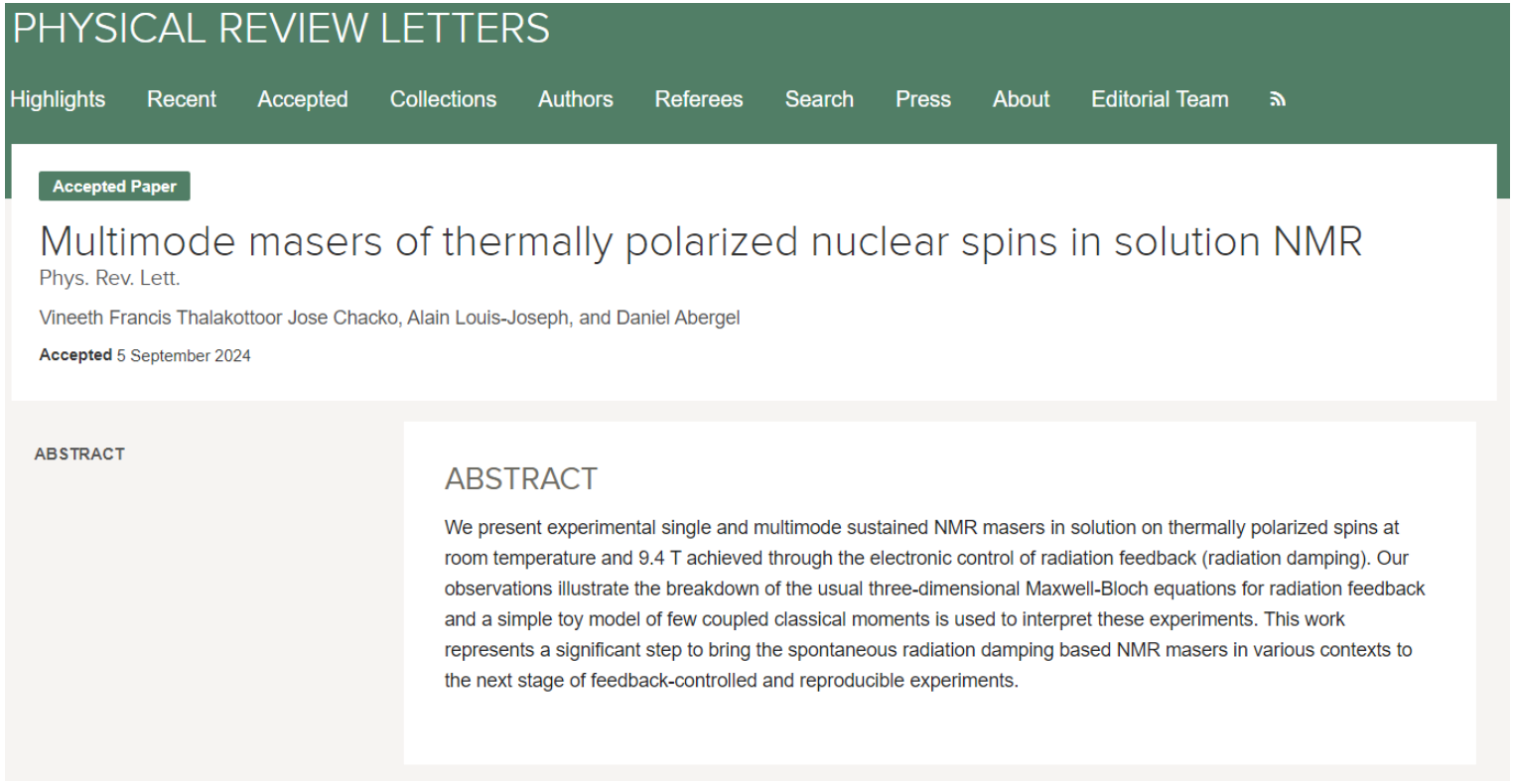
PyOR: A versatile NMR Simulator

Beta version: Jeener-B-24.08.24 (available on GitHub)

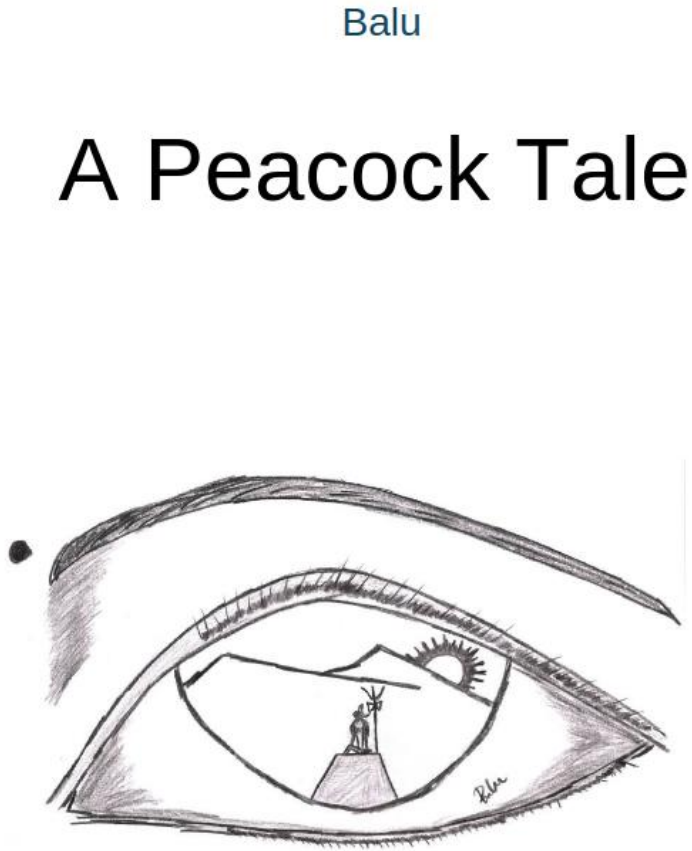
Vineeth Francis Thalakottoor Jose Chacko

LBM, UMR 7203, ENS-SU, Group Meeting

Let's begin with good news



A dream come true, I never thought it would happen so soon.

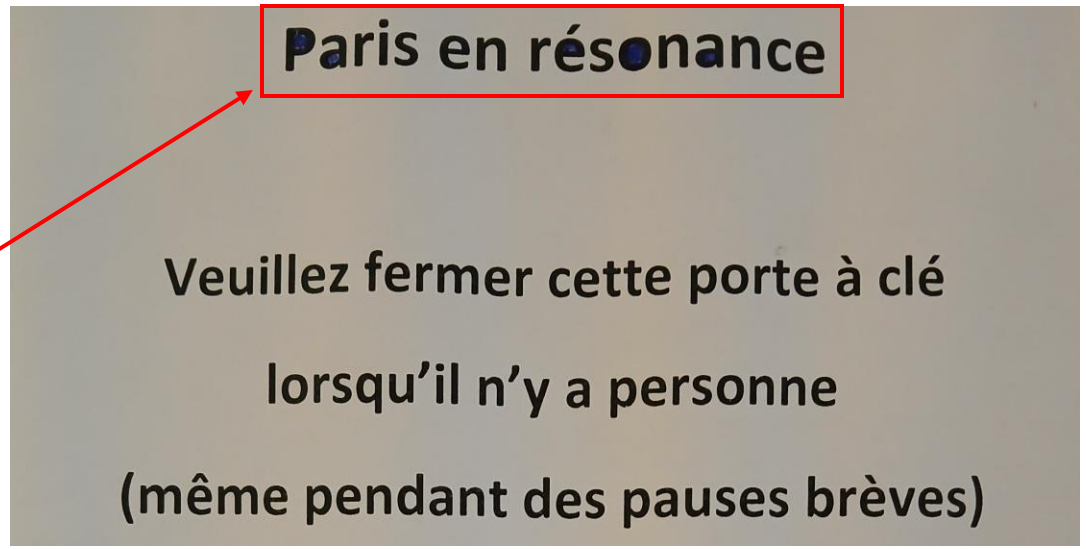


My Book

PyOR

"Everybody can simulate NMR"

A notice in the NMR lab at ENS, Paris



- **Python On Resonance**
- A **collection of Python functions** to simulate NMR spin physics numerically (~ 2500 lines)
- Versatile: From **basic to specialized** NMR experiments
- **Beginners** with basic knowledge of *matrices*, *spin operators* and *Python programming* ***(For anyone interested, I'd be happy to offer a tutorial session.)***

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)
  - Removed in beta version
- *User can easily modify the source code according to their needs*

Why develop PyOR when many NMR simulation packages are available? Why reinvent the wheel?

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

My attitude after PyOR

Why choose Python?

- It is **free**
- **Easy** to learn
- **A lot of packages**: NumPy, SciPy, SymPy, Matplotlib, …
- Great **online** support
- ***I support Free Software and open source packages***

The origin of PyOR

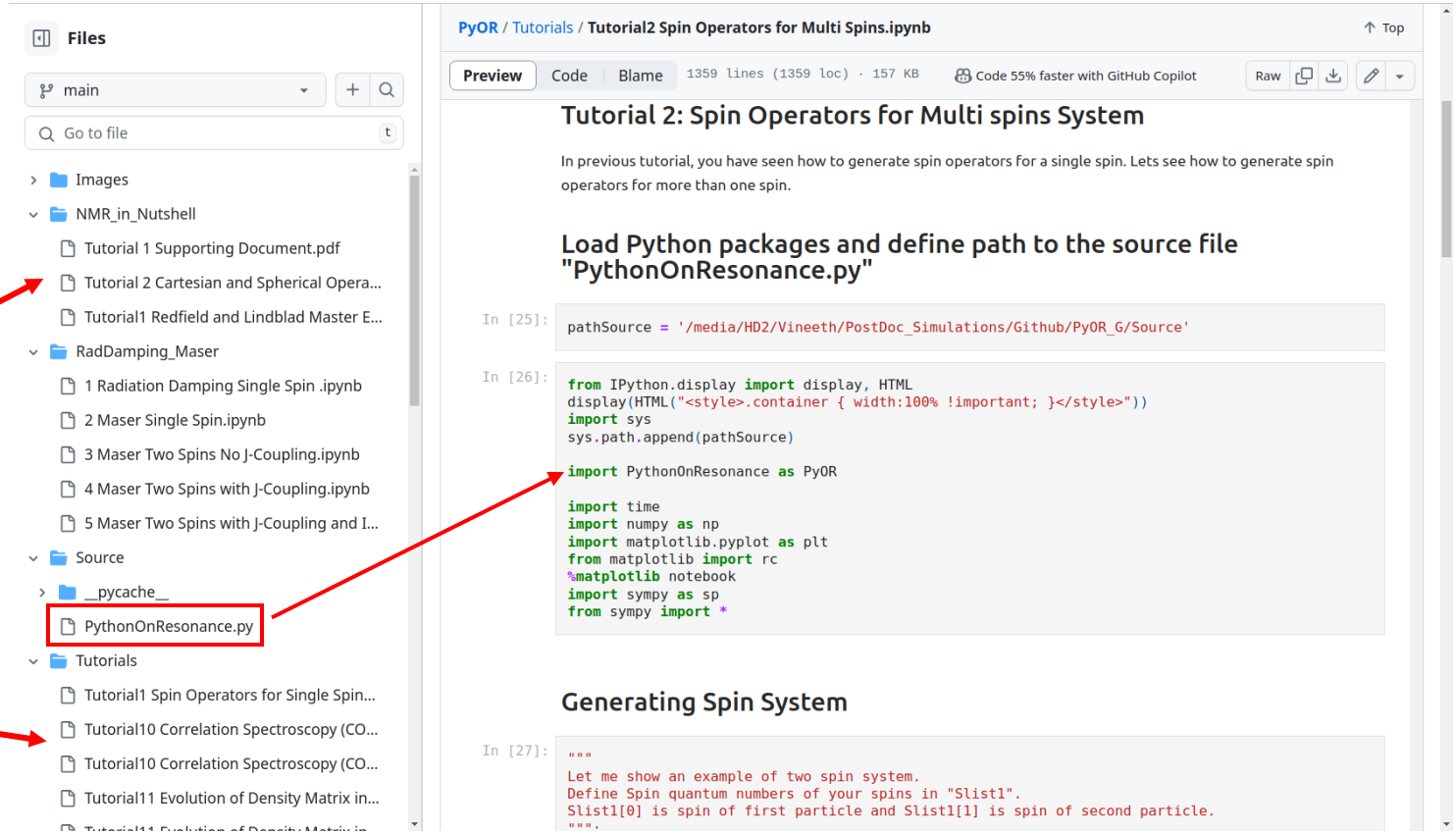
- I was curious to understand "**Avoided Crossing**" and "**SLIC**"
  - I wrote a general a general way to create spin operator (PhD)
  - And did not proceed much – "**A silly Conflict with definition of Density Matrix**"
  - The Program was named "**PyOR**", Python On Resonance.

