# PyOR: A Versatile NMR Simulator for the Beginners and Teaching


**Vineeth Francis Thalakottoor Jose Chacko**

**UCCS UMR 8181, University of Lille**

**15th January 2025**

# PyOR (Python On Resonance)

*Motto: **"Everybody can simulate NMR"****

- **Python based NMR Simulator**
- **Why Python?** It's free, easy to learn, a lot packages (Numpy, Scipy, Sympy, Matplotlib,...), great online support.
- **What makes PyOR unique? Great readability of the source code** (*single file*) unlike other popular simulators available.
- **Versatile**: From **basic to specialized** NMR experiments
- **Programming (user)**: Jupyter Notebook or text file
- https://github.com/VThalakottoor/PyOR-Jeener-Beta

** Required basic knowledge of spin operators

# Features: Spin Operators (Heart of PyOR)

- Particle with any spin quantum number
  - ½, 1, 3/2, 2, ….
- Any number of particles
  - ½ (Single spin system)
  - ½, ½ (Two spin system)
  - ½, ½, ½ (Three spin system)
  - ½, 1 (Two spin mixed system)
- Example: Spin half system

```
Spin_list = [1/2]

hbarEQ1 = True

System = PyOR.Numerical_MR(Spin_list,hbarEQ1)

Sx,Sy,Sz = System.SpinOperator()
```

```
Matrix(Sx[0])
```

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

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

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

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

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

- Example: Two spin half system

```
Spin_list = [1/2,1/2]
hbarEQ1 = True
System = PyOR.Numerical_MR(Spin_list,hbarEQ1)
Sx,Sy,Sz = System.SpinOperator()
```

```
Matrix(Sx[0])
```

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

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

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

First spin

```
Matrix(Sx[1])
```

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

```
Matrix(Sy[1])
```

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

Second spin

# Features: Hamiltonians

- Zeeman Hamiltonian

```
# Gyromagnetic Ratio
Gamma = [System.gammaH1,System.gammaH1]
# B0 Field in Tesla, Static Magnetic field (B0) along Z
B0 = 9.4
# Rotating Frame Frequency
OmegaRF = [-System.gammaH1*B0,-System.gammaH1*B0]
# Offset Frequency in rotating frame (Hz)
Offset = [10.0,20.0]
# generate Larmor Frequencies
LarmorF = System.LarmorFrequency(Gamma,B0,Offset)
# Lab Frame Hamiltonian
Hz_lab = System.Zeeman(LarmorF,Sz)
# Rotating Frame Hamiltonian
Hz = System.Zeeman_RotFrame(LarmorF,Sz,OmegaRF)
```

```
Matrix(System.Matrix_Round(Hz/(2.0*np.pi), 2))
```

$$\begin{bmatrix} -15.0 & 0 & 0 & 0 \\ 0 & 5.0 & 0 & 0 \\ 0 & 0 & -5.0 & 0 \\ 0 & 0 & 0 & 15.0 \end{bmatrix}$$

- J coupling Hamiltonian

```
Jlist = np.zeros((len(Spin_list),len(Spin_list)))
Jlist[0][1] = 1
Jcoupling_Strong = True
if Jcoupling_Strong:
    Hj = System.Jcoupling(Jlist,Sx,Sy,Sz)
else:
    Hj = System.Jcoupling_Weak(Jlist,Sz)
```

```
Matrix(Hj/(2.0*np.pi))
```

$$\begin{bmatrix} 0.25 & 0 & 0 & 0 \\ 0 & -0.25 & 0.5 & 0 \\ 0 & 0.5 & -0.25 & 0 \\ 0 & 0 & 0 & 0.25 \end{bmatrix}$$

- B1 Hamiltonian
- Dipole Hamiltonian
- Define your own Hamiltonian

# Features: Superoperator

- Liouville-von Neumann Eq.
  - Form 1

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

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

  - Form 2

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

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

- Commutation Hamiltonian Superoperator

$$\hat{H}_0 = H_0 \otimes 1_d - 1_d \otimes H_0^\top$$

```
Hz_L = System.CommutationSuperoperator(Hz)
```

- Vectorize density matrix

```
rho_in_L = System.Vector_L(rho_in)
rhoeq_L = System.Vector_L(rhoeq)
```

# Features: Relaxation

- Redfield Master Equation (special case: Auto Corr. Dipolar relaxation)
  - Form 1

$$\frac{d}{dt}\rho = \frac{-i}{\hbar}[H_0, \rho] + \sum_{q=-2}^{2}\sum_{p} S(\omega_p^q)[A_{2p}^q, [(A_{2p}^q)^\dagger, \rho - \rho_{eq}]]$$

  - Form 2

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

$$\hat{R} = \sum_{q=-2}^{2}\sum_{p} S(\omega_p^q)\hat{A}_{2p}^q(\hat{A}_{2p}^q)^\dagger$$

- Phenomenological relaxation
- Random field fluctuation
- Homonuclear and Heteronuclear dipolar relaxation (any number of spin pairs)
- Lindblad master equation: Homonuclear and Heteronuclear dipolar relaxation
- Compute relaxation rates
- Define relaxation mechanism

