

# PyOR: A versatile NMR Simulator



Vineeth Thalakottoor

LBM, UMR 7203, ENS-SU, Group Meeting

# PyOR

*"Everybody can simulate NMR"*

- Python On Resonance
- A collection of Python functions to simulate NMR spin physics numerically
- Versatile: From basic to specialized NMR experiments
- Beginners with basic knowledge of matrices, spin operators and Python programming

Paris en résonance

Veillez fermer cette porte à clé  
lorsqu'il n'y a personne  
(même pendant des pauses brèves)

# Main Features

- Generate Spin Operators, ( $S_x$ ,  $S_y$ ,  $S_z$ ,  $S_+$  and  $S_-$ ) for a system with any number of particles with any spin quantum number
- Generate Spherical Operator Basis for arbitrary system
- Hamiltonians: Zeeman (Lab and Rotating Frame), B1, J coupling and Dipolar Coupling
- Solve Liouville Equation in Hilbert Space or Liouville Space
  - Unitary Propagation
  - Solve ODEs
- Relaxation
  - Redfield Master Equation
  - Lindblad Master Equation
- Radiation Damping and NMR Masers (Multi-mode and J Coupling)
- *User can easily modify the source code according to their needs*

# Motivations to build PyOR

- I was curious to understand "Avoided Crossing"

**Representation of population exchange at level  
anti-crossings**

**Bogdan A. Rodin<sup>1,2</sup> and Konstantin L. Ivanov<sup>1,2</sup>**

- Simulate multi-mode MASER with J-Coupling
- The driving force behind PyOR was my desire to escape the loneliness I experienced in academia.

# Why reinvent the wheel?

*Why develop PyOR when many NMR simulation packages are available?*

- “The Pleasure of Finding Things Out”
- Deeper understanding of Spin Physics
- No black boxes
- Everything is under control

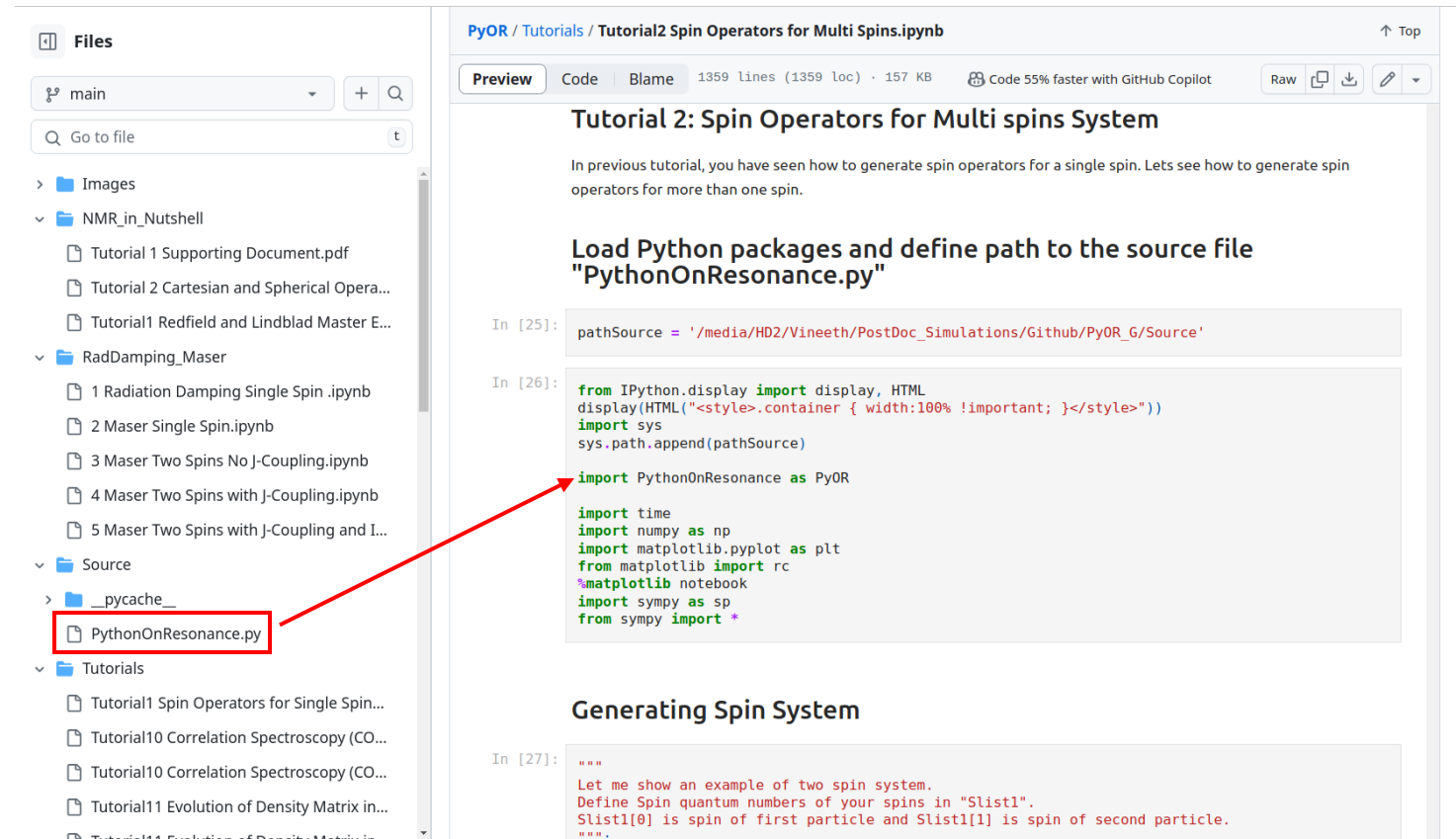


# Why Python?

- It is free
- Easy to learn
- A lot of packages: NumPy, SciPy, SymPy, Matplotlib, ...
- Great online support

# How to use PyOR?

- Install Python in your computer (I prefer Anaconda Python Distribution)
- PyOR Source code: PythonOnResonance.py
- Jupyter Notebook
- PyOR beta version and tutorials are in the Github  
([https://github.com/VThalakottoor/PyOR\\_beta](https://github.com/VThalakottoor/PyOR_beta))
  - Programming
  - Descriptive
- **Modify the source code according to your need**



The screenshot displays a Jupyter Notebook environment. On the left, the 'Files' sidebar shows a directory structure with folders like 'Images', 'NMR\_in\_Nutshell', 'RadDamping\_Maser', 'Source', and 'Tutorials'. The file 'PythonOnResonance.py' is highlighted with a red box. A red arrow points from this file to the code cell in the notebook. The notebook title is 'PyOR / Tutorials / Tutorial2 Spin Operators for Multi Spins.ipynb'. The code cell 'In [26]:' contains the following Python code:

```
In [26]: pathSource = '/media/HD2/Vineeth/PostDoc_Simulations/Github/PyOR_G/Source'

In [26]: 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 *
```

Below the code cell, the text 'Generating Spin System' is visible. The code cell 'In [27]:' contains the following text:

```
In [27]: """
Let me show an example of two spin system.
Define Spin quantum numbers of your spins in "Slist1".
Slist1[0] is spin of first particle and Slist1[1] is spin of second particle.
"""
```



# Final requirement for PyOR: An NMR Book






Let us see how to use PyOR

# First, load PyOR

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

```
1 pathSource = '/media/HD2/Vineeth/PostDoc_Simulations/Github/PyOR_v1.0/Source'
```

```
1 from IPython.display import display, HTML
2 display(HTML("<style>.container { width:100% !important; }</style>"))
3 import sys
4 sys.path.append(pathSource)
5
6 import PythonOnResonance as PyOR
7
8 import time
9 import numpy as np
10 import matplotlib.pyplot as plt
11 from matplotlib import rc
12 %matplotlib notebook
13 import sympy as sp
14 from sympy import *
```



Enter your path here

# Define Spin System

```
"""  
Define Spin quantum numbers of your spins  
""";
```

```
Slist1 = [2]
```

```
"""
```

```
Try also
```

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

```
Slist1 = [1]
```

```
Slist1 = [3/2]
```

```
Slist1 = [2]
```

```
Slist1 = [5/2]
```

```
So on ...
```

```
and compare with results in:
```

```
https://easyspin.org/easyspin/doc/
```

```
""";
```

```
"""
```

```
Let me show an example of two spin system.
```

```
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]
```

```
"""
```

```
Try also
```

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

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

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

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

```
""";
```

# Generate Spin Operators (Heart of PyOR)

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

```
hbarEQ1 = True
```

```
"""  
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)
```

# Documentation

```
1 help(System.SpinOperator)
```

Help on method SpinOperator in module PythonOnResonance:

SpinOperator() method of PythonOnResonance.Numerical\_MR instance

Generate spin operators for all spins: Sx, Sy and Sz

INPUT

-----

nil

OUTPUT

-----

Sx : array [Sx of spin 1, Sx of spin 2, Sx of spin 3, ...]

Sy : array [Sy of spin 1, Sy of spin 2, Sy of spin 3, ...]

Sz : array [Sz of spin 1, Sz of spin 2, Sz of spin 3, ...]

# Matrix Representations

```

1 """
2 Matrix representation (Sympy)
3 In Sx[0], 0 means the index of the spin (0th spin or first spin).
4 """
5 Matrix(Sx[0])

```

$$\begin{bmatrix} 0 & 1.0 & 0 & 0 & 0 \\ 1.0 & 0 & 1.22474487139159 & 0 & 0 \\ 0 & 1.22474487139159 & 0 & 1.22474487139159 & 0 \\ 0 & 0 & 1.22474487139159 & 0 & 1.0 \\ 0 & 0 & 0 & 1.0 & 0 \end{bmatrix}$$

```

1 Matrix(Sy[0])

```

$$\begin{bmatrix} 0 & -1.0i & 0 & 0 & 0 \\ 1.0i & 0 & -1.22474487139159i & 0 & 0 \\ 0 & 1.22474487139159i & 0 & -1.22474487139159i & 0 \\ 0 & 0 & 1.22474487139159i & 0 & -1.0i \\ 0 & 0 & 0 & 1.0i & 0 \end{bmatrix}$$

```

1 Matrix(Sz[0])

```

$$\begin{bmatrix} 2.0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1.0 & 0 \\ 0 & 0 & 0 & 0 & -2.0 \end{bmatrix}$$

```

1 """
2 Matrix representation (Sympy)
3 In Sx[0], 0 means the index of the first spin.
4 And Sx[1], 1 means the index of the second spin.
5 """
6 Matrix(Sz[0]) # Spin operator Sz of first spin