- To understand **multi-mode MASER** from spins which are J-Coupled
  - Restarted the project (One year after the completion of my PhD)
  - **"Conflict with the definition of Density Matrix resolved."**
  - Implemented Radiation Damping and Maser

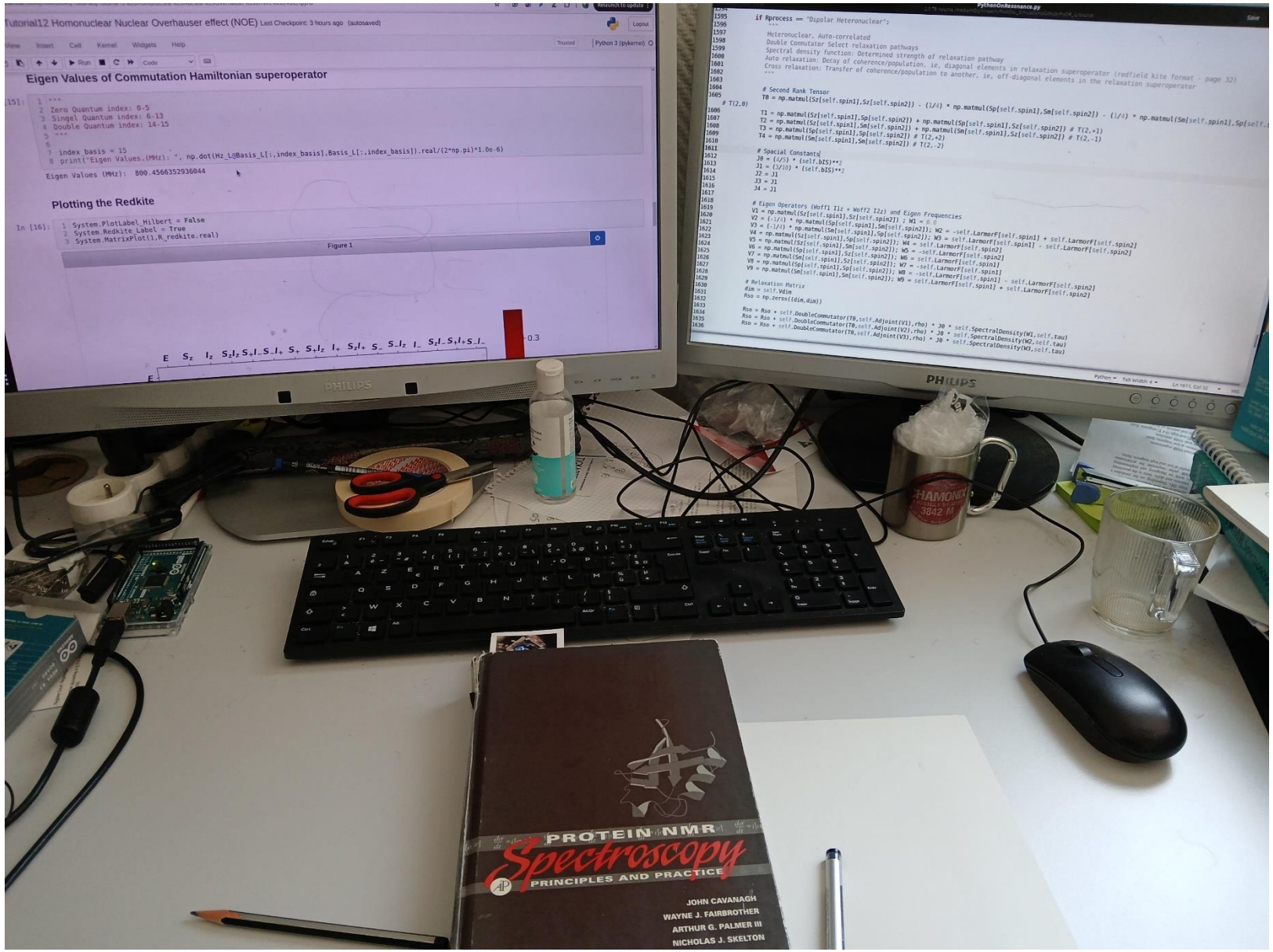
- ***"Solitude" in academia (working alone). Identity Crisis. Job??***
  - ***Show the NMR community what I can do.***
  - **What is going on** in NMR community other than MASER?
  - ***PyOR became my journey to explore the "spin physics involved in NMR."***

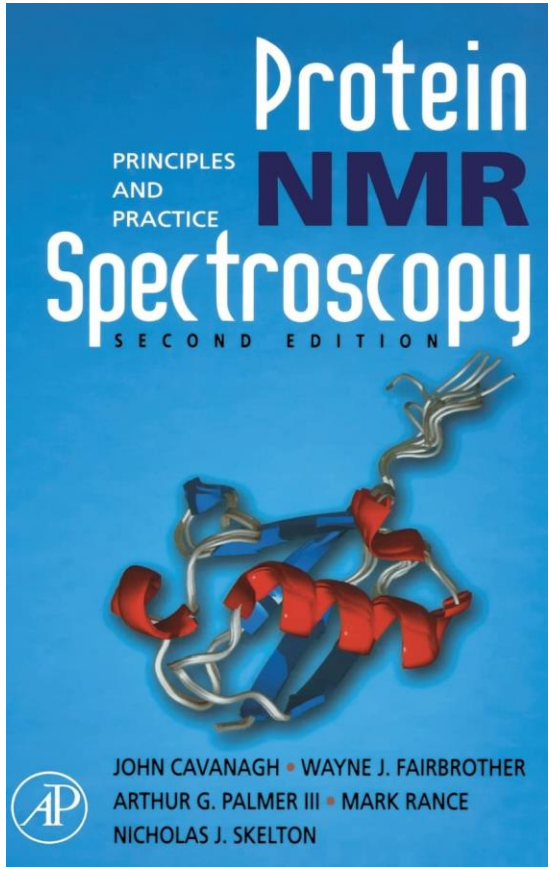
How to use PyOR?

- Install Python in your computer (I prefer **Anaconda Python Distribution**)
- PyOR **Source code**: PythonOnResonance.py
- **Jupyter Notebook**
- PyOR and tutorials are in the **Github**
  - version: Jeener-B-24.08.24
  - https://github.com/VThalakottoor/PyOR_beta
  - Descriptive
  - Programming
- ***Modify the source code according to your need***



Final requirement for PyOR: An NMR Book





I found my Book to start learning NMR, What is yours?

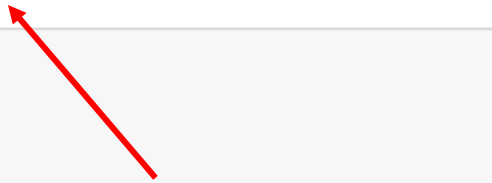
Let us see how to use PyOR

First, import PyOR

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

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

```
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 *
```



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/d
""";
```

Single Spin

```
"""
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]
""";
```

Multiple Spins

Generate Spin Operators

```
"""
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)
```

Heart of PyOR (First Born)

Documentation

Any PyOR functions

```
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
    -----
    nill

    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

```
"""
Matrix represenatation (Sympy)
In Sx[0], 0 means the index of the spin (0th spin or first spin).
""";
Matrix(Sx[0])
```

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

```
Matrix(Sy[0])
```

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

```
Matrix(Sz[0])
```

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

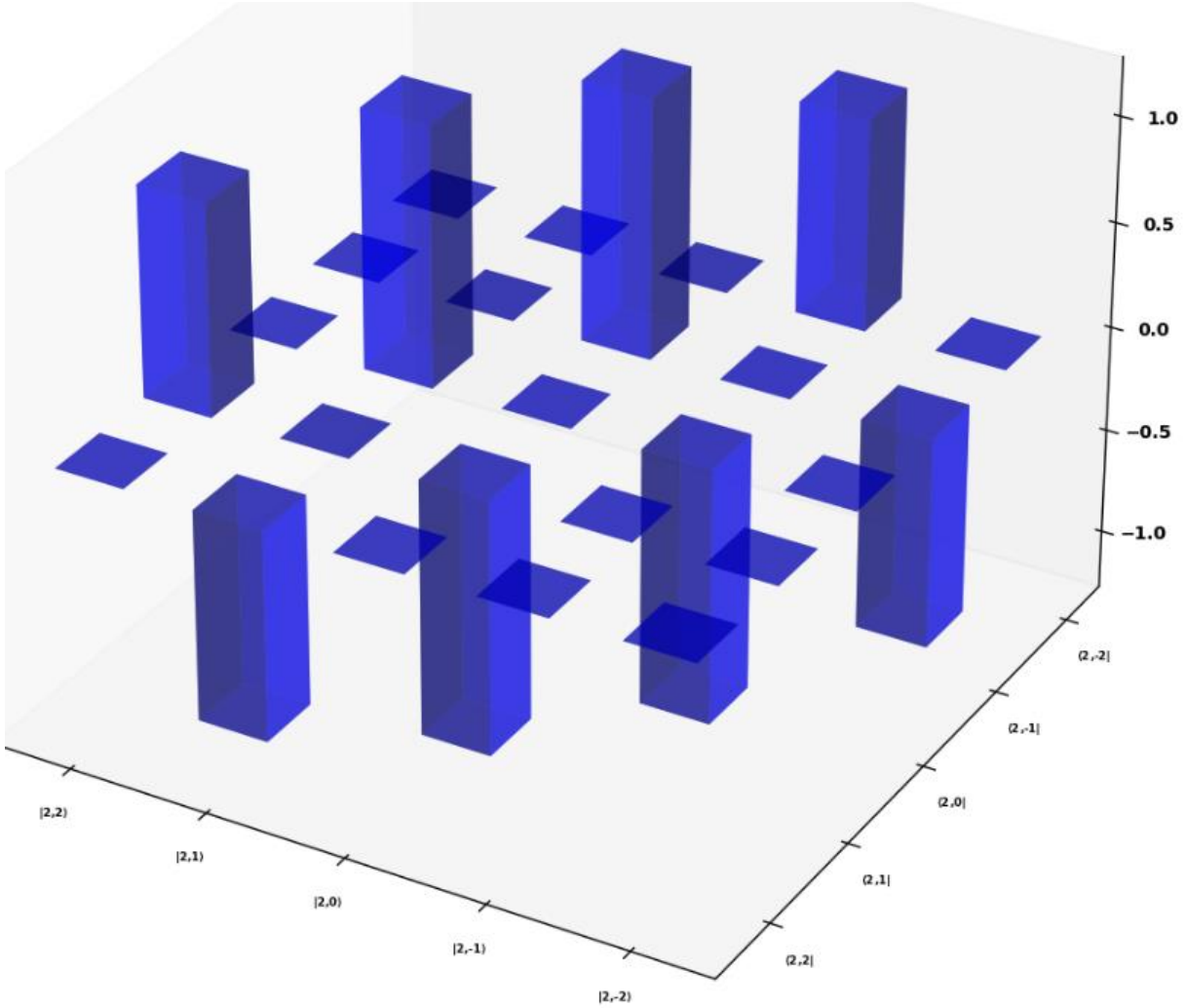
```
"""
Matrix represenatation (Sympy)
In Sx[0], 0 means the index of the first spin.
And Sx[1], 1 means the index of the second spin.
""";
Matrix(Sz[0]) # Spin operator Sz of first spin
```

