QOSF - Problem 1

September 25, 2020

1 QOSF Problem-1: Generating a Random State Using Parametric Circuits

In this program, we will use multi-layer ansatz circuits to generate a random 4-qubit state. We will also see how the optimization error depends on the number of layers.

We will be coding using IBM's open-source Quantum SDK QISKit.

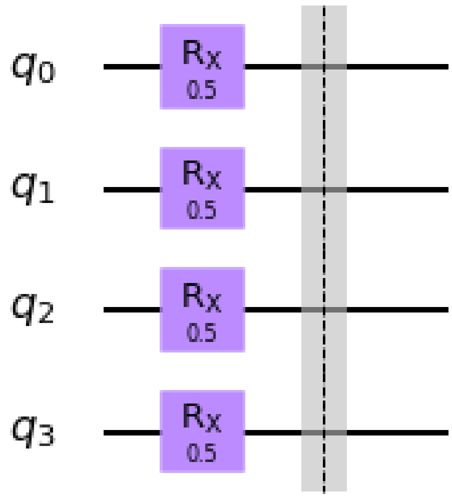
[1]:
```
import numpy as np
np.random.seed(20)

from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
```

2 Building Blocks of The Parametric Circuit

2.1 Odd Block

[2]:
```
# defining odd block
def oddblock(params):
    qc = QuantumCircuit(4)

    qc.rx(params[0], 0)
    qc.rx(params[1], 1)
    qc.rx(params[2], 2)
    qc.rx(params[3], 3)

    qc.barrier()

    return qc

# visualizing oddblock to check
oddblock([0.5, 0.5, 0.5, 0.5]).draw(output='mpl')
```

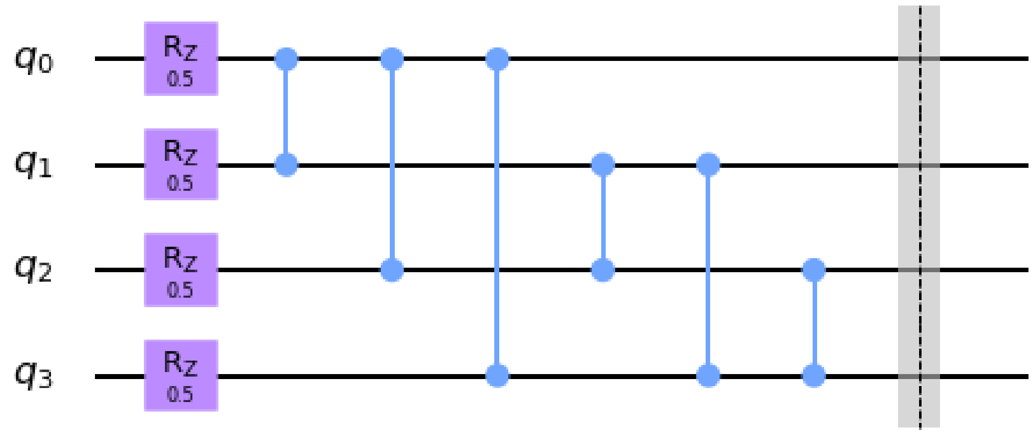
[2]:



2.2 Even Block

[3]:
```
#defining even block

def evenblock(params):
    qc = QuantumCircuit(4)

    qc.rz(params[0], 0)
    qc.rz(params[1], 1)
    qc.rz(params[2], 2)
    qc.rz(params[3], 3)

    qc.cz(0, 1)
    qc.cz(0, 2)
    qc.cz(0, 3)

    qc.cz(1, 2)
    qc.cz(1, 3)

    qc.cz(2, 3)

    qc.barrier()

    return qc

# visualizing evenblock to check
evenblock([0.5, 0.5, 0.5, 0.5]).draw(output='mpl')
```