# Features: Solve Liouville-von Neumann Equation

- State of the system: Density matrix
  - Form 1 (Hilbert space)

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

  - Form 2 (Liouville space)

$$\tilde{\rho} = [\rho_{11}, \rho_{12}, \rho_{13}, \rho_{14}, \ldots . \rho_{44}]^{\top}$$

- Solve Liouville-von Neumann Eq.
  - Unitary propagation

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

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

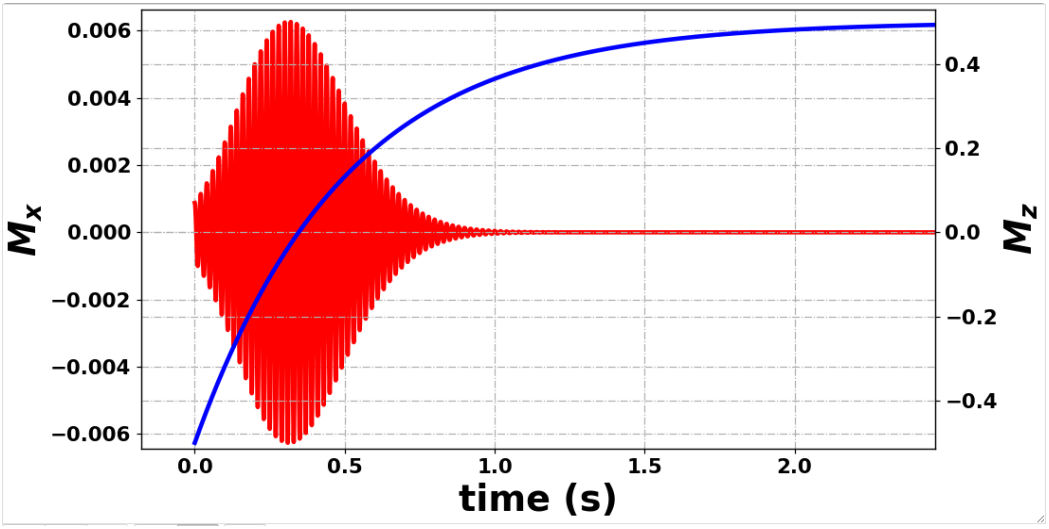
  - Solve Ordinary Diff. Eq

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

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

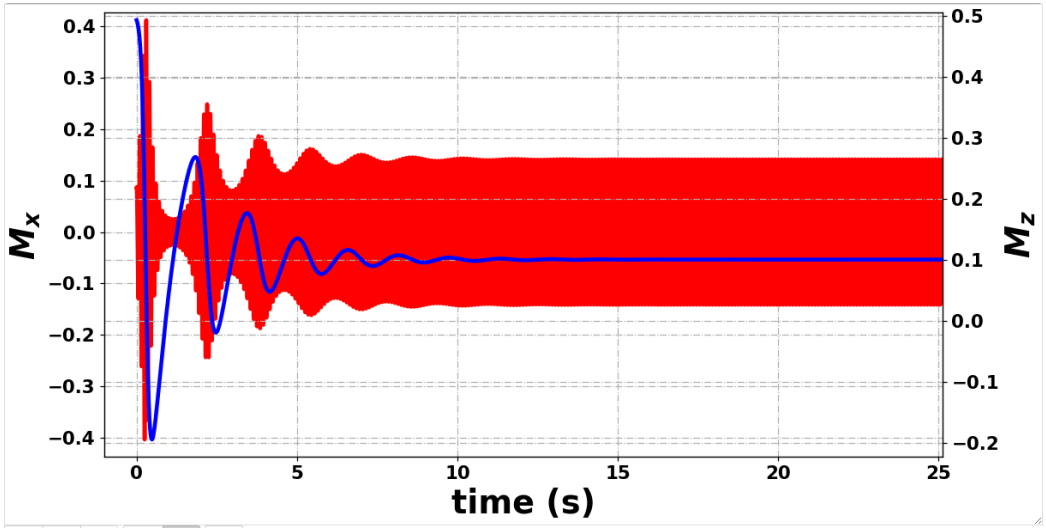
# Features: Radiation Damping (not in beta version)

- Radiation Damping




```
RDgain = [30]
RDphase = [0]
Rdamping = True
System.RDparameters(RDgain,RDphase,Rdamping)
```

- Maser



# Features: Product Operators Basis

- Decomposition of density matrix in terms of product operator basis

$$\rho = \sum_i a_i B_i$$

- Spherical tensor product operator
  - Any spin quantum number
  - Any number of spins
- Special case: Spin half
  - Cartesian
  - Zeeman
  - Spherical