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

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

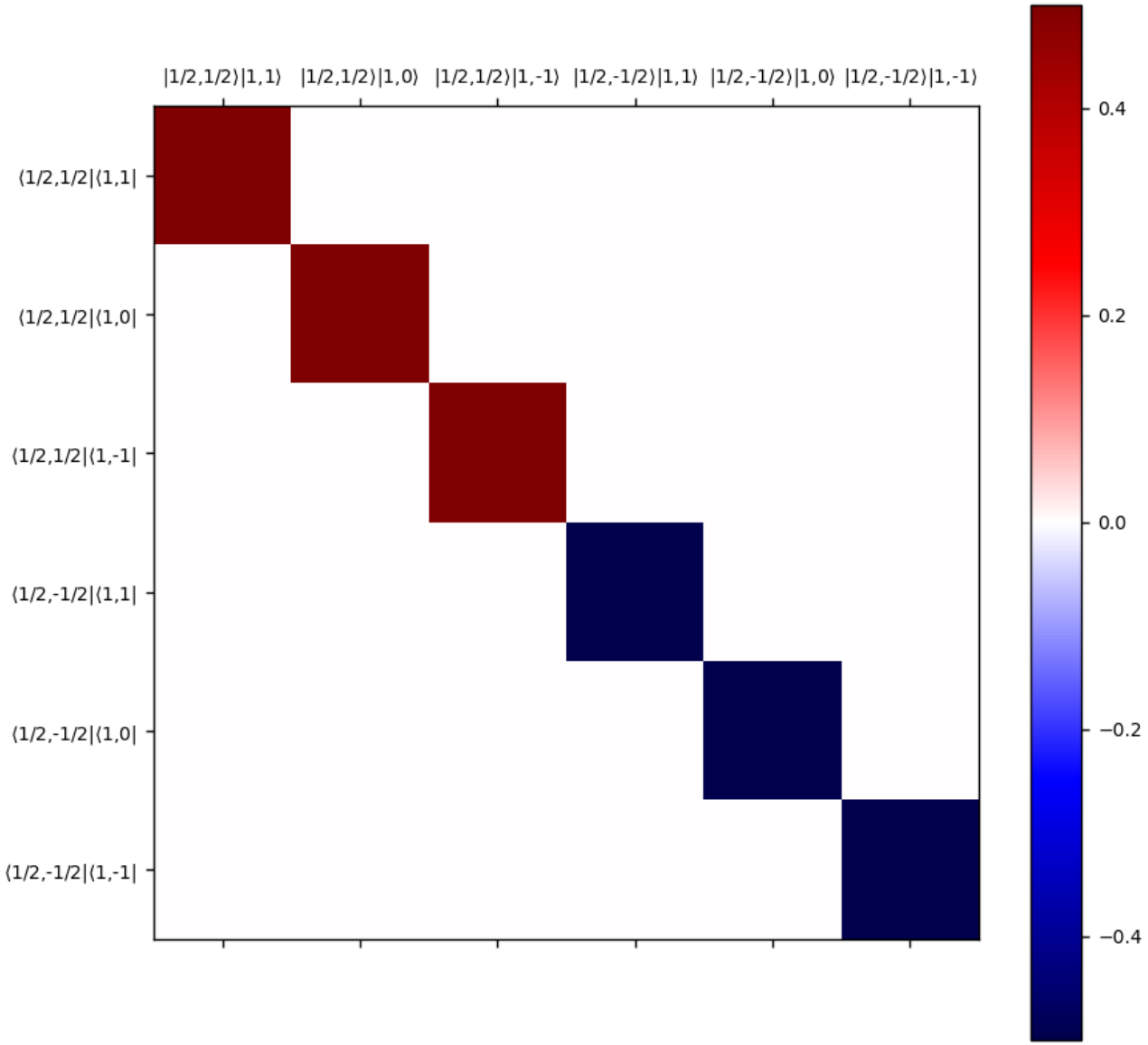
$$\begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1.0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.0 \end{bmatrix}$$

Matrix Representations




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

Matrix Representations




```
System.MatrixPlot(1,Sz[0].real)
```

Pre-defined Constants

```
1 print("Planck constant, h = ",System.pl)
2 print("Planck constant, hbar = ",System.hbar)
3 print("Permitivity 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
Permitivity 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)

```
"""
Gyromagnetic Ratio
Gamma = [Gyromagnetic Ratio spin 1, Gyromagnetic Ratio spin 1, ...]
""";
Gamma = [System.gammaH1]

"""
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 = [20] # Offset frequency in Hz

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

```
Larmor Frequency in MHz:  [-400.22803765]
```

Hamiltonian

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

Zeeman Hamiltonian (Rotating Frame)

```
"""
"OmegaRF" is list of rotating frame frequencies
""";
OmegaRF = [-System.gammaH1*B0]

"""
Hamiltonian in the rotating frame
""";
Hzr = System.Zeeman_RotFrame(LarmorF, Sz, OmegaRF)
```

B1 Hamiltonian

```
"""
So we have a spin half particle sitting at static magnetic
When a RF field is applied the magnetic dipole of the spin
The energy of the particle in this case is given by B1 Fie
""";

"""
List of RF amplitude (Hz) or Nutation frequency, "Omega1"
""";
Omega1 = [100] # Hz

"""
List of RF signal phase in degree
""";
Omega1Phase = [0] # deg

"""
B1 field hamiltonian
""";
HzB1 = System.Zeeman_B1(Sx,Sy,Omega1,Omega1Phase)
```

J Coupling Hamiltonian

```
"""
Define J Coupling between each spins, Jlist[0][3] means J coupling between 1st spin and 4th spin.
"""

Jlist = np.zeros((len(Slist1),len(Slist1)))
Jlist[0][3] = 7
Jlist[0][4] = 7
Jlist[1][3] = 7
Jlist[1][4] = 7
Jlist[2][3] = 7
Jlist[2][4] = 7
Jlist[3][5] = 5
Jlist[4][5] = 5

Hj = System.Jcoupling(Jlist,Sx,Sy,Sz)
```

```
"""
Define J Coupling between each spins, Jlist[0
"""

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

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

Weak Coupling

Initialize Density Matrix

```
"""
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_FreqUnitsTOEn
    HT_approx = False # High Temperature App
    T = 300 # Temperature in Kelvin
    rho_in = System.EqulibriumDensityMatrix(
    rhoeq = rho_in.copy()
else:
    rho_in = np.sum(Sz,axis=0) # Initial Den
    rhoeq = np.sum(Sz,axis=0)  # Equlibrium
    print("Trace of density metrix = ", np.t
```

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

```
"""
Second Case: Initial Desnity Matrix at Thermal Equlibrium
""";

Thermal_DensMatrix = True

if Thermal_DensMatrix:
    Hz_EnUnit = System.Convert_FreqUnitsTOEnergy(Hz)
    HT_approx = False # High Temperature Approximation is False
    T = 1.2 # 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 =  1.0
```

Liouville-von Neumann Equation

Hilbert Space ⟷ **Swap** ⟷ **Liouville Space**

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

$$\rho(t) = e^{iH_0 t}\rho(0)e^{-iH_0 t}$$

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

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

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

$$\tilde{\rho}(t) = e^{-i\hat{H}_0 t}\tilde{\rho}(0)$$

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

Basis Operators: 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:

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

Basis Operators

```
"""
Spherical Operator Basis for single spin half particle
let S = 1/2
""";

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)]
```

Single Spin Half Particle

Product Operators (Two spin half)

```
"""
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
"""

Coh_order = [0,-1, 0, 1]
Dic_1 = ["E1 ","Im ","Iz ","Ip "]
Dic_2 = ["E2 ","Sm ","Sz ","Sp "]
Dic_3 = ["E3 ","Rm ","Rz ","Rp "]

"""
Sorting options: 'normal', 'negative to positive', 'zero to high'
"""

sort = 'negative to positive'
indexing = False
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
```

Three Single Spin Half Particle

```
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]
```

```
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 ']
```

Product Operators (Three spin half)

```
"""
Product operator basis for three spin half particles
"""

indexing = True
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,
```

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

```
Coherence order (Three spin half particles):  [-3, -2, -2, -2, -2, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3]
```

```
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]', '
[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 R
'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
'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
'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
'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
'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
'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
'Ip Sp E3 [61]', 'Ip Sp Rz [62]', 'Ip Sp Rp [63]']
```

Product Operator, $I\_S\_R\_$

```
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 Product Operators

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

```
"""
Dictionary
""";
dic
```

```
['E$', 'Ix', 'Iy$', 'Iz']
```

Single Spin Half

```
'''
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, dic = System.TwoSpinOP(Sx,Sy,Sz,Sp,Sm,Basis)
```

Two Spin Half

Projection of Density Matrix

```
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 Zeeman States (Three spin half)

```
"""
Basis Ket
""";
Kets = System.Basis_Ket()
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>']
```

```
"""
Basis Bra
""";
Bras = System.Basis_Bra()
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 Transformation (Operator)

```
"""
Now lets see how to get the eigen vectors of the Zeman Hamiltonian (lab frame)
""";
B_Z = System.ZBasis_H(Hz)
```

Zeeman Basis

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

```
"""
Singlet Triplet Basis
""";
B_ST = System.STBasis(B_Z)
```

Basis: T_-, T_0, T_+, S_0

Singlet Triplet Basis

```
"""
Basis Transformation
""";
U = System.Transform_StateBasis(B_Z,B_ST)
Matrix(U)
```

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

Transformation Matrix

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

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

Zeeman Basis

Change Basis of any Operators

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

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

Singlet Triplet Basis

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

Change Basis of Spin Operators

Hilbert or Liouville ?

Hilbert

Pulse

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

Liouville

Converting to Liouvillian

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

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

Evolution of Density Matrix

```
"""
Options: "No Relaxation", "Phenomenological", "Auto-corre
""";
R1 = None
R2 = None
Rprocess = "Auto-correlated Dipolar Homonuclear"
tau = 10.0e-12
bIS = 30.0e3
System.Relaxation_Constants(R1,R2)
System.Relaxation_Parameters(LarmorF, OmegaRF, tau, bIS)
```

```
"""
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,dt,Npoints,method,Rprocess)
end_time = time.time()
timetaken = end_time - start_time
print("Total time = %s seconds " % (timetaken))
```

Hilbert

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

start_time = time.time()
t, rho_t = System.Evolution_L(rhoeq_L,rho_L,Sx,Sy,Hzr_L - 1j * R_L,dt,Npoints,method)
end_time = time.time()
timetaken = end_time - start_time
print("Total time = %s seconds " % (timetaken))
```

Liouville

Relaxation Super-operator

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

Redfield

```
Rprocess = "Auto-correlated Dipolar Hetronuclear"
tau = 10.0e-12
bIS = 30.0e3
System.Relaxation_Parameters(LarmorF, OmegaRF, tau, bIS)
System.Temperature(T)
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 Hetronuclear"
        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

```
EXP_Z1 = Sz[0]
EXP_Z2 = Sz[1]

t, Mz1 = System.Expectation_H(rho_t,EXP_Z1,dt,Npoints)
t, Mz2 = System.Expectation_H(rho_t,EXP_Z2,dt,Npoints)
```