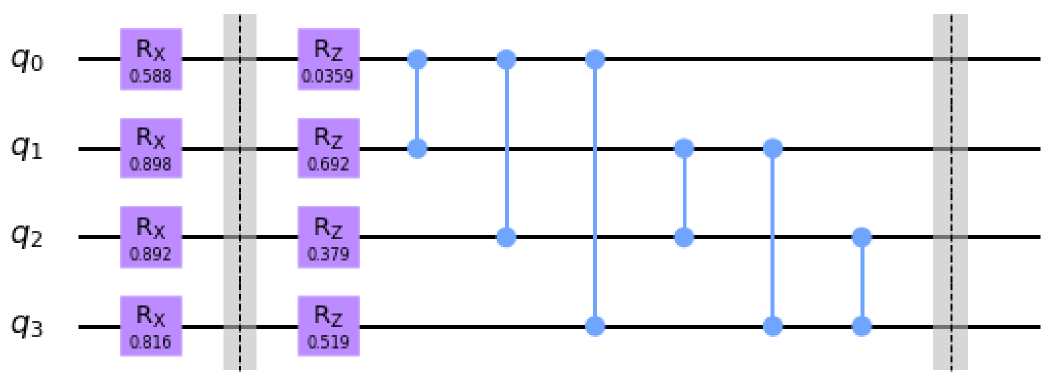
[3]:



2.3 Complete Circuit

[4]:
```
def get_var_form(LAYERS, params):

    circ = QuantumCircuit(4)

    for i in range(LAYERS):
        circ = circ + oddblock(params[8*i:8*i+4])
        circ = circ + evenblock(params[8*i+4:8*i+8])

    return circ

params = np.random.rand(8*1)

# visualizing complete circuit to check
get_var_form(1, params).draw(output='mpl')
```

[4]:



3 Optimizing

3.1 Creating Target State

[5]:
```
#creating random target state
target_state = np.random.rand(16) + np.random.rand(16)*1j

#normalizing target state
target_state = target_state/(np.linalg.norm(target_state))
```

3.2 Configuring Simulator

[6]:
```
# Import Aer
from qiskit import Aer, execute

# Run the quantum circuit on a statevector simulator backend
backend = Aer.get_backend('statevector_simulator')
```

3.3 Creating Cost Function

[7]:
```
def objective_function(params):
    # Obtain a quantum circuit instance from the paramters
    circ = get_var_form(LAYERS, params)

    # Execute the quantum circuit to obtain the probability distribution␣
 ↪associated with the current parameters
    result = execute(circ, backend).result()
    output_state = result.get_statevector(circ)

    # cost
    cost = np.linalg.norm(output_state - target_state)
    return cost

LAYERS = 1
params = np.random.rand(8*LAYERS)
objective_function(params)
```

[7]: 1.2825920097154815

3.4 Running optimizer

[8]:
```
from qiskit.aqua.components import optimizers

# for timing the optimization sessions
import time
```

[9]:
```
def optimize_circuit(params, LAYERS, iteration, bounds):

    # Initialize the COBYLA optimizer
    optimizer = optimizers.COBYLA(maxiter=iteration, tol=0.0001)

    # run optimizer
    ret = optimizer.optimize(num_vars=len(params),␣
 ↪objective_function=objective_function,
                                variable_bounds=bounds, initial_point=params)

    return ret[0], ret[1]
```

Note: The following code-block generates error-results for simulations. You can change the number of optimizer-iterations and the maximum layer count up to which you want to simulate, in the first line.

For 1000 optimizer iterations, and simulating up to 10 layers, the following code-block needs approximately **4** minutes to run.

