_car[1] = B1 = Ix,
so on...
Hope you understand.
'''
System.OperatorBasis('Cartesian')
```

Basis: B0 = $\frac{1}{2}E$, B1 = I_x, B2 = I_y, B3 = I_z, B4 = S_x, B5 = S_y, B6 = S_z, B7 = $2I_xS_z$, B8 = $2I_yS_z$, B9 = $2I_zS_x$, B10 = $2I_zS_y$, B11 = $2I_zS_z$, B12 = $2I_xS_x$, B13 = $2I_xS_y$, B14 = $2I_yS_x$, B15 = $2I_yS_y$

In [10]:

```
'''
A density matrix (rho) can be written as:
rho = a B0 + b B1 + c B2 + ...
where B0, B1,... are operator basis.

Components of initial density matrix in cartesian basis
''';
Matrix(System.DensityMatrix_Components(B_car,rho_in))
```

Out[10]:

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1.0 \\ 0 \\ 0 \\ 1.0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Zeeman Halitonian in Rotating Frame

In [11]:
```
OmegaRF = [-System.gammaH1*B0,-System.gammaC13*B0]
Hzr = System.Zeeman_RotFrame(LarmorF, Sz, OmegaRF)
```

J Coupling Hamiltonian

In [12]:
```
"""
Define J Coupling between each spins, Jlist[0][3] means J coupling between 1st spin and 4t
"""

Jlist = np.zeros((len(Slist1),len(Slist1)))
Jlist[0][1] = 150

Hj = System.Jcoupling_Weak(Jlist,Sz)
```

Relaxation Constant

In [13]:
```
'''
Define longitudinal (R1) and transverse Relaxation (R2)
''';

R1 = 1.0
R2 = 2.0
System.Relaxation_Constants(R1,R2)

'''
Options for "Rprocess": "No Relaxation" or "Phenomenological"
                        or "Random Field Fluxtuation" or "Dipolar"
''';
Rprocess = "No Relaxation"
```

Pulse 90x on H1

In [23]:
```
pulse_angle = 90.0
rho = System.Rotate_H(rho_in,pulse_angle,Sx[0])
```

In [24]:
```
'''
Components of density matrix in cartesian basis after 90x on H1
So for H1; rho = -Iy
''';
Matrix(System.DensityMatrix_Components(B_car,rho))
```

Out[24]:

$$\begin{bmatrix} 0 \\ 0 \\ -1.0 \\ 0 \\ 0 \\ 0 \\ 1.0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

First Delay

In [16]:
```
dt = 1.0e-4
fs = 1.0/dt
delay = 1/(4*Jlist[0][1])
Npoints = int(delay/dt)
print("Number of points in the simulation", Npoints)

"""
option for solver, "method": "Unitary Propagator" or "ODE Solver"
"""
method = "Unitary Propagator"

start_time = time.time()
t, rho_t = System.Evolution_H(rhoeq,rho,Sx,Sy,Sz,Sp,Sm,Hzr + Hj,dt,Npoints,method,Rprocess
end_time = time.time()
timetaken = end_time - start_time
print("Total time = %s seconds " % (timetaken))
```

```
Number of points in the simulation 16
Total time = 0.001171112060546875 seconds
```

In [17]:
```
'''
Components of density matrix in cartesian basis after the first delay
So for H1; rho = a Ix + b Iy + c 2IxSz + d 2IySz
''';
Matrix(System.DensityMatrix_Components(B_car,rho_t[-1]))
```

Out[17]:

$$\begin{bmatrix} 0 \\ -0.0731605413363614 \\ -0.725288077218995 \\ 0 \\ 0 \\ 0 \\ 1.0 \\ 0.681090839232821 \\ -0.0687023212742848 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Pulse 180y on H1

In [18]:

```
pulse_angle = 180.0
rho = System.Rotate_H(rho_t[-1],pulse_angle,Sy[0])
```

In [19]:

```
'''
Components of density matrix in cartesian basis after 180y pulse on H1
So for H1; rho = a Ix + b Iy + c 2IxSz + d 2IySz
''';
Matrix(System.DensityMatrix_Components(B_car,rho))
```

Out[19]:

$$\begin{bmatrix} 0 \\ 0.0731605413363614 \\ -0.725288077218995 \\ 0 \\ 0 \\ 0 \\ 1.0 \\ -0.681090839232821 \\ -0.0687023212742848 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Second Delay

In [20]:
```
dt = 1.0e-4
fs = 1.0/dt
delay = 1/(4*Jlist[0][1])
Npoints = int(delay/dt)
print("Number of points in the simulation", Npoints)

"""
option for solver, "method": "Unitary Propagator" or "ODE Solver"
"""
method = "Unitary Propagator"

start_time = time.time()
t, rho_t = System.Evolution_H(rhoeq,rho,Sx,Sy,Sz,Sp,Sm,Hzr + Hj,dt,Npoints,method,Rprocess
end_time = time.time()
timetaken = end_time - start_time
print("Total time = %s seconds " % (timetaken))
```

```
Number of points in the simulation 16
Total time = 0.001230001449584961 seconds
```

In [21]:
```
'''
Components of density matrix in cartesian basis after the secondst delay
So for H1; rho = -Iy
''';
Matrix(System.DensityMatrix_Components(B_car,rho_t[-1]))
```

Out[21]:

$$\begin{bmatrix} 0 \\ 0 \\ -1.0 \\ 0 \\ 0 \\ 0 \\ 0.999999999999999 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Conclusion

For Hetronuclear spin system, after delay + 180y on I + delay, J coupling and chemical shift have been refocused.

Next tutorial: COSY

Any suggestion? write to me

If you see something is wrong please write to me, so that the PyOR can be error free.

vineethfrancis.physics@gmail.com

In [ ]: