

# Desarrollo e Implementación del Algoritmo HHL

JAVIER GÓMEZ ARRIBAS

29 de noviembre de 2023

## Índice

|   |           |
|---|-----------|
| <b>1. Introducción.</b>                                       | <b>2</b>  |
| <b>2. Preparación para el Algoritmo HHL.</b>                  | <b>3</b>  |
| 2.1. Autovalores y Autovectores de $A$ . . . . .              | 3         |
| 2.2. Descomposición de $A$ . . . . .                          | 3         |
| 2.3. Descomposición de $ b\rangle$ . . . . .                  | 4         |
| 2.4. Descomposición de $A^{-1}$ . . . . .                     | 4         |
| 2.5. Despejando $ x\rangle$ . . . . .                         | 4         |
| <b>3. Quantum Phase Estimation(QPE)</b>                       | <b>4</b>  |
| 3.1. Codificación del circuito de la QPE. . . . .             | 5         |
| 3.1.1. Aplicación de la QFT Inversa. . . . .                  | 6         |
| <b>4. Desarrollo y Codificación del Algoritmo HHL</b>         | <b>7</b>  |
| 4.1. Carga de $ b\rangle$ . . . . .                           | 7         |
| 4.2. Aplicamos la Quantum Phase Estimation. . . . .           | 7         |
| 4.3. Operador Inversión. . . . .                              | 8         |
| 4.4. Rotación condicionada del autovalor . . . . .            | 8         |
| 4.5. Post-Selección. . . . .                                  | 9         |
| 4.6. Aplicamos la inversa de los procesos anteriores. . . . . | 9         |
| <b>5. Ejemplo del algoritmo HHL en Quiskit</b>                | <b>10</b> |
| <b>6. Conclusiones del algoritmo HHL.</b>                     | <b>10</b> |

## 1. Introducción.

Todos recordamos del colegio o de algunas asignaturas de matemáticas de la carrera qué era un sistema de ecuaciones lineales y su forma.

$$\begin{cases} x + 5y = 5 \\ 3x - 5y = 3 \end{cases} \quad \begin{cases} y = -2x + 1 \\ 4x + 2y = 3 \end{cases}$$

$$\begin{cases} 2y - 3x = 1 \\ -4y + 6x = -2 \end{cases} \quad \begin{cases} 6x - 5y = -3 \\ 3x + 2y = 12 \end{cases}$$

Este algoritmo resuelve sistemas de ecuaciones lineales bajo ciertas condiciones. Se ha demostrado que el algoritmo HHL es mucho más eficiente que cualquier algoritmo clásico pero bajo una serie de restricciones en el procedimiento. Este algoritmo ha sido desarrollado por Aram Harrow(H), Avinatan Hassidim(H) y Seth Lloyd(L) y presentado en 2009, pero no fue implementado hasta 2013 y finalmente se demostró su funcionamiento y su eficiencia en 2018. Este algoritmo trata de dar una solución más rápida respecto utilizando computadores cuánticos respecto a los clásicos. Debemos comprender que para resolver estos sistemas de ecuaciones debemos transformar la forma que hemos visto anteriormente a la siguiente:

$$A\vec{x} = \vec{b}$$

Ser consciente de las restricciones de este algoritmo es de vital importancia para comprenderlo.  $A$  se trata de una matriz invertible y hermitiana, esta última característica quiere decir que se trata de una matriz que coincide con su matriz transpuesta conjugada. Además  $\vec{b}$  se trata de un vector unitario. Vamos a tratar esta fórmula con el lenguaje correspondiente:

$$A|x\rangle = |b\rangle$$

## 2. Preparación para el Algoritmo HHL.

Previamente a realizar el desarrollo matemático del algoritmo debemos saber que para sacar  $|x\rangle$  debemos realizar una descomposición muy habitual en este tipo de algoritmos. Luego debemos sustituir y despejar. Como comentaba en la introducción partiremos de la siguiente igualdad:

$$A|x\rangle = |b\rangle$$

### 2.1. Autovalores y Autovectores de $A$ .

Para comprender mejor el desarrollo del algoritmo es importante saber primero qué son los autovalores y los autovectores. El autovector de una matriz  $A$  es aquel que multiplicado por  $A$  nos resulta un autovalor de  $A$  por el mismo autovector.

$$A\vec{v} = \lambda\vec{v}$$

$A$ : matriz.  $\vec{v}$ : autovector.  $\lambda$ : autovalor.

Ejemplo:

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

### 2.2. Descomposición de $A$ .

Si  $A$  es hermítica podemos realizar las siguientes descomposiciones. La fórmula se traduce en la suma de los autovalores por el producto interno de los autovectores. Los autovectores forman una base ortonormal en el espacio en el que trabajamos, es decir, podemos descomponer cualquier estado por una combinación lineal de los autovectores, es decir, podemos descomponer  $A$  como:

$$A = \sum_i \lambda_i |u_i\rangle \langle u_i|$$

### 2.3. Descomposición de $|b\rangle$ .

Dada la misma regla establecida en el apartado anterior,  $|b\rangle$  lo podemos descomponer de la siguiente forma:

$$|b\rangle = \sum_i b_i |u_i\rangle$$

### 2.4. Descomposición de $A^{-1}$ .

Como  $A$  se trata de una matriz totalmente invertible podemos crear una descomposición de  $A^{-1}$ . a partir de la descomposición de  $A$  anterior:

$$A^{-1} = \sum_i \frac{1}{\lambda_i} |u_i\rangle \langle u_i|$$

### 2.5. Despejando $|x\rangle$ .

Por tanto, si queremos despejar  $|x\rangle$  en la fórmula original  $A|x\rangle = |b\rangle$ , debemos aplicar las igualdades anteriores quedando así la fórmula  $|x\rangle = A^{-1}|b\rangle$  y en consecuencia la siguiente sentencia:

$$|x\rangle = \sum_i \frac{b_i}{\lambda_i} |u_i\rangle$$

## 3. Quantum Phase Estimation(QPE)

Antes de desarrollar el algoritmo debemos saber qué es la Quantum Phase Estimation(QPE). Se trata de una subrutina muy utilizada en los algoritmos cuánticos junto con la QFT. Simplemente se trata de un circuito que trata de estimar el valor de la fase que estamos añadiendo al estado. El QPE es una analogía cuántica del cálculo de autovalores y autovectores de una matriz. La QPE resuelve la siguiente igualdad:

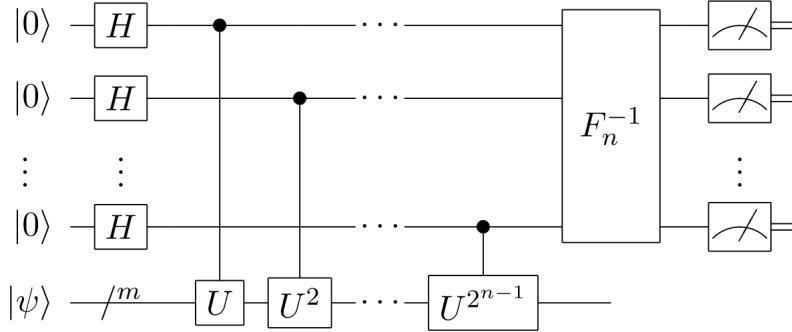
$$U|\psi\rangle = \lambda|\psi\rangle$$

Dada esta igualdad definimos  $U$  como una matriz unitaria, nuestra  $\phi$  es la fase del número complejo. La QPE resuelve el valor de  $\lambda$ . Como trabajamos con matrices unitarias( $U$ ) y gracias a la existencia de las QFT, sabemos que:

$$\lambda = e^{2\pi i\theta}$$

### 3.1. Codificación del circuito de la QPE.

Por un lado tenemos los qubits de estimación, los cuales sometemos a unas puertas Hadamard( $H$ ) para entrelazarlos. Por otro lado va a estar nuestro qubit objetivo, el cual estará sometido a serie de puertas  $U$  controladas por cada qubit de estimación. Tras esto aplicaremos la QFT inversa en los qubits de estimación. Con esto obtendremos una aproximación del  $\theta$  que buscamos. De hecho, cuantos más qubits de estimación añadamos al circuito, nos aproximaremos con más precisión a ese valor aproximado a  $\theta$ .