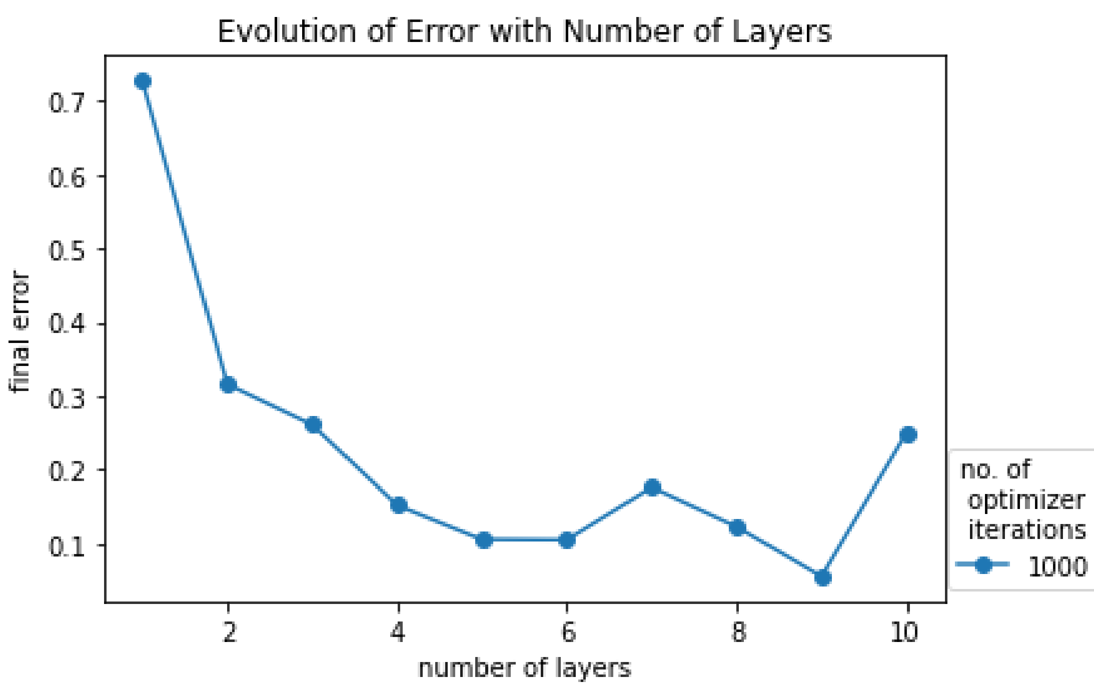
[10]:
```
# no. of iterations and maximum no. of layers
# change these two variables if required
iteration, max_layer_count = 1000, 10

print("Running", iteration, "optimizer iterations per session...")
# timing each iteration
iter_start = time.time()

# simulation results will be stored in this list
data = []

for LAYERS in range(1, max_layer_count+1):
    print("Optimizing a", LAYERS, "layer circuit...")
    layer_start = time.time()

    # Creating random initial parameters in the range of 0 to 2pi
    params = np.random.rand(8*LAYERS)*2*np.pi
    # bounds of the parameters
    bounds = [(0, 2*np.pi)]*len(params)

    # optimizing circuit for the given number of layers
```

```
    solution, error = optimize_circuit(params, LAYERS, iteration, bounds)
    print(iteration, "iterations", LAYERS, "layers... Optimized! \n",␣
↪round(time.time()-layer_start), "seconds for this layer.")

    data.append([LAYERS, error, solution])

print("All circuits optimized for", iteration, "iterations! \n\n",
      round(time.time()-iter_start), "seconds for running", iteration,␣
↪"iterations per session.\n\n")

final_result = [[iteration, data]]
```

```
1000 iterations 2 layers… Optimized!
 18 seconds for this layer.
Optimizing a 3 layer circuit…
1000 iterations 3 layers… Optimized!
 24 seconds for this layer.
Optimizing a 4 layer circuit…
1000 iterations 4 layers… Optimized!
 29 seconds for this layer.
Optimizing a 5 layer circuit…
1000 iterations 5 layers… Optimized!
 35 seconds for this layer.
Optimizing a 6 layer circuit…
1000 iterations 6 layers… Optimized!
 40 seconds for this layer.
Optimizing a 7 layer circuit…
1000 iterations 7 layers… Optimized!
 47 seconds for this layer.
Optimizing a 8 layer circuit…
1000 iterations 8 layers… Optimized!
 52 seconds for this layer.
Optimizing a 9 layer circuit…
1000 iterations 9 layers… Optimized!
 59 seconds for this layer.
Optimizing a 10 layer circuit…
1000 iterations 10 layers… Optimized!
 66 seconds for this layer.
All circuits optimized for 1000 iterations!

 371 seconds for running 1000 iterations per session.
```

3.5 Plot Results for Comparison

[11]:
```
import matplotlib.pyplot as plt

# function to plot results for comparison
def plot_result(max_layer_count, results):

    layer_list = [i for i in range(1, max_layer_count+1)]

    for iter_data in results:
        iter_num, error = iter_data[0], ([i[1] for i in iter_data[1]])

        plt.plot(layer_list, error, label=str(iter_num), marker='o', ls='-')

    plt.xlabel('number of layers')
    plt.ylabel('final error')

    plt.title('Evolution of Error with Number of Layers')

    plt.legend(loc="lower right", bbox_to_anchor=(1.2, 0), title='no. of \n␣
↪optimizer\n iterations')
    plt.show()


plot_result(10, final_result)
```



You can use the following code-block to see the simulation results for a particular number of layers. Just change the "*layers*" variable below.

