Centro Brasileiro de Pesquisas Físicas

Mestrado em Física

Um estudo de sistemas lineares quânticos usando o Qiskit

Aluno: ***Hugo Santana Clemente***

Orientador: ***Prof. Dr. Alexandre Martins de Souza***

12 de julho de 2023

CENTRO BRASILEIRO DE PESQUISAS FÍSICAS
CBPF

Hugo Santana Clemente

Um estudo de sistemas lineares quânticos usando o Qiskit

Dissertação de Mestrado

Dissertação apresentada ao Programa de Pós-Graduação do Centro Brasileiro de Pesquisas Físicas como exigência parcial para obtenção do título de Mestre em Física sob a supervisão do Prof. Dr. Alexandre Martins de Souza.

12 de julho de 2023

"UM ESTUDO DE SISTEMAS LINEARES QUÂNTICOS USANDO O QISKIT"

HUGO SANTANA CLEMENTE

Dissertação de Mestrado em Física apresentada no Centro Brasileiro de Pesquisas Físicas do Ministério da Ciência Tecnologia e Inovação. Fazendo parte da banca examinadora os seguintes professores:

Alexandre Martins de Souza – Orientador/CBPF

Fabio Pereira dos Santos - UFRJ

gov.br Documento assinado digitalmente
LUCAS CHIBEBE CELERI
Data: 28/06/2023 16:18:33-0300
Verifique em https://validar.iti.gov.br

Lucas Chibebe Céleri – UFG

Rio de Janeiro, 28 de junho de 2023.

Às vítimas da COVID-19 e seus familiares.

"O sucesso nasce do querer, da determinação e persistência em se chegar a um objetivo. Mesmo não atingindo o alvo, quem busca e vence obstáculos, no mínimo fará coisas admiráveis."

José de Alencar

Agradecimentos

Aos meus pais Armando e Marli, por todo amor e incentivo.

A todos os professores, pelos bons ensinamentos. Em especial ao meu orientador e professor Alexandre Martins. Obrigado pelo suporte, incentivo e confiança!

Ao CBPF (Centro Brasileiro de Pesquisas Físicas), que me deu a oportunidade de integrar na pós-graduação de física.

Ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico), pela bolsa de estudos possibilitando a dedicação em tempo integral ao programa de pós-graduação.

Enfim, a todos que contribuíram direta ou indiretamente nos meus estudos, o meu muito obrigado.

Resumo

Explorando os fenômenos quânticos, a computação quântica oferece o potencial de reduzir drasticamente o consumo do tempo e energia da computação em alguns problemas específicos em comparação a computação clássica. Porém, o modelo de programação quântica é fundamentalmente diferente da programação de um computador tradicional. As principais empresas do ramo possuem sua própria plataforma de programação. Por exemplo, a IBM desenvolveu o Qiskit, o Google criou o Cirq e a Microsoft o Azure.

Os sistemas de equações lineares constituem uma classe de problemas encontrados principalmente na física, na engenharia e inteligência artificial. Existem métodos clássicos de resolução dos sistemas lineares, porém quando o número de equações aumenta, tais métodos muitas vezes se mostram ineficientes. Neste contexto diversos algoritmos quânticos foram recentemente propostos para acelerar a resolução de sistemas lineares.

A proposta deste trabalho é estudar três diferentes métodos para resolução de sistemas lineares quânticos, o método HHL, o CKS e o VQLS. Para explanar cada método é demonstrado o desenvolvimento passo a passo, comentando suas características, vantagens e implementando exemplos. Todas implementações foram realizadas utilizando a plataforma da IBM, o Qiskit.

Palavras-chaves: Algoritmos quânticos; resolução de sistemas lineares quânticos; HHL; CKS; VQLS; Qiskit.

Abstract

By exploiting quantum phenomena, quantum computing offers the potential to drastically reduce the time and energy consumption of computing on some specific problems compared to classical computing. However, the quantum programming model is fundamentally different from programming a traditional computer. The main companies in this field have their own programming platform. For example, IBM developed Qiskit, Google created Cirq, and Microsoft the Azure.

The linear systems belong to a class of problems found mainly in physics, engineering and artificial intelligence. There are classical methods for solving linear systems, but when the number of equations increases, such methods are often inefficient. In this case, several quantum algorithms have been recently proposed to accelerate the resolution of linear systems.

The purpose of this work is to study three different algorithms for solving quantum linear systems, the HHL method, the CKS method and the VQLS method. To explain each method, we demonstrate the development step by step, commenting about its features, advantages and implementing examples. All implementations were performed using the IBM platform, Qiskit.

Keywords: Quantum algorithms; solving quantum linear systems; HHL; CKS; VQLS; Qiskit.

Sumário

Capítulo 1

Introdução

A computação quântica explora fenômenos quânticos[1], como por exemplo, superposição, emaranhamento e tunelamento quântico afim de executar uma computação com mais eficiência. Em comparação com a computação clássica, também chamada computação digital, a computação quântica oferece o potencial de reduzir drasticamente o consumo do tempo e energia da computação em alguns problemas específicos.

Nos últimos anos, a computação quântica deixou de ser uma área puramente acadêmica e passou para a indústria, para a realidade técnica. Devido as vantagens da computação quântica, com a desaceleração das leis tradicionais de dimensionamento de *hardware* (Lei de Moore[2]), o constante avanço tecnológico tem acarretado no crescente interesse comercial, isto é, no aumento do investimento em tecnologia de computação quântica, em especial a partir da década de 2010. Nessa transição, muitos problemas tecnológicos precisam ser resolvidos. Para isso, foi necessário criar uma ponte entre a ciência quântica e a engenharia com o propósito de construir um computador quântico funcional. Assim, os computadores quânticos estão se tornando disponíveis comercialmente para pessoas além do laboratório acadêmico. Tradicionalmente, físicos e matemáticos estão na vanguarda deste campo. Atualmente o grande objetivo é desenvolver soluções de engenharia para construir um computador plenamente funcional e capaz de resolver problemas de interesse da academia e da indústria.

O modelo de programação quântica é fundamentalmente diferente da programação de um computador tradicional. Atualmente as ferramentas para programação de computadores quânticos estão disponíveis para uma ampla gama de pesquisadores e desenvolvedores. Para popularizar a computação quântica, muitas empresas criaram e disponibilizaram computadores quânticos de acesso remoto ou na nuvem com código aberto. O impacto e a vantagem dessa disponibilidade de código aberto são muitas, principalmente para o desenvolvimento de novas aplicações.

As principais empresas do ramo da computação quântica possuem sua própria plataforma de programação. Por exemplo, a IBM desenvolveu o Qiskit[3], a Google possui o Cirq[4] e a Microsoft o Azure[5]. Além disso, muitas outras empresas também estão desenvolvendo suas próprias plataformas de software para facilitar o usuário a programar computadores quânticos sem precisar conhecer

como as operações quânticas são implementadas no *hardware*. O desenvolvimento das linguagens de programação ocorre não apenas na indústria, mas também nas universidades e centros de pesquisas.

Os computadores quânticos possuem o potencial de resolver problemas de grande relevância, impossíveis de serem resolvidos de maneira eficiente pelos computadores convencionais. Podemos citar algumas áreas onde os computadores podem ser aplicados, por exemplo, no desenvolvimento de novos materiais, na inteligência artificial, cyber-segurança, em problemas de otimização, farmacêutica e na tecnologia em serviços financeiros.

Os computadores quânticos não são apenas mais rápidos que os computadores tradicionais, são fundamentalmente diferentes. Na informática digital, o bit é o elemento fundamental de informação, e somente pode assumir dois estados mutualmente excludentes, que podemos denotar por 0 e 1. Já na computação quântica, a unidade básica de informação é chamado de qubit (abreviação para *quantum bit*). O qubit é um sistema quântico com dois estados quânticos distinguíveis, onde cada estado está associado a um nível lógico possível, 0 ou 1. O estado quântico do qubit pode também ser uma superposição de estados lógicos. Sendo assim, novos fenômenos como superposição e emaranhamento estão presentes na computação quântica. Esses fenômenos são utilizados como recursos computacionais e fazem a computação quântica ser fundamentalmente diferente da clássica.

Apesar do recente avanço tecnológico, a computação quântica está apenas no seu início. Assim como os circuitos integrados levaram a uma revolução no processamento de informações no século passado, impulsionando o crescimento econômico e a produtividade, acredita-se que a computação quântica terá um impacto semelhante. A computação quântica e os algoritmos quânticos representam um novo paradigma para programação e design de algoritmos.

A resolução de sistemas de equações lineares constituem uma classe de problemas matemáticos de grande interesse da indústria e da academia. Em particular sistemas lineares podem ser encontrados em vários problemas de física, na engenharia, inteligência artificial e computação científica. Existem métodos clássicos para resolução de sistemas lineares, porém quando o número de equações aumenta, tais métodos clássicos muitas vezes se mostram ineficientes. Neste contexto, diversos algoritmos quânticos foram recentemente propostos para acelerar a resolução de sistemas lineares.

Neste trabalhamos realizamos um estudo de vários algoritmos quânticos para resolução de sistemas lineares. Os algoritmos foram implementados em computadores quânticos reais e comparados entre si. Para implementação utilizamos a plataforma da IBM, que está disponível como serviço na nuvem. Esta dissertação está organizada da seguinte forma. No capítulo 2 introduzimos alguns conceitos básicos sobre computação quântica, explicamos problemas definidos como sistemas lineares quânticos, como implementar os algoritmos no Qiskit e sobre algoritmos importantes utilizados como subrotinas nos algoritmos de resolução de sistemas lineares quânticos. No capítulo 3 explanamos três diferentes algoritmos quânticos para resolução de sistemas lineares. Eles são definidos como Harrow, Hassadim Lloyd[6], Childs, Kothari Somma[7] e *Variational Quantum Linear Solver*[8] e, para cada um deles

fizemos implementações de problemas e posteriormente faz-se uma comparação entre os diferentes métodos.

Capítulo 2

Conceitos fundamentais

Neste capítulo vamos descrever os conceitos e termos utilizados nos algoritmos quânticos e também o método de implementação dos algoritmos quânticos nos computadores quânticos reais disponibilizados através do IBM Q[3]. Além disso vamos introduzir a classe de problema definido como sistemas lineares quânticos e, definir alguns processos essenciais que serão usados para criação dos algoritmos quânticos para sistemas lineares.

2.1 Conceitos básicos da computação quântica

A computação quântica é feita basicamente através da manipulação de sistemas quânticos. Os detalhes físicos dessa manipulação dependem do design de *hardware* do computador quântico, porém não vamos nos ater a isso. O estado de qualquer sistema quântico é sempre representado por um vetor em um espaço vetorial complexo definido como espaço de Hilbert. Os algoritmos quânticos são expressados como transformações atuando neste espaço vetorial. Agora vamos explicar alguns dos conceitos básicos[1] e terminologia usados na computação quântica.

2.1.1 Qbit

O qbit é a unidade fundamental de informação usados em computadores quânticos, seu nome é a abreviação em inglês de *quantum bit* (bit quântico). Em comparação com a computação clássica, pode ser visto como a generalização de um bit. Mais precisamente, um qbit é um sistema quântico bidimensional. Seu estado pode ser expresso como,

$$|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.1}$$

onde α e β são números complexos tal que $|\alpha|^2 + |\beta|^2 = 1$. Na notação de Dirac, $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ e $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ são as representações dos vetores bases. Logo, o estado de um qbit é o vetor complexo

bidimensional $\begin{pmatrix}\alpha\\ \beta\end{pmatrix}$. Diferente do bit clássico, o estado de um qbit não pode ser medido sem alterá-lo. Ao fazer a medição de um qbit cujo estado é representado pela equação 2.1, o estado inicial vai colapsar para o valor $|0\rangle$ com probabilidade $|\alpha|^2$ ou para $|1\rangle$ com probabilidade $|\beta|^2$.

O estado de qualquer sistema quântico é um vetor unitário em um espaço vetorial complexo. A normalização é necessária para garantir que a probabilidade total de todos as saídas (resultados) da medição seja igual a um. Um computador quântico contém muitos qbits, portanto é preciso saber como representar o estado combinado de um sistema de qbits diante os estados dos qbits individuais. O estado de um sistema de qbits é descrito usando uma operação conhecida como produto tensorial.

Dado os seguintes estados dos qbits $|\phi_1\rangle = \begin{pmatrix}\alpha_1\\ \beta_1\end{pmatrix}$ e $|\phi_2\rangle = \begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix}$, o estado composto dos dois qbits é dado por

$$\begin{aligned} |\phi_1\rangle \otimes |\phi_2\rangle &= \begin{pmatrix}\alpha_1\\ \beta_1\end{pmatrix} \otimes \begin{pmatrix}\alpha_2\\ \beta_2\end{pmatrix} \\ &= \begin{pmatrix}\alpha_1\alpha_2\\ \alpha_1\beta_2\\ \beta_1\alpha_2\\ \beta_1\beta_2\end{pmatrix}. \end{aligned} \tag{2.2}$$

Vale ressaltar que muitas vezes o símbolo do produto tensorial $\otimes$ é retirado, deixando as equações mais simplificadas. Por exemplo, na equação 2.2 o produto $|\phi_1\rangle \otimes |\phi_2\rangle$ abrevia-se como $|\phi_1\phi_2\rangle$.

2.1.2 Superposição e emaranhamento

Qualquer combinação linear de dois estados quânticos, uma vez normalizados, também será um estado quântico válido. Esta combinação define-se como superposição. Diante disso, qualquer estado quântico pode ser expresso como uma combinação linear de alguns estados básicos. Como vimos na equação 2.1, qualquer estado de um qbit pode ser expresso como uma combinação linear de $|0\rangle$ e $|1\rangle$. Da mesma maneira, o estado de qualquer sistema de n qbits pode ser escrito como uma combinação linear normalizada dos 2^n estados da cadeia de bits. A base ortonormal formada pelos 2^n estados da cadeia de bits é chamada de base computacional.

Ao descrever um sistema de qbits é possível encontrar estados que não podem ser descritos como o produto tensorial de cada um dos estados dos qbits, como por exemplo o estado

$$\Phi = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle). \tag{2.3}$$

Estados que possuem essa característica são chamados de estados emaranhados. A existência de estados emaranhados[9] é um fato físico que tem consequências importantes para a computação quântica.

2.1.3 Medidas

O processo de medida pode ser visto como sendo a transformação da informação quântica, armazenada por um sistema quântico, em informação clássica. Quando medimos um qbit, ganhamos um bit de informação clássica, por exemplo, se o qbit colapsar após a medida para os estado $|0\rangle$ ($|1\rangle$), ganhamos um bit de informação clássica no valor 0(1). Como o qbit é um objeto quântico, os resultados da medição são probabilísticos.

Utilizando a notação de Dirac, as probabilidades de medição pode ser representadas como o quadrado dos valores absolutos das projeções do estado quântico do sistema em um estado da base computacional. Portanto, a probabilidade de obter $|0\rangle$ após uma medição do estado $|\phi\rangle$ é representada por $|\langle 0|\phi\rangle|^2$ e a probabilidade de obter $|1\rangle$ é $|\langle 1|\phi\rangle|^2$. Generalizando, a probabilidade de obter a cadeia de bits $x_1...x_n$ depois de medir um estado n qbits $|\phi\rangle$, é dado por $|\langle x_1...x_n|\phi\rangle|^2$ onde os termos x_i tal que $i \in \{1, 2, \ldots, n\}$ são binários.

Existem casos que é necessário fazer a medição em bases diferentes da base computacional. Isso pode ser realizado fazendo uma transformação de mudança de base apropriada nos qbits de registros antes da execução das medidas.

2.1.4 Portas e transformações unitárias

Em um computador quântico, a computação é realizada através da aplicação de transformações unitárias que modificam o estado dos qubits. Uma transformação unitária U é definida como:

$$UU^\dagger = U^\dagger U = I \tag{2.4}$$

onde $U^\dagger$ é o conjugado transposto e I a matriz identidade.

Fazendo uma analogia com as portas lógicas clássicas, como NOT e AND, na computação quântica também utiliza-se de transformações unitárias que são portas lógicas para manipular qbits. As portas quânticas são unitários de transformações que devem satisfazer a equação 2.4, onde a quantidade de qbits de entrada deve ser igual ao número de qbits de saída. Além disso, para cada porta U sempre é possível ter outra porta $U^\dagger$ que desfaça a transformação. Então, diferente das portas clássicas, as portas quânticas são reversíveis. Quando se diz reversível significa que as entradas da porta sempre podem ser reconstruídas através das saídas da mesma porta.

As portas lógicas quânticas podem ser portas de um qbit, ou seja, portas que atuam apenas em um qbit, ou portas de dois ou mais qbits. As portas de um qbit são expressas por rotaçãos $SU2$. Algumas portas importantes (tabela 2.1) de um qbit são: a porta NOT que inverte o estado do qbit, a porta Hadamard que cria superposição e a porta de fase que cria uma fase relativa entre os estados lógicos $|0\rangle$ e $|1\rangle$. As portas de condicionais de 2 qbits, aplicam uma operação unitária U sobre um determinado qbit alvo condicionado ao estado de um outro qbit de controle. A porta $CNOT$, por

exemplo, aplica a porta NOT, ou seja, inverte o qbit alvo apenas se o qbit de controle estiver o estado $|1\rangle$. A tabela 2.1 mostra algumas portas relevantes e sua representação matricial.

Porta / Símbolo	Representação matricial	Tabela verdade
Identidade ou I	$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	Entrada Saída $\|0\rangle$ $\|0\rangle$ $\|1\rangle$ $\|1\rangle$
NOT ou X	$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	Entrada Saída $\|0\rangle$ $\|1\rangle$ $\|1\rangle$ $\|0\rangle$
Y	$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	Entrada Saída $\|0\rangle$ $i\|1\rangle$ $\|1\rangle$ $-i\|0\rangle$
Z	$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	Entrada Saída $\|0\rangle$ $\|0\rangle$ $\|1\rangle$ $-\|1\rangle$
Hadamard ou H	$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	Entrada Saída $\|0\rangle$ $\frac{\|0\rangle + \|1\rangle}{\sqrt{2}}$ $\|1\rangle$ $\frac{\|0\rangle - \|1\rangle}{\sqrt{2}}$
De fase ou P	$\begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$	Entrada Saída $\|0\rangle$ $\|0\rangle$ $\|1\rangle$ $e^{i\theta}\|1\rangle$
CNOT	$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	Entrada Saída $\|00\rangle$ $\|00\rangle$ $\|01\rangle$ $\|01\rangle$ $\|10\rangle$ $\|11\rangle$ $\|11\rangle$ $\|10\rangle$

Tabela 2.1: Portas quânticas de 1 qbit (I, NOT, Y, Z, H e P) e 2 qbits (CNOT) com suas respectivas representações matriciais.

2.1.5 Circuitos e algoritmos quânticos

Os algoritmos quânticos normalmente são apresentadas esquematicamente através dos circuitos quânticos. Na apresentação do circuito, os qbits são representados pelas linhas horizontais e as portas são operações aplicadas nos qbits. Além disso, a leitura deve ser feita da esquerda para a direita e o estado inicial de cada qbit é denotado no início das linhas dos qbits. É possível observar um circuito quântico simples representado na figura 2.1.

Um algoritmo quântico é representado por um circuito quântico que consiste basicamente na aplicação de um conjunto de portas lógicas sobre qbits. A medidas são geralmente feitas no final do circuito. Combinando um conjunto mínimo de portas lógicas, é possível implementar qualquer operação unitária, ou seja, a partir de um número finito de portas é possível implementar qualquer algoritmo quântico.

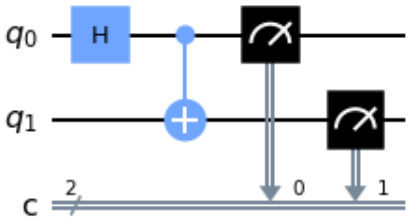


Figura 2.1: Exemplo de circuito quântico que cria um estado emaranhado definido como estado de Bell. Primeiro se aplica um porta Hadamard, depois aplica o $CNOT$ e posteriormente a medida.

2.2 Implementação de algoritmos no Qiskit

O kit de desenvolvimento de software para informação quântica, definido como Qiskit (*Quantum information software development kit*) é disponibilizado pela empresa IBM (*International Business Machines*). O Qiskit[3] é uma estrutura computacional de código aberto desenvolvida para funcionar em várias linguagens, dentre elas estão o Python, Swift e JavaScript. Para este trabalho foi utilizado exclusivamente a linguagem Python para a construção dos circuitos e simulações dos algoritmos quânticos.

Além disso, este código em Python pode ser interpretado no código de computadores quânticos usando acesso remoto a *hardwares* disponíveis de modo gratuito pelo IBM Q (plataforma da IBM Quantum Experience), sendo possível executar circuitos quânticos para obter resultados nos simuladores e nos computadores quânticos reais. Portanto, a ferramenta de desenvolvimento de programação quântica, ou Qiskit é composto pelos seguintes elementos:

- Terra: contém os elementos fundamentais utilizados para criação dos circuitos dos algoritmos quânticos, a estrutura do circuito;
- Aer: contém os recursos para simulações quânticas permitindo a simulação local dos processos quânticos;
- AQUA: contém bibliotecas de algoritmos pré programados;
- Ignis: contém as ferramentas para correção de erros e ruídos quânticos afim de melhorar a permormance e análise;
- IBM Q Provider: contém as ferramentas de acesso do IBM Q.

Tendo introduzido o Qiskit, agora vamos dar um exemplo de aplicação demonstrando alguns dos passos e funções fundamentais usados para implementar um algoritmo quântico. Inicialmente é preciso importar os pacotes que serão utilizados ao longo do algoritmo que está demonstrado pelos seguintes comandos:

```
from qiskit import *
from math import *
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
from qiskit.visualization import plot_histogram
```

Uma vez que importamos os pacotes temos todas as condições necessárias para iniciar a programação dos algoritmos para a execução nos computadores quânticos.

O próximo passo é definir as informações básicas, como o conjunto de qbits a ser utilizado na criação do circuito e o circuito em si conforme observado a seguir.

```
nq = 2
nc = 2
q = QuantumRegister( nq, name = 'q')
c = ClassicalRegister( nc, name = 'c')
qc = QuantumCircuit(q, c, name = 'Circuito_simples')
```

Nesses comandos, nq e nc representam respectivamente o número de qbits e bits do circuito. Vale lembrar que ambos são números inteiros e neste caso teremos a mesma quantidade porém não é uma característica sempre necessária. Portanto, com os bits e qbits definidos (linha 3 e linha 4), a última linha (linha 5) representa a definição do circuito quântico, onde q e c são os qbits e os bits clássicos do sistema.

A tarefa a seguir consiste na adição das portas lógicas e da medição ao final do circuito. Existe uma grande quantidade de portas quânticas portanto não serão definidas todas. Elas serão apresentadas ao decorrer de todo o projeto de acordo com seu uso em específico.

```
qc.h(0)
qc.cx(0,1)
qc.measure(q,c)
qc.draw('mpl')
```

As operações do circuito consiste na aplicação da porta Hadamard H no qbit 0 (linha 1) e de uma porta controlada X cujo qbit de controle é o qbit 0 e o qbit alvo é o qbit 1 (linha 2). Logo após (linha 3) é realizada a medida e dado o comando (linha 4) para visualizar o circuito. A execução deste comando pode ser observado na figura 2.1.

Com o circuito pronto resta apenas sua execução e visualização dos resultados conforme o exemplo abaixo:

```
shots = 2**11
# simulacao
simulador = Aer.get_backend('qasm_simulator')
simulacao = execute(qc,simulador, shots = shots).result()
# hardware real
IBMQ.load_account()
provider = IBMQ.get_provider(hub='ibm-q')
backend = least_busy(provider.backends(filters=lambda x: x.configuration().n_qubits
    >= nq and not x.configuration().simulator and x.status().operational==True))
```

```
job = execute(qc, backend=backend, shots=shots, optimization_level=3)
job_monitor(job)
real = job.result()
# plotando
legend = ['Simulacao', 'Real']
plot_histogram([simulacao.get_counts(), real.get_counts()], legend = ['Simulador','
    Hardware real'])
```

Existem vários métodos e maneiras de otimizar o processo estatístico afim de aproximar os valores emulados no processador quântico do resultado teórico. Dentre uma dessas maneiras é aumentar o número de repetições a serem realizadas de cada circuito definido como *shots*, quanto maior esse número, mais preciso é o resultado.

As linhas 3 e 4 representam a simulação numérica, ou seja, a simulação de um processador ideal. Esse provedor é definido como *'qasm_simulator'* e comporta até 32 qbits. Já nas linhas 6-11, apresentamos os comandos para executar o circuito em dispositivos quânticos reais, neste caso é preciso carregar uma conta pessoal da IBM. Existem vários provedores disponibilizados pela IBM para realização de processos através da computação em nuvem. Para a execução de vários processos no mesmo provedor é feita a criação de listas com ordem de prioridades. A linha 7 é o comando que seleciona o provedor com menos processos a serem realizados e logo após executa e retorna os resultados para o computador local. Também é possível escolher um determinado provedor que podem ser identificados através do site da IBM QE. E por último, nas linhas 13 e 14 são demonstrados os resultados obtidos graficamente, fazendo a comparação entre os dados obtidos pelo simulador e pelo *hardware* real, conforme observado na figura 2.2. Observa-se que, ao comparar os resultados,

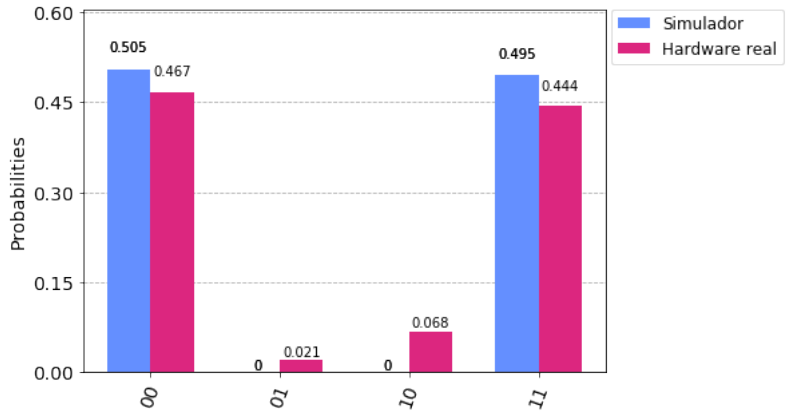


Figura 2.2: Comparação das medidas entre o simulador ideal e dispositivo quântico

o *hardware* real apresenta certa probabilidade nos estados $|01\rangle$ e $|10\rangle$, diferente do simulador ideal. Isso é comum e se da devido aos erros existentes nos dispositivos reais. Apesar disso os resultados são bastante consistentes com o valor teórico.

Nesta seção foi demonstrado apenas uma simples implementação no Qiskit afim de fazer uma introdução. Vale constar que a complexidade dos algoritmos quânticos podem ser bem maiores, conforme os algoritmos que serão desenvolvidos no restante do trabalho.

2.3 Sistemas Lineares Quânticos

Um sistema de equações lineares pode ser escrito com uma equação matricial dada por

$$Ax = b \tag{2.5}$$

onde $A \in R^{N \times N}$ e $x, b \in R^N$. Discutindo brevemente os sistema de equações lineares clássicos, eles geralmente são resolvidos através de duas técnicas: o método de Eliminação de Gauss e o método do Gradiente Conjugado[10]. O método de Eliminação de Gauss consiste na aplicação de uma sequência de operações elementares na matriz afim de preenchê-la com a maior quantidade de zeros possível, obtendo a chamada matriz triangular ou matriz escalonada do sistema. Dentre tais operações estão as trocas de linhas da matriz, a multiplicação da linha por um escalar e a adição do múltiplo de uma linha a outra. E para resolução de sistemas esparsos, o método clássico mais rápido é o método do Gradiente Conjugado que consiste na otimização do sistema de forma iterativa.

Assumindo o mapeamento dos vetores x e b respectivamente para os estados quânticos $|x\rangle$ e $|b\rangle$ tal que a i-ésima componente do vetor b corresponde a amplitude do i-ésimo termo do estado quântico $|b\rangle$ e de forma análoga para o vetor $|x\rangle$, podemos definir o análogo quântico do problema clássico. O problema de sistemas lineares quânticos é definido então como sendo o problema computacional para preparar o estado $|x\rangle$, tal que

$$A\,|x\rangle = |b\rangle\,. \tag{2.6}$$

Vale ressaltar que no problema quântico a saída da computação não são as soluções mas sim um estado quântico que contém uma superposição com todas as soluções possíveis. Para obter todas as soluções é preciso repetir o algoritmo muitas vezes, o que pode destruir qualquer possível ganho em relação ao algoritmos clássicos. Por isso algoritmos quânticos para sistemas lineares são utilizados para obter apenas informação estatística sobre as soluções ou como subrotinas de outros algoritmos. Note também que para a equação 2.6 ser válida a matriz A deve ser hermitiana.

Observando a equação (2.6), fazendo a decomposição espectral da matriz hermitiana A,

$$A = \sum_{j=1}^{N} \lambda_j\,|u_j\rangle\,\langle u_j| \tag{2.7}$$

onde $|u_j\rangle \in H^N$ são os $j-$ésimo autovetores de A e $\lambda_j \in R$ os respectivos autovalores. O lado direito da equação (2.6) pode ser escrito na base dos autovalores de A como

$$|b\rangle = \sum_{j=1}^{N} b_j\,|u_j\rangle\,. \tag{2.8}$$

Tomando a inversa da matriz

$$A^{-1} = \sum_{j=1}^{N} \lambda_j^{-1}\,|u_j\rangle\,\langle u_j|\,, \tag{2.9}$$

agora o sistema apresenta forma diagonal, cuja solução pode ser dada por

$$|x\rangle = A^{-1}\,|b\rangle = \sum_{j=1}^{N} \lambda_j^{-1} b_j\,|u_j\rangle\,. \tag{2.10}$$

Seria possível encontrar um algoritmo quântico para resolver o problema linear quântico? Ou seja, um algoritmo que prepare o estado $|x\rangle$. A resposta é sim, existem mais de uma forma de se fazer isso. A maneira proposta é feita através da decomposição espectral 2.10, porém é possível resolver o problema de várias maneiras possíveis.

2.4 Subrotinas Importantes

Nesta seção vamos discutir e mostrar a implementação de subrotinas essenciais que irão compor os algoritmos quânticos que serão discutidos posteriormente. Vale ressaltar que cada subrotina não possui relação com as outras, ou seja, são processos independentes.

2.4.1 Simulação Hamiltoniana

A evolução ou simulação Hamiltoniana obedece o postulado de Schrodinger

$$i\partial_t |\psi\rangle = H |\psi\rangle \tag{2.11}$$

onde $i^2 = -1$ representa a unidade imaginária e H é o operador Hamiltoniano. A solução quando H é independente do tempo da equação (2.11) representa a evolução temporal do estado $|\psi\rangle$ e é dada por

$$|\psi(t)\rangle = e^{-iHt} |\psi\rangle \tag{2.12}$$

onde o termo e^{-iHt} é unitário. Todo operador unitário pode ser escrito como $U = e^{iHt}$.

O problema de simulação quântica[11] consiste em simular a evolução de um sistema quântico em um computador quântico, ou seja, implementar a operação U. Para realizar essa tarefa precisamos decompor U em termos portas lógicas de maneira eficiente. Esse não é um problema simples e em muitos casos uma decomposição eficiente não exite.

No problema linear quântico, a matriz A precisa ser hermitiana e, portanto pode ser identificado como uma Hamiltoniana. No algoritmos HHL precisamos de técnicas de simulação para implementar a operação e^{iAt} para vários valores de t, esta simulação será usada na etapa de estimativa de fase.

2.4.2 Transformada de Fourier Quântica

A transformada de Fourier quântica (QFT - Quantum Fourier Transform)[12] é o análogo quântico da transformada discreta de Fourier. Seja uma série de valores complexos $x_0, x_1, ..., x_{N-1}$, a transformada de Fourier é definida como uma operação

$$x_0, x_1, ..., x_{N-1} \rightarrow y_0, y_1, ..., y_{N-1} \tag{2.13}$$

onde

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \; e^{\frac{2\pi i}{N} jk}. \tag{2.14}$$

Realçando que as bases de Fourier são ortogonais, ou seja, a transformada de Fourier é uma transformação unitária.

Seja uma superposição dos estados $|0\rangle, ..., |N-1\rangle$ com respectivas amplitudes $x_0, x_1, ..., x_{N-1}$. Podemos então definir que a QFT como a transformação unitária entre bases computacionais , conforme descrito abaixo

$$\sum_{j=0}^{N-1} x_j \, |j\rangle \rightarrow \sum_{k=0}^{N-1} y_k \, |k\rangle \, . \tag{2.15}$$

O valor assumido de N são potências de dois, $N = 2^n$ onde n são números naturais. As bases $|0\rangle, |1\rangle, ..., |N-1\rangle$ são estados da base de N qbits, ou seja, $|0\rangle \equiv |0...00\rangle, |1\rangle \equiv |0...01\rangle, |2\rangle \equiv |0...10\rangle, ..., |N-1\rangle \equiv |1...11\rangle$. De forma geral, cada estado $|j\rangle$ é escrito como

$$j = \underbrace{j_1 j_2 ... j_n}_{binário} = \underbrace{j_1 2^{n-1} + j_2 2^{n-2} + ... + j_n 2^0}_{decimal} \tag{2.16}$$

para todo $j = 0, 1, ..., N-1$ onde $j_k \in \{0, 1\}$ com $k = 0, 1, ..., n$. Através das notações representadas podemos reescrever a QFT de um estado base como

$$|j\rangle = |j_1 j_2 ... j_n\rangle \quad \rightarrow \quad \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{\frac{2\pi i}{N} jk} \, |k\rangle \tag{2.17}$$

e após algumas manipulações matemáticas

$$|j\rangle \rightarrow \frac{1}{2^{n/2}} \big(\, |0\rangle + e^{2\pi i \cdot (0 j_n)} \, |1\rangle \, \big) \otimes \big(\, |0\rangle + e^{2\pi i \cdot (0 j_{n-1} j_n)} \, |1\rangle \, \big) \otimes ... \otimes \big(\, |0\rangle + e^{2\pi i \cdot (0 j_1 j_2 ... j_n)} \, |1\rangle \, \big). \tag{2.18}$$

A representação dos estado de entrada e saída da QFT são mostrados na figura (2.3) onde fica aparente a diferenciação entre as bases computacionais e de Fourier.

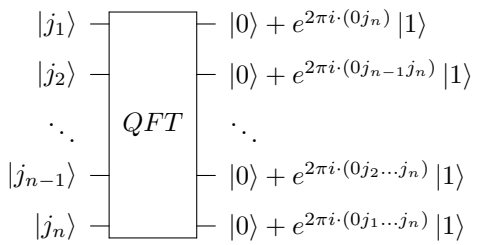


Figura 2.3: Diagrama de transformação de entrada e saída do QFT

Para a construção do circuito da QFT será necessário introduzir a porta unitária R_k definida como

$$R_k = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{pmatrix} \tag{2.19}$$

onde k são números naturais. Um circuito quântico pode ser construído para implementar o algoritmo da QFT conforme demonstrado na figura 2.4. Neste circuito a ordem da saída é dada de maneira inversa. Portanto, para retornar ao resultado demonstrado na 2.3 basta aplicar a porta *Swap* na saída dos qbits de maneira que q_0 troca de estado com q_{n-1}, q_1 troca de estado com q_{n-2}, q_2 troca de estado com q_{n-3} e assim por adiante.

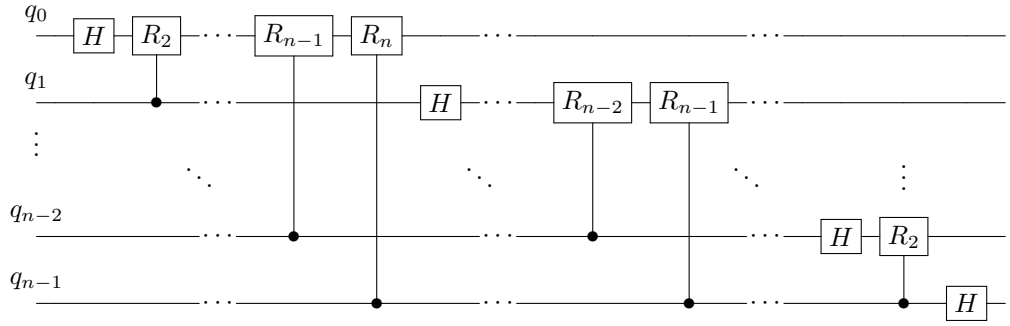


Figura 2.4: Circuito QFT

Além da QFT, para o desenvolvimento dos circuitos que serão estudados posteriormente vamos precisar da transforma quântica inversa de Fourier, representada por $\text{QFT}^\dagger$. A inversa da transformada de Fourier é implementada aplicando a inversa de cada porta quântica mostrada na figura 2.4 na ordem inversa.

