ex13_solution

May 20, 2025

Computational Quantum Physics - PHYS 463

Lecturer: Prof. G. Carleo

Assistants: alessandro.sinibaldi@epfl.ch, linda.mauron@epfl.ch, lorenzo.fioroni@epfl.ch

0.1 Exercise 13.0 A brief introduction to Qiskit

In order to simulate quantum computers on classical hardware (and also to access real hardware online) we will resort to the IBM Quantum Computing framework Qiskit (currently migrating to Qiskit 2).

Give a comprehensive view of what Qiskit can do is a task that goes beyond the scope of this course, for this reason we will restrict to prepare quantum circuits and measure observables on it.

[1]:
```
# First install Qiskit
#!pip install qiskit matplotlib pylatexenc qiskit_ibm_provider qiskit_aer␣
↪qiskit_ibm_runtime
```

[2]:
```
import numpy as np
import scipy.linalg as la
import matplotlib.pyplot as plt

import qiskit
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit_aer import QasmSimulator, StatevectorSimulator
from qiskit.visualization import plot_histogram
```

Create you first circuit Circuits are defined using a `QuantumRegister` and a `ClassicalRegister` that keep track of the operations made on qubits and classical bits respectively.

[3]:
```
n_qubits = 1

qc = QuantumCircuit(QuantumRegister(n_qubits), ClassicalRegister(n_qubits))
```

Every circuit has a set of gates already defined

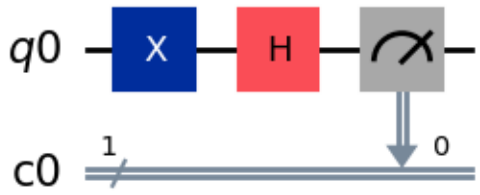
[4]:
```
# We call the gate and pass as an argument the qubit on which it will act
# See https://qiskit.org/documentation/stubs/qiskit.circuit.QuantumCircuit.html␣
↪for all the available gates
```

```
qc.x(0)
qc.h(0)

# We want to measure the qubit we prepared and save the result in the classical␣
  ↪bit
qc.measure(qubit=0, cbit=0)
```

[4]:
```
<qiskit.circuit.instructionset.InstructionSet at 0x1225f0910>
```

[5]:
```
# We can easily draw the circuit
qc.draw('mpl')
```

[5]:

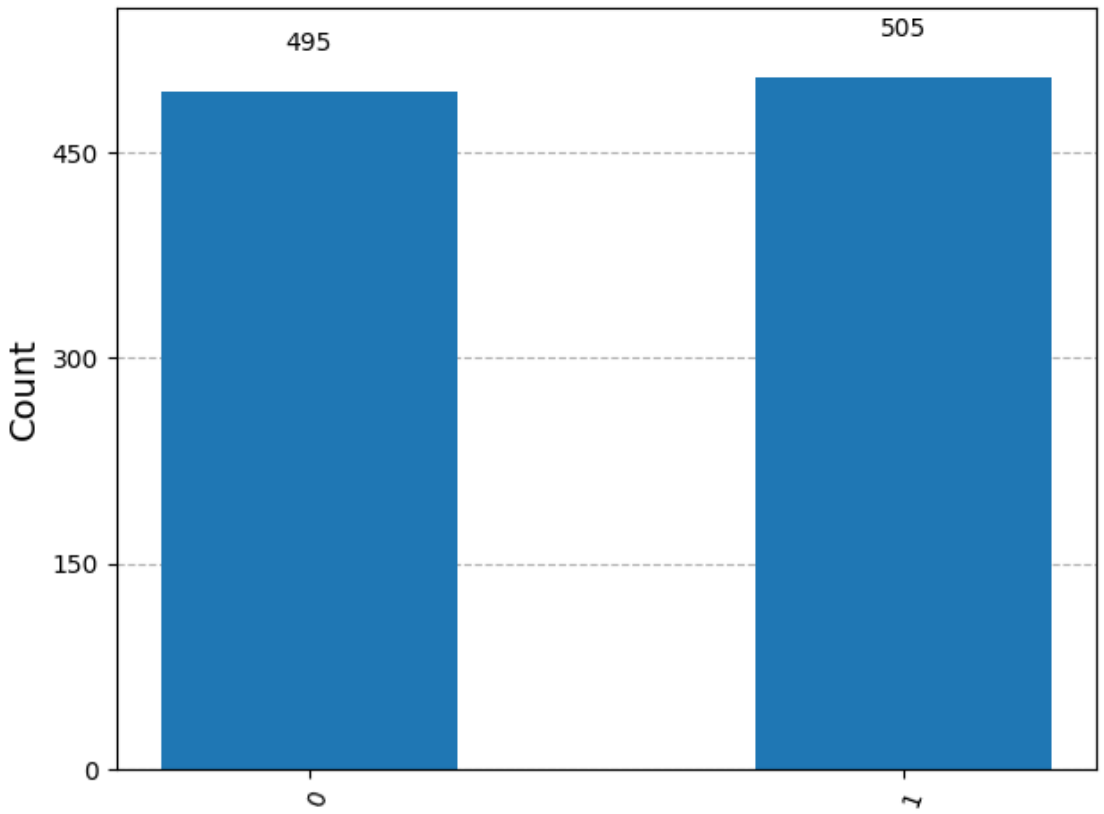


Now we want to perform the simulation and plot the results

[6]:
```
simulator = QasmSimulator()
results   = simulator.run(qc, shots=1000).result()
```

[7]:
```
plot_histogram(results.get_counts())
```

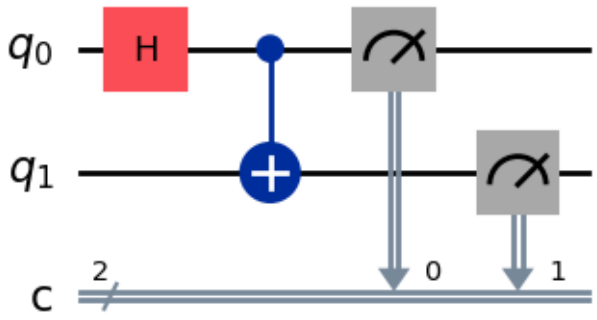
[7]:



Preparing Bell states Now we want to prepare a two qubit entangled state (a Bell state) on the quantum computer

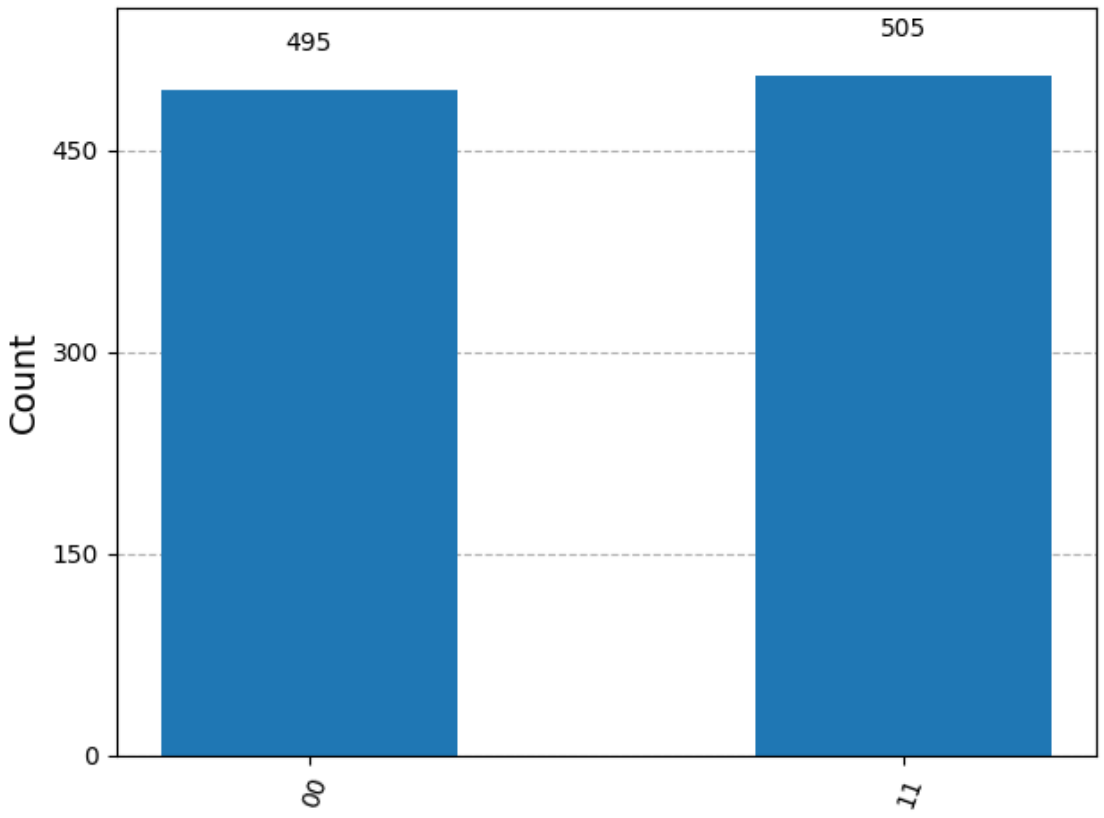
[8]:
```
# shortcut for QuantumCircuit(QuantumRegister(2), ClassicalRegister(2))
qc2 = QuantumCircuit(2,2)

qc2.h(0)
qc2.cx(0,1)

# Which qubits to measure and where to save the results
qc2.measure(qubit=[0,1], cbit=[0,1])
qc2.draw('mpl')
```