- Example: Two spin half (spherical)

```
sort = 'negative to positive'
Index = False
Normal = True
Basis_PMZ, coh_PMZ, dic_PMZ = System.ProductOperators_SpinHalf_PMZ(sort,Index,Normal)
```

```
print(dic_PMZ)
```

```
['Im1Im2', 'Im1Iz2', 'Iz1Im2', 'Im1', 'Im2', 'Im1Ip2', 'Ip1Im2', 'Iz1Iz2', 'Iz1', 'Iz2', 'ID', 'Ip1Iz2', 'Iz1Ip2',
'Ip1', 'Ip2', 'Ip1Ip2']
```

```
print(coh_PMZ)
```

```
[-2, -1, -1, -1, -1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 2]
```

```
Matrix(Basis_PMZ[0])
```

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

```
Matrix(OpB_H["Im1Im2"])
```

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

```
OpB_H = System.String_to_Matrix(dic_PMZ, Basis_PMZ)
```

# Documentation

```python
Spin_list = [1/2,1/2]
hbarEQ1 = True
System = PyOR.Numerical_MR(Spin_list,hbarEQ1)
Sx,Sy,Sz = System.SpinOperator()
```

```python
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, ...]
```

```python
Sp,Sm = System.PMoperators(Sx,Sy)
help(System.PMoperators)
```

```
Help on method PMoperators in module PythonOnResonance:

PMoperators(Sx, Sy) method of PythonOnResonance.Numerical_MR instance
    Generate spin operators for all spins: Sp (Sx + j Sy) and Sm (Sx - j Sy)
    INPUT
    -----
    Sx, Sy

    OUTPUT
    ------
    Sp : array [Sp of spin 1, Sp of spin 2, Sp of spin 3, ...]
    Sm : array [Sm of spin 1, Sm of spin 2, Sm of spin 3, ...]
```

# Example: NOE
# (Hilbert and Liouville space)

# Define Spin System (Case: Two spin half)

## Define the Path to PyOR source code, PythonOnResonance.py

```
pathSource = '/media/HD2/Vineeth/PostDoc_Simulations/Github/PyOR_V1/Source'
```

## Load Python packages

```
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 (Two Spin Half)

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

## Define the unit of Hamiltonian

hbarEQ1 is True, then unit of Hamiltonian is in angular frequency

```
hbarEQ1 = True
```

## Generate the spin operators: Sx, Sy and Sz

```
System = PyOR.Numerical_MR(Spin_list,hbarEQ1)
Sx,Sy,Sz = System.SpinOperator()
```

## Generate the spin operators: S+ and S-

```
Sp,Sm = System.PMoperators(Sx,Sy)
```

## Generating Zeeman Hamiltonian (Lab and Rotating Frame)

```
# Gyromagnetic Ratio
Gamma = [System.gammaH1,System.gammaH1]
# B0 Field in Tesla, Static Magnetic field (B0) along Z
B0 = 9.4
# Rotating Frame Frequency
OmegaRF = [-System.gammaH1*B0,-System.gammaH1*B0]
# Offset Frequency in rotating frame (Hz)
Offset = [10.0,20.0]
# generate Larmor Frequencies
LarmorF = System.LarmorFrequency(Gamma,B0,Offset)
# Lab Frame Hamiltonian
Hz_lab = System.Zeeman(LarmorF,Sz)
# Rotating Frame Hamiltonian
Hz = System.Zeeman_RotFrame(LarmorF,Sz,OmegaRF)
```

```
Larmor Frequency in MHz:  [-400.22802765 -400.22803765]
```

## Zeeman Hamiltonian in Liouville Space (Zeeman Basis)

```
Hz_L = System.CommutationSuperoperator(Hz)
```

## Zeeman Basis Kets

```
Kets = System.Basis_Ket()
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>']
```

## Zeeman Basis Bras

```
Bras = System.Basis_Bra()
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|']
```

## Zeeman Basis states

```
Basis_Zeeman_state = System.ZBasis_H(Hz_lab)
```

### Matrix represenation of Zeeman Basis states

```
Matrix(Basis_Zeeman_state[0])
```

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

## Initialize Density Matrix

```
Thermal_DensMatrix = False

if Thermal_DensMatrix:
    # Spin temperature of individual spins (initial) Kelvin
    Tin = [300.0,300.0]

    # Spin temperature of individual spins (equlibrium) Kelvin
    Tfi = [300.0,300.0]

    # High Temperature
    HT_approx = False

    # Initial Density Matrix
    rho_in = System.EqulibriumDensityMatrix_Advance(LarmorF,Sz,Tin,HT_approx)

    # Equlibrium Density Matrix
    rhoeq = System.EqulibriumDensityMatrix_Advance(LarmorF,Sz,Tfi,HT_approx)
else:
    rho_in = np.sum(Sz,axis=0)
    rhoeq = np.sum(Sz,axis=0)
```

```
System.DensityMatrix_Components_Dictionary(Basis_PMZ,dic_PMZ,rho_in)
```

```
Density Matrix = 1.0 Iz1 + 1.0 Iz2
```

## Converting initial and equlibrium density matrix into Liouvillian

```
rho_in_L = System.Vector_L(rho_in)
rhoeq_L = System.Vector_L(rhoeq)
```

## Pulse (Hilbert Space)