Implementação

Nesta seção vamos descrever a implementação da QFT no Qiskit. A informação é armazenada de diferente maneira nas bases computacionais e de Fourier. Na base computacional a informação é armazenada inteiramente nos estados $|0\rangle$ ou $|1\rangle$ para cada qbit. Já na base de Fourier a informação em cada qbit é dada por rotações em diferentes ângulos sob o eixo Z da esfera de Bloch. Para

diferenciar o estado na base de Fourier da base computacional será utilizado o símbolo " $\sim$ " sobre a base de Fourier.

Na implementação do circuito (figura 2.5) vamos utilizar um total de 4 qbits ($n = 4$) onde o estado inicial é $|1010\rangle$. O estado inicial de cada um dos qbits é demonstrado na figura 2.6 pela esfera de Bloch, na base computacional. Vale ressaltar que a leitura da ordem dos qbits é inversa, isto é, os dígitos da base são colocados de forma contrária: $q_3 = 1,\ q_2 = 0,\ q_1 = 1$ e $q_0 = 0$.

Já a figura 2.7 mostra o estado dos qbits na base de Fourier, ou seja, após a aplicação do QFT. Transformando a base computacional para decimal temos que $|j\rangle = |1010\rangle_2 \equiv |10\rangle_{10}$.

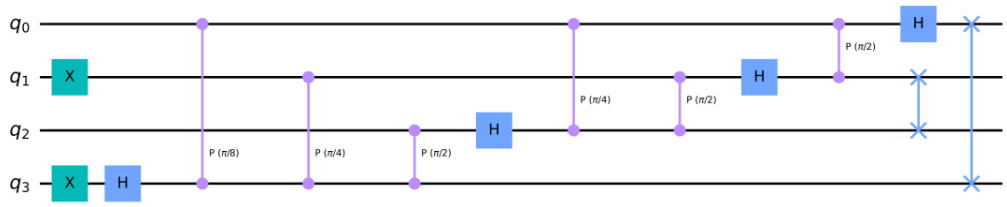


Figura 2.5: Circuito QFT com estado inicial $|1010\rangle$.

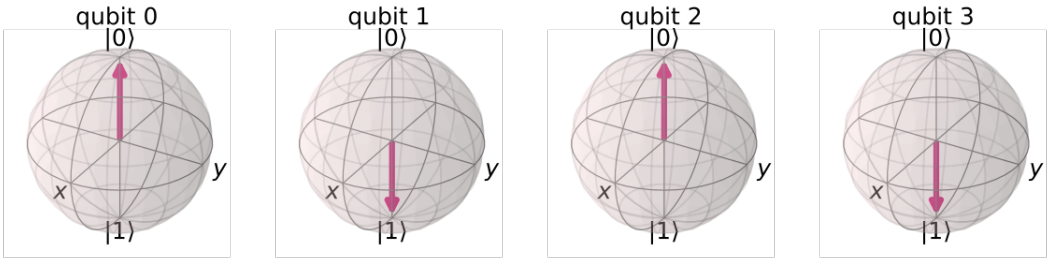


Figura 2.6: Visualização base computacional do estado $|1010\rangle$

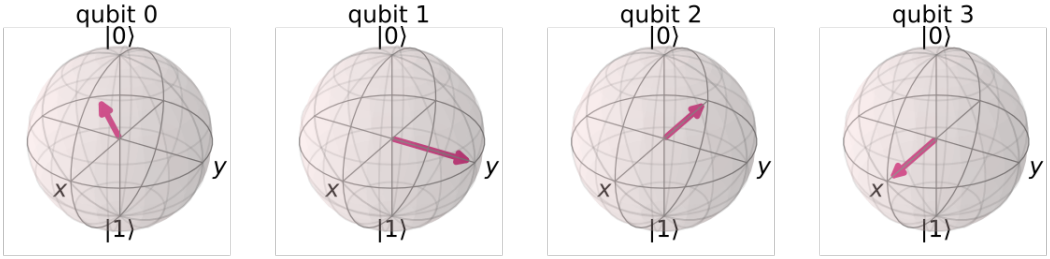


Figura 2.7: Visualização base de Fourier do estado $|\tilde{1010}\rangle$

Após a aplicação do QFT, o estado dos qbits é uma superposição entre os estado $|0\rangle$ e $|1\rangle$, ou seja, cada qbit é representado no plano x, y e rotacionado sobre o eixo Z. O número de voltas sobre tal

eixo é calculado conforme a equação $\frac{2^{nq} \times |j\rangle_{10}}{2^n}$ onde nq representa a ordenação do qbit ($q_0 = 0$, $q_1 = 1$, $q_2 = 2$, $q_3 = 3$ etc), $|j\rangle_{10}$ é a representação decimal da base computacional e n a quantidade de qbits. Com isso temos que q_0 foi rotacionado em $\frac{2^0 \times 10}{2^4} = \frac{10}{16}$ voltas, q_1 em $\frac{20}{16}$ voltas, q_2 em $\frac{40}{16}$ voltas e q_3 em $\frac{80}{16}$ voltas conforme pode ser observado na figura 2.7.

Afim de recuperar o resultado original dado na base computacional basta aplicar a transformada inversa ($\text{QFT}^\dagger$) com a entrada na base de Fourier $\left|\widetilde{1010}\right\rangle$ representado pelo circuito na figura 2.8. O resultado obtido na medição é dado pela figura 2.9 onde recupera- se o estado final $|1010\rangle$.

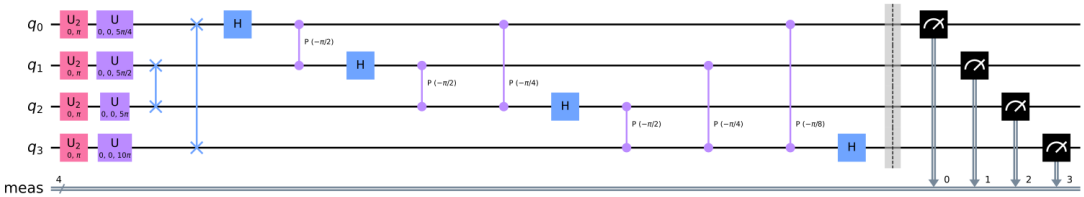


Figura 2.8: Circuito QFT Inversa com estado inicial $\left|\tilde{1010}\right\rangle$

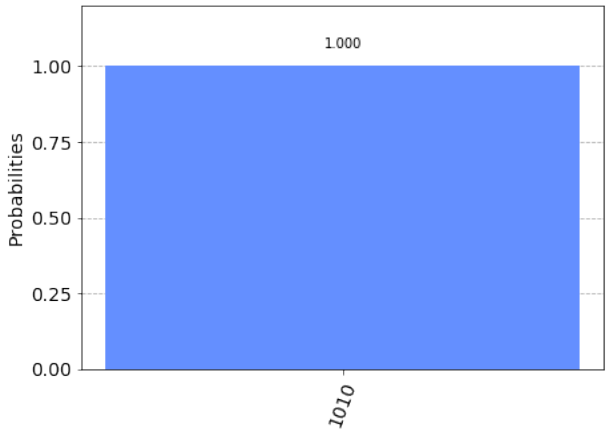


Figura 2.9: Medida do circuito QFT Inversa

2.4.3 Estimativa de Fase Quântica

A Estimativa de Fase Quântica (QPE - *Quantum Phase Estimation*)[13] é uma das subrotinas mais importantes para a computação quântica, servindo como fundamento central para a construção de muitos outros algoritmos quânticos além do HHL.

Dado um operador unitário U, o algoritmo estima o valor de φ_j onde

$$U\left|u_j\right\rangle = e^{2\pi i\varphi_j}\left|u_j\right\rangle. \tag{2.20}$$

Aqui $|u_j\rangle$ é um autovetor e $e^{2\pi i\varphi_j}$ é um autovalor do operador unitário U. Como todos os autovalores possuem norma igual a um, a QPE é capaz de estimar $\varphi_j \in [0, 1]$.

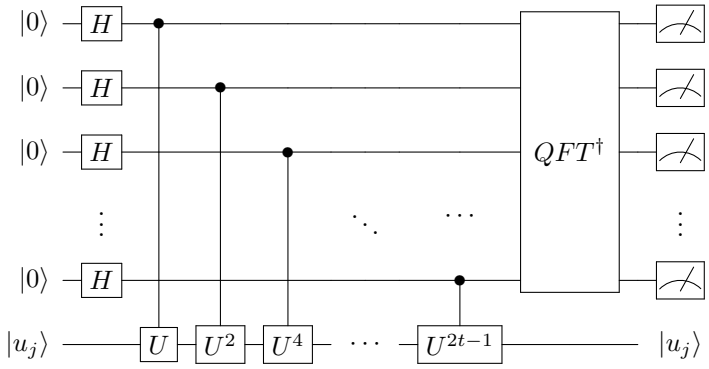


Figura 2.10: Circuito QPE

O circuito quântico do algoritmo de estimativa de fase está representado na figura (2.10). Ele possui dois tipos de registros: um de tamanho t preparado no estado $|0\rangle^{\otimes t}$ e outro com m qbits, inicializado no estado $|u_j\rangle$ que é o autovetor de U. A escolha de t determina o quão preciso será representado os autovalores.

O primeiro passo do circuito consiste em aplicar a porta Hadamard nos t qbits, ou seja,

$$|0\rangle^{\otimes t} \to \frac{1}{2^{t/2}}(|0\rangle + |1\rangle)^{\otimes t}. \tag{2.21}$$

O próximo passo é a aplicação das portas controladas U, onde U^{2^0} opera em $|u_j\rangle$ com o primeiro qbit de controle, U^{2^1} opera em $|u_j\rangle$ com o segundo qbit de controle e assim sucessivamente. Então,

$$U^{2^j}\left|u_j\right\rangle = U^{2^j-1}U\left|u_j\right\rangle = U^{2^j-1}e^{2\pi i\varphi}\left|u_j\right\rangle = ... = e^{2\pi i 2^j\varphi}\left|u_j\right\rangle. \tag{2.22}$$

Vale ressaltar que essa etapa é preciso aplicar as técnicas de simulação quântica como explicado na seção 2.4.1. Após a aplicação das t operações controladas temos o estado final

$$\frac{1}{2^{t/2}}\left(\left|0\right\rangle + e^{2\pi i 2^{t-1}\varphi}\left|1\right\rangle\right) \otimes ... \otimes \left(\left|0\right\rangle + e^{2\pi i 2^0\varphi}\left|1\right\rangle\right) = \frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi ik\varphi}\left|k\right\rangle \otimes \left|u_j\right\rangle \tag{2.23}$$

cuja equação identifica-se com a saída do algoritmo QFT (equação (2.18).

Dando seguimento ao circuito, agora é aplicado a QFT inversa. Aplicando-a é obtido o estado final

$$\frac{1}{2^{t/2}}\sum_{k=0}^{2^t-1} e^{2\pi i k\varphi}\,|k\rangle\otimes|u_j\rangle \to \frac{1}{2^t}\sum_{x=0}^{2^t-1}\sum_{k=0}^{2^t-1} e^{-\frac{2\pi i k}{2^t}(x-\varphi)}\,|x\rangle\otimes|u_j\rangle\,. \tag{2.24}$$

Analisando o somatório, quando a condição $\dfrac{x-\varphi}{2^t}=0$ for satisfeita vai existir somente amplitudes de probabilidades finitas onde $\sum_{k=0}^{2^t-1} e^0 = 2^t$. Caso contrário, devido a interferência destrutiva, a amplitude será $\sum_{k=0}^{2^t-1} e^{-\frac{2\pi i k}{2^t}(x-\varphi)} = 0$. E como último passo encontramos o estado final do circuito que consiste na medição dos t qbits

$$\frac{1}{2^t}\sum_{x=0}^{2^t-1}\sum_{k=0}^{2^t-1} e^{-\frac{2\pi i k}{2^t}(x-\varphi)}\,|x\rangle\otimes|u_j\rangle \to \left|2^t\varphi\right\rangle\otimes|u_j\rangle \tag{2.25}$$

onde o estado dos qbits auxiliares é a representação binária da fase φ com t dígitos.

Implementação

Como exemplo descreveremos aqui a implementação da QFT em dois casos distintos: no primeiro a fase pode ser escrita com apenas dois dígitos e na segunda a fase não possui representação binária finita.

As portas unitárias de 1-qbit podem ser decompostas fazendo uma parametrização através dos ângulos de rotações de Euler (θ,ϕ,λ) e uma fase global γ. Neste caso, a porta U será decomposta como $U = e^{i\gamma}U_3(\theta,\phi,\lambda) = Z(\phi)Y(\theta)Z(\lambda)$ onde

$$U_3(\theta,\phi,\lambda) = \begin{pmatrix} \cos(\theta) & -e^{i\lambda}\sin(\theta) \\ e^{i\phi}\sin(\theta) & e^{i(\phi+\lambda)}\cos(\theta) \end{pmatrix}, \tag{2.26}$$

Z e Y são as portas de rotação unitárias respectivamente do eixo Z e no eixo Y.

Portanto, o primeiro passo para a implementação é descobrir os valores das constantes θ,ϕ,λ e γ de acordo com o problema. Vamos utilizar duas matrizes A como exemplo,

$$A_1 = \frac{1}{4}\begin{pmatrix} \frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{pmatrix} \quad \text{e} \quad A_2 = \frac{1}{2}\begin{pmatrix} \frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{pmatrix} \tag{2.27}$$

onde $U = e^{2\pi i A}$.

Exemplo 1: Matriz A_1

De início é necessário descobrir os autovetores da matriz que serão usados como entrada conforme observado na figura 2.10. Diante disso, os autovetores da matriz serão

$$|u_1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad \text{e} \quad |u_2\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.28}$$

e, respectivamente, o valor de U ou autovalores serão

$$\lambda_1 = e^{2\pi i \frac{1}{4}} \quad \text{e} \quad \lambda_2 = e^{2\pi i \frac{1}{2}}. \tag{2.29}$$

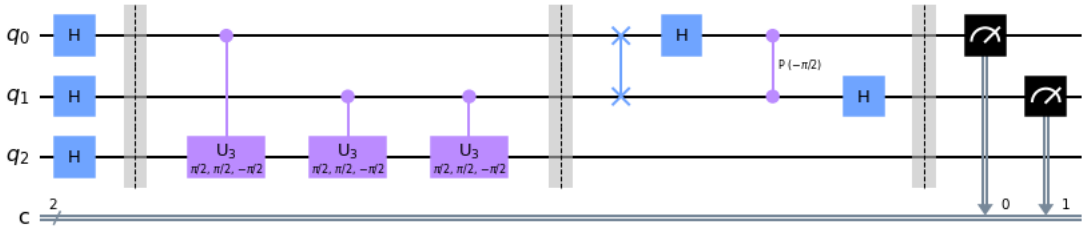


Figura 2.11: Circuito QPE da matriz A_1 com 2 qbits registradores

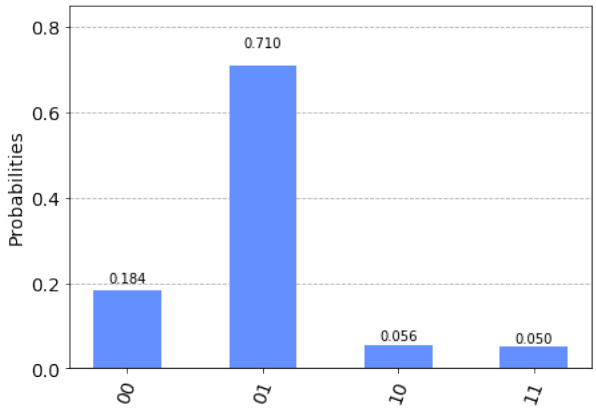


Figura 2.12: Medida referente ao circuito da figura 2.11

Utilizando $|u_2\rangle$ como entrada e dois qbits registros teremos o circuito dado pela figura 2.11 cujas medidas são demonstrados na figura 2.12, observa-se que a maior probabilidade de medida é dado no estado $|10\rangle$ (Obs: a medição dos qbits é invertida portanto é necessário inverter a ordem dos dígitos).

Portanto, de acordo com a equação 2.25, fazendo a medição nos $t = 2$ qbits do circuito encontramos que

$$\left|2^2\varphi\right\rangle = |k\rangle = |10\rangle_2 \equiv |2\rangle_{10} \tag{2.30}$$

onde $|k\rangle$ é a base de maior probabilidade medida. Representando o resultado na forma da equação 2.20,

$$U\,|u_2\rangle = e^{2\pi i \cdot \frac{2}{4}}\,|u_2\rangle \tag{2.31}$$

e fazendo a comparação encontramos $\varphi = \dfrac{1}{2}$. Tal resultado equivale ao resultado encontrado através de λ_2 (equação 2.29) pois utilizamos o autovetor $|u_2\rangle$. Apesar disso, será possível aumentar a probabilidade de medida ($P(|10\rangle) = 0,71$)? Para responder a pergunta vamos aumentar a quantidade de qbits registros para três conforme o circuito da figura 2.13.