```

$$\begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.5 \end{bmatrix}$$

```

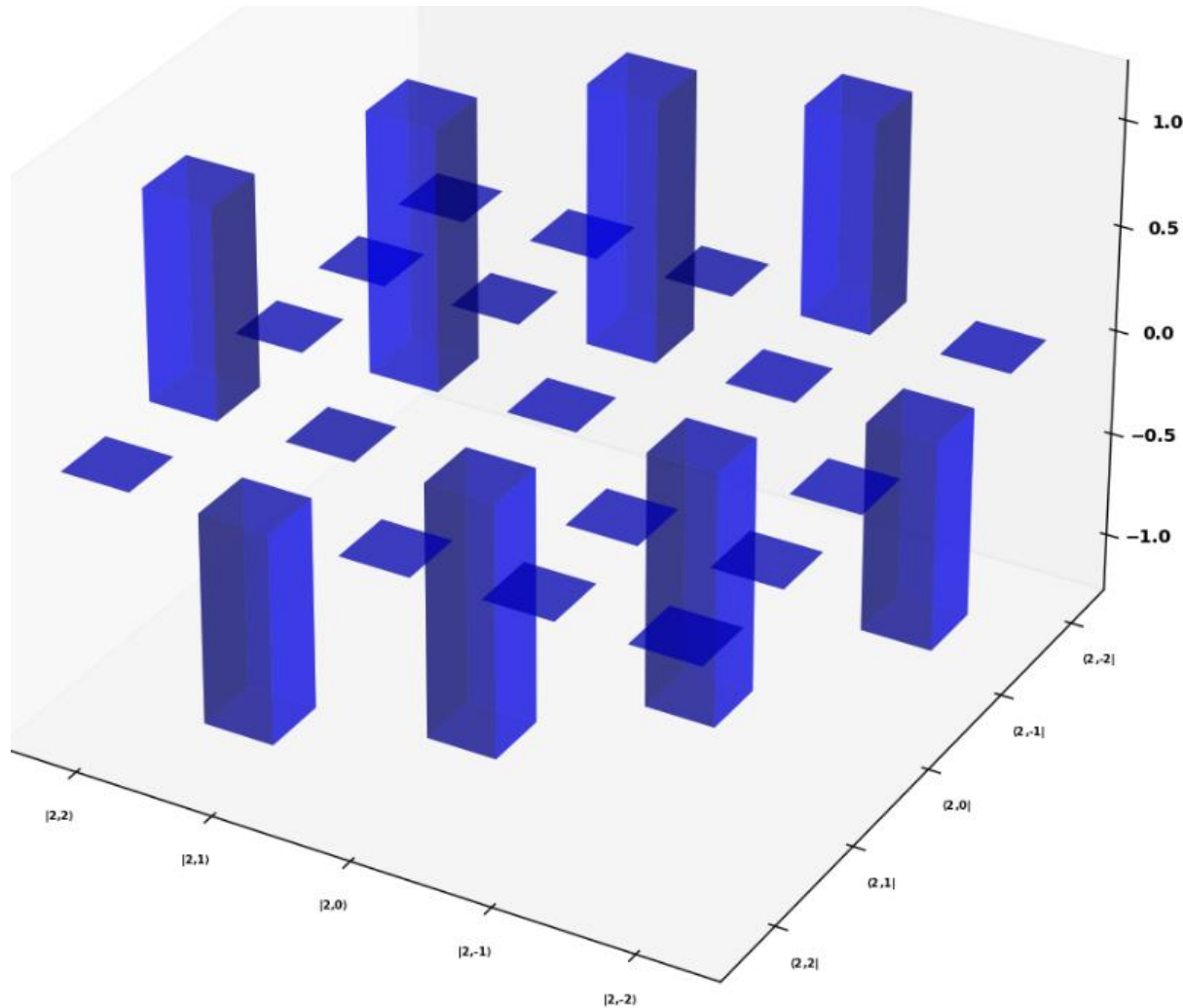
1 Matrix(Sz[1]) # Spin operator Sz of second spin

```

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

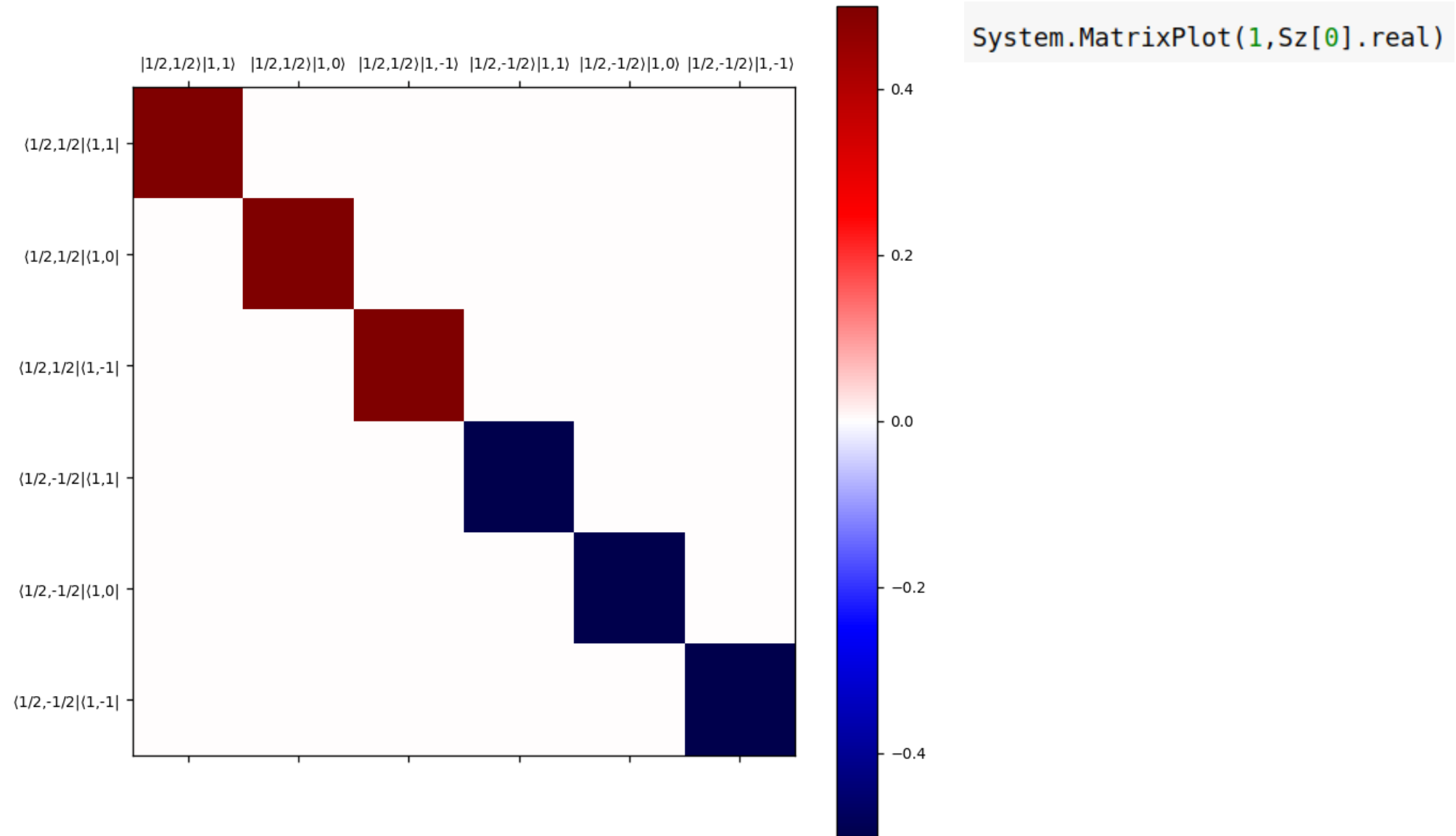


# Matrix Representations



```
System.MatrixPlot3D(2,Sy[0].imag)
```

# Matrix Representations



# Pre-defined Constants

```
1 print("Planck constant, h = ",System.pl)
2 print("Planck constant, hbar = ",System.hbar)
3 print("Permittivity of free space = ",System.ep0)
4 print("Permeability of free space = ",System.mu0)
5 print("Boltzmann constant = ",System.kb)
```

Planck constant, h = 6.626e-34

Planck constant, hbar = 1.054e-34

Permittivity of free space = 8.854e-12

Permeability of free space = 1.2566370614359173e-06

Boltzmann constant = 1.38e-23

```
1 print("Gyromagnetic ratio, electron = ",System.gammaE)
2 print("Gyromagnetic ratio, H1 = ",System.gammaH1)
3 print("Gyromagnetic ratio, C13 = ",System.gammaC13)
4
5 """
6 and gyromagnetic ratio of N14, N15, O17 and F19
7 """;
```

Gyromagnetic ratio, electron = -176100000000.0

Gyromagnetic ratio, H1 = 267522000.0

Gyromagnetic ratio, C13 = 67282800.0

# Zeeman Hamiltonian (Lab Frame)

```
1  """
2  Gyromagnetic Ratio
3  Gamma = [Gyromagnetic Ratio spin 1, Gyromagnetic Ratio spin 1, ...]
4  """;
5  Gamma = [System.gammaH1]
6
7  """
8  Define the field of the spectrometer, B0 in Tesla.
9  """
10 B0 = 9.4
11
12 """
13 Define the chemical Shift of individual spins
14 Offset = [chemical Shift spin 1, chemical Shift spin 1, ..]
15 """
16 Offset = [20] # Offset frequency in Hz
17
18 """
19 Function "LarmorF" give the list Larmor frequencies of individual spins in lab frame
20 """
21 LarmorF = System.LarmorFrequency(Gamma,B0,Offset)
```

Larmor Frequency in MHz: [-400.22803765]

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

# Zeeman Hamiltonian (Rotating Frame)

```
1  """
2  "OmegaRF" is list of rotating frame frequencies
3  """;
4  OmegaRF = [-System.gammaH1*B0]
5
6  """
7  Hamiltonian in the rotating frame
8  """;
9  Hzr = System.Zeeman_RotFrame(LarmorF, Sz, OmegaRF)
```

# B1 Hamiltonian

```
1  """
2  So we have a spin half particle sitting at static magnetic
3  When a RF field is applied the magnetic dipole of the spin
4  The energy of the particle in this case is given by B1 Fie
5  """;
6
7  """
8  List of RF amplitude (Hz) or Nutation frequency, "Omega1"
9  """;
10 Omega1 = [100] # Hz
11
12 """
13 List of RF signal phase in degree
14 """;
15 Omega1Phase = [0] # deg
16
17 """
18 B1 field hamiltonian
19 """;
20 HzB1 = System.Zeeman_B1(Sx,Sy,Omega1,Omega1Phase)
```



# J Coupling Hamiltonian

```
1  """
2  Define J Coupling between each spins, Jlist[0][3] means J coupling between 1st spin and 4th spin.
3  """
4
5  Jlist = np.zeros((len(Slist1),len(Slist1)))
6  Jlist[0][3] = 7
7  Jlist[0][4] = 7
8  Jlist[1][3] = 7
9  Jlist[1][4] = 7
10 Jlist[2][3] = 7
11 Jlist[2][4] = 7
12 Jlist[3][5] = 5
13 Jlist[4][5] = 5
14
15 Hj = System.Jcoupling(Jlist,Sx,Sy,Sz)
```

```
1  """
2  Define J Coupling between each spins, Jlist[0
3  """
4
5  Jlist = np.zeros((len(Slist1),len(Slist1)))
6  Jlist[0][1] = 150
7
8  Hj = System.Jcoupling_Weak(Jlist,Sz)
```

