量子计算

量子计算编程基础课程讲义

Quantum Computing Course

赵晓菲

量子计算

量子计算编程基础课程讲义

by

赵晓菲

参编教师　赵睿、蒲晓妮、高丽

院系:　信息与控制工程学院

基金:　中国高校创新基金

日期:　2021.10-2023.10

企业:　本源量子

封面:　兰州白塔山西麓

摄影:　赵晓菲 2023 年 7 月 4 日拍摄于七里河建兰南路

兰州石化职业技术大学

Lanzhou Petrochemical University of Vocational Technology

目录

插图

表格

1

叠加态、纠缠和可逆性

1.1. 前言

什么是量子计算机？[1] 回答这个问题涵盖了量子力学（Quantum mechanics，QM）、量子信息理论（Quantum Information Theory，QIT）和计算机科学（Computer Science，CS）。

现今的计算机，不论是理论上的图灵机如图1.1，还是实际上的个人电脑、高性能计算机、笔记本电脑、平板电脑、智能手机等，都是基于经典物理学的。它们受到地域性的限制（操作只有局部影响），并受到经典物理学[1]实际的限制，即“**系统在任何时刻只能处于一个状态**”。

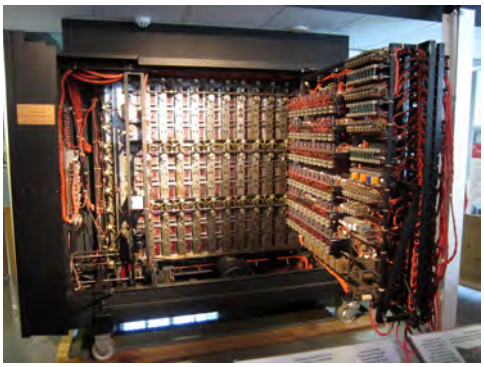
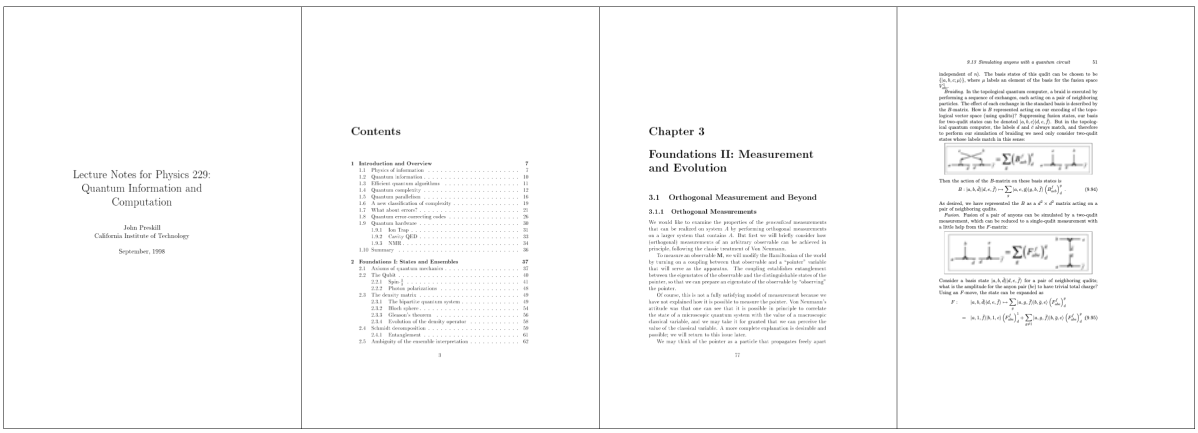


图 1.1: 在 Bletchley Park 的早期图灵机。Bletchley Park 是英国政府机构，位于英格兰中部的 Buckinghamshire 郡。该机构成立于第二次世界大战期间，旨在破译德国和其它敌方国家的加密通信。在战争期间，Bletchley Park 聚集了一批英国最聪明的数学家、工程师、语言学家和密码学家，他们的任务是破解德国的加密通信，为盟军提供重要情报支持。在 Bletchley Park，使用了各种技术手段，包括电子机械装置、密码机和语言学分析等，以破译加密通信。该机构最为著名的成就之一是破解 Enigma 密码系统，这是纳粹德国用于保护军事通信的一种机械密码机，使用了一系列复杂的密码方法。通过对 Enigma 密码系统的破解，Bletchley Park 帮助盟军获取了大量的战略情报，对于战争的胜利进程起到了重要作用。Bletchley Park 的历史是英国在二战期间的一个重要组成部分，同时也为现代密码学、计算机科学和信息技术的发展做出了贡献。目前，Bletchley Park 已成为一个开放的博物馆和教育中心，向公众展示和解释这段历史，并向人们展示这个时期的科技成就和创新精神。图片由 Douglas Hoyt 拍摄：`https://www.flickr.com/photos/douglashoyt/8235850748/`

[1]经典物理学是指在相对论和量子力学出现之前，对物理现象进行研究的传统物理学分支。经典物理学主要涉及物理学中最基本的领域，包括力学、电学、热学、光学和声学等。经典物理学的研究对象通常是宏观物体和宏观现象，其中最为著名的经典物理学理论包括牛顿力学、电磁学马克思韦尔方程、热力学第一、二定律和光的波动理论等。尽管经典物理学被相对论和量子力学所替代，但它仍然是物理学中重要的基础和入门学科，也是许多实际应用的基础。

然而，现代量子物理告诉我们，世界的运行方式可能大不相同。量子系统认为可以处于多个不同状态的“超位置”，其演化过程中会表现出干涉效应。另外，空间上分离的量子系统之间可能会发生纠缠，操作可能会因此具有“非局部”效应[2]。

量子计算是研究基于量子力学原理的计算机的计算能力和其他特性的领域。它结合了 20 世纪最重要的两个科学领域：**量子力学**[3]和**计算机科学**[4] 。探索量子力学的一个重要的目标是寻找比解决同样问题的任何经典算法快得多的量子计算方法或者量子计算范式。


Lecture Notes for Physics 229:
Quantum Information and
Computation

John Preskill
California Institute of Technology

September, 1998

Contents

Chapter 3

Foundations II: Measurement and Evolution

3.1 Orthogonal Measurement and Beyond

3.1.1 Orthogonal Measurements

图 1.2: 《Lecture notes for physics 229: Quantum information and computation》图1.1。这是一篇关于量子信息和计算的讲义的文献引用，作者为 John Preskill，发表于 1998 年，发表在《加州理工学院》期刊上。该文献记录了作者在加州理工学院教授的物理学 229 课程的讲义。这门课程介绍了量子信息和计算的基本概念，包括量子比特、量子纠缠、量子电路、量子算法等内容，并讨论了量子计算的应用前景。这门课程对后来量子计算和量子信息领域的发展产生了重要影响，被认为是这一领域的经典教材之一。该文献可以被视为介绍量子计算和量子信息领域的重要文献之一，对这一领域的初学者和研究者都有很大的参考价值。感兴趣的同学可以问赵老师索要改讲义 xifizhao@gmail.com。

量子计算始于 20 世纪 80 年代初，由**尤里 · 伊万诺维奇 · 马宁**[5] [3, 4] 、**理查德 · 费曼**[6] [5, 6] 和

[2]“非局部”效应是指在某个系统中，两个或多个空间上分离的部分之间存在相互作用或联系，这种联系是通过超距离传递信息或作用而发生的。这种联系并不依赖于空间上的距离，而是通过某种非局部的方式进行传递。

在物理学中，“非局部”效应的概念经常出现在量子力学中，例如量子纠缠效应。量子纠缠是指当两个或多个量子系统处于纠缠状态时，它们之间的相互作用是非局部的，即它们之间的相互作用并不受到它们之间距离的影响。这意味着当一个量子系统发生测量时，它可能会影响与之纠缠的另一个量子系统，即使它们之间的距离非常远，这种现象被称为“非局部”效应。非局部效应在量子信息学和量子计算等领域有重要应用。

[3]由普朗克、爱因斯坦、玻尔、海森堡、薛定谔等人于 1900 年至 1925 年间发展

[4]其诞生可以追溯到图灵 1936 年的论文 [2]

[5]尤里 · 伊万诺维奇 · 马宁（Yuri Ivanovich Manin）是一位著名的数学家，生于 1937 年，是苏联和俄罗斯的国家科学院院士，也是美国国家科学院外籍院士。他是数学领域中的重要人物，对代数几何、数论、数学物理等领域作出了卓越的贡献，他也是现代数学的一些基本概念和理论的奠基人之一。

马宁在代数几何和数论领域的研究成就非常突出，他对于椭圆曲线、数域、代数簇、模形式、保型形式等领域都做出了杰出的贡献，他也在数学物理领域中的工作取得了重要成果。马宁获得了多项国际数学奖项，包括 1982 年的菲尔兹奖和 1993 年的沃尔夫数学奖等。此外，他还是国际数学界著名杂志《数学年刊》（Annals of Mathematics）的编辑之一。

[6]理查德 · 费曼（Richard Feynman）是一位著名的美国理论物理学家，生于 1918 年，逝世于 1988 年。他是 20 世纪最具影响力的物理学家之一，同时也是一位杰出的教育家、科学普及者和散文家。费曼在量子电动力学和粒子物理学的研究方面作出了重要贡献，并因此获得了 1965 年的诺贝尔物理学奖。

费曼的贡献涉及许多领域，包括量子电动力学、强相互作用、拓扑物理学、计算机科学等。他发明了一种著名的计算方法，称为费曼图，这种图形方法能够清晰地表示物理过程，为理解粒子物理学提供了强有力的工具。此外，他也在物理学教育领域作出了杰出的贡献，他的讲授风格幽默诙谐、生动活泼，深受学生和同行们的喜爱。他的讲义和书籍，如《费曼物理学讲义》等，在物理学教育中被广泛使用。

保罗·贝尼奥夫[7] [7] 提出了模拟量子计算机的建议。在 1985 年，**大卫·迪奥克**[8]定义了通用量子图灵机 [8]，将其应用扩展到数字领域。加州理工学院量子信息研究所主任**约翰·普雷斯基尔** (John Preskill) 教授的教材讲义《Lecture notes for physics 229: Quantum information and computation》[9, 10] 中有更多关于量子计算机早期历史的介绍，有兴趣的同学可以去阅读。再接下来的几年中，**大卫·迪奥克**和**理查德·约瑟夫**[9] [11]、**丹尼尔·西蒙** [12] 首先提出的量子算法以及由**伊桑·伯恩斯坦**和**乌梅什·瓦齐拉尼** [13] 发展出的量子复杂性理论。然而，在 1994 年**彼得·肖尔**非常意外地发现了整数因子分解和离散对数问题的有效量子算法——肖尔算法（关于这段历史，有一段非常不错的彼得 · 肖尔本人采访的视频以及根据此段视频制作的动画片，感兴趣的同学可以联系赵老师组织观看）[14]，肖尔算法是建立在 Simon 的工作之上，他的意义在于由于我们现在的大多数经典密码学是基于这两个问题的计算难度来保证的，因此实际构建和使用量子计算机将使我们能够破解大多数当前的经典密码系统，尤其是基于 RSA 系统 [15, 16]。当然，**查尔斯·贝内特**和**吉尔·布拉萨** [17] 提出的量子密码学形式即使对于量子计算机也是不可破解的。以下是探索量子计算的三个原因，包括了解决实际问题和哲学层面上的：

- 使现有的经典计算机已经很强大和廉价的微型化过程（摩尔定律）已经达到了量子效应要发生的微观水平。
- 利用量子效应可以极大地加速某些计算（有时呈指数级），甚至使某些经典计算机无法实现的事情成为可能。本课程的主要目的是详细解释量子计算编程（算法、密码学等）的这些优点。
- 最后，可以说理论计算机科学的主要目标是“研究自然界允许我们使用的最强大的计算设备的能力和限制”。由于我们对自然的当前理解是量子力学的，理论计算机科学应该研究量子计算机的能力，而不是经典计算机的能力。

在开始这门课程之前，让我们问一个实践方面的问题：建造量子计算机需要什么条件？此时此刻，还为时过早？第一台小型 2 比特量子计算机是在 1997 年建造的，2001 年一台 5 比特量子计算机被用来成功因式分解“15”[18]。此后，对不同技术路线的实验进展稳步但依旧缓慢。目前最先

[7]保罗 · 贝尼奥夫（Paul Benioff）是一位美国计算机科学家和量子物理学家，生于 1934 年，逝世于 2021 年。他是计算机科学中量子计算理论的先驱之一，并且是最早研究量子计算机的科学家之一。

在 20 世纪 80 年代初，贝尼奥夫就开始研究量子计算机的理论，并在 1982 年提出了著名的“量子图灵机”（quantum Turing machine）概念，为后来的量子计算机研究提供了基础。他也是第一批认识到量子计算机在某些领域将优于传统计算机的人之一，并且他的工作启发了后来量子计算领域的发展。

此外，贝尼奥夫还在物理学中作出了重要贡献，他是最早将量子力学与信息理论相结合的科学家之一，他的研究在量子信息理论、量子算法和量子物理学的交叉领域有广泛应用。他还是一位出色的教育家，在科学教育和科学普及方面也做出了贡献。

[8]**大卫·迪奥克**（David Deutsch）是一位英国物理学家和量子计算机科学家，生于 1953 年。他是量子计算理论的先驱之一，也是量子信息科学的奠基人之一。

迪奥克在 20 世纪 80 年代初开始研究量子计算理论，提出了著名的“量子图灵机”（quantum Turing machine）模型，并在 1985 年提出了量子并行性的概念，这些成果对后来量子计算机的发展产生了深远的影响。此外，他还提出了量子误差纠正的概念，这在保护量子信息方面起到了重要作用。

迪奥克还在物理学中作出了贡献，尤其是在量子力学的基本问题和哲学方面。他提出了“多世界诠释”（Many-worlds interpretation）的量子力学解释，认为量子力学中的不确定性并不是真正的随机，而是来自于各种可能性的同时发生。他的工作在哲学、科学和文化领域都产生了广泛的影响。

迪奥克获得了多项国际奖项，包括 2002 年的欧洲物理学会量子电子学奖和 2010 年的英国皇家学会皇家奖章等。他还是多本著作的作者，其中包括《量子计算机和量子通信》（Quantum Computation and Quantum Communication）等经典著作。

[9]理查德 · 约瑟夫（Richard Jozsa）是一位英国数学家和计算机科学家，生于 1957 年。他是量子计算和量子信息领域的重要人物之一，同时也是量子算法领域的奠基人之一。

Jozsa 在 20 世纪 90 年代提出了著名的 Jozsa 算法，该算法可以判断任意的黑盒量子函数是否为恒等函数或者平衡函数，是量子计算领域中最早的重要算法之一。此外，他还提出了其他一些著名的量子算法，如 Grover 搜索算法的一种变体，以及求解线性方程组的量子算法等。

Jozsa 也是量子信息理论领域的重要研究者，他的研究工作涉及量子信息压缩、量子纠错等领域，并发展了许多经典信息理论的量子版本。他还曾获得过多个重要奖项，包括 2005 年的欧洲物理学会量子电子学奖和 2007 年的英国皇家学会皇家奖章等。

进的实现方式使用超导量子比特和离子阱量子比特。在撰写本文的时候，据英国《新科学家》周刊网站 11 月 9 日报道，国际商用机器公司（IBM）已经制造出迄今为止最大的量子计算机。这台名为“鱼鹰”（Osprey）的计算机包含 433 个量子比特，是该公司此前打破纪录的 127 个量子比特计算机的 3 倍多，是谷歌公司包含 53 个量子比特的“西克莫”计算机[10]的 8 倍多。

图 1.3: IBM Osprey 芯片组的示意图。IBM Osprey 是一种用于构建量子计算机的芯片组，是 IBM 量子计算机的核心部件之一。这个芯片组包括多个量子位和多个控制电路，用于在量子计算机中执行量子门操作，从而实现量子算法的运行。图片来源：IBM 官网 `www.ibm.com`

在理论上，一个 433 量子比特的处理器可以处理 2^{433} 的复数希尔伯特空间，远远超出任何经典系统。但是在量子计算过程中，处理器内部的大量数据（被称为“状态向量”）需要使用单量子位门和双量子位门进行处理。大多数量子算法都需要至少运行同样大量的门循环，这导致门数量远大于算法深度。错误在量子计算中也会迅速增加，因此需要错误纠正和错误缓解技术。

以 99% 的双量子比特错误率为例（没有错误缓解），使用具有 422 个双量子比特门的算法时，您有 1.28% 的机会获得良好的结果。而在 99.9% 的错误率下，这个数字将会下降到 65%。这表明错误率的高低对于量子计算的结果至关重要。因此，需要开发出更加可靠的量子硬件和量子错误纠正技术来提高量子计算的准确性和可靠性。

物理实现量子计算机所面临的实际问题似乎十分严峻。噪声[11]和退相干问题[12]在理论上在一定

[10]” 西克莫” 是指 Google Quantum AI 实验室在 2019 年推出的一款量子计算机处理器，该处理器包含了 53 个超导量子比特。这是谷歌目前推出的量子计算机处理器中最大的一个，也是在公共领域中可用的最大量子计算机之一。

” 西克莫” 处理器的目标是实现量子优势，即通过利用量子计算的优势解决经典计算难以处理的问题。与传统的二进制计算机不同，量子计算机使用量子比特作为基本计算单元，可以在同一时间处理多个计算，从而具有处理大规模数据的能力。这种能力使得量子计算机在解决一些特定问题时具有优势，如优化、模拟和密码学等领域。

” 西克莫” 处理器的推出标志着谷歌在量子计算领域取得了重大进展，并成为了该领域的领军者之一。

[11]在量子计算中，噪声问题是一个非常严重的挑战。由于量子计算机中的量子比特是非常脆弱的，很容易受到周围环境的影响，例如温度、电磁干扰和量子比特之间的相互作用等。这些环境因素会导致量子比特产生误差和失真，从而对量子计算机的准确性和可靠性产生影响，这就是量子计算机的噪声问题。

在实际的量子计算机中，错误的来源是多种多样的。例如，有些量子比特可能会因为与环境的相互作用而失去相干性；有些量子比特可能会因为与其他量子比特的相互作用而发生失真；还有些量子比特可能会因为随机性引起的噪声而产生错误。

噪声问题不仅会影响量子计算机的运行速度，还会对算法的正确性和可靠性产生严重影响。为了解决这个问题，需要采用各种技术来对量子比特进行纠错和缓解。其中一些技术包括量子错误纠正（Quantum Error Correction，QEC）、量子优化和量子模拟等。这些技术可以帮助我们提高量子计算机的准确性和可靠性，并进一步推动量子计算的发展

[12]在量子计算中，” 相干性”（Coherence）是指一个量子比特能够保持在一个明确定义的状态（例如，$|0\rangle$ 或 $|1\rangle$ 或它们的叠加态 $|\rangle$ 的时间长度。量子计算中的” 退相干”（Decoherence）问题是指量子比特在和环境相互作用时，失去它的相干性和纠缠状态，导致量子计算的错误和失效。

在量子计算中，相干性是实现量子算法所必需的，因为它允许量子比特之间的纠缠和相互作用。相干性的丢失是由于量子比特与它周围的环境（如热噪声、振动、磁场等）相互作用而引起的。这种相互作用会导致量子比特的相位和振幅失真，从而使得量子比特不能维持其原有的纠缠状态，而是演化成经典态。

退相干问题对于量子计算的实现非常关键，因为它限制了量子计算机的规模和精度。如果不能有效地控制和抑制退相干，量子计算机就不能够进行准确和可靠的计算。因此，为了解决退相干问题，研究者们采用了多种技术，如使用量子错误纠正码来保护量子比特，使用量

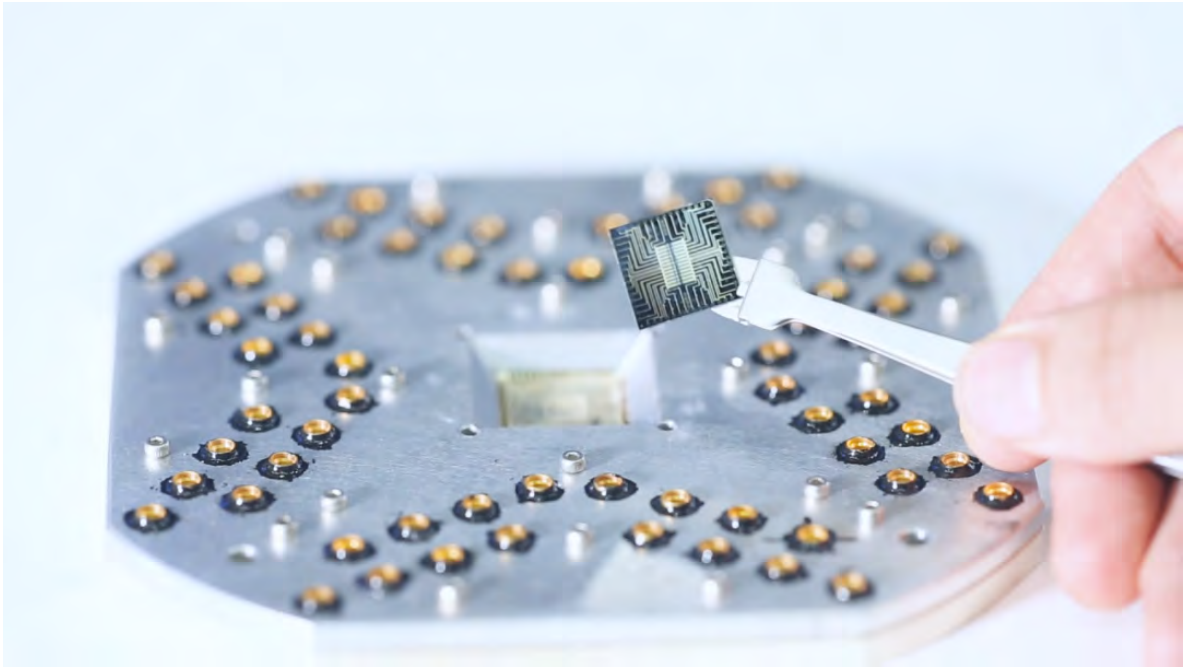


图 1.4: 本源超导 24 比特量子芯片。图片来源：`https://www.guancha.cn/`

程度上已经被量子纠错码和容错计算的发现所解决，但这些问题在实践中并没有被完全解决。另一方面，科学家认识到，物理实现量子计算机的领域仍处于起步阶段，而经典计算机也曾面临和解决过许多艰巨的技术问题，有趣的是，这些在经典计算机发展中遇到的问题甚至与现在量子计算机面临的问题相同（例如，噪声抑制和纠错）。另外，尽管实现完整量子计算机面临的困难似乎令人望而却步，但涉及量子通信的更有限的应用已经在一定程度上取得了成功，例如通过纠缠和经典通信发送量子比特的传送（teleportation），以及 BB84 量子密钥分发的一些版本现在甚至已经商业化。

即使量子计算的理论永远不会实现为现在像手机、笔记本电脑一样真正的大规模物理计算设备，量子计算机仍然是一个极为有趣的想法，将在计算之外产生一系列成果。

- 在物理学方面，它可能会改善我们对量子力学的理解。关于纠缠和哈密顿复杂度的新兴理论已经在某种程度上做到了这一点。
- 在计算机科学方面，量子计算理论扩展和丰富了经典复杂性理论，可能有助于解决其中的一些问题虽然实现完整的量子计算机面临的实际问题似乎十分艰巨。

总结：说量子计算是涉及最广泛的大学学科可能有些夸张，但它确实跨足了许多学科领域。量子计算的研究和应用在诸多领域中都发挥着重要作用，这使得量子计算成为了一个高度跨学科的领域。以下是一些量子计算涉及的主要学科：

- **物理学**：量子计算的基础建立在量子力学之上，这是现代物理学的一个重要分支。量子计算机的设计、构建和实现都需要对量子力学和相关物理概念有深入的理解。
- **计算机科学**：量子计算机使用量子比特（qubits）作为基本信息单位，与传统计算机使用的比特（bits）不同。因此，量子计算需要独特的算法和计算方法，这使得计算机科学家在量子计算领域具有关键作用。

子纠缠和量子纠错技术来控制量子比特的相干性和纠缠状态，以及使用低温和高真空等环境条件来减少退相干的影响。

- **数学**：量子计算依赖于复杂数学理论，包括线性代数、概率论、离散数学等。研究量子算法需要深入了解这些数学领域。
- **材料科学与工程**：量子计算机的制造需要研究和开发新型量子材料以及相关技术。材料科学家和工程师在这方面发挥着重要作用。
- **电子工程**：量子计算机的硬件设计、微电子制造和量子比特控制等方面需要电子工程的专业知识。
- **化学**：量子化学是一个研究化学现象的理论框架，量子计算在量子化学模拟和分子结构预测方面具有潜在优势。
- **生物学**：量子计算在生物学中的应用包括基因组学数据处理、蛋白质折叠模拟以及药物设计等。
- **信息论与密码学**：量子计算机具有处理复杂数学问题的潜力，这使得它在信息论和密码学领域具有巨大价值。量子密码学已经成为信息安全领域的一个重要分支。

因此，虽然称量子计算涉及最广泛的大学学科可能夸大其词，但它的确是一个高度跨学科的领域，吸引了来自各种背景的研究人员。

1.1.1. 量子力学

量子力学是研究原子、分子、原子核以及基本粒子等微观粒子行为和相互作用的物理学分支。它是 20 世纪初诞生的一种革命性的理论，为我们理解和描述自然界微观现象提供了一个全新的框架。

与经典物理学相比，量子力学在描述原子尺度的微观粒子时具有明显的优势。在经典物理学中，物体的位置和动量可以被同时精确测量，物体遵循牛顿运动定律。然而，对于原子尺度的微观粒子，这种描述已经不再适用。量子力学揭示了在微观尺度下，物体的行为表现出波粒二象性，即它们既具有波动性，又具有粒子性。

量子力学的主要特点包括：

- **波函数**（Wave Function）：在量子力学中，波函数是描述量子系统的一种数学工具。

  它是一个复数函数，可以用来预测量子系统的物理性质和行为。波函数通常用希腊字母 ψ 表示，它的形式可以是一维、二维或三维的，具体取决于所描述的物理系统的性质和结构。

  波函数包含了关于量子系统的全部信息，包括位置、动量、自旋和能量等性质。在量子力学中，波函数的演化是由薛定谔方程来描述的，它可以预测波函数随时间的演化以及系统的能量本征态和本征值。

  波函数的模平方值代表了找到一个粒子在某一特定位置的概率。例如，对于一个单独的电子，它的波函数可以用来预测电子在空间中的分布，而它的模平方值则表示了在一个特定位置找到这个电子的概率。

  最后，波函数是描述量子系统的基本数学工具，在量子力学中发挥着至关重要的作用。通过使用波函数，可以预测量子系统的各种物理性质和行为，并且波函数的作用也在实验中得到了广泛的验证。
- **测量与坍缩**：总的来说，在测量之前，微观粒子处于一种概率性的状态（即量子叠加态）。在进行测量后，波函数会坍缩成一个确定的状态。

具体讲就是，在量子力学中，测量是指通过某种方式来获取一个量子系统的信息。测量的结果是一个经典的物理量，例如位置、动量或能量，它通常以一个确定的数值表示。然而，在量子力学中，测量并不总是产生确定的结果，而是具有一定的概率性。具体地说，测量的结果是由波函数的性质和测量过程中的随机性决定的。

在量子力学中，测量会导致波函数的坍缩，即波函数的一部分会消失，而其余部分会转化为一个确定的状态。例如，对于一个处于叠加态的量子比特，进行测量会导致它处于两个状态中的一个，而另一个状态将被消除。在这个过程中，波函数的坍缩是不可逆的，而且是由测量过程本身所决定的。

测量与波函数的坍缩是量子力学中的两个基本概念。它们是量子力学理论中的核心，也是解释量子世界的重要基础。它们的存在说明了量子世界与经典世界的巨大差异，而且在现代科技中也有广泛应用。

- **测不准原理**（Heisenberg's Uncertainty Principle）：由维尔纳 · 卡尔 · 海森堡[13]在 1927 年提出，表明某些物理量（如位置和动量）不能同时被精确测量。这一原理揭示了量子世界的根本不确定性，是量子力学中的一个基本原理，测不准原理揭示了在量子尺度下，某些物理量对的测量存在着根本性的限制。

  根据测不准原理，对于一个量子系统，例如一个微观粒子，我们不能同时精确测量某一对互补物理量，如位置（x）和动量（p）。换句话说，当我们试图更精确地测量一个物理量（如位置）时，与之相关的另一个物理量（如动量）的不确定性将增加。这种限制并非是由测量工具的不精确性导致的，而是量子系统本身固有的性质。

  数学上，测不准原理可以表示为：

$$\Delta x \cdot \Delta p \geq \frac{\overline{h}}{2} \tag{1.1}$$

  其中，Δx 表示位置的不确定性，Δp 表示动量的不确定性，$\overline{h}$ 表示约化普朗克常数（即普朗克常数除以 2π）。这个表达式说明了，位置和动量的不确定性乘积的最小值受到约化普朗克常数的限制。

  测不准原理对量子力学的发展产生了深远影响，强调了在微观世界中，概率和不确定性是根本性的特征。这与经典物理学中的决定性观念形成了鲜明对比。测不准原理的发现对量子力学的基础和应用产生了重大影响，如量子计算、量子通信和量子加密等领域。

- **量子纠缠** Quantum entanglement：两个或多个微观粒子的状态可以相互关联，即使它们相隔很远，对其中一个粒子的测量会立即影响另一个粒子的状态。这种现象被爱因斯坦称为“鬼

[13] **维尔纳 · 卡尔 · 海森堡**（Werner Karl Heisenberg，1901 年 12 月 5 日 – 1976 年 2 月 1 日）是一位德国理论物理学家，被认为是量子力学创立者之一。他在原子物理学、核物理学和粒子物理学等领域做出了重要贡献，尤其是他关于量子力学的不确定性原理，对现代物理学产生了深远影响。

海森堡 1925 年提出了一种矩阵力学的形式体系，这是量子力学发展的一个重要里程碑。矩阵力学为描述原子系统的动力学行为提供了一种新的数学框架，不同于经典力学的方法。矩阵力学与薛定谔的波动力学最终证明是等价的，两者共同构成了量子力学的基础。

海森堡在 1927 年提出了著名的海森堡不确定性原理，这是量子力学中最重要和最具挑战性的观念之一。不确定性原理表明，在相同时间内，粒子的位置和动量不能同时被精确测量。类似地，粒子的能量和时间也受到不确定性原理的限制。这一原理揭示了量子世界的根本性质，即在微观尺度上，物理现象具有概率性和不确定性。

海森堡在核物理学领域的工作也非常重要。他在 1932 年提出了一个描述原子核的力学模型，称为液滴模型。这一模型为后来的核物理研究提供了理论基础。此外，他还参与了德国在第二次世界大战期间的核研究项目，尽管这个项目最终未能实现核武器的研制。

海森堡在 1932 年获得了诺贝尔物理学奖，以表彰他在量子力学方面的杰出贡献。他的工作对现代物理学产生了深远影响，尤其是在量子力学、核物理学和粒子物理学等领域。

魅般的超距作用”。

量子纠缠是量子力学中一种特殊的现象，它描述了两个或多个量子系统之间的非常强的关联。当量子系统被纠缠时，一个系统的状态将会立即影响另一个系统的状态，即使它们之间的距离相当遥远。这种现象超越了经典物理学范畴，因为它看似违反了经典物理中的局部实效原理[14]。

量子纠缠起初是由爱因斯坦、波多尔斯基和罗森于 1935 年提出的，他们认为这种现象是量子力学理论的不完备之处，因为它违反了他们所认为的物理现实观念。后来，贝尔（Bell）提出了贝尔不等式，用以检验这种纠缠现象是否真实存在。随后的实验表明，贝尔不等式在某些情况下确实被违反了，这为量子纠缠提供了强有力的证据。

量子纠缠现象在量子信息学领域具有重要应用价值，例如量子通信、量子计算和量子密码学。

所以说量子力学不仅推翻了经典物理学的许多观念，而且为现代科学技术的发展提供了基础，如半导体、激光、核能、量子计算等领域。尽管量子力学的某些现象直观上难以理解，但它已经成为描述自然界微观现象的最成功和最完善的理论。

1.1.2. 量子信息理论

量子信息理论是一门研究量子系统中信息处理和传输的跨学科领域，它将量子力学的基本原理应用于信息科学和通信技术。量子信息理论的核心思想是利用量子力学的特性（如量子叠加和量子纠缠）来实现对信息的高效处理、安全传输和存储。

量子信息理论主要包括以下几个方面：

- **量子计算**：量子计算是量子信息理论中的一个关键概念，它利用量子力学的基本原理进行信息处理和计算。与经典计算机使用比特（0 或 1）作为基本信息单位不同，量子计算使用量子比特（qubit）作为基本信息单位。量子比特可以处于 0、1 或者它们的线性叠加态，即同时处于多个状态。这使得量子计算机在理论上可以进行高度并行的计算，从而在某些问题上比经典计算机更加高效。

  量子计算中的基本操作是量子门（Quantum Gates），类似于经典计算中的逻辑门。量子门在量子比特上执行特定的操作，例如保持不变、翻转或者实现两个量子比特之间的纠缠。这些操作都是可逆的，并且满足一定的幺正性质。通过组合这些基本量子门，可以构建复杂数学和逻辑运算的量子算法。

  量子计算的一个显著优势是在某些问题上可以显著超越经典计算机的性能。著名的量子算法包括 Shor 算法（用于整数分解和解决 RSA 加密问题）和 Grover 算法（用于无序数据库搜索）。这些算法在量子计算机上的运行速度比经典计算机快得多，为解决许多实际问题提供了全新的途径。

  然而，量子计算也面临着许多挑战，如实现可扩展的、可控的量子硬件以及解决与量子比特相互作用和保真度相关的问题。尽管如此，量子计算仍被认为是未来计算技术的一个重要方向。

[14] 局部实效原理（Local Realism）是指在经典物理学中，物体或系统之间的相互作用必须是局部的，即它们之间的作用是通过在空间和时间上相互接近的点之间传递信息或者因果关系来实现的。换句话说，在经典物理世界中，系统之间的相互作用不可能瞬间跨越很远的距离。

局部实效原理基于两个基本概念：

实效性（Realism）：这个概念表示物体的属性或状态在进行测量之前就已经确定，不依赖于观察者的知识或测量行为。局部性（Locality）：这个概念表示在空间上分离的物体之间的相互作用不能瞬时发生，而是受到光速的限制，即信息不能超过光速传播。

- **量子通信**：量子通信是量子信息理论中的一个重要领域，它利用量子力学的原理进行信息的传输和处理。与经典通信系统使用经典比特（0 或 1）作为信息载体不同，量子通信使用量子比特（qubit）作为信息载体，通常采用光子（光量子）来实现量子比特的传输。

  量子通信的主要优势在于它提供了一种具有更高安全性的通信方式。在量子通信中，信息的传输过程利用了量子纠缠和量子隐形传态等现象，使得在理论上可以实现无条件安全的通信。这是因为任何对量子信道的监听或窃取行为都会导致信道状态的改变，从而被通信双方察觉。这种安全性是经典通信系统无法实现的。

  量子通信的一个典型应用是量子密钥分发（Quantum Key Distribution, QKD）。量子密钥分发利用量子通信技术在通信双方之间分发密钥，以实现安全的加密通信。著名的量子密钥分发协议包括 BB84 协议[15]和 E91 协议[16]等。量子密钥分发系统已经在一些实际场景中得到了应用，如银行、政府和军事通信等领域。

  虽然量子通信具有显著的优势，但它仍然面临着许多技术挑战，如提高信道传输距离、降低误码率和实现大规模量子网络的构建等。研究人员正致力于克服这些挑战，以实现量子通信在更广泛领域的应用。

- **量子密码学**：量子密码学（Quantum Cryptography）是一个研究如何利用量子力学原理来实现信息安全的学科。量子密码学的主要目标是为通信双方提供无条件安全的通信方式，即使在面对拥有无限计算能力的攻击者时也能保证通信的安全性。

  量子密码学最著名的应用领域是量子密钥分发（Quantum Key Distribution, QKD）。量子密钥分发利用量子通信技术在通信双方之间安全地传输密钥，以实现安全的加密通信。在量子密钥分发过程中，由于量子测量的不可克隆性和不可逆性，任何试图窃听通信的行为都会引入错误，从而被通信双方察觉。这种安全性是经典密码学和通信系统无法实现的。

  量子密码学包括以下主要部分：

  - **量子密钥分发**：如前述，量子密钥分发是量子密码学的核心应用。著名的量子密钥分发协议包括 BB84 协议和 E91 协议等。
  - **量子隐形传态**：量子隐形传态是利用量子纠缠现象实现在远程的两个地点之间传输未知量子态的一种技术。在量子隐形传态中，通信双方需要共享一对纠缠的量子比特，并利用经典通信信道来传输关于测量结果的信息。
  - **量子隐写术**：量子隐写术是一种使用量子信道来隐藏信息的技术。通过量子隐写术，可以将信息“隐藏”在量子信道中，使窃听者无法察觉到信息的存在。
  - **量子签名**：量子签名是一种基于量子力学原理的数字签名技术。与经典数字签名相比，量子签名在理论上可以提供更强的安全性。

  尽管量子密码学在理论上具有无条件安全的优势，但在实际应用中，由于设备的缺陷、信道损耗等因素，量子密码学的安全性仍然需要通过严密的数学证明和实际测试来保障。量子密码学在政府、军事、金融等领域具有广泛的应用潜力。

[15] BB84 协议是量子密钥分发（Quantum Key Distribution, QKD）领域的一个著名协议，由查尔斯 · 贝内特（Charles H. Bennett）和吉列斯 · 布拉萨德（Gilles Brassard）于 1984 年提出，因此得名 BB84。该协议利用量子力学原理实现密钥在通信双方之间的安全分发，保障通信的安全性。

[16] E91 协议是一个量子密钥分发（Quantum Key Distribution, QKD）协议，由 Artur Ekert 于 1991 年提出，因此得名 E91。该协议利用量子纠缠的特性实现密钥在通信双方之间的安全分发，保障通信的安全性。E91 协议与之前提到的 BB84 协议有相似之处，但在原理上存在一些不同。

- **量子纠错**：量子纠错（Quantum Error Correction）是量子信息处理领域中的一个重要概念，它主要研究如何在量子计算和量子通信过程中纠正由于噪声和其他误差引入的错误。由于量子比特（qubit）非常敏感，容易受到外部环境和设备缺陷的影响，因此在量子计算和量子通信过程中保持量子态的稳定性是至关重要的。量子纠错技术正是为解决这一问题而发展起来的。

  量子纠错的基本原理与经典纠错编码有相似之处，都是通过冗余编码的方式来检测和纠正错误。然而，量子纠错面临着比经典纠错更为复杂的挑战。量子比特可以处于 0、1 或者它们的线性叠加态，这使得量子纠错编码需要考虑更多的情况。此外，由于量子力学的不确定性原理和测量的不可逆性，量子纠错过程中不能直接对量子态进行测量，否则会破坏量子态本身。

  量子纠错编码的一个重要概念是量子纠错码（Quantum Error-Correcting Code），它是一种将逻辑量子比特编码为多个物理量子比特的方法，从而在物理量子比特出现错误时，可以通过冗余信息检测和纠正错误。著名的量子纠错码包括 Shor 算法，如算法1[17]和 Steane 代码，如算法2[18]等。

  在量子纠错过程中，除了需要设计有效的纠错码之外，还需要开发一系列用于错误检测和纠正的量子算法和量子门。这些算法和量子门应当满足一定的幺正性质，以保证在纠错过程中不引入新的错误。

  量子纠错是实现大规模、可靠的量子计算和量子通信的关键技术之一。尽管目前的量子纠错技术仍然面临许多挑战，但随着量子信息技术的不断发展，量子纠错的应用前景十分广阔。

Shor 代码的纠错过程包括以下几个步骤：

1. **对编码后的 9 个物理量子比特进行错误检测，这通常通过一系列受控非门（CNOT 门）和受控受控非门（Toffoli 门）来实现。在此过程中，需要注意的是不能直接对量子比特进行测量，以避免破坏量子态本身。**
2. 分析错误检测的结果，确定错误的类型和位置。
3. 应用适当的量子门（如 X 门或 Z 门）来纠正检测到的错误。
4. 对修复后的量子比特进行解码，恢复原始的逻辑量子比特。

Steane 代码的纠错过程包括以下几个步骤：

1. **对编码后的 7 个物理量子比特进行错误检测。这通常通过一系列受控非门（CNOT 门）以及辅助量子比特来实现。在此过程中，同样需要注意不能直接对量子比特进行测量，以避免破坏量子态本身。**
2. 分析错误检测的结果，确定错误的类型和位置。
3. 应用适当的量子门（如 X 门或 Z 门）来纠正检测到的错误。
4. 对修复后的量子比特进行解码，恢复原始的逻辑量子比特。

[17]Shor 代码是一种量子纠错码，由 Peter Shor 于 1995 年首次提出。Shor 代码是一种能够纠正任意单个量子比特错误的纠错码。Shor 代码的提出标志着量子计算领域的一个重要突破，因为它证明了在量子计算过程中可以纠正错误，从而增强了实现大规模量子计算机的可行性。

[18]Steane 代码是一种量子纠错码，由 Andrew Steane 于 1996 年首次提出。Steane 代码是一种能够纠正任意单个量子比特错误的纠错码，与 Shor 代码一样，它可以同时检测和纠正单个量子比特的位翻转错误（X 错误）和相位翻转错误（Z 错误）。Steane 代码的提出进一步推动了量子纠错领域的发展。

Algorithm 1 Shor's Code Error Correction (量子纠错：Shor 代码)

procedure ShorErrorCorrection($|\psi\rangle$)
    准备辅助量子比特，状态为 $|0\rangle$
    使用 Shor 代码对 $|\psi\rangle$ 进行编码
    应用 CNOT 门进行错误检测
    测量辅助量子比特
    利用测量结果确定错误的类型和位置
    if 检测到位翻转错误 **then**
        应用 X 门进行纠正
    end if
    if 检测到相位翻转错误 **then**
        应用 Z 门进行纠正
    end if
    对纠正后的量子比特进行解码，恢复原始状态 $|\psi\rangle$
end procedure

Algorithm 2 Steane's Code Error Correction (量子纠错：Steane 代码)

procedure SteaneErrorCorrection($|\psi\rangle$)
    准备辅助量子比特，状态为 $|0\rangle$
    使用 Steane 代码对 $|\psi\rangle$ 进行编码
    应用 CNOT 门进行错误检测
    测量辅助量子比特
    利用测量结果确定错误的类型和位置
    if 检测到位翻转错误 **then**
        应用 X 门进行纠正
    end if
    if 检测到相位翻转错误 **then**
        应用 Z 门进行纠正
    end if
    对纠正后的量子比特进行解码，恢复原始状态 $|\psi\rangle$
end procedure

量子信息理论是一个充满挑战和潜力的领域，它为未来信息技术的发展提供了全新的可能。随着量子计算、量子通信和量子密码学等技术的不断发展，量子信息理论有望在许多领域产生深远的影响，如密码安全、高性能计算、人工智能等。

1.1.3. 计算机科学与量子计算

计算机科学与量子计算是两个相关的但有所区别的领域。计算机科学是研究计算机系统设计、软件开发和信息处理的理论和实践的学科。它包括各种子领域，如算法设计、数据结构、编程语言、操作系统、数据库管理、人工智能等。计算机科学的核心是经典计算理论，基于经典比特（bit）进行信息处理。

量子计算则是计算机科学的一个子领域，专注于开发和理解基于量子力学原理的新型计算模型。量子计算利用量子比特（qubit）作为信息单位，它可以同时处于多个状态（即量子叠加态），从而在某些问题上实现比经典计算更快的速度。量子计算涉及到量子算法设计、量子计算机硬件实现、量子纠错等方面的研究。

计算机科学与量子计算的关系可以总结为以下几点：

- 量子计算是计算机科学的一个子领域，但它采用了截然不同的计算模型。
- 量子计算基于量子力学原理，而计算机科学主要依赖于经典物理学。
- 计算机科学与量子计算之间存在一定的交叉与互动。例如，计算机科学中的算法设计和优化方法可以应用于量子算法，而量子计算的发展也可能对计算机科学中的问题提供新的解决方案。
- 量子计算的发展可能对计算机科学的未来产生深远影响。如果实现可扩展的量子计算机，它可能会在许多领域（如密码学、优化问题等）实现比经典计算更高效的解决方案，从而推动计算机科学的进步。
- 量子计算对计算机科学的教育和研究方法产生影响。随着量子计算的发展，计算机科学专业可能需要加入量子计算相关的课程，以便学生能够掌握这一领域的基本概念和技能。此外，研究人员需要开发新的模拟工具和编程语言来实现量子算法和量子计算机的设计。
- 计算机科学与量子计算在硬件设计方面有所区别。传统计算机硬件基于经典物理学原理，如晶体管和集成电路。而量子计算硬件则需要利用量子力学现象，如超导量子比特、离子阱、光子量子比特等。这些技术在原理和实现上与传统计算机硬件有很大差别。
- 计算机科学与量子计算在工业应用方面有所区别。传统计算机科学技术已经广泛应用于各个行业，如金融、医疗、通信等。而量子计算目前尚处于发展初期，尽管在某些特定问题上具有潜在优势，但其在工业应用方面的普及和实际价值仍需时间和技术突破。
- 计算机科学与量子计算在安全性和隐私保护方面有所区别。量子计算对现有的密码学体系产生挑战，如 Shor 算法可以在量子计算机上快速分解大整数，从而破解 RSA 等加密算法。这促使计算机科学研究人员开发新的抗量子加密技术，如基于格的密码学、基于哈希的签名算法等。

计算机科学与量子计算是两个相互关联但有所区别的领域。量子计算作为计算机科学的一个子领域，以量子力学原理为基础，具有潜在的并行性和高效性。量子计算的发展可能对计算机科学产生深远的影响，推动信息技术领域的进步。

对于我们的目的，我们将重点关注量子计算机与经典计算机的区别所在。

每台经典计算机（即非量子计算机）都可以用量子力学来描述，因为量子力学是物理宇宙的基础。然而，经典计算机没有利用量子力学所提供的特定属性和状态来进行计算。

为了深入研究我们在量子计算机中使用的特定属性，让我们先讨论量子力学的一些关键概念：

- 如何表示量子系统中的叠加态?
- 什么是纠缠?
- 可逆性、计算和物理系统之间的联系是什么?

1.2. 量子力学基础

在本课程中，我们广泛使用狄拉克符号、线性代数和其他工具；读者被鼓励在需要时参考本文后面的数学章节进行复习。

1.2.1. 狄拉克符号

狄拉克符号（Dirac notation），又称为 bra-ket 表示法，是一种用于描述量子力学中的态和算符的简洁符号体系。这种表示法由著名物理学家保罗 · 狄拉克（Paul Dirac）[19]引入，以便更方便地处理量子力学中的数学计算。

在狄拉克符号中，量子态用一个被称为 ket 的符号表示，如：$|\psi\rangle$。这表示一个量子态，可以是一个量子比特（qubit）或多个量子比特组成的系统。另一方面，bra 符号表示一个复共轭[20]的态，如：$|\psi\rangle$。通过 bra 和 ket 的组合，可以计算内积、外积以及算符作用等操作。

狄拉克符号在量子计算中具有重要意义，因为它为处理量子比特（qubit）和量子门等核心概念提供了一种简洁有效的表示方法。量子计算中的基本单位是量子比特，其状态可以用狄拉克符

[19]**保罗 · 狄拉克**（Paul Dirac，1902 年 8 月 8 日 – 1984 年 10 月 20 日）是一位英国理论物理学家，被认为是 20 世纪最重要的物理学家之一。他在量子力学和量子场论方面的开创性工作为现代物理学的发展奠定了基础。狄拉克的研究涉及许多领域，包括电动力学、粒子物理、引力理论以及数学物理。

狄拉克最著名的贡献之一是狄拉克方程，这是一个描述相对论性电子行为的方程。狄拉克方程成功地将量子力学与狭义相对论结合起来，为电子和其他基本粒子的性质提供了深刻的理解。狄拉克方程的一个重要结果是预言了电子的反粒子——正电子的存在，这在后来的实验中得到了证实。

狄拉克还对量子力学的数学表示和形式体系做出了重要贡献。他引入了一种称为狄拉克符号（bra-ket 表示法）的表示方法，用以简化量子力学的数学计算。这种表示法在现代物理学和量子计算中被广泛应用。

狄拉克还对粒子物理学的发展产生了重要影响。他提出了狄拉克海这一概念，即所有负能量电子状态的集合。这一理论为量子场论的发展奠定了基础，并为物质和反物质之间的对称性提供了理论支持。

狄拉克在 1933 年与奥地利物理学家埃尔温 · 薛定谔共同获得了诺贝尔物理学奖，以表彰他们在原子理论和量子力学方面的杰出贡献。狄拉克对现代物理学的影响是深远的，他的工作为量子力学、相对论性粒子物理学以及量子信息科学等领域的发展奠定了基础。

[20]**复共轭**（complex conjugate）是一种与给定复数相关的另一个复数，它的实部与原复数相同，但虚部的符号相反。复共轭的概念主要用于复数运算，尤其是在复平面上表示和操作复数时非常有用。

给定一个复数 z，表示为：

$$z = a + bi$$

其中 a 和 b 是实数，i 是虚数单位，满足 $i^2 = -1$。那么 z 的复共轭表示为：

$$z* = a - bi$$

可以看到，实部相同，而虚部的符号相反。

复共轭在复数运算中有许多重要性质。例如：

- 两个复数的乘积与它们的复共轭的乘积相等：$(z_1 z_1)^* = z_1^* z_2^*$；
- 一个复数与其复共轭的乘积等于该复数的模的平方：$|z|^2 = zz^*$；
- 如果一个复数的实部或虚部为零，则该复数等于其复共轭；
- 两个复数的和的复共轭等于它们各自复共轭的和：$(z_1 + z_1)^* = z_1^* + z_2^*$。

在量子力学和量子计算中，复共轭在表示和处理复数概率振幅时起到重要作用。例如，在狄拉克符号中，bra 表示一个量子态的复共轭，用于计算量子态的内积和概率分布。

号表示为：

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{1.2}$$

其中，α 和 β 是复数，它们的模平方之和等于 1（即 $|\alpha|^2 + |\beta|^2 = 1$）。$|0\rangle$ 和 $|1\rangle$ 分别表示量子比特的基本态，类似于经典计算中的 0 和 1。

在量子计算中，狄拉克符号还可以用于表示量子门（quantum gate）。量子门是量子计算中的基本操作，可以理解为量子比特状态的变换。例如，Pauli-X 门可以表示为：

$$\mathsf{X} = |0\rangle\langle 1| + |1\rangle\langle 0| \tag{1.3}$$

在这里，X 是一个 2x2 的矩阵，可以作用于一个量子比特的状态，从而实现状态的变换。

狄拉克符号在量子计算中起到了关键作用，它为表示和处理量子态、量子门等核心概念提供了一种简洁有效的方法。这使得量子计算的理论研究和实际实现更加清晰和直观。

1.2.2. 量子计算基础

根据量子力学的原理，只有在进行测量时，系统才被设置为一个确定的状态。在测量之前，系统处于不确定的状态；在我们测量它们之后，它们就处于一个确定的状态。如果我们有一个系统，当进行测量时可以采取两种离散状态之一，我们可以用狄拉克符号表示这两种状态，如 $|0\rangle$ 和 $|1\rangle$。然后，我们可以将状态的叠加表示为这些状态的线性组合，例如公式1.2, 其中，α 和 β 是复数，且满足

$$\alpha^2 + \beta^2 = 1 \tag{1.4}$$

，以确保总概率为 1。

公式1.2是量子计算中的一个基本概念，表示一个量子比特（qubit）的量子态。在量子计算中，量子比特是基本的信息单位，与经典计算中的比特（bit）类似。然而，与经典比特不同的是，量子比特可以同时处于 0 和 1 的状态，这种现象被称为**量子叠加**。

在这个公式中，$|0\rangle$ 和 $|1\rangle$ 分别表示量子比特的基本态，即类似于经典比特中的 0 和 1。而 $|\psi\rangle$ 是一个一般的量子态，可以通过基本态的线性组合表示。α 和 β 是复数系数，它们的模平方之和等于 1，这个条件确保了量子态的概率解释是合理的。α 和 β 的值决定了量子态 $|\psi\rangle$ 的具体形式。当我们测量这个量子态时，它会坍缩为 $|0\rangle$ 或 $|1\rangle$，并以 $|\alpha|^2$ 的概率得到 $|0\rangle$，以 $|\beta|^2$ 的概率得到 $|1\rangle$，即一个量子比特可以同时处于多个状态的线性组合。这种特性使得量子计算具有潜在的并行性和高效性，可以在某些问题上实现比经典计算更快的速度。

在量子系统中，我们可以将多个粒子纠缠在一起，这意味着它们之间存在一种关联性，无论它们之间有多远的距离。例如，如果两个粒子是纠缠的，当我们改变其中一个粒子的状态时，另一个粒子的状态也会相应地改变，即使它们之间相隔数千万英里。

量子计算机的关键优势在于它们能够在计算中利用这种纠缠现象和叠加态。另外，量子计算机中的运算通常是可逆的，这意味着可以通过相反的运算将计算结果。

当我们将一个量子系统测量时，它会坍缩到一个确定的状态。但在测量之前，它处于叠加态，可以表示为一组基矢量的线性组合：

$$|\psi\rangle = \sum_i c_i |i\rangle \tag{1.5}$$

其中 c_i 是一组复数系数，$|i\rangle$ 是一组基矢量。在测量之前，我们无法知道系统处于哪个基矢量上，只能计算出处于每个基矢量的概率为 $|c_i|^2$。在测量之后，系统会坍缩到一个确定的基矢量上，概率等于 $|c_i|^2$。

在量子计算机中，我们利用叠加态和量子纠缠来加速计算。量子计算机的计算方式与经典计算机不同，可以利用叠加态和纠缠来处理信息，从而在某些情况下比经典计算机更高效。

1.2.3. 叠加原理

举例来说，让我们考虑光的一个性质，它展示了状态的叠加。光具有一种固有属性，称为偏振，我们可以用它来展示状态的叠加。在几乎所有日常生活中看到的光线（例如来自太阳的光），没有偏振的优先方向。偏振状态可以通过偏振滤波器来选择，这是一种带有轴的薄膜，只允许与该轴平行的偏振光通过。叠加态是指一个量子系统处于多个基态的线性组合。叠加态是量子计算中的核心概念，它体现了量子系统在某种程度上可以同时存在于多个状态，这与经典计算中的二进制位（0 或 1）有很大的不同。

使用单个偏振滤波器，我们可以选择光的一个偏振，例如垂直偏振，我们可以表示为 $|\uparrow\rangle$. 水平偏振，我们可以表示为 $|\rightarrow\rangle$，是垂直偏振的正交状态。这些状态一起形成了任何偏振光的基础。也就是说，任何偏振状态 $|\psi\rangle$ 都可以被写成这些状态的线性组合。我们使用希腊字母来表示系统的状态

$$|\psi\rangle = \alpha\,|\uparrow\rangle + \beta\,|\rightarrow\rangle \tag{1.6}$$

公式1.6中系数 α 和 β 是称为振幅的复数。系数 α 与垂直偏振相关，系数 β 与水平偏振相关。它们在量子力学中有重要的解释。在选择垂直偏振的偏振滤波器之后，我们可以引入第二个偏振滤波器。假设我们将第二个滤波器的轴与第一个垂直。那么第二个滤波器会透过多少光？如果你对这个问题的答案是否定的，那么你是正确的。水平状态 $|\rightarrow\rangle$ 是第一个状态的正交，因此第一个垂直滤波器后没有任何水平偏振。

现在假设我们将第二个偏振滤波器的轴朝着第一个的对角线（即垂直 $\uparrow$ 和水平 $\rightarrow$ 之间的对角线）方向旋转 45°。现在我们问同样的问题——第二个滤波器会透过多少光？如果你对这个问题的答案是否定的，你可能会惊讶地发现答案是肯定的。实际上，我们会看到一些光透过第二个滤波器。如果所有经过第一个滤波器的光都是垂直偏振，那么这是如何发生的？原因在于我们可以将垂直偏振表示为对角线分量的叠加。也就是说，让 $|\nearrow\rangle$ 表示 45° 偏振，让 $|\nwarrow\rangle$ 表示 -45° 偏振，我们可以写成：

$$|\uparrow\rangle = \frac{1}{\sqrt{2}}\,|\nearrow\rangle + \frac{1}{\sqrt{2}}\,|\nwarrow\rangle \tag{1.7}$$

从几何直觉上预期，垂直状态包含相等部分的 $|\nearrow\rangle$ 和 $|\nwarrow\rangle$。

正是因为这个原因，我们会看到一些光透过第二个滤波器。也就是说，垂直偏振可以被写成状态的叠加，其中一个状态恰好是我们允许通过第二个滤波器的 45° 对角线状态 $|\nearrow\rangle$。由于 $|\nearrow\rangle$ 只是叠加中的一个术语，因此并不是所有的光都通过滤波器，但是一些光确实通过了。在这种情况下，传输的量恰好是 $1/2$。（更正式地说，传输光的强度是入射光的 $1/2$。）这个值是由 Born 法则确定的，我们现在来讨论。

马克斯·玻恩（Max Born）[21]在他 1926 年的论文中证明，状态振幅的模平方是测量后该状态

[21] **马克斯·玻恩**（Max Born，1882 年 - 1970 年）是一位德国籍的英国理论物理学家和数学家，因在量子力学领域的开创性贡献而获得广泛认可。他在量子力学的建立和发展过程中扮演了关键角色，并因此荣获了 1954 年的诺贝尔物理学奖。

玻恩出生于德国汉诺威，曾在哥廷根大学、柏林洪堡大学和爱丁堡大学等学府学习和工作。他的研究生导师是著名数学家大卫·希尔伯特（David Hilbert），这使得玻恩在数学和理论物理方面接受到了良好的教育。

出现的概率 [38]。在这种情况下，由于振幅是 $1/\sqrt{2}$，因此测量到该状态的概率为 $1/2$。注意，我们选择了振幅为 $1/\sqrt{2}$，以使状态归一化，这样振幅的模平方之和将等于一。这使我们能够使用玻恩法则将振幅连接到测量的概率上。

1.3. 玻恩规则

在一个量子态的叠加中，一个态的振幅模平方是测量得到该态的概率。此外，叠加中所有可能态的振幅模平方之和等于 1。所以，对于态 $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$，我们有：

$$|\alpha|^2 + |\beta|^2 = 1$$

虽然在上面的偏振示例中，两个态各有 50% 的概率，但如果我们观察其他物理系统，可能会有 75% 的概率分布或其他概率分布。经典力学和量子力学之间的一个关键区别是振幅（而非概率）可以是复数。

换句话说，出现在玻恩规则陈述中的系数 α 和 β 可以是复数，例如 $i = \sqrt{-1}$ 或 $(1+i)\sqrt{-1}$ 。只有在我们取这些振幅的模平方时，我们才得到实数，即实际的概率。请参阅第 11 章以复习复数及如何确定复数模平方。

量子叠加本身就足够奇特，量子力学描述了一种特殊类型的叠加，这进一步挑战了我们的想象力：纠缠。

1935 年，当爱因斯坦与波多尔斯基和罗森共同发表了关于量子纠缠的论文 [19] 时，他们的目标是攻击量子力学的基石（这篇论文现在被称为 EPR 论文）。尽管爱因斯坦因 1905 年关于光电效应的量子性质的工作获得了诺贝尔奖，但他直到晚年仍在抨击量子力学的含义。

爱因斯坦在 1952 年写道，量子力学对他来说似乎是

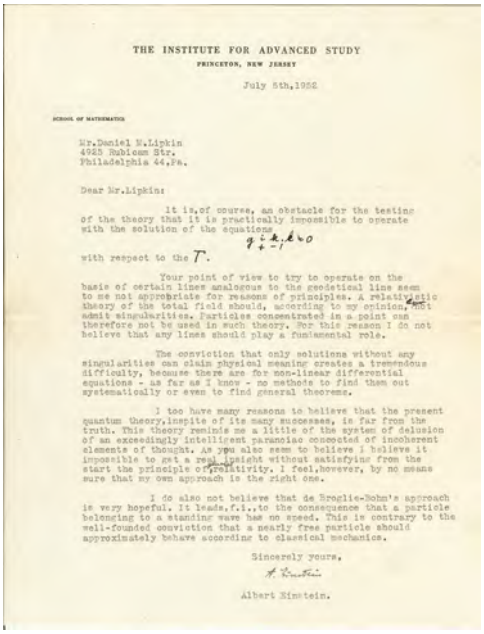
“a system of delusion of an exceedingly intelligent paranoiac concocted of incoherent elements of thought（一个极度聪明的妄想狂用不连贯的思想元素捏造的幻觉体系）”

如图1.5。他希望 EPR 论文能证明他认为的量子力学的不足之处。

在 20 世纪初，玻恩开始了他在量子力学领域的研究。他与其他科学家，如沃纳 · 海森堡（Werner Heisenberg）和尼尔斯 · 玻尔（Niels Bohr）等，一起发展了这一革命性的理论体系。在 1920 年代，玻恩对波恩概率解释的提出和数学公式的精确描述，成为量子力学的核心组成部分。玻恩概率解释说明了波函数的平方与粒子存在的概率密度之间的关系。

除了量子力学领域的贡献外，马克斯 · 玻恩还在晶体学、固态物理学和光学等领域取得了成果。在第二次世界大战期间，由于他的犹太血统，他逃离了德国，并在英国继续了他的学术生涯。1954 年，玻恩因在量子力学领域的贡献获得了诺贝尔物理学奖。

马克斯 · 玻恩是一位在物理学领域产生深远影响的科学家，他的研究对量子力学的发展起到了关键性作用。



THE INSTITUTE FOR ADVANCED STUDY
PRINCETON, NEW JERSEY

July 5th,1952

SCHOOL OF MATHEMATICS

Mr.Daniel M.Lipkin
4925 Rubicam Str.
Philadelphia 44,Pa.

Dear Mr.Lipkin:

It is,of course, an obstacle for the testing of the theory that it is practically impossible to operate with the solution of the equations

$$g_{\mu\nu,k}+g_{k\nu,\mu}+g_{\mu k,\nu}=0$$

with respect to the Γ.

Your point of view to try to operate on the basis of certain lines analogous to the geodetical line seem to me not appropriate for reasons of principles. A relativistic theory of the total field should, according to my opinion, not admit singularities. Particles concentrated in a point can therefore not be used in such theory. For this reason I do not believe that any lines should play a fundamental role.

The conviction that only solutions without any singularities can claim physical meaning creates a tremendous difficulty, because there are for non-linear differential equations - as far as I know - no methods to find them out systematically or even to find general theorems.

I too have many reasons to believe that the present quantum theory,inspite of its many successes, is far from the truth. This theory reminds me a little of the system of delusion of an exceedingly intelligent paranoiac connected of incoherent elements of thought. As you also seem to believe i believe it impossible to get a real insight without satisfying from the start the principle of relativity. I feel,however, by no means sure that my own approach is the right one.

I do also not believe that de Broglie-Bohm's approach is very hopeful. It leads,i.a.,to the consequence that a particle belonging to a standing wave has no speed. This is contrary to the well-founded conviction that a nearly free particle should approximately behave according to classical mechanics.

Sincerely yours,

Albert Einstein.

图 1.5: 爱因斯坦给丹尼尔 · M · 利普金的签名信，普林斯顿，1952 年 7 月 5 日。图片来自于 christieshttps://www.christies.com/en/lot/lot-6210437

ERP 论文 [19] 表明，如果你拿两个相互纠缠的粒子，然后测量其中一个，这将自动触发第二个粒子的相关状态，即使两者相距很远；这是 EPR 希望用来证明量子力学本身存在缺陷的看似不合逻辑的结果。讽刺的是，我们现在认为纠缠是量子力学的基石。纠缠发生在不可分离的态的叠加中。

这种“鬼魅般的距离作用”似乎与我们的直觉和先前的物理学相悖。据报道，合著者中年龄最小的波多尔斯基将论文泄露给《纽约时报》，以便向公众突显这对量子力学大厦的攻击。1935 年 5 月 4 日，《纽约时报》以“爱因斯坦攻击量子理论”为标题，在头版刊登了这篇报道。

如今，纠缠不仅被接受为标准量子力学的一部分，我们还将在后面的章节中看到，我们可以利用纠缠来进行新颖类型的计算和通信。从信息论的角度来看，纠缠是一种不同的信息编码方式。如果我们有两个纠缠在一起的粒子，关于它们的信息不是局部地编码在每个粒子中，而是编码在两者之间的关联中。

约翰 · 普雷斯基尔喜欢用两种书的比喻来解释纠缠和非纠缠 [20] [22] ：在普通的、非纠缠的书中，我们可以像平常一样阅读每一页的信息。然而，在纠缠的书中，每一页都包含看似胡言乱语的内容。信息被编码在页面之间的关联中，而不是每一页单独的内容。这正是薛定谔在创造“纠缠”这个术语时所表达的：

[22]这篇论文的主要内容是关于”NISQ（Noisy Intermediate-Scale Quantum）” 时代及其之后量子计算的研究，作者 John Preskill 提出了许多观点和建议。
论文的主要观点如下：
NISQ 时代的量子计算机具有一定的规模和噪声，因此其能力相对有限，但仍然有潜力在某些领域取得突破。
发展量子纠缠和量子误差校正等技术，可以提高量子计算机的可靠性和性能。
发展量子模拟和量子优化等领域的算法，可以在实际问题中展现量子计算机的优势。
在 NISQ 时代中，需要进一步发展量子硬件和量子软件，同时也需要建立一些实际问题的测试平台，以便在实际应用中测试量子计算机的性能。
在未来，量子计算机可能会推动量子物理、量子信息和量子工程等领域的发展，从而引领新的科技革命。
论文的主要实验手段是理论分析和综述，对已有的研究成果进行了总结和归纳。作者提出的观点和建议，对于当前和未来的量子计算机研究具有重要的指导作用。
该论文的主要贡献是提出了 NISQ 时代及其之后量子计算的研究方向和目标，对量子计算机的发展和应用具有指导作用。

另一种表达这种奇特情况的方式是：”对整体的最佳可能了解并不一定包括对所有部分的最佳可能了解。”[21]

薛定谔[23]进一步指出，在他看来，纠缠不仅仅是量子力学描述的现象之一，“而是量子力学的特征，使其完全脱离了经典思维模式。”[21]。

1.4. 纠缠

如果两个系统的测量结果以比经典世界更强的方式与另一个系统的状态相关，那么这两个系统处于被称为纠缠的量子力学叠加态的特殊情况。换句话说，这两个系统的状态是不可分的。

现在我们已经介绍了量子力学的两个核心思想——叠加和纠缠，让我们转向另一个不太常被涉及的基本概念——**信息的物质性**。当**罗尔夫·兰道尔**[24]提出以下问题时，他开启了一条新的探索方向：寻求更快、更紧凑的计算电路，直接引发了这样一个问题：

“这方面的进展在物理上有什么终极限制？... 我们可以证明，或者至少强烈暗示，信息处理过程中不可避免地伴随着一定程度的热量产生。[22]”

换句话说，计算基本单位过程中能量损耗的下限是多少？由于兰道尔和其他人的研究，我们现在相信确实存在这样一个限制；这被称为兰道尔界限（LB）。更具体地说，擦除 n 位比特的能量成本是 $nkT\ln 2$，其中 k 是玻尔兹曼常数，T 是以开尔文表示的计算设备周围热汇的温度，$\ln 2$ 当然是 2 的自然对数（约 0.69315）。这个限制是不可逆计算的最小能量损耗。

兰道尔承认，这个最小值不一定是系统能量消耗的限制因素。

当然，很明显，热噪声和能量耗散的要求在目前计算机组件中的比例是完全可以忽略的。然而，计算出的耗散是一个绝对最小值。[22]

兰道尔将逻辑不可逆定义为“设备的输出不能唯一确定输入”的条件。然后他声称，“逻辑不可逆性... 反过来意味着物理不可逆性，而后者伴随着耗散效应。”这是由热力学第二定律推导出的，该定律指出，一个系统的总熵不能减少，而且更具体地说，在不可逆过程中必须增加。关于可逆性、热力学和计算的更多背景知识，请参阅费曼的《计算讲义》[84]

在经典计算中，我们使用不可逆计算。例如，布尔”或”（表示为 ∨）门具有以下真值表，其中 0 表示“假”，1 表示“真”：

注意，输出值 1 不能唯一地追溯到一组输入。我们可以通过输入组合得到该输出；一旦转到输出，输入状态就丢失了。这并没有违反信息守恒，因为信息已经转化为耗散热量。

亦或门也是不可逆的，同样适用于经典计算中的通用 NAND 门。NAND 表示“非与”，是布尔与运算符的逆。通过检查它的真值表来验证 NAND 是不可逆的：

[23]**薛定谔**（Erwin Schrödinger，1887 年 8 月 12 日 – 1961 年 1 月 4 日），是一位奥地利物理学家，曾获得 1933 年的诺贝尔物理学奖。他被誉为现代量子力学的奠基人之一，对于波动力学方程的建立做出了重要贡献。

薛定谔在 1918 年提出了著名的薛定谔方程，用以描述量子体系的运动和行为。他也因此被誉为“波动力学之父”，并成为了量子力学的重要代表人物之一。他还提出了著名的“薛定谔的猫”思想实验，用来探讨量子力学中的测量问题和量子态的纠缠问题。

薛定谔在物理学领域的贡献不仅限于量子力学，他还在许多其他领域做出了重要的贡献，如在物态方程、热力学、色散理论、宇宙学等领域。

薛定谔的学术成就对于现代物理学的发展有着深远的影响。他不仅在理论物理学方面取得了杰出的成就，而且还对科学哲学、科学教育和科学传播做出了重要的贡献。

[24]**罗尔夫·兰道尔**，Rolf Landauer (1927–1999) 是一位德国裔美国物理学家，他在信息理论和统计物理学方面作出了重大贡献。他在 IBM 托马斯·J·沃森研究中心工作了大部分的职业生涯。

罗尔夫最为人所知的贡献是他在 1961 年提出的罗尔夫原理，该原理阐述了在某些条件下，计算机操作的物理限制。他的理论指出，信息的擦除在物理上必然伴随着能量的耗散，这与当时的普遍观点相反。这一原理在纳米尺度科学和量子计算机领域具有重要意义。

他的另一个重要贡献是关于电子在固体中输运的理论，这对理解和设计微电子设备非常重要。Landauer 的工作对现代信息和计算理论产生了深远影响。

X	Y	X ∨ Y
0	0	0
0	1	1
1	0	1
1	1	1

X	Y	X	Y
0	0	1	
0	1	1	
1	0	1	
1	1	0	

“在量子计算中，我们将自己限制在可逆逻辑运算上”[23] 这句话的意思是，在量子计算中，我们确实经常将自己限制在可逆逻辑运算上。这是由于量子机械的基本动力学是可逆的，也就是说，如果你知道一个系统的当前状态，你就可以准确地推算出它的过去和未来状态。因此，量子计算的基本操作——量子门，也必须是可逆的。

在经典计算中，许多逻辑运算都是不可逆的。例如，如果你看到一个 AND 门的输出是 0，你无法确定输入是什么。然而，在量子计算中，所有的操作都必须是可逆的。这意味着，如果你知道输出，你就可以准确地推算出输入。例如，量子计算中的 CNOT 门（受控非门）就是一种可逆逻辑门。

这种限制在可逆逻辑运算的原因还有另一个层面，那就是罗尔夫 · 兰道尔提出的兰道尔原理，该原理表明，每个不可逆的运算都会导致能量的耗散。因此，基于量子力学的可逆性和能源效率的考虑，量子计算通常都限制在可逆逻辑运算上。稍后在本书中，我们将考虑哪些量子算子组合是通用的。现在，让我们专注于量子计算中的要求，将我们的算子集限制为可逆门。

这个要求源于不可逆操作的性质：如果我们执行不可逆操作，我们就丢失了信息，因此测量结果。此时我们的计算周期将结束，我们将无法继续执行程序。相反，通过将所有门限制为可逆算子，只要我们能保持系统的相干性，我们就可以继续将算子应用于我们的量子比特集。当我们说可逆时，我们指的是理论上无噪声的量子计算机。在一个有噪声的量子计算机中，我们当然无法反转操作。

1.5. 量子计算的可逆性

除测量外，量子计算中使用的所有算子都必须是可逆的。

量子计算的可逆性是指在量子计算过程中，每个量子操作（通常由量子门实现）都是可逆的。这意味着每个量子门操作都具有一个逆操作，该逆操作可以完全恢复输入量子态。在量子力学中，这个特性体现在量子操作（量子门）是幺正的。幺正操作具有以下性质：其逆操作就是其共轭转置（在量子力学中，共轭转置通常表示为酉矩阵）。

可逆性是量子计算与经典计算的一个重要区别。在经典计算中，有些操作是不可逆的，例如逻辑与门（AND gate）和逻辑或门（OR gate）。在这些操作中，输出信息会丢失输入信息的一部分，因此无法通过输出状态唯一地确定输入状态。然而，在量子计算中，这种信息丢失是不允许的，因

为它会破坏量子态的相干性。

可逆性在量子计算中具有重要意义，因为它保留了量子系统的相干性，并允许量子算法在量子计算过程中探索可能的解空间[25]。这使得量子计算在某些问题上具有优越性，例如大整数因式分解（Shor 算法）和无序数据库搜索（Grover 算法）。这些量子算法利用量子计算的特性，比经典计算方法更高效地解决了相应的问题。

1.5.1. 计算表达

可逆性在量子计算中的表示主要体现在量子门的性质上。在量子计算中，量子门是对量子比特进行操作的基本单元。为了满足可逆性，量子门必须是幺正的。幺正矩阵是一个满足以下性质的矩阵：它的共轭转置（酉矩阵）等于它的逆矩阵。用数学表示，设 U 是一个量子门，那么有：

$$U^\dagger U = UU^\dagger = I \tag{1.8}$$

其中，U 是一个正矩阵，$U^\dagger$ 表示 U 的共轭转置（酉矩阵），I 表示单位矩阵。这意味着 U 的逆操作就是 $U^\dagger$。

在量子计算中，常见的量子门如 X、Y、Z 门，以及 Hadamard 门、CNOT 门、Toffoli 门等都是幺正的。这些量子门都可以通过它们的共轭转置来实现逆操作，从而实现可逆性。

可逆性在量子计算中具有重要意义，因为它保证了量子态在操作过程中的相干性。由于量子门是幺正的，量子态在经过量子门操作后仍然保持在同一希尔伯特空间（量子态所在的线性空间）内。这使得量子计算能够在一定程度上利用量子叠加和量子纠缠等现象，实现对于某些问题的优越性能。

1.6. 总结

在本章中，我们研究了量子力学系统的四个基本原则：叠加、玻恩规则、纠缠和可逆计算。这四个原则对于理解经典计算与量子计算之间的差异至关重要，我们将在本书后面进一步学习。

[25]在数学和计算机科学中，解空间是指包含了一个给定问题所有可能解的集合。解空间可以被用来描述一系列的可能情况，或者是一个问题的所有可能答案。

例如，考虑一个简单的线性代数问题，比如线性方程组。这个方程组的解空间是由所有满足该方程组的解组成的向量空间。

在优化或搜索问题中，解空间可能包含了所有可能的候选解，而求解的过程就是在这个解空间中寻找最优解或满足特定条件的解。

解空间的大小和结构取决于具体问题的性质和约束条件。有的问题解空间可能非常大，甚至是无限的，而有的问题解空间可能就只有有限的几个解。

2

量子力学简史

自量子力学领域诞生伊始，其被用于进行新颖且有趣计算方式的可能性就一直潜藏在我们的视野中。叠加原理和纠缠现象可以形成非常强大的计算模式，关键在于如何构建一个我们能够轻松操作和测量的系统。

而在此之前，我们简单回忆一下量子力学的简单历史。

2.1. 一九七九年：提出计算理论

尽管**理查德·费曼**[1]通常被认为是量子计算机的创始人，但还有几位研究者更早的提出了这个想法。1979 年，阿贡国家实验室的年轻物理学家**保罗·贝尼奥夫**[2]提交了一篇名为“计算机作为物

[1]**理查德·菲利普·费曼**（Richard Phillips Feynman，1918 年 5 月 11 日－ 1988 年 2 月 15 日）是一位美国理论物理学家，他在量子力学、量子场论和粒子物理学等领域做出了重要贡献。费曼以他的卓越教育技巧、对物理学的深刻见解以及对于科学传播的独特风格而闻名。

费曼在量子电动力学（QED）领域的工作是他最著名的贡献之一。量子电动力学是描述光子与物质相互作用的理论，是量子场论的一个重要分支。费曼开发了一种称为费曼图的图形表示法，用于直观地描述和计算粒子间的相互作用。这种表示法不仅简化了复杂数学计算，还为物理学家提供了更深入的理解。

费曼还对超流体和固体物理学等领域做出了贡献。他的费曼路径积分方法为处理量子系统的时间演化提供了一种新的途径。此外，费曼还参与了美国曼哈顿计划，该计划致力于开发原子弹。

费曼以其传授知识的能力和对科学探索的热情而著称。他的一系列讲座——《费曼物理学讲义》（The Feynman Lectures on Physics）被认为是物理学领域的经典教材，涵盖了从力学到量子力学的广泛主题。

1986 年，费曼被任命为挑战者号航天飞机事故调查委员会的成员，他揭示了事故发生的原因，并在公众面前进行了直观的演示。这使得费曼在非物理学领域也获得了广泛的声誉。

费曼在 1965 年与 Julian Schwinger 和 Sin-Itiro Tomonaga 共同获得了诺贝尔物理学奖，以表彰他们在量子电动力学方面的杰出贡献。费曼在物理学界和公众中享有盛誉，他的工作为现代物理学的发展奠定了基础。

[2]**保罗·贝尼奥夫**（Paul Benioff，1930 年－ 2022 年 3 月 29 日）是一位美国理论物理学家，他在量子计算领域的早期工作中发表了一系列具有开创性的论文。贝尼奥夫被认为是量子计算理论的奠基人之一，他的研究对这一领域的发展产生了重要影响。

在 1980 年代初，保罗 · 贝尼奥夫开始对使用量子力学描述的计算机系统进行研究。在 1980 年，他发表了一篇题为“The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines”的论文。在这篇论文中，贝尼奥夫提出了一种量子力学哈密顿量描述的图灵机模型，将经典计算机与量子物理原理相结合。

贝尼奥夫的工作表明，量子力学规律可以应用于描述计算过程，并指出了在这种情况下可能出现的一些潜在优势。虽然贝尼奥夫的早期工作没有明确提出有效利用量子叠加和纠缠等量子现象进行计算的方法，但他确实为后来的量子计算理论奠定了基础。

在贝尼奥夫之后，理查德 · 费曼、大卫 · 德谢尔巴等科学家也对量子计算进行了研究，最终导致了量子计算这一领域的迅速发展。如今，量子计算已经成为计算机科学和物理学的前沿领域，有望在搜索、优化和加密等方面实现比经典计算更高效的解决方案。保罗 · 贝尼奥夫的工作为量子计算领域的发展奠定了基石，使其在计算机科学史上具有重要地位。

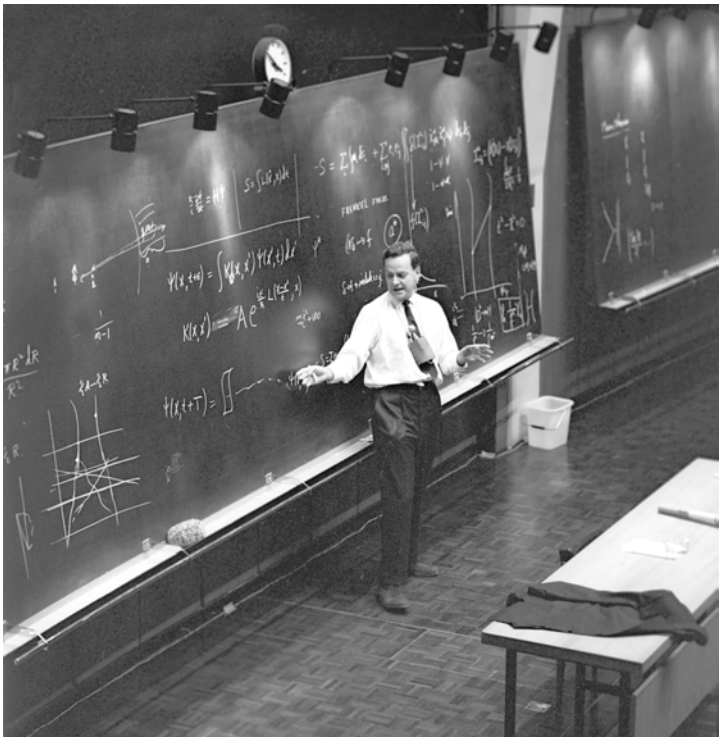


图 2.1: 1965 年，**理查德-费曼**在获得当年的诺贝尔物理学奖后，在日内瓦欧洲核子研究中心粒子物理实验室的一个研讨会上发言。图片来源：`https://physicsworld.com/a/richard-feynman-from-a-to-z/`

理系统：用图灵机表示的计算机的微观量子力学哈密顿模型”的论文 [24]。在这篇论文中，贝尼奥夫阐述了量子计算的理论基础，并提出这样的计算机可以被构建的：

“整个计算过程由一个在给定哈密顿作用下演化的纯态描述。因此，随着计算的进行，图灵机的所有组成部分都由具有确定相位关系的状态描述。这些模型的存在至少表明，应该检查构建这样的相干机器的可能性”。

2.2. 一九八零年：量子计算机思想

1980 年，**尤里 · 马尼**[3]在他的“Computable and uncomputable”（《可计算与不可计算》）一书中阐述了量子计算的核心思想 [25]。然而，这本书是用俄语写的，直到多年后才被翻译。

1981 年，费曼发表了题为“用计算机模拟物理现象”的演讲 [5]。在这次演讲中，他认为经典系统不能充分地表示量子力学系统：

“... 自然不是经典的，该死，如果你想模拟自然，你最好让它是量子力学的，这真是个了不起

[3]**尤里 · 伊万诺维奇 · 马尼**（Yuri Ivanovich Manin，1937 年－）是一位俄罗斯数学家，他的研究领域涵盖了代数、几何、数论和数学物理。马尼以在多项数学领域的突破性贡献而著名，也因其在量子计算领域的早期工作对该领域产生的影响而备受瞩目。

尤里 · 马尼在 1960 和 1970 年代对代数几何和数论等领域的研究做出了重要贡献。他在这些领域的工作包括对多项式方程的根的研究、对椭圆曲线的研究以及在代数几何中开创性地引入了模空间的概念。此外，马尼还对数学物理中的量子场论和共形场论等领域进行了研究。

在量子计算方面，尤里 · 马尼与理查德 · 费曼和保罗 · 贝尼奥夫等科学家是该领域的先驱者之一。在 1980 年代，马尼提出了一种基于线性代数的量子计算模型，并开始研究将量子力学应用于计算过程的可能性。虽然他的工作并没有直接导致现代量子计算框架的发展，但马尼的早期工作确实为量子计算理论的发展奠定了基础。

尤里 · 马尼在数学界享有盛誉，曾获得多个国际数学奖项，包括 1990 年的国际数学家大会（ICM）颁发的格奥尔格 · 康托尔奖章（Georg Cantor Medal）和 2002 年颁发的博雷尔奖章（Borel Medal）。作为一位多产的数学家，马尼不仅对代数、几何和数论等领域做出了贡献，还为量子计算领域的发展提供了理论基础。

的问题，因为它看起来并不容易...”