```
flip_angle1 = 0.0   # Flip angle Spin 1
flip_angle2 = 180.0 # Flip angle Spin 2

rho = System.Rotate_H(rho_in,flip_angle1,Sy[0])
rho = System.Rotate_H(rho,flip_angle2,Sy[1])
```

```
System.DensityMatrix_Components_Dictionary(Basis_PMZ,dic_PMZ,rho)
```

```
Density Matrix = 1.0 Iz1 + -1.0 Iz2
```

## Pulse (Liouville Space)

```
rho_L = System.Rotate_L(rho_in_L,flip_angle1,Sy[0])
rho_L = System.Rotate_L(rho_L,flip_angle2,Sy[1])
```

## Relaxation in Hilbert Space

```
R1 = None
R2 = None

# Correlation Time
tau = [10.0e-12]

# Dipolar coupling constant (Hz)
bIS = [30.0e3]

Rprocess = "Auto-correlated Dipolar Homonuclear"

System.Relaxation_Constants(R1,R2)
System.Relaxation_Parameters(LarmorF, OmegaRF, tau, bIS)
```

## Relaxation Rate Spin 1

```
R1_rate = System.RelaxationRate_H(Sz[0],Sz[0],Rprocess,R1,R2,Sx,Sy,Sz,Sp,Sm)
Rcross_rate = System.RelaxationRate_H(Sz[1],Sz[0],Rprocess,R1,R2,Sx,Sy,Sz,Sp,Sm)
Longit_relaxa = R1_rate + Rcross_rate
R2_rate = System.RelaxationRate_H(Sp[0],Sp[0],Rprocess,R1,R2,Sx,Sy,Sz,Sp,Sm)

print("T1 = %.5f and T2 = %.5f" % ((1.0/R1_rate).real, (1.0/R2_rate).real))
print("R1 = %.5f and R2 = %.5f" % ((R1_rate).real, (R2_rate).real))
print("Cross Relaxation rate = %.5f and time = %.5f" % (Rcross_rate.real, 1.0/Rcross_rate.real))
print("Longitudinal Relaxation rate = %.5f and time = %.5f" % (Longit_relaxa.real, 1.0/Longit_relaxa.real))
```

```
T1 = 5.63856 and T2 = 5.63482
R1 = 0.17735 and R2 = 0.17747
Cross Relaxation rate = 0.08856 and time = 11.29210
Longitudinal Relaxation rate = 0.26591 and time = 3.76070
```

## Evolution of Density Matrix in Hilbert Space

```
dt = 0.0001
AQ = 50.0
Npoints = int(AQ/dt)

method = "ODE Solver"
ode_solver = 'DOP853'
System.ODE_Method(ode_solver)

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

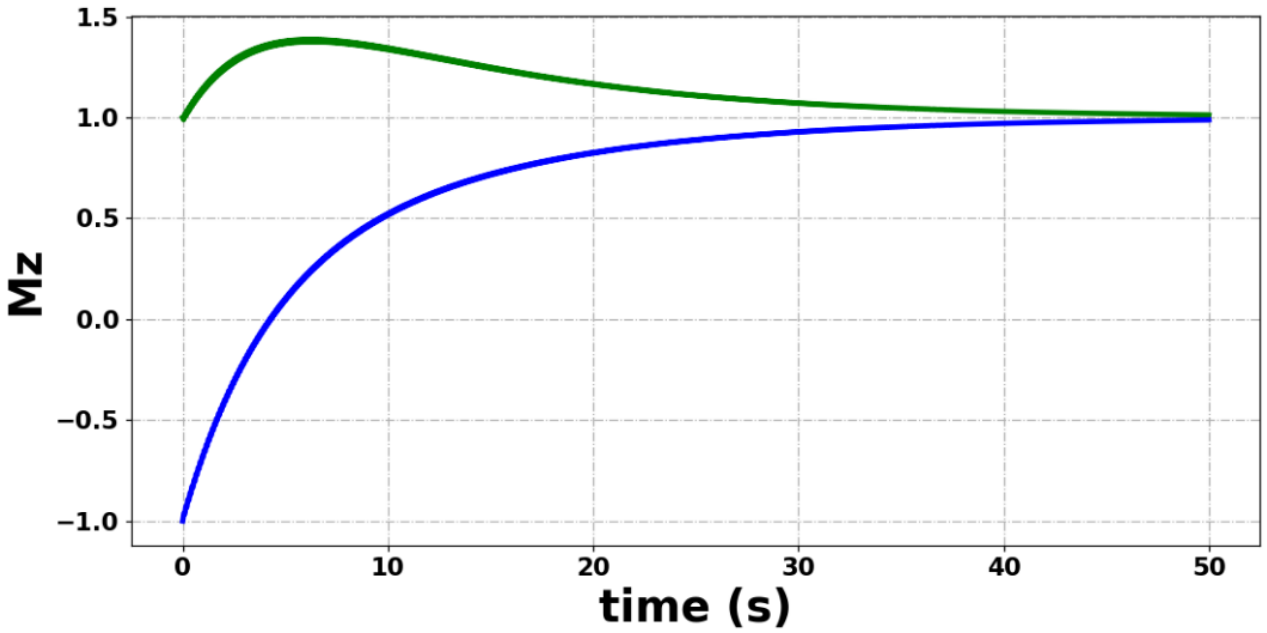
```
Total time = 33.43313264846802 seconds
```