Weak Coupling



# Initialize Density Matrix

```
1  """
2  We will generate Initial Density Matrix in two ways:
3  First we will generate a density matrix as we prefer say, Sz.
4  Second we will create density matrix at thermal equilibrium
5
6  First Case
7  """;
8
9  Thermal_DensMatrix = False
10
11 if Thermal_DensMatrix:
12     Hz_EnUnit = System.Convert_FreqUnitsTOEn
13     HT_approx = False # High Temperature App
14     T = 300 # Temperature in Kelvin
15     rho_in = System.EquilibriumDensityMatrix(
16     rhoeq = rho_in.copy()
17 else:
18     rho_in = np.sum(Sz,axis=0) # Initial Den
19     rhoeq = np.sum(Sz,axis=0) # Equilibrium
20     print("Trace of density metrix = ", np.t
```

Trace of density metrix = 0j

```
1  """
2  Second Case: Initial Desnity Matrix at Thermal Equilibrium
3  """;
4
5  Thermal_DensMatrix = True
6
7  if Thermal_DensMatrix:
8     Hz_EnUnit = System.Convert_FreqUnitsTOEnergy(Hz)
9     HT_approx = False # High Temperature Approximation is False
10    T = 1.2 # Temperature in Kelvin
11    rho_in = System.EquilibriumDensityMatrix(Hz_EnUnit,T,HT_approx)
12    rhoeq = rho_in.copy()
13 else:
14    rho_in = np.sum(Sz,axis=0) # Initial Density Matrix
15    rhoeq = np.sum(Sz,axis=0) # Equilibrium Density Matrix
16    print("Trace of density metrix = ", np.trace(rho_in))
```

Trace of density metrix = 1.0

# Liouville-von Neumann Equation

## Hilbert Space

$$\frac{d}{dt}\rho = \frac{-i}{\hbar}[H_0, \rho]$$

$$\rho(t) = e^{iH_0t}\rho(0)e^{-iH_0t}$$

$$\rho = \begin{bmatrix} \rho_{11} & \rho_{12} & \rho_{13} & \rho_{14} \\ \rho_{21} & \rho_{22} & \rho_{23} & \rho_{24} \\ \rho_{31} & \rho_{32} & \rho_{33} & \rho_{34} \\ \rho_{41} & \rho_{42} & \rho_{43} & \rho_{44} \end{bmatrix}$$

## Liouville Space

$$\frac{d}{dt}\tilde{\rho} = \frac{-i}{\hbar}\hat{H}_0\tilde{\rho}$$

$$\tilde{\rho} = \begin{bmatrix} \rho_{11} \\ \rho_{12} \\ \rho_{13} \\ \rho_{14} \\ \rho_{21} \\ \vdots \\ \rho_{44} \end{bmatrix}$$

$$\frac{d}{dt}\tilde{\rho} = \frac{-i}{\hbar}(H_0 \otimes \mathbb{1} - \mathbb{1} \otimes H_0^T)\tilde{\rho}$$

$$\tilde{\rho}(t) = e^{-i\hat{H}_0t}\tilde{\rho}(0)$$

# Basis Operators (Example: 3 Three spin half) (Irreducible spherical tensor Operators)

An arbitrary square matrix of dimension  $(2s + 1) \times (2s + 1)$ ,  $A$  can be written as linear combination of spherical tensor operators:

$$A = \sum_{k=0}^{2s} \sum_{q=-k}^k a_q^k T_q^k, \text{ where } a_q^k = \text{Tr}[(T_q^k)^\dagger A]$$

```
1  """
2  Spherical Operator Basis for single spin half particle
3  let S = 1/2
4  """;
5
6  pol_basis_half, Coherence_order, LM_state = System.Spherical_OpBasis(1/2)
```

Coherence Order: [0, -1, 0, 1]

LM state: [(0, 0), (1, -1), (1, 0), (1, 1)]

# Basis Operators (Example: 3 Three spin half)

```
1 """
2 First make product operator basis for two spin half particle and then with the third, Spin 1 -> I, Spin 2 -> S and Spin 3 -> R
3 """
4
5 Coh_order = [0, -1, 0, 1]
6 Dic_1 = ["E1 ", "Im ", "Iz ", "Ip "]
7 Dic_2 = ["E2 ", "Sm ", "Sz ", "Sp "]
8 Dic_3 = ["E3 ", "Rm ", "Rz ", "Rp "]
9
10 """
11 Sorting options: 'normal', 'negative to positive', 'zero to high'
12 """
13
14 sort = 'negative to positive'
15 indexing = False
16 product_basis_2half, Coh_order_2, Dic_2 = System.ProductOperator(pol_basis_half, Coh_order, Dic_1, pol_basis_half, Coh_order, Dic_2, sort, in
```

```
1 print("Coherence order (Two spin half particles): ", Coh_order_2)
```

Coherence order (Two spin half particles): [-2, -1, -1, -1, -1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2]

```
1 print("Dictionary of operator basis for two spin half particles: ", Dic_2)
```

Dictionary of operator basis for two spin half particles: ['Im Sm ', 'E1 Sm ', 'Im E2 ', 'Im Sz ', 'Iz Sm ', 'E1 E2 ', 'E1 Sz ', 'Im Sp ', 'Iz E2 ', 'Iz Sz ', 'Ip Sm ', 'E1 Sp ', 'Iz Sp ', 'Ip E2 ', 'Ip Sz ', 'Ip Sp ']

# Basis Operators (Example: 3 Three spin half)

```
1 """
2 Product operator basis for three spin half particles
3 """
4
5 indexing = True
6 product_basis_3half, Coh_order_3, Dic_3 = System.ProductOperator(product_basis_2half, Coh_order_2, Dic_2, pol_basis_half, Coh_order, Dic_3,
```

```
1 print("Coherence order (Three spin half particles): ", Coh_order_3)
```

[illegible]

```
1 print("Dictionary of operator basis for three spin half particles: ", Dic_3)
```

Dictionary of operator basis for three spin half particles: ['Im Sm Rm [0]', 'Im Sm E3 [1]', 'Im Sm Rz [2]', 'Im Sz Rm [3]', 'Im Sz Rz [4]', 'Im Sz Rm [5]', 'Iz Sm Rm [6]', 'Im Sm Rp [7]', 'E1 Sm E3 [8]', 'E1 Sm Rz [9]', 'Im E2 E3 [10]', 'Im E2 Rm [11]', 'Im Sz Rz [13]', 'Iz Sm E3 [14]', 'Iz Sm Rz [15]', 'E1 E2 Rm [16]', 'E1 Sz Rm [17]', 'Im Sp Rm [18]', 'Iz E2 Rm [19]', 'Ip Sm Rm [21]', 'E1 Sm Rp [22]', 'Im E2 Rp [23]', 'Im Sz Rp [24]', 'Iz Sm Rp [25]', 'E1 E2 E3 [26]', 'E1 E2 Rz [27]', 'E1 Sz Rz [29]', 'Im Sp E3 [30]', 'Im Sp Rz [31]', 'Iz E2 E3 [32]', 'Iz E2 Rz [33]', 'Iz Sz E3 [34]', 'Iz Sz Rz [35]', 'Ip Sm Rz [37]', 'E1 Sp Rm [38]', 'Iz Sp Rm [39]', 'Ip E2 Rm [40]', 'Ip Sz Rm [41]', 'E1 E2 Rp [42]', 'E1 Sz Rp [43]', 'Iz E2 Rp [45]', 'Iz Sz Rp [46]', 'Ip Sm Rp [47]', 'E1 Sp E3 [48]', 'E1 Sp Rz [49]', 'Iz Sp E3 [50]', 'Iz Sp Rz [51]', 'Ip E2 Rz [53]', 'Ip Sz E3 [54]', 'Ip Sz Rz [55]', 'Ip Sp Rm [56]', 'E1 Sp Rp [57]', 'Iz Sp Rp [58]', 'Ip E2 Rp [59]', 'Ip Sp E3 [61]', 'Ip Sp Rz [62]', 'Ip Sp Rp [63]']

### Product Operator, $I_S R_S$

```
1 Matrix(product_basis_3half[0])
```

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



# Density Matrix in terms of Basis Operators

```
1  """
2  Basis Operators in Cartesian
3  """;
4  Basis = 'Cartesian spin half'
5  B_CarT, dic = System.SingleSpinOP(Sx,Sy,Sz,Sp,Sm,Basis)
```

```
1  """
2  Dictionary
3  """;
4  dic
['E$', 'Ix', 'Iy$', 'Iz']
```

**Single Spin  
Half**

```
1  '''
2  Operator Basis
3  Option: 'Cartesian spin half' and 'PMZ spin half'
4  All the 16 operator basis are loaded in the matrix, 'B_car'
5  ''';
6  Basis = 'Cartesian spin half'
7  B_car, dic = System.TwoSpinOP(Sx,Sy,Sz,Sp,Sm,Basis)
```

**Two Spin Half**

**Projection of Density Matrix**

```
1  System.DensityMatrix_Components_Dictionary(B_car,dic,rho)
```

Density Matrix = 0.0 E + 0.0 Sx + -1.0 Sy + 0.0 Sz + 0.0 Ix + 0.0 Iy + 1.0 Iz + 0.0 Sx Iz + 0.0 Sy Iz + 0.0 Sz Ix + 0.0 Sz Iy + 0.0 Sz Iz + 0.0 Sx Ix + 0.0 Sx Iy + 0.0 Sy Ix + 0.0 Sy Iy +

# Basis States (Example: 3 Three spin half)

```
1  """
2  Basis Ket
3  """;
4  Kets = System.Basis_Ket()
5  display(Kets)
```

```
[ '|1/2,1/2>|1/2,1/2>|1/2,1/2>',
  '|1/2,1/2>|1/2,1/2>|1/2,-1/2>',
  '|1/2,1/2>|1/2,-1/2>|1/2,1/2>',
  '|1/2,1/2>|1/2,-1/2>|1/2,-1/2>',
  '|1/2,-1/2>|1/2,1/2>|1/2,1/2>',
  '|1/2,-1/2>|1/2,1/2>|1/2,-1/2>',
  '|1/2,-1/2>|1/2,-1/2>|1/2,1/2>',
  '|1/2,-1/2>|1/2,-1/2>|1/2,-1/2>' ]
```

```
1  """
2  Basis Bra
3  """;
4  Bras = System.Basis_Bra()
5  display(Bras)
```

```
[ '<1/2,1/2|<1/2,1/2|<1/2,1/2|',
  '<1/2,1/2|<1/2,1/2|<1/2,-1/2|',
  '<1/2,1/2|<1/2,-1/2|<1/2,1/2|',
  '<1/2,1/2|<1/2,-1/2|<1/2,-1/2|',
  '<1/2,-1/2|<1/2,1/2|<1/2,1/2|',
  '<1/2,-1/2|<1/2,1/2|<1/2,-1/2|',
  '<1/2,-1/2|<1/2,-1/2|<1/2,1/2|',
  '<1/2,-1/2|<1/2,-1/2|<1/2,-1/2|' ]
```