然后，他列举了量子计算机应具备的特征，以便发挥其作用。然而，当时费曼和物理学界并不清楚如何构建这样的机器。

贝尼奥夫、马尼和费曼为量子计算机敞开大门后，研究人员开始研究可以在量子计算机上运行的算法的性质。1985 年，牛津大学的物理学家**大卫·德意志**[4]在他的一篇论文中 [8] 提出了一个更全面的量子计算框架。在这项工作中，他详细描述了量子算法的样子，并预测“有一天，建造量子计算机在技术上将成为可能。”

2.3. 一九九零年：两个重要的算法

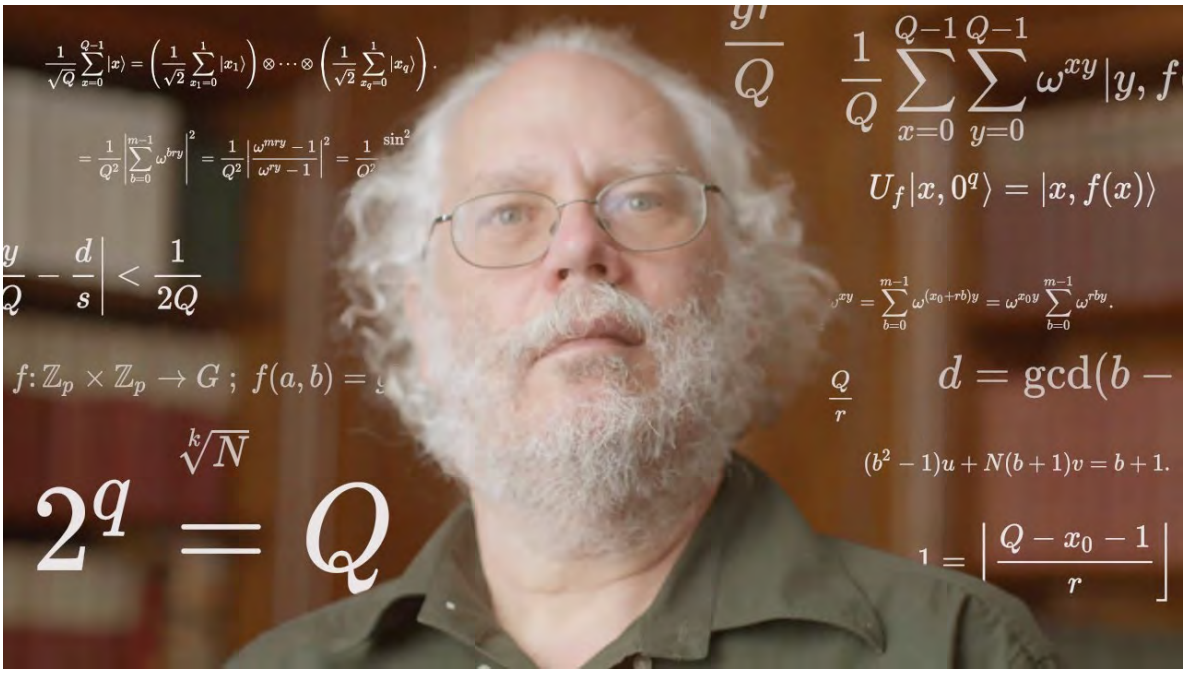


图 2.2: 彼得 · 舒尔，图片来源

随着这些开创性研究的展开，量子计算领域逐渐取得了重要的突破。研究人员开始开发出一系列新的量子算法，这些算法在理论上能够在量子计算机上显著加速问题求解。其中最著名的例子是 1994 年，**彼得·舒尔**[5]提出了著名的舒尔算法，该算法能够在量子计算机上以指数级加速解决大整数因式分解问题。这一发现引起了广泛关注，因为它对现代加密系统具有潜在的巨大影响。

[4]**大卫·德意志**（David Deutsch，1953 年生）是一位英国理论物理学家、哲学家和科普作家，主要以在量子计算和多世界解释（Many-Worlds Interpretation）方面的开创性工作而闻名。他在牛津大学获得博士学位，并在那里担任访问教授。德意志被认为是量子计算领域的奠基人之一，他的工作为量子计算机的发展奠定了基础。

在 1992 年，德意志提出了一个通用量子计算机的概念，称为“德意志-约瑟夫逊（Deutsch-Jozsa）算法”。该算法在理论上展示了量子计算机在解决特定问题时可能比经典计算机更高效。这一发现激发了后续的量子计算研究，包括著名的 Shor 的算法，能高效地分解大整数，从而对现有的加密系统构成威胁。

德意志在多世界解释方面的工作也非常重要。多世界解释是量子力学的一种解释，认为每当量子系统的状态发生叠加时，实际上会产生一个平行宇宙。这一观点在科学界引起了广泛的讨论和争议。

此外，大卫 · 德意志也是一位杰出的科普作家。他的两本著作——《量子计算机时代》（The Fabric of Reality，1997 年）和《创造开始》（The Beginning of Infinity，2011 年）——探讨了科学、哲学和历史等多个领域的话题，受到了广泛关注和好评。

大卫 · 德意志是一位在量子计算和多世界解释领域产生深远影响的科学家，对现代科学的发展产生了重要影响。

[5]**彼得·舒尔**（Peter Shor，1959 年生）是一位美国数学家和理论计算机科学家，因在量子计算领域的杰出贡献而闻名，尤其是发现了著名的“Shor 算法”，这一算法使得量子计算机能够高效地分解大整数。他的这项工作对现代密码学和量子计算的发展产生了深远影响。

舒尔在 1985 年从麻省理工学院（MIT）获得博士学位，并在贝尔实验室工作了一段时间。后来，他加入了麻省理工学院（MIT）的数学

另一个重要的量子算法是 1996 年**洛夫·库马尔·格罗弗**[6]提出的 Grover 搜索算法，它在量子计算机上可以实现对无序数据库的平方根加速搜索。这一算法在很多实际应用中具有广泛的潜在价值，例如优化搜索引擎和其他搜索任务。

Algorithm 3 格罗弗算法

procedure Grover(n, f) ▷ n 是量子比特的数量，f 是函数
    初始化 n 个量子比特到状态 $|\psi\rangle = |0\rangle^{\otimes n}$
    对 $|\psi\rangle$ 应用哈达玛变换 $H^{\otimes n}$
    for $\sqrt{N}$ 次 **do**
        应用 oracle U_f ▷ 翻转解状态的符号
        应用哈达玛变换 $H^{\otimes n}$
        执行条件相移（除了 $|0\rangle^{\otimes n}$ 的所有量子比特都获得 -1 相位）
        应用哈达玛变换 $H^{\otimes n}$
    end for
    测量量子态
end procedure

自那时以来，量子计算领域已经取得了显著的进展。研究人员已经成功地制造了实际的量子计算机原型，尽管这些原型的规模和计算能力尚不足以实现量子优势——即量子计算机在某些任务上显著优于经典计算机的阶段。然而，随着量子计算技术的不断发展和进步，人们对量子计算机的实际应用和潜在影响抱有越来越高的期望。

今天，量子计算正迅速发展，政府、企业和学术界都在积极投资研究和开发。随着量子计算技术的不断成熟，未来有望解决许多当今难以应对的计算挑战，例如在药物设计、材料科学、加密和优化问题等领域开辟新的可能性。

例如：**尼尔·西蒙**（Daniel Simon）在 1994 年进行了一项研究，概述了一个量子计算机明显比经典计算机解决得更快的问题 [12]。更具体地说，西蒙的算法是一种在量子计算机上运行的算法，由美国计算机科学家**丹尼尔·西蒙**在 1994 年提出。这个算法是量子计算领域的早期突破之一，它展示了量子计算相对于经典计算的优势。问题的核心是一个“黑盒”函数 f，它有一个特殊的性质：存在一个未知的字符串 s，对于所有输入 x，$f(x) = f(x \oplus s)$，其中 $\oplus$ 表示位的异或操作。也就是说，f 在某种周期性模式下重复。我们的任务是找出这个字符串 s。在经典计算中，解决这个问题的最有效方法是直接的暴力搜索，即尝试所有可能的输入，直到找到一个匹配的周期模式。但这需要 $\Omega(2^{n/2})$ 的时间复杂度，其中 n 是输入的位数。也就是说，在最糟糕的情况下，我们需要检查大约 $2^{n/2}$ 个不同的输入。西蒙的算法利用量子力学的叠加和干涉效应，在量子计算机上解决了这个

系，成为一名教授。

1994 年，彼得 · 舒尔提出了 Shor 算法，这是一个量子计算算法，可以有效地分解大整数。在密码学中，大整数分解被认为是困难的问题，许多现代加密系统（如 RSA 算法）正是基于这种困难性来确保信息安全。因此，Shor 算法对现有的加密系统构成了潜在威胁，引发了对量子计算和密码学的广泛关注。

舒尔的这项工作激发了量子计算领域的研究热潮，使得许多科学家和工程师开始关注和研究量子计算机的潜力和可能的实际应用。此外，彼得 · 舒尔还在理论计算机科学和编码理论等领域取得了显著成果。

彼得 · 舒尔是一位在量子计算和密码学领域产生重要影响的科学家，他的工作对现代科学和技术发展具有深远意义。

[6]洛夫 · 库马尔 · 格罗弗 Lov Kumar Grover 是一位印度裔美国计算机科学家，他在 Bell Labs 工作。他最为人所知的贡献是在 1996 年发明了 Grover's algorithm，这是一种在无序数据库中进行搜索的量子算法。

Grover 的算法在没有任何排序信息的情况下，可以在根号 N 的时间复杂度下找到一个特定的记录，其中 N 是数据库的大小。这比经典计算机上可能的最快方法（线性搜索，需要查看所有 N 个记录）要快得多，因此，Grover 的算法在理论上展示了量子计算的优势。

Grover 的工作对量子计算机的早期发展产生了重要影响，他的算法仍然是最知名和最常用的量子搜索算法之一。

问题，时间复杂度仅为 $O(n)$。这意味着，随着问题规模的增加，我们需要的计算步骤数量以线性的速度增加，而不是指数的速度。因此，西蒙的算法提供了一个明确的证据，证明在某些问题上，量子计算机在理论上可以显著地超越经典计算机。这就是我们所说的“量子优势”或“量子超越”。

图 2.3: 洛斯阿拉莫斯实验室（Los Alamos National Laboratory）是美国最著名的国家实验室之一，位于新墨西哥州的洛斯阿拉莫斯市。该实验室成立于 1943 年，最初是为了开发原子弹而设立的，是曼哈顿计划的主要组成部分之一。如今，洛斯阿拉莫斯实验室是美国国防部和国家能源部的重要研究机构，致力于国家安全、能源、环境和生命科学等方面的研究。在量子计算领域，洛斯阿拉莫斯实验室一直处于领先地位，并与多家企业和研究机构合作推动量子计算技术的发展。，图片来源量子客`https://www.qtumist.com/post/15498`

在**丹尼尔·西蒙**研究算法之前，洛斯阿拉莫斯实验室的**赛斯·劳埃德**[7]在《科学》杂志上发表了一篇论文，描述了如何构建一个可行的量子计算机 [26]。他提出，一个向单元发送脉冲的系统可以表示一个量子态：

”脉冲式、弱耦合量子系统阵列为量子计算提供了一个可能实现的基础。阵列中的基本单元可以是量子点、核自旋、聚合物中的局域电子态，或者任何与邻近体系局部相互作用且可以通过共振光脉冲在不同态之间切换的多态量子系统。”

量子计算领域在这些基础性研究的推动下不断取得突破，如今已经发展成为一个备受关注的前沿技术。随着量子计算技术的进一步发展，未来可能会出现越来越多的应用场景，从而推动计算科学和其他领域的创新。

劳埃德意识到：所提出的设备具有纯粹的量子力学信息处理能力，远超已经展示的传统数字能力。其中最重要的一种能力是，通过在适当的共振频率上简单地应用脉冲，但长度与完全切换比

[7]赛斯 · 劳埃德（Seth Lloyd）是一位美国物理学家，生于 1960 年。他是麻省理工学院（MIT）机械工程系、物理系、核工程系和工程系统分区的教授，也是 MIT 量子工程中心的主任。

劳埃德在量子计算、量子通信和量子生命科学等领域做出了重要贡献。他是量子信息理论的先驱之一，在这个领域提出了许多创新的想法，包括量子比特的概念、量子密码学和量子计算的基础概念。

劳埃德的研究还包括热力学、统计物理学、信息理论和生物物理学。他是美国物理学会和美国文理科学院的会员，曾获得过美国物理学会杰出青年奖和麻省理工学院的教学奖。

特所需的长度不同，可以将比特置于 0 和 1 的叠加状态。这样的比特具有多种用途，包括生成随机数。这是第一个实用的量子计算机方法。有趣的是，劳埃德注意到从量子系统生成随机数的可能用途；这已经成为量子计算社区近期研究的一个话题。请参阅例如 [27] 和 [28]。

彼得 · 肖尔在 1994 年是新泽西贝尔实验室（Bell Labs）数学部门的一名研究员。肖尔研究了 Deutsch、BV 和 Simon 的工作，并意识到他可以构造一种将大数分解为两个质数因子的算法；在经典计算机上分解大数被认为是棘手的，但肖尔的分解算法在量子计算机上运行迅速。分解大数当然是公钥加密（PKC）的核心难题，就像 RSA 算法 [15] 实现的那样，这种加密是现今几乎所有互联网通信的基础。这包括安全地发送信用卡号、银行付款和确保在线消息系统的安全。

基于 RSA 的加密依赖于将大数分解为两个质数因子的单向困难。生成大数很容易；我们只需将两个因子相乘。然而，给定一个任意大的数，找到它的两个质数因子是指数级困难的。

Algorithm 4 Shor 算法

procedure Shor(N) ▷ N 是待分解的整数
    随机选择一个整数 $a < N$
    计算 $g = \gcd(a, N)$
    if $g > 1$ **then**
        return g ▷ 我们找到了一个因子
    else
        使用量子傅里叶变换找到 $a \mod N$ 的周期 r
        if r 是奇数或者 $a^{r/2} \equiv -1 \mod N$ **then**
            回到开始
        else
            return $\gcd(a^{r/2} \pm 1, N)$
        end if
    end if
end procedure ▷ 在随后的章节我们将详细学习这个算法

受西蒙启发，肖尔意识到我们可以使用量子计算机解决与分解问题等价的另一个问题；实际上，分解问题等价于西蒙在他的论文 [29] 中解决的周期查找问题。他还意识到，BV 描述的量子傅立叶变换（QFT）正是他在测量之前设置每个量子比特振幅所需的，以便测量结果可以以高概率从量子计算中获得所需的答案。

2.4. 二零零零年：量子计算机诞生

肖尔的突破使更多研究人员投入到量子算法的研究中，因为此时已经很明确，如果建立了量子计算机，它们将非常强大。事实上，肖尔的算法是最早在早期量子计算物理系统上展示的算法之一。2001 年，Isaac Chuang 等人在核磁共振（NMR）系统上实现了肖尔的算法，将数字 15 分解为一个示例 [30]。

在肖尔之后，Lov Grover 为量子算法库贡献了一个搜索算法的速度提升，该算法可以在量子计算机上实现 [31]。Grover 的算法仅实现了二次加速，而不是指数加速（如肖尔的算法），但这仍然具有重要意义。二次加速意味着，如果一个算法在经典计算机上需要 $O(N)$ 步，我们可以在量子计算机上用 $O(\sqrt{N})$ 步实现相同的目标。在 Grover 的论文发布的 1996 年 5 月之后的几个月，

Farhi 和 Gutmann 为 Grover 的算法制定了一个连续时间哈密顿版本的框架 [32]

。这引入了哈密顿预言子和实现量子计算连续时间模型的概念，这与基于门的方法不同。

在一组研究人员发现可以在量子计算机上运行具有加速优势的算法时，另一组人在量子计算机的物理实现方面取得了进展。1999-2001 年，中村康伸（Yasunobu Nakamura）构建并展示了一个功能完善的、可控的超导量子比特 [33, 34]。中村使用约瑟夫森结（Josephson junction）创建一个可在两个状态之间操作的二级系统。实现量子计算机的另一种方法是捕获和操纵离子。1995 年，Cirac 和 Zoller 提出将离子阱作为执行量子计算的物理系统 [35] 。在这个设置中，激光用于将原子电离，然后将其固定在电势中。

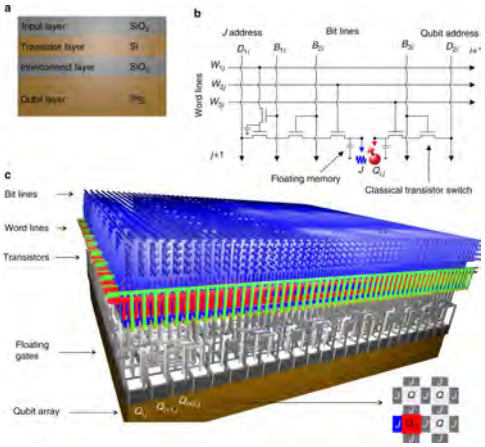


图 2.4: 一个基于自旋的量子计算机的硅 CMOS 架构，该架构的物理量子处理器，电路控制以及单元模块的物理架构。具体来说，子图 a 展示了硅片上的双层结构，其中含有 2D 量子位阵列的底部层是富含硅-28 同位素的硅层，而顶部的硅层则形成用于操作量子位的晶体管。这些晶体管通过氧化物区域使用多晶硅（或其他金属）电路进行互连。子图 b 展示了控制一个 Q-gate 和一个 J-gate 的电路，从而实现所需的逐行、逐列或全局操作，如文章的正文所述。子图 c 展示了操作包含 480 个量子位的单元模块的物理架构。右下角的插图显示了通过量子位平面进行的平面视图剖面。每个 J 门和量子位都通过 b 中展示的电路进行连接。图片来源文献 [36]

随着量子计算领域活动的增加，该领域的研究人员明确了构成量子计算机和计算的条件。1996 年，**大卫 · 迪文森佐**[8]以这种方式概述了量子计算机的关键标准迪文森佐的量子计算机准则为：

- 可扩展的具有良好特性的物理系统与良好定义的量子比特；
- 量子比特之间的可行且可扩展的操作；
- 对量子比特的长相干时间；
- 通用量子门的可实现性；
- 可实现的量子态制备；
- 可实现的量子比特测量。

这些准则已成为评估量子计算机实现方法的基本指南。

在量子计算机硬件的发展过程中，研究人员还在努力提高量子计算机的容错性。这意味着在不影响计算结果准确性的情况下，应对量子计算机中的错误。在 1995 年和 1996 年，Shor 和 Steane

[8]大卫 · 迪文森佐（David DiVincenzo）是一位美国物理学家，出生于 1957 年。他是德国莱布尼茨研究所的研究员，同时也是阿亚赛隆基金会的主席。

迪文森佐在量子计算和量子信息处理领域做出了杰出的贡献，提出了著名的 DiVincenzo 标准，这个标准成为了衡量量子计算机是否成功的重要标准之一。

他还是量子误差校正、量子通信、量子协议和量子算法等领域的专家。迪文森佐曾担任美国物理学会量子信息小组的主席，并荣获了美国物理学会颁发的 Oliver E. Buckley 凝聚态物理学奖、伦敦皇家物理学会颁发的 Dirac 奖章、和美国文理科学院的会员身份。

分别提出了用于量子计算机的容错编码方案 [37, 38]。后来，其他容错编码方案 [37, 38]。后来，其他容错编码方法和纠错技术也得到了发展，使得量子计算机在保证结果准确性的同时更加健壮。

2.5. 本章总结

自 20 世纪 90 年代以来，量子计算领域已经取得了显著的进展。从理论的发展到实际的量子计算机实现，量子计算机的应用前景不断扩大。量子计算机具有解决传统计算机难以处理的问题的潜力，例如在密码学、优化和模拟方面的应用。虽然量子计算机的发展仍然面临诸多挑战，但它们已经在多个方面展示出强大的潜力，为未来的科技创新和发展奠定了基础。

- 一个可扩展的物理系统，其中量子比特彼此独立，并且能够准确计算系统中的量子比特数量（不允许误差）。系统可以通过希尔伯特空间进行精确表示。
- 能够将任何量子比特的状态初始化为计算基础中的确定状态。例如，如果计算基向量是 $|0\rangle$ 和 $|1\rangle$，将所有量子比特设置为状态 $|0\rangle$。
- 系统的量子比特必须能够保持其状态。这意味着系统必须与外界隔离，否则量子比特将失去相干性。允许一些状态衰减（，其中是一个小量）。在实践中，系统的量子比特必须保持其状态足够长的时间，以便确保在操作之间，量子比特没有因外部影响而改变状态。
- 系统必须能够对量子比特状态应用一系列酉算子。系统还必须能够同时对两个量子比特应用酉算子。这涉及到这些量子比特之间的纠缠。正如迪文森佐在他的论文中所说：

  “... 量子计算机不同部分之间的纠缠是好的；量子计算机与其环境之间的纠缠是坏的，因为它对应于失去相干性”[39]
- 系统能够对每个量子比特进行“强”测量。通过强测量，迪文森佐是指测量结果显示“量子态属于某个特定厄米算子的正交本征态，同时将系统的波函数不可逆地投影到相应的本征函数中。”这意味着系统中的测量技术实际上确实测量了被测量属性的量子比特状态，并使量子比特保持在该状态。迪文森佐希望防止具有弱测量的系统，换句话说，测量技术不足以与量子比特耦合，使其处于新测量状态。当他撰写论文时，许多系统的耦合度不足以保证投影到新状态。

在接下来的章节中，我们将更详细地探讨构建量子计算机的各种方法以及如何在构建完成后编程。现在让我们将注意力转向量子比特和量子计算中使用的算子。

3

量子位，算子和测量

在本章中，我们将介绍量子比特和用于操纵量子比特状态的核心——算子集。

量子比特是一个量子位。量子比特与经典比特相似，它可以取 0 或 1 作为状态，但它与比特不同之处在于它还可以取一系列连续值，表示状态的叠加。

虽然通常我们使用两级量子比特系统来构建量子计算机，使用两级量子比特系统来构建量子计算机指的是基于两个能够稳定存在的量子态之间相互转换的系统构建量子计算机。这两个能够稳定存在的量子态[1]可以是一个自旋上和下的电子态[2]、两个不同能级的原子态[3]、两个超导量子比特之间的库珀对等。

量子计算机中的量子比特通常可以同时处于多个状态的叠加态，这种性质被称为叠加态的特性。使用两级量子比特系统可以实现对这种叠加态进行控制，从而实现量子计算中的量子纠缠和量子门操作等重要的量子信息处理任务。例如，使用两级量子比特系统的量子计算机可以实现基本的量子门操作，如 CNOT 门、Hadamard 门和相位门等，以及一些量子算法和量子通信协议。

但我们也可以选择其他类型的计算架构。例如，我们可以使用三级系统的量子三态构建量子计算机。我们可以将其视为具有 0、1 或 2 的状态，或者这些状态的叠加。

这种单元的更一般术语是 qudit, 译作量子位或量子比特；量子比特和量子三态是 qudit 的特定实例，可以是任意数量状态的计算单元。例如，加州大学伯克利分校的 Siddiqi 实验室设计了一个基于量子三态的量子计算机 [40]。在一个量子三态系统中，我们可以表示比具有相同数量计算单元的量子比特系统更多的状态。

---

[1]量子态指的是描述一个量子系统的状态，它用于描述系统所处的量子态空间中的一个向量或波函数。量子态可以用来描述量子系统的物理性质和状态，例如位置、动量、自旋等。

在量子力学中，量子态是由所谓的 Hilbert 空间中的复向量表示的。这个向量通常用符号“$|\psi\rangle$”表示，其中“ψ”代表向量的名称。这个向量在某些特定的测量下将具有一定的概率，这些测量可以对量子系统的状态进行观察和测量。

量子态的性质可以通过运用量子力学中的数学工具来计算和描述，例如薛定谔方程和量子力学中的算符。量子态的特殊性质包括叠加态的性质，即一个量子系统可以处于多个态的叠加态，这使得量子计算机能够处理大量数据并且在某些情况下比经典计算机更加高效。

[2]电子态是指一个原子或分子中的电子所处的能级或状态，它用来描述电子的物理性质和状态。在原子或分子中，电子通过在不同的能级中运动来维持其状态和性质。电子态的特性可以通过测量电子的能量、自旋和角动量等物理量来确定。电子态在化学和材料科学等领域中具有重要的应用，例如在描述化学反应和材料性质等方面起着重要的作用。

[3]原子态指的是一个原子所处的能级或状态，它用来描述原子的物理性质和状态。在原子中，电子通过在不同的能级中运动来维持其状态和性质。

一个 100 个量子比特的系统可以处理 2^{100} 个状态（1:26765E+30），1:26765E+30 是一个 16 进制数，表示的是一个二进制数的值。在这个 16 进制数中，前面的“1”代表这个二进制数是正数，后面的数字和字母（26765E+30）表示这个二进制数的值。

具体来说，这个 16 进制数对应的二进制数是：

$$0010 \quad 0110 \quad 0111 \quad 0110 \quad 0101 \quad 1110 \quad 1100 \quad 0011 \quad 0000$$

这个二进制数的长度是 100 位，正好与 100 个量子比特的系统的状态数相同。这个二进制数的每一位表示了 100 个量子比特系统中某一特定叠加态的存在或缺失。因此，这个 16 进制数是用来表示 100 个量子比特系统的状态数的一种简便的方式。而一个量子三态系统可以处理 3^{100} 个状态（5:15378E+47），这个数字比前者大 17 个数量级。换句话说，要表示与 100 个量子比特系统相同的数字空间，我们只需要约 63 个量子三态（$log_3(2^{100})$）。由于构建量子三态系统更加困难，目前主流的量子计算机是基于量子比特的。无论我们选择量子比特、量子三态还是其他 qudit 数量，这些系统都可以运行其他系统可以运行的任何算法，即它们可以相互模拟。

在量子力学中，我们用向量表示状态，用矩阵表示算子，并使用狄拉克表示法代替传统的线性代数符号来表示向量和其他抽象概念。

3.0.1. 什么是量子比特?

物理量子比特是一个二级量子力学系统。正如我们将在构建量子计算机的章节中看到的，有很多方法可以构建一个物理量子比特。我们可以将量子比特表示为一个二维复数希尔伯特空间 $\mathbb{C}^2$。量子比特在任何给定时间的状态可以用这个复数希尔伯特空间中的一个向量来表示。

其中：希尔伯特空间 是数学中的一个概念，它是由德国数学家**大卫·希尔伯特**[4]在 19 世纪末提出的。希尔伯特空间是一个向量空间，其中的向量是无限维的复数序列，并且满足一些特定的性质。希尔伯特空间在量子力学中发挥着重要的作用，因为它是用来描述量子力学中的态空间和物理量的空间的。

在量子力学中，希尔伯特空间被用来描述量子系统的状态和物理量。量子力学中的态矢量是希尔伯特空间中的向量，它可以表示量子系统的状态，而量子力学中的物理量则是希尔伯特空间中的厄米算符，它可以用来描述测量量子系统的物理量的结果。

希尔伯特空间的性质和数学方法在量子力学中具有广泛的应用，例如在描述量子比特和量子纠缠等方面。希尔伯特空间的基本概念和数学工具对于理解量子力学中的量子态和量子测量等概念至关重要。

希尔伯特空间配备了内积，它允许我们确定代表量子比特状态的两个向量之间的相对位置。我们用 $|u|v\rangle$ 表示向量 $|u\rangle$, 和 $|v\rangle$ 的内积，如果 $|u\rangle$ 和 $|v\rangle$ 是正交的即这表示两个向量之间的夹角是 90 度，它们在空间中的方向是互相垂直的。正交向量在数学和物理中都是非常重要的概念，因为它们可以被用来描述不同维度的空间、投影和基底等概念。在量子力学中，正交的态矢量可以表示

[4] **大卫·希尔伯特**（David Hilbert，1862-1943）是一位德国数学家，被誉为 20 世纪最重要的数学家之一。他在数学的多个领域都做出了开创性的贡献，特别是在函数论、代数学、数学物理学和逻辑学等领域。

希尔伯特最著名的成就之一是创立了希尔伯特空间的概念，这是用来描述无限维向量空间的一个基本工具。希尔伯特空间在量子力学中发挥着重要的作用，因为它是用来描述量子力学中的态空间和物理量的空间的。

希尔伯特还在代数学和数学物理学等领域做出了许多重要贡献，他提出了希尔伯特问题，这是一个包含 23 个未解决问题的数学难题清单。其中最著名的是希尔伯特第十三问题，它为后来的代数学、群论和拓扑学等领域的发展奠定了基础。

希尔伯特的贡献被广泛地认为对现代数学和物理学的发展有着深远的影响。他曾在多个国际数学家大会上发表过著名的演讲，其中最著名的是 1900 年在巴黎的一次演讲，提出了 23 个重要的未解决问题，激励了许多数学家的研究工作。

互相垂直的量子态，它们之间的内积为零，因此不能同时测量到它们的取值，所以内积为 0，如果 $|u\rangle = |v\rangle$，则内积为 1。为了表示两个或多个量子比特，我们可以将希尔伯特空间进行张量积运算，以表示量子比特的组合状态。

可以用下面显示的向量来表示状态 $|0\rangle$ 和 $|1\rangle$，我们称这两个为二级系统的计算基础。然后，我们可以将以矩阵形式表示的算子应用于状态空间中的向量。

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

其中：

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

表示处于 $|0\rangle$ 状态的量子比特的态矢量，它是一个列向量，第一个分量为 1，第二个分量为 0。

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

表示处于 $|1\rangle$ 状态的量子比特的态矢量，同样也是一个列向量，第一个分量为 0，第二个分量为 1。

3.0.2. 量子算子

在基于门的量子计算机中，我们用来演化量子比特状态的算子是幺正的，因此是可逆的。有些算子是幺正的、可逆的和可对合的（即它们是自己的逆算子）；其他的则不是。一个可测量的量，或可观察的量，是一个厄米算子；因此，量子计算机中的测量从系统输出实值。我们可以互换地使用算子和门这两个术语。

除了两个向量的内积，线性代数还给我们提供了外积。这是当我们取两个向量并形成一个矩阵（而内积给我们一个标量）的时候。例如，如果我们取外积 $|0\rangle\langle 0|$，我们会产生以下算子：

$$|0\rangle\langle 0| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{3.1}$$

类似地，我们可以取其他三个组合的外积来产生这些矩阵：

$$|0\rangle\langle 1| = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{3.2}$$

$$|1\rangle\langle 1| = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \tag{3.3}$$

$$|1\rangle\langle 0| = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \tag{3.4}$$

我们可以将两个矩阵的和取为一个幺正矩阵，如：

$$|0\rangle\langle 1| + |1\rangle\langle 0| = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3.5}$$

这就是 X 或 NOT 算子。量子比特可以处于计算基状态 0 或 1 中，或者处于这两个状态的叠加态中。我们如何表示多个状态的叠加？我们可以将状态空间的计算基的线性组合表示为叠加。

3.0.3. 状态的叠加

我们将状态的叠加表示为状态空间的计算基的线性组合。叠加中的每一项都有一个复数系数或振幅。

在单个量子比特的情况下，使用两个计算基向量，两个状态叠加的例子分别是：

$$|+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{3.6}$$

公式3.6表示的是一个量子比特的态矢量，它被称为“+ 态”或“哈达玛态”。在这个公式中，$|0\rangle$ 和 $|1\rangle$ 分别表示量子比特处于“0 态”和“1 态”的态矢量。公式中的“:=”表示定义，它意味着我们定义 $|+\rangle$ 为右边的式子。

公式右边的式子中，$\frac{1}{\sqrt{2}}$ 是一个归一化系数，保证态矢量的模长平方为 1。$|0\rangle + |1\rangle$ 表示两个态矢量的叠加，它们的内积为 0，即它们是正交的。这个叠加操作相当于将量子比特同时处于两种状态的叠加态，即

$$|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{3.7}$$

和

$$|-\rangle := \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{3.8}$$

这个公式表示的是一个量子比特的态矢量，它被称为“-态”。在这个公式中，$|0\rangle$ 和 $|1\rangle$ 分别表示量子比特处于“0 态”和“1 态”的态矢量。公式中的“:=”表示定义，它意味着我们定义 $|-\rangle$ 为右边的式子。

公式右边的式子中，$\frac{1}{\sqrt{2}}$ 是一个归一化系数，保证态矢量的模长平方为 1。$|0\rangle - |1\rangle$ 表示两个态矢量的差，它们的内积为 0，即它们是正交的。这个差分操作相当于将量子比特同时处于两种状态的差分态，即

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{3.9}$$

因此，$|-\rangle$ 表示量子比特处于一个等概率叠加了“0 态”和“1 态”相位取负的态矢量。在量子计算中，Hadamard 门可以将一个量子比特从 $|0\rangle$ 转化为 $|+\rangle$，或者从 $|1\rangle$ 转化为 $|-\rangle$。

这两个状态在 $|1\rangle$ 状态上的差异是一个负号。更正式地说，我们称这种差异为相对相位。相位这个词在物理学中有很多含义——在这个上下文中，它指的是一个角度。负号与欧拉恒等式 2[5]中的角度 π（180°）有关：

$$e^{i\pi} = -1 \tag{3.10}$$

[5]欧拉恒等式 2 是一组数学等式，描述了复数的三角函数，指数函数和自然对数函数之间的关系。它的表达式如下：

$$e^{i\theta} = \cos\theta + i\sin\theta$$

其中，e 是自然常数，i 是虚数单位，θ 是角度。这个公式可以看作是欧拉恒等式的一种特殊形式，因为当 $\theta = \pi$ 时，公式左侧的 $e^{i\pi}$ 等于 -1，而公式右侧的 $\cos\pi = -1$，$\sin\pi = 0$，所以两侧相等。

欧拉恒等式在数学、物理和工程等领域中都有广泛的应用，它将复数的指数函数和三角函数联系起来，为许多问题的求解提供了便利。在量子计算中，欧拉恒等式也被广泛应用，因为它能够将相位旋转操作表示为一个酉矩阵，从而方便地在量子算法中使用。

相对相位对于量子算法具有根本性的重要性，因为它们允许发生有构性干涉和破坏性干涉。例如，如果我们计算上述状态的和，我们得到：

$$\frac{1}{\sqrt{2}}(|+\rangle + |-\rangle) = \frac{1}{2}|0\rangle + |1\rangle + \frac{1}{2}(|0\rangle - |1\rangle) = |0\rangle \tag{3.11}$$

这个公式表示的是一个等式关系，左侧的表达式为一个量子比特的叠加态，其中 $|+\rangle$ 和 $|-\rangle$ 分别表示量子比特处于“+ 态”和“-态”（即哈达玛门作用后的态矢量），右侧的表达式表示同一个量子比特的混合态，其中 $|0\rangle$ 和 $|1\rangle$ 分别表示量子比特处于“0 态”和“1 态”。这个等式告诉我们，左侧的叠加态和右侧的混合态是等价的，它们所代表的量子比特状态是相同的。

现在我们来看等式右侧的具体计算过程。首先，我们可以将等式右侧的式子按照括号中的符号分成两个部分：

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

我们发现，第一个部分中的 $|0\rangle$ 和 $|1\rangle$ 是相加的，而第二个部分中的 $|0\rangle$ 和 $|1\rangle$ 是相减的。因此，我们可以将这个式子简化为：

$$|0\rangle$$

这里的等号成立，因为两边的态矢量代表的是同一个量子比特状态。所以，我们说 $|1\rangle$ 状态的振幅发生了破坏性干涉——不同的相对相位使它们相加为零。另一方面，$|0\rangle$ 状态的振幅发生了有构性干涉——它们具有相同的符号（相对相位），因此它们不相加为零，因此我们得到 $|0\rangle$ 状态作为结果。

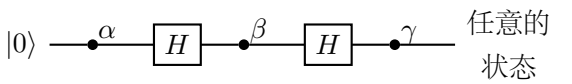
在这里，$|0\rangle$ 状态的振幅发生了破坏性干涉，而 $|1\rangle$ 状态的振幅发生了有构性干涉。在这个例子中，我们没有得到完全相同的 $|1\rangle$ 状态——它乘以一个负号。正如我们在上面看到的，我们可以将这个负号解释为一个角度（或相位）$e^{i\pi}$。在这里，它应用于整个状态，而不仅仅是叠加中的一个项。我们称这种类型的相位为全局相位。

虽然确实 $-|1\rangle$ 状态不完全等于 $|1\rangle$ 状态，但我们将在后续章节中看到，全局相位变化对量子测量没有影响。也就是说，测量 $-|1\rangle$ 状态和测量 $|1\rangle$ 状态得到的测量统计数据是完全相同的。在这种情况下，我们通常说这两个状态在全局相位上是相等的。

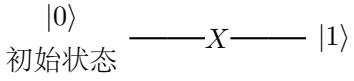
因此，在量子计算中，我们通常会忽略全局相位的差异，因为它们对测量结果没有实际影响。然而，相对相位在量子算法中起着至关重要的作用，因为它们可以导致不同状态的振幅发生有构性或破坏性干涉。这是量子计算中充分利用量子力学特性的一个关键方面，为量子算法提供了优势。在后续章节中，我们将更深入地研究如何利用相位和叠加来实现高效的量子算法。

3.0.4. 量子电路图

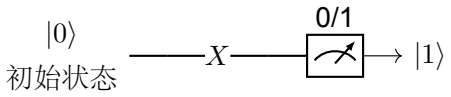
我们使用电路图来表示量子电路。我们从左到右构建和阅读这些图；我们可以把电路图想象成乐谱，阅读方向也是一样的。Barenco 等人提出了一些我们今天在量子计算中使用的基本操作符 [41]。Fredkin 和 Toffoli[42, 43] 通过两个三元操作符扩充了这个集合。



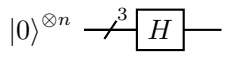
3.0.5. 初始状态



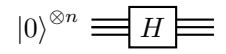
这个量子电路表示的是一个单量子比特的 X 门操作。X 门是量子计算中的一种基本门，也称为量子比特的 NOT 门。它作用在一个单量子比特上，将 $|0\rangle$ 态变为 $|1\rangle$ 态，将 $|1\rangle$ 态变为 $|0\rangle$ 态。因此，这个电路的伪算法可以表示为：将一个初始为 $|0\rangle$ 的量子比特应用一个 X 门，将其变为 $|1\rangle$。



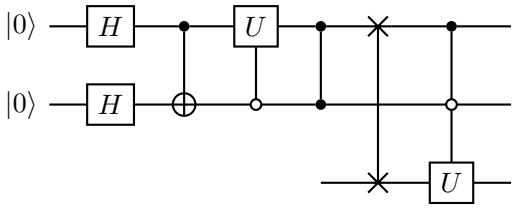
这个量子电路表示的是一个单量子比特的 X 门操作，之后进行了一次测量。X 门是量子计算中的一种基本门，也称为量子比特的 NOT 门。它作用在一个单量子比特上，将 $|0\rangle$ 态变为 $|1\rangle$ 态，将 $|1\rangle$ 态变为 $|0\rangle$ 态。测量操作是量子计算中的一种基本操作，用来获取量子系统的经典信息。这个电路的伪算法可以表示为：将一个初始为 $|0\rangle$ 的量子比特应用一个 X 门，将其变为 $|1\rangle$，然后进行一次测量，将结果输出到经典比特中。在这个电路中，meter 表示测量，0/1 表示输出 0 或 1 的概率。



这个量子电路表示的是一个 n 量子比特的 Hadamard 门操作。Hadamard 门是量子计算中的一种基本门，它将一个单量子比特从 $|0\rangle$ 态变为 $\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$ 态，将一个单量子比特从 $|1\rangle$ 态变为 $\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$ 态。对于 n 个量子比特的系统，Hadamard 门作用于所有量子比特的叠加态将会变成一个均匀叠加态，即 $\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle$。因此，这个电路的伪算法可以表示为：将 n 个初始为 $|0\rangle$ 的量子比特应用一个 Hadamard 门，将其置于均匀叠加态上，即 $|0\rangle^{\otimes n}\rightarrow\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle$。



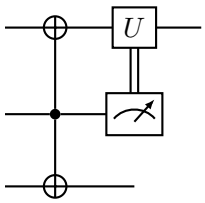
这个量子电路表示的是一个 n 量子比特的 Hadamard 门操作，在这种方式下，每一个方块代表一个量子比特，不同方块之间使用不同的颜色标识。在这个电路中，Hadamard 门作用于第一个量子比特，将其从 $|0\rangle$ 态变为 $\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)$ 态。$[arrows, yshift = 0.1cm]l[arrows]l[arrows, yshift = -0.1cm]l$ 表示在第一个和第二个量子比特之间插入一个空格，以便区分不同的量子比特。因此，这个电路的伪算法可以表示为：将 n 个初始为 $|0\rangle$ 的量子比特中的第一个量子比特应用一个 Hadamard 门，将其置于均匀叠加态上，即 $|0\rangle^{\otimes n}\rightarrow\frac{1}{\sqrt{2^n}}\sum_{x=0}^{2^n-1}|x\rangle$。



这个量子电路表示的是一个基于量子相位估计（QPE）算法的量子电路，其中包含了 Hadamard 门、CNOT 门、控制 U 门、SWAP 门、Tofolli 门等基本量子门。该电路的输入为两个量子比特，其中第一个量子比特作为 QPE 算法的控制比特，第二个量子比特作为量子电路的输出。该电路的具体操作过程比较复杂，可以简单地概括为以下几步：

- 在第一个量子比特上应用 Hadamard 门，将其置于均匀叠加态上。
- 在第二个量子比特上应用 U 门，对其进行量子相位变换。
- 将第一个量子比特与第二个量子比特进行 CNOT 门操作。
- 对第一个量子比特应用 Tofolli 门，将相位信息从第二个量子比特传递到第一个量子比特和一个辅助比特中。
- 对辅助比特应用多次控制 U 门，以实现相位估计。
- 对第一个量子比特和辅助比特应用 SWAP 门，将它们的状态交换。
- 对第一个量子比特应用多次 CNOT 门，将相位信息反馈到第二个量子比特上。
- 对第一个量子比特和第二个量子比特应用 Hadamard 门和 Tofolli 门，得到最终的输出结果。

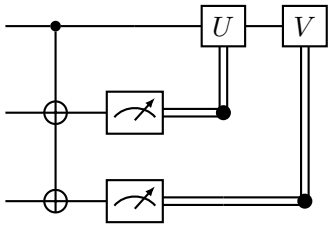
总的来说，该量子电路的操作过程涉及到多个量子门的组合和交错，利用了量子相位估计算法的原理，用于求解一些特定的量子计算问题。



这个量子电路表示的是一个基于量子迭代相位估计算法的量子电路，其中包含了 Tofolli 门、控制 U 门、CNOT 门和量子测量等基本量子门。该电路的输入为两个量子比特，其中第一个量子比特作为控制比特，第二个量子比特作为目标比特。该电路的具体操作过程可以简单地概括为以下几步：

- 在目标比特上应用 U 门，进行量子相位变换。
- 将控制比特和目标比特进行 Tofolli 门操作。
- 对控制比特和目标比特进行 CNOT 门操作。
- 对目标比特进行量子测量，得到测量结果。

该电路的主要作用是用来估计 U 门的相位，进而求解一些特定的量子计算问题。具体而言，该电路可以通过多次迭代来逐步精确地估计 U 门的相位，从而实现类似于经典计算中的二分查找算法的效果。

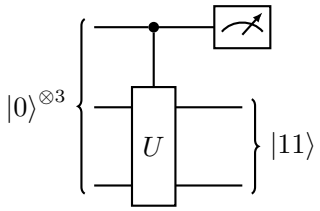


这个量子电路表示的是一个基于量子相位估计算法的量子电路，其中包含了两个控制门（CNOT 门）和两个量子测量。该电路的输入为两个量子比特，其中第一个量子比特作为控制比特，第二个量子比特作为目标比特。该电路的具体操作过程可以简单地概括为以下几步：

- 在目标比特上应用 U 门，进行量子相位变换。

- 将控制比特和目标比特进行 CNOT 门操作。
- 将目标比特和另一个新的辅助比特进行 CNOT 门操作。
- 在辅助比特上应用 Hadamard 门和 T 门，将其置于特定的超位置态上。
- 将辅助比特和目标比特进行控制 V 门操作。对目标比特和辅助比特进行量子测量，得到测量结果。

该电路的主要作用是用来估计 U 门和 V 门之间的相位关系，进而求解一些特定的量子计算问题。具体而言，该电路可以通过多次迭代来逐步精确地估计相位关系，从而实现求解一些经典算法难以解决的问题。

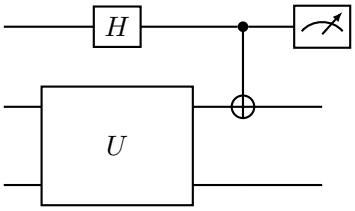


这个量子电路表示的是一个基于量子相位估计算法的量子电路，其中包含了一个控制门（CNOT 门）、一个待估计的相位变换门 U 和两个量子测量。该电路的输入为三个量子比特，其中第一个量子比特作为控制比特，后面两个量子比特作为目标比特。

该电路的操作过程可以简单地概括为以下几步：

- 在目标比特上应用 U 门，进行量子相位变换。
- 将控制比特和第二个目标比特进行 CNOT 门操作。
- 对第一个目标比特进行量子测量，得到测量结果。
- 将第一个目标比特和第二个目标比特进行控制 U 门操作。
- 对第二个目标比特进行量子测量，得到测量结果。

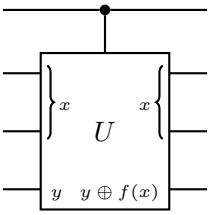
该电路的主要作用是用来估计 U 门和控制 U 门的控制相位之间的关系，从而实现求解一些特定的量子计算问题。具体而言，该电路可以通过多次迭代来逐步精确地估计相位关系，从而实现求解一些经典算法难以解决的问题。



这个电路描述了一个量子随机游走算法，它的大致步骤如下：

- 将一个初始量子比特（通常是 $|0\rangle$）放在一个超级位置态中，即 $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$。
- 通过应用 Hadamard 门将初始量子比特放置在一个均匀叠加态上，即 $\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle$。
- 将第一个量子比特和一个辅助量子比特连接在一起，并应用一个 Unitary 门，表示一个随机行走过程，其中 Unitary 门是一个由两个不同的旋转角度的旋转矩阵组成的操作，可以将量子比特从一个状态转移到另一个状态。
- 对辅助量子比特进行测量，并根据结果应用一个控制 NOT 门，从而将第一个量子比特的状态更新为目标状态。
- 重复执行步骤 3-4，直到满足停止条件。

3.0.6. Grover 算法量子电路

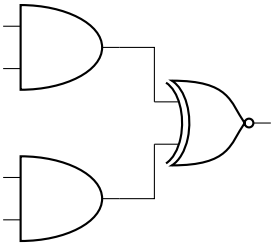


这是一个量子电路，用于描述量子查询算法中的 Grover 算法。这个电路包含了 4 个量子比特和一个未知函数 f(x)，其中，x 和 y 分别表示输入和输出的量子比特。下面是对这个电路的详细解释：

- 在这个电路中，我们将两个量子比特 x 和 y 作为输入和输出。其中，x 是一个 n 比特的量子寄存器，y 是一个 1 比特的输出寄存器。
- 电路的第一个门是一个控制 NOT 门，或称为 CNOT 门，作用在第 1 和第 2 个量子比特上。它将第 2 个比特作为目标比特，第 1 个比特作为控制比特，并在第 1 个比特为 $|1\rangle$ 时对第 2 个比特进行取反操作。因此，这个门可以表示为 $|x,0\rangle \to |x,f(x)\rangle$，其中 $f(x)$ 表示一个未知函数。
- 电路的第二个门是一个受控旋转门，它可以对一个量子比特进行旋转操作，并且旋转角度是根据输入 x 和未知函数 f(x) 计算得到的。在这个电路中，我们使用了一个受控 U 门，它将第 1 个比特作为控制比特，第 2、3、4 个比特作为目标比特，并且将输入 x 传递给未知函数 f(x)。这个门可以表示为 $|x,y,0,0\rangle \to |x,y,0,f(x)\rangle$。
- 电路的第三个门是一个受控反转门，它将第 4 个比特的取反作用于第 3 个比特上。这个门可以表示为 $|x,y,0,f(x)\rangle \to |x,y,f(x),f(x)\rangle$。
- 最后，我们将第 4 个比特传递到输出寄存器 y 中，即 $|x,y,f(x),f(x)\rangle \to |x,y,y\oplus f(x),f(x)\rangle$。

整个电路中的关键部分是受控 U 门，它可以将 Grover 算法的搜索过程表示为一个量子电路，并且实现了搜索过程中的幅度放大和相位反转操作

3.1. 与经典门的比较



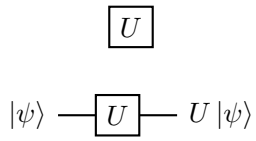
这个电路描述了一个有两个 AND 门和一个 XNOR 门的逻辑电路，其中两个 AND 门的输出分别连接到 XNOR 门的输入。

在经典计算中，我们有一组常用的门：AND（与门）、NOT（非门）、OR（或门）、NAND（与非门）、XOR（异或门）、FANOUT（扩散门）等。我们可以使用这些门的组合来进行经典计算中的任何计算。能运行这些门的经典计算机是图灵完备的或通用的。实际上，我们可以证明，仅用 NAND 门就足以构造所有其他的经典操作符 [198]。我们可以使用这些基本构件来构建经典电路，例如半

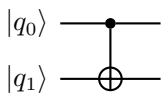
加器电路。然而，AND、OR、XOR、NAND 和 FANOUT 门在量子计算中都无法使用。AND、OR、XOR 和 NAND 门都不是可逆的。FANOUT 门在量子计算中也不被允许，因为它涉及到状态的复制或克隆，这将违反无克隆定理。在主要的经典门中，只有 NOT 操作符可以在量子计算领域使用，因为它是可逆的，且不涉及克隆。

3.2. 量子算符

现在让我们来看一下常用的量子算符。我们用包含代表该算符的字母的框跨越线来表示单量子比特算符。对于二元门，我们用一个跨越两个量子线的算符框表示，对于三元算符，我们用一个跨越三个线的算符框表示，依此类推。请注意，我们本可以选择不同的算符集来实现通用量子计算；选择的算符集是任意的，只要满足通用性测试就足够了，我们将在本章后面介绍这个测试。以下是一元和二元算符的表示：



其中，$|\psi\rangle$ 放在左侧和右侧，并且算符 U 作用于 $|\psi\rangle$，将其转换为 $U|\psi\rangle$。



其中，上图是一个二元算符，这个量子电路描述的是一个 CNOT (Controlled-NOT) 门的操作。

CNOT 门是一个两量子比特的量子逻辑门，其中一个量子比特作为“控制比特”，另一个量子比特作为“目标比特”。在这个电路中，$|q_0\rangle$ 是控制比特，$|q_1\rangle$ 是目标比特。

3.3. 单元运算符

现在我们来介绍一元量子算符的集合。我们首先要研究的是 Pauli 矩阵。这三个矩阵以及单位矩阵和它们的所有 i 和 j 倍构成了所谓的 Pauli 群。首先是 X，它是 NOT 算符（也称为位翻转算符，可以表示为 σ_x）：

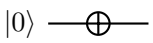
3.3.1. NOT 算符

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{3.12}$$

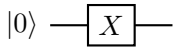
如果我们将 X 应用于 $|0\rangle$，得到：

$$\begin{aligned} X|0\rangle &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0\times 1 + 1\times 0 \\ 1\times 1 + 0\times 0 \end{pmatrix} \\ &= \begin{pmatrix} 0+0 \\ 1+0 \end{pmatrix} \\ &= \begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= |1\rangle \end{aligned}$$

我们可以用电路图表示量子比特的初始状态和应用于它们的算符。我们用以下符号在电路图中表示 X 算符：



这与操作符名称在框内的约定不同，你也可能在电路图中遇到这种约定。



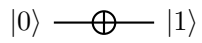
可以用表示 X 算符为

$$X := |0\rangle\langle 1| + |1\rangle\langle 0| \tag{3.13}$$

将 X 算符应用如下

$$X|j\rangle = |j \oplus 1\rangle \tag{3.14}$$

其中 $j \square 0, 1$。这里的 $\oplus$ 操作表示模 2 加法[6]，$j \oplus 1$ 相当于 NOT 操作。所以，如果我们从状态 $|0\rangle$ 开始并应用 NOT，我们得到：



[6]"Addition modulo-2" 是一种特殊的加法运算，也被称为"binary addition" 或"XOR operation"（异或运算）。它的规则很简单：在这种运算中，只有当两个加数不同时结果为 1，否则结果为 0。

具体来说：

- 0 + 0 (模 2) = 0 英文表达 0 + 0 (mod 2) = 0
- 1 + 0 (模 2) = 1 英文表达 1 + 0 (mod 2) = 1
- 0 + 1 (模 2) = 1 英文表达 0 + 1 (mod 2) = 0
- 1 + 1 (模 2) = 0 英文表达 1 + 1 (mod 2) = 1

注意最后一种情况，1 + 1 的结果是 0，而不是我们在普通加法中看到的 2。这就是为什么这种运算被称为"modulo-2"：任何结果都是对 2 取模的结果。这种加法运算在数字电路和信息理论中都有广泛应用，例如在一些校验位和纠错码的计算中。

在量子计算中，类似的运算可以通过 CNOT 门来实现，因为 CNOT 门实现的就是一个模 2 加法。当控制比特为 1 时，目标比特被翻转；当控制比特为 0 时，目标比特保持不变。这与上面的模 2 加法规则相符合。

3.3.2. Y 算符

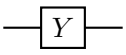
接下来我们有 Y 算符，也表示为 σ_y，它围绕 y 轴旋转状态向量：

$$Y := \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \tag{3.15}$$

因此，如果我们将其应用于 $|1\rangle$ 状态，我们有：

$$\begin{aligned} y|0\rangle &= \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 \times 1 + -i \times 0 \\ i \times 1 + 0 \times 0 \end{pmatrix} \\ &= \begin{pmatrix} 0 - i \\ 0 + 0 \end{pmatrix} \\ &= \begin{pmatrix} -i \\ 0 \end{pmatrix} \\ &= -i|0\rangle \end{aligned}$$

电路图中 Y 算符的表示是：



3.3.3. Z 算符

接着是 Z 算符，也表示为 σ_z，它围绕 z 轴旋转状态向量（也称为相位翻转算符，因为它将其翻转 180 度或 pi 弧度）：

$$Z := \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{3.16}$$

如果我们将 Z 应用于计算基态，我们有

$$Z|j\rangle = (-1)^j |j\rangle \tag{3.17}$$

或者对于特殊情况 $j = 0$，用矩阵形式表示：

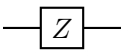
$$
\begin{aligned}
Z\left|j\right\rangle &= (-1)^{j}\left|j\right\rangle \\
&= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 1\times 1+0\times 0 \\ 0\times 1+-1\times 0 \end{pmatrix} \\
&= \begin{pmatrix} 1+0 \\ 0+0 \end{pmatrix} \\
&= \begin{pmatrix} 1 \\ 0 \end{pmatrix} \\
&= (-1)^{0}\left|0\right\rangle \\
&= \left|0\right\rangle
\end{aligned}
$$

对于 $j=1$ 的情况，我们有：

$$
\begin{aligned}
Z\left|j\right\rangle &= (-1)^{j}\left|j\right\rangle \\
&= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 1\times 0+0\times 1 \\ 0\times 0+-1\times 1 \end{pmatrix} \\
&= \begin{pmatrix} 0+0 \\ 0-1 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ -1 \end{pmatrix} \\
&= (-1)^{1}\left|1\right\rangle \\
&= -\left|1\right\rangle
\end{aligned}
$$

注意，我们可以将比特翻转算符 X 与相位翻转算符 Z 相乘，以产生具有全局相位移动 i 的 Y 算符。也就是说，$Y=iXZ$。

电路图中 Z 算符的表示是：



3.3.4. 相位移动算符

接下来我们来看更一般的相位移动算符。当我们应用这个算符时，我们保持状态 $|0\rangle$ 不变，将状态 $|1\rangle$ 旋转角度（或相位）φ，如矩阵所示：

$$
R_{\varphi} := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{pmatrix} \tag{3.18}
$$

所以泡利 Z 算符只是 R_φ 的一个特例，其中 $\varphi = \pi$。让我们回忆一下，根据欧拉恒等式，$e^{i\pi} = -1$，所以我们可以用-1 替换 Z 矩阵中的 $e^{i\pi}$。

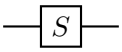
R 算符的电路图表示为：

两个是 R_φ 矩阵特例的附加相位移动算符。

首先，S 算符，其中 $\phi = \frac{\pi}{2}$：

$$S := \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \tag{3.19}$$

S 算符将状态绕 z 轴旋转 90°。S 算符的电路图表示为：

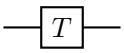


3.3.5. T 算符

接下来让我们来看 T 算符，它使状态绕 z 轴旋转 45°。如果我们给' 赋值为 $\frac{pi}{4}$，那么：

$$T := \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix} \tag{3.20}$$

请注意，$S = T^2$。换句话说，如果我们将 T 矩阵应用到表示状态的向量上，然后将 T 再次应用到第一个操作产生的结果向量上，我们就实现了与一次应用 S 相同的结果（45°+45° = 90°）。T 算符的电路图表示为：



3.3.6. Hadamard 算符

现在让我们来看 Hadamard 算符。这个算符在量子计算中至关重要，因为它使我们能够将一个确定的计算基态转换为两个状态的叠加态。Hadamard 矩阵是：

$$H := \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{3.21}$$

实际上，是数学家**约翰·西尔维斯特**[7]开发了这个矩阵，但我们以**雅克·阿达马**[8]命名 Hadamard 算符的电路图表示为：

[7]**约翰·约瑟夫·西尔维斯特**（John Joseph Sylvester）是一位英国数学家，活动于 19 世纪。他在不少数学领域都有重要的贡献，包括矩阵理论、不变量理论、和数论。

西尔维斯特于 1814 年出生于伦敦，他在剑桥大学学习，但因为他是犹太人，而当时的剑桥对犹太人有歧视，因此他没有获得学位。尽管如此，他还是在各种场合进行了数学研究，并发表了许多重要的论文。

在他的职业生涯中，西尔维斯特曾在许多地方任教，包括美国的约翰斯·霍普金斯大学。他与其他数学家，如詹姆斯·约瑟夫·西尔维斯特（James Joseph Sylvester），一起工作，并发表了许多重要的研究成果。

西尔维斯特在 1897 年去世，但他在数学领域的影响力持续至今。他的许多理论和方法，例如矩阵理论和不变量理论，现在仍然是现代数学的重要组成部分。

[8]**雅克·阿达马**（Jacques Hadamard）是一位法国数学家，活动于 19 世纪和 20 世纪初。他在许多数学领域都有重要的贡献，包括复

如果我们将 Hadamard 应用于状态 $|0\rangle$，我们得到：

$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1\times 1+1\times 0 \\ 1\times 1+-1\times 0 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1+0 \\ 1+0 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \\ &= \frac{|0\rangle+|1\rangle}{\sqrt{2}} \end{aligned}$$

对于状态 $|1\rangle$，我们有：

$$\begin{aligned} H|1\rangle &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} 0 \\ 1 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1\times 0+1\times 1 \\ 1\times 0+-1\times 1 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}}\begin{pmatrix} 0+1 \\ 0-1 \end{pmatrix} \\ &= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ -1 \end{pmatrix} \\ &= \frac{|0\rangle-|1\rangle}{\sqrt{2}} \end{aligned}$$

因此，我们可以看到 H 算符将计算基态投影到状态叠加 $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ 或 $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$，具体取决于初始状态。

概率的和必须为 1，因为测量后会出现其中一个状态。在讨论二元算符之前，让我们定义恒等算符，然后确定哪些算符可以表示为其他算符的序列。恒等算符只是保持量子比特当前状态的矩阵。对于一个量子比特，我们可以使用：

$$I := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{3.22}$$

分析、数论、拓扑学和偏微分方程。

阿达马于 1865 年出生在法国巴黎。他在巴黎高等师范学校（École Normale Supérieure）学习，并在那里开始他的研究生涯。在他的职业生涯中，阿达马在许多不同的学校教书，包括巴黎大学和普林斯顿大学。

阿达马可能最为人所知的工作是他对素数分布的研究，这被称为阿达马定理。他也为解决偏微分方程提供了重要的方法，这被称为阿达马法则。此外，他还对概率论和统计力学做出了贡献。

阿达马在 1963 年去世，但他在数学领域的贡献对现代数学有着深远的影响。

在介绍了一元算符集合之后，我们可以展示以下恒等式：

$$HXH = Z$$

$HXH = Z$ 描述的是两个 Hadamard 门（H）在一个 Pauli-X 门（X）的两侧的组合效果等同于一个 Pauli-Z 门（Z）。这是一个重要的量子门恒等式，即两个 Hadamard 门和一个 Pauli-X 门的组合可以实现 Pauli-Z 门的功能。所以，$HXH = Z$ 的含义就是，如果你先应用一个 Hadamard 门，然后应用一个 Pauli-X 门，再然后应用另一个 Hadamard 门，那么你得到的最终效果就和直接应用一个 Pauli-Z 门一样。

$$HZH = X$$

$HZH = X$ 描述的是两个 Hadamard 门（H）在一个 Pauli-Z 门（Z）的两侧的组合效果等同于一个 Pauli-X 门（X）。这是一个重要的量子门恒等式，即两个 Hadamard 门和一个 Pauli-Z 门的组合可以实现 Pauli-X 门的功能。所以，$HZH = X$ 的含义就是，如果你先应用一个 Hadamard 门，然后应用一个 Pauli-Z 门，再然后应用另一个 Hadamard 门，那么你得到的最终效果就和直接应用一个 Pauli-X 门一样。这种变换在量子计算和量子信息处理中是很常见的。

$$HYH = -Y$$

$HYH = -Y$ 描述的是两个 Hadamard 门（H）在一个 Pauli-Y 门（Y）的两侧的组合效果等同于一个负的 Pauli-Y 门（-Y）。这是一个重要的量子门恒等式，即两个 Hadamard 门和一个 Pauli-Y 门的组合可以实现负的 Pauli-Y 门的功能。所以，$HYH = -Y$ 的含义就是，如果你先应用一个 Hadamard 门，然后应用一个 Pauli-Y 门，再然后应用另一个 Hadamard 门，那么你得到的最终效果就和直接应用一个负的 Pauli-Y 门一样。这种变换在量子计算和量子信息处理中是很常见的。

$$H^\dagger = H$$

$H^\dagger = H$ 表示 Hadamard 门（H）是自己的厄米共轭（hermitian conjugate），也称为伴随（adjoint）。在这种情况下，Hadamard 门是自伴的（self-adjoint）或厄米的（hermitian），这意味着它是自己的伴随。因此，$H^\dagger = H$ 的含义是，如果你应用 Hadamard 门，然后应用它的伴随，那么你得到的就是应用 Hadamard 门两次的效果，这就是应用恒等操作（即没有改变任何东西）。这是因为 Hadamard 门是自伴的，所以它的伴随就是它自己。

$$H^2 = I$$

在量子计算中，$H^2 = I$ 表示的是当你连续两次应用 Hadamard 门（H）时，你会得到一个恒等操作（I）。换句话说，对一个量子态连续应用两次 Hadamard 门，将会把这个量子态恢复到它的原始状态。

这个公式的一个直观的理解方式是，Hadamard 门的作用是把量子比特的基态 $|0\rangle$ 和 $|1\rangle$ 转换为它们的叠加态。如果你连续两次应用 Hadamard 门，那么你首先会得到一个叠加态，然后这个叠加态会被再次转换回它的原始基态。

数学上，这可以通过 Hadamard 门的矩阵表示进行验证。Hadamard 门的矩阵表示是：

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

如果你计算这个矩阵的平方，你会发现结果就是单位矩阵 I：

$$H^2 = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$$

所以，$H^2 = I$ 就是说连续两次应用 Hadamard 门相当于没有做任何操作。

3.4. 二元算符

在物理学和量子力学中，”二元算符”通常指的是作用于两个量子系统（例如，两个量子比特或两个粒子）的算符。这些算符可以描述两个系统之间的相互作用或者联合演化。例如，量子计算中的受控非门（CNOT 门）就是一个二元算符，它在一个量子比特上进行操作，但这个操作是否进行取决于另一个量子比特的状态。

现在让我们考虑两个量子比特或二元算符。在一个双量子比特系统中，按照惯例，我们使用以下计算基态：

$$|00\rangle = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \quad |01\rangle = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \quad |10\rangle = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \quad |11\rangle = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{3.23}$$

3.4.1. SWAP 算符

首先让我们讨论 SWAP 算符。SWAP 将状态 $|01\rangle$ 变为 $|10\rangle$，当然，也将 $|10\rangle$ 变为 $|01\rangle$。我们可以用以下矩阵表示这个算符：

$$SWAP = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.24}$$

并将其应用于表示状态 $|01\rangle$ 的四维向量，如下所示：

$$SWAP\,|01\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \tag{3.25}$$

$$= \begin{pmatrix} 1\times 0 + 0\times 1 + 0\times 0 + 0\times 0 \\ 0\times 0 + 0\times 1 + 0\times 0 + 0\times 0 \\ 0\times 0 + 1\times 1 + 0\times 0 + 0\times 0 \\ 0\times 0 + 0\times 1 + 0\times 0 + 1\times 0 \end{pmatrix} \tag{3.26}$$

$$= \begin{pmatrix} 0+0+0+0 \\ 0+0+0+0 \\ 0+1+0+0 \\ 0+0+0+0 \end{pmatrix} \tag{3.27}$$

$$= \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \tag{3.28}$$

$$= |10\rangle \tag{3.29}$$

这个公式描述了 SWAP（交换）门在量子态 $|01\rangle$ 上的操作。SWAP 门是一个二元量子门，它交换两个量子比特的状态。这个公式的结构如下：

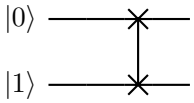
- 首先，公式左侧的 $SWAP\,|01\rangle$ 表示我们将要在状态 $|01\rangle$ 上应用 SWAP 门。
- 然后，我们看到 SWAP 门的矩阵表示，它是一个 4×4 的矩阵。这个矩阵的对角线元素都是 1，这表示当两个量子比特处于相同的状态（即，它们都是 $|00\rangle$ 或 $|11\rangle$）时，SWAP 门不会改变这个状态。同时，这个矩阵的其它两个 1 分别在第二行第三列和第三行第二列的位置，这表示当两个量子比特处于不同的状态时（即，一个是 $|0\rangle$，另一个是 $|1\rangle$），SWAP 门会交换这两个状态。
- 然后，我们看到状态 $|01\rangle$ 的列向量表示。这是一个 4×1 的向量，其中的元素对应于量子态 $|00\rangle$，$|01\rangle$，$|10\rangle$ 和 $|11\rangle$。在这个例子中，第二个元素是 1，表示我们的初始状态是 $|01\rangle$。
- 在应用 SWAP 门后，我们得到的新的状态向量是

$$\begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

，这对应于量子态 $|10\rangle$。

所以，整个公式的意思是，当我们在量子态 $|01\rangle$ 上应用 SWAP 门，我们会得到新的量子态 $|10\rangle$，即两个量子比特的状态被交换了。

请确保这个算符应用于两个量子比特计算基向量之一会得到期望的结果。对于 SWAP 算符的电路图，我们使用：



3.4.2. 受控非门（CNOT）

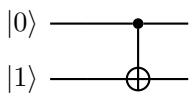
现在我们来到量子计算的关键算符——受控非门（CNOT）。在这个二元算符中，我们将第一个量子比特视为控制量子比特，第二个量子比特视为目标量子比特。如果控制量子比特处于状态 |0□，那么我们不对目标量子比特做任何操作。然而，如果控制量子比特处于状态 |1□，那么我们将对目标量子比特应用非门（X）算符。我们使用 CNOT 门在量子计算机中纠缠两个量子比特。我们可以用以下矩阵表示 CNOT：

$$CNOT := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.30}$$

例如，我们计算 CNOT 对状态 $|10\rangle$ 的作用如下：

$$CNOT\,|10\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0+0+0+0 \\ 0+0+0+0 \\ 0+0+0+0 \\ 0+0+1+0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = |11\rangle \tag{3.31}$$

对于电路图，我们以这种方式表示 CNOT：



3.4.3. SWAP 和 CNOT 算符的恒等式

$$SWAP_{ij} = CNOT_{ji}CNOT_{ji}CNOT_{ji} \tag{3.32}$$

这个等式说明我们可以用三个受控非门（CNOT 门）构建一个交换（SWAP）门。这在量子计算中非常有用，因为 CNOT 门是一种通用的量子门，可以在任何量子计算设备上实现。

等式的左边是一个 SWAP 门，它交换 i 和 j 两个量子比特的状态。

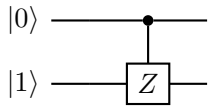
等式的右边是三个 CNOT 门的序列。在这个序列中，每个 CNOT 门都是在 i 和 j 之间进行操作的。具体来说，第一个 CNOT 门检查 i 的状态，如果 i 的状态是 $|1\rangle$，那么它就把 j 的状态翻转；否则，它不做任何事情。第二个 CNOT 门也做同样的检查，但这次是检查 j 的状态，并可能翻转 i 的状态。最后，第三个 CNOT 门再次检查 i 的状态，并可能翻转 j 的状态。

通过这样的操作，如果 i 和 j 的初始状态不同，那么这三个 CNOT 门的效果就是交换 i 和 j 的状态，从而实现了 SWAP 门的功能。如果 i 和 j 的初始状态相同，那么这三个 CNOT 门的效果就是保持 i 和 j 的状态不变，这也符合 SWAP 门的定义。

需要注意的是，这个等式使用的是 i 和 j 的指标来区分不同的 CNOT 门，这在描述复杂的量子电路时是很常见的做法。

3.4.4. 控制算符：受控 Z 门 CZ

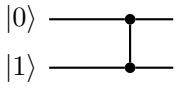
现在让我们转向另一个控制算符：CZ。在这里，我们有一个控制量子比特和一个目标量子比特，就像 CNOT 一样；然而，在这个操作中，如果控制量子比特处于状态 $|1\rangle$，那么我们将对目标量子比特应用 Z 算符。我们可以用电路图表示 CZ 算符如下所示：



以及矩阵形式：

$$CZ := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} \tag{3.33}$$

我们可以用电路图表示 CZ 算符，如下：



请注意，与 CNOT 门不同，CZ 门是对称的：我们可以选择任何一个量子比特作为控制或目标，最终都会得到相同的结果。这就是为什么我们可以在电路图中用两个线路上的点来表示 CZ 门。

3.5. 三元算符

我们已经讨论了一元和二元算符。现在让我们考虑三元或 3 量子比特算符。首先，我们有 Toffoli 算符，也称为 CCNOT 门 [42]。就像在 CNOT 算符中一样，我们有控制量子比特和目标量子比特。在这种情况下，前两个量子比特是控制量子比特，第三个是目标量子比特。两个控制量子比特都必须处于 $|1\rangle$ 状态才能修改目标量子比特。另一种思考方式是，前两个量子比特（x 和 y）必须满足布尔与函数——如果结果为真，则对目标量子比特 z 应用非运算（NOT）。我们可以用下面的方式表示这个操作：

$$(x, y, z)(x, y, (z \oplus xy)) \tag{3.34}$$

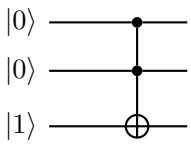
或者，作为矩阵表示：

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{3.35}$$

作为示例，我们将此门应用于状态 $|110\rangle$：

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = |111\rangle \tag{3.36}$$

量子电路表示：



3.5.1. Fredkin 门

接下来，让我们考虑 Fredkin 门，也称为 CSWAP 门 [43]

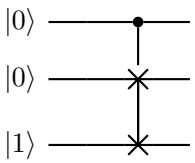
。当我们应用此操作符时，第一个量子比特是控制量子比特，另外两个是目标量子比特。如果第一个量子比特处于 $|0\rangle$ 状态，我们什么都不做；如果它处于 $|1\rangle$ 状态，我们将另外两个量子比特互相交换。表示这个操作的矩阵是：

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.37}$$

例如，Fredkin 门应用于 $|110\rangle$ 得到：

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = |101\rangle \tag{3.38}$$

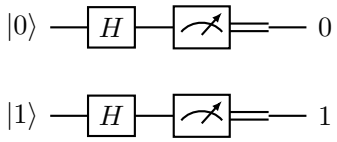
在电路图中，我们使用这个符号表示 Fredkin 操作符。



3.5.2. 与经典门的比较

在经典计算中，我们有一组常用的门：AND、NOT、OR、NAND、XOR、FANOUT 等。我们可以使用这些门的组合来执行经典计算中的任何计算。能够运行这些门的经典计算机是图灵完备或通用的。实际上，我们可以证明，仅使用 NAND 门就足以构造所有其他经典运算符 [44]。我们可以使用这些基本构建块构建经典电路，例如半加器电路。然后，我们可以使用这些元素构建一个全加器：

在量子计算中，AND、OR、XOR、NAND 和 FANOUT 都不能使用。AND、OR、XOR 和 NAND 门不可逆。FANOUT 门在量子计算中不允许使用，因为它涉及状态的复制或克隆，这将违反无克隆定理。在主要的经典门中，只有 NOT 运算符可以在量子计算中使用，因为它是可逆的且不涉及克隆。



在这个示例中，$|[innersep = 4pt, minimumwidth = 1.5pt, minimumheight = 1.5pt]|\quad [arrows]l$ 命令插入了一个 Hadamard 门，$|[meter, label = [mylabel],]|[arrows]l$ 命令插入了一个测量门，表示在经典测量后，量子比特坍缩为一个经典比特。，并且 $[arrows]l$ 分别表示测量后得到的两个可能的经典结果 0 和 1。

注意，这个电路表示的是两个初始状态分别为 $|0\rangle$ 和 $|1\rangle$ 的量子比特经过 Hadamard 操作后，被测量并坍缩为经典比特的过程。

3.5.3. 量子运算符的普适性

如果 NAND 门对于经典计算是普适的，那么是否存在一种门或一组门对于量子计算是普适的呢？实际上，有几种一元和二元运算符的组合可以导致普适性。没有一组一元门可以单独实现量子计算的普适性。其中两个产生普适性的门集是：

当与具有实系数的基础变换一元运算符（例如 H）配对时，Toffoli 门对于量子计算是普适的 [45]。另一组普适性的门集是 fCNOT；T；H g [46, 23]。

3.5.4. Gottesman-Knill 和 Solovay-Kitaev

Gottesman-Knill 定理是量子信息学中的一个重要定理，它涉及到一类特殊的量子计算，被称为稳定子计算。

稳定子计算的特点是，它们只使用以下几种元素：

初始状态为 $|0\rangle$ 的量子比特 Hadamard 门、Phase 门和 Pauli 门任意的量子比特测量和经典控制 Gottesman-Knill 定理的内容是：任何使用以上元素构成的量子计算，都可以在经典的多项式时间内精确模拟。换句话说，这类量子计算并不能实现超越经典计算的速度。这个定理的重要性在于，它揭示了量子计算的一些基本限制，并为理解量子计算的优势提供了更深入的视角。

然而，需要注意的是，尽管 Gottesman-Knill 定理显示了一类量子计算可以被经典计算有效模拟，但这并不意味着所有的量子计算都可以。实际上，已经有多种证据表明，一般的量子计算在某些任务上可以显著地超越经典计算，比如分解大整数和搜索无序数据库等问题。

Gottesman-Knill 定理指出，仅使用 Clifford 门构建的电路可以在以下条件下在经典计算机上高效模拟：

- 在计算基础上的状态准备在标准基础上的测量任何条件于测量结果的经典控制

Clifford 算子组由集合 $C = \{CNOT, S, H\}$[9] [47] 生成。

在此交汇点值得考虑的另一个定理是 Solovay-Kitaev 定理。

Solovay-Kitaev 定理是量子计算中的一个重要结果，关于量子门的近似合成。定理的主要内容是：给定一个量子门集（比如通用量子门集），对任意的一个单量子比特门，我们都可以在对数深度的复杂度下，使用给定的量子门集来近似它。

更具体地说，如果你有一个想要实现的目标单量子比特门 U，但是你的量子计算机只能直接实现一组特定的量子门（这组门被称为“门集”），Solovay-Kitaev 定理告诉我们，你可以通过组合（或“合成”）你可用的那些门，来构造一个非常接近 U 的门。而且，定理还告诉我们，为了达到 ε 精度，你所需要的门的数量大约是 $O(\log^c(1/\epsilon))$，其中 c 是一个常数，通常取 4。

这个定理在实际的量子计算中非常重要，因为实际的量子计算机通常只能直接实现一组有限的、固定的量子门，但是我们可能希望能够实现更复杂的、由这些基本门组合而成的量子操作。Solovay-Kitaev 定理给出了一种实现这个目标的方法，尽管这个方法可能需要大量的基本门。

该定理指出，如果一组单量子比特门生成 $SU(2)$ 的密集子集，其中 $SU(2)$ 是大小为 2 x 2 的幺正矩阵的特殊幺正群，则可以保证该集合可以快速填充 $SU(2)$，即可以使用给定生成集的短序列从好的逼近任何所需的门 [48]。该定理推广到多量子比特门和 $SU(d)$ 的算子 [48]。所有有限的通用门集都可以用一个有限数量的门模拟任意一个给定的门集，模拟精度为 。更准确地说，如果 L 是电路的大小（即门的数量），则 L 的近似值 L_0 可以用有限数量的门表示，其数量可以用大 O 表示法表示为：

$$L_0 = O\left(L^{\log \frac{4L}{}}\right) \tag{3.39}$$

其中，O 是大 O 符号，表示上限，log 是以 2 为底的对数。这个式子表示，我们可以用数量有限的门集，使用不超过 $L^{\log \frac{4L}{}}$ 个门，来逼近任何给定的门集，精度为 。

深度 D 的近似值 D_0 有一个有界的深度，可以用大 O 表示法指定为：

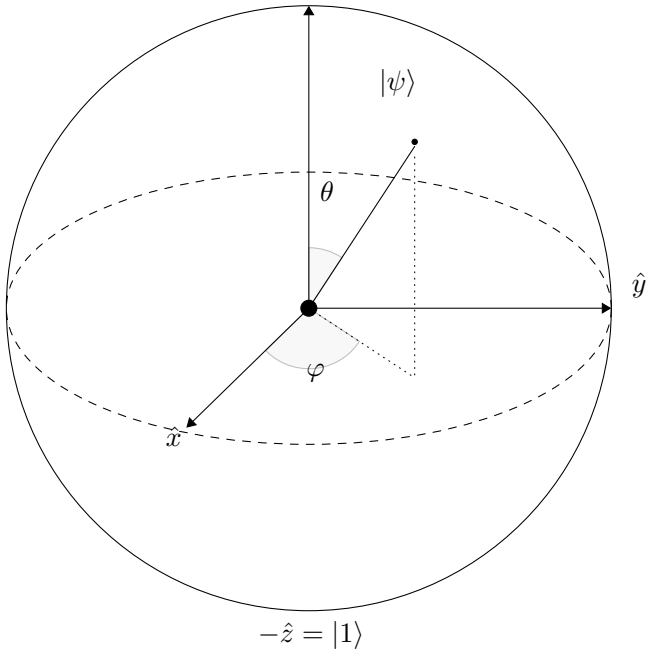
$$D_0 = O\left(L^{\log \frac{4D}{}}\right) \tag{3.40}$$

其中，O 是大 O 符号，表示上限，log 是以 2 为底的对数。这个式子表示，我们可以用数量有限的门集，使用不超过 $L^{\log \frac{4D}{}}$ 个门，来逼近任何给定的门集，精度为 。

因此，这些表达式说明模拟非常高效，效率优于多项式时间。

[9] https://grove-docs.readthedocs.io/en/latest/vqe.html

3.6. The Bloch Sphere



有几种方法可以表示量子比特的状态：

1. 我们可以用 Dirac 符号写出量子态。例如，如果我们有一个量子比特处于状态 $|0\rangle$，然后应用 X 算子，那么我们将在状态 $|1\rangle$ 中找到该量子比特（假设没有外界干扰）。

$$X|0\rangle = |1\rangle$$

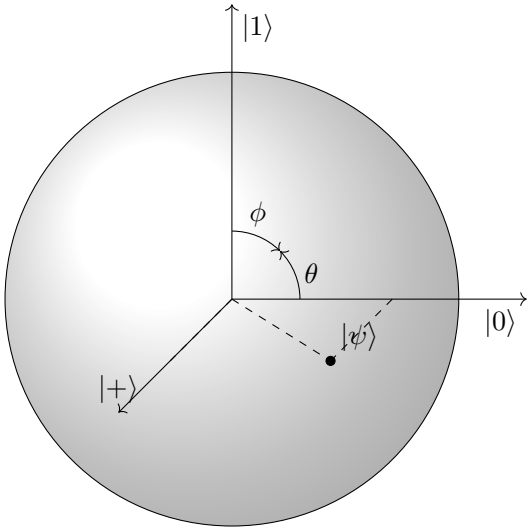
2. 我们可以使用 Bloch 球来表示单个量子比特的状态。量子计算中的任何状态都可以表示为一个向量，该向量从原点开始，终止于单位 Bloch 球的表面。通过对状态向量应用幺正算子，我们可以将状态移动到球上的不同位置。我们约定球的两个对点分别是球的顶部 $|0\rangle$ 和底部 $|1\rangle$。

如图 3.3 所示，使用 Bloch 球进行可视化的一个优点是，我们可以表示叠加态，例如

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

如我们在 x 轴上看到的那样。我们还可以区分包含不同相位的状态，如沿 x 轴和 y 轴显示的状态所示。

让我们回到我们之前讨论的计算普适性。现在我们介绍了 Bloch 球，另一种考虑满足通用计算的门集的方法是使我们能够到达 Bloch 球上的任意点。有关 Bloch 球上量子比特的交互式可视化，请参见本书的在线网站。现在我们已经介绍了在量子计算中使用的主要幺正算子，让我们转向量子计算的状态测量。



在这个图中，球面表示所有可能的单量子比特状态。$|0\rangle$、$|1\rangle$ 和 $|+\rangle$ 分别是球面的 x、y、z 轴。$|\psi\rangle$ 是一个量子态，用一个点来表示。θ 和 ϕ 表示 $|\psi\rangle$ 在球面上的极角和方位角。

3.7. 测量假设

在经典物理中，测量似乎是一个直截了当的过程。测量的行为被认为不会对我们正在测量的物品产生任何影响。此外，我们有能力测量系统的一个属性，获取读数，然后测量另一个属性，并确信第一个已经观测到的属性仍然保留其观察值。但在量子力学中却不是这样，此时测量的行为对观测结果有着深远的影响。

基于量子力学的原理，我们可以将测量假设表述为：

每个可测量的物理量 o 都由一个相应的厄米算符 O 描述，并作用于状态 Ψ 上。

根据这个假设，与每个属性相关联的都存在一个厄米算符，我们称之为可观察量。例如，可观测量 $\hat{x}$ 与粒子的位置相关联。

一个厄米算符等于它的伴随（即复共轭转置）。如果 O 是厄米算符，则我们可以说 $O = O^{\dagger}$（。厄米算符具有其本征值保证为实数的优良性质。当测量物理系统的动量或位置等属性时，我们需要指定一个实数。

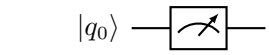
测量的每个可能结果都是可观察量的本征值，并由 $|P|\psi\rangle|^2$ 表示，其中 P 是投影到可观察量的本征空间的投影算符。由于我们对向量进行了归一化，因此测量后的状态由可观察量的单位本征向量表示，其本征值为 λ_i。

我们之前讨论了一个系统可以处于两个或多个状态的叠加态中。我们可以将其表示为正交归一基向量的线性组合。例如：

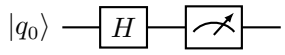
$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{3.41}$$

这引出了一个问题，即我们如何设置测量方式，以获得所需的输出。这又取决于每个状态的振幅，因为正如先前讨论的那样，振幅的模的平方是该状态在测量时出现为输出的概率（伯恩规则）。

我们可以在量子电路中用如下方式表示测量：



现在我们有了我们的幺正算子和测量算子，我们将构建一些基本的量子电路。让我们想一想这个电路的作用：



$$|0\rangle_H \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \rightarrow \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{3.42}$$

我们从两个量子比特开始，称它们为 q0 和 q1，每个量子比特都准备好处于状态 $|0\rangle$。然后我们对 q0 应用 Hadamard 算子，将其置于状态叠加态中

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \tag{3.43}$$

这是一个非常简单的量子电路，只涉及到一个量子比特（$|q_0\rangle$），并且只执行了两个操作：一个 Hadamard 门操作和一个测量。

让我们逐步解析这个电路：

：这表示电路开始时有一个量子比特，状态为 $|0\rangle$。

：这表示在 $|q_0\rangle$ 上应用了一个 Hadamard 门。Hadamard 门是一种量子逻辑门，可以将 $|0\rangle$ 和 $|1\rangle$ 状态变为叠加态。具体来说，它可以把 $|0\rangle$ 变为 $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$，把 $|1\rangle$ 变为 $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$。

：这表示对 $|q_0\rangle$ 进行了测量。量子测量会将量子比特的状态“塌缩”为一个确定的经典状态，即 $|0\rangle$ 或 $|1\rangle$。在测量后，量子比特的状态将成为测量结果。

：这表示电路的结束。所以，这个电路的意思是：我们开始时有一个处于 $|0\rangle$ 状态的量子比特，然后我们在其上应用了 Hadamard 门，使其进入了一个叠加态，最后我们测量这个量子比特，并获得一个经典的测量结果，即 0 或 1。

3.7.1. 测量假设

每个可测量的物理量 o 都由一个相应的厄米算符 O 描述，并作用于状态 $|\psi\rangle$ 上。

根据这个假设，与每个属性相关联的都存在一个厄米算符，我们称之为可观察量。例如，可观测量 x_O 与粒子的位置相关联。

我们回忆一下，一个厄米算符等于它的伴随（即复共轭转置）。如果 O 是厄米算符，则我们可以说 $O = O^\dagger$（有关厄米算符的更多讨论请参见第 12 章）。厄米算符具有其本征值保证为实数的优良性质。当测量物理系统的动量或位置等属性时，我们需要指定一个实数。

测量的每个可能结果都是可观察量的本征值 λ，并由 P 表示，其中 P 是投影到可观察量的本征空间的投影算符。由于我们对向量进行了归一化，因此测量后的状态由可观察量的单位本征向量表示，其本征值为 λ。

我们之前讨论了一个系统可以处于两个或多个状态的叠加态中。我们可以将其表示为正交归一基向量的线性组合。例如：

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{3.44}$$

这引出了一个问题，即我们如何设置测量方式，以获得所需的输出。这又取决于每个状态的振幅，因为正如先前讨论的那样，振幅的模的平方是该状态在测量时出现为输出的概率（伯恩规则）。

我们可以在量子电路中用如下方式表示测量：

$$\text{(测量算子)} \quad \longrightarrow \quad |\psi\rangle \qquad \downarrow \text{(幺正算子)} \qquad |0\rangle$$

现在我们有了我们的幺正算子和测量算子，我们将构建一些基本的量子电路。让我们想一想这个电路的作用：

$$|0\rangle_H \rightarrow \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \rightarrow \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{3.45}$$

我们从两个量子比特开始，称它们为 q_0 和 q_1，每个量子比特都准备好处于状态 $|0\rangle$。然后我们对 q_0 应用 Hadamard 算子，将其置于状态

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \tag{3.46}$$

这将它变为两个状态的叠加态：

$$\frac{1}{\sqrt{2}}(|0\rangle_q |0\rangle_{q1} + |1\rangle_q |1\rangle_{q1}) \tag{3.47}$$

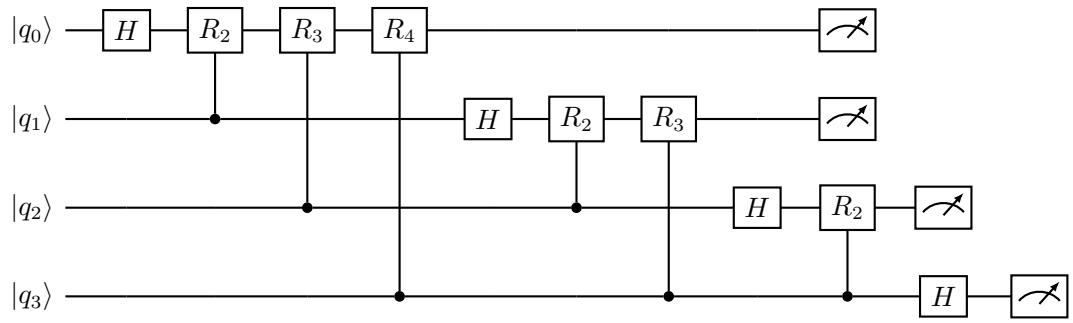
接下来，我们在 q_0 和 q_1 上应用 CNOT。这将两个量子比特缠绕起来，因此我们现在有了两个量子比特的组合、不可分离状态：

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{3.48}$$

我们因此创建了一个贝尔态，或 EPR 对。然后我们对 q_0 进行测量，以 50/50 的概率找到实值输出的 0 或 1 值。

3.8. Computation-in-Place

“Computation-in-Place”是指在大多数基于门的量子计算中，信息是通过量子比特状态的演化来表示的，随着连续应用幺正算符而演变。这种方式与经典计算形成了鲜明的对比，在经典计算中，我们需要将数据传输到各种内存和计算寄存器中。在大多数量子计算中，所有的处理都在量子比特上进行。在量子计算中进行测量后，我们输出实值比特，可以与控制量子处理器的 CPU 共享，如果必要，可以在经典计算机上进一步处理。现在，我们来到了量子计算中的一个关键问题：如果测量基于每个量子比特状态振幅的模的平方，那么我们如何预先确定哪些量子比特将成为输出？Deutsch、Jozsa、Bernstein、Vazirani、Shor 等人认为，我们可以通过在测量之前设置振幅来影响输出，以获得特定计算任务所需的输出结果。实现这个目标的一种方法是量子 Fourier 变换 (QFT——不要与量子场论混淆!)。在测量之前，在所有量子比特上应用 QFT，可以在测量时获得相位信息而不是振幅信息。QFT 是量子计算中的一种有效过程：对于 2n 个振幅的离散 Fourier 变换，只需要应用 Hadamard 和相移算符 $O(n^2)$ 次（其中 n 是量子比特的数量）。我们将在本文稍后更详细地介绍 QFT。在这里，我们展示了 n 为 4 时 QFT 的电路。在我们关注量子硬件之前，让我们在下一章深入了解计算复杂性。这将为我们提供理解哪些问题适合在量子计算机上处理的基础。



量子傅立叶变换 (Quantum Fourier Transform, QFT) 的电路图，它在四个量子比特上执行 QFT。以下是每个步骤的详细解释：

对于每个量子比特 $|q_i\rangle$，从 $i = 0$ 到 3，应用 Hadamard 门和一系列受控相位门。Hadamard 门将量子比特从计算基态（$|0\rangle$ 或 $|1\rangle$）转换到叠加态，而受控相位门根据一个量子比特的状态改变另一个量子比特的相位。对于每个量子比特 $|q_i\rangle$，从 $i = 0$ 到 3，应用一系列受控非门。这些门实现了两个量子比特之间的纠缠。在每个量子比特上执行测量操作（由 |[meter,label=[my label],]|[arrows]| 表示）。这会导致量子比特坍缩到特定的基态（$|0\rangle$ 或 $|1\rangle$），并给出一个经典的测量结果。这个电路是 QFT 的一个例子，它是量子算法中的关键组成部分，例如 Shor 的因数分解算法。QFT 本质上是离散傅立叶变换的量子版本，它可以在量子态的振幅上产生复杂的干涉模式，这在经典计算中是难以实现的。

4

复杂性理论

由于量子计算提供了一种替代计算的方法，因此考虑哪些在经典计算框架下被认为是不可行的问题现在在这种新的范式下是可行的是合乎逻辑的。为了做到这一点，让我们考虑一系列问题类别。

4.1. 问题与算法

首先，让我们明确区分计算问题（或任务）、算法和程序。一个计算问题的例子可能是：给定一个包含 n 个数字的列表，如何按照递增的顺序将这些数字排序?

针对这个问题，我们可以提出各种算法来解决，包括快速排序、归并排序、插入排序等。算法是独立于硬件的、用于解决计算问题的一种策略或方法。我们通常试图找到那些能够有效解决问题的算法。而程序则是在特定的编程语言中对算法的实现。

算法分析是研究算法运行所需资源的领域。我们可以按照时间（即执行步骤的数量）和空间（即内存使用量）来对算法进行分类。相对应的，计算复杂度理论是研究问题本身固有难度的领域；接下来我们将介绍一些重要的问题类别。

在定义复杂度类别时，我们通常关注的是决策问题；这些问题可以用二进制的“是”或“否”来回答。决策问题的复杂度分析确定了找到答案所需的计算资源。为了确定复杂度，我们通常关注最坏情况下的性能。

4.2. 时间复杂性

在之前的章节中，我们讨论了使用大 O 符号表示问题的最坏情况的上限。例如，如果我们有一系列需要排序的项目，大 O 时间复杂度将取决于我们选择的排序算法。以下是一些常见排序算法的复杂度顺序：

插入排序：$O(n^2)$

归并排序：$O(n \log(n))$

Timsort[1]：$O(n \log(n))$

[1]Timsort 是一种混合排序算法，源自插入排序和归并排序。这种算法在对特定类型的数据进行排序时，具有很好的性能。

如果 n 很大，这些差异可能非常显著。在分析算法所需的计算资源时，我们进行渐近分析，即当输入 n 变得非常大时需要的资源。请参见图 4.1，其中列出了常见的大 O 时间计算顺序的图表。

除了大 O 符号表示算法的最坏情况的上限之外，我们还可以考虑大 Ω 符号，它表示最坏情况下的下限。现在添加到上面的列表中，我们可以比较这些排序算法所需计算资源的下限和上限：

- **插入排序**：$\Omega(n)$、$O(n^2)$
- **归并排序**：$\Omega(n \log n)$、$O(n \log n)$
- **Timsort**：$\Omega(n)$、$O(n \log n)$

我们可以观察到，对于插入排序和 Timsort，它们的最优时间复杂度（大 Ω）显然优于它们的最坏时间复杂度（大 O）。然而，因为我们需要为最坏的情况做准备，所以我们通常关注的是大 O 复杂度。

当一个算法的最坏情况时间复杂度（大 O）和最优情况时间复杂度（大 Ω）相匹配时，我们引入了第三个度量方式。对于这些算法，我们可以描述它们的大 Θ 复杂度，因为它们的大 O 和大 Ω 是相匹配的。

为了更正式地表述，我们说一个函数 f(n)（它可以表示运行一个大小为 n 的算法所需的时间）与一个函数 g(n)（它可以表示给定 n 个输入运行另一个算法所需的时间）是同阶的，当且仅当 f(n)/g(n) 的极限值（当 n 趋向于无穷大时）是有限的。

用符号表示：$f(n) = O(g(n))$ 当且仅当 $\lim_{n\to\infty} sup|\frac{f(n)}{g(n)}|$ 在这个表达式中，lim sup 是极限上限或上确界。根据 Hardy 和 Littlewood [49] 以及 Knuth [50] 的描述，我们声明：$f(n) = \Omega(g(n))$ 当且仅当 $g(n) = O(f(n))$

和

$f(n) = \Theta(g(n))$ 当且仅当 $f(n) = O(g(n))$ 和 $f(n) = \Omega(g(n))$

还有两个符号称为 o 和 ω。ω 提供了一个严格的上限（从 O 中删除了等式条件），而 ω 提供了一个严格的下限（从 Ω 中删除了等式条件）。

我们还介绍了完整性的概念。如果问题 G 是 H 类的一部分，并且我们可以这里还介绍了一种计算复杂度的符号——o 和 ω。o 提供了一个严格的上界（从 O 中删除了等式条件），而 ω 提供了一个严格的下界（从 Ω 中删除了等式条件）。

最后，让我们介绍一下完备性的概念。如果一个问题 G 属于类 H，并且我们能够证明类 H 中的所有问题都可以归约到 G，那么我们就说 G 是 H 完备的。换句话说，如果我们有一个子程序 S，当运行时可以解决问题 G，并且这个子程序也可以解决 H 类中的任何问题，那么我们就可以说 H 类中的所有问题都可以归约到 G，G 是 H 完备的。

4.3. 复杂性类别

首先，让我们定义一些经典计算的常见复杂性类别：

Timsort 最初在 Python 中被设计出来，现在已经被许多其他的编程语言，包括 Java、Swift 和 V8 JavaScript 引擎所采用。Timsort 的主要优点是，它在最坏的情况下，也有着良好的时间复杂度 O(n log n)。

Timsort 的工作原理是，它首先在数据中寻找已经有序的部分，这些部分被称为“runs”。然后，它将这些 runs 通过归并排序合并起来。Timsort 会根据数据的特点来调整排序策略，如果数据已经大部分有序，它可以在近乎线性的时间内完成排序。这使得 Timsort 在处理实际数据时，通常比其他 O(n log n) 复杂度的排序算法更有效率。

4.3.1. 多项式时间

在计算复杂度理论中，“P”代表“多项式时间”（Polynomial Time）。具体来说，如果一个问题可以在多项式时间内（即，所需的计算步骤数为输入大小的某个多项式函数）在确定性图灵机（或等价地说，任何常规的序列计算模型）上被解决，那么这个问题就属于 P 类。

在这里，输入大小通常被定义为问题描述的长度。例如，对于一个排序问题，输入大小就是待排序列表的长度；对于一个图问题，输入大小可能是图中的边数和顶点数。

多项式时间的定义是基于这样一个观察：当问题的规模增大时，如果问题的解决时间只是问题规模的某个多项式函数，那么我们通常认为这个问题是可以有效解决的。在这种情况下，即使问题规模有所增加，解决问题所需的计算时间也不会变得过于庞大。这是与指数时间相比的结果，因为当问题规模增加时，指数函数的增长速度要远远超过多项式函数。

举个例子，如果一个问题的解决方法的时间复杂度是 $O(n^2)$，其中 n 是问题的规模（例如，输入数据的数量），那么当 n 增大一倍时，所需的计算时间会增加四倍。相比之下，如果解决方法的时间复杂度是指数的，例如 $O(2^n)$，那么当 n 增大一倍时，所需的计算时间将变为原来的平方。

因此，P 类包含了所有可以在合理时间内解决的问题。这包括许多我们在日常生活中遇到的问题，如排序和搜索问题，以及许多更复杂的问题，如网络流量优化和线性规划。

在计算复杂度理论中，我们通常用大 O 符号表示一个算法的时间复杂度。如果一个问题可以在 $O(n^k)$ 时间内解决，其中 n 是输入大小，k 是一个常数，那么这个问题就属于 P 类。这意味着该问题可以在多项式时间内解决。

例如，考虑一个简单的排序问题，如冒泡排序。冒泡排序的基本步骤是比较相邻的元素，如果它们的顺序错误就交换它们，直到没有更多的需要交换的元素，排序就完成了。这个算法的时间复杂度是 $O(n^2)$，其中 n 是待排序元素的数量。

为什么是 $O(n^2)$ 呢？因为冒泡排序的过程中，我们可能需要遍历待排序元素的列表 n 次，每次遍历都需要进行 n 次操作（比如比较和可能的交换）。因此，总的操作数量是 n*n，即 n^2。所以，冒泡排序的时间复杂度是 $O(n^2)$，它属于 P 类问题。

注意，当我们说一个问题属于 P 类，我们并不是说这个问题的任何解决方法都可以在多项式时间内完成，而是说存在至少一种解决方法可以在多项式时间内完成。例如，虽然冒泡排序的时间复杂度是 $O(n^2)$，但我们也有其他的排序算法，如快速排序，其平均时间复杂度是 $O(n \log n)$，这是一个更优的多项式时间复杂度。所以，排序问题属于 P 类问题。

4.3.2. 非确定多项式时间

非确定性多项式时间（NP）是计算复杂性理论中的一个类别，它包含了所有可以在多项式时间内验证解决方案正确性的问题。这些问题可能在经典计算机上难以解决，但一旦给出了一个解决方案，我们可以容易地检查它是否正确。

非确定性的概念来自于非确定性图灵机，这是一种理论上的计算模型，它可以同时探索多个可能的解决方案路径。在这种模型中，如果存在一个路径能够得到正确的解决方案，那么我们就说这个问题是 NP 类的。

一个著名的 NP 问题是旅行商问题（Traveling Salesman Problem, TSP）。在 TSP 问题中，给定一组城市和每对城市间的旅行距离，旅行商需要找到一条能够访问每个城市一次并返回原点的最短路径。找到这样的路径是非常困难的，因为可能的路径数量随着城市数量的增加呈指数级增长。然而，如果给出了一个路径，我们可以很容易地计算其总距离，从而验证它是否是最短路径。因此，TSP 是 NP 类问题的一个例子。

值得注意的是，P 类是 NP 类的一个子集，因为如果一个问题可以在多项式时间内解决，那么它的解决方案当然也可以在多项式时间内验证。然而，是否所有的 NP 问题都是 P 类问题，也就是 P 是否等于 NP，是计算机科学中的一个未解决的问题，也是“千禧年大奖难题”之一。

非确定性多项式时间（NP）并不直接涉及到一个明确的数学计算方法。其主要关注的是问题的性质，即问题的解是否能在多项式时间内被验证，而非问题的具体解决方案如何被计算出来。

举个例子，假设有一个典型的 NP 问题，称为” 子集和问题”。给定一个整数集合，你的任务是确定是否存在这个集合的一个非空子集，使得子集中所有数字的总和等于 0。这个问题没有已知的多项式时间解决方案，因此它不一定属于 P 类。然而，如果你找到了一个解（即一个总和为 0 的子集），那么我们可以在多项式时间内验证这个解是正确的（只需将子集中的所有数字相加，看总和是否为 0）。因此，子集和问题属于 NP 类。

值得注意的是，虽然我们知道所有 P 类问题也属于 NP 类，但是我们并不知道所有 NP 问题是否都属于 P 类。这就是著名的”P vs NP” 问题，至今仍是一个未解决的问题。如果能证明 P=NP 或者 P≠NP，那么这将是计算机科学的一大突破。

4.3.3. 多项式空间

” 多项式空间”（Polynomial space）是一个在计算复杂性理论中使用的概念，用来衡量解决问题所需的内存或空间的数量。如果一个问题可以在多项式空间内解决，那么它就是属于 PSPACE 类别的问题。

多项式空间（PSPACE）是指在图灵机模型中，那些所有确定性和非确定性的计算问题，他们的空间复杂度是输入大小的多项式函数。换句话说，如果问题的大小是 n，那么解决这个问题所需的内存或空间不会超过 n 的某个固定次数的幂。

PSPACE 是重要的复杂性类别之一，因为它在理论上定义了在有限的计算资源下可以解决的问题的范围。例如，如果一个问题是 PSPACE 完全的（PSPACE-Complete），那么就意味着如果我们有一个多项式时间的算法可以解决这个问题，那么就可以在多项式时间内解决 PSPACE 中的所有问题。这样的问题通常被认为是困难的，因为至今还没有找到任何能在多项式时间内解决 PSPACE 完全问题的算法。

” 多项式空间”（Polynomial space）的概念来自计算机科学的计算复杂性理论，它描述的是解决问题所需的内存或存储空间的量。在数学或计算上，它通常不涉及具体的计算过程，而是一个问题解决方案所需空间的度量。

在图灵机模型中，我们可以将问题的解决过程看作是在一个无限长的纸带上进行的，纸带被划分为了许多的小格子。多项式空间的问题是指那些在解决过程中，所需使用的纸带格子数量是输入大小的多项式函数的问题。

具体来说，假设我们有一个问题，其输入大小为 n。如果我们可以找到一个算法，该算法在解决该问题的过程中，所使用的纸带格子数量不超过 n^k（这里的 k 是一个正整数），那么我们就说这个问题可以在多项式空间内解决。这就是计算多项式空间的过程。

需要注意的是，一个算法是否可以在多项式空间内解决某个问题，并不涉及具体的计算步骤，而是关注的是解决问题所需的存储空间的量。因此，算法的具体实现和优化可能会影响其空间复杂度，从而影响问题是否属于多项式空间。

4.3.4. 有界错误概率多项式时间

”Bounded-error probabilistic polynomial time” (BPP) 是复杂性理论中的一个类别，用于分类某些类型的决策问题。这个类别包含了那些可以在概率图灵机（一种可以进行随机计算的理论计算机模型）上在多项式时间内给出正确答案的问题，其错误概率小于 1/3。这个错误概率 1/3 是常数，可以被任何小于 1/2 的常数所替换。

具体来说，我们说一个问题属于 BPP，如果存在一个多项式时间的算法，使得对于任何给定的输入：

”YES”，那么算法至少有 2/3 的概率输出”YES”。”NO”，那么算法至少有 2/3 的概率输出”NO”。

这个错误概率的界限（1/3）是可以被调整的，因为通过多次运行算法并取最常见的输出作为结果，可以将错误概率减小到任何想要的程度。例如，如果你运行算法 100 次，取最常见的结果，那么错误概率将远小于 1/3。

一个经典的 BPP 算法的例子是 Miller-Rabin 素数测试。这个算法是用来检查一个数是否是素数。给定一个输入 n，它会随机选择一个数 a，然后进行一些计算来检查 n 是否是素数。这个算法可能会出错，但是如果 n 确实是一个素数，那么它至少有 3/4 的概率会正确地识别出来。如果 n 不是素数，那么它至少有 3/4 的概率会正确地识别出来。因此，通过多次运行这个算法并取最常见的结果，我们可以使错误的概率降低到任意小。

所以，BPP 算法的运行和验证过程主要涉及到概率和统计，而不是具体的数学计算步骤。这是因为这些算法的正确性是基于概率的，而不是确定性的。

BPP 是非常重要的一个复杂性类别，因为它描述了我们能够在实践中解决的问题的范围。许多实际的算法，如素数测试和近似算法，都是 BPP 的例子。

4.3.5. "Bounded-error Quantum Polynomial time" (BQP)

”Bounded-error Quantum Polynomial time” (BQP) 是复杂性理论中的一个类别，用于分类在量子计算机上可以在多项式时间内解决的决策问题，并且错误率有界。

BQP 中的问题有一个很重要的特性，即它们可以在量子计算机上通过量子算法在多项式时间内解决。这是因为量子计算机利用了量子力学的一些特性，如叠加状态和纠缠，使其能够在某些情况下比经典计算机更快地解决问题。

一个问题被认为在 BQP 中，如果存在一个量子算法，使得对于任何给定的输入：

- 如果正确答案是”YES”，那么算法至少有 2/3 的概率测量到”YES” 的结果。
- 如果正确答案是”NO”，那么算法至少有 2/3 的概率测量到”NO” 的结果。

