-----------TASK 2: Brief Explanation-------------

--QASM Simulator is used (from Qiskit aer) to generate noise.
--Circuit consists of 2 wires (initialized as |0>), 1 RY gate on each, and then a CNOT with Wire-1 as Target.
--Circuit outputs the Probablity of each state (|00>,|01>,|10>,|11>).
--Gate-Parameters (Theta[0,1]) are randomly initialzed and converged to local minimas at pi/2 & pi respectively.
--Natural Gradient Descent is used to optimize the Prob. Dist. to the Desired Values (0,0.5,0.5,0).
--Diffirent degrees of Sampling (1/10/100/1000) are done / iteration of GDO.
--Bonus-Question answered at the end

In [ ]:
```
#Importing libraries
import qiskit
from qiskit import Aer
from math import pi
import pennylane as qml
from pennylane import numpy as np
import random
import pandas as pd
```

In [230]:
```
np.set_printoptions(suppress=True)   #Supressing Scientific Notation
#QASM Simulator used to generate noise
dev = qml.device('qiskit.aer', wires=2,backend='qasm_simulator',shots = 1000)
desired_probs = np.array([0,0.5,0.5,0])   #Desired State Probaility Distribution
```

In [3]:
```
#Defining the Circuit
@qml.qnode(dev)
def circuit(thetas):
    qml.RY(thetas[0],wires=0)
    qml.RY(thetas[1],wires=1)
    qml.CNOT(wires = [0,1])
    return qml.probs(wires=[0,1])   #Returns State Probability Distribution
```

In [4]:
```
#Defining the Cost Function
def cost_function(thetas):
    return sum(abs(circuit(thetas)-desired_probs))
```

In [256]:
```
#Optimization of State Probablity Distributions
iterations = [1,10,100,1000]   #Sampling Values
final_thetas = []
final_probs = []
eta = 0.01   #Learning Rate
steps = 200
for iter in iterations:
    dev.shots = iter   #Changing the sampling value
    #Randomly initializing Gate Parameters
    init_thetas = np.array([np.pi*random.random(),2*np.pi*random.random()])
    opt = qml.QNGOptimizer(eta)   #Natural Gradient Descent
    thetas_new = init_thetas
    for _ in range(steps):
          #Metric-Tensor is explicitly Provided to Optimizer
            thetas_new = opt.step(cost_function,thetas_new,metric_tensor_fn=circuit.metric_tensor)
    final_thetas.append(thetas_new)
    final_probs.append(circuit(thetas_new))
```

In [323]:
```
#Displaying & Comparing Results
final_thetas = np.array(final_thetas)*(180/(np.pi))   #Converting from radians to degrees
final_probs = np.array(final_probs)
pd.options.display.float_format = "{:,.2f}".format   #Rounding to 2 Decimals
row_labels = ['1', '10', '100', '1000']
column_labels =  ['|00>', '|01>', '|10>', '|11>']
column_labels2 = ['Theta-0','Theta-1']
df_probs =  pd.DataFrame(final_probs, columns=column_labels, index=row_labels)
df_thetas = pd.DataFrame(final_thetas, columns=column_labels2, index=row_labels)
df_probs.style.set_caption('State Probabilties for Each Sampling Case')
df_thetas.style.set_caption('Parameter Values for Each Sampling Case')
display(df_probs)
print("Probability Distribution \nfor Different Iterations\n\n")
display(df_thetas)
print("Paramter Values for\nDifferent Iterations\n\t(in Degrees)\n\n")
```

	|00>	|01>	|10>	|11>
1	0.00	0.48	0.52	0.00
10	0.00	0.49	0.51	0.00
100	0.00	0.48	0.51	0.01
1000	0.00	0.49	0.51	0.00

```
Probability Distribution
for Different Iterations
```

	Theta-0	Theta-1
1	91.34	177.04
10	91.99	182.31
100	91.05	189.86
1000	91.18	182.59

```
Paramter Values for
Different Iterations
        (in Degrees)
```

---Bonus Question---

--The Circuit constructed above can output |01> - |10> when the Parameter for RY-gate on Wire-0 is -pi/2,3pi/2..etc.
--Thus, we initialize theta[0] b/w (0-Pi), so gradient descent converges it to the local minima at pi/2.
--Thus outputing |01> + |10> always.