# Basis State / Operator Transformation

```

1  """
2  Now lets see how to get the eigen vectors of the Zeman Hamiltonian (lab frame)
3  """
4  B_Z = System.ZBasis_H(Hz)

```

$|1/2, 1/2\rangle |1/2, 1/2\rangle, |1/2, 1/2\rangle |1/2, -1/2\rangle, |1/2, -1/2\rangle |1/2, 1/2\rangle, |1/2, -1/2\rangle |1/2, -1/2\rangle$

```

1  """
2  Singlet Triplet Basis
3  """
4  B_ST = System.STBasis(B_Z)

```

Basis:  $T_-, T_0, T_+, S_0$

```

1  """
2  Basis Transformation
3  """
4  U = System.Transform_StateBasis(B_Z, B_ST)
5  Matrix(U)

```

$$\begin{bmatrix} 1.0 & 0 & 0 & 0 \\ 0 & 0.707106781186547 & 0.707106781186547 & 0 \\ 0 & 0 & 0 & 1.0 \\ 0 & 0.707106781186547 & -0.707106781186547 & 0 \end{bmatrix}$$

```

6  Hj = System.Jcoupling(Jlist, Sx, Sy, Sz)
7  Matrix(Hj / (2.0 * np.pi))

```

$$\begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & -1.25 & 2.5 & 0 \\ 0 & 2.5 & -1.25 & 0 \\ 0 & 0 & 0 & 1.25 \end{bmatrix}$$

```

4  Hj_ST = System.Operator_BasisChange(Hj, U)
5  Matrix(System.Matrix_Tol(Hj_ST, 1.0e-10) / (2.0 * np.pi))

```

$$\begin{bmatrix} 1.25 & 0 & 0 & 0 \\ 0 & 1.25 & 0 & 0 \\ 0 & 0 & 1.25 & 0 \\ 0 & 0 & 0 & -3.75 \end{bmatrix}$$

```

1  Sx_ST = System.SpinOperator_BasisChange(Sx, U)
2  Matrix(System.Matrix_Tol(Sx_ST[0], 1.0e-10))

```

# Hilbert or Liouville ?

## Hilbert

### Pulse

```
1  """
2  Rotate the magnetization about Y-axis, by an angle theta.
3  """
4  pulse_angle = 90.0
5  rho = System.Rotate_H(rho_in,pulse_angle,np.sum(Sy,axis=0))
```

## Liouville

### Converting to Liouvillian

```
1  Hz_L = System.CommutationSuperoperator(Hz)
2  Hzr_L = System.CommutationSuperoperator(Hzr)
3  rho_in_L = System.Vector_L(rho_in)
4  rhoeq_L = System.Vector_L(rhoeq)
```

```
1  """
2  Rotate the magnetization about Y-axis, by an angle theta.
3  """
4  pulse_angle = 90.0
5  rho_L = System.Rotate_L(rho_in_L,pulse_angle,np.sum(Sy,axis=0))
```

# Evolution of Density Matrix

```
1  """
2  Options: "No Relaxation", "Phenomenological", "Auto-corre
3  """;
4  R1 = None
5  R2 = None
6  Rprocess = "Auto-correlated Dipolar Homonuclear"
7  tau = 10.0e-12
8  bIS = 30.0e3
9  System.Relaxation_Constants(R1,R2)
10 System.Relaxation_Parameters(LarmorF, OmegaRF, tau, bIS)
```

```
14 """
15 option for solver, "method": "Unitary Propagator" or "ODE Solver"
16 """
17 method = "Unitary Propagator"
18
19 start_time = time.time()
20 t, rho_t = System.Evolution_H(rhoeq, rho, Sx, Sy, Sz, Sp, Sm, Hrz, dt, Npoints, method, Rprocess)
21 end_time = time.time()
22 timetaken = end_time - start_time
23 print("Total time = %s seconds " % (timetaken))
```

Hilbert

```
6  """
7  option for solver, "method": "Unitary Propagator", "Relaxation" or "ODE Solver"
8  """
9  method = "Relaxation"
10
11 start_time = time.time()
12 t, rho_t = System.Evolution_L(rhoeq_L, rho_L, Sx, Sy, Hrz_L - 1j * R_L, dt, Npoints, method)
13 end_time = time.time()
14 timetaken = end_time - start_time
15 print("Total time = %s seconds " % (timetaken))
```

Liouville

# Relaxation Super-operator

```
1  """
2  Options: "No Relaxation", "Phenomenological", "Auto-correlated Dipolar Homonuclear"
3  """;
4  R = None
5  Rprocess = "Auto-correlated Dipolar Homonuclear"
6  tau = 10.0e-12
7  bIS = 30.0e3
8  System.Relaxation_Parameters(LarmorF, OmegaRF, tau, bIS)
9  R_L = System.Relaxation_L(Rprocess, R, Sx, Sy, Sz, Sp, Sm)
```

**Redfield**

```
5  Rprocess = "Auto-correlated Dipolar Hetrnuclear"
6  tau = 10.0e-12
7  bIS = 30.0e3
8  System.Relaxation_Parameters(LarmorF, OmegaRF, tau, bIS)
9  System.Temperature(T)
10 R_L = System.Relaxation_Lindblad(Rprocess, Sx, Sy, Sz, Sp, Sm)
```

**Lindblad**



# Write your own Relaxation Mechanism

```
def Relaxation_Lindblad(self, Rprocess, Sx, Sy, Sz, Sp, Sm):  
    """  
    Lindblad Relaxation in Liouville Space  
  
    INPUT  
    ----  
    Rprocess: "No Relaxation" or "Auto-correlated Dipolar Homonuclear" or "Auto-correlated Dipolar Heteronuclear"  
    Sx: Spin Operator Sx  
    Sy: Spin Operator Sy  
    Sz: Spin Operator Sz  
    Sp: Spin Operator Sp  
    Sm: Spin Operator Sm  
  
    OUTPUT  
    ----  
    Rso: Relaxation Superoperator  
  
    """  
    if Rprocess == "No Relaxation":  
        """  
        No Relaxation  
        """  
        Rso = np.zeros((self.Ldim, self.Ldim))  
  
    if Rprocess == "Auto-correlated Dipolar Homonuclear":  
        """  
        Auto-correlated Dipolar Homonuclear Relaxation  
        Extreme Narrowing  
        """  
        Rso = np.zeros((self.Ldim, self.Ldim), dtype=np.cdouble)  
        m = [-2, -1, 0, 1, 2]  
        for i in m:  
            Rso = Rso + (-1)**i * self.SpectralDensity_Lb(i * self.LarmorF[0], self.tau) * self.Lindblad_Dissipator(self.Spherical_Tensor([0, 1],  
2, i, Sx, Sy, Sz, Sp, Sm), self.Spherical_Tensor([0, 1], 2, -i, Sx, Sy, Sz, Sp, Sm))  
        Rso = Rso * (-6/5) * self.bIS**2
```

If Rprocess == "Your Relaxation Mechanism":

"""

Short Description

"""

Your code here

# Relaxation Mechanisms Implemented

- **Redfield Master Equation**

- *Hilbert Space*

- Phenomenological, Auto-Correlated Random Field Fluctuation, Auto-Correlated Homonuclear, Auto-Correlated Heteronuclear

- *Liouville Space*

- Phenomenological, Auto-Correlated Random Field Fluctuation, Auto-Correlated Homonuclear, Auto-Correlated Heteronuclear, Cross Correlated CSA-Dipolar Heteronuclear

- **Lindblad Master Equation**

- *Liouville Space*

- Auto-Correlated Homonuclear, Auto-Correlated Heteronuclear

# Detection – Expectation Value

```
1 EXP_Z1 = Sz[0]
2 EXP_Z2 = Sz[1]
3
4 t, Mz1 = System.Expectation_H(rho_t, EXP_Z1, dt, Npoints)
5 t, Mz2 = System.Expectation_H(rho_t, EXP_Z2, dt, Npoints)
```

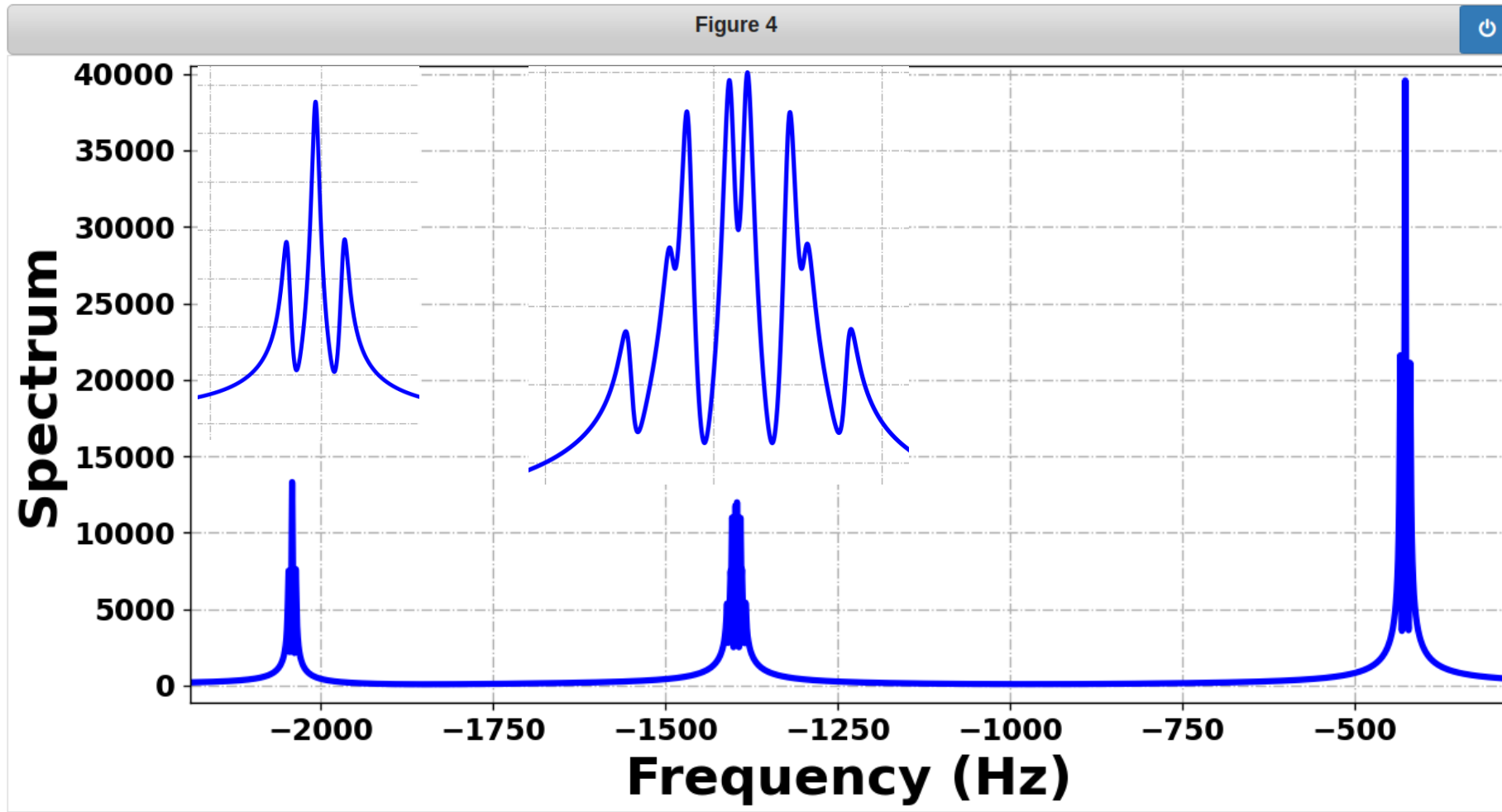
**Hilbert**

```
1 EXP_T = np.sum(Sx,axis=0) + 1j * np.sum(Sy,axis=0)
2 EXP_Z = np.sum(Sz,axis=0)
3
4 LEXP_T = System.Detection_L(EXP_T)
5 LEXP_Z = System.Detection_L(EXP_Z)
6
7 t, Mp = System.Expectation_L(rho_t, LEXP_T, dt, Npoints)
8 t, Mz = System.Expectation_L(rho_t, LEXP_Z, dt, Npoints)
```

