Aplicações da Matriz Densidade_v1

April 15, 2021

1 Aplicações do operador densidade

Neste notebook, usaremos os conceitos aprendidos até agora, incluindo as ferramentas computacionais, em algumas aplicações físicas. Aproveitaremos, também, para extender algumas ideias, no sentido de descrever sistemas mais complexos, envolvendo múltiplas partículas.

[1]:
```
import sympy as sy
import numpy as np
import sympy.physics as phys
from sympy import symbols, Matrix, sin, cos, exp, diff, integrate
```

[2]:
```
sx = phys.matrices.msigma(1)  # sigma_x
sy = phys.matrices.msigma(2)  # sigma_y
sz = phys.matrices.msigma(3)  # sigma_z
I = Matrix([[1,0],[0,1]])     # unit matrix
```

1.1 Exemplo 1: sistemas compostos

Vamos considerar um sistema composto de duas partículas (ou dois sistemas físicos quaisquer) que, por simplicidade, vamos supor ser de dois níveis. Sejam A e B dois sistemas fechados de dois níveis representados pelas matrizes ρ_A e ρ_B:

$$\rho_A = \begin{pmatrix} 1/3 & 0 \\ 0 & 2/3 \end{pmatrix}; \quad \rho_B = \begin{pmatrix} 3/8 & 0 \\ 0 & 5/8 \end{pmatrix},$$

:: ***¡Desafio!*** :: *Para praticar, tente representar ρ_A e ρ_B como operadores...*

A matriz densidade do sistema composto é dada pelo **produto tensorial**

$$\begin{aligned} \rho_{AB} &= \rho_A \otimes \rho_B \\ \rho_A \otimes \rho_B &= \begin{pmatrix} 1/3 & 0 \\ 0 & 2/3 \end{pmatrix} \otimes \begin{pmatrix} 3/8 & 0 \\ 0 & 5/8 \end{pmatrix} \\ &= \begin{pmatrix} 1/8 & 0 & 0 & 0 \\ 0 & 5/24 & 0 & 0 \\ 0 & 0 & 1/4 & 0 \\ 0 & 0 & 0 & 5/12 \end{pmatrix} \end{aligned}$$

Como podemos facilmente verificar abaixo, usando **sympy** e **numpy**...

[3]:
```
rho_A = Matrix([['1/3', '0'],['0', '2/3']])
rho_B = Matrix([['3/8', '0'],['0', '5/8']])
```

[4]:
```
rho_A
```

[4]:
$$\begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{2}{3} \end{bmatrix}$$

[5]:
```
rho_B
```

[5]:
$$\begin{bmatrix} \frac{3}{8} & 0 \\ 0 & \frac{5}{8} \end{bmatrix}$$

Oserve que o produto ρ_A^2 produz o resultado esperado..

[6]:
```
rho_A*rho_A
```

[6]:
$$\begin{bmatrix} \frac{1}{9} & 0 \\ 0 & \frac{4}{9} \end{bmatrix}$$

Assim como, usando os métodos definidos para o **objeto** Matrix, podemos verificar que $\rho_A \cdot \rho_A^{-1} = I$

[7]:
```
rho_A*(rho_A.inv())
```

[7]:
$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Usando a biblioteca **sympy**, calculamos o produto tensorial com TensorProduct( )

[8]:
```
from sympy.physics.quantum import TensorProduct, Dagger
rho_AB = TensorProduct(rho_A, rho_B)
rho_AB
```

[8]:
$$\begin{bmatrix} \frac{1}{8} & 0 & 0 & 0 \\ 0 & \frac{5}{24} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{5}{12} \end{bmatrix}$$

Neste exemplo, para facilitar a visualização e identificação das operações do produto tensorial, foram usadas matrizes diagonais (estados mistos, como mostra o $\mathrm{Tr}(\rho_A^2)$, calculado abaixo)

[9]:
```
(rho_A**2).trace()
```

[9]:
$$\frac{5}{9}$$

mas poderíamos fazer o mesmo para qualquer matrix densidade, de qualquer dimensão. Mas adiante veremos outros exemplos, que exploram mais essa opções.