## Expectation Value (Hilbert Space)

```
det_Z1 = Sz[0]
det_Z2 = Sz[1]

t, signal_Z1 = System.Expectation_H(rho_t,det_Z1,dt,Npoints)
t, signal_Z2 = System.Expectation_H(rho_t,det_Z2,dt,Npoints)
```

```
System.PlottingMulti(1,[t,t],[signal_Z1,signal_Z2],"time (s)","Mz",["green","blue"])
```

```
<IPython.core.display.Javascript object>
```



# Relaxation in Liouville Space

```
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)
```

## Evolution of Density Matrix Liouville Space

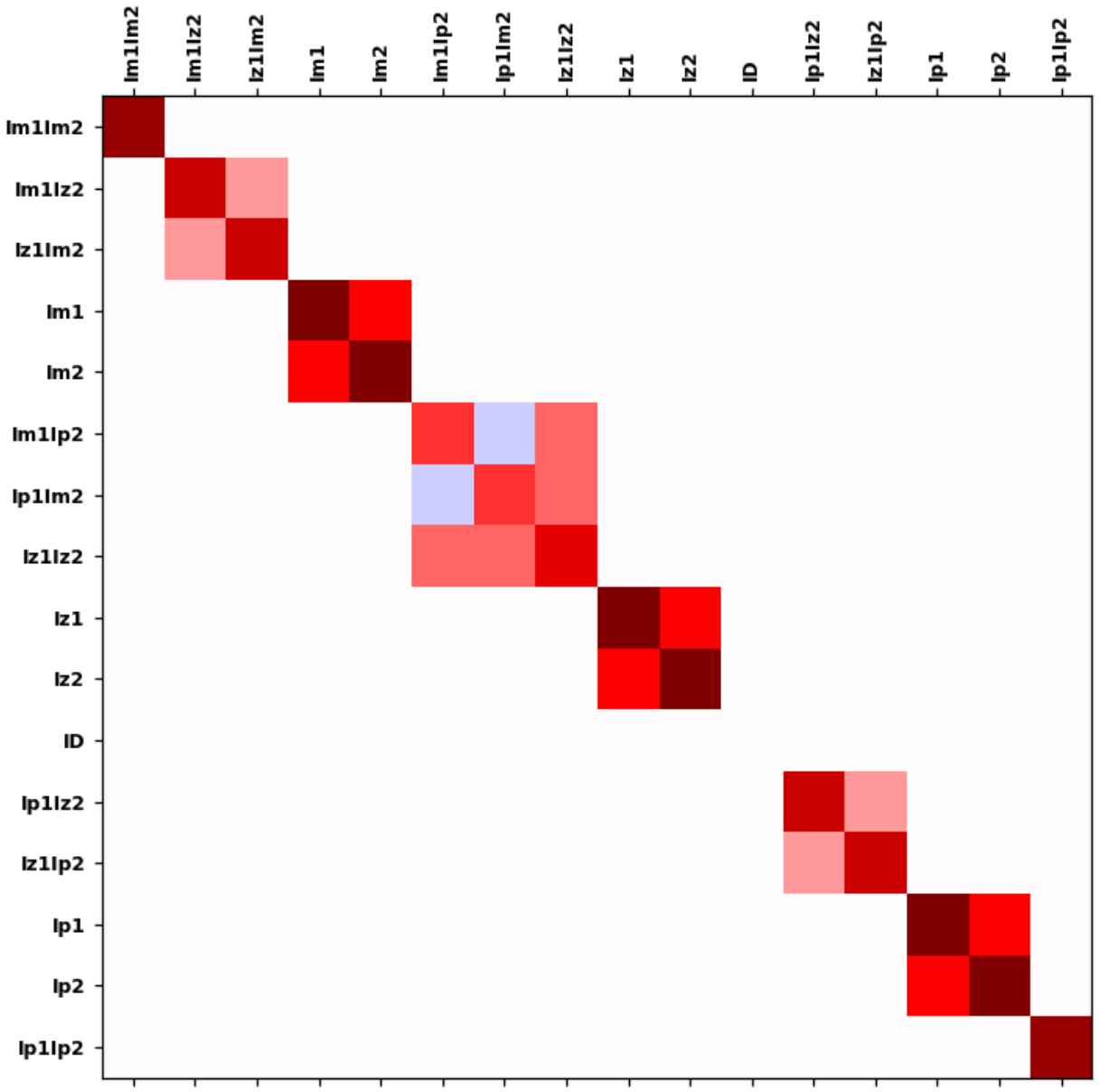
```
method = "ODE Solver"
System.ODE_Method('DOP853')

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

```
(16, 500000)
Total time = 3.557140827178955 seconds
```

# Expectation value (Liouville Space)

```
LEXP_Z1 = System.Detection_L(det_Z1)
LEXP_Z2 = System.Detection_L(det_Z2)
```