[8]:



[9]:
```
results2 = simulator.run(qc2, shots=1000).result()
plot_histogram(results2.get_counts())
```

[9]:



Measuring an observable What if we want to measure a specific observable (as an example `X^I`) on the prepared state?

Qiskit gives the possibility to do so. The procedure at the moment is a little bit involved, but future release will simplify this.

[10]:
```
from qiskit.quantum_info import SparsePauliOp                    # To create␣
 ↪operators
from qiskit.primitives import StatevectorEstimator as Estimator     # to␣
 ↪estimate expectation values
```

[11]:
```
n_shots     = 1000
estimator   = Estimator()
```

[12]:
```
ob = SparsePauliOp.from_list([("XI",1.0)])

# Compose the observable with the quantum circuit you want to measure
qc_meas = QuantumCircuit(2)
qc_meas.h(0)
qc_meas.cx(0,1)
qc_meas.cx(1,0)
qc_meas.cx(0,1)

# Measure
pubs = (qc_meas, ob)
qc_measured = estimator.run([pubs]).result()[0]
```

[13]:
```
print("Value:",qc_measured.data.evs)
```

```
Value: 0.9999999999999998
```

Running your circuit on hardware Suppose now we want to run our circuit on real hardware instead of simulators, how do we do that?

We must change the backend of our calculations, and we need an IBM Quantum account.

[14]:
```
from qiskit_ibm_runtime import QiskitRuntimeService
```

[15]:
```
# Find your API token in the homepage of the IBM Quantum website (you can skip␣
 ↪if you already did it)
#QiskitRuntimeService.save_account(channel="ibm_quantum", token="<IQP_TOKEN>",␣
 ↪overwrite=True)
```

[16]:
```
# Then load your credentials
service = QiskitRuntimeService(channel='ibm_quantum', instance='ibm-q/open/
 ↪main')
```

```
/var/folders/3n/2qkt_9ts32v_c5xkvv3d61mr0000gp/T/ipykernel_49184/204554355.py:2:
DeprecationWarning: The "ibm_quantum" channel option is deprecated and will be
sunset on 1 July. After this date, "ibm_cloud" and "local" will be the only
valid channels. For information on migrating to the new IBM Quantum Platform on
the "ibm_cloud" channel, review the migration guide
```

```
https://quantum.cloud.ibm.com/docs/migration-guides/classic-iqp-to-cloud-iqp .
  service = QiskitRuntimeService(channel='ibm_quantum',
instance='ibm-q/open/main')
```

[17]:
```
## See the backends you have access to
service.backends(simulator=False)
```

[17]:
```
[<IBMBackend('ibm_brisbane')>, <IBMBackend('ibm_sherbrooke')>]
```

[18]:
```
# Now decide which backend to use
hw_backend = service.backend('ibm_brisbane')
```

[19]:
```
# You can also use the backend to import a noisemodel
from qiskit_aer.noise import NoiseModel
from qiskit_aer.primitives import Estimator as AerEstimator

noise_model = NoiseModel.from_backend(hw_backend)
print(noise_model)
```

```
NoiseModel:
  Basis gates: ['delay', 'ecr', 'for_loop', 'id', 'if_else', 'measure', 'reset',
'rz', 'switch_case', 'sx', 'x']
  Instructions with noise: ['ecr', 'reset', 'id', 'measure', 'x', 'sx']
  Qubits with noise: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56,
57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96,
97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126]
  Specific qubit errors: [('ecr', (1, 0)), ('ecr', (2, 1)), ('ecr', (3, 2)),
('ecr', (4, 3)), ('ecr', (4, 5)), ('ecr', (4, 15)), ('ecr', (6, 5)), ('ecr', (6,
7)), ('ecr', (7, 8)), ('ecr', (8, 9)), ('ecr', (10, 9)), ('ecr', (10, 11)),
('ecr', (11, 12)), ('ecr', (12, 17)), ('ecr', (13, 12)), ('ecr', (14, 0)),
('ecr', (14, 18)), ('ecr', (15, 22)), ('ecr', (16, 8)), ('ecr', (16, 26)),
('ecr', (17, 30)), ('ecr', (18, 19)), ('ecr', (20, 19)), ('ecr', (20, 33)),
('ecr', (21, 20)), ('ecr', (21, 22)), ('ecr', (22, 23)), ('ecr', (24, 23)),
('ecr', (24, 34)), ('ecr', (25, 24)), ('ecr', (26, 25)), ('ecr', (27, 26)),
('ecr', (28, 27)), ('ecr', (28, 29)), ('ecr', (28, 35)), ('ecr', (30, 29)),
('ecr', (30, 31)), ('ecr', (31, 32)), ('ecr', (32, 36)), ('ecr', (33, 39)),
('ecr', (34, 43)), ('ecr', (35, 47)), ('ecr', (36, 51)), ('ecr', (37, 38)),
('ecr', (39, 38)), ('ecr', (40, 39)), ('ecr', (40, 41)), ('ecr', (41, 53)),
('ecr', (42, 41)), ('ecr', (42, 43)), ('ecr', (43, 44)), ('ecr', (44, 45)),
('ecr', (46, 45)), ('ecr', (46, 47)), ('ecr', (48, 47)), ('ecr', (48, 49)),
('ecr', (50, 49)), ('ecr', (50, 51)), ('ecr', (52, 37)), ('ecr', (52, 56)),
('ecr', (53, 60)), ('ecr', (54, 45)), ('ecr', (54, 64)), ('ecr', (55, 49)),
('ecr', (55, 68)), ('ecr', (56, 57)), ('ecr', (57, 58)), ('ecr', (58, 59)),
('ecr', (58, 71)), ('ecr', (59, 60)), ('ecr', (60, 61)), ('ecr', (62, 61)),
```

