Para a implementação do protocolo, iniciaremos carregando as seguintes bibliotecas e funções. Vale ressaltar que as bibliotecas nos fornecem os recursos necessários para o desenvolvimento das operações. As funções **TensorProduct** e **sqrt** foram importadas para tornar as operações mais objetivas.

[1]:
```
import numpy as np
import math as mt
import sympy as sp
from sympy.physics.quantum import TensorProduct as TP
from math import sqrt
import sys
import random
```

Qubits iniciais

Para iniciar o protocolo, iremos definir os possíveis estados de entrada dos qubits que serão operacionalizados. As bases da Computação Quântica definem as bases $|0>$ e $|1>$ e sua representação matricial é descrita por $\begin{bmatrix}1\\0\end{bmatrix}$ e $\begin{bmatrix}0\\1\end{bmatrix}$ respectivamente.

Operadores quânticos

Devem ser definidos também, todas os operadores/portas lógicas que atuarão durante o protocolo. Sendo eles os operadores ***CNOT***, ***Hadamard***, ***Pauli-X***, ***Pauli-Z*** e ***Identidade***. Suas representações matriciais são descritas por:

$$CNOT = \begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&1\\0&0&1&0\end{bmatrix}, H = \frac{1}{\sqrt{2}}\begin{bmatrix}1&1\\1&-1\end{bmatrix}, X = \begin{bmatrix}0&1\\1&0\end{bmatrix}, Z = \begin{bmatrix}1&0\\0&-1\end{bmatrix} eI = \begin{bmatrix}1&0\\0&1\end{bmatrix}.$$

[2]:
```
#Variáveis que descrevem os qubits
qbit0= np.matrix([1,0]).transpose()
qbit1= np.matrix([0,1]).transpose()

#Variáveis que descrevem os operadores quânticos
#CNOT
CNOT = np.matrix([[1,0,0,0],[0,1,0,0],[0,0,0,1],[0,0,1,0]])

#Hadamard
```

```
H = 1/sqrt(2)*(np.matrix([[1,1], [1,-1]]))

# Pauli-X
X = np.matrix([[0, 1], [1, 0]])

# Pauli-Z
Z = np.matrix([[1, 0], [0, -1]])

#Identidade
I = np.matrix ([[1,0], [0,1]])
```

Emaranhamento

O protocolo é iniciado com a realização do emaranhamento de modo que aplicaremos a porta ***H*** no estado $|0>$ e em seguida, definiremos o qubit alvo e o qubit controle para a aplicação da porta ***CNOT***. O qubit alvo e controle são definidos realizando o produto tensorial do primeiro pelo segundo de modo que: $AC = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ Vale ressaltar que nesse caso estamos emaranhando dois qubits no estado $|0>$ criando uma das Bases de Bell que, nesse caso é descrita por:

$$\Phi_+ = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

[3]:
```
#Aplicação da porta Hadamard no qubit-0
Hqbit0= H*qbit0

#Definição Alvo e Controle
AC= TP(Hqbit0,qbit0)

#Aplicação da porta CNOT
Bell00 = CNOT * AC
```

Início do protocolo

Iremos agora definir o estado do qubit que será teletransportado. Para isso é solicitado ao usuário que insira as probabilidades de α e β desejados. Lembrando que estas devem

estar normalizadas e sua soma deve ser igual a 1, caso contrário o programa não pode continuar operando. Como o estado do qubit generalizado é definido por:

$$|\psi >= \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Atribuímos os valores inseridos de α e β ao estado inicial.

[4]:
```
#Determinação do qubit para o protocolo
estado_alfa = float(input('Qual o estado alfa do Qubit que deseja␣
 ↪enviar?'))
estado_beta = float(input('Qual o estado beta do Qubit que deseja␣
 ↪enviar?'))



#Verifica se a soma é igual a 1 (condição de normalização)
if not mt.isclose(estado_alfa  + estado_beta, 1.0):
    print("Erro: A soma das entradas não é igual a 1.")
    sys.exit()

#Determina a mensagem a ser enviada
alfa_inicial = (estado_alfa * qbit0)
beta_inicial = (estado_beta * qbit1)
estado_inicial = alfa_inicial + beta_inicial

print(estado_inicial)
```