[12]:
```
# change the layer variable to the number of layers for which you want to see␣
 ↪the result
layer = 10


solution = final_result[0][1][layer-1][2]

# Obtain a quantum circuit instance from the paramters
circ = get_var_form(layer, solution)

# Create a Quantum Program for execution
result = execute(circ, backend).result()
output_state = result.get_statevector(circ)

# cost
error = np.linalg.norm(output_state - target_state)

print("Target State:", target_state)
print("Output State:", output_state)
print("Error:", error)
```

```
Target State: [0.20394689+0.23875695j 0.06008825+0.1526246j
0.08441061+0.19567112j
 0.22274811+0.26022131j 0.24270962+0.14290956j 0.26357822+0.15434775j
 0.24030463+0.21059879j 0.01136493+0.20172577j 0.03617185+0.08331914j
 0.23287639+0.0208688j  0.07415108+0.23912681j 0.07898287+0.14909188j
 0.26584037+0.10204495j 0.29440542+0.15828471j 0.17410751+0.0817177j
 0.05541705+0.09625006j]
Output State: [0.24183412+0.14933466j 0.11107756+0.11450198j
0.01931973+0.1808261j
 0.15833298+0.25165699j 0.28525828+0.13351815j 0.31737773+0.14801199j
 0.2672598 +0.25213391j 0.08274669+0.25154055j 0.04590854+0.07644953j
 0.19337966+0.0020517j  0.06127235+0.21482642j 0.11974764+0.16486555j
 0.21406406+0.0649974j  0.27951539+0.20075111j 0.1169302 +0.0018492j
 0.03274196+0.15917639j]
Error: 0.2488978767168584
```

3.6 Additional

The following block of code can be run to see the dependence of final error on the numbmer of optimization iterations. You can change the list ITER_LIST below, to see the results for a particular number of iterations.

Note: For the given max_layer_count and ITER_LIST, the code-block requires approximately **30** minutes to run.

[13]:
```
max_layer_count = 8
ITER_LIST = [100, 250, 500, 750, 1000, 3000]

iter_results = []
for iteration in ITER_LIST:
    print("Running", iteration, "optimizer iterations per session...")

    # results for each iteration will be stored in this list
    data = []

    # timing each iteration
    iter_start = time.time()

    for LAYERS in range(1, max_layer_count+1):
        print("Optimizing a", LAYERS, "layer circuit...")
        layer_start = time.time()
        # Creating random initial parameters in the range of 0 to 2pi
        params = np.random.rand(8*LAYERS)*2*np.pi

        # bounds of the parameters
        bounds = [(0, 2*np.pi)]*len(params)

        # optimizing circuit for the given number of layers
        solution, error = optimize_circuit(params, LAYERS, iteration, bounds)
        print(iteration, "iterations", LAYERS, "layers... Optimized! \n",␣
↪round(time.time()-layer_start), "seconds for this layer.")

        data.append([LAYERS, error, solution])

    print("All circuits optimized for", iteration, "iterations! \n\n",
          round(time.time()-iter_start), "seconds for running", iteration,␣
↪"iterations per session.\n\n")

    iter_results.append([iteration, data])
```

```
Running 100 optimizer iterations per session…
Optimizing a 1 layer circuit…
100 iterations 1 layers… Optimized!
 1 seconds for this layer.
Optimizing a 2 layer circuit…
100 iterations 2 layers… Optimized!
 2 seconds for this layer.
Optimizing a 3 layer circuit…
100 iterations 3 layers… Optimized!
 2 seconds for this layer.
```