('ecr', (62, 63)), ('ecr', (62, 72)), ('ecr', (63, 64)), ('ecr', (65, 64)),
('ecr', (65, 66)), ('ecr', (67, 66)), ('ecr', (67, 68)), ('ecr', (69, 68)),
('ecr', (69, 70)), ('ecr', (73, 66)), ('ecr', (74, 70)), ('ecr', (74, 89)),
('ecr', (75, 90)), ('ecr', (76, 75)), ('ecr', (77, 71)), ('ecr', (77, 76)),
('ecr', (77, 78)), ('ecr', (79, 78)), ('ecr', (79, 80)), ('ecr', (80, 81)),
('ecr', (81, 72)), ('ecr', (81, 82)), ('ecr', (82, 83)), ('ecr', (83, 92)),
('ecr', (84, 83)), ('ecr', (85, 73)), ('ecr', (85, 84)), ('ecr', (85, 86)),
('ecr', (86, 87)), ('ecr', (87, 88)), ('ecr', (88, 89)), ('ecr', (91, 79)),
('ecr', (92, 102)), ('ecr', (93, 87)), ('ecr', (93, 106)), ('ecr', (94, 90)),
('ecr', (94, 95)), ('ecr', (95, 96)), ('ecr', (97, 96)), ('ecr', (97, 98)),
('ecr', (98, 91)), ('ecr', (99, 98)), ('ecr', (100, 99)), ('ecr', (100, 110)),
('ecr', (101, 100)), ('ecr', (101, 102)), ('ecr', (102, 103)), ('ecr', (104,
103)), ('ecr', (105, 104)), ('ecr', (105, 106)), ('ecr', (107, 106)), ('ecr',
(108, 107)), ('ecr', (108, 112)), ('ecr', (109, 96)), ('ecr', (110, 118)),
('ecr', (111, 104)), ('ecr', (112, 126)), ('ecr', (113, 114)), ('ecr', (114,
109)), ('ecr', (114, 115)), ('ecr', (116, 115)), ('ecr', (116, 117)), ('ecr',
(117, 118)), ('ecr', (118, 119)), ('ecr', (120, 119)), ('ecr', (121, 120)),
('ecr', (122, 111)), ('ecr', (122, 121)), ('ecr', (122, 123)), ('ecr', (124,
123)), ('ecr', (125, 124)), ('ecr', (125, 126)), ('reset', (0,)), ('reset',
(1,)), ('reset', (2,)), ('reset', (3,)), ('reset', (4,)), ('reset', (5,)),
('reset', (6,)), ('reset', (7,)), ('reset', (8,)), ('reset', (9,)), ('reset',
(10,)), ('reset', (11,)), ('reset', (12,)), ('reset', (13,)), ('reset', (14,)),
('reset', (15,)), ('reset', (16,)), ('reset', (17,)), ('reset', (18,)),
('reset', (19,)), ('reset', (20,)), ('reset', (21,)), ('reset', (22,)),
('reset', (23,)), ('reset', (24,)), ('reset', (25,)), ('reset', (26,)),
('reset', (27,)), ('reset', (28,)), ('reset', (29,)), ('reset', (30,)),
('reset', (31,)), ('reset', (32,)), ('reset', (33,)), ('reset', (34,)),
('reset', (35,)), ('reset', (36,)), ('reset', (37,)), ('reset', (38,)),
('reset', (39,)), ('reset', (40,)), ('reset', (41,)), ('reset', (42,)),
('reset', (43,)), ('reset', (44,)), ('reset', (45,)), ('reset', (46,)),
('reset', (47,)), ('reset', (48,)), ('reset', (49,)), ('reset', (50,)),
('reset', (51,)), ('reset', (52,)), ('reset', (53,)), ('reset', (54,)),
('reset', (55,)), ('reset', (56,)), ('reset', (57,)), ('reset', (58,)),
('reset', (59,)), ('reset', (60,)), ('reset', (61,)), ('reset', (62,)),
('reset', (63,)), ('reset', (64,)), ('reset', (65,)), ('reset', (66,)),
('reset', (67,)), ('reset', (68,)), ('reset', (69,)), ('reset', (70,)),
('reset', (71,)), ('reset', (72,)), ('reset', (73,)), ('reset', (74,)),
('reset', (75,)), ('reset', (76,)), ('reset', (77,)), ('reset', (78,)),
('reset', (79,)), ('reset', (80,)), ('reset', (81,)), ('reset', (82,)),
('reset', (83,)), ('reset', (84,)), ('reset', (85,)), ('reset', (86,)),
('reset', (87,)), ('reset', (88,)), ('reset', (89,)), ('reset', (90,)),
('reset', (91,)), ('reset', (92,)), ('reset', (93,)), ('reset', (94,)),
('reset', (95,)), ('reset', (96,)), ('reset', (97,)), ('reset', (98,)),
('reset', (99,)), ('reset', (100,)), ('reset', (101,)), ('reset', (102,)),
('reset', (103,)), ('reset', (104,)), ('reset', (105,)), ('reset', (106,)),
('reset', (107,)), ('reset', (108,)), ('reset', (109,)), ('reset', (110,)),
('reset', (111,)), ('reset', (112,)), ('reset', (113,)), ('reset', (114,)),
('reset', (115,)), ('reset', (116,)), ('reset', (117,)), ('reset', (118,)),

```
('reset', (119,)), ('reset', (120,)), ('reset', (121,)), ('reset', (122,)),
('reset', (123,)), ('reset', (124,)), ('reset', (125,)), ('reset', (126,)),
('id', (0,)), ('id', (1,)), ('id', (2,)), ('id', (3,)), ('id', (4,)), ('id',
(5,)), ('id', (6,)), ('id', (7,)), ('id', (8,)), ('id', (9,)), ('id', (10,)),
('id', (11,)), ('id', (12,)), ('id', (13,)), ('id', (14,)), ('id', (15,)),
('id', (16,)), ('id', (17,)), ('id', (18,)), ('id', (19,)), ('id', (20,)),
('id', (21,)), ('id', (22,)), ('id', (23,)), ('id', (24,)), ('id', (25,)),
('id', (26,)), ('id', (27,)), ('id', (28,)), ('id', (29,)), ('id', (30,)),
('id', (31,)), ('id', (32,)), ('id', (33,)), ('id', (34,)), ('id', (35,)),
('id', (36,)), ('id', (37,)), ('id', (38,)), ('id', (39,)), ('id', (40,)),
('id', (41,)), ('id', (42,)), ('id', (43,)), ('id', (44,)), ('id', (45,)),
('id', (46,)), ('id', (47,)), ('id', (48,)), ('id', (49,)), ('id', (50,)),
('id', (51,)), ('id', (52,)), ('id', (53,)), ('id', (54,)), ('id', (55,)),
('id', (56,)), ('id', (57,)), ('id', (58,)), ('id', (59,)), ('id', (60,)),
('id', (61,)), ('id', (62,)), ('id', (63,)), ('id', (64,)), ('id', (65,)),
('id', (66,)), ('id', (67,)), ('id', (68,)), ('id', (69,)), ('id', (70,)),
('id', (71,)), ('id', (72,)), ('id', (73,)), ('id', (74,)), ('id', (75,)),
('id', (76,)), ('id', (77,)), ('id', (78,)), ('id', (79,)), ('id', (80,)),
('id', (81,)), ('id', (82,)), ('id', (83,)), ('id', (84,)), ('id', (85,)),
('id', (86,)), ('id', (87,)), ('id', (88,)), ('id', (89,)), ('id', (90,)),
('id', (91,)), ('id', (92,)), ('id', (93,)), ('id', (94,)), ('id', (95,)),
('id', (96,)), ('id', (97,)), ('id', (98,)), ('id', (99,)), ('id', (100,)),
('id', (101,)), ('id', (102,)), ('id', (103,)), ('id', (104,)), ('id', (105,)),
('id', (106,)), ('id', (107,)), ('id', (108,)), ('id', (109,)), ('id', (110,)),
('id', (111,)), ('id', (112,)), ('id', (113,)), ('id', (114,)), ('id', (115,)),
('id', (116,)), ('id', (117,)), ('id', (118,)), ('id', (119,)), ('id', (120,)),
('id', (121,)), ('id', (122,)), ('id', (123,)), ('id', (124,)), ('id', (125,)),
('id', (126,)), ('x', (0,)), ('x', (1,)), ('x', (2,)), ('x', (3,)), ('x', (4,)),
('x', (5,)), ('x', (6,)), ('x', (7,)), ('x', (8,)), ('x', (9,)), ('x', (10,)),
('x', (11,)), ('x', (12,)), ('x', (13,)), ('x', (14,)), ('x', (15,)), ('x',
(16,)), ('x', (17,)), ('x', (18,)), ('x', (19,)), ('x', (20,)), ('x', (21,)),
('x', (22,)), ('x', (23,)), ('x', (24,)), ('x', (25,)), ('x', (26,)), ('x',
(27,)), ('x', (28,)), ('x', (29,)), ('x', (30,)), ('x', (31,)), ('x', (32,)),
('x', (33,)), ('x', (34,)), ('x', (35,)), ('x', (36,)), ('x', (37,)), ('x',
(38,)), ('x', (39,)), ('x', (40,)), ('x', (41,)), ('x', (42,)), ('x', (43,)),
('x', (44,)), ('x', (45,)), ('x', (46,)), ('x', (47,)), ('x', (48,)), ('x',
(49,)), ('x', (50,)), ('x', (51,)), ('x', (52,)), ('x', (53,)), ('x', (54,)),
('x', (55,)), ('x', (56,)), ('x', (57,)), ('x', (58,)), ('x', (59,)), ('x',
(60,)), ('x', (61,)), ('x', (62,)), ('x', (63,)), ('x', (64,)), ('x', (65,)),
('x', (66,)), ('x', (67,)), ('x', (68,)), ('x', (69,)), ('x', (70,)), ('x',
(71,)), ('x', (72,)), ('x', (73,)), ('x', (74,)), ('x', (75,)), ('x', (76,)),
('x', (77,)), ('x', (78,)), ('x', (79,)), ('x', (80,)), ('x', (81,)), ('x',
(82,)), ('x', (83,)), ('x', (84,)), ('x', (85,)), ('x', (86,)), ('x', (87,)),
('x', (88,)), ('x', (89,)), ('x', (90,)), ('x', (91,)), ('x', (92,)), ('x',
(93,)), ('x', (94,)), ('x', (95,)), ('x', (96,)), ('x', (97,)), ('x', (98,)),
('x', (99,)), ('x', (100,)), ('x', (101,)), ('x', (102,)), ('x', (103,)), ('x',
(104,)), ('x', (105,)), ('x', (106,)), ('x', (107,)), ('x', (108,)), ('x',
(109,)), ('x', (110,)), ('x', (111,)), ('x', (112,)), ('x', (113,)), ('x',
```