```
t, MZ_1 = System.Expectation_L(rho_t,LEXP_Z1,dt,Npoints)
t, MZ_2 = System.Expectation_L(rho_t,LEXP_Z2,dt,Npoints)
```



# Look detailed notebook in Github

PyOR-Jeener-Beta / Future Version

/ **Introduction to PyOR - NOE (Hilbert and Liouville Space - ODE Solver).ipynb**

**VThalakottoor** Introduction to PyOR (NOE, ODE Solver, Sparse) 267c35d · 2 months ago

**Preview** Code | Blame 6440 lines (6440 loc) · 746 KB

## Python On Resonance (PyOR)

### Everybody can simulate NMR

### Version: Jeener

### Tutorial: Introduction to PyOR - NOE (Hilbert and Liouville Space - ODE Solver)

Author: Vineeth Thalakottoor

Email: vineethfrancis.physics@gmail.com

# Example: Ethanolamine Spectra

## Define Spin System (Four Spin Half)

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

## Define the unit of Hamiltonian

```
hbarEQ1 = True
```

### Generate the spin operators: Sx, Sy, Sz, Sp and Sm

```
# Call Class "Numerical_MR" from PyOR package
System = PyOR.Numerical_MR(Spin_list,hbarEQ1)

Sx,Sy,Sz = System.SpinOperator()

Sp,Sm = System.PMoperators(Sx,Sy)
```

```
# Offset Frequency in rotating frame (Hz)
ref_freq = System.gammaH1*B0
Freq_1 = System.PPMtoHz(2.35,ref_freq)
Freq_2 = System.PPMtoHz(2.35,ref_freq)
Freq_3 = System.PPMtoHz(3.1,ref_freq)
Freq_4 = System.PPMtoHz(3.1,ref_freq)
Offset = [Freq_1,Freq_2,Freq_3,Freq_4]
```

```
# generate Larmor Frequencies
LarmorF = System.LarmorFrequency(Gamma,B0,Offset)
```

```
Larmor Frequency in MHz:  [-500.14569182 -500.14569182 -500.14606693 -500.14606693]
```

```
# Lab Frame Hamiltonian
Hz_lab = System.Zeeman(LarmorF,Sz)
```

```
# Rotating Frame Hamiltonian
Hz = System.Zeeman_RotFrame(LarmorF,Sz,OmegaRF)
```

## Generating Zeeman Hamiltonian (Lab and Rotating Frame)

```
# Gyromagnetic Ratio
Gamma = [System.gammaH1,System.gammaH1,System.gammaH1,System.gammaH1]
```

```
# B0 Field in Tesla, Static Magnetic field (B0) along Z
B0 = System.L500
```

```
# Rotating Frame Frequency
OmegaRF = [-System.gammaH1*B0,-System.gammaH1*B0,-System.gammaH1*B0,-System.gammaH1*B0]
```

## Generating J Coupling Hamiltonian

```
Jlist = np.zeros((len(Spin_list),len(Spin_list)))
Jlist[0][1] = 11.1
Jlist[0][2] = 3.9
Jlist[0][3] = 6.69
Jlist[1][2] = 6.69
Jlist[1][3] = 3.9
Jlist[2][3] = 12.3

Jcoupling_Strong = True

if Jcoupling_Strong:
    Hj = System.Jcoupling(Jlist,Sx,Sy,Sz)
else:
    Hj = System.Jcoupling_Weak(Jlist,Sz)
```

### Initialize Density Matrix

```
Thermal_DensMatrix = False

if Thermal_DensMatrix:
    # Spin temperature of individual spins (initial) Kelvin
    Tin = [300.0,300.0]

    # Spin temperature of individual spins (equlibrium) Kelvin
    Tfi = [300.0,300.0]

    # High Temperature
    HT_approx = False

    # Initial Density Matrix
    rho_in = System.EqulibriumDensityMatrix_Advance(LarmorF,Sz,Tin,HT_approx)

    # Equlibrium Density Matrix
    rhoeq = System.EqulibriumDensityMatrix_Advance(LarmorF,Sz,Tfi,HT_approx)
else:
    rho_in = np.sum(Sz,axis=0)
    rhoeq = np.sum(Sz,axis=0)
```

## Pulse (Hilbert Space)

```
flip_angle = 90.0

rho = System.Rotate_H(rho_in,flip_angle,np.sum(Sy,axis=0))
```

## Relaxation

```
R1 = 0.0
R2 = 0.0

Rprocess = "No Relaxation"
System.Relaxation_Constants(R1,R2)
```

## Evolution of Density Matrix