A próxima etapa do protocolo consiste na aplicação da porta ***CNOT*** nos três qubits que compõem o estado geral do sistema. Os três qubits são o estado inicial a ser teletransportado atuando como controle e o par emaranhado atuando como alvo. A atuação ocorre apenas no qubit presente no local **A**, porém afeta probabilisticamente o qubit no local **B**. Para definir o estado geral são somados os produtos tensoriais de todos os estados dos qubits. Em seguida, a porta ***CNOT*** é dimensionada para atuar sob três qubits com a realização do produto tensorial entre ***CNOT*** e ***I*** e por último multiplicamos a porta ***CNOT*** pela soma dos estados dos qubits. Esse estado é chamado de $|\psi_1 >$ e pode ser definido por:

$$|\psi_1 >= \frac{1}{\sqrt{2}} \left\{ \alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \right\}$$

Em seguida, a porta $\boldsymbol{H}$ é aplicada no qubit a ser teletransportado. Sua aplicação foi realizada em cada um dos estados do qubit de maneira isolada para melhor observação da modificação dos estados deste. Após a aplicação da porta $\boldsymbol{H}$ o estado $|\psi_2>$ é definido por:

$$|\psi_2>=\frac{1}{2}\left\{\alpha\begin{bmatrix}1\\1\end{bmatrix}\begin{bmatrix}1\\0\\0\\1\end{bmatrix}+\beta\begin{bmatrix}1\\-1\end{bmatrix}\begin{bmatrix}0\\1\\1\\0\end{bmatrix}\right\}$$

[5]:
```
#Estado geral (produto tensorial entre os estados de todos os qubits␣
 ↪do sistema)
psi_0 = TP(qbit0, qbit0, qbit0) + TP(qbit0, qbit1, qbit1) +␣
 ↪TP(qbit1,qbit0, qbit0) + TP(qbit1,qbit1,qbit1)

#Dimensionamento da porta CNOT para 3 qubits
CNOT_I = TP(CNOT,I)

#Aplicação da porta CNOT
psi_1 = CNOT_I * psi_0

#Aplicação da porta Hadamard
H_estado_alfa = H * alfa_inicial
H_estado_beta = H * beta_inicial
```

Para realizar a Medição e a Reconstrução do estado inicial, iremos separar os estados associados às probabilidades α e β de modo que:

$$\alpha\begin{bmatrix}1\\1\end{bmatrix}=\alpha\begin{bmatrix}1\\0\end{bmatrix}+\alpha\begin{bmatrix}0\\1\end{bmatrix}\quad e\quad \beta\begin{bmatrix}1\\-1\end{bmatrix}=\beta\begin{bmatrix}1\\0\end{bmatrix}-\beta\begin{bmatrix}0\\1\end{bmatrix}$$

[6]:
```
#Separação dos estados Alfa e Beta

estadoa0 = float(H_estado_alfa[0][0])
estadoa1 = float(H_estado_alfa[1][0])
estadob0 = float(H_estado_beta[0][0])
estadob1 = float(H_estado_beta[1][0])

#Determinação dos estados do qbit enviado
a0 = np.matrix([estadoa0,0]).transpose()
a1 = np.matrix([0,estadoa1]).transpose()
b0 = np.matrix([estadob0,0]).transpose()
```

```
b1 = np.matrix([0,estadob1]).transpose()
```

Para realizar a medição, foram estabelecidos os possíveis estados presentes em **A**, e um deles foi escolhido de maneira aleatória, lembrando que a probabilidade de colapso em cada um dos estados é de $\frac{1}{4}$. Os estados que não foram escolhidos foram deletados, simulando o colapso do sistema.

[7]:
```
#Medição

estado_00 = TP(qbit0, qbit0)
estado_11 = TP(qbit1, qbit1)
estado_10 = TP(qbit1, qbit0)
estado_01 = TP(qbit0, qbit1)

group_estados = [estado_00,estado_11,estado_10,estado_01]
chosen_estados = random.choice(group_estados)
Medição = chosen_estados

del estado_00, estado_11, estado_10, estado_01

print(Medição)
```

As etapas a seguir acontecem no local **B** com o qubit do par emaranhado. Segundo o estado colapsado enviado por **A**, o qubit em **B** terá um estado correspondente de acordo com a relação:

Estado Medido em **A**	Estado do qubit emaranhado em **B**
$\|00>$	$\alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
$\|11>$	$\alpha \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \beta \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
$\|01>$	$\alpha \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \beta \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
$\|10>$	$\alpha \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \beta \begin{bmatrix} 0 \\ 1 \end{bmatrix}$

[8]:
```
#Define o estado em B em função da medição realizada em A

if  np.all(Medição == (TP(qbit0,qbit0))):
    estado_tp = (a0-b1)
```

```
elif  np.all(Medição == (TP(qbit1,qbit1))):
    estado_tp = (a1-b0)
elif  np.all(Medição == (TP(qbit0,qbit1))):
    estado_tp = (a1+b0)
elif np.all(Medição == (TP(qbit1,qbit0))):
    estado_tp = (a0+b1)
```