这里的 2/3 是任意小于 1/2 的常数，通过多次运行算法并取最常见的结果，可以将错误率降低到任意小。

一个著名的 BQP 的例子是 Shor 的算法，它是一个用于整数分解的量子算法。对于大的合数，Shor 的算法在量子计算机上运行的速度比在经典计算机上已知的任何算法都要快。这对于许多现代的加密系统来说是一个重大的威胁，因为它们的安全性是建立在整数分解是一个在经典计算机上难以解决的问题的基础上的。

4.3.6. Exact quantum polynomial time

”Exact Quantum Polynomial time” (EQP) 是计算复杂性理论中的一个类别，它包含那些可以在量子计算机上在多项式时间内精确解决的决策问题。

量子计算机利用量子力学的一些特性，如叠加状态和纠缠，使得它在某些情况下能够比经典计算机更快地解决问题。在 EQP 中，我们关注的是那些在量子计算机上可以精确解决，而不只是近似解决的问题。

一个问题被认为在 EQP 中，如果存在一个量子算法，使得对于任何给定的输入：

如果正确答案是"YES"，那么算法总是精确地输出"YES"。如果正确答案是"NO"，那么算法总是精确地输出"NO"。也就是说，EQP 算法不会产生错误。然而，这种类别的问题相对较少，因为在许多情况下，量子算法只能给出近似答案（这就导致了 BQP 类别的产生）。

需要注意的是，EQP 类别的存在并不意味着所有的问题都可以在量子计算机上精确解决。事实上，许多重要的问题在量子计算机上仍然是困难的，包括一些在经典计算机上也是困难的问题，如 NP-完全问题。

4.3.7. NP

"Quantum Merlin-Arthur" (QMA) 是计算复杂性理论中的一个类别，它被视为量子版本的 NP（nondeterministic polynomial time）类别。在 QMA 中的决策问题可以通过一个量子证据（或者说是一个量子态），在多项式时间内由一个量子计算机（Arthur）验证其正确性。

Merlin-Arthur 类别中的问题通常被描述为 Merlin 尝试去说服 Arthur 一个事实的真实性，而 Arthur 则是一个悲观的多疑者，只有在看到一个有效证据后才会相信。在经典版本的 Merlin-Arthur 类别中，证据是一个经典的比特串，而在量子版本中，证据变成了一个量子态。

具体来说，一个决策问题在 QMA 中，如果存在一个量子算法，使得对于任何给定的输入：

- 如果正确答案是"YES"，那么存在一个量子态，使得算法接受这个量子态为证据的概率至少为 2/3。
- 如果正确答案是"NO"，那么对于任何量子态，算法接受这个量子态为证据的概率最多为 1/3。

这里的 2/3 和 1/3 是可以被调整的常数，只要它们满足"YES" 实例的接受概率大于"NO" 实例的接受概率即可。

QMA 类别的一个经典问题是量子满足性问题（Quantum Satisfiability Problem）。

经典复杂性类别	量子复杂性类别
P (确定性多项式时间)	P (确定性多项式时间)
NP (非确定性多项式时间)	BQP (有界错误量子多项式时间)
PSPACE (多项式空间)	QSPACE (量子多项式空间)
BPP (有界错误概率多项式时间)	EQP (精确量子多项式时间)
MA (Merlin-Arthur)	QMA (量子 Merlin-Arthur)

表 4.1: 经典与量子复杂性类别表格

4.4. 量子计算与 Church-Turing 假设

这里的 Church-Turing 假设，又称为 Church-Turing 论题或者可计算性论题，是一个关于算法和计算的重要理论。它主张：任何人类直观上可以做的" 有效" 或者" 系统的" 计算都可以通过一个具有固定数量的简单但精确的指令（即一个算法或者程序）在一个 Turing 机上完成。这个假设在理论计算机科学和形式语言理论中起着基础性的作用。

而量子计算则是一种新的计算范式，利用量子力学的一些特性（如叠加和纠缠）来进行信息处理。量子计算机在某些问题上，例如整数分解和搜索无序数据库，可以比经典计算机更快。

” 量子计算与 Church-Turing 假设” 这个主题可能会探讨量子计算如何与这个传统的计算理论相适应，或者说量子计算如何可能改变我们对于可计算性和复杂性的理解。

4.4.1. Church-Turing Thesis (CTT)

“如果一个算法可以在任何硬件上执行（比如现代个人计算机），那么就存在一个等效的通用图灵机（UTM）的算法，可以执行完全相同的任务 [23] 。”

Church-Turing 论题（CTT，Church-Turing Thesis），又称图灵-丘奇论题，是计算理论中的一个基本原则。这个论题并不是一个被严格证明过的定理，而是一种广泛接受的假设，用于定义什么是“计算”以及什么是“算法”。

Church-Turing 论题的主要内容可以概括为：任何在物理世界中可计算的函数，都可以通过一个图灵机来计算。换句话说，图灵机可以模拟任何“有效的”计算过程。这里的“有效”通常被理解为一个人类可以通过纸和笔在有限的时间内完成的计算过程。

这个论题最初是由**阿隆佐 · 丘奇**[2]和**艾伦 · 图灵**[3]在 1936 年独立提出的。丘奇提出了 λ 演算，而图灵提出了图灵机，他们分别用这两种方式来定义什么是“有效计算”。然后他们证明了 λ 演算和图灵机的等价性，即他们可以模拟彼此的计算过程。

Church-Turing 论题对于计算机科学有着深远的影响。首先，它提供了一个理论框架，用于讨论什么问题可以被计算解决，以及什么问题不能被计算解决。此外，它也为研究计算复杂性提供了基础，例如 P 问题和 NP 问题。

尽管 Church-Turing 论题至今还没有被证明，但是我们还没有找到任何可以超越图灵机能力的计算模型。包括量子计算在内的现代计算模型，虽然在某些问题上比图灵机更有效率，但是他们

[2]**阿隆佐 · 丘奇**（Alonzo Church，1903 年 6 月 14 日 – 1995 年 8 月 11 日）是一位美国数学家和逻辑学家，他对数理逻辑和理论计算机科学的发展做出了重要贡献。

丘奇最为人所知的工作之一就是他在 1936 年提出的 λ 演算（lambda calculus）。这是一种形式系统，用于研究函数定义、函数应用和递归。λ 演算在计算理论的发展中起到了关键的作用，它为计算机科学提供了基础，特别是在函数式编程语言的设计中，λ 演算的影响深远。

丘奇也因为提出了被称为 Church-Turing 论题（Church-Turing Thesis）的假设而闻名。这个假设是与艾伦 · 图灵（Alan Turing）同时独立提出的，它提出了对“有效计算”或“机械计算”能力的形式化描述。这个论题在理论计算机科学，特别是在计算复杂性理论中，扮演了基础性的角色。

此外，丘奇还提出了所谓的”Church-Rosser 定理”，这个定理是关于 λ 演算的一个重要性质，它在理论计算机科学和数理逻辑中都有重要的应用。

在他的职业生涯中，丘奇在普林斯顿大学和洛杉矶加利福尼亚大学担任教职，他指导了许多后来在数理逻辑和计算机科学领域取得重要成就的研究生，包括艾伦 · 图灵和斯蒂芬 · 科尔 · 克莱尼（Stephen Cole Kleene）。

[3]**艾伦 · 图灵**（Alan Turing，1912 年 6 月 23 日 – 1954 年 6 月 7 日）是一位英国的数学家、逻辑学家和计算机科学的先驱人物。他在理论计算机科学和人工智能领域的贡献特别突出，被广泛认为是计算机科学的奠基人之一。

图灵的主要成就包括以下几个方面：

- 图灵机：在 1936 年，图灵提出了一种理想化的计算设备，称为图灵机，用来形式化算法和计算过程的概念。图灵机至今仍是理论计算机科学中研究算法和计算复杂性的基本模型。
- 图灵可判定性问题：图灵证明了某些问题是无法被图灵机解决的，最著名的例子就是停机问题（Halting Problem）。
- Church-Turing 论题：图灵提出了所谓的图灵-Church 假设，认为任何可以被人类通过机械过程完成的计算，都可以被图灵机完成。这个假设在理论计算机科学中有着基础性的地位。
- 破解德国” 恩尼格玛” 密码机：在第二次世界大战期间，图灵在英国政府密码学学校（Government Code and Cypher School）工作，他设计了一种被称为” 炸弹”（bomb）的机器，用于破解德国使用” 恩尼格玛” 密码机加密的信息，这对于盟军最终赢得战争起到了关键的作用。
- 人工智能：图灵在人工智能领域的贡献也非常重要。他提出了著名的” 图灵测试”，作为评价一个机器是否可以达到人类智能的标准。

图灵在 1954 年去世，年仅 41 岁，然而他的贡献对于现代计算机科学和人工智能的影响深远。

仍然不能计算图灵机不能计算的函数。

这个猜想随后更新以考虑算法的效率。算法效率是指运行算法所使用的特定资源的量化。在比较一组算法的效率时，保持一致性非常重要；如果我们分析一种算法的步数，我们应该对比较中的所有其他算法进行相同的分析，而不是在分析过程中转向内存资源的分析。对算法的有效运行的额外要求给我们带来了强 Church-Turing 假设（SCTT）。

4.4.2. 强 Church-Turing 假设（SCTT）

强 Church-Turing 论题（Strong Church-Turing Thesis，简称 SCTT），有时也被称为 Church-Turing 宇宙性论题，是原始 Church-Turing 论题的一个扩展。强 Church-Turing 论题不仅声明所有”自然” 或” 合理” 的计算模型都可以被图灵机模拟，而且声称这种模拟可以在多项式时间内进行。

更具体地说，强 Church-Turing 论题主张任何” 物理上可实现的” 或” 有效的” 计算过程，都可以在多项式时间复杂度内由一个确定性的（或概率的）图灵机模拟。原始的 Church-Turing 论题并没有对模拟过程的时间复杂度进行限制。

强 Church-Turing 论题在计算机科学的一些领域，特别是复杂性理论和量子计算中，有着重要的地位。在量子计算的背景下，这个论题经常被用来讨论量子计算机是否能够在一些问题上超越经典计算机。

值得注意的是，尽管强 Church-Turing 论题在理论计算机科学中被广泛接受，但它并没有被严格证明，因此还存在争议。特别是在量子计算领域，有些人认为强 Church-Turing 论题可能在某些情况下不成立，因为有证据显示量子计算机可能在某些问题上比经典计算机更有效率。

初始的强 Church-Turing 论题声明所有的算法过程都可以由一个通用图灵机（Universal Turing Machine，UTM）高效地模拟 [23] 。然而，随后的研究发现，可能存在违反强 Church-Turing 论题的情况，即使用随机性的算法。例如，Solovay 和 Strassen 展示了一个使用随机性的算法能够测试一个数的素性 [51]。通过有限次地重复这个算法，我们几乎可以确定地得到正确答案。

后来，研究人员证明存在一个高效的多项式时间算法用于测试素性 [52] ，但历史上，最初的洞见是一个使用随机性的算法可以完成这个任务，这导致了对强 Church-Turing 论题的修正。因此，强 Church-Turing 论题被更新为扩展的 Church-Turing 论题（Extended Church-Turing Thesis，ECTT）。

扩展的 Church-Turing 论题（ECTT）考虑了使用随机性的算法，它声明任何物理上可实现的或有效的计算过程，都可以在多项式时间复杂度内由一个概率的图灵机（也就是可以生成随机数的图灵机）模拟。这是对强 Church-Turing 论题的一种扩展，以适应随机算法的情况。

4.4.3. Extended Church-Turing Thesis (ECTT)

扩展的 Church-Turing 论题（Extended Church-Turing Thesis，ECTT）是对原始的 Church-Turing 论题（CTT）以及强 Church-Turing 论题（SCTT）的进一步扩展。在原始的 Church-Turing 论题中，它提出任何在物理世界中可计算的函数，都可以通过一个图灵机来计算。而强 Church-Turing 论题则进一步提出，这些计算过程都可以在多项式时间内由一个确定性的图灵机模拟。

然而，研究人员发现，存在一些算法可以利用随机性来获得优势，这些算法可以在多项式时间内完成，但是确定性的图灵机可能需要超过多项式时间来模拟。因此，扩展的 Church-Turing 论题应运而生。

扩展的 Church-Turing 论题（ECTT）主张，任何物理上可实现的或有效的计算过程，都可以在多项式时间内由一个概率的图灵机（也就是可以生成随机数的图灵机）模拟。这是对强 Church-Turing

论题的一种扩展，以适应随机算法的情况。

这个论题在理论计算机科学的一些领域，特别是复杂性理论和量子计算中，有着重要的地位。同样，它并没有被严格证明，仍然存在一些争议和例外情况。例如，量子计算机被认为可能违反扩展的 Church-Turing 论题，因为有证据显示量子计算机可能在某些问题上比概率图灵机更有效率。

这是对扩展 Church-Turing 论题（Extended Church-Turing Thesis，ECTT）的一个重要修正，特别是在量子计算的背景下。

这个修正的基础是，量子计算机能够利用量子力学的原理（例如叠加态和量子纠缠）来执行计算，这可能使得它们在解决某些问题上，比任何经典的（包括概率的）图灵机都要快。这是因为量子计算机可以在计算过程中同时探索多个可能的解，而经典的图灵机则只能逐一检查每个可能的解。

这种情况下，任何算法过程都可以由量子计算机有效地模拟，这挑战了扩展 Church-Turing 论题的主张。尽管这个论题声称任何物理上可实现的或有效的计算过程，都可以在多项式时间内由一个概率的图灵机模拟，但是量子计算机可能在某些情况下违反这个论题，因为它们在某些问题上可能比概率图灵机更有效率。

这个修正并不是说扩展 Church-Turing 论题是错误的，而是指出了它的一种局限性，即在量子计算的背景下，它可能不再完全适用。这也表明了量子计算机可能的强大潜力，以及在理论计算机科学中对量子计算理论的研究的重要性。

4.4.4. Quantum Extended Church-Turing Thesis (QECTT)

量子扩展 Church-Turing 论题（Quantum Extended Church-Turing Thesis，QECTT）是对扩展 Church-Turing 论题（ECTT）的进一步修正，考虑了量子计算的情况。

这个论题主张，任何物理上可实现的或有效的计算过程，都可以在多项式时间内由一个量子图灵机（也就是可以执行量子操作的图灵机）模拟。这是对扩展 Church-Turing 论题的一种扩展，以适应量子算法的情况。

量子扩展 Church-Turing 论题的提出，是基于量子计算机在处理某些问题时可能具有的优势。特别是，有一些问题，量子计算机可能在多项式时间内解决，而经典的图灵机（包括概率图灵机）可能需要指数时间才能解决。因此，量子扩展 Church-Turing 论题实际上是在声明，量子计算机的能力是超越经典计算机的最高极限的。

这个论题在理论计算机科学的一些领域，特别是复杂性理论和量子计算中，有着重要的地位。它是理论计算机科学对量子计算潜力的一种重要认识，并为进一步探索量子计算提供了理论基础。

5

构建量子计算机

现在，我们已经介绍了量子计算机的基本工作原理，让我们讨论如何在物理上实现这些设备。有许多不同的门式量子计算机体系结构和设计，每种都有其优缺点。在本章中，我们将介绍量子计算硬件的主要范式。由于技术正在迅速发展，请查看本书的在线网站以获取更新信息。

下面的每种体系结构都需要一组经典计算机来控制系统。例如，如图 5.1 所示，超导量子比特计算机由传统计算机控制。我们在经典计算机上使用高级语言开发量子电路协议，然后可以操作量子系统来应用电路中的运算符。在测量后，QC 的输出是经典信息，该信息反馈到经典计算机进行读取或进一步处理。

我们可以将计算中的 QC 部分视为更大计算的子程序，其中许多任务可能在经典范畴内完成。例如，在应用 Shor 算法时，许多任务由经典计算机执行，然后将难点——周期发现算法（相当于质因数分解）——作为子程序发送到 QC。然后将该输出集成回经典平台。

5.1. 评估一台量子计算机

在量子硬件方面出现了许多进展。以下是一份有用的核对表，以分析该领域工程进展的潜在影响

- **通用性**：首先要问的一个问题是，被介绍为量子计算机的硬件平台是否是图灵完备的，或者是通用的。量子计算机的通用性指的是其是否具备图灵完备性，即它是否可以模拟任何图灵机。一个量子计算机如果具备图灵完备性，则意味着它能够模拟任何可计算的算法，并且可以在理论上解决任何问题。在实际应用中，图灵完备性是量子计算机能够完成广泛计算任务的重要基础。因此，许多量子计算机的设计和构建都以实现图灵完备性为目标。例如，该设备可能是非通用的退火器。我们可以参考第 2 章中描述的 DiVincenzo 标准来进行测试。例如，DiVincenzo 要求量子计算机中的量子比特必须是可单独寻址的。如果我们有一个由原子集合组成的双级系统，但其中的量子比特不能单独寻址，那么该系统将无法通过量子计算测试。
- **保真度**：量子比特的保真度，即量子比特在进行计算过程中，能够保持在量子态中的时间长度和状态的准确度。保真度通常用 1 减去错误率来计算，即保真度等于 1 减去量子比特执行

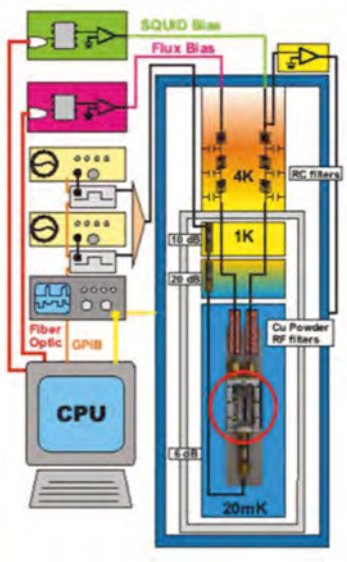


图 5.1: 经典中央处理器控制超导量子计算机量子计算系统中，使用经典计算机的中央处理器（CPU）来控制和管理超导量子计算机的运行。超导量子计算机的运行需要精确的控制和调度，包括量子比特的初始化、量子门的操作以及量子态的测量等步骤，这些都需要通过外部设备来实现。这就是经典 CPU 的作用。它接收用户的指令，然后生成相应的控制信号，通过与超导量子计算机连接的控制线路，对超导量子计算机进行精确的控制和调度。同时，经典 CPU 还负责接收和处理超导量子计算机的输出信号，例如量子测量的结果。这些信息可以用来指导后续的量子运算，或者直接作为计算结果返回给用户。因此，虽然超导量子计算机是量子计算系统的核心部分，但它的正常运行是离不开经典 CPU 的。

操作时出错的概率。在量子计算中，保持量子比特的高保真度非常重要，因为它直接影响到计算的正确性和精度。因此，保真度是衡量量子计算机性能和质量的重要指标之一。虽然将焦点放在每个平台上量子比特的数量上是很诱人的，但我们建议首先检查这些量子比特的保真度的声称。保真度是衡量量子比特在计算中保持相干性的能力的指标；具体而言，它被计算为：1-误差率。当我们面对一个系统时，检查单个和双量子比特操作下量子比特的保真度是有用的。例如，当跨越两个量子比特进行 CNOT 时，保真度更难维持，而应用单比特操作（如 X 或 Y）时更容易维持保真度。

- **可扩展性**：能否实现可扩展性，即能否在更大规模的计算中实现高效的量子计算。在量子计算中，同时保持多个量子比特的相干态非常困难，随着量子比特数目的增加，量子计算机的复杂度和噪声等问题也会随之增加。因此，要实现实用的量子计算机，需要解决实现大规模量子比特的技术难题，并建立可靠的错误校正和容错机制，以确保计算的可靠性和精度。因此，可扩展性是量子计算机设计和开发过程中必须考虑的一个关键因素。该架构是否可扩展到 106 个及以上的量子比特？虽然在此阶段在多个硬件框架上创建 NISQ 阶段的量子计算机有一定好处，但检查在该体系结构中实现容错平台的能力也同样重要。
- **量子比特**：量子计算机的 Qubits 指的是量子比特，它是量子计算机中的基本计算单元。量子比特与经典比特不同，可以处于多个状态的叠加态中，并且可以进行纠缠等量子特性操作。量子计算机中的计算过程是通过对量子比特进行一系列操作来实现的，这些操作可以被描述为量子逻辑门。由于量子比特的特殊性质，量子计算机具有远远超越经典计算机的计算能力和速度。量子比特的数目是衡量量子计算机规模和计算能力的重要指标之一。一旦考虑了上述问题，我们就可以关注量子比特的数量。维持越来越多量子比特的同时相干性是一个技术挑战。此外，还要注意量子比特的架构特定限制，例如最近邻连接。例如，在某些平台上，如果我们对相邻的两个量子比特进行操作，则由于可能的串扰，同时不能使用其他相邻的量子

比特。

- **电路深度**：量子计算机的 Circuit depth 指的是量子电路的深度，即执行一个量子算法所需的操作次数。在量子计算中，一个量子电路通常由一系列的单量子比特操作和双量子比特门操作组成，每个操作都需要一定的时间来执行。Circuit depth 是衡量量子计算机效率和速度的一个重要指标，因为它直接影响到计算的执行时间和可扩展性。在实际应用中，通过设计和优化量子电路来降低 Circuit depth，可以提高量子计算机的计算效率和速度。这是指在相干性失效之前我们可以实现多少操作。一个 100,000 量子比特的计算机可能很好，但如果在失去相干性之前不能执行超过几个操作，那么它的价值是有限的。
- **逻辑连接**：量子计算机的 Logical connectivity 指的是量子比特之间的逻辑连接关系，即能否实现任意两个量子比特之间的双量子门操作。在量子计算中，双量子门操作是实现量子比特之间相互作用和量子纠缠的重要手段。如果量子计算机中的量子比特之间存在逻辑连接限制，例如只能在某些特定的量子比特之间实现双量子门操作，那么就需要通过引入逻辑 SWAP 操作来实现更广泛的逻辑连接，这会增加量子电路的复杂度和错误率。因此，Logical connectivity 是衡量量子计算机可扩展性和性能的一个重要指标。我们是否可以在任何一对量子比特上实现两量子比特门，还是只能在某些对上实现？有限的逻辑连接要求在我们的算法中插入逻辑 SWAP 操作，以有效模拟更大的连接性。更多操作意味着更多的噪声和误差。
- **云访问**：量子计算机的 Cloud access 指的是其在云平台上的可用性和易用性。由于大多数组织无法购买和构建自己的量子计算机，因此它们通常会依赖于学术和商业提供商来提供云端访问。因此，Cloud access 是衡量量子计算机可用性和实用性的一个重要因素。对于量子计算机云服务提供商，他们需要确保量子计算机的可用性、可靠性和安全性，并提供易用的接口和工具，使用户能够轻松地使用量子计算机进行计算。在云计算中，还需要考虑重置时间和平台维护等因素，以确保满足服务级别协议（SLA）的要求。硬件是否易于通过云平台提供？大多数组织不太可能购买或构建自己的量子计算机，而是依赖一系列学术和商业供应商来提供云访问。在此类别中考虑的标准包括计算之间的重置时间以及保持平台可用以满足其服务级别协议所需的工作量。

5.1.1. qubits

与经典计算机的比特（只能为 0 或 1）不同，量子比特可以同时存在于多个状态。这是通过量子叠加和纠缠这两种特性实现的，这使得量子计算机在处理特定类型的问题时，比传统的数字计算机更加强大和快速。

在你引用的文章的上下文中，”qubits” 可能是指在 FT-QuMA 微体系结构中的量子比特。这些量子比特需要通过微体系结构进行精细控制和操作，以执行量子计算任务，并通过错误纠正和容错机制来降低错误率。

5.1.2. Quantum-Classical Interface

在量子计算中通常是指连接量子处理器和经典处理器的部分，它负责在量子和经典系统之间转换和传递信息。

在量子计算机中，量子处理器负责执行量子操作，例如对量子比特的操作。然而，大部分的控制逻辑、错误处理和数据处理仍然在经典处理器上完成。这是因为这些任务在经典计算机上执行通常更有效率，也更可靠。

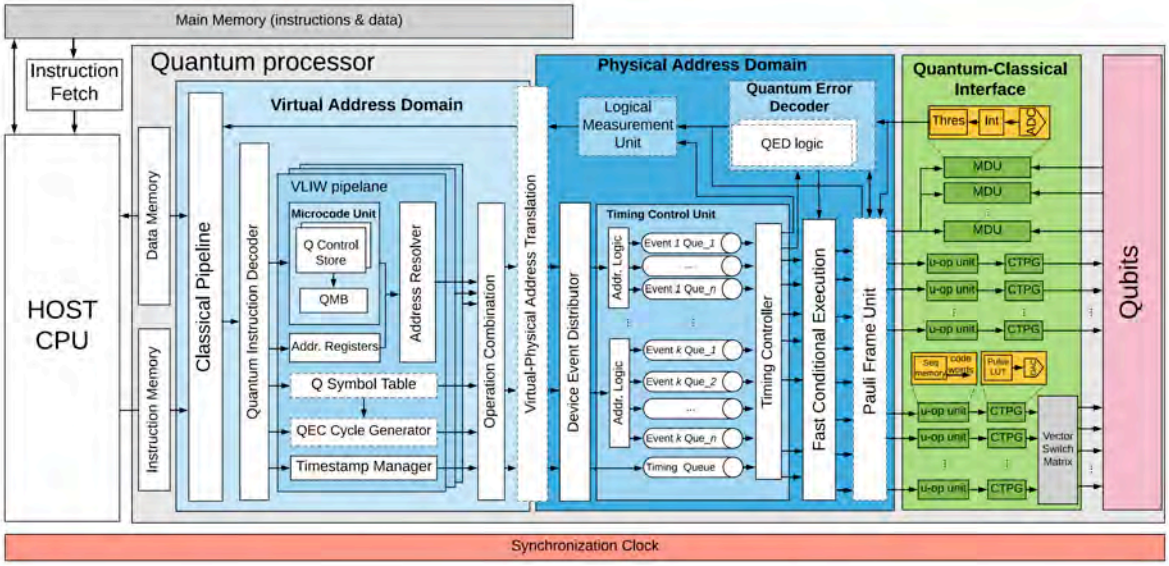


图 5.2: FT-QuMA 的容错量子控制微体系结构描绘了一种被称为 FT-QuMA 的容错量子控制微体系结构（Fault-Tolerant Quantum Control Microarchitecture）的概述。FT-QuMA 是对前一版本 QuMA_v2 的升级和改进，图中的虚线块突出显示了改变的部分。在量子计算中，微体系结构（Microarchitecture）负责实现量子处理器的实际操作。它的设计和实现对于量子计算机的性能至关重要。微体系结构需要处理多种挑战，包括噪声管理、定时控制、纠错编码等等。FT-QuMA 的设计目标是提供一种可以容忍错误的量子控制微体系结构。具体来说，它通过在微体系结构级别集成纠错功能，以增强量子计算的可靠性。这种纠错能力对于实现大规模、实用的量子计算至关重要，因为现实的量子系统总是受到各种噪声和错误的影响。图中的黑色细线表示数字信号，灰色粗线表示模拟信号。数字信号通常用于控制和同步微体系结构的各个部分，而模拟信号则直接用于操作量子位。设计和实现一个高性能的量子控制微体系结构需要精细地管理这两种类型的信号。然而，具体的 FT-QuMA 结构以及其如何实现容错能力，需要查阅文章的详细内容来理解。不同的量子计算系统可能会采用不同的微体系结构设计，以适应其特定的硬件和算法需求。

量子-经典接口的任务是在这两个不同的处理器之间建立通信。它必须能够将经典信号转换为可以对量子比特进行操作的量子信号，反之亦然。例如，它可能需要将经典计算机的指令转换为适当的激光脉冲，以对量子比特进行操作。另一方面，它也需要将从量子比特获取的信息转换为经典数据，以供经典计算机进一步处理。

在图5.2 中，Quantum-Classical Interface 可能包含一些模拟-数字转换（ADC）和数字-模拟转换（DAC）的设备，以及一些控制电路和信号处理电路。它可能也包含一些特定的硬件和软件，以实现特定的通信协议和错误处理策略。具体的设计和实现可能会根据具体的量子计算系统和应用需求而有所不同。

MDU

在量子计算微体系结构中，MDU 可能是”Measurement Demultiplexer Unit” 的缩写。

在量子计算过程中，我们需要对量子比特进行测量，然后将这些测量结果传输到经典计算机进行进一步处理。然而，因为量子比特的数量可能非常大，并且每个量子比特都可能需要独立地被测量，所以我们需要一种方法来有效地处理这些测量结果。

这就是 Measurement Demultiplexer Unit (MDU) 的作用。MDU 的任务是收集来自量子比特的测量结果，然后将它们分配到适当的经典处理器或内存单元。这样，经典计算机就可以独立地处理每个量子比特的测量结果，而不需要一次处理所有的结果。

具体来说，MDU 可能包括一些缓冲器、复用器和解复用器，以及一些控制逻辑和信号处理电路。这些部件的设计和实现可能会根据具体的量子计算系统和应用需求而有所不同。

Thres ← int ← ADD

”ADD” 通常表示加法操作，”int” 可能表示整数，”Thres” 可能表示阈值。箭头 ← 通常表示数据或控制流的方向，或者是赋值操作。因此，”Thres ← int ← ADD” 可能表示：执行某种加法操作，将结果（一个整数）赋值给”int”，然后将”int” 的值赋值给”Thres”。

在量子-经典界面的”Measurement Demultiplexer Unit” 的上下文中，这可能是一个对测量结果进行处理的操作，比如对测量结果的概率值进行加法运算，然后将结果用作阈值。

Thres

”Thres” 可能是”Threshold” 的缩写。在量子测量中，测量结果通常是概率性的，表示在测量时找到量子系统在某个特定状态的概率。阈值可能用于确定是否检测到了有效的信号或者事件。例如，如果测量结果的概率大于某个阈值，那么就认为检测到了有效的信号。

在”Measurement Demultiplexer Unit” 的上下文中，”Thres” 是用来决定是否将测量结果发送到不同的通道的阈值。这可能是一个方法，用于在量子-经典界面中管理和处理量子测量的结果。

u-op unit

在量子计算中，“u-op”可能是“unitary operation”的缩写，即酉操作或酉变换。酉操作是量子计算中的基本操作，它们在量子比特（qubits）上执行并改变其状态。酉操作有一些重要的性质，包括它们是可逆的。

ICTPG

5.1.3. 物理地址域 Physical Address Domain

在计算和特别是与体系结构相关的环境中，“物理地址域”通常指的是计算机系统内的实际物理内存位置。这与“虚拟地址域”形成对比，虚拟地址域是对物理内存的一个抽象层，允许更灵活和有效地使用内存资源。

在像 FT-QuMA 这样的量子控制微体系结构的环境中，” 物理地址域” 可能指的是系统内量子比特（qubits）的实际物理位置或标识符。在量子计算机的路由和控制操作中，这将是必要的，特别是在容错设计中，某些量子比特可能需要被换出或者以其他方式进行管理，以减轻错误的影响。

5.1.4. 虚拟地址域 Virtual Address Domain

在计算中，” 虚拟地址域” 通常是指通过操作系统或某种抽象层将物理内存映射到一个更大或更易于管理的地址空间的过程。这种抽象允许程序更有效地使用内存，同时还提供了一种保护不同程序之间不会相互干扰的方式。

在 FT-QuMA 这种量子控制微体系结构的中，” 虚拟地址域” 是指对量子比特（qubits）的抽象表示或标识。这可能涉及到对 qubits 进行编程和控制，同时提供一种方式来管理和减少量子计算中的错误。

有了这个介绍，让我们按字母顺序考虑领先的 QC 架构。请查阅本书在线网站，以获取其他论文和更新的链接。有关本章中概述的方法的更多背景信息，请参见 [53][126]。

5.2. 中性原子

5.2.1. 范式表达

中性原子量子计算机的计算过程可以用量子力学的数学框架来描述。每个中性原子比特可以用两个能量不同的自旋态（通常是一个基态和一个激发态）来表示。在实验中，通过适当的激光束和磁场可以将比特从一个自旋态转移到另一个自旋态。这些操作可以表示为旋转矩阵，它们是特

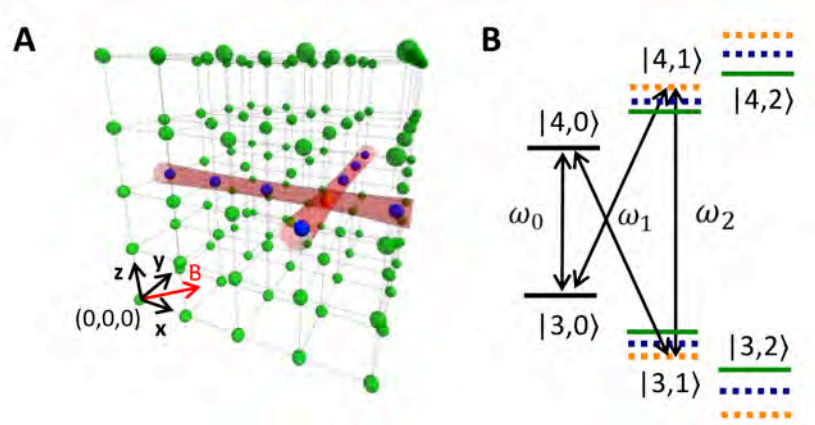


图 5.3: 如何在三维光晶格中精确地选址并操控中性原子子图 A 描述了如何在一个 5x5x5 的中性原子阵列中定向寻址每一个原子。每一个寻址光束可以在 5 微秒内沿任意原子线平行移动，因此可以选择任意一个位点作为交点，也就是说，可以实现对任意一个原子的选址。寻址光束是圆偏振的，140 mG 的磁场位于同一平面内。子图 B 描述的是关于选址过程中目标原子的地面状态能级结构。目标原子会经历两倍于其他任何原子的交流 Stark 位移（由橙色虚线表示），这意味着，在初始的储存基底 $|3,0\rangle$ 和 $|4,0\rangle$ 下，只有目标原子与频率为 $\omega 1$ 的光共振。当目标原子被转移到计算基底 $|3,1\rangle$ 和 $|4,1\rangle$ 后，只有它与频率为 $\omega 2$ 的光共振。这就使得我们可以精确地选定并操控目标原子，而不会影响到其他原子。总的来说，改图描述了一种高效且精确的寻址和操控中性原子阵列中单个原子的方法，这对于量子计算和量子信息处理具有重要意义。图片来源 [54]

殊的幺正矩阵。类似于其他量子计算机架构，量子门操作可以用矩阵乘法来实现。比如，我们可以用单比特旋转门（如 X、Y 和 Z 门）和双比特控制 NOT 门（CNOT 门）来实现任何量子门，从而进行量子计算。

量子计算的输出是一个量子态，其表示为一个由比特状态组成的向量。我们可以通过对比特进行测量来读取这个状态，每个比特测量的结果是一个二进制值（0 或 1）。通常情况下，我们对一系列比特进行测量，这些比特组成一个寄存器。每次测量的结果是一组二进制值，也就是经典计算机可以读取和处理的输出。

5.2.2. 计算过程

中性原子量子计算机的计算过程可以用量子力学的数学框架来描述。每个中性原子比特可以用两个能量不同的自旋态（通常是一个基态 $|0\rangle$ 和一个激发态 $|1\rangle$）来表示。在实验中，通过适当的激光束和磁场可以将比特从一个自旋态转移到另一个自旋态。这些操作可以表示为旋转矩阵，它们是特殊的幺正矩阵 U。类似于其他量子计算机架构，量子门操作可以用矩阵乘法来实现。比如，我们可以用单比特旋转门（如 Pauli X、Y 和 Z 门）和双比特控制 NOT 门（CNOT 门）来实现任何量子门 U，从而进行量子计算。

量子计算的输出是一个量子态 $|\psi\rangle$，其表示为一个由比特状态组成的向量：

$$|\psi\rangle = \sum_{i=0}^{2^n-1} c_i |i\rangle \tag{5.1}$$

其中 n 是比特数，i 是一个 n 位二进制数，c_i 是一个复数系数，表示状态 $|i\rangle$ 的幅度。当我们对这个态进行测量时，每个比特的结果是一个二进制值（0 或 1）。通常情况下，我们对一系列比特进行测量，这些比特组成一个寄存器。每次测量的结果是一组二进制值，也就是经典计算机可以读取和处理的输出。

5.2.3. 现状

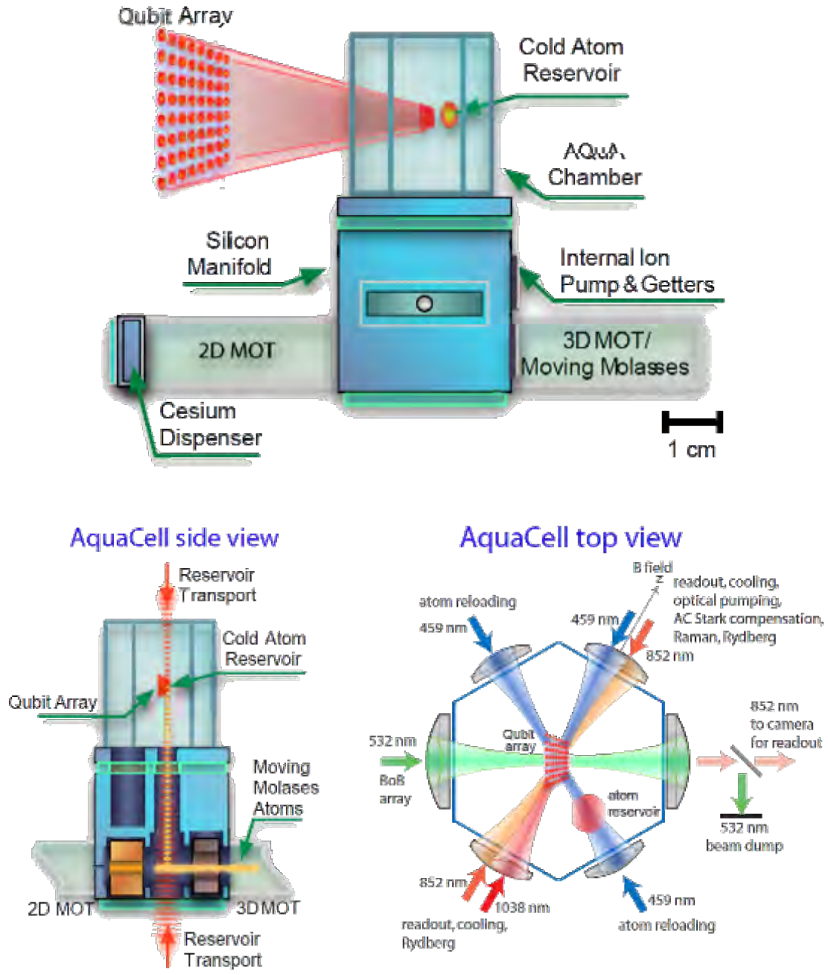


图 5.4: 中性原子系统的建立 Anderson 小组正在创建一个二维的铯 (Cs) 原子阵列。每个铯原子在阵列中的理论最大寿命是 100 秒。这个寿命足够长，以至于我们可以连续监视原子的损失，并根据需要在阵列的各个位置重新装载原子。我们的项目旨在从一个包含数百万铯原子的磁光学陷阱 (MOT) 中取出一个铯原子，并将其运输到阵列的一个空位点。因此，一个铯原子的恒定循环将始终确保阵列有充足的 Rydberg 铯原子供应。这个铯原子阵列将作为进行量子算法所必需的量子比特。图片来源科罗拉多大学博尔德分校（University of Colorado Boulder）的 JILA 研究所的网站。JILA 是一个跨学科的研究机构，聚焦于物理学、化学和生物物理学领域的前沿研究。图片来源：`https://jila.colorado.edu/dzanderson/research/neutral-atom-quantum-computing`

中性原子是一种有趣的量子计算方法。虽然束缚离子研究已经进行了一段时间，但是最近一些实验室已经加强了对中性原子集合的控制能力。

为了实现中性原子系统，工程师可以在原子集合周围设置四束激光束，形成一个磁光陷阱（MOT）。实验室通常使用铯（Cs）或铷（Rb）原子进行这项工作。通过使用这个四重激光系统限制原子，我们可以将原子冷却到毫开尔文温度。现在我们在一个储备池中拥有数亿个中性原子，我们可以将其中少量的中性原子转移到一个可寻址的阵列中（见图 5.2）。哈佛大学的 Lukin 实验室在中性原子系统方面取得了良好的进展 [55, 56]。David Weiss 和他在宾州州立大学的团队也专注于中性原子平台，并展示了一个 Stern-Gerlach 灵感的中性原子实验 [54, 57]。中性原子系统符合 DiVincenzo 的量子计算机标准：量子比特是可分离和可寻址的，它们可以保持它们的状态并且我们可以对它们进行测量。Saffman 等人对在中性原子系统上实现门提供了良好的综述 [58]。现在的挑战是扩展这样的系统。参见 [59] 以获取有用的综述。

哈佛大学的 Lukin 实验室

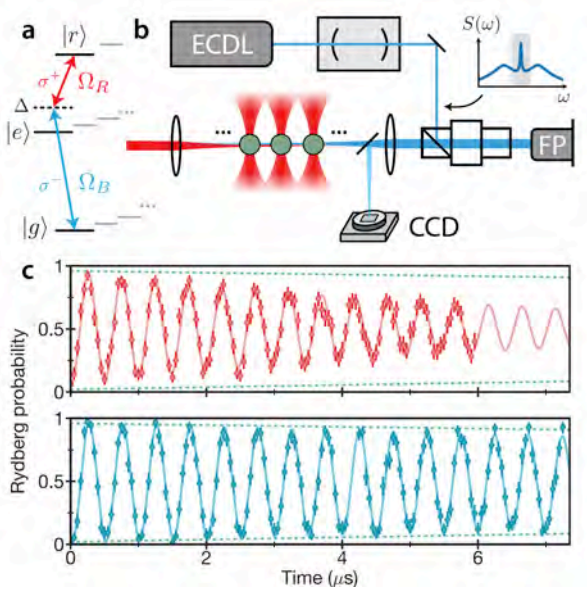


图 5.5: 哈佛大学单原子拉比振荡哈佛大学的 Lukin 实验室在中性原子系统方面取得了的进展。图片来源：[56]

哈佛大学的 Lukin 实验室在中性原子系统方面是一项实验设置和单原子拉比振荡[1]的研究。以下是对图5.5这段文献的概述：

1. 图 5.5描述了实验设置和单原子拉比振荡。子图 (a) 展示了基态 $|g\rangle = |5S_{1/2}, F = 2, m_F = -2>$ 与 $|r\rangle = |70S, J = 1/2, m_J = -1/2>$ 之间通过中间态 $|e\rangle = |6P_{\frac{3}{2}}, F = 3, m_F = -3>$ 进行耦合。
2. 子图 (b) 展示了激光与参考腔的锁定方式。参考腔的窄传输窗口（插图中的阴影区域）抑制了高频相位噪声。传输的光用于注入锁定法布里-珀罗（FabryPerot）激光二极管。激光二极管的输出焦点对准了光学镊子中的原子阵列，小型取样器对准参考 CCD 相机用于对齐。
3. 子图 (c) 展示了共振的双光子耦合引起的 $|g_i|$ 和 $|r_i|$ 之间的拉比振荡。上图是从以前在 [60] 中使用的设置中得到的典型测量结果。下图显示了新设置的典型结果，拟合的相干时间为 27(4) 微秒。每个数据点是根据 50-100 次重复测量两个相同耦合的原子得出的，这两个原子相隔 23 微米，因此它们实际上是非相互作用的。

在所有的图中，误差条标记了 68% 的置信区间，实线是对实验数据的拟合，虚线表示了从数值模型预期的对比度。

所以，图5.5是对一项量子物理实验的详细解释，该实验涉及到使用高级技术精确地控制和研究原子态。在这项实验中，研究人员能够观察和测量原子的特性，例如拉比振荡，这是量子力学中的一个重要概念，描述的是一个二能级系统在外部激励下跃迁的现象。

[1]拉比振荡（Rabi oscillations）是量子力学中的一个重要概念，由物理学家伊萨多尔·拉比（Isidor Isaac Rabi）首次描述。

在两级量子系统中（如一个原子的两个能级，一个量子比特的两个状态），如果这两个状态之间的能量差与外部激励（例如电磁场）的频率匹配，该系统可能会在两个状态之间振荡。这种现象就被称为拉比振荡。

例如，考虑一个原子有两个能级，低能级和高能级。如果我们用一个电磁场来激励这个原子，电磁场的频率与两个能级之间的能量差相匹配，那么原子就会从低能级跳跃到高能级。然后，如果我们继续施加电磁场，原子会掉回到低能级。这个上下跳跃的过程会反复发生，形成振荡。这种振荡就是拉比振荡。

拉比振荡在量子信息处理、量子计算和量子控制中都有重要的应用。例如，在量子计算中，通过精确控制拉比振荡的时间，可以实现对量子比特的精确操作，如量子门的实现。

宾州州立大学

宾州州立大学关于中心原子的研究论文为"Stern–Gerlach detection of neutral-atom qubits in a state-dependent optical lattice"，由 Tsung-Yao Wu, Aishwarya Kumar, Felipe Giraldo, 和 David S Weiss 撰写，发表在 2019 年的《Nature Physics》杂志上 [57]。

文章的主要内容是关于一种新的检测中性原子量子比特（qubits）的方法，这种方法基于 Stern-Gerlach 效应[2]和状态依赖的光晶格。Stern-Gerlach 实验是量子力学中的一个基本实验，它展示了原子的自旋量子化。在这篇文章中，作者用类似的方法来检测中性原子量子比特的状态。

文章中描述了一个实验设置，其中使用了光晶格来捕获和操纵原子。光晶格是由交叉的激光束形成的周期性势能结构，可以用来捕获和操纵原子。在这个实验中，光晶格的形状可以根据原子的内部状态进行调整，这使得原子的状态可以通过他们在光晶格中的位置来检测。

实验的结果显示，这种新的检测方法能够以高精度和无损的方式测量中性原子量子比特的状态。这种无损的检测方法对于量子信息处理和量子计算具有重要的应用潜力，因为它们需要对量子系统进行精确和高效的控制和测量。

总的来说，这篇文章报告了一种新的量子比特检测方法，这种方法结合了 Stern-Gerlach 效应和状态依赖的光晶格，为实现高精度无损的量子比特测量提供了一种可能的途径。

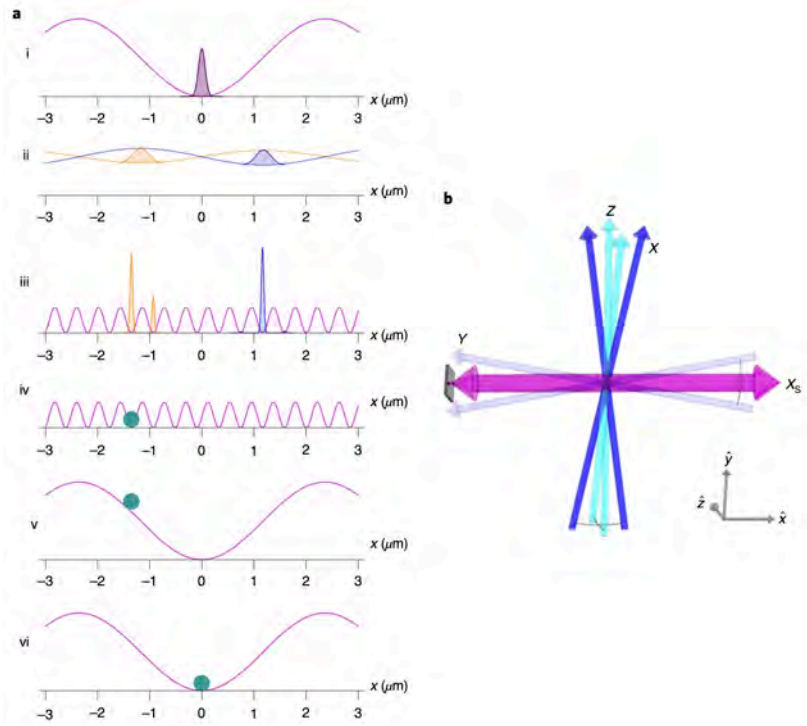


图 5.6: 宾州州立大学关于斯特恩-格拉克在随状态变化的光学晶格中对中性原子量子比特的检测。图片来源：[57]

[2]Stern-Gerlach 效应，是由德国物理学家奥托·斯特恩（Otto Stern）和瓦尔特·格尔拉赫（Walter Gerlach）在 1922 年首次观察到的一个物理现象。这个实验是一个里程碑式的实验，因为它是首次明确展示了原子的自旋量子化特性，从而对量子力学的发展产生了深远影响。

在 Stern-Gerlach 实验中，一束银原子被通过一个非均匀磁场。由于银原子的自旋，他们在磁场中的行为会受到自旋状态的影响。如果银原子的自旋是连续的，那么我们期望原子束在通过磁场后在屏幕上形成连续的分布。然而，实际观测到的结果是，银原子束被分裂成两个明确的点，这表明银原子的自旋只有两个可能的取值，即自旋向上和自旋向下。

这个结果是量子力学的一个关键证据，因为它表明粒子的自旋（以及其他量子性质）不是连续的，而是量子化的。这是一个从经典物理学到量子物理学的重大转变，因为在经典物理学中，这样的性质通常被认为是连续的。自此以后，Stern-Gerlach 实验被用作教授量子力学基本原理的一个重要工具，特别是关于粒子的自旋和量子化的概念。

宾州州立大学关于一种无损态检测的概述，这种检测方法用于中性原子量子比特（qubits）。这一方法的主要步骤和相关概念在图5.6中进行了详细的解释和描绘。

子图 (a) 描述了状态检测的步骤：

- **预测量状态**。紫色曲线代表在 x 方向（X 晶格）上的状态无关势能。紫色阴影区域代表原子在振动基态和两个状态的等概率叠加态的波函数；
- **状态依赖晶格的位移**。势能已经绝热地转变为两个较浅的状态依赖势能，其中每个状态的势能和波函数分别用橙色和蓝色表示；
- **将原子转移到 XS**。对于每个内部状态，具有显著波函数振幅的 XS 晶格位点的数量取决于 XS 和状态依赖的 X 势能最小值的相对位置，这些在我们的实验中并非固定的；
- **在 XS 中成像原子**。波函数已经被投影到一个单一的晶格位点上。现在，原子在许多内部状态和几个振动能级之间分布，由青色圆圈表示。其位置用于状态分配；
- 将原子转回到 X；
- 原子在一个四分之一振荡周期后在 X 的底部。在这一点，再次拍摄一个图像。

子图 (b) 是晶格光束的示意图。X，Y 和 Z 晶格由一对线性偏振激光束组成，它们以 10° 的角度交叉（在图中绘制为 20°，但否则按比例），给出 4.8 μm 的晶格间距。这些光束在原子处聚焦，高斯腰部为 75 μm。XS 晶格（带双箭头的紫色）是通过反射形成的，产生 0.42 μm 的晶格间距。XS 光束在原子处的高斯腰部大约为 150 μm。

所以，该段描述是关于一个精细的实验设置，该设置用于无损地检测中性原子量子比特的状态。在这个实验中，研究人员使用复杂的激光束阵列和原子陷阱，以及精细的操控和测量技术，以便精确地控制和测量原子的量子状态。

5.2.4. NMR

核磁共振（Nuclear Magnetic Resonance，NMR）是一种物理现象，它涉及到核自旋在磁场中的行为。在这个背景下，一个核自旋可以被看作是一个量子比特，或简称为量子位（qubit）。

在量子计算中，一个量子位是信息的基本单位，类似于经典计算中的位（bit）。然而，与经典的位不同，一个量子位不仅可以处于 0 和 1 的状态，还可以处于这两种状态的叠加，这就允许我们在一个量子位上同时处理多个可能的解。

在 NMR 设备中，可以使用核自旋作为量子位。具体来说，一个核自旋可以处于两种状态，通常被标记为“上”和“下”，或者“+1/2”和“-1/2”，这对应于量子位的 0 和 1。通过操作这些核自旋，我们可以在它们上执行量子计算。

为了操作核自旋，我们可以使用磁场和无线电频率（RF）脉冲。通过改变磁场或施加 RF 脉冲，我们可以改变核自旋的状态，或者使两个核自旋相互作用，这就允许我们在 NMR 设备中执行量子门，从而进行量子计算。

尽管 NMR 设备可以用来执行量子计算，但它们在实践中面临一些重要的限制。例如，NMR 信号是从大量的分子样品中平均得到的，这使得我们无法访问或控制单个量子系统的状态。此外，NMR 量子计算的可扩展性也受到限制，因为随着量子位数量的增加，必要的磁场和 RF 脉冲的复杂性也会显著增加。

5.3. NV Center-in-Diamond

NV Center-in-Diamond” 是指金刚石中的氮空位中心（Nitrogen-Vacancy Center）。

在固体物理学中，氮空位中心是金刚石晶格中的一种特殊缺陷，由一个氮原子（Nitrogen）和一个相邻的空位（Vacancy）组成。这种缺陷在量子物理学中具有独特的性质，包括在室温下保持量子相干性，以及对光和微波的响应，使其成为一种潜在的量子比特（qubit）。

氮空位中心具有强大的潜力，可以用作单光子源、磁力计和电子自旋系统，这使其在量子信息处理、纳米尺度传感以及生物成像等领域有广泛的应用前景。在量子计算机领域，氮空位中心被视为一种可能的实现量子比特的物理系统。

然而，尽管氮空位中心具有很多吸引人的性质，但要将其用于实际的量子计算还面临许多技术挑战，包括提高量子相干时间、实现大规模集成以及实现高效的量子操作等。

在针对量子计算的金刚石中的氮空位（NV）中心方法中，金刚石晶格中有两个碳原子缺失，其中一个被氮离子取代。由此形成的系统构成了一种顺磁性缺陷，可以作为量子比特。量子比特状态可以通过微波场进行操控，并可光学读出。许多实验室已经展示了基础的 NV 系统 [61, **?**, 62]。

5.3.1. 进展

莱比锡大学

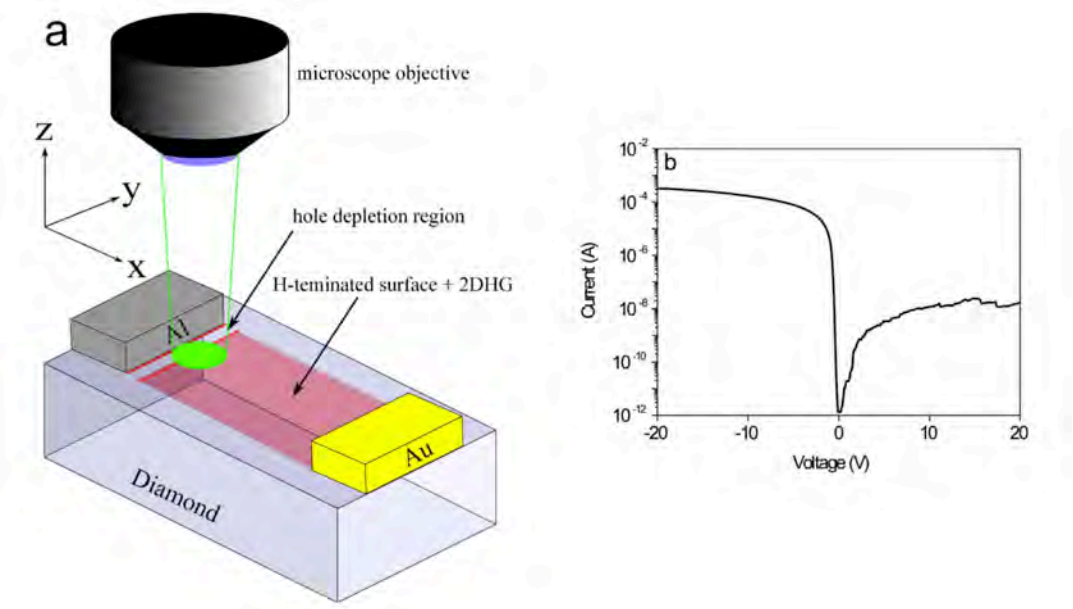


图 5.7: 莱比锡大学关于在氢化钻石表面的铝 Schottky 二极管的结构以及其电流-电压特性的研究。图片来源：[61]

图5.7描述了一个在氢化钻石表面的铝 Schottky 二极管的结构以及其电流-电压特性。

在图5.7 子图 (a) 中，作者展示了这个二极管的示意图。在这个图中，一个共焦微光致发光（μPL）测量在铝接触的边缘（绿点）进行，以研究在耗尽区内 NV 中心发射的偏压电压依赖性变化。NV 中心是钻石晶格中的一种缺陷，它具有特殊的光学和磁学性质，使得它在量子信息科学和生物成像中有很大的应用潜力。

在图5.7 子图 (b) 中，作者展示了这个铝 Schottky 二极管的电流-电压特性。这个测量在室温条件下（T = 300 K）进行。这个图表显示了二极管的电流随着施加的电压的变化，这是一个重要的特性，可以用来描述和理解这个器件的电学行为。

总的来说，这个图展示了一个新的设备和一种新的方法来控制钻石中的单个 NV 中心的电荷状态，这对于利用这些中心进行量子信息处理和生物成像有重要的应用潜力。

除了掺杂氮之外，一些实验室已经尝试用硅来掺杂 [110, 76]。这些以及其他材料各自具有其独特的优点和缺点 [53, 122]。对于 NV 方法的有用回顾，请参见 [55, 71]。

研究者目前正在探究在这个平台上成功应用两量子比特操作符的可能性（参见 [135, 28, 171, 109]）。

5.3.2. Photonics

光子学是研究光（即光子）和其应用的科学领域。它的应用非常广泛，包括光纤通信、激光技术、光谱学等等。近年来，光子学也被应用到量子计算中，利用光子来实现量子比特（或称为量子位，qubit）和量子门。

光子具有许多属性，例如偏振状态、轨道角动量，以及其在空间中的位置等等，这些属性都可以用来定义一个量子位。例如，一个光子的两种偏振状态（比如水平偏振和垂直偏振）可以用来定义一个量子位的 0 和 1 状态。

在基于光子的量子计算系统中，量子门是通过对光子的操作来实现的。这些操作可以是线性的（例如偏振转换器、波片等）或非线性的（例如通过 Kerr 效应等）。线性的操作通常比较容易实现，但是要实现一个通用的量子计算系统，我们还需要至少一种非线性的操作。

最常见的基于光子的量子计算模型是线性光学量子计算（Linear Optical Quantum Computing，LOQC）。在 LOQC 中，量子门是通过线性光学元件（例如分束器和相位转换器）和单光子检测器来实现的。尽管这个模型在理论上已经被证明是通用的，但在实践中实现它仍然面临一些挑战，例如需要高效的单光子源和单光子检测器，以及需要处理光子的损失和噪声等问题。

总的来说，基于光子的量子计算是一个非常活跃的研究领域，它提供了一种可能的路径来实现大规模的量子计算。

光子学也可以用于构建基于门的量子计算。线性光学量子计算（LOQC）使用线性光学元素（例如镜子，分束器和相位移器）来处理量子信息 [63, 64, 65][7，120，123]。这些光学元素保持了输入光的相干性，因此等效地在有限数量的量子位上应用一个酉变换。然而，在真空中，光子不会相互作用，它们只能通过另一个介质间接地相互作用。

在 2001 年，Knill、Laflamme 和 Milburn（KLM）[64][120] 提出了一种使用单光子，线性光学，光探测和后处理实现可扩展量子计算的方法。这些元素使得仅使用线性光学元素就能应用非线性操作 [65][123]。另一种范例是所谓的单向，基于测量或群集态量子计算机（MBQC）[66, 67, 68, 69][179、180、181、182]，它首先准备高度纠缠的多粒子群集态。然后，为了实现量子电路，必须执行一系列单量子位投影测量。每次后续的测量选择都由前一次测量的结果驱动。

任意两量子位处理在量子计算电路模型中需要三个连续的纠缠门的等效操作 [70][101]，这超出了用自由空间量子光学实际构造和维护的复杂度水平 [71, 72][143，52]。光子芯片可以利用整个基于硅的基础设施来微型化 LOQC 并降低成本 [37][201]。布里斯托大学的研究人员与中国国防科技大学的研究人员一起，在硅光子芯片上展示了一个光子量子处理器。该处理器生成两个光子量子位，在其中执行任意的两量子位酉变换，包括任意纠缠操作 [73][178]。这个量子处理器是使用成熟的互补金属氧化物半导体（CMOS）兼容处理制造的，包括 200 多个光子组件；该处理器被编程以实现 98 种不同的两量子位酉变换，平均量子过程保真度为 93.2 ± 4.5%。

半导体量子晶体管

确定性光子 - 光子相互作用的缺乏是这种方法中量子计算的一个挑战。要用单个光子确定性地控制光信号需要与量子存储器进行强相互作用。与量子发射体耦合的纳米光子学结构可能提供

一种有吸引力的方法，在紧凑的固态设备中实现单光子非线性特性。最近，对于使用固态量子存储器实现的第一个单光子开关和晶体管的实现，研究人员在马里兰大学和联合量子研究所使用半导体芯片 [74][211]。该装置允许一个光子切换其他光子，因此产生强大且可控的光子 - 光子相互作用。它由一个与纳米光学腔强耦合的自旋量子位组成（这是由 Duan 和 Kimble 首先提出的想法 [75][72]）。他们将半导体薄膜与量子点相结合；它们一起位于阵列的中心。阵列形成一个光子晶体，使用布拉格反射机制，其中光在陷阱中来回反弹。这使得量子点能够以单个电子的自旋性质存储有关光子的信息。通过使用这个晶体管，他们应该能够将量子门应用于光子。适用于量子信息处理的可扩展设备将需要更高的效率，因为光子损失构成了光子量子位的主要误差源。

拓扑光子芯片

拓扑光子芯片是一种采用拓扑光子学原理设计和制造的光子集成芯片。拓扑光子学是一门研究光子与特定拓扑结构相互作用的学科。光子是光的粒子，拓扑学是数学的一个分支，研究空间结构的性质。拓扑光子芯片在光子学、材料科学和拓扑学等领域的交叉研究中取得了许多创新性的成果。

拓扑光子芯片具有以下几个显著特点：

- **带隙**：在特定的光子能带范围内，光子无法通过拓扑光子结构。这种现象类似于半导体材料中的电子能带隙，可以用于实现光的筛选、调制和控制。
- **边缘态**：拓扑光子结构可以产生一种称为边缘态的光子模式。边缘态具有单向传播的特点，不受结构缺陷的影响，因此可以实现低损耗、高稳定性的光波导传输。
- **鲁棒性**：由于边缘态的存在，拓扑光子芯片对结构缺陷和散射损耗具有很高的鲁棒性，可以在复杂环境中实现高性能的光子器件和系统。

拓扑光子芯片在通信、计算、传感和量子信息等领域具有广泛的应用前景。例如，拓扑光子芯片可以用于实现光纤通信中的波长选择开关、光子集成电路中的波导和耦合器等关键器件。此外，拓扑光子芯片还可以用于开发新型量子信息处理设备，例如拓扑量子比特和拓扑量子门。

拓扑绝缘体是一种异乎寻常的材料，其在其体内绝缘但在其表面导电 [76, 77][105，177]。这些状态表现出非常出色的性质，如单向传播和噪声的稳健性。自这些物态的发现以来，使用集成光学 [78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 38, 93, 94, 95][138，167，240，229，99，183，100，32，153，239，54，162，134，125，221，220，244，33，154] 已经观察到了几种拓扑效应。

拓扑光子学是可扩展量子计算的一个有前途的选项，因为它们不需要强磁场，具有固有的高相干性，室温操作和易于操作的优点。最近，来自澳大利亚皇家墨尔本理工大学的科学家与米兰理工大学和苏黎世联邦理工学院的研究人员合作，开发了一个编码，处理和传输量子信息的拓扑光子芯片 [96][214]。他们使用光子芯片证明了拓扑状态可以通过复制已知的 Hong-Ou-Mandel（HOM）实验进行量子干涉，可见度为 93.1 ± 2.8％[97][112]。

5.3.3. Spin Qubits

Spin Qubits（自旋量子比特）是一种基于电子自旋的量子比特表示方法。量子比特（qubit）是量子计算的基本单元，与经典计算中的比特（bit）类似。然而，量子比特具有超越经典比特的性质，如叠加态和纠缠态等。

自旋是一种内在的量子力学特性，与电子的磁性密切相关。电子的自旋可以有两个方向，通常表示为“上”和“下”，分别对应自旋量子数为 +1/2 和-1/2。在 Spin Qubits 中，我们可以将这两个自

旋状态作为量子比特的基本状态，即 |0> 和 |1>。通过操纵电子自旋，我们可以实现量子比特的初始化、操作和读取等功能。

Spin Qubits 具有一些优势，如较长的相干时间、较小的空间尺寸和与现有半导体技术的兼容性等。这使得 Spin Qubits 在固态量子计算领域具有广泛的应用前景。实际上，许多实验室已经成功地在硅、氮化镓等半导体材料中实现了 Spin Qubits 的操作和控制。

然而，Spin Qubits 也面临着一些挑战，例如实现高保真度的量子操作、提高纠缠态生成速率和构建大规模集成的量子处理器等。为了克服这些挑战，研究人员正在不断探索新的物理机制、材料和技术。

基于硅的自旋量子位技术代表了量子计算的另一种方法。如果我们可以使用普通半导体材料构建量子位，那么我们就可以通过利用集成芯片行业几十年的专业知识来扩展这样的系统。然而，在标准 CMOS 平台上开发稳定且可寻址的量子位非常困难。虽然已经展示了初步的原型，但很难将这些量子位稳定地扩展。在这些早期的尝试中，一对量子点被放置在硅半导体衬底上的传统 CMOS 设置中的源和汇之间。整个装置被放置在冷却冰箱中，将其降至约 1K 的温度；这比超导量子位所需的温度要高得多。然后使用微波脉冲将单量子位操作符应用于量子位 [98][230]。英特尔公司，HRL 实验室以及剑桥大学，代尔夫特理工大学，哈佛大学和新南威尔士大学（UNSW）的实验室正在开发基于硅的自旋量子位方法。

5.3.4. Superconducting Qubits

超导量子比特（Superconducting Qubits）是一种基于超导电路的量子比特表示方法。在量子计算中，量子比特（qubit）是基本的信息处理单元，类似于经典计算中的比特（bit），但具有量子特性，如叠加态和纠缠态。

超导电路是由超导材料制成的电路，在低于特定临界温度时，超导材料能够以零电阻状态传导电流。超导量子比特利用超导电路中的约瑟夫森结（Josephson junction）制作而成，约瑟夫森结在超导电路中充当非线性元件的角色，使得超导量子比特的能级间距具有非等间距特性。

超导量子比特有多种实现方式，常见的有：

- **Charge Qubits（电荷量子比特）**：基于超导电路中的电荷态，用量子线路中的电荷数表示量子比特状态。
- **Flux Qubits（磁通量子比特）**：基于超导电路中的磁通量态，利用量子线路中的磁通量表示量子比特状态。
- **Transmon Qubits（跨越单子量子比特）**：一种改进的电荷量子比特，通过增大约瑟夫森结电容，使电荷噪声对量子比特的影响降低。

超导量子比特具有一定的优势，如较快的操作速度、较高的保真度和与微波电路技术的兼容性等。这使得超导量子比特成为当前量子计算领域的主要研究方向之一。例如，IBM、谷歌和 Rigetti 等公司和研究机构都在开发基于超导量子比特的量子计算机。

然而，超导量子比特也面临着一些挑战，例如提高量子比特的相干时间、减少量子操作错误率和实现大规模量子计算机的集成等。为了克服这些挑战，研究人员正在不断探索新的物理机制、超导材料和量子控制技术。

一些团队正在使用超导量子位构建量子计算机。其核心设计基于由 Josephson 结构组成的 Cooper 对量子位 [99][42]。微波引线连接到量子位上以控制它。通过向物理量子位发送特定频率

的微波脉冲以控制其时间，用户可以应用一系列的单量子位操作符。整个装置必须冷却至 10mK 以下才能操作。该系统也被屏蔽以防止磁场和其他可能导致失相的因素干扰。

NAS 报告总结了各种类型的超导量子位 [100][159，C-1 页]：固定频率与可调谐量子位：可调谐量子位可以进行校准和纠正，以应对由制造过程中的变化或设备老化导致的量子位频率变化。一个优点是一个微波信号可以控制多个量子位，这是硬件上的节省。获得这个优势需要一个额外的控制信号来调整频率，并为噪声进入量子位增加了一个额外的路径。目前用于数字超导量子计算的两种最常见的量子位是“跃迁态量子位”，它分为单结构不可调谐和双结构可调谐两种形式，以及“通量量子位”... 两种跃迁态设计都被用于领先的工作中。

静态与可调谐耦合：量子位之间的静态耦合-例如，通过使用电容器或电感器来介导相互作用-是一种固定的设计“一直开启”的耦合。通过使两个量子位共振来打开耦合，并通过调整量子位来关闭耦合。然而，即使在关闭状态下，仍存在一些残余耦合。可以通过在两个量子位之间添加第三个物体（另一个耦合量子位或谐振器）来进一步减小调谐。

许多团队正在开发超导量子位计算机，包括：Google，IBM，Rigetti，QCI 等（见图 5.8）[101, 102][165，190]。请查看本书的 GitHub 网站以获取该领域的更新信息。Krantz 等人对这种方法提供了有用的综述 [103][124]。

5.3.5. Topological Quantum Computation

拓扑量子计算（Topological Quantum Computation）是一种基于拓扑量子系统的量子计算方法。拓扑量子计算的核心思想是利用几何拓扑学的概念和方法，通过操纵量子系统的拓扑性质来实现量子计算。由于拓扑性质具有一定的稳定性和鲁棒性，拓扑量子计算具有潜在的抗噪声和低错误率的优势。

拓扑量子计算的关键组成部分是拓扑量子比特（Topological Qubits），它们是基于任意子（Anyons）的量子态。任意子是一种存在于特定拓扑量子系统中的准粒子，它们的交换统计行为介于费米子和玻色子之间。通过对任意子进行特定顺序的交换操作，可以实现拓扑量子比特之间的纠缠和量子门操作。

拓扑量子计算的优势在于它具有天然的容错性。由于任意子交换操作仅依赖于其拓扑性质而非局部细节，这使得拓扑量子计算对局部扰动和噪声具有很高的鲁棒性。理论上，拓扑量子计算可以实现低错误率的量子操作，从而降低量子错误纠正的需求。

然而，拓扑量子计算也面临着一些挑战。首先，实验上实现拓扑量子比特和任意子的操作依然十分困难，尚未完全实现。此外，拓扑量子计算目前仅限于特定类型的拓扑系统和任意子，这可能限制其在量子计算中的应用范围。尽管如此，拓扑量子计算在量子计算领域仍具有广泛的研究价值和潜在应用前景。

除了我们上面提到的平台，还有许多其他构建量子计算设备的尝试。拓扑方法利用了任意子（anyon）的独特属性。任意子是一种既不是玻色子（如光子）也不是费米子（如电子）的二维准粒子。通过编织 4D 时空中任意子的路径，理论上可以创建一个量子计算系统，该系统对于减弱效应是强健的（见图 5.9）。这是由于系统的编织性质——即使对于系统有一系列微小的扰动，系统的拓扑不会改变，因此它保持相干，量子计算可以继续进行。有关该方法的更多背景，请参见 Roy 和 DiVincenzo 的文章 [104][192]。

理论物理学家 Alexei Kitaev 首次提出了拓扑量子计算的想法 [105][119]。Freedman，Kitaev 等人随后证明了这样一个拓扑计算机是图灵完备的 [106][90]。然后 Freedman 和其他人在 Microsoft 启动了一个项目，以研究实现拓扑量子计算机的物理实现。有关更有用的调查，请参见 [107][127]。

5.3.6. Trapped Ion

Trapped Ion（离子阱）指的是一种技术，通过产生适当的静电或磁场来捕获并操控单个或多个带电粒子（离子）。离子阱技术在许多领域都有重要应用，如质谱分析、原子钟、基本物理研究以及量子计算等。

在量子计算领域，离子阱技术可以用于构建离子阱量子比特（Trapped Ion Qubits），这是一种基于离子内部态（如能级、自旋等）的量子比特表示方法。通过精确地操纵激光或微波场，可以实现对离子量子态的初始化、操作和测量等功能。

离子阱量子比特具有以下优势：

- **较长的相干时间**：离子阱内的离子相对隔离，能够抵抗来自外部环境的噪声和干扰，因此具有较长的相干时间，有利于实现高保真度的量子操作。
- **高度纠缠**：离子阱量子比特间的相互作用可以通过离子的共振振动模式实现，从而方便地生成多离子纠缠态，为量子计算提供基础。
- **精确操控**：离子阱量子比特可以通过激光或微波场进行精确操控，实现高精度的量子操作。

然而，离子阱量子比特也面临一些挑战，如提高离子捕获和操控的速度、实现大规模集成的离子阱量子处理器以及降低操作中的加热和噪声等。为了克服这些挑战，研究人员正在探索新的离子阱技术、物理机制和量子控制策略。

离子阱量子计算已成为量子计算领域的重要研究方向之一，许多实验室和公司（如 Honeywell、IonQ 等）正在开发基于离子阱技术的量子计算机。

在被困离子方法中，用激光将钇原子（或其他元素）离子化并困在电势中形成一个量子比特的线。然后使用额外的激光来测量量子比特的状态。支持被困离子系统的人指出，他们的系统能够在不需要冷却到 mK 范围的情况下运行，而其他方法则需要这样的冷却。Cirac 和 Zoller 在这个领域做了早期的工作 [35][59]。马里兰大学学院公园分校的 Chris Monroe 和杜克大学的 Jungsang Kim 一直在这个领域活跃，并报道了一些进展 [12，21]。其他团队包括 NIST[108][35]、牛津 [109][139]、因斯布鲁克 [110][92]、MIT[111][131] 和 ETH[112][86] 等。

图 5.10 详细说明了被困离子系统中可以实现的两种量子比特类型：光学量子比特和超精细量子比特。光学量子比特利用了基态亚稳态之间能量水平的差异；超精细量子比特区分两个不同的基态。有关被困离子方法的有用综述，请参见 [113][50] 和 [100][159]。

6

量子机编程开发库

量子计算机编程的开发库是指一系列用于设计、模拟和实现量子算法的软件工具。这些开发库使得程序员能够编写量子计算程序并在量子计算机硬件或经典计算机上进行仿真。以下是一些常用的量子计算开发库：

- **Qiskit（Quantum Information Science Kit）**：由 IBM 开发的一套开源量子计算开发库，使用 Python 编程语言。Qiskit 允许用户设计量子电路、运行量子算法并将其与 IBM 的云端量子计算机相连接。
- **Cirq**：由谷歌量子计算团队开发的一套开源量子计算开发库，同样使用 Python 编程语言。Cirq 旨在支持量子电路设计、模拟以及对谷歌量子处理器的操作。
- **Microsoft Quantum Development Kit**：由微软开发的量子计算开发库，使用 Q 编程语言。Q 是一种专门为量子计算设计的编程语言，旨在简化量子算法的开发和测试。微软量子开发工具包还提供了用于量子电路模拟的经典库。
- **ProjectQ**：一个开源量子计算开发库，使用 Python 编程语言。ProjectQ 提供了一套用于设计、模拟和优化量子电路的工具，支持多种量子计算机硬件平台。
- **QuTiP（Quantum Toolbox in Python）**：一个开源量子计算和量子信息处理的 Python 库，提供了丰富的量子系统模拟和分析工具。
- **Strawberry Fields**：由 Xanadu 开发的一套用于光子量子计算的开源软件库，使用 Python 编程语言。Strawberry Fields 专注于连续变量量子计算，并提供了用于光子量子电路设计和模拟的工具。

以上开发库为量子计算领域提供了丰富的软件支持，使得研究人员和开发者能够更方便地设计、实现和优化量子算法，推动量子计算技术的发展。

随着对量子计算的兴趣不断增长，该领域的开发库和工具数量也越来越多。所有主要的编程语言都有相应的开发环境和模拟器，包括 Python、C/C++、Java 等。

许多领先的量子计算研究中心已将 Python 作为构建量子电路的首选语言。选择 Python 的原因之一是它是一种灵活的高级语言，允许程序员专注于解决问题而无需过多关注形式细节。例如，

Python 是动态类型的（意味着变量类型不必由程序员声明），并且是一种解释性语言（意味着它不必预编译为二进制可执行文件）。出于这些原因和其他原因，Python 对于新用户具有相对容易的学习曲线，并已经得到了量子计算社区的强烈支持。

在本章中，我们将概述这些库的工作原理，并为每个框架提供代码示例。在接下来的章节中，我们将通过使用这些库的算法实现示例进行更详细的说明。这些量子开发库都具有我们在本书中涵盖的所有主要一元和二元运算符的方法。一些库提供了三元运算符的内置模块，当然如果库中没有提供，也可以自行构造。

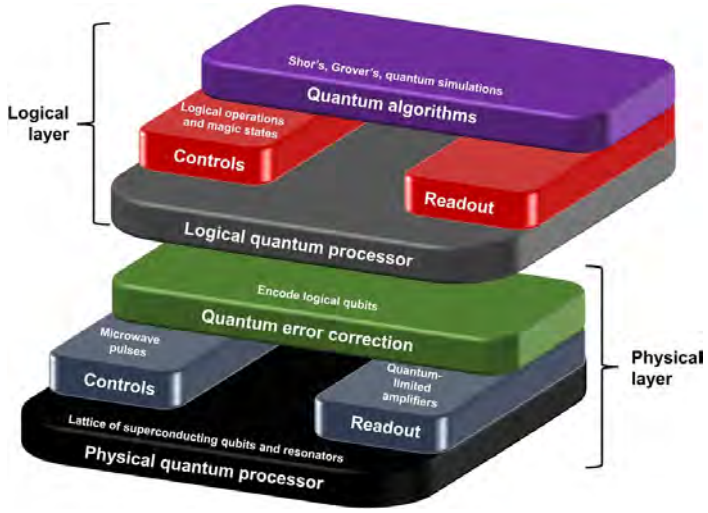


图 6.1: 量子信息处理器的系统视图量子信息处理器包含两个层面：物理层和逻辑层。**物理层**：这个层面包含物理量子处理器，其具有输入和输出线路，受量子纠错（Quantum Error Correction，QEC）处理器的控制。物理层主要负责实现基本的量子操作和纠错机制。它是直接与物理硬件交互的层次，涉及量子位的实际创建和操作，以及纠错码的实施。这一层面的操作通常涉及高级别的复杂性和噪声，需要精细的控制和纠错策略来保证计算的精度和稳定性。**逻辑层**：在这个层面，定义了编码的量子位，并且进行了所需的量子算法的逻辑操作。逻辑层是在物理层之上的一个抽象层，它定义了经过纠错编码的逻辑量子位，并执行逻辑操作。这一层面的操作是用户定义的量子算法，它们通常比物理层的操作更抽象，更依赖于理论和算法设计。这两个层面的设计和实现是实现大规模、可靠的量子计算的关键。物理层需要有效地控制和操纵量子位，以及实施强大的纠错策略来抵抗噪声和错误。逻辑层则需要设计高效的量子算法，并能够通过纠错编码来保护量子信息，以实现复杂的量子计算任务。

图6.1显示了量子计算堆栈的原理图，它从最高层的量子算法和应用到最低层的量子计算机的物理实现范围。许多组件位于这些层之间，如控制、读出和未来的量子错误纠正模块。量子编程语言（QPL）用于与堆栈的顶部接口 [114] 。量子语言包括低级指令，指示在哪些量子位上执行哪些门。由于手动编程会非常繁琐，因此在编程量子计算机时，我们会与开发库中的高级量子编程语言进行交互。现在有许多 QPL 可用，包括函数和命令式语言。函数集包括：Qiskit、LIQUiji、Q 和 Quipper。命令式 QC 编程语言包括：Cirq、Scaffold 和 ProjectQ。有关开源量子计算框架的有用审查，可以参阅 [115] 。

6.1. 量子计算机和 QC 模拟器

云量子计算机使得在真实的量子硬件上运行算法成为可能，开发库使这种能力成为可能。为了在量子计算机上运行代码之前测试代码，大多数量子计算框架提供在经典计算机上运行的量子模拟器。这个模拟器可以在本地或云端运行。由于它在经典计算机上运行，当然无法处理实际的量子状态，但测试代码的语法和流程很有帮助。

经典计算机模拟量子电路的难点主要在于量子状态的存储。一个 n 量子比特（qubit）的系统，

其最通用的状态需要存储 2^n 个复数来描述系统的波函数。每增加一个量子比特，需要的存储空间就翻倍，这就是所谓的”指数爆炸”问题。

例如，假设每个复数使用 1 字节来存储，那么对于 30 量子比特的系统，需要存储 2^{30} 字节，也就是 1GB 的内存。对于 40 量子比特的系统，需要 1TB 的内存，而对于 50 量子比特的系统，需要 1PB（1 千万 GB）的内存。这些内存需求已经接近现代超级计算机的极限，即使只是对数量较少的量子比特进行模拟。

基本的量子计算机模拟方法是通过表示量子电路作用在初始状态上的酉矩阵（unitary transformation）U。然后，模拟器算法只需要进行矩阵乘法，得到最后的状态 $|0\rangle$。然而，由于上述指数爆炸的问题，这种方法对于大规模的量子系统是无法实现的。

$$|\psi\rangle' = U\,|\psi\rangle$$

其中，$|\psi\rangle$ 表示初始的量子态，U 表示量子电路所对应的幺正变换，$|\psi\rangle'$ 表示经过量子电路作用后的最终量子态。

为了解决这个问题，可以使用一些技术来减少内存消耗。一种常见的方法是只存储波函数本身，然后将单比特门和双比特门作用于波函数上。例如，要对第 i 个量子比特应用单量子门 G，则需要将每个下标的第 i 位不同的幅值 α 进行以下操作：

请注意，这种方法需要将电路的整个幺正矩阵（一个 2n×2n 的矩阵）存储在内存中（除了存储波函数）。一种改善这些内存需求的方法是仅将单比特门和双比特门分别应用于波函数。要将单量子比特门：

$$G = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} \tag{6.1}$$

应用于第 i 个量子比特，需要对每个振幅进行矩阵乘积，其中位于第 i 位的指数不同 [116, 117]

$$a_{*\cdots*0_i*\cdots*} = G_{11}a_{*\cdots*0_i*\cdots*} + G_{12}a_{*\cdots*1_i*\cdots*}$$
$$a_{*\cdots*1_i*\cdots*} = G_{21}a_{*\cdots*0_i*\cdots*} + G_{22}a_{*\cdots*1_i*\cdots*}$$

被视为描述两量子比特门在状态向量或波函数模拟器[1]上的操作的更新方程。这些方程说明了当一个两量子比特门作用在量子比特上时，量子比特状态的变化。这里的 $a_{*\cdots*0_i*\cdots*}$ 和 $a_{*\cdots*1_i*\cdots*}$ 表示量子态的复振幅，而 $G_{11}, G_{12}, G_{21}, G_{22}$ 是二元门的矩阵元素。

这种模拟器的工作方式是遍历量子电路中的所有单量子比特和两量子比特门，即按照特定的顺序一次处理电路中的每个量子门。在模拟量子电路的过程中，模拟器通常从电路的起始点开始，依次处理每个量子门，直到电路的结束点。，并应用相应的更新方程。它可以模拟通用的量子计算，但对于大规模的量子系统，这种模拟可能非常耗时和计算密集。

这段话也提到了量子程序在发送到量子后端之前需要进行编译，这一过程将高级别的量子操作转换为实际的物理操作，这些操作在较低级别的语言——量子汇编或指令语言中表达——然后发送到计算机。

[1]波函数模拟器是一种用于模拟量子系统的工具，它使用量子态的波函数表示来模拟量子比特的行为和交互。在量子力学中，波函数是一个复数函数，它提供了关于量子系统的完整信息。特别地，在量子计算中，波函数是用来描述一个或多个量子比特的状态的。

在实际的量子计算中，一个波函数模拟器可以用于模拟量子电路的行为，例如通过模拟量子门的效果以及量子比特之间的纠缠等。这可以帮助研究人员理解和设计量子算法，以及预测实际量子计算机的行为。

然而，模拟大规模的量子系统是非常计算密集的。对于 n 个量子比特的系统，其状态空间的大小为 2^n。这意味着，随着量子比特数量的增加，所需的计算资源将指数级增长。这就是为什么我们需要实际的量子计算机来执行大规模的量子计算，因为它们可以直接操纵和利用量子态，而无需通过经典计算来模拟这些行为。

在物理层面，这些门是通过对量子比特进行物理操作来实现的。这些物理操作可以包括微波脉冲、激光脉冲或作用于量子比特的其他交互作用，具体取决于使用的量子比特的物理实现。

6.2. Cirq

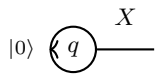


图 6.2: Cirq 程序中的电路

图中表示一个有一个量子比特的电路，该比特初始化为 $|0\rangle$，然后应用了一个 X 门（即 NOT 门），然后进行测量。

简介

Cirq 是谷歌开发的一款量子计算库，旨在支持量子计算的研究和实际应用。作为一个开源的 Python 库，Cirq 提供了一套丰富的工具和接口，使研究人员和开发者能够轻松地构建和模拟量子算法，以及与量子硬件进行交互。随着量子计算技术的不断发展和成熟，Cirq 为实现大规模量子计算提供了关键支持。

主要特性

Cirq 的设计旨在解决量子计算中的关键挑战。以下是 Cirq 的一些主要特性：

- **易用性**：Cirq 提供了直观易用的 API，使得开发者能够轻松地构建量子电路、执行量子操作和模拟量子算法。
- **灵活性**：Cirq 支持广泛的量子门和操作，同时允许用户自定义新的量子门，以满足特定需求。
- **硬件感知**：Cirq 考虑了实际量子硬件的特点和限制，支持硬件约束和噪声模型，有助于实现有效的量子算法。
- **与量子硬件的兼容性**：Cirq 提供了与现有量子硬件和云服务的接口，包括谷歌的量子计算机以及其他第三方平台。

核心组件

Cirq 库包含了一系列核心组件，用于创建、模拟和优化量子电路：

- **量子比特（Qubit）**：Cirq 使用 Qubit 对象表示量子比特，这是量子电路中的基本单位。
- **量子门（Gate）**：Cirq 提供了一系列预定义的量子门（如 X、Y、Z 门、Hadamard 门、CNOT 门等），同时允许用户自定义新的量子门。
- **量子操作（Operation）**：Cirq 将量子门应用于特定的量子比特，生成量子操作。操作可以组合成量子电路或其他复杂结构。
- **量子电路（Circuit）**：Cirq 使用 Circuit 对象表示量子电路，可以将多个量子操作按顺序组织起来，形成一个完整的量子计算过程。
- **模拟器（Simulator）**：Cirq 提供了多种模拟器，用于在经典计算机上模拟量子电路的行为，包括理想情况下的量子计算以及考虑硬件噪声的情况。
- **优化器（Optimizer）**：Cirq 包含了一系列优化器，用于优化量子电路，减少门操作的数量，降低错误率，从而提高量子计算的性能。

- **调度（Scheduling）**：Cirq 支持对量子电路的调度和时间规划，以便将量子操作在硬件上高效地执行。
- **量子云服务接口**：Cirq 提供了与谷歌及其他第三方量子云平台的接口，允许用户将量子电路部署到实际的量子硬件上。

应用范围

Cirq 的功能和特性使其在量子计算领域具有广泛的应用价值。以下是 Cirq 在量子计算中的一些应用场景：

- **量子算法研究**：Cirq 为研究人员提供了一个方便的平台，可以用于构建、模拟和分析各种量子算法，例如 Deutsch-Jozsa 算法、Shor 算法、Grover 搜索算法等。
- **量子错误纠正**：Cirq 可以用于实现量子纠错码和量子纠错技术，以提高量子计算的鲁棒性和可靠性。
- **量子机器学习**：Cirq 为实现基于量子计算的机器学习算法提供了工具和支持，这些算法有望在某些场景中超越经典机器学习方法。
- **量子优化**：Cirq 可以用于解决复杂的组合优化问题，如旅行商问题、图着色问题等，这些问题在经典计算机上难以解决。
- **与量子硬件的交互**：Cirq 支持与实际量子硬件的交互，帮助开发者熟悉量子硬件的特性，为实际应用做好准备。

总结

总之，Cirq 是谷歌开发的一款量子计算库，旨在支持量子计算的研究和实际应用。Cirq 提供了一套丰富的工具和接口，以帮助研究人员和开发者轻松地构建和模拟量子算法，以及与量子硬件进行交互。Cirq 的功能和特性使其在量子计算领域具有广泛的应用价值，为实现大规模量子计算提供了关键支持。

Cirq 是 Google 的量子计算开发库。Cirq 使开发人员能够构建和执行由本书中涵盖的所有常见一元、二元和三元运算符组成的量子电路。在本书的大部分代码示例中，我们将使用 Cirq。为了熟悉这种语言，下面提供了一个 Cirq 的示例程序 [60]。该程序创建一个具有一个量子比特的量子电路，并在其上执行 NOT 运算，然后进行测量。该电路被模拟多次，其测量结果显示在控制台中。完整程序如下所示：1。

在这个电路中，我们首先在网格图案中设置了一个量子比特。Cirq 还允许根据需要线性设置量子比特，因为线性和二维量子比特阵列是近期量子计算机架构中最流行的。然后，我们将 NOT 运算符应用于量子比特。如果量子比特在之前处于状态 j0i，则现在处于状态 j1i，反之亦然。然后，我们测量量子比特以输出测量结果的经典位。注意，在测量操作中使用关键字参数 key=’m’，可以通过 Cirq TrialResult 的直方图方法轻松访问测量结果。在这个简单的程序中不是必需的，但在更复杂的程序中，这可能是一个有用的工具。

接下来的代码行获取一个 QC 模拟器并执行电路十次。然后，执行电路的结果被打印到屏幕上。该程序的一个样本输出如下所示。电路被以 ASCII 文本形式打印出来，这是在 Cirq 中绘制电路的标准方式，而结果则由一系列二进制数字表示。正如我们预期的那样，在无噪声的 QC 模拟器上，所有的测量结果都等于 1。最后，注意，测量结果的字符串由提供给测量操作的关键字参数 m 标记。

我们可以在布洛赫球上（参见图 6.3）表示这个电路中 X 运算符的应用。我们回忆一下，布洛赫球表示极点处的计算基础态 j0i 和 j1i。通过将 X 门应用于 j0i 的准备好的状态，我们将量子比特移动到状态 j1i。如果然后应用一个 Hadamard 门，我们可以用布洛赫球上的水平向量表示新的量子比特状态，对应于：j0i □ j1i p 2

对于许多运算符的电路，我们可以使用来自 Q-Ctrl（以及附带网站上列出的其他网站）的动态布洛赫球模拟软件来可视化幺正变换。

6.2.1. Qiskit

简介

Qiskit（Quantum Information Science Kit）是 IBM 开发的一款量子计算软件库，旨在推动量子计算的研究和实际应用。作为一个开源的 Python 库，Qiskit 提供了一系列工具和接口，使研究人员和开发者能够轻松地构建和模拟量子算法，以及与量子硬件进行交互。Qiskit 的发展与 IBM 量子计算机的推广密切相关，对于量子计算领域的普及和发展具有重要意义。

主要特性

Qiskit 的设计旨在解决量子计算中的关键挑战。以下是 Qiskit 的一些主要特性：

- **易用性**：Qiskit 提供了直观易用的 API，使得开发者能够轻松地构建量子电路、执行量子操作和模拟量子算法。
- **灵活性**：Qiskit 支持广泛的量子门和操作，同时允许用户自定义新的量子门，以满足特定需求。
- **硬件感知**：Qiskit 考虑了实际量子硬件的特点和限制，支持硬件约束和噪声模型，有助于实现有效的量子算法。
- **与量子硬件的兼容性**：Qiskit 提供了与现有量子硬件和云服务的接口，包括 IBM 的量子计算机以及其他第三方平台。

核心组件

Qiskit 库包含了一系列核心组件，用于创建、模拟和优化量子电路：

- **量子比特（Qubit）**：Qiskit 使用 Qubit 对象表示量子比特，这是量子电路中的基本单位。
- **量子门（Gate）**：Qiskit 提供了一系列预定义的量子门（如 X、Y、Z 门、Hadamard 门、CNOT 门等），同时允许用户自定义新的量子门。
- **量子操作（Operation）**：Qiskit 将量子门应用于特定的量子比特，生成量子操作。操作可以组合成量子电路或其他复杂结构。
- **量子电路（Circuit）**：Qiskit 使用 Circuit 对象表示量子电路，可以将多个量子操作按顺序组织起来，形成一个完整的量子计算过程。
- **模拟器（Simulator）**：Qiskit 提供了多种模拟器，用于在经典计算机上模拟量子电路的行为，包括理想情况下的量子计算以及考虑硬件噪声的情况。
- **优化器（Optimizer）**：Qiskit 包含了一系列优化器，用于优化量子电路，减少门操作的数量，降低错误率，从而提高量子计算的性能。
- **调度（Scheduling）**：Qiskit 支持对量子电路的调度和时间规划，以便将量子操作在硬件上高效地执行。

- **量子云服务接口**：Qiskit 提供了与 IBM Quantum Experience 及其他第三方量子云平台的接口，允许用户将量子电路部署到实际的量子硬件上。

应用范围

Qiskit 的功能和特性使其在量子计算领域具有广泛的应用价值。以下是 Qiskit 在量子计算中的一些应用场景：

- **量子算法研究**：Qiskit 为研究人员提供了一个方便的平台，可以用于构建、模拟和分析各种量子算法，例如 Deutsch-Jozsa 算法、Shor 算法、Grover 搜索算法等。
- **量子错误纠正**：Qiskit 可以用于实现量子纠错码和量子纠错技术，以提高量子计算的鲁棒性和可靠性。
- **量子机器学习**：Qiskit 为实现基于量子计算的机器学习算法提供了工具和支持，这些算法有望在某些场景中超越经典机器学习方法。
- **量子优化**：Qiskit 可以用于解决复杂的组合优化问题，如旅行商问题、图着色问题等，这些问题在经典计算机上难以解决。
- **与量子硬件的交互**：Qiskit 支持与实际量子硬件的交互，帮助开发者熟悉量子硬件的特性，为实际应用做好准备。

总之，Qiskit 是 IBM 开发的一款量子计算库，旨在推动量子计算的研究和实际应用。Qiskit 提供了一系列工具和接口，以帮助研究人员和开发者轻松地构建和模拟量子算法，以及与量子硬件进行交互。Qiskit 的功能和特性使其在量子计算领域具有广泛的应用价值，对于量子计算领域的普及和发展具有重要意义。

量子信息科学工具包（Quantum Information Science Kit，简称 Qiskit）是由 IBM 创建的量子计算开发库。Qiskit 库是一个用于编程量子计算机的灵活框架。Qiskit 开发库由四个核心模块组成，分布在量子计算堆栈中：

Qiskit Terra：Terra 为在电路和脉冲级别组成量子程序提供核心元素，并根据特定物理量子处理器的约束进行优化。Qiskit Aer：Aer 提供了一个 C++ 模拟器框架以及用于构建噪声模型的工具，用于执行在真实设备上运行时发生的错误的现实嘈杂模拟。Qiskit Ignis：Ignis 是一个用于理解和减轻量子电路和设备中噪声的框架。Qiskit Aqua：Aqua 包含一个跨领域量子算法库，可在此基础上构建近期量子计算的应用。除了 Aqua 之外，截至撰写时，这些组件都会自动安装在 Qiskit 中。Aqua 模块可以单独安装，需要具有核心 Qiskit 库的工作安装。为了展示 Qiskit 中的编码语法，我们在下面包含了之前在 Cirq 中展示的相同示例程序。

