PyOR: Expanding Quantum Simulations with Object-Oriented Design

Vineeth Francis Thalakottoor

Group Seminar 11th March 2025

Python On Resonance (PyOR)

- Powerful and flexible numerical NMR simulator.
- *For spin physics enthusiasts and teaching spin physics.*
- System: ***Ideal any number of spins with any quantum number (but, RAM limitation).***
- Space: ***Hilbert and Liouville.***
- Relaxation: ***Redfield and Lindblad.***
- ***Radiation damping and Maser/Raser.***
- ***Readability of the source code.***
- Many other useful functions.
- ***User can modify the source code for their needs.***

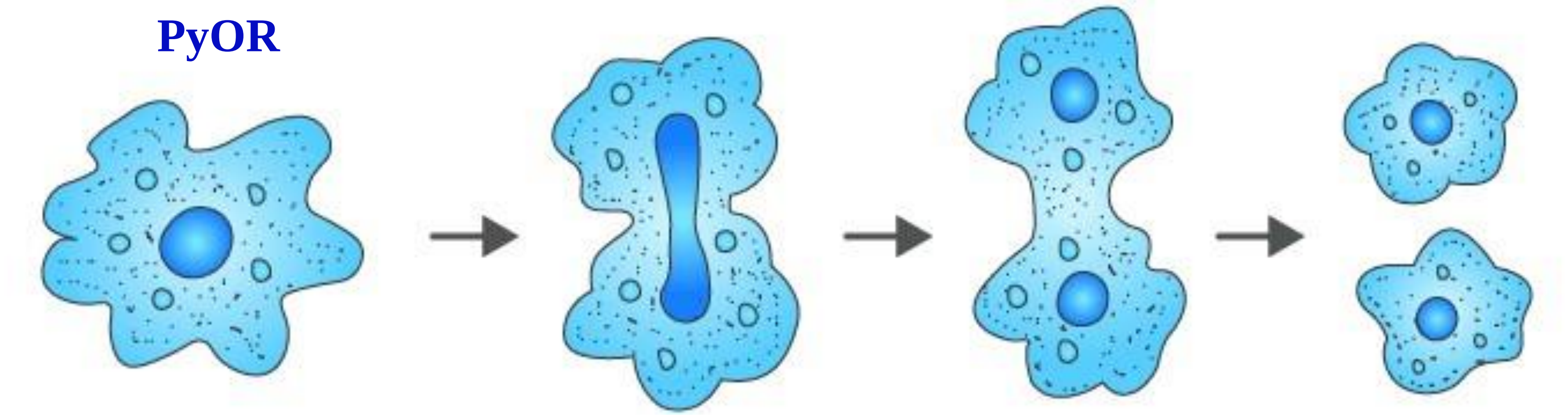
Questions

- There are a lot of NMR/Quantum simulators, then why wasting time by reinventing?
  - "Pleasure of finding things out".
- Is it efficient than other software? What about computing speed?
  - PyOR is not competing with any software, it is for understanding spin physics.
  - Optimizing PyOR will happen in a later stage, currently I am focusing on coding.

Outline

- PyOR Family
- Shaped pulse implementation for NMR simulations
    - Radiation damping
    - Relaxation
- Introducing Quantum Objects

What is new in PyOR Family?



PyOR Magnetic Resonance (*PyOR Beta*) : NMR (liquid and solid) and EPR simulations

PyOR Quantum (*inspired by QuTiP*): Magnetic resonance, NV Centrers, Polarized atoms, Atomic magnetometry, Quantum Computing, …

A work in progress

PyOR Magnetic Resonance (PyOR Jeener Beta)

- To simulate NMR and EPR experiments
- Implemented
  - *Spin operators*
    - Any number of spins (tested 6 spins)
    - Any quantum number
  - *Hilbert Space*
    - Lindblad master equation
    - Redfield master equation
  - *Liouville Space (Sparce and dense)*
    - Lindblad master equation
    - Redfield master equation
  - Many useful functions
- Work in progress
  - ssNMR, EPR, DNP and other hyperpolarization techniques

- **PythonOnResonance_MagneticResonance.py**
  - *class MagneticResonance*
    - For simulations
  - *class Fanalyzer*
    - For data analysis of multimode maser/raser

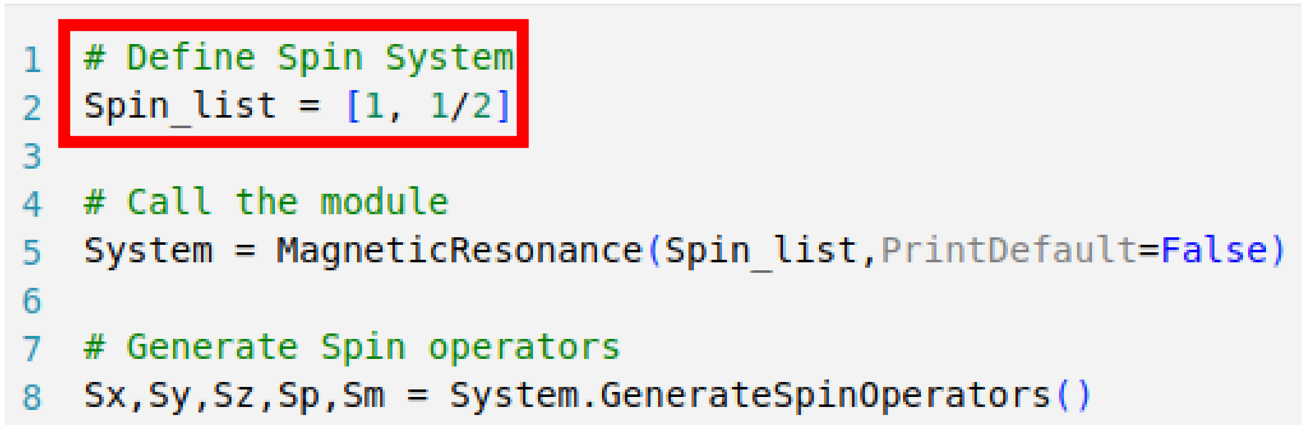
PyOR Quantum

- To Simulate magnetic resonance, NV centers, polarized atoms, atomic magnetometry, Quantum Computing, …
- *Solvers*
  - **Schrödinger equation**
  - **Liouville-von Neumann equation**
    - Lindblad master equation
      - Hilbert space
      - Liouville space

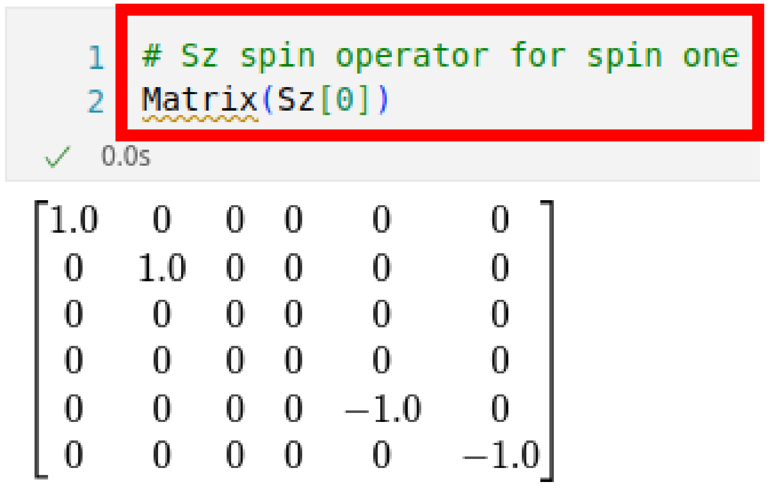
- **PythonOnResonance_Quantum.py**
  - *class QunObj*
    - Quantum objects (*Type: ket, bra, operators*)
  - *class QuantumSystem*
    - Generate spin operators of the system and subsystem
    - Generate Zeeman and coupled states (*ket*) of the quantum system
    - Generate equilibrium density matrix
  - *class QuantumLibrary*
    - Useful functions for simulations

Spin Operators: PyOR Magnetic Resonance Vs PyOR Quantum

- ***PyOR Magnetic Resonance***



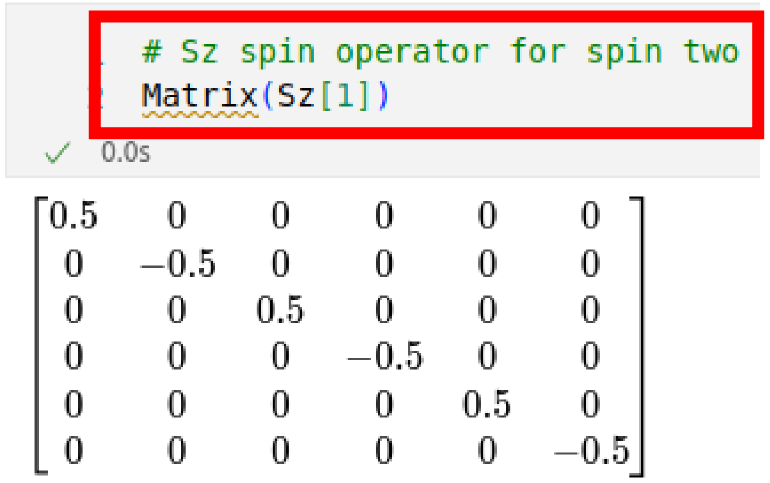
```python
# Define Spin System
Spin_list = [1, 1/2]

# Call the module
System = MagneticResonance(Spin_list,PrintDefault=False)

# Generate Spin operators
Sx,Sy,Sz,Sp,Sm = System.GenerateSpinOperators()
```



```python
# Sz spin operator for spin one
Matrix(Sz[0])
```

✓ 0.0s

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



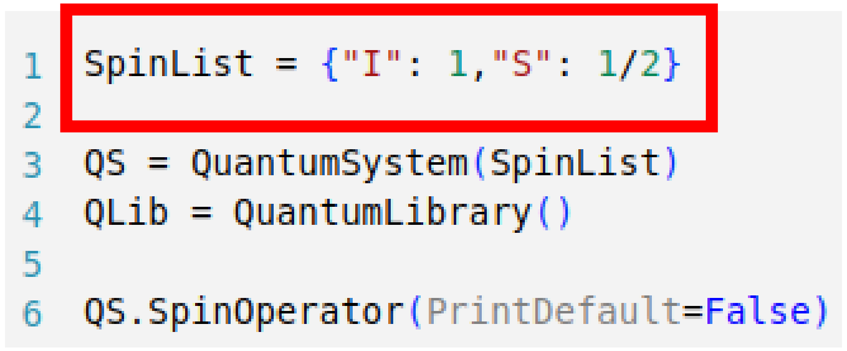
```python
# Sz spin operator for spin two
Matrix(Sz[1])
```

✓ 0.0s

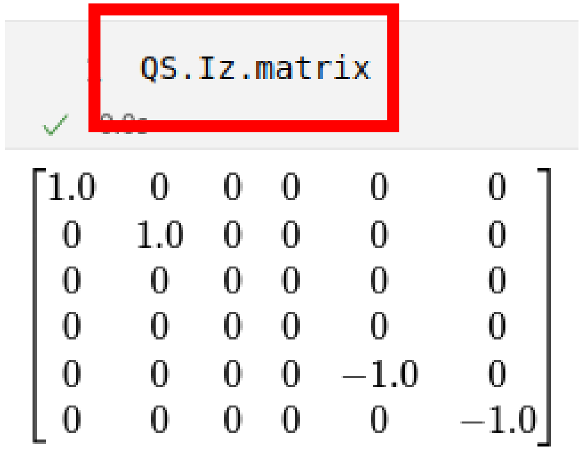
$$\begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.5 \end{bmatrix}$$

Spin Operators: PyOR Magnetic Resonance Vs PyOR Quantum

- ***PyOR Quantum***



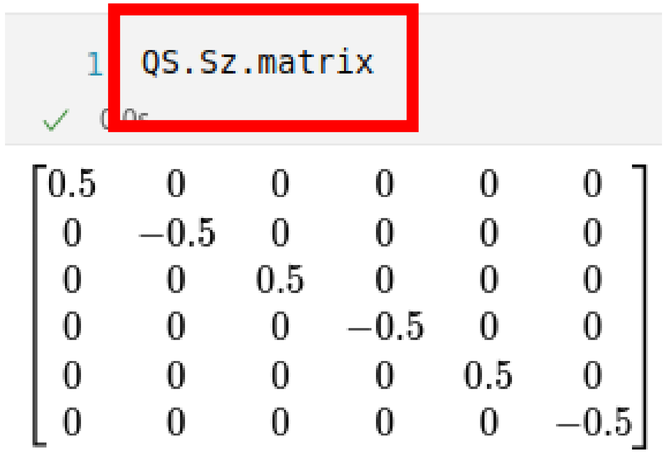
```python
SpinList = {"I": 1,"S": 1/2}

QS = QuantumSystem(SpinList)
QLib = QuantumLibrary()

QS.SpinOperator(PrintDefault=False)
```



```python
QS.Iz.matrix
```

✓ 0.0s

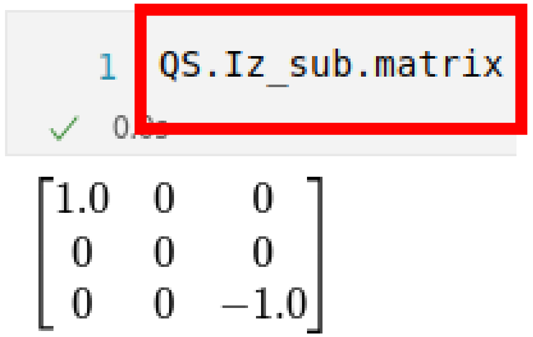
$$\begin{bmatrix} 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1.0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1.0 \end{bmatrix}$$



```python
QS.Sz.matrix
```

✓ 0.0s

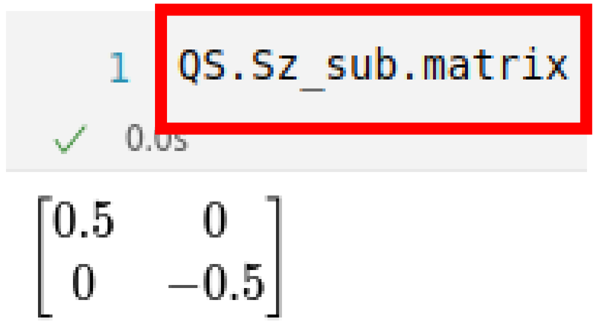
$$\begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.5 \end{bmatrix}$$



```python
QS.Iz_sub.matrix
```

