Memoria de prácticas: CITMAga

Nicolás Fernández Otero

19 de diciembre de 2024

Índice

1. Objetivo de las prácticas

Las prácticas fueron realizadas en el Centro de Investigación y Tecnología Matemática de Galicia (CITMAga), y el principal objetivo fue el estudio de la viabilidad de la computación cuántica en la resolución de sistemas lineales con aplicaciones físicas. Además, uno de los objetivos principales de las prácticas fue la ejecución del algoritmo de resolución de sistemas lineales (denominado HHL) sobre la máquina real del CESGA, denominada Qmio. Como yo ya tenía base sobre el funcionamiento de la computación cuántica y de los distintos algoritmos cuánticos (ejemplo de trabajo anterior), pude comenzar las prácticas repasando los distintos algoritmos necesarios para el HHL, en vez de comenzar desde cero.

2. Justificación de la tarea realizada

La resolución de sistemas lineales es una tarea que aparece como paso intermedio de una gran parte de problemas, como en la resolución de ecuaciones diferenciales en algún proceso físico, en el entrenamiento de distintos modelos de regresión, en el mundo de la computación gráfica, etc. Estos problemas pueden ser planteados de forma matricial de la manera:

$$Ax = b, \quad x, b \in \mathbb{C}^n, A \in \mathcal{M}_{n\times n}(\mathbb{C}).$$

Para resolver estos problemas existen diversos métodos muy eficientes: gradiente conjugado [5], descomposición QR [3], método de Jacobi, etc. Todos estos métodos tienen órdenes de ejecución polinómicos. Por ejemplo, el gradiente conjugado, uno de los métodos más utilizados, tiene orden $\mathcal{O}(n^2)$ en general [11]. Además, el método puede alcanzar orden $\mathcal{O}(n \cdot s \cdot \sqrt{\kappa} \cdot \log \varepsilon)$ en el caso de que la matriz A sea definida positiva [4], donde κ denota el condicionamiento de la matriz.

La computación cuántica es un paradigma de computación basado en distintos principios de la mecánica cuántica. Aprovechando sus características, es posible alcanzar mejoras exponenciales en distintos problemas. En concreto, los autores Harrow A., Hassidim A. y Lloyd S. desarrollaron en 2008 un algoritmo cuántico para la resolución de sistemas lineales que añade una mejora exponencial [4]. Este método tiene varios problemas, como que necesita matrices hermitianas o que la preparación de los datos hace que desaparezca su mejora exponencial. Existen diversos artículos que cubren estos problemas, como [15], que se usó como base para el estudio.

A partir del interés del CESGA (Centro de Supercomputación de Galicia) en la computación cuántica, el CITMAga, elaboró un informe (enlace) sobre el tema. Varios de sus investigadores, Fernando Adrián Fernández Tojo y Francisco Javier Fernández Fernández, mostraron interés sobre un estudio más en profundidad del uso de la computación cuántica en la resolución de sistemas lineales, además de la utilización de la máquina real del CESGA.

El campo de la computación cuántica es bastante nuevo y aún se encuentra en crecimiento. El proceso en el mismo es a veces difícil, debido a la falta de computadores cuánticos con las características necesarias para las labores actuales. Sin embargo, la preparación de algoritmos y mejoras que permitan aumentos en la velocidad de los métodos clásicos es necesaria. Esta es la principal motivación de este trabajo. Además, la mayor parte de la literatura relacionada con este campo es, en ocasiones, muy complicada, y es necesario un estudio en profundidad para poder comprender qué se trata. Mediante este estudio, también se busca dar una visión más general y sencilla del algoritmo, para que así personas que no tengan tanto conocimiento, o aquellas que busquen una idea general puedan entenderlo.

3. Contexto de trabajo

- **Contexto en el que se enmarcan las prácticas:** Como se explica en el anterior apartado, este trabajo nace de la necesidad de investigación sobre si podrían existir métodos eficientes para la resolución de sistemas lineales mediante la computación cuántica.

- **Antecedentes:** El antecedente dentro de la organización es la elaboración de un informe sobre la computación cuántica para el CESGA. Dentro del campo de la computación cuántica, el mejor antecedente encontrado es [15], ya que amplía de manera muy completa el trabajo original de los autores del algoritmo HHL. Este artículo se utiliza como base en las implementaciones existentes en versiones de Qiskit (obsoletas) [6]. Además, también existen otros artículos muy interesantes que amplían la información sobre el HHL, como puede ser por ejemplo [18], que trata sobre la aplicación del algoritmo a matrices densas.
- **Recursos informáticos disponibles:** El recurso informático disponible más importante fue el acceso al Qmio, que pudo ser conseguido por la afiliación de un doctor investigador. Gracias a este acceso pude realizar distintas pruebas en un ordenador real, siendo muy útil para esta investigación.
- **Bibliografía:** La bibliografía completa se encuentra en la última página del documento, e incluirá todas aquellos elementos que se hayan utilizado tanto para realizar las prácticas, como para escribir el informe.
- **Manuales, tutoriales y otros cursos:** Para realizar la implementación de los distintos algoritmos y circuitos cuánticos se utilizó el lenguaje de programación Python [9], más en concreto el módulo Qiskit [6], desarrollado por IBM para computación cuántica. Qiskit tiene documentación, aunque, en mi opinión, es de bastante baja calidad de momento. Sin embargo, el Qiskit textbook, aunque obsoleto ahora por el lanzamiento de la versión 1.0.0 de Qiskit, permite entender de manera muy completa toda la base y los algoritmos tanto de manera teórica como práctica. El problema de que el libro esté obsoleto se soluciona buscando las funciones de Qiskit 1.0 que realizan las mismas operaciones. Para aprender (recordar) el uso del Finisterrae III, el *cluster* de supercomputación del CESGA, además de resolver los problemas provenientes de la instalación de la VPN tanto como del uso básico de las colas en el Qmio, utilicé el manual ofrecido por el CESGA en su portal de usuarios (enlace). Para la conexión y uso del Qmio, se uso el paquete de Python asociado, cuya documentación puede ser encontrada en el siguiente enlace (enlace). Esta documentación es actualmente algo pobre, pero esto es esperado ya que es de reciente elaboración, y se entiende que mejorará con el tiempo. Además, para conocer mejor el funcionamiento del ordenador cuántico del CESGA, participé en un curso avanzado del Qmio, en el que aprendí mucho sobre la teoría e innovación que hay detrás de los ordenadores cuánticos y sobre su funcionamiento. En este curso pude también dialogar con Andrés Gómez Tato, líder del grupo de proyectos del CESGA, que me dio algún consejo para el proyecto.
- **Grupo de trabajo:** El grupo de trabajo, como consecuencia, estaba formado por mi, además de mis tutores, que tenían un papel de consejeros, que me indicaban qué dirección seguir en caso de dudas. Cabe también mencionar que las prácticas fueron realizadas en modalidad híbrida, y que para algún problema de cálculos matemáticos o dudas generales, los compañeros que se encontraban en el edificio del CITMAga de manera presencial también me ayudaron a resolverlo.

4. Metodología

Debido a la naturaleza del trabajo de investigación, la metodología fue cambiante y dependía mucho de la fase del proyecto. Al inicio de las prácticas, gran parte del tiempo consistió en entender el funcionamiento del algoritmo, además de buscar implementaciones que pudiesen resultar útiles como base para la implementación final, además de definir un método para poder realizar las distintas mediciones necesarias. Posteriormente, al tener una base para generar los circuitos necesarios, comenzó una técnica de codificación y prueba debido a que el problema fue afrontado en pequeños paquetes de trabajo, formados por cada parte de los elementos a probar. Este método pudo ser utilizado de manera eficiente al trabajar de manera individual. Además, todo el trabajo realizado fue completamente documentado, tanto en las propias Jupyter Notebooks, haciendo comentarios y explicaciones de lo que se iba a realizar, como en un documento de texto, en el cual se indicaban los resultados y se hacían explicaciones sobre los mismos. El código fue almacenado en un repositorio de GitHub (enlace al repositorio), lo que me permitía realizar control de versiones, además de poder pasar de manera sencilla archivos dentro de la máquina del CESGA.

La modalidad pactada por mi y mis tutores para la realización de las prácticas fue híbrida, realizando

las primeras 3 semanas en la oficina de Santiago, y el mes de Julio en remoto, pudiendo, aún así ir a la oficina cualquier día, ya que tenía un lugar de trabajo durante la duración de las prácticas.

Finalmente, tuve reuniones periódicas con mis tutores de la empresa (generalmente una o dos veces por semana), de modo presencial o telemático, en las que discutíamos los avances realizados y los problemas que podían surgir. Mediante estas reuniones también decidíamos en conjunto cuáles serían los siguientes pasos a seguir en la investigación. Sin embargo, la responsabilidad de qué dirección seguir, recaía principalmente en mí.

5. Descripción del trabajo realizado

Esta sección estará dedicada a describir todo el trabajo realizado, pasando desde su concepción hasta la realización de las pruebas. También se explicará el interés que tiene este trabajo tanto en el mundo de la computación cuántica, como en el ámbito general de la computación. En primer lugar, se hará una pequeña introducción a la computación cuántica, en la que se explicará el principal algoritmo trabajado el denominado HHL [4]. Posteriormente, se tratará todo lo realizado en el proceso de investigación, tanto los métodos como los resultados obtenidos. Finalmente, se comentarán las distintas pruebas realizadas en el ordenador cuántico real. Para comprender mejor los conceptos principales tratados en las siguientes secciones, es muy recomendable leer el anexo A, en el que se hace una breve introducción a los conceptos de la computación cuántica.

5.1. Algoritmo HHL

Este es el algoritmo principal que se desarrolló durante las prácticas. En ellas se estudió su funcionamiento, tanto teórico como real; se analizó su eficiencia, tanto en número de puertas como en tiempo; y se probaron distintas modificaciones que teóricamente mejoran su eficiencia. En este apartado se explicará la base del funcionamiento de este algoritmo, y en los sucesivos se explicará el trabajo que se realizó sobre el mismo, además de la explicación de las distintas pruebas realizadas sobre el ordenador cuántico del CESGA.

El **algoritmo HHL** fue descrito por primera vez por Harrow, Hassidim y Lloyd en [4]. En este artículo definen teóricamente un algoritmo cuántico para la resolución de problemas lineales. Tomaremos la siguiente notación:

$$A\boldsymbol{x} = \boldsymbol{b}, \text{ donde } A \in \mathcal{M}_{N\times N}(\mathbb{R}), N = 2^{n_b}, n_b \in \mathbb{N}, \text{ es hermitiana y } \boldsymbol{b}, \boldsymbol{x} \in \mathbb{R}^N.$$

Definida esta notación, el algoritmo define los siguientes pasos, suponiendo que comenzamos con todos los registros en el estado 0 (es el estándar, en caso de empezar en 1, una puerta X lo pasa al estado 0):

1. Carga del vector $\boldsymbol{b}$ en los primeros n_b registros. Es importante recalcar que, debido a nuestra definición previa de cúbit, la suma de las amplitudes al cuadrado deberá valer 1, por lo que este método deberá trabajar tanto con el vector de lados derecho como con la solución normalizados, denotadas por $|b\rangle, |x\rangle$ respectivamente. Entonces tendremos que:

$$|0\rangle_{n_b} \mapsto |b\rangle_{n_b} = \sum_{i=0}^{N-1} b_i |i\rangle_{n_b}. \tag{1}$$

2. Aplicación del algoritmo cuántico **QPE** (*Quantum Phase Estimation*) a los $n_b + n_l$ primeros registros. El algoritmo QPE, dado U un operador unitario y $|\psi\rangle$ un estado, calcula θ tal que $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$. El problema se define mediante una matriz hermitiana, por lo que esta admite una descomposición espectral por sus autovalores. Por consiguiente, para aplicar QPE, podemos tomar como unitario $U = e^{iAt}, t \in \mathbb{R}$, dependiente de los autovalores de la matriz, consiguiendo como consecuencia el estado:

$$\sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle_{n_l} |u_j\rangle_{n_b}, \tag{2}$$

donde $|u_j\rangle_{n_b}$ es el j-ésimo autovector de la matriz A, $|\lambda_j\rangle_{n_l}$ es la representación binaria en n_l bits de $2^{n_l}\lambda_j$, y β_j son los componentes de $|b\rangle$ en la base $|u_j\rangle_{n_b}$. El valor de n_l puede ser elegido arbitrariamente para aumentar la precisión de la aproximación, pero este debe ser al menos mayor que n_b. Tras este paso conseguimos un estado formado por la combinación lineal de los valores de $\boldsymbol{b}$ y los autovalores de la matriz A.

3. Realización de la inversión de los autovalores, lo que permite la resolución del problema. Este es el paso más importante, y es el que realmente introduce la mejora cuántica del algoritmo. A partir de este momento el algoritmo se hace bastante difícil de seguir. Se seguirán indicando las fórmulas que lo rigen, pero se explicarán los pasos en mayor profundidad para que sea entendible.
   Para realizar la inversión, utilizaremos un conjunto de rotaciones condicionadas. Estas rotaciones son la versión controlada de la puerta $P(\alpha)$ descrita en la sección anterior. Los controles serán los elementos $|\lambda_j\rangle_{n_l}$, es decir, los elementos que definen los autovalores del problema, y el objetivo será un nuevo registro, denominado auxiliar, que es el que posteriormente decidirá si la solución del algoritmo es o no correcta. El estado tras las rotaciones es el siguiente:

$$\sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle_{n_l} |u_j\rangle_{n_b} \left(\sqrt{1-\tfrac{C^2}{\lambda_j^2}}\,|0\rangle + \tfrac{C}{\lambda_j}\,|1\rangle\right), \tag{3}$$

   donde C es una constante de normalización (se tomará el menor de los autovalores). Es importante entender que este es el estado completo del circuito, donde cada uno de los registros que existen en el circuito tienen el estado asociado al ket que tiene el mismo tamaño, por ejemplo, lo que está entre paréntesis (multiplicado por β_j) es el estado en el que se encuentra el registro auxiliar.

4. Aplicación del algoritmo QPE^{-1} (o $\text{QPE}^{\dagger}$), es decir, el inverso del QPE. Este paso deshace la transformación realizada en el paso 2. Permite volver a un estado inicial en los registros, pero tras haber modificado el auxiliar, además de mantener $|b\rangle$ en la base de autovectores de A. El estado obtenido es el siguiente (esperable tras las aclaraciones):

$$\sum_{j=0}^{N-1} \beta_j |0\rangle_{n_l} |u_j\rangle_{n_b} \left(\sqrt{1-\tfrac{C^2}{\lambda_j^2}}\,|0\rangle + \tfrac{C}{\lambda_j}\,|1\rangle\right). \tag{4}$$

5. Medición del registro auxiliar. En este paso se mide el registro auxiliar (además del resto del circuito en una QPU real), en el caso de que la medición devuelva el estado $|0\rangle$, se descartará el circuito, y deberán repetirse todos los pasos. Si el registro auxiliar se encuentra en el estado $|1\rangle$, esto indicará que el resto del circuito se encuentra en el siguiente estado:

$$\left(\sqrt{\frac{1}{\sum_{j=0}^{N-1} |\beta_j|^2/|\lambda_j|^2}}\right) \sum_{j=0}^{N-1} \frac{\beta_j}{\lambda_j} |0\rangle_{n_l} |u_j\rangle_{n_b}, \tag{5}$$

   lo que nos devuelve la solución de $|x\rangle$ escalada, ya que la norma del estado deberá ser 1. Es decir, si tomamos las amplitudes relativas al registro de tamaño n_b, y dividimos por la norma resultante tendremos $|x\rangle = \frac{\boldsymbol{x}}{\|\boldsymbol{x}\|}$, habiendo entonces resuelto el problema. En la Figura 1 podemos ver una vista general del circuito descrito.