Isso também poderia ser feito, facilmente, usando a biblioteca **numpy**.

[10]:
```
A = np.matrix([[1/3, 0],[0, 2/3]])
B = np.matrix([[3/8, 0],[0, 5/8]])
AB = np.kron(A,B)
AB
```

[10]:
```
matrix([[0.125     , 0.        , 0.        , 0.        ],
        [0.        , 0.20833333, 0.        , 0.        ],
        [0.        , 0.        , 0.25      , 0.        ],
        [0.        , 0.        , 0.        , 0.41666667]])
```

[11]:
```
AB.trace()
```

[11]:
```
matrix([[1.]])
```

[12]:
```
print(AB**2)
(AB**2).trace()
```

```
[[0.015625   0.         0.         0.        ]
 [0.         0.04340278 0.         0.        ]
 [0.         0.         0.0625     0.        ]
 [0.         0.         0.         0.17361111]]
```

[12]:
```
matrix([[0.29513889]])
```

Em geral, nos casos que envolvem apenas cálculos numéricos e desempenho é importante, é sempre preferível usar a biblioteca **numpy**.

1.1.1 Exercício sugerido

Para praticar um pouco mais tudo que aprendemos até agora, expressse todos os operadores densidade acima usando a notação de operadore, ao invés da representação matricial.

[ ]:
```
```

1.2 Exemplo 2: operador reduzido e Traço parcial

Suponha, apenas para fixar ideia, que o sistema do **Exemplo 1** representasse duas partículas de spin-1/2, e que inicialmente nos fosse dado apenas a matriz densidade do sistema composto AB.

Essa informação permite-nos, facilmente, calcular as probilidades dos estados produto. Por exemplo...

Faça você mesmo: Como poderíamos calcular a probabilidade de observar o estado $|{\downarrow}{\uparrow}\rangle$?

[13]:
```
#Dica: a resposta é 1/4...
#Tente calcular você mesmo, antes de resolvermos juntos na aula
```

Uma sitação diferente, mas ainda simples, é perguntar, para os estados de duas partículas, a **probabilidade de observar uma medida onde a partícula B está no estado** $|{\downarrow}\rangle$. Essa medida também

pode ser respondida a pela observação direta da matriz densidade total do sistema.

DIY: Tente fazer sua previsão, antes de resolvermos...

Porém, se tivéssemos apenas a matriz total, como poderíamos responder, em geral, as perguntas abaixo?

No sistema do **Exemplo 1**, se fosse feita uma medida na partícula B para determinar a projeção do spin na direção $\hat{z}$, qual seria a probabilidade de medir o estado $|\downarrow\rangle$?

E se a medida fosse na direção $\hat{x}_B$, qual seria a probabilidade de observar o estado $|\uparrow; x\rangle$? Qual seria o valor esperado de $\langle S_x \rangle$ para cada partícula, se medidas separadamente?

DIY: Tente fazer sua previsão, antes de resolvermos.

Para perguntas desse tipo, podemos usar a matriz densidade reduzida do sistema composto (i.e., as densidades ρ_A e ρ_B), que podem ser calculada a partir da matriz densidade total usando o conceito de **Traço parcial**

$$\rho_A = \mathrm{Tr}_B\left(\rho_{AB}\right)$$

onde o símbolo $\mathrm{Tr}_B(\cdot)$ significa, em palavras, fazer o *traço* apenas sobre o espaço de Hilbert do sistema B. Na prática, o traço parcial é um mapa linear definido abaixo

1.2.1 Traço parcial

O traço parcial pode ser definido por

$$\mathrm{Tr}_B\left(\,|a\rangle\,\langle a|\otimes|b\rangle\,\langle b|\,\right) = |a\rangle\,\langle a|\;\mathrm{Tr}(\,|b\rangle\,\langle b|\,),$$

portanto,

$$\mathrm{Tr}_B\left(\,\rho_A\otimes\rho_B\,\right) = \rho_A\,\mathrm{Tr}(\,\rho_B\,) = \rho_A$$

pois $\mathrm{Tr}(\rho_A) = 1$, como qualquer operador densidade.