**Liouville**

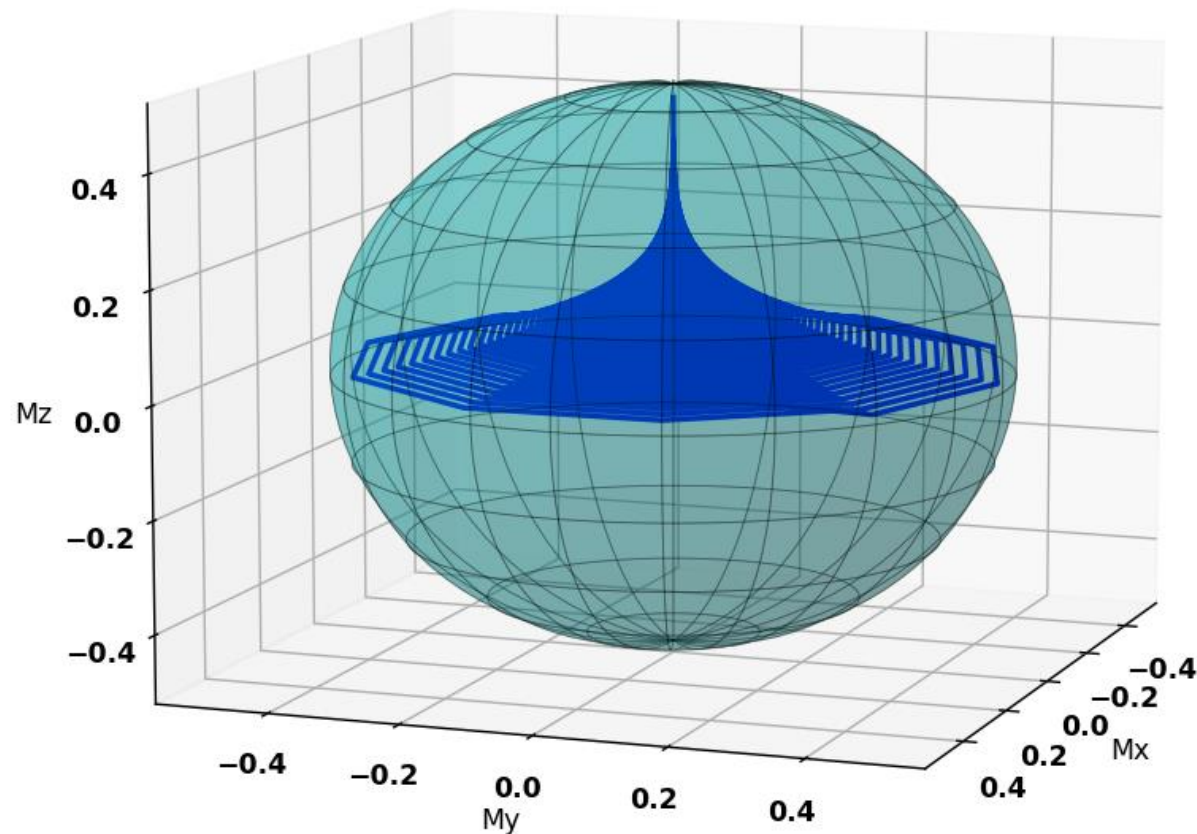
# Simulation Results Visualization: Plotting (Ethanol Spectra)

```
1 System.Plotting(4,freq,np.absolute(spectrum),"Frequency (Hz)","Spectrum","blue")
```



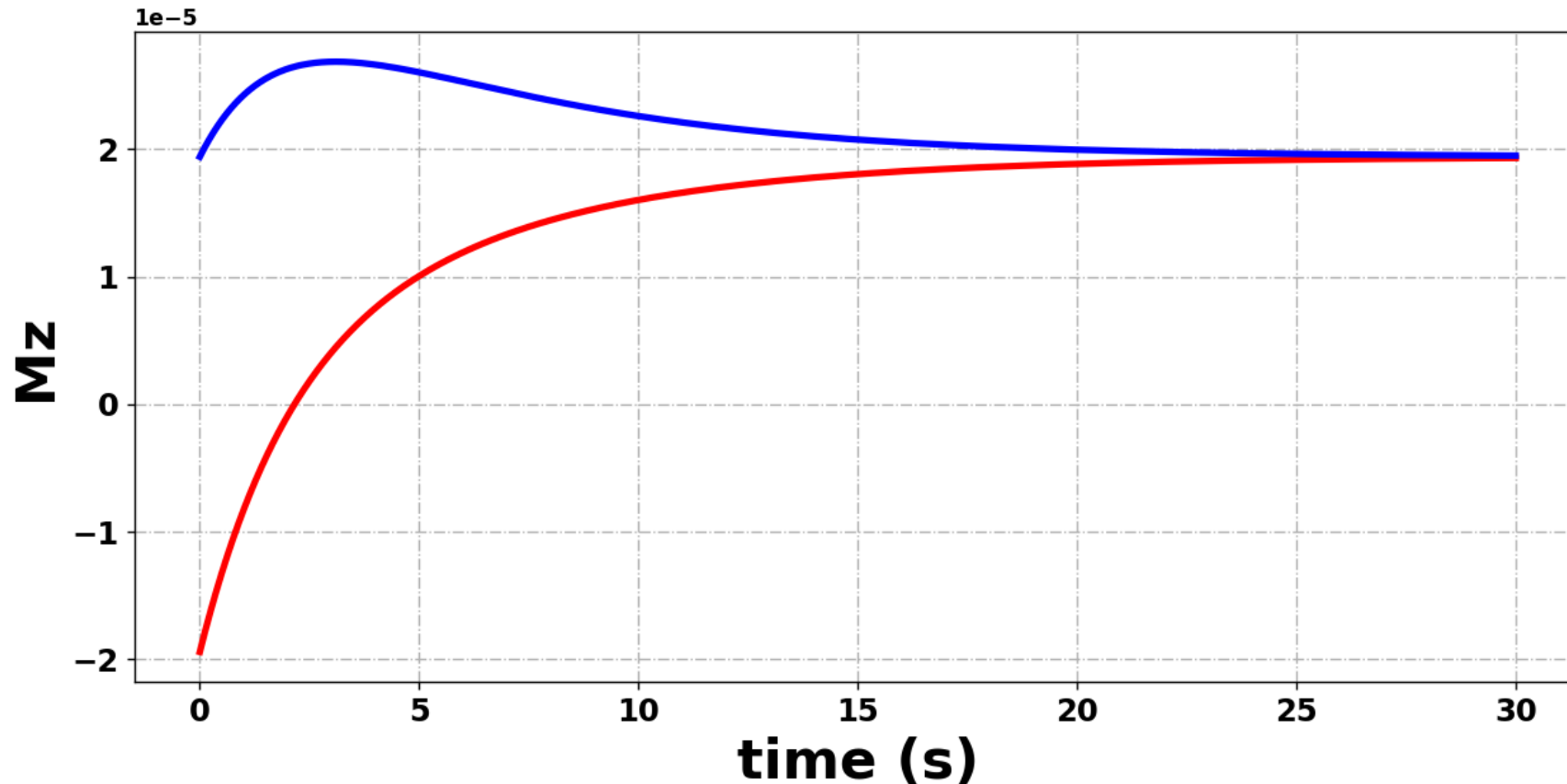
# Simulation Results Visualization: Sphere

```
1 plot_vector = False
2 scale_datapoints = 2
3 System.PlottingSphere(8,Mp.real,Mp.imag,Mz,rhoeq,np.sum(Sz,axis=0),plot_vector,scale_datapoints)
```