(114,)), ('x', (115,)), ('x', (116,)), ('x', (117,)), ('x', (118,)), ('x',
(119,)), ('x', (120,)), ('x', (121,)), ('x', (122,)), ('x', (123,)), ('x',
(124,)), ('x', (125,)), ('x', (126,)), ('sx', (0,)), ('sx', (1,)), ('sx', (2,)),
('sx', (3,)), ('sx', (4,)), ('sx', (5,)), ('sx', (6,)), ('sx', (7,)), ('sx',
(8,)), ('sx', (9,)), ('sx', (10,)), ('sx', (11,)), ('sx', (12,)), ('sx', (13,)),
('sx', (14,)), ('sx', (15,)), ('sx', (16,)), ('sx', (17,)), ('sx', (18,)),
('sx', (19,)), ('sx', (20,)), ('sx', (21,)), ('sx', (22,)), ('sx', (23,)),
('sx', (24,)), ('sx', (25,)), ('sx', (26,)), ('sx', (27,)), ('sx', (28,)),
('sx', (29,)), ('sx', (30,)), ('sx', (31,)), ('sx', (32,)), ('sx', (33,)),
('sx', (34,)), ('sx', (35,)), ('sx', (36,)), ('sx', (37,)), ('sx', (38,)),
('sx', (39,)), ('sx', (40,)), ('sx', (41,)), ('sx', (42,)), ('sx', (43,)),
('sx', (44,)), ('sx', (45,)), ('sx', (46,)), ('sx', (47,)), ('sx', (48,)),
('sx', (49,)), ('sx', (50,)), ('sx', (51,)), ('sx', (52,)), ('sx', (53,)),
('sx', (54,)), ('sx', (55,)), ('sx', (56,)), ('sx', (57,)), ('sx', (58,)),
('sx', (59,)), ('sx', (60,)), ('sx', (61,)), ('sx', (62,)), ('sx', (63,)),
('sx', (64,)), ('sx', (65,)), ('sx', (66,)), ('sx', (67,)), ('sx', (68,)),
('sx', (69,)), ('sx', (70,)), ('sx', (71,)), ('sx', (72,)), ('sx', (73,)),
('sx', (74,)), ('sx', (75,)), ('sx', (76,)), ('sx', (77,)), ('sx', (78,)),
('sx', (79,)), ('sx', (80,)), ('sx', (81,)), ('sx', (82,)), ('sx', (83,)),
('sx', (84,)), ('sx', (85,)), ('sx', (86,)), ('sx', (87,)), ('sx', (88,)),
('sx', (89,)), ('sx', (90,)), ('sx', (91,)), ('sx', (92,)), ('sx', (93,)),
('sx', (94,)), ('sx', (95,)), ('sx', (96,)), ('sx', (97,)), ('sx', (98,)),
('sx', (99,)), ('sx', (100,)), ('sx', (101,)), ('sx', (102,)), ('sx', (103,)),
('sx', (104,)), ('sx', (105,)), ('sx', (106,)), ('sx', (107,)), ('sx', (108,)),
('sx', (109,)), ('sx', (110,)), ('sx', (111,)), ('sx', (112,)), ('sx', (113,)),
('sx', (114,)), ('sx', (115,)), ('sx', (116,)), ('sx', (117,)), ('sx', (118,)),
('sx', (119,)), ('sx', (120,)), ('sx', (121,)), ('sx', (122,)), ('sx', (123,)),
('sx', (124,)), ('sx', (125,)), ('sx', (126,)), ('measure', (0,)), ('measure',
(1,)), ('measure', (2,)), ('measure', (3,)), ('measure', (4,)), ('measure',
(5,)), ('measure', (6,)), ('measure', (7,)), ('measure', (8,)), ('measure',
(9,)), ('measure', (10,)), ('measure', (11,)), ('measure', (12,)), ('measure',
(13,)), ('measure', (14,)), ('measure', (15,)), ('measure', (16,)), ('measure',
(17,)), ('measure', (18,)), ('measure', (19,)), ('measure', (20,)), ('measure',
(21,)), ('measure', (22,)), ('measure', (23,)), ('measure', (24,)), ('measure',
(25,)), ('measure', (26,)), ('measure', (27,)), ('measure', (28,)), ('measure',
(29,)), ('measure', (30,)), ('measure', (31,)), ('measure', (32,)), ('measure',
(33,)), ('measure', (34,)), ('measure', (35,)), ('measure', (36,)), ('measure',
(37,)), ('measure', (38,)), ('measure', (39,)), ('measure', (40,)), ('measure',
(41,)), ('measure', (42,)), ('measure', (43,)), ('measure', (44,)), ('measure',
(45,)), ('measure', (46,)), ('measure', (47,)), ('measure', (48,)), ('measure',
(49,)), ('measure', (50,)), ('measure', (51,)), ('measure', (52,)), ('measure',
(53,)), ('measure', (54,)), ('measure', (55,)), ('measure', (56,)), ('measure',
(57,)), ('measure', (58,)), ('measure', (59,)), ('measure', (60,)), ('measure',
(61,)), ('measure', (62,)), ('measure', (63,)), ('measure', (64,)), ('measure',
(65,)), ('measure', (66,)), ('measure', (67,)), ('measure', (68,)), ('measure',
(69,)), ('measure', (70,)), ('measure', (71,)), ('measure', (72,)), ('measure',
(73,)), ('measure', (74,)), ('measure', (75,)), ('measure', (76,)), ('measure',
(77,)), ('measure', (78,)), ('measure', (79,)), ('measure', (80,)), ('measure',

```
(81,)), ('measure', (82,)), ('measure', (83,)), ('measure', (84,)), ('measure',
(85,)), ('measure', (86,)), ('measure', (87,)), ('measure', (88,)), ('measure',
(89,)), ('measure', (90,)), ('measure', (91,)), ('measure', (92,)), ('measure',
(93,)), ('measure', (94,)), ('measure', (95,)), ('measure', (96,)), ('measure',
(97,)), ('measure', (98,)), ('measure', (99,)), ('measure', (100,)), ('measure',
(101,)), ('measure', (102,)), ('measure', (103,)), ('measure', (104,)),
('measure', (105,)), ('measure', (106,)), ('measure', (107,)), ('measure',
(108,)), ('measure', (109,)), ('measure', (110,)), ('measure', (111,)),
('measure', (112,)), ('measure', (113,)), ('measure', (114,)), ('measure',
(115,)), ('measure', (116,)), ('measure', (117,)), ('measure', (118,)),
('measure', (119,)), ('measure', (120,)), ('measure', (121,)), ('measure',
(122,)), ('measure', (123,)), ('measure', (124,)), ('measure', (125,)),
('measure', (126,))]
```