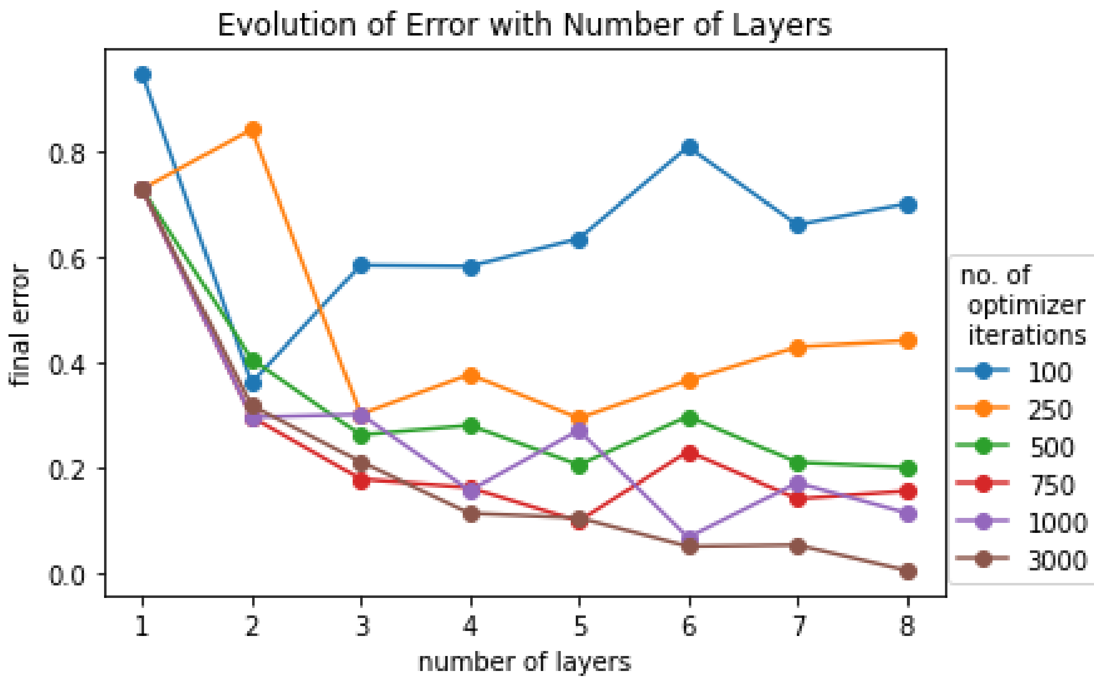
```
Optimizing a 4 layer circuit…
100 iterations 4 layers… Optimized!
 3 seconds for this layer.
Optimizing a 5 layer circuit…
100 iterations 5 layers… Optimized!
 4 seconds for this layer.
Optimizing a 6 layer circuit…
100 iterations 6 layers… Optimized!
 4 seconds for this layer.
Optimizing a 7 layer circuit…
100 iterations 7 layers… Optimized!
 5 seconds for this layer.
Optimizing a 8 layer circuit…
100 iterations 8 layers… Optimized!
 5 seconds for this layer.
All circuits optimized for 100 iterations!

 26 seconds for running 100 iterations per session.


Running 250 optimizer iterations per session…
Optimizing a 1 layer circuit…
250 iterations 1 layers… Optimized!
 3 seconds for this layer.
Optimizing a 2 layer circuit…
250 iterations 2 layers… Optimized!
 5 seconds for this layer.
Optimizing a 3 layer circuit…
250 iterations 3 layers… Optimized!
 6 seconds for this layer.
Optimizing a 4 layer circuit…
250 iterations 4 layers… Optimized!
 7 seconds for this layer.
Optimizing a 5 layer circuit…
250 iterations 5 layers… Optimized!
 9 seconds for this layer.
Optimizing a 6 layer circuit…
250 iterations 6 layers… Optimized!
 10 seconds for this layer.
Optimizing a 7 layer circuit…
250 iterations 7 layers… Optimized!
 12 seconds for this layer.
Optimizing a 8 layer circuit…
250 iterations 8 layers… Optimized!
 13 seconds for this layer.
All circuits optimized for 250 iterations!

 64 seconds for running 250 iterations per session.
```

```
Running 500 optimizer iterations per session…
Optimizing a 1 layer circuit…
500 iterations 1 layers… Optimized!
 3 seconds for this layer.
Optimizing a 2 layer circuit…
500 iterations 2 layers… Optimized!
 9 seconds for this layer.
Optimizing a 3 layer circuit…
500 iterations 3 layers… Optimized!
 12 seconds for this layer.
Optimizing a 4 layer circuit…
500 iterations 4 layers… Optimized!
 15 seconds for this layer.
Optimizing a 5 layer circuit…
500 iterations 5 layers… Optimized!
 18 seconds for this layer.
Optimizing a 6 layer circuit…
500 iterations 6 layers… Optimized!
 20 seconds for this layer.
Optimizing a 7 layer circuit…
500 iterations 7 layers… Optimized!
 24 seconds for this layer.
Optimizing a 8 layer circuit…
500 iterations 8 layers… Optimized!
 26 seconds for this layer.
All circuits optimized for 500 iterations!

 126 seconds for running 500 iterations per session.


Running 750 optimizer iterations per session…
Optimizing a 1 layer circuit…
750 iterations 1 layers… Optimized!
 2 seconds for this layer.
Optimizing a 2 layer circuit…
750 iterations 2 layers… Optimized!
 14 seconds for this layer.
Optimizing a 3 layer circuit…