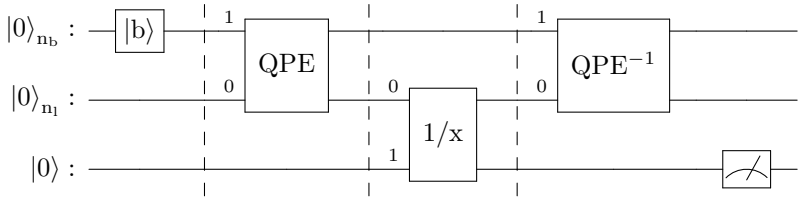


Figura 1: *Circuito HHL.*

Tomando como base [4], nacen otros artículos que resuelven distintos problemas que surgen de [4]. Un artículo muy importante, y en el que se basa todo mi trabajo a partir de aquí, es el escrito por Carrera Vázquez, Hiptmair y Woerner [15] en el que se discuten los problemas prácticos que nacen de una implementación real del HHL, además de dar aproximaciones y otros métodos para solucionarlos o, al menos, mitigarlos. Dentro de los principales problemas describe, los siguientes:

- La inicialización del vector $\boldsymbol{b}$. Para ello, se describen distintas maneras, como puede ser el uso de aritmética cuántica. Sin embargo, el artículo se fija más en un caso concreto, y es aquel en el que $\boldsymbol{b}$ esté descrito por una función analítica tal que $\boldsymbol{b}_i = f(x_i), x_i = i/(N-1)$. Para este caso, se puede utilizar una aproximación de $\boldsymbol{b}$ mediante un polinomio de grado d [15]. Su implementación se realiza mediante rotaciones y fue una investigación que duró bastante tiempo, debido a las distintas pruebas realizadas. Debido a ser algo más tangencial, esta se explica en el anexo B, pero se recomienda su lectura tras el apartado siguiente.
- La simulación hamiltoniana. Este es el paso de construir un circuito que actue como $e^{-iAt}, t \in \mathbb{R}$. Esto no es para nada trivial, además es necesario aplicar el circuito 2 veces, ya que tenemos tanto el paso que ejecuta QPE como QPE^{-1}. El artículo trata sobre matrices A tridiagonales simétricas de Toeplitz. Esto lo que quiere decir es que cada una de las diagonales de la matriz es constante, además por ser simétrica, las dos subdiagonales tienen el mismo valor. Estas matrices aparecen en problemas de resolución de ecuaciones en derivadas parciales utilizando un método de diferencias finitas. Además tienen muy buenas propiedades, lo que permite realizar la denominada "*troterization*", basado en el trabajo de Suzuki, Lie y Trotter [12]. En este paso, la matriz se divide en 3 matrices hermitianas: la primera contiene la diagonal inicial, $H_1 = aI_{n_b}$; la segunda se define por $H_2 = bI_{n_b-1} \otimes X$, siendo X la matriz de Pauli, es decir, es una matriz por bloques en el que cada bloque es una matriz simétrica 2×2 con diagonal principal a 0; la tercera es idéntica a H_2, pero encapsulando esa matriz dentro de otra con 2 filas y columnas más, con todo 0. Utilizando esto, se consiguen resultados muy buenos reduciendo mucho la complejidad del circuito. Esta parte no se investigó en el estudio, ya que parecía irse del alcance del proyecto, además de que la propia clase que implementaba esto en Qiskit [6] ya utilizaba esta aproximación.
- También dentro del anterior proceso, el artículo comenta el uso de extrapolación de Richardson para la simulación hamiltoniana [8],[1]. Esto se utiliza en combinación con el método anterior para reducir el orden de manera cuadrática. De manera similar al anterior punto, no se indagó más en este método, aunque parece conseguir muy buenos resultados por lo que indica el artículo.
- Inversión de autovalores. En este paso pasamos del circuito del estado recogido en la ecuación (2) al estado de la ecuación (3). Para construir este circuito se pueden tomar diversas rutas, pero el artículo comenta una opción, que es la utilización de una aproximación de $\arcsin(C/x)$. Para ello, se eligen intervalos, dependiendo del valor de $N_l = 2^{n_l}$, una vez elegidos estos intervalos, se utiliza la interpolación de Chebyshev [2], para aproximar $\arcsin(C/x)$ para un polinomio de grado d. Esta aproximación se tratará algo más en otro apartado.

- Observables. El artículo determina que, aunque posible, el extraer la solución del circuito produce un aumento del orden que eliminaría la mejora exponencial que este ofrece. Por lo tanto, este indica qué observables pueden ser extraídos del circuito sin perder la mejora exponencial. Estos son:
  - La norma de la solución $\|\boldsymbol{x}\|$.
  - El producto de la solución por una matriz tridiagonal Toeplitz B, $F_B(x) = \boldsymbol{x}^T B \boldsymbol{x}$.
  - La media absoluta de la solución.

  Estos observables pueden ser medidos de manera sencilla tras haber añadido alguna puerta específica y luego realizar la medición. En este estudio solo se tuvo en cuenta el cálculo de la norma, ya que no encontré aplicaciones directas de ninguno de los otros dos observables.

Una vez entendido este artículo y el funcionamiento del algoritmo HHL, pude comenzar a realizar una primera aproximación, en el que podría comenzar a realizar pruebas de funcionamiento.

5.2. Implementación, observables y primeras pruebas

Para comenzar a hacer la implementación, busqué algún ejemplo o alguna implementación ya existente, y descubrí que Qiskit [6] tenía una implementación. Sin embargo, esta se encontraba obsoleta, además de no encontrarse en la última versión de Qiskit, que es con la que quería trabajar, ya que es la que se utiliza en el módulo existente en el CESGA. Para afrontar estos problemas, decidí hacer una implementación del algoritmo utilizando como base el ya existente. Para ello, fui utilizando aquellas cosas que fuesen correctas en la función de Qiskit, modificando las que estaban mal o de una manera poco fiable. Finalmente llegué al código base que es el que utilicé para realizar las primeras pruebas de funcionamiento. Cabe destacar que este algoritmo ya implementaba la opción de realizar la inversión de autovalores, así que mantuve esta opción, pero arreglando alguna cosa que estaba mal, como puede ser el orden en el que se realizaba la misma (Qiskit toma el orden inverso al habitual en sus soluciones). El siguiente circuito es similar a la Figura 1, sin embargo, aquí se definen los registros como cúbits individuales. En este caso, el circuito es de dimensión $N = 2^{n_b} = 4$.

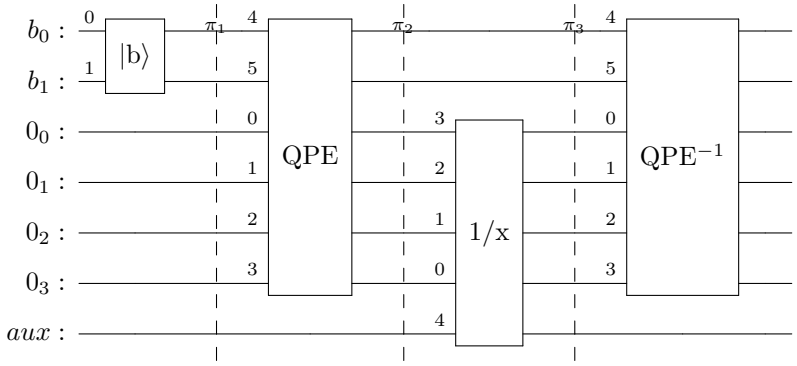


Figura 2: *Circuito HHL obtenido de mi función de Qiskit.*

Una vez finalizada esta implementación, ya con distintos elementos añadidos, como el permitir también realizar la inicialización del vector de lados derechos como una función, creé una clase de Python, en la que se incluyen todos los métodos necesarios para la generación y resolución de un problema de este tipo. Esto hace mucho más cómoda la gestión de los problemas y simplifica el probar distintas cosas que se tratarán posteriormente.

Es importante conocer los 2 tipos de ejecuciones (o simulaciones) que se pueden realizar mediante un circuito cuántico. Estas son:

- ***Statevectors*:** Tipo de ejecución que realiza una simulación exacta del resultado del circuito, devolviendo el estado $|\psi\rangle$ en el que se encuentra el circuito al final. Para ello, se realiza el producto

matricial del estado inicial del circuito por todas aquellas matrices que definen el circuito, ya que como se indica en la sección anterior, todas las puertas son matrices, y añadiendo la matriz identidad en aquellos cúbits a los que no afecte la puerta, conseguimos definir todo el circuito como productos de matrices. Esto produce una solución del circuito devuelta como un vector complejo de longitud $2^{n_b+n_l+1}$ exacto. No obstante, al utilizar una matriz real, la parte imaginaria es despreciable. Las soluciones de este circuito son muy distintas de una simulación real, sin embargo, son muy útiles para ver si el circuito funciona bien además de obtener directamente los resultados del problema.

- ***Shots*:** Esta ejecución es la ejecución real que se realiza en las QPUs. El circuito se ejecuta de manera completa, y al final se mide, lo que produce un colapso del estado a una cadena binaria. Cada repetición de este proceso se denomina un *shot* del circuito, y para conseguir una solución, se deben realizar varias medidas. Por lo general se toman números de shots de potencias enteras de 2, siendo el número habitual por precisión: $2^{13} = 8192$. Este tipo de medida es inexacta y probabilística, haciendo que si el número de shots tiende a infinito, se consiga la solución exacta del circuito.

En el trabajo se realizaron ambos tipos de medida, siendo utilizados cada uno de ellos para diferentes motivos. Las simulaciones mediante *statevectors* se utilizaron para comprobar que las distintas partes del circuito funcionaban, además de hallar soluciones de manera mucho más veloz. Las simulaciones por shots fueron utilizadas para comprobar la utilidad de los circuitos en un hardware real.

5.2.1. Cálculo del resultado a partir del circuito

Ahora que se tiene el circuito, además de la manera de ejecutarlo, será necesario extraer la solución del mismo, algo que no es siempre algo trivial debido a su implementación. Como se indica en la anterior sección, en el paso 5 del algoritmo HHL, en el caso de que el registro auxiliar mida 0, la solución deberá ser descartada, pero, ¿cómo se haría para comprobar eso en este caso, en el que se obtienen todas las mediciones a la vez, o en que se obtienen las amplitudes del estado? Esto se explicará en este apartado. Para ello hagamos la distinción entre los dos tipos de mediciones que se explicaron anteriormente.

- **Statevectors:** Como se indica anteriormente, al simular un circuito se obtiene un vector complejo de longitud $2^{n_b+n_l+1}$, pero de estos datos, solo nos interesan los $N_b = 2^{n_b}$ elementos colocados en el registro en el que se encuentre la inicialización de $|b\rangle$, en nuestro caso, el primer registro (tomando un registro como un conjunto de cúbits). Sin embargo, estos datos solo son útiles en el caso en el que el registro auxiliar tome también el valor 1. Para conseguir este efecto utilizaremos el hecho de estas amplitudes que nos devuelve no son mas que los valores α_j tales que $|\psi\rangle = \sum_{j=0}^{N-1} \alpha_j |j\rangle_N$, con $N = 2^{n_b+n_l+1}$. Suponiendo que el registro $|b\rangle$ se sitúa al inicio del circuito, tomando los N_b primeros valores tales que el autoestado tenga un 1 en la primera posición (esto es efecto de que Qiskit tome al revés los cúbits, ya que el registro auxiliar se coloca al final del circuito, pero su valor es el primero de la cadena binaria), conseguiremos nuestra solución sin normalizar. Para normalizarla, simplemente tomaremos el vector resultante y lo dividiremos por su norma. Veremos que la solución no es igual que la exacta, pero esto es esperable, ya que como se indica en el paso 2 del HHL, se realiza una aproximación de los autovalores de la matriz A.

```python
st = Statevector(self.qc).data.real
num = int(len(st)/2)
st = st[num:num+2**self.nb]
sol = st/np.linalg.norm(st)
```

(a) *Solución mediante Statevectors.*

```python
sol=np.linalg.solve(matrix,vector/np.linalg.norm(vector))
norm=np.linalg.norm(sol)
sol2=sol/norm

sol1=hhl.solution()
print(f'Solución mediante HHL: {sol1}')
print(f'Solución exacta: {sol2}')
```

✓ 0.0s

```
Solución mediante HHL: [0.39206738 0.5884583  0.5884583  0.39206738]
Solución exacta: [0.39223227 0.58834841 0.58834841 0.39223227]
```

(b) *Comparación con solución exacta.*

Figura 3: *Ejecución mediante statevectors.*

- **Shots:** La extracción de la solución tras la simulación con shots puede ser más complicada, pero utilizaremos para ella el concepto básico de cúbit y sus amplitudes. Una vez realizada la simulación tendremos un diccionario de Python, es decir, una lista formada por una clave, la cadena binaria de la solución medida, y el valor, el número de veces que se ha medido esa solución. Se contará el número de veces que aparece cada valor, y después dividiendo entre el número total de *shots*, se calculará la probabilidad de medir esa cadena. Ahora bien, ya que dado el estado $|\psi\rangle = \sum_{j=0}^{N-1} \alpha_j |j\rangle_N$, α_j^2 es la probabilidad de medir este estado, se tendrá que si $\mathbb{P}_j$ es la probabilidad de medir la cadena $|j\rangle_N$, entonces $\alpha_j \approx \sqrt{\mathbb{P}_j}$. Por lo tanto, podremos utilizar esto para calcular las amplitudes del vector de estado del circuito. Este es el método general, que se utiliza en el caso de necesitar todas las amplitudes, sin embargo, podemos aumentar la eficiencia de este método en el caso del HHL, ya que únicamente nos hacen falta N_b amplitudes. Podemos calcular solo estas amplitudes para calcular el resultado del problema. Para extraer únicamente estas amplitudes podemos utilizar el mismo método que en caso anterior, haciendo estadísticas de las primeras N_b cadenas que tengan un 1 en el primer bit. Para obtener la solución normalizada $|x\rangle$, será necesario normalizar el vector de amplitudes extraído.

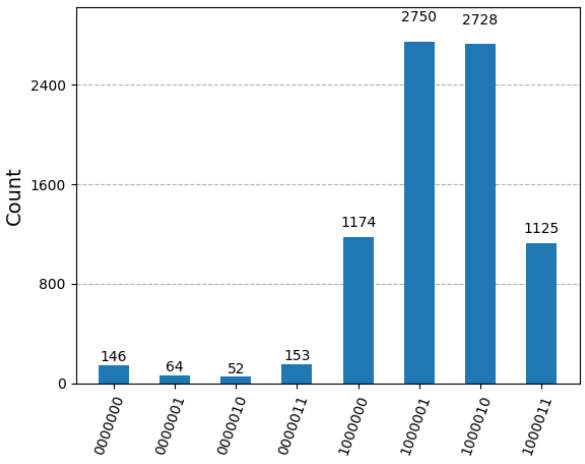
  Cabe notar que, debido a la naturaleza probabilística de esta simulación, los resultados van a ser mucho más imprecisos que los obtenidos a partir de *statevectors*. Una manera de mejorar la precisión es aumentar el número de *shots*, lo que hace que aumente el tiempo de ejecución, perdiendo entonces la ventaja cuántica en el proceso. No hay manera de evitarlo, es la naturaleza de este tipo de algoritmos.

```python
shots = 8192

counts = hhl.get_counts(shots)

prob_amplitudes = hhl.prob_from_counts_hhl(counts)

print("Solución por shots:", np.sqrt(prob_amplitudes))
print(f'Solución exacta: {sol2}')
plot_histogram(counts)
```

✓ 0.1s

```
Solución por shots: [0.40141352 0.58609373 0.58254225 0.39497577]
Solución exacta: [0.39223227 0.58834841 0.58834841 0.39223227]
```

(a) *Solución mediante shots.*



(b) *Histograma del número medido de shots.*

Figura 4: *Ejecución mediante shots.*

Poder calcular la solución normalizada es un gran hito dentro del proyecto, ya que podemos realizar comprobaciones de funcionamiento y comparaciones con la solución normalizada. Sin embargo, ver que funciona no es lo que se busca con estos algoritmos, sino que se busca que sean un reemplazo de los algoritmos clásicos, al menos en situaciones donde sea posible. Como consecuencia, en un algoritmo de resolución de problemas lineales queremos calcular la solución del problema original, no la del problema normalizado. Para esto, necesitaremos calcular la norma de la solución a partir del circuito.

