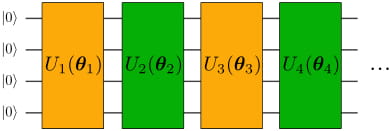
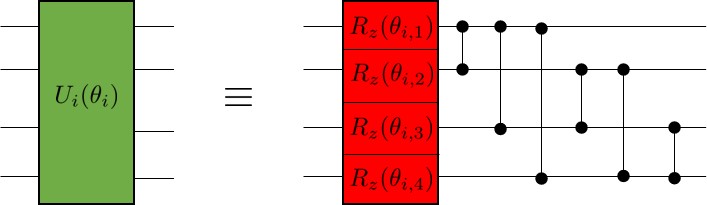
**QC exercises**

These tasks have been designed

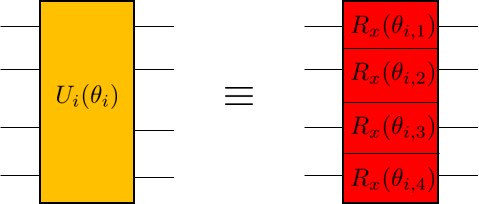
Where the number of layers, denoted with L, has to be considered as a parameter. We call ¨Layer¨ the combination of 1 yellow + 1 green block, so, for example, U1 + U2 is a layer. The odd/even variational blocks are given by:



Even blocks



Odd blocks



The angles θ\_(i,n)are variational parameters, lying in the interval (0,2π), initialized at random. Double qubit gates are CZ gates.

Report with a plot, as a function of the number of layers, L, the minimum distance

ϵ= min\_θ || |ψ(θ)> - |ϕ> ||

Where |ϕ> is a randomly generated vector on 4 qubits and the norm || | v> ||, of a state | v>, simply denotes the square root of the sum of the modulus square of the components of |v >. The right set of parameters θ\_(i,n) can be found via any method of choice (e.g. grid-search or gradient descent)

Bonus question:

Try using other gates for the parametrized gates and see what happens.

**Complete in task1.py**

**200 layers in all, in each layer, U1 stands for even**

**Rz as R1[0:7]（[[exp(-1j\*theta/2), exp(1j\*theta/2)], [exp(-1j\*theta/2), exp(1j\*theta/2)]]）; CNOT as CNOT**

**U2 stands for odd**

**Rx as R2[0:7];(Commented [[cos(theta/2), -sin(theta/2)],[cos(theta/2), sin(theta/2)]])**

**Bonus:**

**As the comments , substitute Rx with Ry:**

**RY as new R2[0:7];(uncommented [[cos(theta/2), 1j\*sin(theta/2)],[cos(theta/2), 1j\*sin(theta/2)]])**

**Figures: err\_200.png,(detailed trend with layer in 100 epochs sees err\_100.png), theta(digree)\_200, theta(rad)\_200.png, Bonus: err\_200(Bonus).png, Bones\_theta(digree) \_200.png, Bones\_theta\_200.png**

Task 2

Implement a circuit that returns |01> and |10> with equal probability (50% for each).

Requirements :

The circuit should consist only of CNOTs, RXs and RYs.

Start from all parameters in parametric gates being equal to 0 or randomly chosen.

You should find the right set of parameters using gradient descent (you can use more advanced optimization methods if you like).

Simulations must be done with sampling (i.e. a limited number of measurements per iteration) and noise.

Compare the results for different numbers of measurements: 1, 10, 100, 1000.

Bonus question:

How to make sure you produce state |01> + |10> and not |01> - |10> ?

(Actually for more careful readers, the “correct” version of this question is posted below:

How to make sure you produce state |01⟩ + |10⟩ and not any other combination of |01> + e(i\*phi)|10⟩ (for example |01⟩ - |10⟩)?)

**0.Initial from |0> |0> two qubits**

**RX(theta) on q0 ： cos(theta)|0>+sin(theta)|1>**

**Input Output (equivalent to H)**

**00**

**10**

**Result:（cos(theta)|0>+sin(theta)|1>）|0>**

**CX(CNOT) : CNOT(（cos(theta)|0>+sin(theta)|1>）|0>**

**Input Output**

**00 00**

**10 11**

**Result: cos(theta)|00>+sin(theta)|11>**

**RY(pi) on q1 ： |1>|0> + |0>|1>**

**Input Output (equivalent to ancilla, not x)**

**00 01**

**11 10**

**Result: cos(theta)|01>+sin(theta)|10>**

**By hand:**

**Prob(‘01’) = Prob(‘00’) = 0.5**

**Prob(‘10’) = Prob(‘11’) = 0.5**

**Implement on qiskit see the qiskit.png (h,s,h  RX(theta) ; cx; h,h  not )**

**And the circuit on circuit.png**

**With different measurement numbers: fig1.png- fig4.png**

**1:**

**{**

**‘01’: 1,**

**‘10’: 0}**

**10:**

**{**

**‘01’:6,**

**‘10’: 4}**

**100:**

**{**

**‘01’:53,**

**‘10’: 47}**

**1000:**

**{**

**‘01’:535,**

**‘10’: 465}**

**Complete in task2.txt**

**Bonus: apply RY(theta) to rotate more, in case :from |01>- |10> to |01>+ |10> just use theta = pi**

Task 3

Please write a simple compiler – program, which translates one quantum circuit into another, using a restricted set of gates.