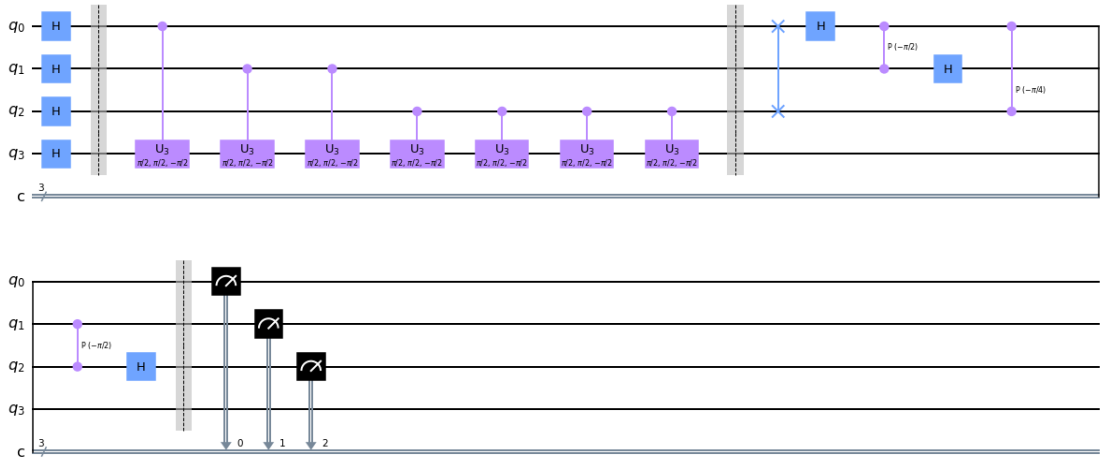


Figura 2.13: Circuito QPE da matriz A_1 com 3 qbits registradores

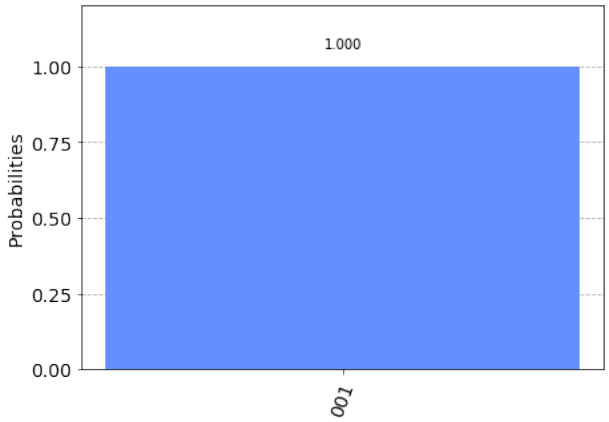


Figura 2.14: Medida referente ao circuito da figura 2.13

Ao aumentar o número de qbits também aumentou a precisão do circuito alcançando probabilidade de sucesso máxima ($P(|100\rangle) = 1$) conforme observado na figura 2.14. Vale ressaltar que novamente a fase obtida é exatamente a fase φ pois neste caso φ pode ser representada com apenas dois bits

$$|2^3\varphi\rangle = |100\rangle_2 \equiv |4\rangle_{10} \rightarrow \varphi = \frac{1}{2}. \tag{2.32}$$

Exemplo 2: Matriz A_2

Utilizando o mesmo processo do Exemplo 1 da matriz A_1, os autovetores da matriz A_2 serão idênticos

$$|u_1\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \quad \text{e} \quad |u_2\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \tag{2.33}$$

porém autovalores diferentes

$$\lambda_1 = e^{2\pi i \frac{1}{3}} \quad \text{e} \quad \lambda_2 = e^{2\pi i \frac{2}{3}}. \tag{2.34}$$

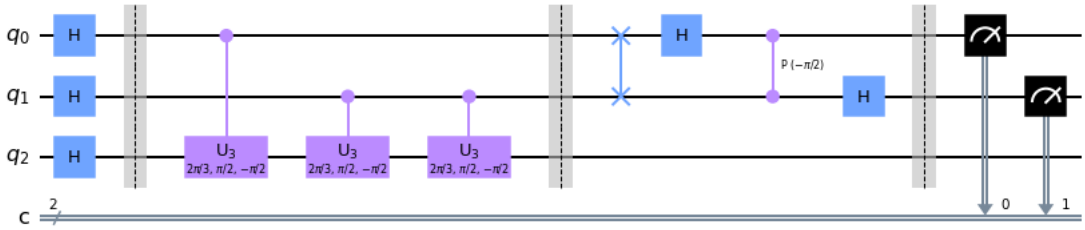


Figura 2.15: Circuito QPE da matriz A_2 com 2 qbits registradores

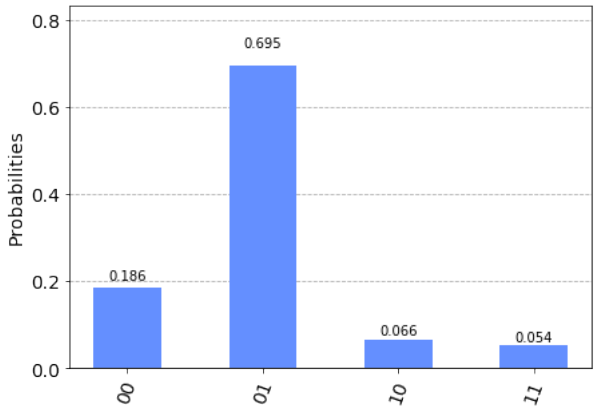


Figura 2.16: Medida referente ao circuito da figura 2.15

Implementando no circuito com entrada $|u_2\rangle$ e dois qbits registradores (figura 2.15) observa-se na medida (figura 2.16) com maior probabilidade ($P(|10\rangle) = 0.695$) dada por

$$|2^3\varphi\rangle = |10\rangle_2 \equiv |2\rangle_{10} \rightarrow \varphi = \frac{1}{2} \tag{2.35}$$

que possui um erro de aproximadamente 25% do valor esperado ($\varphi = \frac{2}{3} \approx 0.666$). Será possível melhorar o resultado? A resposta é sim, neste caso a acurácia do circuito aumenta com uma maior quantidade de qbits registradores.

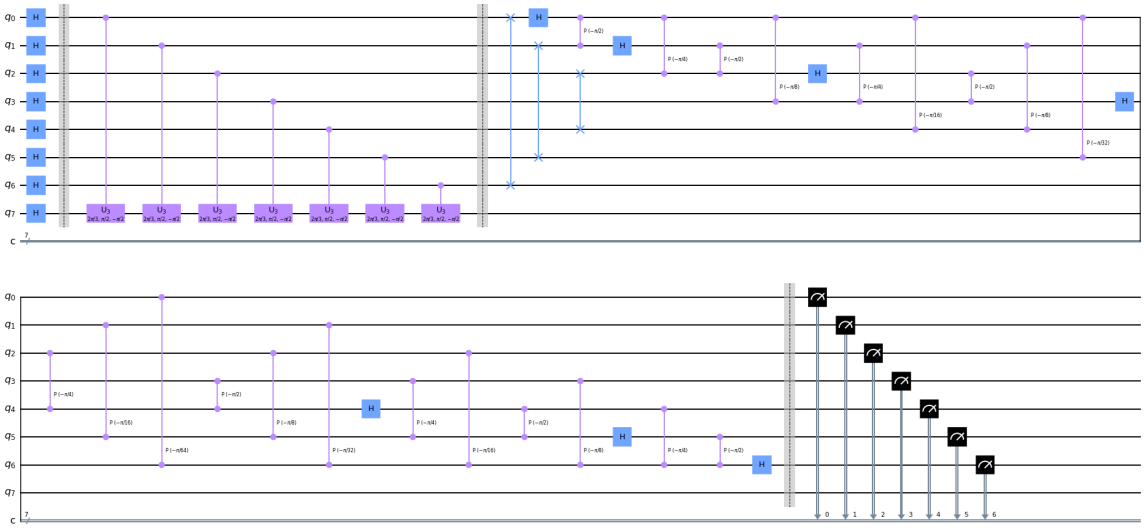


Figura 2.17: Circuito QPE da matriz A_2 com 7 qbits registradores

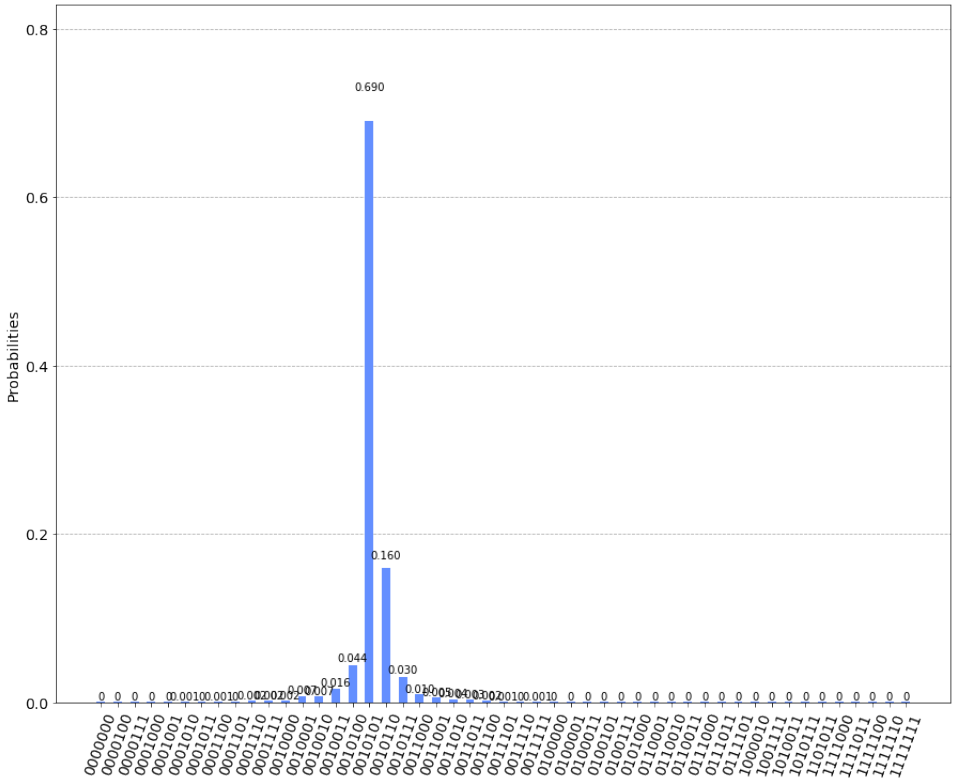


Figura 2.18: Medida referente ao circuito da figura 2.17

A figura 2.17 demonstra o mesmo exemplo porém utilizando 7 qbits registradores ($t = 7$) e pela medição final (figura 2.18) a maior probabilidade é dada por $P(|1010100\rangle) = 0.692$. Portanto a fase

é

$$\left|2^7\varphi\right\rangle = \left|1010100\right\rangle_2 \equiv \left|84\right\rangle_{10} \rightarrow \varphi = \frac{84}{128} \approx 0,656, \tag{2.36}$$

apresentando um erro aproximado de $1,5\%$, ou seja, uma precisão bem maior em comparação ao circuito utilizando apenas 2 qbits registradores.

2.5 Codificação de dados

Em um algoritmo para resolver sistemas lineares é preciso preparar o estado $|b\rangle$ de entrada, essa tarefa não é simples e pode comprometer a eficiência do algoritmo. Esta seção será utilizada para discutir os diferentes tipos de codificação dos dados.

Inicialmente vamos definir o estado quântico

$$\sum_{j=1}^{N} c_j |j\rangle \tag{2.37}$$

onde $\{c_j\}_{j=1}^{N}$ normalizado satisfaz $\sum_{j=1}^{N} |c_j|^2 = 1$. Neste caso os dados são codificados em quantidades analógicas, isto é, amplitudes complexas do estado quântico. Definindo assim a transformação unitária de codificação analógica $U_A(\{c_j\})$ como

$$U_A(\{c_j\}) |0\rangle \rightarrow \sum_{j=1}^{N} c_j |j\rangle . \tag{2.38}$$

No outro caso vamos codificar $m-$bits de dados binários no conjunto de qbits[14]. Seja N e $d_j = \{d_j^k\}_{k=1}^{m}$ o número de dados binários e conjunto de bits respectivamente. Aqui os dados são armazenados no estado de codificação digital dado por

$$\frac{1}{\sqrt{N}} \sum_{j=1}^{N} |j\rangle |d_j\rangle = \frac{1}{\sqrt{N}} \sum_{j=1}^{N} |j\rangle \left| d_j^m d_j^{m-1} \cdots d_j^2 d_j \right\rangle . \tag{2.39}$$

Portanto, definimos a transformação unitária de codificação digital $U_D(\{d_j\})$ como

$$U_D(\{d_j\}) |j\rangle |0\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{j=1}^{N} |j\rangle |d_j\rangle \tag{2.40}$$

onde U_D geralmente é chamado de QRAM (*Quantum Random Acess Memory*) [15][16][17][18]. No caso clássico, o algoritmo RAM (*Random Acess Memory*) recebe um endereço com o índice de memória e em seguida carrega os dados deste endereço de memória em um registrador de saída. A funcionalidade do QRAM é a mesma porém o endereço e o registro de saída são registradores quânticos apresentando a possibilidade de superposição de múltiplos valores. Os algoritmos para sistemas lineares usam em geral a codificação analógica, sendo possível passar da codificação analógica para digital através de um algoritmo quântico de conversão digital-analógica[19].

Capítulo 3

Algoritmos quânticos para sistemas lineares

3.1 Harrow, Hassidim Lloyd e implementação

O primeiro algoritmo quântico para sistemas lineares foi desenvolvido por Harrow, Hassidim e Lloyd e publicado em 2009[6] ficando conhecido como HHL. O algoritmo a ser discutido[20] utiliza as rotinas explicadas nas seções anteriores, portanto, para o algoritmo ser eficiente é importante existir uma maneira eficiente de simular a operação e^{iAt}. O HHL está mostrado na figura 3.1 e somente possui vantagem sobre algoritmos clássicos se a matriz A for hermitiana, esparsa e bem condicionada.

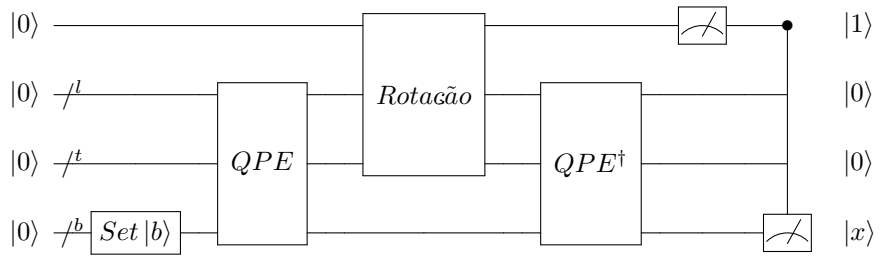


Figura 3.1: Circuito HHL

Na figura (3.1), observa-se que o algoritmo utiliza de um conjunto de registros quânticos, todos inicializados no estado $|0\rangle$. O registro b é utilizado para inicializar $|b\rangle$ e posteriormente é transformado no estado solução $|x\rangle$. O registro t e l serão responsáveis pelo armazenamento dos autovalores da matriz A e o primeiro qubit indicado sem índice serve apenas como qubit auxiliar.

Inicialmente, o estado $|b\rangle$ será inicializado no registro b. Seja $|b\rangle$ descrito como uma superposição dos autovetores $|u_j\rangle$ conforme descrito pela equação 2.8, logo

$$|0\rangle\,|0\rangle_l\,|0\rangle_t\,|0\rangle_b \rightarrow |0\rangle\,|00\rangle_{lt}\,|b\rangle_b = \sum_{j=1}^{N} b_j\,|0\rangle\,|00\rangle_{lt}\,|u_j\rangle_b\,. \tag{3.1}$$

Vale ressaltar que a maneira como $|b\rangle$ é inicializado não está prescrita pelo algoritmo porém possui fundamental importância. A eficiência do algoritmo está condicionada a existência de uma maneira eficiente de preparar o estado $|b\rangle$.

No segundo passo o algoritmo de estimativa de fase é aplicado relacionando U com a matriz A,

$$U|u_j\rangle = e^{iAt_0}|u_j\rangle = e^{i\lambda_j t_0}|u_j\rangle \tag{3.2}$$

onde t_0 é o tempo de evolução Hamiltoniana e λ_j é a estimativa da fase φ. Comparando com a equação 2.20 encontramos que $i\lambda_j t_0 = 2\pi i\varphi$ e escolhendo $t_0 = 2\pi/2^t$ temos $\varphi = \lambda_j/2^t$. Logo, após a aplicação da QPE, o registro t e l recebe os autovalores estimados λ_j de U, resultando na transformação

$$\sum_{j=1}^{N} b_j|0\rangle|00\rangle_{lt}|u_j\rangle_b \rightarrow \sum_{j=1}^{N} b_j|0\rangle|\lambda_j\rangle_{lt}|u_j\rangle_b\,. \tag{3.3}$$

A próxima etapa do circuito consiste na rotação no qubit auxiliar. Seja a porta unitária de rotação controlada $R_y(\theta)$ definida por

$$R_y(\theta) = \begin{pmatrix} \cos\left(\frac{\theta}{2}\right) & -\sin\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right) & \cos\left(\frac{\theta}{2}\right) \end{pmatrix}. \tag{3.4}$$

Essa subrotina de rotação tem como entrada o qubit auxiliar (registro sem índice) inicializado no estado $|0\rangle$ e os registros lt onde está mapeado o autovalor $|\lambda_j\rangle$. Aplicando a porta $R_y(\theta)$ no qubit auxiliar encontra-se

$$R_y(\theta)|0\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \sin\left(\frac{\theta}{2}\right)|1\rangle \tag{3.5}$$

entretanto é desejado uma rotação de θ tal que $\sin\left(\frac{\theta}{2}\right) = \frac{C}{\lambda_j}$ onde C é um valor real predeterminado que satisfaz a condição $|C| < 1$. Diante as condições temos que $\theta = 2\ \arcsin\left(\frac{C}{\lambda_j}\right)$. Portanto, rotacionando o qubit auxiliar no circuito observa-se a seguinte transformação

$$\begin{aligned} \sum_{j=1}^{N} b_j|0\rangle|\lambda_j\rangle_{lt}|u_j\rangle_b &\rightarrow \sum_{j=1}^{N} b_j\left[R_y\left(2\ \arcsin\left(\frac{C}{\lambda_j}\right)\right)|0\rangle\right]|\lambda_j\rangle_{lt}|u_j\rangle_b \\ &= \sum_{j=1}^{N} b_j\left[\sqrt{1-\frac{C^2}{\lambda_j^2}}|0\rangle + \frac{C}{\lambda_j}|1\rangle\right]|\lambda_j\rangle_{lt}|u_j\rangle_b\,. \end{aligned} \tag{3.6}$$

Seguindo as etapas do circuito, o próximo passo é a decomputação. Ela consiste em resetar os registros l e t aplicando as operações inversas das subrotinas performadas, causando o desemaranhamento dos registros l, t e b produzido pela QPE. Portanto basta aplicar $QPE^\dagger$, resultando em

$$\sum_{j=1}^{N} b_j\left[\sqrt{1-\frac{C^2}{\lambda_j^2}}|0\rangle + \frac{C}{\lambda_j}|1\rangle\right]|\lambda_j\rangle_{lt}|u_j\rangle_b \rightarrow \sum_{j=1}^{N} b_j\left[\sqrt{1-\frac{C^2}{\lambda_j^2}}|0\rangle + \frac{C}{\lambda_j}|1\rangle\right]|00\rangle_{lt}|u_j\rangle_b\,. \tag{3.7}$$

Finalmente, fazendo a medição do qubit auxiliar a função de onda colapsa para o estado $|0\rangle$ ou $|1\rangle$. No caso em que a medida for $|0\rangle$ o resultado é descartado e a computação é repetida até encontrar o estado $|1\rangle$. Quando $|1\rangle$ for obtido, o estado final $|x\rangle$ é preparado no registro b conforme detalhado a seguir,

$$\sum_{j=1}^{N} b_j \left[\sqrt{1-\frac{C^2}{\lambda_j^2}}\,|0\rangle + \frac{C}{\lambda_j}\,|1\rangle\right] |0\rangle_l\,|0\rangle_t\,|u_j\rangle_b \to \sum_{j=1}^{N} b_j \left[\frac{C}{\lambda_j}\,|1\rangle\right] |0\rangle_l\,|0\rangle_t\,|u_j\rangle_b \tag{3.8}$$

$$\propto |1\rangle\,|0\rangle_l\,|0\rangle_t \sum_{j=1}^{N} \frac{b_j}{\lambda_j}\,|u_j\rangle_b = |1\rangle\,|0\rangle_l\,|0\rangle_t\,|x\rangle_b\,. \tag{3.9}$$

Um fator importante na performance do algoritmo HHL esta ligado aos fatores κ e s. O fator κ consiste na razão entre o maior e o menor autovalor de A. Quando tal número condicional κ cresce, A se torna mais próximo a uma matriz que não pode ser invertida e a solução se torna menos estável, mais sensível a pertubações. Esta matriz é dita bem condicionada e o algoritmo assume valores de A entre $\{\frac{1}{\kappa}, \kappa\}$. Já o fator s está ligado à esparsidade da matriz. Ele consiste na quantidade de elementos não-nulos da matriz A por linha. Além disso existe o fator ϵ que é o erro dado pela distância entre o estado de saída do circuito e estado ideal $|x\rangle$.