这个 Qiskit 程序与 Cirq 中的程序非常相似，只是由于语言设计、语法和符号的不同而有细微差别。在导入 Qiskit 开发库后，程序声明了一个具有一个量子比特的量子和经典寄存器，然后用它创建一个电路。注意这与 Cirq 中的做法不同，Cirq 中（1）从不显式创建经典寄存器，（2）量子比特仅在添加操作时才在电路中引用。继续通过 Qiskit 程序，接下来的几行将适当的操作（NOT 和测量）添加到电路中，然后通过绘制电路打印出来。

Qiskit 具有绘制和将电路另存为文件的功能，除了打印出文本表示。图 6.5 展示了使用下面的代码绘制的这个程序的电路：

在打印电路之后，声明了一个用于执行量子电路的后端。在最后几行中，执行电路，检索结果，最后将测量统计数据（计数）打印到屏幕上。该程序的一个示例输出如下所示。请注意，电路也将在上面的程序中打印到控制台-我们在这里省略了文本表示。

在 Qiskit 中，测量结果以字典形式存储（一个由键值对组成的 Python 数据类型），其中键是位字符串，值是每个位字符串被测量的次数。在这里，这个输出表示测量中出现的唯一位字符串是 1。与 Cirq 输出类似，由于我们在没有激活噪声模型的情况下在量子计算模拟器上运行该程序，所有测量结果都是预期的 1。

总结一下，Qiskit 是一个由 IBM 创建的量子计算开发库，旨在提供一个灵活的量子计算机编程框架。Qiskit 开发库由四个核心模块组成，分布在量子计算堆栈中。Qiskit 可以方便地用于构建和模拟量子电路，同时提供了与硬件的交互接口。其程序与 Cirq 非常相似，但由于语言设计、语法和符号的不同而存在一些细微差别。Qiskit 具有绘制和将电路另存为文件的功能，可以方便地分析和查看量子电路。

6.2.2. Forest

简介

Forest 是 Rigetti Computing 开发的量子计算软件库，旨在推动量子计算的研究和应用。Forest 提供了一系列工具和接口，使研究人员和开发者能够轻松地构建和模拟量子算法，并与量子硬件进行交互。Forest 的设计考虑了实际量子硬件的特点和限制，有助于实现有效的量子算法。作为一个开源软件库，Forest 对量子计算领域的普及和发展具有重要意义。

主要特性

Forest 的设计旨在解决量子计算中的关键挑战。以下是 Forest 的一些主要特性：

- **易用性**：Forest 提供了直观易用的 API，使得开发者能够轻松地构建量子电路、执行量子操作和模拟量子算法。
- **灵活性**：Forest 支持广泛的量子门和操作，同时允许用户自定义新的量子门，以满足特定需求。
- **硬件感知**：Forest 考虑了实际量子硬件的特点和限制，支持硬件约束和噪声模型，有助于实现有效的量子算法。
- **与量子硬件的兼容性**：Forest 提供了与 Rigetti 量子计算机以及其他第三方平台的接口，允许用户将量子电路部署到实际的量子硬件上。

核心组件

Forest 库包含了一系列核心组件，用于创建、模拟和优化量子电路：

- **量子比特（Qubit）**：Forest 使用 Qubit 对象表示量子比特，这是量子电路中的基本单位。
- **量子门（Gate）**：Forest 提供了一系列预定义的量子门（如 X、Y、Z 门、Hadamard 门、CNOT 门等），同时允许用户自定义新的量子门。
- **量子操作（Operation）**：Forest 将量子门应用于特定的量子比特，生成量子操作。操作可以组合成量子电路或其他复杂结构。
- **量子电路（Circuit）**：Forest 使用 Circuit 对象表示量子电路，可以将多个量子操作按顺序组织起来，形成一个完整的量子计算过程。
- **模拟器（Simulator）**：Forest 提供了多种模拟器，用于在经典计算机上模拟量子电路的行为，包括理想情况下的量子计算以及考虑硬件噪声的情况。

- **优化器（Optimizer）**：Forest 包含了一系列优化器，用于优化量子电路，减少门操作的数量，提高量子算法的效率。
- **编译器（Compiler）**：Forest 的编译器可以将量子电路转换为适合实际量子硬件的形式。它会考虑硬件的约束和特性，生成一个优化过的量子电路。
- **硬件接口（Hardware Interface）**：Forest 提供了与 Rigetti 量子计算机以及其他第三方平台的接口，使得用户可以将量子电路部署到实际的量子硬件上进行测试。

应用领域

Forest 适用于各种量子计算相关的研究和应用领域，包括：

- **量子算法研究**：Forest 提供了强大的工具和接口，使研究人员能够快速地设计和测试新的量子算法。
- **量子计算模拟**：Forest 的模拟器可以帮助研究人员在经典计算机上模拟量子电路的行为，探究量子计算的原理和性能。
- **量子优化**：Forest 包含了一系列优化器，可以用于优化量子电路，提高量子算法的效率和鲁棒性。
- **量子硬件测试**：通过 Forest 的硬件接口，用户可以将量子电路部署到实际的量子硬件上，评估量子计算机的性能和可靠性。
- **量子计算教育**：Forest 作为一个开源软件库，可以帮助学生和教师更好地了解量子计算的原理和应用，推动量子计算领域的普及和发展。

总结

Rigetti Computing 开发的 Forest 是一个功能强大、易用的量子计算软件库，旨在推动量子计算的研究和应用。Forest 提供了一系列工具和接口，使研究人员和开发者能够轻松地构建和模拟量子算法，并与量子硬件进行交互。Forest 的设计考虑了实际量子硬件的特点和限制，有助于实现有效的量子算法。通过 Forest，用户可以在各种量子计算相关的研究和应用领域取得显著成果。作为一个开源软件库，Forest 对量子计算领域的普及和发展具有重要意义。

6.2.3. Quantum Development Kit

简介

Quantum Development Kit（QDK）是微软为量子计算开发者推出的一套全面的开发工具集。QDK 的核心组件是 Q（Q-sharp）编程语言，这是一种专门为量子计算设计的高级编程语言。QDK 提供了一系列工具和资源，帮助开发者编写、测试和运行 Q 程序，从而实现量子算法的开发和模拟。QDK 的目标是降低量子计算的学习门槛，推动量子计算领域的创新和发展。

Q# 编程语言

Q# 是 QDK 的核心组件，它是一种与 C# 和 Python 兼容的领域特定编程语言。Q# 的设计旨在充分发挥量子计算的优势，同时克服其固有的挑战。Q# 提供了一套丰富的内置库，支持各种量子门、操作和算法。此外，Q# 还提供了一些先进的特性，如类型安全、模块化设计和函数式编程元素，使得开发者能够编写高效且可靠的量子代码。

主要组件

QDK 包含了一系列工具和资源，帮助开发者构建和运行 Q 程序：

- **编译器**：QDK 提供了一个 Q 编译器，将 Q 代码转换为可在量子计算机或模拟器上运行的形式。
- **模拟器**：QDK 包含了多种模拟器，用于在经典计算机上模拟量子程序的行为。这包括理想情况下的量子计算以及考虑硬件噪声和误差的情况。
- **资源估计器**：QDK 提供了一个资源估计器，用于分析量子程序的性能，如所需的量子门数量、深度、宽度等。
- **调试和诊断工具**：QDK 集成了一系列调试和诊断工具，如断点、单步执行、变量监视等，帮助开发者快速定位和解决问题。
- **文档和示例**：QDK 附带了丰富的文档和示例代码，帮助开发者快速上手 Q 编程和量子算法设计。

（1）

应用领域

QDK 适用于各种量子计算相关的研究和应用领域，包括：

- **量子算法研究**：QDK 提供了强大的工具和接口，使研究人员能够快速地设计和测试新的量子算法。
- **量子计算模拟**：QDK 的模拟器可以帮助研究人员在经典计算机上模拟量子程序的行为，探究量子计算的原理和性能。
- **量子优化**：QDK 提供了一些工具和方法，用于优化量子电路和算法，提高量子计算的效率和鲁棒性。
- **量子软件开发**：QDK 使得开发者能够编写高质量的量子软件，为实际量子硬件提供可靠的支持。
- **量子计算教育**：QDK 作为一套全面的开发工具集，可以帮助学生和教师更好地了解量子计算的原理和应用，推动量子计算领域的普及和发展。

与其他量子软件库的比较

QDK 与其他量子软件库（如 Qiskit、Cirq 和 Forest）有一些相似之处，例如都提供了量子编程语言、模拟器和优化器等工具。然而，QDK 具有一些独特的优势：

- **Q# 编程语言**：Q# 是一种专为量子计算设计的高级编程语言，与 C# 和 Python 兼容，提供了丰富的内置库和先进的编程特性。
- **与经典计算的紧密集成**：QDK 强调量子程序与经典程序的协同作用，使得开发者能够在一个统一的环境中编写混合的量子-经典程序。
- **跨平台支持**：QDK 支持多种操作系统（如 Windows、macOS 和 Linux），并提供了与多种量子硬件平台的接口。

总结

微软推出的 Quantum Development Kit 是一套全面的量子计算开发工具集，旨在降低量子计算的学习门槛，推动量子计算领域的创新和发展。QDK 的核心组件是 Q 编程语言，这是一种专门为量子计算设计的高级编程语言。QDK 提供了一系列工具和资源，帮助开发者编写、测试和运行 Q 程序，从而实现量子算法的开发和模拟。QDK 适用于各种量子计算相关的研究和应用领域，包括量子算法研究、量子计算模拟、量子优化、量子软件开发和量子计算教育。

Quantum Development Kit 量子开发工具包（QDK）是微软推出的一套量子计算开发库。与之前基于 Python 的库（如 Cirq、Qiskit 和 pyQuil）不同，QDK 包含了一种名为 Q#（发音为“Q sharp”）的专门用于编写量子程序的语言。与基于 Python 的库相比，Q 语言在以下几个方面有所不同：我们需要显式声明类型，使用花括号而非 Python 中的缩进。此外，使用 QDK 执行程序需要三个单独的文件：

- 以.qs 结尾的文件，用于存储量子操作（类似于 Python 中的函数）
- 以.cs 结尾的驱动文件，用于在主程序中执行量子操作
- 以.csproj 结尾的文件，用于定义项目并包含有关计算机体系结构和程序包引用的元数据

我们在下面的示例程序中展示了 Q 中执行与其他语言相同的“非门和测量”电路。首先，我们展示定义驱动文件中将使用的量子操作的.qs 文件。在第一行，我们为操作定义一个命名空间。驱动文件将使用此命名空间来访问这些操作。接下来的两行相当于 Python 中的导入程序包——它们使 QDK 中定义的操作可供我们在程序中使用（例如 X 门）。接着，我们声明了一个名为 Set 的操作，它输入一个期望的计算基态和一个任意的量子态；然后改变量子态以匹配期望的状态。这是通过在计算基础上测量量子比特，然后在必要时执行非门操作来实现的。

接下来，我们定义 NotAndMeasure 操作，该操作设置量子电路并测量用户指定次数的量子比特。具体来说，该操作使用之前定义的操作将量子比特设置为 0 状态，执行非门操作，然后测量，并记录测量到的 1 的数量。在完成总的重复次数后，返回测量到的 1 的数量。

为了使用这个文件，我们必须定义一个以.cs 结尾的单独驱动文件，用于执行这些操作。用于执行此程序的驱动文件如下所示。

在这里，我们导入 System，它允许我们将结果打印到控制台，并使用 QDK 中的量子计算模拟器所需的工具。接下来，我们调用上面.qs 文件中声明的命名空间，并在驱动类中定义一个 Main 函数。这个 Main 函数使用量子计算模拟器运行 NotAndMeasure 操作，然后将结果打印到控制台。该程序的输出如下所示。

与其他语言一样，我们得到了正确的输出，即测量全 1。最后一个需要执行此代码的.csproj 文件包含在此书的 GitHub 网站上。我们在这里省略了程序，因为对于 QDK 中的每个项目，它几乎是相同的。

总结一下，微软的量子开发工具包（QDK）提供了一种名为 Q 的专用量子编程语言，以及用于构建和模拟量子电路的工具。与其他基于 Python 的量子库（如 Cirq、Qiskit 和 pyQuil）相比，Q 具有不同的语法和结构，需要用户显式声明类型，并使用花括号而非缩进。

QDK 的一个显著特点是，它使用单独的文件（.qs、.cs 和.csproj）来管理和组织量子程序的不同部分，使程序更加模块化。另外，QDK 与经典计算紧密集成，强调量子程序与经典程序之间的协同作用，有助于实现混合的量子-经典程序。

在我们展示的示例中，Q 程序通过在计算基础上测量量子比特并执行非门操作来设置和测量量子电路。尽管 Q 与其他量子库在语法和结构上有所不同，但它们的核心功能和目标相似，即为

量子计算的发展和应用提供支持。

6.3. 开发库总结

开发库总结

在为这些库展示的示例程序中，我们已经看到，在构建量子电路方面，大多数框架都非常相似。它们的通用步骤如下：

- 构建由量子和/或经典寄存器组成的量子电路
- 向电路中添加操作
- 模拟电路

然而，各种库之间还是存在一些差异。例如，在 Cirq 中，量子比特与量子电路分开定义，而在 Qiskit 中，量子比特需要作为量子电路的输入。类似地，在 Qiskit 中，所有的量子比特必须在执行操作之前分配到量子寄存器中，但在 pyQuil 中编程时，可以动态分配量子比特。

尽管这些差异可能看起来很小，但各个开发库所包含的功能有较大差异。例如，有些库包括模拟噪声的能力，将算法编译到任意架构，访问用于调试的波函数等。这些差异导致某些算法在特定库中更容易实现。虽然一个开发库通常足以应对大多数任务，但具备多个库的经验有助于为特定情况选择合适的库。

6.3.1. 使用库

在本书的其余部分，我们将使用这些开发库来探讨组成量子计算基础和更多近期量子计算方法的核心算法。虽然 Cirq 是主要使用的库，但其他一些库中的程序也用于进行比较。在继续阅读本文的核心部分之前，我们简要介绍一下其他尚未涉及的开发库。

6.3.2. 其他开发库

正如前面所提到的，在云量子计算出现之前，许多量子计算机模拟器是用从 C++（例如 Quantum++）到 Java（例如 jSQ）到 Rust（例如 QCGPU）等语言开发的。在 https://quantiki.org/wiki/list-qc-simulators 上可以找到量子计算机模拟器的完整列表。虽然许多这样的模拟器已经过时或被弃用，但其中一些仍在积极开发和改进。实际上，虽然人们普遍认为经典算法在模拟量子电路方面难以实现，但找到更快速模拟更多量子比特的新方法仍是一个活跃的研究领域。

除了上面的库，还有其他一些值得关注的库：ProjectQ，一种带有高性能 C++ 量子计算机模拟器的 Python 库；Strawberry Fields，一种围绕连续变量量子计算构建的 Python 库；以及 Ocean，一种用于在 D-Wave 量子计算机上进行量子退火的 Python 库。此外，还有一种名为 XACC 的硬件独立量子编程框架也在积极开发中。要获得开源量子软件项目的完整列表，请参阅本书的配套网站。

本章向您介绍了各种量子计算库，包括 Cirq、Qiskit、pyQuil、QDK 等。虽然这些库在编程语法和结构上有所不同，但它们的核心功能和目标是相似的，即为量子计算的发展和应用提供支持。各个库所包含的功能和特点可能因库而异，因此熟悉多个库将有助于为特定场景选择合适的库。

在本书接下来的部分，我们将利用这些量子开发库来研究量子计算领域的核心算法和最新方法。虽然 Cirq 是主要使用的库，但我们也会在其他库中展示一些程序以进行比较。通过深入了解这些库，您将能够更好地应对实际中的量子计算问题，并在未来的量子计算应用中发挥关键作用。

6.4. 附加量子程序

从原理上讲，了解如何编程电路、模拟它们以及访问测量结果是未来章节中编写量子算法的核心技能。虽然简单的“NOT 和测量”程序包含了这三个关键特性，但它仍然简单。为了帮助弥补到即将到来的更复杂程序的差距，我们在本节中包括了两个附加的示例量子程序。

6.4.1. 贝尔态

一个常见的电路模式是我们用来从基态准备四个贝尔态之一的一组操作符

C1 jˆ iD p Œj00iCj11i□ 2

这可以很容易地验证，这样的电路由第一个量子比特上的哈达玛尔门和两个量子比特之间的 CNOT 门组成，其中第一个量子比特是控制量子比特。这种结构在量子隐形传态电路中出现，用于在两方之间创建纠缠量子比特对。

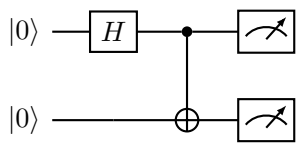
用 Cirq 编写的用于准备贝尔态的电路如下所示。出于教学目的，我们在贝尔态准备电路之后添加测量，以查看可能的测量结果。

Listing 6.1: 用 Cirq 库创建了一个贝尔态准备电路

```
# 导入Cirq库
import cirq

# 获取量子比特和电路
qreg = [cirq.LineQubit(x) for x in range(2)]
circ = cirq.Circuit()

# 添加贝尔态制备电路
circ.append([cirq.H(qreg[0]),
             cirq.CNOT(qreg[0], qreg[1])])

# 显示电路
print("Circuit")
print(circ)

# 添加测量
circ.append(cirq.measure(*qreg, key="z"))

# 模拟电路
sim = cirq.Simulator()
res = sim.run(circ, repetitions=100)

# 显示结果
print("\nMeasurements:")
print(res.histogram(key="z"))
```

请注意，在这个 Cirq 程序中，我们使用测量键通过直方图方法访问测量结果。这个程序的一个示例输出如下所示：

Bell State Preparation Circuit



在这里，电路图展示了由一个哈达玛尔门和一个 CNOT 门组成的贝尔态制备电路。之后，测量结果列举了测量到的 0 和 3 状态的数量。注意这是位字符串的二进制表示，即 0 代表 00，3 代表 11。正如预期的那样，这两个结果是程序中唯一的测量结果，这意味着测量结果是完全相关的：当一个量子比特测量为 0.1/时，另一个量子比特总是测量为 0.1/。

具有参数的门有几个量子门是根据角度定义的，例如标准旋转门 $R_x(\theta)$, $R_y(\theta)$ 和 $R_z(\theta)$。在许多量子算法中，称为变分量子算法，这些角度或参数会被迭代调整以最小化成本。例如，在变分量子本征求解器中，调整门参数以最小化哈密顿量的期望值 $h(\theta)\langle H|\theta\rangle$。无论应用如何，这类变分量子算法都严重依赖于更新和更改门参数的能力。

因此，大多数量子计算开发库都包含内置的功能和方法来处理带参数的门。在以下程序中，我们演示了 Cirq 中的这个功能。特别是，我们设置了一个带有一个参数化门的简单量子电路，对一组参数执行该算法，并绘制测量结果。

c

我们在此程序中强调几个关键组件。首先，Cirq 中的符号用于表示在执行电路之前将确定的参数的数值。当电路被打印（见下文）时，符号的名称将显示在电路中。

接下来，扫描是一个符号将要采用的值集合。Cirq 中的模拟器有一个名为 run_sweep 的方法，用于在扫描中的所有值上执行电路。在每个值上，电路将根据重复次数关键字参数进行多次模拟。程序中剩下的代码用于绘制 run_sweep 的结果，如图 6.8 所示。

现在我们已经探讨了各种量子计算开发库，接下来让我们来看看三个协议 - 量子隐形传态、超密编码和贝尔不等式测试 - 然后转向建立该领域的量子算法宗旨。

7

遥感、超密集编码和贝尔不等式

两个最引人入胜的量子电路使我们能够以在经典世界中不可能的方式传输信息。在本章中，我们将学习如何构建这两个电路。然后，我们将研究量子力学中的一个基本进展，即贝尔不等式。

7.1. 量子传送

量子传送尽管名字如此，但并不传送任何实物。它可以在任何距离完全安全地传输量子比特的状态。从量子力学的提出到我们意识到这个框架为我们提供了一种新型安全通信方式，经历了超过 70 年的时间。这个协议是由贝内特和布拉萨德等人于 1993 年开发的 [118][26]；并于 1997 年得到实验验证 [119][41]。贝内特和布拉萨德在 1984 年还开发了量子密钥分发，称为 BB84[17][25]。

量子传送的一个关键洞察是，我们可以将纠缠态视为一种资源。我们可以使用纠缠态（也称为 EPR 对或贝尔态）来完成一系列无法使用经典手段完成的任务。

在量子传送中，爱丽丝是发送者，她希望将量子比特 Q 的状态传输给接收者鲍勃。该算法总共需要三个量子比特。让我们详细介绍一下这个协议：

我们用三个量子比特设置系统：（a）爱丽丝有一个量子比特 Q，状态为 j^i。爱丽丝希望以安全的方式将状态 j^i 传输给鲍勃。（b）为了实现这个目标，爱丽丝还开始使用另外两个量子比特，我们将其标记为 R 和 S。其中一个量子比特，比如 S，将被发送给鲍勃，另一个将留在爱丽丝那里。实际上，如果量子比特是光子，S 可以通过光纤等量子通道发送。爱丽丝用量子比特 R 和 S 准备一个贝尔态。这是通过在量子比特 R 上应用 Hadamard 门，然后在 R 和 S 之间进行 CNOT 操作，以 R 为控制量子比特。此时，爱丽丝将量子比特 S 发送给鲍勃。爱丽丝现在对她的原始量子比特 Q 和她的 EPR 对的一半 R 进行贝尔测量。这是通过在量子比特之间进行 CNOT 操作，以 Q 为控制量子比特，然后在 Q 上执行 Hadamard 门，最后在计算基础上测量两个量子比特来完成的。测量后，爱丽丝现在有两位经典信息，每个测量量子比特各得一位。爱丽丝现在通过一个经典通信渠道将这两位传输给鲍勃。注意，她的测量结果有四种可能的输出：00、01、10 和 11。5. 根据鲍勃从爱丽丝那里收到的比特串，他对自己的量子比特 S 执行一系列操作。每个测量结果的操作字典如下所示。执行相应的操作可以确保鲍勃的量子比特 S 处于与爱丽丝原始量子比特 Q 相同的状态——尽管爱丽丝和鲍勃都不知道这个状态是什么！

在图 7.2 和下面所示的电路图中，可以看到两种表示量子传送的方法；它们是等效的，熟悉不同呈现电路的方法是很有用的：让我们强调一下关于传送的以下几点：

注意爱丽丝如何在 R 上应用 H 门，然后在 R 和 S 之间进行 CNOT 操作来准备贝尔态。注意爱丽丝向鲍勃传输两位经典信息——这些在量子电路中用双线表示。在这个电路中，爱丽丝通过 EPR 对和两个经典比特成功地将 j^i 状态传输给鲍勃。我们还可以将这种传输表示为 Œqq□ C Œcc□ Œq□ 其中 Œqq□ 表示 EPR 对，Œcc□ 表示一对经典比特，Œq□ 是我们希望传输的量子比特的状态。虽然量子传送是我们在量子通信中可以使用的工具，但它也在量子计算中有应用 [120][58]。我们可以使用量子传送在可扩展量子计算机中创建模块化架构，将量子态从一个模块发送到另一个模块（参见 [120][58]）。

7.2. Superdense Coding

超密集编码超密集编码是一种通过仅发送一个量子比特从发送方到接收方来传输经典比特的方法。如果爱丽丝希望通过一个经典通道将两个经典比特传输给鲍勃，她将不得不使用两个比特。然而，通过超密集编码，她可以通过仅传输一个量子比特来传递这两个比特。

这个协议最初由贝内特和维斯纳 [121][27] 开发，后来被进一步指定为一种安全通信协议 [122][226]。安东 · 齐林格于 1995 年实验上展示了超密集编码传输 [123][144]。

为了实现超密集编码，爱丽丝首先准备一个 EPR 对。然后，她对这对中的一半执行四种操作之一。假设这是一对光子。要创建贝尔对，爱丽丝首先将 Hadamard 应用于她的光子，然后将 CNOT 应用于两个光子，就像在为量子传送协议准备 EPR 对的情况下一样。这对光子现在是纠缠的。

现在爱丽丝选择她希望将四个经典状态中的哪一个作为要发送给鲍勃的预期消息。根据她选择发送的消息，爱丽丝对她的光子应用一个特定的量子算子。

如果爱丽丝想要发送爱丽丝应用的操作是 00 I（恒等算子）01 X 10 Z 11 ZX（先应用 X，然后应用 Z）

接下来，她通过一个保持纠缠的量子通信渠道将她的光子发送给鲍勃。收到光子后，鲍勃将 Hadamard 应用于爱丽丝的光子，然后将 CNOT 应用于光子对。然后他进行测量。结果将是两个经典比特的信息。让我们回想一下，测量输出是经典信息。

我们可以用以下简写表示超密集编码：Œq□ C Œqq□ Œcc□

这表示通过一个量子比特（Œq□）和一个 EPR 对（Œqq□），我们可以成功地传输两个经典比特的信息（Œcc□）。

总结一下，超密集编码允许在发送单个量子比特的情况下传输两个经典比特的信息。这是通过使用纠缠的量子比特对（EPR 对）以及根据要发送的经典信息选择适当的量子操作来实现的。一旦发送方将其量子比特发送给接收方，接收方可以通过解纠缠和测量来获得两个经典比特的信息。

值得注意的是，超密集编码与量子传送有相似之处，它们都利用了量子纠缠作为一种资源。然而，两者之间的目标和用途有所不同：量子传送用于在不知道原始状态的情况下将量子比特的状态安全地传输给另一个接收者，而超密集编码用于在发送单个量子比特的情况下传输两个经典比特的信息。

超密集编码的应用包括量子通信和量子计算中的信息传输。例如，它可以用于提高量子通信的带宽效率，或者在分布式量子计算中传输信息。

7.3. 量子传输和超密集通信的代码

量子隐形传态和超密编码的代码

下面用 Cirq 提供了一个量子隐形传态的程序。该程序在 Alice 的量子比特中编码一个随机量子态，并打印出其 Bloch 球的.x; y; z/ 分量。然后，它执行量子隐形传态电路并打印出 Bob 量子比特的 Bloch 球.x; y; z/ 分量。下面显示了该程序的一个示例输出。如可见，Alice 和 Bob 的量子比特的 Bloch 球分量是相同的 - 换句话说，量子比特已经从 Alice ” 传送” 到 Bob。

下面用 Cirq 提供了一个超密编码的程序。

Listing 7.1: 量子隐形传态

```
"""Cirq 中的量子隐形传态。修改自以下示例:
    quantum_teleportation.py:
https://github.com/quantumlib/Cirq/tree/master/examples
"""
# 导入
import random
import cirq

def make_quantum_teleportation_circuit(ranX, ranY):
    """返回一个量子隐形传态电路。"""
    circuit = cirq.Circuit()
    msg, alice, bob = cirq.LineQubit.range(3)
    # 在 Alice 和 Bob 之间创建贝尔态
    circuit.append([cirq.H(alice), cirq.CNOT(alice, bob)])
    # 为信息创建一个随机态
    circuit.append([cirq.X(msg)**ranX, cirq.Y(msg)**ranY])
    # 对信息和 Alice 的纠缠量子比特进行贝尔测量
    circuit.append([cirq.CNOT(msg, alice), cirq.H(msg)])
    circuit.append(cirq.measure(msg, alice))
    # 使用贝尔测量的两个经典比特来恢复
    # Bob 的纠缠量子比特上的原始量子信息
    circuit.append([cirq.CNOT(alice, bob), cirq.CZ(msg, bob)])
    return msg, circuit

def main():
    # 编码一个随机态进行传送
    ranX = random.random()
    ranY = random.random()
    msg, circuit = make_quantum_teleportation_circuit(ranX, ranY)
    # 模拟电路
    sim = cirq.Simulator()
    message = sim.simulate(cirq.Circuit.from_ops(
        [cirq.X(msg)**ranX, cirq.Y(msg)**ranY]))
    # 打印 Alice 量子比特的 Bloch 球
    print("Alice 的量子比特的 Bloch 球:")
    b0X, b0Y, b0Z = cirq.bloch_vector_from_state_vector(
        message.final_state, 0)
    print("x: ", round(b0X, 4),
        "y: ", round(b0Y, 4),
        "z: ", round(b0Z, 4))
    # 显示隐形传态电路
    print("\n电路:")
    print(circuit)
    # 记录模拟的最终状态
    final_results = sim.simulate(circuit)
```

```
    # 打印 Bob 量子比特的 Bloch 球
    print("\nBob 的量子比特的 Bloch 球:")
    b2X, b2Y, b2Z = cirq.bloch_vector_from_state_vector(
        final_results.final_state, 2)
    print("x: ", round(b2X, 4),
        "y: ", round(b2Y, 4),
        "z: ", round(b2Z, 4))

if __name__ == '__main__':
    main()
```

下面显示了该程序对消息 01 的一个示例输出:

```
Bloch sphere of Alice's qubit:
x: 0.654 y: -0.6177 z: -0.4367
Bloch sphere of Bob's qubit:
x: 0.654 y: -0.6177 z: -0.4367
```

如可见，Bob 收到的消息正是 Alice 通过超密编码发送的消息。

下面提供一个用 Cirq 编写的超密集编码的程序:

Listing 7.2: 超密编码

```
"""Cirq 中的超密编码。"""
# 导入
import cirq

# 用于可视化输出的辅助函数
def bitstring(bits):
    return ''.join('1' if e else '0' for e in bits)

# 创建两个量子和经典寄存器
qreg = [cirq.LineQubit(x) for x in range(2)]
circ = cirq.Circuit()

# 每个消息的操作字典
message = {"00": [],
           "01": [cirq.X(qreg[0])],
           "10": [cirq.Z(qreg[0])],
           "11": [cirq.X(qreg[0]), cirq.Z(qreg[0])]}

# Alice 创建一个贝尔对
circ.append(cirq.H(qreg[0]))
circ.append(cirq.CNOT(qreg[0], qreg[1]))

# Alice 选择要发送的消息
m = "01"
print("Alice 发送的消息 =", m)

# Alice 用适当的量子操作对她的消息进行编码
circ.append(message[m])

# Bob 用贝尔基测量
circ.append(cirq.CNOT(qreg[0], qreg[1]))
circ.append(cirq.H(qreg[0]))
circ.append([cirq.measure(qreg[0]), cirq.measure(qreg[1])])

# 打印电路
```

```
print("\n电路:")
print(circ)

# 在模拟器后端上运行量子电路
sim = cirq.Simulator()
res = sim.run(circ, repetitions=1)

# 打印 Bob 收到的消息: 电路的结果
print("\nBob 收到的消息 =", bitstring(res.measurements.values()))
```

下面是这个程序对信息 01 的输出示例。

```
Alice's sent message = 01
Circuit:
0: ---H---@---X---@---H---M---
          |   |
1: -------X-------X-------M---
Bob's received message = 01
```

可以看出，Bob 收到的信息正是 Alice 通过超密集编码发送的信息。

7.4. 贝尔不等式测试

现在让我们来看另一个代码演示：贝尔不等式测试。在简要描述实验之后，我们将详细介绍一个用 Cirq 模拟它的完整程序。

贝尔不等式测试最容易通过一个涉及两名玩家，爱丽丝和鲍勃的合作游戏来理解，他们根据裁判的输入做出决策。爱丽丝和鲍勃在游戏过程中被分开（比如说坐在不同的房间里），不能在游戏中进行交流。在游戏的每一轮，裁判给爱丽丝发一个比特，叫做 x，给鲍勃发一个比特，叫做 y。根据比特的值，爱丽丝向裁判发送她自己的一个比特 a(x)，鲍勃也向裁判发送一个比特 b(y)。裁判观察这两个比特，然后决定爱丽丝和鲍勃是赢还是输。赢得这一轮的条件是：

a(x) □ b(y) = x * y

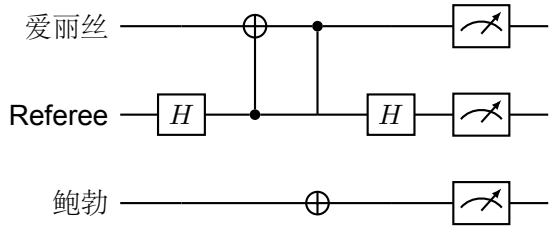
其中 □ 表示模 2 加法（或等效地，XOR）。爱丽丝和鲍勃的目标是尽可能多地赢得比赛。虽然他们在游戏中不能交流，但他们可以在游戏开始前见面并制定策略。一个示例策略可能是“爱丽丝总是回传 a(x)=x，鲍勃总是回传 b(y)=0”。由于 a(x) 和 b(y) 每个都有两个可能的值，爱丽丝和鲍勃可以实现四种可能的确定性策略。此外，由于整个游戏中只涉及四个比特，所以不难列举出所有可能的结果，并看看哪种策略让爱丽丝和鲍勃赢得最多的回合（或者等效地，以最高的概率赢得每一轮）。

通过分析每种策略，可以看到爱丽丝和鲍勃最多只能赢得 75% 的时间。实现这个胜率的策略是 1，其中 a(x)=b(y)=0，以及 4，其中 a(x)=b(y)=1。因此，最佳的经典策略最多只能赢得 75% 的时间。

当我们允许爱丽丝和鲍勃之间进行量子策略时，会发生有趣的现象。所谓量子策略，我们是指允许爱丽丝和鲍勃在他们的策略中使用纠缠作为资源。正如我们在本书中看到的，纠缠允许物理系统中比经典关联更强的关联。如果允许爱丽丝和鲍勃共享纠缠的量子比特，他们可以令人惊讶地以更高的概率赢得贝尔不等式测试游戏！最佳的量子策略实现了一个胜利概率为 $\cos^2(/8)$□□□85%□

这个游戏的量子策略如下面的电路图所示。这里，顶部（第一个）量子比特属于爱丽丝，从顶部数第三个量子比特属于鲍勃。电路的第一部分在爱丽丝和鲍勃的量子比特之间创建纠缠。然后，裁判向爱丽丝和鲍勃发送一个随机比特。在电路中，这是通过在一个“新鲜的量子比特”（处于 |0> 状态的量子比特）上执行哈达玛门操作来实现的，从而产生等概率的叠加态。爱丽丝和鲍勃然后执

行一个受控非门操作，然后记录结果。下面，我们展示了用 Cirq 设置这个电路的完整程序，并模拟贝尔不等式测试的量子策略。



这个代码会生成一个包含 Alice、Referee 和 Bob 的量子比特的电路，其中 Alice 和 Bob 的量子比特在开始时通过 CNOT 门纠缠。随后，Referee 的量子比特经过一个 H 门，然后对 Alice 的量子比特进行受控非门操作。最后，Referee 的量子比特再次经过一个 H 门，然后对 Alice 和 Bob 的量子比特进行测量。

Listing 7.3: 创建了一个贝尔测试电路，并打印出电路的结构

```
import numpy as np
import cirq

def main():
    # 创建电路
    circuit = make_bell_test_circuit()
    print('电路: ')
    print(circuit)

    # 运行模拟
    print()
    repetitions = 1000
    print(f'正在模拟{repetitions}次...')
    result = cirq.Simulator().run(program=circuit, repetitions=repetitions)

    # 收集结果
    a = np.array(result.measurements['a'][:, 0])
    b = np.array(result.measurements['b'][:, 0])
    x = np.array(result.measurements['x'][:, 0])
    y = np.array(result.measurements['y'][:, 0])

    # 计算获胜百分比
    outcomes = a ^ b == x & y
    win_percent = len([e for e in outcomes if e]) * 100 / repetitions

    # 打印数据
    print()
    print('结果: ')
    print('a:', bitstring(a))
    print('b:', bitstring(b))
    print('x:', bitstring(x))
    print('y:', bitstring(y))
    print('(a XOR b) == (x AND y):\n ', bitstring(outcomes))
    print(f'获胜率: {win_percent}%')

def make_bell_test_circuit():
    # Alice、Bob 和裁判员的量子比特
    alice = cirq.GridQubit(0, 0)
    bob = cirq.GridQubit(1, 0)
```

```
    alice_referee = cirq.GridQubit(0, 1)
    bob_referee = cirq.GridQubit(1, 1)

    circuit = cirq.Circuit()

    # 准备 Alice 和 Bob 之间的共享纠缠态
    circuit.append([
        cirq.H(alice),
        cirq.CNOT(alice, bob),
        cirq.X(alice)**-0.25,
    ])

    # 裁判员掷硬币
    circuit.append([
        cirq.H(alice_referee),
        cirq.H(bob_referee),
    ])

    # 玩家根据裁判员的硬币做一个 sqrt(X) 变换
    circuit.append([
        cirq.CNOT(alice_referee, alice)**0.5,
        cirq.CNOT(bob_referee, bob)**0.5,
    ])

    # 然后记录结果
    circuit.append([
        cirq.measure(alice, key='a'),
        cirq.measure(bob, key='b'),
        cirq.measure(alice_referee, key='x'),
        cirq.measure(bob_referee, key='y'),
    ])

    return circuit

def bitstring(bits):
    return ''.join('1' if e else '_' for e in bits)

if __name__ == '__main__':
    main()
```

这个程序的一个示例输出如下所示，我们只模拟了 75 次以更轻松地可视化输出。此程序的输出如下解释。第一行显示了 Alice 和 Bob 的比特串。这些是每个玩家发送回裁判员的比特。下一行显示发送给 Alice（x）和 Bob（y）的比特串。最后，获胜条件 a.x/˚b.y/ D xy 以每一轮的比特串形式呈现。获胜率是从此比特串中 1 的数量计算出来的，这些 1 表示获胜。

```
Simulating 75 repetitions...
Results
a: 1_1111_1_1111_11_1111_1_1_____1_1__
11_1__111_1_1_111_1111__11_1111_1_1____
b: 1_1__11_1_1_1_11__11__1_1_1_____1___
_11__11_1_1___11__1__1111_1_1___1__11_1
x: 11_11_1_1111_____1_11____11_1__111_1
1__1_11_1___11_111111111__11__1_111__11
y: _1_11111111__11_11__1__1111___1111__
1__111_1__111___1__11__1_111_11___1___1
(a XOR b) == (x AND y):
  1_11111_11__1111111111111_11111111111
```

```
  1_1111_11111111111111_11_111_11111__11
Win rate: 84.0%
```

如上所示，我们在这个例子中获得了 84% 的获胜率，这比完全经典策略所能达到的要高。请注意，使用更多的重复次数（即模拟更多的轮数）是推荐的，以便看到到达最优获胜率 $\cos^2(/8) = 85\%$□□□□□□

7.5. 概要

量子隐形传态、超密编码和贝尔不等式测试是量子处理器中最引人入胜的电路之一。现在让我们转向证明量子优势的量子算法的经典典范。

8

早期算法

本章将深入探讨一些基础的量子算法。我们将它们称为“基础算法”，这是因为它们在量子计算早期阶段就已开发，并且它们是首批证明量子计算能够带来显著计算速度提升的算法。虽然一些算法所需要的量子计算机现在还无法实现，但通过对它们的深入理解，我们能更好地预见未来可能达到的计算能力。同时，这些算法的一些变种已经被证明可以在无噪声 [124] 甚至有噪声的条件下 [125] 在近期量子计算机中展现优势。

	早期量子算法	**NISQ 时代算法**
代表性算法	Shor 的算法、Grover 的算法	QAOA、量子机器学习
硬件要求	需要大规模、完全误差纠正的量子计算机	可在存在噪声的量子设备上运行
理论 vs 实用	更侧重于理论，显示量子优势	更强调实用性和在现有硬件上的可行性
问题类型	整数因子分解、无序搜索	组合优化问题、机器学习任务

表 8.1: 早期量子算法和 NISQ 时代算法的比较

8.1. 代码遍历

首先，我们将关注一类被称为“黑箱”或“查询模型”的量子算法。在量子计算中，oracle（有时被称为黑箱）是一种抽象的概念，通常被用来描述一个未知的函数或过程。这个函数或过程能够对某个特定的输入进行某种操作或返回某种输出。在许多量子算法中，包括 Deutsch-Jozsa 算法和 Grover's 搜索算法，oracle 是一个关键的组成部分。这些算法的设计思路是，给定一个 oracle，我们可以利用量子系统的特性（如叠加和纠缠）来比经典方法更有效地查询这个 oracle。举个例子，在 Grover's 搜索算法中，我们有一个未知的函数 f，它在某个特定的输入 x 下返回 1，而在所有其他输入下返回 0。我们的目标是找到这个特定的输入 x。在这种情况下，oracle 就是这个函数 f，我们通过反复查询这个 oracle 并利用量子干涉效应，可以在 $O(\sqrt{N})$ 的时间复杂度内找到这个特定的输入 x，其中 N 是搜索空间的大小。这比经典方法（需要 $O(N)$ 的时间复杂度）快很多。需要注意的是，oracle 在这里更像是一种理论工具，而不是实际可实施的操作。在实际的量子计算实

类别	问题/算法	使用的范例	硬件	模拟匹配
反函数计算	Grover's 算法	GO	QX4	中等
	Bernstein-Vazirani	n.a.	QX4, QX5	高
数论应用	Shor's 分解算法	QFT	QX4	中等
代数应用	线性系统	HHL	QX4	低
	矩阵元素群表示法	QFT	ESSEX	低
	矩阵乘积验证	GO	n.a.	n.a.
	子组同构	QFT	none	n.a.
图形应用	量子随机漫步	n.a.	VIGO	中等-低
	最小生成树	GO	QX4	中等-低
	最大流	GO	QX4	中等-低
	近似量子算法	SIM	QX4	高
学习应用	量子主成分分析 (PCA)	QFT	QX4	中等
	量子支持向量机 (SVM)	QFT	none	n.a.
	分区函数	QFT	QX4	中等-低
量子模拟	薛定谔方程模拟	SIM	QX4	低
	横向伊辛模型模拟	VQE	none	n.a.
量子实用工具	状态准备	n.a.	QX4	中等
	量子断层成像	n.a.	QX4	中等
	量子纠错	n.a.	QX4	中等

表 8.2: 研究的量子算法概览。包括 Grover 算子（GO），量子傅立叶变换（QFT），Harrow-Hassidim-Lloyd 算法（HHL），变分量子本征值求解器（VQE），以及直接哈密顿模拟（SIM）。模拟匹配列显示了硬件量子结果与模拟器结果结果的匹配程度。

现中，我们需要找到实现这个 oracle 的量子电路。对于一些特定的问题，我们可能已经知道如何构造这样的电路，但对于一般的问题，构造 oracle 可能是一个困难的问题。

在这些模型中，有一个基础函数存在，但是我们并不了解它的内部运作。然而，我们可以构造一个所谓的'oracle' 函数，通过查询特定的输入-输出对来对其进行探索。更精确地说，我们可以在量子寄存器中使用特定的输入来查询'oracle' 函数，并以可逆的方式将'oracle' 函数的输出写入寄存器中。也就是，我们可以访问一个'oracle' O_f，满足以下的性质：

$$O_f |x\rangle = |x \oplus f(x)\rangle \tag{8.1}$$

其中 $\oplus$ 代表模 2 加法。显然，O_f 是可逆的，因为它是自反的。这可能一开始看起来有些令人困惑——我们如何构造一个实现 O_f 的电路？我们如何确定它是一个高效的电路？

查询模型的量子算法值得我们深思的一个原因是它们为步骤（门）的数量提供了一个下界。每次查询都至少占用算法的一步，因此，如果不能通过查询高效地执行，那么通过门来高效执行也绝对不可能。因此，查询模型可以用于证明某个快速量子算法的不存在性。

然而，查询模型还可以用于证明相对于'oracle' 的快速量子算法的存在性。我们可以允许量子计算机和经典计算机访问相同的'oracle'，并比较两者的性能。我们可以证明在经典和量子情况下查询次数的下界或精确值，从而有可能证明相对于'oracle' 的计算优势。例如，Deutsch 的算法和 Bernstein-Vazirani 算法都是具有可证明的相对加速性质的量子算法。

最后，如果我们能够找到一种方法以多项式方式扩展输入寄存器的数量来实现 oracle，那么我们就可以找到"真正的"（即非相对）量子加速。这就是量子因下面的算法可以与现有的计算机算法相比，具有量子优势：

8.2. Deutsch-Jozsa 算法

Deutsch 的算法是首个明确展示量子计算相较于经典计算优势的算法。在 Deutsch 问题中，我们获得了一个计算一位布尔函数的黑箱。也就是，一个接收一位输入并输出一位的函数。我们可以表示函数 f 为

$$f : \{0,1\} \to \{0,1\} \tag{8.2}$$

这是一个数学函数的表示方式，这个函数名为 f，它的定义域是集合 {0,1}，值域也是集合 {0,1}。

我们可以想象，比如，正如 David Deutsch 所指出的，黑箱函数计算的是某些复杂的函数，比如路由算法，输出（0 或 1）指示选择了哪条路线 [66]。

在这种情况下，函数 f 接受一个二进制输入（0 或 1）并返回一个二进制输出（0 或 1）。这样的函数在计算机科学中经常出现，尤其是在逻辑运算和布尔代数中。对于这样的函数，存在四种可能的映射，分别为：

- 无论输入是 0 还是 1，函数 f 都返回 0（也就是常数函数 f0）；
- 无论输入是 0 还是 1，函数 f 都返回 1（也就是常数函数 f1）；
- 函数 f 返回输入的值（也就是恒等函数 fx，如果输入是 0，输出是 0；如果输入是 1，输出是 1）；
- 函数 f 返回输入值的反（也就是非函数 fxN，如果输入是 0，输出是 1；如果输入是 1，输出是 0）。

如果你用经典的工具来处理这个问题，你至少需要查询函数两次：首先查看输入为 0 时的输出，然后查看输入为 1 时的输出。David Deutsch 的重大发现是，你在量子计算机上只需要查询一次！原始的 Deutsch 算法处理的是一位布尔 oracle[65] 的情况，而 Deutsch-Jozsa（DJ）算法将这种方法推广，以处理 n 个输入的布尔函数 [67]。不难看出，用经典的工具，你至少需要查询 n 位布尔函数 n 次。而 DJ 算法只需要查询一次就能解决这个问题。

”Deutsch-Jozsa 证明的量子优势在经典情况下，为了区分常数函数[1]和平衡函数[2]，必须至少查询两次一位布尔函数。对于一个 n 位的布尔函数，必须查询 n 次。然而，在量子计算机上使用 DJ 算法，我们只需查询一次。”

我们现在转向量子方法来解决这个问题。在上文中，我们描述了一个非可逆的经典位函数，例如常数函数 f_0 和 f_1。也就是说，知道输出的值并不能让我们唯一确定输入的值。然而，为了在量子计算机上运行，我们需要使这个计算可逆。将一个经典的非可逆函数变为可逆的一个技巧是在额外的寄存器中计算值（或者等价地，在额外的寄存器中存储函数的输入）。假设我们有一个 n 位

[1]常数函数是一种特殊的函数，它的特点是无论输入是什么，输出都是固定的常数值。换句话说，常数函数的函数值（即输出）与输入无关。

举一个简单的例子，我们可以定义一个常数函数 $f(x) = 5$。无论 x 的值是多少（比如 1、2、3 等等），$f(x)$ 的值总是 5。

在你给出的布尔函数的例子中，常数函数 f0 和 f1 就是指无论输入是 0 还是 1，输出都是 0 或 1。对于函数 f0，无论输入是 0 还是 1，输出都是 0。对于函数 f1，无论输入是 0 还是 1，输出都是 1。

[2]在布尔函数的语境中，平衡函数是指其输出在所有可能输入上的 0 和 1 出现的次数相同。也就是说，对于给定的所有输入，输出为 0 的次数与输出为 1 的次数相等。

举例来说，在你提到的一位输入布尔函数的情况中，有两个平衡函数：函数 fx 和函数 fxN。函数 fx 当输入为 0 时输出 0，输入为 1 时输出 1；函数 fxN 正好相反，输入为 0 时输出 1，输入为 1 时输出 0。你可以看到，在所有可能的输入（0 和 1）中，这两个函数的输出 0 和 1 各出现一次，所以它们是平衡的。

这个概念可以扩展到具有多位输入的布尔函数。对于一个 n 位输入的布尔函数，如果在所有的 2^n 个可能输入中，函数输出 0 和 1 的次数都是 $2^{(n-1)}$，那么我们就称这个函数是平衡的。

输入 x，我们正在计算一个（可能非可逆的）布尔函数 fx。然后我们可以通过作用在 $n+1$ 个量子比特上的酉矩阵 U_f 来实现这个计算：

$$U_f(|x\rangle\,|y\rangle) := |x\rangle\,|y \oplus f(x)\rangle \tag{8.3}$$

在这里，$|x\rangle$ 和 $|y\rangle$ 是量子态，也就是量子比特（或称为 qubit）的状态。符号 $\oplus$ 表示模 2 加法，也被称为异或（XOR）操作。函数 U_f 是一个酉矩阵，它将一个量子态映射到另一个量子态。这里，U_f 操作在两个输入态 $|x\rangle$ 和 $|y\rangle$ 上，并生成新的量子态 $|x\rangle\,|y \oplus f(x)\rangle$。在这个表达式中，$|x\rangle$ 保持不变，而 $|y\rangle$ 被转换为 $|y \oplus f(x)\rangle$。换句话说，$|y\rangle$ 的新状态是其原始状态和函数 f 在 $|x\rangle$ 上的值的模 2 加法（或异或操作）。这个操作是可逆的，这是量子计算的关键特性。它允许我们在执行复杂计算时保留所有中间步骤的信息，这是经典计算机无法做到的。

这个算法中的一个核心思想是，将在与计算基础不同的基础上进行测量。如果我们在计算基础（例如，z 基础）上进行测量，那么我们将无法获得量子优势，因为我们有两个基础状态，$|0\rangle$ 和 $|1\rangle$，它们对应于经典的比特，0 和 1。使这个算法工作的一个技巧是我们将在 Hadamard 基态中进行测量，这是 $|0\rangle$ 和 $|1\rangle$ 的叠加态 [137]。图 8.2 展示了 DJ 的电路图。

让我们看看如何在 Cirq 中实现这些函数。f_0 执行了变换

$$|00\rangle \to |00\rangle\ \ |01\rangle \to |01\rangle\ \ |10\rangle \to |10\rangle\ \ |11\rangle \to |11\rangle$$

这只是恒等变换，即一个空的电路。f_1 执行了变换

$$|00\rangle \to |01\rangle\ \ |01\rangle \to |00\rangle\ \ |10\rangle \to |11\rangle\ \ |11\rangle \to |10\rangle$$

这是对第二个量子比特的比特翻转门。

为了理解这个新定义的可逆运算符是如何工作的，我们将以计算 $U_{f_x}(|0\rangle\,|0\rangle)$ 为例进行演示。其中 $|00\rangle$ 是 $|0\rangle\,|0\rangle$ 的简写。那么，根据定义：

$$U_{f_x}(|00\rangle) = U_{f_x}(|0\rangle\,|0\rangle) := |0\rangle\,|0 \oplus f_x(0)\rangle = |0\rangle\,|0 \oplus 1\rangle = |0\rangle\,|1\rangle \tag{8.4}$$

这是一个量子操作符（U_{f_x}）作用在一个特定的两量子比特态（$|00\rangle$）上的例子。

在这个例子中，f_x 是一个函数，其定义域和值域都是二进制集合 0,1。在这个例子中，$f_x(0) = 1$。

操作符 U_{f_x} 定义为：$U_{f_x}(|x\rangle\,|y\rangle) = |x\rangle\,|y \oplus f_x(x)\rangle$，这里 $\oplus$ 表示模 2 加法（也被称为异或操作）。

所以，当我们将 U_{f_x} 应用到态 $|00\rangle$ 上，我们得到：$U_{f_x}(|00\rangle) = U_{f_x}(|0\rangle\,|0\rangle) = |0\rangle\,|0 \oplus f_x(0)\rangle = |0\rangle\,|0 \oplus 1\rangle = |01\rangle$。

这表示在第二个量子比特上进行了一个比特翻转操作。原始的态是 $|00\rangle$，结果的态是 $|01\rangle$。

对每个 $|0\rangle\,|1\rangle$，$|1\rangle\,|0\rangle$ 和 $|1\rangle\,|1\rangle$ 执行 U_{f_x}；然后让我们验证 f_x 是否执行了这样的变换：

$$|00\rangle \to |00\rangle\ \ |01\rangle \to |01\rangle\ \ |10\rangle \to |11\rangle\ \ |11\rangle \to |10\rangle$$

这不过是从第一个量子比特到第二个量子比特的 CNOT（受控非）操作。最后，fx_N 执行了以下变换：

$$|00\rangle \to |01\rangle\ \ |01\rangle \to |00\rangle\ \ |10\rangle \to |10\rangle\ \ |11\rangle \to |11\rangle$$

这是一个从第一个量子比特到第二个量子比特的 CNOT 操作，然后在第二个量子比特上进行位翻转。我们可以将这些函数封装到一个字典中，该字典将 oracle 的名字映射到电路中执行此函数所需的操作：

```
# 导入Cirq库
import cirq
# 获取两个量子比特，分别为数据量子比特和目标量子比特
q0, q1 = cirq.LineQubit.range(2)
# oracle函数的字典
oracles = {'0': [], '1': [cirq.X(q1)], 'x': [cirq.CNOT(q0, q1)],
        'notx': [cirq.CNOT(q0, q1), cirq.X(q1)]}
```

图 8.1: 实现 oracle 函数的 Python 代码

上述代码导入了 Cirq 库，这是一个用于模拟和运行量子电路的 Python 库。然后，你定义了两个量子比特，分别是数据量子比特和目标量子比特。接着，你创建了一个字典，其中的键是 oracle 的名称，值是用于执行相应 oracle 函数的 Cirq 操作的列表。

这里是具体的解释：

- '0'：这个 oracle 函数总是返回 0，不需要做任何操作，所以对应的列表为空。
- '1'：这个 oracle 函数总是返回 1，这需要对第二个量子比特（$q1$）执行位翻转（X 门）操作。
- 'x'：这个 oracle 函数返回输入值（即，如果输入为 0 则返回 0，输入为 1 则返回 1）。这需要对 $q0$ 和 $q1$ 执行 CNOT 门操作，使 $q1$ 的值变为 $q0$ 和 $q1$ 的值的异或。
- 'notx'：这个 oracle 函数返回输入值的非（即，如果输入为 0 则返回 1，输入为 1 则返回 0）。这需要对 $q0$ 和 $q1$ 执行 CNOT 门操作，然后对 $q1$ 执行位翻转（X 门）操作。

如果想查看执行这些 oracle 函数后量子比特的状态，你需要创建一个量子电路，将这些操作添加到电路中，然后运行电路。

让我们现在来看看 Deutsch 的算法。假设我们可以访问之前定义的可逆 oracle 函数。通过对我们不可逆的经典函数的类似论证，你可以证明你无法通过只使用一次这个 oracle 来区分平衡函数和常数函数。但现在我们可以提出这样的问题：如果我们被允许在叠加状态中查询这个函数，即，如果我们可以使用量子计算的能力会怎样？

Deutsch 能够证明，你可以使用量子计算机，只通过单次查询函数，就解决这个问题。要理解这是如何工作的，我们需要两个简单的洞见。假设我们将第二个量子比特准备在叠加态中：

$$|-\rangle = \frac{1}{\sqrt{2}}(|1\rangle - |1\rangle) \tag{8.5}$$

并应用 oracle。利用操作符 U_f 的线性性得到第二个等式，并通过观察得到第三个等式，我们可以验证如下：

$$U_f |x\rangle |-\rangle = U_f |x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle), \tag{8.6}$$

$$= |x\rangle \frac{1}{\sqrt{2}}(|f(x)\rangle - |f(x) \oplus 1\rangle), \tag{8.7}$$

$$= (-1)^{f(x)} |x\rangle |-\rangle . \tag{8.8}$$

其中 f 是一个二进制函数。这个表达式描述的是量子门 U_f 在量子态 $|x\rangle|-\rangle$ 上的效果。在这里，$|-\rangle$ 是一个量子叠加态，表示为 $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$，也就是说这是一个包含了所有可能结果的叠加态。

逐步分析这个过程。

首先，U_f 作用于 $|x\rangle|-\rangle$：

$U_f|x\rangle|-\rangle = U_f|x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$

然后，由 U_f 的定义，我们有：

$U_f|x\rangle|0\rangle = |x\rangle|f(x)\rangle$

$U_f|x\rangle|1\rangle = |x\rangle|f(x)\oplus 1\rangle$

因此，当 U_f 作用于 $|x\rangle|-\rangle$，我们得到：

$U_f|x\rangle|-\rangle = \frac{1}{\sqrt{2}}(|x\rangle|f(x)\rangle - |x\rangle|f(x)\oplus 1\rangle)$

接下来，我们注意到当 $f(x)=0$ 时，$|f(x)\rangle = |0\rangle$ 和 $|f(x)\oplus 1\rangle = |1\rangle$。反之，当 $f(x)=1$ 时，$|f(x)\rangle = |1\rangle$ 和 $|f(x)\oplus 1\rangle = |0\rangle$。所以，我们可以将量子态 $|f(x)\rangle - |f(x)\oplus 1\rangle$ 写成 $(-1)^{f(x)}|-\rangle$。

因此，我们最终得到：

$U_f|x\rangle|-\rangle = (-1)^{f(x)}|x\rangle|-\rangle$

这是所谓的相位反馈技巧。通过将 U_f 作用在处于叠加态的目标量子比特上，函数的值最终出现在全局相位中，从而反馈我们需要的关于函数是常数还是平衡的信息；这些信息被编码在相位中。

我们如何利用这一点来区分常数和平衡函数呢？请注意，对于常数函数，应用的相位对于所有的输入 $|x\rangle$ 是相同的，而对于平衡函数，相位对于 x 的每一个值都是不同的。为了使用相位反馈技巧对每个神谕进行操作，我们对第一个量子比特应用以下变换：

$$f_0 \to I$$

$$f_1 \to -I$$

$$f_x \to Z$$

$$f_{\bar{x}} \to -Z$$

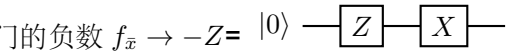
这里的 I 是单位门，Z 是 Pauli Z 门，它们都是常用的量子门。

- 对于常数函数 f_0 和 f_1，无论输入 x 的值是什么，输出都是固定的 0 或 1。这些函数被映射到单位门 I 或者其负数 $-I$ 上。单位门不改变输入态，因此这种映射能够保持输入的状态，符合常数函数的特性。
- 对于平衡函数 f_x 和 $f_{\bar{x}}$，对于输入 x，输出是 x 或者 x 的补。这些函数被映射到 Z 门或者其负数 $-Z$ 上。Z 门会将基态 $|0\rangle$ 和 $|1\rangle$ 分别映射到 $|0\rangle$ 和 $-|1\rangle$，这种性质与平衡函数的特性相符。

U_{f_0} 对应的是单位门 I，其对量子态没有任何改变。U_{f_0} = $|0\rangle$ ——

U_{f_1} 对应的是 -I，即单位门的负数 U_{f_1} = $|0\rangle$ —[X]—

Z 门：U_{f_x} = $|0\rangle$ —[Z]—

$f_{\bar{x}}$ 对应的是 -Z，即 Z 门的负数 $f_{\bar{x}} \to -Z$= $|0\rangle$ 

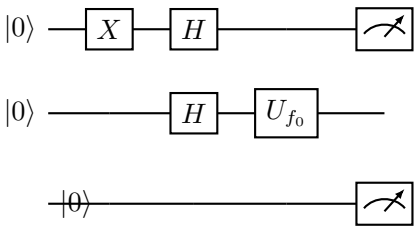
现在我们只需要区分第一个量子比特上的单位门（identity gate）和 Z 门；我们可以通过回忆以下内容来实现这一点：

$$HZH = X$$

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

这意味着我们可以通过在相位翻转前后应用 Hadamard 门，将相位翻转转化为比特翻转。如果我们观察常数函数和平衡函数，我们会发现常数函数对应的量子门与单位门 (I) 成比例，而平衡函数对应的量子门与 Pauli-X 门 (X) 成比例。如果我们将基态 $|0\rangle$ 输入到这个寄存器中，那么在常数函数的情况下，我们只会观察到基态 $|0\rangle$，而在平衡函数的情况下，我们会观察到基态 $|1\rangle$。换句话说，我们可以通过仅进行一次神谕查询来区分常数函数和平衡函数。

```
# 导入 Cirq 库
import cirq

# 获取两个量子比特，分别作为数据比特和目标比特
q0, q1 = cirq.LineQubit.range(2)

# 神谕的字典表示
oracles = {'0': [], '1': [cirq.X(q1)], 'x': [cirq.CNOT(q0, q1)], 'notx': [cirq.CNOT(q0, q1), cirq.X
    (q1)]}

def deutsch_algorithm(oracle):
    """根据实现神谕的操作生成 Deutsch 算法的电路."""
    yield cirq.X(q1)
    yield cirq.H(q0), cirq.H(q1)
    yield oracle
    yield cirq.H(q0)
    yield cirq.measure(q0)

# 显示每个神谕的电路
for key, oracle in oracles.items():
    print('神谕 {} 的电路...'.format(key))
    print(cirq.Circuit.from_ops(deutsch_algorithm(oracle)), end="\n\n")

# 获取一个模拟器
simulator = cirq.Simulator()

# 对每个神谕执行电路以区分常数和平衡函数
for key, oracle in oracles.items():
    result = simulator.run(cirq.Circuit.from_ops(deutsch_algorithm(oracle)), repetitions=10)
    print('神谕：{:<4} 结果：{}'.format(key, result))

\caption{生成 Deutsch 算法}
\label{fig:Deutsch_code}
```



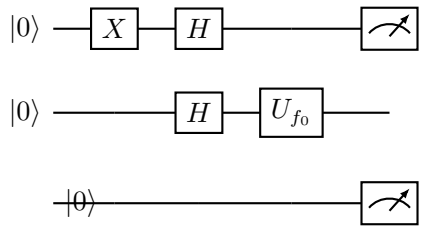
将 Deutsch 问题扩展到具有 n 个布尔输入的布尔函数，而不仅仅是前面提到的单输入函数。我们看到，通过使用 Deutsch 算法，我们可以将经典的两次查询速度加倍，从经典的两次查询减少到量子的一次查询。

如果我们能够仅通过一次查询 n 位神谕，并确定函数是常数还是平衡的，那么我们的时间复杂度将为 $O(1)$，这比 $O(n)$ 的查询要快得多。对于单比特输入的布尔函数，它们要么是常数函数，要么是平衡函数。对于具有两个输入比特的布尔函数，存在两个常数函数 $f(x_0, x_1) = 0$ 和 $f(x_0, x_1) = 1$，同时存在 $\binom{4}{2} = 6$ 个平衡函数。下面的代码为你提供了查询这些函数的两个操作。

简而言之，这段描述指的是通过一次查询就能确定布尔函数是常数还是平衡的情况下，时间复杂度为 $O(1)$，相对于需要 $O(n)$ 次查询的情况而言，具有显著的加速效果。对于单比特输入的布尔函数，它们要么是常数函数，要么是平衡函数。对于两个输入比特的布尔函数，有两个常数函数和六个平衡函数。代码提供了查询这些函数所需的两个操作。

```
"""Deustch-Jozsa algorithm on three qubits in Cirq."""
# 导入 Cirq 库
import cirq

# 获取三个量子比特，其中两个是数据比特，一个是目标比特
q0, q1, q2 = cirq.LineQubit.range(3)

# 常数函数的神谕
constant = ([], [cirq.X(q2)])

# 平衡函数的神谕
balanced = (
    [cirq.CNOT(q0, q2)],
    [cirq.CNOT(q1, q2)],
    [cirq.CNOT(q0, q2), cirq.CNOT(q1, q2)],
    [cirq.CNOT(q0, q2), cirq.X(q2)],
    [cirq.CNOT(q1, q2), cirq.X(q2)],
    [cirq.CNOT(q0, q2), cirq.CNOT(q1, q2), cirq.X(q2)]
)

def your_circuit(oracle):
    """生成一个包含三个量子比特的 Deustch-Jozsa 算法的电路."""
    # 相位反馈技巧
    yield cirq.X(q2), cirq.H(q2)
    # 输入比特的等概叠加态
    yield cirq.H(q0), cirq.H(q1)
    # 查询函数
    yield oracle
    # 干涉以获取结果，将最后一个比特设为 |1>
    yield cirq.H(q0), cirq.H(q1), cirq.H(q2)
    # 最后的 OR 门将结果放入最终比特
    yield cirq.X(q0), cirq.X(q1), cirq.CCX(q0, q1, q2)
    yield cirq.measure(q2)
```

```
34
35 # 获取一个模拟器
36 simulator = cirq.Simulator()
37
38 # 执行常数值函数的电路
39 print('对常数值函数的结果')
40 for oracle in constant:
41     result = simulator.run(cirq.Circuit.from_ops(your_circuit(oracle)), repetitions=10)
42     print(result)
43
44 # 执行平衡函数的电路
45 print('对平衡函数的结果')
46 for oracle in balanced:
47     result = simulator.run(cirq.Circuit.from_ops(your_circuit(oracle)), repetitions=10)
48     print(result)
```



现在我们可以看到如何查询一个具有 n 位布尔输入的神谕，以检查它是常数函数还是平衡函数。

8.2.1. The Bernstein-Vazirani 算法

现在让我们转向本书之前讨论过的贝恩斯坦-瓦齐拉尼（Bernstein-Vazirani，简称 BV）算法。与 DJ 算法类似，BV 算法的目标也是确定一个黑盒布尔函数的性质。尽管 DJ 算法展示了量子计算相对于经典计算的优势，但如果我们允许存在一定的错误率，那么这种优势会消失：经典和量子方法的时间复杂度都在 $O(1)$ 的量级。

BV 算法是第一个展示了即使在允许存在误差的情况下，量子计算和经典计算之间存在明显差异的算法，也就是真正的非确定性加速。以下是 BV 问题的描述：给定一个未知的 n 个输入的函数：$f(x_{n-1}, x_{n-2}, ..., x_1, x_0)$；令 a 是一个未知的非负整数，小于 2^n。定义 $f(x)$ 为将任意的整数 x 与 a 进行模 2 求和。因此，函数的输出为：

$$a \oplus x = a_0x_0 \oplus a_1x_1 \oplus a_2x_2 \oplus ... \oplus a_nx_n$$

要求在一次查询中找到 a 的值。

与 DJ 算法一样，我们准备了两组量子比特的状态：数据寄存器比特和目标比特。数据寄存器比特的初始状态设置为 $|0\rangle$，而目标比特设置为 $|1\rangle$。然后，我们对两组量子比特都应用 Hadamard 门，使它们处于叠加态。对数据寄存器比特应用的 Hadamard 门将我们准备好在 X 基础上进行测量。

我们接下来对数据寄存器比特应用幺正门 U_f，然后对这些比特应用 H 门，并进行测量。由于我们只应用了一次 U_f，所以 BV 算法的时间复杂度是 $O(1)$。

在 Bernstein-Vazirani (BV) 算法中，我们通过应用 U_f 幺正门对数据寄存器进行变换，该幺正门根据函数 f 的定义进行操作。然后，我们对数据寄存器的比特应用 H 门，将它们置于叠加态。最后，我们进行测量以获取结果。

由于 BV 算法中只进行了一次 U_f 操作，所以它的时间复杂度是 $O(1)$，即常数时间复杂度。这意味着无论输入规模的大小，算法的执行时间都是固定的，因此具有高效性。

```
# 导入所需库
import random
import cirq

def main():
    """执行贝恩斯坦-瓦齐拉尼算法。"""
    # 量子比特数量
    qubit_count = 8
    # 从电路中采样的次数
    circuit_sample_count = 3
    # 选择要使用的量子比特
    input_qubits = [cirq.GridQubit(i, 0) for i in range(qubit_count)]
    output_qubit = cirq.GridQubit(qubit_count, 0)
    # 为神谕选择系数并创建查询电路
    secret_bias_bit = random.randint(0, 1)
    secret_factor_bits = [random.randint(0, 1) for _ in range(qubit_count)]
    oracle = make_oracle(input_qubits, output_qubit, secret_factor_bits, secret_bias_bit)
    print('秘密函数: \nf(x) = x*<{}> + {} (mod 2)'.format(
        ', '.join(str(e) for e in secret_factor_bits),
        secret_bias_bit))

    # 将神谕嵌入到特定的量子电路中，仅查询一次
    circuit = make_bernstein_vazirani_circuit(input_qubits, output_qubit, oracle)
    print('\n电路: ')
    print(circuit)

    # 从电路中采样几次
    simulator = cirq.Simulator()
    result = simulator.run(circuit, repetitions=circuit_sample_count)
    frequencies = result.histogram(key='result', fold_func=bitstring)
    print('\n采样结果: \n{}'.format(frequencies))

    # 检查是否找到了秘密值
    most_common_bitstring = frequencies.most_common(1)[0][0]
    print('\n最常匹配秘密因子: \n{}'.format(
        most_common_bitstring == bitstring(secret_factor_bits)))

def make_oracle(input_qubits, output_qubit, secret_factor_bits, secret_bias_bit):
    """实现函数 f(a) = a*factors + bias (mod 2) 的门操作。"""
    if secret_bias_bit:
        yield cirq.X(output_qubit)
    for qubit, bit in zip(input_qubits, secret_factor_bits):
        if bit:
            yield cirq.CNOT(qubit, output_qubit)

def make_bernstein_vazirani_circuit(input_qubits, output_qubit, oracle):
    """使用一次查询解决 f(a) = a*factors + bias (mod 2) 的问题。"""
    c = cirq.Circuit()
    # 初始化量子比特
    c.append([
        cirq.X(output_qubit),
        cirq.H(output_qubit),
        cirq.H.on_each(*input_qubits),
    ])
```

```
    # 查询神谕
    c.append(oracle)
    # 在 X 基础上进行测量
    c.append([
        cirq.H.on_each(*input_qubits),
        cirq.measure(*input_qubits, key='result')
    ])
    return c

def bitstring(bits):
    """将位列表转换为二进制字符串。"""
    return ''.join(str(int(b)) for b in bits)

if __name__ == '__main__':
    main()
```

在这里，我们通过应用 X 门将目标（或输出）寄存器初始化为 $|1\rangle$，并通过应用 H 门将数据寄存器比特从 $|0\rangle$ 状态转换到 $|+\rangle/|-\rangle$ 基态。然后，我们查询神谕（oracle），对每个输入比特应用 H 门，并测量这些输入比特。这样，我们就得到了我们所寻找的答案 a，并且只需进行一次查询。因此，无论输入有多少个，我们都可以在 $O(1)$ 的时间复杂度内执行这个算法。

在 Bernstein-Vazirani (BV) 算法中，通过初始化目标比特为 $|1\rangle$ 并将数据寄存器比特设置为 $|+\rangle/|-\rangle$，我们对函数的输入进行了叠加态的准备。然后，我们应用了查询神谕的操作，根据函数的定义对输入进行变换，并在 X 基础上进行测量以获得函数的隐藏参数 a。由于我们只进行了一次查询，所以无论输入的规模如何，算法的时间复杂度都是 $O(1)$，即常数时间复杂度。这意味着无论输入有多少个，算法的执行时间都是固定的，因此具有高效性。

```
"""
=== EXAMPLE OUTPUT for BV ===
Secret function:
f(x) = x*<0, 1, 1, 1, 0, 0, 1, 0> + 1 (mod 2)
Sampled results:
Counter({' 01110010' : 3})
Most common matches secret factors:
True
"""
```

8.3. Simon's Problem

紧随 BV 的结果之后，Daniel Simon[3] 展示了一种能力，即在量子计算机上，他能够比在经典计算机上更快地确定函数的周期性。

让我们回顾一下，一个函数可以将两个不同的输入映射到相同的输出，但不能将同一个输入映射到两个不同的输出。换句话说，2:1 是可以接受的，但 1:2 则不行。例如，函数 $fx = x^2$ 将每个输入都平方，实际上是一个函数；两个不同的输入，即 1 和-1，都映射到 1，即，fx 是 2:1。

如果我们确定函数是 2:1 类型，那么我们接下来的挑战就是研究函数的周期；这是 Simon 问题的核心目标。以下是用更正式的语言概述的问题：

[3]Daniel R. Simon 是一位著名的计算机科学家，他在量子计算领域有着重要的贡献。他最为人所知的贡献是提出了被称为"Simon's problem"的问题，这个问题是量子算法研究的重要基础。

Simon 的问题阐述了一种特定的黑箱函数场景，即对于一个特定的二进制函数，我们需要确定是否存在两个不同的输入映射到相同的输出，以及这种情况发生的频率。Simon 证明了，对于这个问题，量子计算机在理论上能够比经典计算机更快地找到答案。

Simon 的这项工作对于量子计算的发展产生了深远的影响。特别是，Simon 的问题直接启发了 Peter Shor 发展出他的著名的 Shor 的量子因式分解算法，这个算法在理论上能够威胁到当前主要的公钥密码体系的安全性。

这个问题考虑了一个实现函数映射的预言器，该函数将一个 n 位字符串映射到一个 m 位字符 $f:\{0,1\}^n \to \{0,1\}^m$，其中 $m \geqq n$。我们假设函数 f 是 $1:1$ 类型的函数（每个输入给出不同的输出）或 $2:1$ 类型的函数（两个输入给出相同的输出），并且非零周期 s 属于 $\{0,1\}^n$，使得对于所有的 x 和 x_0，我们有 $fx = fx_0$，当且仅当 $x_0 = x \oplus s$，其中 $\oplus$ 对应于模 2 的加法。问题是确定函数 f 的类型，如果它是 $2:1$，还要确定周期 s。[62]

Simon 算法（Simon's algorithm）Simon 算法可以解决的问题与 Deutsch-Jozsa 算法相同，但 Simon 算法通常被认为是比 Deutsch-Jozsa 算法更强大的算法。它可以在 O(n)次查询内解决问题，而经典算法需要 2^n-1□□□□$Grover$□□□□□$Grover's search algorithm$□$Grover$□□□□□□□□□□□□□□□□□□□□□□

8.3.1. The Deutsch-Jozsa Algorithm

Deutsch-Jozsa 问题 [?]：对于 $N = 2^n$，我们被给定一个 $x \in 1^N$，其中要么 （1）所有的 x_i 都有相同的值（"常数"），要么 （2）$N/2$ 个 x_i 是 0，另外 $N/2$ 个是 1（"平衡的"）。目标是找出 x 是常数还是平衡的。

Deutsch 和 Jozsa 的算法如下。我们从 n 个量子比特的零状态 $|0^n\rangle$ 开始，对每个量子比特应用 Hadamard 变换，应用一个查询（用它的 ± 形式表示），再次对每个量子比特应用 Hadamard 变换，然后测量最终状态。作为一个幺正变换，该算法将是 $H^{\otimes n}O_{x,\pm}H^{\otimes n}$。我们在图 8.2中画出了相应的量子电路（时间再次从左到右推进）。请注意，进入查询的电线数为 n，而不是 N；这个电线序列上的基态指定了一个 n 位地址。

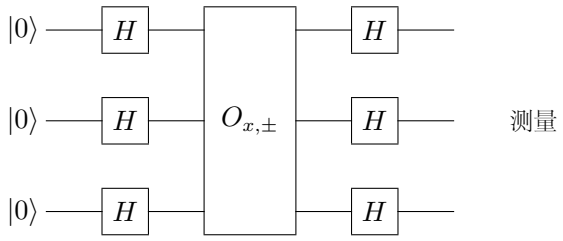


图 **8.2:** The Deutsch-Jozsa algorithm for $n = 3$

让我们跟随这些操作的状态。最初我们有状态 $|0^n\rangle$。根据第**??**页上的公式（**??**），在第一次哈达玛尔变换之后，我们得到了所有 i 的均匀叠加:

$$\frac{1}{\sqrt{2^n}} \sum_{i \in 1^n} |i\rangle \,. \tag{8.9}$$

$O_{x,\pm}$ 查询将其变为

$$\frac{1}{\sqrt{2^n}} \sum_{i \in 1^n} (-1)^{x_i} |i\rangle \,. \tag{8.10}$$

再次应用第二批哈达玛尔变换（由公式（**??**）给出）得到最后的叠加态

$$\frac{1}{2^n} \sum_{i \in 1^n} (-1)^{x_i} \sum_{j \in 1^n} (-1)^{i \cdot j} |j\rangle \,, \tag{8.11}$$

其中 $i \cdot j = \sum_{k=1}^{n} i_k j_k$ 与之前相同。由于 $i \cdot 0^n = 0$ 对于所有 $i \in 1^n$ 成立，我们可以看到在最

终叠加态中 $|0^n\rangle$ 的幅度是

$$\frac{1}{2^n}\sum_{i\in 1^n}(-1)^{x_i}=\begin{cases}1 & \text{如果对于所有 } i \text{ 都有 } x_i=0,\\ -1 & \text{如果对于所有 } i \text{ 都有 } x_i=1,\\ 0 & \text{如果 } x \text{ 是平衡的。}\end{cases} \tag{8.12}$$

因此，如果 x 是常数，最后的观测结果将得到 $|0^n\rangle$，如果 x 是平衡的，将得到其他状态。因此，Deutsch-Jozsa 问题可以使用仅 1 个量子查询和 $O(n)$ 其他操作来确定解决（Deutsch 和 Jozsa 的原始解决方案使用了 2 个查询，1 个查询解决方案来自于 [?]）。

相反，很容易看出，任何经典的确定性算法至少需要 $N/2+1$ 个查询：如果它只进行了 $N/2$ 个查询并且只看到 0，那么正确的输出仍然未确定。然而，如果我们允许一个小的错误概率，经典算法可以有效地解决这个问题：在两个随机位置查询 x，如果这些位相同，则输出"常数"，如果它们不同，则输出"平衡"。如果 x 是常数，这个算法输出正确答案的概率为 1，如果 x 是平衡的，输出正确答案的概率为 1/2。

8.3.2. 编码

Deutsch 算法是第一个展示量子计算优于经典计算的明显优势的算法。在 Deutsch 问题中，我们得到了一个计算一个比特布尔函数的黑盒。也就是说，一个输入一个比特并输出一个比特的函数。我们可以表示函数 f 为：f W f0;1g ! f0;1g (8.2) 例如，正如 David Deutsch 指出的，我们可以想象黑盒函数在计算某个复杂的函数，例如路由算法，输出（0 或 1）表示选择哪条路线[4][66]。

恰好有四个一输入一输出布尔函数：2 x f0 f1 fx fxN 00101 10110 其中前两个是常数函数，f0 和 f1。也就是说，它们总是输出一个常数值。我们称其他两个，fx 和 fxN 为平衡的。在它们的输入 0 和 1 上，它们的真值表中 0 和 1 的数量相等。

现在我们可以陈述 Deutsch 的问题：给定一个一比特输入和一比特输出的布尔函数，通过尽可能少的查询该函数，确定该函数是平衡的还是常数的。

如果你用经典方法来解决这个问题，你需要至少查询函数两次：首先查看输入为 0 时的输出，然后查看输入为 1 时的输出。David Deutsch 的惊人发现是，在量子计算机上你只需要一个查询！最初的 Deutsch 算法处理这种一比特布尔神谕的情况 [65]，而 Deutsch-Jozsa（DJ）算法将方法推广，以处理 n 个输入的布尔函数 [67]。不难看出，用经典方法，一个人必须至少查询一个 n 比特布尔函数 n 次。DJ 算法只需一个查询就能解决这个问题。

8.3.3. Deutsch-Jozsa 展示的量子优势

在经典计算中，要区分常数函数和平衡函数，必须对一比特布尔函数进行两次查询。对于一个 n 比特布尔函数，必须查询 n 次。然而，在使用 DJ 算法的量子计算机上，只需要查询一次。

现在我们转向量子方法解决这个问题。在上文中，我们描述了一个不可逆的经典比特函数，例如常数函数 f0 和 f1。也就是说，知道输出值并不能让我们唯一地确定输入值。然而，为了在量子计算机上运行这个计算，我们需要使这个计算可逆。一个将经典非可逆函数变为可逆函数的技巧是在额外的寄存器中计算值（或等效地，在额外的寄存器中存储函数的输入）。假设我们有一个 n 比特输入 x，我们正在计算一个（可能是非可逆的）布尔函数 f .x/。然后我们可以通过一个作用在 nC1 量子比特上的酉矩阵 Uf 来实现这一点：Uf .jxijyi/ WD jxijy ˚ f .x/i 这里的符号 ˚ 表示模 2 加法（又称 XOR）；在上面的表达式中，我们已经确定了 Uf 如何作用在所有计算基态 jxi 和单个输出量

[4] http://www.quiprocone.org/Protected/DD$_lectures.htm$

子比特 jyi 上。由于我们已经在计算基上定义了 Uf，我们通过线性性将其定义扩展到状态空间中的所有向量。要看到这是可逆的，可以注意到将变换应用两次会使状态恢复到其原始形式。更进一步，Uf 是酉矩阵，因为它将正交归一化的计算基映射到自身，因此保持它作用的向量的长度。

这个算法中的一个核心思想是我们将在与计算基不同的基上测量。如果我们在计算基（例如 z 基）上测量，那么我们将不会获得量子优势，因为我们有两个基态，j0i 和 j1i，它们对应于经典比特 0 和 1。使这个算法奏效的技巧之一是我们将在 Hadamard 基态上测量，该基态是 j0i 和 j1i 的叠加 [126][137]。图 8.2 显示了 DJ 的电路图。

让我们看看如何在 Cirq 中实现这些功能。f0 实现了变换在 Cirq 中，我们可以使用以下代码来实现这些功能

```
import cirq

# 定义量子比特
q0, q1 = cirq.LineQubit.range(2)

# 实现 f0 函数（恒定函数）
def f0(q0, q1):
    pass

# 实现 f1 函数（恒定函数）
def f1(q0, q1):
    yield cirq.X(q1)

# 实现 fx 函数（平衡函数）
def fx(q0, q1):
    yield cirq.CNOT(q0, q1)

# 实现 fxN 函数（平衡函数）
def fxN(q0, q1):
    yield cirq.CNOT(q0, q1)
    yield cirq.X(q1)
```

现在我们已经定义了四个布尔函数（f0、f1、fx 和 fxN），我们可以在 Cirq 中实现 Deutsch-Jozsa 算法。这个算法的关键是使用 Hadamard 门以及测量 Hadamard 基态。以下是使用 Cirq 实现 Deutsch-Jozsa 算法的示例：

```
# 创建电路
circuit = cirq.Circuit()

# 初始化量子比特（应用Hadamard门）
circuit.append([cirq.H(q0), cirq.H(q1)])

# 应用布尔函数（例如，f0）
circuit.append(f0(q0, q1))

# 再次应用Hadamard门
circuit.append([cirq.H(q0), cirq.H(q1)])

# 测量
circuit.append([cirq.measure(q0), cirq.measure(q1)])

# 输出电路
print(circuit)

# 模拟电路
```

```
simulator = cirq.Simulator()
result = simulator.run(circuit, repetitions=1000)

# 输出结果
print(result.histogram(key='0,1'))
```

这个代码创建了一个量子电路，初始化量子比特，应用布尔函数（例如 f0），然后再次应用 Hadamard 门。接下来，我们在 Hadamard 基态上测量量子比特，并模拟电路运行。最后，我们输出测量结果的直方图。

$$|00\rangle \rightarrow |00\rangle$$

$|01\rangle \rightarrow |01\rangle$

$|10\rangle \rightarrow |10\rangle$

$|11\rangle \rightarrow |11\rangle$ (8.13) 这只是恒等变换，即一个空电路。f1 实现了变换。

$$|00\rangle \rightarrow |01\rangle$$

$|01\rangle \rightarrow |00\rangle$

$|10\rangle \rightarrow |11\rangle$

$|11\rangle \rightarrow |10\rangle$ (8.14) 这是一个在第二个量子比特上的比特翻转门。为了理解这个新定义的可逆算子是如何工作的，我们将以计算 $U_{f_x}(|0\rangle\,|0\rangle)$ 为例进行演示。回顾一下，$|00\rangle$ 是 $|0\rangle\,|0\rangle$ 的简写。那么，根据定义:

$$U_{f_x}(|00\rangle) = U_{f_x}(|0\rangle\,|0\rangle) := |0\rangle\,|0 \oplus f_x(0)\rangle = |0\rangle\,|0 \oplus 1\rangle = |0\rangle\,|1\rangle \tag{8.15}$$

值得一提的是，我们可以在每个 $|0\rangle|1\rangle$,$|1\rangle|0\rangle$ 和 $|1\rangle|1\rangle$ 的前面计算 U_{f_x}；然后让我们检查一下 f_x 是如何进行转换的。我们检查一下，f_x 是否制定了转换:

$$|00\rangle \rightarrow |00\rangle$$

$|01\rangle \rightarrow |01\rangle$

$|10\rangle \rightarrow |11\rangle$

$|11\rangle \rightarrow |00\rangle$ (8.16)

这不过是从第一个量子比特到第二个量子比特的一个 CNOT 门。最后，fxN 实现了变换

$$|00\rangle \rightarrow |01\rangle$$

$|01\rangle \rightarrow |00\rangle$

$|10\rangle \rightarrow |10\rangle$

$|11\rangle \rightarrow |11\rangle$ (8.17) 它是一个从第一个量子比特到第二个量子比特的 CNOT 门，后面跟着一个在第二个量子比特上的比特翻转。我们可以将这些函数封装到一个字典中，该字典将 oracle 名称映射到执行此功能所需的电路中的操作:

Listing 8.1: 导入 Cirq 库

```
# 导入Cirq库
import cirq

```

```
# 获取两个量子比特，分别为数据量子比特和目标量子比特
q0, q1 = cirq.LineQubit.range(2)

# 定义oracle字典
oracles = {'0': [], '1': [cirq.X(q1)], 'x': [cirq.CNOT(q0, q1)],
           'notx': [cirq.CNOT(q0, q1), cirq.X(q1)]}
```

现在让我们转到 Deutsch 的算法。假设我们可以访问之前定义的可逆 oracle 函数。通过对我们的不可逆经典函数进行类似的论证，你可以证明仅使用一次这个 oracle 就无法区分平衡函数和常数函数。但现在我们可以问这样一个问题：如果我们可以在叠加态中查询这个函数，即如果我们可以使用量子计算的力量，会发生什么?