5.2.2. Norma de la solución

En esta sección utilizaremos la misma notación utilizada por el artículo [15]. Denotamos por $|x\rangle$ a la solución normalizada, directamente extraída del circuito y $\boldsymbol{x}$ a la solución del problema original. Entonces, como indica el artículo, ya que no se realiza aún la aproximación del vector de lados derechos, se tendrá que dado $M_1 := |1\rangle\langle 1| \otimes I_{n_b}$:

$$\mathbb{P}_1 := \langle\psi_1| M_1^\dagger M |\psi_1\rangle = \frac{C^2\|\boldsymbol{x}\|^2}{\|\boldsymbol{b}\|^2} \implies \|\boldsymbol{x}\|^2 = \frac{\mathbb{P}_1\|\boldsymbol{b}\|^2}{C^2} \iff \|\boldsymbol{x}\| = \frac{\sqrt{\mathbb{P}_1}\|\boldsymbol{b}\|}{C}. \tag{6}$$

Es importante notar que, aunque $\mathbb{P}_1$ se defina como un producto escalar, este no define más que la probabilidad de medir 1 en el registro auxiliar, por lo que podemos tomar esta probabilidad experimental, obteniendo los mismos resultados. Esto es más importante para la simulación por shots, ya que calcular la probabilidad de medir 1 en el registro auxiliar es mucho más sencilla que calcular el estado total del vector y posteriormente multiplicar por la matriz M_1, además de añadir más error. Una vez que tengamos esta probabilidad, debido a que

$$\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|} \approx \frac{|x\rangle}{\||x\rangle\|} \implies \boldsymbol{x} \approx \frac{\|\boldsymbol{x}\|}{\||x\rangle\|}|x\rangle, \tag{7}$$

podremos calcular la solución real del sistema en el caso de que la probabilidad de medir 1 no se vea muy deformada por el ruido inducido por el circuito. Este problema se tratará en un apartado posterior. Suponemos aquí que la igualdad no es estricta, ya que como se comenta anteriormente, el circuito utiliza una aproximación de los autovalores, por lo que la solución puede o no ser exacta, existiendo una dependencia muy grande en la naturaleza de estos y lo buena que pueda ser esta aproximación.

Ahora que tenemos las herramienta necesarias para ejecutar el algoritmo y calcular su solución veamos algún problema interesante al que se le puede aplicar este método.

5.2.3. Ecuación diferencial

Como ya se indicó en una sección anterior, los sistemas de ecuaciones son muy comunes en la práctica, muchas veces como subproblemas. Un ejemplo de este caso es la siguiente ecuación diferencial con condiciones de frontera de tipo Dirichlet:

$$\begin{cases} -\Delta u(x) = -u''(x) = 1, & \in [0,1], \\ u(0) = u(1) = 0. \end{cases} \tag{8}$$

Esta ecuación en derivadas parciales aparece en distintos problemas de difusión, y es muy útil en la práctica. Para resolverla, podemos utilizar métodos exactos o métodos numéricos, donde podemos encontrar el **método de las diferencias finitas**.

El método de las diferencias finitas es un método numérico para la resolución de ecuaciones diferenciales basado en el cálculo de derivadas mediante diferencias finitas. Utilizando las aproximaciones siguientes:

$$\Delta u(x) = u''(x) \approx \frac{u(x+h) + u(x-h) - 2u(x)}{h^2}, \tag{9}$$

se tendrá un sistema de ecuaciones lineales $A\boldsymbol{x} = \boldsymbol{b}$, donde:

$$A = \frac{1}{h^2}\begin{bmatrix}-1 & 2 & -1\end{bmatrix} = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & 0 & \dots & 0 \\ -1 & 2 & -1 & \dots & 0 \\ 0 & -1 & 2 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 2 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix}, \tag{10}$$

donde $h = \frac{1}{N}$, siendo N el tamaño del problema. Sin embargo, para realizar la resolución del problema, descartaremos este factor ya que simplemente es un factor de escala del problema. Además, como vemos, la matriz es una matriz tridiagonal Toeplitz, es decir, el tipo de matrices que trata [15] y matrices con muy buenas propiedades.

Una vez tenemos el problema definido podemos realizar pruebas de funcionamiento y de error, para distintos tamaños de N. Hay que tener en cuenta que para dimensiones superiores a 32, la construcción del circuito es muy costosa. Esto se debe a que la implementación de Qiskit de QPE genera circuitos de enorme tamaño (circuito de más de un millón de puertas para tamaños superiores a 32), debido al uso de la transformada de Fourier cuántica (QFT). El alto tamaño de los circuitos de QPE, es el motivo que existan distintas aproximaciones variacionales (algoritmos híbridos) que buscan realizar este algoritmo de forma heurística pero más veloz. En la Figura 3b y la Figura 4a se observa la solución normalizada para el problema (8) de dimensión 4 resuelta mediante *statevectors* y *shots* respectivamente. En la Figura 5 se muestran los resultados para una ejecución de tamaño 32. Como se indica anteriormente, estos circuitos tardan mucho en generarse (aproximadamente 10 minutos) y en ejecutarse (alrededor de 8 minutos con *statevectors*), pese a ser circuitos de 15 cúbits.

```
Solution using HHL: [0.02801853 0.05427968 0.07878532 0.10153662 0.12253761 0.14179011
 0.15929618 0.17505762 0.18907429 0.20134564 0.21186999 0.22064523
 0.22766985 0.2329396  0.2364535  0.23821102 0.23821102 0.2364535
 0.2329396  0.22766985 0.22064523 0.21186999 0.20134564 0.18907429
 0.17505762 0.15929618 0.14179011 0.12253761 0.10153662 0.07878532
 0.05427968 0.02801853]
Exact solution: [0.02801728 0.05428348 0.07879859 0.10156263 0.12257559 0.14183747
 0.15934827 0.17510799 0.18911663 0.20137419 0.21188066 0.22063606
 0.22764038 0.23289362 0.23639578 0.23814686 0.23814686 0.23639578
 0.23289362 0.22764038 0.22063606 0.21188066 0.20137419 0.18911663
 0.17510799 0.15934827 0.14183747 0.12257559 0.10156263 0.07879859
 0.05428348 0.02801728]
```

Figura 5: *Solución para un circuito de dimensión 32 (15 cúbits).*

En la Figura 6 se muestran los resultados sin normalizar calculados por los dos métodos para un circuito de dimensión 4 (datos más legibles), además de una comparación con la solución analítica real.

```
1  sol_n = hhl.solution(norm=False)
2  sol_r = np.linalg.solve(matrix,vector)
3  ampl = ampl*hhl.norm_from_counts(counts)
4  print('Solución mediante statevectors: ',sol_n)
5  print('Solución mediante shots:\t',ampl)
6  print('Solución analítica:\t\t',sol_r)
7
✓ 0.0s
Solución mediante statevectors:  [1.99916739 3.00057257 3.00057257 1.99916739]
Solución mediante shots:         [2.02739045 2.98610671 3.01288847 1.98400094]
Solución analítica:              [2. 3. 3. 2.]
```

Figura 6: *Comparación de la solución normalizada mediante los 2 métodos.*

Aunque la solución sea muy buena, en el caso de querer hacer una comparación real de la precisión de una estimación por *shots*, debemos compararla no con la solución analítica, sino con la simulación por *statevectors*. Esto se debe a que los *statevectors* nos indican el máximo teórico de precisión que puede conseguir el circuito. Otra cosa a notar, es que estas simulaciones son simulaciones "perfectas", es decir sin el ruido presente el el hardware actual NISQ (Noisy Intermediate-Scale Quantum). Estos dispositivos, como el Qmio presente en el CESGA, tienen mucho ruido, además de una disminución de la fiabilidad con la aplicación de puertas sucesivas. Podremos ver este comportamiento en un apartado posterior.

5.3. Interpolación de Chebyshev

Además de la aproximación tratada en el anexo B, existen otras aproximaciones, como la aproximación de la inversión de los autovalores mediante la interpolación de Chebyshev [2]. En esencia, buscamos calcular el valor del $\arcsin(C/x)$ en el intervalo $[a, N_l - 1]$, siendo $N_l = 2^{n_l}$. Para ello, se tomarán intervalos $[a_i, a_{i+1}], i = 1, \ldots, M$, con $a_1 = a = 2^{\frac{2n_l}{3}}, ai + 1 = 5a_i$ y $M = \lceil \log_5(\frac{N_l - 1}{a}) \rceil$. Tomaremos entonces para cada uno de estos intervalos la interpolación de Chebyshev descrita en [2], que aproximará la función $\arcsin(C/x)$ mediante un polinomio de grado arbitrario d. Cuanto mayor sea el grado, mejor será la aproximación, sin embargo, la complejidad de evaluar un polinomio de este estilo escala cuadráticamente con el grado, lo que hace que no podamos aumentar el número de manera descontrolada.

El artículo [15] entra en mayor profundidad, dando límites para los errores, además de valores óptimos para las distintas variables de esta aproximación (como C). Utilizando estos valores, además de la función de Python que implementa la interpolación de Chebyshev para un conjunto de intervalos dados, podemos implementar esta aproximación. La implementación de este método estaba hecha pero no probada en la implementación de Qiskit que tomé como base, así que tuve que realizar varios cambios para que funcionase, siendo el más importante cambiar el orden de los cúbits al realizar la rotación. Esto se debe a que los cúbits en Qiskit se encuentran colocados en orden inverso al que se utilizarían en los cálculos manuales. Utilizando esta implementación, tenemos que la solución obtenida es una buena aproximación, como se puede ver en la Figura 7, aunque peor que la del circuito exacto que se puede ver en la Figura 3b.

```
Solucion usando HHL: [0.37499135 0.59948435 0.59948435 0.37499135]
Solución exacta:     [0.39223227 0.58834841 0.58834841 0.39223227]
```

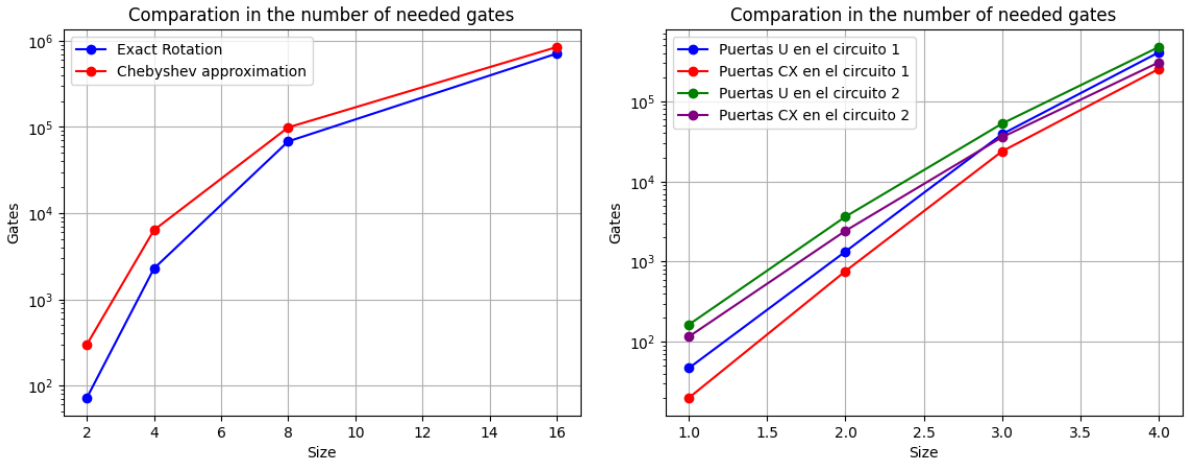
Figura 7: *Comparación de la solución mediante interpolación de Chebyshev.*

5.3.1. Comparación del número de puertas

A partir de esta aproximación, viendo el aumento del error y que el circuito añadía un mayor número de cúbits auxiliares, me pareció muy interesante hacer un pequeño estudio sobre el número total de puertas utilizadas por estos circuitos, como evolucionaban con el tamaño del problema, y como variaban entre métodos. Esto toma importancia sobre todo en la ejecución de este algoritmo en una QPU real, debido al denominado **tiempo de coherencia**. El tiempo de coherencia es el intervalo de tiempo en la que el estado de los cúbits pueden ser considerados coherentes, es decir, que su fase es previsible. En los ordenadores cuánticos reales, el tiempo de coherencia es bajo, por lo que un circuito muy extenso no podrá ser ejecutado en un ordenador cuántico real. Este estudio también se realizó sobre la aproximación del vector de lados derechos, sin embargo, debido al resto de puntos negativos que tiene esa aproximación, no se vio lo suficientemente importante como para mencionar. Sin embargo, para esta aproximación el punto negativo proviene de la duración de la misma, además del error, que es bueno. En la Figura 8a se muestra una gráfica que representa el crecimiento de las puertas.

Además de este crecimiento, puede ser interesante ver el tipo de puertas que utiliza cada circuito, ya que las puertas de un único cúbit tienen un tiempo de ejecución menor, además de añadir poco error. Si descomponemos al mínimo el circuito, este se representa únicamente por puertas de dos tipos: $U(\theta, \varphi, \lambda)$ y

CX. Si consideramos el circuito 1, el circuito que realiza la rotación exacta y circuito 2 el que utiliza la aproximación tendremos la Figura 8b.



(a) *Evolución del número de puertas necesarias.* **(b)** *Número de puertas U y CX para cada circuito*

Figura 8: *Gráficas sobre el número de puertas*

Vemos por consiguiente, que las puertas aumentan de manera lineal en la gráfica logarítmica, lo que implica que tienen un crecimiento exponencial. Esto es malo, y como se comentaba anteriormente en el documento, es el principal problema del algoritmo QPE. Entonces, si consideramos la ejecución en una máquina real, probablemente no podamos ejecutar el circuito para tamaños muy grandes directamente sobre la QPU debido al alto número de puertas. Podemos hacer una pequeña aproximación de hasta que valor podremos ejecutar el circuito. Para ello, se tomaron tiempos aproximados de vida de los cúbits en las QPU, además del tiempo que tarda en aplicarse una puerta. Los tiempos de vida de los cúbits son aproximadamente de $t_q =\sim 50 - 100\,\mu s$ y los tiempos de aplicación de las puertas son de $t_g =\sim 10 - 50\,ns$ aproximadamente. Utilizando esto, se toma el peor y el mejor caso, resultando así en un intervalo de puertas de

$$N_{puertas} = [1000, 10000], \text{ es decir, } \mu_{puertas} = 5000.$$

Como consecuencia, el circuito de tamaño 2 será siempre realizable, mientras el de tamaño 4 podrá o no serlo dependiendo de la QPU con la que se trabaje, siendo más posible que se pueda ejecutar el circuito con rotaciones exactas. Para tamaños superiores, los circuitos no podrán ser ejecutados dentro del tiempo de coherencia de los cúbits.

Como conclusión, esta aproximación es realizable y consigue los resultados esperados, sin embargo, debido al aumento de puertas, el tiempo del circuito también aumentará, por lo que no será útil para reducir el tiempo del circuito. Esto añadido al aumento de cúbits auxiliares, hacen que el error producido por una QPU real haga que la solución se consiga con mucha menor precisión.

5.4. Qmio

Finalmente, en esta sección se tratarán los resultados obtenidos en el Qmio el ordenador cuántico propiedad del CESGA. El Qmio, aunque se vea como un ordenador cuántico, realmente es un *cluster* formado por nodos de HPC, entre los cuales se encuentra un nodo con conexión a la QPU. Dentro de estos nodos también se incluyen otros nodos HPC clásicos (x86), además de nodos preparados para simulación cuántica de *statevectors* (A64). Tras escribir una pequeña propuesta de proyecto pude acceder a esta máquina para realizar los experimentos realizados. Los experimentos serán los siguientes:

- Comprobación del funcionamiento del algoritmo, además de determinar el máximo tamaño ejecutable.

- Comprobación de la utilidad de la aproximación del vector de lados derechos $\boldsymbol{b}$.
- Comprobación del funcionamiento de la aproximación de la inversión por interpolación de Chebyshev.
- Utilización de simulación de *statevectors* mediante Qulacs [13].

En primer lugar, tras aprender el funcionamiento de la partición cuántica gracias al curso en el que participé y a otros videos de cursos, colgados en el canal de Youtube del CESGA, comencé probando el funcionamiento del algoritmo en una QPU real. Para ejecutar en la QPU hay que hacer varias tareas al circuito generado:

1. Definir un objeto que permita la comunicación entre el programa y el ordenador cuántico, en el caso de Qiskit se denomina: *QmioBackend*.

2. Simplificar el circuito a puertas básicas.

3. Eliminar las instrucciones de "reset". QAT, el compilador cuántico, que traduce las instrucciones a las ondas electromagnéticas necesarias no permite esta instrucción, por lo que deberán ser eliminadas. Este es el motivo por el que es necesario el anterior paso.

4. Transpilar el circuito, es decir, transformar el circuito en un conjunto de puertas que pueda utilizar la QPU, además de indicar la posición real que tomará cada registro lógico del circuito. La posición real que toma para un circuito de dimensión 2 se puede ver en la Figura 11.

5. Ejecutar el circuito mediante la función `run`. Nos devolverá, igual que en Qiskit, un diccionario con las ejecuciones del circuito, que se puede analizar o graficar, como se observa en la Figura 9.