A complexidade do HHL, $O\left(\frac{s^2\kappa^2}{\epsilon}\log(N)\right)$ comparando com o melhor caso do algoritmo clássico, $O(N\kappa s\log(1/\epsilon))$ é exponencialmente mais rápido no tamanho da matriz porém mais lento para matrizes mal condicionadas e não esparsas.

Implementação

Como exemplo, nesta seção apresentamos a simulação do algoritmo HHL no Qiskit utilizando das subrotinas já implementadas e discutidas. Nesta implementação em específico o circuito (figura 3.2) foi otimizado para resolver qualquer problema no formato onde

$$A = \begin{pmatrix} a & b \\ b & a \end{pmatrix} \quad \text{e} \quad |b\rangle = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \tag{3.10}$$

tal que a, b, θ são reais.

Vale discutir também que há diferentes maneiras de extrair o resultado (vetor solução $|x\rangle$) do circuito. A solução desejada é encontrada quando a leitura do último qubit for $|1\rangle$, o segundo e o terceiro qubit for $|0\rangle$ e o primeiro ou $|0\rangle$ ou $|1\rangle$. Mediante isso, queremos apenas os resultados apresentados diante os estados finais $|0001\rangle$ ou $|1001\rangle$.

Diante as questões discutidas será utilizada a seguinte matriz como exemplo

$$A = \begin{pmatrix} \frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{pmatrix} \tag{3.11}$$

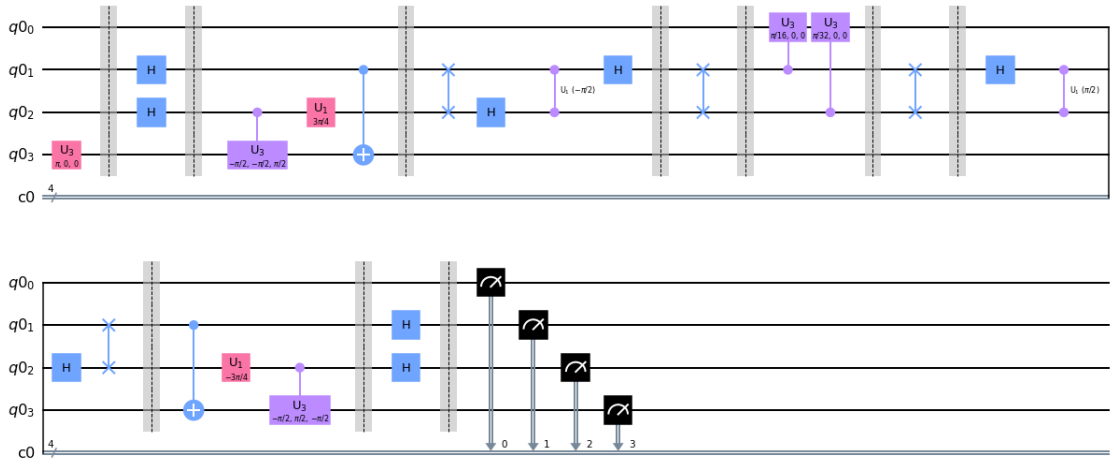


Figura 3.2: Circuito HHL

e para $|b\rangle$, o valor de θ será modificado de acordo com cada simulação. Será utilizado quatro valores distintos para $\theta = \{\pi, \pi/2, -0.57, -2.23\}$ porém o resultado obtido pelo sistema é um vetor não normalizado, por exemplo, para $\theta = \pi$ a solução é $|x\rangle = \begin{pmatrix} -0,250 \\ 0,750 \end{pmatrix}$. Logo, o resultado deve ser obtido pelo estado normalizado, ou seja, $|x\rangle_{norm} = \begin{pmatrix} -0,316 \\ 0,949 \end{pmatrix}$. Na tabela 3.1 e na figura 3.3 são demonstradas as saídas e soluções para o exemplo.

θ	π	$\pi/2$	$-2,23$	$-0,57$
Teoria	$\begin{pmatrix} -0,316 \\ 0,949 \end{pmatrix}$	$\begin{pmatrix} 0,707 \\ 0,707 \end{pmatrix}$	$\begin{pmatrix} 0,555 \\ -0,816 \end{pmatrix}$	$\begin{pmatrix} 0,868 \\ -0,496 \end{pmatrix}$
Simulação	$\begin{pmatrix} -0,316 \pm 0\% \\ 0,949 \pm 0\% \end{pmatrix}$	$\begin{pmatrix} 0,707 \pm 0\% \\ 0,707 \pm 0\% \end{pmatrix}$	$\begin{pmatrix} 0,577 \pm 3,8\% \\ -0,817 \pm 0,1\% \end{pmatrix}$	$\begin{pmatrix} 0,869 \pm 0,1\% \\ -0,495 \pm 0,2\% \end{pmatrix}$

Tabela 3.1: Resultados com erros relativos referentes ao vetor solução $|x\rangle$ para diferentes valores de θ

Observando os resultados da tabela 3.3 pode-se dizer que os valores da simulação em comparação a teoria são muito próximos após a normalização. O maior erro relativo encontrado foi na simulação de $theta = -2,23$ com erro igual a $3,8\%$.

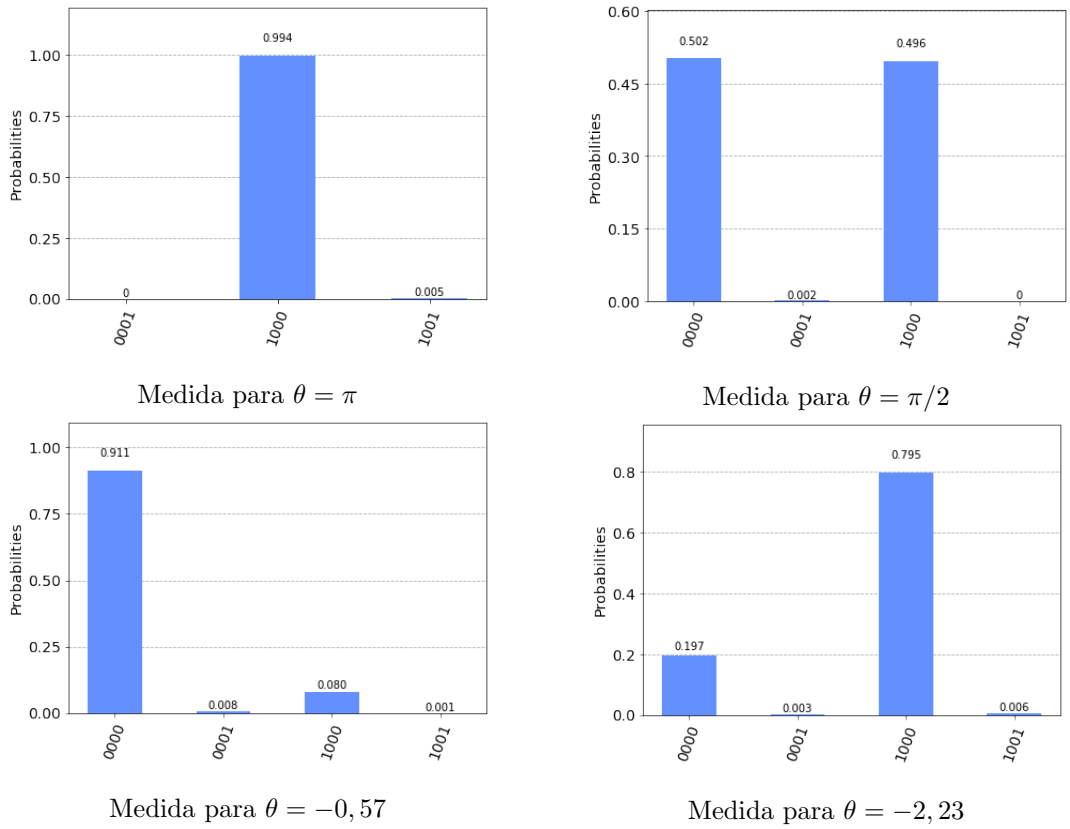


Figura 3.3: Medidas referentes ao circuito HHL para diferentes valores de θ

3.2 Childs, Kothari Somma e implementação

Conforme discutido na seção anterior, o algoritmo HHL é um procedimento eficiente para resolução de sistemas lineares quânticos porém sua performance é limitada pela estimativa de fase, que utiliza de operadores unitários para estimar seus autovalores com precisão ϵ. Portanto sua complexidade é dada por $O(1/\epsilon)$.

Afim de reduzir o erro de dependência da estimativa de fase foi proposto um modelo que consiste na variação do algoritmo HHL. O algoritmo a ser discutido a seguir foi desenvolvido por Childs, Kothari e Somma[7] onde será definido como algoritmo CKS. Ele consiste basicamente em substituir a estimativa de fase por algum método de aplicação direta do inverso da matriz, melhorando a precisão do algoritmo exponencialmente. O circuito do algoritmo CKS pode ser observado na figura 3.4.

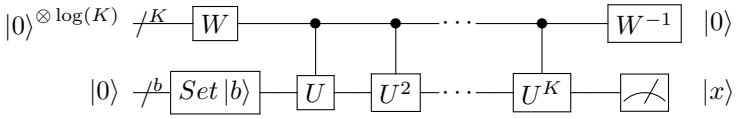


Figura 3.4: Circuito CKS

Neste caso, a inicialização de $|b\rangle$ no registro l obedece o mesmo princípio do algoritmo HHL. Já a matriz A deve ser decomposta numa combinação linear de K matrizes unitárias U_k,

$$A = \sum_{k=1}^{K} \alpha_k U_k \tag{3.12}$$

onde α são os coeficientes da expansão. A partir dos coeficientes da expansão podemos criar a porta W. Ela representa o circuito que cria uma superposição a partir de $\log(K)$ qubits no estado $|0\rangle$ mediante a seguinte transformação,

$$|0\rangle^{\otimes \log(K)} \rightarrow \frac{\sum_k \alpha_k |k\rangle}{\sum_k ||\alpha_k||^2}. \tag{3.13}$$

Com W definido, as matrizes unitárias U_k são aplicadas em $|b\rangle$ tendo como controle o estado da equação 3.13. Após isso, o registro b tem como resultado a inversa da matriz A sobre o estado $|b\rangle$, ou seja, $A^{-1}|b\rangle$ que é o resultado desejado. Por fim, fazendo a medição em tal registro temos o estado de saída $|x\rangle$.

Como exemplo, desenvolvendo o circuito W no caso particular para matrizes reais $A_{2\times 2}$ onde apenas duas constantes de expansão são suficientes. O circuito unitário será construído utilizando a porta Hadamard e uma porta de fase $P(\phi)$ definida por

$$P(\phi) = \begin{pmatrix} 1 & 0 \\ 0 & \exp^{i\phi} \end{pmatrix}. \tag{3.14}$$

Neste caso em específico a formação do circuito W consiste na aplicação de H no estado inicial $|0\rangle$

$$|0\rangle \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \tag{3.15}$$

logo após a aplicação da porta $P(\phi)$

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + \exp^{i\phi}|1\rangle), \tag{3.16}$$

e novamente a aplicação da porta Hadamard

$$\frac{1}{\sqrt{2}}(|0\rangle + \exp^{i\phi}|1\rangle) \rightarrow \frac{1}{2}[(1+\exp^{i\phi})|0\rangle + (1-\exp^{i\phi})|1\rangle)]. \tag{3.17}$$

Com isso, após a aplicação de W os coeficientes da expansão alfas serão $\alpha_1(|1\rangle) = \frac{1}{2}[1 - cos(\phi)]$, que no circuito CKS controla a aplicação da porta U_1 e $\alpha_2(|0\rangle) = \frac{1}{2}[1 + cos(\phi)]$, que corresponde a aplicação da porta U_2.

Implementação

Como exemplo de implementação, vamos definir a matriz A tal que

$$A = \begin{pmatrix} 1,5 & 0,5 \\ 0,5 & 1,5 \end{pmatrix} \quad \text{e} \quad |b\rangle = \begin{pmatrix} cos(2\pi/3) \\ sen(2\pi/3) \end{pmatrix}. \tag{3.18}$$

Utilizaremos apenas duas constantes α_k e duas matrizes unitárias U_k na expansão da matriz onde U_1: porta X e U_2: porta $U3$.

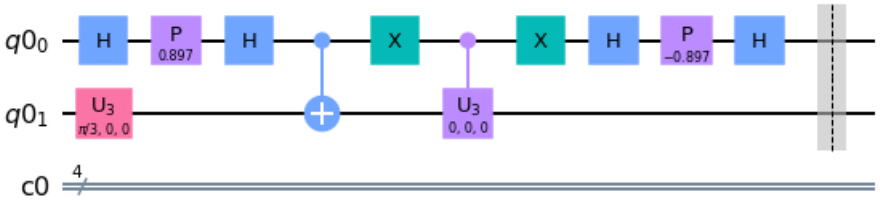


Figura 3.5: Circuito CKS implementado

A figura 3.5 demonstra o circuito implementando a matriz A e $|b\rangle$ cujo resultado final corresponde ao estado final

$$|x\rangle = \begin{pmatrix} 0,569 \\ -0,797 \end{pmatrix}. \tag{3.19}$$

Comparando com o resultado teórico onde $|x\rangle_{teoria} = \begin{pmatrix} 0,607 \\ -0,795 \end{pmatrix}$ obtemos uma fidelidade de aproximadamente 94%. O cálculo das fidelidades são definidos como produto interno entre os resultados teóricos e medidos.

Além do circuito CKS é importante dizer que existem outras variações do HHL. O HHL usa como base a decomposição espectral da matriz A (equação 2.7) e o algoritmo de estimativa de fase para estimar os autovalores. Por isso não é apenas o erro que fica limitado, pois a implementação da porta $U = e^{iAt}$ só é possível ser feita de maneira eficiente se A for uma matriz esparsa, assim o HHL também e sensivel ao parâmetro s.

O algoritmo proposto por Wossnig et. al[21] em 2018 usa como base a decomposição de A em termos de seus valores singulares. Tal algoritmo substitui a estimativa de fase quântica por um algoritmo quântico de estimativa de valores singulares. O algoritmo linear quântico resultante é independente da esparcidade. Outro ponto importante é o condicionamento da matriz. Quando A é mal condicionada, a probabilidade de sucesso do algoritmo diminui, ou seja, precisamos repetir o algoritmo mais vezes para encontrar a resposta. Em 2013, A. Ambainis[22] propôs adicionar a etapa final do HHL um algoritmo de "Amplificação de Amplitude", dessa maneira é possível reduzir a dependência em k de k^2 para $k\log_3(k)$.

3.3 *Variational quantum linear solver* e implementação

Os algoritmos explicados no capítulo anterior requerem uma grande quantidade de qubits e que o erro por parte seja extremamente baixo para funcionar. Assim métodos variacionais que utilizam algoritmos híbridos, isto é, utilizam métodos computacionais clássicos e quânticos para a resolução de sistemas lineares tem sido propostos recentemente como por exemplo o algoritmo VQLS[8].

A estrutura do VQLS (*Variational Quantum Linear Solver*) é similar a qualquer algoritmo de minimização, ou seja, consiste em minimizar certos parâmetros a partir de uma função definida como função custo. Neste caso os parâmetros definem uma sequência de portas lógicas quânticas e a minimização da função custo é feito através de otimizador clássicos. A saída do circuito VQLS é análoga ao HHL, porém por possuir o caráter variacional, permite ser utilizado nos computadores quânticos NISQ[23] enquanto o HHL requer mais qubits e necessita que o computador quântico seja tolerante a erro, ou seja, uma capacidade maior dos *hardwares* quânticos ainda não disponível atualmente.

O termo NISQ (*Noisy Intermediate-Scale Quantum*) se refere aos computadores quânticos atuais em que o *hardware* é limitado no número de qubits disponíveis e a profundidade do circuito é limitada pelo aumento do ruído que surge dentro de um sistema quântico. Como tal, muitos dos algoritmos quânticos mais eminentes (como os mencionados acima) têm aplicações práticas limitadas, levando os pesquisadores a se concentrarem no desenvolvimento de algoritmos quânticos amigáveis ao NISQ.

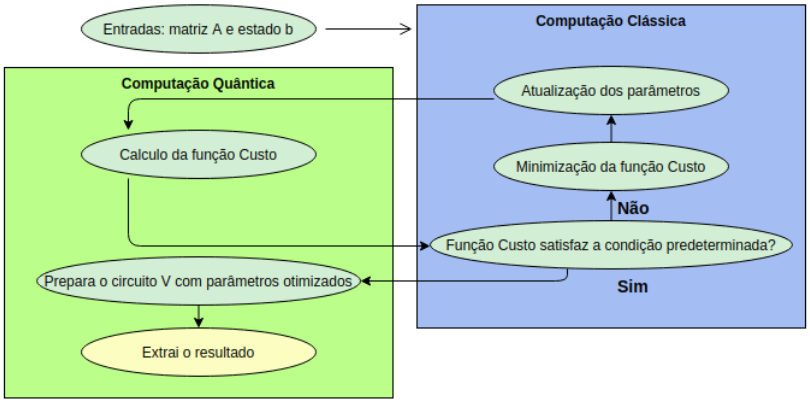


Figura 3.6: Diagrama VQLS

Conforme observado na figura 3.6, a entrada do algoritmo recebe a matriz $2^n \times 2^n$ A que é denotada como a combinação linear de L matrizes unitárias $A_1, A_2, ..., A_L$,

$$A = \sum_{l=1}^{L} c_l A_l \tag{3.20}$$

onde c_l são números complexos arbitrários.