```
Highest_Larmor_Frequency = Freq_4
fs = 4 * Highest_Larmor_Frequency
dt = 1.0/fs
AQ = 5.0
Npoints = int(AQ/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,Hz+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 31008
Total time = 0.29144835472106934 seconds
```

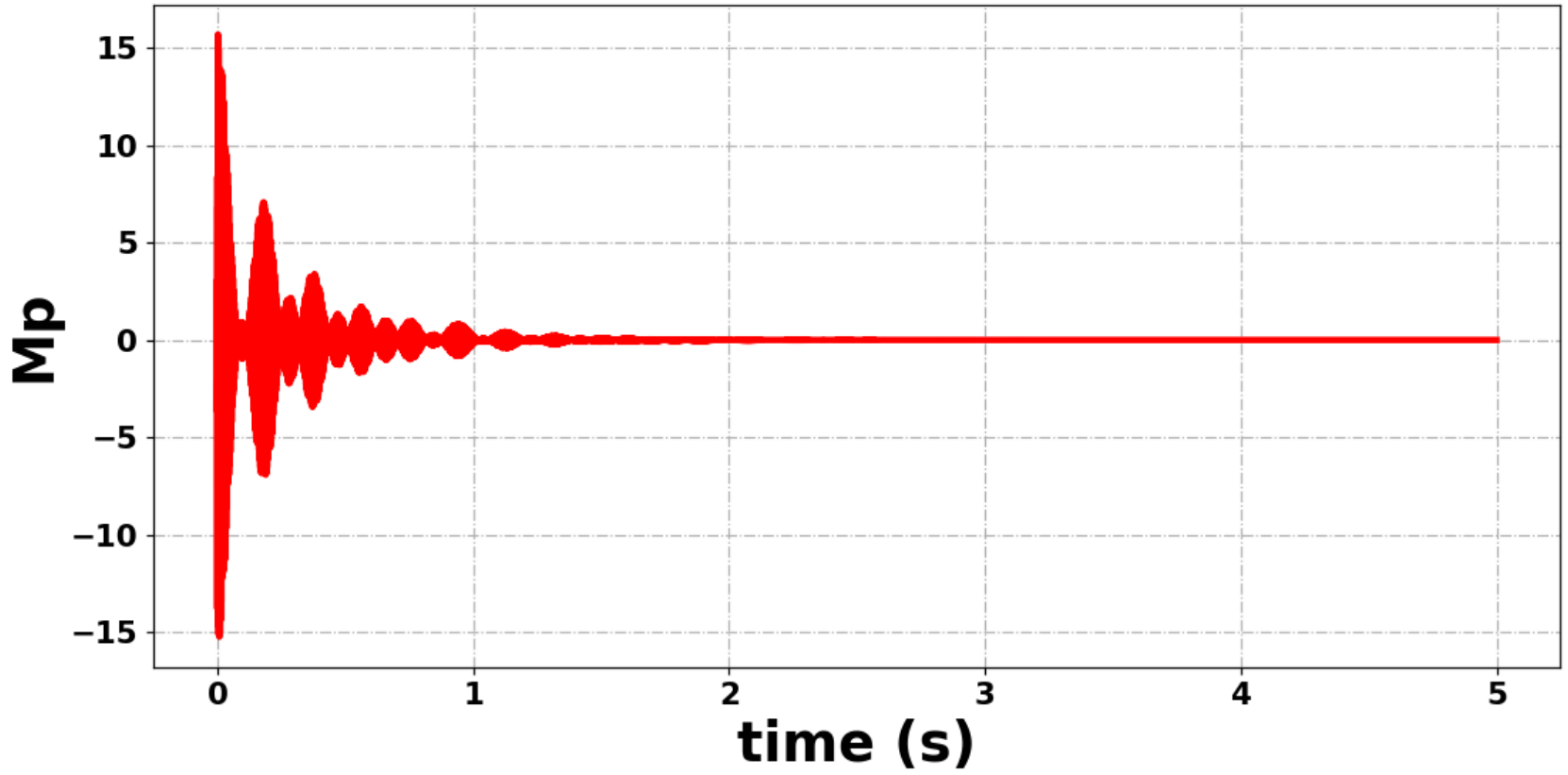
## Expectation value

```
t, Mp = System.Expectation_H(rho_t,np.sum(Sp,axis=0),dt,Npoints)
```

## Windowing