✓ 0.0s

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

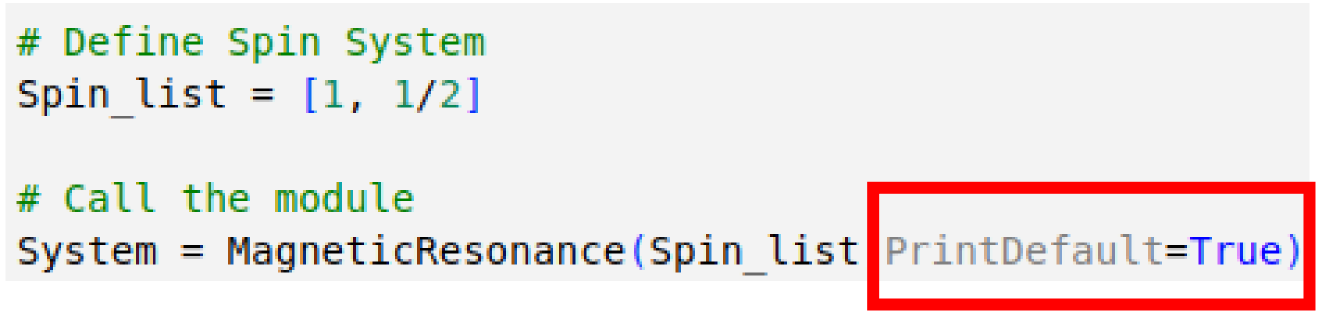


```python
QS.Sz_sub.matrix
```

✓ 0.0s

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

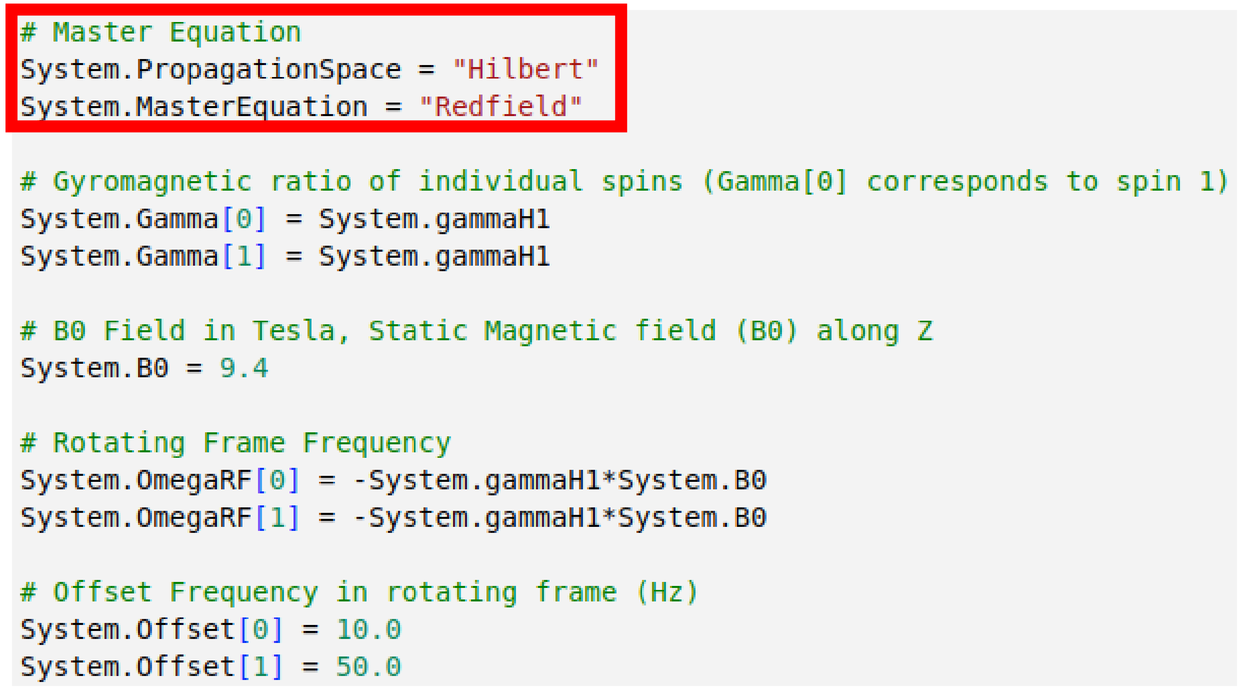
PyOR Mag. Res.: Default Simulation Parameters



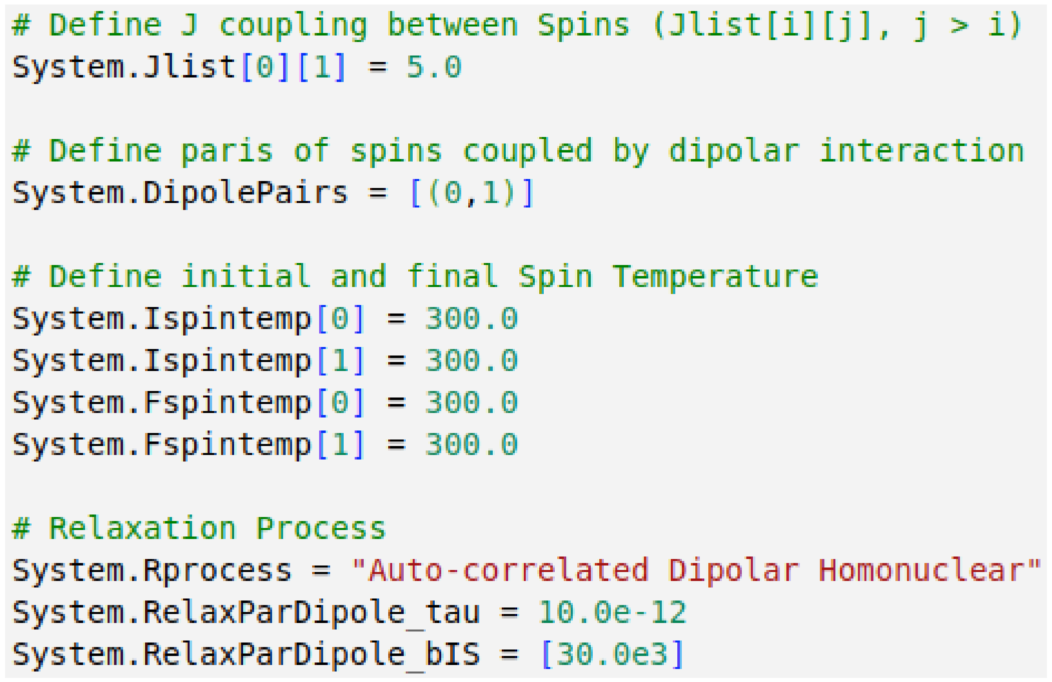
```python
# Define Spin System
Spin_list = [1, 1/2]

# Call the module
System = MagneticResonance(Spin_list PrintDefault=True)
```

```
PyOR default parameters/settings
--------------------------------

Define energy units: hbarEQ1 =  True

Define the matrix tolerence (make matrix elements less than tolarence value to zero): MatrixTolarence =  1e-06

Define the gyromagnetic ratios: Gamma =  [0, 0]

Define the static field along Z: B0 =  None

Define rotating frame frequency: OmegaRF =  [0, 0]

Define the offset frequencies of the spins: Offset =  [0, 0]

Do you want to print the larmor frequency: print_Larmor =  True

Define the J coupling: Jlist =
 [[0. 0.]
 [0. 0.]]

Define the spin paris dipolar coupled: DipolePairs =  []
```

PyOR Mag. Res.: User Simulation Parameters



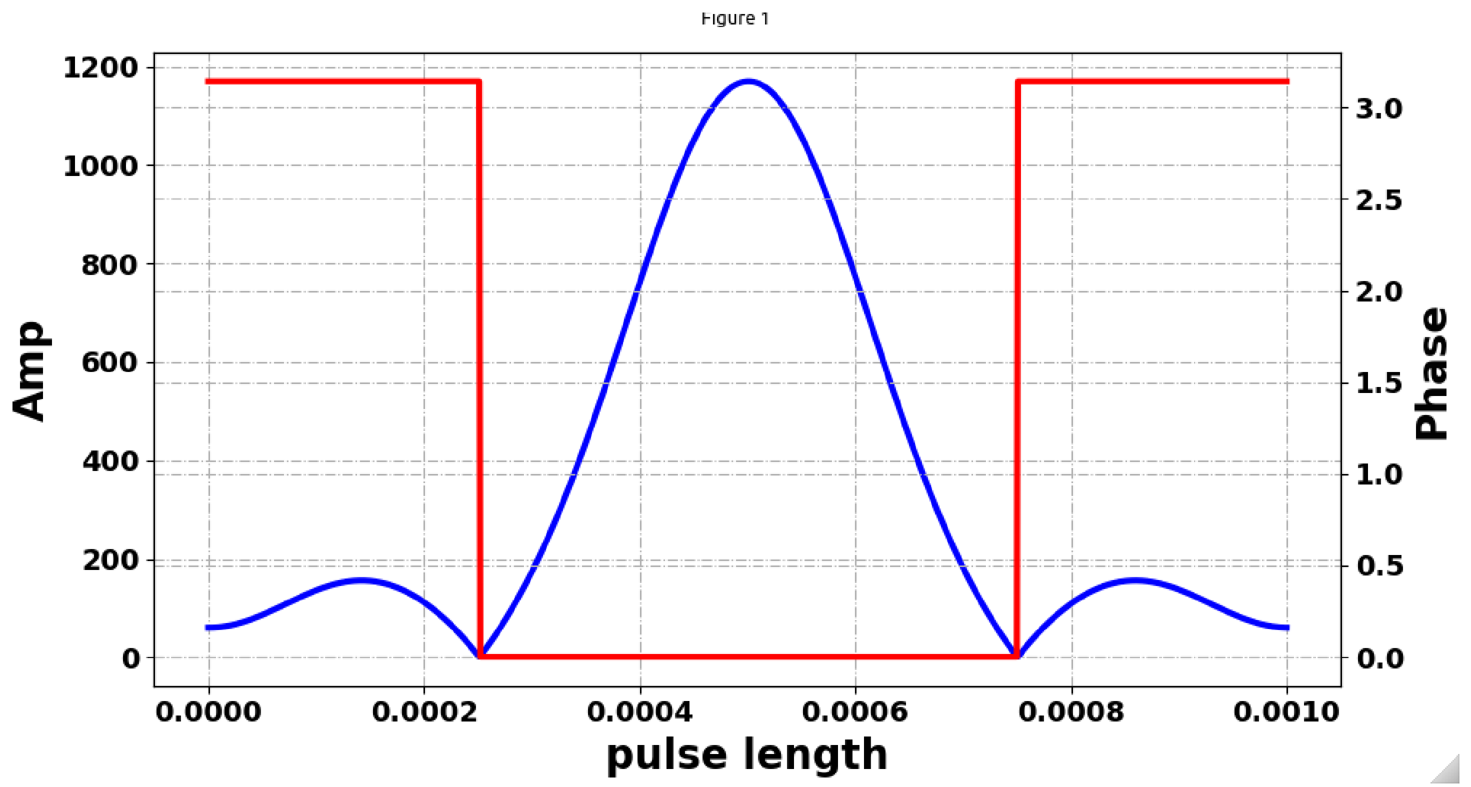
```
# Master Equation
System.PropagationSpace = "Hilbert"
System.MasterEquation = "Redfield"

# Gyromagnetic ratio of individual spins (Gamma[0] corresponds to spin 1)
System.Gamma[0] = System.gammaH1
System.Gamma[1] = System.gammaH1

# B0 Field in Tesla, Static Magnetic field (B0) along Z
System.B0 = 9.4

# Rotating Frame Frequency
System.OmegaRF[0] = -System.gammaH1*System.B0
System.OmegaRF[1] = -System.gammaH1*System.B0

# Offset Frequency in rotating frame (Hz)
System.Offset[0] = 10.0
System.Offset[1] = 50.0
```



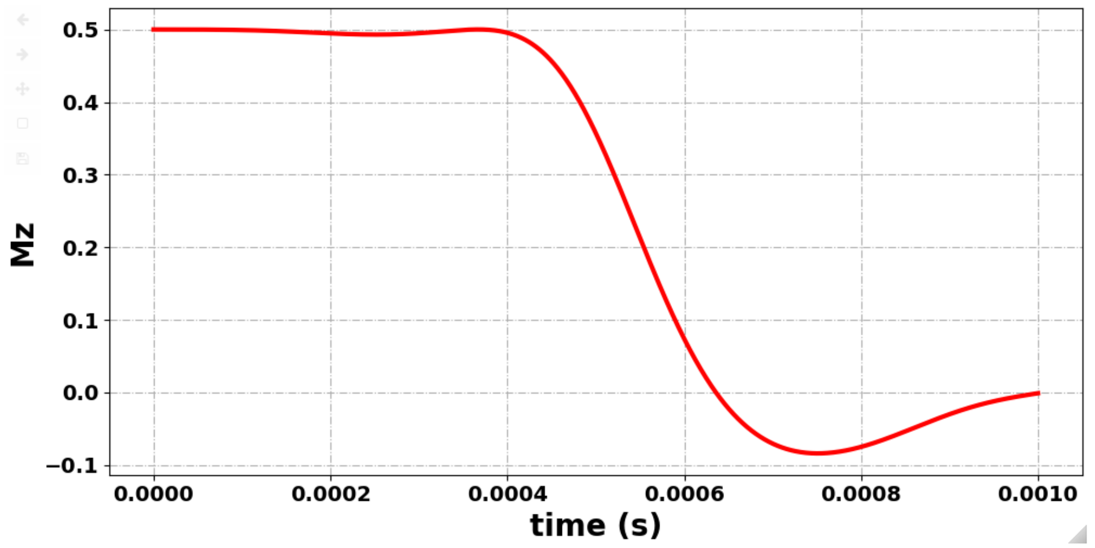
```
# Define J coupling between Spins (Jlist[i][j], j > i)
System.Jlist[0][1] = 5.0

# Define paris of spins coupled by dipolar interaction
System.DipolePairs = [(0,1)]

# Define initial and final Spin Temperature
System.Ispintemp[0] = 300.0
System.Ispintemp[1] = 300.0
System.Fspintemp[0] = 300.0
System.Fspintemp[1] = 300.0

# Relaxation Process
System.Rprocess = "Auto-correlated Dipolar Homonuclear"
System.RelaxParDipole_tau = 10.0e-12
System.RelaxParDipole_bIS = [30.0e3]
```

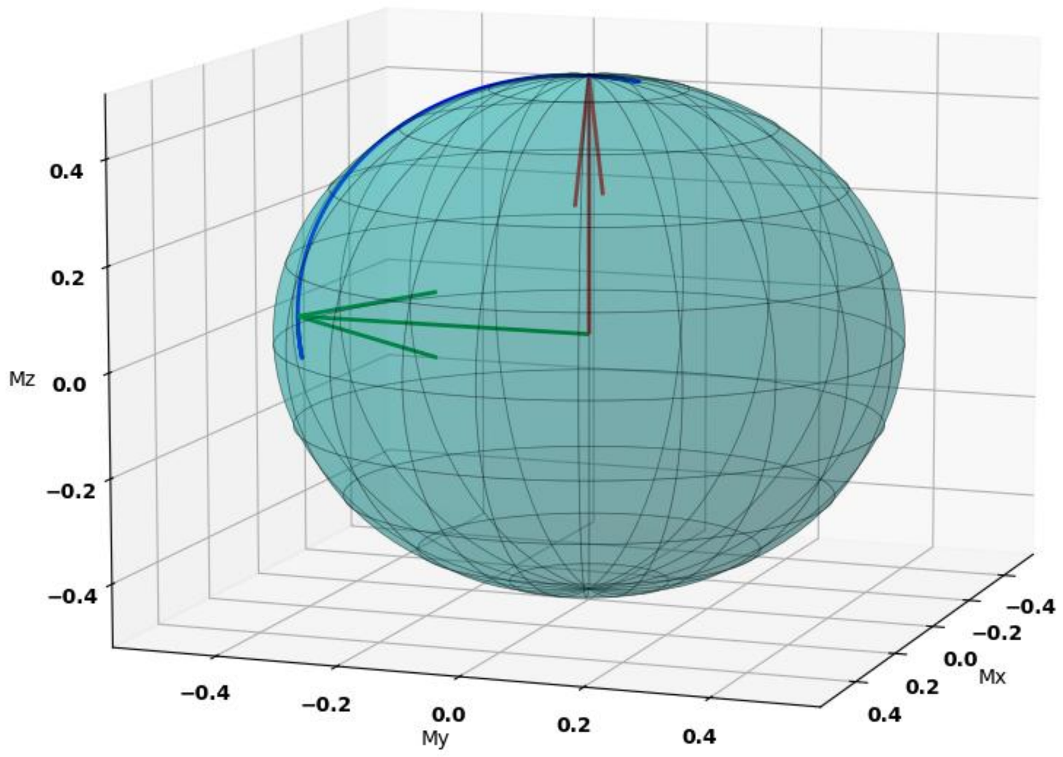
PyOR Mag. Res.: Read Shaped Pulse Bruker File

```
# Shape file
pulseFile = '/opt/topspin4.1.4/exp/stan/nmr/lists/wave/Rsnob.1000' # Rsnob.1000 or square.1000 or Gaus1.1000
pulseLength = 1000.0e-6
RotatioAngle = 90.0
t, amp, phase = System.ShapedPulse_Bruker(pulseFile,pulseLength,RotatioAngle)
```