Siguiendo estos pasos, podemos comprobar el funcionamiento del circuito para cualquiera de los experimentos que se buscan realizar. Empecemos por el primero, la comprobación del funcionamiento del algoritmo sobre hardware real. Para ello, definiremos un script de python que cree el circuito y realice los pasos explicados anteriormente. Tras la ejecución para dimensión 2, obtendremos la Figura 9. Además, los valores normalizados obtenidos son correctos, como se puede observar en la Figura 10. Sin embargo, como se observa en la misma figura, el resultado normalizado que se obtiene no es correcto, ya que debido al ruido, la probabilidad de medir $|1\rangle$ en el bit auxiliar se ve completamente distorsionada, obteniendo un resultado erróneo.

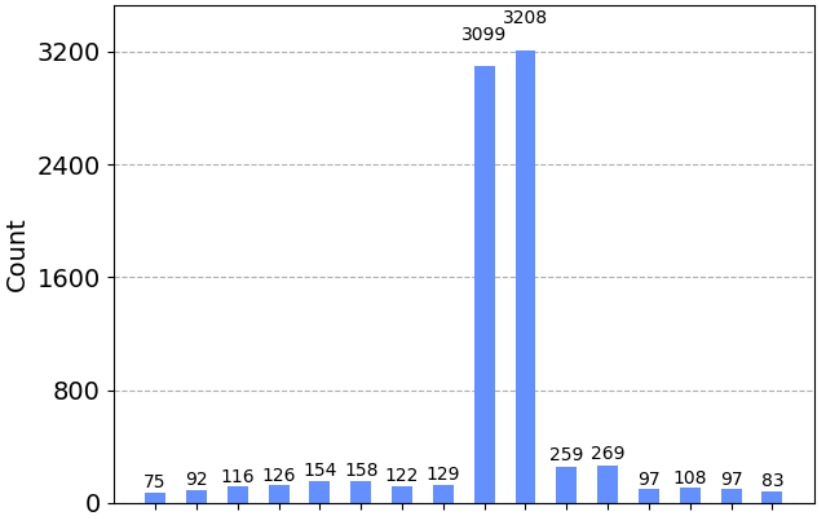


Figura 9: *Ejecución de un circuito de dimensión 2 sobre el Qmio.*

```
Estimated amplitudes of the solution: [0.71209555 0.70208256]
Solution:  [0.93614745 0.92298399]
```

Figura 10: *Solución sobre la QPU de un circuito de dimensión 2.*

Para obtener resultados con sentido, debemos indicar que el circuito se transpile con un nivel de optimización de 2. El nivel de optimización funciona de la misma manera que un compilador clásico e indica que optimizaciones se deben añadir en la compilación del circuito para la máquina real. Este nivel, además de aumentar la velocidad del circuito, reduce el ruido de la QPU, lo que permite obtener resultados como los que se observan en la Figura 9. Tras la transpilación del circuito, este se encuentra ya preparado para la ejecución sobre el Qmio, es decir se encuentra mapeado sobre 4 de los 32 cúbits que tiene esta QPU. La localización depende de la calidad y localización de los cúbits en el chip, utilizando en este caso los números 23, 29, 30 y 31.

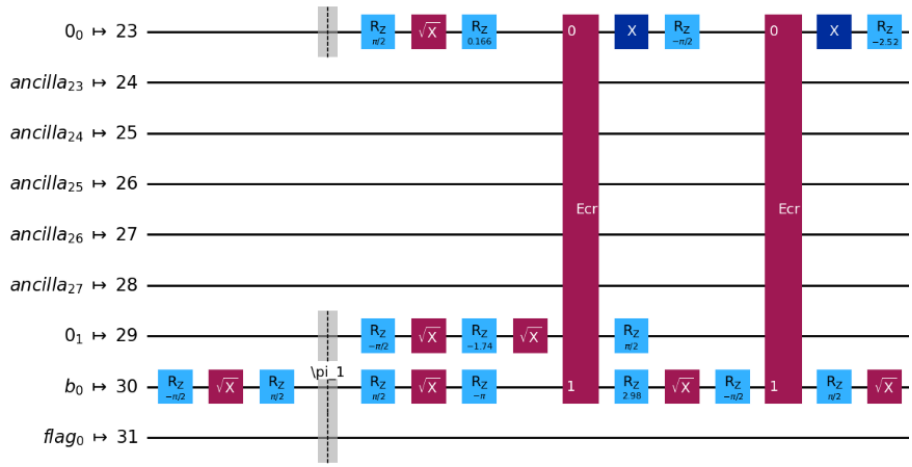


Figura 11: *Colocación de los cúbits en la QPU con nivel de optimización 2.*

Como podemos observar, viendo la topología (Figura 12), tiene mucho sentido esta disposición de los elementos en la QPU. En el circuito base, existen interacciones entre el vector de lados derechos b y los registros intermedios, además de interacciones entre los registros intermedios y el registro auxiliar. Así, para reducir mucho el número de puertas SWAP, se colocan en una posición en la que existan conexiones directas entre los cúbits, además e utilizar una posición (la 30), que permita la ejecución fácil de puertas de múltiples cúbits.

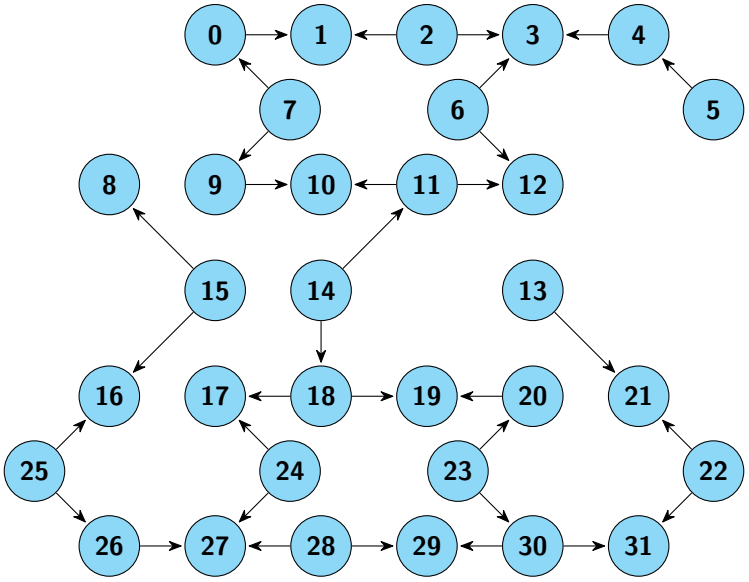
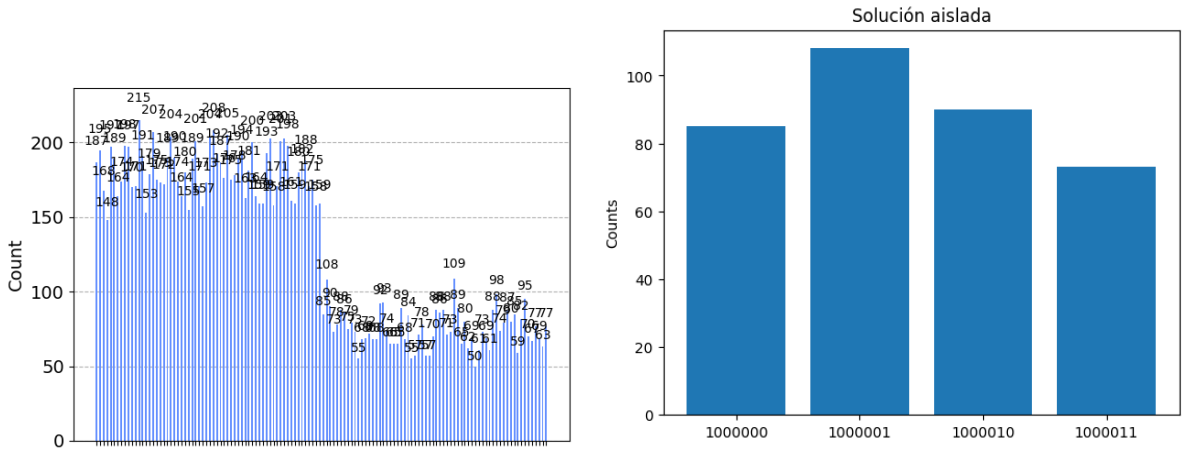


Figura 12: *Topología de la QPU del Qmio.*

No obstante, al aumentar el número de cúbits, la QPU no permite su ejecución, ya que el circuito es demasiado grande. Probando distintos niveles de "*repetition_period*", el circuito sigue fallando por otros motivos, por lo que no se pueden ejecutar circuitos de tamaño superior a 2 directamente sobre la QPU. Para dar una posible solución a este problema podemos tomar 2 rutas: utilizar el simulador cuántico `FakeQmio`, que produce los mismos resultados que la QPU real, pero permite la ejecución de circuitos de hasta 32 cúbits con un número de *shots* arbitrario; utilizar otra metodología, que puede ser una hibridación del circuito, ejecutando los QPE en el simulador *FakeQmio* o mediante *statevectors* y la rotación de los autovalores sobre la QPU.

En primer lugar se probó la primera opción, ya así se podría comprobar el circuito y ver los límites reales que alcanzaría la segunda opción. Sin embargo, la ejecución de circuitos de tamaño superior a 2 tiene un problema muy grande. Como se ve en la Figura 8a, el circuito tiene alrededor de 2000 puertas, siendo aproximadamente la mitad puertas de resonancia cruzada (puerta de dos cúbits), estas añaden mucho ruido al circuito, disminuyendo gravemente la fiabilidad, obteniendo resultados incoherentes, muy dañados por el ruido. En la Figura 13 se puede ver una ejecución completa de dimensión 4, además de las columnas de la solución aisladas. Es importante indicar que el ruido en este caso, al ser una simulación es bastante predecible, por lo que se podría hacer un análisis del ruido en el caso de que la solución fuese constante, y luego restar este ruido sobre la solución obtenida. En el caso de dimensiones superiores sucede lo mismo, pero las gráficas son demasiado ilegibles (si ya no lo eran las de dimensión 4).



(a) *Ejecución completa.* **(b)** *Solución aislada.*

Figura 13: *Ejecución para dimensión 4 con 16384 shots.*

5.4.1. Hibridación

Como segunda alternativa, utilizaremos hibridación de simulador y QPU real para la ejecución del circuito de tamaño superior. En primer lugar, deberemos separar la función que gener circuitos HHL en circuitos distintos:

- Inicialización de b y QPE
- Inversión de los autovalores
- QPE^{-1}

Estos circuitos pueden ser ejecutados de manera independiente, y debido a que en cada uno de ellos se le añade una inicialización al estado del anterior, la composición de los mismos devuelve el resultado correcto, con poco error añadido. Este método fue probado anteriormente a su uso sobre el Qmio, y los resultados obtenidos mediante ejecución completa por *shots* (todo simulado sin ruido) producía malos resultados. Investigando un poco, el motivo de ello era que el número de shots era muy bajo para conseguir una buena solución. Sin embargo, en el Qmio, el número máximo de *shots* que se pueden realizar son 8192. Para solucionar esto, implementé la misma función, pero que dividiese el número total de *shots* requerido por el usuario en lotes de máximo 8192 *shots*. Una vez finalizadas las ejecuciones, se juntarían el total de ejecuciones realizadas, además de los resultados, utilizando el total para calcular las probabilidades necesarias. Este método conseguía los mismos resultados, teniendo la desventaja de ser más lento. No obstante, era la única manera de "garantizar precisión" en el Qmio.

Sin embargo, si algo nos había indicado el experimento anterior, el ruido añadido es muy alto para conseguir soluciones con sentido. Mediante este método sucede lo mismo, con un mayor nivel de aleatoriedad, debido que se hacen más simulaciones, pero con un menor nivel de ruido, ya que la fiabilidad de la QPU no se ve tan forzada. Sin embargo, los resultados siguen siendo malos, como se observa en la Figura 14, en la que se prueban 2 simulaciones: primero ejecutando las QPE con *statevectors*, y la segunda utilizando *FakeQmio* para la estimación de fase.

```
Solucion con 1 y 3 con statevectors: [0.64254422 0.57253893 0.32845672 0.38916871]
Tiempo: 86.95765495300293
Solucion con todo simulado: [0.46119238 0.47140452 0.49119227 0.56904264]
Tiempo: 32.436134815216064
```

Figura 14: *Ejecución del circuito híbrido, utilizando 10 repeticiones (81920 shots) en el primer caso.*

Aunque este método funciona, para circuitos de mayor tamaño que 8, la rotación, lamentablemente,

también es demasiado grande como para ejecutar en la QPU, por lo que no se puede escalar de la manera esperada.

5.4.2. Aproximaciones

Una vez finalizado el análisis del funcionamiento del circuito podemos comenzar la prueba del funcionamiento de las aproximaciones. Como se trata en secciones anteriores, estas aproximaciones tienen diversos problemas, por lo que sobre el Qmio, solo se comprobó si funcionaban y si estos problemas detectados eran tan serios como se esperaba. Estas aproximaciones son las siguientes:

- **Aproximación del vector de lados derechos:** Como se esperaba, con esta aproximación, la probabilidad de medir $|1\rangle$ en el bit auxiliar es tan bajo, que las mediciones que se producen de los estados de la solución únicamente se dan por el ruido añadido por la QPU. Esto sucede incluso para simulaciones de dimensión 2. De esta manera, confirmamos que esta aproximación no es nada factible en la práctica.

```
Solución esperada: [0.62193269 0.75252747 0.21069217 0.05010937]
Solución obtenida: [0.46422193 0.56548842 0.50716078 0.45553133]
```

Figura 15: *Ejecución para la aproximación del vector de lados derechos.*

- **Aproximación de la inversión de autovalores:** Esta aproximación sí que funciona, pero añade dificultad para leer y obtener los datos específicos. Sin embargo, de la misma manera que el circuito con la rotación exacta, para dimensiones superiores a 2, los resultados no tenían sentido debido al ruido. En la Figura 16 (ejecutada en QPU) podemos ver un resultado similar al circuito exacto de dimensión 2, con dos picos diferenciables.

Comprobamos así las conclusiones sacadas de los respectivos apartados, viendo que los resultados obtenidos mediante las aproximaciones son peores, además de añadir un mayor ruido a la QPU, mediante el uso de un mayor número de puertas.

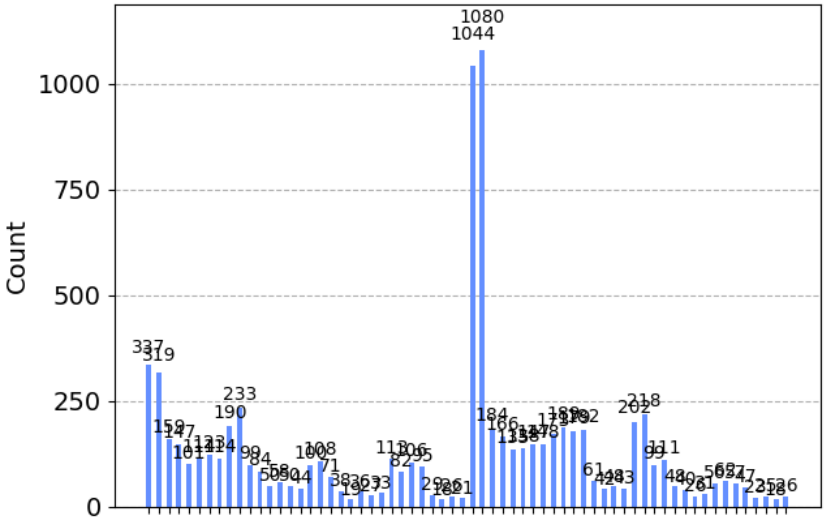


Figura 16: *Histograma de los shots con la aproximación de la inversión de autovalores para dimensión 2.*

5.4.3. Simulación en Qulacs

Qulacs [13] es un módulo de Python para la simulación de circuitos de alta velocidad desarrollado por la universidad japonesa de Osaka. Este añade todas las funciones necesarias para la construcción

y simulación de circuitos. Qulacs produce mejoras significativas en la simulación de circuitos mediante *statevectors* (sin tener en cuenta el *overhead* procedente de la transformación del circuito), como se puede ver en la Figura 18, por lo que convenía probar este método. Sin embargo, no existe una conexión directa con Qiskit, por lo que tuve que implementar una función que pasase un circuito de Qiskit a un circuito Qulacs. Sin embargo, durante el proceso, vi que en la partición A64, donde pensaba simular los circuitos Qulacs, no está disponible Qiskit, por lo que tuve que buscar otra alternativa. Para ello, utilicé un lenguaje intermedio, Open QASM 3, que permite tratar los circuitos como un fichero de texto plano. Qiskit permite transformar cualquier circuito a un QASM, posteriormente, podía parsear el fichero y construir un circuito QASM. Estos ficheros QASM permiten también ver el tamaño del circuito para 16 y 32 bits, con ficheros de texto plano que pesan 15 y 270 MB respectivamente. Una vez listo el código, podíamos utilizar las mejoras del Qmio, que añade una modificación de Qulacs para HPC, que permite el uso de OpenMPI (*Message Passing Interface*), uno de los métodos de paralelización más importantes e utilizados.