Agora, de acordo com o estado associado em B, uma sequência de operações deverão ser realizadas conforme a relação:

Estado do qubit emaranhado em **B**	Operação realizada
$\alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix}$	Nenhuma operação
$\alpha\begin{bmatrix}0\\1\end{bmatrix} - \beta\begin{bmatrix}1\\0\end{bmatrix}$	X e Z
$\alpha\begin{bmatrix}0\\1\end{bmatrix} + \beta\begin{bmatrix}1\\0\end{bmatrix}$	X
$\alpha\begin{bmatrix}1\\0\end{bmatrix} - \beta\begin{bmatrix}0\\1\end{bmatrix}$	Z

[12]:
```
if np.all(estado_tp == (a0-b1)):
    estado_final = estado_tp
elif np.all(estado_tp == (a1-b0)):
    estado_final = Z * X * estado_tp
elif np.all(estado_tp == (a1+b0)):
    estado_final = X * estado_tp
elif np.all(estado_tp == (a0+b1)):
    estado_final = Z * estado_tp

estado_teletransportado = estado_final*sqrt(2)

alfa = float(estado_teletransportado[0][0])
beta = float(estado_teletransportado[1][0])

alfa_final = np.matrix([alfa,0]).transpose()
beta_final = np.matrix([0,beta]).transpose()
```

Se durante o processo, algum ruído ocorresse, o estado do qubit teletransportado seria alterado. A simulação de uma interação com os ruídos ***bitflip*** e ***phaseflip*** foi implementada de modo que o usuário pode selecionar se o teletransporte teve ou não ruído e em caso afirmativo, qual ruído ocorreu. O ruído do tipo ***bitflip*** inverte o estado

do qubit, ou seja, se este era $|0>$ passa a ser $|1>$ e vice-versa. O ruído do tipo ***phaseflip*** inverte a fase do qubit.

[ ]:
```
#Aplicação de ruído
#Determinando os ruídos
bitflip = np.matrix([[0, 1], [1, 0]])
phaseflip = np.matrix([[1, 0], [0, -1]])
noisebit = 0

# Escolha do ruído aplicado
while True:
    noise = input('Escolha o ruído:[0]- sem ruído, [1]- bitflip, [2]-␣
 ↪phaseflip: ')
    if noise in ['0', '1', '2']:
        noise = int(noise)
        break
    else:
        print("Opção inválida. Por favor, digite 0, 1, 2 ou 3.")

if noise == 1:
    noisebit = bitflip
elif noise == 2:
    noisebit = phaseflip
elif noise == 0:
    noisebit = 1

aplicação_noise_a = noisebit * alfa_final
aplicação_noise_b = noisebit * beta_final

alfa_noise = aplicação_noise_a
beta_noise = aplicação_noise_b

alfa_final = alfa_noise
beta_final = beta_noise
```

Por fim, para verificar se o teletransporte funcionou e ainda, se houve ou não ruído e qual a natureza deste, a sequência de testes abaixo foi implementada. Neles, comparamos os estados associados as probabilidades α e β antes(inicial) e depois(final) do teletransporte.

[11]:
```
if np.all((alfa_final == alfa_inicial) & (beta_final == beta_inicial)):
    print('Teletransporte concluído com sucesso e sem presença de␣
 ↪ruídos', alfa_final+beta_final)
else:
    if np.all((alfa_inicial[0][0] == alfa_final[1][0]) &␣
 ↪(alfa_inicial[1][0] == alfa_final[0][0]) & (beta_inicial[1][0] ==␣
 ↪beta_final[0][0]) & (beta_inicial[0][0] == beta_final[1][0])):
        print("Mensagem teletransportada com ruído do tipo bitflip",␣
 ↪alfa_final+beta_final)
    elif np.all((alfa_inicial[0][0] == alfa_final[0][0]) &␣
 ↪(alfa_inicial[1][0] == -alfa_final[1][0]) & (beta_inicial[1][0] ==␣
 ↪-beta_final[1][0]) &(beta_inicial[0][0] == beta_final[0][0])):
        print("Mensagem teletransportada com ruído do tipo phaseflip",␣
 ↪alfa_final+beta_final)
    else:
        print("Erro de teletransporte")
```

Vale ressaltar que o protocolo acima é uma simplificação de uma situação real e tem como finalidade compreender os procedimentos associados a ele.