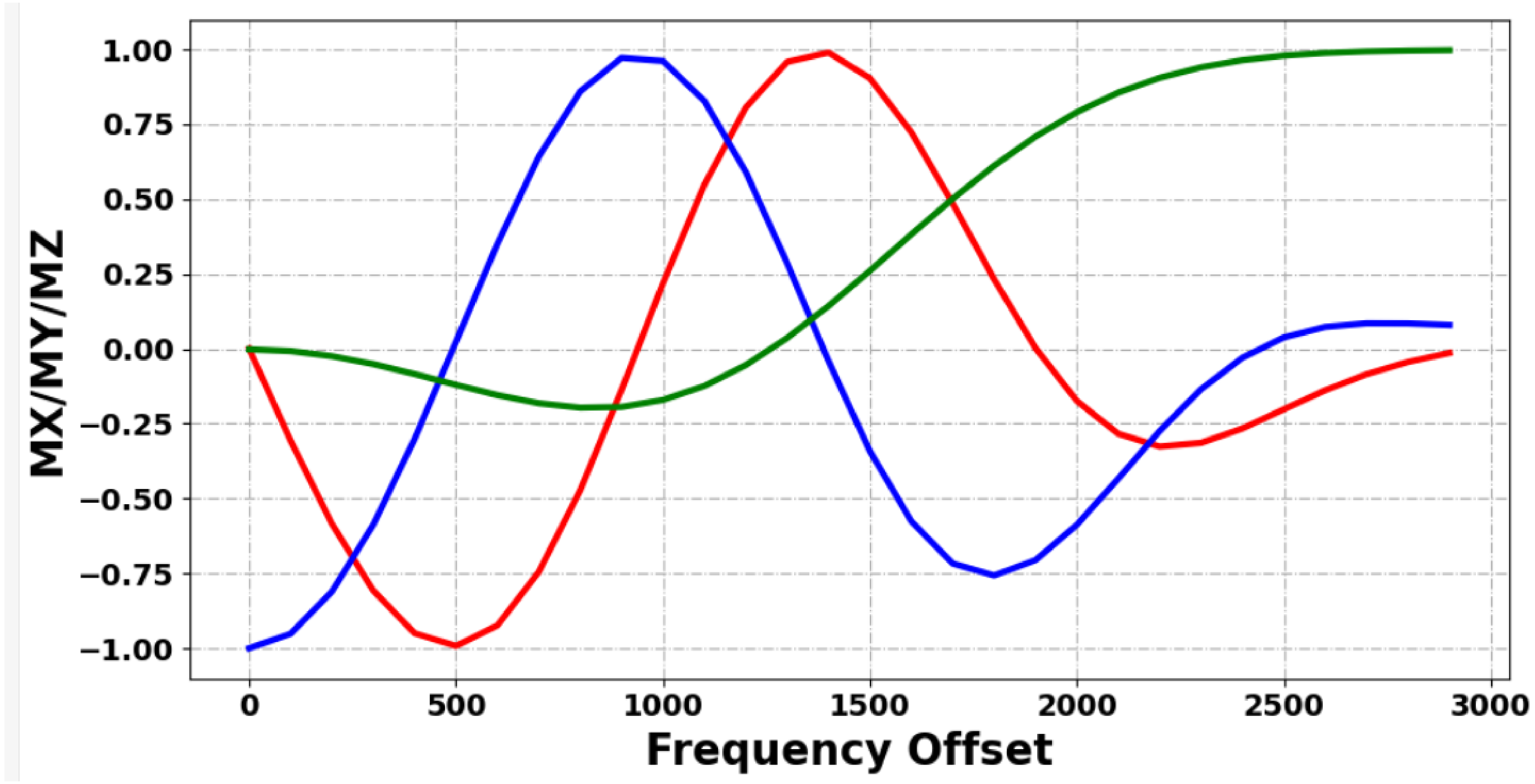


PyOR Mag. Res.: Simulate Rsnob



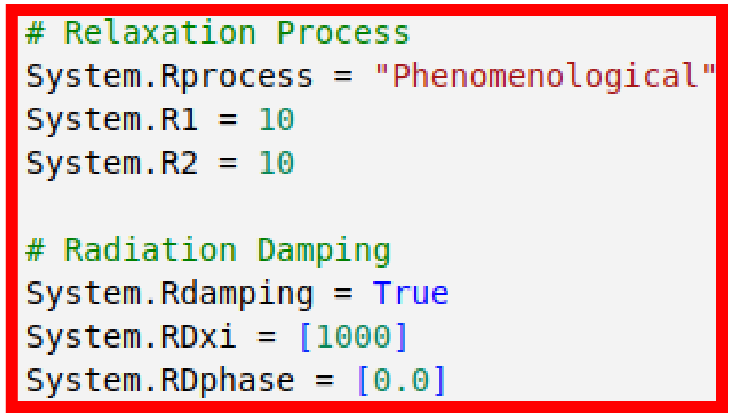


PyOR Mag. Res.: Simulate Rsnob Magnetization Frequency Profile

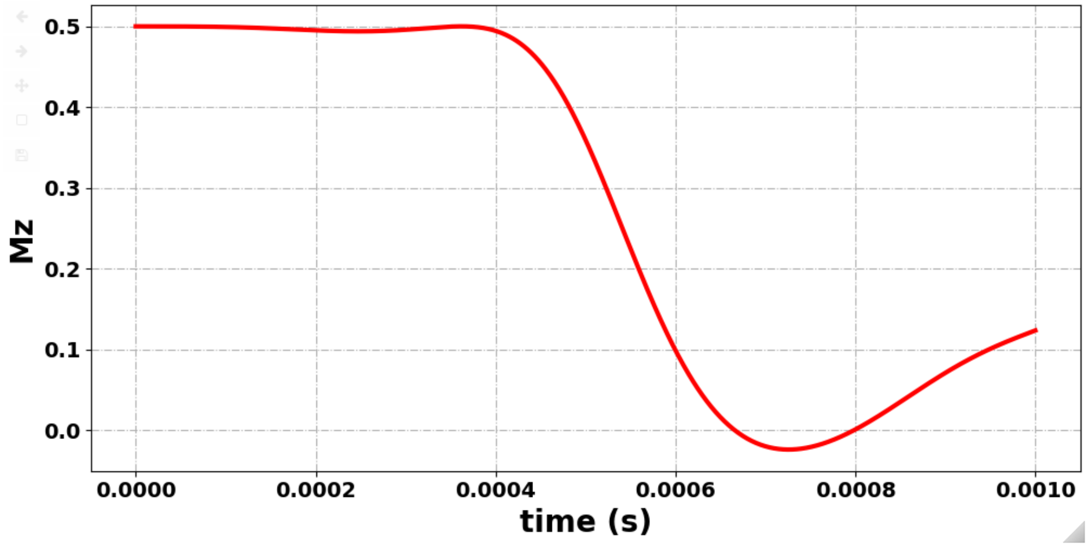


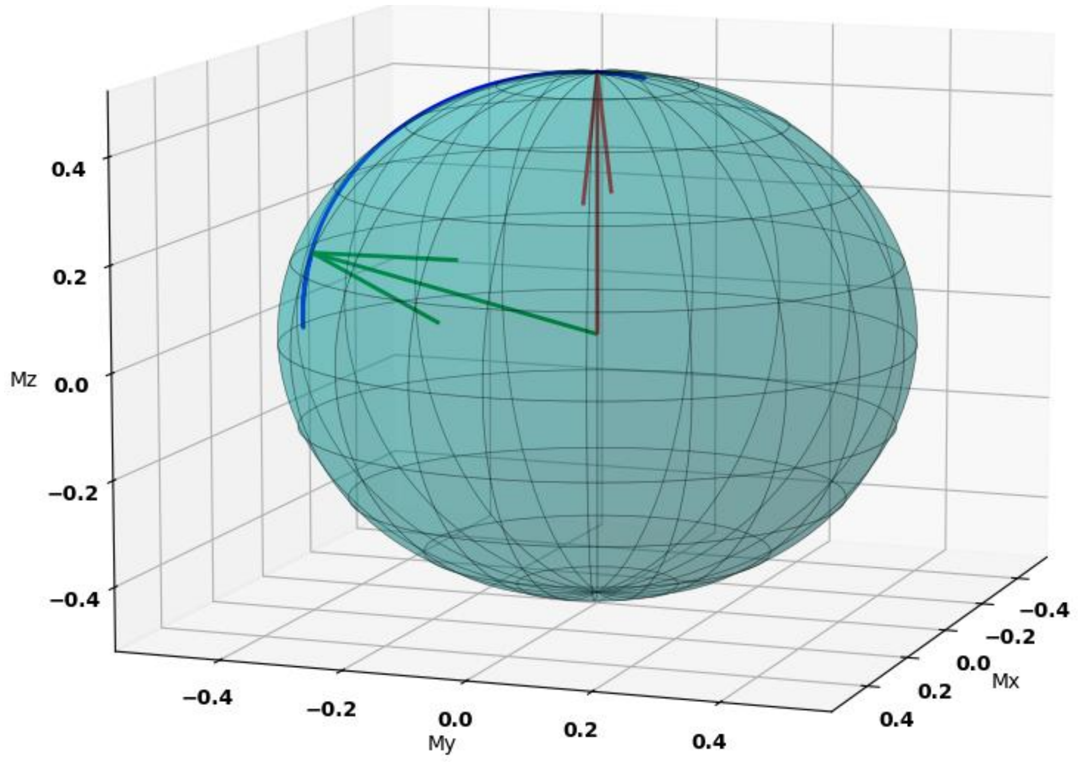
Mx (Blue), My (Red) and Mz (Green)

PyOR Mag. Res.: Simulate Rsnob with Relaxation and Radiation damping



```
# Relaxation Process
System.Rprocess = "Phenomenological"
System.R1 = 10
System.R2 = 10

# Radiation Damping
System.Rdamping = True
System.RDxi = [1000]
System.RDphase = [0.0]
```





PyOR Mag. Res.: Lindblad master equation (Hilbert space)

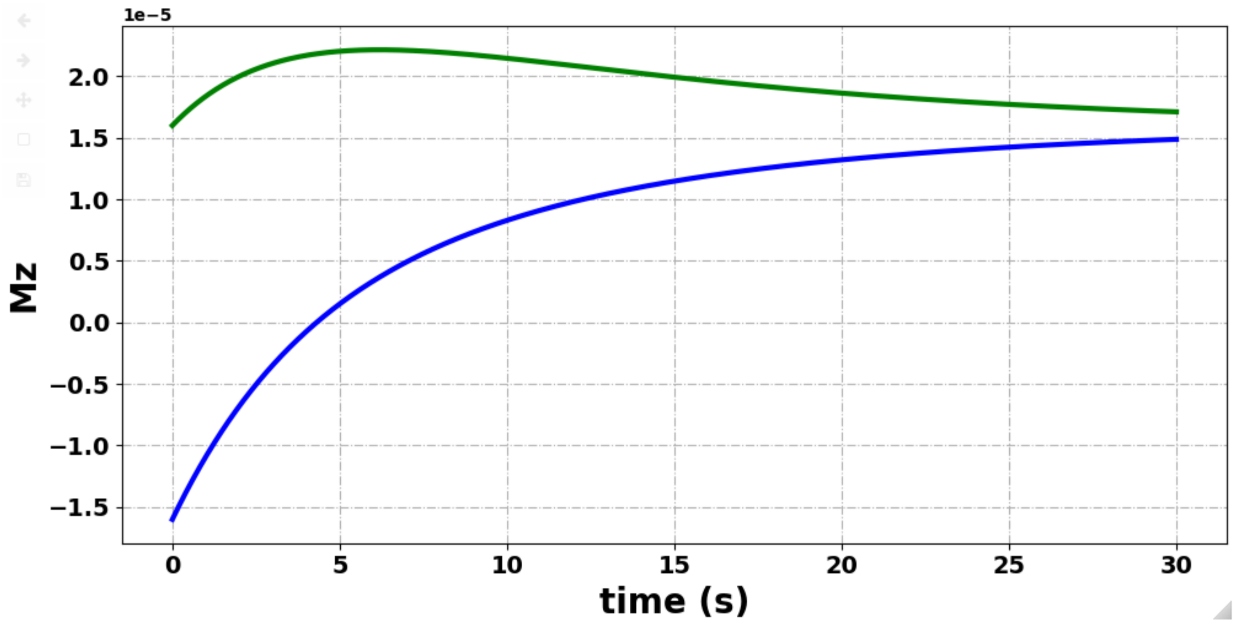
```
# Master Equation
System.PropagationSpace = "Hilbert"
System.MasterEquation = "Lindblad"
```

```
Thermal_DensMatrix = True

if Thermal_DensMatrix:
    # High Temperature
    HT_approx = False

    # Initial Density Matrix
    rho_in = System.EqulibriumDensityMatrix(System.Ispintemp,HT_approx)

    # Equlibrium Density Matrix
    rhoeq = System.EqulibriumDensityMatrix(System.Fspintemp,HT_approx)
```

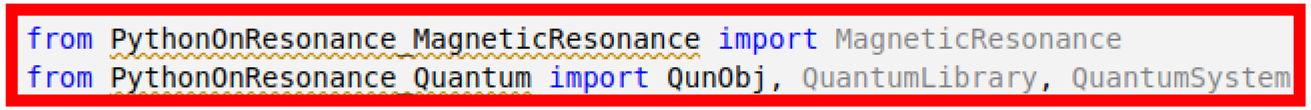
```
System.AcqDT = 0.0001
System.AcqAQ = 30.0
System.OdeMethod = 'DOP853'
System.PropagationMethod = "ODE Solver Lindblad"

start_time = time.time()
t, rho_t = System.Evolution(rhoeq,rho,Hz+Hj)
end_time = time.time()
timetaken = end_time - start_time
print("Total time = %s seconds " % (timetaken))
```

- Example: NOE

```
# Relaxation Process
System.Rprocess = "Auto-correlated Dipolar Homonuclear"
System.RelaxParDipole_tau = 10.0e-12
System.RelaxParDipole_bIS = [30.0e3]
```

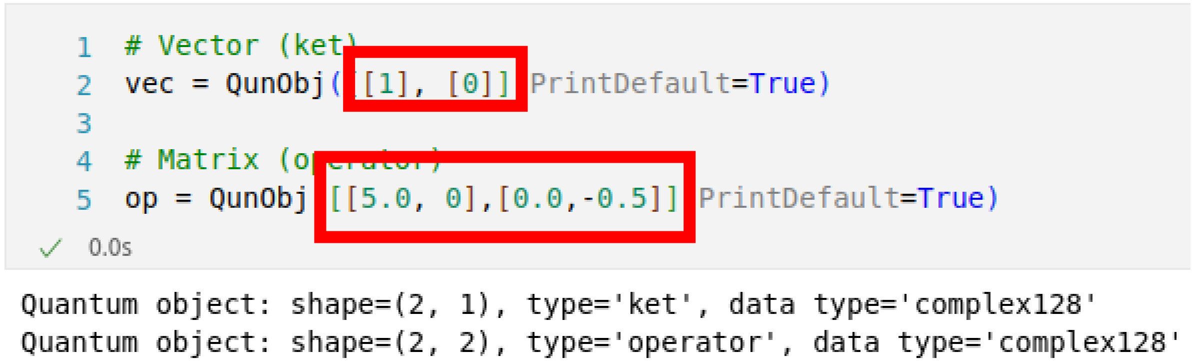


PyOR Quantum: Quantum Objects



```python
from PythonOnResonance_MagneticResonance import MagneticResonance
from PythonOnResonance_Quantum import QunObj, QuantumLibrary, QuantumSystem
```