?!?:: Ok, mas como calcular na prática a partir da matriz total??

¡DIY!: Pense um pouco, e tente fazer sozinho(a) antes de resolvermos juntos.

[14]:
```
sys_A = TensorProduct(rho_A,I)
sys_B = TensorProduct(I,rho_B)
```

[15]:
```
sys_A
```

[15]:
$$\begin{bmatrix} \frac{1}{3} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & \frac{2}{3} \end{bmatrix}$$

[16]:
```
sys_B
```

[16]:
$$\begin{bmatrix} \frac{3}{8} & 0 & 0 & 0 \\ 0 & \frac{5}{8} & 0 & 0 \\ 0 & 0 & \frac{3}{8} & 0 \\ 0 & 0 & 0 & \frac{5}{8} \end{bmatrix}$$

[17]:
```
sys_A*sys_B
```

[17]:
$$\begin{bmatrix} \frac{1}{8} & 0 & 0 & 0 \\ 0 & \frac{5}{24} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{5}{12} \end{bmatrix}$$

Ou seja, temo o resultado (esperado):

$$(\rho_A \otimes \hat{I}) \cdot (\hat{I} \otimes \rho_B) = \rho_{AB} S_A = \rho_A \otimes \hat{I} S_B = \hat{I} \otimes \rho_B$$

Temos o produto das matrizes S_A e S_B. Note que podermos fazer

$$S_B = S_A^{-1} \cdot S_A \cdot S_B$$

[18]:
```
sys_A.inv()*sys_A*sys_B
```

[18]:
$$\begin{bmatrix} \frac{3}{8} & 0 & 0 & 0 \\ 0 & \frac{5}{8} & 0 & 0 \\ 0 & 0 & \frac{3}{8} & 0 \\ 0 & 0 & 0 & \frac{5}{8} \end{bmatrix}$$

1.2.2 Aplicações nos problemas propostos

Vamos agora aplicar esse conceito nos problemas (perguntas) proposto acima.

- Qual a probabilidade de encontrar o sistema B (segunda partícula) no estado $|\downarrow\rangle$

  O projetor do estado $|\downarrow\rangle_B$ é $\hat{P}_d = |\downarrow\rangle \langle\downarrow|$

[19]:
```
up = Matrix([[0],[1]])
down = Matrix([[0],[1]])
pd = TensorProduct(down.T,down)
op1 = TensorProduct(I,pd)
```

[20]:
```
up
```

[20]:
$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

[21]:
```
down
```

[21]:

$$\begin{bmatrix}0\\1\end{bmatrix}$$

[22]:
```
pd
```

[22]:
$$\begin{bmatrix}0 & 0\\0 & 1\end{bmatrix}$$

[23]:
```
op1
```

[23]:
$$\begin{bmatrix}0 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

[24]:
```
(rho_AB*op1)
```

[24]:
$$\begin{bmatrix}0 & 0 & 0 & 0\\0 & \frac{5}{24} & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & \frac{5}{12}\end{bmatrix}$$

[25]:
```
(rho_AB*op1).trace()
```

[25]:
$$\frac{5}{8}$$

[26]:
```
rho_AB[1,1]+rho_AB[3,3]
```

[26]:
$$\frac{5}{8}$$

- Qual o valor esperado de $\langle S_x \rangle$ para a partícula B?

  O operador $_x $ é dado por

$$S_x = \frac{\hbar}{2}\begin{pmatrix}0 & 1\\1 & 0\end{pmatrix}$$

[27]:
```
sx  # a menos de uma constante multiplicativa
```

[27]:
$$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$$

[28]:
```
op2 = TensorProduct(I,sx)
op2
```

[28]:
$$\begin{bmatrix}0 & 1 & 0 & 0\\1 & 0 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$$

Assim, podemos facilmente calcular o valor esperado $\langle S_x \rangle_B$

$$\langle S_x \rangle_B = \mathrm{Tr}\left(\rho\left(\hat{I} \otimes S_x\right)\right)$$

[29]:
```
(rho_AB*op2)
```

[29]:
$$\begin{bmatrix} 0 & \frac{1}{8} & 0 & 0 \\ \frac{5}{24} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & \frac{5}{12} & 0 \end{bmatrix}$$

[30]:
```
(rho_AB*op2).trace()
```

[30]:
0

Portanto, o $\langle S_x \rangle_B = 0$, como esperado.