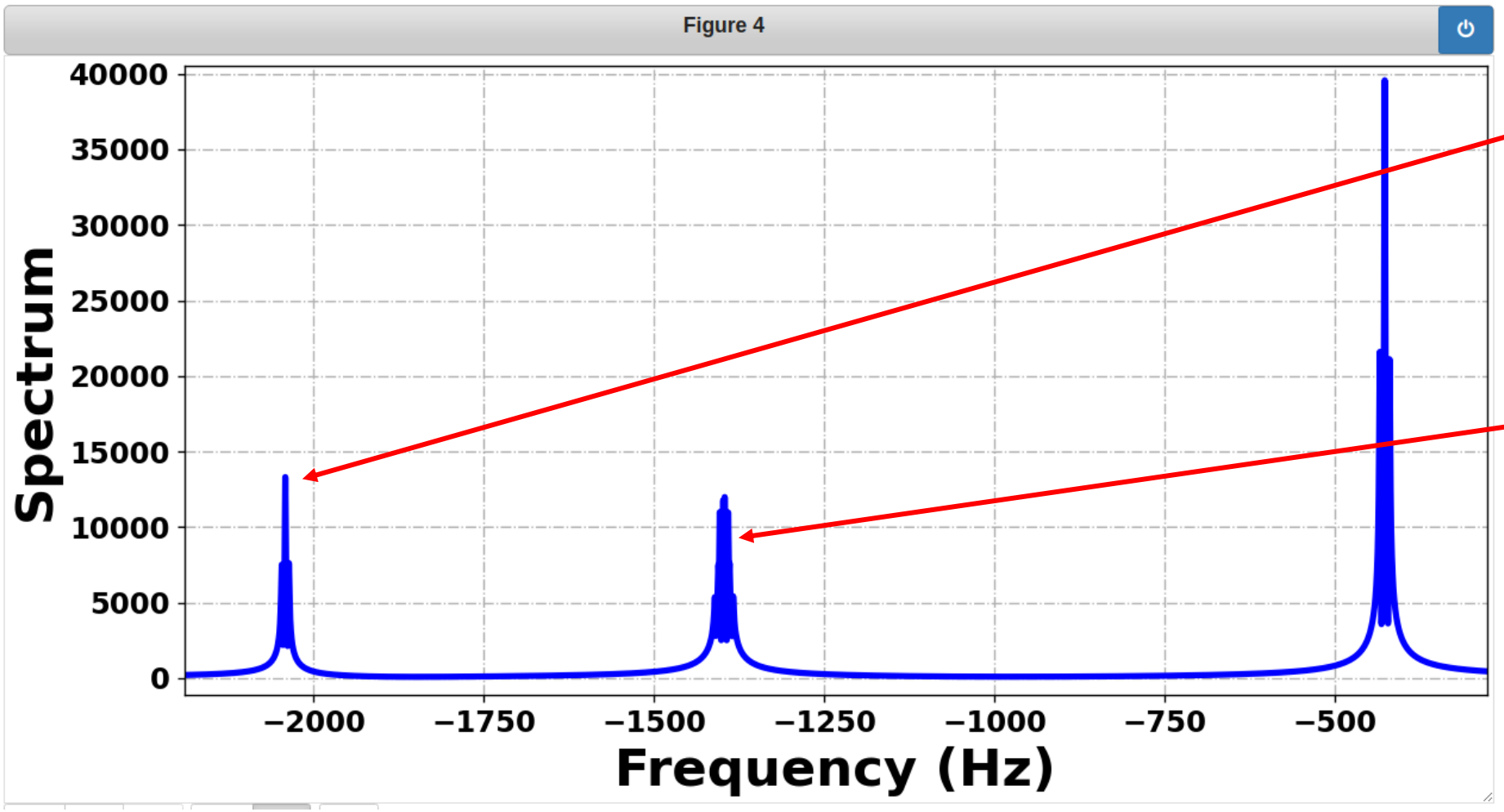
Hilbert

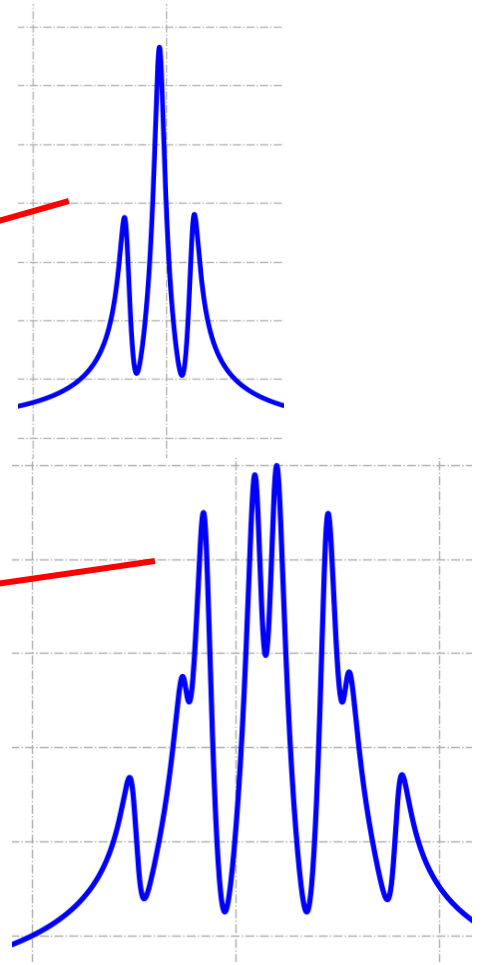
```
EXP_T = np.sum(Sx,axis=0) + 1j * np.sum(Sy,axis=0)
EXP_Z = np.sum(Sz,axis=0)

LEXP_T = System.Detection_L(EXP_T)
LEXP_Z = System.Detection_L(EXP_Z)

t, Mp = System.Expectation_L(rho_t,LEXP_T,dt,Npoints)
t, Mz = System.Expectation_L(rho_t,LEXP_Z,dt,Npoints)
```

Liouville

Simulation Results Visualization:
Plotting (Ethanol Spectra – First Result with PyOR)

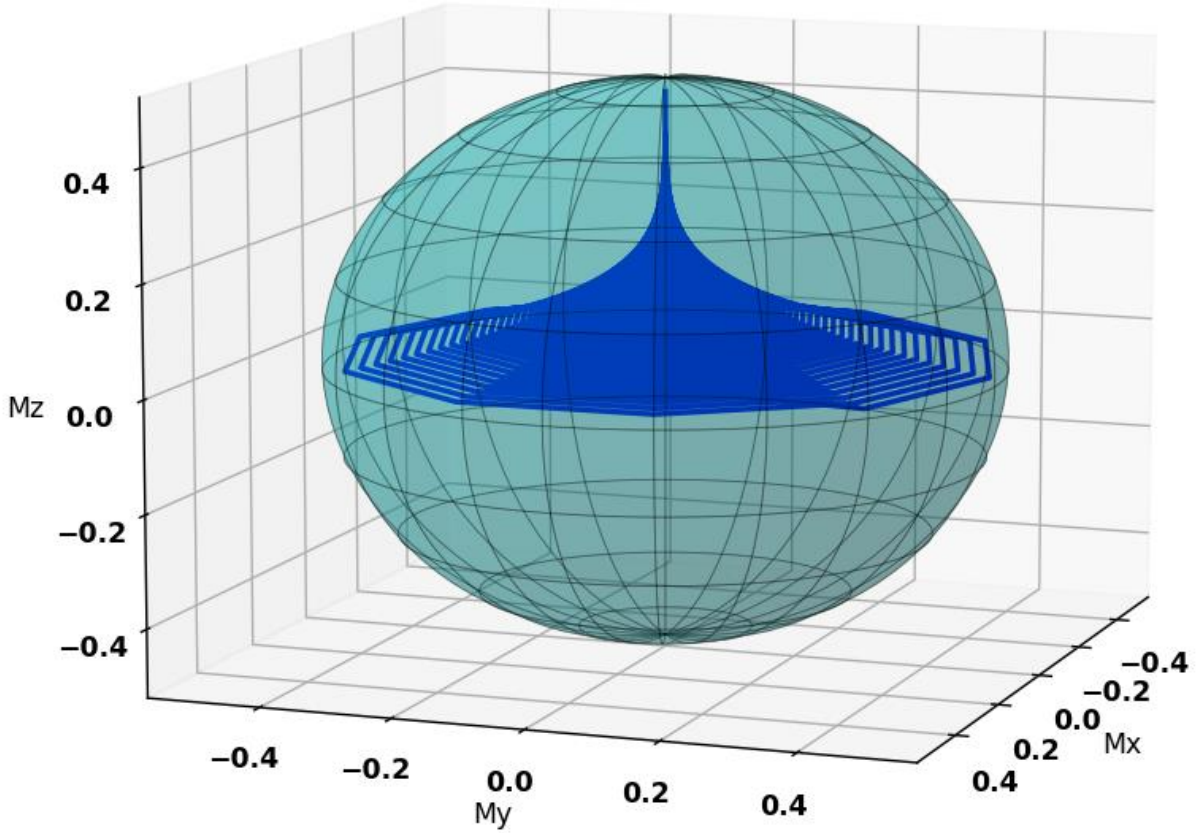
```
1 System.Plotting(4,freq,np.absolute(spectrum),"Frequency (Hz)","Spectrum","blue")
```





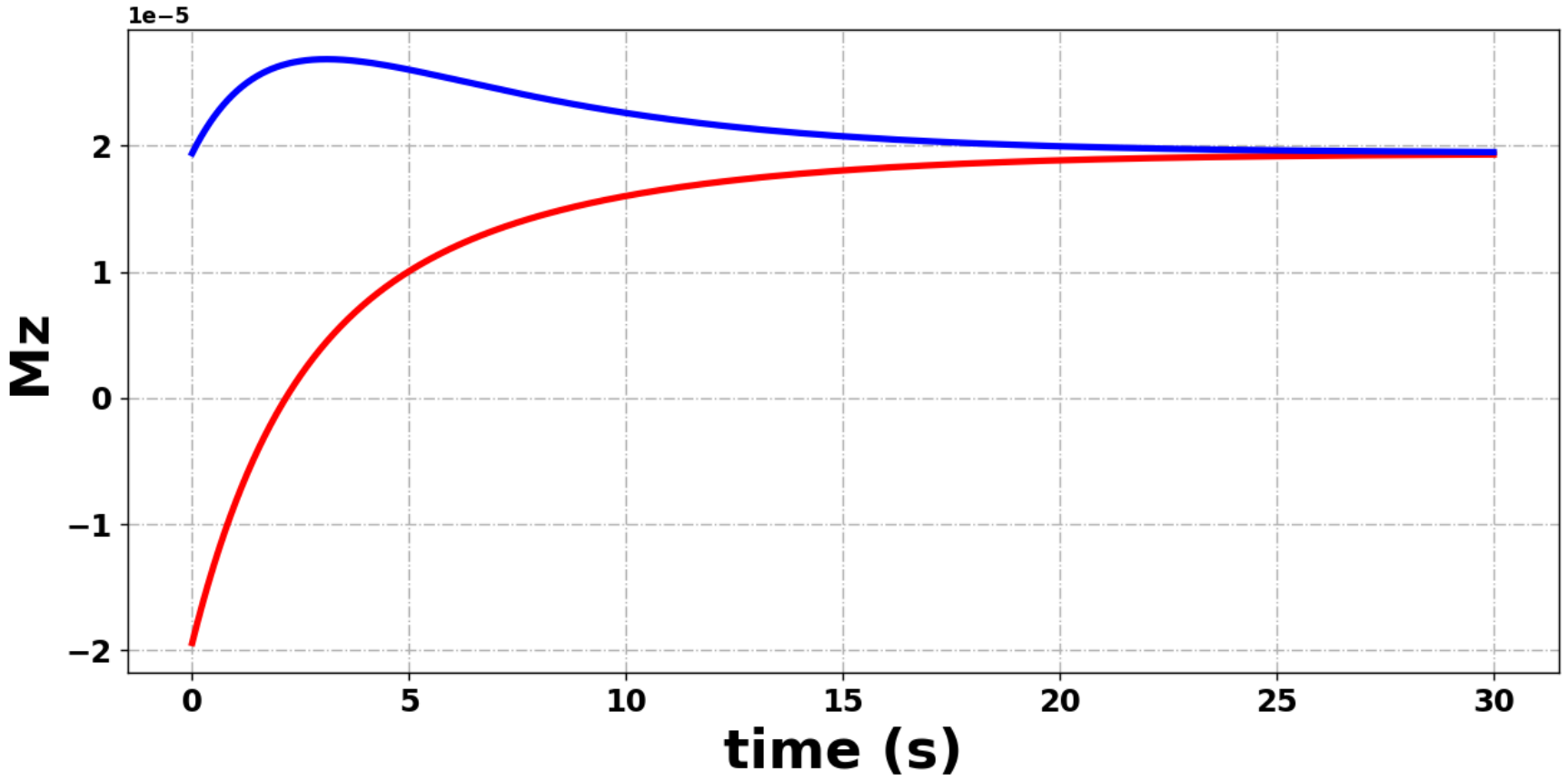
Simulation Results Visualization:
Sphere