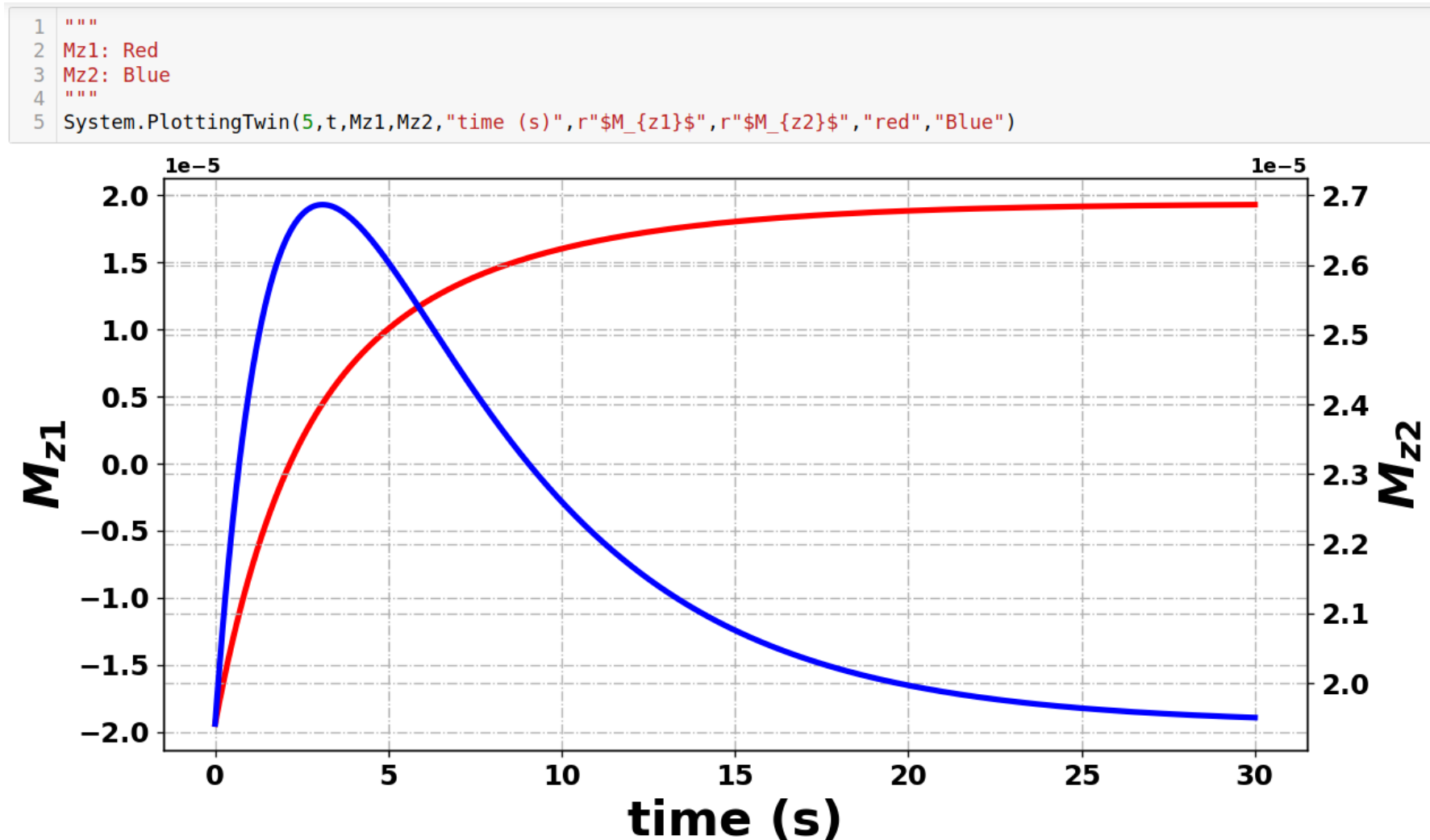


# Simulation Results Visualization: Multiple Plots (NOE Lindblad Homonuclear)

```
1 ""  
2 Mz1: Red  
3 Mz2: Blue  
4 ""  
5 System.PlottingMulti(4,[t,t],[Mz1,Mz2],"time (s)","Mz",["red","blue"])
```



# Simulation Results Visualization: Twin Axis Plots (NOE Lindblad Homonuclear)

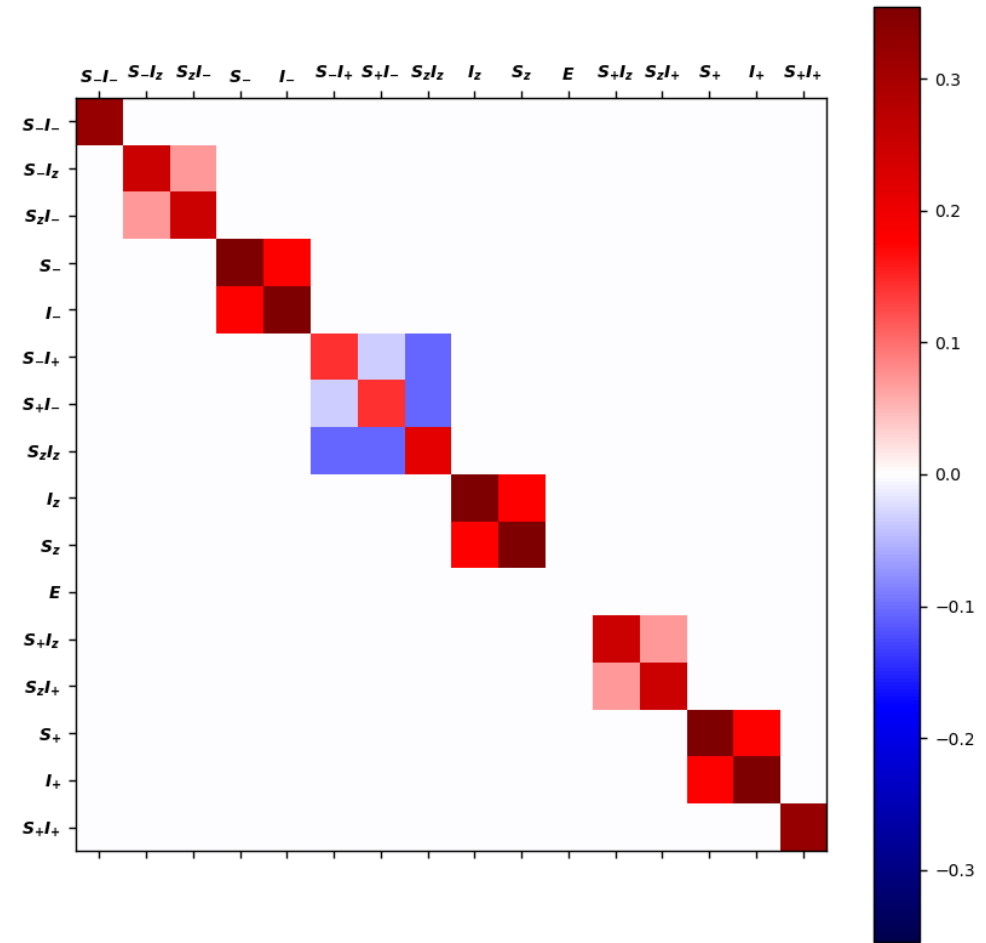


# Simulation Results Visualization: RedKite

```
Coherenceorder = "-2,-1,0,1,2"  
R_redkite, Basis_L = System.Transform_Redkite(R_L,Sp,Sm,Sz,Coherenceorder)
```

$S_-I_-, S_-I_z, S_zI_-, S_-, I_-, S_-I_+, S_+I_-, S_zI_z, I_z, S_z, E, S_+I_z, S_zI_+, S_+, I_+, S_+I_+$

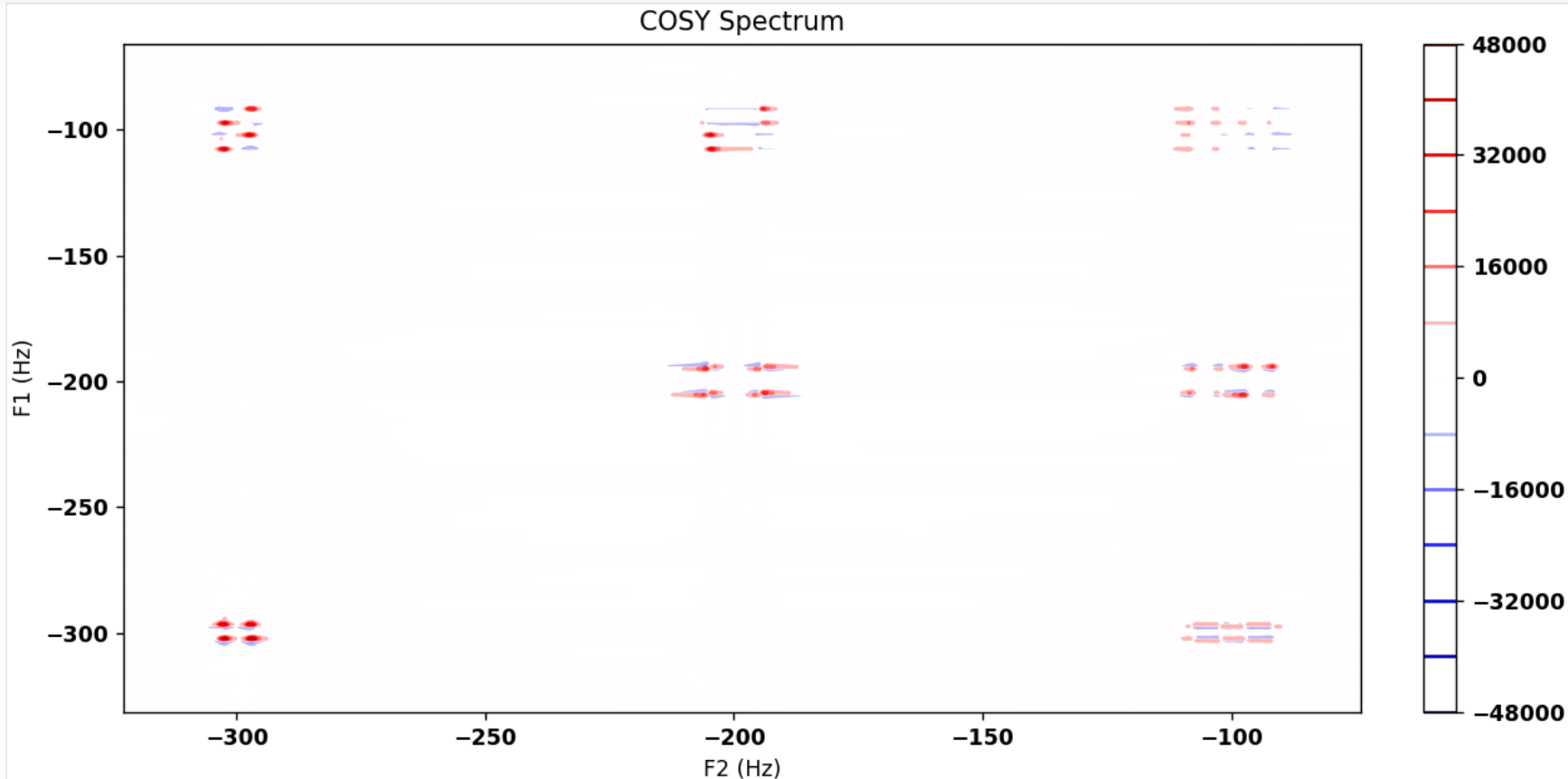
```
System.PlotLabel_Hilbert = False  
System.Redkite_Label_SpinDynamica = True  
System.MatrixPlot(1,R_redkite.real)
```