Deutsch 证明了只需要用到一次函数查询，你就可以使用量子计算机解决这个问题。要理解这是如何工作的，我们需要两个简单的洞察。假设我们将第二个量子比特准备成叠加态

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{8.18}$$

并应用 oracle。利用运算符 Uf 的线性性得到第二个方程式，并通过观察得到第三个方程式，我们可以验证

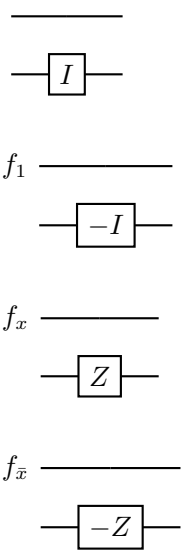
$$U_f |x\rangle |-\rangle = U_f |x\rangle \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |x\rangle \frac{1}{\sqrt{2}}(|fx\rangle - |f(x)\oplus\rangle 1) = (-1)^{f(x)} |x\rangle |-\rangle \tag{8.19}$$

这就是相位反弹技巧。将 Uf 应用到处于叠加态的目标上，函数的值最终会成为全局相位，从而反映出我们需要了解的函数是常数型还是平衡型的信息；信息被编码在相位中。

我们如何利用这一点来区分常数函数和平衡函数呢? 注意，对于常数函数，应用的相位对于所有输入 jxi 都是相同的，而对于平衡函数，相位对于每个 x 的值都是不同的。为了对每个 oracle 使用相位反弹技巧，我们在第一个量子比特上应用以下变换。

$$f_0 \to I$$

$\mathsf{f}_1 \to -I$
$\mathsf{f}_x \to Z$
$\mathsf{f}_{\bar{x}} \to -Z$ (8.20)



现在我们只需要区分第一个量子比特上的恒等门和 Z 门；我们可以通过回想起以下公式来实现这一点：

$$HZH = X \tag{8.21}$$

这意味着我们可以通过在相位翻转之前后应用 Hadamard 门来将相位翻转转换为比特翻转。如果我们观察常数函数和平衡函数，我们会发现常数函数将与 I 成比例，而平衡函数将与 X 成比例。如果我们将 j0i 输入到这个寄存器中，那么在第一种情况下，我们将只看到 j0i，在第二种情况下，我们将看到 j1i。换句话说，我们可以通过一次查询 oracle 来区分常数和平衡。

Listing 8.2: 导入 Cirq 库

```
# Import the Cirq Library
import cirq

# 获得两个量子比特，分别为数据量子比特和目标量子比特
q0, q1 = cirq.LineQubit.range(2)

# 神谕辞典
oracles = {'0': [], '1': [cirq.X(q1)], 'x': [cirq.CNOT(q0, q1)],
           'notx': [cirq.CNOT(q0, q1), cirq.X(q1)]}

def deutsch_algorithm(oracle):
    """Yields a circuit for Deustch' s algorithm given operations implementing the oracle."""
    yield cirq.X(q1)
    yield cirq.H(q0), cirq.H(q1)
    yield oracle
    yield cirq.H(q0)
    yield cirq.measure(q0)

# 显示所有口令的每个电路
for key, oracle in oracles.items():
    print('Circuit for {}...'.format(key))
    print(cirq.Circuit.from_ops(deutsch_algorithm(oracle)), end="\n\n")

# 获得一个模拟器
simulator = cirq.Simulator()

# 为每个甲骨文执行电路，以区分常数和平衡数。
for key, oracle in oracles.items():
    result = simulator.run(
        cirq.Circuit.from_ops(deutsch_algorithm(oracle)),
        repetitions=10
    )
    print('oracle: {:<4} results: {}'.format(key, result))
```

现在让我们将 Deutsch 问题扩展到 n 个布尔输入的布尔函数，而不仅仅是像上面的单输入函数。我们看到，通过 Deutsch 算法，我们可以将速度提高一倍，从经典的两个查询到量子的一个查询。如果我们能够查询一个 n 位的 Oracle，仅查询一次并确定函数是常数还是平衡的，我们就可以将时间复杂度从 O(n) 提高到 O(1)，这是一个显著的加速。所有的一位输入布尔函数都是常数或平衡的。对于具有两个输入比特的布尔函数，有两个常数函数 f(x0, x1) = 0 和 f(x0, x1) = 1，而有 6 个平衡函数。以下代码给出了查询这些函数的 2 个操作。

Listing 8.3: Cirq 中三个量子比特上的 Deustch-Jozsa 算法

```
# 导入Cirq库
```

```
import cirq

# G等三个量子比特--两个数据和一个目标量子比特
q0, q1, q2 = cirq.LineQubit.range(3)

# 常数函数的奥义
constant = ([], [cirq.X(q2)])

# 平衡函数的奥秘
balanced = ([cirq.CNOT(q0, q2)],
            [cirq.CNOT(q1, q2)],
            [cirq.CNOT(q0, q2), cirq.CNOT(q1, q2)],
            [cirq.CNOT(q0, q2), cirq.X(q2)],
            [cirq.CNOT(q1, q2), cirq.X(q2)],
            [cirq.CNOT(q0, q2), cirq.CNOT(q1, q2), cirq.X(q2)])

def deutsch_jozsa_circuit(oracle):
    """Yields a circuit for the Deustch-Jozsa algorithm on three qubits."""
    # 相位回旋术
    yield cirq.X(q2), cirq.H(q2)
    # 在输入位上进行等量叠加
    yield cirq.H(q0), cirq.H(q1)
    # 查询功能
    yield oracle
    # 干扰得到的结果，将最后一个量子位放入|1>中
    yield cirq.H(q0), cirq.H(q1), cirq.H(q2)
    # 最后一个OR门，将结果放入最后一个量子位
    yield cirq.X(q0), cirq.X(q1), cirq.CCX(q0, q1, q2)
    yield cirq.measure(q2)

# 获得一个模拟器
simulator = cirq.Simulator()

# 执行恒值函数神谕的电路
print('Your result on constant functions:')
for oracle in constant:
    result = simulator.run(cirq.Circuit.from_ops(deutsch_jozsa_circuit(oracle)), repetitions=10)
    print(result)

# 执行平衡函数神谕的电路
print('Your result on balanced functions:')
for oracle in balanced:
    result = simulator.run(cirq.Circuit.from_ops(deutsch_jozsa_circuit(oracle)), repetitions=10)
    print(result)
```

我们现在可以看到如何查询一个 n 位布尔输入的神谕，以检查它是否是常数或平衡。

8.4. Bernstein-Vazirani 算法

现在让我们转向本书中早先讨论过的 Bernstein-Vazirani（BV）算法 [127][29]。与 DJ 一样，BV 的目标也是确定黑盒布尔函数的性质。虽然 DJ 证明了量子计算比经典计算具有优势，但是如果我们允许一定的误差率，那么这种优势会消失：经典和量子方法的时间复杂度都是 O（1）[126][137]。BV 是开发的第一个算法，它表明即使考虑到错误，量子计算和经典计算也存在真正的非确定性加速差异。以下是 BV 问题的陈述：给定 n 个输入的未知函数：f.xn 1; xn 2; :::; x1; x0/；让 a 是小于 2n 的未知非负整数。让 f.x/对任何其他这样的整数 x 进行模 2 求和，乘以 a。因此，函数的输

出为：a x D a0x0˚a1x1˚a2x2:::: 在一个查询中找到 a [128][151]。就像在 DJ 中一样，我们准备了两组量子比特的状态：数据寄存器比特和目标比特。数据寄存器比特设置为 j0i，目标比特设置为 j1i。然后，我们对两组量子比特应用 H，将它们置于叠加状态。应用于数据寄存器比特的 H 准备我们在 X 基础上进行测量。然后，我们应用单量子门 Uf 和 H 到数据寄存器比特，然后测量这些比特。由于我们只应用了 Uf 一次，因此 BV 的时间复杂度为 O（1）。

Listing 8.4: Cirq 中的 Bernstein-Vazirani 算法

```
# Imports
import random
import cirq

def main():
    """执行BV算法."""
    # 量子比特的数量
    qubit_count = 8
    # 从电路中取样的次数
    circuit_sample_count = 3

    # 选择要使用的量子比特
    input_qubits = [cirq.GridQubit(i, 0) for i in range(qubit_count)]
    output_qubit = cirq.GridQubit(qubit_count, 0)

    # 为甲骨文挑选系数并创建一个电路来查询它
    secret_bias_bit = random.randint(0, 1)
    secret_factor_bits = [random.randint(0, 1) for _ in range(qubit_count)]
    oracle = make_oracle(input_qubits, output_qubit, secret_factor_bits, secret_bias_bit)
    print(f"Secret function:\nf(x) = x*<{', '.join(str(e) for e in secret_factor_bits)}> + {
        secret_bias_bit} (mod 2)")

    # 将神谕嵌入到一个特殊的量子电路中，精确地查询它一次
    circuit = make_bernstein_vazirani_circuit(input_qubits, output_qubit, oracle)
    print("\nCircuit:")
    print(circuit)

    # 从电路中取样数次
    simulator = cirq.Simulator()
    result = simulator.run(circuit, repetitions=circuit_sample_count)
    frequencies = result.histogram(key='result', fold_func=bitstring)
    print("\nSampled results:")
    print(frequencies)

    # 检查我们是否真的找到了秘密值
    most_common_bitstring = frequencies.most_common(1)[0][0]
    print("\nMost common matches secret factors:")
    print(most_common_bitstring == bitstring(secret_factor_bits))

def make_oracle(input_qubits, output_qubit, secret_factor_bits, secret_bias_bit):
    """实现函数f(a)=a*因素+偏置（mod 2）的门。"""
    if secret_bias_bit:
        yield cirq.X(output_qubit)
    for qubit, bit in zip(input_qubits, secret_factor_bits):
        if bit:
            yield cirq.CNOT(qubit, output_qubit)

def make_bernstein_vazirani_circuit(input_qubits, output_qubit, oracle):
```

```
    """用一个查询解决f(a)=a*因素+偏向（mod 2）中的因素。"""
    circuit = cirq.Circuit()
    # 初始化量子比特
    circuit.append([cirq.X(output_qubit), cirq.H(output_qubit), cirq.H.on_each(*input_qubits)])

    # 查询神谕
    circuit.append(oracle)

    # 在X基础上的测量
    circuit.append([cirq.H.on_each(*input_qubits), cirq.measure(*input_qubits, key='result')])

    return circuit

def bitstring(bits):
    """从一个可迭代的比特中创建一个比特字符串."""
    return ''.join(str(int(b)) for b in bits)

if __name__ == '__main__':
    main()
```

在这里，我们使用 X 操作将目标（或输出）寄存器初始化为 1，并通过应用 H 将数据寄存器量子比特从状态 j0i 转换为 jCi / j0i 基础。然后我们查询 oracle，对每个输入量子比特应用 H 并测量输入量子比特。这样我们就得到了我们正在寻找的答案 a，在一次查询中。因此，无论我们有多少输入，我们都可以在 O.1/ 的时间内执行此算法。

```
"""
=== EXAMPLE OUTPUT for BV ===
Secret function:
f(x) = x*<0, 1, 1, 1, 0, 0, 1, 0> + 1 (mod 2)
Sampled results:
Counter({' 01110010' : 3})
Most common matches secret factors:
True
"""
```

8.5. Simon 问题

在 BV 的结果之后，Daniel Simon 展示了量子计算机在确定函数周期性方面比经典计算机快指数倍的能力。

让我们回忆一下，一个函数可以将两个不同的输入映射到相同的输出，但不能将相同的输入映射到两个不同的输出。换句话说，2:1 是可以接受的，但 1:2 不可以。例如，将每个输入平方的函数 $f(x) = x^2$ 实际上是一个函数；两个不同的输入，即 1 和-1 都映射到 1，即 $f(x)$ 是 2:1 的。有关函数和单射、满射和双射的回顾，请参见第 III 部分。

如果我们确定函数是 2:1 类型，则我们的下一个挑战是研究函数的周期性，这是 Simon 问题的核心目标。以下是该问题更正式的概述：

实现将 n 位字符串映射到 m 位字符串 $f : \{0,1\}^n \to \{fx_0{}^m\}$ 的函数的 oracle，其中 $m \geq n$，并且 f 是 1:1 类型函数（每个输入都给出不同的输出）或 2:1 类型函数（两个输入给出相同的输出），其中非零周期 $x \in \{0,1\}^n$，使得对于所有 x, x_0 我们有 $f(x) = f_0(x_0)$ 当且仅当 $x_0 = x \oplus s$，其中 $\oplus$ 表示模 2 加法。问题是确定函数 f 的类型，并且如果它是 2:1，则确定周期 s[?]。

这是一个在计算理论和量子计算中常见的问题。问题考虑的是一个”oracle”（在理论计算中，”oracle” 可以被视为一个黑盒子，它可以在一个步骤中实现我们需要的函数或操作）。这个 oracle

实现了一个从 n 位字符串映射到 m 位字符串的函数 $f \square \{0,1\}^n \to \{0,1\}^m$，其中 $m \leq n$。这个函数 f 被保证是 $1-1$ 类型函数（即每个输入产生不同的输出）或 $2-1$ 类型函数（两个输入产生相同的输出）。在 $2-1$ 类型函数的情况下，它具有非零周期 s，s 是 n 位长的字符串（$s \in \{0,1\}^n$）。”周期” 的意思是，如果我们取一个输入 x 并对其加上周期 s（模 2 加法），我们将得到另一个输入 x'，对于这个输入，函数 f 的输出与原来的输入 x 相同，即 $f(x) = f(x')$。这个问题的目标是确定函数 f 的类型，如果它是 $2-1$ 类型的，还需要确定其周期 s。这种类型的问题在量子计算中很重要，因为它们通常与找到隐藏的子空间或周期性结构有关，这是一些量子算法（如 Shor's 算法或 Simon's 算法）的关键步骤。

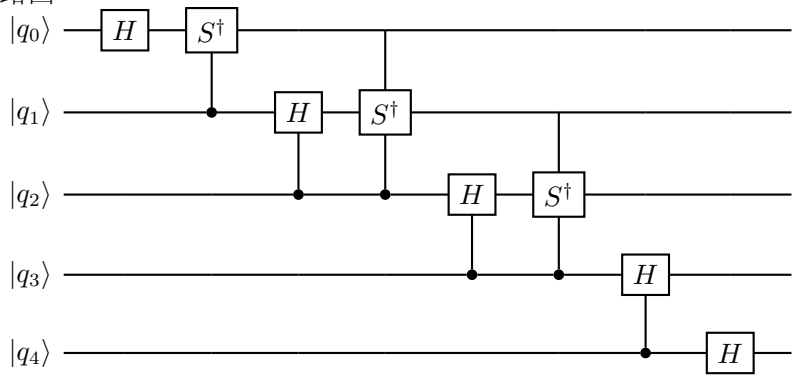
Simon 问题与 Deutsch 问题的思路相同——您面对一个 oracle——一个黑盒函数，您可以观察特定输入的输出，但不能观察底层函数。挑战是确定此黑盒过程是否会将两个不同的输入发送到相同的输出，并确定其发生频率。请注意，在这些算法中，我们不会发现黑盒中底层函数是什么，只会发现输入和输出之间的关系。实际的函数可能非常复杂，可能需要更多的步骤来分析输入/输出关系。我们在本章进一步探讨了 oracle 的概念。

Simon 成功地证明了我们可以在量子计算机上指数级地快速解决此问题，以确定函数的周期性，而经典计算机无法实现。Shor 在开发用于分解大型数的算法时，基于这一关键结果意识到，找到函数的周期性和分解大型复合数这两个问题实际上是同构的。

20 年后，研究人员实际上成功地使用量子系统确定了函数的周期性。

8.6. 量子傅里叶变换

在第 3 章中，我们讨论了量子傅里叶变换（QFT）作为一种设置测量振幅的方法，以便有利于我们需要的含有信息的量子比特。我们可以在 NISQ 硬件上直接实现 QFT 电路。这是 QFT 的标准电路图：



这是一个 5-qubit 的 QFT 电路示例。这个电路包含了一些常见的量子门，如 Hadamard 门（H）、控制相位门（Controlled phase gates，这里用 S 表示）和他们的相反门（用 $S^\dagger$ 表示）。这些门被组合在一起，以便对输入的量子比特串进行傅立叶变换。

这个电路的具体规模（即 qubit 数）可以根据你的需要进行调整。增加或减少行数即可增加或减少 qubit 数。

现在让我们逐步介绍 QFT 的代码，因为我们将在接下来的 Shor 算法部分中使用这种技术。我们将逐步解释程序的细节，首先讨论高级别细节，然后再讨论实现细节。

首先，我们导入了必要的程序包，包括 Cirq 库。

```
"""
创建并模拟一个4量子比特系统上的量子傅里叶变换（QFT）电路。
```

```
"""
# 导入库
import numpy as np
import cirq
```

接下来，我们定义主函数，它简单地调用电路生成函数，然后运行它–在这里是在模拟器上。然后程序打印出应用 QFT 后波函数的最终状态。

```
def main():
    """演示量子傅里叶变换。"""
    # 创建电路并显示
    qft_circuit = generate_2x2_grid_qft_circuit()
    print('电路: ')
    print(qft_circuit)
    # 模拟并收集最终状态
    simulator = cirq.Simulator()
    result = simulator.simulate(qft_circuit)
    # 显示最终状态
    print('\n最终状态')
    print(np.around(result.final_state, 3))
```

接下来，我们定义一个用于构建 QFT 电路的辅助函数。这个函数会在输入的量子比特上产生一个受控制的 Rz 旋转门和一个 SWAP 门。

```
def cz_and_swap(q0, q1, rot):
    """在输入的量子比特上生成控制-RZ门和SWAP门。"""
    yield cirq.CZ(q0, q1)**rot
    yield cirq.SWAP(q0, q1)
```

最后，我们使用这个辅助函数来编写整个电路，这个过程在下面的函数中完成。首先，我们定义一个 2×2 的量子比特网格并将它们标记为 a 到 d。在许多量子计算系统中，有限制条件，只有最近邻的量子比特才能相互作用。这意味着我们无法应用上述标准 QFT 电路。相反，我们需要应用包括 SWAP 操作的修改后的 QFT 电路，如图 8.3 所示。这是我们将在示例中实现的电路。

我们按照图表中的说明应用一系列的 Hadamard 和控制旋转操作。在这个示例中，我们对 $1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0$ 的向量执行量子傅里叶变换，这意味着在初始态 $|0000\rangle$ 上作用 QFT 电路。

```
def generate_2x2_grid_qft_circuit():
    """在2 x 2平面量子比特结构上返回一个QFT电路。
    电路参考自https://arxiv.org/pdf/quant-ph/0402196.pdf。
    """
    # 定义一个2*2的量子比特方格
    a, b, c, d = [cirq.GridQubit(0, 0), cirq.GridQubit(0, 1),
                  cirq.GridQubit(1, 1), cirq.GridQubit(1, 0)]
    # 创建电路
    circuit = cirq.Circuit.from_ops(
        cirq.H(a),
        cz_and_swap(a, b, 0.5),
        cz_and_swap(b, c, 0.25),
        cz_and_swap(c, d, 0.125),
        cirq.H(a),
        cz_and_swap(a, b, 0.5),
        cz_and_swap(b, c, 0.25),
        cirq.H(a),
        cz_and_swap(a, b, 0.5),
        cirq.H(a),
```

```
        strategy=cirq.InsertStrategy.EARLIEST
    )
    return circuit
```

最后，我们可以通过调用 main 函数来运行此电路：

```
if __name__ == '__main__':
    main()
```

输出：

```
FinalState
[0.25+0.j 0.25+0.j 0.25+0.j 0.25+0.j 0.25+0.j 0.25+0.j 0.25+0.j
    0.25+0.j
 0.25+0.j 0.25+0.j 0.25+0.j 0.25+0.j 0.25+0.j 0.25+0.j 0.25+0.j
0.25+0.j]
```

8.7. Shor's Algorithm

我们假设 Alice 想通过互联网发送私人信息给 Bob。Alice 的信息很可能被恶意监听者 Eve 截获。如果 Alice 只是发给 Bob 一个便条，这虽然令人尴尬，但并无大碍。但是如果 Alice 发送给 Bob 她的信用卡号码，这就是个大问题。如何通过互联网安全地发送消息?

密码学是制作和破解秘密代码的研究。密码学是指编写秘密代码，而密码分析是指破解这些代码。RSA 密码学是一种流行的加密方式，可以通过互联网安全地传输信息。RSA 的命名方式是对其开发的三位先驱 Rivest[5]、**阿迪·沙米尔** [6]和 **莱纳德·阿德曼** [7] 的致敬 [15] 。

RSA 密码学的核心假设是：将两个大质数相乘是一个陷门函数[8]；将两个大质数相乘很容易，但在乘法已经发生之后找到这两个质因数很难。在本章后面，我们将看到一个容错的量子计算机

[5]Ronald L. Rivest）是美国计算机科学家和密码学家，是 RSA 加密算法的共同发明人之一。他于 1947 年生于美国，毕业于麻省理工学院（MIT），并在该校任教。Rivest 在密码学领域有着丰富的研究经验和深厚的专业知识，他的研究涵盖了密码算法、网络安全、数据隐私等方面。

与 Adi Shamir 和 Leonard Adleman 合作，Ron Rivest 于 1977 年提出了 RSA 加密算法，这是一种非对称加密算法，广泛应用于数据加密、数字签名和密钥交换等领域。RSA 算法的安全性基于大数分解问题，它在现代密码学中扮演着重要角色，并为信息安全提供了可靠的保障。

除了 RSA 算法，Ron Rivest 还对其他密码学算法和协议进行了研究，并在密码学社区中享有盛誉。他的工作对于密码学的发展和应用有着重要的影响。

[6]Adi Shamir（阿迪 · 沙米尔）是以色列计算机科学家和密码学家，同时也是 RSA 加密算法的共同发明人之一。他于 1952 年生于以色列，毕业于以色列理工学院（Technion）并在该校任教。Shamir 在密码学领域有着广泛的研究和贡献。

与 Ron Rivest 和 Leonard Adleman 合作，Adi Shamir 于 1977 年提出了 RSA 加密算法，该算法以三位发明者的姓氏命名。RSA 算法是一种非对称加密算法，利用大数分解问题的困难性保护数据的机密性和完整性。该算法在现代密码学中被广泛应用于数字签名、密钥交换、安全通信等领域。

Adi Shamir 在密码学领域的研究工作不仅仅局限于 RSA 算法，他还对其他密码学算法和协议做出了重要贡献。他被认为是公钥密码学和密码学应用领域的权威人物之一，他的工作对密码学的发展和应用产生了深远的影响。

[7]Leonard Adleman（莱纳德 · 阿德曼）是一位美国计算机科学家和分子生物学家，同时也是 RSA 加密算法的共同发明人之一。他于 1945 年生于美国，毕业于加州大学伯克利分校，并在加州大学洛杉矶分校（UCLA）担任教授职位。

Leonard Adleman 与 Ron Rivest 和 Adi Shamir 合作，于 1977 年提出了 RSA 加密算法。该算法以三位发明者的姓氏命名，是一种基于大数分解困难性的非对称加密算法。RSA 算法的广泛应用使得现代通信和信息安全领域得以实现安全的数据传输和加密保护。

除了密码学，Leonard Adleman 还在分子生物学领域有着杰出的贡献。他是 DNA 计算的先驱之一，通过在实验室中利用 DNA 分子进行计算，展示了生物系统在计算领域的潜力。

Leonard Adleman 因其在计算机科学和分子生物学领域的创新和贡献而获得了许多奖项和荣誉，包括图灵奖（Turing Award），该奖项被认为是计算机科学界最高荣誉之一。

[8]" 陷门函数"（Trapdoor Function）是密码学中的一个重要概念。陷门函数是一种数学函数，它在一个方向上很容易计算，但在相反方向上却很困难。

具体来说，对于一个陷门函数，给定输入，可以很容易地计算出输出。然而，如果只有输出，要逆向计算出输入则非常困难，除非知道特定的秘密信息，也称为" 陷门"。

陷门函数在密码学中扮演着重要的角色，特别是在公钥密码学中。例如，RSA 加密算法中的大数分解问题被视为一个陷门函数。在 RSA

将有能力克服因子分解过程中的困难，这将使整个 RSA 方案面临风险 [29]！这将引领我们进入后量子密码学领域，这是数学、物理和计算机科学交叉领域的一个迷人的领域。

1994 年，Peter Shor 发表了他的里程碑论文，建立了一种量子算法，用于素数因子分解 [29]。将一个数字分解成质因数的问题归结为找到一个因子，因为如果您可以找到一个因子，您可以使用它来除以原始数字并考虑较小的因子。最终，使用这种分治策略，我们可以完全将数字分解成质数。他找到任何数 n 的一个因子的技术是找到一个特定函数 f 的周期 r，然后利用这个函数的周期知识来找到一个数字的因子。

8.7.1. 函数的周期

将两个大质数相乘的因子分解问题，在某种程度上等价于寻找函数的周期问题。为了了解函数的周期是什么，可以考虑将某个数字（如 2）提高到越来越高的幂次方，然后对两个质数（如 91 = 13 × 7）取模。例如：

n	n mod 91
20	1
21	2
22	4
23	8
24	16
25	32
26	64
27	37
28	74
...	...

我们可以看到，在模运算的约束下，数值无法无限增大。例如，当我们考虑 2 的幂时，我们发现 $2^6(mod 91) = 64$，但是下一个更高的幂，$2^7(mod 91) = 37$，数值并未无限增大。

8.4 练习

请确定 2 的幂是否会在模 91 的运算下循环回到数字 1。更准确地说，找到一个大于 0 的最小数字 n，使得

$$2^n(mod 91) = 1$$

如果你勤奋不懈地尝试解决这个问题，你会发现，是的，$2^{12}(mod 91) = 1$，12 是满足这个条件的最小的大于 0 的数字。这是不是意味着它一定会回到 1 呢？我们称数字 12 为函数

$$f(n) = 2^n(mod 91)$$

的周期。

更一般地说，如果我们有一个函数

$$f(n) = a^n(mod N)$$

其中 a 是与 N 互质的某个数字（即，a 和 N 的最大公因数为 1），那么我们定义函数 f 的周期为使

中，使用一个容易计算的公钥进行加密，但要解密需要使用一个陷门（私钥），从而保证了数据的安全性。

陷门函数的概念和应用对于密码学中的加密、数字签名、身份验证等领域具有重要意义，它们为安全通信和信息保护提供了基础和保障。

得 $f(n) = 1$ 的最小的大于 0 的数字 n。这个数字 n 也被称为元素 a 在群 $(\mathbb{Z}/N\mathbb{Z})^\times$ 中的序数，其中 $(\mathbb{Z}/N\mathbb{Z})^\times$ 是一个乘法群，其基础集合是所有与 N 互质的数字，二进制操作是模 N 乘法[9]。

我们可以通过这个方法找到一个函数在模运算下的周期，这在数学和计算机科学中都有很多应用。

8.5 练习：请验证 $(\mathbb{Z}/N\mathbb{Z})^\times$，其基础元素集合是 $\{1, 2, \ldots, N-1\}$ 中与 N 互质的数字子集，其二进制运算是模 N 的乘法，实际上是一个群！对于任何数字 N，群 $(\mathbb{Z}/N\mathbb{Z})^\times$ 有多少个元素？你能找到一个关联 $(\mathbb{Z}/N\mathbb{Z})^\times$ 中元素数量和数字 N 的模式吗?

你在找到函数 $f(x) = 2^n (mod 97)$ 的周期上所遇到的困难并不是独一无二的。即使是（经典的）计算机也会很难找到这个函数的周期！**彼得·肖尔**意识到，我们可以利用量子计算来快速找到这样一个函数的周期。我们现在将解释如何将函数的周期用作输入，以输入到可以破解 RSA 密码的因式分解算法中。

函数的周期作为因式分解算法的输入

假设我们被要求对某个数 N 进行因式分解，并且我们知道如何找到任何模函数的周期，如上所述。请记住，因式分解 N 的问题可以转化为查找 N 的任何因子的问题。因此，让我们看看如何利用我们找到模函数周期的能力来找到 N 的因子：

1. 随机选择一个小于 N 的数字 a。
2. 使用扩展欧几里得算法计算 $\gcd(a, N)$[10]。
3. 如果 $\gcd(a, N) \neq 1$，即，a 和 N 不互质，那么 a 已经是 N 的一个非平凡因子[11]，我们的任务就完成了。否则，找到模函数 $f(n) = a^n \mod N$ 的周期 r。
4. 如果 r 是奇数，或者 $a^{r/2} \equiv -1 \mod N$，那么这个 a 不适用，需要选择另一个数字重新开始。
5. 否则，经典数论保证 $\gcd(a^{r/2} + 1, N)$ 或 $\gcd(a^{r/2} - 1, N)$ 将会是 N 的一个非平凡因子。

练习：运行上述算法以分解数字 N = 21。

成功解决上述练习的方法如下：

1. 从 $a = 2$ 开始，因为
2. $\gcd(a, N) = \gcd(2, 21) = 1$，
3. （由于 $\gcd(a, N) = \gcd(2, 21) = 1$，省略步骤 3），
4. 发现模函数 $f(n) = 2^n \mod 21$ 的周期为 $r = 6$，

[9] $(\mathbb{Z}/N\mathbb{Z})^\times$ 是数论中一个重要的概念，它表示模 N 的乘法群。首先，我们来解释一下符号。这里，$\mathbb{Z}$ 是整数的集合，$N\mathbb{Z}$ 是 N 的整数倍构成的集合，$\mathbb{Z}/N\mathbb{Z}$ 表示的是整数对 N 取模后得到的结果的集合，也就是说，$\mathbb{Z}/N\mathbb{Z}$ 是由 0 到 N-1 这 N 个整数构成的集合。然后，$(\mathbb{Z}/N\mathbb{Z})^\times$ 表示的是 $\mathbb{Z}/N\mathbb{Z}$ 中所有和 N 互质的元素构成的集合，也就是说，这些元素都是在模 N 的乘法下有乘法逆元的整数。这个集合在模 N 的乘法下构成一个群，所以我们称它为模 N 的乘法群。例如，如果 N=4，那么 $\mathbb{Z}/4\mathbb{Z}$ 是集合 0,1,2,3，而 $(\mathbb{Z}/4\mathbb{Z})^\times$ 是集合 1,3，因为只有 1 和 3 与 4 互质。这个概念在数论和密码学中有很多应用，例如在 RSA 加密算法中，就涉及到了模 N 的乘法群。

[10] 这个表达式"$\gcd(a, N) \neq 1$" 在数学上表示的是 a 和 N 的最大公约数（greatest common divisor，gcd）不等于 1。

最大公约数是两个或多个整数共有的最大的能够同时整除他们的数。例如，12 和 18 的最大公约数是 6。

如果两个数的最大公约数是 1，那么我们说这两个数是互质的，也就是他们之间没有除 1 以外的公约数。例如，15 和 28 的最大公约数是 1，所以他们是互质的。

因此，"$\gcd(a, N) \neq 1$" 的意思是 a 和 N 不是互质的，也就是说，他们有除 1 以外的公约数。

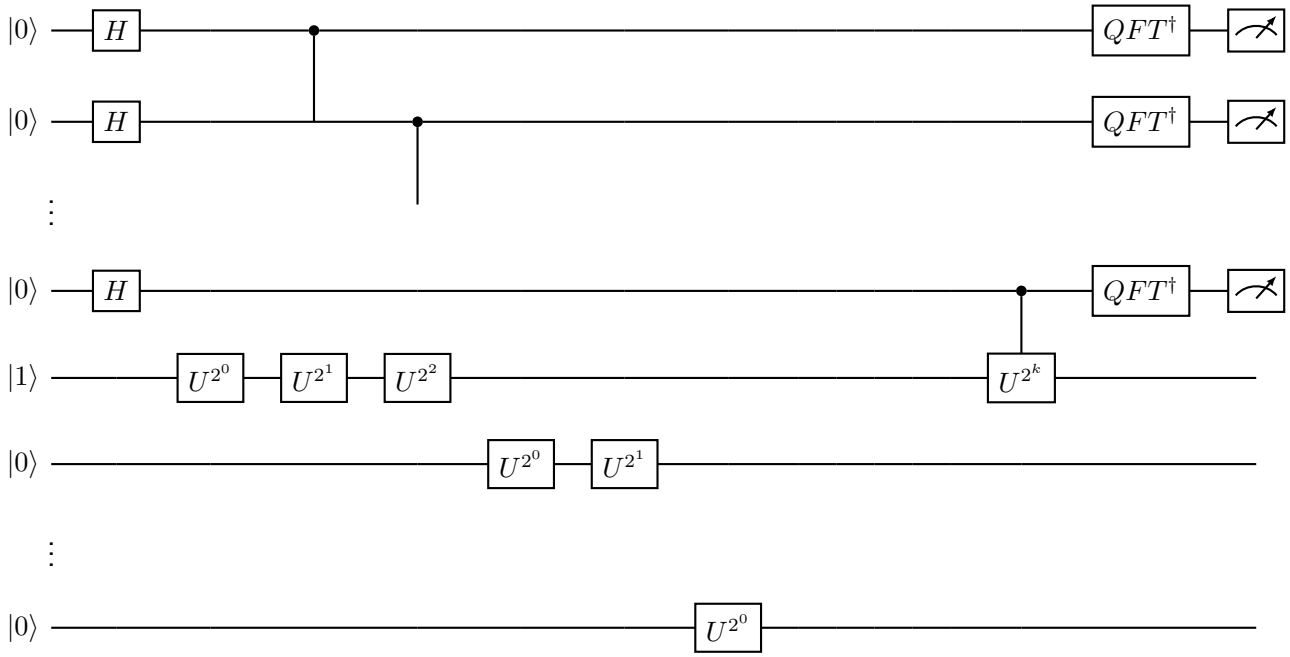
[11] " 非平凡因子" 是一个数学术语，用于描述一个数的因子（除数），该因子既不是 1，也不是这个数本身。

例如，考虑数字 12。它的因子有 1, 2, 3, 4, 6, 12。其中，1 和 12 是平凡因子，因为他们对于所有的数都是因子。而 2, 3, 4, 6 则是非平凡因子，因为他们只能被一部分数字整除。

所以，当我们说找到一个数的非平凡因子，意味着我们找到了一个既不是 1 也不是这个数本身的因子。在很多数学问题，尤其是关于质数和因式分解的问题中，找到非平凡因子是非常重要的一步。

5. $r = 6$ 既不是奇数，也不满足等式 $a^2 = -1 \mod N$，
6. $\gcd(a^2 + 1, N) = \gcd(2^2 + 1, 21) = \gcd(8 + 1, 21) = \gcd(9, 21) = 3$ 和 $\gcd(a^{2r} - 1, N) = \gcd(2^6 - 1, 21) = \gcd(63 - 1, 21) = \gcd(62, 21) = 7$ 是 $N = 21$ 的两个非平凡因子。

我们可以看到，能够找到任何模函数的周期是进行因式分解的关键。正是本书前面描述的量子傅里叶变换（QFT），让我们能够找到周期！肖尔在开发这个算法时受到了 Simon 算法和 BV 算法的启发。他基于 BV 的 QFT 使用和 Simon 方法中的周期查找，最后得出了他的数因子分解算法。以下是肖尔算法的电路图：



现在让我们来看一个 Shor's 算法样例编码的演示。以下代码来自[12]：

```
"""
toddwildey/shors-python
@toddwildey toddwildey 使用状态向量在 Python 3.X 中实现了 Shor 的算法
470 行 (353 sloc) 12.1 KB
#!/usr/bin/env python
shors.py: Shor 的量子整数因式分解算法"""
import math
import random
import argparse
__author__ = "Todd Wildey"
__copyright__ = "Copyright 2013"
__credits__ = ["Todd Wildey"]
__license__ = "MIT"
__version__ = "1.0.0"
__maintainer__ = "Todd Wildey"
__email__ = "toddwildey@gmail.com"
__status__ = "Prototype"
def printNone(str):
  pass
def printVerbose(str):
  print(str)
printInfo = printNone
#      量子组件
```

[12] https://github.com/toddwildey/shors-python

```
class Mapping:
  def __init__(self, state, amplitude):
     self.state = state
     self.amplitude = amplitude
class QuantumState:
  def __init__(self, amplitude, register):
     self.amplitude = amplitude
     self.register = register
     self.entangled = {}
  def entangle(self, fromState, amplitude):
     register = fromState.register
     entanglement = Mapping(fromState, amplitude)
     try:
       self.entangled[register].append(entanglement)
     except KeyError:
       self.entangled[register] = [entanglement]
  def entangles(self, register = None):
     entangles = 0
     if register is None:
       for states in self.entangled.values():
         entangles += len(states)
    else:
       entangles = len(self.entangled[register])
    return entangles
class QubitRegister:
  def __init__(self, numBits):
    self.numBits = numBits
    self.numStates = 1 << numBits
    self.entangled = []
    self.states = [QuantumState(complex(0.0), self) for x in
         range(self.numStates)]
    self.states[0].amplitude = complex(1.0)
  def propagate(self, fromRegister = None):
    if fromRegister is not None:
       for state in self.states:
         amplitude = complex(0.0)
         try:
           entangles = state.entangled[fromRegister]
           for entangle in entangles:
             amplitude += entangle.state.amplitude *
                  entangle.amplitude
           state.amplitude = amplitude
         except KeyError:
           state.amplitude = amplitude
    for register in self.entangled:
       if register is fromRegister:
continue
       register.propagate(self)
# Map 将把任何映射转换为单位张量，前提是映射返回的每个元素 v
# 都满足 v * v.conjugate() = 1
#
def map(self, toRegister, mapping, propagate = True):
  self.entangled.append(toRegister)
  toRegister.entangled.append(self)
  # 创建协变/逆变表示
  mapTensorX = {}
  mapTensorY = {}
```

```
  for x in range(self.numStates):
     mapTensorX[x] = {}
     codomain = mapping(x)
     for element in codomain:
       y = element.state
       mapTensorX[x][y] = element
       try:
         mapTensorY[y][x] = element
       except KeyError:
         mapTensorY[y] = { x: element }
  # 标准化映射:
  def normalize(tensor, p = False):
     lSqrt = math.sqrt
     for vectors in tensor.values():
       sumProb = 0.0
       for element in vectors.values():
         amplitude = element.amplitude
         sumProb += (amplitude * amplitude.conjugate()).real
       normalized = lSqrt(sumProb)
       for element in vectors.values():
         element.amplitude = element.amplitude / normalized
  normalize(mapTensorX)
  normalize(mapTensorY, True)
  # 量子态纠缠
  for x, yStates in mapTensorX.items():
    for y, element in yStates.items():
       amplitude = element.amplitude
       toState = toRegister.states[y]
       fromState = self.states[x]
       toState.entangle(fromState, amplitude)
       fromState.entangle(toState, amplitude.conjugate())
  if propagate:
    toRegister.propagate(self)
def measure(self):
  measure = random.random()
  sumProb = 0.0
# 选择一个状态
finalX = None
 finalState = None
     for x, state in enumerate(self.states):
       amplitude = state.amplitude
       sumProb += (amplitude * amplitude.conjugate()).real
       if sumProb > measure:
         finalState = state
         finalX = x
         break
     # 如果找到了状态, 更新系统
     if finalState is not None:
       for state in self.states:
         state.amplitude = complex(0.0)
       finalState.amplitude = complex(1.0)
       self.propagate()
     return finalX
  def entangles(self, register = None):
     entangles = 0
     for state in self.states:
       entangles += state.entangles(None)
```

```
    return entangles
  def amplitudes(self):
    amplitudes = []
    for state in self.states:
      amplitudes.append(state.amplitude)
    return amplitudes
def printEntangles(register):
  printInfo("纠缠态:␣" + str(register.entangles()))
def printAmplitudes(register):
  amplitudes = register.amplitudes()
  for x, amplitude in enumerate(amplitudes):
    printInfo('状态␣#' + str(x) + '␣的振幅:␣' +
        str(amplitude))
def hadamard(x, Q):
  codomain = []
  for y in range(Q):
    amplitude = complex(pow(-1.0, bitCount(x & y) & 1))
    codomain.append(Mapping(y, amplitude))
  return codomain
# 量子模运算
def qModExp(a, exp, mod):
  state = modExp(a, exp, mod)
  amplitude = complex(1.0)
  return [Mapping(state, amplitude)]
# 量子傅立叶变换
def qft(x, Q):
  fQ = float(Q)
  k = -2.0 * math.pi
  codomain = []
  for y in range(Q):
    theta = (k * float((x * y) % Q)) / fQ
    amplitude = complex(math.cos(theta), math.sin(theta))
    codomain.append(Mapping(y, amplitude))
  return codomain
```

现在我们已经定义了纠缠和量子傅立叶变换（QFT）的函数，我们可以定义核心的寻找周期的函数。请记住，这是必须在量子硬件上运行的关键子程序。

```
def findPeriod(a, N):
  # 确定N的位数
  nNumBits = N.bit_length()
  # 输入的位数是N的两倍减1
  inputNumBits = (2 * nNumBits) - 1
  # 如果2的inputNumBits次方小于N的平方，那么inputNumBits增加1
  inputNumBits += 1 if ((1 << inputNumBits) < (N * N)) else 0
  # Q是2的inputNumBits次方
  Q = 1 << inputNumBits
  printInfo("正在寻找周期...")
  printInfo("Q␣=␣" + str(Q) + "\ta␣=␣" + str(a))
  # 创建四个寄存器
  inputRegister = QubitRegister(inputNumBits)
  hmdInputRegister = QubitRegister(inputNumBits)
  qftInputRegister = QubitRegister(inputNumBits)
  outputRegister = QubitRegister(inputNumBits)
  printInfo("已生成寄存器")
  printInfo("对输入寄存器执行哈达玛门")
  # 对输入寄存器执行哈达玛门
  inputRegister.map(hmdInputRegister, lambda x: hadamard(x, Q), False)
```

```
  printInfo("哈达玛门执行完成")
  printInfo("将输入寄存器映射到输出寄存器，其中f(x)是a^x␣mod␣N")
  # 对输入寄存器进行模幂运算并映射到输出寄存器
  hmdInputRegister.map(outputRegister, lambda x: qModExp(a, x, N), False)
  printInfo("模幂运算完成")
  printInfo("对输出寄存器执行量子傅立叶变换")
  # 对输入寄存器执行量子傅立叶变换并映射到QFT输入寄存器
  hmdInputRegister.map(qftInputRegister, lambda x: qft(x, Q), False)
  # 传播信息
  inputRegister.propagate()
  printInfo("量子傅立叶变换完成")
  printInfo("对输出寄存器进行测量")
  # 对输出寄存器进行测量
  y = outputRegister.measure()
  printInfo("测量了输出寄存器\ty␣=␣" + str(y))
  # 输出寄存器的振幅和纠缠度，这部分代码被注释了
  # printAmplitudes(inputRegister)
  # printAmplitudes(qftInputRegister)
  # printAmplitudes(outputRegister)
  # printEntangles(inputRegister)
  printInfo("对周期寄存器进行测量")
  # 对QFT输入寄存器（周期寄存器）进行测量
  x = qftInputRegister.measure()
  printInfo("测量了QFT寄存器\tx␣=␣" + str(x))
  if x is None:
    return None
  printInfo("通过连分数寻找周期")
  # 通过连分数找到候选的周期
  r = cf(x, Q, N)
  printInfo("候选周期\tr␣=␣" + str(r
))
return r
```

现在我们可以定义将在经典硬件上运行的函数。

```
BIT_LIMIT = 12
# 位计数
def bitCount(x):
  sumBits = 0
  while x > 0:
     # 累加x的二进制表示中1的个数
     sumBits += x & 1
     # 右移一位
     x >>= 1
  return sumBits

# 最大公约数
def gcd(a, b):
  while b != 0:
    # 通过欧几里得算法求最大公约数
    tA = a % b
    a=b
    b = tA
  return a

# 扩展欧几里得算法
def extendedGCD(a, b):
  fractions = []
```

```
24    while b != 0:
25      fractions.append(a // b)
26      tA = a % b
27      a=b
28      b = tA
29    return fractions
30
31  # 连分数
32  def cf(y, Q, N):
33    fractions = extendedGCD(y, Q)
34    depth = 2
35    def partial(fractions, depth):
36      c=0
37      r=1
38      for i in reversed(range(depth)):
39        tR = fractions[i] * r + c
40        c=r
41        r = tR
42      return c
43
44    r=0
45    for d in range(depth, len(fractions) + 1):
46       tR = partial(fractions, d)
47       if tR == r or tR >= N:
48         return r
49       r = tR
50    return r
51
52  # 模幂运算
53  def modExp(a, exp, mod):
54    fx = 1
55    while exp > 0:
56      if (exp & 1) == 1:
57         fx = fx * a % mod
58       a = (a * a) % mod
59       exp = exp >> 1
60    return fx
61
62  # 随机选择一个数
63  def pick(N):
64    a = math.floor((random.random() * (N - 1)) + 0.5)
65    return a
66
67  # 检查候选值
68  def checkCandidates(a, r, N, neighborhood):
69    if r is None:
70      return None
71    # 检查倍数
72    for k in range(1, neighborhood + 2):
73       tR = k * r
74       if modExp(a, a, N) == modExp(a, a + tR, N):
75         return tR
76    # 检查下限
77    for tR in range(r - neighborhood, r):
78       if modExp(a, a, N) == modExp(a, a + tR, N):
79         return tR
80    # 检查上限
```

```
  for tR in range(r + 1, r + neighborhood + 1):
     if modExp(a, a, N) == modExp(a, a + tR, N):
       return tR
  return None
```

现在我们准备定义一个函数，它将调用我们创建的所有其他函数。这个函数将反复测试是否已经找到了周期。

```
def shors(N, attempts = 1, neighborhood = 0.0, numPeriods = 1):
  if(N.bit_length() > BIT_LIMIT or N < 3):
     return False
  periods = []
  neighborhood = math.floor(N * neighborhood) + 1
  printInfo("N = " + str(N))
  printInfo("邻域范围 = " + str(neighborhood))
  printInfo("周期个数 = " + str(numPeriods))
  for attempt in range(attempts):
     printInfo("\n尝试次数 #" + str(attempt))
     a = pick(N)
     while a < 2:
        a = pick(N)
     d = gcd(a, N)
     if d > 1:
       printInfo("在经典计算中找到因子，重新尝试")
       continue
     r = findPeriod(a, N)
     printInfo("检查候选周期、附近值和倍数")
     r = checkCandidates(a, r, N, neighborhood)
     if r is None:
       printInfo("未找到周期，重新尝试")
       continue
     if (r % 2) > 0:
       printInfo("周期为奇数，重新尝试")
       continue
     d = modExp(a, (r // 2), N)
     if r == 0 or d == (N - 1):
       printInfo("周期是平凡的，重新尝试")
       continue
     printInfo("找到周期\t周期 r = " + str(r))
     periods.append(r)
     if(len(periods) < numPeriods):
        continue
     printInfo("\n找到所有周期的最小公倍数")
     r = 1
     for period in periods:
       d = gcd(period, r)
       r = (r * period) // d
     b = modExp(a, (r // 2), N)
     f1 = gcd(N, b + 1)
     f2 = gcd(N, b - 1)
     return [f1, f2]
  return None
```

最后，我们为命令行功能定义了各种标志。

```
def parseArgs():
  parser = argparse.ArgumentParser(description='模拟对N进行 Shor 算法因式分解.')
  parser.add_argument('-a', '--attempts', type=int, default=20,
```

```
      help='要执行的量子尝试次数')
  parser.add_argument('-n', '--neighborhood', type=float,
      default=0.01, help='检查候选者的邻域大小（作为N的百分比）')
  parser.add_argument('-p', '--periods', type=int, default=2,
      help='确定最小公倍数之前要获取的周期数')
  parser.add_argument('-v', '--verbose', type=bool, default=True,
      help='详细模式')
  parser.add_argument('N', type=int, help='要分解的整数␣N')
  return parser.parse_args()

def main():
  args = parseArgs()
  global printInfo
  if args.verbose:
    printInfo = printVerbose
  else:
    printInfo = printNone
  factors = shors(args.N, args.attempts, args.neighborhood, args.periods)
  if factors is not None:
    print("因子:\t" + str(factors[0]) + ",␣" + str(factors[1]))

if __name__ == "__main__":
  main()
```

这就是–著名的肖尔算法。虽然我们还没有容错的硬件来为任何有意义的大钥匙运行肖尔算法，但它说明了量子计算的潜力。尽管 Shor 的算法被证明是在多项式时间内运行的（即，在整数中的位数的多项式要进行因子化），但可以做很多工作来减少这个多项式的常数因素和总体资源需求。参见 Gidney 和 Ekera 的工作 [129]，以讨论 Shor 算法的资源需求。下面是一个使用这个程序来计算因子 15 的例子：

```
N = 15
Neighborhood = 1
Number of periods = 2
Attempt #0
Finding the period...
Q = 256 a = 8
Registers generated
Performing Hadamard on input register
Hadamard complete
Mapping input register to output register, where f(x) is a^x mod N
Modular exponentiation complete
Performing quantum Fourier transform on output register
Quantum Fourier transform complete
Performing a measurement on the output register
Output register measured y = 1
Performing a measurement on the periodicity register
QFT register measured x = 192
Finding the period via continued fractions
Candidate period r = 4
Checking candidate period, nearby values, and multiples
Period found r = 4
Attempt #1
Found factors classically, re-attempt
Attempt #2
Found factors classically, re-attempt
Attempt #3
Finding the period...
```

```
Q = 256 a = 2
Registers generated
Performing Hadamard on input register
Hadamard complete
Mapping input register to output register, where f(x) is a^x mod N
Modular exponentiation complete
Performing quantum Fourier transform on output register
Quantum Fourier transform complete
Performing a measurement on the output register
Output register measured y = 2
Performing a measurement on the periodicity register
QFT register measured x = 128
Finding the period via continued fractions
Candidate period r = 2
Checking candidate period, nearby values, and multiples
Period found r = 4
Finding least common multiple of all periods
Factors: 5, 3
```

在这里，我们可以看到 Shor's 算法的量子部分执行了四次（标记为"尝试# 1"至"尝试# 4"）。在其中两次尝试中，电路成功地找到了周期，而在另外两次中，由于运气好，经典方法找到了因子，因此程序重新尝试了量子部分。在找到周期两次之后，Shor's 算法的经典部分开始执行，其中计算了找到的所有周期的最小公倍数。从这个结果中，正确地确定了质因数为 3 和 5。

8.8. 在 Qiskit 上的 Shor 算法

以下是使用 Qiskit 编写的 Shor's 算法的详细解释：

- **导入库**：首先，我们导入了所需的库，如 Qiskit、math、numpy 和 fractions。
- **设置模数 N**：我们希望找到 N 的质因数。在这个例子中，N = 15。
- **定义辅助函数**：我们定义了两个辅助函数，find_gcd 和 qpe_amod15。
- **find_gcd**：此函数用于计算给定整数 a 和 N 的最大公约数（gcd）。
- **qpe_amod15**：此函数用于计算 a 的模 15 序列周期。它首先创建一个量子电路，其中 n 个输入量子比特和 4 个附加量子比特。然后，它使用量子模数指数化 (QPE) 和逆量子傅里叶变换 (QFT) 来计算周期。
- **实现 Shor's 算法**：shors_algorithm 函数实现了 Shor's 算法的主要部分。它首先随机选择一个整数 a，然后计算 a 和 N 的最大公约数。如果这个数不是 1，那么我们已经找到了一个非平凡因子。否则，我们使用 qpe_amod15 函数找到 a 的序列周期 r。
- **分析结果**：我们检查计算出的序列周期 r 是否有效。如果 r 是偶数，且 $a^{(\frac{r}{2})}$ 不等于 $N-1$（模 N），那么我们可以通过计算 $a^{(\frac{r}{2})}\pm 1, N)$ 来找到 N 的非平凡因子。
- **返回结果**：shors_algorithm 函数最后返回找到的质因数。
- **运行 Shor's 算法并输出结果**：我们调用 shors_algorithm 函数并打印出找到的质因数。

需要注意的是，实际量子计算机上运行 Shor's 算法可能需要错误校正和其他技术来确保算法的准确性。在这个例子中，我们使用了 Qiskit 的 Aer 模拟器在经典计算机上模拟量子计算过程。

Listing 8.5: Qiskit 编写的 Shor's 算法

```
# 导入所需的库
```

```
from qiskit import QuantumCircuit, Aer, transpile, assemble
from qiskit.visualization import plot_histogram
from math import gcd
from numpy.random import randint
import numpy as np
from fractions import Fraction

# 模数 N, 我们希望找到其质因数
N = 15

# 检查 a 和 N 是否有公因数
def find_gcd(a, N):
    return gcd(a, N)

# 计算序列周期
def qpe_amod15(a):
    # n 是所需的 qubit 数量
    n = 8
    qc = QuantumCircuit(4+n, n)
    # 初始化输入 qubit
    for q in range(n):
        qc.h(q)     # 全部设置为 |+>
    qc.x(3+n)     # 设置输出为 |1>
    # 实现 U 算子
    qc.append(a**(2**0) % 15, [0] + [i+n for i in range(4)])
    qc.append(a**(2**1) % 15, [1] + [i+n for i in range(4)])
    qc.append(a**(2**2) % 15, [2] + [i+n for i in range(4)])
    qc.append(a**(2**3) % 15, [3] + [i+n for i in range(4)])
    qc.append(a**(2**4) % 15, [4] + [i+n for i in range(4)])
    qc.append(a**(2**5) % 15, [5] + [i+n for i in range(4)])
    qc.append(a**(2**6) % 15, [6] + [i+n for i in range(4)])
    qc.append(a**(2**7) % 15, [7] + [i+n for i in range(4)])
    # 应用逆QFT
    qc.append(qpe.qft_dagger(n), range(n))
    qc.measure(range(n), range(n))
    # 模拟器运行电路
    aer_sim = Aer.get_backend('aer_simulator')
    t_qc = transpile(qc, aer_sim)
    qobj = assemble(t_qc)
    # 获取并返回测量结果
    result = aer_sim.run(qobj).result()
    return result.get_counts()

# Shor's 算法的主要部分
def shors_algorithm(N):
    # 随机选择 1 < a < N 的整数
a = randint(2, N)
print(f"随机选择整数 a = {a}")
# 检查 a 和 N 是否有公因数
gcd_result = find_gcd(a, N)
if gcd_result != 1:
    print(f"找到一个非平凡因子: {gcd_result}")
    return gcd_result

# 使用量子算法寻找 a 的序列周期 r
counts = qpe_amod15(a)
# 提取结果中的主要周期
```

```
phase = max(counts, key=counts.get)
phase = int(phase, 2)

# 检查测量的位数
if phase == 0:
    print("测量失败，未找到序列周期")
    return None

# 使用连分数算法找到 r
r = Fraction(phase / (2**8)).limit_denominator(15).denominator

# 检查 r 是否有效
if r % 2 != 0 or (a**(r//2)) % N == N - 1:
    print("测量失败，未找到有效的序列周期")
    return None

# 计算非平凡因子
factor1 = find_gcd(a**(r//2) - 1, N)
factor2 = find_gcd(a**(r//2) + 1, N)

# 返回找到的因子
if factor1 == N or factor2 == N:
    print("测量失败，未找到非平凡因子")
    return None
else:
    print(f"找到非平凡因子: {factor1} 和 {factor2}")
    return factor1, factor2
```

Listing 8.6: Cirq 编写的 Shor's 算法

```
import cirq
import numpy as np
from numpy.random import randint
from math import gcd
from fractions import Fraction

# 模数 N，我们希望找到其质因数
N = 15

# 检查 a 和 N 是否有公因数
def find_gcd(a, N):
    return gcd(a, N)

# 定义 QPE 和 QFT 能用到的门
# ... 这里需要实现QPE和QFT

# 计算序列周期
def qpe_amod15(a):
    # n 是所需的 qubit 数量
    n = 8
    # 创建量子比特和经典比特
    qubits = cirq.LineQubit.range(n + 4)
    cbits = cirq.LineQubit.range(n)
    # 初始化电路
    circuit = cirq.Circuit()

    # 初始化输入 qubit
    for q in qubits[:n]:
```

```
        circuit.append(cirq.H(q))

    # 设置输出为 |1>
    circuit.append(cirq.X(qubits[3 + n]))

    # 实现 U 算子
    # ... 这里需要实现 U 算子

    # 应用逆 QFT
    # ... 这里需要实现逆 QFT

    # 测量
    circuit.append(cirq.measure(*qubits[:n], key='result'))

    # 运行模拟器
    simulator = cirq.Simulator()
    result = simulator.run(circuit, repetitions=1024)

    # 返回测量结果
    return result.histogram(key='result')

# Shor's 算法的主要部分
def shors_algorithm(N):
    # ... 上面的代码内容保持不变

# 运行 Shor's 算法并输出结果
factors = shors_algorithm(N)
print(f"质因数分解结果: {factors}")
```

8.9. Grover's Search Algorithm

Grover's 搜索算法

1996 年，Lov Grover 证明了我们可以在量子计算机上获得与经典计算机相比的二次速度提升的算法搜索 [31]。虽然这不是指数级速度提升，但仍然具有重要意义。

搜索问题可以设置如下。给定一个函数 $f(x)$，使得 $f(a^*) = -1$，而函数的所有其他输出都为 1，找到 a。换句话说，我们正在寻找一种详尽的搜索算法；特别是，我们正在寻求一种算法搜索协议。算法搜索协议是一种可以通过对搜索结果进行函数求值来验证我们找到了问题中的项目的方法。因此，我们已经排除了找到分析方法的可能性，现在必须进行暴力搜索。

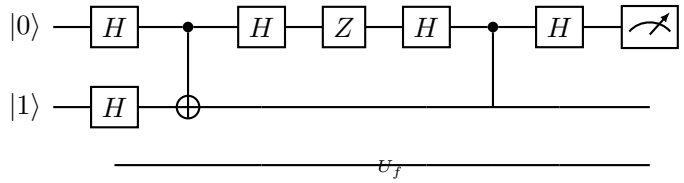
在经典计算机上，这将需要对 $x = \{0, n\}$ 的某个范围进行详尽的搜索，需要 n 次操作；或者最好情况下，$\frac{n}{2}$ 步，如果我们认为平均搜索一半范围后可以找到目标。然而，在量子计算机上，我们可以做得更好。我们可以在 $O(\sqrt{n})$ 步内找到 a，而不是 n 或 $\sqrt{n}2$。Bennett 等人随后证明了，在量子计算机上运行并解决算法搜索问题的任何算法，最好会查询 $\Omega(\sqrt{n})$ 次 oracle；因此 Grover's 算法是最优的 [130] 。

Grover's 算法比 DJ 和 BV 稍微复杂一些。在这里，我们需要应用三个酉算子，后两个我们在循环中实现，直到找到我们的目标。按照 David Deutsch 的方式，让我们称这三个算子为 H、M 和 XX [13]。

与我们之前研究的其他算法一样，我们将数据输入量子比特准备为状态 $|0\rangle$，将输出量子比特准备为状态 $ket1$。然后，我们将 H 应用于所有数据输入量子比特和输出量子比特。

[13] http://www.quiprocone.org/Protected/DD$_{l}ectures.htm$

现在我们查询 oracle：



Listing 8.7: Grover 在 Cirq 的算法

```
import random
import cirq

def set_io_qubits(qubit_count):
    """添加指定数量的输入和输出量子比特."""
    input_qubits = [cirq.GridQubit(i, 0) for i in range(qubit_count)]
    output_qubit = cirq.GridQubit(qubit_count, 0)
    return (input_qubits, output_qubit)

def make_oracle(input_qubits, output_qubit, x_bits):
    """实现函数 {f(x) = 1 如果 x==x', f(x) = 0 如果 x!= x'}."""
    # 制作 oracle
    yield (cirq.X(q) for (q, bit) in zip(input_qubits, x_bits) if not bit)
    yield (cirq.TOFFOLI(input_qubits[0], input_qubits[1], output_qubit))
    yield (cirq.X(q) for (q, bit) in zip(input_qubits, x_bits) if not bit)

def make_grover_circuit(input_qubits, output_qubit, oracle):
    """在 sqrt(N) 次尝试中找到 oracle 识别的值."""
    c = cirq.Circuit()
    # 初始化量子比特
    c.append([
        cirq.X(output_qubit),
        cirq.H(output_qubit),
        cirq.H.on_each(*input_qubits),
    ])
    # 查询 oracle
    c.append(oracle)
    # 构造 Grover 算子
    c.append(cirq.H.on_each(*input_qubits))
    c.append(cirq.X.on_each(*input_qubits))
    c.append(cirq.H.on(input_qubits[1]))
    c.append(cirq.CNOT(input_qubits[0], input_qubits[1]))
    c.append(cirq.H.on(input_qubits[1]))
    c.append(cirq.X.on_each(*input_qubits))
    c.append(cirq.H.on_each(*input_qubits))
    # 测量结果
    c.append(cirq.measure(*input_qubits, key='result'))
    return c

def bitstring(bits):
    return ''.join(str(int(b)) for b in bits)

def main():
    qubit_count = 2
    circuit_sample_count = 10
    # 设置输入和输出量子比特
    (input_qubits, output_qubit) = set_io_qubits(qubit_count)
    # 选择 x' 并制作可以识别它的 oracle
```

```
    x_bits = [random.randint(0, 1) for _ in range(qubit_count)]
    print(f'Secret bit sequence: {x_bits}')
    # 制作 oracle（黑箱）
    oracle = make_oracle(input_qubits, output_qubit, x_bits)
    # 将 oracle 嵌入到实现 Grover 的算法的量子电路中
    circuit = make_grover_circuit(input_qubits, output_qubit, oracle)
    print('Circuit:')
    print(circuit)
    # 对电路进行多次采样
    simulator = cirq.Simulator()
    result = simulator.run(circuit, repetitions=circuit_sample_count)
    frequencies = result.histogram(key='result', fold_func=bitstring)
    print(f'Sampled results:\n{frequencies}')
    # 检查我们是否真的找到了秘密值
    most_common_bitstring = frequencies.most_common(1)[0][0]
print(f'Most common bitstring: {most_common_bitstring}')
print(f'Found a match: {most_common_bitstring == bitstring(x_bits)}')

if name == 'main':
main()
```

我们现在运行该代码，并获得这个作为例子的输出：

```

"""
=== EXAMPLE OUTPUT ===
Secret bit sequence: [1, 0]
Sampled results:
Counter({'10': 10})
Most common bitstring: 10
Found a match: True
"""
```

8.10. 总结

在本章中，我们探讨了一组典型的量子算法。这些来自 20 世纪 80 年代和 90 年代的突破性成果奠定了量子优势的潜力。尽管我们现在还没有足够的硬件来运行具有实际意义规模的 Shor 和 Grover 算法，但它们是对未来的强大提醒。在下一章中，我们将介绍一系列用于 NISQ 时代的量子计算方法。

9

NISQ 时代量子计算范式

NISQ，或”Noisy Intermediate-Scale Quantum”，是指当前阶段（大约在 2020 年至 2030 年之间）的量子计算机。这些量子计算机通常包含几十到几百个量子比特（qubits），但受限于噪声和计算错误，因此它们被称为” 噪声中等规模” 的量子计算机。

在 NISQ 时代，研究人员正在尝试找到能够在这些设备上有效运行的应用。这是一个挑战，因为噪声和错误会限制量子计算机能够执行的计算的复杂性和准确性。然而，即使在这些限制下，NISQ 设备也可能能够执行某些类型的计算，这对于经典计算机来说是困难或不可能的。

NISQ 设备的一个关键应用领域是量子模拟，即使用量子计算机模拟量子系统，如分子或材料的性质。这种类型的模拟对于经典计算机来说极为困难，但对于量子计算机来说，由于它们的量子性质，这种任务可能会更容易。

此外，NISQ 设备还可能在优化问题、机器学习和金融模型等领域有用。然而，这些应用的具体效能和实用性还有待进一步研究。

然而，值得注意的是，NISQ 设备尚未实现量子优势，即量子计算机在某个任务上超越最好的经典计算机的能力。这是因为 NISQ 设备的噪声和错误率过高，导致它们在执行大规模的量子计算时结果不准确。

总的来说，NISQ 时代是量子计算发展中的一个重要阶段，尽管有许多挑战，但也为研究人员提供了研究量子计算潜能的独特机会。在本节中，我们将介绍一系列可以在 NISQ 处理器上运行的量子计算程序。我们将涵盖优化、化学、机器学习和其他领域的方法。

- **优化**: 量子近似优化算法（QAOA）QAOA 是一种量子计算方法，用于解决组合优化问题，如最大割问题、旅行商问题等。QAOA 结合了量子计算的优势和经典优化技术，使其适用于 NISQ 设备。
- **化学**: 量子相位估计（QPE）和变分量子本征求解器（VQE）量子计算在化学领域的应用主要集中在解决分子哈密顿量的本征值问题，以确定分子的能量和结构。QPE 和 VQE 是两种常用的求解这类问题的方法。QPE 需要较高的量子计算资源，而 VQE 是一种混合量子-经典算法，更适合 NISQ 设备。
- **机器学习**: 量子支持向量机（QSVM）和量子神经网络（QNN）量子计算在机器学习领域的

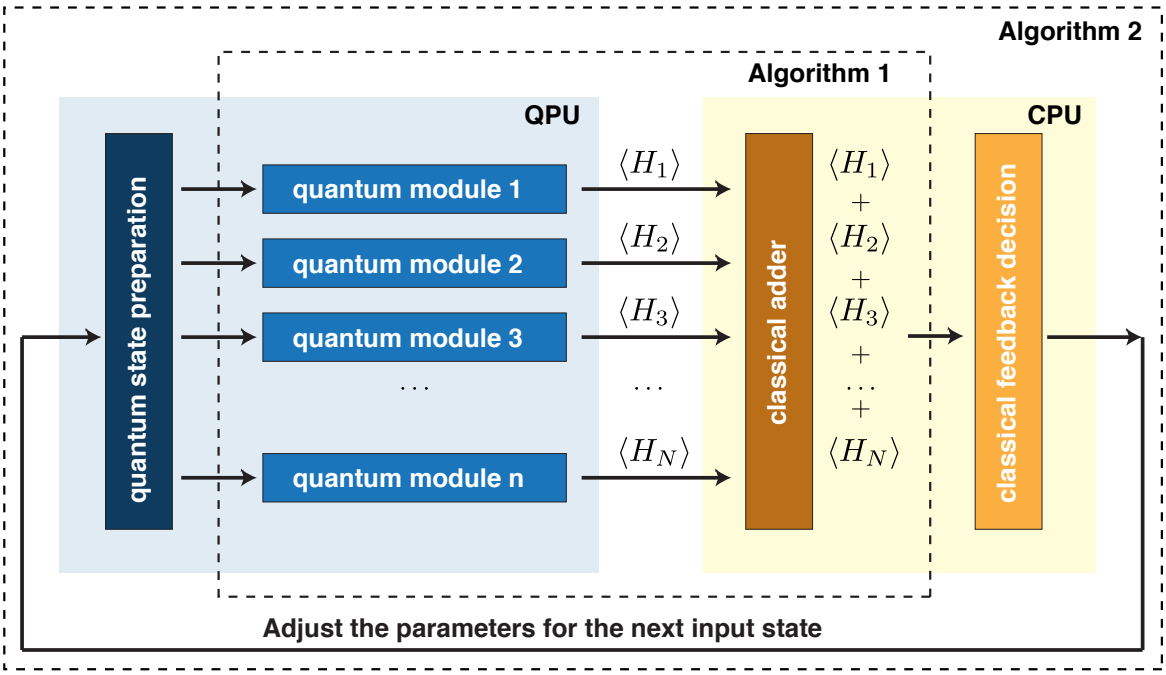


图 9.1: 一种基于光子量子处理器的量子变分本征值求解器的架构，该架构基于光子量子处理器，并且提供了一种实现量子计算的方法。图中显示了一个光子量子处理器的示意图，该处理器由一系列波导和光学元件组成。该架构中的核心思想是利用量子光学技术来实现量子比特的控制和测量操作。在图中，方框表示量子比特（qubits），用线段表示波导。每个波导代表一个光学路径，用于传递光子和实现量子操作。光子的路径可以通过控制光学元件（如相移器和波导耦合器）来调整，从而实现量子门操作。图中还展示了连接量子比特的光学线路，这些线路允许量子比特之间进行相互作用和纠缠。这种纠缠是量子计算的核心要素，可以用于实现量子算法和求解量子问题。该架构的关键点是通过优化参数化量子电路来求解量子系统的本征值问题。通过适当选择参数化电路，结合经典计算资源进行优化，可以找到能量函数的本征值和相应的本征态。这种方法可以用于模拟和求解各种化学和物理系统的量子特性。图片来源 [131]

应用主要关注提高分类和回归任务的性能。QSVM 和 QNN 是两种常用的量子机器学习方法。QSVM 利用量子计算实现内积计算，从而提高分类性能。QNN 则是量子与神经网络相结合的一种方法，利用量子比特和量子门构建类似于经典神经网络的结构。

- **其他领域**: 量子计算在其他领域的应用还包括解决线性系统（HHL 算法）、解决微分方程（量子线性组合算法）等。这些算法在某些情况下可以比经典算法更快地解决问题，但它们的适用性和扩展性可能受到 NISQ 设备的限制。

NISQ 时代的量子计算为许多领域提供了新的可能性。虽然它们可能无法立即实现广泛的量子优势，但它们为实现未来更大规模的量子计算奠定了基础。

9.1. 变分量子本征求解器

首先让我们研究一下变分量子本征求解器（VQE）[131]。我们可以使用 VQE 找到代表系统哈密顿量的大型矩阵的本征值。在许多情况下，我们寻找的是最低本征值，它表示系统的基态能量。我们还可以使用 VQE 和 VQE 类算法计算其他本征值，它们表示激发态能量[1][132, 133]。VQE 是解决问题的混合量子/经典方法的一个很好的例子（有关 VQE 的更多信息，请参见 [131, 134, 101,

[1]在量子力学中，当我们讨论一个粒子（例如电子）在原子或分子中的能量状态时，我们通常会引用其" 基态" 和" 激发态"。

基态是粒子能量最低的状态，通常被认为是该粒子的" 自然" 或" 默认" 状态。当粒子获得足够的能量（例如，通过吸收光子）时，它可能跃迁到一个更高能量的状态，这就是所谓的激发态。

每个激发态都对应于一个特定的能量水平，这个能量水平高于基态。激发态的能量可以通过量子力学方程，如薛定谔方程，来计算。一旦粒子处于激发态，它通常会通过发射光子的方式回到基态，这个过程被称为" 退激发"。

135, 136]。虽然 VQE 最初是为了寻找哈密顿量的基态而开发的，但我们可以用它来找到我们可以用量子电路表示的任何给定目标函数的最小值。这大大扩展了这种变分方法的应用空间。

在变分方法中，我们参数化一个量子态 $|\psi(\theta)\rangle$，其中 θ 是一组参数。VQE 解决的问题如下：

给定一个哈密顿量 H，通常来自诸如分子氢或水等物理系统，通过求解以下优化问题来近似地找到基态能量（H 的最小特征值）：

$$\min_{\theta} \langle\psi(\theta)| H |\psi(\theta)\rangle \tag{9.1}$$

这个方程表达的是变分量子本征求解器（Variational Quantum Eigensolver，VQE）的核心问题。在这里，目标是找到一组参数 θ，使得量子态 $|\psi(\theta)\rangle$ 的期望值在哈密顿量 H 下达到最小。

在此方程中：

$$\langle\psi(\theta)| H |\psi(\theta)\rangle$$

- $|\psi(\theta)\rangle$ 是参数化的量子态，也被称为 ansatz。这个状态是由一组参数 θ 控制的，这些参数可以通过优化算法进行调整。
- H 是哈密顿量，它描述了量子系统的能量。
- $\langle\psi(\theta)|H|\psi(\theta)\rangle$ 是量子态 $|\psi(\theta)\rangle$ 在哈密顿量 H 下的期望值。这是通过将哈密顿量 H 作用在量子态 $|\psi(\theta)\rangle$ 上，然后计算结果状态的内积得到的。
- "minθ" 表示的是要找到一组参数 θ，使得期望值达到最小。

因此，该方程的目标是找到一组参数 θ，使得量子态 $|\psi(\theta)\rangle$ 在哈密顿量 H 下的期望值最小。这对应于找到哈密顿量的最低本征值（即最小能量）及其对应的本征态，这就是 VQE 的主要目标。

根据量子力学的变分原理[2]，量

$$|\psi(\theta)\rangle$$

绝不会小于基态能量。因此，通过最小化这个量，我们得到了基态能量的近似值。

在 VQE 算法中，$|\psi(\theta)\rangle$ 是在量子计算机上准备的，因此 ansatz 通常是由参数化的量子门生成的，例如旋转门 $R_\sigma(\varphi)$ ，其中 σ 是保尔算子，以及其他"静态"量子门，如 CNOT 或控制 Z。

为了 VQE 的目的，我们将假定我们的哈密顿量是写成保尔算子张量积[3]的和，其权重是恒定

在化学和物理中，激发态的能量通常对应于原子或分子的特性，例如它们的颜色（由于不同的激发态会吸收和发射不同波长的光），以及它们在化学反应中的行为。在量子计算中，通过操控量子比特（即量子版本的" 二进制位"）的基态和激发态，可以实现各种复杂的计算任务。

[2]量子力学中的变分原理是用于求解量子系统的基态（即最低能量态）的一个非常有用的方法。这个原理的关键思想是：对于任何一种可能的量子态，该态对应的期望能量总是大于或等于系统真实的基态能量。

换句话说，如果你有一个量子系统的哈密顿量 H（描述了该系统的能量），并且你选择了一个特定的量子态 $|\psi\rangle$（这个态可以是任何可能的态，但通常是参数化的，以便可以通过改变参数来调整态），那么这个态的期望能量 $E = \langle\psi|H|\psi\rangle$ 总是大于或等于系统的基态能量。

因此，通过变分原理，我们可以通过寻找能够最小化期望能量的量子态来估计一个量子系统的基态和基态能量。这是通过调整量子态 $|\psi\rangle$ 的参数来实现的。这种方法并不总是能够找到精确的基态，但它可以给出一个上界，而且在许多情况下，这个上界是足够接近真实基态的。

在量子计算中，变分量子本征求解器（Variational Quantum Eigensolver，VQE）就是基于这个原理的，它是一种在有噪声的中等规模量子（NISQ）设备上求解基态问题的有效方法。

[3]在数学和物理学中，张量积是一种二元运算，它将一对向量空间（或一般来说，模）映射到另一个向量空间（或模）。它是向量积和标量积的推广，可以用来创建更高维度和更复杂的结构。

如果我们有两个向量空间 V 和 W，那么它们的张量积记作 $V \otimes W$。如果 $v \in V$ 和 $w \in W$，那么 $v \otimes w$ 就是 $V \otimes W$ 中的一个元素。$v \otimes w$ 的结果是一个新的向量，这个向量包含了 v 和 w 的" 混合" 信息。这个结果向量的维度是 V 和 W 维度的乘积。

在量子力学中，我们经常使用张量积来描述复合量子系统。例如，如果我们有两个量子比特，每个量子比特都可以处于两个状态 $|0\rangle$ 或 $|1\rangle$，那么两个量子比特的整体状态可以用它们状态的张量积来表示。例如，如果第一个量子比特处于状态 $|0\rangle$，而第二个量子比特处于状态 $|1\rangle$，那么整个系统的状态就是 $|0\rangle \otimes |1\rangle$，通常简写为 $|01\rangle$。

要注意的是，张量积不满足交换律，也就是说，通常情况下 $v \otimes w \neq w \otimes v$。这是因为张量积包含了元素的顺序信息。

系数 [132]。

$$H = \sum_{i=1}^{m} c_i H_i \tag{9.2}$$

注意，保利算子的张量积形成了厄米矩阵的基，因此原则上任何哈密顿量都可以用这种方式表示。然而，这可能导致与系统大小呈指数关系的项数。对于这种一般情况，不同的表示对于限制哈密顿量中的项数以及因此限制量子算法所需资源至关重要。对于当前的讨论，我们将注意力限制在哈密顿量的形式（9.2）上，其中 m 最多以系统大小的多项式增长，即 $m = O(n^k)$ 这对于许多感兴趣的物理系统来说是一个合理的假设。

VQE 算法使用量子电路计算每个项 H_i 的期望值，然后经典地添加总能量。经典优化器改变 ansatz 波函数的值以最小化总能量。找到一个近似最小值后，VQE 返回基态能量及其本征态。

VQE 的另一个应用是误差缓解[4]。McClean 等人探讨了 VQE 的这个方面：

在这里，我们通过引入变分态准备[5]的可解通道模型[6]为变分方法可以自动抑制甚至非系统性退相干误差的猜想提供证据。此外，我们展示了变分量子-经典方法如何适应测量和经典计算的更一般层次结构，使得可以通过额外的经典资源获得越来越精确的解决方案 [137]。

关于这个主题不断增长的文献中探索使用子空间扩展实现误差缓解的方法 [138]。

下面，我们展示了一个使用 pyQuil 和 Grove 库 [97][7]为简单哈密顿量实现 VQE 的程序。让我们分步讲解这个程序并解释每个部分。首先，我们导入所需的软件包并连接到量子虚拟机（QVM）。

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from pyquil.quil import Program
import pyquil.api as api
from pyquil.paulis import sZ
from pyquil.gates import RX, X, MEASURE
from grove.pyvqe.vqe import VQE
```

接下来，我们设置一个 ansatz，在这种情况下，它是绕 x 轴的旋转矩阵，带有一个参数。

[4]"Error mitigation" 是量子计算中的一个重要概念。在理想情况下，量子计算机可以在没有任何错误的情况下运行。然而，在实际中，由于各种因素（包括环境噪声、硬件不完善、操作错误等），量子比特（qubit）的状态可能会被扰动，导致计算结果出现错误。这种现象被称为量子衰减或量子噪声。

错误缓解（Error Mitigation）是一种在量子计算中尽可能减少这些错误影响的技术。它并不直接消除错误，而是通过各种策略尽可能降低错误带来的影响，从而提高量子计算的准确性。这些策略可能包括用复杂的错误模型来预测错误，或者使用纠错码来纠正可能出现的错误。

错误缓解在当前的中等规模有噪声量子（NISQ）设备中尤其重要，因为这些设备的错误率相对较高，而我们又无法直接实现完全的量子错误纠正。错误缓解为我们提供了一种在现有技术条件下尽可能提高计算精度的方法。

[5]变分态准备（Variational State Preparation）是一种在量子计算中使用的策略，其目标是找到一种有效的方式来制备（即生成）目标量子态。

在变分态准备的过程中，我们选择一个参数化的量子电路（也被称为 ansatz），它可以通过改变其中的参数来生成不同的量子态。然后，我们定义一个损失函数来量化当前生成的量子态与目标量子态之间的差距。在许多情况下，这个损失函数可以是两个态的保真度的负值。

然后，我们使用一个经典的优化算法（例如梯度下降或量子自然梯度等）来调整电路中的参数，以最小化这个损失函数。通过这种方式，我们可以调整电路，使其生成的量子态尽可能接近目标量子态。

变分态准备在许多量子计算应用中都很有用，包括量子机器学习、量子化学以及其他需要准备特定量子态的问题。

[6]" 可解通道模型" 是指的一种可以被精确解析的量子通道模型。在量子信息理论中，量子通道是描述开放量子系统的主要工具，它描述了量子态如何因为与环境的相互作用而演化。

一个典型的例子是量子比特受到相位翻转或比特翻转的影响，这可以用量子错误通道来描述。例如，一个可以解析的量子通道模型可能会描述一个量子比特在一段时间后以某种概率发生翻转。

这些模型被称为" 可解"，是因为我们可以精确地计算出量子态在经过这些通道后的状态。然而，一般的量子通道可能很复杂，不能直接解析，因此需要使用数值方法或者近似方法来处理。

[7]https://grove-docs.readthedocs.io/en/latest/vqe.html

```
# 定义创建ansatz的函数
def small_ansatz(params):
    """返回一个带有一个参数的ansatz程序。"""
    return Program(RX(params[0], 0))

# 用一个参数示例值展示ansatz
print("用参数示例值展示的ansatz: ")
print(small_ansatz([1.0]))
```

在这段代码中，定义了一个函数 small_ansatz，它接受一个参数列表（在这个例子中只有一个参数），并返回一个应用于第 0 个量子比特上的旋转 X 门（RX 门）的 ansatz 量子程序。然后，这个函数被用于生成一个具有示例参数值（在这个例子中，参数值为 1.0）的 ansatz，并打印出来。

程序这部分的输出显示了 ansatz 作为 pyQuil 电路的结果如下：

```
用参数示例值展示的ansatz:
RX(1.0) 0
```

接下来，我们设置一个哈密顿量；正如我们上面所提到的，任何哈密顿量都可以表示为保罗算符的张量积的线性组合，因为这些算符形成厄米矩阵的基。在实际应用中，哈密顿量首先必须转换为量子比特算符，以便可以使用量子计算机测量期望值。如果哈密顿量（9.2）中有 m 个非平凡的、不同的项，则需要计算 m 个不同的期望值。VQE 中的每个量子电路计算一个期望值，因此需要运行 m 个不同的量子电路。

为了简化说明，我们考虑一个简单的保罗算符 H = Z 的情况（请注意，在这一节中，H 指的是哈密顿量，而不是哈达玛尔算符）。我们使用从 Grove 导入的 VQE 类创建一个 VQE 算法的实例，并计算 ansatz 中示例角度的期望值。

```
# 用一个参数示例值展示ansatz
print("用参数示例值展示的ansatz: ")
print(small_ansatz([1.0]))

# 创建一个哈密顿量 H = Z_0
hamiltonian = sZ(0)

# 使用Nelder-Mead最小化方法创建一个VQE实例
vqe_inst = VQE(minimizer=minimize,
               minimizer_kwargs={'method': 'nelder-mead'})

# 在一个特定角度（比如2.0弧度）手动检查VQE
angle = 2.0
print("在角度␣=␣{}时哈密顿量的期望值".format(angle))
print(vqe_inst.expectation(small_ansatz([angle]), hamiltonian,
10000, qvm))
```

在这段代码中，首先创建了一个哈密顿量 hamiltonian，它表示的是作用在第 0 个量子比特上的 Pauli Z 门（或者说 Z 算符）。然后，创建了一个 VQE 实例 vqe_inst，并指定了用于最小化的方法（在这个例子中是 Nelder-Mead 方法）。最后，这个 VQE 实例被用于计算在一个特定角度（在这个例子中，角度为 2.0 弧度）时，给定的 ansatz 状态关于哈密顿量的期望值，并打印出来。这个期望值是通过在一个量子虚拟机 qvm 上进行 10000 次测量得到的。

为了了解优化问题的全貌，我们可以在范围 $(0, 2\pi)$ 内扫描一组值。在这里，由于我们的 ansatz 中只有一个参数，这在计算上是廉价的。对于具有更多参数的较大 ansatz，实现网格搜索以覆盖所有可能值是不可行的，因此必须使用经典优化算法来找到一个近似最小值。

```
# 遍历一系列角度并绘制期望值（无采样）
angle_range = np.linspace(0.0, 2.0 * np.pi, 20)
exact = [vqe_inst.expectation(small_ansatz([angle]), hamiltonian,
    None, qvm)
       for angle in angle_range]
# 绘制精确期望值
plt.plot(angle_range, exact, linewidth=2)

# 遍历一系列角度并绘制期望值（有采样）
sampled = [vqe_inst.expectation(small_ansatz([angle]), hamiltonian,
    1000, qvm)
        for angle in angle_range]
# 绘制采样期望值
plt.plot(angle_range, sampled, "-o")

# 绘图选项
plt.xlabel('Angle [radians]')
plt.ylabel('Expectation value')
plt.grid()
plt.show()
```

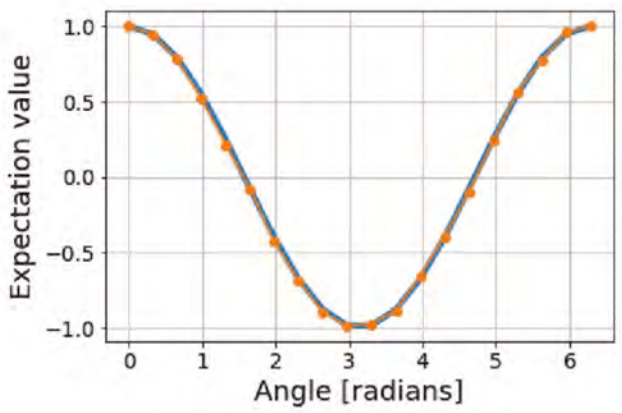


图 9.2: 简单哈密顿的预期值，在所有角度范围内（从 0 到 2π），计算了在给定的波函数 ansatz：$|\phi(\theta)\rangle = R_x(\theta)$（这是通过在初始状态 $|0\rangle$ 上施加以角度为参数的旋转 X 门得到的）中，简单哈密顿量 $H = Z$ 的期望值。

在这段代码中，首先遍历了一个从 0 到 2π 的角度范围，并计算了在每个角度下，给定的 ansatz 状态关于哈密顿量的期望值，这些期望值是通过在量子虚拟机 qvm 上进行精确计算得到的，然后将这些期望值绘制在图上。然后，再次遍历这个角度范围，但这次是通过在 qvm 上进行 1000 次采样来计算期望值，并将这些期望值也绘制在同一张图上。最后，设置了图的 x 轴标签为”Angle [radians]”，y 轴标签为”Expectation value”，并显示了网格线，然后显示了这张图。

该程序部分生成的图像如图 9.1 所示。在这里，我们可以直观地看到，汉密尔顿量的最小能量（以任意单位表示）出现在波函数 ansatz 中的角度 $\theta = \pi$。如前所述，对于需要更多 ansatz 参数的较大汉密尔顿量，枚举所有角度的期望值是不可行的。相反，必须使用优化算法来遍历优化景观，理想情况下找到全局最小值。

下面展示了用 SciPy Optimize 包实现的 Nelder-Mead 优化算法的示例:

```
# Do the minimization and return the best angle
initial_angle = [0.0]
result = vqe_inst.vqe_run(small_ansatz, hamiltonian, initial_angle,
    None, qvm=qvm)
```

```
print("\nMinimum energy =", round(result["fun"], 4))
print("Best angle =", round(result["x"][0], 4))
```

段代码中，首先设置了一个初始角度为 0.0，并调用 vqe_inst.vqe_run 方法进行最小化。vqe_run 方法接受 ansatz 量子程序、哈密顿量、初始参数、采样次数（在这个例子中没有采样，所以是 None）以及一个量子虚拟机 qvm。

然后，打印出最小化结果，这包括得到的最小能量（即哈密顿量的期望值在最优参数下的值）和达到最小能量的参数值（在这个例子中，这就是最佳角度）。

程序最后这部分的输出是：

```
Minimum energy = -1.0
Best angle = 3.1416
```

如您所见，优化器能够找到全局最小能量 $E = \langle\psi| H |\psi\rangle = -0.1$（以任意单位表示）的正确角度 $\theta = \pi$。

9.1.1. 噪声条件下变分量子本征求解器

VQE（变分量子本征求解器）算法是为了在近期的量子计算机上有效运行而设计的。因此，分析该算法在噪声环境下的表现，例如在 NISQ（噪声中等规模量子）处理器上，是非常重要的。在上文中，我们使用无噪声的 QVM（量子虚拟机）来模拟 VQE 中的电路。现在，我们可以考虑一个带有特定噪声模型的 QVM，并再次运行 VQE 算法。

在量子计算中，噪声是一个重要的考虑因素，因为现实的量子计算机都有一定的噪声水平，这可能会影响算法的性能。因此，我们不仅要在理想的无噪声环境中测试算法，还要在模拟实际噪声条件的环境中测试算法。

下面显示的代码块在 pyQuil 中设置了一个带有噪声的 QVM，并证明了它的确是带有噪声的。注意，这个程序是前一个程序的扩展，假设所有的包仍然被导入。

```
# 创建一个噪声模型，每个时间步的每个门都有10%的概率出现噪声
pauli_channel = [0.1, 0.1, 0.1]
noisy_qvm = api.QVMConnection(gate_noise=pauli_channel)

# 检查模拟器是否确实是噪声的
p = Program(X(0), MEASURE(0, 0))
res = noisy_qvm.run(p, [0], 10)
print(res)
```

在这段代码中，首先创建了一个噪声模型，该模型为每个时间步的每个门都设置了 10% 的概率出现 Pauli 噪声（分别是 X、Y、Z 门的噪声）。然后，使用这个噪声模型创建了一个带有噪声的量子虚拟机 noisy_qvm。

然后，为了检查这个模拟器是否确实是带有噪声的，创建了一个程序，该程序在第 0 个量子比特上应用了一个 X 门，然后对其进行测量。这个程序在 noisy_qvm 上运行 10 次，然后打印出结果。如果这个模拟器确实是带有噪声的，那么这 10 次的运行结果应该会有一些变化。

这个程序的输出（测量量子态 $|1\rangle$）证明了模拟器确实是有噪声的—否则，我们永远不会看到测量到的比特为 0。

```
"Outcome of NOT and MEASURE circuit on noisy simulator:"
[[0], [1], [1], [1], [0], [1], [1], [1], [1], [0]]
```

现在我们有了一个噪声模拟器，我们可以在噪声下运行 VQE 算法。在这里，我们修改了经典的优化器，从一个更大的单线开始，这样我们就不会卡在初始最小值。然后，我们可以看到与之前相同的景观图（能量与角度），但现在是在有噪声的情况下。

```
# 更新VQE实例中的最小化函数，使其以更大的初始单纯形开始
vqe_inst.minimizer_kwargs = {"method": "Nelder-mead",
                       "options":
                          {"initial_simplex": np.array([[0.0],
                              [0.05]]),
                           "xatol": 1.0e-2}
                          }
# 遍历一系列角度并通过采样绘制期望值
sampled = [vqe_inst.expectation(small_ansatz([angle]), hamiltonian,
    1000, noisy_qvm)
        for angle in angle_range]
# 绘制采样的期望值
plt.plot(angle_range, sampled, "-o")
# 绘图选项
plt.title("VQE on a Noisy Simulator")
plt.xlabel("Angle [radians]")
plt.ylabel("Expectation value")
plt.grid()
plt.show()
```

首先更新了 VQE 实例中的最小化方法和相关参数，然后遍历一系列角度，并在带有噪声的量子虚拟机（QVM）上运行 VQE，计算在各个角度下的哈密顿量期望值，并绘制出期望值随角度变化的图像。

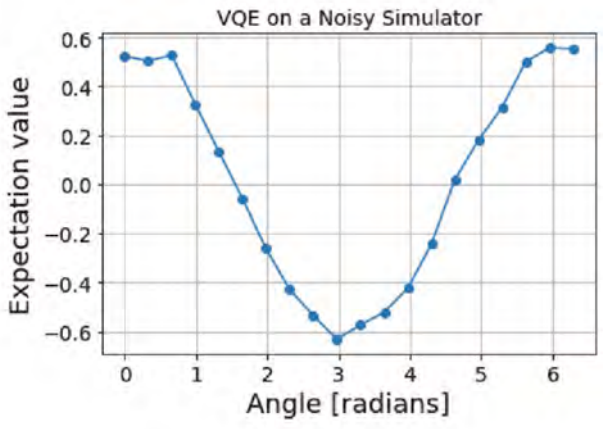


图 9.3: 通道噪声的模拟器上运行 VQE 的结果。

程序生成的示例图如图9.3所示。在这里，我们注意到整体的形状大致相同，但略有扭曲。曲线的最小值仍然接近最优值的位置，即 $\theta = \pi$，但能量的值在这里垂直偏移。在这里，最小能量约为 -0.6（以任意单位为单位），而在无噪声情况下，最小能量为 -1.0。

然而，由于最小值仍然出现在 $\theta = \pi$ 附近，VQE 对噪声表现出一定的鲁棒性。最优参数仍然可以找到，并且最小能量的垂直偏移可以在经典后处理中考虑。

Pauli 通道噪声不是唯一我们可以考虑的噪声模型。我们发现 VQE 在测量噪声方面具有与上述相同的鲁棒性——景观曲线具有相同的一般形状，最小值再次接近 $\theta = \pi$。

9.1.2. 更为复杂的 Ansatz（假设函数）

如前所述，更大的哈密顿量可能需要一个具有更多参数的 Ansatz 来更精确地近似基态波函数。在 pyQuil 中，我们可以通过添加更多的门来增加程序中的参数数量，如下所示：

```
# 定义一个具有两个参数的变分量子电路
def smallish_ansatz(params):
    """返回具有两个参数的变分量子电路."""
    return Program(RX(params[0], 0), RZ(params[1], 0))

# 打印具有两个门和两个参数的变分量子电路（使用示例值）
print("带有两个门和两个参数的变分量子电路（使用示例值）:")
print(smallish_ansatz([1.0, 2.0]))

# 获取一个VQE实例
vqe_inst = VQE(minimizer=minimize, minimizer_kwargs={'method': 'nelder-mead'})

# 执行最小化操作并返回最佳角度
initial_angles = [1.0, 1.0]
result = vqe_inst.vqe_run(smallish_ansatz, hamiltonian, initial_angles, None, qvm=qvm)

# 打印最小能量和最佳角度
print("\n最小能量␣=", round(result["fun"], 4))
print("最佳角度␣=", round(result["x"][0], 4))
```

在上面的程序中，我们创建了一个包含两个门和两个参数的 Ansatz，将其打印出来，然后使用该 Ansatz 运行了 VQE 算法。程序的一个示例输出如下：

```
"带有两个门和两个参数的变分量子电路（使用示例值）:"
RX(1.0) 0
RZ(2.0) 0
Minimum energy = -1.0
Best angle = 3.1416
```

在这里，我们看到最小化器能够使用新的 Ansatz 找到精确的基态能量。这是预期的——因为我们知道可以通过仅使用一个 Rx 门来最小化哈密顿量的期望值，所以第二个 Rz 门是多余的。然而，对于更大的非平凡哈密顿量，情况可能不是这样的，可能需要更多的参数。

在本节中，为了清晰地展示，我们使用简单的试验 Ansatz。通常，选择适当的 Ansatz 和良好的初始参数起始点对于成功实现 VQE 至关重要。随机生成的 Ansatz 可能会在大电路尺寸时梯度消失 [139]，因此使得参数优化极其困难，甚至是实际上不可能的。出于这些原因，实践中使用结构化的 Ansatz，如单位耦合簇或 QAOA，而不是参数化的随机量子电路。

9.2. 量子化学

现在，我们将探讨量子化学或更一般的量子模拟应用。在量子模拟中，我们试图模拟波函数在某个哈密顿量 H 下的动态演化，如薛定谔方程所示：

$$i\frac{\partial |\psi\rangle}{\partial t} = H|\psi\rangle \tag{9.3}$$

这是薛定谔方程的时间依赖形式。它描述了一个量子系统随着时间的演化。在这个方程中：

- i 是虚数单位。
- $\frac{\partial|\psi\rangle}{\partial t}$ 是量子态 $|\psi\rangle$ 关于时间 t 的导数，描述了随着时间推移，系统的状态如何变化。

- H 是哈密顿算子，表示系统的总能量。
- $|\psi\rangle$ 是量子系统的状态，包含了描述系统所有可能性的复振幅。

这个方程的物理含义是：系统的状态随时间的变化由系统的哈密顿算子（即系统的总能量）驱动。系统的哈密顿算子作用在当前的系统状态上，给出了系统状态随时间变化的速度。在给定初始条件（即初始的量子态 $|\psi\rangle$）后，我们可以通过解这个方程来预测系统在任何时间点的状态。

其中我们设置 $\bar{h} = 1$。易于编写时间演化算符

$$U(t) = exp(-iHt) \tag{9.4}$$

它通过将初始状态 t 演化到时间的最终状态 $|\phi 0\rangle$。

然而，即使哈密顿量是稀疏的，经典计算系统上的幺正时间演化[8]算符 $U(t)$ 的计算通常仍然非常困难。相反，利用量子计算进行量子模拟为我们提供了一种更简便的方法来计算幺正时间演化。量子模拟对于与时间有关的过程的经典模拟有着相同的功能。也就是说，它使我们能够分析复杂物理系统的动态，计算可观察量的性质，并利用这两者来进行新的预测或者与实验结果进行比较。例如，O'Malley 等人展示了利用 VQE 和 QPE 进行量子模拟 [101]，计算了氢分子的势能面[9]。

对于量子化学应用来说，模拟分子哈密顿量的量子模拟非常有用。在接下来的程序中，我们将使用 Cirq 和 OpenFermion[10]来演示如何模拟初始状态在哈密顿量作用下的演化。在这里，为了教学目的，我们将随机生成哈密顿量和初始状态。

现在，我们将逐步介绍这个程序。首先，我们导入必要的包并定义一些常量供我们的模拟使用。具体地，我们将定义量子位数 n、模拟的最终时间 t 以及随机数生成器的种子，以保证结果的可重复性。

```
# Imports
import numpy
import scipy
import cirq
```

[8] 幺正时间演化是指量子系统在哈密顿量作用下随时间演化的过程，其中演化算符是幺正的。根据量子力学中的薛定谔方程，幺正时间演化可以用幺正算符 $U(t)$ 来描述，它将初始的量子态 $|\psi(0)\rangle$ 演化到时间 t 后的量子态 $|\psi(t)\rangle$：

$$|\psi(t)\rangle = U(t)|\psi(0)\rangle$$

幺正时间演化保持量子态的归一性和内积，它是时间的反演对称的，也是可逆的。幺正时间演化算符满足幺正条件，即 $U(t)^\dagger U(t) = U(t)U(t)^\dagger = I$，其中 $U(t)^\dagger$ 是 $U(t)$ 的厄米共轭。

通过解薛定谔方程或者利用其他方法，可以计算出具体的幺正时间演化算符 $U(t)$，用于描述量子系统在给定的哈密顿量下随时间的演化。幺正时间演化在量子计算和量子模拟中起着重要的作用，用于模拟和探索量子系统的动态行为以及计算量子算法的时间复杂度。

[9] 氢分子的势能面指的是描述氢分子内部原子之间相互作用的势能函数。它是一个二维或三维的势能曲面，其中自变量通常是原子之间的核间距离（例如，氢原子核之间的距离）。

在氢分子中，两个氢原子核（质子）围绕共同的电子云运动。这两个原子核之间的相互作用可以通过量子力学中的薛定谔方程和分子结构理论来描述。势能面可以用势能函数的形式表示，它描述了在不同核间距离下的氢分子的总能量。

势能面的形状和特征直接影响着氢分子的振动和旋转运动以及其光谱特性。通过计算势能面，可以预测氢分子的能级结构、振动频率、转动惯量等物理性质，并与实验结果进行比较。势能面的计算对于理解和研究氢分子的结构、动力学行为以及化学反应过程非常重要。

[10] OpenFermion 是一个开源的量子化学软件包，用于处理和模拟量子化学问题。它提供了一套工具和算法，可以用于处理和分析分子的结构、性质和反应。

OpenFermion 是基于 Python 编程语言的软件包，它的设计目标是为量子计算和量子模拟提供方便的接口和功能。它提供了一系列用于处理量子化学信息的工具，包括处理分子结构、生成量子比特的量子比特哈密顿量、构建量子化学哈密顿量、计算能量和梯度、执行变分量子算法等。

OpenFermion 还集成了其他量子计算框架，如 Cirq 和 PyQuil，使得在量子计算环境中使用量子化学工具更加方便。它还提供了与常用量子化学软件包的接口，例如 Psi4 和 PySCF，以便获取和处理量子化学数据。

OpenFermion 的目标是为研究人员和开发者提供一个强大而灵活的工具，用于研究和解决量子化学问题，并推动量子计算在化学领域的应用和发展。它在量子化学模拟、量子化学算法开发和优化以及量子化学问题的求解方面具有广泛的应用价值。

```
import openfermion
import openfermioncirq
# 设置量子比特的数量、模拟时间和种子，以实现可重复性
n_qubits = 3
simulation_time = 1.0
random_seed = 8317
```

在下面的代码块中，我们生成一个随机的矩阵形式的哈密顿量。为了运行任何带有这个哈密顿量的量子电路，它首先必须用量子算符来表示。接下来的几行代码使用 OpenFermion 的功能来完成这个任务。

```
# 生成随机单体算子
T = openfermion.random_hermitian_matrix(n_qubits, seed=random_seed)
print("Hamiltonian:", T, sep="\n")
# 计算OpenFermion "FermionOperator "形式的哈密顿
H = openfermion.FermionOperator()
for p in range(n_qubits):
   for q in range(n_qubits):
      term = ((p, 1), (q, 0))
      H += openfermion.FermionOperator(term, T[p, q])
print("\nFermion operator:")
print(H)
```

以上代码段是使用 OpenFermion 软件包在 Python 中生成随机的单体算符并计算对应的费米算符形式的哈密顿算符。让我们逐行解释代码：

```
# 生成随机单体算子
T = openfermion.random_hermitian_matrix(n_qubits, seed=random_seed)
print("Hamiltonian:", T, sep="\n")
```

这部分代码生成了一个随机的 $n_{qubits} \times n_{qubits}$ 大小的厄米矩阵，即随机的单体算符。这里的 n_qubits 是量子位数，random_seed 是用于生成随机数的种子。生成的矩阵 T 表示了随机单体算符的系数矩阵。通过打印输出，可以查看生成的随机单体算符的值。

```
# 计算OpenFermion "FermionOperator"形式的哈密顿
H = openfermion.FermionOperator()
for p in range(n_qubits):
   for q in range(n_qubits):
      term = ((p, 1), (q, 0))
      H += openfermion.FermionOperator(term, T[p, q])
print("\nFermion operator:")
print(H)
```

这部分代码使用生成的随机单体算符 T 来构建对应的费米算符形式的哈密顿算符。通过使用 OpenFermion 提供的 FermionOperator 类，我们可以逐个元素遍历 T 矩阵，并将每个非零元素添加到费米算符 H 中。最后，我们打印输出费米算符 H 的结果，展示了哈密顿算符的费米算符形式。

这段代码的目的是演示使用 OpenFermion 进行量子化学计算中的哈密顿算符处理。通过生成随机单体算符并计算对应的费米算符形式的哈密顿算符，我们可以对量子化学系统进行建模和计算。

这一部分程序的输出是：

```
Hamiltonian:
[[ 0.53672126+0.j -0.26033703+3.32591737j 1.34336037+1.54498725j]
 [-0.26033703-3.32591737j -2.91433037+0.j -1.52843836+1.35274868j]
 [ 1.34336037-1.54498725j -1.52843836-1.35274868j 2.26163363+0.j ]]
Fermion operator:
```

```
(0.5367212624097257+0j) [0^ 0] +
(-0.26033703159240107+3.3259173741375454j) [0^ 1] +
(1.3433603748462144+1.544987250567917j) [0^ 2] +
(-0.26033703159240107-3.3259173741375454j) [1^ 0] +
(-2.9143303700812435+0j) [1^ 1] +
(-1.52843836446248+1.3527486791390022j) [1^ 2] +
(1.3433603748462144-1.544987250567917j) [2^ 0] +
(-1.52843836446248-1.3527486791390022j) [2^ 1] +
(2.261633626116526+0j) [2^ 2]
```

第一部分显示哈密顿量的矩阵形式，接下来的部分显示矩阵在 OpenFermion 算符形式下的表示。在这里，OpenFermion 符号 pq□□□□□□□□□□p□q□□□□□□□□□□□□□□□□□□□□□□$\mathsf{a}_p^\dagger a_q$□ 的乘积。

现在，我们已经将哈密顿量转换为可用的形式，可以开始构建我们的电路。在量子模拟算法中，通常首先将电路旋转到哈密顿量的本征态基。这是通过（经典地）对角化哈密顿量，然后使用 OpenFermion 构建执行这个基变换的电路来完成的。

```
# Diagonalize T and obtain basis transformation matrix (aka "u")
eigenvalues, eigenvectors = numpy.linalg.eigh(T)
basis_transformation_matrix = eigenvectors.transpose()
# Initialize the qubit register
qubits = cirq.LineQubit.range(n_qubits)
# Rotate to the eigenbasis
inverse_basis_rotation = cirq.inverse(
   openfermioncirq.bogoliubov_transform(qubits,
       basis_transformation_matrix)
circuit = cirq.Circuit.from_ops(inverse_basis_rotation)
```

现在我们可以添加对应于哈密顿进化的门。由于我们是在哈密顿的特征基上，这对应于 Pauli-Z 旋转的对角线算子，其中旋转角度与特征值和最终模拟时间成正比。最后，我们改变基数，回到计算基数。