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



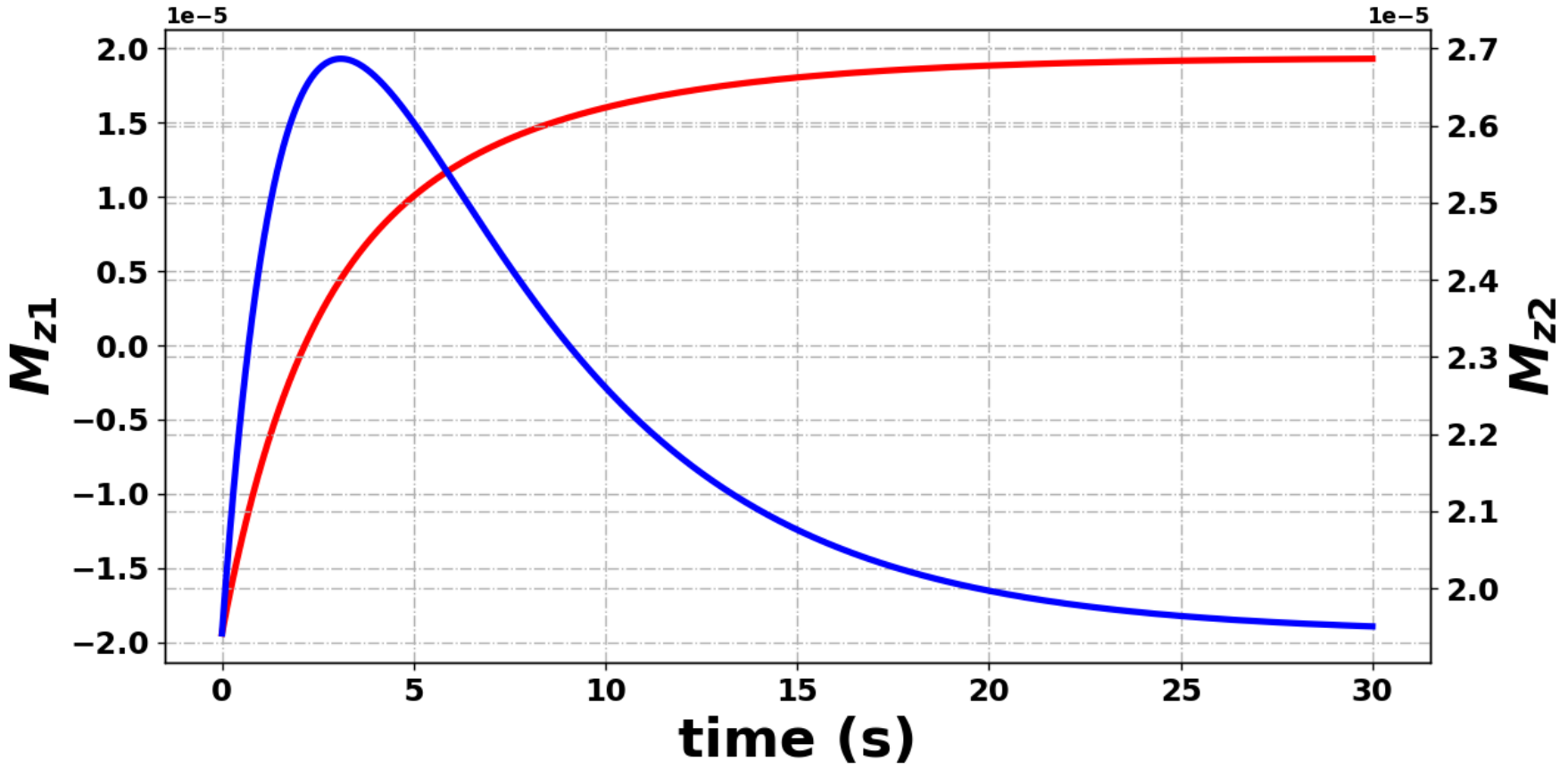
Simulation Results Visualization:
Multiple Plots (NOE Lindblad Homonuclear)

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



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

```
"""
Mz1: Red
Mz2: Blue
"""
System.PlottingTwin(5,t,Mz1,Mz2,"time (s)",r"$M_{z1}$",r"$M_{z2}$","red","Blue")
```

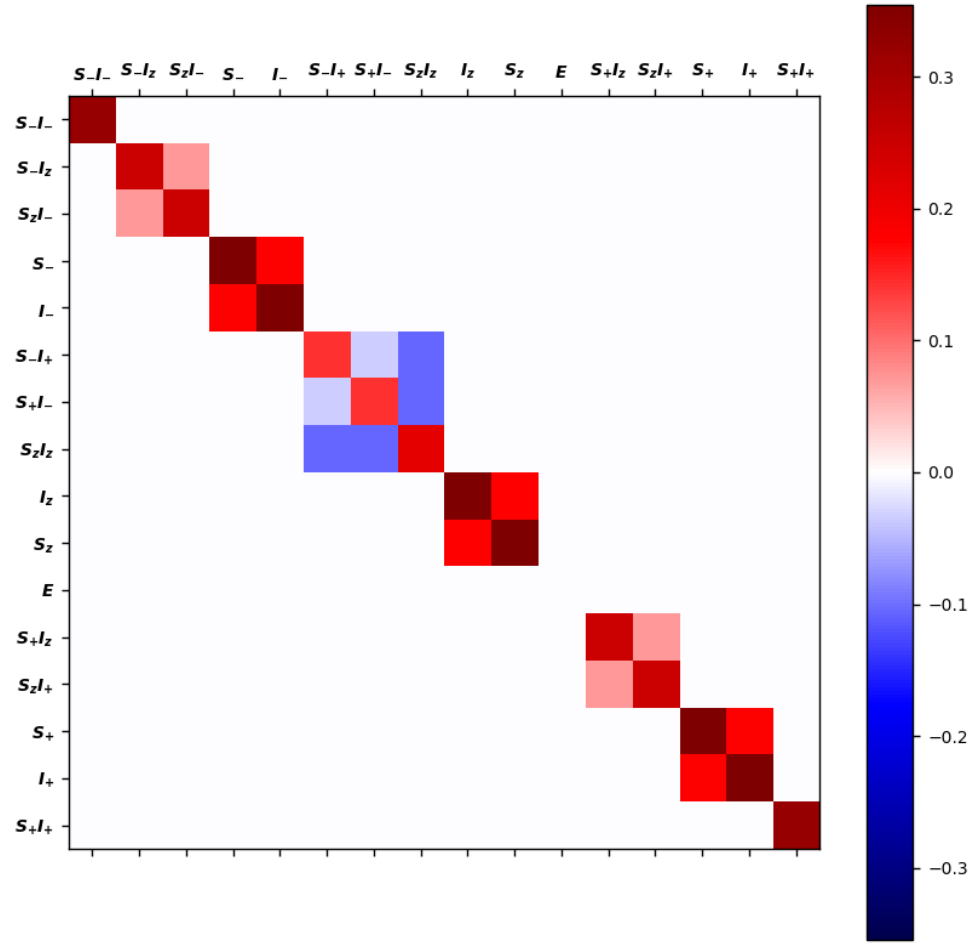


Simulation Results Visualization:
RedKite

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

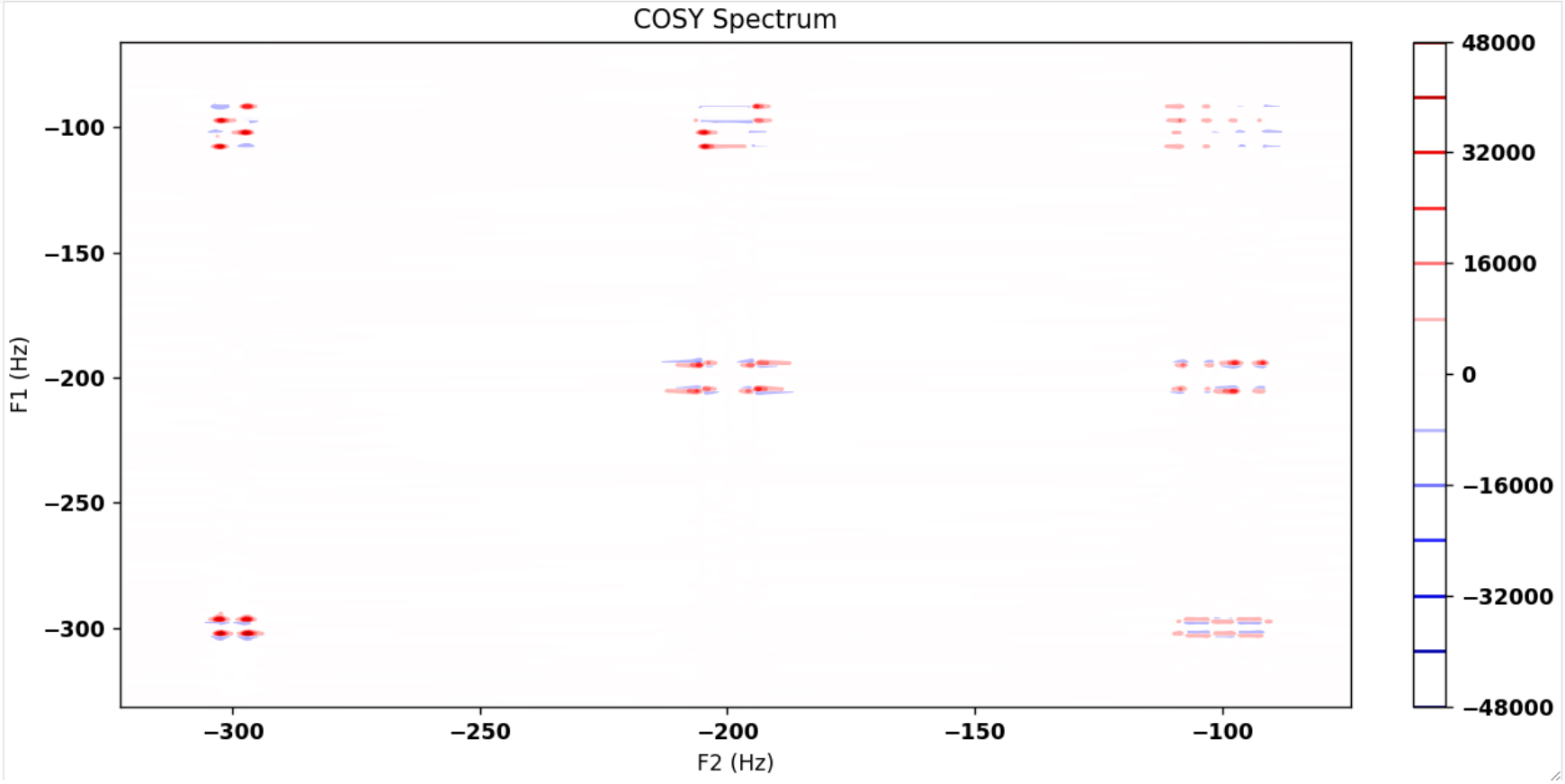
$$S\_I\_, S\_I_z, S_zI\_, S\_, I\_, S\_I_+, S_+I\_, S_zI_z, I_z, S_z, E, S_+I_z, S_zI_+, S_+, I_+, S_+I_+$$

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



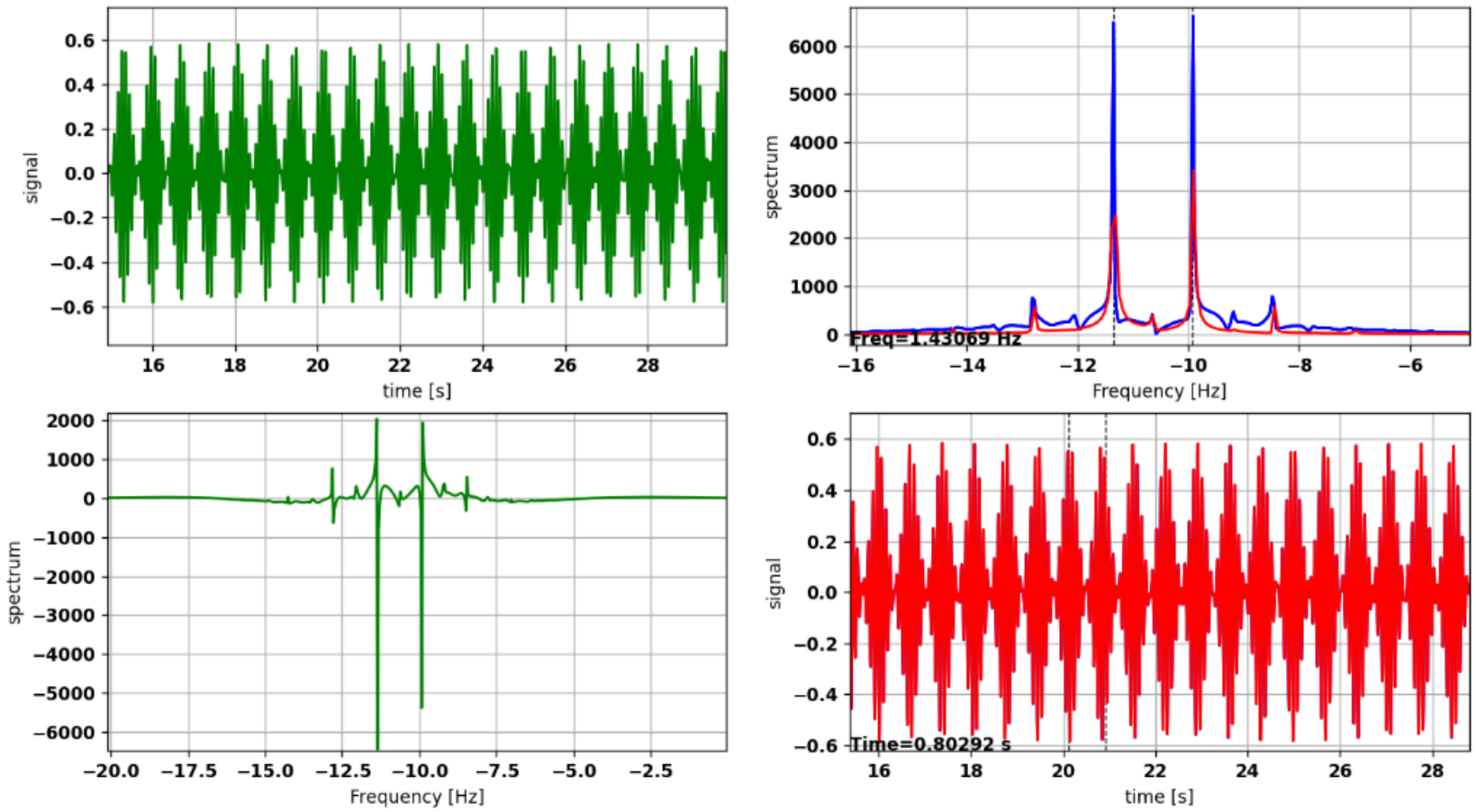
Simulation Results Visualization:
Contour Plotting (COSY)