[20]:
```
noisy_estimator = AerEstimator(
    backend_options={
        "method": "density_matrix",
        "noise_model": noise_model,
    },
    run_options={"shots": n_shots},
)
```

[21]:
```
# Measure
qc_measured_noisy = noisy_estimator.run(qc_meas, ob).result()
```

[22]:
```
print(f"Value: {qc_measured_noisy.values[0]} ± {qc_measured_noisy.
  ↪metadata[0]['variance'].real}")
```

```
Value: 0.946 ± 0.10508400000000007
```

0.2 Exercise 13.1 : Simulating the Heisenberg Chain

[23]:
```
from qiskit.quantum_info        import SparsePauliOp
```

[24]:
```
from qiskit.quantum_info    import Statevector
from qiskit.circuit.library import PauliEvolutionGate
```

We will use the XXX Heisenberg Hamiltonian H_{Heis} as defined below

$$H_{\text{Heis}} = \sum_{\langle ij \rangle}^{N} J \left(\sigma_x^{(i)} \sigma_x^{(j)} + \sigma_y^{(i)} \sigma_y^{(j)} + \sigma_z^{(i)} \sigma_z^{(j)} \right).$$

N is the number of spin-1/2 particles in model. The operators σ_x, σ_y, and σ_z are the usual Pauli operators where the i and j superscripts label which qubit they act on. For example, $\sigma_x^{(1)}$ would be the σ_x operator acting on only qubit 1. This version of the general Heisenberg spin model is called XXX because the same J value multiplies each pair of Pauli operators. The sum notation $\langle ij \rangle$ means the sum is over nearest neighbors (only qubits next to each other interact), and J is the interaction strength, which we will set $J = 1$.

0.2.1 Classical simulation of the Heisenberg chain

We can use Qiskit to have a classical simulation of the Heisenberg chain that we will use as a benchmark for the quantum computations.

[25]:
```
# Returns the matrix representation of the XXX Heisenberg model
def H_heis(n_spins):

    # using SparsePauliOp from Terra 0.20 is very compact
    XXs = SparsePauliOp.from_sparse_list([["XX", [i,i+1], 1] for i in␣
 ↪range(n_spins-1)], n_spins)
    YYs = SparsePauliOp.from_sparse_list([["YY", [i,i+1], 1] for i in␣
 ↪range(n_spins-1)], n_spins)
    ZZs = SparsePauliOp.from_sparse_list([["ZZ", [i,i+1], 1] for i in␣
 ↪range(n_spins-1)], n_spins)

    # Sum interactions
    H = XXs + YYs + ZZs

    # Return Hamiltonian
    return H
```

[26]:
```
def U_heis(n_spins,t):
    import scipy.linalg as la
    # Compute XXX Hamiltonian
    # For efficiency, can also be done with csr using H_heis(n_spins).
 ↪to_matrix(sparse=True)
    # and then use csr_array defined in the previous lectures
    H = H_heis(n_spins).to_matrix()

    # Return the exponential of -i multipled by time t multipled by the XXX␣
 ↪Heisenberg Hamiltonian
    return la.expm(-1j*H*t)
```

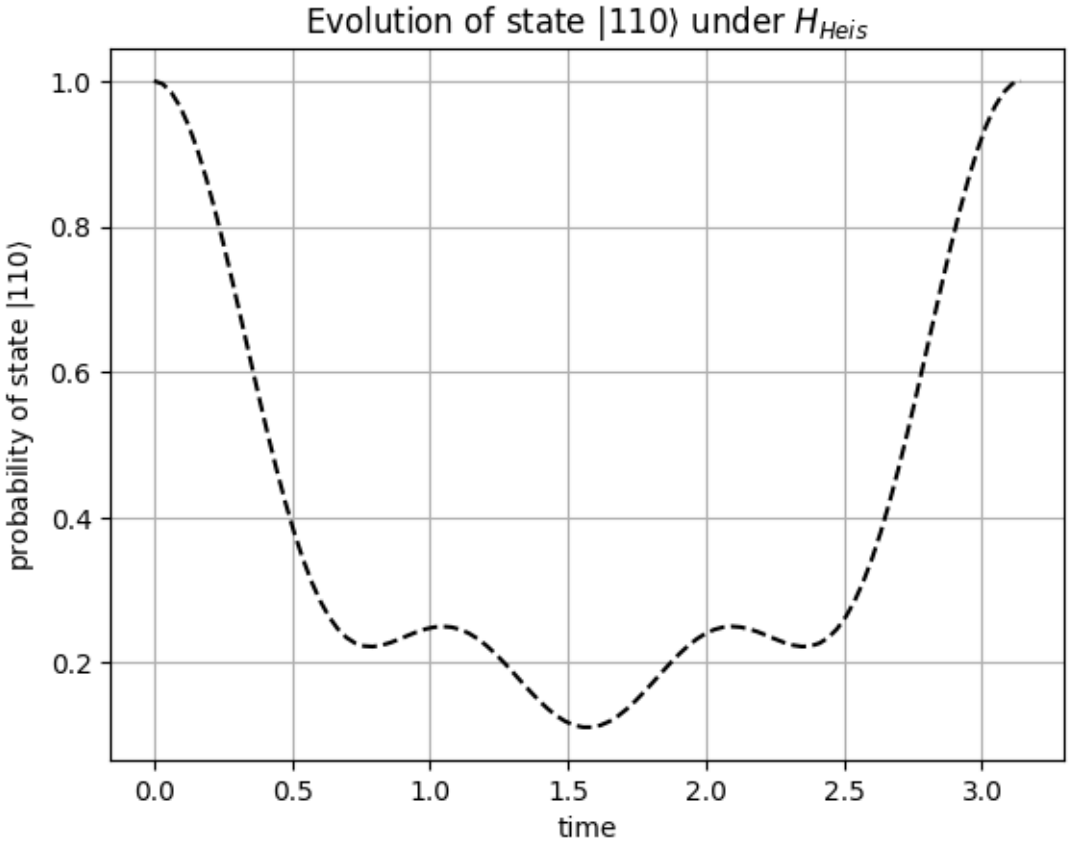
Now that we have the time evolution operator we can apply it to an inital state of choice and study its time dependent properties.

We consider a system of $N = 3$ spins with the initial state $|\psi_0\rangle = |110\rangle$ and evolve it up until total time $T = \pi$.

We will measure the overlap with the initial state at every time t, namely $O(t) = |\langle 110|U(t)|110\rangle|^2$.