```
# Add diagonal phase rotations to circuit
for k, eigenvalue in enumerate(eigenvalues):
   phase = -eigenvalue * simulation_time
   circuit.append(cirq.Rz(rads=phase).on(qubits[k]))
# Finally, change back to the computational basis
basis_rotation = openfermioncirq.bogoliubov_transform(
   qubits, basis_transformation_matrix
)
circuit.append(basis_rotation)
```

我们现在在我们的量子电路中构建时间演化算符。在下面的代码中，我们首先获取一个随机的初始状态。请注意，本程序仅供演示目的。在实际情况下，我们会使用许多非随机的技术来确定初始状态：

```
# Initialize a random initial state
initial_state = openfermion.haar_random_vector(
   2 ** n_qubits, random_seed).astype(numpy.complex64)
```

现在我们使用矩阵指数计算时间演化，然后使用 QC 模拟器进行模拟。在使用两种方法获取最终状态后，我们计算两者的保真度（重叠的平方）并打印出其值：

```
# Numerically compute the correct circuit output
hamiltonian_sparse = openfermion.get_sparse_operator(H)
exact_state = scipy.sparse.linalg.expm_multiply(
```

```
    -1j * simulation_time * hamiltonian_sparse, initial_state
)
# Use Cirq simulator to apply circuit
simulator = cirq.google.XmonSimulator()
result = simulator.simulate(circuit, qubit_order=qubits,
                     initial_state=initial_state)
simulated_state = result.final_state
# Print final fidelity
fidelity = abs(numpy.dot(simulated_state,
    numpy.conjugate(exact_state)))**2
print("\nfidelity␣=", round(fidelity, 4))
```

本节代码的输出是：

```
fidelity = 1.0
```

上述结果表明，我们的量子电路与解析演化完全相同！当然，对于更大的系统，解析演化无法计算，我们必须仅依赖量子计算机。这个小型的原理性计算表明了这种方法的有效性。最后，我们提到 Cirq 具有将此量子电路编译为 Google 的 Xmon 体系结构量子计算机以及 IBM 的量子计算机的功能。下面的代码段显示了如何执行此操作：

```
# Compile the circuit to Google's Xmon architecture
xmon_circuit = cirq.google.optimized_for_xmon(circuit)
print("\nCircuit␣optimized␣for␣Xmon:")
print(xmon_circuit)
# Print out the OpenQASM code for IBM's hardware
print("\nOpenQASM␣code:")
print(xmon_circuit.to_qasm())
```

下面，我们包含了 Cirq 为此电路生成的 OpenQASM 代码：

```
// Generated from Cirq v0.4.0
OPENQASM 2.0;
include "qelib1.inc";
// Qubits: [0, 1, 2]
qreg q[3];
u2(pi*-1.0118505646, pi*1.0118505646) q[2];
u2(pi*-1.25, pi*1.25) q[1];
u2(pi*-1.25, pi*1.25) q[0];
cz q[1],q[2];
u3(pi*-0.1242949803, pi*-0.0118505646, pi*0.0118505646) q[2];
u3(pi*0.1242949803, pi*-0.25, pi*0.25) q[1];
cz q[1],q[2];
u3(pi*-0.3358296941, pi*0.4881494354, pi*-0.4881494354) q[2];
u3(pi*-0.5219350773, pi*1.25, pi*-1.25) q[1];
cz q[0],q[1];
u3(pi*-0.328242091, pi*0.75, pi*-0.75) q[1];
u3(pi*-0.328242091, pi*-0.25, pi*0.25) q[0];
cz q[0],q[1];
u3(pi*-0.2976584908, pi*0.25, pi*-0.25) q[1];
u3(pi*-0.7937864503, pi*0.25, pi*-0.25) q[0];
cz q[1],q[2];
u3(pi*-0.2326621647, pi*-0.0118505646, pi*0.0118505646) q[2];
u3(pi*0.2326621647, pi*-0.25, pi*0.25) q[1];
cz q[1],q[2];
u3(pi*0.8822298425, pi*0.4881494354, pi*-0.4881494354) q[2];
u3(pi*-0.2826706001, pi*0.25, pi*-0.25) q[1];
cz q[0],q[1];
```

```
u3(pi*-0.328242091, pi*0.75, pi*-0.75) q[1];
u3(pi*-0.328242091, pi*-0.25, pi*0.25) q[0];
cz q[0],q[1];
u3(pi*-0.3570821075, pi*0.25, pi*-0.25) q[1];
u2(pi*-0.25, pi*0.25) q[0];
rz(pi*0.676494835) q[0];
cz q[1],q[2];
u3(pi*0.1242949803, pi*0.9881494354, pi*-0.9881494354) q[2];
u3(pi*-0.1242949803, pi*0.75, pi*-0.75) q[1];
cz q[1],q[2];
u2(pi*-0.0118505646, pi*0.0118505646) q[2];
u2(pi*-0.25, pi*0.25) q[1];
rz(pi*-0.4883581348) q[1];
rz(pi*0.5116418652) q[2];
```

9.3. 量子近似优化算法 Quantum Approximate Optimization Algorithm QAOA

虽然前面介绍的两种量子计算方法主要用于物理和化学应用，但量子近似优化算法（Quantum Approximate Optimization Algorithm，QAOA）主要用于一般的优化问题。Farhi 等人引入了 QAOA 来处理这些问题 [140, 140]。在该算法中，目标是最大化或最小化一个代价函数，

$$C(b) = \sum_{a=1}^{m} C_a(b) \tag{9.5}$$

该函数可以表示为 m 个子句 $C_a(b)$ 的总和，其中 $b \in \{0,1\}^n$ 是一个位串，或者等效地，$z_i \in \{-1,+1\}^n$ 是自旋。这是因为位串和自旋之间存在着双射映射关系，即 Bits、band 和 spins 之间通过双射映射 $z = 1 - 2b \iff b = (1-z)/2$ 相关联。因此，任何涉及位的问题都可以转化为涉及自旋的问题，反之亦然。

MaxCut 是一个可以在常规图上使用 QAOA 的问题的例子 [140]。其代价函数可以用自旋形式表示为：

$$C(z) = \frac{1}{2} \sum_{(i,j)} (1 - z_i z_j) \tag{9.6}$$

在这里，求和是在图中的边 $\langle i,j \rangle$ 上进行的，每个子句[11]$(1 - z_i z_j)$ 只有在自旋 z_i 和 z_j 是反平行（即具有不同的值）时才会对代价产生非零项。需要注意的是，" 反平行"[12]是指两个方向完全相反的状态或物体。

更一般情况下，这个问题考虑了每条边之间的任意权重 w_{ij}。这意味着具有较大权重 w_{ij} 的子句会对代价产生更高的贡献。我们在下文中将讨论这种情况。

[11]在逻辑和数学推理中，" 子句" 指的是一个由逻辑变量或命题构成的布尔表达式的一部分。布尔表达式可以由多个子句组成，而每个子句又可以包含一个或多个逻辑变量，用于描述逻辑条件或命题之间的关系。

在 MaxCut 问题中，子句是用来描述图的边的关系的一部分。每个子句表示两个顶点之间的连接关系，其中的布尔变量表示两个顶点的自旋状态。对于每条边 $\langle i,j \rangle$，子句 $(1 - z_i z_j)$ 表示当顶点 i 和顶点 j 的自旋状态反平行时，子句取值为 1；否则，子句取值为 0。

因此，MaxCut 问题的代价函数可以表示为多个子句的总和，每个子句的取值取决于对应边上的自旋状态。通过优化这个代价函数，我们可以寻找最大化切割的方案，即将图的顶点分为两个集合，使得切割边的权重总和最大化。

[12]" 反平行" 是指两个方向完全相反的状态或物体。在物理学中，如果两个向量的方向相反，它们被称为反平行。例如，如果一个向量指向东方，而另一个向量指向西方，它们就是反平行的。在磁学中，当两个磁场或磁矩的方向相反时，它们也被称为反平行。

为了将问题转化为量子近似优化算法的一般输入形式，我们将每个自旋升级为一个 Pauli-Z 算符（它具有本征值 ±1）。代价函数可以写成一个代价哈密顿量的形式：

$$C \equiv H_C = \frac{1}{2}\sum_{(i,j)} w_{ij}(I - \sigma_z^{(i)}\sigma_z^{(j)}) \tag{9.7}$$

在这里，I 表示单位算符，$\sigma_z^{(i)}$ 表示第 i 个自旋上的 Pauli-Z 算符。可以观察到，在计算基下，这个代价哈密顿量是对角化的。这个对角化形式是量子近似优化算法的一般输入形式，下面我们将进一步讨论它。

QAOA 的步骤如下。给定一个形如公式9.5的代价哈密顿量 $H_C \equiv H$，我们定义幺正算符：

$$U(H_C, \gamma) := e^{i\gamma H_C} = \prod_{\alpha=1}^{m} e^{i\gamma C_\alpha} \tag{9.8}$$

其中，γ 是一个参数。值得注意的是，等式的第二部分成立的原因是每个子句 C_α 在计算基础下是对角的，因此对于所有 $\alpha, \beta \in \{1, \ldots, m\}$，都有 $[C_\alpha, C_\beta] = 0$。此外，根据前面的讨论，我们可以将这种情况理解为代价哈密顿量 H_C 在时间 γ 下的模拟（即演化）。由于每个子句 C_α 都只能取整数值，我们可以将“时间”γ 限制在 0 和 2π 之间。因此，我们也可以将这个参数 γ 视为旋转角度。

这个等式详述了量子近似优化算法（QAOA）中的一个关键步骤。其中，H_C 是问题的代价哈密顿量，γ 是一个参数，C_α 是代价函数的子句。

等式定义了一个量子门操作 $U(H_C, \gamma)$，它作用于量子比特系统，通过指数运算将代价哈密顿量 H_C 乘以复数相位因子 $e^{i\gamma}$。

具体地，等式中的乘积符号 $\prod$ 表示了一个从 $\alpha = 1$ 到 m 的乘积，每一项都是一个指数运算 $e^{i\gamma C_\alpha}$。这意味着对于每个子句 C_α，我们都将其与相位因子 $e^{i\gamma}$ 相乘。

换句话说，我们对每个子句 C_α 乘以相位因子 $e^{i\gamma}$ 并求指数，然后将这些结果相乘，这样我们就得到了整个代价哈密顿量的量子门操作 $U(H_C, \gamma)$。这个量子门操作是 QAOA 算法中的关键部分，用于在量子计算机上处理优化问题。

接下来，我们定义混合哈密顿量的算符 $B \equiv H_B$：

$$B \equiv H_B = \sum_{j=1}^{n} \sigma_x^{(j)} \tag{9.9}$$

其中，$\sigma_x^{(j)}$ 是在自旋 j 上的 Pauli-X 算符。然后，我们形成对应的幺正算符：

$$U(H_B, \beta) := e^{i\beta B} = \prod_{j=1}^{n} e^{i\beta\sigma_x^{(j)}} \tag{9.10}$$

请注意，这里的第二个等式成立，是因为混合哈密顿量中的所有项都是对易的。我们可以将 $e^{i\beta\sigma_x^{(j)}}$ 视为一个绕自旋 j 上的 x 轴旋转 2β 的旋转，其中 $0 \leqq \beta < \pi$。

现在，我们将幺正算符 $U(H_C, \gamma)$ 和 $U(H_B, \beta)$ 交替应用 p 次，形成 QAOA 电路。这个电路的参数是 γ 和 β。在经典计算机上，我们可以使用一些优化算法，如梯度下降，来确定这些参数，以最大化或最小化代价函数 $C_H(\beta, \gamma)$。

QAOA 是一个可广泛应用于各类优化问题的量子算法。然而，为了将其实际应用于特定的问题，我们需要对 QAOA 进行特定的调整。

有了这些定义，我们可以说明 QAOA 量子部分的步骤：

1. 从初始状态开始，该状态是所有比特串（自旋）的等概率叠加态：

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{b\in\{0,1\}^n} |b\rangle \tag{9.11}$$

通过对每个量子比特应用哈达玛门 $H^{\otimes n}|0\rangle^{\otimes n}$ 来实现。

2. 通过实施 $U(H_C,\gamma)$，让代价哈密顿量 H_C 随角度 γ 演化。
3. 通过实施 $U(H_B,\beta)$，让代价哈密顿量 H_B 随角度 β 演化。
4. 重复步骤（2）和（3）p 次，每一步使用不同的参数 γ_i,β_i，形成状态

$$|\gamma,\beta\rangle := \prod_{i=1}^{p} U(H_B,\beta_i)U(H_C,\gamma_i)\,|s\rangle \tag{9.12}$$

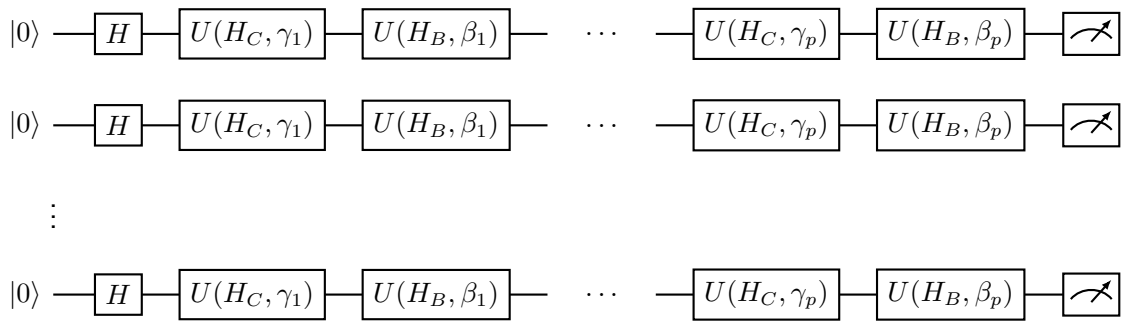
5. 在计算基础上进行测量，以计算此状态下 H_C 的期望值：

$$F_p(\gamma,\beta) := \langle\gamma,\beta|\,|H_C|\,|\gamma,\beta\rangle \tag{9.13}$$

6. 使用（经典的）优化算法来（近似）计算 $F_p(\gamma,\beta)$ 的最大或最小值。另外，如果你有其他方法来确定最佳角度，你也可以使用这些方法。
7. 从电路 9.12 的输出分布中采样，得到一组比特串 b。最可能的比特串编码了代价函数的近似最优值。

Algorithm 5 量子优化算法

1: 初始化一个状态，使其为所有比特串（自旋）的等概率叠加态：$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_{b\in\{0,1\}^n} |b\rangle$ 通过对每个量子比特应用哈达玛门 $H^{\otimes n}|0\rangle^{\otimes n}$ 来实现。
2: 通过实施 $U(H_C,\gamma)$，让代价哈密顿量 H_C 随角度 γ 演化。
3: 通过实施 $U(H_B,\beta)$，让代价哈密顿量 H_B 随角度 β 演化。
4: **for** $i = 1,\ldots,p$ **do**
5: 使用不同的参数 γ_i,β_i，重复步骤（2）和（3），形成状态 $|\gamma,\beta\rangle := \prod_{i=1}^{p} U(H_B,\beta_i)U(H_C,\gamma_i)\,|s\rangle$
6: **end for**
7: 在计算基础上进行测量，以计算此状态下 H_C 的期望值：$F_p(\gamma,\beta) := \langle\gamma,\beta|\,|H_C|\,|\gamma,\beta\rangle$
8: 使用（经典的）优化算法来（近似）计算 $F_p(\gamma,\beta)$ 的最大或最小值。另外，如果你有其他方法来确定最佳角度，你也可以使用这些方法。
9: 从电路 9.12 的输出分布中采样，得到一组比特串 b。最可能的比特串编码了代价函数的近似最优值。



绝热定理[13]表明，即使在受到扰动的情况下，系统仍然保持在其本征态[14]，只要这个扰动足够缓慢和渐进，并且该态的本征值与系统的其他本征值（其谱系）之间有一个间隙 [141, 142, 143]。换句话说，如果我们有一个处于测量状态的系统，并且该状态与系统的其他可能状态有足够的间隙，那么如果我们足够慢地扰动系统，它就不会跳跃到另一个本征态。可以使用绝热定理来证明：

$$\lim_{p\to\infty} \max_{\gamma,\beta} F_p(\gamma,\beta) = \max_b C(b) \tag{9.14}$$

公式9.14中的 $F_p(\gamma,\beta)$ 是 QAOA 在给定参数 γ 和 β 下，经过 p 轮迭代后得到的目标函数的期望值。这个期望值是在量子态 $|\gamma,\beta\rangle$ 下对应的代价哈密顿量 H_C 的期望值。$\max_{\gamma,\beta} F_p(\gamma,\beta)$ 表示的是在所有可能的 γ 和 β 参数值下，这个期望值能够达到的最大值。$C(b)$ 是目标函数，它是一个关于比特串 b 的函数，QAOA 算法的目标就是找到能够最大化（或最小化）这个函数的比特串 b。$\max_b C(b)$ 表示的是在所有可能的比特串 b 下，目标函数能够达到的最大值，也就是问题的最优解。

这个公式9.14的含义是说，如果我们可以进行无限多次的 QAOA 迭代（即 $p \to \infty$），并且我们能够找到使 $F_p(\gamma,\beta)$ 达到最大值的参数 γ 和 β，那么我们就能够找到问题的最优解。换句话说，QAOA 在理论上是可以找到问题的最优解的，只要我们可以进行足够多的迭代，并且能够找到最优的参数。

9.3.1. QAOA 的实施实例

为了更好地理解 QAOA 如何工作，我们现在转向一个实现。在这个例子中，我们将横向场伊辛模型[15]作为代价哈密顿量：

$$H_C = -\sum_{\langle i,j\rangle} J_{ij}\sigma_z^{(i)}\sigma_z^{(j)} - \sum_i h_i\sigma_x^{(i)} \tag{9.15}$$

为了简化呈现，我们将横向场[16]系数设为零 ($h_i = 0$) 并将每个相互作用系数设为 1 $J_{ij} = 1$。哈

[13]绝热定理是量子力学中的一个重要定理，它描述了物理系统在缓慢改变其哈密顿量（即系统的总能量表达式）时的行为。这个定理基本上说明，如果一个量子系统开始于其哈密顿量的某个本征态，那么在哈密顿量缓慢变化的过程中，系统将会保持在对应的本征态。这就是所谓的“绝热演化”。

更具体地说，假设一个系统开始于其哈密顿量 H 的一个本征态，然后我们缓慢地改变哈密顿量 H 到一个新的哈密顿量 H'。如果这个过程足够缓慢，那么系统将在过程结束时处于 H' 的对应本征态。

这个定理在量子计算和量子信息理论中有着广泛的应用。例如，量子绝热计算就是根据这个定理设计的，其基本思想是从一个容易准备和处理的初始哈密顿量开始，然后缓慢地改变它，使之逼近一个代表计算问题的目标哈密顿量，从而找到问题的解。

[14]在量子力学中，" 本征态" 是一个特别的状态，它对应于物理系统的一个特定的可观测量（如位置、动量、能量等）具有确定的值。

在形式上，本征态被定义为某个物理算符（对应于物理量）的本征函数。例如，考虑一个算符 A 和一个状态向量（或波函数）ψ，如果它们满足以下的本征方程：

$$A\psi = a\psi$$

那么我们就说 ψ 是算符 A 的一个本征态，对应的本征值是 a。这意味着如果我们在状态 ψ 下测量物理量 A，我们总会得到确定的结果 a。

本征态是量子力学的一个关键概念，因为它们为描述和理解量子系统提供了一个基础。例如，一个量子系统的任何状态都可以被表示为系统哈密顿量的本征态的叠加，这在量子力学的很多应用中都是很重要的。

[15]横向场伊辛模型" 是指在统计物理中使用的伊辛模型的一种变体，它考虑了物体中的磁场或其他横向力的影响。

一般来说，伊辛模型是一个理论模型，用于描述固体物理中的磁性行为，具体来说，是相转变和铁磁性的行为。这个模型假设一个材料可以被看作是一个磁性原子或粒子的网格，每个粒子可以有两种状态，通常被描述为" 上" 和" 下"，或者"+1" 和"-1"。

在最简单的伊辛模型中，粒子只与其最近邻的粒子相互作用。横向场伊辛模型在这个基础上增加了一个额外的项，这个项代表了粒子受到的横向力，这通常是一个磁场，其方向垂直于粒子的" 上" 和" 下" 的状态。这个模型在量子计算和量子信息理论中特别有用，因为它可以描述一个量子比特（qubit）的行为，这是量子计算中的基本单元。

[16]" 横向场" 是指垂直于某一特定方向的场。其用法通常取决于上下文。

例如，在物理学中，如果你正在研究一个具有确定方向的系统，比如一维的磁性材料，其中的粒子可能有" 上" 和" 下" 的自旋状态，那

密顿量可以通过直接的方式进行修改以对其进行推广，但在初次接触 QAOA 时，这些细节并不重要。另一个原因是该系统在解析上是平凡的——因此，我们可以将 QAOA 找到的解与精确解进行比较。通过做这些简化，我们的代价哈密顿量的形式为

$$H_C = -\sum_{\langle i,j \rangle} \sigma_z^{(i)} \sigma_z^{(j)} \tag{9.16}$$

我们将考虑的图（即，自旋的排列）是在一个 2D 网格上的最近邻配置。因此，我们需要一种方式来实现酉算子

$$U(H_C, \gamma) := e^{-iH_C\gamma} = \prod_{\langle i,j \rangle} e^{i\gamma z_i z_j} \tag{9.17}$$

为了简化，从这里开始我们将用 Z_i 替换 $\sigma_z^{(i)}$。为了实现这个完整的酉算子，我们需要一系列的门来实现每个 $e^{i\gamma z_i z_j}$ 项，其中 i 和 j 是图中的邻居。将其重新缩放并考虑实现酉算子 $e^{i\pi\gamma z_i z_j}$ 会更加方便。要理解如何做到这一点，注意到算子 $Z \oplus Z$ 在计算基础上是对角线的，因此 $e^{i\pi\gamma Z\otimes Z}$ 只是每个对角元素的指数（乘以 $i\pi\gamma$）。

$$exp(i\pi\gamma Z \otimes Z) = \begin{bmatrix} e^{i\pi\gamma} & 0 & 0 & 0 \\ 0 & e^{i\pi\gamma} & 0 & 0 \\ 0 & 0 & e^{i\pi\gamma} & 0 \\ 0 & 0 & 0 & e^{i\pi\gamma} \end{bmatrix} \tag{9.18}$$

为了对如何用标准门来实现这个算子有一些直观的理解，注意到控制-Z 门是对角线的，其中 $C(Z) = \text{diag}(1, 1, 1, -1)$。写作 $-1 = e^{i\pi}$，我们看到 $C(Z) = \text{diag}(1, 1, 1, e^{i\pi})$ 因此

$$C(Z^\gamma) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\pi\gamma} \end{bmatrix} \tag{9.19}$$

这给了我们在最终的酉算子 9.18 中我们想要实现的一个对角项。为了得到其他项，我们可以在适当的量子位上应用 X 算子。例如，

$$(I \otimes X) \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{i\pi\gamma} \end{bmatrix} (I \otimes X) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & e^{i\pi\gamma} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9.20}$$

我们可以继续这种方式来得到所有四个对角元素，然后通过简单地将它们相乘（利用对角矩阵的乘积是对角线的事实）得到完整的酉算子 9.18。在下面的 Cirq 代码中，我们编写一个函数，它给我们一个实现酉算子 $e^{i\pi\gamma Z_i Z_j}$ 的电路。然后，我们通过打印出一个例子量子比特 i 和 j 的电路以及一个任意值，并确保这个电路的酉矩阵是我们所期望的，来测试我们的函数。

么”横向场”通常指的是垂直于这个方向的场。如果你向这样一个系统施加一个磁场，并且这个磁场的方向垂直于粒子的自旋方向（也就是说，它是横向的），那么这个磁场就会影响粒子的行为。

在电磁学中，”横向场”可能指的是垂直于传播方向的电场或磁场，这是在描述电磁波（如光）的性质时常用的术语。

在以上这些情况中，”横向场”通常涉及到一个额外的力或影响，这个力或影响垂直于系统中粒子的某些特性或行为的主要方向。

```
# 导入库
import numpy as np
import matplotlib.pyplot as plt
import cirq

# 函数实现在量子比特a, b上以角度gamma的ZZ门
def ZZ(a, b, gamma):
    """返回一个实现exp(-i \pi \gamma Z_i Z_j)的电路。"""
    # 获取一个电路
    circuit = cirq.Circuit()

    # 提供第四个对角线组件
    circuit.append(cirq.CZ(a, b)**gamma)

    # 提供第三个对角线组件
    circuit.append([cirq.X(b), cirq.CZ(a,b)**(-1 * gamma), cirq.X(b)])

    # 提供第二个对角线组件
    circuit.append([cirq.X(a), cirq.CZ(a,b)**-gamma, cirq.X(a)])

    # 提供第一个对角线组件
    circuit.append([cirq.X(a), cirq.X(b), cirq.CZ(a,b)**gamma,
                    cirq.X(a), cirq.X(b)])

    return circuit

# 确保电路产生正确的矩阵
qreg = cirq.LineQubit.range(2)
zzcirc = ZZ(qreg[0], qreg[1], 0.5)
print("Circuit for ZZ gate:", zzcirc, sep="\n")
print("\nUnitary of circuit:", zzcirc.to_unitary_matrix().round(2),
sep="\n")
```

这段代码的输出如下：

```
Circuit for ZZ gate:
0: ---@-----------@------------X---@--------X---X---@-------X---
|||||
1: ---@^0.5---X---@^-0.5---X-------@^-0.5-------X---@^0.5---X---
Unitary of circuit:
[[0.+1.j 0.+0.j 0.+0.j 0.+0.j]
 [0.+0.j 0.-1.j 0.+0.j 0.+0.j]
 [0.+0.j 0.+0.j 0.-1.j 0.+0.j]
 [0.+0.j 0.+0.j 0.+0.j 0.+1.j]]
```

通过与 9.18 比较，我们可以看到这个电路确实实现了所需的酉算子。注意，这个电路并不是最优的——一个简单的简化是移除对量子比特 0 的连续 X 操作，还有其他可能的优化。然而，这样的优化在这里不会关心。

在下一个代码块中，我们定义了一个 2x2 的量子比特网格。

```
ncols = 2
nrows = 2
qreg = [[cirq.GridQubit(i,j) for j in range(ncols)] for i in
    range(nrows)]
```

然后我们编写函数来实现算子 $U(H_C, \gamma)$ 和 $U(H_B, \beta)$

```
# 实现代价哈密顿量的函数
```

```
def cost_circuit(gamma):
    """返回代价哈密顿量的电路。"""
    circ = cirq.Circuit()
    for i in range(nrows):
        for j in range(ncols):
            if i < nrows - 1:
                circ += ZZ(qreg[i][j], qreg[i + 1][j], gamma)
            if j < ncols - 1:
                circ += ZZ(qreg[i][j], qreg[i][j + 1], gamma)
    return circ

# 实现混合哈密顿量的函数
def mixer(beta):
  """生成 U(H_B, beta) 层（混合层）"""
  for row in qreg:
    for qubit in row:
      yield cirq.X(qubit)**beta
```

这些函数让我们能够构造整个 QAOA (量子近似优化算法) 电路。下面的函数为任意数量 p 的参数构建这个电路。

```
# 构建 QAOA 电路的函数
def qaoa(gammas, betas):
    """返回一个 QAOA 电路。"""
    circ = cirq.Circuit()

    # 在每个量子比特上添加 Hadamard 门
    circ.append(cirq.H.on_each(*[q for row in qreg for q in row]))

    # 按照参数的数量，添加代价和混合层
    for i in range(len(gammas)):
        circ += cost_circuit(gammas[i])
        circ.append(mixer(betas[i]))

    return circ
```

现在，我们可以为给定的参数集构建我们的 QAOA 电路，我们可以计算在最终状态 9.12 下的代价哈密顿量的期望值。为了简便，我们使用 Cirq 的能力来访问波函数以计算这个期望值，而不是从电路本身进行采样。下面的函数展示了如何在应用电路后访问波函数：

```
def simulate(circ):
    """返回应用电路后的波函数。"""
    sim = cirq.Simulator()
    return sim.simulate(circ).final_state
```

下一个函数使用波函数来评估期望值：

```
def energy_from_wavefunction(wf):
    """从波函数计算伊辛模型的每个位点的能量。"""
    # Z 是一个 (n_sites x 2**n_sites) 数组。每一行包括在算符中的 2**n_sites 个非零项，
    # 这个算符是一个量子比特上的 Pauli-Z 矩阵乘以其他量子比特上的单位矩阵。第 (i*n_cols + j) 行对应
        的是量子比特 (i,j)。
    Z = np.array([(-1)**(np.arange(2**nsites) >> i)
                  for i in range(nsites-1,-1,-1)])
    # 创建对应于所有最近邻量子比特对之间的相互作用能量的算符
    ZZ_filter = np.zeros_like(wf, dtype=float)
    for i in range(nrows):
        for j in range(ncols):
```

```
        if i < nrows-1:
            ZZ_filter += Z[i*ncols + j]*Z[(i+1)*ncols + j]
        if j < ncols-1:
            ZZ_filter += Z[i*ncols + j]*Z[i*ncols + (j+1)]
    # 能量的期望值除以位点的数量
    return -np.sum(np.abs(wf)**2 * ZZ_filter) / nsites
```

最后，为了方便，我们定义了一个函数，可以直接从一组参数计算能量/成本。这个函数使用参数来构建电路，然后获取最终状态的波函数，最后使用前面的函数计算能量/成本。

```
def cost(gammas, betas):
    """返回问题的成本函数。"""
    wavefunction = simulate(qaoa(gammas, betas))
    return energy_from_wavefunction(wavefunction)
```

这些函数为 QAOA 提供了设置，现在我们可以优化参数以最小化成本。出于教学目的，我们实现了 $p = 1$ 层的 QAOA，并进行了网格搜索，绘制了每个参数 γ 和 β 的二维成本景观[17]。下面给出了在一系列参数上进行网格搜索的函数：

```
def grid_search(gammavals, betavals):
    """在所有参数值上进行网格搜索。"""
    # 创建一个全零的二维矩阵来存储每组参数的成本值
    costmat = np.zeros((len(gammavals), len(betavals)))
    # 遍历所有的gamma值
    for (i, gamma) in enumerate(gammavals):
        # 遍历所有的beta值
        for (j, beta) in enumerate(betavals):
            # 计算给定参数的成本，并存储在相应的位置上
            costmat[i, j] = cost([gamma], [beta])
    # 返回成本矩阵
    return costmat
```

最后，这是在主脚本中使用这个函数并绘制成本景观的代码：

```
# 获取一系列参数
gammavals = np.linspace(0, 1.0, 50)  # 生成从0到1.0的50个等差数列，作为gamma参数的取值范围
betavals = np.linspace(0, np.pi, 75)  # 生成从0到 的75个等差数列，作为beta参数的取值范围

# 使用网格搜索在所有参数值上计算成本
costmat = grid_search(gammavals, betavals)

# 绘制成本景观
plt.imshow(costmat, extent=(0, 1, 0, np.pi), origin="lower",
    aspect="auto")  # 在图像上显示成本矩阵，其中颜色深浅表示成本大小
plt.colorbar()  # 添加颜色条，以便了解颜色与成本的对应关系
plt.show()  # 显示图像
```

程序这一部分的输出显示在图 9.3 中。如我们所见，成本景观中存在大量的对称性。这种现象在变分量子算法中很常见。除了源自 Ising Hamiltonian 的自然对称性外，成本景观的对称性和周期性形式还源于 ansatz 电路中的对称性。利用这些对称性可以帮助经典优化算法更快地找到一个好的解决方案。

我们现在可以通过取成本景观中的最小值的坐标来得到一组最优参数。下面这个简短的代码块就是做这个，并打印出这些参数下的成本的数值。

[17]“景观”指的是在参数空间中，成本函数的变化图像。你可以将它想象成一个地形图，其中高度表示成本函数在给定参数下的值，而二维平面表示参数 γ 和 β 的可能值。因此，“绘制二维成本景观”就是创建一个图形，图形显示了在所有可能的 γ 和 β 值上，成本函数的变化情况。

```
# 从成本值的网格中获取坐标
gamma_coord, beta_coord = np.where(costmat == np.min(costmat))  # 寻找成本矩阵中的最小值，并返回其坐标

# 从坐标中获取值
gamma_opt = gammavals[gamma_coord[0]]  # 根据坐标得到对应的最优gamma值
beta_opt = betavals[beta_coord[0]]  # 根据坐标得到对应的最优beta值
```

现在我们有了最佳参数，我们可以用这些参数运行 QAOA 电路，并在计算基础上进行测量，得到解决我们最初优化问题的比特串。下面的函数运行该电路并返回测量结果。

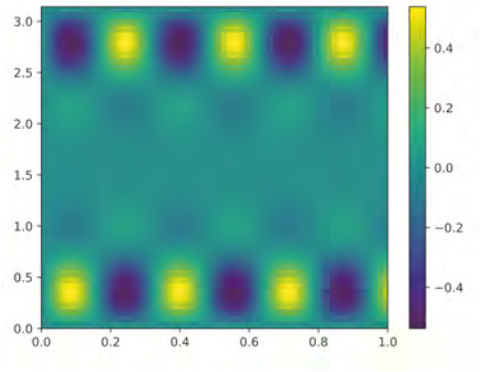


图 9.4: 通过一层 QAOA 计算出的 Ising Hamiltonian 的成本景观。

```
def get_bit_strings(gammas, betas, nreps=10000):
    """在计算基础上测量QAOA电路以获取比特串。"""
    # 创建 QAOA 电路
    circ = qaoa(gammas, betas)
    # 添加测量操作
    circ.append(cirq.measure(*[qubit for row in qreg for qubit in
        row], key='m'))
    # 模拟电路
    sim = cirq.Simulator()
    # 运行模拟，并重复多次以获取统计结果
    res = sim.run(circ, repetitions=nreps)
    # 返回结果
    return res
```

最后，我们使用上面找到的最优参数对电路进行采样。然后，我们解析输出并打印出从电路中采样得到的两个最常见的比特串。

```
# 进行采样以获取比特，并转换为直方图
bits = get_bit_strings([gamma_opt], [beta_opt])  # 使用最优参数进行采样
hist = bits.histogram(key="m")  # 将采样结果转换为直方图

# 获取最常见的比特
top = hist.most_common(2)  # 获取最常见的两个比特串

# 打印出测量到的两个最常见的比特串
print("\nMost common bitstring:")
print(format(top[0][0], "#010b"))
print("\nSecond most common bitstring:")
print(bin(top[1][0]))
```

以下是这部分代码的一个样本输出：

```
Most common bitstring:
0b000000000

Second most common bitstring:
0b111111111
```

These bitstrings are exactly the ones we would expect to minimize our cost function! Recall the Ising Hamiltonian we considered, which when written classically has the form

$$C(z) = -\sum_{\langle i,j \rangle} z_i z_j \tag{9.21}$$

这里 $z_i = \pm 1$ 表示自旋。对于我们的比特串输出，$b = 0$ 对应自旋向上 $(z = 1)$，$b = 1$ 对应自旋向下 $(z = -1)$。由于相反的自旋 $z_i \neq z_j$ 会在求和中产生正贡献（别忘了前面的整体负号!），所以成本函数的最小值发生在所有自旋对齐的时候。也就是，对所有的 i, j，有 $z_i = z_j$。我们测量到的比特串对应所有自旋向下对齐或所有自旋向上对齐。因此，这些比特串确实产生了我们成本函数的最小值，而 QAOA 能够成功地优化成本。

对于具有更复杂成本函数的更大的优化问题，可能需要更多的 QAOA ansatz 层（即 $p > 1$）。更多的层意味着变分量子电路中有更多的参数，这导致了更困难的优化问题。这样的优化问题不能仅通过对值的网格搜索来解决，因为这很快就变得难以处理。相反，必须使用基于梯度或无梯度的优化算法来计算大约最优的参数集。

这个 QAOA 实现的完整程序可以在本书的在线网站上找到。

9.3.2. 量子处理器上的机器学习

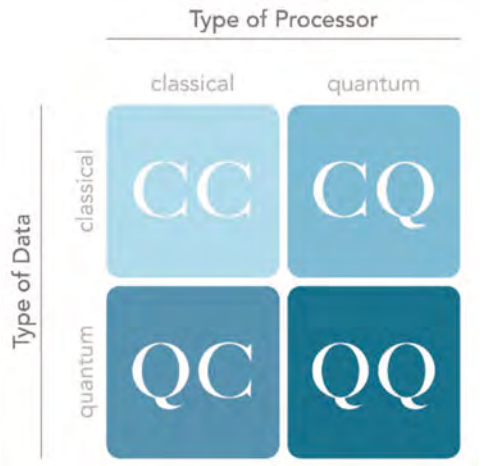


图 9.5: 机器学习数据类型和处理器类型。

量子计算机在机器学习领域的应用正在被多个研究团队探索；我们自然会想知道 QC 是否为我们在这个领域提供了任何优势。在量子机器学习 (QML) 中，加速并不是我们应该考虑的唯一优势。可能有机会使用 QC 直接处理来自保留了该传感器的全部量子信息的量子传感器的数据。图 9.4 指出了匹配量子数据与量子处理的潜力。在 QC 上直接分析数据流以查找模式的分类器可能比将数据传输到经典计算机更好。许多团队在这个领域发表了文章，包括：

- Alan Aspuru-Guzik 及其同事已经探索了量子机器学习以及混合经典-量子模型 [144, 145]。

- Rigetti 团队致力于在经典-量子混合方法上进行无监督机器学习 [146]。

  主要观点：该论文探讨了在混合量子计算机上进行无监督机器学习的方法。它介绍了一种基于量子比特和经典比特组成的混合量子计算机的实验平台，并通过实验验证了在该平台上实现无监督机器学习的可行性。

  实验模型：该论文采用了一个基于超导量子比特和经典比特的混合量子计算机实验平台。在该平台上，使用无监督机器学习算法进行实验，并通过比较实验结果和经典计算模型的性能来评估混合量子计算机在无监督机器学习任务中的潜力和优势。论文还描述了实验中使用的硬件设置和量子电路设计。

- Farhi 和 Neven 提出了在量子处理器上使用神经网络进行分类的方法 (QNNs)[147]。

  主要观点：该论文探讨了在近期量子处理器上使用量子神经网络进行分类任务的方法。它提出了一种基于量子比特的神经网络模型，并研究了该模型在解决分类问题上的潜力和限制。

  实验模型：该论文采用了一种基于量子比特的神经网络模型，并在近期量子处理器上进行实验。在该模型中，使用量子比特作为神经元，通过量子门操作和测量来构建和训练神经网络。论文还描述了在实验中使用的量子处理器和量子电路设计。通过比较实验结果和经典分类模型的性能，评估了近期量子处理器上量子神经网络在分类任务中的可行性和优势。

- Wittek 和 Gogolin 在量子平台上探索了 Markov 逻辑网络 [148]。

  主要观点：该论文研究了在马尔可夫逻辑网络中应用量子增强推理的方法。它探讨了如何利用量子计算的特性来提高马尔可夫逻辑网络中的推理能力，以实现更高效和精确的推断。

  实验模型：该论文基于马尔可夫逻辑网络和量子计算的理论，设计了一种量子增强推理模型，并在实验中应用该模型进行推理任务。论文还描述了实验中使用的马尔可夫逻辑网络模型和量子增强推理算法的具体实现细节。通过比较实验结果和传统推理方法的性能，评估了量子增强推理在马尔可夫逻辑网络中的优势和潜力。

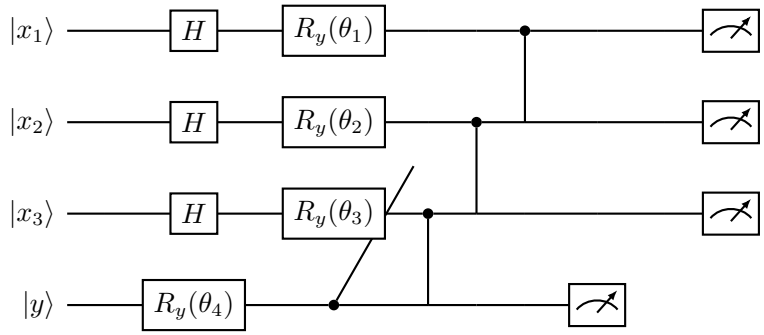
我们首先定义 QNN：

Listing 9.1: QNN

```
import cirq
import numpy as np

class ZXGate(cirq.ops.eigen_gate.EigenGate,
cirq.ops.gate_features.TwoQubitGate):
"""带有可变权重的ZX门。"""
def __init__(self, weight=1):
    """初始化ZX门的相位。
    Args:
        weight: 旋转角度，周期为2
    """
    self.weight = weight
    super().__init__(exponent=weight) # 处理权重不等于1的情况

def _eigen_components(self):
    return [
        (1, np.array([[0.5, 0.5, 0, 0],
                      [0.5, 0.5, 0, 0],
                      [0, 0, 0.5, -0.5],
                      [0, 0, -0.5, 0.5]])),
```

```
        (-1, np.array([[0.5, -0.5, 0, 0],
                       [-0.5, 0.5, 0, 0],
                       [0, 0, 0.5, 0.5],
                       [0, 0, 0.5, 0.5]]))
    ]

# 这让权重可以是一个符号，有助于参数化。
def _resolve_parameters_(self, param_resolver):
    return ZXGate(weight=param_resolver.value_of(self.weight))

# 该门在ASCII图中的样子
def _circuit_diagram_info_(self, args):
    return cirq.protocols.CircuitDiagramInfo(
        wire_symbols=('Z', 'X'),
        exponent=self.weight)
# 数据量子位的总数

INPUT_SIZE = 9
data_qubits = cirq.LineQubit.range(INPUT_SIZE)
readout = cirq.NamedQubit('r')

# 电路的初始参数

params = {'w': 0}

def ZX_layer():
"""在每个数据量子位和读出之间添加一个ZX门。
所有门都使用相同的cirq.Symbol权重。"""
for qubit in data_qubits:
yield ZXGate(cirq.Symbol('w')).on(qubit, readout)

qnn = cirq.Circuit()
qnn.append(ZX_layer())
qnn.append([cirq.S(readout)**-1, cirq.H(readout)]) # 基础转换
```

我们构建的量子神经网络（QNN）电路可以如下所示进行可视化：



接下来，我们定义函数，让我们能够访问 QNN 的 Z 值期望和损失函数：

```
def readout_expectation(state):
  """接受一个状态的描述，以0和1的数组形式表示，
  并返回在读出比特上的Z的期望值。
  使用模拟器来精确计算波函数。"""
  # 将状态方便地表示为一个整数
  state_num = int(np.sum(state * 2 ** np.arange(len(state))))
  resolver = cirq.ParamResolver(params)
  simulator = cirq.Simulator()
  # 明确指定比特顺序，以便知道哪个比特是读出比特
```

```
  result = simulator.simulate(qnn, resolver,
      qubit_order=[readout] + data_qubits,
      initial_state=state_num)
  wf = result.final_state
  # 由于我们指定了比特顺序，读出比特的Z值是最高有效位。
  Z_readout = np.append(np.ones(2 ** INPUT_SIZE),
      -np.ones(2 ** INPUT_SIZE))
  # 使用np.real消除+0j项
  return np.real(np.sum(wf * wf.conjugate() * Z_readout))


def loss(states, labels):
  loss = 0
  for state, label in zip(states, labels):
    loss += 1 - label * readout_expectation(state)
  return loss / (2 * len(states))


def classification_error(states, labels):
  error = 0
  for state, label in zip(states, labels):
    error += 1 - label * np.sign(readout_expectation(state))
  return error / (2 * len(states))
```

现在生成一些数据：

```
def make_batch():
  """生成一组标签，然后使用这些标签生成输入。
  label = -1 对应状态中的大多数0，label = +1 对应大多数1。"""
  np.random.seed(0)  # 为了演示的一致性
  labels = (-1) ** np.random.choice(2, size=100)  # 较小的批量大小将加速计算
  states = []
  for label in labels:
    states.append(np.random.choice(2, size=INPUT_SIZE,
                    p=[0.5 - label * 0.2, 0.5 + label * 0.2]))
  return states, labels

states, labels = make_batch()
```

最后，我们可以在参数空间上进行蛮力搜索，以找到最佳的 QNN：

```
linspace = np.linspace(start=-1, stop=1, num=80)
train_losses = []
error_rates = []
for p in linspace:
 params = {'w': p}
 train_losses.append(loss(states, labels))
 error_rates.append(classification_error(states, labels))
plt.plot(linspace, train_losses)
plt.xlabel('Weight')
plt.ylabel('Loss')
plt.title('Loss as a Function of Weight')
plt.show()
```

我们可以绘制损失函数作为权重的函数图，以了解网络的性能。这在图 9.5 中有所体现。最小损失约为 0.2，与线性模型可以获得的结果相当。如 [147][82] 所讨论的，更复杂的 QNN 可以实现更多功能。

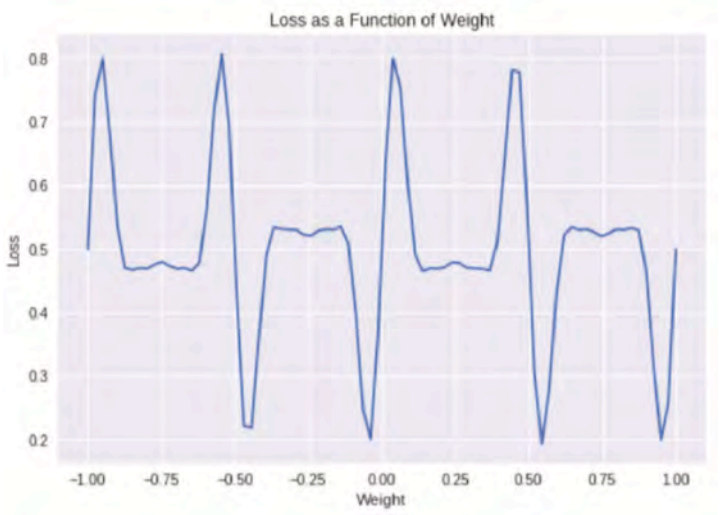


图 9.6: QNN 的损失函数与权重的关系图：权重的选择应使损失最小。

对于这种分类问题，QNN 是否优于经典模型尚未可知。更一般地说，在量子机器学习领域中，优势似乎在早期阶段并不明显。直到最近，一种用于推荐系统的量子算法在最佳已知经典算法上实现了指数级加速 [149] 。简而言之，推荐系统的一般思想如下：给定一个包含 m 个用户及其对 n 个产品的反馈的不完整的偏好矩阵 P，输出特定用户的好推荐。在这里，“不完整”意味着矩阵的某些条目丢失——也就是说，不是每个用户都对每个产品提供了反馈。

在 [149] 的工作之前，最好的经典算法的运行时间随矩阵维数 mn 增长而线性增加。量子推荐算法的运行时间随 mn 的多项式对数级别增长，具体而言为

$$O(poly(k)polylog(m,n))$$

，其中 poly 是 P 的条件数。虽然这是 QML 领域的一个主要成果，但是一个新的基于该量子算法的突破性经典算法也实现了矩阵维数的多项式对数级别缩放 [150] 。

视角不同，这或许是量子机器学习的优点或缺点。优点在于，经典算法直接受到了量子算法的启发——如果没有 QML，我们可能永远不会有这种想法。缺点在于，QML 领域的一个主要成果被“去量子化”了。在量子机器学习领域，例如 [151, 152, 153]，还在进行大量的研究，以探索这个相对年轻的领域的可能性和前景。

9.4. 量子相位估计

量子相位估计（Quantum Phase Estimation，QPE），也称为相位估计算法（Phase Estimation Algorithm，PEA），是一种用于确定一个酉算子的本征值的算法。本征值问题的形式如下：

$$Ax = \lambda x \tag{9.22}$$

其中 $A \in \mathbb{C}^{2m \times 2m}, x \in \mathbb{C}^{2m}$ 和 $\lambda \in \mathbb{C}$。本征值问题在数学和物理学中应用广泛。在数学中，应用范围从图论到偏微分方程。在物理学中，应用包括计算系统的哈密顿量的最小本征值——系统的基态能量——对于核、分子、材料和其他物理系统。此外，在机器学习中，用于降低特征向量维数的算法——主成分分析（PCA）——在其核心拥有一个本征值问题。本征值问题的应用范围涵盖了各个学科的广泛领域。

在量子情况下，我们关心的是找到一个酉算子 U 的本征值。由于酉算子的定义（$U^\dagger U = I$），一个酉算子的本征值具有模为 1 的特点，即 $|\lambda| = 1$。因此，酉算子 U 的任何本征值 λ 都可以写成下面的形式：

$$\lambda = e^{2\pi i\varphi} \tag{9.23}$$

其中 $0 \leq \varphi < 1$ 为相位。这就是算法名称“量子相位估计”的来源。通过估计 φ，我们可以得到 λ 的估计值。

在二进制小数表示中，假设相位 φ 可以精确地用 n 位表示，即

$$\varphi = 0.\varphi_1\varphi_2\ldots\varphi_n \tag{9.24}$$

这是相位 φ 的二进制小数表示。其中，对于每个 φ_k 其中 k 是二进制位 $\varphi \in \{0, 1\}$ 中的一位。我们也可以将其等价地写为：

$$\varphi = \sum_{k=1}^{n} \varphi_k 2^k \tag{9.25}$$

关键是理解控制幺正算子在本征态 $|\psi\rangle$ 上的作用。明确地，假设 U 是作为 QPE 算法的输入的幺正算子，使得

$$U|\psi\rangle = \lambda|\psi\rangle \tag{9.26}$$

现在暂时假设我们有本征态 $|\psi\rangle$。这并不是 QPE 的要求——实际上，如果我们知道 $|\psi\rangle$，我们可以直接在量子计算机上实现 $U|\psi\rangle$，但它简化了解释。现在，假设我们在第一个寄存器中准备了等叠加态（忽略归一化因子），并在第二个寄存器中准备了 U 的本征态 $|\psi\rangle$：

$$(|0\rangle + |1\rangle) \otimes |\psi\rangle = |0\rangle|\psi\rangle + |0\rangle|\psi\rangle \tag{9.27}$$

现在，如上所述，我们在该状态上实现了一个控制-U 操作，其产生以下状态：

$$|0\rangle|\psi\rangle + |1\rangle U|\psi\rangle = |0\rangle|\psi\rangle + e^{2\pi i 0.\psi_1\cdots\psi_n}|1\rangle|\psi\rangle \tag{9.28}$$

$$= (|0\rangle + e^{2\pi i 0.\psi_1\cdots\psi_n}|1\rangle) \otimes |\psi\rangle \tag{9.29}$$

请注意，第二个寄存器未更改。由于 $|\psi\rangle$ 是 U 的本征态，因此它不受控制操作的影响。那我们为什么要这样做呢？我们将关于相位的信息编码到第一个寄存器中。具体来说，第一个寄存器中的状态获得了相对相位 $e^{2\pi i 0.\psi_1\cdots\psi_n}$。

现在，根据相位估计，我们要实现 $k = 0, \ldots, n-1$ 的 2^k 次控制-U 操作。我们已经在 $k = 0$ 的情况下完成了上述操作。现在考虑 U^2 的作用：

$$\begin{aligned} U^2|\psi\rangle &= \lambda^2|\psi\rangle \\ &= e^{2\pi i(2\psi)}|\psi\rangle \\ &= e^{2\pi i 0.\varphi_2\ldots\varphi_n}|\psi\rangle \end{aligned}$$

for $k = 0, \ldots, n-1$. Hence, we can transform (9.30) under a controlled-U 2k as

$$|0\rangle|\psi\rangle + |1\rangle|\psi\rangle \mapsto |0\rangle|\psi\rangle + |1\rangle U^{2^k}|\psi\rangle$$
$$= (|0\rangle + e^{2\pi i 0.\varphi_{k+1}\ldots\varphi_n}|1\rangle) \otimes |\psi\rangle$$

这个表达式描述了一个量子系统在经过一个控制门操作后的演化过程。让我逐步解释。

首先，$|0\rangle$ 和 $|1\rangle$ 是量子比特的基态。$|\psi\rangle$ 是另一个量子系统的态。

表达式的第一行展示了一个初始态，它是将量子比特 $|0\rangle$ 和量子系统 $|\psi\rangle$ 的态进行张量积（tensor product）得到的。可以将其理解为量子比特和量子系统的初态为纠缠态。

然后，表达式中的 U^{2^k} 表示一个幺正操作，这是一个控制门操作。在这里，U 是一个单量子比特门，而 k 是一个非负整数。U^{2^k} 意味着 U 操作连续作用 2^k 次。这种形式的幺正操作通常在量子算法和量子纠错中使用。

接下来，我们考虑表达式的第二行。它给出了经过控制门操作后的演化结果。

在第二行中，我们可以将控制门操作作用在量子比特上，并保持量子系统不变。这意味着，如果量子比特的态为 $|0\rangle$，那么量子系统的态仍然是 $|\psi\rangle$。但是，如果量子比特的态为 $|1\rangle$，那么量子系统的态会乘以一个相位因子 $e^{2\pi i 0.\varphi_{k+1}\ldots\varphi_n}$。

这个相位因子涉及到一个小数 $0.\varphi_{k+1}\ldots\varphi_n$，其中 $\varphi_{k+1},\ldots,\varphi_n$ 是二进制小数的位。这个小数决定了相位因子的大小和复数的旋转角度。具体而言，$0.\varphi_{k+1}\ldots\varphi_n$ 是一个二进制小数的形式，表示一个角度在 $[0,1)$ 之间。

最后，表达式的最后一行展示了经过控制门操作后整个系统的态。$|0\rangle + e^{2\pi i 0.\varphi_{k+1}\ldots\varphi_n}|1\rangle$ 是量子比特的态，而 $|\psi\rangle$ 是量子系统的态。两者通过张量积连接在一起，形成了整个系统的态。

总结起来，给定一个量子比特和一个量子系统的初始纠缠态，经过控制门操作后，量子比特的态保持不变，而量子系统的态则会乘以一个相位因子。这个相位因子是由控制门操作的次数和一个二进制小数决定的。

s

10

应用与量子霸权

在这项工作中，我们已经踏上了量子计算的领域之旅；我们探索了它的理论基础，讨论了推动该领域发展的关键研究和里程碑，并涵盖了一系列硬件方法和量子计算方法。

在工程方面，我们仍然面临着艰巨的挑战，需要扩展至超过 106 个量子比特。一旦我们实现了容错的量子计算，将会有更多的应用可能性。在当前的 NISQ（噪声中间规模量子）体制下，还有大量的工作要做，包括探索测试案例和为纠错机器做准备。

Noisy Intermediate-Scale Quantum（NISQ）技术是指处于噪声临界状态的，数量可达到几百量子比特的量子计算系统。这种技术属于量子计算的一种过渡阶段，因为这些量子系统在处理的时候会受到噪声的影响，也不能进行错误修正。

NISQ 设备之所以重要，是因为它们是当前能够构建的最大的量子设备，也是我们通往大规模，纠错量子计算设备的桥梁。尽管它们的操作在一定程度上受到了噪声的影响，但它们还是能够执行一些有用的任务，特别是对于那些经典计算机难以解决的问题。

以下是 NISQ 技术的一些重要特性：

- **规模**：NISQ 设备的规模是中等的，即量子比特的数量大约在 50 到几百个之间。这是因为，量子比特数量增加会导致系统更容易受到环境噪声的影响，并且错误率[1]也会显著增加。
- **噪声**：NISQ 设备是噪声的，这意味着量子操作并不总是准确的。噪声可以通过环境干扰[2]，材

[1]错误率指的是在进行计算、传输或处理数据时发生错误的频率或比例。它是衡量系统或过程准确性的指标，表示出实际错误与期望正确结果之间的差异。错误率通常以百分比、小数或分数的形式表示，值越低表示系统的可靠性越高。在量子计算中，错误率是指在量子操作和量子比特的操作过程中发生错误的概率。高错误率会影响计算结果的准确性和可靠性，因此降低错误率是实现可靠量子计算的重要目标。

[2]环境干扰指的是在量子计算系统中由于外部环境的影响而导致的干扰现象。在量子计算中，量子比特需要处于特定的叠加态和纠缠态来进行计算和信息处理。然而，外部环境的噪声和干扰可以破坏量子比特的相干性和纠缠性，从而影响计算的准确性和可靠性。

环境干扰可以来自多个来源，如热噪声、振动、电磁波辐射和杂散信号等。这些干扰可能导致量子比特的相位、振幅或能级发生不可逆的变化，使得计算过程中产生误差。此外，环境干扰还可能导致量子比特之间的耦合效应、退相干速率的增加以及量子门操作的错误执行。

为了抵抗环境干扰，量子计算研究中采用了多种技术，如量子纠错码、量子噪声抑制和量子控制技术。这些技术旨在减少环境干扰对量子系统的影响，提高量子计算的可靠性和稳定性。

料缺陷[3]，设备不稳定性[4]等方式引入。这种噪声的存在限制了 NISQ 设备能够有效地执行的计算长度[5]。

- **无错误纠正**：由于 NISQ 设备的噪声水平和尺度，它们无法有效地实现量子错误纠正。量子错误纠正需要大量的冗余量子比特[6]，以及高精度的操作，这在 NISQ 设备上是无法实现的。
- **应用领域**：尽管存在这些挑战，NISQ 设备仍然可以在一些特定的应用领域发挥作用。例如，它们可以用于执行量子模拟，优化，以及一些机器学习算法。这些应用领域通常涉及到那些对于经典计算机来说难以处理的问题。

NISQ 设备是我们通向大规模，纠错量子计算的过渡阶段。尽管它们受到噪声的影响并且无法进行错误纠正，但它们仍然有可能在某些应用领域中找到有价值的用途。

10.1. 应用

随着量子计算领域的发展，一些量子计算的应用正在变得清晰起来。请查看在线网站，并参考 [154] 和 [20] 了解有关量子计算应用的最新信息。

在文章 [154] 中，作者们指出了商业化量子技术所面临的挑战，并提出了一些解决方案和发展方向。他们认为，虽然量子技术在理论上已经取得了重大进展，但要实现商业化应用仍然存在许多技术和工程上的难题。

首先，作者们强调了量子技术的可扩展性和稳定性问题。当前的量子系统对于噪声和误差非常敏感，而且随着系统规模的增大，这些问题变得更加严重。为了商业化量子技术，需要研发更稳

[3]材料缺陷是指在固体材料中存在的结构上的不完整或不规则性。它们可以是材料内部的点缺陷、线缺陷或面缺陷，也可以是材料表面的缺陷。材料缺陷可以是原子层面上的缺陷，如空位、间隙原子、杂质原子等，也可以是宏观尺度上的缺陷，如晶体缺陷、晶界、位错等。

材料缺陷对材料的物理、化学和力学性质产生显著影响。它们可以改变材料的电子结构、导电性、热导性、机械性能和化学反应性等。一些缺陷可能会引起材料的非理想行为，如导致材料的疲劳、断裂、腐蚀或电子器件的失效。另一方面，一些材料缺陷也可以用于调控材料的特性，例如在半导体材料中引入杂质原子来调节导电性。

研究材料缺陷有助于理解材料的结构与性能之间的关系，并为材料设计、工程和制备提供指导。控制和调控材料缺陷可以改善材料的性能和功能，进而推动材料科学和工程的发展。

[4]设备不稳定性指的是设备或系统在运行过程中出现不一致、不可靠或不稳定的行为。它表示设备的性能或功能可能会随着时间、环境条件或其他因素的变化而发生波动或变化。

设备不稳定性可能导致设备的输出结果或性能出现偏差、波动或不可预测的变化。这可能包括测量或传感器设备的输出值的变化、控制系统的失效或错误、信号传输的干扰或中断等。设备不稳定性可能由多种因素引起，包括温度变化、电压波动、机械振动、材料老化、设计缺陷或制造误差等。

设备不稳定性对于许多领域和应用都是一个关键问题，特别是在科学研究、工业生产、通信系统和精密测量等领域。不稳定的设备可能导致数据的不准确性、产品质量的下降、系统性能的降低或操作过程的不可靠性。为了解决设备不稳定性的问题，通常需要采取措施，如校准和调整设备、改进设计和制造过程、使用稳定性更高的元件或采用自适应控制策略等。

[5]计算长度（computational length）是指在计算理论和计算复杂性理论中用于衡量算法或计算问题复杂性的度量。它通常用于评估计算任务所需的计算资源，例如时间和空间。

计算长度可以用来表示一个算法执行所需的计算步骤的数量或所需的计算资源的数量。在时间复杂性分析中，计算长度通常以算法的时间复杂度的形式表示，表示算法执行所需的计算步骤或操作的数量。在空间复杂性分析中，计算长度可以表示算法所需的内存或存储空间的数量。

计算长度的概念对于理解和比较不同算法的效率和复杂性非常重要。通过分析计算长度，可以评估算法的运行时间、内存使用量以及计算问题的可解性。计算长度还可以用于确定最佳算法选择、优化算法实现以及评估计算系统的性能和资源需求。

[6]冗余量子比特（redundant quantum bit）是指在量子计算系统中添加额外的量子比特来增强系统的纠错能力和容错性。冗余量子比特的引入是为了应对环境干扰、噪声和其他错误对量子信息的影响。

在传统的量子比特表示中，每个量子比特都承载一个特定的量子态。然而，由于环境噪声和量子操作的误差，量子比特可能会发生错误，导致信息丢失或退相干。为了弥补这些错误，冗余量子比特的思想被引入，通过在系统中引入额外的量子比特来增加冗余度。

通过冗余量子比特的使用，量子计算系统可以使用纠错编码和纠错算法来检测和纠正量子比特上的错误。冗余量子比特允许系统通过对多个量子比特的测量和操作来检测和纠正单个量子比特的错误。当错误发生时，系统可以利用冗余的信息进行错误检测和修复，从而保护量子信息的完整性。

冗余量子比特在量子纠错码和量子容错技术中发挥着重要作用。通过引入冗余量子比特并设计合适的纠错编码方案，可以提高量子计算系统的可靠性和稳定性，使其能够在存在噪声和干扰的环境中进行可靠的量子计算。

定和可靠的量子系统，以及有效的纠错和误差校正方法。

- 提到了量子技术的可扩展性问题。他们指出当前的量子系统在面对噪声和误差时非常敏感，并且随着系统规模的增大，这些问题变得更加严重。这就意味着要实现商业化的量子技术，需要克服可扩展性方面的挑战。

  作者们认为，为了解决这个问题，需要研发更稳定和可靠的量子系统。这可能涉及到设计和制造更高质量的量子比特，改进量子纠错和误差校正的方法，以及寻找更好的材料和构建技术，以提高系统的性能和可扩展性。

  量子技术的可扩展性是实现商业化应用的关键因素之一。只有当量子系统能够在大规模和实际环境中稳定运行时，才能满足商业化应用的需求。因此，作者们强调了在解决可扩展性问题方面的研究和发展的重要性，以推动量子技术的商业化进程。
- 在文献中，作者们提到了量子技术的稳定性问题。他们指出当前的量子系统对于噪声和误差非常敏感，这限制了量子技术的可靠性和稳定性。在商业化量子技术的过程中，解决稳定性问题是至关重要的。

  作者们认为，为了解决稳定性问题，需要进行更深入的研究和发展。这可能包括改进量子硬件的设计和制造，以减少噪声和误差的影响。另外，发展有效的纠错和误差校正方法也是提高稳定性的关键。这些方法可以帮助纠正由于噪声和误差引起的错误，并提高量子系统的可靠性。

  稳定性问题的解决还需要在材料选择、构建技术和系统控制方面取得进展。寻找更好的材料和构建技术可以提高量子系统的性能和稳定性。同时，改进系统的控制方法可以减少对系统的干扰，提高其稳定性和可靠性。

  综上所述，作者们强调了解决量子技术稳定性问题的重要性。只有通过克服稳定性方面的挑战，才能实现商业化量子技术，并使其能够在实际应用中稳定运行。因此，稳定性问题的研究和发展是推动量子技术商业化的关键因素之一。

其次，作者们提到了量子算法和应用的开发问题。尽管已经存在一些理论上的量子算法，但将它们应用于实际问题仍然面临许多挑战。商业化量子技术需要开发更多实用的量子算法，以及与经典计算机系统的集成方法。

此外，作者们还讨论了量子通信和量子安全性的重要性。量子通信可以实现高度安全的通信方式，而量子安全性可以保护敏感数据和信息。商业化量子技术需要解决量子通信的可靠性和扩展性问题，并加强量子安全性的研究和应用。

在文章的结尾，作者们呼吁加强学术界、工业界和政府之间的合作，共同努力推动量子技术的商业化进程。他们相信，在未来五年内，通过解决技术挑战、加强研发和促进合作，商业化量子技术将取得重大突破，并对各个领域产生深远的影响。

10.1.1. 量子模拟和化学

目前，高性能的经典计算机被用于对新的分子组合进行建模。这项工作帮助研究人员开发新材料、新型药物以及其他应用的化合物。量子计算机很可能在这个领域为我们提供新的能力。已经有一些方法，如在第 9 章讨论的 VQE 和量子化学模拟方法，显示出了有希望的结果。请参考以下的附加示例：[155, 134, 156]

10.1.2. 从概率分布中抽样调查

我们在许多应用中使用分布抽样，例如模式识别和概率推断。借助量子计算机，我们可以从更大的分布中进行抽样。这就是分布抽样被用来展示量子霸权的原因之一，我们稍后在本章中会详细描述。

10.1.3. 用量子计算机提高线性代数的速度

线性代数在工业上有很多应用。矩阵反转是一种常见的技术，例如，可以用于计算电磁模式来设计天线 [20] 。我们在第 9 章中介绍的 HHL 技术是一种可能被证明对这些应用有价值的方法。

10.1.4. 优化

工业领域存在着众多的优化应用，包括：货车路径规划、在线广告竞价策略以及电动汽车电池组成中不同化学物质的混合。越来越明显的是，量子计算机可以用于优化这些类型的系统。

10.1.5. 张量网络

一个有前途的研究领域是将量子计算应用于张量网络（TNs）。本书的在线网站包含了一些介绍 TNs 的好参考资料。

诸如 MERA、MPS、TTNs 和 PEPS 等不同的张量网络架构正显示出在物理学以及深度学习网络等其他领域探索问题的有用工具的潜力。几个研究小组已经展示了张量网络的多种应用 [157, 158, 159, 160, 161, 162] 。

10.2. 量子霸权

I

“量子霸权”这个术语最早由普雷斯基尔（Preskill）[7]于 2012 年首次提出，指的是一项在量子计算机上能够高效执行，超越现有最先进经典超级计算机能力的计算任务 [163] 。需要立即注意的是——由于“霸权”这个词可能会引起混淆——这指的是满足特定标准的任何计算任务，而不一定是有用的任务。

用于展示霸权的算法不需要具有广泛的应用，只需要相对于经典计算机在无法处理该算法的情况下，在量子处理器上能够高效运行的明确能力 [164]。

在本节中，我们讨论研究人员考虑用来展示量子霸权的问题。无论用于演示的特定问题如何，量子霸权都是物理学和计算机科学历史上的里程碑成就。虽然在量子处理器上进行了许多原理性的量子计算，但这将是第一个在足够大的规模上执行的实验，以揭示一个可通过实验证实的计算分离。

这对通过大规模计算验证量子力学具有重要意义。事实上，我们可以将霸权实验视为贝尔实验的计算类比 [165]。正如贝尔实验否定了局部隐藏变量模型一样，对于纠错量子计算机的霸权实验将否定扩展的丘奇-图灵论题（ECTT），该论题在第 4 章中讨论过，它断言任何算法过程都可以通过使用概率图灵机 PTM 进行高效模拟。精确控制如此规模的量子系统也是工程和实验物理学的巅峰成就之一。

[7]普雷斯基尔（Preskill）指的是 John Preskill，他是一位著名的理论物理学家和量子信息科学家。他是加州理工学院（Caltech）的教授，专注于量子信息和量子计算领域的研究。普雷斯基尔在 2012 年提出了“量子霸权”这个术语，将其定义为一种在量子计算机上能够高效执行、超越经典计算机能力的计算任务。他的研究对于量子计算和量子信息的发展做出了重要贡献，并在学术界享有很高的声誉。

在本节的剩余部分，我们将讨论正在考虑用于展示量子霸权的主要问题。

10.2.1. 随机电路取样

从量子电路的输出分布中进行抽样是展示量子霸权最自然的问题之一。要在经典计算机上模拟这个过程，必须进行线性代数和矩阵计算，以确定在执行量子电路后的波函数的最终状态（由单位算符的张量积表示）。然而，量子计算机通过在单位算符的物理实现下简单地按时间演化来自然地执行这个计算。

经典模拟量子电路的方法通常在量子比特数上呈指数级增长。具体而言，对于 n 个量子比特的最一般、完全纠缠的状态，波函数中有 2^n 个复幅需要跟踪。即使对于中等规模的 n，这个数量也迅速超过了当前最强大的超级计算机的内存限制。

前缀	字节数	最大量子比特数
千 (KB)	10^3	$2^{8,000}$
兆 (MB)	10^6	$2^{8,000,000}$
吉 (GB)	10^9	$2^{8,000,000,000}$
太 (TB)	10^{12}	$2^{8,000,000,000,000}$
拍 (PB)	10^{15}	$2^{8,000,000,000,000,000}$
艾 (EB)	10^{18}	$2^{8,000,000,000,000,000,000}$
泽 (ZB)	10^{21}	$2^{8,000,000,000,000,000,000,000}$
尧 (YB)	10^{24}	$2^{8,000,000,000,000,000,000,000,000}$

表 10.1: 前缀和对应字节数的表格；最后一列显示了在给定内存下可以存储的最大量子比特数，假设量子比特的最一般状态以双精度存储幅度。一字节等于 8 位。

每个波函数中的振幅通常是一个复数，这意味着每个振幅需要存储两个实数浮点数。假设这些浮点数以双精度格式存储，即每个浮点数占用 8 个字节。在这些假设下，存储波函数所需的总内存为：$2n$ 个振幅 $\times 2$ 个实数/振幅 $\times 2^3$ 字节/实数即，2^{n+4} 字节。回想一下，一个千字节被定义为 2^{10} 字节，一个兆字节是 2^{20} 字节，依此类推；请参考表格10.1。

当前最先进的超级计算机的 RAM 大小通常为拍字节到艾字节的范围。基于我们之前关于存储波函数内存需求的论证，我们可以估计任何特定经典系统对量子电路模拟的上限范围。

这就是将量子电路抽样作为展示量子霸权的候选问题的基本思想 [166, 167] 。现在有多种模拟量子电路的方法——从电路的单位算符的显式构造到张量网络的收缩——但它们都面临着指数级的复杂度问题。现在让我们更详细地考虑随机电路抽样作为展示量子霸权的演示问题，按照 Boixo 等人的工作 [168, 164, 169] 。在这里，研究人员正在考虑从随机量子电路的输出分布中进行抽样的问题。

量子霸权实验中考虑的特定随机电路是根据以下规则构建的 [164]：

1. 每个量子比特开始时应用哈达玛门。
2. 在二维网格中，按照水平和垂直模式交替，在相邻的量子比特之间应用控制-Z (CZ) 门。请注意，在任何特定的周期中，并非所有相邻的量子比特都会通过 CZ 门连接，并且不同周期中 CZ 门的数量可能会有所不同。
3. 根据以下标准，对未受 CZ 门影响的量子比特应用来自集合 $\{X^{1/2}, Y^{1/2}, T\}$ 的单比特门：

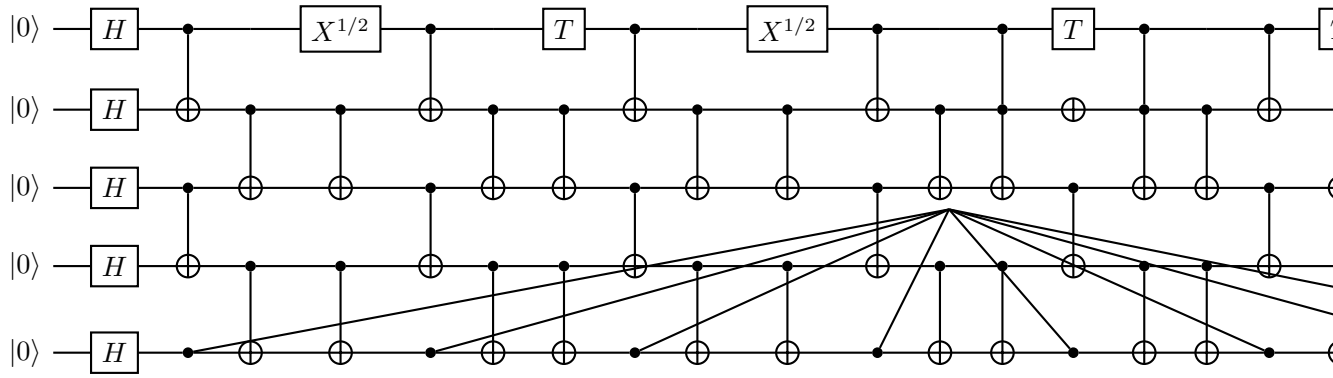
- 如果前一个周期在给定的量子比特上有一个 CZ 门，则如果可能，对该量子比特应用随机选择的非对角线一元门。
- 如果前一个周期在给定的量子比特上有一个非对角线一元门，则如果可能，对该量子比特应用 T 门。
- 如果在前几个周期中给定的量子比特上没有应用一元门（除了初始的哈达玛门），则对该量子比特应用 T 门。请注意，此规则是一个“if”而不是“iff”。也就是说，T 门可以跟随另一个一元门；该规则只是说明如果在前几个周期中在该量子比特上没有应用一元门，则我们必须在当前周期中放置一个 T 门。前面的两个标准优先于此标准。
- 如果对于给定的量子比特没有满足上述任何标准，则在当前周期不对该量子比特应用一元门。

4. 对给定数量的周期（决定深度），重复步骤（2）和（3）。
5. 在计算基或哈达玛（X）基上进行测量。

现在我们可以将这些规则整合到一个用于构建霸权电路的程序中。下面提供了一个使用 Cirq 演示此功能的示例程序。

```
import cirq

def build_supremacy_circuit(qubits, depth):
    circuit = cirq.Circuit()

    # 第1步：在每个量子位上应用哈达玛德门
    for qubit in qubits:
        circuit.append(cirq.H(qubit))

    # 第二和第三步：应用CZ和单量子位门
    for cycle in range(depth):
        for i in range(len(qubits)):
            # 在相邻的量子比特之间应用CZ门
            if i < len(qubits) - 1:
                circuit.append(cirq.CZ(qubits[i], qubits[i+1]))

            # 根据给定的标准应用单比特门
            if cycle % 2 == 0:
                if i % 2 == 0:
                    circuit.append(cirq.X**(1/2)(qubits[i]))  # 对满足条件的量子比特应用 X^(1/2) 门
                else:
                    circuit.append(cirq.Y**(1/2)(qubits[i]))  # 对满足条件的量子比特应用 Y^(1/2) 门
            else:
                if i % 2 == 1:
                    circuit.append(cirq.T(qubits[i]))  # 对满足条件的量子比特应用 T 门