Sin embargo, debido, supongo, a la manera del circuito, o a una mala codificación, esta implementación no obtenía las mejoras esperadas utilizando MPI-Qulacs. Esto se puede ver por el tiempo, además de las estadísticas del uso de la CPU, donde se mostraba que solo se usaba un 2 %, lo que indica que únicamente utilizaba un core. Al contrario, si se hacía la implementación de la misma manera, pero usando Qulacs base, si que existía paralelización, utilizando un 50 % de la CPU, tardando muchísimo menos tiempo. Uno de los trabajos futuros sería comprobar este problema. Las soluciones y el tiempo requerido se pueden ver en la Figura 17.

```
Time with Qulacs:  208.86134934425354
Solución: [0.02561564 0.04962462 0.07202865 0.09282879 0.11202872 0.12963012 0.14563486 0.16004459 0.17285919 0.18407814 0.19369991 0.20172259
 0.20814477 0.21296258 0.21617513 0.21778193 0.21778193 0.21617513 0.21296258 0.20814477 0.20172259 0.19369991
 0.18407814 0.17285919 0.16004459 0.14563486 0.12963012 0.11202872 0.09282879 0.07202865 0.04962462 0.02561564]

Time with MPI-Qulacs:  14011.598337650299
Solución: [0.02561564 0.04962462 0.07202865 0.09282879 0.11202872 0.12963012 0.14563486 0.16004459 0.17285919 0.18407814 0.19369991 0.20172259
 0.20814477 0.21296258 0.21617513 0.21778193 0.21778193 0.21617513 0.21296258 0.20814477 0.20172259 0.19369991
 0.18407814 0.17285919 0.16004459 0.14563486 0.12963012 0.11202872 0.09282879 0.07202865 0.04962462 0.02561564]
```

Figura 17: *Comparación de las soluciones y tiempos de Qulacs y Qulacs-MPI para el circuito de dimensión 32.*

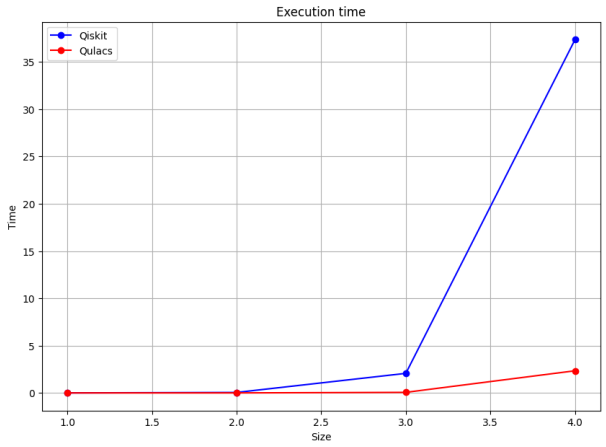


Figura 18: *Diferencia del tiempo de ejecución entre Qulacs y Qiskit (sin contar el overhead)*

5.4.4. Paralelización

Veamos ahora una posibilidad futura para mejorar la eficiencia de los algoritmos anteriormente mostrados, la paralelización. En este caso, la paralelización podría consistir en ejecutar varios circuitos al mismo tiempo. En una simulación, podemos hacer esto usando diferentes módulos de paralelización en Python, como Dask o Multiprocessing. Sin embargo, en un ordenador cuántico real podemos tener 2 enfoques diferentes:

- **Ejecutar múltiples circuitos en la misma QPU**: Podemos mapear múltiples circuitos en una única topología de QPU. En este método nos aprovechamos de la paralelización de las puertas existente en la computación cuántica, por lo que correr 2 circuitos independientes tarda lo mismo que un único circuito (y así es, la QPU permite correr 2 circuitos de tamaño 2, con la misma velocidad de correr 1). Este método es realmente agresivo ya que se genera una cantidad de ruido muy alta al duplicar el tamaño del circuito. Al probar este método sobre la QPU del Qmio, observamos el ruido esperado.

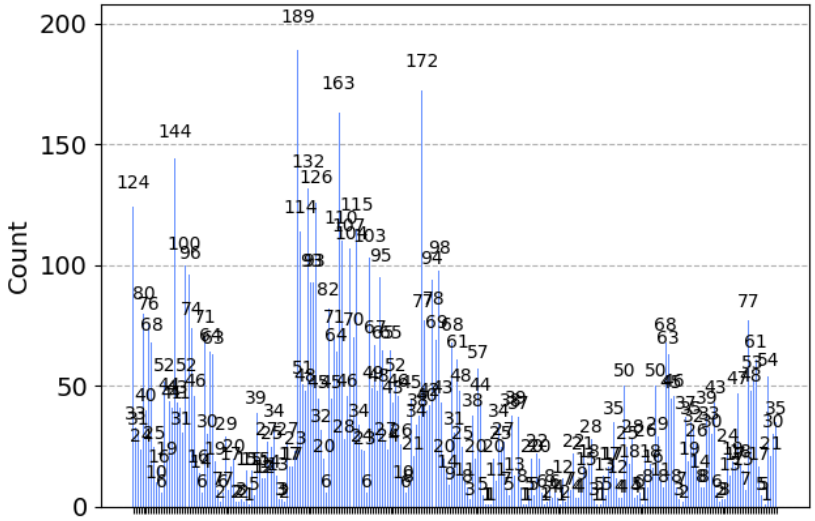


Figura 19: *Doble circuito en QPU.*

- **Paralelización de los circuitos y sus instrucciones clásicas**: Enfoque mucho menos agresivo. Podemos dividir las N ejecuciones de los circuitos cuánticos, y el cálculo clásico que hace funcionar el enfoque híbrido en N hilos diferentes. Mediante esto, conseguimos que mientras un hilo esté ejecutando el circuito, el resto que ya lo hicieron, ejecuten instrucciones clásicas. Sin embargo, tal como está diseñado actualmente el ordenador cuántico, además de que los circuitos que se pueden ejecutar completos sobre QPU tardan muy poco tiempo, no será útil, por lo que no se probará.

Para finalizar esta sección, como una curiosidad muy interesante, cualquier tipo de perturbación afecta a la QPU. El viernes 12 de julio, la QPU tenía un nivel muy alto y poco predecible de ruido (siendo normalmente las ejecuciones casi idénticas a las del FakeQmio). En el CESGA, los eventos, o conferencias se realizan los viernes. Por lo tanto, este ruido anormal se debe probablemente a algún evento que tiene lugar en el CESGA, ya sea un curso o algún otro tipo de presentación.

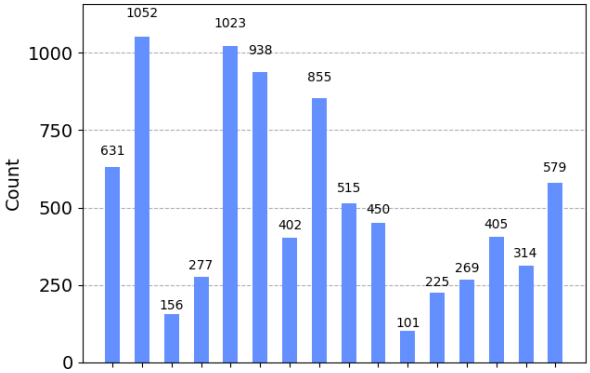


Figura 20: *Simulación con ruido anormal.*

5.5. Cronograma

Para medir el avance de las prácticas, mantuve un diario, en el que apuntaba el trabajo realizado cada día, además de los resultados conseguidos, para tener un más fácil acceso en la redacción de esta memoria. A partir del diario, se realizó el siguiente cronograma:

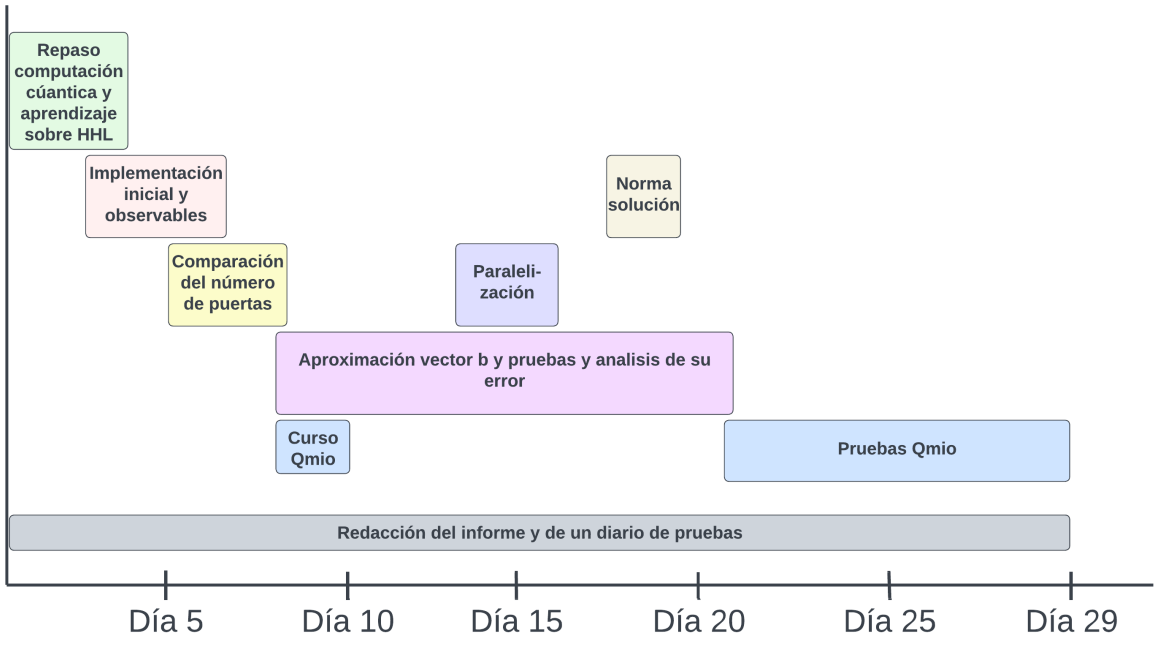


Figura 21: *Cronograma de las prácticas.*

6. Descripción del equipo de trabajo

Como se avanza en la sección 3, debido a ser un trabajo de investigación, el equipo de trabajo estaba formado por mi, en función de Investigador Principal, que avanzaba en el análisis de información y en la realización de código. Además yo mismo dictaba aquellas cosas en las que más me interesaba indagar y trabajaba sobre ellas. En el caso de quedarme estancado, tener dudas, o cualquier otro problema, contaba con dos excelentes tutores, investigadores del CITMAga y profesores de la Universidad de Matemáticas

de Santiago de Compostela del Departamento de Estadística, Análisis Matemático y Optimización, más concretamente del área de Análisis Matemático. Estos son:

- Fernando Adrián Fernández Tojo.
- Francisco Javier Fernández Fernández.

Ellos me ayudaron mucho durante toda la realización de las prácticas, facilitándome el material necesario en cada momento, proporcionándome ayuda cuando no entendía alguna cosa más matemática que no hubiese trabajado en la carrera, o simplemente indicándome algún tema con el que continuar. Además todas sus respuestas eran muy rápidas, contestando a los correos que enviaba casi al momento, estando siempre muy pendientes e implicados en el proyecto, incluso en momentos en los que estaban ocupados.

En cuanto a la metodología de colaboración, debido a que yo era el único miembro investigador, esto no aplica, sin embargo, como ya se indicó en la sección 4, todo el trabajo fue almacenado en un repositorio de Github, y en el caso de que tanto, yo, mis tutores o un tercero necesitase alguna parte del código, podía consultarlo ahi.

En mi opinión, el equipo de trabajo que se me fue asignado fue muy bueno y estoy muy agradecido por su trabajo y por su apoyo, gracias al cual pude realizar el proyecto de manera óptima.

7. Evaluación de la experiencia y lecciones aprendidas

Debido a la naturaleza de estas prácticas, algunos campos requeridos no aplican desde una perspectiva pura de ingeniería de software, por lo que los adaptaré a la naturaleza investigadora de mi proyecto.

- **A nivel de Análisis:** Considerando análisis, tanto como el análisis del problema, como de la bibliografía relacionada, este proyecto me hizo aprender mucho de cómo debía tratar nuevos conocimientos. En primer lugar, es más importante entender qué es lo importante del documento, sobre todo de [15], que usé como base para todo, ya que abordarlo todo era demasiado para la duración de las prácticas. También aprendí mucho sobre el análisis de código tras leer mucha cantidad de código en repositorios de Github, además de tratar con muchas documentaciones (en mejor o peor estado). De manera similar que con la bibliografía, mirar en profundidad el código puede ser complicado y en ocasiones puedes quedar más liado que al inicio, por lo que parar a la que alcances un nivel suficiente de entendimiento suele ser suficiente. Finalmente, comprendí la importancia que tienen las reuniones, tanto para comprender más el proyecto a realizar, eliminar problemas y tratar las posibles rutas de avance del proyecto.
- **A nivel de Diseño:** Aunque hacía pequeños esquemas antes de programar los distintos métodos, el diseño en mayor profundidad no fue algo a lo que le di importancia, sobre todo al implementar funciones directamente basadas en los pasos dados por los distintos artículos. Sin embargo, si que observé que si había planeado con anterioridad el código, esta implementación era más rápida, además de, por lo general, tener menos errores y estar mejor documentada. Como consecuencia, la lección aprendida ha sido que para otros proyectos que tenga en un futuro, realizar un diseño inicial básico de cualquier sistema software agiliza cualquier paso posterior. Una cosa que si que diseñé en mayor profundidad fueron las pruebas para realizar en el Qmio, ya que no sabía en un primer momento cuánto tiempo tendría para hacerlas. Gracias a diseñar estas pruebas, su proceso de ejecución fue rápido, ya que sabía lo que quería probar y cómo hacerlo.
- **A nivel de Implementación:** La implementación de todos estos algoritmos fue realizada utilizando Python. En la carrera este lenguaje se utiliza muy poco, y es muy importante conocerlo, sobre todo para cualquier tarea relacionada con el aprendizaje máquina o la inteligencia artificial. Además, pude mirar en mayor profundidad el funcionamiento del módulo de Qiskit, que probablemente utilice en la realización del TFG. También se priorizó bastante la legibilidad de las distintas *notebooks* y otro tipo de código, incluyendo comentarios y documentación para las funciones. Debido a que como al repositorio le añadí una licencia de código abierto, esto permite que otros usuarios puedan tomar mis funciones, modificarlas de manera sencilla y usarlas a su antojo.

