

# Programming\_Week\_1&2\_Symbolics\_Python

October 11, 2022

## 1 A really quick and brief intro to symbolic Mathematics in Python (For quantum information)

```
[1]: #Importing necessary modules
from sympy import *
from sympy.physics.quantum import TensorProduct,Dagger
from sympy import sqrt
from sympy.physics.quantum.qubit import Qubit, matrix_to_qubit, represent, matrix_to_density
from sympy.physics.quantum.gate import HadamardGate
from sympy.physics.quantum.qapply import qapply
from sympy.physics.quantum.gate import CNOT
from sympy.physics.quantum.gate import X,Y,Z
```

## 2 Definition of Symbols

In sympy you need to tell the system which variables are to be considered symbols, for that we use the symbols function

```
[2]: , , , =symbols('theta phi alpha beta')
```

Once symbols are defined you can do computations on them, similar to the way you do with numbers, we use the same symbols for addition, subtraction, multiplication and division, let us write an expression that combines all of them

```
[3]: var=*(+)/( *- *)
var
```

```
[3]: 
$$\frac{\alpha(\phi + \theta)}{-\alpha\beta + \alpha\phi}$$

```

We may simplify an expression for example in the denominator there is a common factor of  $\alpha$

```
[4]: var.simplify()
```

```
[4]: 
$$-\frac{\phi + \theta}{\beta - \phi}$$

```

### 3 Vectors and Matrices

Both vectors and matrices can be created from python lists, the difference in the creation of vectors and matrices is whether you use a list or a nested list. Let us construct both to get some practice. Let us start by defining the vector

$$|\Psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$$

To create a vector we simply do `Matrix([element1, element2, ....., elementn])`

```
[5]: Ψ=Matrix([ , ])
      Ψ
```

```
[5]:  $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ 
```

**Task 1:** Try creating the vector

$$|00\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and assign the name `vect1`

```
[6]: vect1=Matrix() # Complete this line
```

```
[7]: vect1==represent(Qubit('00')) # Check task 1
```

```
[7]: False
```

Matrices are defined quite similarly, all we need to do is input a nested list, for example let us create the following matrix:

$$M = \begin{pmatrix} 0 & i\alpha \\ -i\alpha & 0 \end{pmatrix}$$

```
[8]: M=Matrix([[0,I* ],[-I* ,0]]) # Complete this line
      M
```

```
[8]:  $\begin{bmatrix} 0 & i\alpha \\ -i\alpha & 0 \end{bmatrix}$ 
```

Matrix products are obtained simply by using the standard python multiplication operator, for example acting  $M$  on  $|\Psi\rangle$

```
[9]: M*Ψ
```

```
[9]:  $\begin{bmatrix} i\alpha\beta \\ -i\alpha^2 \end{bmatrix}$ 
```

In quantum the symbol  $\dagger$  stands for conjugate transpose, to find the conjugate of an object in sympy we simply use the conjugate function, while the transpose is obtained by the .T method, using  $|\Psi\rangle$  we can exemplify this

```
[10]:  $\Psi.T$ 
```

```
[10]:  $\begin{bmatrix} \alpha & \beta \end{bmatrix}$ 
```

```
[11]: conjugate( $\Psi$ )
```

```
[11]:  $\begin{bmatrix} \overline{\alpha} \\ \overline{\beta} \end{bmatrix}$ 
```

Then for  $(|\Psi\rangle)^\dagger = \langle\Psi|$  we simply combine the two operations

```
[12]: conjugate( $\Psi$ ).T
```

```
[12]:  $\begin{bmatrix} \overline{\alpha} & \overline{\beta} \end{bmatrix}$ 
```

We may now compute the inner product which is denoted as  $\langle\Psi|\Psi\rangle$

```
[13]: ((conjugate( $\Psi$ ).T)* $\Psi$ )
```

```
[13]:  $\begin{bmatrix} \alpha\overline{\alpha} + \beta\overline{\beta} \end{bmatrix}$ 
```

Due to it coming from matrix multiplication sympy returns a vector of just one element, we may obtain the result simply by indexing it

```
[14]: ((conjugate( $\Psi$ ).T)* $\Psi$ )[0]
```

```
[14]:  $\alpha\overline{\alpha} + \beta\overline{\beta}$ 
```

**Subjective advice :** I like to see  $x\bar{x}$  as  $|x|^2$  to do it you may simply substitute  $\bar{x}$  for  $\frac{|x|^2}{x}$ , though not powerful on it's own it can be helpful if incorporated in more complicated calculations