Inicialmente, será inicializado o estado $|b\rangle$. Importante ressaltar que essa etapa também foi

necessária em todos os algoritmos mencionados no capítulo anterior. Assumindo um estado quântico $|b\rangle$, é necessário que tal estado seja eficientemente preparado por um circuito quântico, representado pela operação unitária U, ou seja,

$$|b\rangle = U|0\rangle . \tag{3.21}$$

Após a preparação inicial de $|b\rangle$ faremos uma tentativa de preparar o estado $|x\rangle$ aplicando o circuito V dependente de parâmetros reais clássicos $\alpha = \alpha_1, \alpha_2, ...$ que são entradas do computador quântico. Diante o ansatz $V(\alpha)$, será preparado uma potencial solução dada por

$$|x(\alpha)\rangle = V(\alpha)|0\rangle \tag{3.22}$$

e através de um circuito quântico estima-se a função custo $C(\alpha)$. A função custo quantifica a projeção entre $AV(\alpha)|0\rangle$ e $|b\rangle$, sendo a função mínima quando $AV(\alpha)|0\rangle$ está próximo a $|b\rangle$ e máxima quando $AV(\alpha)|0\rangle$ é ortogonal a $|b\rangle$. Mais detalhes sobre a função custo serão discutidos posteriormente.

O valor de $C(\alpha)$ retorna ao computador clássico que ajusta os parâmetros α através de uma otimização clássica afim de reduzir a função custo. Tal processo é repetido várias vezes até que o custo seja menor que um determinado parâmetro γ ($C(\alpha) \leq \gamma$). Ao atingir dada condição, os parâmetros estão otimizados. Neste momento vamos atribuir então que

$$\alpha = \alpha_{opt}. \tag{3.23}$$

Através dos parâmetros otimizados é preparado o estado quântico final

$$|x(\alpha_{opt})\rangle = V(\alpha_{opt})|0\rangle . \tag{3.24}$$

Essa é a saída do algoritmo. Podemos observar que tal saída é similar a saída do HHL, porém aqui temos uma aproximação. Diante o contexto geral do algoritmo VQLS apresentado, vamos analisar suas subrotinas.

Função custo

Conforme dito anteriormente, a função custo quantifica quanto a componente $A|x(\alpha)\rangle$ tem de ortogonalidade com $|b\rangle$, ou seja, se $|\psi\rangle = A|x(\alpha)\rangle$ está proporcionalmente próximo de $|b\rangle$.

Introduzindo a projeção Hamiltoniana como

$$H = I - |b\rangle\langle b| \tag{3.25}$$

a função custo pode ser observada como o valor esperado da Hamiltoniana

$$C = \langle\psi| H |\psi\rangle = \langle\psi| (I - |b\rangle\langle b|) |\psi\rangle = \langle\psi|\psi\rangle - \langle\psi|b\rangle\langle b|\psi\rangle \tag{3.26}$$

Normalizando a função,

$$\hat{C} = \frac{C}{\langle\psi|\psi\rangle} = \frac{\langle\psi|\psi\rangle}{\langle\psi|\psi\rangle} - \frac{\langle\psi|b\rangle\langle b|\psi\rangle}{\langle\psi|\psi\rangle} = 1 - \frac{|\langle b|\psi\rangle|^2}{\langle\psi|\psi\rangle} \tag{3.27}$$

onde a função será mínima quando $|\psi\rangle$ está próximo a $|b\rangle$ (valor de $\langle\psi|b\rangle$ próximo de um) e máxima quando $|\psi\rangle$ é ortogonal a $|b\rangle$ (valor de $\langle\psi|b\rangle$ igual a zero).

Portanto, observando a função custo devemos encontrar uma forma de calcular os seguintes termos: $|\langle b|\psi\rangle|^2$ e $\langle\psi|\psi\rangle$. Para tal, existe uma subrotina quântica que permite fazer isso que é definida como Circuito Teste Hadamard que será discutido com mais detalhes mais adiante. Para estimar cada um dos termos são desenvolvidos diferentes circuito quânticos utilizando o circuito teste Hadamard, e, somente após o recebimento dos valores dos termos é calculado a função custo.

Fazendo uma avaliação dos termos de forma independente, desenvolvendo inicialmente $\langle\psi|\psi\rangle$ temos que

$$\begin{aligned}
\langle\psi|\psi\rangle &= \langle x|A^\dagger A|x\rangle \quad \text{onde } |x\rangle = V|0\rangle \text{ e } A = \sum_l c_l A_l && (3.28)\\
&= \langle 0|V^\dagger A^\dagger A V|0\rangle && (3.29)\\
&= \langle 0|V^\dagger \left(\sum_l c_l A_l\right)^\dagger \left(\sum_l c_l A_l\right) V|0\rangle && (3.30)\\
&= \sum_{l,l'} c_l c_{l'}^* \langle 0|V^\dagger A_{l'}^\dagger A_l V|0\rangle && (3.31)
\end{aligned}$$

e para o outro termo,

$$\begin{aligned}
|\langle b|\psi\rangle|^2 &= |\langle b|A|x\rangle|^2 \quad \text{onde } |b\rangle = U|0\rangle \ , \ |x\rangle = V|0\rangle \text{ e } A = \sum_l c_l A_l && (3.32)\\
&= |\langle 0|U^\dagger A V|0\rangle|^2 && (3.33)\\
&= \langle 0|U^\dagger A V|0\rangle \langle 0|V^\dagger A^\dagger U|0\rangle && (3.34)\\
&= \langle 0|U^\dagger \left(\sum_l c_l A_l\right) V|0\rangle \langle 0|V^\dagger \left(\sum_l c_l A_l\right)^\dagger U|0\rangle && (3.35)\\
&= \sum_{l,l'} c_l c_{l'}^* \langle 0|U^\dagger A_l V|0\rangle \langle 0|V^\dagger A_{l'}^\dagger U|0\rangle && (3.36)
\end{aligned}$$

porém, na implementação do circuito (que será discutido mais adiante) a saída será de valores reais pois os resultados são os valores esperados. Portanto

$$\langle 0|U^\dagger A V|0\rangle = (\langle 0|U^\dagger A V|0\rangle)^* = \langle 0|V^\dagger A^\dagger U|0\rangle \tag{3.37}$$

logo,

$$|\langle b|\psi\rangle|^2 = \sum_{l,l'} c_l c_{l'}^* \langle 0|U^\dagger A_l V|0\rangle \langle 0|U^\dagger A_{l'}^\dagger V|0\rangle . \tag{3.38}$$

A avaliação dos termos é exclusivamente feita para implementação do circuito teste Hadamard demonstrando as relações que devem ser aplicadas conforme será explicado a seguir.

Circuito Teste Hadamard

Como discutido anteriormente, o circuito Teste Hadamard é uma subrotina quântica que permite implementar o cálculo dos termos da função custo. Basicamente, dado um circuito unitário F e algum estado quântico $|\phi\rangle$, o teste Hadamard permite encontrar o valor esperado de F em relação ao estado $|\phi\rangle$, ou seja, $\langle\phi| F |\phi\rangle$.

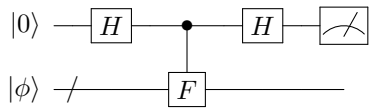


Figura 3.7: Circuito Teste Hadamard

Para demonstrar o funcionamento do circuito vamos seguir o diagrama da figura 3.7 por partes. Inicialmente aplica-se a Hamard no primeiro qubit:

$$|0\rangle \otimes |\phi\rangle \rightarrow \frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |\phi\rangle = \frac{|0\rangle \otimes |\phi\rangle}{\sqrt{2}} + \frac{|1\rangle \otimes |\phi\rangle}{\sqrt{2}} \tag{3.39}$$

Logo após aplica-se o circuito controlado unitário F:

$$\frac{|0\rangle \otimes |\phi\rangle}{\sqrt{2}} + \frac{|1\rangle \otimes |\phi\rangle}{\sqrt{2}} \rightarrow \frac{|0\rangle \otimes |\phi\rangle}{\sqrt{2}} + \frac{|1\rangle \otimes F |\phi\rangle}{\sqrt{2}} \tag{3.40}$$

E novamente a porta Hadamard no primeiro qubit:

$$\begin{aligned} \frac{|0\rangle \otimes |\phi\rangle}{\sqrt{2}} + \frac{|1\rangle \otimes F |\phi\rangle}{\sqrt{2}} &\rightarrow \frac{|0\rangle \otimes |\phi\rangle + |1\rangle \otimes |\phi\rangle + |0\rangle \otimes F |\phi\rangle - |1\rangle \otimes F |\phi\rangle}{2} \\ &= \frac{|0\rangle}{2} \otimes (I + F) |\phi\rangle + \frac{|1\rangle}{2} \otimes (I - F) |\phi\rangle \end{aligned} \tag{3.41}$$

Fazendo a medição no primeiro qubit, a probabilidade de obter o estado $|0\rangle$ será dada por

$$\begin{aligned} P(0) &= \langle\phi| \frac{(I+F)}{2} \frac{(I+F^\dagger)}{2} |\phi\rangle = \langle\phi| \frac{(\overbrace{I^2}^{=I} + F + F^\dagger + \overbrace{FF^\dagger}^{=I})}{4} |\phi\rangle \\ &= \frac{1}{4}(\langle\phi| 2I |\phi\rangle + \langle\phi| F |\phi\rangle + \langle\phi| \overbrace{F^\dagger}^{=F^*} |\phi\rangle) \\ &= \frac{1}{2}(1 + \mathrm{Re}\,\langle\phi| F |\phi\rangle) \end{aligned} \tag{3.42}$$

e fazendo o processo análogo, a probabilidade de obter o estado $|1\rangle$ será

$$P(1) = \frac{1}{2}(1 + \mathrm{Re}\,\langle\phi| F |\phi\rangle). \tag{3.43}$$

Tomando a diferença entre as probabilidades encontramos a saída esperada

$$P(0) - P(1) = \mathrm{Re}\,\langle\phi| F |\phi\rangle). \tag{3.44}$$

Portanto, para obter o valor esperado de qualquer circuito unitário F em relação ao estado $|\phi\rangle$, basta aplicar o circuito teste Hadamard e tomar a diferença das probabilidades dada pela equação 3.44. Por exemplo, implementando o circuito teste Hadamard conseguimos calcular o termo $\langle\psi|\psi\rangle$ mediante as seguintes relações:

$$\begin{aligned} \langle\psi|\psi\rangle &= \langle 0| V^\dagger A^\dagger A V |0\rangle & (3.45) \\ &= \langle\phi| F |\phi\rangle \quad \text{, ou seja,} \quad |\phi\rangle = |0\rangle \quad \text{e} \quad F = V^\dagger A^\dagger A V & (3.46) \end{aligned}$$

cujo circuito quântico é demonstrado na figura 3.8.

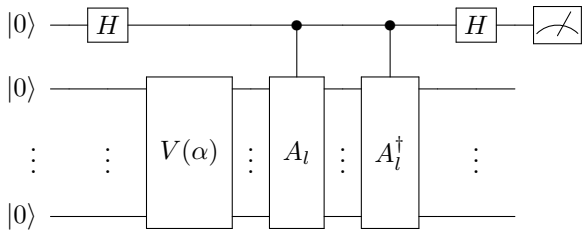


Figura 3.8: Circuito quântico usado para computar o termo $\langle\psi|\psi\rangle$.

Fazendo o processo análogo, para o termo $\langle b|\psi\rangle$ encontramos as relações:

$$\begin{aligned} \langle b|\psi\rangle &= \langle 0| U^\dagger A V |0\rangle & (3.47) \\ &= \langle\phi| F |\phi\rangle \quad \text{, ou seja,} \quad |\phi\rangle = |0\rangle \quad \text{e} \quad F = U^\dagger A V & (3.48) \end{aligned}$$

cujo circuito é demonstrado na figura 3.9.

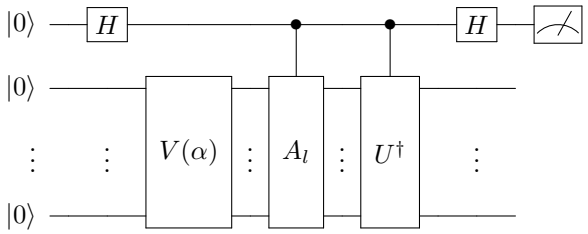


Figura 3.9: Circuito quântico usado para computar o termo $\langle b|\psi\rangle$.

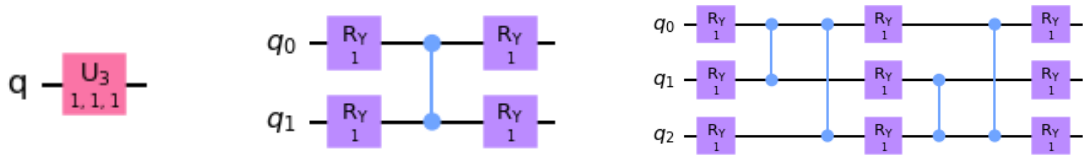
Implementação

Conforme demonstrado pela equação 3.20, o algoritmo VQLS recebe a matriz A que é decomposta em uma combinação linear de unitários A. Na implementação, a configuração de A_n vai corresponder a combinação linear de dois diferentes tipos de portas unitárias. Por exemplo

$$A = 0,3 \cdot I + 0,5 \cdot Z_2 + 0,2 \cdot Z_3 \tag{3.49}$$

onde I é a porta identidade, Z a porta Z. Os qubits que as portas são aplicadas são representados pelos seus índices. Por exemplo, Z_2 significa que a porta Z foi aplicada no terceiro qubit.

O Ansatz $V(\alpha)$ será uma sequência de portas que irá preparar a potencial solução $|x\rangle$ a partir de $|0\rangle$. Existem várias maneiras de criar essa estrutura porém na nossa implementação vamos utilizar uma estrutura fixa que varia somente os parâmetros α das portas de rotação e não o circuito em si. Vale ressaltar que cada estrutura deve conseguir rotacionar todos os valores dentro da esfera de Bloch.



(a) (b) (c)

Figura 3.10: Circuito fixo do Ansatz $V(\alpha)$ para diferentes tamanhos de matrizes. (a) Matriz 2×2; (b) Matriz 4×4; (c) Matriz 8×8.

Além disso, para a função custo precisa-se calcular os termos $\langle\psi|\psi\rangle$ e $\langle b|\psi\rangle$ cujo exemplos de circuitos podem ser observados na figura 3.11 e 3.12.

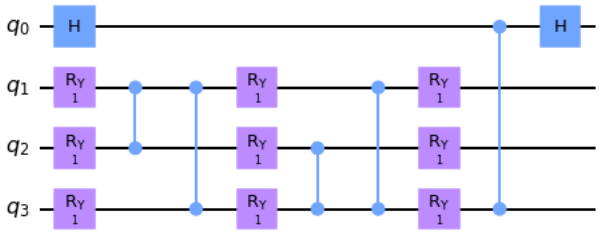


Figura 3.11: Exemplo circuito de implementação para calcular $\langle\psi|\psi\rangle$ de uma matriz $A_{8\times8}$

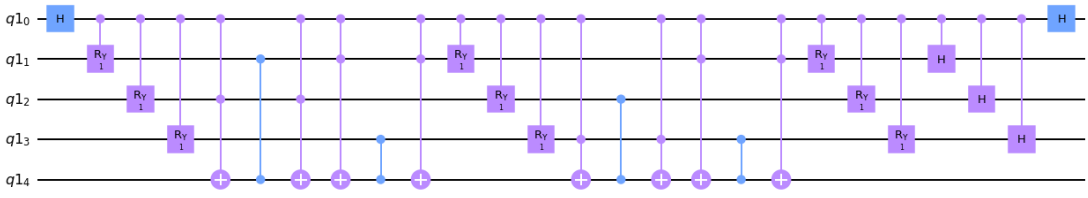


Figura 3.12: Exemplo de circuito de implementação para calcular $\langle b|\psi\rangle$ de uma matriz $A_{8\times8}$ com $b = H_1H_2H_3$

Com todos os termos computados, a função custo é calculada. Por meio da computação clássica é feito o processo de minimização dos parâmetros da função custo. Para toda a implementação será utilizado o método COBYLA como o método de minimização clássico com um número máximo de iterações sendo 200.

Após a iteração, os parâmetros otimizados são obtidos e usados novamente na computação quântica para a criação do circuito Ansatz final. Portanto, o último passo é construir o circuito Ansatz com os parâmetros obtidos (α_{opt}) e fazer a medição do estado final afim de obter o vetor solução $|x\rangle$.

Sabendo todo o processo da implementação, agora vamos extrair alguns resultados das simulações. Para isso será utilizado alguns exemplos com diferentes tamanhos da matriz A apresentados a seguir. Vale ressaltar que tais exemplos a seguir não foram executados nos *hardwares* reais, são simulações.

Exemplo 1: Matriz $A_{2\times2}$

Seja

$$A = 1,5 \cdot I + 0,5 \cdot X_1 \quad \text{e} \quad |b\rangle = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}. \tag{3.50}$$

O processo de minimização atingiu o número máximo de iterações e o valor final da função custo foi de $7,00535363051813e-09$. O circuito ansatz com os parâmetros otimizados é demonstrado na figura 3.13.

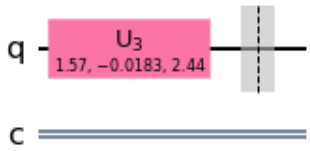


Figura 3.13: Ansatz $V(\alpha_{opt})$ referente ao Exemplo 1

O vetor solução final é obtido através da simulação e pelo hardware quântico real, ambos resultados são demonstrados na tabela a seguir.

	Teoria	Simulação	Real
$\|x\rangle$	$\begin{pmatrix} 0,70710 \\ 0,70710 \end{pmatrix}$	$\begin{pmatrix} 0,70710 \\ 0,70710 \end{pmatrix}$	$\begin{pmatrix} 0,71320 \\ 0,70093 \end{pmatrix}$
Fidel.	1	1	0,999

Tabela 3.2: Resultados obtidos pela matriz 2×2 na simulação e no computador quântico real.

Os valores representam um bom resultado com fidelidade de $99,9\%$ da execução no computador real diante do valor teórico.

Exemplo 2: Matriz $A_{4\times4}$

Seja

$$A = 0,4 \cdot I + 0,6 \cdot Z_2 \quad \text{e} \quad |b\rangle = \begin{pmatrix} 1/2 \\ 1/2 \\ 1/2 \\ 1/2 \end{pmatrix}. \tag{3.51}$$

Neste caso teremos o número de iteração alcançado em 108 com a função custo sendo $6,592253409820614e{-}09$.