- Quantum Objects

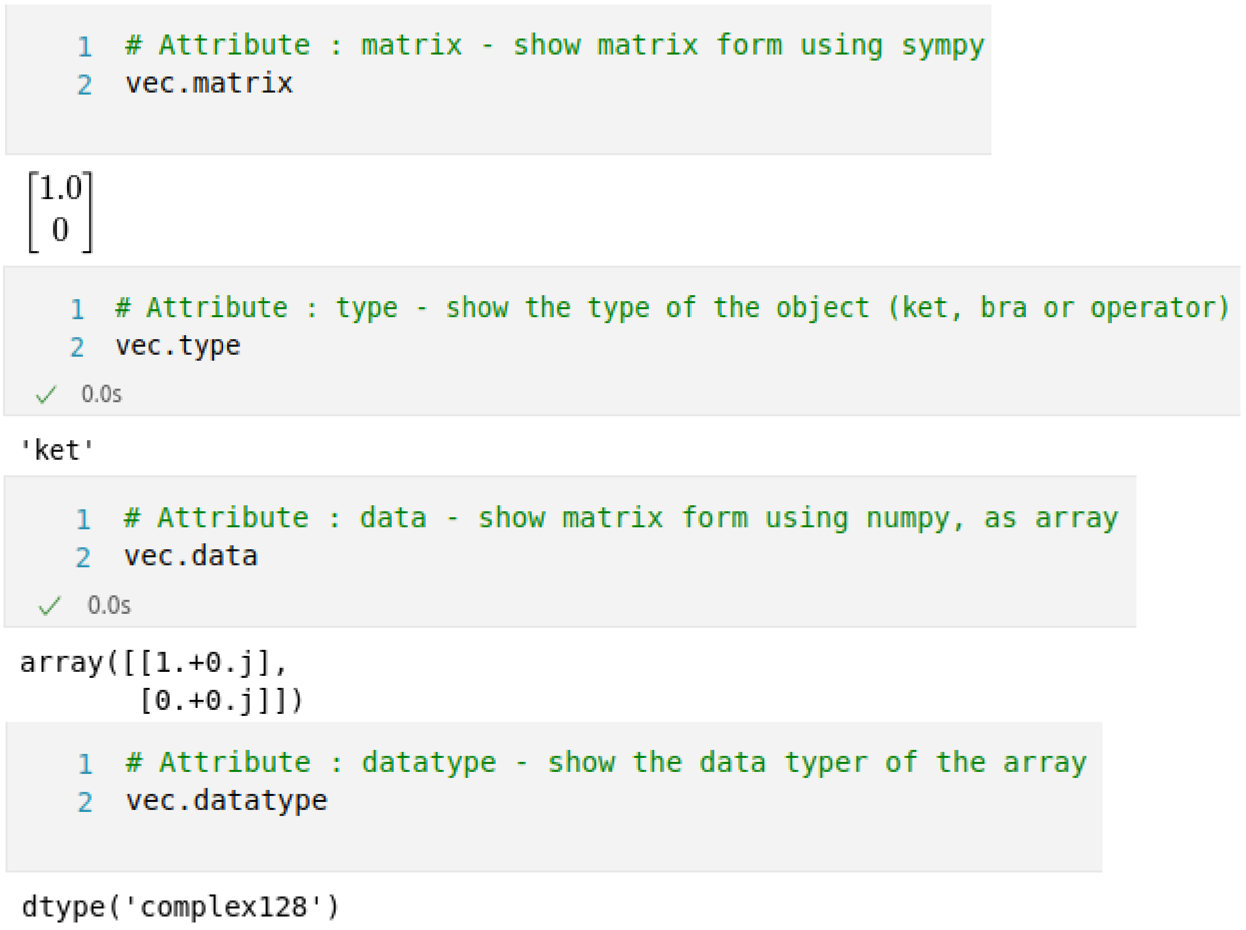


```python
# Vector (ket)
vec = QunObj([[1], [0]], PrintDefault=True)

# Matrix (operator)
op = QunObj([[5.0, 0],[0.0,-0.5]], PrintDefault=True)
```

✓ 0.0s

```
Quantum object: shape=(2, 1), type='ket', data type='complex128'
Quantum object: shape=(2, 2), type='operator', data type='complex128'
```

- Attributes



```python
# Attribute : matrix - show matrix form using sympy
vec.matrix
```

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

```python
# Attribute : type - show the type of the object (ket, bra or operator)
vec.type
```

✓ 0.0s

```
'ket'
```

```python
# Attribute : data - show matrix form using numpy, as array
vec.data
```

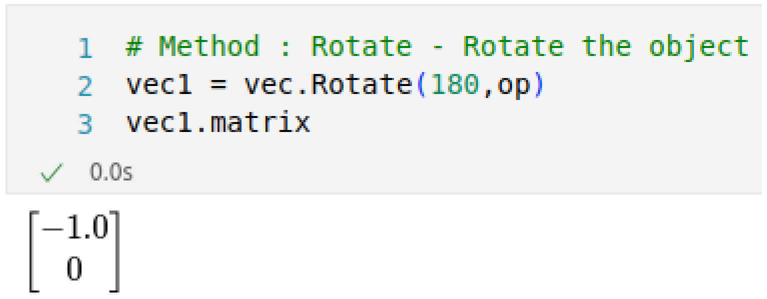
✓ 0.0s

```
array([[1.+0.j],
       [0.+0.j]])
```

```python
# Attribute : datatype - show the data typer of the array
vec.datatype
```

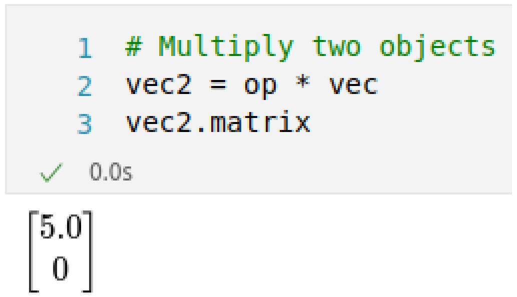
```
dtype('complex128')
```

PyOR Quantum: Quantum Objects (methods)



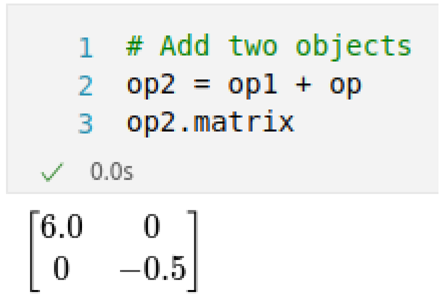
```
# Method : Rotate - Rotate the object
vec1 = vec.Rotate(180,op)
vec1.matrix
```
✓ 0.0s

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



```
# Multiply two objects
vec2 = op * vec
vec2.matrix
```
✓ 0.0s

$$\begin{bmatrix} 5.0 \\ 0 \end{bmatrix}$$

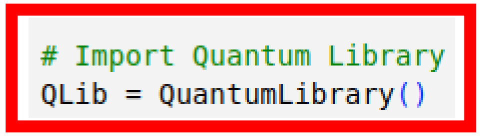


```
# Add two objects
op2 = op1 + op
op2.matrix
```
✓ 0.0s

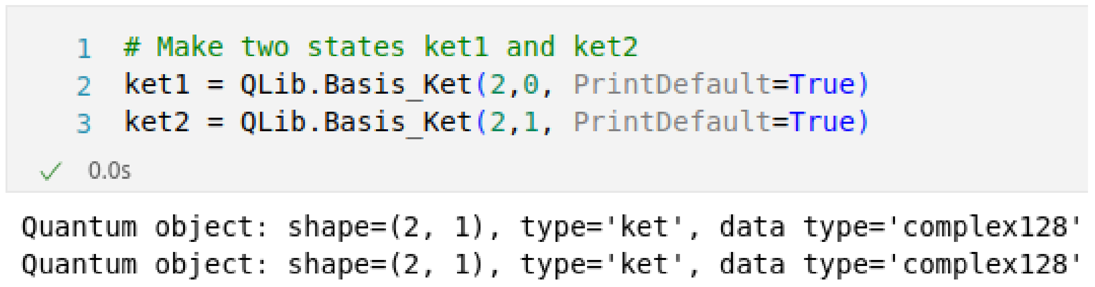
$$\begin{bmatrix} 6.0 & 0 \\ 0 & -0.5 \end{bmatrix}$$

- Norm()
- Normalize()
- Conjugate()
- Transpose()
- ConjugateTranspose()
- Innerproduct()
- OuterProduct()
- Adjoint()
- …

PyOR Quantum: Quantum Library (methods)



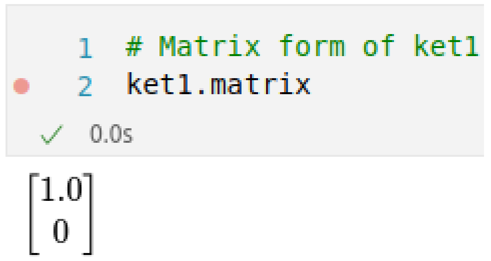
```python
# Import Quantum Library
QLib = QuantumLibrary()
```



```python
# Make two states ket1 and ket2
ket1 = QLib.Basis_Ket(2,0, PrintDefault=True)
ket2 = QLib.Basis_Ket(2,1, PrintDefault=True)
```

✓ 0.0s

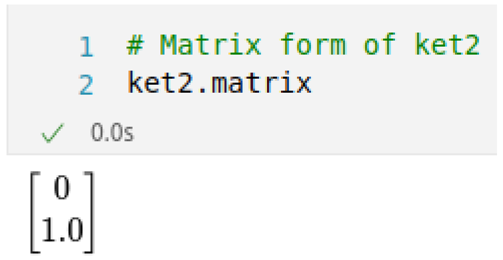
```
Quantum object: shape=(2, 1), type='ket', data type='complex128'
Quantum object: shape=(2, 1), type='ket', data type='complex128'
```



```python
# Matrix form of ket1
ket1.matrix
```

✓ 0.0s

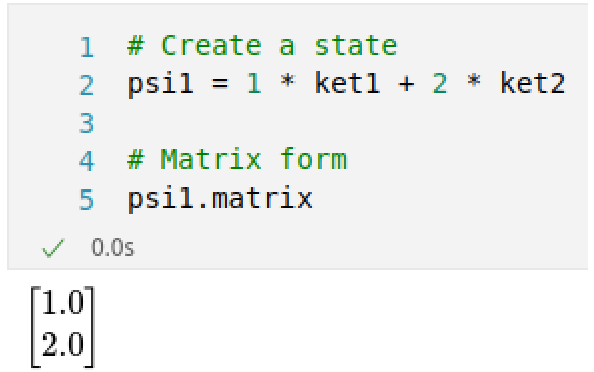
$$\begin{bmatrix} 1.0 \\ 0 \end{bmatrix}$$



```python
# Matrix form of ket2
ket2.matrix
```

✓ 0.0s

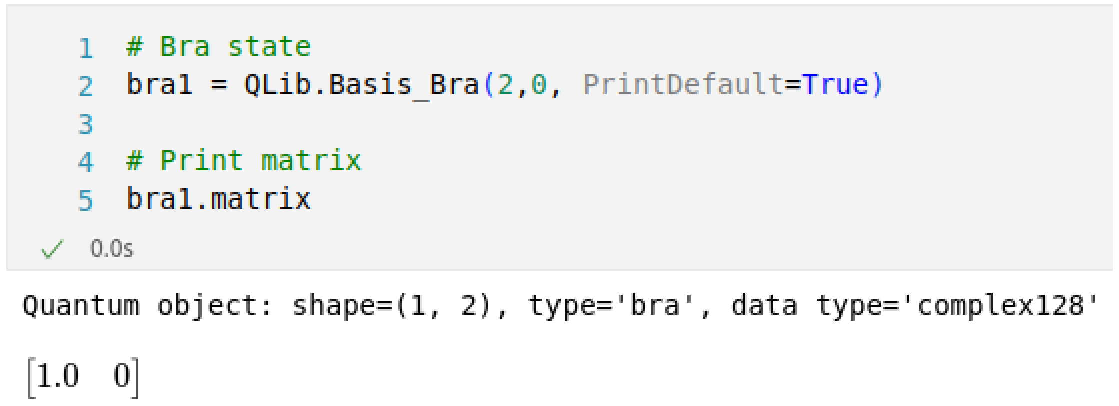
$$\begin{bmatrix} 0 \\ 1.0 \end{bmatrix}$$



```python
# Create a state
psi1 = 1 * ket1 + 2 * ket2

# Matrix form
psi1.matrix
```

✓ 0.0s

$$\begin{bmatrix} 1.0 \\ 2.0 \end{bmatrix}$$



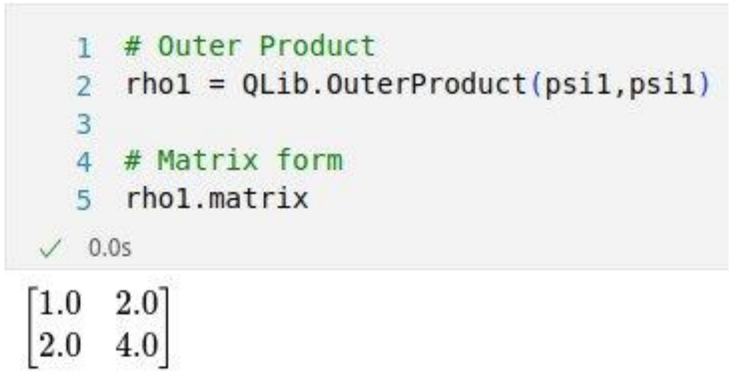
```python
# Bra state
bra1 = QLib.Basis_Bra(2,0, PrintDefault=True)

# Print matrix
bra1.matrix
```

✓ 0.0s

```
Quantum object: shape=(1, 2), type='bra', data type='complex128'
```

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

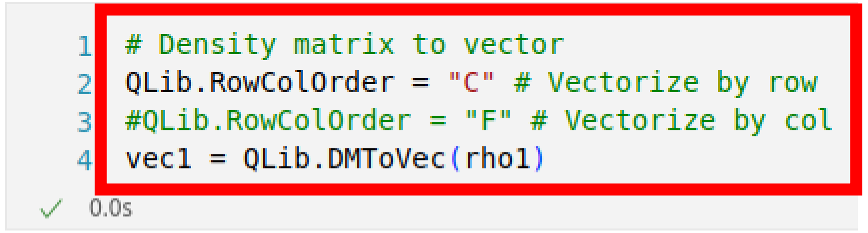
PyOR Quantum: Quantum Library (methods) …



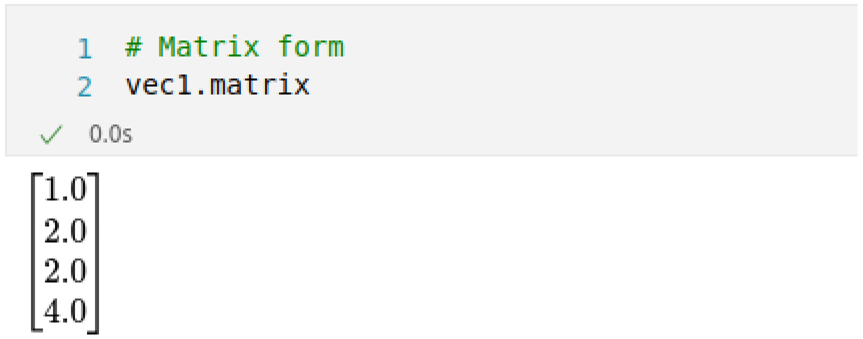
```
# Outer Product
rho1 = QLib.OuterProduct(psi1,psi1)

# Matrix form
rho1.matrix
```
✓ 0.0s

$$\begin{bmatrix}1.0 & 2.0\\2.0 & 4.0\end{bmatrix}$$

Vectorize Density Matrix

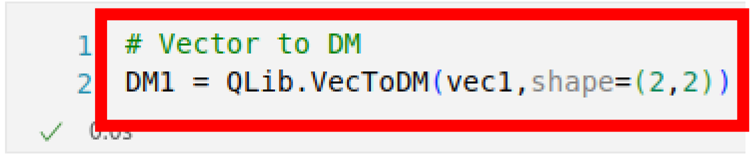


```
# Density matrix to vector
QLib.RowColOrder = "C" # Vectorize by row
#QLib.RowColOrder = "F" # Vectorize by col
vec1 = QLib.DMToVec(rho1)
```
✓ 0.0s