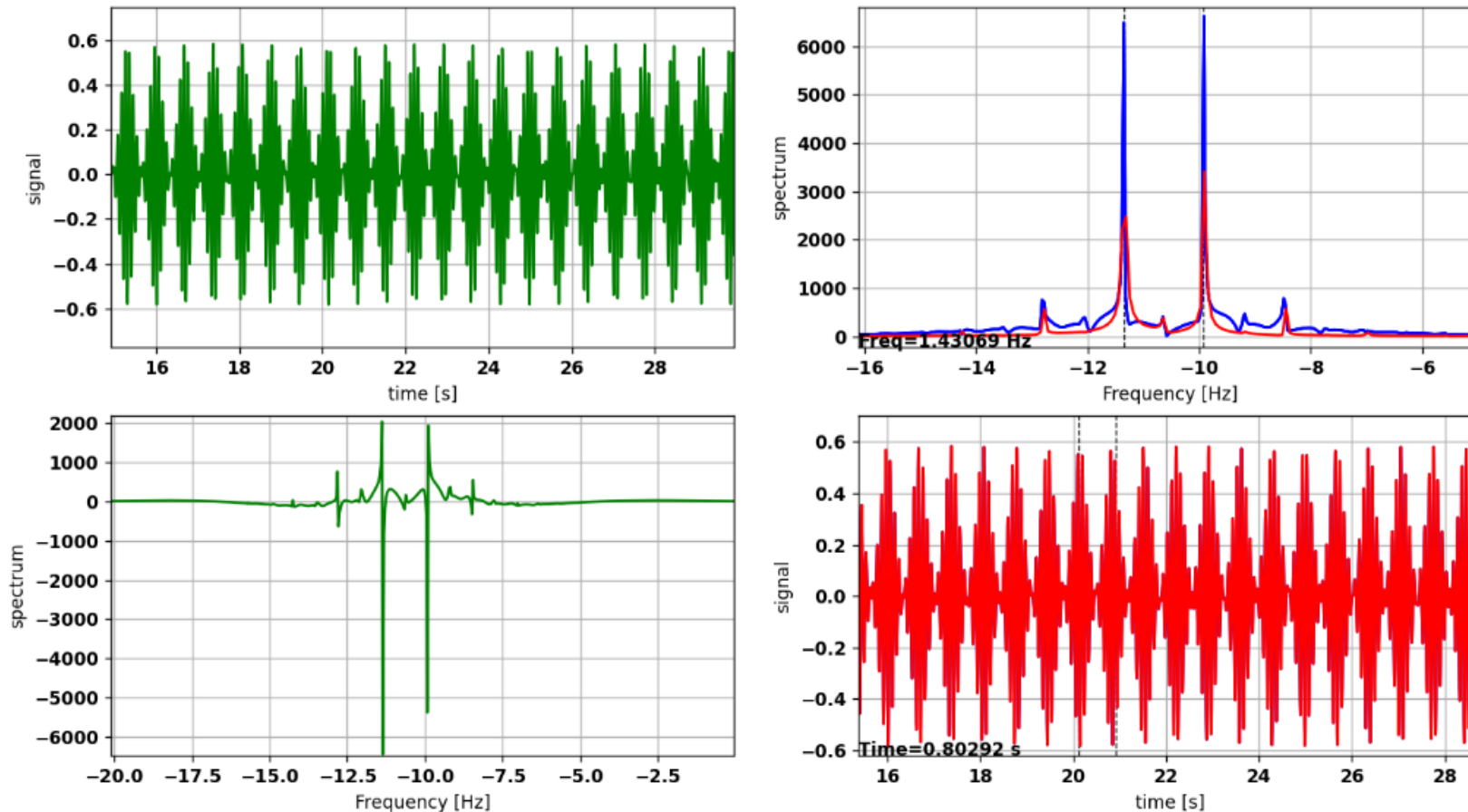
# Simulation Results Visualization: Contour Plotting (COSY)

```
1 # Contour Plot
2 PH0 = 45
3 spectrum_PH0_2D = System.PhaseAdjust_PH0(spectrum,PH0)
4 System.PlottingContour(4,F2,F1,spectrum_PH0_2D,"F2 (Hz)","F1 (Hz)","COSY Spectrum")
```

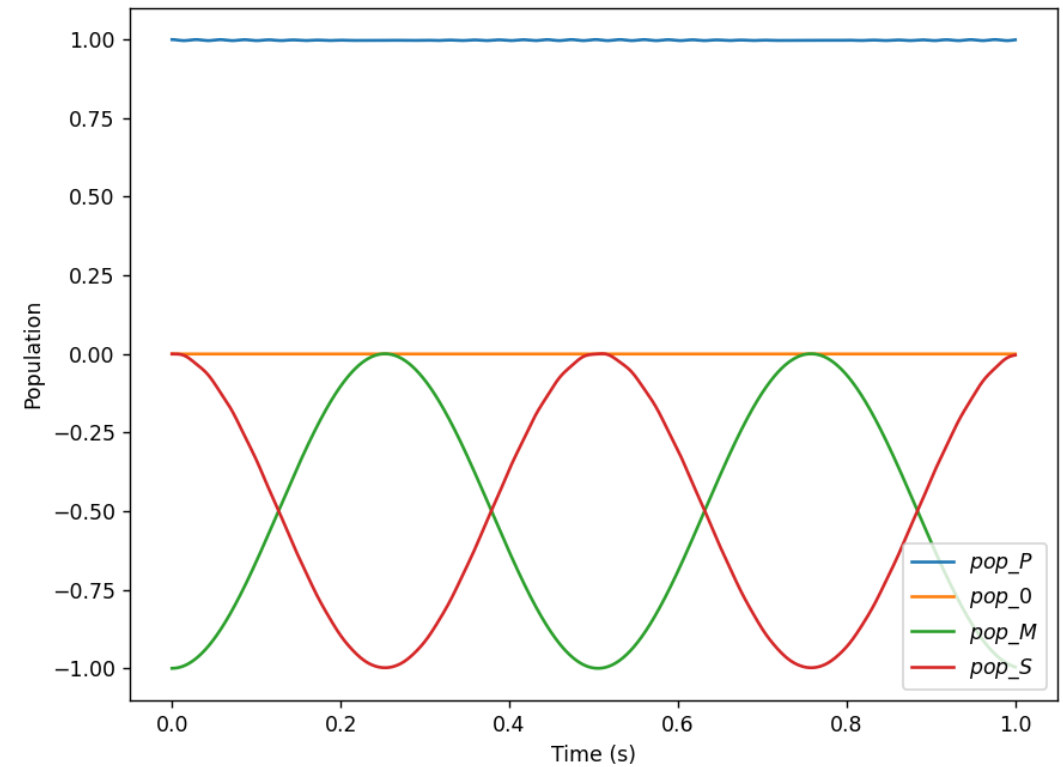
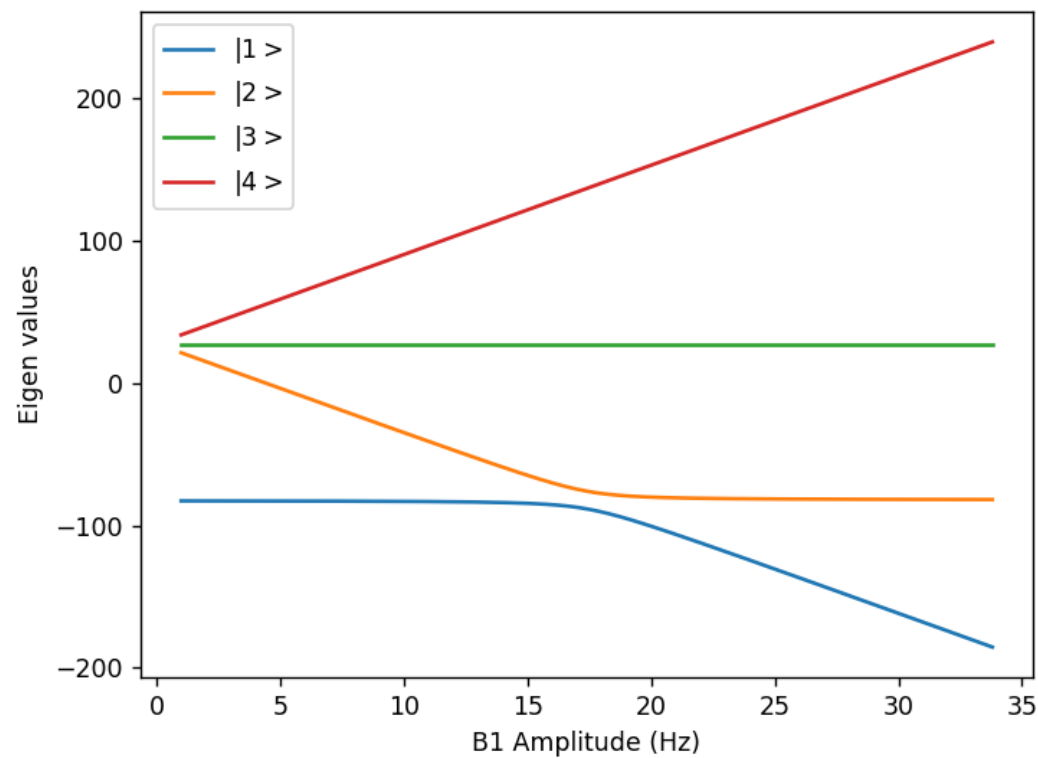