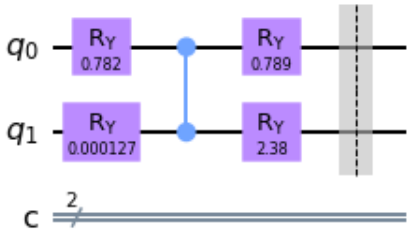


Figura 3.14: Ansatz $V(\alpha_{opt})$ referente ao Exemplo 2

A estrutura do algoritmo do Ansatz otimizado pode ser observado na figura 3.14, cujo vetor solução é demonstrado na tabela 3.3.

	Teoria	Simulação	Real
$\|x\rangle$	$\begin{pmatrix} 0,13867 \\ 0,13867 \\ -0,69337 \\ -0,69337 \end{pmatrix}$	$\begin{pmatrix} 0,13848 \\ 0,13922 \\ -0,69682 \\ -0,68983 \end{pmatrix}$	$\begin{pmatrix} -0,16563 \\ -0,10152 \\ 0,73207 \\ 0,65293 \end{pmatrix}$
Fid.	1	0,999	0,994

Tabela 3.3: Resultados obtidos pela matriz 4×4 na simulação e no computador quântico real.

Exemplo 3: Matriz $A_{8\times8}$

Seja a matriz A dada pela seguinte combinação linear

$$A = 0,7 \cdot I + 0,3 \cdot Z_3 \tag{3.52}$$

e $|b\rangle$ definido como

$$|b\rangle = \begin{pmatrix} 1/2\sqrt{2} \\ 1/2\sqrt{2} \\ 1/2\sqrt{2} \\ 1/2\sqrt{2} \\ 1/2\sqrt{2} \\ 1/2\sqrt{2} \\ 1/2\sqrt{2} \\ 1/2\sqrt{2} \end{pmatrix} \tag{3.53}$$

Alguns valores do processo iterativo são mostrados na seguinte tabela

N^o Iteração	Função custo
1	0,8759892988984588
50	0,09836948063178164
100	0,007283569445728388
150	0,006105271713996996
200_{max}	0,0055462866558434465

onde é possível observar a contínua redução da função custo conforme o maior número de iterações. Com os parâmetros otimizados, a estrutura do ansatz é demonstrada na figura 3.15,

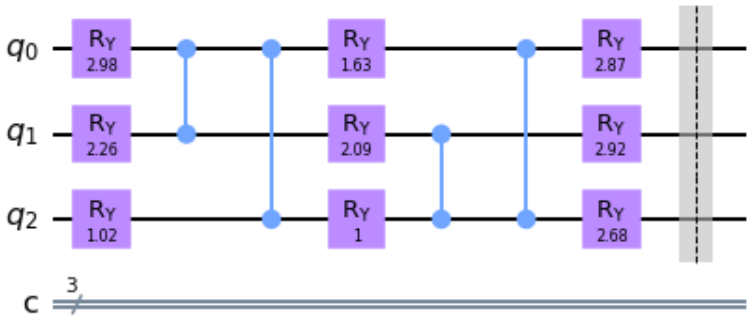


Figura 3.15: Ansatz $V(\alpha_{opt})$ referente ao Exemplo 3

cujo vetor solução é demonstrado na tabela 3.4.

O resultado da simulação em relação ao valor teórico continua com uma alta fidelidade de $99,9\%$ e, com relação ao valor obtido do dispositivo real houve uma pequena redução para $93,5\%$.

	Teoria	Simulação	Real
$\|x\rangle$	$\begin{pmatrix} 0,18569 \\ 0,18569 \\ 0,18569 \\ 0,18569 \\ 0,46423 \\ 0,46423 \\ 0,46423 \\ 0,46423 \end{pmatrix}$	$\begin{pmatrix} -0,18576 \\ -0,18581 \\ -0,18641 \\ -0,18550 \\ -0,46153 \\ -0,46597 \\ -0,46558 \\ -0,46355 \end{pmatrix}$	$\begin{pmatrix} 0,31626 \\ 0,27343 \\ 0,28011 \\ 0,29992 \\ 0,39309 \\ 0,44474 \\ 0,41995 \\ 0,35792 \end{pmatrix}$
Fid.	1	0,999	0,935

Tabela 3.4: Resultados obtidos pela matriz 8×8 na simulação e no computador quântico real.

Exemplo 4: Matriz $A_{16\times16}$

Seja a matriz A dada pela seguinte combinação linear

$$A = 0,7 \cdot I + 0,3 \cdot Z_4 \tag{3.54}$$

e $|b\rangle$ definido como

$$|b\rangle = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{pmatrix} \tag{3.55}$$

O processo de minimização atingiu novamente o número máximo de iterações. O valor final da função custo foi de $0,0018007215234696616$ cujo circuito V otimizado é demonstrado na figura 3.16.

Os resultados obtidos podem ser observados na tabela 3.5 e continuam com alta fidelidade onde a menor fidelidade obtido foi, novamente no *hardware* real. Portanto, o algoritmo VQLS utilizado neste projeto possui boa confiabilidade e, conforme aumenta a dimensão e/ ou diminui o condicionamento das matrizes do problema, tende a diminuir a fidelidade dos resultados executados no computador real quântico.

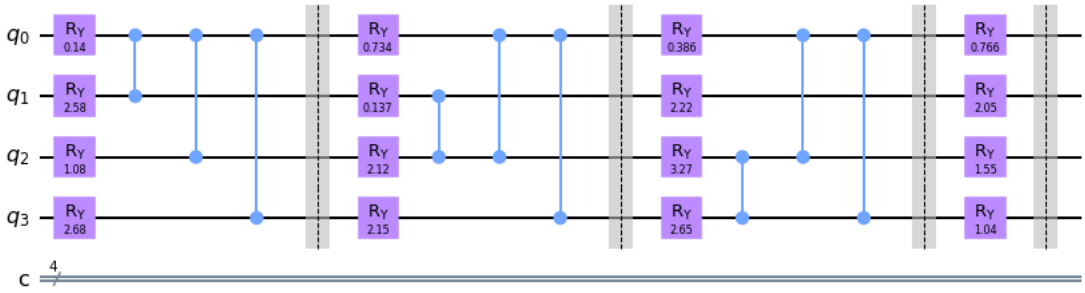


Figura 3.16: Ansatz $V(\alpha_{opt})$ referente ao Exemplo 4

	Teoria	Simulação	Real
$\|x\rangle$	$\begin{pmatrix} 0,13130 \\ 0,13130 \\ 0,13130 \\ 0,13130 \\ 0,13130 \\ 0,13130 \\ 0,13130 \\ 0,13130 \\ 0,32826 \\ 0,32826 \\ 0,32826 \\ 0,32826 \\ 0,32826 \\ 0,32826 \\ 0,32826 \\ 0,32826 \end{pmatrix}$	$\begin{pmatrix} 0,13247 \\ 0,13451 \\ 0,12297 \\ 0,13166 \\ 0,13165 \\ 0,12197 \\ 0,13358 \\ 0,13924 \\ 0,32232 \\ 0,34618 \\ 0,32024 \\ 0,33883 \\ 0,33039 \\ 0,32548 \\ 0,34162 \\ 0,29922 \end{pmatrix}$	$\begin{pmatrix} 0,11172 \\ 0,08760 \\ 0,14037 \\ 0,09009 \\ 0,11679 \\ 0,11247 \\ 0,16945 \\ 0,05820 \\ 0,30963 \\ 0,30258 \\ 0,36322 \\ 0,34569 \\ 0,31554 \\ 0,34755 \\ 0,35175 \\ 0,33264 \end{pmatrix}$
Fid.	1	0,998	0,984

Tabela 3.5: Resultados obtidos pela matriz 16×16 na simulação e no computador quântico real.

3.4 Comparação entre os algoritmos

Afim de descobrir o melhor método dentre os estudados neste trabalho vamos fazer uma comparação de resultados resolvendo o mesmo problema utilizando os diferentes algoritmos de solução: HHL, CKS e VQLS. Para quantificar a precisão calculamos a fidelidade definida como produto interno entre o valor teórico e o valor medido. O problema foi resolvido dada uma matriz 2×2 como definida a seguir:

$$A = \begin{pmatrix} \frac{3}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{3}{2} \end{pmatrix} \quad \text{e} \quad |b\rangle = \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix}. \tag{3.56}$$

Variamos os valores de θ segundo valores predeterminados $\theta \in \{0, \pi/4, \pi/3, \pi/2, \pi, 2\pi/3\}$ e, além de implementar simulações (cujos resultados são indicados pelos índices "sim), o problema também foi executado no *hardware* quântico real (indicadas pelos índices "$real$).

$\theta = 0$	$\|x\rangle$	Fid.
Teoria	$\begin{pmatrix} 0,949 \\ -0,316 \end{pmatrix}$	1
HHL_{sim}	$\begin{pmatrix} 0,948 \\ -0,316 \end{pmatrix}$	0,9995
HHL_{real}	$\begin{pmatrix} 0,810 \\ -0,585 \end{pmatrix}$	0,9535
VQLS_{sim}	$\begin{pmatrix} 0,948 \\ -0,316 \end{pmatrix}$	0,9995
VQLS_{real}	$\begin{pmatrix} 0,973 \\ -0,230 \end{pmatrix}$	0,9960
CKS_{sim}	$\begin{pmatrix} 0,975 \\ -0,226 \end{pmatrix}$	0,9967
CKS_{real}	$\begin{pmatrix} 0,886 \\ -0,460 \end{pmatrix}$	0,9862

Tabela 3.6: Resultados obtidos com $\theta = 0$.

$\theta = \pi/4$	$\|x\rangle$	Fid.
Teoria	$\begin{pmatrix} 0,836 \\ -0,548 \end{pmatrix}$	1
HHL_{sim}	$\begin{pmatrix} 0,836 \\ -0,549 \end{pmatrix}$	0,9997
HHL_{real}	$\begin{pmatrix} 0,757 \\ -0,653 \end{pmatrix}$	0,9906
VQLS_{sim}	$\begin{pmatrix} 0,835 \\ -0,549 \end{pmatrix}$	0,9989
VQLS_{real}	$\begin{pmatrix} 0,844 \\ -0,535 \end{pmatrix}$	0,9987
CKS_{sim}	$\begin{pmatrix} 0,862 \\ -0,508 \end{pmatrix}$	0,9990
CKS_{real}	$\begin{pmatrix} 0,850 \\ -0,520 \end{pmatrix}$	0,9955

Tabela 3.7: Resultados obtidos com $\theta = \pi/4$.

$\theta = \pi/3$	$\|x\rangle$	Fid.
Teoria	$\begin{pmatrix} 0,795 \\ -0,607 \end{pmatrix}$	1
HHL_{sim}	$\begin{pmatrix} 0,794 \\ -0,607 \end{pmatrix}$	0,9996
HHL_{real}	$\begin{pmatrix} 0,791 \\ -0,611 \end{pmatrix}$	0,9997
VQLS_{sim}	$\begin{pmatrix} 0,795 \\ -0,607 \end{pmatrix}$	1
VQLS_{real}	$\begin{pmatrix} 0,795 \\ -0,605 \end{pmatrix}$	0,9992
CKS_{sim}	$\begin{pmatrix} 0,798 \\ -0,601 \end{pmatrix}$	0,9992
CKS_{real}	$\begin{pmatrix} 0,865 \\ -0,498 \end{pmatrix}$	0,9899

Tabela 3.8: Resultados obtidos com $\theta = \pi/3$.

$\theta = \pi/2$	$\|x\rangle$	Fid.
Teoria	$\begin{pmatrix} 0,707 \\ -0,707 \end{pmatrix}$	1
HHL_{sim}	$\begin{pmatrix} 0,707 \\ -0,707 \end{pmatrix}$	1
HHL_{real}	$\begin{pmatrix} 0,673 \\ -0,739 \end{pmatrix}$	0,9982
VQLS_{sim}	$\begin{pmatrix} 0,707 \\ -0,707 \end{pmatrix}$	1
VQLS_{real}	$\begin{pmatrix} 0,709 \\ -0,704 \end{pmatrix}$	0,9989
CKS_{sim}	$\begin{pmatrix} 0,707 \\ -0,707 \end{pmatrix}$	1
CKS_{real}	$\begin{pmatrix} 0,722 \\ -0,686 \end{pmatrix}$	0,9954

Tabela 3.9: Resultados obtidos com $\theta = \pi/2$.

Agora vamos analisar os resultados demonstrados nas tabelas 3.6 - 3.11. Comparando os valores finais ($|x\rangle$) com o valor teórico calculamos a fidelidade. Quanto mais próximo de 1 o valor da fidelidade, mais preciso é o resultado.

Dentre todos os resultados, as piores fidelidades encontradas foram nos métodos HHL_{real} para $\theta = 0$ onde a fidelidade é igual a $0,9535$ e no CKS_{sim} para $\theta = \pi/4$ onde a fidelidade é igual a $0,9967$. E dentre todos diferentes métodos utilizados, o que apresentou os melhores resultados foi o método $VQLS$. Ressaltando que a fidelidade para o VQLS é sempre maior que $0,99$, evidenciando seu caráter mais robusto enquanto que nos outros casos a fidelidade é menor.

$\theta = \pi$	$\|x\rangle$	Fid.
Teoria	$\begin{pmatrix} -0,316 \\ 0,949 \end{pmatrix}$	1
HHL_{sim}	$\begin{pmatrix} -0,316 \\ 0,949 \end{pmatrix}$	1
HHL_{real}	$\begin{pmatrix} -0,555 \\ 0,832 \end{pmatrix}$	0,9649
VQLS_{sim}	$\begin{pmatrix} -0,316 \\ 0,949 \end{pmatrix}$	1
VQLS_{real}	$\begin{pmatrix} -0,309 \\ 0,944 \end{pmatrix}$	0,9935
CKS_{sim}	$\begin{pmatrix} -0,226 \\ 0,975 \end{pmatrix}$	0,9967
CKS_{real}	$\begin{pmatrix} -0,495 \\ 0,867 \end{pmatrix}$	0,9792

Tabela 3.10: Resultados obtidos com $\theta = \pi$.

$\theta = 2\pi/3$	$\|x\rangle$	Fid.
Teoria	$\begin{pmatrix} 0,607 \\ -0,795 \end{pmatrix}$	1
HHL_{sim}	$\begin{pmatrix} 0,607 \\ -0,794 \end{pmatrix}$	0,9996
HHL_{real}	$\begin{pmatrix} 0,636 \\ -0,772 \end{pmatrix}$	0,9997
VQLS_{sim}	$\begin{pmatrix} 0,607 \\ -0,795 \end{pmatrix}$	1
VQLS_{real}	$\begin{pmatrix} 0,593 \\ -0,802 \end{pmatrix}$	0,9975
CKS_{sim}	$\begin{pmatrix} 0,601 \\ -0,798 \end{pmatrix}$	0,9992
CKS_{real}	$\begin{pmatrix} 0,629 \\ -0,775 \end{pmatrix}$	0,9979

Tabela 3.11: Resultados obtidos com $\theta = 2\pi/3$.

Geralmente os valores dos métodos simulados apresentam maior fidelidade em relação aos mesmos métodos implementados no *hardware*. Isso evidencia os erros do *hardware* quântico. Em alguns casos a fidelidade obtida na simulação é de ≈ 1 e, no *hardware* quântico, $\approx 0,95$. Essa diferença é devido aos erros presente do *hardware* real, que são grandes mesmo para um problema simples de dimensão 2×2. Apenas para o método VQLS as fidelidades obtidas no *hardware* quântico e na simulação estão sempre compatíveis.

Capítulo 4

Conclusão

Nesta dissertação estudamos três diferentes métodos para resolução de sistemas lineares quânticos $A|x\rangle = |b\rangle$, o método de Harrow, Hassidim & Lloyd, o de Childs, Kothari & Somma e o *Variational quantum linear solver*. Cada um deles foi explanado e implementado em um computador quântico real através do Qiskit. Além dos métodos estudados, existem outros algoritmos propostos na literatura, porém de implementação mais difícil. Cada algoritmo é otimizado para uma situação diferente, quanto mais complexo, mais difícil é sua implementação nos hardwares atuais, onde os erros na implementação são grandes.

Cada um dos algoritmos quânticos estudados utilizam subrotinas que são processos essenciais que compõem cada algoritmo e definem suas propriedades. Cada uma dessas subrotinas (referenciado na seção 2.4) não possui relação com as outras, ou seja, são processos independentes. A primeira subrotina de relevância para a solução de sistemas lineares quânticos é a simulação hamiltoniana, que nada mais é do que a simulação da evolução de um sistema quântico caracterizado por uma Hamiltoniana H. O problema da simulação Hamiltoniana consiste em implementar uma operação unitária $U = e^{-iHt}$ decomposta em portas lógicas de maneira eficiente. A segunda rotina de relevância é a transformada de Fourier Quântica (QFT), que consiste na transformação das bases computacionais nas bases de Fourier. E, por fim, o problema linear quântico também utiliza um algoritmo para codificar dados clássicos e também o algoritmo de estimativa de fase quântica (QPE), onde dado o operador U, este processo estima o valor da fase φ_j tal que $U|u_j\rangle = e^{2\pi i\varphi_j}|u_j\rangle$.

Na seção 3 vimos os três algoritmos de resolução para os sistemas lineares que são mais simples de serem implementados nos computadores quânticos atuais. Inicialmente discutimos sobre o HHL, que foi o primeiro algoritmo quântico proposto para resolver sistemas lineares quânticos. Tal algoritmo utiliza-se da QPE como sua componente principal. Um fator importante na performance do algoritmo HHL está ligado aos fatores κ e s. O fator κ é o número de condicionamento da matriz e é definido como sendo a razão entre o maior e o menor autovalor da matriz A. Quando tal número de condicionamento κ cresce, A se torna mais próxima a uma matriz que não pode ser invertida e a solução se torna menos estável. Já o fator s está ligado à esparsidade da matriz.

O algoritmo HHL é um procedimento eficiente para resolução de sistemas lineares quânticos se a matriz A for bem condicionada e esparsa, porém sua performance em termos de precisão é limitada pela estimativa de fase. Visando reduzir esse erro de dependência desenvolveu-se o algoritmo CKS. Basicamente, este algoritmo substitui a estimativa de fase por algum método de aplicação direta do inverso da matriz, melhorando a precisão do algoritmo exponencialmente. Portanto o algoritmo CKS consiste numa variação do HHL porém não é único, existem outras variações. Na tabela (4.1), apresentamos as principais características de alguns dos algoritmos propostos para resolver sistemas lineares quânticos.