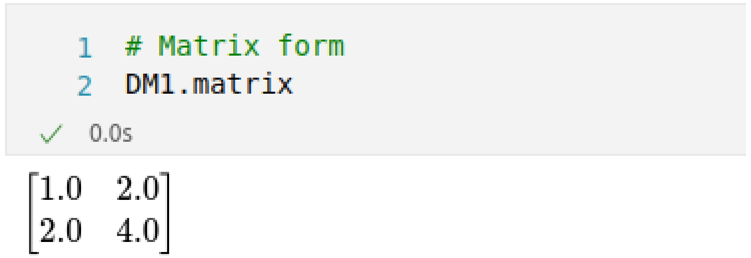
```
# Matrix form
vec1.matrix
```
✓ 0.0s

$$\begin{bmatrix}1.0\\2.0\\2.0\\4.0\end{bmatrix}$$



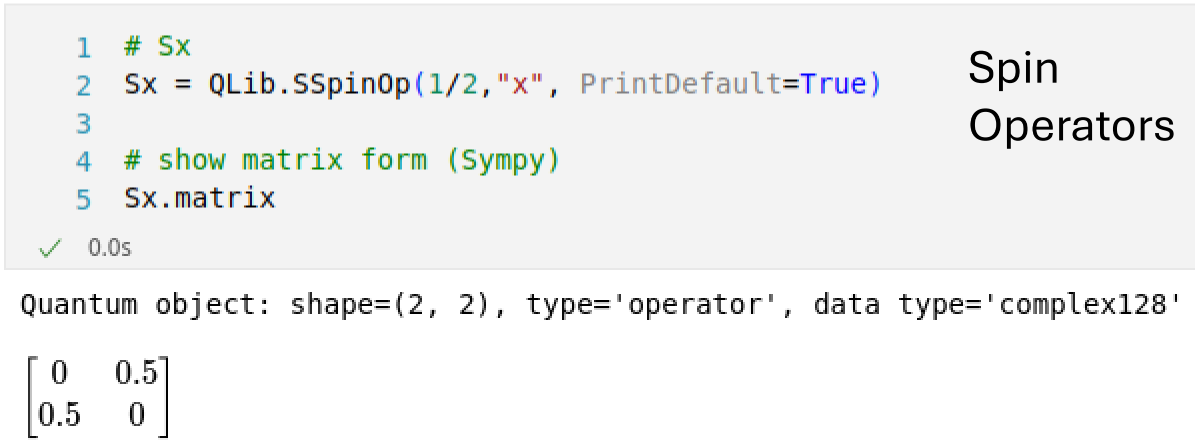
```
# Vector to DM
DM1 = QLib.VecToDM(vec1,shape=(2,2))
```
✓ 0.0s

Vector to Density matrix



```
# Matrix form
DM1.matrix
```
✓ 0.0s

$$\begin{bmatrix}1.0 & 2.0\\2.0 & 4.0\end{bmatrix}$$



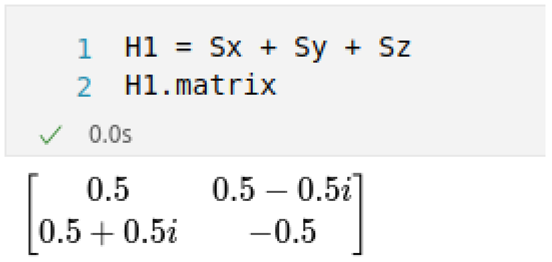
```
# Sx
Sx = QLib.SSpinOp(1/2,"x", PrintDefault=True)

# show matrix form (Sympy)
Sx.matrix
```
✓ 0.0s

```
Quantum object: shape=(2, 2), type='operator', data type='complex128'
```

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

Spin Operators

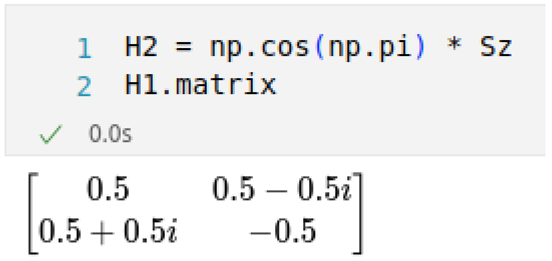
PyOR Quantum: Quantum Library (Hamiltonians, Tensor Products)



```python
H1 = Sx + Sy + Sz
H1.matrix
```

✓ 0.0s

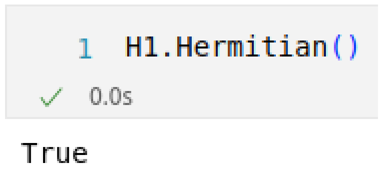
$$\begin{bmatrix} 0.5 & 0.5 - 0.5i \\ 0.5 + 0.5i & -0.5 \end{bmatrix}$$



```python
H2 = np.cos(np.pi) * Sz
H1.matrix
```

✓ 0.0s

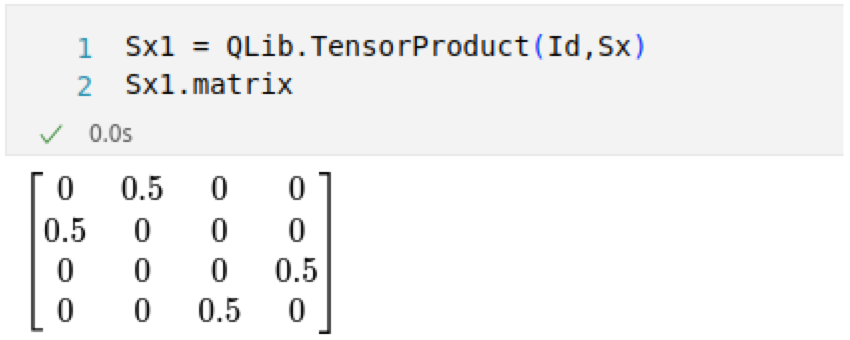
$$\begin{bmatrix} 0.5 & 0.5 - 0.5i \\ 0.5 + 0.5i & -0.5 \end{bmatrix}$$



```python
H1.Hermitian()
```

✓ 0.0s

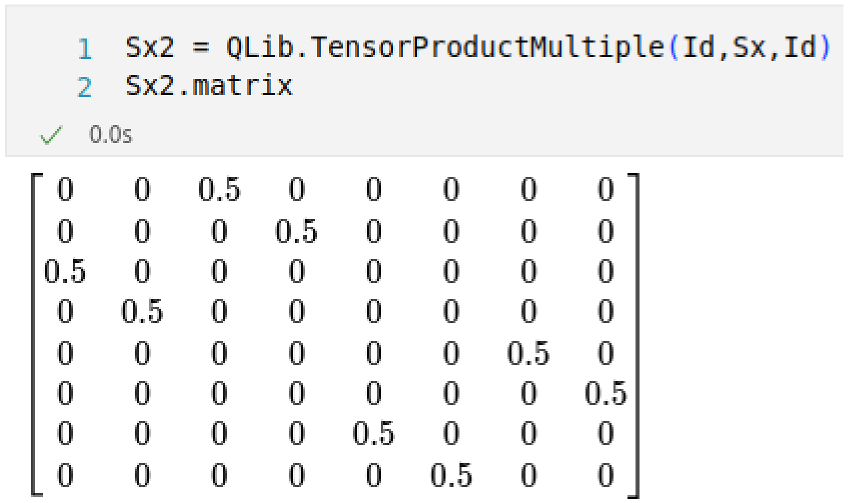
```
True
```



```python
Sx1 = QLib.TensorProduct(Id,Sx)
Sx1.matrix
```

✓ 0.0s

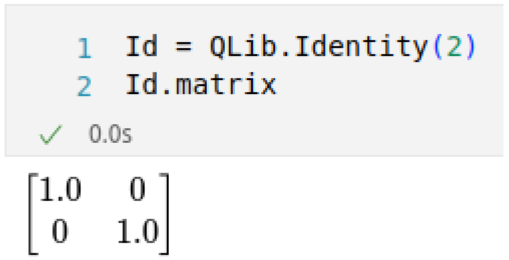
$$\begin{bmatrix} 0 & 0.5 & 0 & 0 \\ 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 \\ 0 & 0 & 0.5 & 0 \end{bmatrix}$$



```python
Sx2 = QLib.TensorProductMultiple(Id,Sx,Id)
Sx2.matrix
```

✓ 0.0s

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

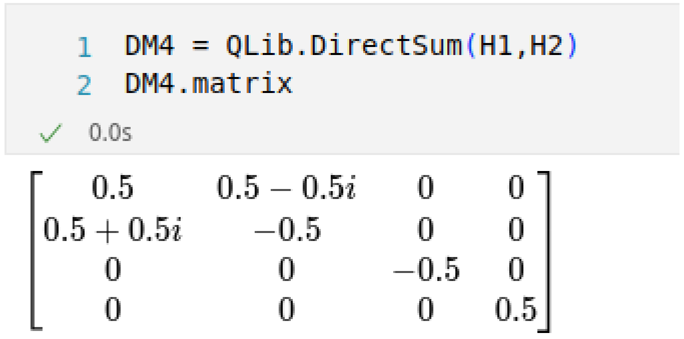


```python
Id = QLib.Identity(2)
Id.matrix
```

✓ 0.0s

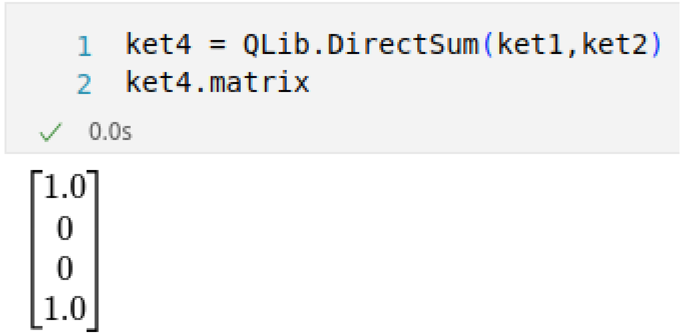
$$\begin{bmatrix} 1.0 & 0 \\ 0 & 1.0 \end{bmatrix}$$

PyOR Quantum: Quantum Library (Direct Sum and Block Extractor)



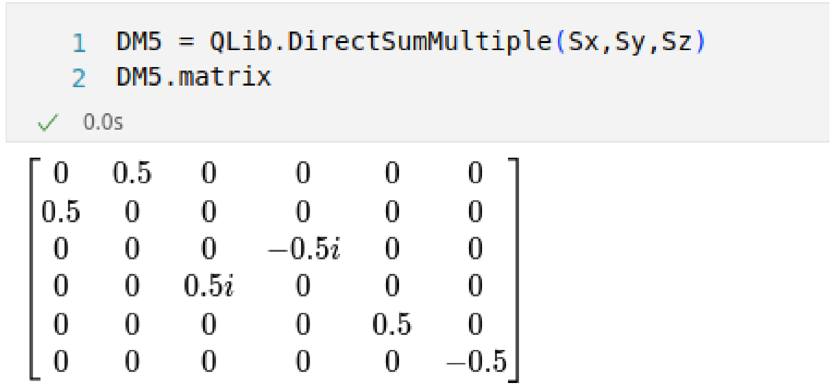
```
DM4 = QLib.DirectSum(H1,H2)
DM4.matrix
```
✓ 0.0s

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



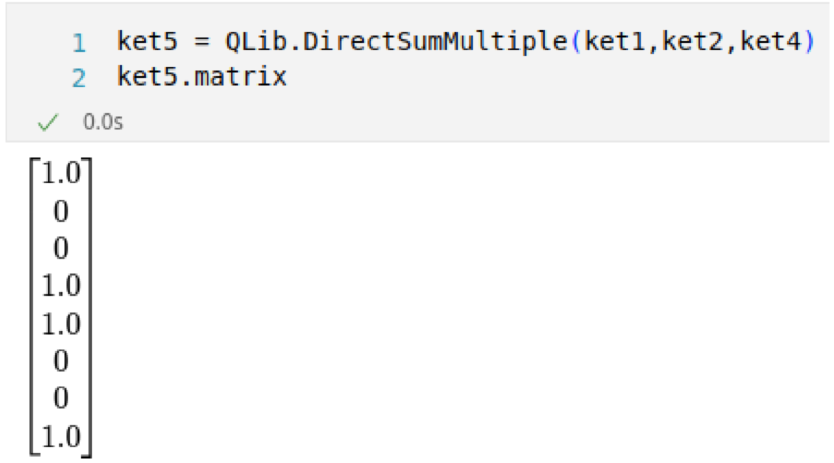
```
ket4 = QLib.DirectSum(ket1,ket2)
ket4.matrix
```
✓ 0.0s

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



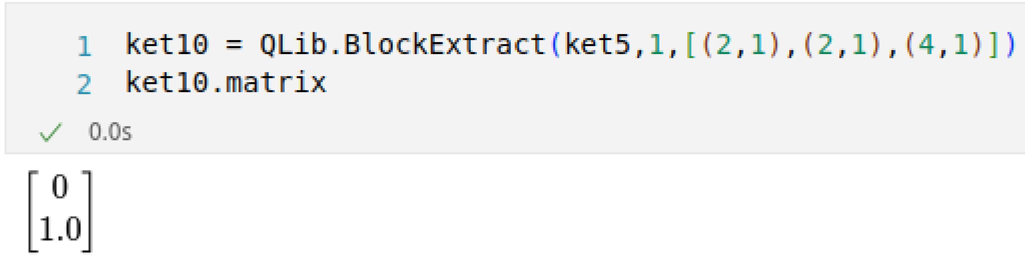
```
DM5 = QLib.DirectSumMultiple(Sx,Sy,Sz)
DM5.matrix
```
✓ 0.0s

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



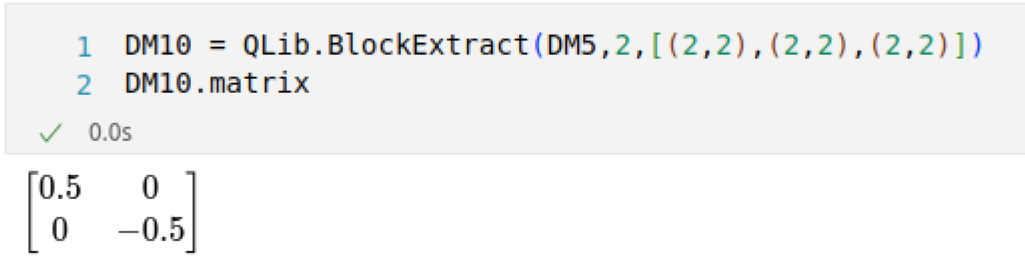
```
ket5 = QLib.DirectSumMultiple(ket1,ket2,ket4)
ket5.matrix
```
✓ 0.0s

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



```
ket10 = QLib.BlockExtract(ket5,1,[(2,1),(2,1),(4,1)])
ket10.matrix
```
✓ 0.0s

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



```
DM10 = QLib.BlockExtract(DM5,2,[(2,2),(2,2),(2,2)])
DM10.matrix
```
✓ 0.0s

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

PyOR Quantum: Quantum Library (Evolution: Schrödinger equation)

```python
SpinList = {"I": 1/2}

QS = QuantumSystem(SpinList)
QLib = QuantumLibrary()

QS.SpinOperator(PrintDefault=False)
```

```python
# Initial State
vec = QLib.Basis_Ket(2,0)
vec.matrix
```
✓ 0.0s

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

```python
# Rotate by 90 deg about X axis
vec = vec.Rotate(90,QS.Ix)
vec.Round(3).matrix
```
✓ 0.0s

