example-spin-chain-model

June 25, 2014

1 QuTiP example: Physical implementation of Spin Chain Qubit model

Author: Anubhav Vardhan (anubhavvardhan@gmail.com)

For more information about QuTiP see http://qutip.org

```
In [1]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
In [2]: from qutip import *
```

```
In [3]: from qutip.qip.models.circuitprocessor import *
```

```
In [4]: from qutip.qip.models.spinchain import *
```

1.1 Hamiltonian:

$$H = -\frac{1}{2}\sum_{n}^{N} h_n \sigma_z(n) - \frac{1}{2}\sum_{n}^{N-1}[J_x^{(n)}\sigma_x(n)\sigma_x(n+1) + J_y^{(n)}\sigma_y(n)\sigma_y(n+1) + J_z^{(n)}\sigma_z(n)\sigma_z(n+1)]$$

The linear and circular spin chain models employing the nearest neighbor interaction can be implemented using the SpinChain class.

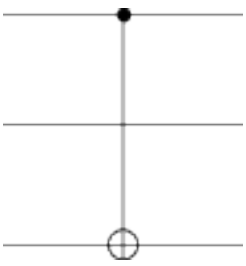
1.2 Circuit Setup

```
In [5]: N = 3
        qc = QubitCircuit(N)

        qc.add_gate("CNOT", targets=[0], controls=[2])

        qc.png
```

Out[5]:



The non-adjacent interactions are broken into a series of adjacent ones by the program automatically.

In [6]:
```
U_ideal = gate_sequence_product(qc.unitary_matrix())

U_ideal
```

Out[6]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

1.3 Circular Spin Chain Model Implementation

In [7]:
```
p1 = CircularSpinChain(N, correct_global_phase=True)

U_list = p1.run(qc)

U_physical = gate_sequence_product(U_list)

U_physical.tidyup(atol=1e-5)
```

Out[7]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

In [8]:
```
(U_ideal - U_physical).norm()
```

Out[8]: 0.0

The results obtained from the physical implementation agree with the ideal result.

In [9]:
```
p1.qc0.gates
```

Out[9]: [Gate(CNOT, targets=[0], controls=[2])]

The gates are first convert to gates with adjacent interactions moving in the direction with the least number of qubits in between.

In [10]:
```
p1.qc1.gates
```

```
Out[10]: [Gate(CNOT, targets=[0], controls=[2])]
```

They are then converted into the basis [ISWAP, RX, RZ]

```
In [11]: p1.qc2.gates
```

```
Out[11]: [Gate(GLOBALPHASE, targets=None, controls=None),
          Gate(ISWAP, targets=[2, 0], controls=None),
          Gate(RZ, targets=[0], controls=None),
          Gate(RZ, targets=[2], controls=None),
          Gate(RX, targets=[2], controls=None),
          Gate(RZ, targets=[2], controls=None),
          Gate(RZ, targets=[2], controls=None),
          Gate(ISWAP, targets=[2, 0], controls=None),
          Gate(RZ, targets=[0], controls=None),
          Gate(RX, targets=[0], controls=None),
          Gate(RZ, targets=[0], controls=None),
          Gate(RZ, targets=[0], controls=None)]
```

The time for each applied gate:

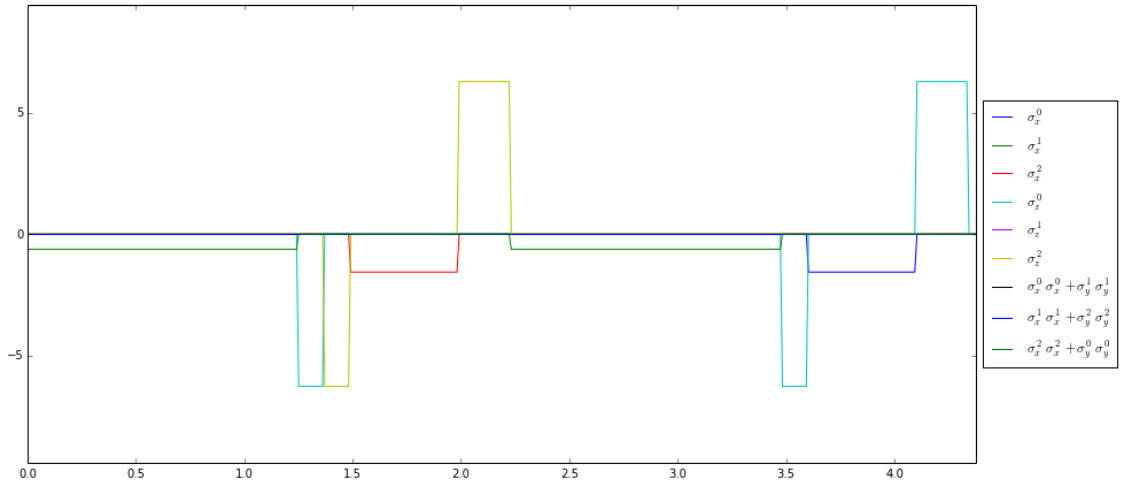
```
In [12]: p1.T_list
```

```
Out[12]: [1.25, 0.125, 0.125, 0.5, 0.125, 0.125, 1.25, 0.125, 0.5, 0.125, 0.125]
```

The pulse can be plotted as:

```
In [13]: p1.plot_pulses()
```

```
Out[13]: (<matplotlib.figure.Figure at 0x7fb48ca65898>,
          <matplotlib.axes.AxesSubplot at 0x7fb48ca6c860>)
```