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

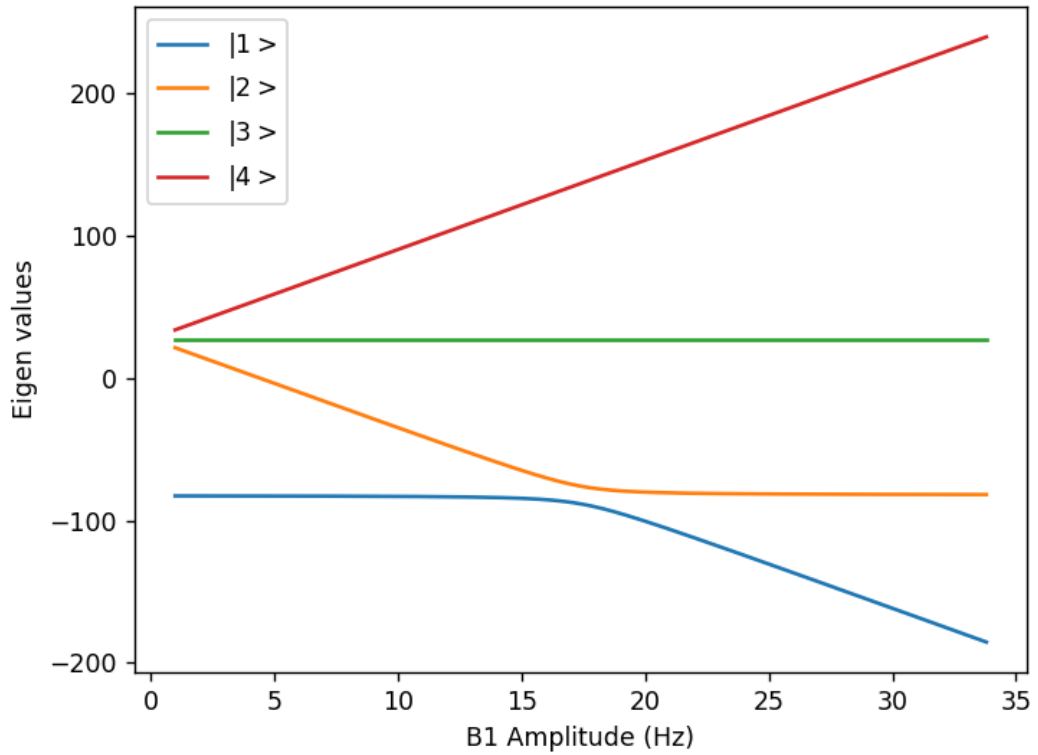


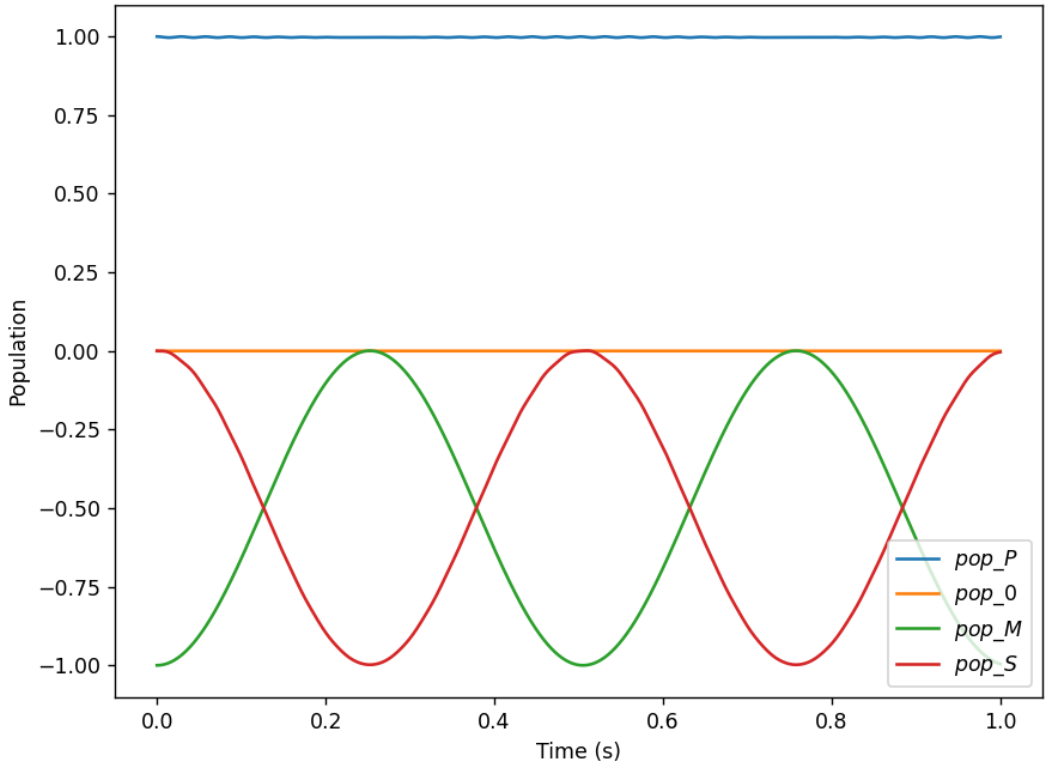
Simulation Results Visualization: Multi-mode Analyzer

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



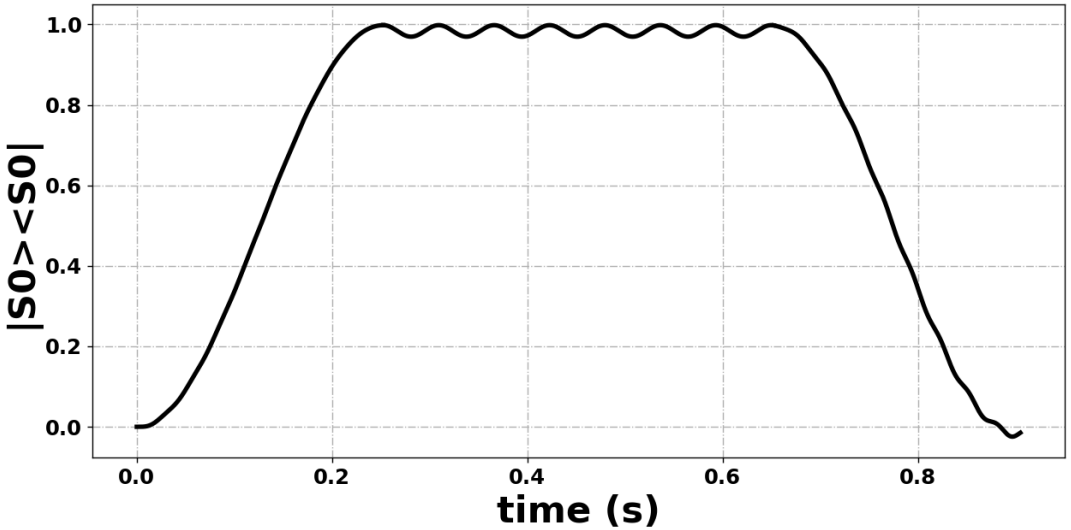
Simulation of specialized NMR Experiments:
SLIC (Avoided Crossing)