**Task 3.** A matrix is said to be unitary if  $U^\dagger U = \mathcal{I}$ , under what condition is M a unitary matrix, remember that  $\dagger$  stands for conjugate transpose, find the condition and implement it here

```
[15]: #task 3
```

**The computational basis** Remember from the lecture that the computational basis is given by:

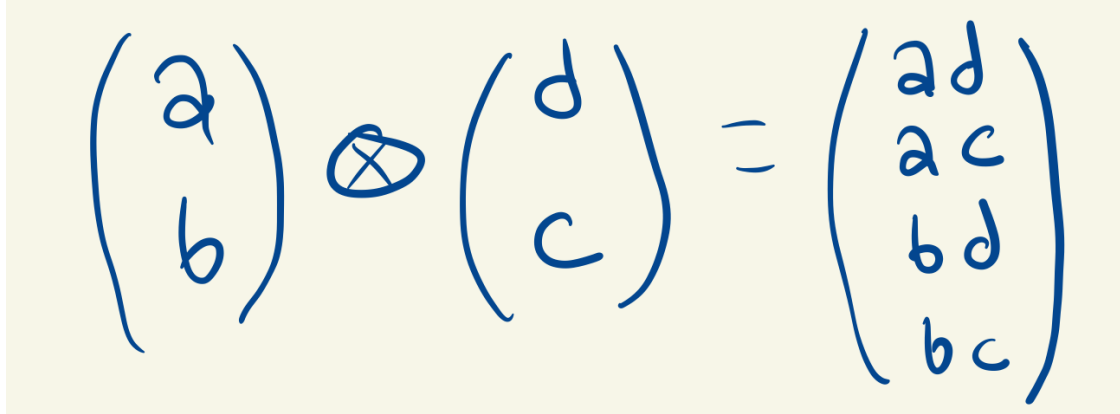
```
[16]: zero=Matrix([1,0])
zero
```

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

```
[17]: one=Matrix([1,0])
one
```

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

Now remember from the lecture that if we deal with more than one qubit we can obtain the appropriate basis from the tensor product


$$\begin{pmatrix} a \\ b \end{pmatrix} \otimes \begin{pmatrix} d \\ c \end{pmatrix} = \begin{pmatrix} ad \\ ac \\ bd \\ bc \end{pmatrix}$$

In sympy that product is done by simply calling `kroncker_product(A,B)`

[18]: `kroncker_product(zero,zero)`

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

Now the outer product can also be found using this function, we simply have to pass a bra (row vector) and a ket (column vector) instead of two column vectors. So to obtain  $|0\rangle\langle 0|$  we simply do:

[19]: `kroncker_product(zero,zero.T)`

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

For this one, we may also just use normal matrix multiplication as a shortcut

[47]: `zero*zero.T`

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

**Task 4** Define the pauli matrices, the hadamard (H) and the two dimentional identity

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

[20]: `# Complete this cell  
#z=Matrix()  
#x=  
#y=`

```
#iden=
```

**Task 5** Obtain  $H \otimes Y \otimes H \otimes Z(\theta|0000\rangle + \alpha|1111\rangle)$

**The sympy quantum module** The sympy quantum library provides shortcuts to the above operations using an implementation based on the bra-ket notation (so that the matrix products are only done when necessary) and the rest of the time it implements the transformations like we would do in pen and paper, In order to construct a state in this framework we use the computational basis, particularly we use their Qubit function

```
[21]: Qubit('0')
```

```
[21]: |0>
```

We can convert the ket notation to the matrix one by using represent

```
[22]: represent(Qubit('0'))
```

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

Conversely we can go from vector notation to bra-ket notation using the matrix\_to\_qubit function

```
[23]:  $\Psi$ 
```

```
[23]:  $\begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ 
```

```
[24]: matrix_to_qubit( $\Psi$ )
```

```
[24]:  $\alpha|0\rangle + \beta|1\rangle$ 
```

Most common Operators and gates are predefined in the sympy quantum module, for example the pauli matrices are under the names X,Y,Z respectively. The CNOT gate is under CNOT, and the Hadamard under HadamardGate, in this module the quantum gates require an input, that input is the qubit they act on, qubits are counted from right to left and the counting starts at 0 for example to write the operator X acting on the first qubit we have:

$$(\mathcal{I} \otimes X)|00\rangle = X_0|00\rangle = |01\rangle$$

```
[25]: X(0)*Qubit('10')
```

```
[25]:  $X_0|10\rangle$ 
```