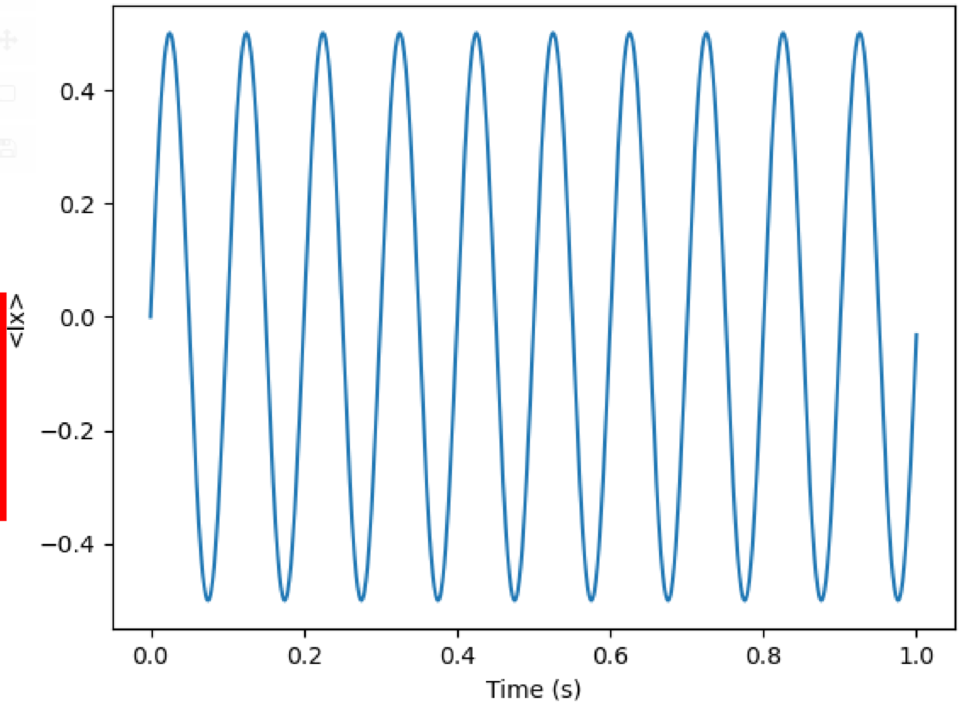
$$\begin{bmatrix} 0.707 \\ -0.707i \end{bmatrix}$$

```python
# Hamiltonian
Hz = 2.0 * np.pi * 10 * QS.Iz
Hz.matrix
```
✓ 0.0s

$$\begin{bmatrix} 31.4159265358979 & 0 \\ 0 & -31.4159265358979 \end{bmatrix}$$

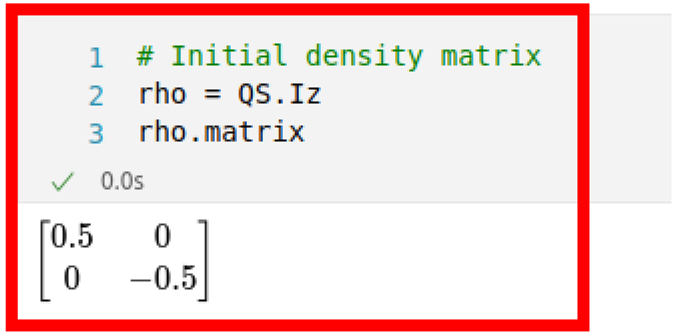
```python
# Evolution
QLib.AcqAQ = 1
QLib.AcqDT = 0.001
t, vect = QLib.Evolve_SE_UProp(vec,Hz)
```
✓ 0.7s

```python
# Expectation
t, signal_SE = QLib.Expectation(t,vect,QS.Ix)
```



PyOR Quantum: Quantum Library (Evolution: Liouville-von Neumann equation)

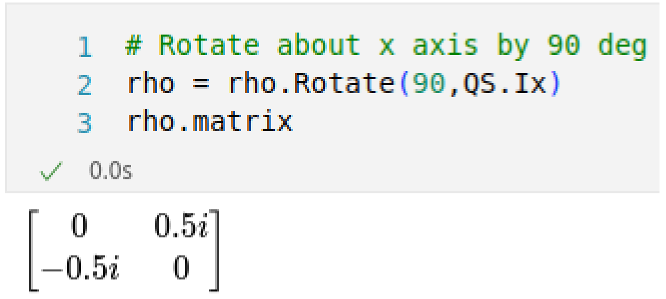
- Hilbert space



```
# Initial density matrix
rho = QS.Iz
rho.matrix
```

✓ 0.0s

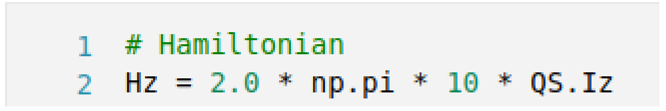
$$\begin{bmatrix} 0.5 & 0 \\ 0 & -0.5 \end{bmatrix}$$



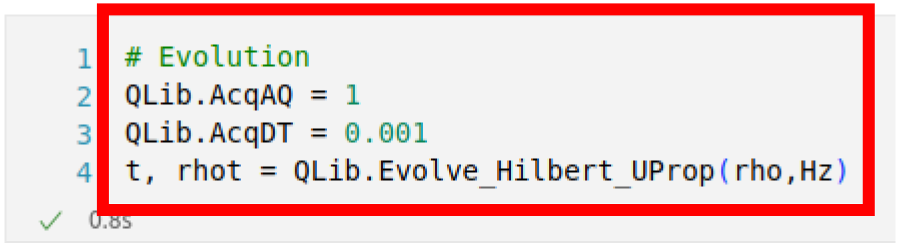
```
# Rotate about x axis by 90 deg
rho = rho.Rotate(90,QS.Ix)
rho.matrix
```

✓ 0.0s

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

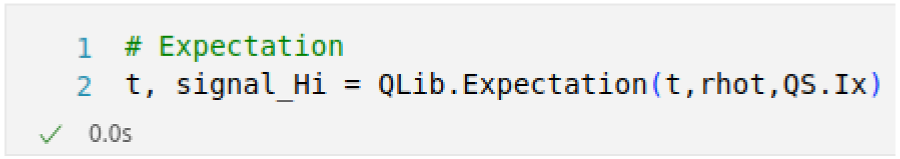


```
# Hamiltonian
Hz = 2.0 * np.pi * 10 * QS.Iz
```



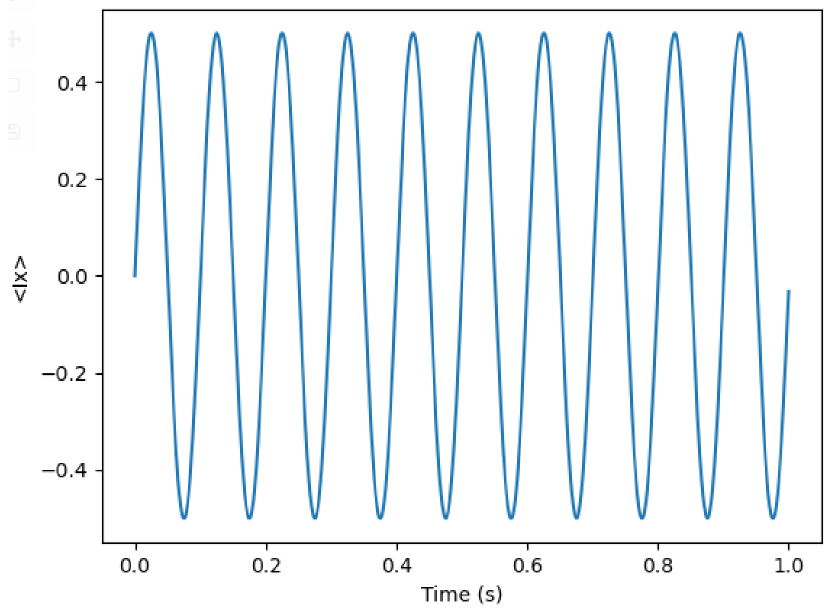
```
# Evolution
QLib.AcqAQ = 1
QLib.AcqDT = 0.001
t, rhot = QLib.Evolve_Hilbert_UProp(rho,Hz)
```

✓ 0.8s



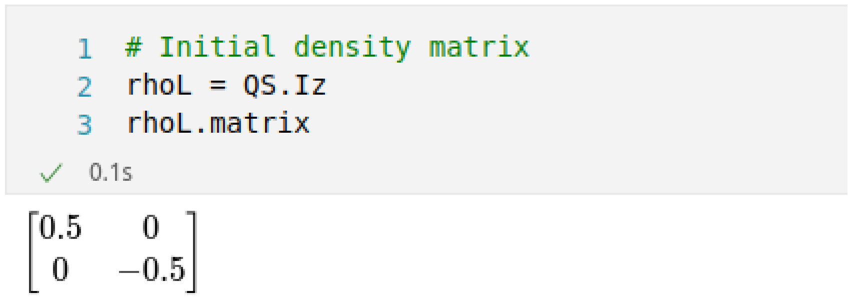
```
# Expectation
t, signal_Hi = QLib.Expectation(t,rhot,QS.Ix)
```

✓ 0.0s



PyOR Quantum: Quantum Library (Evolution: Liouville-von Neumann equation)

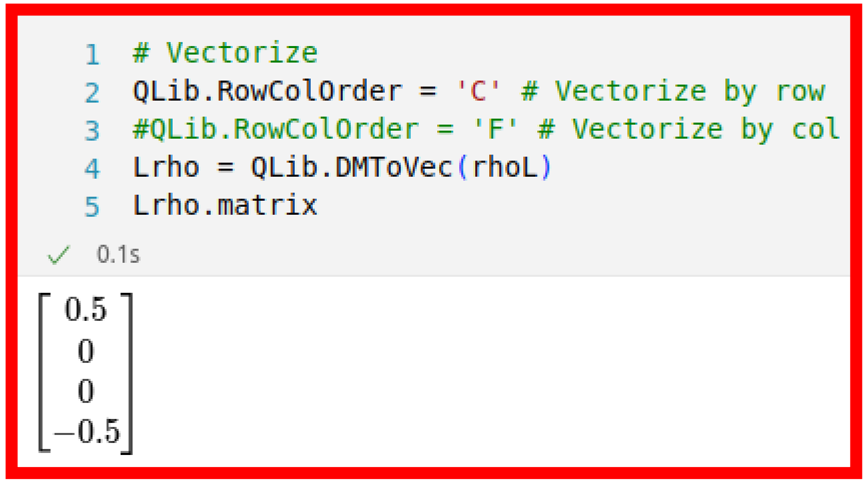
- Liouville space



```python
# Initial density matrix
rhoL = QS.Iz
rhoL.matrix
```

✓ 0.1s

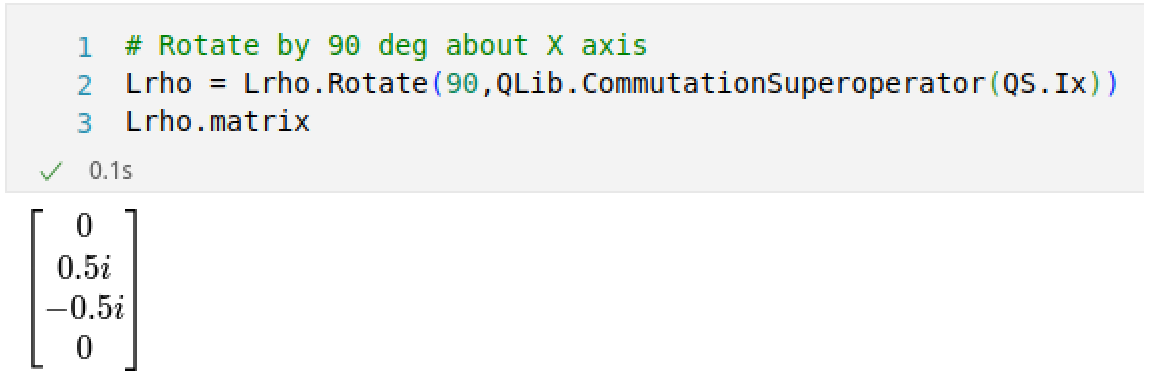
$$\begin{bmatrix} 0.5 & 0 \\ 0 & -0.5 \end{bmatrix}$$



```python
# Vectorize
QLib.RowColOrder = 'C' # Vectorize by row
#QLib.RowColOrder = 'F' # Vectorize by col
Lrho = QLib.DMToVec(rhoL)
Lrho.matrix
```

✓ 0.1s

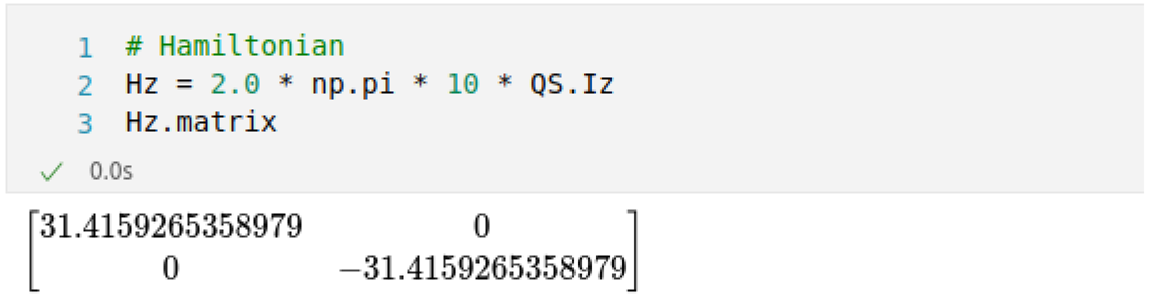
$$\begin{bmatrix} 0.5 \\ 0 \\ 0 \\ -0.5 \end{bmatrix}$$



```python
# Rotate by 90 deg about X axis
Lrho = Lrho.Rotate(90,QLib.CommutationSuperoperator(QS.Ix))
Lrho.matrix
```

✓ 0.1s

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

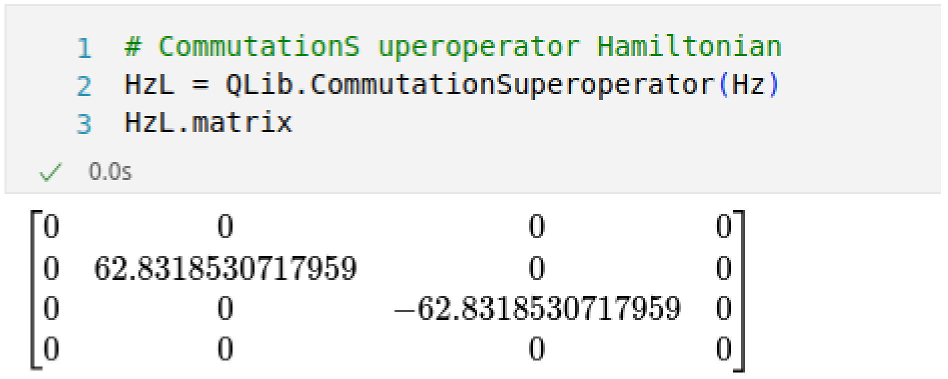


```python
# Hamiltonian
Hz = 2.0 * np.pi * 10 * QS.Iz
Hz.matrix
```

✓ 0.0s

$$\begin{bmatrix} 31.4159265358979 & 0 \\ 0 & -31.4159265358979 \end{bmatrix}$$

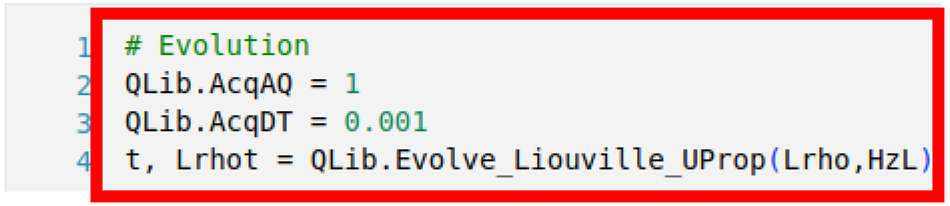
PyOR Quantum: Quantum Library (Evolution: Liouville-von Neumann equation ...)