1.4 Linear Spin Chain Model Implementation

```
In [14]: p2 = LinearSpinChain(N, correct_global_phase=True)

         U_list = p2.run(qc)
```

```
U_physical = gate_sequence_product(U_list)

U_physical.tidyup(atol=1e-5)
```

Out[14]:

Quantum object: dims = [[2, 2, 2], [2, 2, 2]], shape = [8, 8], type = oper, isherm = True

$$\begin{pmatrix} 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 \\ 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.000 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.000 & 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

```
In [15]: (U_ideal - U_physical).norm()
```

```
Out[15]: 0.0
```

The results obtained from the physical implementation agree with the ideal result.

```
In [16]: p2.qc0.gates
```

```
Out[16]: [Gate(CNOT, targets=[0], controls=[2])]
```

The gates are first convert to gates with adjacent interactions moving in the direction with the least number of qubits in between.

```
In [17]: p2.qc1.gates
```

```
Out[17]: [Gate(SWAP, targets=[0, 1], controls=None),
          Gate(CNOT, targets=[1], controls=[2]),
          Gate(SWAP, targets=[0, 1], controls=None)]
```

They are then converted into the basis [ISWAP, RX, RZ]

```
In [18]: p2.qc2.gates
```

```
Out[18]: [Gate(GLOBALPHASE, targets=None, controls=None),
          Gate(ISWAP, targets=[0, 1], controls=None),
          Gate(RX, targets=[0], controls=None),
          Gate(ISWAP, targets=[0, 1], controls=None),
          Gate(RX, targets=[1], controls=None),
          Gate(ISWAP, targets=[1, 0], controls=None),
          Gate(RX, targets=[0], controls=None),
          Gate(GLOBALPHASE, targets=None, controls=None),
          Gate(ISWAP, targets=[2, 1], controls=None),
          Gate(RZ, targets=[1], controls=None),
          Gate(RZ, targets=[2], controls=None),
          Gate(RX, targets=[2], controls=None),
          Gate(RZ, targets=[2], controls=None),
          Gate(RZ, targets=[2], controls=None),
          Gate(ISWAP, targets=[2, 1], controls=None),
          Gate(RZ, targets=[1], controls=None),
          Gate(RX, targets=[1], controls=None),
```

```
          Gate(RZ, targets=[1], controls=None),
          Gate(RZ, targets=[1], controls=None),
          Gate(GLOBALPHASE, targets=None, controls=None),
          Gate(ISWAP, targets=[0, 1], controls=None),
          Gate(RX, targets=[0], controls=None),
          Gate(ISWAP, targets=[0, 1], controls=None),
          Gate(RX, targets=[1], controls=None),
          Gate(ISWAP, targets=[1, 0], controls=None),
          Gate(RX, targets=[0], controls=None)]
```

The time for each applied gate:

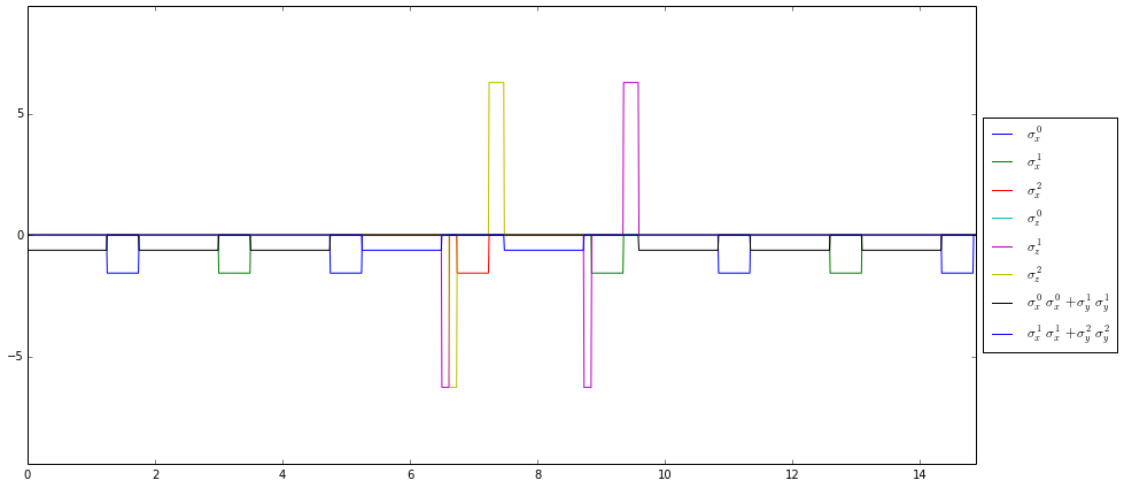
```
In [19]: p2.T_list
```

```
Out[19]: [1.25,
          0.5,
          1.25,
          0.5,
          1.25,
          0.5,
          1.25,
          0.125,
          0.125,
          0.5,
          0.125,
          0.125,
          1.25,
          0.125,
          0.5,
          0.125,
          0.125,
          1.25,
          0.5,
          1.25,
          0.5,
          1.25,
          0.5]
```

The pulse can be plotted as:

```
In [20]: p2.plot_pulses()
```

```
Out[20]: (<matplotlib.figure.Figure at 0x7fb48ca37940>,
          <matplotlib.axes.AxesSubplot at 0x7fb48c8719b0>)
```



1.4.1 Software versions:

```
In [21]: from qutip.ipynbtools import version_table
         version_table()
```

```
Out[21]: <IPython.core.display.HTML at 0x7fb48c3b33c8>
```