- **A nivel de Evaluación:** La evaluación fue realizada, quitando las pruebas sobre el Qmio de manera superficial, eligiendo distintos casos de prueba. En el caso de error, me metía en el código y observaba qué era lo que podía fallar. Sin embargo, viendo lo mucho mejores que fueron las pruebas sobre el Qmio, debería haber utilizado en todo momento un método de pruebas y evaluación más sistemático, que permita encontrar errores y arreglarlos de manera sencilla y rápida.
- **A nivel de Mantenimiento:** El mantenimiento consistió en retomar funciones previamente realizadas y añadir elementos que las hiciese más genéricas, sin eliminar sus funcionalidades anteriores. Esto fue sencillo gracias al uso de los argumentos no obligatorios de Python, además de la buena documentación de las funciones. También puede considerarse dentro de este grupo la división de las *notebooks* en otras más pequeñas y específicas, pudiendo localizar lo que quería de manera más rápida. Esto tendría que haberlo hecho antes, por lo que una lección aprendida, fue esta, dividir las tareas e investigaciones en otras más pequeñas y específicas, lo que hace que te fijes más en ella y te vayas menos por las ramas.
- **A nivel de Algoritmos:** Aunque el punto principal de estudio fueron distintos algoritmos cuánticos, por lo que aprendí mucho sobre ellos, dentro de algoritmos clásicos, no utilicé nada que no hubiese utilizado antes. Además, la implementación de los métodos clásicos (y en general de las funciones), no fue realizada pensando en una eficiencia máxima, priorizando siempre una mayor legibilidad.
- **A nivel de Trabajo Colaborativo:** Como ya se indicó anteriormente, no tuve un trabajo colaborativo de manera directa, pero pude observar y comprobar lo importante que es tener tanto un repositorio de Git o simplemente una carpeta compartida a la que el mundo pueda acceder y ver el trabajo realizado y sus avances.
- **A nivel de Gestión de proyectos:** Pude aprender en mayor profundidad lo importante que es la gestión de tiempos para un proyecto, viendo que no se puede realizar todo lo que se desea. También comprobé lo importante que es la documentación y la comunicación para un proyecto. El diario que escribí con todos mis experimentos y las reuniones fueron muy útiles para mis tutores para comprender mejor el estado de la investigación.
- **Otras aprendizajes que se consideren relevantes:** Me pareció muy importante la capacidad crítica de decisiones que tuve que tener en el trabajo, viendo lo que era y no era importante. Esto pudo ser difícil al inicio, pero a la que entendía más el problema, pude ver lo que era importante, y aquellas cosas a las que debía prestar una mayor atención. También mejoré mucho en la lectura (sobre todo en diagonal) de artículos científicos, entendiendo mucho más qué hay y que no hay que leer de ellos. Finalmente, me gustaría recalcar todo el aprendizaje realizado sobre física cuántica, además del funcionamiento de los ordenadores cuánticos de manera específica gracias al curso avanzado del Qmio. Aunque ya tenía algo de base, en estas prácticas asenté conceptos de física cuántica, autoestados, bases de medición, ondas, que no comprendía del todo bien.

En mi opinión, estas prácticas me han gustado mucho, y me permitieron conocer otros elementos del mundo de la computación cuántica, aún en desarrollo. Gracias a ellas he podido aprender nuevos algoritmos que no había visto en cualquier otro estudio sobre el tema, reforzar mis conocimientos en campos que necesitaré para realizar los trabajo de fin de grado, y seguir practicando el lenguaje de programación Python. También pude ver desde otra perspectiva cómo es el entorno de trabajo de otra empresa relacionada a la investigación científica distinta al CESGA (y es bastante distinto). Todos los compañeros del CITMAga se portaron muy bien conmigo y me ayudaron en casos en los que fuera posible, sobre todo mis tutores. Estoy super agradecido con ellos por todo el trabajo extra que realizaron para ayudar a mi investigación, incluso en momentos de alta carga de trabajo por la USC. Gracias a ellos, pude ver el interés que tenía tratar este algoritmo (existen muchos trabajos teóricos, pero pocos prácticos) además de ver dónde se pueden usar cosas aprendidas en la carrera, donde muchas veces no te cuentan sus usos reales. Me hubiese gustado tener algo más de tiempo, en los que podría haber explorado conceptos como el error explícito de las distintas aproximaciones y el circuito en general, otros observables para el circuito... Recomendaría estas prácticas a cualquier persona que tenga interés en la investigación y a conocer el mundo de la computación cuántica, que aunque tenga bastante carga matemática, es super interesante y puede llegar a ser muy importante en un futuro. Como punto negativo puede ser la sensación

agridulce del resultado de las ejecuciones sobre una máquina real, lo que nos enseña que estas aún están en un punto muy inicial, y cualquier ventaja obtenida por la computación cuántica se ve paralizada por el hardware actual. Además, en relación a las prácticas, estas requerían ya un conocimiento previo del campo (tanto de matemáticas como de computación cuántica), ya que en caso de no tenerlo, probablemente no se podría haber hecho ni la mitad de lo realizado.

8. Aplicación de los conocimientos/competencias de los estudios

Para esta sección se tendrán en cuenta los conceptos aprendidos por las dos carreras (Matemáticas e Ingeniería Informática), sin embargo se hará un mayor hincapié en aquellos conceptos aprendidos en el GREI.

Aunque gran parte de los conocimientos necesarios utilizados en el proyecto fueron aprendidos de 0, hay otros que no, que sí que se trataron por completo en la carrera. Varios de ellos están relacionados con el uso del supercomputador del CESGA. Gracias al conocimiento del funcionamiento del sistema de colas tratado en asignaturas como Sistemas Operativos o Ingeniería de los Computadores, pude utilizar el sistema de manera sencilla. El uso de MPI fue tratado tanto en la asignatura de Computación distribuida como en Arquitectura de Computadores. Como un conocimiento Finalmente, para comprender la transpilación del circuito, además de distintos conceptos del curso de Qmio, fueron necesarios conceptos de la asignatura de Compiladores e Intérpretes. Por parte de la carrera de matemáticas, todo el conocimiento sobre álgebra matricial y vectorial fue altamente necesario para comprender los nuevos conceptos. Además, para este proyecto específico, el conocimiento de métodos numéricos para la resolución de sistemas lineales también fue necesario.

Sin embargo, algunos conceptos tratados en alguna asignatura tuvieron que ser aprendidos de otra manera, ya que los conceptos explicados en la carrera no eran suficientes. Por ejemplo, el conocimiento muy vago de Python tratado en varias asignaturas, en las cuales no había que programar de manera directa, no fue suficiente y tuve que aprender más de Python, que aunque no es muy complicado, era necesario conocer qué módulos y funciones existían para realizar las tareas necesarias. Además, dentro del ámbito de las matemáticas, tuve que mirar algún concepto tratado muy poco en la carrera, como puede ser ver más sobre álgebra tensorial o la transformada de Fourier.

Existen muchos conocimientos nuevos obtenidos a partir de las prácticas, proveniendo todos del área de la computación cuántica. Dentro de este área, aprendí sus bases, sus algoritmos principales (QFT, QPE, Shor, Grover, algoritmos variacionales, etc.) y sus ventajas y problemas. Esto no se aprende en ninguna asignatura del grado actualmente, pero es posible que si la tecnología mejora sus prestaciones, pueda existir una asignatura (probablemente optativa) en la que se estudien estos conceptos.

Cualquier concepto aprendido es valioso en cierta medida, sin embargo me quedaría con 2 principalmente: la experiencia de trabajo en un entorno profesional y la capacidad investigativa para la realización del TFG y otras investigaciones futuras. La primera es muy importante, y es algo que no se ve en la carrera de ninguna manera, salvo en las prácticas (tanto extracurriculares como curriculares). Sin esta experiencia, nuestra futura entrada al mundo laboral sería mucho más dura y difícil. La segunda es más importante para completar los estudios restantes. Para acabar las carreras, solo me queda la realización de ambos TFG (los dos están relacionados con la investigación), lo que va a resultar más sencillo gracias al trabajo realizado en las prácticas, ya que puedo utilizar las lecciones aprendidas tratadas en el anterior apartado. Además el TFG de Ingeniería Informática es sobre computación cuántica, por lo que esta experiencia será muy útil.

Referencias

[1] Butcher, J.C.: Differential and Difference Equations, chap. 1, pp. 1–53. John Wiley Sons, Ltd (2016). DOI https://doi.org/10.1002/9781119121534.ch1. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119121534.ch1

[2] Deuflhard, P., Hohmann, A.: Numerical analysis in modern scientific computing (2003). DOI 10.1007/978-0-387-21584-6. URL https://doi.org/10.1007/978-0-387-21584-6

[3] Golub, G., Van Loan, C.: Matrix Computations. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press (2013). URL https://books.google.es/books?id=X5YfsuCWpxMC

[4] Harrow, A.W., Hassidim, A., Lloyd, S.: *Quantum Algorithm for Linear Systems of Equations*. Physical Review Letters **103**(15) (2009). DOI 10.1103/physrevlett.103.150502. URL http://dx.doi.org/10.1103/PhysRevLett.103.150502

[5] Hestenes, M.R., Stiefel, E.: *Methods of conjugate gradients for solving linear systems*. Journal of research of the National Bureau of Standards **49**, 409–436 (1952)

[6] Javadi-Abhari, A., Treinish, M., Krsulich, K., Wood, C.J., Lishman, J., Gacon, J., Martiel, S., Nation, P.D., Bishop, L.S., Cross, A.W., Johnson, B.R., Gambetta, J.M.: *Quantum computing with Qiskit* (2024). DOI 10.48550/arXiv.2405.08810

[7] Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge University Press (2010)

[8] Richardson, L.F.: *IX. The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam*. Philosophical transactions of the Royal Society of London. Series A, Containing papers of a mathematical or physical character **210**(459-470), 307–357 (1911). DOI 10.1098/rsta.1911.0009. URL https://doi.org/10.1098/rsta.1911.0009

[9] van Rossum, G.: *Python tutorial*. Tech. Rep. CS-R9526, Centrum voor Wiskunde en Informatica (CWI), Amsterdam (1995)

[10] Saff, E.B., Totik, V.: *Polynomial Approximation of Piecewise Analytic Functions*. Journal of the London Mathematical Society **s2-39**(3), 487–498 (1989). DOI https://doi.org/10.1112/jlms/s2-39.3.487. URL https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/jlms/s2-39.3.487

[11] Shewchuk, J.R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep., USA (1994)

[12] Suzuki, M.: *General theory of fractal path integrals with applications to many-body theories and statistical physics*. Journal of Mathematical Physics **32**(2), 400–407 (1991). DOI 10.1063/1.529425. URL https://doi.org/10.1063/1.529425

[13] Suzuki, Y., Kawase, Y., Masumura, Y., Hiraga, Y., Nakadai, M., Chen, J., Nakanishi, K.M., Mitarai, K., Imai, R., Tamiya, S., Yamamoto, T., Yan, T., Kawakubo, T., Nakagawa, Y.O., Ibe, Y., Zhang, Y., Yamashita, H., Yoshimura, H., Hayashi, A., Fujii, K.: *Qulacs: a fast and versatile quantum circuit simulator for research purpose*. Quantum **5**, 559 (2021). DOI 10.22331/q-2021-10-06-559. URL http://dx.doi.org/10.22331/q-2021-10-06-559

[14] Tojo, F.A.F., Fernández, F.J.F., Brage, F.J.P.: *Las Matemáticas en la era de la Computación Cuántica: nuevas fronteras* (2023)

[15] Vazquez, A.C., Hiptmair, R., Woerner, S.: *Enhancing the Quantum Linear Systems Algorithm Using Richardson Extrapolation.* ACM Transactions on Quantum Computing **3**(1), 1–37 (2022). DOI 10.1145/3490631. URL http://dx.doi.org/10.1145/3490631

[16] Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P., SciPy 1.0 Contributors: *SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python.* Nature Methods **17**, 261–272 (2020). DOI 10.1038/s41592-019-0686-2

[17] Woerner, S., Egger, D.J.: *Quantum risk analysis.* npj Quantum Information **5**(1) (2019). DOI 10.1038/s41534-019-0130-6. URL http://dx.doi.org/10.1038/s41534-019-0130-6

[18] Wossnig, L., Zhao, Z., Prakash, A.: *Quantum Linear System Algorithm for Dense Matrices.* Physical Review Letters **120**(5) (2018). DOI 10.1103/physrevlett.120.050502. URL http://dx.doi.org/10.1103/PhysRevLett.120.050502

A. Introducción a la computación cuántica

Esta sección estará basada en el documento que mis tutores prepararon el año pasado para el CESGA [14], en el cual se trata la computación cuántica desde un punto más matemático, lo que a mi me fue útil debido a mis conocimientos de la carrera. Gracias a esto, logré entender mejor algún concepto que en mi anterior base de conocimiento (Qiskit Textbook) se trataba desde un punto de vista más físico, que me queda más lejos.

El concepto más importante de la computación cuántica es el concepto de cúbit o bit cuántico. Es la unidad base de información de un sistema de computación cuántica y a partir de lo que nace toda la computación cuántica. Matemáticamente, considerando el **espacio vectorial** sobre $\mathbb{C}^2$, y la notación bra-ket (si un ket es un vector entendido como una matriz columna, el bra es su matriz traspuesta conjugada), tendremos que $|\psi\rangle \in \mathbb{C}^2$ es un **cúbit** si

$$\||\psi\rangle\| = \sqrt{\langle\psi|\psi\rangle} = 1,$$

es decir, un cúbit es un elemento unitario del espacio $\mathbb{C}^2$. Por consiguiente, podemos definir una base ortonormal del espacio vectorial, más en específico, la base canónica, a partir de la cual se puede definir cualquier elemento del mismo. Los elementos de la base son los siguientes:

$$\boldsymbol{e_0} = |0\rangle = \begin{pmatrix}1\\0\end{pmatrix}, \quad \boldsymbol{e_1} = |1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$$

Por lo tanto, para cualquier cúbit $|\psi\rangle$, se tendrá

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,,$$

donde α y β serán las **amplitudes** del cúbit y $|\alpha|^2 + |\beta|^2 = 1$. La base $\mathcal{B} = \{|0\rangle\,, |1\rangle\}$ se denominará **base computacional**. Además de esta base se pueden definir muchas otras, pero esta es la más interesante y útil desde un punto de vista general. Se denomina **estado de superposición** a cualquier estado en el cual $|\alpha| \neq 1, |\beta| \neq 1$. El nombre proviene a que estos estados pueden ser alcanzados mediante una combinación de los estados básicos.

Las **mediciones** colapsan el estado de un sistema en superposición a uno de los vectores de la base computacional. Sea $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,$, se tendrá que

- La probabilidad de medir el estado $|0\rangle$ será $|\alpha|^2$.
- La probabilidad de medir el estado $|1\rangle$ será $|\beta|^2$.

Es interesante ver que como tenemos elementos en $\mathbb{C}^2$ tales que la suma de sus amplitudes es 1, podemos parametrizar el espacio como una esfera dentro de $\mathbb{R}^3$, denominada **Esfera de Bloch**. Tomando la parametrización clásica de la esfera mediante 2 ángulos: θ (latitud), φ (longitud) y utilizando propiedades de las funciones trigonométricas y la exponencial compleja, dado $(x, y, z) \equiv (\theta, \varphi)$:

$$|\psi\rangle = |\psi(\theta,\varphi)\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\varphi}\sin\left(\frac{\theta}{2}\right)|1\rangle\,.$$

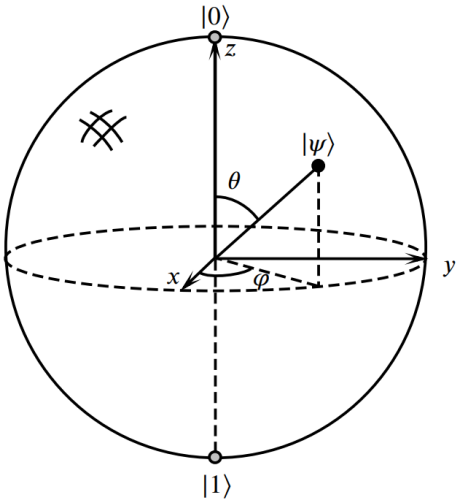


Figura 22: *Esfera de Bloch, [7].*

Sin embargo, un solo cúbit no es suficiente para resolver problemas, por lo que debemos aumentar el número de cúbits.