# Simulation Results Visualization: Multi-mode Analyzer

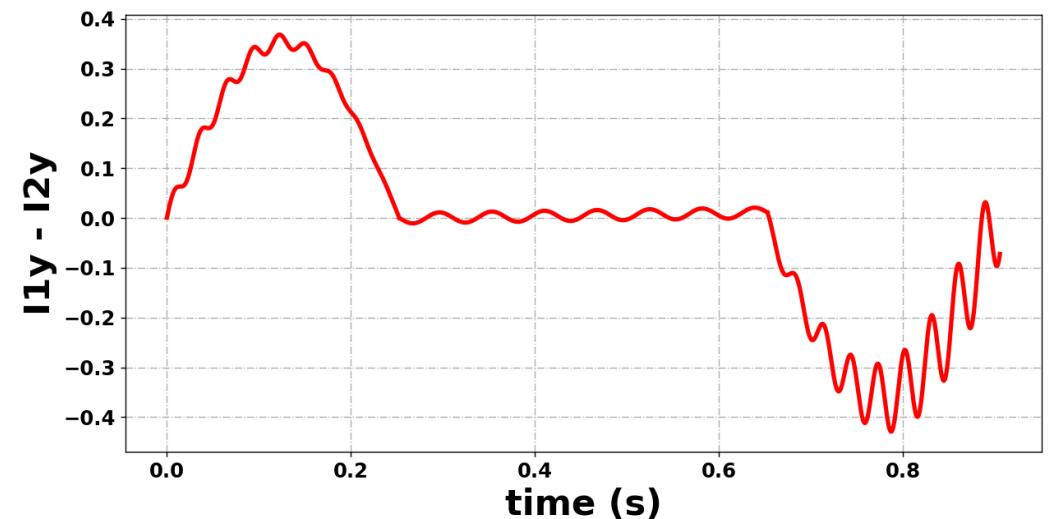
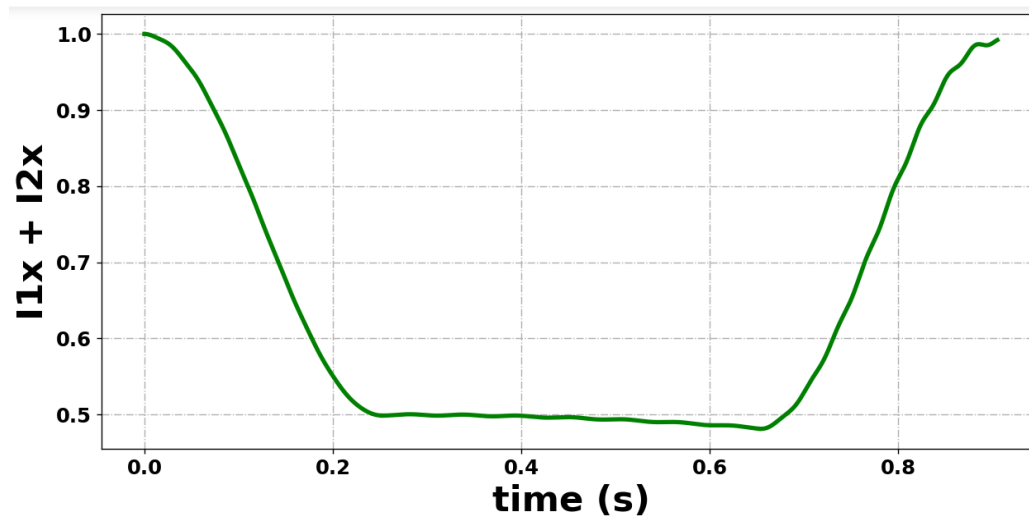
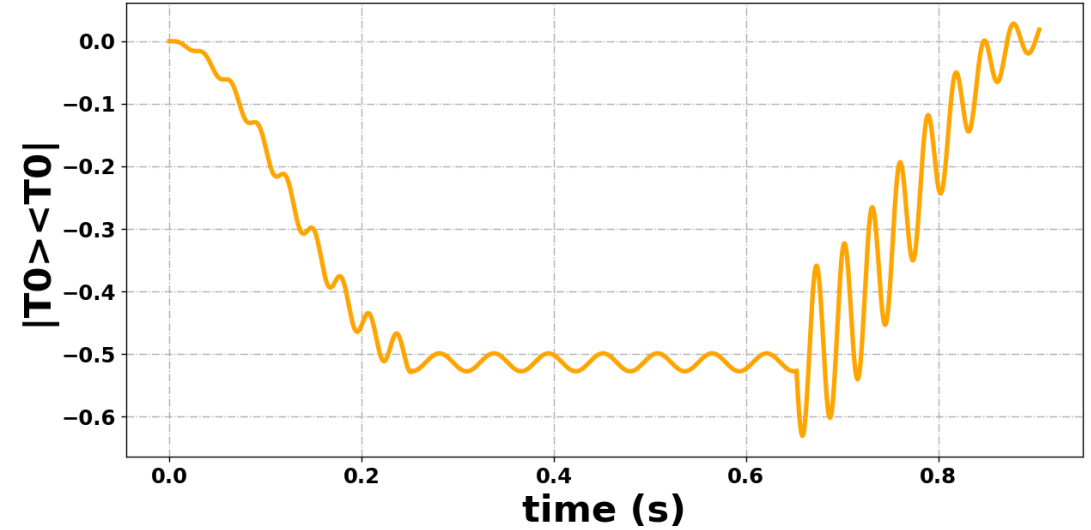
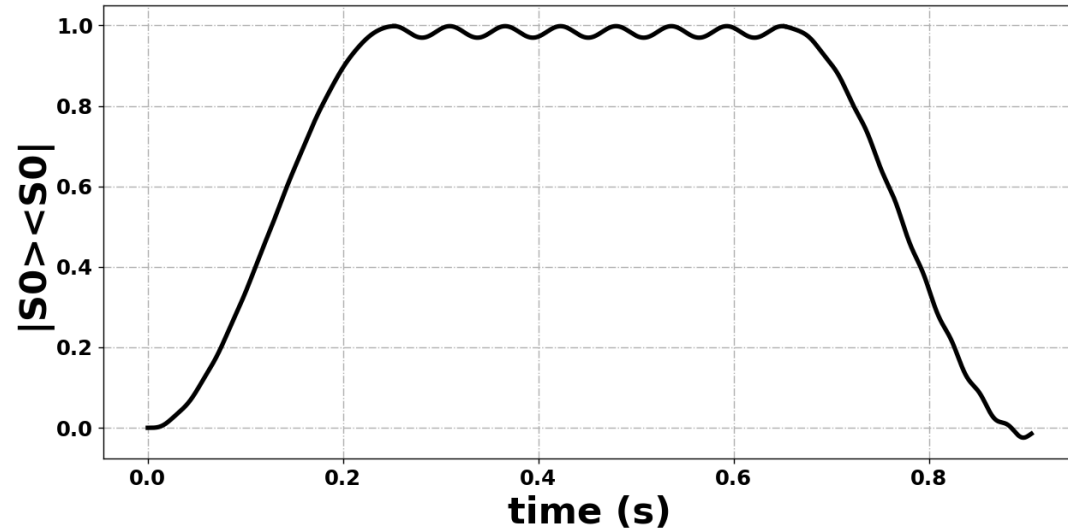
```
1 fig, fourier = System.PlottingMultimodeAnalyzer(t,freq,signal,spectrum)
```



# Simulation of specialized NMR Experiments: SLIC (Avoided Crossing)

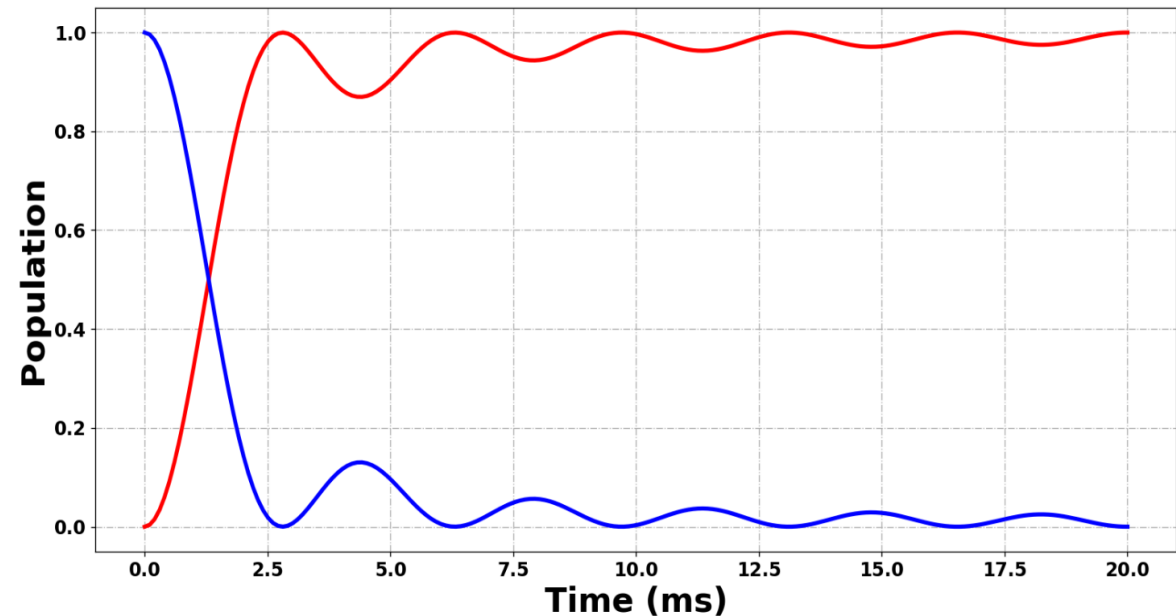
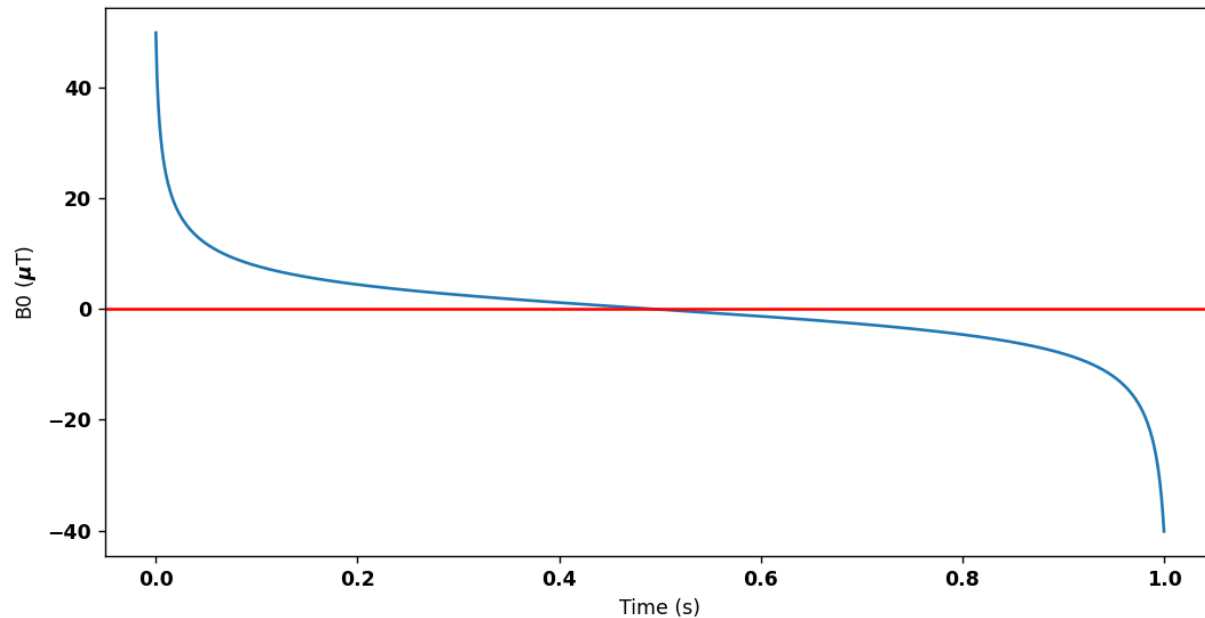


# Simulation of specialized NMR Experiments: SLIC (DeVience PRL)



# Simulation of specialized NMR Experiments: FIRE – dDNP (Stern - JACS)

- Field Inversion Results in Enhancement
- Constant Adiabatic Inversion
- $^1\text{H}$  to  $^{13}\text{C}$



# Future

- Sparse Matrices
- Shaped Pulses
- Average Hamiltonian and Floquet Theory
- Anything I see interesting

# Feedback and Suggestions: LinkedIn, Twitter, Email



**Mohamed Sabha** @ma\_sabba · Jul 22

I fully support Vineeth's lovely work on implementing magnetic resonance simulations using code written from scratch, and am impressed by the breadth (which ranges from classic exps i.e. INEPT/INADEQUATE to more specialist sequences like SLIC).

Good stuff 🤔



**Vineeth Thalakottoor** @VThalakottoor · Jul 22

Dear Lindblad, I am coming for you; I am done with Redfield. See the cool features implemented and to implement in PyOR at [github.com/VThalakottoor/...](https://github.com/VThalakottoor/...) Check the tutorial for the examples. If you like to add any other features and if you see mistakes, please write in the comments.



13



931



**Rudraksha Dutta Majumdar, Ph.D.** (He/Him) · 1st

1mo ...

Sr. NMR Scientist

Very interesting! I would be definitely interested in taking it for a spin once you release it on GitHub. do you have or planning to have support for arbitrary RF waveforms in the pulse sequences? I have a collection of Python scripts for shaped RF pulse generation that I am happy to contribute. Includes shape families such as BURP, SNOB, E-family, SLR, Gaussian Cascade, Adiabatic etc

Like | Reply · 1 Reply



**Quentin Stern** · 1st

5d ...

Postdoctoral researcher at the Northweste...

That's really cool to see that you choose our work as an example for your tutorial 😊 this project is probably the neatest I've worked on so far, where theory and experiment perfectly met. Congrats for your beautiful work with

Love · 🍷🍷🍷 3 | Reply

# Acknowledgement

Daniel Abergel and ANR (ANR-22-CE29-0006-01–DynNonlinPol)

**Anbe Sivam**