Para entender este circuito lo reflejaremos matemáticamente. En primera instancia debemos aplicar las puertas Hadamard a los  $n$  qubits de estimación:

$$|\psi\rangle|0\rangle^n \xrightarrow{H} |\psi\rangle \left( \frac{|0\rangle + |1\rangle}{\sqrt{2}} \right)^n$$

Sabiendo que  $\lambda = e^{2\pi i\theta}$  visto en el punto anterior, podemos sustituir en la fórmula que resuelve la QPE, quedando de esta forma:

$$U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$$

Y dependiendo un número  $k$  de puertas  $U$  y qubits de estimación nos queda:

$$U|\psi\rangle^k = e^{2\pi i\theta k}|\psi\rangle$$

Con esto podemos sacar factor común  $2^n$ :

$$\frac{1}{\sqrt{2^n}}|\psi\rangle ((|0\rangle + |1\rangle) (|0\rangle + |1\rangle) \dots)$$

Con los últimos desarrollos llegamos estamos aplicando las  $U$ :

$$\frac{1}{\sqrt{2^n}}|\psi\rangle \left( (|0\rangle + e^{2\pi i\theta 2^{n-1}}|1\rangle) (|0\rangle + e^{2\pi i\theta 2^{n-2}}|1\rangle) \dots (|0\rangle + e^{2\pi i\theta}|1\rangle) \right)$$

Por lo que nos queda en resumidas cuentas

$$\frac{1}{\sqrt{2^n}}|\psi\rangle \sum_{k=0}^{2^n-1} e^{2\pi i\theta k}|k\rangle$$

Este sería el estado de nuestro circuito antes de aplicar la QFT inversa.

### 3.1.1. Aplicación de la QFT Inversa.

Sabemos que la QFT mandaba el estado  $|\psi\rangle$  en el siguiente estado:

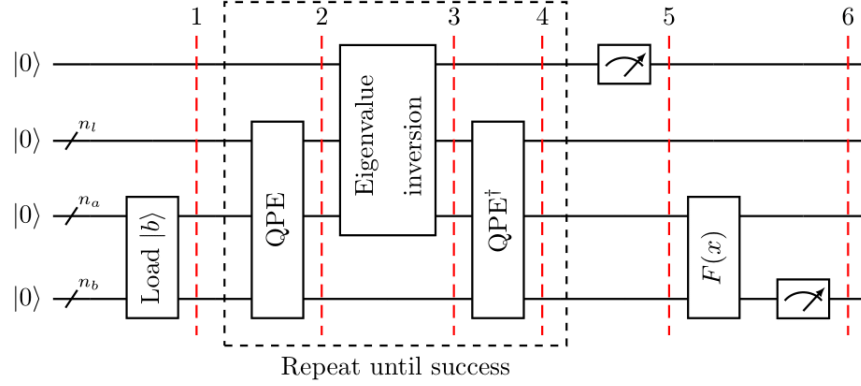
$$QFT|x\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi i x k}{2^n}}|k\rangle$$

Si sustituimos  $|x\rangle$  por  $2^n\theta$ :

$$QFT|2^n\theta\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi i 2^n \theta k}{2^n}}|k\rangle$$

Realizando la inversa de esta operación obtendremos  $\theta$ , que es la fase que queremos obtener.

## 4. Desarrollo y Codificación del Algoritmo HHL



Para explicar el circuito del algoritmo HHL vamos a distinguirlo en una serie de pasos: cargar  $|b\rangle$ , QFT, inversión del estado, QFT inversa y mediciones.

### 4.1. Carga de $|b\rangle$ .

$$|b\rangle = \sum_i b_i |u_i\rangle$$

### 4.2. Aplicamos la Quantum Phase Estimation.

Esta es la puerta por la que pasan los qubits para poder estimar la fase:

$$U = e^{iAt} := \sum_j e^{i\lambda_j t} |u_j\rangle \langle u_j|$$

Dados los autovalores y los autovectores de A:

$$\{\lambda_1, \dots, \lambda_N\} \rightarrow \{e^{i\lambda_1 t}, \dots, e^{i\lambda_N t}\}$$

$$\{|u_1\rangle, \dots, |u_N\rangle\}$$

Podemos ver en resumidas cuentas que el estado de  $|x\rangle$  es:

$$\sum_j b_j |u_j\rangle |\lambda_j\rangle$$

### 4.3. Operador Inversión.

Ya hemos visto un poco de qué va la QPE y lo importante que es para nuestro circuito. Pero aún debemos saber qué es y cómo funciona el operador inversión que hemos mencionado. Matemáticamente hemos distinguido la inversión de nuestro estado de la rotación, pero en qiskit todo el paso de invertir-rotar-desinvertir, se hace en un mismo módulo. Este operador no se suele programar de manera eficiente y tampoco lo haremos nosotros. Se trata de una puerta que se representa así:

$$Ry(2\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

Si sometemos un qubit a esta puerta, provocaremos una rotación del estado del qubit en función de lo que digamos que valga  $\theta$ . Decimos que no es un operador eficiente porque utiliza  $n(2^n)$  puertas Toffoli y es muy costoso.