Teniendo $\mathbb{H} := \mathbb{C}^2$, entonces, al tener un sistema de n cúbits, sus estados se encontrarán en el espacio producto tensorial (visto como un producto de Kronecker) de n espacios $\mathbb{H}$, es decir: $\mathbb{H}^{\otimes n} = \mathbb{H} \otimes \cdots \otimes \mathbb{H} = \mathbb{C}^{2^n}$. Denominamos ***n*-cúbits** (a partir de este momento, simplemente cúbits) a un elemento $|\psi\rangle \in \mathbb{C}^{2^n}$ tal que $\||\psi\|| = 1$. De la misma manera en la que definimos el espacio, podemos definir las bases como el producto tensorial de los elementos de las bases. Por ejemplo, para 2 cúbits, la base computacional será:

$$\mathcal{B} = \{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}.$$

Siguiendo el patrón, podemos definir la base $\mathcal{B}$ para un número de cúbits n como el conjunto de las representaciones en binario desde el 0 hasta el número $2^n - 1$ de los vectores definidos como 0 en todas las posiciones menos un 1 en la posición que describe el número (más 1). Para simplificar la notación, se definirá $|i\rangle_n$ como el elemento i de la base en notación decimal. Por ejemplo, para 3 cúbits, tendremos $|110\rangle = |6\rangle_3$. Dentro de todos los estados posibles a partir de n cúbits, existen algunos que provienen de un producto de estados básicos de un único cúbit, mientras que otros que no. Estos últimos se denominan estados **entrelazados** y, físicamente, las propiedades de estos estados y de los cúbits que las forman dependen unas de las otras. Un ejemplo de un estado entrelazado para 2 cúbits es:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

Finalmente, veamos el elemento final que permite moverse desde un estado cuántico a otro, las denominadas **puertas cuánticas**. Podemos diferenciar las puertas según si afectan a uno o a varios cúbits. Sin embargo, la base matemática de todas es la misma. Una puerta $\mathcal{A}$ es una matriz unitaria compleja que se le aplica a un estado $|\psi\rangle$ para que pase a otro estado $|\phi\rangle = \mathcal{A}|\psi\rangle$. Veamos las puertas más importantes:

- **Puertas de un solo cúbit**: Este tipo de puertas afecta a un solo cúbit, por lo que son matrices 2×2. Las principales puertas son:
  - Puertas X, Y y Z de Pauli, que producen rotaciones de 180° del vector sobre cada uno de los ejes. Esto puede verse muy bien sobre la esfera de Bloch. Se pueden definir de la siguiente manera:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},$$

- Puerta **Hadamard** o H. Esta puerta induce sobre el cúbit un estado de probabilidades iguales para cada autoestado de la base computacional. Se define de la siguiente manera:

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

- Puerta de **desplazamiento de fase** (Phase-shift). Esta puerta induce una fase α sobre el el vector básico $|1\rangle$. Se define de la siguiente manera:

$$|0\rangle + |1\rangle \mapsto |0\rangle + e^{i\alpha} |1\rangle \implies P(\alpha) = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\alpha} \end{pmatrix}.$$

Todas estas puertas pueden ser expresadas por la denominada puerta universal U parametrizada:

$$U(\theta, \varphi, \lambda) = \begin{pmatrix} e^{-i(\varphi+\lambda)/2} \cos(\theta/2) & -e^{-i(\varphi+\lambda)/2} \sin(\theta/2) \\ e^{i(\varphi+\lambda)/2} \sin(\theta/2) & e^{i(\varphi+\lambda)/2} \cos(\theta/2) \end{pmatrix}.$$

- **Puertas de múltiples cúbits**: Estas puertas afectan a múltiples cúbits. Pueden definirse para tantos cúbits como se deseen. Existen algunas puertas como la Hadamard para varios cúbits que pueden ser representadas como el producto tensorial de matrices H, ya que es equivalente a aplicar puertas de un cúbit sobre los cúbits necesarios. Otras puertas importantes son:
  - Puertas de **rotación controlada**. Dentro de este grupo se encuentran la rotación controlada en cada uno de los ejes, dando lugar a las puertas CX (también llamada CNOT), CY y CZ. Estas puertas producen una rotación de 180° en el bit objetivo si el bit de control se encuentra a 1. Se puede extender esta idea a múltiples controles y objetivos, dando lugar a puertas de más cúbits. Una puerta importante es la denominada **Toffoli**, que es la puerta CCNOT, es decir una rotación en X doblemente controlada. Estas puertas inducen entrelazamiento cuántico.

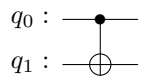


Figura 23: *Puerta CNOT.*

  - Puertas **SWAP**, intercambia la posición de 2 cúbits. Muy útil en la práctica, ya que en una QPU (*Quantum Processing Unit*) no todos los cúbits están interconectados, por lo que para realizar interacción entre ellos deben intercambiarse las posiciones constantemente. Esta puerta no puede ser implementada en una QPU, por lo que se utilizan conjuntos de puertas equivalentes. Una manera de realizarla es la siguiente:

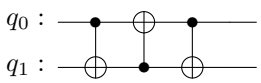


Figura 24: *Puerta SWAP mediante 3 puertas CNOT.*

Mediante el encadenamiento de estas puertas, podemos crear circuitos cuánticos, que nos permiten resolver nustros problemas, como el caso del algoritmo a estudiar, el HHL.

B. Aproximación del vector de lados derechos

Esta aproximación nace originalmente de la dificultad práctica de iniciar un registro cuántico con una función de onda de una amplitud específica. Qiskit implementa una función que realiza esta inicialización mediante la rotación de ángulos calculados en cada uno de los cúbits que se quieren inicializar. Sin embargo, el artículo [15], ofrece otra alternativa, basada en distintos principios de la aritmética cuántica y utilizando rotaciones controladas en el eje Y. En esencia, el funcionamiento de este paso es similar al del paso 4 del algoritmo descrito en la ecuación (3).

El método de preparación asume que $\boldsymbol{b}$ se define a partir de una función analítica tal que:

$$(\boldsymbol{b})_i = f(x_i), \text{ con } x_i = \frac{i}{N-1}, \tag{11}$$

siendo N la dimensión del problema. A partir de esto, podemos implementar el operador F que inicializa el registro a $\boldsymbol{b}$, de la siguiente manera:

1. Preparamos un estado de máxima superposición (todos las amplitudes con la misma probabilidad) en los n_b cúbits. Como se indica en una sección anterior, esto puede realizarse mediante puertas H. Además se añade un cúbit auxiliar al circuito.

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle_{n_b} |0\rangle . \tag{12}$$

2. Aplicamos el operador F de manera similar a la rotación de los autovalores del paso 4 del algoritmo HHL, mediante puertas $CRY(\theta)$ sobre el cúbit auxiliar, es decir, rotaciones controladas en el eje Y de ángulo θ. Mediante este operador pasaremos al estado:

$$\sum_{i=0}^{N-1} |i\rangle_{n_b} \left(\sqrt{1 - \frac{f^2(x_i)}{\|\boldsymbol{b}\|_\infty^2}} |0\rangle + \frac{f(x_i)}{\|\boldsymbol{b}\|_\infty} |1\rangle \right). \tag{13}$$

3. Finalmente, igual que en el algoritmo HHL, mediremos el valor del cúbit auxiliar. En el caso de que la medición sea $|0\rangle$, se descartará la inicialización. Si el cúbit auxiliar se encuentra en $|1\rangle$ al realizar la medición, el valor de los registros de tamaño n_b contendrá las amplitudes de $\boldsymbol{b}$. De hecho, al aplicar esta inicialización al HHL, el resultado sólo es correcto en el caso de que los dos cúbits auxiliares se encuentren en estado $|1\rangle$.

Ahora bien, ¿cómo podemos definir el operador F de manera eficiente?. Mediante aritmética cuántica puede realizarse, pero el *overhead* de esta hace que este método elimine la eficiencia del algoritmo HHL. Para ello, utilizaremos una aproximación de la función analítica $f/\|\boldsymbol{b}\|_\infty$ por una secuencia de polinomios de grado creciente p_f, como describe [10]. Una vez obtenido p_f, podemos utilizar el método descrito en [17] para definir un operador que obtiene en el cúbit auxiliar el estado $(\cos(p_f(x_i))|0\rangle + \sin(p_f(x_i))|1\rangle)$. Ahora, utilizando la linealidad local del seno alrededor del $x = 0$, podremos considerar $cp_f(x)$, con $c \in (0, 1]$ pequeño, con lo que obtendremos que $\sin(cp_f(x_i)) = cp_f(x_i) + \mathcal{O}(c^3)$. Lo que nos da un polinomio que podemos implementar. Pero, ¿cómo? Lo veremos en la siguiente sección.

B.1. Descomposición de funciones e implementación

En esta sección veremos como se realiza la implementación del operador tratado anteriormente, es decir la implementación de un operador F_c tal que:

$$F_c |i\rangle_{n_b} |0\rangle \approx |i\rangle_{n_b} \left(\sqrt{1 - (cp_f(x_i))^2} |0\rangle + cp_f(x_i) |1\rangle \right). \tag{14}$$

Como se indicaba anteriormente, utilizaremos rotaciones controladas en el eje Y para mapear la función. Sin embargo, en primer lugar, debemos descomponer la función p_f de una manera en la que se pueda definir para múltiples cúbits.

Dado un polinomio de grado N: $p(x) = a_N x^N + \cdots + a_1 x + a_0$, aplicado a n_b cúbits se realizará mediante 2^{n_b} rotaciones controladas. Los ángulos de las rotaciones dependerán de las amplitudes y el grado del polinomio. Para calcularlas, debemos calcular el valor del polinomio $p(x)$ para $x \in \{0, 1, \ldots, 2^{n_b-1}\}$. Una vez calculado estos valores, utilizaremos la representación en binario de los mismos, viendo que cúbits hacen aportaciones a cada uno de los valores. Sea $x = \beta_{n_b} \ldots \beta_1 \beta_0$ la representación binaria del entero x, aquellos $\beta_i = 1, \forall i = 0, \ldots, n_b$, serán los controles de la rotación. El ángulo de la rotación se definirá por el valor de $p(x)$ en caso de que x sea una potencia entera de 2, en otro caso:

$$\theta = N\left(p(x) - \textstyle\sum_{i=0}^{n_b-1} \beta_i p(2^i)\right), \tag{15}$$

tomando p sin el término independiente. Las contribuciones del término independiente se tendrán en cuenta en $x = 0$ resultando en $\theta = Na_0$, y realizando la rotación únicamente sobre el cúbit auxiliar. Por ejemplo, tomando $\boldsymbol{b}$ de dimensión 2 y tomando un polinomio de grado 2 $p(x) = a_2x^2 + a_1x + a_0$, tendremos:

$$\begin{aligned}
\theta_0 &= 2a_0, \\
\theta_1 &= Np(1) = 2(a_2 + a_1), \\
\theta_2 &= Np(2) = 2(4a_2 + 2a_1), \\
\theta_3 &= N(p(3) - p(1) - p(2)) = 2(9a_2 + 3a_1 - a_2 - a_1 - 4a_2 - 2a_1) = 2(4a_2).
\end{aligned}$$

Obteniendo entonces el siguiente circuito:

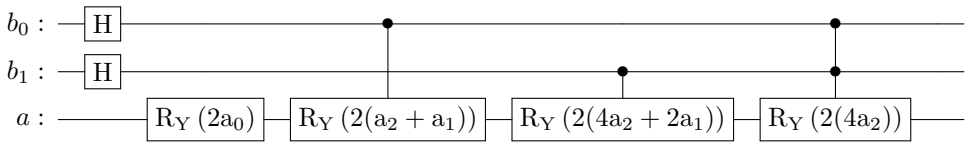
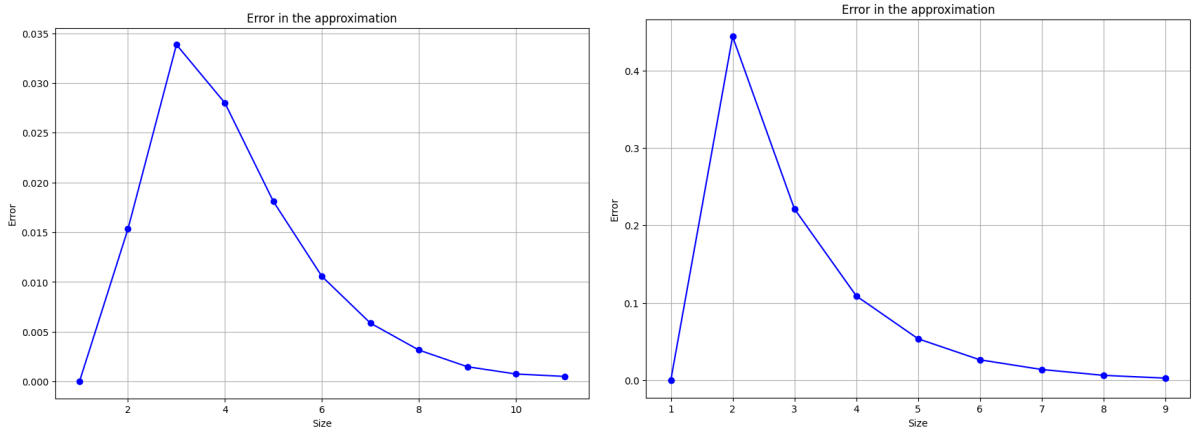


Figura 25: *Circuito que mapea* $\boldsymbol{b}$ *en un registro de tamaño* 2.

Sin embargo, nosotros estamos implementando cp_f, por lo que en todas las rotaciones, deberemos multiplicar el ángulo θ por el valor de c dado.

A partir de esto, se puede implementar esta aproximación en Qiskit (esta implementación fue íntegramente realizada por mí). En la Figura 26 se muestra la evolución del error cuadrático medio (ECM) de la norma de la aproximación para distintas funciones y tamaños del circuito.



(a) *Aproximación de* $f(x) = x^2 + x$, $c = 10^{-6}$.

(b) *Aproximación de* $f(x) = x^3 + 3x^2 - 5x$, $c = 10^{-6}$.

Figura 26: *Gráficas de error para distintas funciones.*

Como podemos observar en estas gráficas, el error aumenta rápidamente al inicio, pero al aumentar el tamaño este disminuye de manera muy rápida (ya que la norma del error se estabiliza). Podemos estudiar la evolución de estas funciones.

Para estudiar el error podemos utilizar distintos métodos de interpolación (como la interpolación de Lagrange) o métodos de adecuación (como distintos métodos de *fitting*). En primer lugar, podemos utilizar interpolación de Lagrange. De la primera figura obtenemos la siguiente expresión:

$$f(x) = 3{,}24 \cdot 10^{-5}x^8 - 5{,}48 \cdot 10^{-3}x^7 + 0{,}01x^6 - 0{,}04x^5 + 0{,}20x^4 - 0{,}64x^3 + 1{,}21x^2 - 1{,}21x + 0{,}47$$

De la segunda figura extraemos los siguiente expresión:

$$g(x) = -2{,}09 \cdot 10^{-5}x^8 + 9{,}23 \cdot 10^{-4}x^7 - 0{,}02x^6 + 0{,}18x^5 - 1{,}15x^4 + 4{,}52x^3 - 10{,}58x^2 + 13{,}18x - 6{,}14$$

Sin embargo, una interpolación polinómica, aunque puede ser útil para mostrar toda la evolución del error, no hace una aproximación buena de la evolución del mismo al aumentar mucho el tamaño del problema. Para ello, no se tomarán los primeros valores medidos, en los que existe un crecimiento lineal. Utilizaremos ahora el otro método, la mejor adecuación de los valores a una función dada. Observando el perfil de las gráficas, las funciones que más se parecen son:

- Función exponencial del tipo $a + be^{-cx}$
- Función inversa del tipo $a + bx^{-c}$

Utilizando la función de Scipy [16] `curve_fit` se puede calcular los coeficientes a, b, c que más se adecuen a la función dada. Posteriormente podemos graficar los resultados obtenidos, dando lugar a las siguientes figuras:

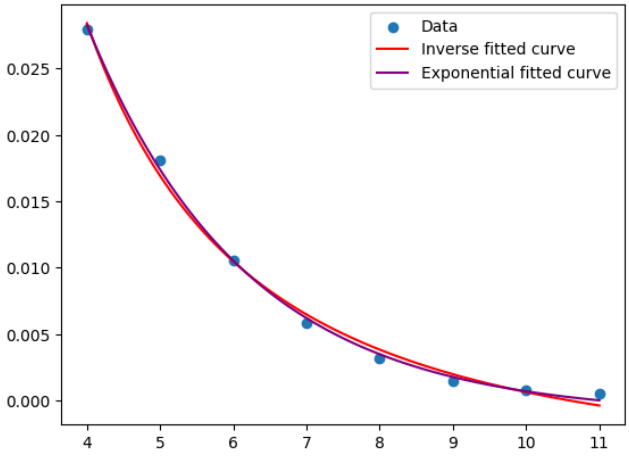


Figura 27: *Adecuación para la función* $x^2 + x$.

En este caso, la ecuaciones que definen las curvas son:

$$f(x) = -5{,}63 \cdot 10^{-3} + \frac{0{,}44}{x^{1{,}85}}, \quad g(x) = -1{,}13 \cdot 10^{-3} + 0{,}19e^{-0{,}46x}$$

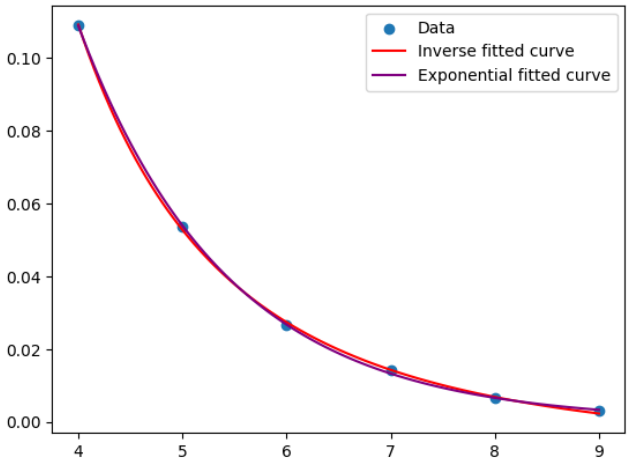


Figura 28: *Adecuación para la función* $f(x) = x^3 + 3x^2 - 5x$.