- Qual a probabilidade de medir o estado $|\uparrow; x\rangle$ numa medida $\hat{x}_B$?

¡Faça você!: Pense um pouco, e tente fazer sozinho(a) essa parte...

[31]:
```
#Dica: comece expressando o estado (up)_x na base original...
```

1.3 Exemplo 3: usando o QuTiP

Nos exemplos acima utilizamos o **sympy**, e mostramos como calcular o produto tensorial (*Kronecker product*) usando o **numpy**. Claro que poderíamos também, facilmente, fazer tudo numericamente, no *numpy*, dado que temos valores numéricos apenas, mas ao invés de repetir isso aqui, eu vou mostrar como fazer essas operações usando a outra ferramente que eu apresentei para vocês: o *QuTiP (Quantum Toolbox in Python)*.

[32]:
```
import qutip as qt
r_A = qt.Qobj(A)  # utilizando as matrizes A e B já definidas acima (numpy)
r_B = qt.Qobj(B)
```

[33]:
```
r_A
```

[33]:
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.333 & 0.0 \\ 0.0 & 0.667 \end{pmatrix}$$

[34]:
```
r_B
```

[34]:
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.375 & 0.0 \\ 0.0 & 0.625 \end{pmatrix}$$

[35]:
```
rho = qt.tensor(r_A,r_B)  # produto tensorial
rho
```

[35]: Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 0.125 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.208 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.250 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.417 \end{pmatrix}$$

Neste caso, para obter o traço parcial $\mathrm{Tr}_B(\cdot)$, basta fazer

[36]:
```
rho.ptrace(1)  # traço parcial Tr_B(), para obter operador da segunda partícula
```

[36]: Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.375 & 0.0 \\ 0.0 & 0.625 \end{pmatrix}$$

Note que o mesmo não funcionaria partindo da matriz densidade completa (sem o produto tensorial).

Mostrarei isso usando a matriz **AB**, já definida acima.

[37]:
```
AB  # matriz definida acima (usando numpy)
```

[37]:
```
matrix([[0.125     , 0.        , 0.        , 0.        ],
        [0.        , 0.20833333, 0.        , 0.        ],
        [0.        , 0.        , 0.25      , 0.        ],
        [0.        , 0.        , 0.        , 0.41666667]])
```

[38]:
```
r = qt.Qobj(AB)
r
```

[38]: Quantum object: dims = [[4], [4]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 0.125 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.208 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.250 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.417 \end{pmatrix}$$

Preste atenção nas características dos dois objetos Qobj(), em particular em dims.

Eles são, efetivamente, objetos diferentes, como podemos ver tentado tomar o traço parcial

[39]:
```
r.ptrace(1)  # este erro é de propósito, para mostrar a dif. dos objetos␣
↪envolvidos.
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-39-300a3eebe6c5> in <module>
----> 1 r.ptrace(1)  # este erro é de propósito, para mostrar a dif. dos objetos
 ↪envolvidos.

~\Anaconda3\lib\site-packages\qutip\qobj.py in ptrace(self, sel, sparse)
   1353             return q.tidyup() if settings.auto_tidyup else q
   1354         else:
-> 1355             return _ptrace_dense(self, sel)
   1356
   1357     def permute(self, order):

~\Anaconda3\lib\site-packages\qutip\qobj.py in _ptrace_dense(Q, sel)
   2192         sel = np.asarray(sel)
   2193     sel = list(np.sort(sel))
-> 2194     dkeep = (rd[sel]).tolist()
   2195     qtrace = list(set(np.arange(nd)) - set(sel))
   2196     dtrace = (rd[qtrace]).tolist()

IndexError: index 1 is out of bounds for axis 0 with size 1
```

Vamos calcular os mesmos resultados usando o QuTiP...