[27]:
```
# Define array of time points
ts      = np.linspace(0, np.pi, 100)
n_spins = 3

# Define initial state |110>
init_state = np.array(Statevector.from_label("110"))
```

```
# Compute probability of remaining in |110> state over the array of time points
 # init_state is the array corresponding to |110>
 # @ is short hand for matrix multiplication
 # U_heis(t) is the unitary time evolution at time t
 # init_state@U_heis@init_state returns the inner product <110|U_heis(t)|110>
 #  np.abs(...)**2 is the modulus squared of the innner product which is the␣
 ↪expectation value,
 # or probability, of remaining in |110>


probs_110_exact = [np.abs(init_state@U_heis(n_spins,t)@init_state)**2 for t in␣
 ↪ts]

# Plot evolution of |110>
plt.plot(ts, probs_110_exact,linestyle="dashed",color="black")
plt.xlabel('time')
plt.ylabel(r'probability of state $|110\rangle$')
plt.title(r'Evolution of state $|110\rangle$ under $H_{Heis}$')
plt.grid()
plt.show()
```



0.2.2 Quantum simulation of the Heisenberg chain

0.2.3 Part I: The 2 spin case

For the 2 spin case the Hamiltonian has the following form:

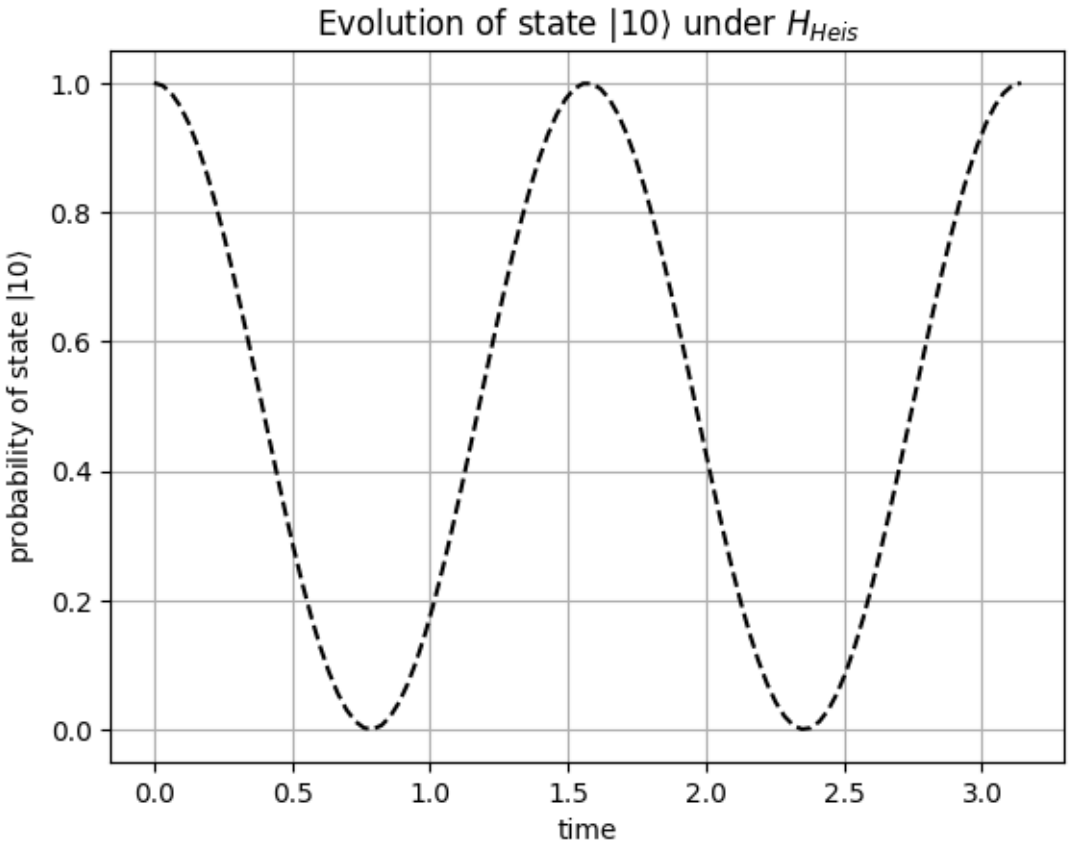
$$H_{\text{Heis2}} = \sigma_x^{(0)}\sigma_x^{(1)} + \sigma_y^{(0)}\sigma_y^{(1)} + \sigma_z^{(0)}\sigma_z^{(1)}$$

We notice that the Pauli operator pairs ($\sigma_x^{(i)}\sigma_x^{(j)}$, $\sigma_y^{(i)}\sigma_y^{(j)}$, and $\sigma_z^{(i)}\sigma_z^{(j)}$) commute. This means that the exponential decomposition using Trotterization of a 2 spins Hamiltonian (H_{Heis2}) is exactand gets us closer to a gate implementation of $U_{\text{Heis2}}(t)$

$$U_{\text{Heis2}}(t) = \exp\left(-itH_{\text{Heis2}}\right) \tag{1}$$
$$U_{\text{Heis2}}(t) = \exp\left(-it\sigma_x^{(0)}\sigma_x^{(1)}\right)\exp\left(-it\sigma_y^{(0)}\sigma_y^{(1)}\right)\exp\left(-it\sigma_z^{(0)}\sigma_z^{(1)}\right) \tag{2}$$

Qiskit already includes these operations, but we will see how we can construct the three operations using only single qubit rotations and CNOTs.

[28]:
```
# First, create a classical simulation to compare

# Define array of time points
ts      = np.linspace(0, np.pi, 100)
n_spins = 2

# Define initial state |10>
init_state = np.array(Statevector.from_label("10"))
probs_10_exact = [np.abs(init_state@U_heis(n_spins,t)@init_state)**2 for t in␣
 ↪ts]

# Plot evolution of |10>
plt.plot(ts, probs_10_exact,linestyle="dashed",color="black")
plt.xlabel('time')
plt.ylabel(r'probability of state $|10\rangle$')
plt.title(r'Evolution of state $|10\rangle$ under $H_{Heis}$')
plt.grid()
plt.show()
```



We will start from the $R_{ZZ}(\theta)$ rotation gate.

Once this gate is decomposed, we can act on the two qubits with R_Y and R_X to construct R_{XX} and R_{YY}, respectively.

The $R_{ZZ}(\theta)$ is a two qubit gate with the following structure:

$$R_{ZZ}(\theta) = \cos(\theta) I \otimes I - i \sin(\theta) Z \otimes Z = \begin{bmatrix} \cos(\theta) - i\sin(\theta) & 0 & 0 & 0 \\ 0 & \cos(\theta) + i\sin(\theta) & 0 & 0 \\ 0 & 0 & \cos(\theta) + i\sin(\theta) & 0 \\ 0 & 0 & 0 & \cos(\theta) - i\sin(\theta) \end{bmatrix}$$

Since there are only diagonal elements, we can start from a single qubit rotation $R_Z = \cos(\theta) I - i \sin(\theta) Z$, which is already diagonal

$$I \otimes R_Z(\theta) = \begin{bmatrix} \cos(\theta) - i\sin(\theta) & 0 & 0 & 0 \\ 0 & \cos(\theta) + i\sin(\theta) & 0 & 0 \\ 0 & 0 & \cos(\theta) - i\sin(\theta) & 0 \\ 0 & 0 & 0 & \cos(\theta) + i\sin(\theta) \end{bmatrix}$$

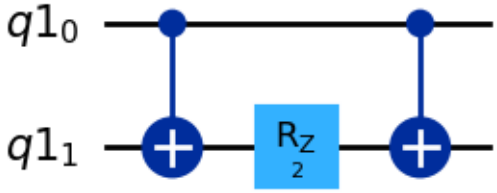
and then invert the third and the fourth element of the diagonal. We can switch 3rd and 4th rows by multiplying this matrix by a CNOT and then switch 3rd and 4th columns by mmultiplying the result by another $CNOT$ (see the $CNOT$ definition in the lecture note).

The final result is therefore $CNOT_{12} * I \otimes R_Z(\theta) * CNOT_{12}$.

[29]:
```
def R_zz(t):

    '''
    Circuit for R_zz(t), as derived

    Args:
        - t: parameter of the rotation

    Returns:
        A QuantumCircuit implementing the R_zz(t) rotation with name ZZ
    '''

    ZZ_qr = QuantumRegister(2)
    ZZ_qc = QuantumCircuit(ZZ_qr, name='ZZ')
    ZZ_qc.cx(0,1)
    ZZ_qc.rz(2 * t, 1) #<- Rotations are defined as R(\theta) = e^{-iP \theta/
↪2} in Qiskit
    ZZ_qc.cx(0,1)
    return ZZ_qc
```

[30]:
```
R_zz(1).draw("mpl")
```

[30]:



Now we can apply a basis change using

$$R_Y(\theta) = \exp\left(-i\frac{\theta}{2}Y\right) = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

apply R_{ZZ} and then change the basis back to the original in order to apply the R_{XX}.