En este segundo caso, las ecuaciones que definen las curvas son:

$$f(x) = -8{,}83 \cdot 10^{-3} + \frac{6{,}70}{x^{2{,}91}}, \quad g(x) = 5{,}54 \cdot 10^{-5} + 1{,}81e^{-0{,}70x}$$

Podemos ver que la aproximación por la función exponencial es mejor. Por lo tanto tendremos que el perfil del ECM es una función exponencial negativa, como nos esperábamos, ya que el crecimiento del tamaño del problema es exponencial y existe una estabilización del error.

Sin embargo, en todas estas pruebas, el valor de c se mantiene en un número pequeño y constante. Sin embargo, su valor es más importante de lo que parece, y es lo que se estudiará en el siguiente apartado.

B.2. Problema del valor de c y análisis de Fourier de la probabilidad

Utilizando la implementación tratada anteriormente, se puede añadir en la función de construcción de circuitos que permita introducir el vector de lados derechos $\boldsymbol{b}$ como una función. Habiendo realizado esto, podemos comenzar a hacer pruebas de la aproximación para el HHL, obteniendo muy buenos resultados del problema, como se muestra en la Figura (29).

```
Solución con HHL (aproximado):   [0.25540959 0.51306228 0.63945808 0.51247089]
Solución con HHL:                [0.25540959 0.51306227 0.63945808 0.51247089]
Solución real:                   [0.25607376 0.51214752 0.6401844  0.51214752]
```

Figura 29: *Simulación mediante statevectors del circuito HHL para la función* x

Sin embargo, esto no es la realidad del método, ya que si vemos las amplitudes sin normalizar devueltas por el circuito, tendremos que estas son muy pequeñas (del orden de 10^{-6}). Como las amplitudes del vector de estados se traducen en probabilidades de ser medido cada estado, se tendrá que la probabilidad de medir la solución será del orden de 10^{-12}, es decir, imposible en la práctica. Este es el problema que indica [15], sobre que la probabilidad de medir $|1\rangle$ en el bit auxiliar decrezca al disminuir el valor de c. Se deberá comprobar como varía la probabilidad de medir $|1\rangle$ variando el valor de c.

En este estudio se comparará la influencia del valor de c en la probabilidad de medir cualquiera de los estados de la solución, en particular, se tomará la norma de la probabilidad, lo que permite ver cuán posible es en general medir la solución. En primer lugar, se comprobó la influencia de c para distintos órdenes de su valor, tomando para cada orden k los valores 10^{-k} y $5 \cdot 10^{-k}$, con $k \in 2, 3, 4, 5$. Además, para $k = 1$ se tomó algún otro valor. También se incluyó el 1, para completar el intervalo máximo. El

análisis se realizó para varias dimensiones del problema, pero para dimensiones superiores o iguales a 8 la posibilidad de medir la solución es tan baja que no merece la pena realizar este análisis. De manera contraria, para dimensión 2, las posibilidades eran importantes pero crecían únicamente de manera lineal, sin que pasase nada raro. Por consiguiente, el caso más interesante es el caso para dimensión 4. Como se observa en Figura 30, en la que hay que tener cuenta que ambos ejes están en escala logarítmica, para que se vea mejor la diferencia entre los puntos, la solución escala de manera lineal hasta 0,1, pero a partir de este punto tiene un comportamiento extraño. Exploremos este comportamiento más en profundidad.

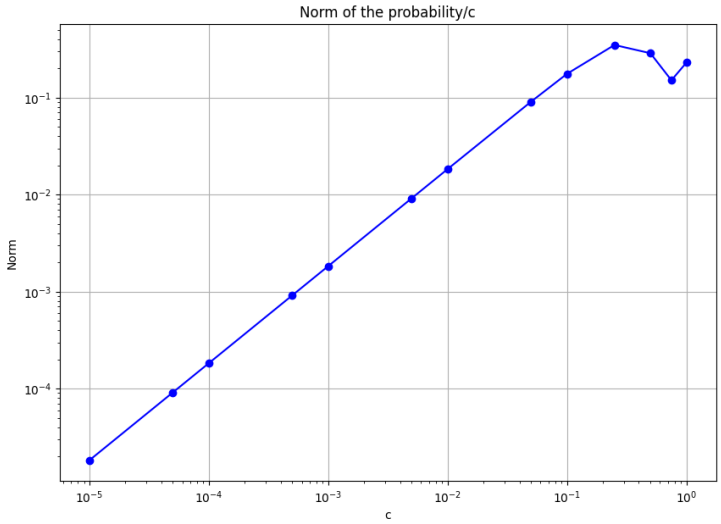
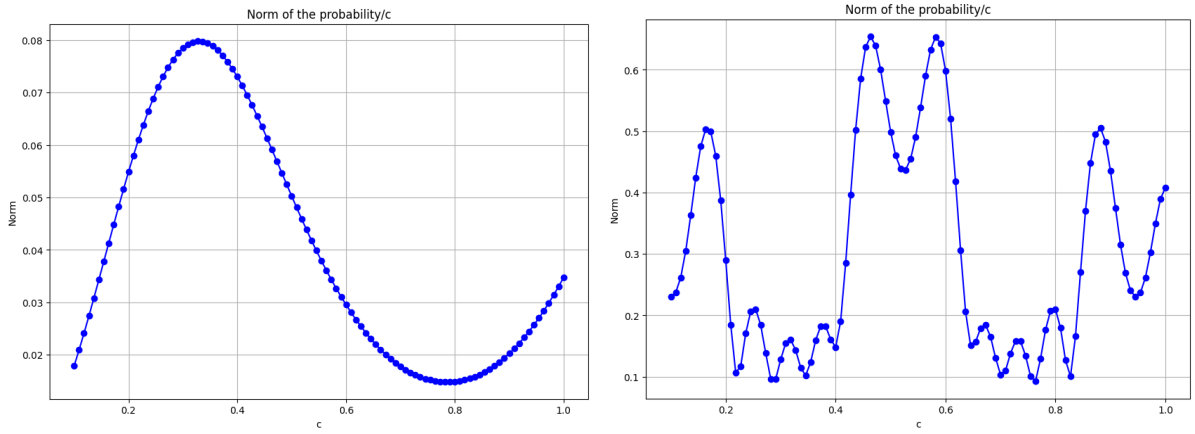


Figura 30: *Norma de la probabilidad de medir la solución para dimensión 4.*

Para explorar el comportamiento anormal, se tomarán una gran cantidad de muestras de c (100), para conseguir una buena evolución discreta. Haciendo esto se pueden conseguir gráficas muy buenas, que claramente provienen de funciones. En la Figura 31 se puede ver algún ejemplo:



(a) *Probabilidades para la función* $x^2 - x$.

(b) *Probabilidades para la función* $x^3 + x$.

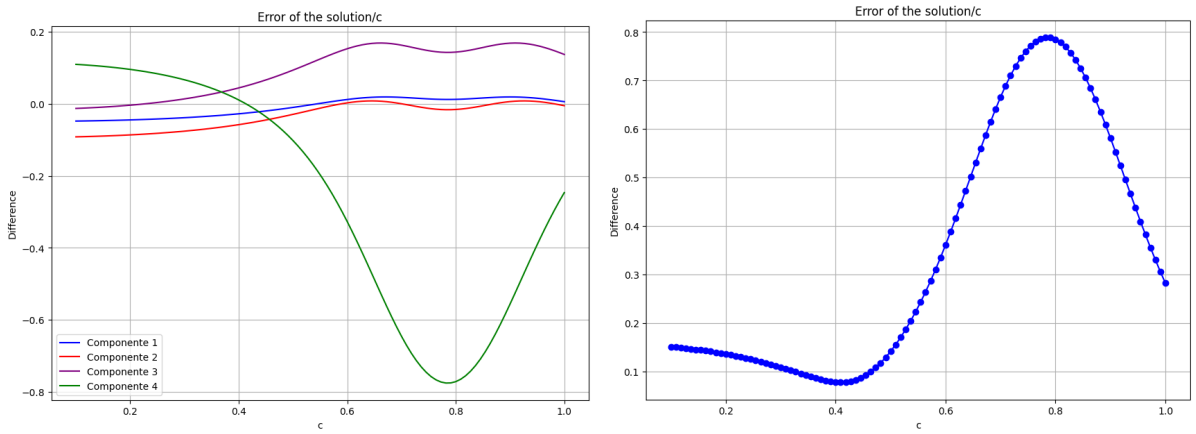
Figura 31: *Gráficas de probabilidad para varias funciones*

Viendo estas y otras funciones distintas del mismo grado, podemos hacer una pequeña aproximación sobre los valores de c para los que se alcanza el máximo de probabilidad:

- En 1 para las funciones de grado 1. Esto se debe a que el crecimiento lineal que se observa en la Figura 30 se mantiene para cualquier valor de c.

- Alrededor de [0,2, 0,4] para las funciones de grado 2.
- Mucha mayor variedad, pero aproximadamente en [0,3, 0,6] para las funciones de grado superior.

Sin embargo, el error aumenta cuando aumentamos el valor de c. En la Figura 32, la gráfica de la izquierda muestra las lineas error para los 4 elementos de la solución. Podemos ver que se consiguen buenos resultados aproximadamente hasta 0.4, aumentando el mucho el error para valores de c mayores. A la derecha podemos observar su norma.



(a) *Error para la función* $x^2 - x$.

(b) *Norma del error para la función* $x^2 - x$.

Figura 32: *Errores de la solución del problema para la función* $x^2 - x$.

Si queremos intentar extraer un valor óptimo (lo máximo que se pueda) del para c, probablemente el mejor valor sea aproximadamente $c = 0{,}3$, que permite reducir el error mientras la probabilidad toma un buen valor. Sin embargo esto depende completamente de la función utilizada, salvo para las funciones lineales, en las que el mejor valor podría ser $c = 0{,}5$, ya que se obtiene una buena aproximación con una alta probabilidad.

Finalmente, a partir de la Figura 31b, función que claramente es una suma de ondas, nació la curiosidad de la posibilidad de hacer un análisis de Fourier, para calcular las ondas que la forman. Cualquier onda puede descomponerse en una suma de funciones sinusoidales definidas por una frecuencia f, una amplitud A y una fase φ tal que $y(t) = A\cos(2\pi ft + \varphi)$ (o con el seno, ya que un coseno es un seno de fase $\varphi = \pi/2$). Así, una onda formada la suma de N ondas básicas será $y(t) = \sum_{i=0}^{N} A_i \cos(2\pi f_i t + \varphi_i)$. Sin embargo, calcular estas ondas básicas es complicado. Para ello, se puede utilizar una herramienta muy potente en el procesamiento de señales de cualquier tipo: la **transformada de Fourier**. Esta función es una transformación, que pasa la función al dominio de la frecuencia, en la que se pueden extraer las distintas frecuencias básicas que forman una onda. Analíticamente, la transformada de Fourier se define como la siguiente función:

$$\hat{f}(\xi) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} f(x) e^{-i\xi\, x} dx. \tag{16}$$

Sin embargo, debido a que no estamos en un dominio continuo sino en uno discreto, tendremos que usar la transformada de Fourier discreta, más en específico, la transformada de Fourier rápida (FFT por sus siglas en inglés). La principal diferencia de esta transformada discreta es el paso de una integral a un sumatorio. Aplicando este algoritmo a nuestros datos, podremos obtener los picos que se generan en los datos resultantes, las que serán las frecuencias de los cosenos que forman la función original. Por último, para comprobar la eficacia, además de calcular las frecuencias de manera más precisa, podemos hacer una adecuación, de manera similar a la realizada en la anterior subsección. Para dar un valor inicial a la amplitud A_i, tomaremos el valor del eje Y de la frecuencia asociada. Las fases iniciarán siempre en 0. Así, obtenemos los resultados mostrados en la Figura 33:

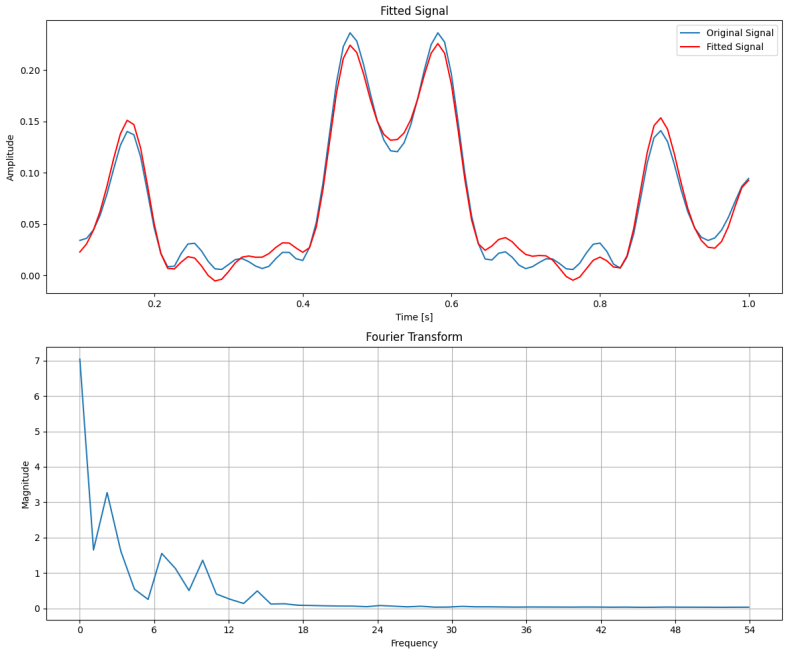


Figura 33: *Adecuación de la probabilidad de la función* $f(x) = x^3 + x$.

Además, la función para hacer el *fitting* nos devuelve todos los argumentos necesarios de la función obtenida, a partir de los cuales tendremos la siguiente expresión:

$$f(t) = 0{,}08\cos(2\pi 2{,}47t - 1{,}86) + 0{,}04\cos(2\pi 7{,}24t - 1{,}82) - 0{,}03\cos(2\pi 9{,}46t + 0{,}32) \\ - 0{,}01\cos(2\pi 14{,}34t - 0{,}08) + 0{,}11\cos(2\pi 0{,}58t - 121{,}31) \quad (17)$$

Como podemos observar, esta expresión tiene 5 términos, mientras que en la gráfica de $\hat{f}$ solo podemos diferenciar 4 picos. Sin embargo, viendo la función original, podemos ver que el último término, el coseno de baja frecuencia aparece en la función, por lo que es necesario. Vemos que los resultados son muy buenos, así que confirmamos nuestras sospechas, y es que esta función de error proviene de una suma de términos sinusoidales. El motivo probablemente sea la naturaleza del método, ya que estamos implementando una función analítica mediante rotaciones en el eje Y. Las rotaciones en el eje Y se definen mediante una matriz del siguiente tipo:

$$RY(\theta) = \begin{pmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{pmatrix}, \quad (18)$$

además de utilizar rotaciones similares en el algoritmo QPE. Esto podría llegar a causar este comportamiento. Sin embargo, debería realizarse algún estudio más exhaustivo, además de buscar más específicamente de dónde vienen estos términos y si se puede encontrar un patrón. Pero esto no era el objetivo de la investigación, por lo que se dejó aquí. Para otras funciones la aproximación por una suma de cosenos no consigue tan buenos resultados, pero sigue consiguiendo aproximaciones cercanas. Alguna de estas ejecuciones pueden verse en la *notebook* "b_side_approx.ipynb" del repositorio de GitHub.

Como conclusión, vemos que aunque esta aproximación funciona, y produce resultados teóricos, en la práctica no puede ser llevada a un entorno real, ya que en primer lugar, añade un cúbit extra en el circuito, lo que aumenta su complejidad; y en segundo lugar, solo conseguimos medidas en aquellos casos en los que la precisión es mala. Esto se acentúa con el ruido añadido por los equipos NISQ actuales, haciendo que la solución obtenida carezca de sentido, ya que no se sabe si la lectura de los valores es correcta o simplemente es el ruido de la máquina. Este es el principal motivo, por el que se rechazará el uso de esta aproximación.