a

February 8, 2025

[ ]:
```
pip install git+https://github.com/MIROptics/ECC2025.git
```

[2]:
```
import numpy as np
from qiskit import QuantumCircuit, QuantumRegister
from scipy.linalg import expm
from qiskit.circuit import Parameter
from qiskit.quantum_info import SparsePauliOp, Operator, process_fidelity
from ECC2025.testing import test_5
```

La formula de Trotter permite aproximar la evolución de un Hamiltoniano,

$$H = \sum_j H_j$$

mediante un producto de evoluciones rápidas,

$$U(t) \approx \left(\prod_j e^{-itH_j/m} \right)^m .$$

La calidad de esta aproximación depende del número de pasos m. La formula de Trotter tiene aplicaciones en simulación de materiales, hasta resolución de ecuaciones diferenciales. Para estas aplicaciones es fundamental tener una solución de buena calidad, lo cual puede lograrse aumentando el valor de m. Sin embargo, existen otras estrategías que pueden mejorar la calidad sin aumentar los pasos de Trotter. La clave es darse cuenta distintas permutaciones de los operadores H_j dan distintas aproximaciones. Consideremos el siguiente ejemplo

$$H = X + Y + Z,$$

donde $\{X, Y, Z\}$son matrices de Pauli. Las dos siguientes transformaciones son approximaciones válidad de$U = e^{-itH}$(con$m = 1$),

$$U_1 = e^{-itX} e^{-itY} e^{-itZ}, \qquad U_2 = e^{-itZ} e^{-itY} e^{-itX}.$$

Sin embargo, estas aproximaciones no tienen la misma calidad.

[3]:
```
H = SparsePauliOp.from_list( [('X',1),('Y',1),('Z',1)] )
def U(t):
    return expm( -1j*H.to_matrix()*t )

t = Parameter('t')
```

```
m = 1
```

[4]:
```
qc = QuantumCircuit(1)
for _ in range(m):
    qc.rx( 2*t/m, 0 )
    qc.ry( 2*t/m, 0 )
    qc.rz( 2*t/m, 0 )

U_1 = Operator( qc.assign_parameters([0.2]) )
process_fidelity( U_1, target=Operator(U(0.2)) )
```

[4]:
```
0.9949598835996103
```

[5]:
```
qc = QuantumCircuit(1)
for _ in range(m):
    qc.rz( 2*t/m, 0 )
    qc.ry( 2*t/m, 0 )
    qc.rx( 2*t/m, 0 )

U_2 = Operator( qc.assign_parameters([0.2]) )
process_fidelity( U_2, target=Operator(U(0.2)) )
```

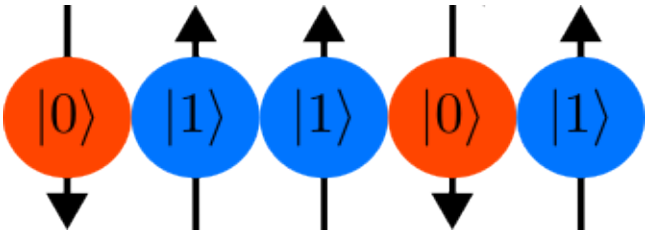
[5]:
```
0.9957866040417919
```

Podemos ver que U_1tiene una fidelidad del99.49%yU_2del99.57%, es decir U_2 es mejor! Hacer una buena elección en el orden del producto es crucial para alcanzar una buena aproximación en un número alto de qubits.

Consideremos ahora el modelo de Heisenberg, descrito por el siguiente Hamiltoniano