Si queremos programar de forma eficiente este operador podemos hacer un circuito de  $n$  qubits, y someter cada uno a una puerta de control, de tal manera que, si el  $q_0$  está a 1 haremos una rotación de 1, si el  $q_1$  está a 1, rotamos a un medio el estado, si el  $q_2$  está a 1, haremos una rotación de un cuarto, y así sucesivamente. El problema de esta optimización es que con las inversas de los números no podemos operar correctamente.

### 4.4. Rotación condicionada del autovalor

De esta forma conseguiremos rotar el qubit auxiliar según el autovalor. Se utiliza una constante de normalización  $C$  que hace que sea posible el cálculo del estado.

$$\sum_{j=0}^{N-1} b_j \left| \frac{1}{\lambda_j} \right\rangle |u_j\rangle \left( \sqrt{1 - \frac{C^2}{\lambda_j^2}} |0\rangle + \frac{C}{\lambda_j} |1\rangle \right)$$



#### 4.5. Post-Selección.

La Post-Selección es un proceso que atraviesan ciertos algoritmos cuánticos, se trata de precesar el circuito de tal manera que si el estado del qubit auxiliar es 1, seguimos desarrollando, pero si da 0 reiniciamos el sistema.

$$|x\rangle = K \sum_{j=0}^{N-1} \frac{b_j}{\lambda_j} |u_i\rangle \left| \frac{1}{\lambda_j} \right\rangle$$

#### 4.6. Aplicamos la inversa de los procesos anteriores.

Para corregir y dar forma al resultado debemos invertir nuestros pasos una vez hemos completado los procesos anteriores. Es decir, rotamos a la inversa y aplicamos la QFT inversa. Así es como queda nuestro estado:

$$|x\rangle = K \sum_{j=0}^{N-1} \frac{b_j}{\lambda_j} |u_i\rangle$$

Repasando lo visto, hemos cargado  $|b\rangle$ , utilizamos Quantum Phase Estimation con  $|b\rangle$  y unos qubits auxiliares para codificar los autovalores, luego hemos aplicado la inversión para obtener 1 partido por el autovalor, luego aplicamos una rotación condicionada por el autovalor y finalmente limpiamos los qubits operando la inversión y la QPE inversas. De esta forma los qubits de estimación vuelven a su estado y obtenemos nuestro resultado  $|x\rangle$ .

## 5. Ejemplo del algoritmo HHL en Quiskit

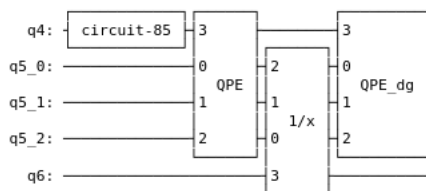
```
In [6]: import numpy as np
        from qiskit.algorithms.linear_solvers.numpy_linear_solver import NumPyLinearSolver
        from qiskit.algorithms.linear_solvers.hhl import HHL
        matrix = np.array([[1, -1/3], [-1/3, 1]])
        vector = np.array([1, 0])
        solucion_ingenua = HHL().solve(matrix, vector)
        solucion_clasica = NumPyLinearSolver().solve(matrix, vector / np.linalg.norm(vector))

In [7]: print('classical state:', solucion_clasica.state)

classical state: [1.125 0.375]

In [9]: print('naive state:')
        print(naive_hhl_solution.state)

naive state:
q4: circuit-85
q5_0:
q5_1:
q5_2:
q6:
```



```
In [11]: print('classical Euclidean norm:', solucion_clasica.euclidean_norm)
        print('naive Euclidean norm:', solucion_ingenua.euclidean_norm)

classical Euclidean norm: 1.1858541225631423
naive Euclidean norm: 1.1858541225631376
```

Esta codificación trata los componentes del algoritmo HHL como cajas negras. Vemos que las funciones *HHL.solve()* y *NumPyLinearSolver()* son las que resuelven el problema.

## 6. Conclusiones del algoritmo HHL.

Este algoritmo proporciona un método exponencialmente más rápido para estimar características de la solución de un conjunto de ecuaciones lineales, que es un problema omnipresente en la ciencia y la ingeniería, tanto por sí solo como como una subrutina en problemas más complejos. Al igual que muchos algoritmos cuánticos, el HHL es capaz de superar con creces la eficacia de cualquier otro algoritmo clásico, pero debemos saber que sólo para una serie de sistemas limitada.