[41]:
```
I2 = qt.qeye(2)
up = qt.Qobj(np.matrix([[1],[0]]))
down = qt.Qobj(np.matrix([[0],[1]]))
pd = qt.tensor(down.dag(),down)
op1 = qt.tensor(I2,pd)
```

[42]:
```
down
```

[42]:
Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket

$$\begin{pmatrix} 0.0 \\ 1.0 \end{pmatrix}$$

[43]:
```
down.dag()
```

[43]:
Quantum object: dims = [[1], [2]], shape = (1, 2), type = bra

$$\begin{pmatrix} 0.0 & 1.0 \end{pmatrix}$$

[44]:
```
I2
```

[44]:

Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$

[45]:
```
pd
```

[45]:
Quantum object: dims = [[1, 2], [2, 1]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$$

[46]:
```
op1
```

[46]:
Quantum object: dims = [[2, 1, 2], [2, 2, 1]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

[47]:
```
rho
```

[47]:
Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 0.125 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.208 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.250 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.417 \end{pmatrix}$$

[48]:
```
rho*op1  # este erro é de propósito, para mostrar a dif. dos objetos envolvidos.
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-48-3670142e2a67> in <module>
----> 1 rho*op1  # este erro é de propósito, para mostrar a dif. dos objetos␣
 ↪envolvidos.

~\Anaconda3\lib\site-packages\qutip\qobj.py in __mul__(self, other)
    553
    554             else:
--> 555                 raise TypeError("Incompatible Qobj shapes")
    556
    557         elif isinstance(other, np.ndarray):

TypeError: Incompatible Qobj shapes
```

Ooops! O que aconteceu aqui??

(*Dica*: objserve os tipos de objetos que forma multiplicados...)

Ok, vamos fazer da maneira certa, desta vez. Para isso usaremos a função que cria a matriz densidade diretamente a partir do vetor de estado, como mostrado abaixo

[ ]:
```
pd = qt.ket2dm(down)
pd
```

[ ]:
```
op1 = qt.tensor(I2,pd)
op1
```

[ ]:
```
rho*op1
```

Sucesso!!

Agora basta tomar o traço, como fizemos antes.

[ ]:
```
(rho*op1).tr()
```

Na dúvida, vamos verificar se é o mesmo resultado...

[ ]:
```
5/8 == 0.625
```

Qual o valor esperado de $\langle S_z \rangle$ no estado ρ (sistema total)

[ ]:
```
sx = qt.sigmax()
sy = qt.sigmay()
sz = qt.sigmaz()
sz
```

[49]:
```
szA = qt.tensor(sz,I2)
szB = qt.tensor(I2,sz)
Sz = qt.tensor(sz,sz)
Sz
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-8b962692e992> in <module>
----> 1 szA = qt.tensor(sz,I2)
      2 szB = qt.tensor(I2,sz)
      3 Sz = qt.tensor(sz,sz)
      4 Sz

~\Anaconda3\lib\site-packages\qutip\tensor.py in tensor(*args)
     98     if not all([isinstance(q, Qobj) for q in qlist]):
     99         # raise error if one of the inputs is not a quantum object
--> 100         raise TypeError("One of inputs is not a quantum object")
    101
```

```
    102     out = Qobj()

TypeError: One of inputs is not a quantum object
```

[50]:
```
rho*Sz
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-50-83dc4791fd8b> in <module>
----> 1 rho*Sz

NameError: name 'Sz' is not defined
```

[ ]:
```
(rho*Sz).tr()
```

Outra forma de obter o mesmo resultado é usando a função **qutip.expect(** *oper, state* **)**.

[ ]:
```
qt.expect(Sz,rho)
```

1.4 Exemplo 4: estado emaranhado

Vamos explorar um pouco mais o conceito de matriz densidade total e reduzida em um sistema composto de duas partículas de spin-1/2, mas agora considerando **estados puros** e superposições.