750 iterations 3 layers… Optimized!
 18 seconds for this layer.
Optimizing a 4 layer circuit…
750 iterations 4 layers… Optimized!
 22 seconds for this layer.
Optimizing a 5 layer circuit…
750 iterations 5 layers… Optimized!
 26 seconds for this layer.
```

```
Optimizing a 6 layer circuit…
750 iterations 6 layers… Optimized!
 30 seconds for this layer.
Optimizing a 7 layer circuit…
750 iterations 7 layers… Optimized!
 36 seconds for this layer.
Optimizing a 8 layer circuit…
750 iterations 8 layers… Optimized!
 40 seconds for this layer.
All circuits optimized for 750 iterations!

 188 seconds for running 750 iterations per session.


Running 1000 optimizer iterations per session…
Optimizing a 1 layer circuit…
1000 iterations 1 layers… Optimized!
 5 seconds for this layer.
Optimizing a 2 layer circuit…
1000 iterations 2 layers… Optimized!
 18 seconds for this layer.
Optimizing a 3 layer circuit…
1000 iterations 3 layers… Optimized!
 24 seconds for this layer.
Optimizing a 4 layer circuit…
1000 iterations 4 layers… Optimized!
 30 seconds for this layer.
Optimizing a 5 layer circuit…
1000 iterations 5 layers… Optimized!
 35 seconds for this layer.
Optimizing a 6 layer circuit…
1000 iterations 6 layers… Optimized!
 40 seconds for this layer.
Optimizing a 7 layer circuit…
1000 iterations 7 layers… Optimized!
 47 seconds for this layer.
Optimizing a 8 layer circuit…
1000 iterations 8 layers… Optimized!
 53 seconds for this layer.
All circuits optimized for 1000 iterations!

 253 seconds for running 1000 iterations per session.


Running 3000 optimizer iterations per session…
Optimizing a 1 layer circuit…
3000 iterations 1 layers… Optimized!
 2 seconds for this layer.
```

```
Optimizing a 2 layer circuit…
3000 iterations 2 layers… Optimized!
 56 seconds for this layer.
Optimizing a 3 layer circuit…
3000 iterations 3 layers… Optimized!
 72 seconds for this layer.
Optimizing a 4 layer circuit…
3000 iterations 4 layers… Optimized!
 91 seconds for this layer.
Optimizing a 5 layer circuit…
3000 iterations 5 layers… Optimized!
 107 seconds for this layer.
Optimizing a 6 layer circuit…
3000 iterations 6 layers… Optimized!
 126 seconds for this layer.
Optimizing a 7 layer circuit…
3000 iterations 7 layers… Optimized!
 144 seconds for this layer.
Optimizing a 8 layer circuit…
3000 iterations 8 layers… Optimized!
 160 seconds for this layer.
All circuits optimized for 3000 iterations!

 758 seconds for running 3000 iterations per session.
```

[14]:
```
plot_result(max_layer_count, iter_results)
```



3.7 References

[1] Quantum Computation and Quantum Information, M. A. Nielsen & I. L. Chuang

[2] Simulating Molecules using VQE, Qiskit Textbook

[ ]:

```

```