```
Mp_w = System.WindowFunction(t,Mp,3.0)
```

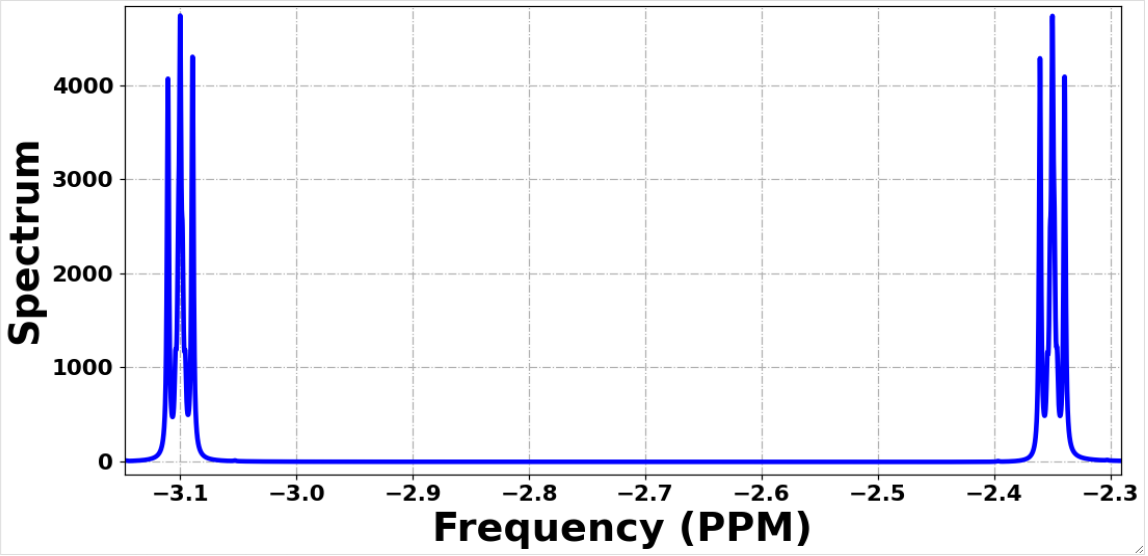
```
System.Plotting(1,t,Mp_w,"time (s)","Mp","red")
```

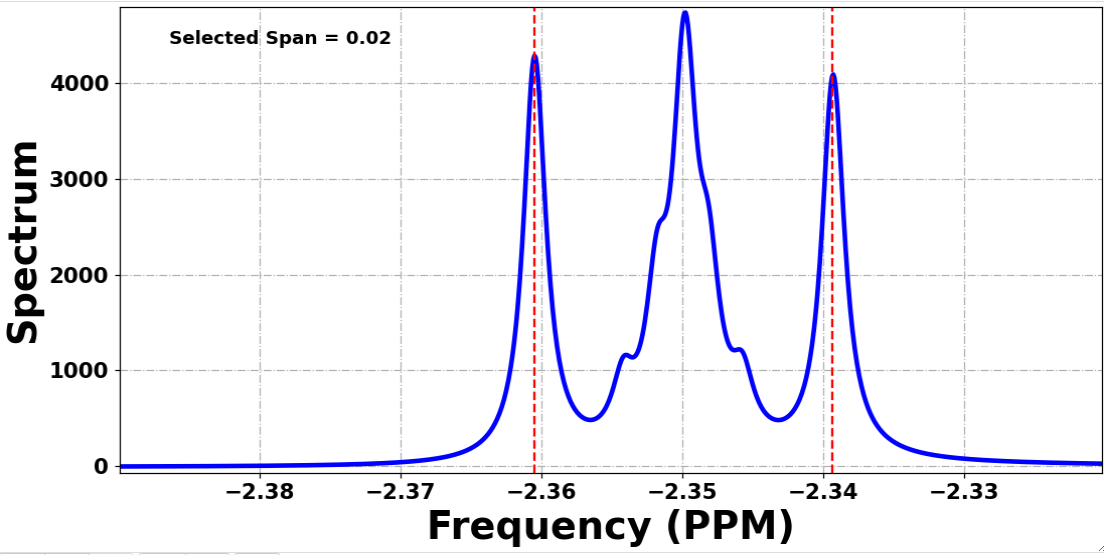


## Fourier Transform

```
fs = 1.0/dt
freq, spectrum = System.FourierTransform(Mp_w,fs,5)
```

```
# Zero order phase correction
PH0 = 45.0
spectrum_PH0 = System.PhaseAdjust_PH0(spectrum,PH0)

# First order phase correction
pivot = -1550.8
slope = 1.1
spectrum_PH1 = System.PhaseAdjust_PH1(freq,spectrum_PH0,pivot,slope)

PPMscale = True
if PPMscale:
    fig,span_selector = System.Plotting_SpanSelector(2,System.HztoPPM(freq,ref_freq),spectrum_PH1,"Frequency (PPM)","Spectrum","blue")
else:
    fig,span_selector = System.Plotting_SpanSelector(2,freq,spectrum_PH1,"Frequency (Hz)","Spectrum","blue")
```



# PyOR Jeener (beta version)

https://github.com/VThalakottoor/PyOR-Jeener-Beta

PyOR-Jeener-Beta Public

Pin Unwatch

main 1 Branch 0 Tags Go to file t Add file Code

VThalakottoor PyOR ENS 16Dec 2024 5adddd6 · 3 weeks ago 48 Commits

| | | |
|---|---|---|
| Future Version | Introduction to PyOR (NOE, ODE Solver, Sparse) | 2 months ago |
| Images | PyOR logo | 5 months ago |
| Messages_Talks | PyOR ENS 16Dec 2024 | 3 weeks ago |
| NMR_in_Nutshell | PyOR Beta | 5 months ago |
| Source | Update PythonOnResonance.py | 5 months ago |
| Tutorials | PyOR Beta | 5 months ago |
| README.md | Update README.md | 5 months ago |

# Future

- **Shaped Pulse**
- **Solid State NMR, Quadrupolar NMR**
- **EPR**
- **Graphical User Interface (GUI)**

# Acknowledgement

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

# PyOR is all yours

"Let what is created surpass the creator"

**Thank You**