    # 第5步：测量
    circuit.append(cirq.measure(*qubits, key='result'))  # 测量量子比特并将结果存储在名为 'result' 的键下

    return circuit

# 使用实例
qubits = cirq.LineQubit.range(5)  # 示例的5量子比特电路
depth = 10  # 电路的深度

```

```
circuit = build_supremacy_circuit(qubits, depth)
print(circuit)
```



在这里，我们为二维量子比特网格指定了行数和列数；通过指定深度或 CZ 门的周期数，确定了量子霸权电路的总深度。该程序的输出如图 10.1 所示。请注意，在任何特定版本或实现的随机电路采样器中，代码模块可能会遵循与上述规则略有不同的规则。

这个包含 n 个 16 量子比特的电路对于经典计算机来说很容易处理，但正如我们所讨论的，经典模拟的难度在 n 上呈指数级增长。对于足够大的量子比特数 n，我们概述了展示量子霸权的步骤 [164]:

- 根据前面所述的规则，生成一个包含 n 个量子比特和给定深度 d 的霸权电路 U。
- 使用 $m \approx 10^3 \square - 10^6$ 次采样电路，得到一个输出分布 $x_1, \ldots, x_m$。
- 使用足够强大的经典计算机计算每个 $j = 1, \ldots, m$ 的 $log(1/pU(x_j))$。其中，$pU(x_j) := |\langle X_j|\psi\rangle|^2$，$|\psi = U\,|0\rangle\rangle$ 是霸权电路的最终状态。
- 计算量:

$$\alpha = H_0 - \frac{1}{m}\sum_{1}^{m} \log \frac{1}{pU(x_j)} \tag{10.1}$$

其中，$H_0 = \log(2^n) + \gamma$ 是一个从比特串中均匀采样的算法的交叉熵。(注意，这里的对数是自然对数。) 这里，$\gamma \approx 0.577$ 是欧拉常数。

对以下内容进行翻译:

一旦计算得到量 α，就可以将其与经典算法 A 模拟量子电路的输出分布 p_A 上的类似量进行比较。注意，交叉熵差异用于衡量算法 A 对典型随机电路 U 结果的预测能力，定义如下:

$$\Delta H(p_A) = H_0 - H(p_A, p_U) \tag{10.2}$$

现在考虑在随机电路集合 R 上，对 $\Delta H(p_A)$ 的期望值，并将其记为 C:

$$C := \mathbb{E}_R[\Delta H(p_A)] \tag{10.3}$$

在 Boixo 等人的霸权实验论文 [164] 中，实际上证明了当满足以下条件时，可以实现量子优势:

$$C \leq \alpha \leq 1 \tag{10.4}$$

请注意，对于足够大的电路，$C \to 0$，并且无法通过数值方法获得 $pU(x_j)$。这意味着根据定义，无法直接测量量 α。然而，可以通过对更大电路进行外推来展示使用随机电路采样的量子霸权。

10.2.2. 证明量子优越性的其他问题

随机电路采样是展示量子霸权的一个非常自然的问题，但并不是唯一的问题。Harrow 和 Montanaro 的最近一篇综述论文 [165] 对其他一些重要的问题进行了讨论，以展示量子与经典的区别。

玻色子采样问题是展示量子霸权的另一个候选问题。最初在 [170] 中提出的玻色子采样问题涉及将 n 个符合条件的光子发送到一个随机生成的 $m \gg n$ 模式（光束分束器）的线性光学网络中；这将生成一个随机的酉旋转矩阵。然后使用探测器从光子的分布中进行采样，这个过程被认为在经典计算上是困难的。已经进行了包含五个光子和九个模式的玻色子采样实验 [171] 。实验系统面临着光子在光学网络中的非平凡损失的挑战。此外，开发更高效的经典采样技术也是通过玻色子采样实现量子霸权的一个挑战。

10.2.3. 量子优势

研究人员创造了几个与经典计算和量子计算之间的区别相关的术语，包括量子优势和量子与经典的区分。量子优势可以指相对于经典计算而言的恒定或线性加速。请参阅 W. Zeng 关于量子与经典计算区别的术语和度量的文章。

10.3. 未来方向

10.3.1. 量子纠错

虽然当今的量子计算机尚未具备足够的量子比特来支持完整的量子纠错（QEC），但关于 QEC 的研究正在增加，对量子计算及其应用产生了重要影响。经典计算可以通过在许多经典比特之间复制状态来进行简单的纠错。然而，量子力学中的无克隆定理阻止我们在量子计算机中采用这种直接的方法。

一种典型的 QEC 方法涉及使用表面码（surface code），将一个逻辑量子比特编码为多个物理量子比特的拓扑态 [172, 173, 174] 。当我们测量这些物理量子比特时，可以看到一种被称为综合码的模式，它是特定错误序列的结果；然后，解码器可以将综合码映射到特定的错误序列。这种解码过程可能适用于机器学习的应用（例如，参见 [175]）。

正如在有关 VQE 的部分中讨论的那样，McClean 等人已经探索了使用子空间扩展进行错误缓解的方法 [138] 。

Ofek 等人的工作讨论了 QEC 的突破点 [176] 。纠错方案还源自其他物理学领域；一些研究人员一直在研究从 Anti-de Sitter/Conformal Field Theory (AdS/CFT) 的对偶框架中衍生出的 QEC 方法 [177] 。QEC 仍然是一个活跃的研究领域，在量子计算硬件设备的扩展中起着至关重要的作用。

10.3.2. 用量子计算机做物理学研究

正如我们在本书的前言中提到的，量子计算机最有趣的潜在用途之一是探索物理学中的未解问题。AdS/CFT 的对偶框架为广义相对论和量子力学之间提供了一个初始映射。Susskind 和其他人猜测使用量子计算机来探索这种对偶性 [178] 。虽然我们离建造足够规模和容错性的量子计算机来运行这样的实验还有很多年的时间，但考虑到我们可能从这样的探索中学到什么仍然是有益的。

这里的关键原则是，在量子计算中，我们不仅仅是在经典计算机上建模超 position 或纠缠态并指向它；实际上，我们正在实现这些态，并且因此可以对它们的动力学提出问题。

10.3.3. 总结

我们预计这个领域的发展速度将非常快，无论是硬件还是软件，我们预测将会有更多的大学和公司探索这些平台如何影响他们的工作。

11

Conclusion

A conclusion...

参考文献

[1] J. D. Hidary and J. D. Hidary, *Quantum computing: an applied approach*. Springer, 2019, vol. 1.

[2] A. Church, “Am turing. on computable numbers, with an application to the entscheidungs problcm. proceedings of the london mathematical society, 2 s. vol. 42 (1936–1937), pp. 230–265.” *The Journal of Symbolic Logic*, vol. 2, no. 1, pp. 42–43, 1937.

[3] Y. I. Manin, “Vychislimoe i nevychislimoe (computable and noncomputable), moscow: Sov,” 1980.

[4] ——, “Classical computing, quantum computing, and shor’s factoring algorithm,” *arXiv preprint quant-ph/9903008*, 1999.

[5] R. P. Feynman, “Simulating physics with computers,” in *Feynman and computation*. CRC Press, 2018, pp. 133–153.

[6] ——, “Quantum mechanical computers,” *Optics news*, vol. 11, no. 2, pp. 11–20, 1985.

[7] P. Benioff, “Quantum mechanical hamiltonian models of turing machines,” *Journal of Statistical Physics*, vol. 29, pp. 515–546, 1982.

[8] D. Deutsch, “Quantum theory, the church–turing principle and the universal quantum computer,” *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[9] J. Preskill, “Lecture notes for physics 229: Quantum information and computation,” *California Institute of Technology*, vol. 16, no. 1, pp. 1–8, 1998.

[10] ——, “Quantum computing 40 years later,” *arXiv preprint arXiv:2106.10522*, 2021.

[11] D. Deutsch and R. Jozsa, “Rapid solution of problems by quantum computation,” *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[12] D. R. Simon, “On the power of quantum computation,” *SIAM journal on computing*, vol. 26, no. 5, pp. 1474–1483, 1997.

[13] E. Bernstein and U. Vazirani, “Quantum complexity theory,” in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 11–20.

[14] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[15] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[16] R. Rivest, “Cryptography. in van leeuwen, j,” *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, pp. 719–756, 1994.

[17] C. H. Bennett and G. Brassard, “Quantum cryptography: Public key distribution and coin tossing,” *arXiv preprint arXiv:2003.06557*, 2020.

[18] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, R. Cleve, and I. L. Chuang, “Experimental realization of an order-finding algorithm with an nmr quantum computer,” *Physical Review Letters*, vol. 85, no. 25, p. 5452, 2000.

[19] A. Einstein, B. Podolsky, and N. Rosen, “Can quantum-mechanical description of physical reality be considered complete?” *Physical review*, vol. 47, no. 10, p. 777, 1935.

[20] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, p. 79, 2018.

[21] E. Schrödinger, “Discussion of probability relations between separated systems,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 31, no. 4. Cambridge University Press, 1935, pp. 555–563.

[22] R. Landauer, “Irreversibility and heat generation in the computing process,” *IBM journal of research and development*, vol. 44, no. 1/2, p. 261, 2000.

[23] M. A. Nielsen and I. Chuang, “Quantum computation and quantum information,” 2002.

[24] P. Benioff, “The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines,” *Journal of statistical physics*, vol. 22, pp. 563–591, 1980.

[25] Y. Manin, “Computable and uncomputable,” *Sovetskoye Radio, Moscow*, vol. 128, 1980.

[26] S. Lloyd, “A potentially realizable quantum computer,” *Science*, vol. 261, no. 5128, pp. 1569–1571, 1993.

[27] M. Hayashi and H. Zhu, “Secure uniform random-number extraction via incoherent strategies,” *Physical Review A*, vol. 97, no. 1, p. 012302, 2018.

[28] S. Aaronson and S.-H. Hung, “Certified randomness from quantum supremacy,” *arXiv preprint arXiv:2303.01625*, 2023.

[29] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[30] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, “Experimental realization of shor’s quantum factoring algorithm using nuclear magnetic resonance,” *Nature*, vol. 414, no. 6866, pp. 883–887, 2001.

[31] L. K. Grover, “Quantum mechanics helps in searching for a needle in a haystack,” *Physical review letters*, vol. 79, no. 2, p. 325, 1997.

[32] E. Farhi and S. Gutmann, “Analog analogue of a digital quantum computation,” *Physical Review A*, vol. 57, no. 4, p. 2403, 1998.

[33] Y. Nakamura, Y. A. Pashkin, and J. Tsai, “Coherent control of macroscopic quantum states in a single-cooper-pair box,” *nature*, vol. 398, no. 6730, pp. 786–788, 1999.

[34] Y. Nakamura, Y. A. Pashkin, and J. S. Tsai, “Rabi oscillations in a josephson-junction charge two-level system,” *Physical Review Letters*, vol. 87, no. 24, p. 246601, 2001.

[35] J. Cirac and P. Zoller, “Quantum computations with cold trapped ions,” *Phys. Rev. Lett*, vol. 74, p. 20.

[36] M. Veldhorst, H. Eenink, C.-H. Yang, and A. S. Dzurak, “Silicon cmos architecture for a spin-based quantum computer,” *Nature communications*, vol. 8, no. 1, p. 1766, 2017.

[37] J. W. Silverstone, J. Wang, D. Bonneau, P. Sibson, R. Santagati, C. Erven, J. O’Brien, and M. Thompson, “Silicon quantum photonics,” in *2016 International Conference on Optical MEMS and Nanophotonics (OMN)*. IEEE, 2016, pp. 1–2.

[38] M. Verbin, O. Zilberberg, Y. E. Kraus, Y. Lahini, and Y. Silberberg, “Observation of topological phase transitions in photonic quasicrystals,” *Physical review letters*, vol. 110, no. 7, p. 076403, 2013.

[39] D. P. DiVincenzo, “Topics in quantum computers,” *Mesoscopic electron transport*, pp. 657–677, 1997.

[40] L. B. Nguyen, G. Koolstra, Y. Kim, A. Morvan, T. Chistolini, S. Singh, K. N. Nesterov, C. Jünger, L. Chen, Z. Pedramrazi *et al.*, “Scalable high-performance fluxonium quantum processor,” *arXiv preprint arXiv:2201.09374*, 2022.

[41] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, “Elementary gates for quantum computation,” *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[42] T. Toffoli, “Reversible computing,” in *Automata, Languages and Programming: Seventh Colloquium Noordwijkerhout, the Netherlands July 14–18, 1980 7*. Springer, 1980, pp. 632–644.

[43] E. Fredkin and T. Toffoli, “Conservative logic,” *International Journal of theoretical physics*, vol. 21, no. 3-4, pp. 219–253, 1982.

[44] H. M. Sheffer, “A set of five independent postulates for boolean algebras, with application to logical constants,” *Transactions of the American mathematical society*, vol. 14, no. 4, pp. 481–488, 1913.

[45] Y. Shi, “Both toffoli and controlled-not need little help to do universal quantum computation,” *arXiv preprint quant-ph/0205115*, 2002.

[46] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, “On universal and fault-tolerant quantum computing,” *arXiv preprint quant-ph/9906054*, 1999.

[47] M. Ozols, “Clifford group,” *Essays at University of Waterloo, Spring*, 2008.

[48] C. M. Dawson and M. A. Nielsen, “The solovay-kitaev algorithm,” *arXiv preprint quant-ph/0505030*, 2005.

[49] G. H. Hardy and J. E. Littlewood, “Some problems of diophantine approximation,” *Acta math*, vol. 37, no. 1, pp. 193–239, 1914.

[50] D. E. Knuth, “Big omicron and big omega and big theta,” *ACM Sigact News*, vol. 8, no. 2, pp. 18–24, 1976.

[51] R. Solovay and V. Strassen, “A fast monte-carlo test for primality,” *SIAM journal on Computing*, vol. 6, no. 1, pp. 84–85, 1977.

[52] M. Agrawal, N. Kayal, and N. Saxena, “Primes is in p,” *Annals of mathematics*, pp. 781–793, 2004.

[53] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O’Brien, “Quantum computers,” *nature*, vol. 464, no. 7285, pp. 45–53, 2010.

[54] Y. Wang, X. Zhang, T. A. Corcovilos, A. Kumar, and D. S. Weiss, “Coherent addressing of individual neutral atoms in a 3d optical lattice,” *Physical review letters*, vol. 115, no. 4, p. 043003, 2015.

[55] H. Pichler, S.-T. Wang, L. Zhou, S. Choi, and M. D. Lukin, “Quantum optimization for maximum independent set using rydberg atom arrays,” *arXiv preprint arXiv:1808.10816*, 2018.

[56] H. Levine, A. Keesling, A. Omran, H. Bernien, S. Schwartz, A. S. Zibrov, M. Endres, M. Greiner, V. Vuletić, and M. D. Lukin, “High-fidelity control and entanglement of rydberg-atom qubits,” *Physical review letters*, vol. 121, no. 12, p. 123603, 2018.

[57] T.-Y. Wu, A. Kumar, F. Giraldo, and D. S. Weiss, “Stern–gerlach detection of neutral-atom qubits in a state-dependent optical lattice,” *Nature Physics*, vol. 15, no. 6, pp. 538–542, 2019.

[58] M. Saffman, T. G. Walker, and K. Mølmer, “Quantum information with rydberg atoms,” *Reviews of modern physics*, vol. 82, no. 3, p. 2313, 2010.

[59] D. S. Weiss and M. Saffman, “Quantum computing with neutral atoms,” *Physics Today*, vol. 70, no. 7, 2017.

[60] H. Bernien, S. Schwartz, A. Keesling, H. Levine, A. Omran, H. Pichler, S. Choi, A. S. Zibrov, M. Endres, M. Greiner *et al.*, “Probing many-body dynamics on a 51-atom quantum simulator,” *Nature*, vol. 551, no. 7682, pp. 579–584, 2017.

[61] C. Schreyvogel, V. Polyakov, R. Wunderlich, J. Meijer, and C. Nebel, “Active charge state control of single nv centres in diamond by in-plane al-schottky junctions,” *Scientific reports*, vol. 5, no. 1, pp. 1–12, 2015.

[62] S. B. van Dam, M. Walsh, M. J. Degen, E. Bersin, S. L. Mouradian, A. Galiullin, M. Ruf, M. IJspeert, T. H. Taminiau, R. Hanson *et al.*, “Optical coherence of diamond nitrogen-vacancy centers formed by ion implantation and annealing,” *Physical Review B*, vol. 99, no. 16, p. 161203, 2019.

[63] C. Adami and N. J. Cerf, “Quantum computation with linear optics,” 1998.

[64] E. Knill, R. Laflamme, and G. J. Milburn, “A scheme for efficient quantum computation with linear optics,” *nature*, vol. 409, no. 6816, pp. 46–52, 2001.

[65] P. Kok, W. J. Munro, K. Nemoto, T. C. Ralph, J. P. Dowling, and G. J. Milburn, “Linear optical quantum computing with photonic qubits,” *Reviews of modern physics*, vol. 79, no. 1, p. 135, 2007.

[66] R. Raussendorf and H. J. Briegel, “A one-way quantum computer,” *Physical review letters*, vol. 86, no. 22, p. 5188, 2001.

[67] R. Raussendorf and H. Briegel, “Computational model underlying the one-way quantum computer,” *arXiv preprint quant-ph/0108067*, 2001.

[68] R. Raussendorf, D. Browne, and H. Briegel, “The one-way quantum computer–a non-network model of quantum computation,” *journal of modern optics*, vol. 49, no. 8, pp. 1299–1306, 2002.

[69] R. Raussendorf, D. E. Browne, and H. J. Briegel, “Measurement-based quantum computation on cluster states,” *Physical review A*, vol. 68, no. 2, p. 022312, 2003.

[70] D. Hanneke, J. Home, J. D. Jost, J. M. Amini, D. Leibfried, and D. J. Wineland, “Realization of a programmable two-qubit quantum processor,” *Nature Physics*, vol. 6, no. 1, pp. 13–16, 2010.

[71] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O’brien, “Experimental realization of shor’s quantum factoring algorithm using qubit recycling,” *Nature photonics*, vol. 6, no. 11, pp. 773–776, 2012.

[72] J. Carolan, C. Harrold, C. Sparrow, E. Martín-López, N. J. Russell, J. W. Silverstone, P. J. Shadbolt, N. Matsuda, M. Oguma, M. Itoh *et al.*, “Universal linear optics,” *Science*, vol. 349, no. 6249, pp. 711–716, 2015.

[73] X. Qiang, X. Zhou, J. Wang, C. M. Wilkes, T. Loke, S. O’Gara, L. Kling, G. D. Marshall, R. Santagati, T. C. Ralph *et al.*, “Large-scale silicon quantum photonics implementing arbitrary two-qubit processing,” *Nature photonics*, vol. 12, no. 9, pp. 534–539, 2018.

[74] S. Sun, H. Kim, Z. Luo, G. S. Solomon, and E. Waks, “A single-photon switch and transistor enabled by a solid-state quantum memory,” *Science*, vol. 361, no. 6397, pp. 57–60, 2018.

[75] L.-M. Duan and H. Kimble, “Scalable photonic quantum computation through cavity-assisted interactions,” *Physical review letters*, vol. 92, no. 12, p. 127902, 2004.

[76] M. Z. Hasan and C. L. Kane, “Colloquium: topological insulators,” *Reviews of modern physics*, vol. 82, no. 4, p. 3045, 2010.

[77] X.-L. Qi and S.-C. Zhang, “Topological insulators and superconductors,” *Reviews of Modern Physics*, vol. 83, no. 4, p. 1057, 2011.

[78] L. Lu, J. D. Joannopoulos, and M. Soljačić, “Topological photonics,” *Nature photonics*, vol. 8, no. 11, pp. 821–829, 2014.

[79] T. Ozawa, H. M. Price, A. Amo, N. Goldman, M. Hafezi, L. Lu, M. C. Rechtsman, D. Schuster, J. Simon, O. Zilberberg *et al.*, “Topological photonics,” *Reviews of Modern Physics*, vol. 91, no. 1, p. 015006, 2019.

[80] J.-S. Xu, K. Sun, Y.-J. Han, C.-F. Li, J. K. Pachos, and G.-C. Guo, “Simulating the exchange of majorana zero modes with a photonic system,” *Nature communications*, vol. 7, no. 1, p. 13194, 2016.

[81] Z. Wang, Y. Chong, J. D. Joannopoulos, and M. Soljačić, “Observation of unidirectional backscattering-immune topological electromagnetic states,” *Nature*, vol. 461, no. 7265, pp. 772–775, 2009.

[82] M. Hafezi, E. A. Demler, M. D. Lukin, and J. M. Taylor, “Robust optical delay lines with topological protection,” *Nature Physics*, vol. 7, no. 11, pp. 907–912, 2011.

[83] M. C. Rechtsman, J. M. Zeuner, Y. Plotnik, Y. Lumer, D. Podolsky, F. Dreisow, S. Nolte, M. Segev, and A. Szameit, “Photonic floquet topological insulators,” *Nature*, vol. 496, no. 7444, pp. 196–200, 2013.

[84] M. Hafezi, S. Mittal, J. Fan, A. Migdall, and J. Taylor, “Imaging topological edge states in silicon photonics,” *Nature Photonics*, vol. 7, no. 12, pp. 1001–1005, 2013.

[85] A. Blanco-Redondo, I. Andonegui, M. J. Collins, G. Harari, Y. Lumer, M. C. Rechtsman, B. J. Eggleton, and M. Segev, “Topological optical waveguiding in silicon and the transition between topological and trivial defect states,” *Physical review letters*, vol. 116, no. 16, p. 163901, 2016.

[86] S. Mittal, S. Ganeshan, J. Fan, A. Vaezi, and M. Hafezi, “Measurement of topological invariants in a 2d photonic system,” *Nature Photonics*, vol. 10, no. 3, pp. 180–183, 2016.

[87] L. Xiao, X. Zhan, Z. Bian, K. Wang, X. Zhang, X. Wang, J. Li, K. Mochizuki, D. Kim, N. Kawakami *et al.*, “Observation of topological edge states in parity–time-symmetric quantum walks,” *Nature Physics*, vol. 13, no. 11, pp. 1117–1123, 2017.

[88] W.-J. Chen, M. Xiao, and C. T. Chan, “Photonic crystals possessing multiple weyl points and the experimental observation of robust surface states,” *Nature communications*, vol. 7, no. 1, p. 13038, 2016.

[89] J. Noh, S. Huang, D. Leykam, Y. Chong, K. Chen, and M. Rechtsman, “Experimental observation of optical weyl points,” in *European Quantum Electronics Conference*. Optica Publishing Group, 2017, p. JSIV_2_1.

[90] F. Li, X. Huang, J. Lu, J. Ma, and Z. Liu, “Weyl points and fermi arcs in a chiral phononic crystal,” *Nature Physics*, vol. 14, no. 1, pp. 30–34, 2018.

[91] Y. E. Kraus, Y. Lahini, Z. Ringel, M. Verbin, and O. Zilberberg, “Topological states and adiabatic pumping in quasicrystals,” *Physical review letters*, vol. 109, no. 10, p. 106402, 2012.

[92] M. Verbin, O. Zilberberg, Y. Lahini, Y. E. Kraus, and Y. Silberberg, “Topological pumping over a photonic fibonacci quasicrystal,” *Physical Review B*, vol. 91, no. 6, p. 064201, 2015.

[93] O. Zilberberg, S. Huang, J. Guglielmon, M. Wang, K. P. Chen, Y. E. Kraus, and M. C. Rechtsman, “Photonic topological boundary pumping as a probe of 4d quantum hall physics,” *Nature*, vol. 553, no. 7686, pp. 59–62, 2018.

[94] A. Blanco-Redondo, B. Bell, M. Segev, and B. Eggleton, “Photonic quantum walks with symmetry protected topological phases,” in *AIP Conference Proceedings*, vol. 1874, no. 1. AIP Publishing LLC, 2017, p. 020001.

[95] S. Mittal, E. A. Goldschmidt, and M. Hafezi, “A topological source of quantum light,” *Nature*, vol. 561, no. 7724, pp. 502–506, 2018.

[96] J.-L. Tambasco, G. Corrielli, R. J. Chapman, A. Crespi, O. Zilberberg, R. Osellame, and A. Peruzzo, “Quantum interference of topological states of light,” *Science advances*, vol. 4, no. 9, p. eaat3187, 2018.

[97] C.-K. Hong, Z.-Y. Ou, and L. Mandel, “Measurement of subpicosecond time intervals between two photons by interference,” *Physical review letters*, vol. 59, no. 18, p. 2044, 1987.

[98] T. Watson, S. Philips, E. Kawakami, D. Ward, P. Scarlino, M. Veldhorst, D. Savage, M. Lagally, M. Friesen, S. Coppersmith *et al.*, “A programmable two-qubit quantum processor in silicon,” *nature*, vol. 555, no. 7698, pp. 633–637, 2018.

[99] V. Bouchiat, D. Vion, P. Joyez, D. Esteve, and M. Devoret, “Quantum coherence with a single cooper pair,” *Physica Scripta*, vol. 1998, no. T76, p. 165, 1998.

[100] E. National Academies of Sciences, Medicine *et al.*, “Quantum computing: progress and prospects,” 2019.

[101] P. J. O’Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding *et al.*, “Scalable quantum simulation of molecular energies,” *Physical Review X*, vol. 6, no. 3, p. 031007, 2016.

[102] S. Rosenblum, Y. Y. Gao, P. Reinhold, C. Wang, C. J. Axline, L. Frunzio, S. M. Girvin, L. Jiang, M. Mirrahimi, M. H. Devoret *et al.*, “A cnot gate between multiphoton qubits encoded in two cavities,” *Nature communications*, vol. 9, no. 1, p. 652, 2018.

[103] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, “A quantum engineer’s guide to superconducting qubits,” *Applied physics reviews*, vol. 6, no. 2, p. 021318, 2019.

[104] A. Roy and D. P. DiVincenzo, “Topological quantum computing,” *arXiv preprint arXiv:1701.05052*, 2017.

[105] A. Y. Kitaev, “Fault-tolerant quantum computation by anyons,” *Annals of physics*, vol. 303, no. 1, pp. 2–30, 2003.

[106] M. Freedman, A. Kitaev, M. Larsen, and Z. Wang, “Topological quantum computation,” *Bulletin of the American Mathematical Society*, vol. 40, no. 1, pp. 31–38, 2003.

[107] V. Lahtinen and J. Pachos, “A short introduction to topological quantum computation,” *SciPost Physics*, vol. 3, no. 3, p. 021, 2017.

[108] J. G. Bohnet, B. C. Sawyer, J. W. Britton, M. L. Wall, A. M. Rey, M. Foss-Feig, and J. J. Bollinger, “Quantum spin dynamics and entanglement generation with hundreds of trapped ions,” *Science*, vol. 352, no. 6291, pp. 1297–1301, 2016.

[109] D. Lucas, C. Donald, J. Home, M. McDonnell, A. Ramos, D. Stacey, J.-P. Stacey, A. Steane, and S. Webster, “Oxford ion-trap quantum computing project,” *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 361, no. 1808, pp. 1401–1408, 2003.

[110] N. Friis, O. Marty, C. Maier, C. Hempel, M. Holzäpfel, P. Jurcevic, M. B. Plenio, M. Huber, C. Roos, R. Blatt *et al.*, “Observation of entangled states of a fully controlled 20-qubit system,” *Physical Review X*, vol. 8, no. 2, p. 021012, 2018.

[111] D. R. Leibrandt, J. Labaziewicz, V. Vuletić, and I. L. Chuang, “Cavity sideband cooling of a single trapped ion,” *Physical review letters*, vol. 103, no. 10, p. 103001, 2009.

[112] C. Flühmann, T. L. Nguyen, M. Marinelli, V. Negnevitsky, K. Mehta, and J. Home, “Encoding a qubit in a trapped-ion mechanical oscillator,” *Nature*, vol. 566, no. 7745, pp. 513–517, 2019.

[113] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, “Trapped-ion quantum computing: Progress and challenges,” *Applied Physics Reviews*, vol. 6, no. 2, p. 021314, 2019.

[114] R. LaRose, “Overview and comparison of gate level quantum software platforms,” *Quantum*, vol. 3, p. 130, 2019.

[115] M. Fingerhuth, T. Babej, and P. Wittek, “Open source software in quantum computing,” *PloS one*, vol. 13, no. 12, p. e0208561, 2018.

[116] E. S. Fried, N. P. Sawaya, Y. Cao, I. D. Kivlichan, J. Romero, and A. Aspuru-Guzik, “qtorch: The quantum tensor contraction handler,” *PloS one*, vol. 13, no. 12, p. e0208510, 2018.

[117] M. Smelyanskiy, N. P. Sawaya, and A. Aspuru-Guzik, “qhipster: The quantum high performance software testing environment,” *arXiv preprint arXiv:1601.07195*, 2016.

[118] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, “Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels,” *Physical review letters*, vol. 70, no. 13, p. 1895, 1993.

[119] D. Boschi, S. Branca, F. De Martini, L. Hardy, and S. Popescu, “Experimental realization of teleporting an unknown pure quantum state via dual classical and einstein-podolsky-rosen channels,” *Physical Review Letters*, vol. 80, no. 6, p. 1121, 1998.

[120] K. S. Chou, J. Z. Blumoff, C. S. Wang, P. C. Reinhold, C. J. Axline, Y. Y. Gao, L. Frunzio, M. Devoret, L. Jiang, and R. Schoelkopf, “Deterministic teleportation of a quantum gate between two logical qubits,” *Nature*, vol. 561, no. 7723, pp. 368–373, 2018.

[121] C. H. Bennett and S. J. Wiesner, “Communication via one-and two-particle operators on einstein-podolsky-rosen states,” *Physical review letters*, vol. 69, no. 20, p. 2881, 1992.

[122] C. Wang, F.-G. Deng, Y.-S. Li, X.-S. Liu, and G. L. Long, “Quantum secure direct communication with high-dimension quantum superdense coding,” *Physical Review A*, vol. 71, no. 4, p. 044305, 2005.

[123] K. Mattle, H. Weinfurter, P. G. Kwiat, and A. Zeilinger, “Dense coding in experimental quantum communication,” *Physical Review Letters*, vol. 76, no. 25, p. 4656, 1996.

[124] S. Bravyi, D. Gosset, and R. König, “Quantum advantage with shallow circuits,” *Science*, vol. 362, no. 6412, pp. 308–311, 2018.

[125] S. Bravyi, D. Gosset, R. Koenig, and M. Tomamichel, “Quantum advantage with noisy shallow circuits,” *Nature Physics*, vol. 16, no. 10, pp. 1040–1045, 2020.

[126] M. Loceff, “A course in quantum computing for the community college. vol. 1,” *Foothill College.—2015*.

[127] E. Bernstein and U. Vazirani, “Proceedings of the 25th annual acm symposium on theory of computing,” *San Diego, CA*, 1993.

[128] N. D. Mermin, *Quantum computer science: an introduction*. Cambridge University Press, 2007.

[129] C. Gidney and M. Ekerå, “How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits,” *Quantum*, vol. 5, p. 433, 2021.

[130] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, “Strengths and weaknesses of quantum computing,” *SIAM journal on Computing*, vol. 26, no. 5, pp. 1510–1523, 1997.

[131] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’brien, “A variational eigenvalue solver on a photonic quantum processor,” *Nature communications*, vol. 5, no. 1, p. 4213, 2014.

[132] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, “The theory of variational hybrid quantum-classical algorithms,” *New Journal of Physics*, vol. 18, no. 2, p. 023023, 2016.

[133] O. Higgott, D. Wang, and S. Brierley, “Variational quantum computation of excited states,” *Quantum*, vol. 3, p. 156, 2019.

[134] D. Wecker, M. B. Hastings, and M. Troyer, “Progress towards practical quantum variational algorithms,” *Physical Review A*, vol. 92, no. 4, p. 042303, 2015.

[135] D. Wang, O. Higgott, and S. Brierley, “Accelerated variational quantum eigensolver,” *Physical review letters*, vol. 122, no. 14, p. 140504, 2019.

[136] S. Sim, Y. Cao, J. Romero, P. D. Johnson, and A. Aspuru-Guzik, “A framework for algorithm deployment on cloud-based quantum computers,” *arXiv preprint arXiv:1810.10576*, 2018.

[137] J. R. McClean, M. E. Kimchi-Schwartz, J. Carter, and W. A. De Jong, “Hybrid quantum-classical hierarchy for mitigation of decoherence and determination of excited states,” *Physical Review A*, vol. 95, no. 4, p. 042308, 2017.

[138] J. R. McClean, Z. Jiang, N. C. Rubin, R. Babbush, and H. Neven, “Decoding quantum errors with subspace expansions,” *Nature communications*, vol. 11, no. 1, p. 636, 2020.

[139] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, “Barren plateaus in quantum neural network training landscapes,” *Nature communications*, vol. 9, no. 1, p. 4812, 2018.

[140] E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” *arXiv preprint arXiv:1411.4028*, 2014.

[141] M. Born, “Das adiabatenprinzip in der quantenmechanik,” *Zeitschrift für Physik*, vol. 40, no. 3-4, pp. 167–192, 1927.

[142] M. Born and V. Fock, “Beweis des adiabatensatzes,” *Zeitschrift für Physik*, vol. 51, no. 3-4, pp. 165–180, 1928.

[143] O. V. Besov, “Trudy matematicheskogo instituta imeni va steklova,” in *Proc. Steklov Inst. Math*, vol. 284, 2014, pp. 81–96.

[144] Y. Cao, G. G. Guerreschi, and A. Aspuru-Guzik, “Quantum neuron: an elementary building block for machine learning on quantum computers,” *arXiv preprint arXiv:1711.11240*, 2017.

[145] J. Romero and A. Aspuru-Guzik, “Variational quantum generators: Generative adversarial quantum machine learning for continuous distributions,” *Advanced Quantum Technologies*, vol. 4, no. 1, p. 2000003, 2021.

[146] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. S. Fried, S. Hong *et al.*, “Unsupervised machine learning on a hybrid quantum computer,” *arXiv preprint arXiv:1712.05771*, 2017.

[147] E. Farhi and H. Neven, “Classification with quantum neural networks on near term processors,” *arXiv preprint arXiv:1802.06002*, 2018.

[148] P. Wittek and C. Gogolin, “Quantum enhanced inference in markov logic networks,” *Scientific reports*, vol. 7, no. 1, pp. 1–8, 2017.

[149] I. Kerenidis and A. Prakash, “Quantum recommendation systems,” *arXiv preprint arXiv:1603.08675*, 2016.

[150] E. Tang, “A quantum-inspired classical algorithm for recommendation systems,” in *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, 2019, pp. 217–228.

[151] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, “Supervised learning with quantum-enhanced feature spaces,” *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.

[152] M. Schuld and N. Killoran, “Quantum machine learning in feature hilbert spaces,” *Physical review letters*, vol. 122, no. 4, p. 040504, 2019.

[153] G. Verdon, J. Pye, and M. Broughton, “A universal training algorithm for quantum deep learning,” *arXiv preprint arXiv:1806.09729*, 2018.

[154] M. Mohseni, P. Read, H. Neven, S. Boixo, V. Denchev, R. Babbush, A. Fowler, V. Smelyanskiy, and J. Martinis, “Commercialize quantum technologies in five years,” *Nature*, vol. 543, no. 7644, pp. 171–174, 2017.

[155] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, and M. Troyer, “Elucidating reaction mechanisms on quantum computers,” *Proceedings of the national academy of sciences*, vol. 114, no. 29, pp. 7555–7560, 2017.

[156] J. Olson, Y. Cao, J. Romero, P. Johnson, P.-L. Dallaire-Demers, N. Sawaya, P. Narang, I. Kivlichan, M. Wasielewski, and A. Aspuru-Guzik, “Quantum information and computation for chemistry,” *arXiv preprint arXiv:1706.05413*, 2017.

[157] F. Verstraete and J. I. Cirac, “Renormalization algorithms for quantum-many body systems in two and higher dimensions,” *arXiv preprint cond-mat/0407066*, 2004.

[158] S. R. White, “Density matrix formulation for quantum renormalization groups,” *Physical review letters*, vol. 69, no. 19, p. 2863, 1992.

[159] U. Schollwöck, “The density-matrix renormalization group,” *Reviews of modern physics*, vol. 77, no. 1, p. 259, 2005.

[160] G. Vidal, “Entanglement renormalization,” *Physical review letters*, vol. 99, no. 22, p. 220405, 2007.

[161] P. Hayden, S. Nezami, X.-L. Qi, N. Thomas, M. Walter, and Z. Yang, “Holographic duality from random tensor networks,” *Journal of High Energy Physics*, vol. 2016, no. 11, pp. 1–56, 2016.

[162] A. Milsted and G. Vidal, “Tensor networks as conformal transformations,” *arXiv preprint arXiv:1805.12524*, 2018.

[163] J. Preskill, “Quantum computing and the entanglement frontier,” *arXiv preprint arXiv:1203.5813*, 2012.

[164] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, “Characterizing quantum supremacy in near-term devices,” *Nature Physics*, vol. 14, no. 6, pp. 595–600, 2018.

[165] A. W. Harrow and A. Montanaro, “Quantum computational supremacy,” *Nature*, vol. 549, no. 7671, pp. 203–209, 2017.

[166] A. Bouland, B. Fefferman, C. Nirkhe, and U. Vazirani, “Quantum supremacy and the complexity of random circuit sampling,” *arXiv preprint arXiv:1803.04402*, 2018.

[167] R. Movassagh, “Efficient unitary paths and quantum computational supremacy: A proof of average-case hardness of random circuit sampling,” *arXiv preprint arXiv:1810.04681*, 2018.

[168] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. V. Isakov, V. Smelyanskiy, A. Megrant, B. Chiaro, A. Dunsworth, K. Arya *et al.*, “A blueprint for demonstrating quantum supremacy with superconducting qubits,” *Science*, vol. 360, no. 6385, pp. 195–199, 2018.

[169] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo, “Quantum supremacy is both closer and farther than it appears,” *arXiv preprint arXiv:1807.10749*, 2018.

[170] S. Aaronson and A. Arkhipov, “The computational complexity of linear optics,” in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011, pp. 333–342.

[171] N. Spagnolo, C. Vitelli, M. Bentivegna, D. J. Brod, A. Crespi, F. Flamini, S. Giacomini, G. Milani, R. Ramponi, P. Mataloni *et al.*, “Experimental validation of photonic boson sampling,” *Nature Photonics*, vol. 8, no. 8, pp. 615–620, 2014.

[172] S. B. Bravyi and A. Y. Kitaev, “Quantum codes on a lattice with boundary,” *arXiv preprint quant-ph/9811052*, 1998.

[173] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, “Topological quantum memory,” *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.

[174] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, “Surface codes: Towards practical large-scale quantum computation,” *Physical Review A*, vol. 86, no. 3, p. 032324, 2012.

[175] P. Baireuther, T. E. O’Brien, B. Tarasinski, and C. W. Beenakker, “Machine-learning-assisted correction of correlated qubit errors in a topological code,” *Quantum*, vol. 2, p. 48, 2018.

[176] N. Ofek, A. Petrenko, R. Heeres, P. Reinhold, Z. Leghtas, B. Vlastakis, Y. Liu, L. Frunzio, S. Girvin, L. Jiang *et al.*, "Extending the lifetime of a quantum bit with error correction in superconducting circuits," *Nature*, vol. 536, no. 7617, pp. 441–445, 2016.

[177] A. Almheiri, X. Dong, and D. Harlow, "Bulk locality and quantum error correction in ads/cft," *Journal of High Energy Physics*, vol. 2015, no. 4, pp. 1–34, 2015.

[178] L. Susskind, "Dear qubitzers, gr= qm," *arXiv preprint arXiv:1708.03040*, 2017.

[179] J. Von Neumann, *Mathematical foundations of quantum mechanics: New edition*. Princeton university press, 2018, vol. 53.

[180] S. Axler, *Linear algebra done right*. Springer Science & Business Media, 1997.

[181] M. Artin, "Algebra (vol. 2.)," 2010.

[182] E. G. Rieffel and W. H. Polak, *Quantum computing: A gentle introduction*. MIT Press, 2011.

[183] J. B. Fraleigh, *A first course in abstract algebra*. Pearson Education India, 2003.

[184] D. S. Dummit and R. M. Foote, *Abstract algebra*. Wiley Hoboken, 2004, vol. 3.

[185] J. J. Rotman, *Advanced modern algebra*. American Mathematical Soc., 2010, vol. 114.

[186] F. W. Lawvere and S. H. Schanuel, *Conceptual mathematics: a first introduction to categories*. Cambridge University Press, 2009.

[187] T.-D. Bradley, "What is applied category theory?" *arXiv preprint arXiv:1809.05923*, 2018.

[188] E. Riehl, *Category theory in context*. Courier Dover Publications, 2017.

[189] F. Verstraete, V. Murg, and J. I. Cirac, "Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems," *Advances in physics*, vol. 57, no. 2, pp. 143–224, 2008.

A

线性代数

在量子计算中，有一些数学基础是特别重要的：

- **线性代数**：线性代数是量子计算中最基础的数学工具。量子位的状态可以用复数表示，而量子操作可以表示为线性变换或矩阵的乘法。量子态的叠加和纠缠都可以通过线性代数的概念来描述和分析。
- 希尔伯特空间：在量子力学中，希尔伯特空间是描述量子系统状态的数学空间。量子比特的状态可以用希尔伯特空间中的向量表示。希尔伯特空间提供了描述量子态和量子操作的框架。
- 变换和算符：量子计算中的操作和变换可以用数学算符来表示。例如，量子比特的门操作可以表示为幺正矩阵。量子计算中的测量和观测也可以用算符表示。
- 概率与统计：量子力学中的测量结果是概率性的，因此概率与统计的概念在量子计算中起着重要作用。概率幅和概率分布是描述量子测量结果的数学工具。
- 复数和复数运算：量子计算中使用的数值表示通常是复数。量子态的叠加和纠缠都涉及到复数运算。复数的相位和幅度对量子计算中的相互干涉和幺正性质有重要影响。

这些数学基础为量子计算的理论建模、算法设计和实验分析提供了关键工具。在量子计算领域，数学与物理相结合，共同推动了该领域的发展和研究。

A.1. 引言

二十世纪量子力学最重要的发现之一是由约翰·冯·诺伊曼在他的《量子力学的数学基础》中观察到，整个量子力学可以用线性代数来描述 [179] 。

在本部分，我们将介绍一些基本概念和主题，这些内容是理解量子计算的基础。我们还提供了一些自测题，以帮助您巩固所学知识。

在引言部分，我们将探讨量子计算的动机和目标，以及它与经典计算的区别。我们还会介绍量子比特（qubit）和量子叠加的概念，以及量子态的表示和演化。

练习

概念回顾

- 函数

$$T : \mathbb{R} \to \mathbb{R} \tag{A.1}$$

$$T(x) := x + 1 \tag{A.2}$$

是一个线性变换吗?

- 二元操作与二进制代码有关吗?
- 哪个空间的维度更大：$\mathbb{R}^4$ 还是 $\mathbb{C}^2$?
- 在以下表达式中：
  - $\langle 0|1\rangle$[1]
  - $|0\rangle \langle 1|$
  - $\langle 0|1\rangle |0\rangle$
  - $\langle i|A|j\rangle$，其中 A 是一个矩阵，i 和 j 是数字

哪个是一个数? 哪个是一个向量? 哪个是一个矩阵?

- 给出一个具有非实数特征值的厄米算符的例子。

答案：

- 不是，它明显不是线性的。
- 不是，二元操作是一种特殊类型的函数。
- $\mathbb{R}^4$ 和 $\mathbb{C}^2$ 在 $\mathbb{R}$ 上具有相同的维数。
- 这里是四个表达式在狄拉克符号中的分类：
  - 表达式 $\langle 0|1\rangle$ 是一个数。实际上，$\langle 0|1\rangle$ 等于 0。
  - 表达式 $|0\rangle \langle 1|$ 是一个矩阵，具体而言，是矩阵

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{A.3}$$

  - 表达式 $\langle 0|1\rangle |0\rangle$ 是一个向量。可以看到 $\langle 0|1\rangle$ 是数值 0，而我们在本书前面提到过 $|0\rangle$ 是向量

$$|0\rangle := \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{A.4}$$

展开

$$\langle 0|1\rangle |0\rangle = (\langle 0|1\rangle) |0\rangle = (0) |0\rangle = (0) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \cdot 1 \\ 0 \cdot 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{A.5}$$

在这个计算过程中，我们依次执行以下步骤：

---

[1] $|0\rangle$ 和 $|1\rangle$ 是量子力学中的表示量子比特（qubit）状态的符号。它们代表了量子比特可能处于的两种基本状态。

$|0\rangle$ 表示量子比特处于基态（ground state），通常被称为" 零态"。它是一个定义为 $(1, 0)$ 的二维向量（在 Dirac 符号中表示为列矢量）。

$|1\rangle$ 则表示量子比特处于激发态（excited state），通常被称为" 一态"。它是一个定义为 $(0, 1)$ 的二维向量（在 Dirac 符号中表示为列矢量）。

$|0\rangle$ 和 $|1\rangle$ 是量子计算中最基本的比特状态。它们是正交的，即内积 $\langle 0|1| = \rangle 0$，表示它们彼此不重叠。

这两种基本状态可以通过量子门操作进行转换和操作，从而实现量子计算中的信息处理和运算。在量子算法中，$|0\rangle$ 和 $|1\rangle$ 作为比特的初始状态或中间状态被广泛使用。

- 首先，我们计算内积 $\langle 0|1\rangle$，得到数值 0。

  回忆一下：内积的计算是将行矢量的共轭转置乘以列矢量，然后将乘积的每个元素相加，所以计算过程如下：

$$\begin{aligned}\langle 0|1\rangle &= \begin{pmatrix}1 & 0\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix}\\ &= 1\cdot 0 + 0\cdot 1\\ &= 0 + 0\\ &= 0\end{aligned}$$

  因此，$\langle 0|1\rangle$ 的计算结果为 0。
- 然后，我们将这个数值与量子态 $|0\rangle$ 进行数乘运算[2]，即 $(0)\,|0\rangle$。
- 这里 $|0\rangle$ 是一个列矢量 $\begin{pmatrix}1\ 0\end{pmatrix}$。
- 将数值 0 与列矢量的每个元素进行乘法运算，得到 $\begin{pmatrix}0\cdot 1\ 0\cdot 0\end{pmatrix}$。
- 最终结果是 $\begin{pmatrix}0\\0\end{pmatrix}$，即 $\langle 0|1\rangle\,|0\rangle$ 的值为 $\begin{pmatrix}0\\0\end{pmatrix}$。

- This is a number. In particular, it's a clever way to write the entry in the i th row and j th column of the matrix A.

- 这样的东西是不存在的！厄米算符（Hermitian operators）总是具有实数特征值；尝试证明这个陈述 - 在本节中我们将会证明。

线性代数在量子计算中扮演着重要的角色。量子比特的状态可以用复数向量表示，而量子操作（例如量子门）可以表示为线性变换。线性代数的工具和概念，如线性空间、内积、矩阵运算等，为描述和分析量子系统提供了数学框架。通过线性代数的技巧，我们可以研究量子算法、量子编码、量子误差纠正等问题，并深入理解量子计算的原理和性质。

因此，掌握线性代数的基本概念和技巧对于理解和研究量子计算是至关重要的。它为我们提供了一种抽象的数学语言，用于描述和解决量子计算中的问题，并帮助我们构建更强大和可靠的量子计算系统。

[2]数乘运算是指将一个数（标量）与一个向量的每个元素相乘的运算。在线性代数中，数乘通常表示为 $k\mathbf{v}$，其中 k 是一个数（标量），$\mathbf{v}$ 是一个向量。

数乘运算的规则是将该数与向量的每个元素逐个相乘，得到一个新的向量。如果向量 $\mathbf{v} = \begin{pmatrix}v_1\\v_2\\\vdots\\v_n\end{pmatrix}$，则数乘 $k\mathbf{v}$ 的结果是 $\begin{pmatrix}kv_1\\kv_2\\\vdots\\kv_n\end{pmatrix}$。

数乘运算在线性代数和量子计算中经常被使用，用于调整向量的幅度、表示缩放或放大的操作。在量子计算中，数乘常常用于表示对量子态的幺正操作、调整相位或表示概率幅的变化。

在数学中，乘法运算是加法的补充，它展示了重复组合的概念。例如，将两个数相乘可以理解为在数轴上从零点开始，以一个数为步长重复移动另一个数的次数。

乘法运算可以应用于不同类型的数，包括整数、小数、分数、复数等。它也可以用于代数表达式和矩阵等数学对象的运算。

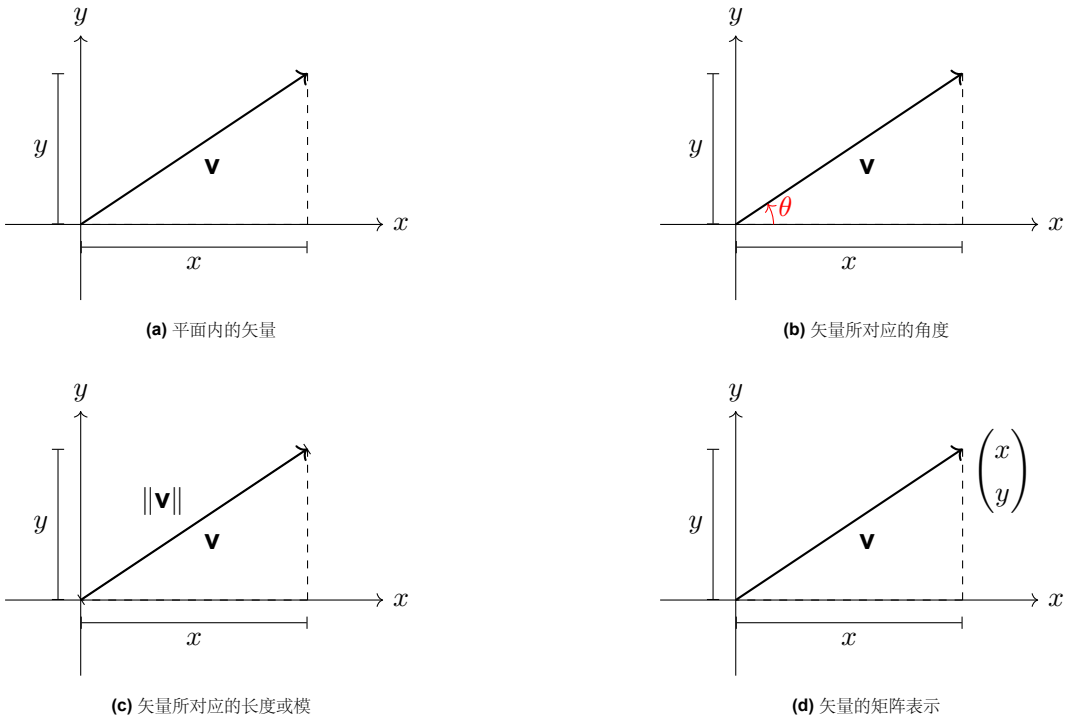
乘法运算在数学和实际生活中具有广泛的应用，它是数学中的基本运算之一，被广泛用于计算、代数、几何、物理学等领域。

A.2. 向量和表示法

向量是线性代数中的核心内容之一。有几种方法可以概念化一个向量。首先，我们可以将向量视为一组有序的数字（一维数组）。例如，下面的向量是由数字 x 如 1 和 y 如 2 组成的有序集合：

$$\begin{pmatrix} x \\ y \end{pmatrix} \text{ 如 } \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{A.6}$$

向量也可以被视为几何对象如图A.1子图 a。例如，我们可以在二维平面上绘制向量 $\begin{pmatrix} x \\ y \end{pmatrix}$，如图A.1子图 d 所示。



(a) 平面内的矢量

(b) 矢量所对应的角度

(c) 矢量所对应的长度或模

(d) 矢量的矩阵表示

图 A.1: 描述矢量的方法

从几何的角度来思考，向量可以具有大小（长度）和方向。例如，根据勾股定理，图A.1子图 a 中的向量具有以下长度：

$$\sqrt{1^2 + 2^2} \tag{A.7}$$

如图A.1子图 a 所示。我们可以通过给出从 x 轴到箭头头部所张角度来描述方向，如图A.1子图 b 所示。我们通常用小写粗体字母（如 v）或手写时使用箭头符号（如 $\overrightarrow{v}$）表示一个向量。当上下文清楚表明 v 是一个向量时，我们会使用简单的符号 v 来表示一个向量。

有时，我们会更明确地书写一个向量，如下所示：

$$v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix} \tag{A.8}$$

to indicate the number of entries in the vector. So, this vector has n entries. Some people like to denote vectors with square brackets, as below

$$v = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \tag{A.9}$$

but it doesn't matter which you use. We prefer round brackets in this discussion.

You may have already encountered the qubit, the quantum analog of the classical bit, earlier in the book. We use the example earlier in the book of a qubit that represents the polarization of light, which can be vertical or horizontal, or in some superposition of these states. As discussed in chapter 1, we can denote the state "vertical polarization"with $|0\rangle$ and the state "horizontal polarization"with $|1\rangle$, or equivalently, $|\uparrow\rangle$and $|\rightarrow\rangle$.

Vectors offer a convenient mathematical notation for these states. For instance, we denote the state "vertical polarization"$|0\rangle = |\uparrow\rangle$ the vector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and the state "horizontal polarization" $|1\rangle = |\rightarrow\rangle$ by the vector $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$.We refer to a vector of the form $\langle\varphi|$ as a "bra"and to a vector of the form $|\varphi\rangle$ as a "ket."So, "bras"are row vectors and "kets"are column vectors. This bra-ket notation was developed by Paul Dirac and is known as Dirac notation [68].

Basic Vector Operations

Now that we covered vector notation let's discuss what we can do with vectors. There are two quite natural operations to consider. The first of these is addition. We can add two vectors in the way you would expect –just add the entries! For example, we add the vectors

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} \; and \; \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{A.10}$$

as follows:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix} := \begin{pmatrix} 1+0 \\ 0+1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{A.11}$$

The notation ":="is used to denote equality that is true by definition and not by happenstance. For example, we would write $1+1 = 2$ and not $1+1 := 2$, since the fact that the sum of 1 and 1 is 2 is not by definition and i[a consequence of other facts. However, we would write $\mathbb{N} := \{0, 1, 2, 3, \dots\}$ to indicate the set of natural numbers, denoted $\mathbb{N}$, is equal to the set $\{0, 1, 2, 3, \dots\}$ by definition and not as a consequence of other facts.

A.3. 矩阵的线性变换

我们之前声称每个线性变换都有一个关联的矩阵，反之亦然。我们的目标是向您证明，实际上，线性变换和矩阵是同一回事。这就证明了线性代数研究矩阵及其上的操作的合理性。

在定义了基的前提下，我们准备定义在两个向量空间之间的线性变换关联的矩阵。

设 V 和 W 是某个域 $\mathbb{F}$ 上的向量空间。是的，它们必须是在相同域上的向量空间——您将会看到为什么！那么，V 和 W 分别有一个基，记为 $\mathcal{B}_V = \{v_1, \ldots, v_n\}$ 和 $\mathcal{B}_W = \{w_1, \ldots, w_n\}$。因此，$V$ 的维度为 n，W 的维度为 m。

设 T 是向量空间 V 和 W 之间的线性变换。因此，$T : V \to W$ 是一个线性变换，也是一个函数。由于 T 是从 V 到 W 的线性变换，对于每个基元素 v_j，其中 $i \in \{1, \ldots, n\}$，都被映射到 W，即 $Tv_j \in W$。由于 W 被向量 $w_i, \ldots, w_m$ 张成，并且 $Tv_j \in W$，存在系数 $a_{1,j}, \ldots, a_{m,j}$，使得 $Tv_j = a_{1,j}w_1 + \cdots + a_{m,j}w_m$，即我们可以将每个 Tv_j 表示为基元素 $w_i, \ldots, w_m$ 的线性组合。

因此，对于 V 的每个基元素 v_j，我们有一个与之关联的域 $\mathbb{F}$ 上的元素列表 $a_{1,j}, \ldots, a_{m,j}$。也许您能看到我们的目标...

我们逐个元素地定义与线性变换 $T : V \to W$ 关联的矩阵 $\mathcal{M}(T : V \to W)_{\mathcal{V},\mathcal{W}}$，相对于基 $\mathcal{B}_v$ 和 $\mathcal{B}_w$。

Definition Matrix associated to a linear transformation 与线性变换相关的定义矩阵

$$(\mathcal{M}(T : V \to W)_{\mathcal{B}_v,\mathcal{B}w}) := ai, j \tag{A.12}$$

其中，$a_{i,j}$ 是 $\mathcal{B}W$ 的第 i 个基向量在线性组合 $Tv_j = a1, jw_1 + \cdots + a_{m,j}w_m$ 中的系数，该线性组合表示了 $\mathcal{B}_V$ 的第 j 个基向量的映射 Tv_j。

练习：

你可能会想为什么我们不能对任意向量空间之间的变换使用相同的方法。更具体地说，为什么我们只对线性变换定义矩阵？你能找出其中的原因吗？

让我们看看如何使用矩阵构建。考虑由以下线性变换 $T : \mathbb{R}^2 \to \mathbb{R}^2$ 描述的情况：

$$T \begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} x + y \\ y - y \end{pmatrix} \tag{A.13}$$

其中，我们采用 $\mathbb{R}^2$ 的标准基作为基：

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} \tag{A.14}$$

至此，我们已经定义了变换 T，向量空间 V 和 W 都是 $\mathbb{R}^2$，而 $\mathcal{B}_V$ 和 $\mathcal{B}_W$ 都是 $\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\}$。

练习：

检验上述给定的变换 T 是否实际上是一个线性变换。

既然你已经验证了变换 T 是线性的，我们可以继续确定构造中的系数 $a_{i,j}$。这就是找出 T 将两个基向量映射到哪里。根据 T 的定义，我们有：

$$T \begin{pmatrix} 1 \\ 0 \end{pmatrix} := \begin{pmatrix} 1 + 0 \\ 1 - 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \tag{A.15}$$

然后，我们将其分解为基向量的线性组合，如下所示：

$$\begin{pmatrix}1\\1\end{pmatrix} = 1 \cdot \begin{pmatrix}1\\0\end{pmatrix} + 1 \cdot \begin{pmatrix}0\\1\end{pmatrix} \tag{A.16}$$

此时，我们可以看出：

$$(\mathcal{M}(T : V \to W)_{\mathcal{B}_V,\mathcal{B}_W})_{1,1} := a_{1,1} = 1 \tag{A.17}$$

以及

$$(\mathcal{M}(T : V \to W)_{\mathcal{B}_V,\mathcal{B}_W})_{2,1} := a_{2,1} = 1 \tag{A.18}$$

当理解向量空间 V、W 及其基 $\mathcal{B}_V$ 和 $\mathcal{B}_W$ 时，我们可以使用简写符号

$$\mathcal{M}(T) \tag{A.19}$$

来表示

练习

通过确定第二个基向量 $\begin{pmatrix}0\\1\end{pmatrix}$ 映射到哪里，并将结果向量分解为基向量 $\begin{pmatrix}1\\0\end{pmatrix}$ 和 $\begin{pmatrix}0\\1\end{pmatrix}$ 的线性组合，验证 $a_{1,2} = 1$ 和 $a_{2,2} = -1$。

在所有这些计算之后，我们意识到线性变换 $T : V = \mathbb{R}^2 \to W = \mathbb{R}^2$ 相对于基向量的矩阵表示为

$$\mathcal{B}_V = \mathcal{B}_W = \left\{ \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}0\\1\end{pmatrix} \right\} \tag{A.20}$$

为

$$\mathcal{M}(T) = \begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}, \tag{A.21}$$

。

练习对于线性变换，构造与线性变换关联的矩阵：

$$T : V = \mathbb{R}^3 \to W = \mathbb{R}^3 \tag{A.22}$$

定义如下：

$$T \begin{pmatrix}x\\y\\z\end{pmatrix} := \begin{pmatrix}1x + 2y + 3z\\4x + 5y + 6z\\7x + 8y + 9z\end{pmatrix} \tag{A.23}$$

相对于标准基向量：

$$\mathcal{B}_V = \mathcal{B}_W = \left\{ \begin{pmatrix}1\\0\\0\end{pmatrix}, \begin{pmatrix}0\\1\\0\end{pmatrix}, \begin{pmatrix}0\\0\\1\end{pmatrix} \right\} \tag{A.24}$$

首先，你应该验证这是一个线性变换。然后，确定每个基向量被映射到的位置。一旦你知道了这一点，将它们分解为基向量的线性组合，并使用这些线性组合的系数来构建矩阵。

除非另有说明，我们将假设向量空间 $\mathbb{R}^n$ 和 $\mathbb{C}^n$ 具有标准基。你可能对我们之前给出的矩阵乘法定义感到困惑。我们希望演示为什么它被定义为这样。这与两个变换（作为函数）的复合与我们定义的矩阵乘法之间的密切关系有关。

考虑以下两个变换 $R : \mathbb{R}^2 \to \mathbb{R}^2$ 和 $S : \mathbb{R}^2 \to \mathbb{R}^2$，定义如下：

$$R\begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} y \\ -x \end{pmatrix} \tag{A.25}$$

和

$$S\begin{pmatrix} x \\ y \end{pmatrix} := \begin{pmatrix} y \\ x \end{pmatrix}. \tag{A.26}$$

。

练习

首先，尝试从几何角度理解这些变换。然后，找出代表这些变换的矩阵 $\mathcal{M}(R)$ 和 $\mathcal{M}(S)$（使用标准基）。然后，回想一下我们如何计算两个矩阵的乘积，并计算以下矩阵乘积：

$$\mathcal{M}(R) \cdot \mathcal{M}(S) \quad \text{和} \quad \mathcal{M}(S) \cdot \mathcal{M}(R) \tag{A.27}$$

它们应该不相等，这可能在你的几何解释中是清晰的！

如果你完成了上述练习，现在你知道 R 和 S 的矩阵分别为：

$$\mathcal{M}(R) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \text{ 和 } \mathcal{M}(S) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{A.28}$$

并且乘积为：

$$\mathcal{M}(R) \cdot \mathcal{M}(S) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \text{ 和 } \mathcal{M}(S) \cdot \mathcal{M}(R) = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \tag{A.29}$$

。

根据给定的矩阵，我们有：

$$\mathcal{M}(R) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \quad \text{和} \quad \mathcal{M}(S) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{A.30}$$

要计算 $\mathcal{M}(R) \cdot \mathcal{M}(S)$，我们将矩阵相乘，得到：

$$\mathcal{M}(R) \cdot \mathcal{M}(S) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{A.31}$$

$$= \begin{pmatrix} (0 \cdot 0) + (1 \cdot 1) & (0 \cdot 1) + (1 \cdot 0) \\ (-1 \cdot 0) + (0 \cdot 1) & (-1 \cdot 1) + (0 \cdot 0) \end{pmatrix} \tag{A.32}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{A.33}$$

因此，$\mathcal{M}(R) \cdot \mathcal{M}(S)$ 的结果是：

$$\mathcal{M}(R) \cdot \mathcal{M}(S) = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{A.34}$$

根据给定的矩阵，我们有：

$$\mathcal{M}(R) = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \quad 和 \quad \mathcal{M}(S) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{A.35}$$

要计算 $\mathcal{M}(S) \cdot \mathcal{M}(R)$，我们将矩阵相乘，得到：

$$\mathcal{M}(S) \cdot \mathcal{M}(R) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix} \tag{A.36}$$

$$= \begin{pmatrix} (0 \cdot 0) + (1 \cdot -1) & (0 \cdot 1) + (1 \cdot 0) \\ (1 \cdot 0) + (0 \cdot -1) & (1 \cdot 1) + (0 \cdot 0) \end{pmatrix} \tag{A.37}$$

$$= \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \tag{A.38}$$

因此，$\mathcal{M}(S) \cdot \mathcal{M}(R)$ 的结果是：

$$\mathcal{M}(S) \cdot \mathcal{M}(R) = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \tag{A.39}$$

好的，现在我们准备计算变换（函数）R 和 S 的复合。请记住，我们从“内到外”进行计算。因此，首先应用 S，然后在 S 之后应用 R。因此，我们可以计算 R ∘ S 为：

$$(R \circ S) \begin{pmatrix} x \\ y \end{pmatrix} = R \left(S \begin{pmatrix} x \\ y \end{pmatrix} \right) = R \begin{pmatrix} y \\ x \end{pmatrix} = \begin{pmatrix} x \\ -y \end{pmatrix} \tag{A.40}$$

接下来，让我们计算 $S \circ R$：

$$(S \circ R) \begin{pmatrix} x \\ y \end{pmatrix} = S \left(R \begin{pmatrix} x \\ y \end{pmatrix} \right) = S \begin{pmatrix} y \\ -x \end{pmatrix} = \begin{pmatrix} -x \\ y \end{pmatrix} \tag{A.41}$$

。

它们并不相同！我们离类比越来越近了。现在，我们找到与变换 R ∘ S 相关联的矩阵。注意两个基向量最终的位置，然后将每个结果表示为另一侧基向量的线性组合。然后，这些线性组合中的系数将帮助我们构建矩阵。

$$(R \circ S) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ -0 \end{pmatrix} = 1 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 0 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{A.42}$$

和

$$(R \circ S) \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ -1 \end{pmatrix} = 0 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + (-1) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{A.43}$$

和

$$(S \circ R) \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \end{pmatrix} = (-1) \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} + 0 \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{A.44}$$

和

$$(S \circ R)\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}-0\\1\end{pmatrix} = 0 \cdot \begin{pmatrix}1\\0\end{pmatrix} + 1 \cdot \begin{pmatrix}0\\1\end{pmatrix} \tag{A.45}$$

现在，我们拥有构建矩阵 $\mathcal{M}(R \circ S)$ 和 $\mathcal{M}(S \circ R)$ 所需的一切!

练习

使用上述计算，确定矩阵 $\mathcal{M}(R \circ S)$ 和 $\mathcal{M}(S \circ R)$。

通过完成上述练习，你现在可以看到与变换 $R \circ S$ 和 $S \circ R$ 关联的矩阵是:

$$\mathcal{M}(R \circ S) = \begin{pmatrix}1 & 0\\0 & -1\end{pmatrix} \text{ 和 } \mathcal{M}(S \circ R) = \begin{pmatrix}-1 & 0\\0 & 1\end{pmatrix} \tag{A.46}$$

而这些矩阵正是我们之前计算的 $\mathcal{M}(R) \cdot \mathcal{M}(S)$ 和 $\mathcal{M}(S) \cdot \mathcal{M}(R)$! 顺便说一句，这并不仅仅是一个巧合。这就是我们定义矩阵乘法的动机所在。

线性变换的复合矩阵，对于所有的线性变换 S 和 T，线性变换的复合矩阵是它们矩阵的乘积:

$$\mathcal{M}(S \circ T) = \mathcal{M}(S) \cdot \mathcal{M}(T) \tag{A.47}$$

A.4. 矩阵运算符

我们将那些从向量空间映射回同一空间的线性变换称为该空间的线性算子。实际上，我们曾经讨论过的所有线性变换（除了我们用来解释矩阵转置操作的一个非方阵）都是线性算子! 在量子力学中，我们对这类算子特别关注。

更重要的是，我们重视可逆的线性算子，因为我们可以利用它们创建量子门和构建量子电路。接下来，我们会介绍行列式，这是一个矩阵的数值特性，它可以表示该矩阵所代表的空间变换是否可逆。

在这里，“线性算子”是指一种特殊的线性变换，它能够将向量空间中的向量映射回同一空间。这类操作在量子力学中非常重要，因为它们通常被用来描述系统的演化过程。

可逆线性算子则更加重要，在量子计算中，我们通常需要设计出可以反向操作的量子门。如果一个算子是可逆的，那么它就有一个逆算子，可以撤销它的操作。

行列式是一个非常有用的数学工具，它可以提供关于线性变换（由矩阵表示）是否可逆的信息。如果一个矩阵的行列式不为零，那么这个矩阵就是可逆的，也就是说，对应的线性变换也是可逆的。这一点对于设计可逆的量子门来说是非常关键的。

A.5. 行列式简介

每个方阵都有一个关联的行列式，它反映了由该矩阵表示的空间线性变换的几何特征。行列式随后描述了线性变换的可逆性，即变换是否可以撤销。

我们所说的变换可以被撤销是什么意思呢? 让我们先看一个可以被撤销的变换的例子，然后再看一个不能被撤销的例子。

首先，一个可以被撤销的变换的例子:

考虑线性变换 R_π 从 R^2 到 R^2 的描述矩阵为

$$R_\pi := \begin{pmatrix}1 & 0\\0 & e^{i\pi}\end{pmatrix} \tag{A.48}$$

练习

回顾本章前面的欧拉公式，并验证

$$e^{i\pi} = \cos(\pi) + i\sin(\pi) = -1 + 0 \cdot i = -1 \tag{A.49}$$

上述练习提醒我们，矩阵 R 实际上可以表示为

$$R_\pi := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{A.50}$$

练习

你可以进一步探究并找出这个矩阵描述的二维空间的变换。弄清楚基向量被映射到哪里，你就能够得到一个想法！

如果你尝试了上述练习，你可能已经发现矩阵 R_π 描述的是二维空间关于 x 轴的镜像。相信这一点，从几何上很明显这个变换是可逆的——只需再次镜像回来！或者你可以再次进行镜像，因为两次关于一条轴的镜像不会产生变化。你可能会想知道为什么这个矩阵的名称中有 π。我们给你一点思路。π 的存在表示该矩阵具有绕 z 轴旋转较大空间 $\mathbb{C}^2$ 的效果，角度为 π 弧度，尽管我们现在不打算深入讨论这个问题。我们希望你能理解这是一个可逆的空间线性变换——暂时就这样吧！你可能还注意到矩阵 R 是矩阵

$$R_\phi := \begin{pmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{pmatrix} \tag{A.51}$$

的特例，这个矩阵在第三章中提到过，其中角度 ϕ 为 π。

现在，让我们考虑一个无法撤销的变换。考虑从 $\mathbb{R}^2$ 到 $\mathbb{R}^2$ 的变换 Pro_{j_x}，其描述矩阵为

练习

与之前一样，你应该尝试找出由矩阵

$$Proj_x := \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{A.52}$$

描述的空间变换是什么。名称可能会给出答案...

因此，如果你尝试了上述练习，希望你确信这个矩阵描述了将向量投影到 x 轴上。例如，将 Proj_x 应用于向量 $(1\ 1)^T$，得到 $(1\ 0)^T$。由于许多向量也通过 Proj_x 映射到 $(1\ 1)^T$，如果我们只看输出，我们就无法知道输入是什么。例如，向量 $(1\ 2)^T$ 也映射到 $(1\ 0)^T$（实际上，任何第一个分量为 1 的向量也是如此！）。由于我们无法从输出中恢复输入，并且无法撤销这个变换，它是不可逆的。

实质上，这个变换将整个空间压缩到 x 轴上。这个想象中的图像应该给人一种无法撤销这个变换的印象。在某种意义上，一个更高维的空间被压缩成一个较低维的空间，因此必然会丢失信息。

在量子计算中，我们不喜欢这样的变换！我们希望代表我们使用的量子逻辑门的矩阵能够保留所有给定的信息，并且希望这些变换是可逆的。在接下来的内容中，我们将看到我们对这些矩阵提出了更高的要求。

所以，到目前为止，希望你确信并不是所有线性变换都可以撤销。我们希望有一个数值不变量来编码线性变换的可逆性。这就是行列式的作用。

让我们通过一个例子来看如何计算一个 2×2 矩阵的行列式。

考虑矩阵

$$A := \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \tag{A.53}$$

我们计算它的行列式如下

$$\mathsf{det}(A) := \begin{vmatrix} 1 & 2 \\ 3 & 4 \end{vmatrix} = 1 \cdot 4 - 2 \cdot 3 = 4 - 6 = -2 \tag{A.54}$$

练习

请尝试研究并描述由该变换描述的二维空间的变换。弄清楚基向量的映射结果，这应该能给你一个想法!

前面的练习希望能让你确信由矩阵 A 描述的二维空间变换是可逆的。

让我们继续这个例子，并计算上面描述的矩阵的行列式

$$\mathsf{det}(R_{\frac{\pi}{2}}) := \begin{vmatrix} 1 & 0 \\ 0 & -1 \end{vmatrix} = 1 \cdot (-1) - 0 \cdot 0 = -1 \tag{A.55}$$

$$\mathsf{det}(Proj_x) := \begin{vmatrix} 1 & 0 \\ 0 & 0 \end{vmatrix} = 1 \cdot 0 - 0 \cdot 0 = 0 \tag{A.56}$$

如果你观察细致，你会注意到与可逆变换相对应的矩阵具有非零行列式，而与不可逆变换相对应的矩阵具有零行列式。

2×2 矩阵的行列式

对于一个 2×2 的矩阵

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{A.57}$$

其行列式为

$$ad - bc \tag{A.58}$$

你可能会想知道如何计算一个 3×3 矩阵的行列式。同样，我们将通过示例来说明

$$
\begin{aligned}
\det\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix} &:= \begin{vmatrix}1&2&3\\4&5&6\\7&8&9\end{vmatrix} \\
&= 1\cdot\begin{vmatrix}5&6\\8&9\end{vmatrix} - 2\begin{vmatrix}4&6\\7&9\end{vmatrix} + 3\cdot\begin{vmatrix}4&5\\7&8\end{vmatrix} \\
&= 1\cdot(5\cdot 9 - 8\cdot 6) - 2\cdot(4\cdot 9 - 7\cdot 6) + 3\cdot(4\cdot 8 - 7\cdot 5) \\
&= 1\cdot(45-48) - 2\cdot(36-42) + 3\cdot(32-35) \\
&= 1\cdot(-3) - 2\cdot(-6) + 3\cdot(-3) \\
&= -3 + 12 - 9 \\
&= 9 - 9 \\
&= 0
\end{aligned}
\tag{A.59}
$$

A.6. 可逆性的等价表述

下面是对于线性变换 T 而言等价的表述：

- 行列式 $\det(T) = 0$：

  这表示 T 的行列式为零。在数学上，一个矩阵的行列式等于零，就表示这个矩阵是“奇异的”或者说“非满秩的”，它不能被逆。这是因为行列式可以被看作是将一个向量从源空间映射到目标空间的体积（或者说，面积）因子。如果行列式等于零，那么映射的结果将全部落在源空间的一个更低维度子空间内，这样的映射就无法被逆。

- T 不可逆：

  如果一个线性变换不可逆，那么它就不能找到一个逆变换，使得逆变换和原变换的复合等于恒等变换。这等价于说，原变换将某些不同的向量映射到了同一点，从而丢失了这些向量的原有信息。在这种情况下，行列式也会为零，因为该线性变换的映射结果落在了一个更低维度的子空间内。

- 表示 T 的矩阵的行向量线性相关：

  如果 T 对应的矩阵的行向量线性相关，那么这个矩阵就不能被逆。这是因为线性相关的行向量意味着在空间映射中存在冗余信息，不能形成一个一一对应的映射关系。在这种情况下，行列式也会为零。

- 表示 T 的矩阵的列向量线性相关：

  与行向量的情况类似，如果 T 对应的矩阵的列向量线性相关，那么这个矩阵也不能被逆。列向量的线性相关表示了空间映射的冗余性。在这种情况下，行列式也会为零。

所有这些属性都是对线性变换是否可以被逆的等价描述。尽管行列式的定义看起来像是一个组合的性质，但实际上，行列式和线性变换的几何性质有着深刻的联系。这是线性代数中的一个惊人而美妙的事实。

A.7. 行列式的几何性质

还有一种更几何化的行列式定义，考虑矩阵：

$$\begin{pmatrix} 1 & 4 \\ 2 & 2 \end{pmatrix} \tag{A.60}$$

再次观察到它将第一个基向量 $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ 映射为向量 $\begin{pmatrix} 1 \\ 2 \end{pmatrix}$，将第二个基向量 $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$ 映射为向量 $\begin{pmatrix} 4 \\ 2 \end{pmatrix}$。这应该使我们相信，由第一象限中基向量形成的单位正方形，如图 12.1 所示，被映射为以以下坐标点为顶点的平行四边形：

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 4 \\ 2 \end{pmatrix}, \begin{pmatrix} 5 \\ 4 \end{pmatrix} \tag{A.61}$$

如图 12.2 所示。

练习

画出这个平行四边形！

在某种意义上，单位正方形在被转化为平行四边形时，它的方向会“翻转”，因为构成单位正方形的向量 $\begin{pmatrix} 1 & 0 \end{pmatrix}$ 和 $\begin{pmatrix} 0 & 1 \end{pmatrix}$ 在形成平行四边形时交换了位置。

单位正方形的面积为 1，而平行四边形的面积为 2。那么这与我们得到的矩阵行列式值为-2 有何联系？关键在于，单位正方形的“翻转”被行列式的负值所编码。保持方向不变的变换对应着正的行列式，而改变方向的变换对应着负的行列式。此外，行列式的绝对值大小编码了单位平行四边形在变换过程中“拉伸”的比例。

总而言之，与线性变换相关的矩阵的行列式在几何上具有丰富的含义，它既编码了该变换是否保持方向，也编码了单位平行六面体（平行四边形的高维泛化）在变换过程中“拉伸”的比例。实际上，这是非常具有几何直观的！图 12.2 便描绘了这个平行四边形。

逆矩阵的 Matrix Inversion

我们之前讨论了矩阵的行列式如何编码其可逆性。但我们如何实际找到逆转换呢？

首先，当我们说逆转换时，我们指的是什么？嗯，转换只是函数的另一个词，因此我们想要找到给定转换的逆函数。然而，找到逆函数可能并不那么简单。为了理解这一点，请尝试找到空间中由以下描述的转换 T 的逆转换：

$$T := \mathbb{R}^3 \to \mathbb{R}^3 \tag{A.62}$$

$$T \begin{pmatrix} x \\ y \\ z \end{pmatrix} := \begin{pmatrix} 1 \cdot x + 2 \cdot y + 3 \cdot z \\ 4 \cdot x + 5 \cdot y + 6 \cdot z \\ 7 \cdot x + 8 \cdot y + 10 \cdot z \end{pmatrix} \tag{A.63}$$

这个矩阵不可逆是因为它的行列式为零。根据前面的讨论，行列式为零意味着线性变换不是可逆的，也就是说存在输入向量对应多个不同的输出向量，无法唯一地确定逆转换。在这种情况下，我们无法找到一个唯一的逆函数来完全还原原始的输入向量。所以，并非所有的转换都是可逆的。

练习

找到与变换 T 相关的矩阵 $\mathcal{T}$，然后计算其行列式。根据对行列式的计算结果，对变换 T 的可逆性作出结论。我们能够求逆 T 吗？

练习

根据给出的 T 的定义，我们能否给出关于这个三维空间变换的合理几何描述?

回顾我们之前关于函数和函数属性的讨论，我们称函数 $f^1 : X \to Y$ 可逆，当且仅当存在一个逆函数 $f^1 : Y \to X$，使得复合函数 $f^1 \circ f : X \to X$ 和 $f^1 \circ f : Y \to Y$ 分别等于对应的恒等函数 I_X 和 I_Y。因此，我们寻找一个变换（函数）T^1，它与变换 T 的复合在两边都是恒等变换。

如果我们找到这样一个 T^1，我们将知道

$$T \circ T^1 = I_{\mathbb{R}^3} \tag{A.64}$$

和

$$T^1 \circ T = I_{\mathbb{R}^3} \tag{A.65}$$

l

现在我们知道空间中的每个线性变换都可以表示为一个矩阵 $\mathcal{M}(T)$，而两个变换的复合类似于它们对应矩阵的乘法，我们可以希望将给定的变换 T 表示为一个矩阵 $\mathcal{M}(T)$。然后我们可以使用这个矩阵描述来找到一个逆矩阵，使得乘以 $\mathcal{M}(T)$ 后得到单位矩阵（对应于恒等函数）。

换句话说，我们将上述复合应用于“矩阵运算”，得到

$$\mathcal{M}(T \circ T^1) = \mathcal{M}(I_{\mathbb{R}^3}) \tag{A.66}$$

和

$$\mathcal{M}(T^1 \circ T) = \mathcal{M}(I_{\mathbb{R}^3}) \tag{A.67}$$

问题是这些矩阵是什么。然而，正如我们之前所说，复合变换的矩阵就是它们对应矩阵的乘积，所以我们有

$$\mathcal{M}(T)\mathcal{M}(T^1) = \mathcal{M}(I_{\mathbb{R}^3}) \tag{A.68}$$

和

$$\mathcal{M}(T^1)\mathcal{M}(T) = \mathcal{M}(I_{\mathbb{R}^3}) \tag{A.69}$$

我们需要找到一个矩阵，使得它在与矩阵 $\mathcal{M}(T)$ 的乘积时，无论在左侧还是右侧，都得到单位矩阵。这是一个非常重要的观察!

现在我们将注意力集中在如何找到这样一个矩阵上。考虑矩阵

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \tag{A.70}$$

让

$$B = \begin{pmatrix} x & y \\ z & w \end{pmatrix} \tag{A.71}$$

是任意的另一个矩阵，并假设 B 具有所需的逆属性，即 $AB = I_2 = BA$。那么，首先我们有

$$AB = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x & y \\ z & w \end{pmatrix} = I_2 \tag{A.72}$$

让我们解释一下

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x & y \\ z & w \end{pmatrix} \tag{A.73}$$

的含义。

回顾一下矩阵乘法的定义。在计算乘积之前，你可以试试看!

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} x & y \\ z & w \end{pmatrix} = \begin{pmatrix} 1 \cdot x + 2 \cdot z & 1 \cdot y + 2 \cdot w \\ 3 \cdot x + 4 \cdot z & 3 \cdot y + 4 \cdot w \end{pmatrix} \tag{A.74}$$

现在，我们希望这个矩阵等于单位矩阵，因此我们需要满足以下方程和不等式：

$$1x + 2z \neq 0 \tag{A.75}$$

$$1y + 2w = 0 \tag{A.76}$$

$$3x + 4z = 0 \tag{A.77}$$

$$3y + 4w \neq 0 \tag{A.78}$$

$$1x + 2z = 3y + 4w \tag{A.79}$$

第二个和第三个方程是显然成立的，但是第一个、第四个和第五个方程需要一些解释。我们希望第一、第四和第五个条目都是非零且相等的，以便我们可以除以它们并使对角线上的条目等于1。

继续练习

试着解上面的方程和不等式系统。

经过足够的努力，你会发现合适的 x, y, z, w 的选择是

$$x = 4, y = -2, z = -3, w = 1 \tag{A.80}$$

因此，我们要找的矩阵是

$$B = \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix} \tag{A.81}$$

我们来计算乘积

$$AB = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 4 & -2 \\ -3 & 1 \end{pmatrix} = \begin{pmatrix} -2 & 0 \\ 0 & -2 \end{pmatrix} \tag{A.82}$$

那不是单位矩阵! 事实上，情况并不糟糕。记得我们要求对角线上的条目非零且相等吗? 现在，你将看到原因。确实，矩阵 B 不起作用，但矩阵 $(\frac{1}{2}) \cdot B$ 起作用（记住如何将一个矩阵乘以一个数）

$$A\left(\begin{pmatrix}\frac{1}{-2}\end{pmatrix}B\right)=\begin{pmatrix}1&2\\3&4\end{pmatrix}\left(\begin{pmatrix}\frac{1}{-2}\end{pmatrix}\quad\begin{pmatrix}4&-2\\-3&1\end{pmatrix}\right) \tag{A.83}$$

$$=\begin{pmatrix}\frac{1}{-2}\end{pmatrix}\begin{pmatrix}1&2\\3&4\end{pmatrix}\begin{pmatrix}4&-2\\-3&1\end{pmatrix} \tag{A.84}$$

$$=\begin{pmatrix}\frac{1}{-2}\end{pmatrix}\begin{pmatrix}-2&0\\0&-2\end{pmatrix} \tag{A.85}$$

$$=\begin{pmatrix}(\frac{1}{2})\cdot(-2)&(\frac{1}{2})\cdot 0\\(\frac{1}{2})\cdot 0&(\frac{1}{2})\cdot(-2)\end{pmatrix} \tag{A.86}$$

$$=\begin{pmatrix}-1&0\\0&-1\end{pmatrix} \tag{A.87}$$

因此，我们有足够的理由将矩阵 $(\frac{1}{2})B$ 称为 A^1！

继续练习

我们应该检查在两边进行乘法是否得到单位矩阵-这样做吧！也就是说，验证 $((-\frac{1}{2}))A=I$。

然后，我们已经找到了矩阵 A 的逆矩阵，所以只需要确定这个逆矩阵代表的线性变换是什么。回想一下，矩阵中的条目恰好表示基元素的映射系数，我们可以轻松地恢复出这个变换：

练习

尝试用函数表示法描述与矩阵 $(\frac{1}{2})\cdot B$ 对应的线性变换。填写下面未完成的方程：

$$\begin{pmatrix}\frac{1}{-2}\end{pmatrix}\cdot B\begin{pmatrix}x\\y\end{pmatrix}:= \tag{A.88}$$

希望你已经想到了方程的右侧应该是向量：