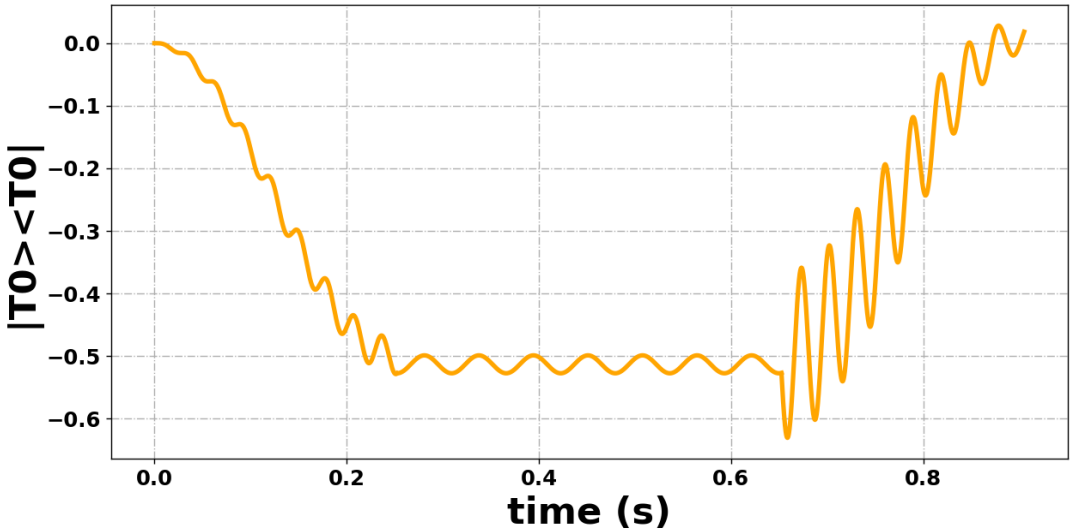


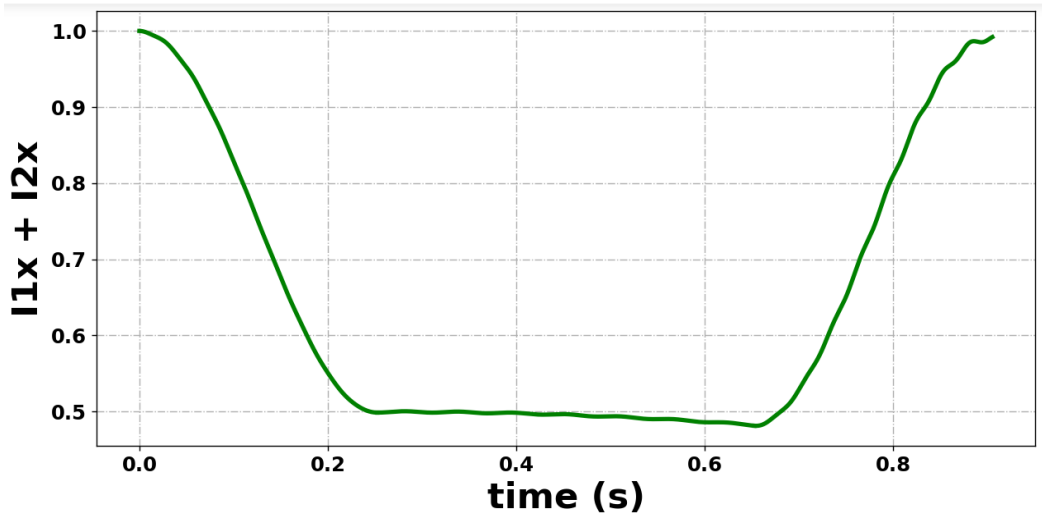


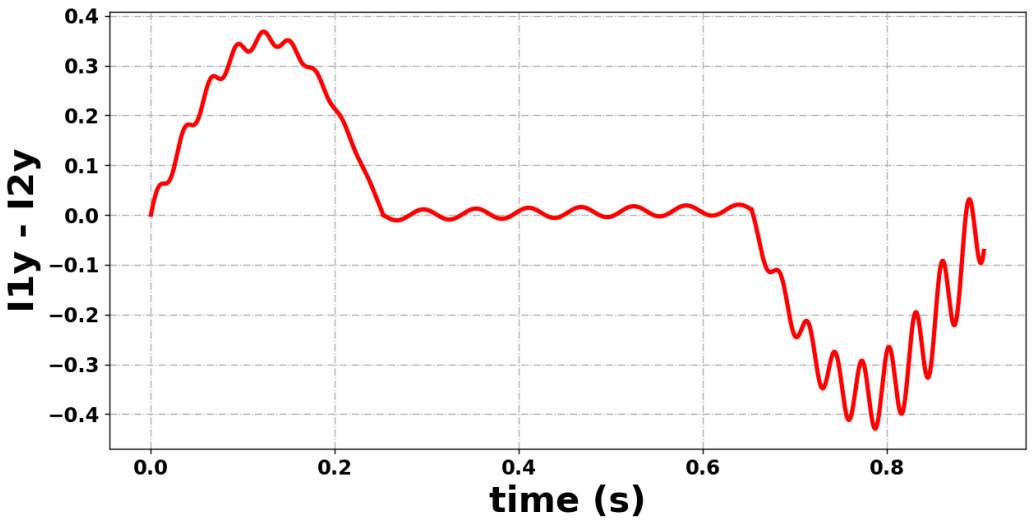
Simulation of specialized NMR Experiments:

SLIC (DeVience PRL)





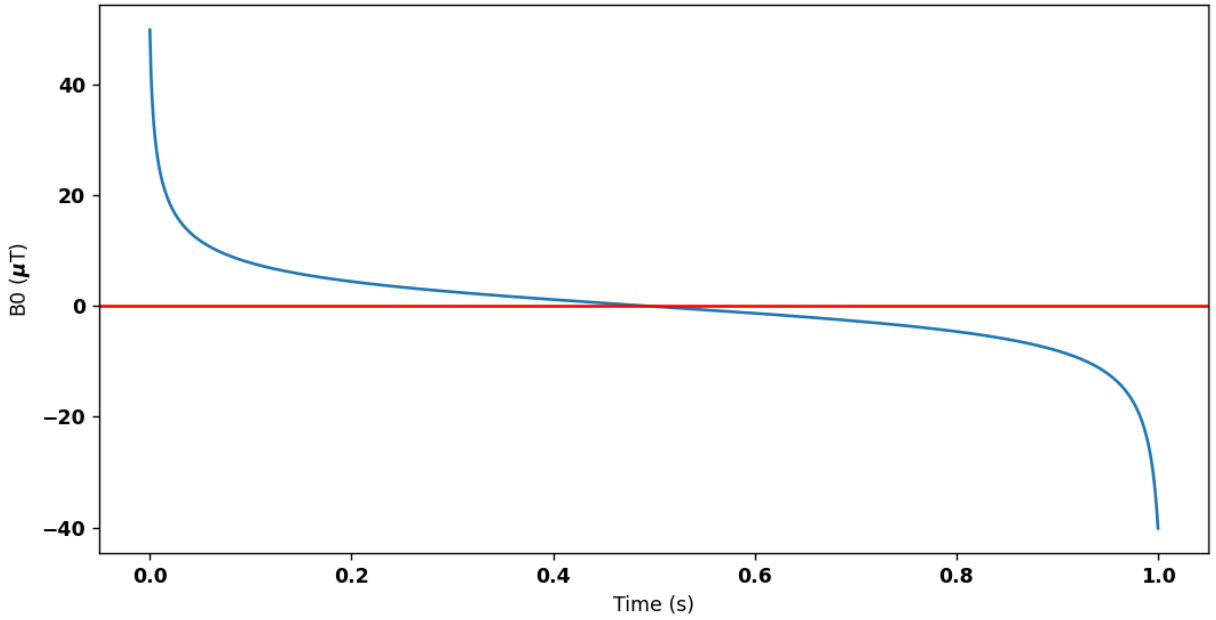


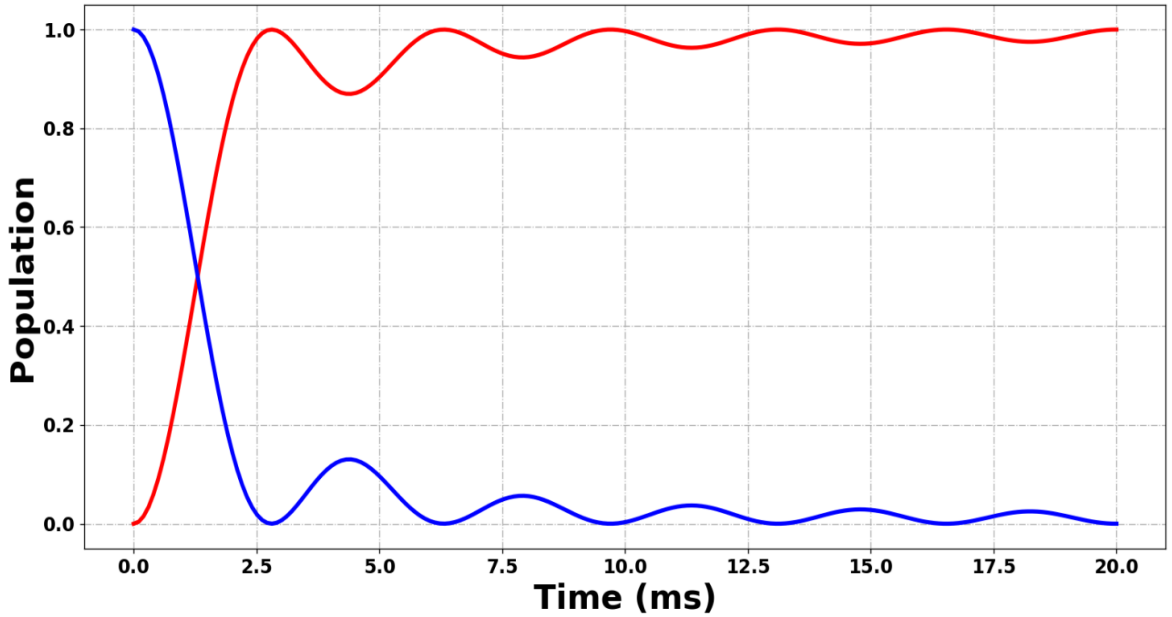


Simulation of specialized NMR Experiments:
FIRE – dDNP (Stern - JACS)

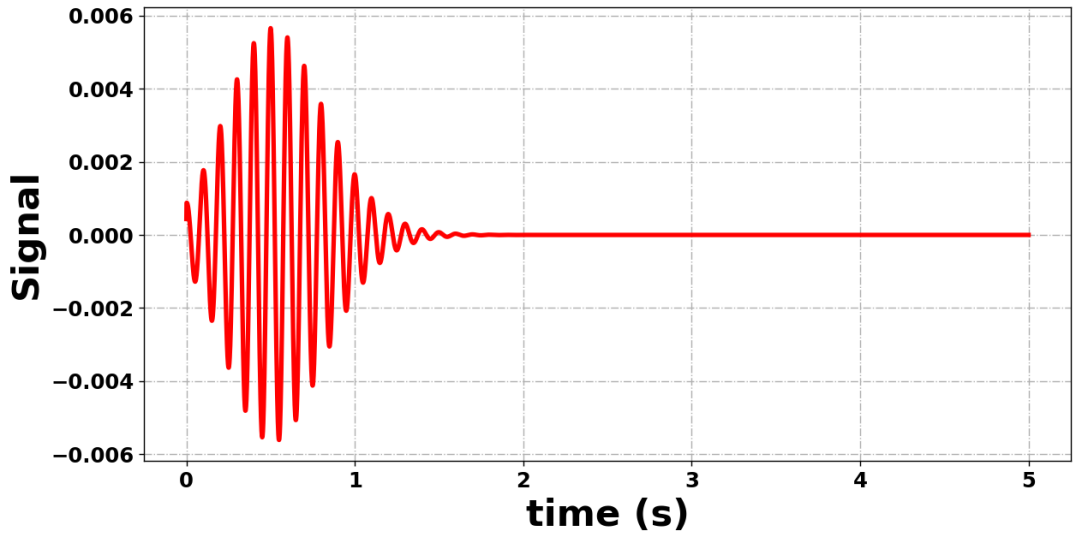
- Field Inversion Results in Enhancement
- Constant Adiabatic Inversion
- 1H to 13 C

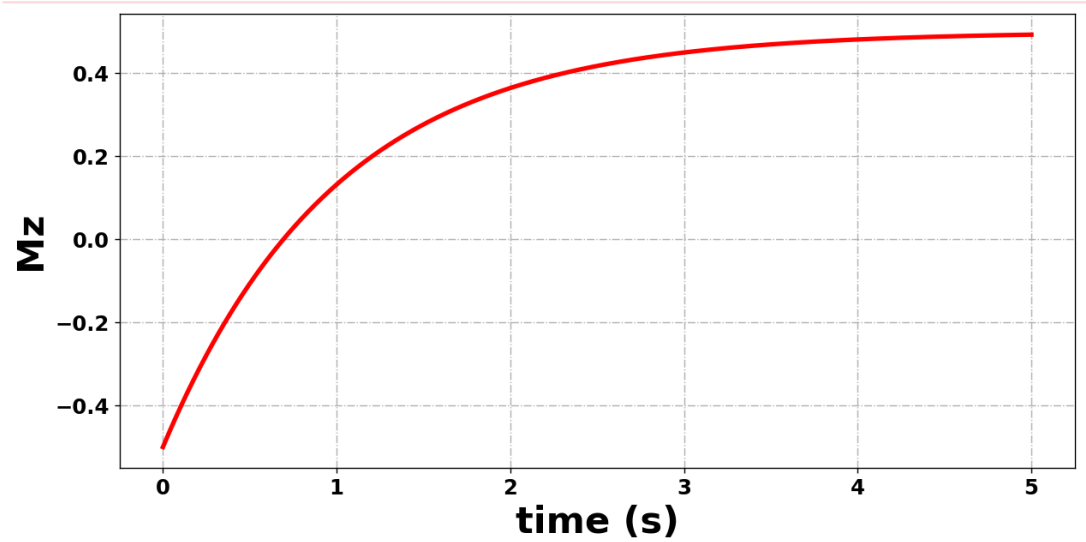
Red: 13C; Blue: 1H





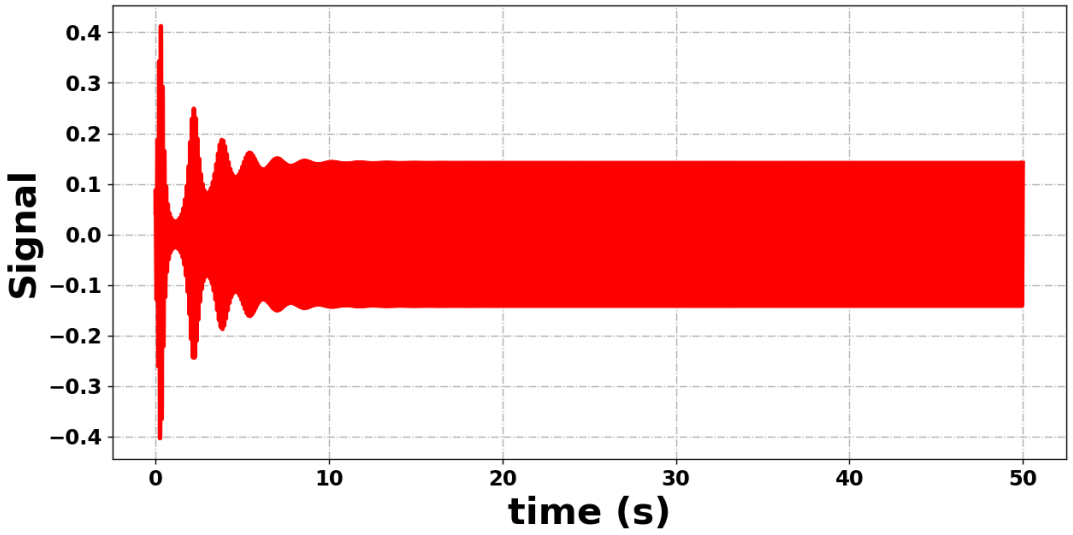
Simulation of specialized NMR Experiments:
Radiation Damping