You need to consider just the basic gates for the input circuit, such as (I, H, X, Y, Z, RX, RY, RZ, CNOT, CZ).

The output circuit should consist only from the following gates: RX, RZ, CZ. In other words, each gate in the original circuit must be replaced by an equivalent combination of gates coming from the restricted set (RX, RZ, CZ) only.

For example, a Hadamard gate after compilation looks like this:

H:⬄

RZ(-pi/2)

RX(pi/2)

RZ(pi/2)

RX = [[np.cos(theta/2), -np.sin(theta/2)],[np.cos(theta/2), np.sin(theta/2)]]

RZ = [[np.exp(-1j\*theta/2), 0],[0,np.exp(-1j\*theta/2)]]

CZ = [[1 0 0 0],[0 1 0 0],[0 0 1 0],[0 0 0 -1]]

H: =1/ sqrt(2) \*[[1,1],[1,-1]] = [[np.exp(-1j\*pi/4), 0],[0,np.exp(-1j\*pi/4)]]\* [[np.cos(pi/4), -np.sin(pi/4)],[np.cos(pi/4), np.sin(pi/4)]]\* [[np.exp(-1j\*pi/4), 0],[0,np.exp(-1j\*pi/4)]]

⬄

RZ(pi/2)

RX(pi/2)

RZ(pi/2)

I: = [[1,0],[0,1]]⬄RZ(2\*pi)

X: = [[0,1],[1,0]]⬄

=-RX(pi)

Y = [[0,-1j],[1j,0]]

= Rx(np.pi)\*Rz(np.pi)\*Rz(np.pi/2)

Z = [[1,0],[0,-1]]

=CZ(np.pi)

RY= RZ(Y)

Analyze what’s the overhead of the compiled program compared to the original one and propose how to improve it. What we mean by overhead is the following: by replacing all the initial gates with the restricted set of gates given in the problem, you will see that the resulting circuit is much more involved than the original one. This is what we called the overhead, and you may think about how to treat this problem, i.e. you could try to simplify as much as possible the resulting circuit.

Task 4

Find the lowest eigenvalue of the following matrix:

[1 0 0 0;

0 0 -1 0;

0 -1 0 0;

0 0 0 1]

using VQE-like circuits, created by yourself from scratch.

In general, this exercise might be pretty difficult, so below there are a couple of tips. We’ve written them in a pale font so that those of you who embrace the challenge don’t look at them accidentally ;)

Requirements:

It requires decomposing the matrix to the sum of Pauli terms.

Decomposition involves only terms consisting of the same matrices, i.e.: II, XX, YY, ZZ, and the coefficients are from the set [-1/2, -1, 0, 1, 1/2].

The ansatz you can use is: (RX I) CX (HI) |00>, where angle in RX is your variational parameter.

You can find an explanation of VQE in this blog post, You can also find links to further resources there.

You can just search through all angles for RX, you don’t need to use any optimizers like gradient descent.

**H = H1 + H2 + H3 + H4 = a\*XX+b\*YY+c\*ZZ+d\*II**

**a\*cos(theta) + b\*cos(theta) +c\*exp(i\*theta/2)+ d = 1**

**-aj\*sin(theta) +bj\*sin(theta) = 0**

**~~a\*exp(-i\*theta/2) + b\*exp(i\*theta/2) = 0~~**

**a\*cos(theta)+ b\*cos(theta)+c\*exp(-i\*theta/2) + d = 0**

**-aj\*sin(theta)-bj\*sin(theta) = -1**

**-aj\*sin(theta)+bj\*sin(theta) = 0**

**a\*cos(theta) + b\*cos(theta) +c\*exp(-i\*theta/2)+ d = -1**

**~~a\*cos(theta)+ b\*cos(theta)+c\*exp(i\*theta/2) + d = 1~~**

**a = -j/(2\*sin(theta)), b = a**

**c = 1/(2\*exp(i\*theta/2)+ exp(-i\*theta/2)), d = 1+j\*cot(theta)-1/(2+exp(-j\*theta))**

**Search see theTask4.py**