Vamos considerar os estados $|\uparrow\uparrow\rangle$ e $|\downarrow\downarrow\rangle$ e sua superposição

[51]:
```
uu = qt.tensor(up,up)       # up, up
dd = qt.tensor(down,down)   # down, down
uu
```

[51]:
Quantum object: dims = [[2, 2], [1, 1]], shape = (4, 1), type = ket

$$\begin{pmatrix} 1.0 \\ 0.0 \\ 0.0 \\ 0.0 \end{pmatrix}$$

[52]:
```
dd
```

[52]:
Quantum object: dims = [[2, 2], [1, 1]], shape = (4, 1), type = ket

$$\begin{pmatrix} 0.0 \\ 0.0 \\ 0.0 \\ 1.0 \end{pmatrix}$$

[53]:
```
# superposiçao
b1 = (uu + dd).unit()
b1
```

[53]:
Quantum object: dims = [[2, 2], [1, 1]], shape = (4, 1), type = ket

$$\begin{pmatrix} 0.707 \\ 0.0 \\ 0.0 \\ 0.707 \end{pmatrix}$$

O estado $|b_1\rangle$ é dado por

$$|b_1\rangle = \frac{|\uparrow\uparrow\rangle + |\downarrow\downarrow\rangle}{\sqrt{2}}$$

que é um **estado puro**.

Vejamos como fica a matriz densidade desse estado de duas partículas

[54]:
```
dm_b1 = qt.ket2dm(b1)
dm_b1
```

[54]:
Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 0.500 & 0.0 & 0.0 & 0.500 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.500 & 0.0 & 0.0 & 0.500 \end{pmatrix}$$

Podemos determinar agora as matrizes densidades de cada partícula, usando o traço parcial...

[55]:
```
dm_A = dm_b1.ptrace(0)
dm_A
```

[55]:
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.500 & 0.0 \\ 0.0 & 0.500 \end{pmatrix}$$

[56]:
```
dm_B = dm_b1.ptrace(1)
dm_B
```

[56]:
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.500 & 0.0 \\ 0.0 & 0.500 \end{pmatrix}$$

Esse é um resultado **surpreendente**, pois as matrizes reduzidas corresponde a estados de **máxima mistura**, como podemos confirmar com o traço de ρ_A^2

[57]:
```
dm_A**2
```

[57]:
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.250 & 0.0 \\ 0.0 & 0.250 \end{pmatrix}$$

[58]:
```
(dm_A**2).tr()
```

[58]:
```
0.4999999999999998
```

Esse resultado ocorre devido ao fenômeno de emaranhamento, presente no estado $|b_1\rangle$, que corresponde a um dos chamados estados de Bell.

[59]:
```
B1 = qt.bell_state()
B1
```

[59]:
Quantum object: dims = [[2, 2], [1, 1]], shape = (4, 1), type = ket

$$\begin{pmatrix} 0.707 \\ 0.0 \\ 0.0 \\ 0.707 \end{pmatrix}$$

[60]:
```
b1 == B1
```

[60]:
```
True
```

Tente explorar, agora, usando o que aprendeu em um estado de superposição que não seja emaranhado e veja como são as matrizes densidades de cada partícula...

Farei um exemplo, usando um estado puro sem superposição, para você aprender o caminho.

[61]:
```
dm_uu = qt.ket2dm(uu)
dm_uu
```

[61]:
Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

[62]:
```
dm_uu.ptrace(0)
```

[62]:
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix}$$

[63]:
```
dm_uu.ptrace(1)
```

[63]:
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix}$$

Note que agora a situação é bem diferente!

[64]:
```
(dm_uu.ptrace(0)**2)
```

[64]:
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix}$$

[65]:
```
(dm_uu.ptrace(0)**2).tr()
```

[65]:
```
1.0
```

Confirmando que trata-se de um estado puro!

Tente estender, agora, a ideia para outros estados de superposição e observe as matrizes densidades reduzidas de cada uma das partículas, verificando, ao final, se são estados puros ou mistos. Por exemplo, fazendo o exercício sugerido abaixo.

1.4.1 Exercício sugerido

Repita os passos acima para o estado

$$|\psi\rangle = \frac{|\uparrow\uparrow\rangle + |\downarrow\uparrow\rangle}{\sqrt{2}}$$

Como seriam as matrizes densidades total e reduzidas em outra base, por exemplo, na base $\hat{x}$ ou $\hat{y}$? Algumas delas tem temos fora da diagonal?

[ ]:

[ ]: