Introducción a OpenKet

Javier Sánchez Villanueva
Universidad Nacional Autónoma de Honduras
Escuela de Física
rokapier@gmail.com

Resumen

Este es un manual introductorio a la librería de Python OpenKet. Esta librería es una herramienta para la manipulación de objetos en mecánica cuántica, tales como vectores en la notación de Dirac, operadores etc. Es un software libre bajo la licencia de GNU-GPL v3, y utiliza otras librerías de software libre, tal como, Sympy, Pylab y Scipy. El proyecto comenzó a finales del 2009 con Vicente Rodríguez, bajo la supervisión de Pablo Barberis-Bolstein, como parte del requisito para su graduación [4]. El Dr. Barberis-Bolstein profesor en el Instituto de Matemática Aplicada de la UNAM, quien estuvo en nuestro país para el CURCCAF en el 2011. Para el uso de OpenKet también se necesitará tener algún conocimiento en Python. OpenKet puede manipular expresiones que contengan objetos como: Kets (vectores), Bras (vectores duales), operadores, operadores adjuntos, operadores de ascenso y descenso, y puede realizar diversas operaciones como: sumar, restar, multiplicar por escalares, obtener el conjunto Hermitiano, aplicar operadores, realizar producto interior, producto exterior, producto tensorial, obtener la representación matricial de un operador, entre otras funciones [21]. OpenKet todavía se encuentra en sus primeros pasos, y por lo tanto tiene un montón de agujeros. Consecuentemente está en pleno desarollo.

Palabras clave: Python, Sympy, Pylab, Scipy, mecánica cuántica, Computación Cuántica, notación de Dirac, Kets, Bras, Operadores, producto interior, producto exterior, producto tensorial

This is an introductory manual to the Python library OpenKet, which is a tool for manipulating objects from quantum mechanics, such as vectors in the form of kets, operators, etc. It is a free software under GNU-GPL v3 license, and uses other free software libraries, such as Sympy, Numpy and Scipy. The project started at the end of 2009 with Vicente Rodríguez under the supervision of Pablo Barberis-Blostein, as part of the requirements for his graduation [4]. Dr. Barberis-Bolstein, professor from the Applied Mathematic Institute at UNAM, who was in our country for the CURCCAF in 2011. We need to have some knowledge in Python to work in OpenKet. OpenKet can manipulate expressions containing objects like: Kets (vectors), Bras (Vector dual), operators, adjoint operators, operators of ascent and descent, and can perform various operations such as addition, subtraction, multiplication by scalar, Hermitian conjugate, apply operators, perform inner product, the outer product, tensor product, get the matrix representation of an operator, and other functions [21]. OpenKet is in its early stages and has a lot of bugs. Consequently is continuously improving.

Keywords: Python, Sympy, Pylab, Scipy, Quantum Mechanic, Quantum Computation, Dirac notation, Kets, Bras, operators, inner product, outer product, tensor product.

I. Instalando Openket

Descargar Openket en la siguiente dirección `http://code.google.com/p/openket/`, descargar los archivos py y el manual en PDF, también aparece las direcciones electrónicas de Vicente Rodríguez y Pablo Barberis. Antes de utilizar OpenKet necesitamos instalar Sympy, Pylab y Scipy.

Se recomienda descargar OpenKet Brief manual desde el sitio del proyecto en google code [21]. Una vez descargado el archivo denominado openketvs0.10.py, lo guardamos en la

carpeta personal y lo reescribimos como openket.py y también nombramos el archivo corevs0.10.py como core.py.

II. Trabajando en Openket

Ahora para trabajar en OpenKet, lo podemos hacer desde la consola de Linux abriendo primero Python
En mi caso, la distribución de Linux que he utilizado es Ubuntu-12.04

```
$ python
```

importamos el archivo openket en python [4]

```
>>> from openket import *
```

En Mecánica Cuántica, los estados se representan como vectores en el espacio de Hilbert [7, 15]. Mediante la notación de Dirac estos vectores se representan como $\mid x\rangle$ y se les llama kets [3, 19, 9].
Veamos un ejemplo: escribamos el Ket de **x**, o sea escribamos $\mid x\rangle$

```
>>> x = var("x")
>>> u = Ket(x); u
| x >
```

Aquí **x** es una variable y **u** es el objeto Ket [4].

Ahora definamos el vector propio $\mid a\rangle$ del operador $\hat{A}$ con valor propio **a**, es decir $\hat{A} \mid a\rangle = a \mid a\rangle$

```
>>> a = var('a')
>>> A = Operator('A')
>>> u = Ket(a, 'A')
>>> u
|a_A >
```

El valor propio del operador es

```
>>> u.eig
a
```

El operador

```
>>> u.op
A
```

Aplicando el operador al vector

```
>>> A*u
a|a_A >
```

También podemos escribir el Bra, o sea el vector dual del vector, por ejemplo

```
>>> x = var("x")
>>> u = Bra(x); u
< x |
```

Openket asume que $\mid x\rangle$ es normalizado [4], por lo tanto si queremos hacer el producto punto $\langle 0 \mid 0\rangle$ [3]

```
>>> Bra(0)*Ket(0)
1
```

O también podemos escribirlo como

```
>>> Adj(Ket(0))*Ket(0)
1
```

El Producto exterior es [19, 20]

```
>>> Ket(0)*Bra(0)
| 0 >< 0 |
```

Ejemplo 1

$\mid \psi\rangle = 2.1 \mid 0\rangle + i \mid 1\rangle$
$\langle \phi \mid = i\langle 0 \mid -4.5\langle 1 \mid$

$\langle \phi \mid \psi\rangle = -2.4i$

$\mid \psi\rangle\langle \phi \mid = 2.1i \mid 0\rangle\langle 0 \mid -9.45 \mid 0\rangle\langle 1 \mid - \mid 1\rangle\langle 0 \mid -4.5i \mid 1\rangle\langle 1 \mid$

En OpenKet lo escribimos así

```
>>> psi = 2.1*Ket(0) + I*Ket(1)
>>> phi = I*Bra(0) - 4.5*Bra(1)
>>> phi*psi
-2.4*I
```

Multiplicando un Ket por un Bra el producto exterior es

```
>>> psi*phi
2.1*I| 0 >< 0 | - 9.45| 0 >< 1 | - | 1 >< 0 | - 4.5*I| 1 >< 1 |
```

Donde $\mathrm{I} = i = \sqrt{-1}$ [4]

III. Producto tensorial

Muchas veces es deseable expandir el espacio de Hilbert, y lo hacemos mediante un proceso llamado producto tensorial [7, 19]. El producto tensorial es una manera de combinar espacios, vectores, u operadores juntos. Suponga que $\mathcal{H}_1$ y $\mathcal{H}_2$ son espacios de Hilbert de dimenciones n y m respectivamente. Entonces el producto tensorial $\mathcal{H}_1 \otimes \mathcal{H}_2$ es un nuevo, espacio de Hilbert mas grande de dimención $n \times m$.

El producto tensorial de dos vectores $\mid \psi_1\rangle$ y $\mid \psi_2\rangle$ de espacios $\mathcal{H}_1$ y $\mathcal{H}_2$, respectivamente, es un vector en $\mathcal{H}_1 \otimes \mathcal{H}_2$, y se escribe $\mid \psi_1\rangle \otimes \mid \psi_2\rangle$. Podemos omitir el símbolo $\otimes$ y escribir $\mid \psi_1\rangle \mid \psi_2\rangle$, o simplemente $\mid \psi_1\psi_2\rangle$ [17, 19]

```
>>> psi1 = var('psi1')
>>> psi2 = var('psi2')
>>> phi1 = var('phi1')
>>> phi2 = var('phi2')
>>> Ket(psi1)*Ket(psi2)
|psi1> |psi2>
>>> (Ket(psi1) + Ket(phi1))*Ket(psi2)
|phi1> |psi2> + |psi1> |psi2>
>>> Ket(psi1)*(Ket(psi2) + Ket(phi2))
|phi2> |psi1> + |psi1> |psi2>
```

IV. Operadores escalera

Veamos los operadores escaleras [4, 3, 19, 7]

$\hat{a}^\dagger \mid n\rangle = \sqrt{n+1} \mid n+1\rangle$
$\hat{a} \mid n\rangle = \sqrt{n} \mid n-1\rangle$

```
>>> a = AnnihilationOperator(); aa = Crea-
tionOperator(); n = var("n")
>>> s = Ket(n); t = Bra(n)
>>> a*s; aa*s; t*a; t*aa
n**(1/2)|n - 1>
(n + 1)**(1/2)|n + 1>
(n + 1)**(1/2)<n + 1|
n**(1/2)<n - 1|
```

V. Representación matricial

```
>>> a, b = var('a b')
>>> X = a*Ket(0)*Bra(1) + b*Ket(1)*Bra(0)
>>> X
a|0><1| + b|1><0|
>>> Qmatrix(X)
[0, a]
[b, 0]
```

VI. Trabajando en Sage

También podemos trabajar con el sistema de álgebra computacional Sage, de manera similar como hemos trabajado en Python [21].

Sage (Software for Algebra and Geometry Experimentation) es un sistema de álgebra computacional basado en Python, creado por William Stein, profesor de la Universidad de Washington; que integra muchos otros paquetes de software libre pre-existentes (como Pari/GP, Singular, Maxima, NTL, GNU GSL, etc.), ofreciendo una interface unificada para todos ellos en Python. Sage está fundamentalmente orientado a problemas de teoría de números, álgebra y geometría, pero también integra paquetes para aplicaciones numéricas, álgebra lineal, teoría de códigos, etc.

```
$ sage

sage: load 'openket.py'
sage: v = Ket(2, 'A')
sage: v
|2_A>
sage: A = Operator('A')
sage: A*v
2|2_A>
sage: u = Ket(0)
sage: u
|0>
sage: Adj(u)
<0|
sage: Adj(u)*u
1
sage: u*Adj(u)
|0><0|
```

VII. Qubits

El bit es un concepto fundamental de la computación clásica e información clásica. La computación cuántica e información cuántica están hechos bajo un concepto similar, el bit cuántico, o Qubit [16, 11]. El sistema cuántico de interés más simple para la computación es un sistema cuántico de dos estados [10, 8]. Los estados cero y uno se pueden escribir como $\mid 0\rangle$ y $\mid 1\rangle$, respectivamente, y un qubit en general se puede escribir como la superposición de estos

dos estados [20, 18]

$| \psi\rangle = \alpha \mid 0\rangle + \beta \mid 1\rangle$ donde $\alpha, \beta \in \mathbb{C}$ y $| \alpha |^2 + | \beta |^2 = 1$.

La medición nos da un bit clásico de información – 0 o 1. La medición mas simple se encuentra en el estado fundamental, y haciendo la medición en $| \psi\rangle$ la base $| 0\rangle, | 1\rangle$ nos da 0 con probabilidad $| \alpha |^2$, y 1 con probabilidad $| \beta |^2$ [20].

Un importante aspecto del procedimiento de medición es que ésta altera el estado del qubit: el efecto de la medición es que el nuevo estado, es exactamente la salida de la medición. *I.e.*, si la salida de la medición en $| \psi\rangle = \alpha \mid 0\rangle + \beta \mid 1\rangle$ nos queda 0, entonces haciendo la medición, el qubit está en el estado $| 0\rangle$. Esto implica que no podemos obtener alguna información adicional sobre α, β repitiendo la medición [5, 20].

Mas generalmente, podemos elegir cualquier base ortogonal $| v\rangle, | w\rangle$ y hacer una medición del qubit en esa base. Para hacer esto reescribimos nuestro estado en esa base: $| \psi\rangle = \alpha' \mid v\rangle + \beta' \mid w\rangle$. El resultado es $| v\rangle$ con probabilidad $| \alpha' |^2$, y $| w\rangle$ con probabilidad $| \beta' |^2$. Si el resultado de la medición en $| \psi\rangle$ nos da $| v\rangle$, entonces como antes, el qubit está en el estado $| v\rangle$.

Ejemplo 2

¿Cuál es el resultado de medir el estado del qubit $| \psi\rangle = \alpha \mid 0\rangle + \beta \mid 1\rangle$, en una base ortogonal $| v\rangle$, $| w\rangle$, donde $| v\rangle = a \mid 0\rangle + b \mid 1\rangle$ y $| w\rangle = b^* \mid 0\rangle - a^* \mid 1\rangle$?. Se puede ver que $| v\rangle$ y $| w\rangle$ son ortogonales, o sea que $\langle w \mid v\rangle = 0$.

$\langle w \mid v\rangle = (b^* \mid 0\rangle - a^* \mid 1\rangle)^\dagger (a \mid 0\rangle + b \mid 1\rangle)$
$\langle w \mid v\rangle = (b\langle 0 \mid - a\langle 1 \mid)(a \mid 0\rangle + b \mid 1\rangle)$
$\langle w \mid v\rangle = ba\langle 0 \mid 0\rangle - a^2\langle 1 \mid 0\rangle + b^2\langle 0 \mid 1\rangle - ab\langle 1 \mid 1\rangle$
$\langle w \mid v\rangle = ba - 0 + 0 - ab$
$\langle w \mid v\rangle = 0$

Aquí hemos utilizado el hecho de que $\langle i \mid j\rangle = \delta_{ij}$.

Ahora, la probabilidad de medir el estado $| \psi\rangle$ y obtener $| v\rangle$ como un resultado es,

$P_\psi(v) = | \langle v \mid \psi\rangle |^2$
$P_\psi(v) = | (a^*\langle 0 \mid + b^*\langle 1 \mid)(\alpha \mid 0\rangle + \beta \mid 1\rangle) |^2$
$P_\psi(v) = | a^*\alpha + b^*\beta |^2$

De manera similar,

$P_\psi(w) = | \langle w \mid \psi\rangle |^2$
$P_\psi(w) = | (b\langle 0 \mid - a\langle 1 \mid)(\alpha \mid 0\rangle + \beta \mid 1\rangle) |^2$
$P_\psi(w) = | b\alpha - a\beta |^2$

Traduciendolo a lenguaje OpenKet

```
    $ python

>>> from openket import *
>>> alpha = var('alpha')
>>> beta = var('beta')
>>> a = var('a')
>>> b = var('b')
>>> psi = alpha*Ket(0) + beta*Ket(1)
>>> v = a*Ket(0) + b*Ket(1)
>>> w = Adj(b)*Ket(0) - Adj(a)*Ket(1)
>>> Adj(w)*v
0
>>> Ppsiv = (Adj(psi)*v)*(Adj(v)*psi)
>>> Ppsiv
(a*conjugate(alpha) + b*conjugate(beta))
*(alpha*conjugate(a) + beta*conjugate(b))
>>> Ppsiw = (Adj(psi)*w)*(Adj(w)*psi)
>>> Ppsiw
(-a*beta + alpha*b)*(-conjugate(a)*conjugate
(beta) + conjugate(alpha)*conjugate(b))
```

Donde Ppsiv = $P_\psi(v)$ y Ppsiw = $P_\psi(w)$

Recordemos que

$| \langle v \mid \psi\rangle |^2 = \langle \psi \mid v\rangle\langle v \mid \psi\rangle = \langle v \mid \psi\rangle^\dagger \langle v \mid \psi\rangle$

Por lo tanto

$P_\psi(v) = (\alpha^*\langle 0 \mid + \beta^*\langle 1 \mid)(a \mid 0\rangle + b \mid 1\rangle)(a^*\langle 0 \mid + b^*\langle 1 \mid)(\alpha \mid 0\rangle + \beta \mid 1\rangle)$

$P_\psi(v) = (a\alpha^* + b\beta^*)(a^*\alpha + b^*\beta)$

$P_\psi(v) = (a^*\alpha + b^*\beta)^\dagger (a^*\alpha + b^*\beta)$

$P_\psi(v) = | a^*\alpha + b^*\beta |^2$

I. Transformaciones unitarias

Los estados cuánticos se pueden manipular exactamente de dos maneras: transformaciones uni-

tarias y mediciones. Las transformaciones unitarias se pueden describir como operaciones matriciales sobre vectores (el vector siendo un estado cuántico) [20]:

$$| \psi\rangle \mapsto U \cdot | \psi\rangle$$

$| \psi\rangle$ es un estado cuántico de n qubits (un vector de dimensión 2^n), y U una matríz unitaria $\in \mathbb{C}^{2^n \times 2^n}$.
Una matríz es unitaria, si satisface la ecuación

$$U^\dagger \cdot U = I$$

II. Puertas cuánticas

En mecánica cuántica, nos referimos a un operador unitario U actuando sobre un qubit como una puerta cuántica [5, 13]

Puerta Hadamard.

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

La puerta Hadamard es una de las mas importantes puertas cuánticas. Note que $H^\dagger = H$, ya que H es real y simétrica, y $H^2 = I$ [10]
Apliando la puerta Hadamard a algún estado [17]

$H | 0\rangle = \frac{1}{\sqrt{2}}(| 0\rangle + | 1\rangle)$
$H | 1\rangle = \frac{1}{\sqrt{2}}(| 0\rangle - | 1\rangle)$

Una propiedad útil es que es auto-inversa, $H = H^{-1}$, y así

$H(\frac{1}{\sqrt{2}}(| 0\rangle + | 1\rangle)) =| 0\rangle$
$H(\frac{1}{\sqrt{2}}(| 0\rangle - | 1\rangle)) =| 1\rangle$

Definamos H en notación de Dirac y desarollemos en Python

$H = \frac{1}{\sqrt{2}}(| 0\rangle\langle 0 | + | 0\rangle\langle 1 | + | 1\rangle\langle 0 | - | 1\rangle\langle 1 |$
)

```
>>> H = (1/sqrt(2))*(Ket(0)*Bra(0) +
Ket(0)*Bra(1) + Ket(1)*Bra(0) - Ket(1)*Bra(1))
>>> Qmatrix(H)
[0.707106781186547, 0.707106781186547]
[0.707106781186547, -0.707106781186547]
>>> H* Ket(0)
0.707106781186547 | 0 > +0.707106781186547 |
1 >
>>> H* Ket(1)
0.707106781186547 | 0 > −0.707106781186547 |
1 >
>>> H*(1/sqrt(2)*(Ket(0) + Ket(1)))
| 0 >
>>> H*(1/sqrt(2)*(Ket(0) - Ket(1)))
| 1 >
```

Puerta NOT. Esta intercambia un bit de 0 a 1 y viceversa [10, 17]

$$NOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$NOT | 0\rangle =| 1\rangle$
$NOT | 1\rangle =| 0\rangle$

$NOT =| 0\rangle\langle 1 | + | 1\rangle\langle 0 |$

```
>>> NOT = Ket(0)*Bra(1) + Ket(1)*Bra(0)
>>> Qmatrix(NOT)
[0, 1]
[1, 0]
>>> NOT*Ket(0)
| 1 >
>>> NOT*Ket(1)
| 0 >
```

Phase Flip [17].

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

La puerta Phase Flip es una puerta NOT actuando sobre la base $| +\rangle = \frac{1}{\sqrt{2}}(| 0\rangle + | 1\rangle)$, $| -\rangle = \frac{1}{\sqrt{2}}(| 0\rangle - | 1\rangle)$

$Z | +\rangle =| -\rangle$
$Z | -\rangle =| +\rangle$

$Z =| 0\rangle\langle 0 | - | 1\rangle\langle 1 |$

```
>>> Z = Ket(0)*Bra(0) - Ket(1)*Bra(1)
>>> Qmatrix(Z)
[1, 0]
[0, -1]
>>> mas = (1/sqrt(2))*(Ket(0) + Ket(1))
>>> Z*mas
0.707106781186547 | 0 > −0.707106781186547 |
1 >
>>> menos = (1/sqrt(2))*(Ket(0) - Ket(1))
>>> Z*menos
0.707106781186547 | 0 > +0.707106781186547 |
1 >
```

III. Dos qubits

Ahora veamos un sistema de dos qubits. Supongamos que tenemos dos qubits.Si estos fueran bits clásicos, entonces habrian cuatro posibles estados, $00, 01, 10, 11$. Respectiamente, un sistema de dos qubits tiene cuatro estados bases computacionales $\mid 00\rangle$, $\mid 01\rangle$, $\mid 10\rangle$, $\mid 11\rangle$. Por el principio de superposición podemos escribir el estado cuántico de los dos qubits como: [11, 12]

$$\mid \psi\rangle = \alpha_{00} \mid 00\rangle + \alpha_{01} \mid 01\rangle + \alpha_{10} \mid 10\rangle + \alpha_{11} \mid 11\rangle$$

donde $\alpha_{ij} \in \mathbb{C}, \Sigma_{ij} \mid \alpha_{ij} \mid^2 = 1$. Por supuesto, ésta es solo la notación de Dirac para el vector unitario en $\mathbb{C}^4$:

$$\begin{pmatrix} \alpha_{00} \\ \alpha_{01} \\ \alpha_{10} \\ \alpha_{11} \end{pmatrix}$$

Para escribir el vector $\mid ab\rangle$, podemos hacerlo de la siguiente manera.
escribamos el siguiente archivo en Python

```
from openket import *

d = ['0', '1']

def ket(x, y):      # Definición de | ab⟩
    xy = var(d[x] + d[y])
    return Ket(xy)

def bra(x, y):      # Definición de ⟨ab |
    xy = var(d[x] + d[y])
    return Bra(xy)
```

Lo guardamos como archivo.py, en la carpeta personal, por ejemplo KetOI.py. Para trabajar en él, hacemos lo siguiente [4, 14, 1, 2, 6]

```
>>> from KetOI import *
>>> ket(0, 0)
| 00 >
>>> ket(0, 1)
| 01 >
>>> ket(1, 0)
| 10 >
>>> ket(1, 1)
| 11 >
```

y escribamos $\mid \psi\rangle$ en Python

```
$ python

>>> from KetOI import *
>>> a = var('alpha00')
>>> b = var('alpha01')
>>> c = var('alpha10')
>>> d = var('alpha11')
>>> psi = a*ket(0, 0) + b*ket(0, 1) + c*ket(1,
0) + d*ket(1, 1)
alpha00| 00 > + alpha01| 01 > + alpha10|
10 > + alpha11| 11 >
```

Ejemplo 3

Veamos, ¿Cuál es la probabilidad de que el primer bit sea igual a cero?, esto es ¿Pr{1st bit = 0}?
Pr{1st bit= 0} = Pr{00} + Pr{01} = $\langle$ psi $\mid 00\rangle\langle 00 \mid$ psi $\rangle$ + $\langle$ psi $\mid 01\rangle\langle 01 \mid$ psi $\rangle$ = $\alpha_{00}\alpha_{00}^* + \alpha_{01}\alpha_{01}^* = \mid \alpha_{00} \mid^2 + \mid \alpha_{01} \mid^2$

```
>>> P00 = (bra(0, 0)*psi)*(Adj(psi)*ket(0,
0))
>>> P00
alpha00*conjugate(alpha00)
>>> P01 = (bra(0, 1)*psi)*(Adj(psi)*ket(0,
1))
>>> P01
alpha01*conjugate(alpha01)
>>> P00 + P01
alpha00*conjugate(alpha00) +
alpha01*conjugate(alpha01)
```

Puerta Controlled NOT

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

El primer bit de la puerta CNOT se conoce como çontrol bit", y el segundo como "target bit". Esto porque el control bit no cambia, mientras que el target bit cambia si y solo si el control bit es 1. [17]

Estados de Bell [17]

Podemos generar los estados de Bell $\mid \Phi^+\rangle = \frac{1}{\sqrt{2}}(\mid 00\rangle + \mid 11\rangle)$ aplicando las puertas de Hadamard y CNOT.
El primer qubit atraviesa una puerta de Hadamard y luego ambos qubits se entrelazan por una puerta CNOT.

Si la entrada del sistema es $\mid 0\rangle \otimes \mid 0\rangle$, entonces la puerta de Hadamar cambia el estado a $\frac{1}{\sqrt{2}}(\mid 0\rangle + \mid 1\rangle) \otimes \mid 0\rangle = \frac{1}{\sqrt{2}} \mid 00\rangle + \frac{1}{\sqrt{2}} \mid 10\rangle$ y después de la puerta CNOT el estado queda como $\frac{1}{\sqrt{2}}(\mid 00\rangle + \mid 11\rangle)$ el estado de Bell $\mid \Phi^{+}\rangle$.

Escribamos CNOT de la siguiente manera
CNOT = $\mid 00\rangle\langle 00 \mid + \mid 01\rangle\langle 01 \mid + \mid 10\rangle\langle 11 \mid + \mid 11\rangle\langle 10 \mid$

aplicando al estado $\frac{1}{\sqrt{2}}(\mid 00\rangle + \mid 10\rangle)$ se puede ver facilmente que esto nos da $\frac{1}{\sqrt{2}}(\mid 00\rangle + \mid 11\rangle)$

```
$ python

>>> from KetOI import *
>>> CNOT = ket(0, 0)*bra(0, 0) + ket(0, 1)*bra(0, 1) + ket(1, 0)*ket(1, 1) + ket(1, 1)*bra(1, 0); CNOT
| 00 >< 00 | + | 01 >< 01 | + | 10 >< 11 | + | 11 >< 10 |
>>> Phi = (1/sqrt(2))*(ket(0, 0) + ket(1, 0)); Phi
0.707106781186547 | 00 > +0.707106781186547 | 10 >
>>> CNOT*Phi
0.707106781186547 | 00 > +0.707106781186547 | 11 >
```

VIII. Conclusiones

La notación de Dirac es una herramienta simple y compacta de definir los estados en mecánica cuántica. OpenKet podría resultar de gran utilidad a la hora de querer representar la notación de Dirac en un computador. Como dicen los autores del proyecto [4], OpenKet todavía se encuentra en sus primeros pasos y se necesita seguir trabajando en él para obtener mejores resultados. Por ejemplo, en OpenKet aún no podemos escribir el estado $\mid xy\rangle$, para este artículo hemos hecho un pequeño trabajo para representar dicho estado, pero debemos generalizarlo mas aún. Esto parece ser una buena motivación para quienes gustan de la mecánica cuántica y la programación.

Referencias

[1] Andrés Marzal, David Llorens e Isabel Graia. *Aprender a programar con Python: Una experiencia docente*. Universitat Jaume I.

[2] Andrés Marzal, Isabel Gracia. *Introducción a la programación con Python*. Departamento de Lenguajes y sistemas informáticos, Universitat Jaume I, 2003.

[3] Gary E. Bowman. *Essential Quantum Mechanics*. Oxford University Press, Department of Physics and Astronomy, Northern Arizona University, 2008.

[4] Víctor S. C. Canela. Openket brief manual.

[5] Ronald de Wolf. Quantum computing: Lecture notes.

[6] Raúl González Duque. *Python para todos*.

[7] Edward G. Harris. *A Pedestrian Approach to Quantum Field Theory*. John Wiley and sons, University of Tennesee, 1972.

[8] Tony Hey. Quantum computing: An introduction.

[9] Hidayath Ansari, Aditya Paramenswaran, Lakulish Antani, Bhaskara Aditya, Ankur Taly and Luv Kumar. Quantum cryptography and quantum computation. May 4, 2009.

[10] Kirk T. McDonald. Physics of quantum computation. July 28, 2011.

[11] Michael A. Nielsen, Issac L. Chuang. Quantum computation and quantum information. 2000.

[12] Michael A. Nielsen, Issac L. Chuang. Quantum computation and quantum information, 10th aniverasry. 2010.

[13] Mark Oskin. Quantum computing-lectures notes.

[14] Paul Grries, Jennifer Campbell, Jason Montojo. *Practical Programing, An Introduction to Computer Science Using Python 3*. Second edition, September, 2013.

[15] Jyh Ying Peng. Quantum computation lecture notes, based on lecture notes by john preskill 1998. 2003.

[16] Lukac Perkowski. Introduction and quantum mechanics.

[17] Phillip Kaye, Raymond Laflamme, Michel Mosca. *An Introduction to Quantum Computing*. Oxford University Press, Oxford New York, 2007.

[18] Rob Pike. *An Introduction to Quantum Computation and Quantum Comunication*. June 23, 2000.

[19] R. Shankar. *Principles of Quantum Mechanics*. Yale University, New Haven, Connecticut, second edition, 1994.

[20] Christian Steinrücken. Quantum computation on non-quantum computers, computer science tripos part ii. July 2004.

[21] Vicente Rodríguez Gómez (FC, UNAM). Software libre para Óptica cuántica y computación cuántica.