Algoritmo	Composição	Complexidade
HHL	Simulação Hamiltoniana, QFT, QPE	$\frac{s^2\kappa^2}{\epsilon}\log(N)$
CKS	Simulação Hamiltoniana, Inversão da matriz	$\frac{\kappa^2}{\epsilon}\log(N)$
VQLS	Simulação Hamiltoniana, Ansatz V, Função Custo, Minimização	$\frac{\kappa}{\epsilon}N$

Tabela 4.1: Características dos algoritmos de resolução de sistemas lineares quânticos

E por último vimos o algoritmo VQLS, que combina métodos computacionais clássicos e quânticos para a resolução de sistemas lineares. Esse algoritmo consiste em minimizar certos parâmetros a partir de uma função definida como função custo. Neste caso os parâmetros definem uma sequência de portas lógicas quânticas e a minimização da função custo é feita através de otimizadores clássicos. A saıda do circuito VQLS é análoga ao HHL, porém por possuir o caráter variacional, este algoritmo permite ser utilizado nos computadores quânticos NISQ, enquanto o HHL requer muitos mais qubits e precisão na implementação de cada porta, ou seja, uma capacidade maior dos hardwares quânticos.

Na implementação do VQLS, a matriz A é decomposta numa combinação linear de unitários A_n. Para a criação do circuito $V(\alpha)$ que consiste na sequência de portas que irá preparar a potencial solução $|x\rangle$ a partir de $|0\rangle$, utilizamos uma estrutura fixa que varia somente os parâmetros α das portas de rotação e não o circuito em si. Porém existem outras maneiras de criar tal estrutura [24][25][26][27]. O método de minimização utilizado foi o método COBYLA (disponível no próprio Qiskit) com o número máximo de iterações sendo 200. Fizemos três diferentes exemplos de implementação, cada um com um tamanho diferente da matriz A.

Afim de comparar os resultados dos três diferentes métodos implementamos o mesmo problema nos diferentes algoritmos modificando apenas o valor de $|b\rangle$. Após a normalização dos estados medidos calculou-se a fidelidade de cada um dos resultados. As piores fidelidades encontradas foram aproximadamente $0,95$ para os métodos HHL e CKS. E dentre todos diferentes métodos utilizados o que apresentou os melhores resultados foi o método $VQLS$.

Vale ressaltar que o método VQLS, além de apresentar os melhores resultados também possui a vantagem de ser utilizado nos computadores NISQ devido a menor quantidade de qbits necessários.

Nesta dissertação utilizamos apenas matrizes 2×2 para compor os algoritmos. Procedemos assim pois o número de qubits e de portas lógicas necessárias para resolver um problema maior aumenta muito para HHL, não sendo viável a sua implementação nas plataformas gratuitas utilizadas. Além disso as fidelidades para o HHL e o CKS já não são boas nos casos de 2×2, devidos aos erros experimentais. Foi possível resolver uma matriz maior apenas no caso do VQLS.

Podemos estender como metas para o futuro a implementação de matrizes de maiores dimensões, porém vale ressaltar que deve se adaptar tais algoritmos e, para executá-los no *hardware* real também é necessário possuir o número suficiente de qbits. No nosso caso utilizamos apenas os computadores quânticos disponíveis de maneira gratuita pela IBM Q que comportam no máximo 7 qbits. Outros modelos de computação, como por exemplo, o *quantum annealing* também oferecem resultados promissores[28], porém estes não foram abordados nesta dissertação.

Apêndice A

Códigos em Python

Exposição de alguns códigos dos principais algoritmos utilizados no desenvolvimento do projeto.

A.1 HHL

```
from qiskit import *
import numpy as np

t = np.pi*3/4
nqubits = 4  # numero total de qubits
nb = 1  # numero de qubits representando a solucao
nl = 2  # numero de qubits representando os autovalores

theta = 1* np.pi  #  ngulo  de |b>
a = 3/2  # elementos diagonais da matriz
b = 1/2  # elementos n o -diagonais da matrix

# Inicializacao registradores
qr = QuantumRegister(nqubits)
qcl = ClassicalRegister(4)

# Criacao QC
qc = QuantumCircuit(qr,qcl)
qrb = qr[0:nb]
qrl = qr[nb:nb+nl]
qra = qr[nb+nl:nb+nl+1]

# Preparacao do estado
qc.ry(2*theta, qrb[0])
qc.barrier(qr[0], qr[1], qr[2], qr[3])

# QPE
for qu in qrl:
    qc.h(qu)
qc.u1(a*t, qrl[0])
qc.u1(a*t*2, qrl[1])
```

```
qc.u3(b*t, -np.pi/2, np.pi/2, qrb[0])
qc.barrier(qr[0], qr[1], qr[2], qr[3])

# Controle U = e^{iAt} -> \lambda_{1}:
params=b*t
qc.u1(np.pi/2,qrb[0])
qc.cx(qrl[0],qrb[0])
qc.ry(params,qrb[0])
qc.cx(qrl[0],qrb[0])
qc.ry(-params,qrb[0])
qc.u1(3*np.pi/2,qrb[0])
qc.barrier(qr[0], qr[1], qr[2], qr[3])

# Controle U = e^{2iAt} -> \lambda_{2}:
params = b*t*2
qc.u1(np.pi/2,qrb[0])
qc.cx(qrl[1],qrb[0])
qc.ry(params,qrb[0])
qc.cx(qrl[1],qrb[0])
qc.ry(-params,qrb[0])
qc.u1(3*np.pi/2,qrb[0])
qc.barrier(qr[0], qr[1], qr[2], qr[3])

# QFT Inversa
qc.h(qrl[1])
qc.rz(-np.pi/4,qrl[1])
qc.cx(qrl[0],qrl[1])
qc.rz(np.pi/4,qrl[1])
qc.cx(qrl[0],qrl[1])
qc.rz(-np.pi/4,qrl[0])
qc.h(qrl[0])
qc.barrier(qr[0], qr[1], qr[2], qr[3])

# Rotacao dos autovalores
t1=(-np.pi +np.pi/3 - 2*np.arcsin(1/3))/4
t2=(-np.pi -np.pi/3 + 2*np.arcsin(1/3))/4
t3=(np.pi -np.pi/3 - 2*np.arcsin(1/3))/4
t4=(np.pi +np.pi/3 + 2*np.arcsin(1/3))/4
qc.cx(qrl[1],qra[0])
qc.ry(t1,qra[0])
qc.cx(qrl[0],qra[0])
qc.ry(t2,qra[0])
qc.cx(qrl[1],qra[0])
qc.ry(t3,qra[0])
qc.cx(qrl[0],qra[0])
qc.ry(t4,qra[0])
qc.barrier()
qc.measure_all()

# executando simulacao
sim = Aer.get_backend('aer_simulator')
shots = 2**10
```

```
simulacao = execute(qc, backend=sim, shots=shots, optimization_level=0)
ressim = simulacao.result()
contagemsim = ressim.get_counts()
```

A.2 CKS

```
from qiskit import QuantumRegister, QuantumCircuit, ClassicalRegister
import numpy as np
from qiskit.visualization import plot_histogram

nqubits = 2  # numero total de qubits
nb = 1  # numero de qubits representando a solucao
nl = 2  # numero de qubits representando os autovalores

thetab = np.pi/3 # angulo de |b>
a = 3/2  # elementos diagonais da matriz
b = 1/2  # elementos nao-diagonais da matrix

# Inicializacao registradores
qr = QuantumRegister(nqubits)
qcl = ClassicalRegister(4)

# Criacao QC
qc = QuantumCircuit(qr,qcl)
qrb = qr[0:nb]
qrl = qr[nb:nb+nl]
qra = qr[nb+nl:nb+nl+1]

# Prepara  o W
phip = 2*a/np.sqrt(a**2+b**2) - 1 # p(0) = 0.5[1+cos(phip)] // p(1) = 0.5[1-cos(phip)
    ]
qc.h(0)
qc.p(phip,0)
qc.h(0)

#Preparacao b
qc.u3(thetab, 0, 0, 1)

#Preparacao U's
qc.cx(0,1)
qc.x(0)
qc.cu3(0, 0, 0, 0,1)
qc.x(0)

#Preparacao W^{-1}
qc.h(0)
qc.p(-phip,0)
qc.h(0)
qc.barrier()
qc.measure(qr[1], 1)

```

```
# executando simulacao
sim = Aer.get_backend('qasm_simulator')
simulacao = execute(qc, backend = sim, shots=shots)
ressim = simulacao.result()
contagemsim = ressim.get_counts()
```

A.3 VQLS

```
import qiskit
from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit import Aer, transpile, assemble
import math
import random
import numpy as np
from scipy.optimize import minimize

# definicao Ansatz fixo
def apply_fixed_ansatz(qubits, parameters):
    for iz in range (0, len(qubits)):
        circ.u3(parameters[0][iz], parameters[1][iz], parameters[2][iz], qubits[iz])

# definicao Teste Hadamard
def had_test(gate_type, qubits, auxiliary_index, parameters):
    circ.h(auxiliary_index)
    apply_fixed_ansatz(qubits, parameters)
    for ie in range (0, len(gate_type[0])):
        if (gate_type[0][ie] == 1):
            circ.cx(auxiliary_index, qubits[ie])
    for ie in range (0, len(gate_type[1])):
        if (gate_type[1][ie] == 1):
            circ.cx(auxiliary_index, qubits[ie])
    circ.h(auxiliary_index)

# ansatz controlados para calcular |<b|psi>|^2 com o teste Hadamard
def control_fixed_ansatz(qubits, parameters, auxiliary, reg):
    for i in range (0, len(qubits)):
        circ.cu3(parameters[0][i], parameters[1][i], parameters[2][i], qiskit.circuit
    .Qubit(reg, auxiliary), qiskit.circuit.Qubit(reg, qubits[i]))

# definindo |b>
def control_b(auxiliary, qubits):
    thetab = np.pi/2
    for ia in qubits:
        circ.cu3(thetab, 0, 0, auxiliary, ia)

# Teste Hadamard para calcular <psi|psi>
def special_had_test(gate_type, qubits, auxiliary_index, parameters, reg):
    circ.h(auxiliary_index)
    control_fixed_ansatz(qubits, parameters, auxiliary_index, reg)
    for ty in range (0, len(gate_type)):
        if (gate_type[ty] == 1):
```

```
            circ.cx(auxiliary_index, qubits[ty])
    control_b(auxiliary_index, qubits)
    circ.h(auxiliary_index)

# custo funcional no circuito
def calculate_cost_function(parameters):
    global opt
    overall_sum_1 = 0
    parameters = [parameters[0:1], parameters[1:2], parameters[2:3]]
    for i in range(0, len(gate_set)):
        for j in range(0, len(gate_set)):
            global circ
            qctl = QuantumRegister(3)
            qc = ClassicalRegister(3)
            circ = QuantumCircuit(qctl, qc)
            backend = Aer.get_backend('aer_simulator')
            multiply = coefficient_set[i]*coefficient_set[j]
            had_test([gate_set[i], gate_set[j]], [1], 0, parameters)
            circ.save_statevector()
            t_circ = transpile(circ, backend)
            qobj = assemble(t_circ)
            job = backend.run(qobj)
            result = job.result()
            outputstate = np.real(result.get_statevector(circ, decimals=100))
            o = outputstate
            m_sum = 0
            for l in range (0, len(o)):
                if (l%2 == 1):
                    n = o[l]**2
                    m_sum+=n
            overall_sum_1+=multiply*(1-(2*m_sum))
    overall_sum_2 = 0
    for i in range(0, len(gate_set)):
        for j in range(0, len(gate_set)):
            multiply = coefficient_set[i]*coefficient_set[j]
            mult = 1
            for extra in range(0, 2):
                qctl = QuantumRegister(3)
                qc = ClassicalRegister(3)
                circ = QuantumCircuit(qctl, qc)
                backend = Aer.get_backend('aer_simulator')
                if (extra == 0):
                    special_had_test(gate_set[i], [1], 0, parameters, qctl)
                if (extra == 1):
                    special_had_test(gate_set[j], [1], 0, parameters, qctl)
                circ.save_statevector()
                t_circ = transpile(circ, backend)
                qobj = assemble(t_circ)
                job = backend.run(qobj)
                result = job.result()
                outputstate = np.real(result.get_statevector(circ, decimals=100))
                o = outputstate
```

```
                m_sum = 0
                for l in range (0, len(o)):
                    if (l%2 == 1):
                        n = o[l]**2
                        m_sum+=n
                mult = mult*(1-(2*m_sum))
            overall_sum_2+=multiply*mult
    print(1-float(overall_sum_2/overall_sum_1))
    return 1-float(overall_sum_2/overall_sum_1)

# definindo elementos da matriz A
coefficient_set = [1.5, 0.5]
gate_set = [[0], [1]]

# funcao minimizacao
out = minimize(calculate_cost_function, x0=[float(random.randint(0,3000))/1000 for i
    in range(0, 3)], method="COBYLA", options={'maxiter':200, 'tol':0.0000000001})
out_f = [out['x'][0:1], out['x'][1:2], out['x'][2:3]]

# criando circuito
circ = QuantumCircuit(1, 1)
apply_fixed_ansatz([0], out_f)
circ.save_statevector()

# executando simulacao
backend = Aer.get_backend('aer_simulator')
t_circ = transpile(circ, backend)
qobj = assemble(t_circ)
job = backend.run(qobj)

result = job.result()
o = result.get_statevector(circ, decimals=10)
a1 = coefficient_set[1]*np.array([[0,1], [1,0]])
a2 = coefficient_set[0]*np.array([[1,0], [0,1]])
a3 = np.add(a1, a2)
```

Referências Bibliográficas

[1] M. A. Nielsen; I. L. Chuang, *Quantum Computation and Quantum Information*. Cambridge University Press 10ed, 2011.

[2] Gordon E. Moore, *Cramming More Components onto Integrated Circuits*. Electronics, 1964.

[3] Abraham Asfaw; Luciano Bello; Yael Ben-Haim; Sergey Bravyi; Nicholas Bronn et.al, *Learn quantum computation using qiskit*. `https://qiskit.org/`.

[4] *Cirq, a python framework for creating, editing, and invoking Noisy Intermediate Scale Quantum (NISQ) circuits*. `https://github.com/quantumlib/Cirq`.

[5] *Microsoft Quantum Development Kit*. `https://github.com/microsoft/Quantum`.

[6] A. W. Harrow; A. Hassidim; S. Lloyd, *Quantum algorithm for linear systems of equations*. Physical Review Letters 103 no. 15, 2009.

[7] A. M. Childs; R. Kothari; R. D. Somma. *Quantum algorithm for systems of linear equations with exponentially improved dependence on precision*. arXiv:1511.02306, 2017.

[8] C. Bravo-Prieto; R. LaRose, M. Cerezo; Y. Subasi; L. Cincio; P. J. Coles, *Variational Quantum Linear Solver: A Hybrid Algorithm for Linear Systems*. Bulletin of the American Physical Society, 2020.

[9] R. Horodecki; P. Horodecki; M. Horodecki; K. Horodecki, *Quantum entanglement*. Reviews of Modern Physics 81, pp. 865-942, 2009.

[10] J. R. Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, 1994.

[11] I. M. Georgescu; S. Ashhab; Franco Nori, *Quantum Simulation*. Reviews of Modern Physics 86, pp. 153-185, 2014.

[12] D. Camps; R. Van Beeumen; C. Yang, *Quantum Fourier transform revisited*. Numerical Linear Algebra with Applications. 28. 10.1002/nla.2331, 2020.

[13] H. Mohammadbagherpoor; Y. -H. Oh; P. Dreher; A. Singh; X. Yu; A. J. Rindos, *An Improved Implementation Approach for Quantum Phase Estimation on Quantum Computers*. IEEE International Conference on Rebooting Computing (ICRC), pp. 1-9, 2019.

[14] J. A. Cortese; T. M. Braje, *Loading classical data into a quantum computer*. arXiv preprint, arXiv:1803.01958, 2018.

[15] I. Chiorescu; N. Groll; S. Bertaina; T. Mori; S. Miyashita, *Magnetic strong coupling in a spin-photon system and transition to classical regime*. Physical Review B 82, 024413, 2010.

[16] D. I. Schuster; A. P. Sears; E. Ginossar; L. DiCarlo; L. Frunzio; J. J. L. Morton; H. Wu; G. A. D. Briggs; B. B. Buckley; D. D. Awschalom; R. J. Schoelkopf, *High-Cooperativity Coupling of Electron-Spin Ensembles to Superconducting Cavities*. Physical Review Letters 105, 140501, 2010.

[17] Y. Kubo; F. R. Ong; P. Bertet; D. Vion; V. Jacques; D. Zheng; A. Dréau; J.-F. Roch; A. Auffeves; F. Jelezko; J. Wrachtrup; M. F. Barthe; P. Bergonzo; D. Esteve, *Strong Coupling of a Spin Ensemble to a Superconducting Resonator*. Physal Review Letters 105, 140502, 2010.

[18] Hua Wu; Richard E. George; Janus H. Wesenberg; Klaus Mølmer; David I. Schuster; Robert J. Schoelkopf; Kohei M. Itoh; Arzhang Ardavan; John J. L. Morton; G. Andrew D. Briggs, *Storage of Multiple Coherent Microwave Excitations in an Electron Spin Ensemble*. Physical Review Letters 105, 140503, 2010.

[19] K. Mitarai; M. Kitagawa; K. Fujii, *Quantum analog-digital conversion*. Physical Review A 99, 012301, 2019.

[20] Y. Cao; A. Daskin; S. Frankel; S. Kais, *Quantum circuit design for solving linear systems of equations*. Molecular Physics 110 no. 15-16, p. 1675–1680, 2012.

[21] L. Wossnig; Z. Zhao; P. Zhikuan; A. Prakash, *Quantum Linear System Algorithm for Dense Matrices*. Physical Review Letters 120 no. 5, 2018.

[22] A. Ambainis, *Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations*. arXiv:1010.4458v2, 2010.

[23] J. Preskill, *Quantum Computing in the NISQ era and beyond*. Quantum 2, p. 79, 2018.

[24] R. LaRose; A. Tikku; É. O'Neel-Judy; L. Cincio; P. J. Coles, *Variational quantum state diagonalization*. npj Quantum Information 5, 57, 2019.

[25] L. Cincio; Y. Subaşı; A. T. Sornborger; P. J. Coles, *Learning the quantum algorithm for state overlap*. New Journal of Physics 20, 113022, 2018.

[26] E. Farhi; J. Goldstone; S. Gutmann, *A quantum approximate optimization algorithm*. arXiv:1411.4028, 2014.

[27] S. Hadfield; Z. Wang; B. O'Gorman; E. G. Rieffel; D. Venturelli; R. Biswas, *From the quantum approximate optimization algorithm to a quantum alternating operator ansatz*. Algorithms 12, 34, 2019.

[28] Alexandre M. Souza; Eldues O. Martins; Itzhak Roditi; Nahum Sá; Roberto S. Sarthour; Ivan S. Oliveira, *An Application of Quantum Annealing Computing to Seismic Inversion*. Frontiers in Physics 9, 2021.