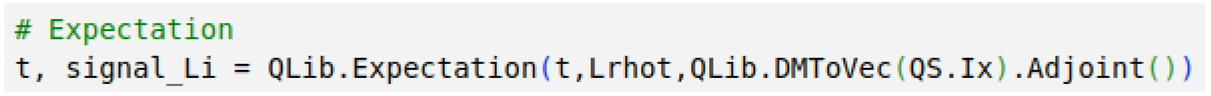
```python
# CommutationS uperoperator Hamiltonian
HzL = QLib.CommutationSuperoperator(Hz)
HzL.matrix
```

✓ 0.0s

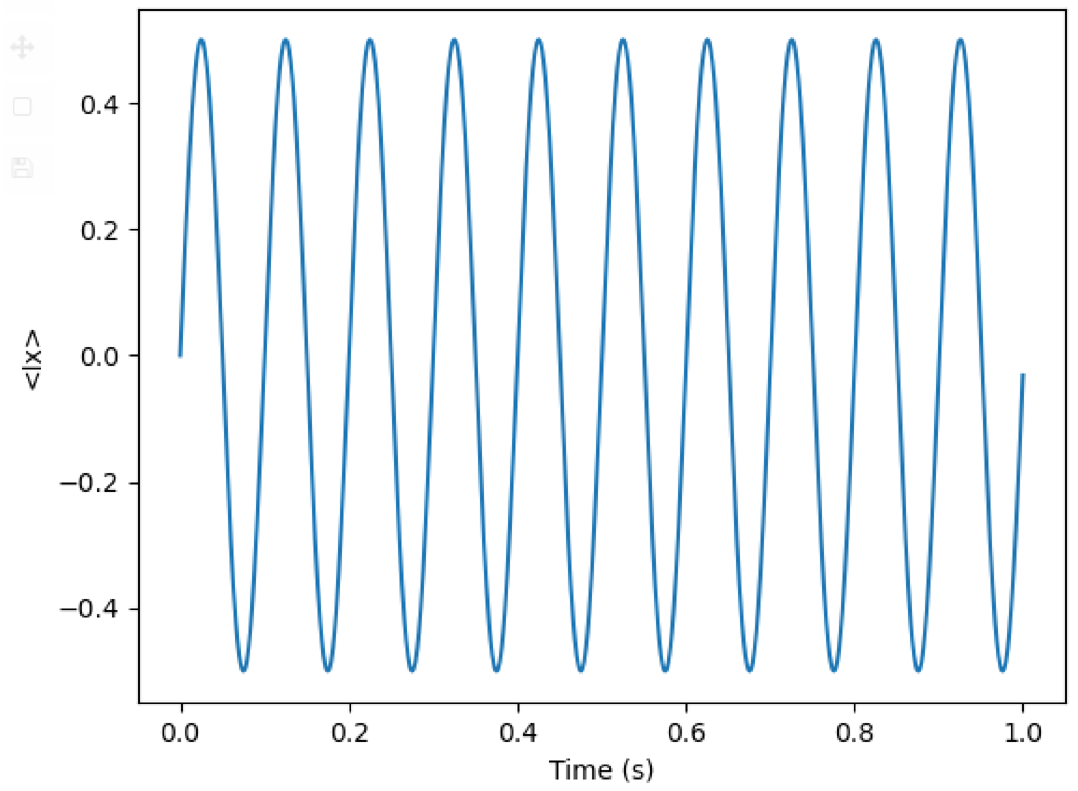
$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 62.8318530717959 & 0 & 0 \\ 0 & 0 & -62.8318530717959 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$



```python
# Evolution
QLib.AcqAQ = 1
QLib.AcqDT = 0.001
t, Lrhot = QLib.Evolve_Liouville_UProp(Lrho,HzL)
```

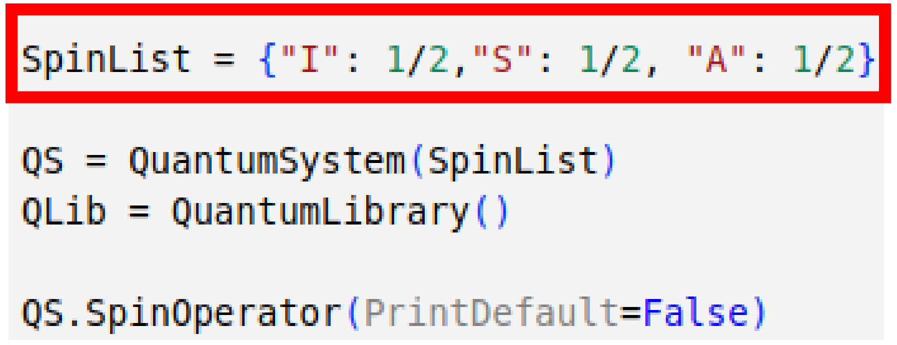


```python
# Expectation
t, signal_Li = QLib.Expectation(t,Lrhot,QLib.DMToVec(QS.Ix).Adjoint())
```

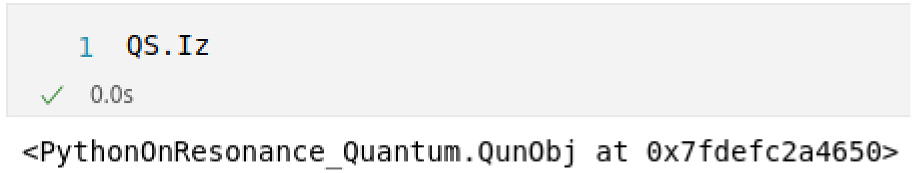


PyOR Quantum: Quantum System

- Spin Operators



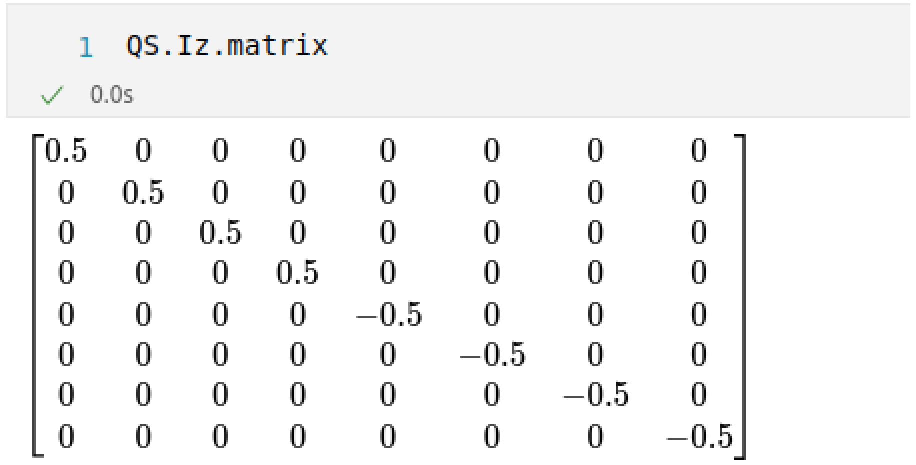
```
SpinList = {"I": 1/2,"S": 1/2, "A": 1/2}

QS = QuantumSystem(SpinList)
QLib = QuantumLibrary()

QS.SpinOperator(PrintDefault=False)
```



```
QS.Iz
```

✓ 0.0s

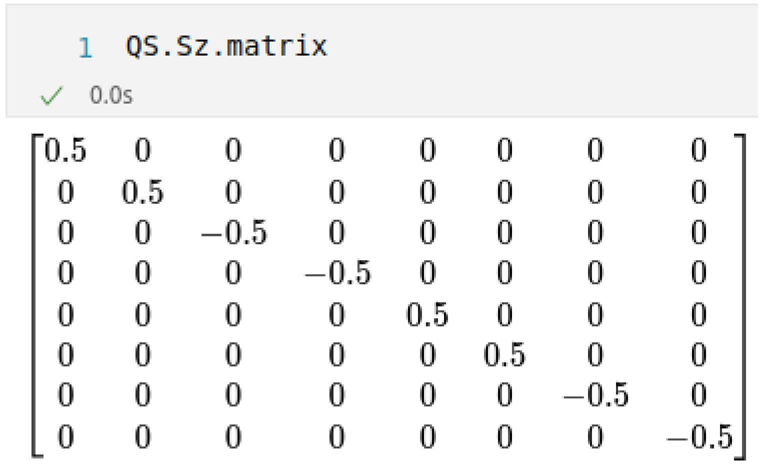
```
<PythonOnResonance_Quantum.QunObj at 0x7fdefc2a4650>
```



```
QS.Iz.matrix
```

✓ 0.0s

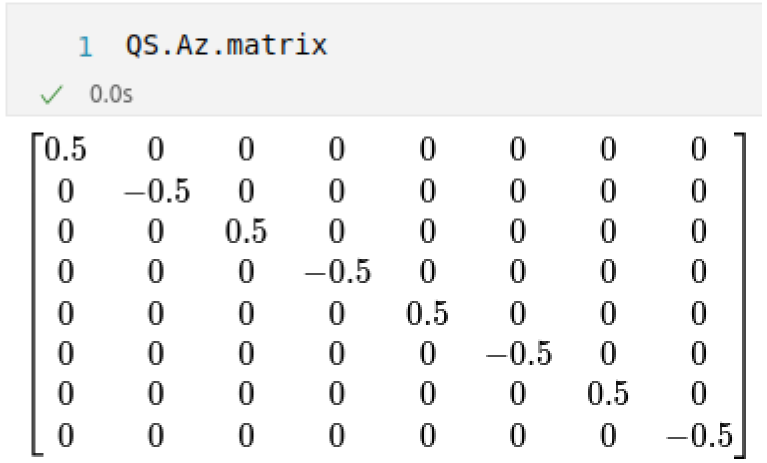
$$\begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 \end{bmatrix}$$



```
QS.Sz.matrix
```

✓ 0.0s

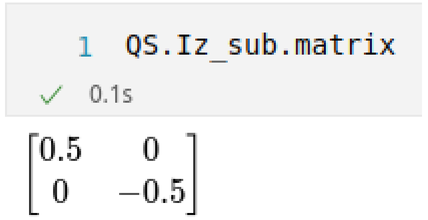
$$\begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 \end{bmatrix}$$



```
QS.Az.matrix
```

✓ 0.0s

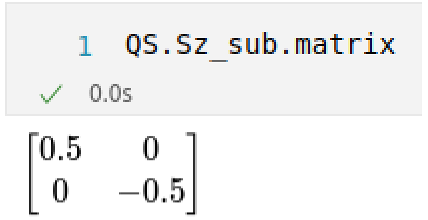
$$\begin{bmatrix} 0.5 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.5 \end{bmatrix}$$



```
QS.Iz_sub.matrix
```

✓ 0.1s

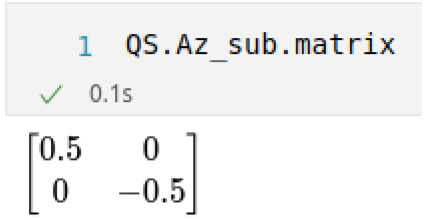
$$\begin{bmatrix} 0.5 & 0 \\ 0 & -0.5 \end{bmatrix}$$



```
QS.Sz_sub.matrix
```

✓ 0.0s

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



```
QS.Az_sub.matrix
```

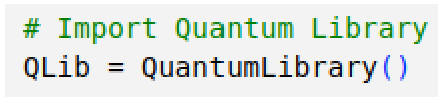
✓ 0.1s

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

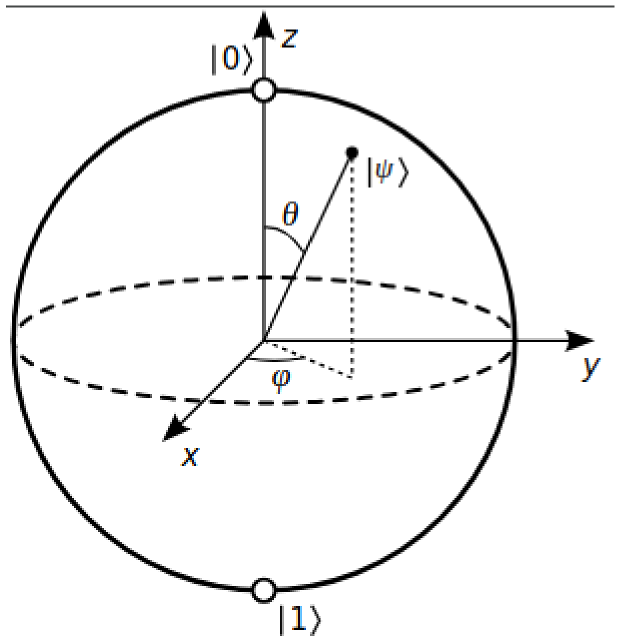
PyOR Quantum: Quantum System

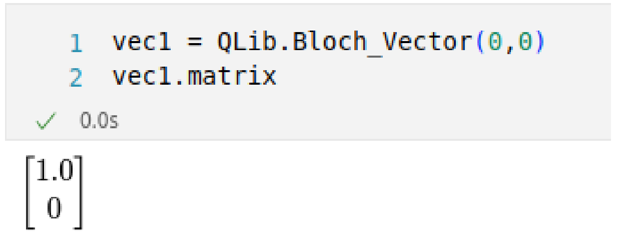
- State
  - QLib.Bloch_Vector
  - QS.StateZeeman
  - QS.States
  - QS.States_General

- Bloch Vector



```python
# Import Quantum Library
QLib = QuantumLibrary()
```

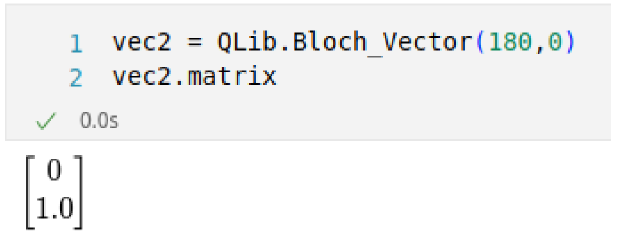




```python
vec1 = QLib.Bloch_Vector(0,0)
vec1.matrix
```

✓ 0.0s

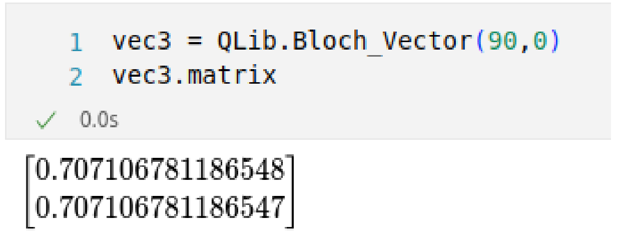
$$\begin{bmatrix}1.0\\0\end{bmatrix}$$



```python
vec2 = QLib.Bloch_Vector(180,0)
vec2.matrix
```

✓ 0.0s

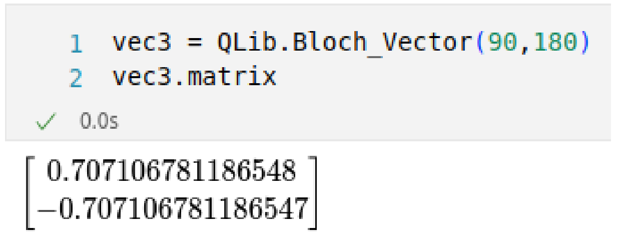
$$\begin{bmatrix}0\\1.0\end{bmatrix}$$



```python
vec3 = QLib.Bloch_Vector(90,0)
vec3.matrix
```

✓ 0.0s

$$\begin{bmatrix}0.707106781186548\\0.707106781186547\end{bmatrix}$$



```python
vec3 = QLib.Bloch_Vector(90,180)
vec3.matrix
```