In particular

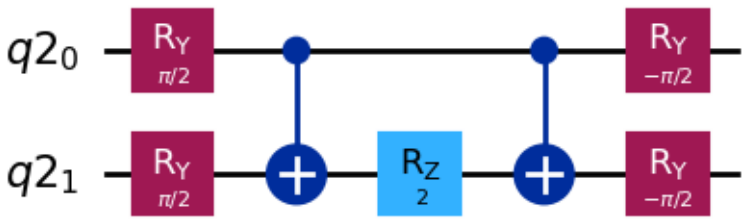
$$R_Y(\pi/2) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}$$

and $R_Y(\pi/2)ZR_Y(-\pi/2) = X$, $R_Y(\pi/2)R_Y(-\pi/2) = I$, therefore

[31]:
```
def R_xx(t):

    '''
    Circuit for R_xx(t)

    Args:
        - t: parameter of the rotation

    Returns:
        A QuantumCircuit implementing the R_xx(t) rotation with name XX
    '''

    XX_qr = QuantumRegister(2)
    XX_qc = QuantumCircuit(XX_qr, name='XX')

    XX_qc.ry(np.pi/2,[0,1])
    XX_qc.cx(0,1)
    XX_qc.rz(2 * t, 1)
    XX_qc.cx(0,1)
    XX_qc.ry(-np.pi/2,[0,1])
    return XX_qc
```

[32]:
```
R_xx(1).draw("mpl")
```

[32]:



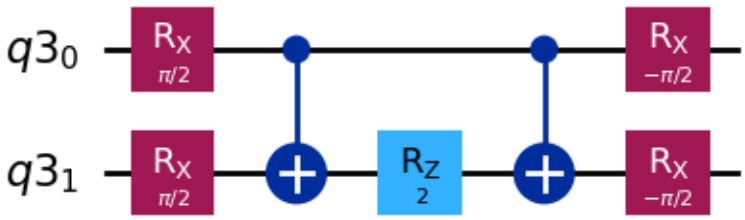
And a similar procedure for R_{YY} with

$$R_X(\theta) = \exp\left(-i\frac{\theta}{2}X\right) = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

[33]:
```
def R_yy(t):

    '''
    Circuit for R_yy(t)

    Args:
        - t: parameter of the rotation

    Returns:
        A QuantumCircuit implementing the R_yy(t) rotation with name YY
    '''

    YY_qr = QuantumRegister(2)
    YY_qc = QuantumCircuit(YY_qr, name='YY')

    YY_qc.rx(np.pi/2,[0,1])
    YY_qc.cx(0,1)
    YY_qc.rz(2 * t, 1)
    YY_qc.cx(0,1)
    YY_qc.rx(-np.pi/2,[0,1])

    return YY_qc
```

[34]:
```
R_yy(1).draw("mpl")
```

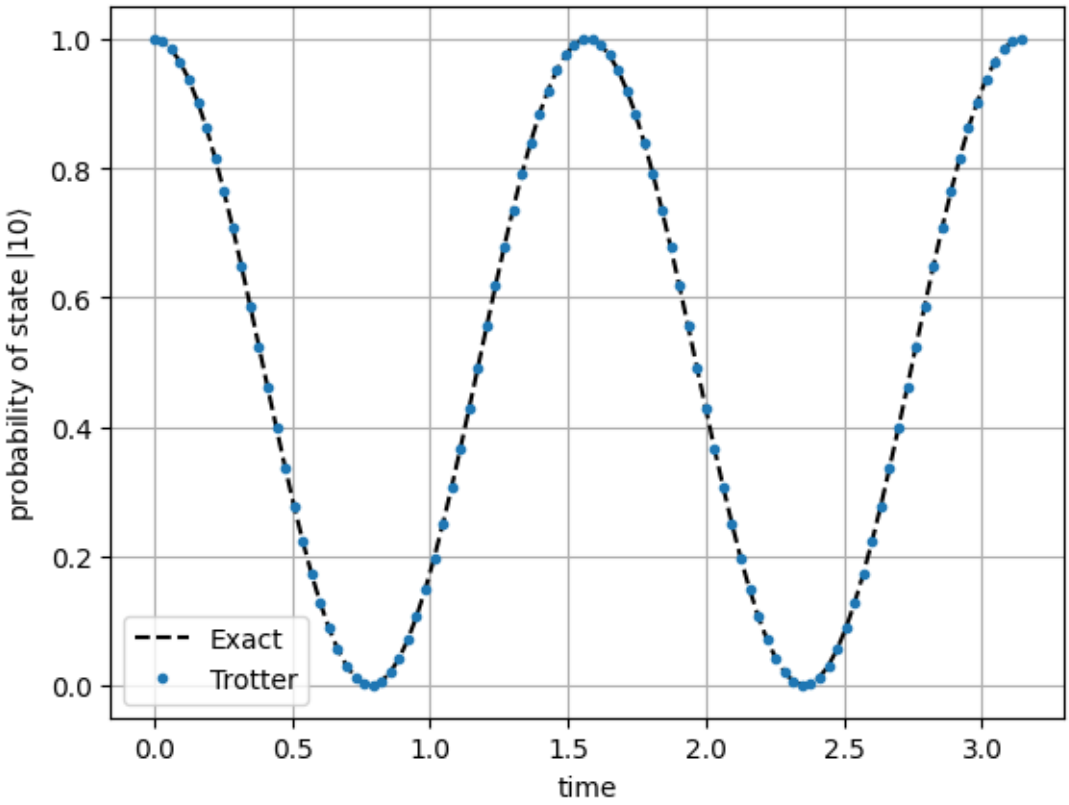
[34]:



[35]:
```
## We want to measure
# |0><0|= (1/2)*(I+Z)
prj_0 = SparsePauliOp(["I","Z"],[0.5,0.5])

# |1><1|= (1/2)*(I-Z)
prj_1 = SparsePauliOp(["I","Z"],[0.5,-0.5])
# and |10><10| =  |1><1|   |0><0|
prj = prj_1 ^ prj_0
```

[36]:
```
# run the simulation
probs_10    = []

for sim_t in ts:

    # Prepare
    qc = QuantumCircuit(2)
    qc.x(1)
    qc = qc.compose(R_xx(sim_t))
    qc = qc.compose(R_yy(sim_t))
    qc = qc.compose(R_zz(sim_t))

    # Measure
    prob = estimator.run([(qc,prj)]).result()[0].data.evs.real
    probs_10.append(prob)
```

[37]:
```
# Now plot the comparison
plt.plot(ts, probs_10_exact,linestyle="dashed",color="black",label="Exact")
plt.plot(ts, probs_10,color="C0",label="Trotter",linestyle="",marker=".")
plt.xlabel('time')
plt.ylabel(r'probability of state $|10\rangle$')

plt.legend()
plt.grid()
plt.show()
```



0.2.4 Part II: The $N > 2$ spin case

Now we will study the more general case.