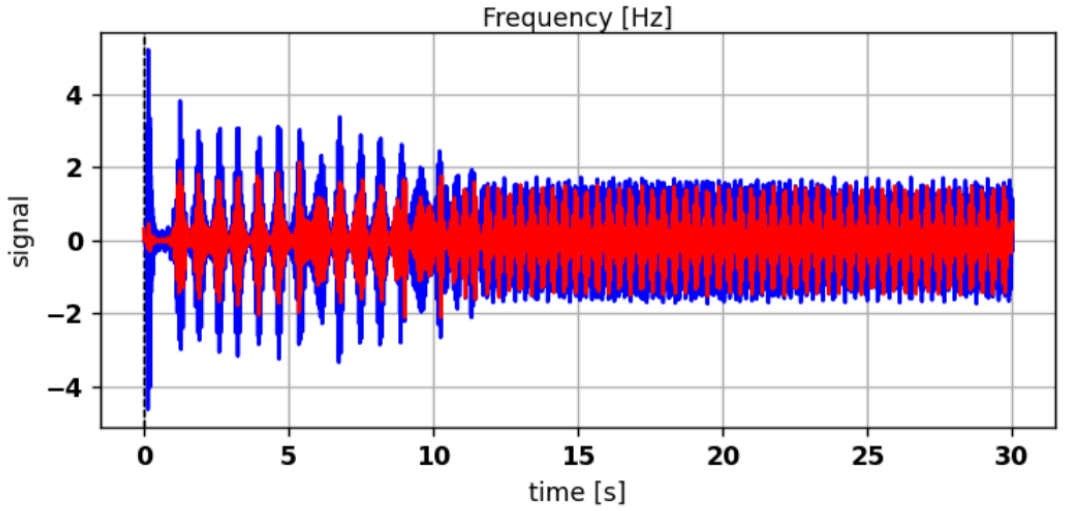


Spin Dynamics (Single Spin) under radiation Damping

Simulation of specialized NMR Experiments:
Masers (single and multi-mode, J Coupling)



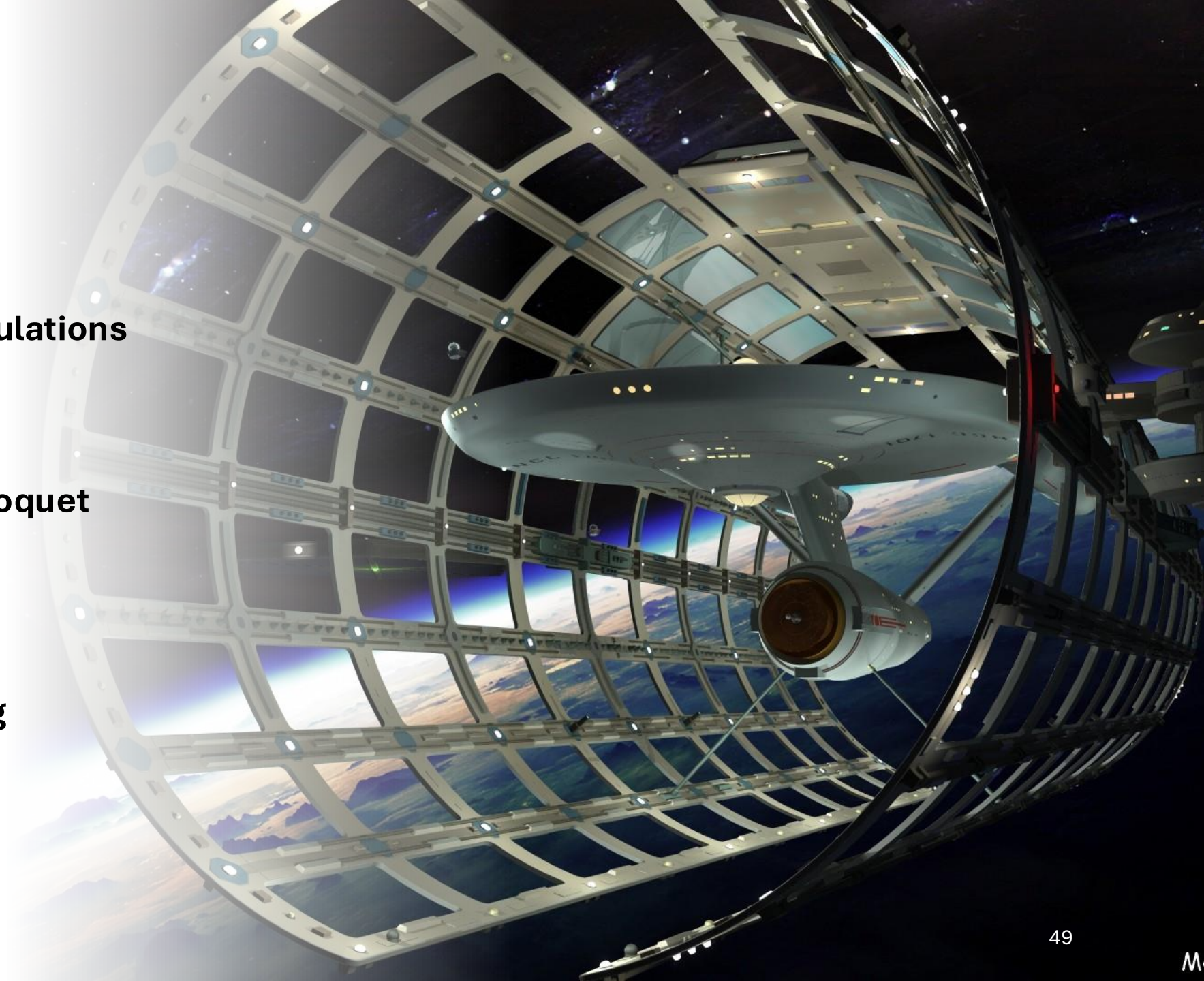
Single Mode (Single Spin)



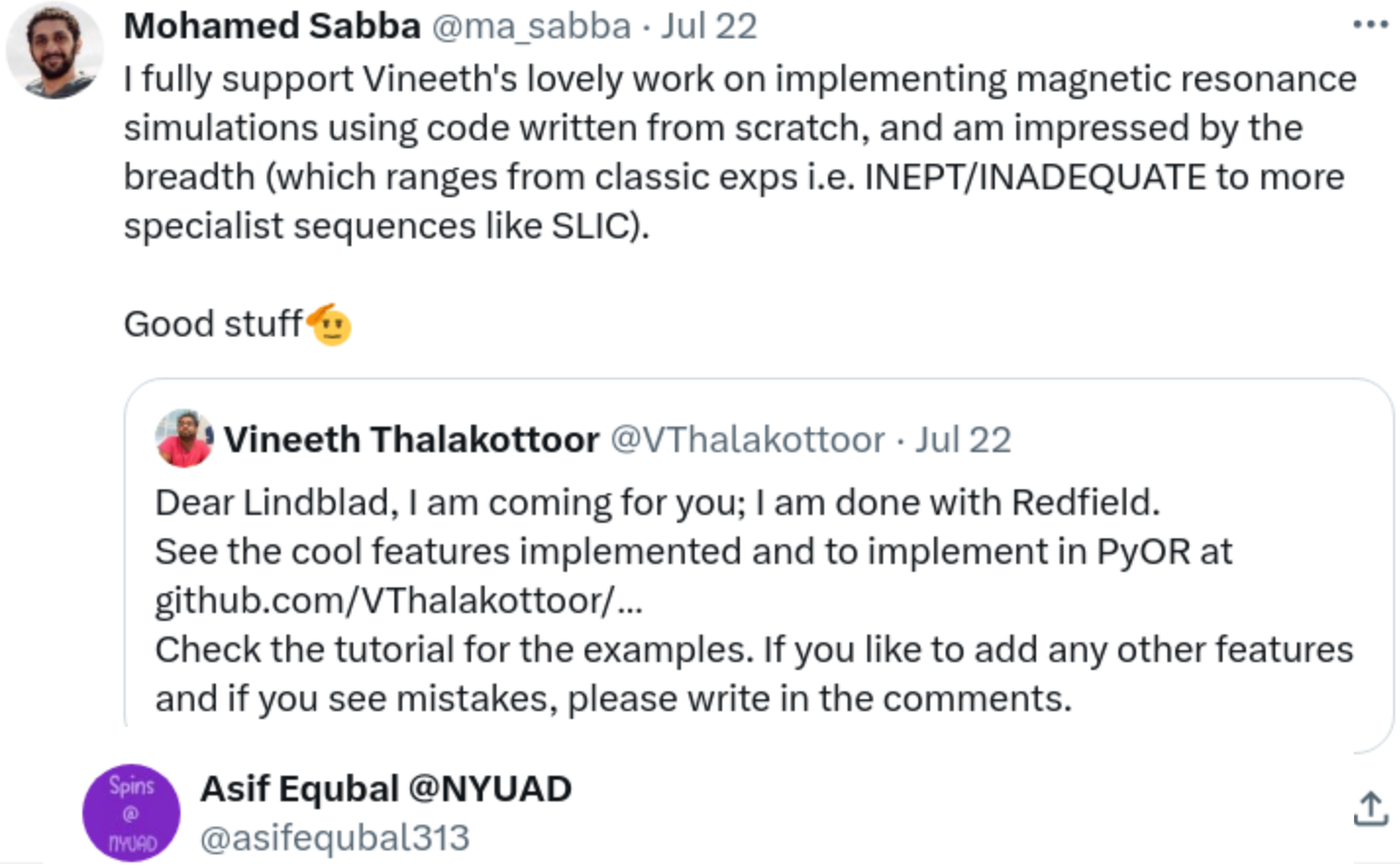
Two Mode (Two Spins, with J coupling)

Future

- More interesting Maser Simulations
- Sparse Matrices
- Shaped Pulses
- Average Hamiltonian and Floquet Theory
- DNP
- Anything I see interesting
- Anything you see interesting



Feedback and Suggestions: LinkedIn, Twitter, Email



Mohamed Sabba @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).

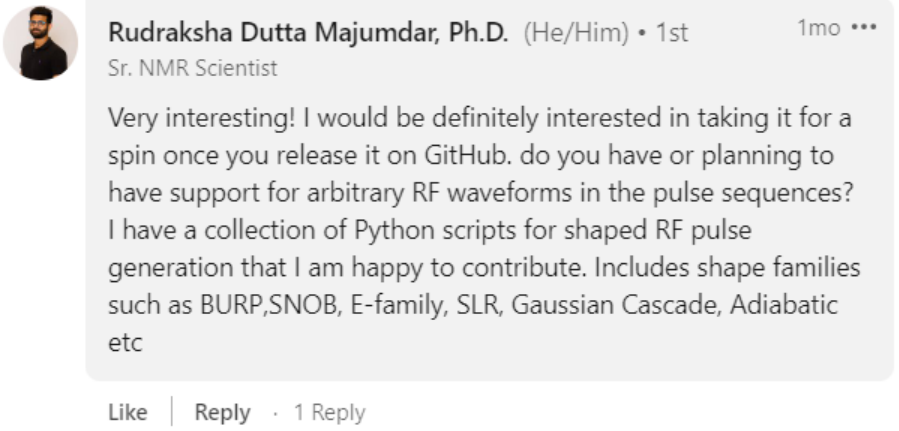
Good stuff🫡

> **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/...
> Check the tutorial for the examples. If you like to add any other features and if you see mistakes, please write in the comments.

Asif Equbal @NYUAD
@asifequbal313

This is remarkable. Simulation is a great tool to learn spin physics.

12:15 PM · Aug 31, 2024 · **90** Views



Rudraksha Dutta Majumdar, Ph.D. (He/Him) • 1st
Sr. NMR Scientist
1mo

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

B. Dillmann
@dillmann_bd

Thanks for this very nice project, very useful !

12:21 PM · Sep 1, 2024 · **15** Views

Acknowledgement

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

PyOR is all yours now

"Let what is created surpass the creator"