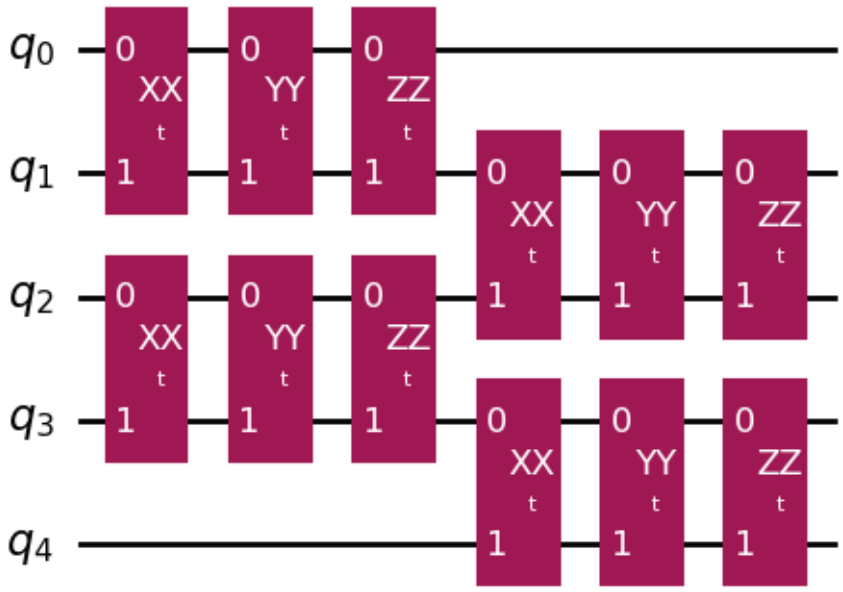
$$H = \sum_{j=0}^{N-2} \left(X_j X_{j+1} + Y_j Y_{j+1} + Z_j Z_{j+1} \right) + h \sum_{j=0}^{N-1} Z_j.$$

Este describe la interacción magnética de una cadena de Nspíns en presencia de un campo magnético externo$h$. Cada spín es representado por un qubit, siendo $|0\rangle$el spín arriba y$|1\rangle$ el spín abajo. Cuando todos los spín están alienados decimos que el sistema exhibe magnetización.



Desafío: Aproxime mediante la formula de Trotter la evolución del modelo de Heisenberg para $N =$ 5qubits. Para esto complete el circuito `Trot_tb_qc` para que implemente **un paso** de approximación de Trotter. Su solución debe tener una fidelidad superior al90%con$m =$ 5pasos de Trotter. Por simplicidad considere$h = 0$.

[6]:
```
num_qubits = 5

# Definimos el parámetro t
t = Parameter('t')
Trot_tb_qr = QuantumRegister(num_qubits, 'q')
Trot_tb_qc = QuantumCircuit(Trot_tb_qr, name='Trot')

### Escriba su solución acá ###

ZZ_qr = QuantumRegister(2)
ZZ_qc = QuantumCircuit(ZZ_qr, name='ZZ')
ZZ_qc.cx(0,1)
ZZ_qc.rz(2 * t, 1)
ZZ_qc.cx(0,1)
ZZ = ZZ_qc.to_instruction()

XX_qr = QuantumRegister(2)
XX_qc = QuantumCircuit(XX_qr, name='XX')
XX_qc.h([0,1])
XX_qc.append(ZZ, [0,1])
XX_qc.h([0,1])
XX = XX_qc.to_instruction()

YY_qr = QuantumRegister(2)
YY_qc = QuantumCircuit(YY_qr, name='YY')
YY_qc.sdg([0,1])
YY_qc.h([0,1])
YY_qc.append(ZZ, [0,1])
YY_qc.h([0,1])
YY_qc.s([0,1])
YY = YY_qc.to_instruction()

for j in [0,2,1,3]:
    Trot_tb_qc.append(XX,[j,j+1])
    Trot_tb_qc.append(YY,[j,j+1])
    Trot_tb_qc.append(ZZ,[j,j+1])

################################

Trot_tb_gate = Trot_tb_qc.to_instruction()

Trot_tb_qc.draw('mpl')
```
[6]:



[7]:
```
def U_trotterize(delta_t, trotter_steps):
    qr = QuantumRegister(num_qubits)
    qc = QuantumCircuit(qr)

    for step in range(trotter_steps):
        qc.append( Trot_tb_gate, list(range(num_qubits)) )

    if qc.num_parameters > 0 :
        qc = qc.assign_parameters({t: delta_t })

    return qc
```

[8]:
```
test_5( U_trotterize )
```

```
Fidelidad= 0.9139858131635302
Felicidades, su solución tiene una fidelidad superior al 90%
```

[ ]:
```

```