Actually, we just need to add a spin to see the difference: the exponential $U_{\text{Heis3}}(t)$ cannot be split into a product of simpler exponentials. However, we can approximate $U_{\text{Heis3}}(t)$ as a product of simpler exponentials through Trotterization. Consider again the 2 spin case, within the larger 3 spin system. As we have seen the Hamiltonian on spins i and j $(i, j \in \{0, 1, 2\})$ is $H_{\text{Heis2}}^{(i,j)} = \sigma_x^{(i)}\sigma_x^{(j)} + \sigma_y^{(i)}\sigma_y^{(j)} + \sigma_z^{(i)}\sigma_z^{(j)}$. Rewritting $U_{\text{Heis3}}(t)$ in terms of the two possible subsystems within the total $N = 3$ system you will simulate,

$$U_{\text{Heis3}}(t) = \exp\left[-it\left(H_{\text{Heis2}}^{(0,1)} + H_{\text{Heis2}}^{(1,2)}\right)\right].$$

$H_{\text{Heis2}}^{(0,1)}$ and $H_{\text{Heis2}}^{(1,2)}$ do not commute, so $U_{\text{Heis3}}(t) \neq \exp\left(-itH_{\text{Heis2}}^{(0,1)}\right)\exp\left(-itH_{\text{Heis2}}^{(1,2)}\right)$. But, this product decomposition can be approximated with Trotterization which says $U_{\text{Heis3}}(t)$ is approximately a short evolution of $H_{\text{Heis2}}^{(0,1)}$ (time $= t/n$) and followed by a short evolution of $H_{\text{Heis2}}^{(1,2)}$ (time $= t/n$) repeated n times

$$U_{\text{Heis3}}(t) = \exp\left[-it\left(H^{(0,1)}_{\text{Heis2}} + H^{(1,2)}_{\text{Heis2}}\right)\right] \tag{3}$$

$$U_{\text{Heis3}}(t) \approx \left[\exp\left(\frac{-it}{n}H^{(0,1)}_{\text{Heis2}}\right)\exp\left(\frac{-it}{n}H^{(1,2)}_{\text{Heis2}}\right)\right]^n . \tag{4}$$

n is the number of Trotter steps, and as n increases, the approximation becomes more accurate. (Note that how a untiary is split up into subsystems for Trotterization is not necessarily unique.)

[38]:
```
## Here we introduce a general function for Heisenberg chain Trotter evolution
## The circuit will not contain the initialisation

def Heisenberg_Trotter(num_spins,trotter_steps,t):

    '''
    Circuit implementing Trotterization of the time evolutiom operator for the␣
 ↪XXX Heisenberg
    model on num_spins.

    Naively, every  Trotter step requires 6*(num_spins-1) CNOTs

    Args:
        - num_spins: int, number of qubits of the system
        - trotter_steps: the number of trotter steps n to implement
        - t: the simulation time we are targeting

    Returns:
        A QuantumCircuit implementing the Trotterization of the time evolutiom␣
 ↪operator for the XXX Heisenberg
        model
    '''
    # Given a target time and a number of Trotter steps, every step will evolve␣
 ↪the
    # circuit for a time step dt = target_time/trotter_steps
    dt = t/trotter_steps

    # Initialize quantum circuit for n_spins
    qr = QuantumRegister(num_spins)
    # A QuantumCircuit can also be initialised without a classical register (in␣
 ↪this case classical bits == qubits)
    qc = QuantumCircuit(qr)

    for _ in range(trotter_steps):
        for i in range(0, num_spins - 1):
            qc.append(R_xx(dt), [i, i+1])
            qc.append(R_yy(dt), [i, i+1])
            qc.append(R_zz(dt), [i, i+1])
```

```
        # Barrier to separate the different Trotter steps in the circuit drawing
        qc.barrier()

    return qc
```

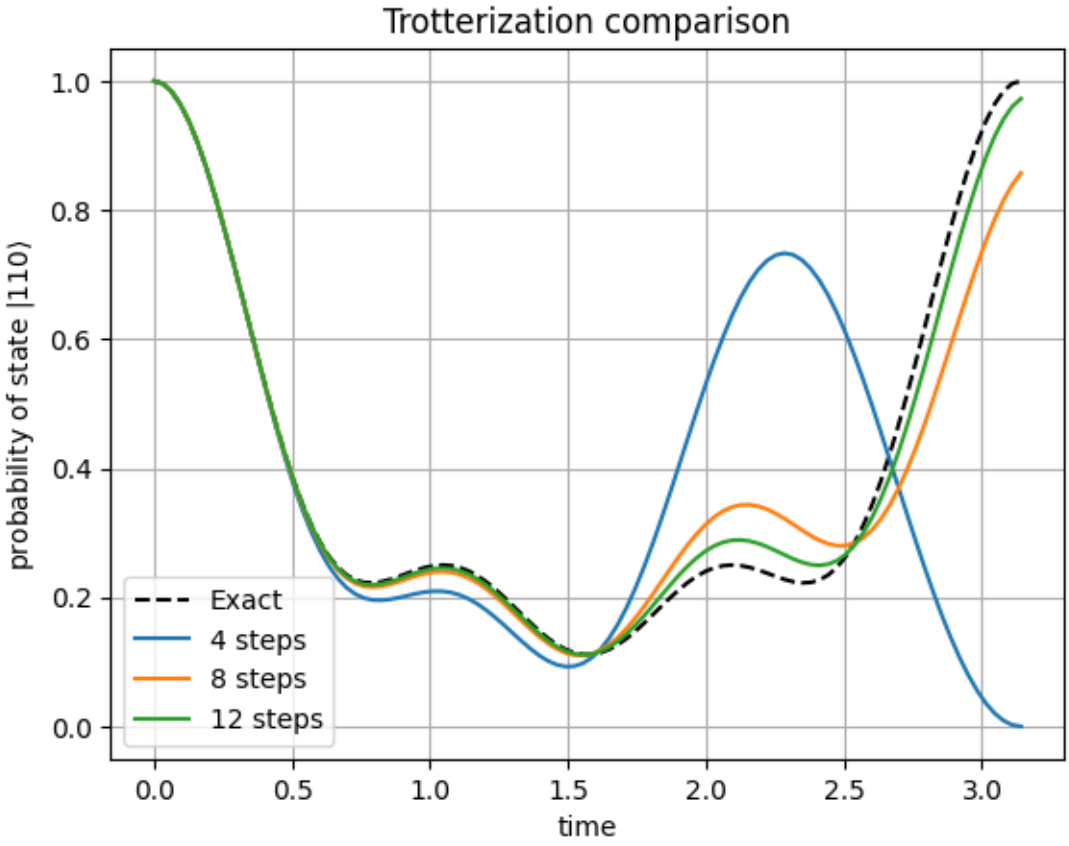
Now we will use the function created to compare the Trotterization with the exact evolution.

[39]:
```
# Simulate the system with a different number of Trotter steps and compare with␣
 ↪the matrix exponentiation
# In this case, we will consider 4, 8 and 12 Trotter steps
probs_110_trott = {4:[],8:[],12:[]}

## We want to measure |110><110|
prj_3 = prj_1 ^ prj_1 ^ prj_0

# We loop over different number of Trotter steps
for n in probs_110_trott.keys():
    for sim_t in ts:
        # Initialize the circuit
        trott_qr = QuantumRegister(3)
        trott_qc = QuantumCircuit(trott_qr)
        trott_qc.x([1,2])

        # Append the Trotterization
        trott_steps = Heisenberg_Trotter(num_spins=3,trotter_steps=n,t=sim_t)
        #trott_qc   = trott_qc.compose(trott_steps)
        trott_qc.append(trott_steps, [trott_qr[0], trott_qr[1], trott_qr[2]])


        # Measure
        prob = estimator.run([(trott_qc,prj_3)]).result()[0].data.evs.real
        probs_110_trott[n].append(prob)

    print(str(n)+" steps completed")
```

```
4 steps completed

8 steps completed

12 steps completed
```

[40]:
```
# Now plot the comparison
plt.plot(ts, probs_110_exact,linestyle="dashed",color="black",label="Exact")
for (i,n) in enumerate(probs_110_trott.keys()):
    plt.plot(ts, probs_110_trott[n],color="C"+str(i),label=str(n)+" steps")
plt.xlabel('time')
plt.ylabel(r'probability of state $|110\rangle$')
plt.title(r'Trotterization comparison')
plt.legend()
```

```
plt.grid()
plt.show()
```



As we can see from the plot , the dynamics gets closer and closer to the exact state evolution as we increase the number of Trotter steps. This comes at the cost of increasing the number of gates in our quantum circuit.