Of course just printing the input in a nice notation is not actually useful, we want to get to the result of the computation, to do that we simply wrap our operations into the qapply function

```
[26]: qapply(X(0)*Qubit('00'))
```

```
[26]: |01>
```

Task 5 was a little painful in matrix notation, we probably would have achieved the answer using paper and pen faster, however using bracket notation is a single line, that is quite readable, again the idea was to compute

$$H \otimes Y \otimes H \otimes Z(\theta|0000\rangle + \alpha|1111\rangle)$$

```
[27]: state= *Qubit('0000')+ *Qubit('1111') #We first defined the superposition on
      ↪which the gates will be applied
      state
```

```
[27]:  $\alpha|1111\rangle + \theta|0000\rangle$ 
```

```
[28]: state=(HadamardGate(3)*Y(2)*HadamardGate(1)*Z(0))*state # We act the operators
      ↪on the state
      state
```

```
[28]:  $H_3Y_2H_1Z_0(\alpha|1111\rangle + \theta|0000\rangle)$ 
```

```
[29]: qapply(state) # We get the result of the computation
```

```
[29]:  $\frac{i\alpha|0001\rangle}{2} - \frac{i\alpha|0011\rangle}{2} - \frac{i\alpha|1001\rangle}{2} + \frac{i\alpha|1011\rangle}{2} + \frac{i\theta|0100\rangle}{2} + \frac{i\theta|0110\rangle}{2} + \frac{i\theta|1100\rangle}{2} + \frac{i\theta|1110\rangle}{2}$ 
```

We broke it down as a review however it could as well be done on a single line

```
[30]: qapply((HadamardGate(3)*Y(2)*HadamardGate(1)*Z(0))*( *Qubit('0000')+ *Qubit('1111'))))
```

```
[30]:  $\frac{i\alpha|0001\rangle}{2} - \frac{i\alpha|0011\rangle}{2} - \frac{i\alpha|1001\rangle}{2} + \frac{i\alpha|1011\rangle}{2} + \frac{i\theta|0100\rangle}{2} + \frac{i\theta|0110\rangle}{2} + \frac{i\theta|1100\rangle}{2} + \frac{i\theta|1110\rangle}{2}$ 
```

**Task 6** Compute the following state

$$(H_3H_2H_1H_0)(X_3Z_2X_1Z_0)(H_3H_2H_1H_0)(\theta|1010\rangle + \alpha|1011\rangle)$$

What is the probability of measuring the state  $|1111\rangle$ ?

```
[31]: # Task 6
```

In this syntax inner and outer products can be computed  $|00\rangle = |0\rangle \otimes |0\rangle$  however this is a particular weak point of this open source library to the best of my knowledge since it requires a little more coding than it should, to see why let us code this step by step, First let us get the tensor product of  $|0\rangle$  with itself

```
[32]: TensorProduct(Qubit('0'),Qubit('0'))
```

```
[32]:  $|0\rangle \otimes |0\rangle$ 
```

This is not terribly useful because this is different from  $|00\rangle$  for the computer, to get one into the other we simply transform the tensor product to vector notation and come back to bra-ket

```
[33]: TensorProduct(Qubit('0'),Qubit('0'))==Qubit('00') # Crude tensor product
```

```
[33]: False
```

```
[34]: matrix_to_qubit(represent(TensorProduct(Qubit('0'),Qubit('0')))) == Qubit('00') #_
      ↪going into vector notation and coming back
```

[34]: True

The inner product on the other hand comes quite naturally we just do a normal product and then use the qapply function, for example having  $|\alpha\rangle = \theta|0\rangle + \beta|1\rangle$  we can quickly find  $\langle\alpha|\alpha\rangle$

```
[35]: alpha = *Qubit('0') + *Qubit('1')
```

```
[36]: qapply((Dagger(alpha)*alpha))
```

[36]:  $\beta\beta^\dagger + \theta\theta^\dagger$

Unfortunately Sympy doesn't know whether our symbols are matrices or numbers or something else, so it keeps the dagger instead of using the bar for conjugate, however as long as you know what you are computing this should not be an issue.

Let us finish this part with a final task

### Task 7

Compute the inner product of the result of task 6 with itself, what does it need to fulfill to be an adequate quantum state?