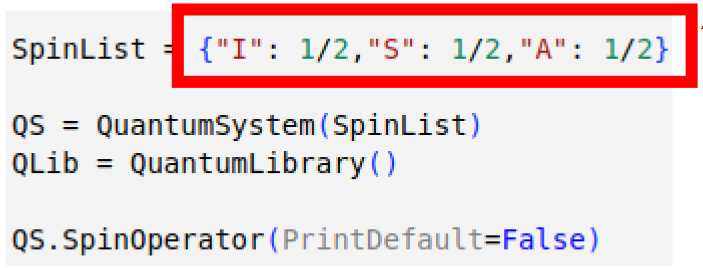
✓ 0.0s

$$\begin{bmatrix}0.707106781186548\\-0.707106781186547\end{bmatrix}$$

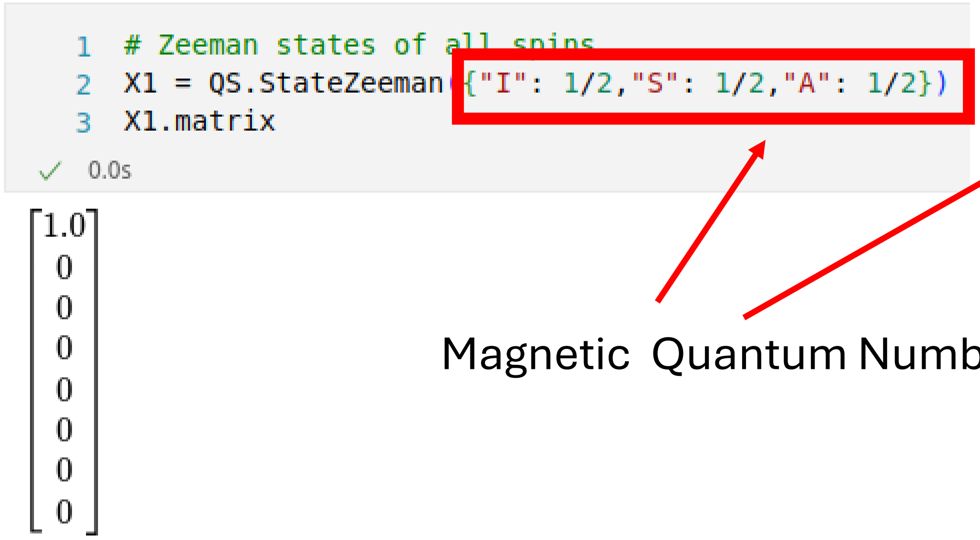
PyOR Quantum: Quantum System

Spin Quantum Number

- Zeeman state



```
SpinList = {"I": 1/2,"S": 1/2,"A": 1/2}

QS = QuantumSystem(SpinList)
QLib = QuantumLibrary()

QS.SpinOperator(PrintDefault=False)
```

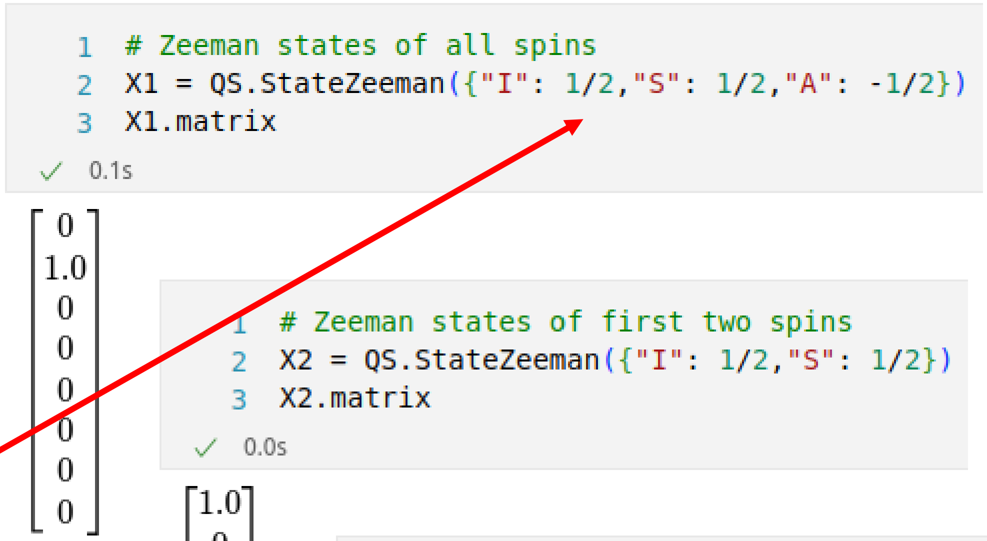


```
# Zeeman states of all spins
X1 = QS.StateZeeman({"I": 1/2,"S": 1/2,"A": 1/2})
X1.matrix
```

✓ 0.0s

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

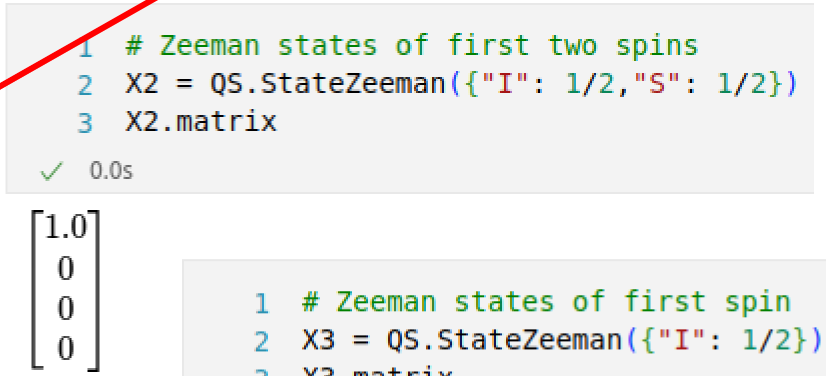
Magnetic Quantum Number



```
# Zeeman states of all spins
X1 = QS.StateZeeman({"I": 1/2,"S": 1/2,"A": -1/2})
X1.matrix
```

✓ 0.1s

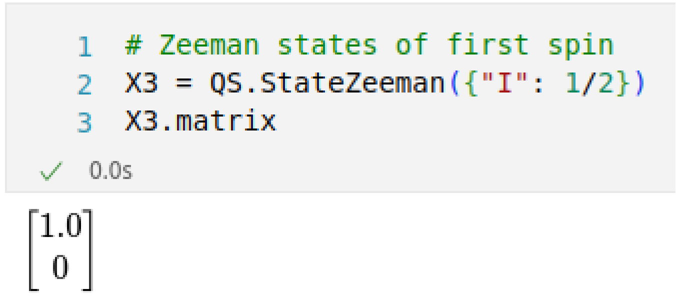
$$\begin{bmatrix} 0 \\ 1.0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



```
# Zeeman states of first two spins
X2 = QS.StateZeeman({"I": 1/2,"S": 1/2})
X2.matrix
```

✓ 0.0s

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



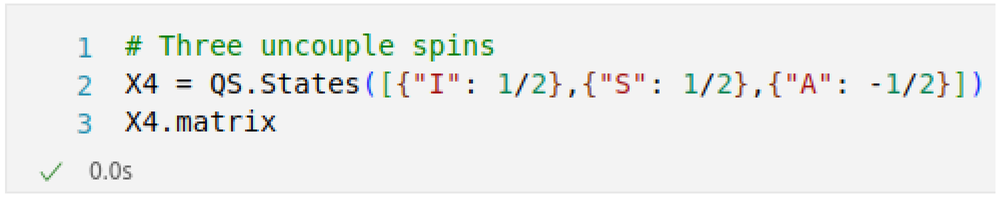
```
# Zeeman states of first spin
X3 = QS.StateZeeman({"I": 1/2})
X3.matrix
```

✓ 0.0s

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

PyOR Quantum: Quantum System

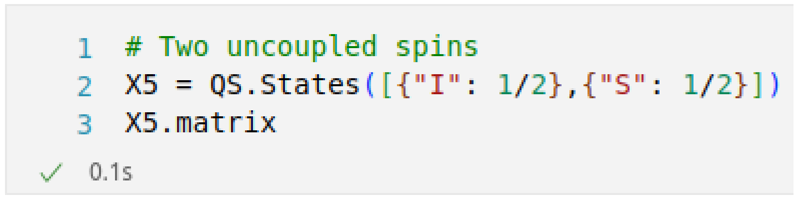
- Coupled State (pairs)



```
# Three uncouple spins
X4 = QS.States([{"I": 1/2},{"S": 1/2},{"A": -1/2}])
X4.matrix
```

✓ 0.0s

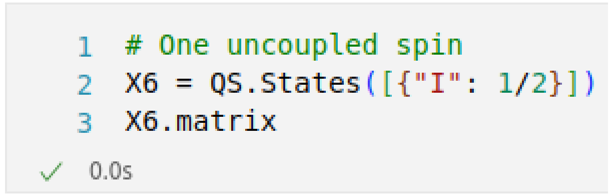
$$\begin{bmatrix} 0 \\ 1.0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



```
# Two uncoupled spins
X5 = QS.States([{"I": 1/2},{"S": 1/2}])
X5.matrix
```

✓ 0.1s

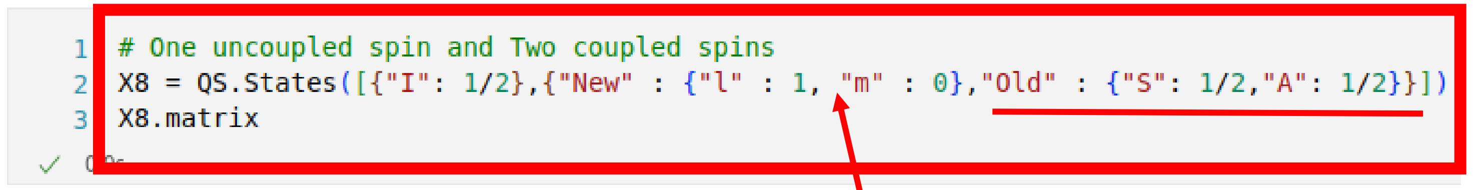
$$\begin{bmatrix} 1.0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$



```
# One uncoupled spin
X6 = QS.States([{"I": 1/2}])
X6.matrix
```

✓ 0.0s

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



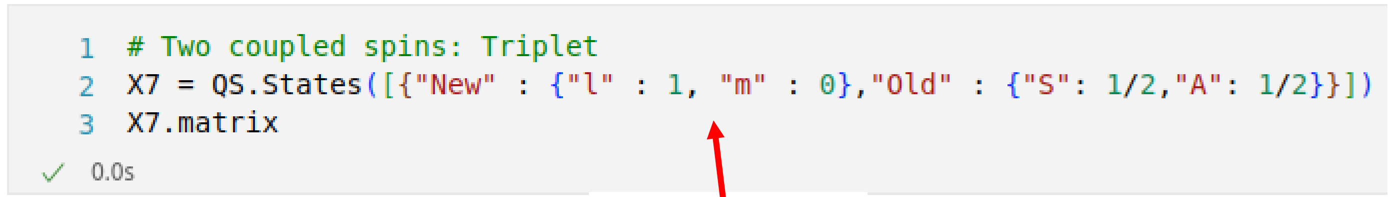
```
# One uncoupled spin and Two coupled spins
X8 = QS.States([{"I": 1/2},{"New" : {"l" : 1, "m" : 0},"Old" : {"S": 1/2,"A": 1/2}}])
X8.matrix
```

✓ 0.0s

$$\begin{bmatrix} 0 \\ 0.707106781186548 \\ 0.707106781186547 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$|\, j,\ m\rangle$$

$$m = -j, -j+1, \ldots, j-1, j$$



```
# Two coupled spins: Triplet
X7 = QS.States([{"New" : {"l" : 1, "m" : 0},"Old" : {"S": 1/2,"A": 1/2}}])
X7.matrix
```

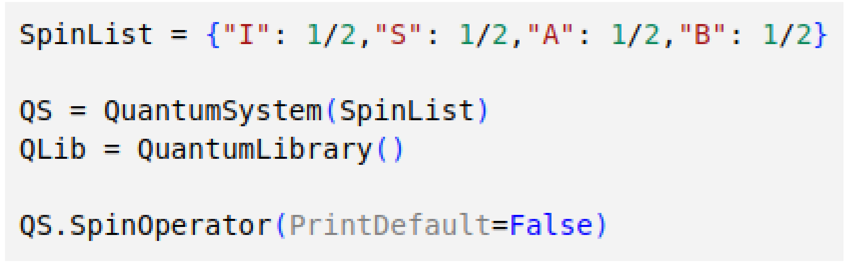
✓ 0.0s

$$\begin{bmatrix} 0 \\ 0.707106781186548 \\ 0.707106781186547 \\ 0 \end{bmatrix}$$

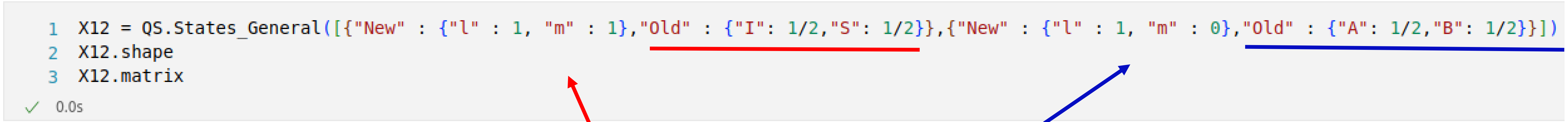
$$|\, 1,\ 0\rangle$$

PyOR Quantum: Quantum System

- Coupled State (multiple)



```
SpinList = {"I": 1/2,"S": 1/2,"A": 1/2,"B": 1/2}

QS = QuantumSystem(SpinList)
QLib = QuantumLibrary()

QS.SpinOperator(PrintDefault=False)
```



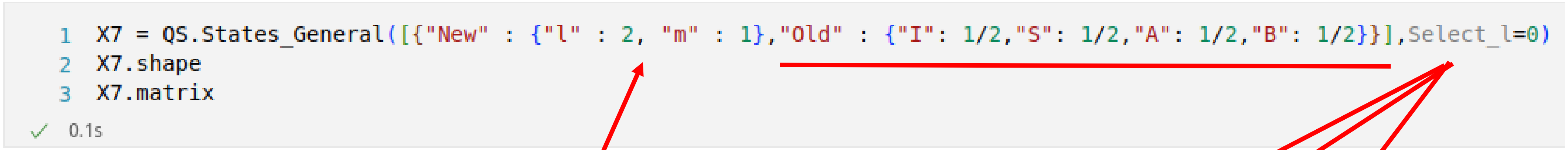
```
X12 = QS.States_General([{"New" : {"l" : 1, "m" : 1},"Old" : {"I": 1/2,"S": 1/2}},{"New" : {"l" : 1, "m" : 0},"Old" : {"A": 1/2,"B": 1/2}}])
X12.shape
X12.matrix
```

✓ 0.0s

$$\begin{bmatrix} 0 \\ 0.707106781186548 \\ 0.707106781186547 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$|\,1,\ 1\rangle \otimes |\,1,\ 0\rangle$$

PyOR Quantum: Quantum System



```
X7 = QS.States_General([{"New" : {"l" : 2, "m" : 1},"Old" : {"I": 1/2,"S": 1/2,"A": 1/2,"B": 1/2}}],Select_l=0)
X7.shape
X7.matrix
```

✓ 0.1s

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

$$|2,1\rangle$$

$$\frac{1}{2} \otimes \frac{1}{2} \otimes \frac{1}{2} \otimes \frac{1}{2} = \overset{0}{2} \oplus \overbrace{\overset{0}{1} \oplus \overset{1}{1} \oplus \overset{2}{1}} \oplus \overbrace{\overset{0}{0} \oplus \overset{1}{0}}$$

For detailed examples, visit my Github

- Main https://github.com/VThalakottoor/PyOR
  - Source
    - PythonOnResonance_jeener_0_9_0.py
    - PythonOnResonance_MagneticResonance.py (soon)
    - PythonOnResonance_Quantum.py (soon)
  - Examples
    - PyOR Magnetic Resonance
      - 1D_EXP, Relaxation, Shape_Pulse, Spin Operators
    - PyOR Quantum
      - Evolution, Quantum Library, Quantum Objects, Quantum system

Future

- Documentation
- PyOR Magnetic Resonance
    - ssNMR, EPR and Hyperpolarization technique
- PyOR Quantum
    - Relaxation (Lindblad master equation: Hilbert and Liouville)
    - NV center, Polarized atoms, atomic magnetometry, Quantum Computing, ...

Acknowledgement

- Daniel Abergel
- ANR