$$\begin{pmatrix}\frac{1}{-2}\end{pmatrix}\cdot B\begin{pmatrix}4\cdot x+-3\cdot y\\-2\cdot x+1\cdot y\end{pmatrix} \tag{A.89}$$

我们成功了！线性变换的逆变换

$$A\begin{pmatrix}x\\y\end{pmatrix}:=\begin{pmatrix}1\cdot x+3\cdot y\\2\cdot x+4\cdot y\end{pmatrix} \tag{A.90}$$

你可能还记得，它的几何效果是将第一象限中的单位正方形转变为行列式一节中描述的平行四边形，那么逆变换是

$$\begin{pmatrix}\frac{1}{-2}\end{pmatrix}\cdot B\begin{pmatrix}x\\y\end{pmatrix}:=\begin{pmatrix}\frac{1}{-2}\end{pmatrix}\begin{pmatrix}4\cdot x+-3\cdot y\\-2\cdot x+1\cdot y\end{pmatrix} \tag{A.91}$$

这很棒，但是这里是否有一个模式？我们能否从这个例子中推导出一般的技巧？也许你注意到了我们在行列式的讨论中提到的这个矩阵。也许你甚至觉得前面的讨论中出现的分数 $\frac{1}{2}$ 前面的 -2 与之有关，因为 -2 就是矩阵 B 的行列式！你是正确的！

一般而言，对于一个 2×2 的矩阵

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{A.92}$$

它的逆矩阵是

$$A^1 := \frac{1}{\mathsf{det}(A)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{A.93}$$

定义
伴随矩阵
矩阵

$$\begin{pmatrix} d & -b \\ c & a \end{pmatrix} \tag{A.94}$$

被称为矩阵 A 的伴随矩阵。

一般而言，对于给定的矩阵 A，我们用 $adj(A)$ 来表示其伴随矩阵。我们上面所发现的可以表述为

$$\begin{aligned} A \cdot (\mathsf{det}(A)^1 \cdot \mathsf{adj}(A)) &= \mathsf{det}(A)^1 \cdot (A \cdot \mathsf{adj}(A)) && \text{(A.95)} \\ &= \mathsf{det}(A)^1 \cdot (\mathsf{det}(A) \cdot I) && \text{(A.96)} \\ &= (\mathsf{det}(A)^1 \cdot \mathsf{det}(A)) \cdot I && \text{(A.97)} \\ &= (1 \cdot \mathsf{det}(A)) \cdot I && \text{(A.98)} \\ &= \mathsf{det}(A) \cdot I && \text{(A.99)} \\ &= 1 \cdot I && \text{(A.100)} \\ &= I && \text{(A.101)} \end{aligned}$$

因此，矩阵 A 的逆矩阵就是由 $\mathsf{det}(A)^1 \cdot \mathsf{adj}(A)$ 描述的矩阵。这让我们意识到我们需要找到一种确定伴随矩阵的方法。令人惊讶的是，伴随矩阵等于被称为矩阵 A 的余子式矩阵的转置。

那么，矩阵的余子式矩阵是什么？我们将给出定义，然后邀请你验证我们上面构建的伴随矩阵确实满足这个定义。

矩阵 A 的余子式矩阵是矩阵 C，其元素 $C_{i,j}$ 是通过删除 A 的第 i 行和第 j 列而形成的 $\square n - 1\square \times \square n - 1\square$ 子矩阵的行列式。我们称 $C_{i,j}$ 为矩阵 A 的第 i, j 个余子式。我们将通过一个例子来进行演示。考虑矩阵

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \tag{A.102}$$

我们计算第 1 行，第 1 列的余子式 $C_{1,1}$。删除 A 的第 1 行和第 1 列后，我们得到较小的 2×2 矩阵

$$\begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix} \tag{A.103}$$

它的行列式就是我们要求的余子式 $C_{1,1}$

$$C_{1,1} := \begin{vmatrix} 5 & 6 \\ 8 & 9 \end{vmatrix} = 5 \cdot 9 - 8 \cdot 6 = 45 - 48 = -3 \tag{A.104}$$

练习

计算余子式 $C_{1,2}$ 和 $C_{1,3}$。

如果你尝试了上面的练习，你会发现

$$C_{1,2} := \begin{vmatrix} 4 & 6 \\ 7 & 9 \end{vmatrix} = 4 \cdot 9 - 7 \cdot 6 = 36 - 42 = -6 \tag{A.105}$$

和

$$C_{1,3} := \begin{vmatrix} 4 & 5 \\ 7 & 8 \end{vmatrix} = 4 \cdot 8 - 5 \cdot 7 = 32 - 35 = -3 \tag{A.106}$$

练习

计算剩下的 6 个余子式 $C_{2,1}, C_{2,2}, C_{2,3}, C_{3,1}, C_{3,2}, C_{3,3}$，然后构建余子式矩阵 C。进一步，对这个矩阵进行转置。然而，如果你试图创建该矩阵的逆矩阵，你将遇到问题！

如果你尝试构建这个矩阵的逆矩阵，你将遇到一个问题，正如上面的练习所述。一切进行得很顺利，直到你计算行列式时——它为 0！

有几种方法可以判断矩阵 A 的行列式为 0，并通过观察它们可以将一些思想联系起来。

首先，矩阵 A 的第一列

$$\begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} \tag{A.107}$$

与其他两列线性相关。

练习

你能找到它们之间的依赖关系吗?

通过一些计算，你会发现第二列的两倍减去第三列的一倍等于第一列:

$$\begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix} = 2 \cdot \begin{pmatrix} 2 \\ 5 \\ 8 \end{pmatrix} + (-1) \cdot \begin{pmatrix} 3 \\ 6 \\ 9 \end{pmatrix} \tag{A.108}$$

如果我们应用之前关于线性变换行列式为 0 的等价表达的定理，我们可以看到第一列对其他两列的线性依赖等价于矩阵 A 的行列式为 0。

现在，我们将进行一些计算，它将导致一个非常有趣的观察，将矩阵的逆矩阵、其行列式和我们上面描述的余子式矩阵联系起来。它起初可能看起来很不寻常，但如果你按照每个等式进行计算，你会看到它们之间的联系！

$$\det(A) = 0 \tag{A.109}$$
$$= -3 - 2 \cdot (-6) - 3 \cdot 3 \tag{A.110}$$
$$= 1 \cdot C_{1,1} - 2 \cdot C_{1,2} + 3 \cdot C_{1,3} \tag{A.111}$$
$$= (-1)^{1+1} \cdot 1 \cdot C_{1,1} + (-1)^{1+2} \cdot 2 \cdot C_{1,2} + (-1)^{1+3} \cdot 3 \cdot C_{1,3} \tag{A.112}$$
$$= (-1)^{1+1} \cdot A_{1,1} \cdot C_{1,1} + (-1)^{1+2} \cdot A_{1,2} \cdot C_{1,2} + (-1)^{1+3} \cdot A_{1,3} \cdot C_{1,3} \tag{A.113}$$

C 表示代数余子式（cofactor），A 表示矩阵 A 的元素。

正如你所看到的，我们可以通过对该行上各个条目的系数与余子式的交替求和来计算行列式。

希望这个例子能让你相信这可能是正确的。为了更加确信，你可以尝试计算 A 沿着第二行的行列式。事实上，你可以尝试计算任意一行或一列上的行列式，并观察它们都得到相同的结果！

A.8. 特征向量和特征值

为了引出下一个主题，让我们来玩玩矩阵

$$A := \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \tag{A.114}$$

练习

首先，你能描述它所表示的空间变换吗?

你会意识到第一个基向量被拉伸了 2 倍，而第二个基向量被拉伸了 3 倍。因此，在这个变换下，这些向量并没有移动，只是被缩放了。你应该将这个例子与之前的例子进行对比

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \tag{A.115}$$

你能找到一个类似的一组向量，它们在与矩阵 B 对应的变换下只被拉伸吗？祝你好运——我们敢打赌你找不到！

在第一个例子中，标准基向量被称为矩阵 A 的特征向量。一般来说，非零向量 v 是线性变换 T 的特征向量，当且仅当存在 $\lambda \in \mathbb{F}$ 使得

$$Tv = \lambda v \tag{A.116}$$

换句话说，T 对 v 的作用只是将向量 v 按标量 λ 进行缩放。

特征向量这个名字从哪里来？德国人对代数的术语和符号有很大的影响，而“eigen”在德语中意思是“自己的”，或者可以理解为“特征的”。因此，特征向量是与矩阵 A 特征相关的向量。这意味着我们完全可以根据所有特征向量来确定矩阵 A。

在大部分情况下，这是正确的。我们之前已经知道线性变换完全由其在空间的一组基上的作用方式来描述。对于上述矩阵 A 的示例，我们非常幸运地从 A 的描述中立即知道它如何作用于标准基向量，因此我们已经基本了解了矩阵 A 的一切。

但是如果基向量不同呢？我们之前已经看到向量空间有多个基，所以在不同于标准基的基上，我们是否能够理解矩阵的行为呢？当然可以。以下是一个这样的情况的例子：

考虑矩阵

$$B = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \tag{A.117}$$

让我们将这个变换应用于向量 $v_1 := \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ 和 $v_2 := \begin{pmatrix} 1 \\ 1 \end{pmatrix}$:

$$Bv_1 = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} = 2 \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 2 \cdot v_1 \tag{A.118}$$

以及

$$Bv_2 = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 3 \end{pmatrix} = 3 \cdot \begin{pmatrix} 1 \\ 1 \end{pmatrix} = 3 \cdot v_1 \tag{A.119}$$

所以，$Bv_1 = 2 \cdot v_1$ 和 $Bv_2 = 3 \cdot v_2$。

现在，让我们对向量 $v := \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ 进行同样的操作。那么，我们有

$$Bv := B \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 3 \end{pmatrix} \tag{A.120}$$

请注意，如果存在一个数 λ 使得

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} = \lambda \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda \cdot 0 \\ \lambda \cdot 1 \end{pmatrix} = \begin{pmatrix} 0 \\ \lambda \end{pmatrix} \tag{A.121}$$

那么 $1 = 0$。因此，不存在这样的数 λ。这等价于说向量 v 与向量 v_1 和 v_2 不同，因为该变换不仅仅是通过某个因子对 v 进行拉伸。

向量 v_1 和 v_2 是矩阵 B 的特征向量，而 v 不是。我们将缩放因子 2 和 3 分别称为特征向量 v_1 和 v_2 的特征值。一般来说，与特征向量 v 相关联的特征值是之前定义的特征向量中出现的标量 λ:

$$T_v = \lambda v \tag{A.122}$$

总体而言，我们将一个变换的特征向量和特征值称为其特征信息。对于二维空间的算子，只能有两个特征向量，而对于 n 维空间的算子，一般可以有多达 n 个特征向量。

练习

找出矩阵

$$Proj_x := \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \tag{A.123}$$

和

$$M := \begin{pmatrix} 2 & 0 \\ -\frac{1}{2} & 3 \end{pmatrix} \tag{A.124}$$

的特征信息。对于 $Proj_x$，找到特征信息应该不难，而对于 M，可能会有一些挑战。

A.9. 基变换

让我们继续使用我们在讨论特征向量时提到的矩阵 B。虽然标准基是很好的，但我们希望展示将基变换为 B 的特征向量可能有几个优势。好吧，我们应该先检查一下 B 的特征向量确实构成了向量空间 $\mathbb{R}^2$ 的基。

练习

验证特征向量

$$v_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, v_2 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \tag{A.125}$$

实际上构成了 $\mathbb{R}^2$ 的基。记住，这意味着我们必须检查 v_1 和 v_2 是线性无关的，并且 v_1 和 v_2 张成 $\mathbb{R}^2$。你应该先用蛮力方法进行检查。然后，思考一下你如何可以应用关于向量空间特定维度基的长度的定理。

我们知道由两个向量组成的标准基确实是 $\mathbb{R}^2$ 的基，因此该定理确保了 $\mathbb{R}^2$ 的任何基元素列表的长度也为 2。根据这一点以及特征向量要么线性无关要么张成的事实，该定理确保它们构成一个基。

我们之前知道描述线性变换的矩阵取决于我们为空间选择的基。让我们将使用的基从标准基变换为特征向量的基，看看矩阵 B 因此而发生的变化！

要执行此操作，我们需要知道与矩阵 B 对应的变换 T 在特征向量 v_1 和 v_2 上的作用，只用 v_1 和 v_2 来表示，但我们已经弄清楚了！回想一下 $Bv_1 = 2 \cdot v_1 = 2 \cdot v_1 + 0 \cdot v_2$，以及 $Bv_2 = 3v_2 = 0 \cdot v_1 + 3 \cdot v_2$。

因此，我们可以根据之前的步骤使用特征向量 v_1 和 v_2 作为基来得到变换 T 的矩阵：

$$\left(\mathcal{M}(T : \mathbb{R}^2 \to \mathbb{R}^2)_{\mathcal{B}^2_{\mathbb{R}}, \mathcal{B}^2_{\mathbb{R}}}\right) = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \tag{A.126}$$

我们可以简写为 $\mathcal{M}(T)$，就像我们之前做过的那样。我们可以看到这个矩阵是对角矩阵；对角矩阵非常有用。为了理解原因，请尝试下面的练习。

练习

找到相对于特征向量 v_1 和 v_2（有时称为特征基）形成的矩阵 $\mathcal{M}(T) = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$ 的平方，计算

$$\mathcal{M}(T)^2 := \mathcal{M}(T) \cdot \mathcal{M}(T) \tag{A.127}$$

然后，计算

$$\mathcal{M}(T)^3 := \mathcal{M}(T) \cdot \mathcal{M}(T) \cdot \mathcal{M}(T) \tag{A.128}$$

接下来，计算

$$\mathcal{M}(T)^4 := \mathcal{M}(T) \cdot \mathcal{M}(T) \cdot \mathcal{M}(T) \cdot \mathcal{M}(T) \tag{A.129}$$

发生了什么？你能计算出 $\mathcal{M}(T)^{100}$ 吗？

练习

现在，尝试对于相对于标准基向量的原始矩阵 $B := \begin{pmatrix} 2 & 1 \\ 0 & 3 \end{pmatrix}$ 进行类似的操作。也就是，计算 B 的平方。然后，计算 B^3。接下来，计算 B^4。你能计算出 B^{100} 吗？

我们愿意打赌，你在处理 M(T) 的练习时找到了方法，但在处理 B 的练习时放弃了——任何人都会这样。M(T) 和 B 之间的区别在于 M(T) 是对角矩阵，而 B 不是。B 无法成为对角矩阵导致在乘法时遇到了相当大的困难，正如你在练习中学到的那样。

因此，问题变成了：我们是否总能将基变换为一个使得我们起始的矩阵成为对角矩阵的基呢？实际上，并不总是这样。

练习

思考一下为什么我们无法找到一个由矩阵

$$T := \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \tag{A.130}$$

描述的变换的特征向量组成的基。具有这种特殊性质的矩阵被称为对角化矩阵。对于能否对角化的矩阵给出一个明确的刻画并不简单，所以我们暂时不深入讨论这个问题。

在这里，我们将量子力学和特征值联系起来。具有实数特征值的线性算符构成了一个特殊类别的算符。在我们之前的一章中回顾了量子力学的测量假设；该假设指出与可测物理属性相关的任何算符都将是共轭算符。我们还没有看到共轭算符，但我们将发现它们是一类特殊的算符，其特征值总是实数，因此可测量。待会我们会看到共轭算符是什么以及为什么它们具有如此显著的性质！

A.10. 特征向量和特征值

A.10.1. 进一步研究内积

进一步研究之前定义的内积揭示了一些显著的特性。其中之一是共轭对称性，它表明对于所有向量 u, v，$\langle u, v\rangle = \overline{\langle v, u\rangle}$。因此，交换向量不会得到相同的结果，而是得到其共轭。乍一看，这可能有些奇怪，但通过一个例子可以帮助我们相信这是正确的。

设 $u := \begin{pmatrix} i \\ 1 \end{pmatrix}$ 和 $v := \begin{pmatrix} 2 \\ i \end{pmatrix}$。那么，

$$\langle u, v\rangle = \left\langle \begin{pmatrix} i \\ 1 \end{pmatrix}, \begin{pmatrix} 2 \\ i \end{pmatrix} \right\rangle = \overline{i} \cdot 2 + \overline{1} \cdot i = (-i) \cdot 2 + 1 \cdot i = -2i + i = -i \tag{A.131}$$

和

$$\langle v, u\rangle = \left\langle \begin{pmatrix} 2 \\ i \end{pmatrix}, \begin{pmatrix} i \\ 1 \end{pmatrix} \right\rangle = \overline{2} \cdot i + \overline{i} \cdot 1 = 2 \cdot i + (-i) \cdot 1 = 2i - i = i \tag{A.132}$$

首先，我们有向量：

$$u = \begin{pmatrix} i\ 1 \end{pmatrix}\ v \qquad\qquad = \begin{pmatrix} 2\ i \end{pmatrix} \tag{A.133}$$

然后，我们计算内积 $\langle u, v\rangle$：

$$\langle u, v\rangle = \begin{pmatrix} \overline{i} & \overline{1} \end{pmatrix} \begin{pmatrix} 2 \\ i \end{pmatrix} = \overline{i} \cdot 2 + \overline{1} \cdot i \tag{A.134}$$

$$= (-i) \cdot 2 + 1 \cdot i \tag{A.135}$$

$$= -2i + i \tag{A.136}$$

$$= -i \tag{A.137}$$

因此，我们得到 $\langle u, v\rangle = -i$。

接下来，我们计算内积 $\langle v, u\rangle$：

$$\langle v, u\rangle = \begin{pmatrix} \overline{2} & \overline{i} \end{pmatrix} \begin{pmatrix} i \\ 1 \end{pmatrix} \tag{A.138}$$

$$= \overline{2} \cdot i + \overline{i} \cdot 1 \tag{A.139}$$

$$= 2 \cdot i + (-i) \cdot 1 \tag{A.140}$$

$$= 2i - i \tag{A.141}$$

$$= i \tag{A.142}$$

因此，我们得到 $\langle v, u\rangle = i$。

根据计算结果，您的推断是正确的。

Exercise

Let $u := \begin{pmatrix} i \\ 2 \end{pmatrix}$ and $v := \begin{pmatrix} 1 \\ i \end{pmatrix}$. Find the inner product $\langle u, v\rangle$. Now, find the inner product $\langle v, u\rangle$. Do you see why we have to conjugate?

练习

设 $u := \begin{pmatrix} i \\ 2 \end{pmatrix}$ 和 $v := \begin{pmatrix} 1 \\ i \end{pmatrix}$。求内积 $\langle u, v\rangle$。然后，求内积 $\langle v, u\rangle$。你是否明白为什么我们需要取共轭?

[3]

共轭对称性还保证了任何向量与其自身的内积是一个实数。为了看到这一点，观察任意向量 v，共轭对称性确保 $\langle v, v\rangle = \overline{\langle v, v\rangle}$（我们将 v 与自身交换!），因此 v 与自身的内积需要等于其共轭值。复数 $a + bi$ 等于共轭值 $a - bi$ 当且仅当 $a - bi$ 与 $a + bi$ 相等，这发生当且仅当 $b = -b$。但是只有当 $b = 0$ 时，$b = -b$ 才成立，也就是说，我们的复数 $a + bi$ 一直都是实数。

[3]内积（Inner product）是线性代数中的一个概念，用于定义向量空间中的向量之间的乘法运算。内积可以用来衡量向量之间的夹角、长度、相似性等性质。

在欧几里得空间中，内积通常被定义为两个向量的点积（Dot product），即将两个向量的对应分量相乘，并将乘积相加得到的结果。对于实数向量空间中的两个向量 **u** 和 **v**，内积可以表示为：

$$\langle \mathbf{u}, \mathbf{v}\rangle = \sum_{i=1}^{n} u_i v_i \tag{A.143}$$

其中，u_i 和 v_i 分别是向量 **u** 和 **v** 的第 i 个分量。

内积具有以下一些重要的性质：- 对称性：$\langle \mathbf{u}, \mathbf{v}\rangle = \langle \mathbf{v}, \mathbf{u}\rangle$ - 线性性质：$\langle \alpha\mathbf{u} + \beta\mathbf{v}, \mathbf{w}\rangle = \alpha\langle \mathbf{u}, \mathbf{w}\rangle + \beta\langle \mathbf{v}, \mathbf{w}\rangle$ - 正定性（正定内积空间）：对于非零向量 **u**，有 $\langle \mathbf{u}, \mathbf{u}\rangle > 0$

内积可以用来定义向量的长度（模）和向量之间的夹角（正交性），并且在许多数学和物理领域中都有广泛应用，如几何、信号处理、机器学习等。

内积也在第一个参数上具有线性性质。因此，对于任意两个向量 u, v 和 w，内积满足以下等式：

$$\langle u+v, w\rangle = \langle u, w\rangle + \langle u+v, w\rangle \tag{A.144}$$

让我们看一个这种现象的例子。设

$$\langle u+v, w\rangle = \left\langle \begin{pmatrix}1\\0\end{pmatrix} + \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix} \right\rangle \tag{A.145}$$

$$= \left\langle \begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix} \right\rangle \tag{A.146}$$

$$= \overline{1}\cdot 1 + \overline{1}\cdot 2 \tag{A.147}$$

$$= 1\cdot 1 + 1\cdot 2 \tag{A.148}$$

$$= 3 \tag{A.149}$$

练习

现在，你计算

$$\langle u, w\rangle + \langle v, w\rangle = \left\langle \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix} \right\rangle + \left\langle \begin{pmatrix}0\\1\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix} \right\rangle \tag{A.150}$$

并验证它等于 $1+2=3$。

下面是对您提供的方程进行详细计算的过程：

首先，我们有向量：

$$u = \begin{pmatrix}1\\0\end{pmatrix} \tag{A.151}$$

$$v = \begin{pmatrix}0\\1\end{pmatrix} \tag{A.152}$$

$$w = \begin{pmatrix}1\\2\end{pmatrix} \tag{A.153}$$

然后，我们计算 $\langle u, w\rangle$：

$$\langle u, w\rangle = \left\langle \begin{pmatrix}1\\0\end{pmatrix}, \begin{pmatrix}1\\2\end{pmatrix} \right\rangle \tag{A.154}$$

$$= \overline{1}\cdot 1 + \overline{0}\cdot 2 \tag{A.155}$$

$$= 1\cdot 1 + 0\cdot 2 \tag{A.156}$$

$$= 1 \tag{A.157}$$

接下来，我们计算 $\langle v, w\rangle$：

$$\langle v, w\rangle = \left\langle \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \end{pmatrix} \right\rangle \tag{A.158}$$

$$= \overline{0} \cdot 1 + \overline{1} \cdot 2 \tag{A.159}$$

$$= 0 \cdot 1 + 1 \cdot 2 \tag{A.160}$$

$$= 2 \tag{A.161}$$

因此，我们有：

$$\langle u, w\rangle + \langle v, w\rangle = 1 + 2 = 3 \tag{A.162}$$

根据计算结果，我们得到 $\langle u, w\rangle + \langle v, w\rangle = 3$。

我们说过内积在第一个参数上是线性的，所以你可能想知道标量乘法在这里是如何体现的。第一个参数的线性性的第二部分是对于任意标量 a 和任意向量 u 和 v，我们有

$$\langle a \cdot u, v\rangle = a \cdot \langle u, v\rangle \tag{A.163}$$

为了验证这个性质，让

$$a = 2, u = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, v = \begin{pmatrix} 3 \\ 4 \end{pmatrix} \tag{A.164}$$

我们计算左边的表达式

$$\langle a \cdot u, v\rangle = \left\langle 2 \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 3 \\ 4 \end{pmatrix} \right\rangle \tag{A.165}$$

$$= \left\langle \begin{pmatrix} 2 \\ 4 \end{pmatrix}, \begin{pmatrix} 3 \\ 4 \end{pmatrix} \right\rangle \tag{A.166}$$

$$= 2 \cdot 3 + 4 \cdot 4 \tag{A.167}$$

$$= 6 + 16 \tag{A.168}$$

$$= 22 \tag{A.169}$$

练习

计算右边的表达式，即 $a \cdot \langle u, v\rangle$，并验证它等于 22。

你可能会想知道是否对于所有的标量 a 和向量 u、v，都有 $\langle u, a \cdot v\rangle$。通过下面的练习来验证这一点。

练习：通过找到一个标量 a 和两个向量 u 和 v，使得 $\langle u, a \cdot v\rangle \neq a\langle u, v\rangle$，证明这不是一个普遍成立的性质。实际上，你可以使用稍后列出的公理证明对于所有的标量 a 和向量 u、v，成立 $\langle u, a \cdot v\rangle = \overline{a}\langle u, v\rangle$。这个内积的性质有时被称为第二个参数的共轭齐次性。

内积的最后一个性质被称为正定性：对于所有的向量 v，$\langle v, v\rangle \geq 0$，并且 $\langle v, v\rangle = 0$ 当且仅当 $v = 0$。

让我们从更抽象的角度来看为什么这是正确的。设 $v = (v_1, v_2, \ldots, v_n)^T$ 是一个向量。那么

$$\langle v, v\rangle := \overline{v_1} \cdot v_1 + \overline{v_2} \cdot v_2 + \cdots + \overline{v_n} \cdot v_n \tag{A.170}$$

回想一下，复数与其共轭的乘积总是一个非负实数（即大于或等于 0），所以对于每个 $i \in \{1, \ldots, n\}$，$\overline{v_i} \cdot v_i$ 是一个实数。好吧，那么一堆（有限个）非负实数的和还是一个实数，所以和

$$\overline{v_1} \cdot v_1 + \overline{v_2} \cdot v_2 + \cdots + \overline{v_n} \cdot v_n \tag{A.171}$$

仍然是一个非负实数，即大于或等于 0。

练习

使自己相信上面的表达式等于 0 当且仅当向量 v 是零向量，即所有分量都为零。

因此，我们已经证明了对于任意向量 v，$\langle v, v\rangle \geq 0$，并且当且仅当 $v = 0$ 时，$\langle v, v\rangle = 0$。

练习

请注意，我们从未说过内积满足第二个参数的线性性！然而，这是正确的，你的任务是使用我们已经列出并验证过的公理来证明它。对第一个参数的线性性将对你的证明至关重要。

作为内积的克朗克三角函数

在前面讨论向量空间的正交归一基时，我们提到了用 Kronecker delta 函数 δ 来描述正交归一基的数学定义。Kronecker delta 函数有一个如此简单的描述，以至于你无法想象为什么它会被命名为某人的名字——尽管 Kronecker 本人并不介意。

定义

Kronecker delta 函数对于集合 $\{1, 2, \ldots n\}$

$$\begin{cases} \delta(i, j) = 1 & \text{如果 } i = j \\ \delta(i, j) = 0 & \text{如果 } i \neq j \end{cases} \tag{A.172}$$

这看起来很熟悉！回想一下我们之前对正交归一基向量的定义。因此，我们可以将正交归一集合 S 称为一个向量集合，其中限制在 S 上的内积是 Kronecker delta 函数！

练习

检验恒等矩阵的元素可以用克罗内克 δ 函数来描述。更准确地说，我们要检验我们可以逐个元素地定义恒等矩阵 I，即 $I_{ij} := \delta(i, j)$。

A.10.2. 厄米算子（Hermitian Operators）

一个线性算子（也可互换地称为矩阵）如果等于其共轭转置（如上所定义），则被称为 Hermitian。Hermitian 算子在量子力学中非常重要。我们经常希望测量与某个状态相关的某个量。通常我们想要的量是算子的特征值，因此我们希望该特征值是实数，以便有效地进行测量。

为什么我们不能使用复数进行测量值得一提的一个微妙的问题是，我们不能使用复数进行测量。假设你想弄清楚 0 和 i 中哪个更大？那么，你要么决定 $i = 0$，$0 < i$，要么决定 $i < 0$。当然，我们不会选择 $i = 0$，因为如果我们这样做，我们可以将方程 $i = 0$ 的每一边乘以 $-i$，得出 $1 = 0$! 让我们尝试选择 $0 < i$。

如果我们选择 $0 < i$，那么在每一边乘以 i 的情况下得到不等式

$$i \cdot 0 < i \cdot i \tag{A.173}$$

%endequation

但是

$$0 < i \cdot i = i^2 = -1 \tag{A.174}$$

总结一下，我们得出了一个结论：根据我们假设的前提，即 $0 < i$，我们推导出了 $0 < 1$。由于 $0 < 1$ 是没有意义的，我们得出结论：我们最初的假设 $0 < i$ 是没有意义的。

那么，如果我们选择 $i < 0$ 呢？从两边同时减去 i，得到

$$i - i < 0 - i \tag{A.175}$$

因此，

$$0 < -i \tag{A.176}$$

将两边乘以 $-i$，得到

$$0 \cdot (-i) < -i \cdot (-i) \tag{A.177}$$

但是

$$-i \cdot (-i) = (-i) \cdot i \cdot (-1) \cdot i \tag{A.178}$$

$$(-1) \cdot (-1) \cdot i \cdot i = 1 \cdot i^2 = -1 \tag{A.179}$$

因此，不等式 $0 \cdot (-1) < -i \cdot (-1)$ 实际上意味着 $0 < -1$。这也是没有意义的，因此我们必须得出结论：最初的假设 $i < 0$ 也是没有意义的。因此，我们发现无论是选择哪种顺序，即 $0 < i$ 还是 $i < 0$，都是没有意义的。假设任何一种顺序都会导致 $0 < -1$，这显然是错误的。因此，我们得出结论：没有一种可接受的方式来对复数进行排序。

为了更精确地说明，假设你想找到一个对复数进行排序的方式，满足我们对实数的常规经验中熟悉的以下合理条件：

- **三分性**：对于所有复数 x, y，我们有三分性：要么 $x < y$，要么 $x > y$，要么 $x = y$。
- **加法性质**：对于所有复数 x, y, z，如果 $x < y$，那么 $x + z < y + z$。
- **乘法性质**：对于所有复数 x, y, z，如果 $0 < z$，那么 $x < y$ 意味着 $xz < yz$。

现在，让我们非常谨慎地证明选择 $0 < i$ 至少与上述公理中的一个矛盾。由于 $0 < i$，根据上述第三条（乘法性质），我们可以看到

$$0 < i \rightarrow 0 \cdot i < i \cdot i = i^2 = -1 \tag{A.180}$$

尽管我们从假设 $0 < i$ 推导出 $0 < 1$ 这个事实令人不安，但从技术上讲，它并不违反上述任何一个公理。根据上述第二条（加法性质），不等式 $0 < -1$ 让我们得出

$$0 < -1 \rightarrow 0 + i < -1 + 1 = 0 \tag{A.181}$$

所以，$1 < 0$。这也令人不安，但它并不直接违反上述任何一个公理。现在，我们最后一次使用第三条（乘法性质），得到

$$1 < 0 \rightarrow 1 + i < 0 \cdot i = 0 \tag{A.182}$$

然后，我们得到 $i < 0$，但我们最初的假设是 $0 < i$。这违反了上述第一条（三分性）的公理！

练习

试着弄清楚以类似的方式定义 $i < 0$ 会出现什么问题。

因此，我们确实希望这些测量结果是实数，否则我们无法理解它们。那么问题就是：为什么厄米矩阵具有实特征值?

厄米算符具有实特征值

尽管我们可以轻松验证一个矩阵是否等于其共轭转置，但通过不同但等价的厄米定义，我们可以简单地证明厄米矩阵具有实特征值。我们还可以定义一个线性算符 $T : V \rightarrow V$ 为厄米算符，如果它是自伴的，我们称之为：

定义

矩阵的共轭转置

矩阵 T 的共轭转置是一个矩阵 T^*，满足对于所有的向量 $u, v \in V$，有 $\langle T_u, v\rangle = \langle u, T^* v\rangle$。

因此，当 $T = T^*$ 时，我们说 T 是自伴的，对于所有的向量 u, v，我们有 $\langle T_u, v\rangle = \langle u, Tv\rangle$。注意，算符的共轭转置就是它的共轭转置！因此，说一个算符是自伴的就是说它等于它的共轭转置。

共轭转置就是共轭转置

线性算符的共轭转置就是它的共轭转置。

现在，如果 v 是厄米矩阵 T 的特征向量，对应的特征值为 λ，我们还可以进一步得出结论：

$$\langle T_v, v\rangle = \langle v, T_v\rangle \tag{A.183}$$

这里，v 是 T 的特征向量，对应的特征值为 λ，意味着 $T_v = \lambda v$。因此，我们可以将 λv 代替 T_v，得到方程 $\langle T_v, v\rangle = \langle v, T_v\rangle$：

$$\langle \lambda v, v\rangle = \langle v, \lambda v\rangle \tag{A.184}$$

现在我们的直觉是要因式分解出 λ，这会导致一些有趣的事情发生。由于内积在第一个参数上的齐次性以及在第二个参数上的共轭齐次性，”因式分解出 λ” 得到了新的方程：

$$\lambda\langle v, v\rangle = \overline{\lambda}\langle v, v\rangle \tag{A.185}$$

现在，我们希望在两边”除以 $\langle v, v\rangle$” 并结束，但我们怎么知道它不是零呢？嗯，我们开始假设 v 是 T 的特征向量，而特征向量的定义要求 v 不为零。然后，我们应用内积的正定性，确保由于 v 不为零，v 与自身的内积 $\langle v, v\rangle$ 也不为零。因此，

$$\lambda = \overline{\lambda} \tag{A.186}$$

还记得复数等于其共轭的意义吗？没错，那么它实际上是一个实数！

值得注意的是，任何实对称矩阵（只有实数元素的对称矩阵）都是 Hermitian 的。你能看出为什么吗?

希望这次对定理证明的探索使你意识到我们在这些定义中要求的所有公理都是真正必要的！你能看出我们在证明中使用了每一个公理吗？

A.11. 单元运算符

在第 3 章中，你肯定遇到了描述量子逻辑门的矩阵的集合，并在第 14 章中列出了它们。这些矩阵很特殊，因为它们是幺正的。幺正矩阵保持向量的长度不变。它们得名于这个性质，即幺正矩阵作用后，单位向量仍然是单位向量。

更准确地说，

定义

幺正算子

对于所有向量 $x, y \in V$，线性算子 $U : V \to V$ 是幺正的，如果

$$\langle x, y\rangle = \langle Ux, Uy\rangle \tag{A.187}$$

即，U 保持内积。

有趣的是，我们也可以等价地定义一个幺正算子（矩阵）$U : V \to V$ 为一个其共轭转置等于其逆的矩阵，即

$$U^\dagger = U^{-1} \tag{A.188}$$

因此，

$$U^\dagger U = UU^\dagger = I \tag{A.189}$$

到目前为止，我们所说的并没有直接暗示幺正算子会保持其作用对象的长度不变，因此让我们在一个练习中验证一下这一点。

练习

通过使用上述给定的定义，验证幺正算子 U 保持其作用对象 v 的长度不变。利用向量 v 的长度等于 $\sqrt{\langle v, v\rangle}$ 这一事实，将原始长度 $\sqrt{\langle v, v\rangle}$ 与 U 作用后 v 的长度 $\sqrt{\langle Uv, Uv\rangle}$ 进行比较。

A.12. 直和与张量积

我们可能希望从我们已经拥有的向量空间集合构建更大的向量空间。有两种常用的方法。一种是直和，另一种是张量积。我们已经在以前的上下文中看到了张量积的概念，例如两个向量或两个矩阵的张量积。之前定义的操作与我们将要描述的操作之间存在关联。

直和

我们首先要强调的一点是，直和操作作用于向量空间，而不是数字、向量或矩阵（尽管敏锐的读者已经认为所有这些只是线性变换！）。张量积也是如此。具体来说，直和是一种二元操作，接受两个向量空间作为输入，并产生另一个通常更“大”的向量空间作为输出。让我们来看一个例子。

考虑一维向量空间 $\mathbb{R}$ 和它自身的一份副本——也就是两份 $\mathbb{R}$。

$\mathbb{R}$ 和它自身的直和被写作

$$\mathbb{R} \oplus \mathbb{R} \tag{A.190}$$

它的定义是

$$\mathbb{R} \oplus \mathbb{R} := \left\{ \begin{pmatrix} x \\ y \end{pmatrix} : x \in \mathbb{R}, y \in \mathbb{R} \right\} \tag{A.191}$$

稍等 - 那只是 $\mathbb{R} \times \mathbb{R}$ 的笛卡尔积而已!

练习回想一下笛卡尔积 $\mathbb{R} \times \mathbb{R}$ 的定义，并将其与直和 $\mathbb{R} \oplus \mathbb{R}$ 进行比较。

为什么我们对同一物体有两个不同的名称? 嗯，它们实际上不是完全相同的，因为直和 $\mathbb{R} \oplus \mathbb{R}$ 具有更多的结构。特别地，直和 $\mathbb{R} \oplus \mathbb{R}$ 是一个（在 $\mathbb{R}$ 上的）向量空间，而笛卡尔积 $\mathbb{R} \times \mathbb{R}$ 只是一个集合。

为了进一步阐述这个概念，考虑集合 $A = \{a, b, c\}$ 和 $B = \{d, e\}$。我们可以创建它们的笛卡尔积

$$A \oplus B := \{(x, y) : x \in A, y \in B\} = \{(a, d), (a, e), (b, d), (b, e), (c, d), (c, e)\} \tag{A.192}$$

现在，我们提出一个问题：直和 $A \oplus B$ 是什么? 根据上面的定义，我们首先有

$$A \oplus B := \left\{ \begin{pmatrix} x \\ y \end{pmatrix} : x \in A, y \in B \right\} \tag{A.193}$$

一个粗略的观察可能让我们相信问题在于笛卡尔积使用了行向量，而直和使用了列向量，但在讨论集合时，我们使用行向量或列向量并没有关系。问题得到进一步阐明：$A \oplus B$ 如何成为一个向量空间? 以及在哪个域上?

你看，为了将 $A \oplus B$ 定义为一个向量空间，你需要赋予它额外的代数结构，使其成为一个阿贝尔群，并且具有由某个指定域定义的场作用。目前，它两者都没有。你可以花一些时间思考一下。你会如何定义 $A \oplus B$ 中两个元素的加法运算?

无论你如何决定，都必须允许特定元素 $(ad)^T$ 和 $(be)^T$ 的加法。当然，我们可以说

$$\begin{pmatrix} a \\ e \end{pmatrix} + \begin{pmatrix} b \\ e \end{pmatrix} = \begin{pmatrix} a + b \\ d + e \end{pmatrix} \tag{A.194}$$

但是 $a + b$ 和 $d + e$ 的含义是什么呢? 请记住，a, b, d 和 e 只是字母，它们没有任何数值意义。因此，重要的观点是，可以合理地形成任意两个集合的笛卡尔积。例如，集合

$$\mathbb{COLORS} := \{\text{紫红色、洋红色、长春花色}\} \tag{A.195}$$

和集合

$$\mathbb{ANMALS} := \{\text{猫，狗，土拨鼠}\} \tag{A.196}$$

的笛卡尔积是

$$\left\{ \begin{array}{c} \text{(紫红色, 猫),(紫红色, 狗),(紫红色土拨鼠),} \\ \text{(洋红色, 猫), (洋红色, 狗), (洋红色土拨鼠),} \\ \text{(长春花色猫), (长春花色, 狗), (长春花色, 土拨鼠)} \end{array} \right\} \tag{A.197}$$

这只是一个集合，并不是一个向量空间。而直和 $\mathbb{R} \oplus \mathbb{R}$，作为先验的定义，是集合

$$\mathbb{R} \oplus \mathbb{R} := \left\{ \begin{pmatrix} x \\ y \end{pmatrix} : x \in \mathbb{R}, y \in \mathbb{R} \right\} \tag{A.198}$$

在域 $\mathbb{R}$ 上的一个向量空间，因为我们可以将加法（得到阿贝尔群结构）定义为通常的向量加法，将场作用定义为通常的标量乘法。

我们还想提到，由于两个向量空间的直和本身就是一个向量空间，所以它有一个维数。事实证明，两个向量空间的直和的维数是它们的维数之和，因此得名直和。更具体地说，对于任何两个在同一域 $\mathbb{F}$ 上的向量空间 V 和 W，

$$dim(V \oplus W) = dim(V) + dim(W) \tag{A.199}$$

即 V 和 W 的直和的维数等于 V 的维数加上 W 的维数。

定义

两个向量空间的直和[4]

向量空间 V 和 W 的直和是向量空间

$$V \oplus W := \left\{ \begin{pmatrix} v \\ w \end{pmatrix} \right\} \tag{A.200}$$

即由所有形如 $\begin{pmatrix} v \\ w \end{pmatrix}$ 的向量所构成的集合，这里 v 来自 V，w 来自 W。

A.12.1. 直和的维度

$$dim(V \oplus W) = dim(V) + dim(W) \tag{A.201}$$

张量积

现在，我们来讨论张量积。两个向量空间的张量积的特别之处在于，它完全由每个空间的基元素的“张量积”来描述。这意味着，如果我们有两个在同一域 F 上的向量空间 V 和 W，且其基分别为 $\mathcal{B}(V) = \{v_1, \ldots, v_m\}$ 和 $\mathcal{B}(W) = \{w_1, \ldots, w_m\}$，那么 V 和 W 的张量积，记作 $V \otimes W$，就是一个以以下形式给出的基的向量空间：

$$\mathcal{B}_{v \otimes w} := \{v_1 \otimes w_1, v_1 \otimes w_2, \ldots, v_1 \otimes w_n, \ldots v_m \otimes w_1, v_m \otimes w_2, \ldots, \otimes w_m \otimes w_n\} \tag{A.202}$$

你可能会注意到，所有可能的 V 的基向量与 W 的基向量的张量积的配对都出现在 V 和 W 的基的“张量积”中。实际上，这总是这样的！也就是说，如果我们有两个在同一域 $\mathbb{F}$ 上的向量空间 V

[4]向量空间的直和（Direct sum）是向量空间的一种构造，用来将两个或多个向量空间结合成一个更大的向量空间。

更具体地说，设有两个向量空间 V 和 W，它们的直和 $V \oplus W$ 是由所有形如 (v, w) 的有序对构成的集合，其中 $v \in V$，$w \in W$。在这个新的向量空间中，定义加法和标量乘法如下：

加法：对于任意两个元素 (v_1, w_1)，$(v_2, w_2) \in V \oplus W$，我们定义他们的和为：$(v_1, w_1) + (v_2, w_2) = (v_1 + v_2, w_1 + w_2)$。标量乘法：对于任意元素 $(v, w) \in V \oplus W$ 和任意标量 c，我们定义标量乘法为：$c \cdot (v, w) = (c \cdot v, c \cdot w)$。这样定义的 $V \oplus W$ 就满足向量空间的所有性质。而且，如果 V 和 W 的维数分别是 n 和 m，那么 $V \oplus W$ 的维数就是 $n + m$。这就是“直和”名字的来源，因为新的向量空间的维数是原来的向量空间的维数之“和”。

和 W，我们想找到它们的张量积 $V \otimes W$，我们至少可以通过张量化所有可能的 V 和 W 的基向量的配对来确定 $V \otimes W$ 的基。尽管这是一个方便的事实，但是它需要一些工作来证明；它通常被称为张量积基定理，强调它是一个待证的结果。

我们需要展示的是，实际上，由所有可能的 V 和 W 的基向量的张量积配对组成的 $V \otimes W$ 的提出的基实际上是一个基，即，所有这些张量积都是线性独立的，并且生成 $V \otimes W$。这是可以相信的，因为每个 V 和 W 的基向量本身就是线性独立的，并且它们生成各自的空间。然而，我们希望你思考为什么这需要一个证明！

不管怎样，我们将避免从公理的角度来定义两个向量的张量积，而是给出一个基中心的观点。如果你以前从未见过两个向量空间的张量积，那么你现在可以这么想：

给定两个具有

基 $\mathcal{B}_V$ 和 $\mathcal{B}_W$ 的向量空间 V 和 W，我们定义它们的张量积 $V \otimes W$ 为基等于它们基的“张量积”的向量空间，即，由所有可能的 V 和 W 的基元素的张量积配对组成的基 $\mathcal{B}_{V\otimes W}$，如上所述。

阅读这篇文章的数学家可能会感到困扰，但我们只是试图给出一个可行的定义。对于为什么可能对这种张量积的定义有任何困扰的读者，我们邀请他们从双线性映射，甚至更深入地，从范畴理论的角度，去研究更多关于张量积的数学处理！[5]

现在，我们来谈谈符号表示。你可能会遇到一些花哨的方式来表示同一个向量空间的多个副本的直和或张量积。例如，

$$\mathcal{H}^{\oplus n} \tag{A.203}$$

只是一种花哨的写法，

$$\underbrace{\mathcal{H} \oplus \cdots \oplus \mathcal{H}}_{n\square} \tag{A.204}$$

另一种表达方式是：向量空间 $\mathcal{H}$ 的 n 个副本的直和。这也可以写成

$$\underset{n}{\oplus}\mathcal{H} \tag{A.205}$$

对于张量积，有类似的表示法

$$\mathcal{H}^{\otimes n} := \underbrace{\mathcal{H} \otimes \cdots \otimes \mathcal{H}}_{n} \tag{A.206}$$

和

$$\underset{n}{\otimes}\mathcal{H} \tag{A.207}$$

这两种符号都指的是 $\mathcal{H}$ 的 n 重张量积，即，向量空间 $\mathcal{H}$ 的 n 个副本的张量积。这些表示几个 $\mathcal{H}$ 的张量积的符号，出现在表示量子寄存器，即，一组量子比特时，我们会在本章的高潮部分，即定义希尔伯特空间时看到。

我们也可以取两个运算符的张量积。把每个运算符想象成一个线性变换，因此是一个矩阵，让我们可以像本章前面那样取它们的张量积！例如，给定二维复空间 $\mathbb{C}^2$ 上的运算符，

[5] 范畴理论通过其普遍性质来定义两个向量空间的张量积。这个定义为张量积提供了一种优雅和精炼的视角，使得证明变得简单，并提供了关于张量积如何与其他向量空间交互的洞察。这种构造也指导了更一般的对象（如模）的张量积的定义！

$$I := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{A.208}$$

和

$$X := \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \tag{A.209}$$

我们可以形成他们的张量积

$$\begin{aligned}
I \otimes X :&= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \otimes X \\
&= \begin{pmatrix} 1 \cdot X & 0 \cdot X \\ 0 \cdot X & 1 \cdot X \end{pmatrix} \\
&= \begin{pmatrix} 1 \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & 0 \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \\ 0 \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & 1 \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix} \\
&= \begin{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix} \\
&= \begin{pmatrix} \begin{pmatrix} 1 \cdot 0 & 1 \cdot 1 \\ 1 \cdot 1 & 1 \cdot 0 \end{pmatrix} & \begin{pmatrix} 0 \cdot 0 & 0 \cdot 1 \\ 0 \cdot 1 & 0 \cdot 0 \end{pmatrix} \\ \begin{pmatrix} 0 \cdot 0 & 0 \cdot 1 \\ 0 \cdot 1 & 0 \cdot 0 \end{pmatrix} & \begin{pmatrix} 1 \cdot 0 & 1 \cdot 1 \\ 1 \cdot 1 & 1 \cdot 0 \end{pmatrix} \end{pmatrix} \\
&= \begin{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \\ \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} & \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{pmatrix} \\
&= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}
\end{aligned} \tag{A.210}$$

总结我们的计算结果，我们已经创建了一个现在在四维复空间 $\mathbb{C}^4$ 上操作的运算符，其效果如下：

$$
\begin{aligned}
(I \oplus X)(|00\rangle) &= (I \oplus X)(|0\rangle \oplus |0\rangle) \\
&= \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \\
&= \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} \\
&= |0\rangle \oplus |1\rangle \\
&= |01\rangle
\end{aligned}
\tag{A.211}
$$

找到张量积 $X \otimes I$。它是否与 $I \otimes X$ 相同?

注意，我们构造了一个 4×4 的矩阵作为两个 2×2 的矩阵的张量积，这符合我们早先关于一个 $(a \times b)$ 矩阵和 $(c \times d)$ 矩阵的张量积是 $(a \times c) \times (b \times d)$ 矩阵的讨论。

好奇的读者可能会尝试通过取二维运算符的张量积来构建第 3 章中讨论的量子运算符。但要小心：这不容易！实际上，你不能通过取两个二维运算符的张量积来构建 CNOT。

A.13. 希尔伯特空间

希尔伯特空间，是一种具有内积结构并且是完备的度量空间，这意味着它是由内积导出的度量引导的任何柯西序列都收敛于该空间中的一个点。希尔伯特空间在数学的许多分支中都有重要应用，特别是在量子力学中，它是理解量子状态和量子力学的运算符等概念的重要工具。在量子计算机的现代定义中，一个量子位被假设为由一个二维复希尔伯特空间来建模。在此，我们引入这一正式的定义。然而，在做这件事之前，我们需要简要讨论一些概念，包括向量空间上的度量，柯西序列，以及向量空间的完备性。这些基本概念对于深入理解和描述希尔伯特空间的性质和应用是必不可少的。

A.13.1. 度量，柯西序列和完备性

谈到向量空间上的度量，我们意味着我们可以通过计算两个向量 u 和 v 的差的范数来测量（因此，“度量”）两个向量之间的距离，即，

$$
\langle u - v, v - u \rangle \tag{A.212}
$$

练习

检查上述定义的两个向量 u 和 v 之间的距离是否等于零当且仅当 $u = v$，等价地，$u - v = 0$。

乍一看，柯西序列可以被认为是“一个序列的数，其项可以尽可能的接近，只要我们看得足够远。”在我们研究柯西序列之前，我们应该理解一下数列收敛的概念。这里有一个实数的柯西序列 $f : \mathbb{N} \to \mathbb{R}$ 的例子：

可以在表**??**看到这些数字正在越来越接近 0，实际上，我们可以通过选择序列中足够远的项使它们尽可能接近 0。

n	$f(n)$
$f(1)$	$\frac{1}{2}$
$f(2)$	$\frac{1}{4}$
$f(3)$	$\frac{1}{8}$
$f(4)$	$\frac{1}{16}$
n	$\frac{1}{2^n}$

表 A.1: 函数 $f(n)$ 的取值

练习验证我们可以通过找到一个数字 n，使得 $f(n) = \frac{1}{2^n}$ 在误差为 0.000001 的情况下接近 0，让上述序列中的数字尽可能接近 0。换句话说，找出我们需要在序列中走多远，才能让项与 0 的差仅为 0.000001 或更小。把 0.000001 视为 $\frac{1}{10^6}$ 可能会有所帮助。

我们说这个数列收敛于 0。我们可以更数学化地表示上述由 $f(n) = \frac{1}{2^n}$ 定义的实数序列 $f : \mathbb{N} \to \mathbb{R}$ 收敛于 0，即，

对于所有的 $\epsilon > 0$，存在一个 $N \in \mathbb{N}$，使得对所有的 $n > N$，都有

$$|f(n) - 0| < \epsilon \tag{A.213}$$

实际上，你在上面的练习中已经找到了在数学描述中序列收敛于 0 的“N”!

当然，这个特定的序列或数字 0 并没有什么特别之处。我们可以用任何序列和数字来玩这个游戏。一般来说，我们说一个序列 $f : \mathbb{N} \to \mathbb{R}$ 收敛于一个数字 L（L 表示“极限”）当且仅当对所有的 $\epsilon > 0$，存在一个 $N \in \mathbb{N}$，使得对所有的 $n > N$，都有

$$|f(n) - L| < \epsilon \tag{A.214}$$

再次强调，直观地，我们应该将这个理解为“一个数字序列收敛到一个数字 L 当且仅当我们仅通过在序列中取得足够远的项，就可以使这些数字尽可能接近 L。”

现在，柯西序列是一种特殊类型的序列，其项，如我们之前所说，“只要我们看得足够远，就可以尽可能地接近。”让我们重新审视上一个序列

n	$f(n)$
$f(1)$	$\frac{1}{2}$
$f(2)$	$\frac{1}{4}$
$f(3)$	$\frac{1}{8}$
$f(4)$	$\frac{1}{16}$
n	$\frac{1}{2^n}$

表 A.2: 函数 $f(n)$ 的取值

注意以下现象：$f(1)$ 和 $f(2)$ 之间的距离有多远？嗯，

$$|f(1) - f(2)| = \left|\frac{1}{2} - \frac{1}{4}\right| = \left|\frac{1}{4}\right| = \frac{1}{4} \tag{A.215}$$

所以它们相距 $\frac{1}{4}$。$f(2)$ 和 $f(3)$ 之间的距离又有多远呢？你应该检查一下，他们相距 $\frac{1}{8}$。$f(3)$ 和 $f(4)$ 呢？现在，他们相距 $\frac{1}{16}$！嗯...

因此，看起来在序列中越往后走，各对项之间的距离就越来越近。让我们将这一点数学化。

我们说一个序列 $f : \mathbb{N} \to \mathbb{R}$ 是柯西序列，当且仅当对于所有的 $\epsilon > 0$，存在一个 $N \in \mathbb{N}$，使得对于所有的 $m, n \in \mathbb{N}$，如果 $m; n > N$，则有 $|f(m) - f(n)| < \epsilon$。

练习

确信这种精确的数学表述的柯西序列与我们关于序列中越往后的项越接近的直观陈述是一致的。然后，检查序列 $f(n) = \frac{1}{2^n}$ 是否确实是一个柯西序列。

因此，我们确信上述序列是柯西的。现在，我们可以问是否有一个数字序列收敛于我们集合中已经存在的数字。你可能会好奇，一个数字序列怎么可能收敛于我们集合中不存在的东西呢... 好吧，让我们再次看一下序列 $f(n) = \frac{1}{2^n}$。

假设我们只考虑大于 0 的数。换句话说，我们正在处理的数字是

$$(0, \infty) := \{x \in \mathbb{R} : x > 0\} \tag{A.216}$$

这个公式定义了一个集合。其中：

- $(0, \infty)$ 是这个集合的符号表示，意味着是所有大于 0 的实数。
- ":=" 表示定义等于，意味着左边被定义为等于右边。
- "$\{x \in \mathbb{R} : x > 0\}$" 表示的是所有大于 0 的实数的集合。其中：
  - "$x \in \mathbb{R}$" 表示 x 是一个实数。
  - ":" 读作"such that"，在中文中我们常说" 使得" 或" 满足条件的"。
  - "$x > 0$" 是这个集合的定义条件，意味着 x 必须大于 0。

所以整个公式可以翻译为："集合 $(0, \infty)$ 被定义为所有大于 0 的实数的集合"。

所以，我们考虑的集合中的数字不能等于 0。那么，前面的序列就有了一个奇特的性质，它是柯西序列且收敛，但并不收敛于我们所在空间的任何一个数，因为它收敛于 0，而 0 并不在我们的集合中！这就是集合 $\{0, \infty\}$ 不完全的一种失败。形式上，我们说一个集合是完全的当且仅当该集合中的任何柯西序列都收敛于该集合中的一个元素。在语境明确的情况下，我们通常会将这个说法简化为"柯西序列收敛"。

现在，我们可以通过允许自己使用更一般的绝对值概念来测量两个对象之间的距离，从而推广所有这些思想。绝对值实际上是一种被称为度量的更一般现象的特例：

定义

度量的定义

度量是一个二元函数 $d : S \times S \to \mathbb{R}$，它从集合 S 的两个副本的笛卡尔积到实数满足以下性质：

- 对所有的 $x, y \in S, d(x, y) \geqq 0$
- 确定性：对所有的 $x, y \in S, d(x, y) = 0$ 当且仅当 $x = y$
- 三角不等式：对所有的 $x, y, z \in S, d(x, z) \leqq d(x, y) + d(y, z)$

绝对值 $|\cdot| : \mathbb{R} \to \mathbb{R}$ 实际上是一个度量，我们邀请你来检查：

练习

通过验证它满足上述三个属性，检查绝对值 $|\cdot| : \mathbb{R} \to \mathbb{R}$ 是否真的是一个度量。同时，确信第三个性质真的应该被称为三角不等式！

但我们说度量是一个更一般的概念！那么什么是更一般的度量呢？好吧，考虑以下方式来测量两个向量之间的“距离”：给定 $\mathbb{R}^2$ 中的两个向量 u 和 v，我们可以通过取它们差的 (L^2) 范数（作为向量）来测量它们之间的距离，即，$d(u,v) := \|u-v\|_2$。但是稍作思考，我们意识到这只是取向量差与其自身的内积的平方根，即，

$$\|u-v\|_2 = \sqrt{\langle u-v, u-v\rangle} \tag{A.217}$$

我们找到了！这是度量的一个非常一般的概念。我们现在可以用以下方式在任何带有内积的向量空间上定义度量。对于任何带有内积 $\cdot,\cdot\rangle$ 的向量空间 V，定义度量

$$d : V \times V \to \mathbb{R} \tag{A.218}$$

通过

$$d(u,v) := \sqrt{\langle u-v, u-v\rangle} \tag{A.219}$$

我们邀请你检查这个定义是否确实满足了前面度量的条件。

练习

通过检查它是否满足前面列出的标准，检查我们上面定义的度量

$$d : V \times V \to \mathbb{R} \tag{A.220}$$

通过

$$d(u,v) := \sqrt{u-v, u-v} \tag{A.221}$$

是否真的是一个度量。

这两个公式的主要区别在于它们的表达方式和使用环境。在一些数学和物理背景下，它们都描述了向量 u 和 v 之间的距离，但使用不同的方法。

1. 第一个公式 $d(u,v) := \sqrt{\langle u-v, u-v\rangle}$ 使用的是内积 $\langle\cdot,\cdot\rangle$，这是一种在欧几里得空间或更一般的内积空间中定义的操作，用于度量两个向量之间的角度和长度。$\langle u-v, u-v\rangle$ 实际上计算的是向量 $u-v$ 的长度的平方，然后取平方根就得到了 u 和 v 之间的距离。

2. 第二个公式 $d(u,v) := \sqrt{u-v, u-v}$ 缺少内积符号 $\langle\cdot,\cdot\rangle$，所以它在这种形式下可能没有明确的意义，除非在某种特定的环境中”$u-v, u-v$” 可以理解为某种计算 u 和 v 之间距离的特殊方式。

对于这两个公式的解释，你可能需要参考你正在学习或工作的具体背景和上下文，因为不同的领域可能会有不同的符号和记号约定。

“我们说内积在向量空间 V 上引入了一个度量，或者说度量是由内积引导的度量。

A.13.2. 内积的公理化定义

让我们简要谈谈内积的公理化定义的概念。本文前面给出的定义服务我们很好，但事实证明，有更多异乎寻常的方式来取两个向量的内积。实际上，我们可能正在一个向量空间中工作，其中的向量是... 矩阵或其他一些数学对象。例如，有一些向量空间，其“向量”包含了所有连续函数 $f : \mathbb{R} \to \mathbb{R}$，其中两个“向量”（实际上，是函数！）$f$ 和 g 的内积定义为

$$\langle f, g\rangle := \int_0^1 f(x)g(x)dx \tag{A.222}$$

是的，内积，以及向量空间本身，可能会很奇特。所以，我们需要一个能捕捉其本质的公理化框架。

定义

内积的公理化定义在一个关于域 $\mathbb{F}$ 的向量空间 V 中（其中 $\mathbb{F}$ 是 $\mathbb{R}$ 或 $\mathbb{C}$），内积 $\langle\cdot,\cdot\rangle$ 是一个二元函数

$$\langle\cdot,\cdot\rangle : V \times V \to \mathbb{F} \tag{A.223}$$

满足以下属性：

- 共轭对称性：对于所有 $u, v \in V$，
$$\langle u, v\rangle = \overline{\langle u, v\rangle} \tag{A.224}$$
- 第一参数的线性：对于所有 $a \in \mathbb{F}, u, v, w \in V$，
$$\langle a, u, v\rangle = a\ cdot\langle u, v\rangle \tag{A.225}$$
和
$$\langle u + v, w\rangle = \langle u, w\rangle + \langle v, w\rangle \tag{A.226}$$
- 正定性：对于所有 $v \in V$，
$$\langle v, v\rangle \geqq 0 \tag{A.227}$$
和
$$\langle v, v\rangle = 0 \text{ 当且仅当 } v = 0 \tag{A.228}$$

习题

根据以上所述的公理，我们邀请你验证我们一直在使用的内积实际上满足这些公理。

最后，我们可以明确地给出希尔伯特空间的定义了！

A.13.3. 希尔伯特空间的定义

定义

希尔伯特空间的定义

希尔伯特空间是在实数或复数领域上的一个向量空间 H，其配备有一个内积 $\langle\cdot,\cdot\rangle$，相对于由内积导出的度量，它是一个完备的度量空间。

希尔伯特空间的定义是这一章的高潮部分，我们想简要地提及希尔伯特空间在量子计算中的应用。我们关注复数领域 $\mathbb{C}$ 上的希尔伯特空间，以下的讨论仅限于这样的空间。

A.13.4. 量子位作为一个希尔伯特空间

量子计算的一个核心概念是量子位（qubit）可以被表示为一个二维的复数希尔伯特空间。我们通常使用字母 H 表示一个量子位，有时也会用更为花哨的 H，代表“希尔伯特”。

量子位是一个向量空间

量子位由一个向量空间来表示——更具体来说，一个希尔伯特空间！

我们有时会将代表量子位的希尔伯特空间称为状态空间。状态空间中的一个状态是状态空间中的一个向量，其 L^2 范数为 1。因此，一个状态就在布洛赫球面上——参考第 3 章。例如，熟悉的向量 $|0\rangle$ 和 $|1\rangle$ 就是一个量子位所在的二维希尔伯特空间（状态空间）$\mathcal{H}$ 中的状态。实际上，$|0\rangle$ 和 $|1\rangle$ 是 $\mathcal{H}$ 的一组正交规范基，如你在我们之前讨论的正交规范基中验证过，因此状态空间中的每个向量都是这两个向量的线性组合，其 L^2 范数为 1。用量子力学的术语来说，每个状态都是这两个状态的叠加！

我们通常将 n 个量子位的集合称为量子寄存器，并常常表示为

$$\underbrace{\mathcal{H}\otimes\mathcal{H}\otimes\cdots\otimes\mathcal{H}}_{n个} \tag{A.229}$$

事实证明，希尔伯特空间的张量积也是一个希尔伯特空间，尽管它的维数通常会更大！

练习

找出基 $\mathcal{B}_{\mathcal{H}^{\otimes 3}}$ 对应于 $\mathcal{H}^{\otimes 3}$，其中 $\mathcal{H}$ 的基为 $\mathcal{B}_{\mathcal{H}}=\{|0\rangle,|1\rangle\}$。根据上一个练习，它应包含 $s^3=8$ 个向量。你能找出 $H^{\otimes n}$ 的基吗？

量子寄存器是张量积

一个由 n 个量子位组成的量子寄存器是一个 2^n 维的向量空间的张量积！

量子计算	线性代数	示例
量子比特（qubit）	二维复希尔伯特空间	$\mathcal{H}=span_{\mathbb{C}\{|0\rangle,|1\rangle\}}$
n-量子比特量子寄存器	二维复希尔伯特空间的 n-重张量积	$\mathcal{H}^{\times n}$
态空间	L^2 范数为 1 的向量（Bloch 球）	$\{\mathsf{v}\in\langle:\langle v\|v\rangle=1\}$
确定态	正交基向量	$|0\rangle,|1\rangle$
态的叠加	L^2 范数为 1 的正交基向量的线性组合（在 Bloch 球上）	$\frac{1}{2}|0\rangle,\frac{1}{2}|1\rangle$
量子逻辑门	酉算子	$X=\begin{pmatrix}0,1\\1,0\end{pmatrix}$
测量算子	投影算子	$\begin{pmatrix}0,1\\1,0\end{pmatrix}$

表 A.3: 量子计算与线性代数的关系

量子计算与线性代数关系的总结：

- **一个量子位可以用一个二维的复数希尔伯特空间来表示。量子位的状态由希尔伯特空间中的一个向量来表示。**

  在量子计算中，我们常常使用希尔伯特空间来表示量子位（qubit）。希尔伯特空间是一种具有内积结构的线性空间，其中的向量可以表示复数和实数。

一个量子位可以被表示为一个二维的复数希尔伯特空间，这是因为在最基本的情况下，一个量子位有两种可能的状态，通常表示为 $|0\rangle$ 和 $|1\rangle$。在这个二维希尔伯特空间中，$|0\rangle$ 和 $|1\rangle$ 是空间的一组基，可以用来表示空间中的任何向量。

量子位的状态由希尔伯特空间中的一个向量来表示。这个向量的每个分量是复数，可以用来描述量子位的概率振幅。换句话说，这个向量的每个分量都与量子位处于特定状态的概率振幅相关。而且，这个向量的长度（或范数）必须为 1，这是因为它代表了概率的分布，而所有可能性的概率总和必须为 1。

因此，我们可以说一个量子位可以用二维的复数希尔伯特空间来表示，而量子位的状态则由希尔伯特空间中的一个向量来表示。

- **更具体地说，表示量子位状态的向量具有 L^2 范数为 1。**

  更具体地说，当我们用一个向量来表示量子位的状态时，这个向量具有一个特殊的性质，即其 L^2 范数为 1。在量子力学中，我们使用概率振幅来描述量子系统的状态。

  概率振幅是一个复数，它描述了量子位处于不同基态的概率幅度。当我们考虑一个量子位的状态时，我们使用一个向量来表示它，该向量的每个分量对应于量子位处于不同基态的概率振幅。

  量子位的状态向量可以表示为：

$$|\psi\rangle = a\,|0\rangle + b\,|1\rangle$$

  其中，$|0\rangle$ 和 $|1\rangle$ 是量子位的基态。而 a 和 b 是复数，它们的模的平方和为 1，即 $|a|^2 + |b|^2 = 1$。

  这个条件确保了量子位的概率分布是归一化的，即所有可能的状态出现的概率之和为 1。通过保持 L^2 范数为 1，我们可以确保量子位的状态是可靠的概率描述。

- **由 n 个量子位组成的量子寄存器是一个 2^n 维的复数希尔伯特空间，由表示量子位的二维希尔伯特空间的 n 次张量积组成。**

  由 n 个量子位组成的量子寄存器可以用一个复数希尔伯特空间来表示，其维度为 2^n。这个复数希尔伯特空间是由表示量子位的二维希尔伯特空间的 n 次张量积组成的。

  具体地，如果我们用 $\mathcal{H}$ 表示一个单个量子位的二维希尔伯特空间，那么由 n 个量子位组成的量子寄存器的希尔伯特空间可以表示为 $\mathcal{H}^{\otimes n}$。这里的符号"$\otimes$" 表示张量积操作，它将 n 个二维希尔伯特空间进行组合。张量积操作可以看作是将量子位的状态向量逐个组合起来，以形成表示整个量子寄存器的向量。

  由于每个二维希尔伯特空间是二维的，具有两个基态，所以 n 个二维希尔伯特空间的 n 次张量积会产生 2^n 个基态。因此，由 n 个量子位组成的量子寄存器的希尔伯特空间是一个 2^n 维的复数希尔伯特空间。

  这个复数希尔伯特空间可以用来表示量子寄存器的所有可能状态，包括它们的叠加态和纠缠态。在量子计算中，我们利用这个 2^n 维的复数希尔伯特空间进行计算和操作，以实现量子算法的运行。

- **空间的计算基向量代表量子位的确定计算状态。**

  空间的计算基向量是指量子位所能处于的确定计算状态。在量子计算中，计算基是描述量子位的基础集合，它由量子位的基态组成。

对于一个单个量子位，计算基包括两个基态，通常表示为 $|0\rangle$ 和 $|1\rangle$。其中，$|0\rangle$ 表示量子位处于基态 0，$|1\rangle$ 表示量子位处于基态 1。这两个基态可以用量子力学的记号来表示，其中的竖线符号表示量子态。

在量子计算中，计算基向量表示了量子位的确定计算状态。当量子位处于计算基向量中的一个时，它处于一个确定的状态，也就是说我们能够以 100% 的概率得到这个基态的测量结果。

例如，对于一个二位量子寄存器（两个量子位的组合），计算基向量可以表示为 $|00\rangle$，$|01\rangle$，$|10\rangle$，$|11\rangle$。这些基向量代表了量子寄存器处于不同的确定计算状态，其中每个量子位可以处于基态 0 或 1。

通过在计算基向量上进行量子操作和测量，我们可以对量子计算进行编码和控制。计算基向量在量子算法和量子门操作中起着重要的作用，它们提供了进行量子计算的基本构建块。

- **状态的叠加是计算基向量的线性组合。**

  状态的叠加是指量子位可以同时处于多个计算基向量的线性组合状态。在量子力学中，量子位的状态可以通过线性组合来表示，其中不同计算基向量的系数被称为概率振幅。

  具体地说，对于一个量子位，它可以处于计算基向量 $|0\rangle$ 和 $|1\rangle$ 所表示的确定计算状态。而在量子计算中，我们允许量子位以叠加的形式处于这些基态之间，即处于两个基态的线性组合。

  假设我们有一个量子位，其状态可以表示为 $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$，其中 a 和 b 是复数，表示量子位处于基态 $|0\rangle$ 和 $|1\rangle$ 的概率振幅。

  这里需要强调的是，概率振幅的平方 $|a|^2$ 和 $|b|^2$ 分别表示量子位处于基态 $|0\rangle$ 和 $|1\rangle$ 的概率。而概率振幅的相对相位和幅度决定了量子位的相对相位和幅度。

  因此，状态的叠加指的是量子位可以处于计算基向量的线性组合状态，其中不同基态的概率振幅相互叠加，并且这些概率振幅遵循量子力学中的线性叠加原理。通过控制概率振幅的系数，我们可以实现量子位的状态叠加和干涉效应，从而进行量子计算中的并行计算和相干操作。

- **量子逻辑门是作用在状态空间上的酉算子。由于酉算子保持其作用的向量的范数，状态被转化为新的状态，而不是空间中的任何向量。因此，我们可以使用酉算子构建量子电路。**

  量子逻辑门是指作用在量子位的状态空间上的酉算子。在量子计算中，逻辑门用于对量子位进行操作和变换，从而实现特定的计算任务。

  一个酉算子是一个线性算子，它保持向量的长度（范数）不变，同时保持向量之间的夹角。这意味着酉算子在量子态空间中执行的操作是保持态矢量的归一性和幅度的变化，而不改变其相对相位和幅度。

  当量子逻辑门作用于一个量子位的状态时，它会将该状态转化为一个新的状态，而不是转化为状态空间中的任意向量。这是因为酉算子保持态矢量的范数不变，即保持概率幅度的大小不变。这种特性是量子计算中非常重要的，因为它确保了计算过程中的量子态保持归一且可逆。

  正是由于酉算子的这些特性，我们可以利用它们来构建量子电路。量子电路是由一系列量子逻辑门按特定顺序组合而成的。通过选择适当的酉算子，并将它们组合在一起，我们可以实现量子计算中所需的各种操作和计算任务。

因此，量子逻辑门的酉性质允许我们在量子计算中进行可逆的、幺正的操作，并确保量子态的保真度和相干性。这使得我们能够利用量子逻辑门来设计和构建复杂的量子算法和量子计算任务。

对线性代数的回顾以表A.3中的表格为主要内容。希望深入理解线性代数的读者，可以阅读以下资料：

- Sheldon Axler 的《线性代数真实应用》（Linear Algebra Done Right）[180] 侧重于线性代数中常见的理论和证明技巧，介绍了矩阵的复杂视角、谱定理（确定线性算子的特征向量是否构成一组基）和行列式。
- Michael Artin 的《代数学》（Algebra）[181] 面向更数学化的读者。它以对初等矩阵及其在高斯消元中的作用的精彩讨论为起点，并为群论提供了很好的视角。
- Gilbert Strang 的教材《线性代数及其应用》非常适合将线性代数应用于科学领域，此外还有免费的 MIT OpenCourseWare 教材作为补充资料。
- 请参考 Rieffel 和 Polak 的教材《量子计算，简单入门》（Quantum Computing, A Gentle Introduction）中的附录 A 和 B，了解量子力学与概率论之间的联系，以及对阿贝尔隐藏子群问题的讨论 [182]。
- 对于对抽象代数感兴趣的读者，以下是一些额外的教材：
  - John Fraleigh 的《抽象代数初级课程》（A First Course in Abstract Algebra）[183]。
  - Dummit 和 Foote 的《抽象代数》（Abstract Algebra）[184]。
  - Joseph Rotman 的《高级现代代数》（Advanced Modern Algebra）[185]。
- F. William Lawvere 和 Stephen H. Schanuel 的《概念数学：范畴论初步》（Conceptual Mathematics: A First Introduction to Categories）[186] 是对范畴论的一个有趣介绍。
- 对于想要更深入了解张量积的精确定义的读者，可以查阅 Tai-Danae Bradley 的博客 Math3ma[6]。
- 对于更具冒险精神的读者，推荐阅读 Bradley 的著作《应用范畴论是什么?》（What is Applied Category Theory?）[187]，了解到范畴论不仅仅是对数学的重新表述，实际上在化学和自然语言处理等领域非常有用！
- Emily Riehl 的《范畴论在背景中》（Category Theory in Context）[188] 是对范畴论的帮助性介绍，适合高年级本科生和研究生阅读。

A.14. 布尔函数

定义：布尔函数

布尔函数 f 是从笛卡尔积

$$\{0,1\}^n \to \{0,1\}^m \tag{A.230}$$

的函数，其中 $\{0,1\}^n$ 表示集合 $\{0,1\}$ 的 n 重笛卡尔积，即

$$\{0,1\}^n := \underbrace{\{0,1\} \times \cdots \times \{0,1\}}_{n\text{次}} \tag{A.231}$$

[6] https://www.math3ma.com

如果您想复习这些术语，笛卡尔积和函数的概念在第 11 章中进行了讨论。布尔函数在计算中自然而然地出现。例如，在第 7 章讨论的 Deutsch-Jozsa 算法中涉及到四个布尔函数。

$$f_0, f_1, f_x, f_{\overline{x}} \tag{A.232}$$

在布尔函数（Boolean function）的环境中，f_0, f_1, f_x, $f_{\overline{x}}$ 这些符号的意义可以被理解为特殊的布尔函数：

1. f_0: 这是常数函数，无论输入是什么，输出都是 ‘0‘。

2. f_1: 这也是常数函数，无论输入是什么，输出都是 ‘1‘。

3. f_x: 这是恒等函数，也就是说，如果输入是 x，那么输出也是 x。在布尔函数的上下文中，x 通常是一个布尔值，即 ‘0‘ 或 ‘1‘。

4. $f_{\overline{x}}$: 这是布尔否定函数。也就是说，如果输入是 x，那么输出是 x 的否定，通常表示为 $\sim x$ 或者 $\neg x$。在布尔函数的上下文中，如果 x 是 ‘0‘，那么 $\sim x$ 或者 $\neg x$ 就是 ‘1‘，反之亦然。

每个函数的定义域和值域都是 $\{0,1\}$。因此，对于这四个函数来说，当我们将它们与上述布尔函数的定义进行比较时，$n = 1$ 且 $m = 1$。

其他布尔函数的例子包括非（NOT）、与（AND）、或（OR）和异或（XOR）。

利率	每年复利次数	一年内的金额
$\frac{100}{1}$	一年 1 次	2 美元
$\frac{100}{2}$	一年 2 次	2.25 美元
$\frac{100}{3}$	一年 3 次	2.37 美元
$\frac{100}{4}$	一年 4 次	2.44 美元
$\frac{100}{5}$	一年 5 次	2.49 美元
$\frac{100}{100}$	一年 100 次	2.70481 美元
$\frac{100}{\infty}$	一年 ∞ 次	e 美元

”The Banker’s Experiment” 通常指的是一个被用来描述计算机操作系统中的死锁（Deadlock）处理机制的实验，它通常被称为”Banker’s Algorithm” 或” 银行家算法”。

这个算法的名称来源于这样一种情况：银行家分配可用的资源（例如贷款），以满足多个客户的需求，但是他们必须确保，即使满足其中一些客户的需求，也有足够的资源来满足其余客户的最大需求，以防止出现债务危机。

在计算机科学中，这个算法被用来避免系统中的进程陷入死锁。系统中的资源被视为银行家，而进程被视为客户。银行家算法通过预先计算资源分配的安全性来避免死锁，确保系统始终保持在一个安全状态，即系统可以满足所有进程的最大需求而不导致死锁。

这个表格是在解释一种被称为” 复利” 的经济概念。复利是一种利息计算方式，其中利息基于初始本金（初始投资或贷款金额）以及之前累积的利息计算。

让我们看看表格中的每一行：

利率是 100%，并且一年只计算一次利息（即，利息每年复合 1 次）。所以，如果你开始时有 1 美元，一年后你将有 2 美元。

利率仍然是 100%，但现在每年复合 2 次。因此，每六个月，你的金额都会增加一半的利息。你开始时有 1 美元，六个月后你有 1.50 美元，然后再过六个月你有 1.50 美元的 150%（也就是 2.25 美元）。

利率是 100%，一年复合 3 次。这意味着每四个月，你的金额都会增加三分之一的利息。因此，你最后得到的金额约为 2.37 美元。

如果你每季度（即每年 4 次）复合一次 100% 的利率，那么一年后你将有约 2.44 美元。

如果你将利率 100% 每年复合 5 次，那么一年后你将有约 2.49 美元。

如果你将利率 100% 每年复合 100 次，那么一年后你将有约 2.70481 美元。

如果你将利率 100% 每年复合无限次，这就是连续复利的概念。在这种情况下，一年后的金额将等于欧拉数 e 美元，约等于 2.71828 美元。

因此，这个表格展示了当你增加复利次数时，你的投资如何增长。当复利次数趋近无限大时，它将趋近于一个特定的值，即欧拉数 e，这是连续复利的基础。

A.15. 对数和指数

首先考虑以数 e 为底的自然对数。e 大约为 2.71，在众多的数学和科学背景下都有出现。以下是一个关于 e 的有趣银行家思维实验的方式：

在银行存入 1 美元，年利率为 $\frac{100}{1}$%，一年后（每年一次），你将赚取 1 美元，因此一年后你将拥有总共 2 美元。现在，允许自己每年获得两次利息，但利率折中为 $\frac{100}{2} = 50\%$。在前六个月后，你将赚取 50 美分，因为 50 美分是已投资的 1 美元的 50%。再过六个月，你将获得你现有的 1.50 美元的 50%，最后你将有 2.25 美元。这比你一年只复利一次的收益要多。如果你一年复利三次，你将有折中的 $\frac{100}{3}$% 的利率。你会发现，在这个利率下，你的收益会略微增加。如果你允许自己以 $\frac{100}{\infty}$% 的折中利率“无限次”复利，你最后将得到 e 美元！

这在图A.2中的表格中进行了总结。

$$ln := log_e : (0, \infty) \to (-\infty, \infty) \tag{A.233}$$

其中“ln”是自然对数。

以下等价性引导了所有将要来的事实：

对数和指数的等价性

$$ln(y) = x :\Leftrightarrow e^x = y \tag{A.234}$$

自然对数的一些初步性质包括：

$$ln(e^x) = x : 和 : e^{ln(x)} = x \tag{A.235}$$

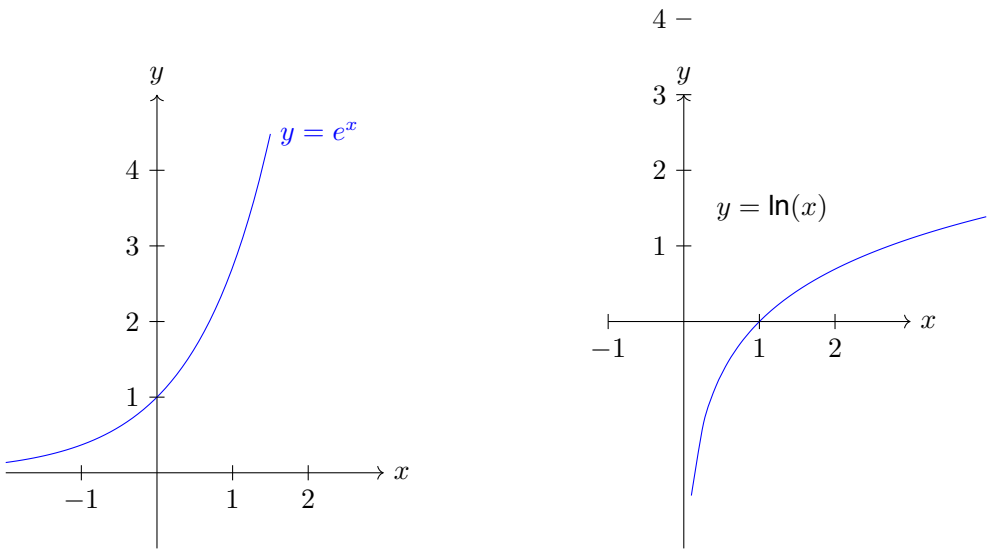
所以，函数 $ln(x)$ 和 e^x 是互为逆函数。

进一步，对于任意两个正实数 a 和 b，

$$ln(a \cdot b) = ln(a) + ln(b) \tag{A.236}$$

实质上是因为

$$e^a e^b = e^{a+b} \tag{A.237}$$



(a) $y = e^x$ 的图像 **(b)** $y = \ln(x)$ 的图

图 A.2: 示例图形

所以，对数把乘积转化为了和!

使用上述性质，我们可以推导出对所有正实数 a、b 和任何实数 c 的以下结论：

- $ln(\frac{b}{a}) = ln(b) - ln(a)$ 因为 $\frac{e^b}{e^a} = e^{b-a}$
- $ln(a^c) = c \cdot ln(a)$ 因为 $(e^a)^c = e^{ac}$

我们还有一个以 2 为底的对数 log_2。与自然对数相似，

$$log_2(y) = x \iff 2^x = y \tag{A.238}$$

事实上，自然对数可以写为 $ln(x) = log_e(x)$，即，明确提到以 e 为底。对于每一个正实数底数，我们都有对应的对数，所以能够进行底数的变换是很方便的。为了将一个对数从一个底数 b 变换到另一个底数 c，我们可以使用公式

$$log_b(a) = \frac{log_c(a)}{log_c(b)} \tag{A.239}$$

记住这个公式的一个技巧是，左边 a 在 b 上面，变换底数到 c 之后，右边 a 仍然在 b 上面。

A.16. 欧拉公式

通过幂级数给出一个证明的暗示（为了简洁起见，避免涉及收敛性问题），关于欧拉的神秘而美妙的公式，将复数的单位范数的极坐标与数字 e 相联系，如前一章所述并在图A.8中所示。

这个示意图表示了欧拉公式 $e^{i\theta} = \cos\theta + i\sin\theta$ 在复数平面中的几何意义：$e^{i\theta}$ 对应于单位圆上的一个点，其实部和虚部分别等于 $\cos\theta$ 和 $\sin\theta$。

$e^{iz} = 1 + (iz) + \frac{(iz)^2}{2!} + \frac{(iz)^3}{3!} + \frac{(iz)^4}{4!} + \cdots$

$= 1 + (iz) + \frac{i^2z^2}{2!} + \frac{i^3z^3}{3!} + \frac{i^4z^4}{4!} + \cdots$

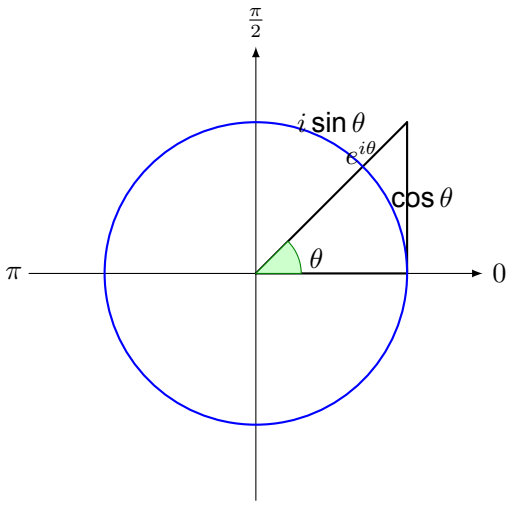


图 A.3: 欧拉公式在复平面中的表示

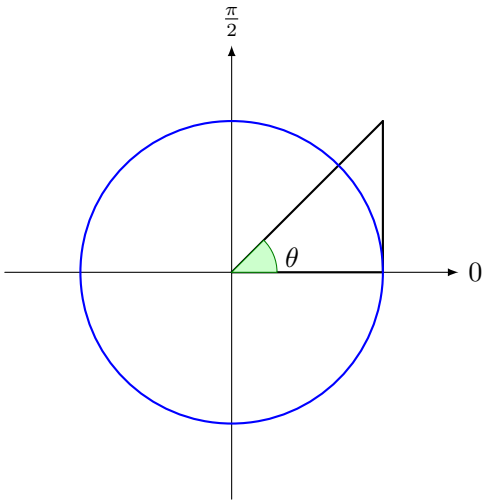


图 A.4: 欧拉公式在复平面中的表示

（复值）余弦和正弦函数的幂级数为：

$$cos(z) = 1 - \frac{z^2}{2!} + \frac{z^4}{4!} - \frac{z^6}{6!} + \cdots \tag{A.240}$$

以及

$$sin(z) = z - \frac{z^3}{3!} + \frac{z^5}{5!} - \frac{z^7}{7!} + \cdots \tag{A.241}$$

（复值）指数函数 $e^z := \exp(z)$ 的幂级数为：

$$e^z := \exp(z) = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!} \cdots \tag{A.242}$$

然后，将复数 iz 代入指数函数的幂级数可得：

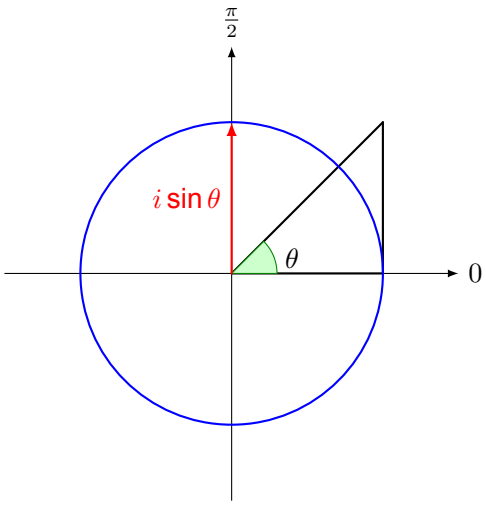


图 A.5: 欧拉公式在复平面中的表示

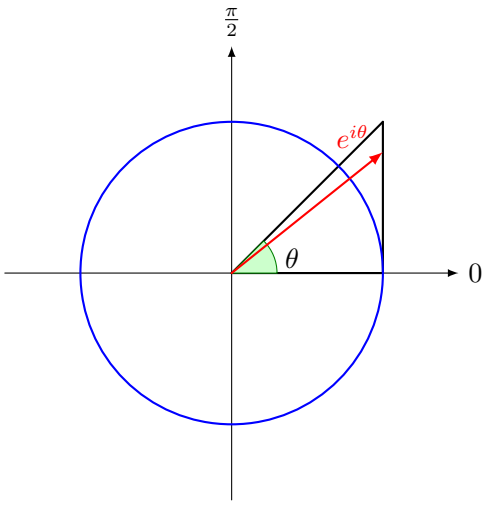


图 A.6: 欧拉公式在复平面中的表示

$$
\begin{aligned}
e^{iz} &= 1 + (iz) + \frac{(iz)^2}{2!} + \frac{(iz)^3}{3!} + \frac{(iz)^4}{4!} + \cdots \\
&= 1 + (iz) + \frac{i^2 z^2}{2!} + \frac{i^3 z^3}{3!} + \frac{i^4 z^4}{4!} + \cdots
\end{aligned}
$$

并记住 $i^2 = -1, i^3 = -i$ 和 $i^4 = 1$，可以进一步推导出：

$$
\begin{aligned}
&= 1 + (iz) + \frac{i^2 z^2}{2!} + \frac{i^3 z^3}{3!} + \frac{i^4 z^4}{4!} + \cdots \\
&= 1 + (iz) + \frac{-1 z^2}{2!} + \frac{-i z^3}{3!} + \frac{z^4}{4!} + \cdots \\
&= 1 + (iz) - \frac{z^2}{2!} - \frac{i z^3}{3!} + \frac{z^4}{4!} + \cdots
\end{aligned}
$$

在这个公式中，i 是虚数单位，满足 $i^2 = -1$，$i^3 = -i$，$i^4 = 1$，这是一种周期性模式。经过一些重新组织，我们得到

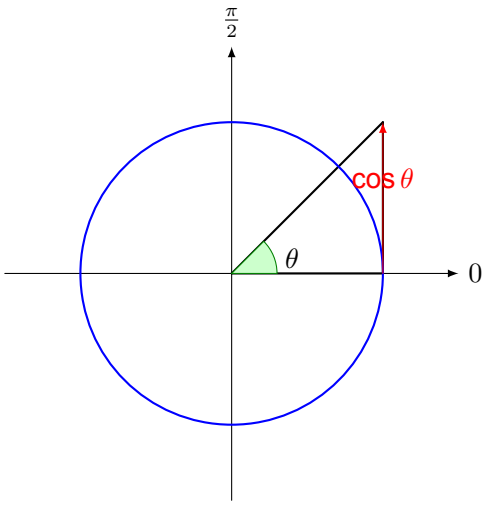


图 A.7: 欧拉公式在复平面中的表示

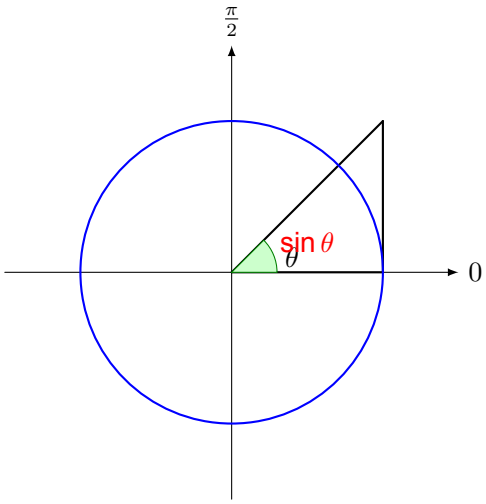


图 A.8: 欧拉公式在复平面中的表示

$$= \left(1 - \frac{z^2}{2!} + \frac{z^4}{4!} - \cdots\right) + \left(iz - \frac{z^3}{3!} + \frac{z^5}{5!} - \cdots\right) \tag{A.243}$$

现在，将 $sin(z)$ 的幂级数乘以 i 可得

$$\begin{aligned} isin(z) &= i\left(z - \frac{z^3}{3!} + \frac{z^5}{5!} - \frac{z^7}{7!}\cdots\right) \\ &= iz - i\frac{z^3}{3!} + i\frac{z^5}{5!} - i\frac{z^7}{7!} + \cdots \\ &= iz - \frac{iz^3}{3!} + \frac{iz^5}{5!} - \frac{iz^7}{7!} + \cdots \end{aligned} \tag{A.244}$$

$$\begin{aligned} isin(z) &= i\left(z - \frac{z^3}{3!} + \frac{z^5}{5!} - \frac{z^7}{7!}\cdots\right) \\ &= iz - i\frac{z^3}{3!} + i\frac{z^5}{5!} - i\frac{z^7}{7!} + \cdots \\ &= iz - \frac{iz^3}{3!} + \frac{iz^5}{5!} - \frac{iz^7}{7!} + \cdots \end{aligned} \tag{A.245}$$

加上 $cos(z)$ 的幂级数和新获得的 $isin(z)$ 表达式，我们得到

$$\begin{aligned} cos(z) + isin(z) = &\left(1 - \frac{z^2}{2!} + \frac{z^4}{4!} - \ldots\right) \\ &+ \left(iz - \frac{z^3}{3!} + \frac{z^5}{5!} - \ldots\right) \end{aligned} \tag{A.246}$$

这正是我们早前为 e^{iz} 找到的表达式！因此，

$$e^{iz} = cos(z) + isin(z) \tag{A.247}$$

使用欧拉公式，我们得到欧拉恒等式：

$$e^{i\pi} + 1 = 0 \tag{A.248}$$

欧拉恒等式是一个非常著名的公式，它连接了数学中五个最重要的数：0、1、π、e 和 i。

这个等式包含了几个重要的数学元素：

1. e 是自然对数的底，约等于 2.71828。是虚数单位，满足 $i^2 = -1$ 的定义。
2. π 是圆周率，约等于 3.14159。
3. 0 和 1 是整数和实数的两个基础元素。

这个等式以这样简洁的方式连接了代数、几何、三角函数和复数等数学领域的重要元素，被许多数学家视为数学之美的极致体现。

B

附录 A：量子计算赛事

B.1. " 司南杯" 量子计算编程挑战赛

B.1.1. 大赛简介

” 司南杯” 量子计算编程挑战赛是由中国计算机学会（CCF）主办的一项国际级的量子计算竞赛。本竞赛旨在推动量子计算技术的发展，并将这一新兴科技引向公众视野，特别是学术界和信息技术行业。

” 司南杯” 量子计算编程挑战赛主要集中在量子计算和量子编程的各个方面，包括但不限于量子算法、量子软件工具、量子机器学习、量子通信、量子错误纠正、量子编程语言以及量子硬件的设计和实现。

图 B.1: ” 司南杯” 量子计算编程挑战赛网站大赛地址：`https://contest.originqc.com.cn/`

本竞赛面向全球开放，吸引了来自不同背景的参赛者，包括学生、研究人员、开发人员和企业

家，以及来自广泛的学科领域，如物理学、计算机科学、工程学、数学等。通过这样的比赛，不仅能够增进公众对于量子计算的理解，同时也为在这一领域内的创新提供了一个重要的平台。

” 司南杯” 量子计算编程挑战赛采用多阶段的形式进行。在初赛阶段，参赛者需在给定的时间内完成一系列预定的编程任务，挑战各种量子算法和量子编程问题。在复赛阶段，优胜者将会被邀请进行更深入、更复杂的量子编程任务。

评审团由一组具有量子计算背景的国际专家组成，他们将根据参赛者的解决方案的创新性、技术深度和应用潜力进行评估。除了主要的奖项之外，还设有一些特别奖项来表彰在某一特定领域做出突出贡献的参赛者。

” 司南杯” 量子计算编程挑战赛不仅是一个评估和挑战量子编程能力的平台，更是一个促进量子计算学术交流、共享最新研究成果和技术进展的重要场所。通过这种方式，中国计算机学会希望能够推动中国乃至全球的量子计算技术的进步，以应对未来的计算挑战。

此外，” 司南杯” 量子计算编程挑战赛也是为了应对量子计算人才的短缺。在这个日益重要的领域中，需要大量具备专业知

识和技能的人才。因此，通过此类竞赛，CCF 希望能够吸引和培养更多的量子计算人才，为未来的量子科技革命奠定坚实的基础。

” 司南杯” 量子计算编程挑战赛是一个富有挑战性和创新性的竞赛，它提供了一个展示量子编程技巧、分享量子计算知识、探索量子科技未来的平台，为推动全球量子计算技术的发展做出了重要贡献。

B.1.2. 2023 年" 司南杯" 量子计算编程挑战赛金融赛道第一名

XXXXXXXXXXXXXXX

XXXXXXXXXXXXX

提供获奖团队相关成果代码、报告、实验整理汇总成案例课程类似 A.2.2 章节在大赛官网没找到

B.1.3. 2023 年" 司南杯" 量子计算编程挑战赛 XX 赛道第一名

XXXXXXXXXXXXXXX

XXXXXXXXXXXXX

提供获奖团队相关成果代码、报告、实验整理汇总成案例课程类似 A.2.2 章节在大赛官网没找到

B.2. QHack 国际大赛

B.2.1. 大赛简介

QHack 2023 是一项全球性的、面向公众的量子计算竞赛，由全球量子计算社区的领军组织主办。这项竞赛聚集了学生、研究员、开发者、科学家和业界专家，共同在量子计算的前沿领域挑战和创新。QHack 的第一届比赛于 2019 年举办。

QHack 的目标是推进量子计算的研究与应用，同时也激发更多人对量子计算技术的兴趣和热情。竞赛提供了一个学习和实践量子计算基础知识和技术的平台，同时也是分享和交流最新研究成果和技术进展的场所。

QHack 2023 的比赛项目涵盖了量子计算的多个关键领域，包括量子算法设计、量子机器学习、量子优化、量子软件开发、量子通信、量子错误纠正等。在这些项目中，参赛者需要运用和展示他们在量子计算领域的专业知识和技术能力。

竞赛分为预选赛和决赛两个阶段。在预选赛阶段，参赛者需要在规定时间内完成一系列在线编程任务，这些任务设计用来测试和评估他们在量子计算技术和编程方面的能力。根据预选赛的成绩，一部分优秀的参赛者将进入决赛阶段。在决赛阶段，参赛者将面临更高级、更具挑战性的量子编程任务，需要他们深入应用量子计算的理论和技术。

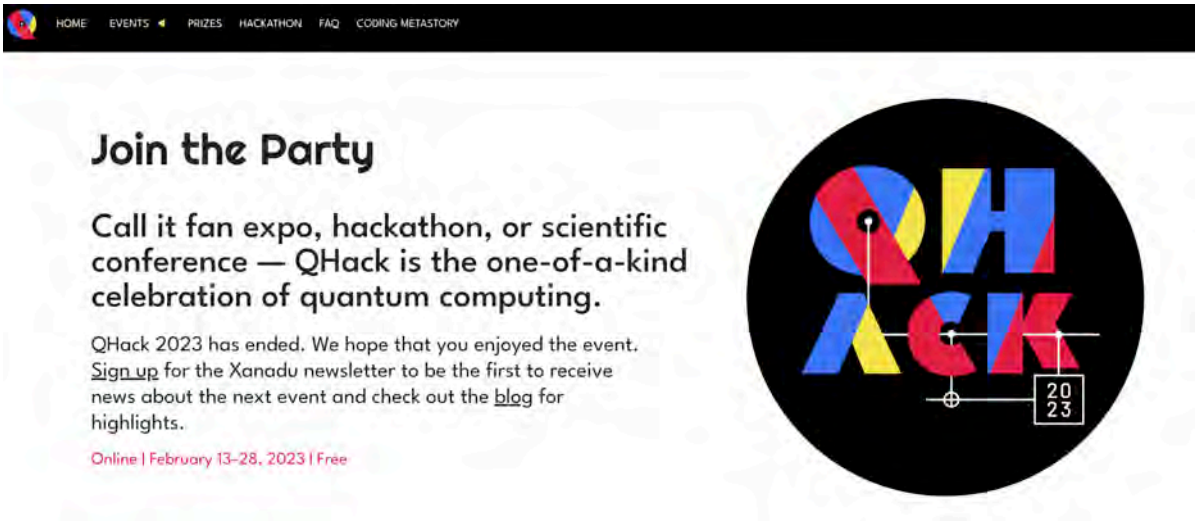


图 B.2: qhack 大赛网站大赛地址：`https://qhack.ai/`

QHack 2023 的评审团由一组国际知名的量子计算专家组成，他们将根据参赛作品的技术深度、创新性、实用性和完成度等多个维度进行评判。除了主要奖项之外，比赛还设有一些特殊奖项，用以表彰在某个特定任务或领域表现出色的参赛者。

除了竞赛之外，QHack 2023 还设置了一系列研讨会、工作坊和主题讲座，邀请全球量子计算领域的顶级专家分享他们的见解和研究成果，探讨量子计算的前沿问题和未来发展趋势。

QHack 2023 是一场规模大、影响广、质量高的量子计算竞赛。它提供了一个实践、学习、交流和创新的平台，对于推动量子计算领域的发展，提升公众对量子计算的理解和认识，培养新一代的量子计算人才，都起到了积极的推动作用。

B.2.2. 2023 年 QHack 大赛 NVIDIA 赛道第一名

基于 PennyLane-Lightning 和 NVIDIA cuQuantum SDK 加速噪声算法的研究

Accelerating Noisy Algorithm Research with PennyLane-Lightning and NVIDIA cuQuantum SDK

“利用 PennyLane-Lightning 和 NVIDIA cuQuantum SDK 加速噪声算法研究”，是由慕尼黑工业大学的 Lion Frangoulis、Cristian Emiliano Godinez Ramirez、Emily Haworth 和 Aaron Sander 合作完成的 。Github 开源地址：https://github.com/aaronleesander/QHack2023/tree。该项目主要探索了模拟噪声量子算法的挑战，这些算法被认为需要大量的计算资源。为了解决这个问题，团队利用了 Xanadu 的 PennyLane-Lightning-GPU 和 NVIDIA 的 cuQuantum SDK 提供的 GPU 工具，使扩大模拟量子计算的模拟规模成为可能，并深入了解噪声对量子算法的影响。他们的分析揭示了噪声对模拟的一般影响，并识别出可以加速开放量子系统和基态优化模拟的区域。

在本项研究中，团队使用了两个工具包：

- Xanadu 的 PennyLane-Lightning-GPU[1]；
- NVIDIA 的 cuQuantum SDK

首先，PennyLane 是一个开源的 Python 库，为量子机器学习，量子计算，和量子化学提供了工具和功能。其主要特点是可以用经典的机器学习优化算法来训练和优化量子电路。PennyLane 的名字源于英国摇滚乐队披头士（The Beatles）的一首歌图B.3，该歌的名字也叫”Penny Lane”。PennyLane 的主要目标是使得量子计算和量子机器学习更加接近经典的机器学习和数据科学的工

[1]主题网站：https://developer.nvidia.com/cuquantum-sdk

作流程。因此，它被设计成可以与 TensorFlow[2]，PyTorch[3]，和 JAX[4]等经典机器学习库无缝集成。PennyLane 的另一个重要特点是它的硬件独立性。它提供了一种统一的编程接口，可以在多种量子处理器（QPU）和量子模拟器上运行，包括 Rigetti，IBM Q，Google Cirq，Microsoft Quantum Development Kit，以及 Strawberry Fields 等。这使得你可以在不同的硬件平台上运行你的量子电路，而无需修改你的代码。

而”PennyLane-Lightning-GPU” 是指 PennyLane 框架中的一个插件，用于将基于 GPU 的加速功能引入到 PennyLane 中。PennyLane 是一个用于量子机器学习和量子优化的开源软件库，用于在量子计算中构建和训练量子神经网络。PennyLane-Lightning-GPU 插件允许用户在进行量子计算任务时利用 GPU 加速。GPU（图形处理器）是一种并行计算设备，可以高效地执行大规模数值计

[2]TensorFlow 是一个开源软件库，用于进行高性能数值计算。由 Google Brain 团队开发，被广泛应用于机器学习和深度学习领域，但其灵活的计算核心也可以被应用于许多其他科学领域。

以下是关于 TensorFlow 的一些关键特性：

1. **灵活性**：TensorFlow 不仅支持深度学习，还支持其他机器学习算法。它可以在各种平台上运行，包括在 CPU 和 GPU 上，无论是在桌面、服务器、移动设备还是边缘设备上。
2. **易于使用**：TensorFlow 提供了高级 API（例如 Keras），使得设计和训练复杂的机器学习模型变得更加容易。此外，TensorFlow 也提供了 TensorBoard，一个可视化工具，可以帮助用户理解、调试和优化 TensorFlow 程序。
3. **高效性能**：TensorFlow 能够充分利用现代硬件的能力，提供了高效的张量运算库，如卷积和矩阵乘法，并且有专门针对 GPU 和 TPU 的优化。
4. **扩展性**：TensorFlow 具有良好的扩展性，可以支持大规模的计算，适合在多台机器上进行分布式计算。
5. **开源和生态系统**：TensorFlow 是开源的，这意味着任何人都可以贡献代码，使其变得更好。此外，TensorFlow 的生态系统包含了大量的库和工具，可以帮助用户构建和部署复杂的机器学习应用。

TensorFlow 的主要构建块是张量（tensor），张量是一个可以在多维数组中保存数据的对象。这些张量在计算图（computation graph）中流动，计算图描述了这些张量之间的运算关系。在 TensorFlow 中，你首先构建一个计算图，然后使用会话（session）来执行图中的运算。这种方式使得 TensorFlow 能够延迟计算，只在需要结果时才进行，同时也能够自动地计算导数，这对于机器学习算法（特别是神经网络）来说非常有用。

[3]PyTorch 是一个用于机器学习的开源 Python 库，由 Facebook 的人工智能研究团队开发。PyTorch 被设计为具有高度灵活性和速度，同时依然提供了方便使用的高级 API。以下是关于 PyTorch 的一些关键特性：

1. **灵活性**：PyTorch 使用动态计算图（Dynamic Computational Graphs），这意味着你可以在运行时改变你的模型。这在研究和开发新的模型时尤其有用，因为你可以逐步调整你的模型并立即看到结果。
2. **直观的接口**：PyTorch 被设计为尽可能接近原生 Python。其 API 设计直观且易于使用，这使得开发者能够更容易理解他们的代码并更快地进行原型设计。
3. **强大的加速**：PyTorch 支持使用 GPU 进行加速，这使得其可以执行大规模的并行运算，对于深度学习任务尤其重要。此外，PyTorch 还支持分布式计算，因此可以处理大型数据集和复杂模型。
4. **扩展性和可交互性**：PyTorch 的扩展性良好，易于与其他 Python 库和软件包（如 NumPy 和 SciPy）进行交互。此外，PyTorch 还拥有一个活跃的社区，提供了大量的预训练模型和扩展库。
5. **工具和库**：PyTorch 提供了一系列的工具和库来帮助开发者更好地开发和部署他们的应用，如用于模型训练和验证的 Torchvision、用于自然语言处理的 Torchtext，以及用于生成模型的 Torchgan 等。

PyTorch 的核心是张量（tensor），它类似于 NumPy 的 ndarray，但可以在 GPU 上运行来加速计算。PyTorch 还提供了自动求导系统（autograd）来自动计算张量的梯度，这在构建和训练神经网络时非常有用。

[4]JAX 是一个开源的 Python 库，由 Google 开发，用于高性能的数值计算。JAX 对 NumPy、SciPy 和 Autograd 的 API 进行了扩展，从而能够提供高效的硬件加速（GPU 和 TPU）和自动微分功能，这对深度学习和科学计算等任务非常有用。以下是关于 JAX 的一些关键特性：

1. **NumPy 兼容**：JAX 提供了一种接口，其语法与 NumPy 完全一致。这使得 NumPy 用户可以无缝地迁移到 JAX，并立即享受到硬件加速和自动微分等功能的优势。
2. **硬件加速**：JAX 可以利用现代硬件的计算能力，包括 GPU 和 TPU，从而实现大规模并行计算。这对于大数据集和复杂的数值计算任务（如深度学习）非常有用。
3. **自动微分**：JAX 提供了自动微分（autograd）的功能，这使得用户可以轻松地计算函数的梯度，无需手动进行复杂的数学推导。自动微分是许多机器学习算法，尤其是神经网络，的核心组件。
4. **即时编译**：JAX 通过 XLA 编译器，可以将 Python 代码即时（JIT）编译为高效的机器代码，从而实现更快的执行速度。
5. **函数变换**：JAX 提供了强大的函数变换工具，如 vmap 和 pmap，用于自动地对函数进行向量化和并行化。

在深度学习和科学计算等领域，JAX 的这些特性使其成为一个非常有力的工具。

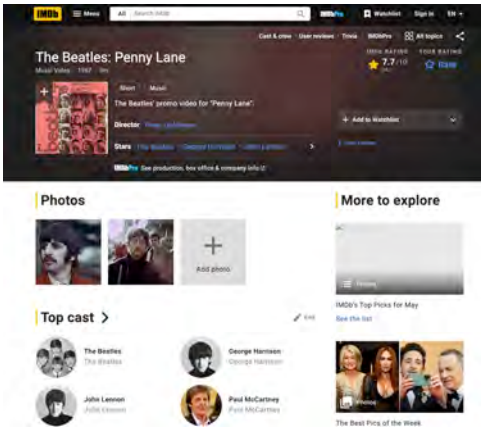


图 B.3: ”Penny Lane” 是英国摇滚乐队 The Beatles 的一首歌曲，由 Paul McCartney 主要创作并领唱，但与习惯一样，该曲以 Lennon-McCartney 的名义发表。这首歌被收录在 1967 年的专辑《Magical Mystery Tour》中。”Penny Lane” 于 1967 年 2 月作为双 A 面单曲发行，另一面是”Strawberry Fields Forever”。这两首歌的发行标志着披头士从传统的流行音乐形式转向更复杂、艺术化的音乐创作。”Penny Lane” 的歌词描述了 McCartney 童年时在利物浦的 Penny Lane 地区的生活，包括那里的银行、消防车、理发店等生动的日常生活细节。其中的角色和场景使得这首歌具有一种迷人的怀旧感。在音乐上，”Penny Lane” 融合了流行、古典和前卫元素，其中包括钢琴、口琴、手铃、小号等众多乐器，展现了 The Beatles 的音乐创新和实验精神。尤其值得一提的是，这首歌的结尾部分有一段由 David Mason 演奏的独奏小号，这段独奏被誉为 Beatles 歌曲中的经典瞬间。”Penny Lane” 在发行后获得了广泛的好评，它在许多国家和地区的单曲榜单上都取得了前列的成绩。在不少音乐评论家和乐迷的心中，这首歌被认为是 The Beatles 最优秀的作品之一，并对后世的流行音乐产生了深远影响。图片来源地址：`https://www.imdb.com/title/tt7593766/`

算，包括矩阵运算和张量操作，这些在量子计算中经常出现。通过使用 GPU，可以加快量子计算任务的执行速度，特别是对于涉及大规模量子系统的计算，可以显著提高性能。PennyLane-Lightning-GPU 插件构建在 PyTorch-Lightning 框架之上，它提供了对 GPU 加速的支持。PyTorch-Lightning 是一个用于训练深度学习模型的高级框架，它简化了模型训练过程中的许多常见任务，并提供了分布式训练和 GPU 加速的功能。PennyLane-Lightning-GPU 利用 PyTorch-Lightning 框架的 GPU 加速功能，将其应用于 PennyLane 的量子计算任务中。使用 PennyLane-Lightning-GPU 插件，用户可以利用 GPU 的并行计算能力来加速量子神经网络的训练和执行。这对于处理复杂的量子任务、大规模的量子系统和需要大量计算资源的量子计算任务非常有用。通过利用 GPU 加速，可以显著缩短量子计算任务的执行时间，提高效率和性能。

最后，NVIDIA 的 cuQuantum SDK 是一套用于加速量子计算工作流的优化库和工具。这套工具包的出现，是为了帮助科学家、开发者和研究人员在经典计算机上模拟量子电路，这对于我们向量子计算的进步至关重要。cuQuantum 的核心特性和优势包括：灵活性：cuQuantum 支持各种量子电路模拟方法，包括状态向量法和张量网络法。状态向量法的特性包括优化的内存管理和数学核心、效率索引位交换、门应用核心，以及对 qubit 集合的概率数组计算。张量网络法的特性包括加速的张量和张量网络收缩、顺序优化、近似收缩和多 GPU 收缩。可扩展性：cuQuantum 可以利用最新 GPU 的多节点、多 GPU 集群的力量，无论是在本地还是在云中。它提供了低级别的 C++ API，用于提供对单个 GPU 和单节点多 GPU 集群的更多控制和灵活性，以及高级别的 Python API，支持无缝的多节点执行。速度：使用 NVIDIA 的 A100 Tensor Core GPU 可以比 CPU 实现在关键量子问题上实现数量级的速度提升，包括随机量子电路、Shor’s 算法和变分量子本征求解器。利用 NVIDIA Selene 超级计算机，cuQuantum 在不到 10 分钟的时间内生成了 Google Sycamore 处理器全电路模拟的样本。集成框架：cuQuantum 与主要的量子电路模拟框架集成，用户可以下载 cuQuantum，以极大地提高所选框架的性能，而无需更改代码。

B.2.3. 项目的简介

在这个项目中，团队考虑模拟噪声量子算法的挑战，这些算法已知需要大量的计算资源。我们通过利用 Xanadu [5]的 PennyLane-Lightning-GPU 和 NVIDIA 的 cuQuantum SDK 中可用的 GPU 工具来解决这个问题，这些工具使团队能够扩大模拟的规模，并对噪声对量子算法的影响有更深入的了解。项目分析揭示了噪声对模拟的一般效应，并确定了可以加速开放量子系统和基态优化模拟的领域。团队希望通过改工作，能够对如何有效模拟噪声量子算法有更好的理解，而有效模拟噪声量子算法这可能对量子计算和寻找 NISQ 时代的用例产生深远影响。

B.2.4. 公式 (1)：分解 Trotter-Suzuki（项目开展研究的数学基础）

在整个项目中，团队使用了一种被称为 Trotter-Suzuki 分解的量子电路来模拟 Hamiltonian 演化的普遍方法。由于量子门只能由酉矩阵[6]表示，所以目标是将 Hamiltonian [7]分解，使量子电路近似其演化，即：希望找到一系列的量子门（即酉矩阵），它们的作用效果等同于由 Hamiltonian 表示的系统演化[8]。这就是所谓的量子模拟，即通过量子电路模拟真实物理系统的演化过程。这是量子计算的一个重要应用，因为很多物理系统的演化过程非常复杂，难以通过经典计算来模拟，而量子系统则可以有效地模拟这些过程。这种近似是按照如下方式实现的：

$$U(t) \approx \prod_{d=1}^{D} \prod_{n=1}^{N} exp(-iH_n t/D) \tag{B.1}$$

在这个方程中，$U(t)$ 是一个量子系统在时间 t 的演化算符。这个算符描述了系统从初始状态演化到时间 t 的状态的变化。

右边的部分是 Trotter-Suzuki [9]分解的核心。这个分解的思想是将总的演化算符分解为一系列更容易处理的小步骤。具体来说，总的 Hamiltonian H 被分解为一系列的子 Hamiltonian H_n，每

[5]Xanadu 是一家加拿大的量子计算公司，总部位于多伦多。该公司成立于 2016 年，致力于开发和商业化量子技术。

Xanadu 的主要目标是构建基于光子的量子计算和量子通信技术。他们专注于开发和推进一种称为" 连续变量量子计算" 的方法，该方法利用量子态中的连续变量（例如光的强度和相位）来进行计算和通信。Xanadu 的核心技术是基于光的量子计算平台，利用可调谐的光学器件和光学探测器来操作和测量量子态。

Xanadu 的量子计算平台基于开放源代码软件库"Pennylane"，该软件库允许开发者使用 Python 编程语言在量子计算中进行实验和模拟。他们还提供了云端的量子计算服务，使研究人员和开发者能够远程访问和使用他们的量子计算资源。

除了量子计算，Xanadu 还专注于量子机器学习和量子光学通信等领域的研究和开发。他们的目标是将量子技术与机器学习、优化算法和通信领域相结合，为解决复杂问题和提供安全通信解决方案提供创新的解决方案。

Xanadu 在量子计算领域积极参与学术界和产业界的合作，与多个大学、研究机构和公司建立了合作伙伴关系。他们的工作得到了量子计算社区的广泛认可，并在学术会议和期刊上发表了大量的研究成果。

总体而言，Xanadu 是一家致力于光子量子计算和量子通信技术开发的公司，他们通过开放源代码和云端服务的方式，推动量子计算在学术和商业领域的应用和发展。

[6]在量子计算中，所有的量子门（即量子计算的基本操作）都可以用酉矩阵来表示。酉矩阵有一个重要的性质，即其逆矩阵等于其共轭转置，这使得酉矩阵的操作可以保持量子系统的总概率为 1（因为量子态的概率分布是由量子态的向量模长平方给出的，而酉矩阵操作不改变这个模长）。

[7]在物理中，Hamiltonian 是一个系统的总能量算符，它描述了系统的能量结构。在量子力学中，系统的时间演化由 Schrödinger 方程描述，该方程中的主要部分就是 Hamiltonian。由 Schrödinger 方程我们知道，一个量子系统的状态随时间的变化（即系统的时间演化）可以通过其 Hamiltonian 来确定。

[8]量子电路的演化：这是量子计算的基本过程，即通过一系列的量子门操作（这些操作由酉矩阵表示）来改变量子系统的状态。

[9]Trotter-Suzuki 分解是一种用于近似求解量子系统的时间演化的方法。特别是，它用于求解由 Hamiltonian（一个描述系统能量的物理量）决定的 Schrödinger 方程。

在很多情况下，量子系统的 Hamiltonian 可以写成一些子 Hamiltonian 的和，每个子 Hamiltonian 对应于系统的某个物理性质。然而，系统的总体演化（由所有子 Hamiltonian 共同决定）通常很难直接求解。

这就是 Trotter-Suzuki 分解派上用场的地方。这种方法的思想是将总体演化分解成一系列更容易处理的小步骤，每一步只包括一个子 Hamiltonian 的影响。然后，这些步骤被依次应用到系统上。

具体来说，Trotter-Suzuki 分解的公式大致如下：

一个都对应于系统的一个特定的物理性质。然后，这些子 Hamiltonian 的演化被逐一地应用到系统上。

在这个方程中，指数函数 exp$(-iH_nt/D)$ 表示了每个子 Hamiltonian H_n 的演化算符。这个算符描述了在时间 t/D 上，由于 H_n 的影响而产生的系统的变化。这里的 D 是一个正数，表示 Trotter-Suzuki 分解的步骤数量。这个值越大，分解的步骤就越多，近似就越准确。

累乘符号 Π 表示这些步骤被依次应用到系统上。首先应用 H_1 的影响，然后是 H_2，以此类推，直到 H_N。然后这个过程重复 D 次。

这个方程描述了如何用一系列更简单的步骤来近似描述一个量子系统的时间演化。这种方法在量子计算和模拟中非常有用，因为它**可以让团队更容易地在有限的资源下进行复杂的模拟**。所以说这个方程是该团队进行研究的最基础前提。

B.2.5. 公式 (2)：Heisenberg 模型的哈密顿量

$$H = -\sum_{i}^{n}[J_x \cdot (S_i^x \oplus s_{i+1}^x) \tag{B.2}$$

$$+ J_y \cdot (S_i^y \oplus s_{i+1}^y) \tag{B.3}$$

$$+ J_z \cdot (S_i^z \oplus s_{i+1}^z) \tag{B.4}$$

$$+ h \cdot S_i^z]. \tag{B.5}$$

这是 Heisenberg 模型的哈密顿量。这是一种在量子力学和统计力学中经常遇到的模型，用来描述一维的量子自旋系统。

在这个模型中，系统由一系列的量子自旋组成，每个自旋可以在空间的三个方向（x，y，z）上有一个值。这就是符号 $(S_i^x), (S_i^y)$, 和 (S_i^z) 所代表的含义，其中 (i) 是自旋的索引[10]。

$(\oplus)$ 符号表示自旋在相应方向上的相互作用，也就是两个相邻自旋在相应方向上的乘积。

$(J_x), (J_y)$, 和 (J_z) 是交换积分，描述了自旋在 x，y，和 z 方向上的相互作用强度。

(h) 是外加的磁场，它作用在每个自旋的 z 方向上，所以它和 (S_i^z) 的乘积描述了这个外加磁场的效果。

这个哈密顿量描述了一个一维的量子自旋系统在一个外加磁场中的行为，以及自旋之间的相互作用。

在这个研究中，这个哈密顿量可能被用于模拟系统的行为，比如计算系统的基态或者模拟开放量子系统的动力学。同时，这个哈密顿量可能还被用于在模拟中引入噪声，以研究噪声对量子系统行为的影响。

为了理解这个概念，我们需要先了解一下量子比特的状态。一个量子比特的状态可以用一个叫做密度矩阵的物理量来描述。在理想的情况下，量子比特的状态是纯的，即其密度矩阵只有一个

$$U(t) \approx (e^{-iH_1t/n}e^{-iH_2t/n}\cdots e^{-iH_Nt/n})^n$$

在这个公式中，$U(t)$ 是系统的总体演化算符，$H_1, H_2, \ldots, H_N$ 是子 Hamiltonian，t 是时间，n 是步骤的数量。可以看到，每个子 Hamiltonian 的影响被独立地应用到系统上，并且这个过程被重复 n 次。

值得注意的是，Trotter-Suzuki 分解只是一个近似方法。当步骤的数量 n 足够大时，这种近似才会变得足够准确。然而，尽管如此，这种方法在量子计算和模拟中仍然非常有用，因为它可以让我们在有限的资源下进行复杂的模拟。

[10]指用来标记粒子自旋状态的标签或编号。例如，对于一个半整数自旋的粒子（如电子），其自旋状态可以是"+1/2" 或"-1/2"，这些就可以被看作是自旋的索引。另一方面，如果我们有一个多粒子系统，每个粒子都有自己的自旋状态，那么我们可能需要一个索引来区分不同粒子的自旋状态。

非零的特征值。然而，如果存在噪声，那么这个量子比特的状态就可能变得混乱，即其密度矩阵可能会有多个非零的特征值。

Depolarizing noise 的效果就是使得量子比特的状态变得更加混乱。具体来说，它会使得量子比特的密度矩阵向最大混合态的方向偏移，最大混合态是一个特殊的混乱状态，其所有的特征值都是相等的。在这种噪声的影响下，量子比特的状态将变得更加随机，其相干性将被破坏。

Depolarizing noise 是量子计算中的一个主要挑战，因为它会破坏量子计算的精度。因此，研究如何纠正或者抑制这种噪声是当前量子计算研究的一个重要领域。

$$K_1 = \sqrt{1-p}I_2, \tag{B.6}$$

$$K_2 = \sqrt{\frac{p}{3}}X, \tag{B.7}$$

$$K_3 = \sqrt{\frac{p}{3}}Y, \tag{B.8}$$

$$K_4 = \sqrt{\frac{p}{3}}Z. \tag{B.9}$$

这些公式定义了退偏噪声的 Kraus 操作符[11]。在量子力学中，Kraus 操作符是一种描述开放量子系统动力学的工具，特别是当系统与其环境发生相互作用时。它们是用来描述量子通道的，量子通道是一个将初始量子态映射到最终量子态的过程。

在这里，(K_1, K_2, K_3, K_4) 是退偏通道的 Kraus 操作符，其中 (I_2, X, Y, Z) 分别是 2×2 单位矩阵和 Pauli 矩阵，它们在量子力学中代表不同的自旋操作。参数 (p) 是退偏概率，这是噪声强度的一个度量。

这些 Kraus 操作符的作用是在模拟中引入噪声。这意味着，当我们通过这个通道传输一个量子态时，这个态会有一定概率发生位翻转（由 (X) 操作符引起）、相位翻转（由 (Z) 操作符引起）、或者位相位翻转（由 (Y) 操作符引起），而这个概率就是 (p)。

在本研究中，这些操作符可能被用来模拟退偏噪声对量子算法的影响，以便研究噪声如何影响算法的性能和结果。

[11]在量子力学和量子信息理论中，Kraus 操作符是一种用于描述开放量子系统动力学的形式。开放量子系统是指与外部环境发生相互作用的量子系统，这种相互作用会引发一些被称为量子噪声的效应。Kraus 操作符（也称为 Kraus 矩阵或 Kraus 表示）是一种数学工具，可以用来描述这些噪声效应。

Kraus 操作符的形式由一组矩阵 $\{K_i\}$ 组成，这些矩阵需要满足以下的归一化条件：

$$\sum_i K_i^\dagger K_i = I$$

其中，$K_i^\dagger$ 是 K_i 的共轭转置，I 是单位矩阵，和等式左边的矩阵具有相同的维度。

给定一个初始量子态（由密度矩阵 ρ 表示），其在 Kraus 操作符作用下的最终态可以由以下公式给出：

$$\rho' = \sum_i K_i \rho K_i^\dagger$$

这个公式告诉我们，每个 Kraus 操作符 K_i 都会对初始态 ρ 产生一种影响，然后这些影响的总和给出了最终态 ρ'。

Kraus 操作符的这种形式给了我们一种强大的工具，可以用来描述和处理量子噪声和量子衰减等复杂的量子效应。

B.2.6. 公式 (4)：Kraus 操作符作用过程

$$K_1 = R_x(s),$$
$$K_2 = R_y(s),$$
$$K_3 = R_z(s)$$

在这里，(K_1, K_2, K_3) 是 Kraus 操作符，它们描述了一个量子系统如何受到噪声的影响。这些操作符是作用在量子态上的，使得量子态发生变化。

$(R_x(s), R_y(s), R_z(s))$ 是分别围绕 x, y, z 方向旋转的旋转操作符。其中 (s) 是旋转角度。

在这个研究中，这些 Kraus 操作符被用来模拟一个特定类型的噪声，即围绕不同轴旋转的噪声。这种噪声可以引起量子比特的相位偏移，这可能对量子计算有影响。

这些 Kraus 操作符也会被用来模拟噪声如何影响量子算法的性能，从而更好地理解噪声对量子计算的影响，并可能找到对抗或利用这种影响的方法。

B.2.7. 公式 (5)：构建量子态

$$\psi^r = \sum_{i}^{2^n} |\psi_i|^2 e_i \tag{B.10}$$

在这个公式中，(ψ^r) 表示一个量子态，它是基于一组系数 (ψ_i) 和相位因子 (e_i) 的加权和。这个公式描述了量子态的构成方式。

具体来说，(ψ_i) 是第 (i) 个状态的系数，表示该状态在量子态中的权重。而 (e_i) 是与该状态相关的相位因子，用于描述相位角度的贡献。

公式的右侧部分是一个求和符号，表示对所有可能的 (i) 进行求和。对于每个 (i)，我们计算 (ψ_i) 的模的平方 $(|\psi_i|^2)$，这表示第 (i) 个状态的概率。然后将其乘以相应的相位因子 (e_i)。

最后，将所有加权的状态相加，得到量子态 (ψ^r)。这个量子态是由基本状态的概率幅度和相位因子构成的。

B.2.8. 公式 (6)：量子态的平均

$$\psi_{avg} = \frac{1}{R} \sum_{r}^{R} \psi^r \tag{B.11}$$

这个公式描述了对一组量子态 ψ^r 进行平均的过程。

在这里，ψ_{avg} 表示平均后的量子态，它是对一组量子态 ψ^r 进行求和并除以数量 R 后得到的。

具体来说，对于每个量子态 ψ^r，我们将它们相加得到总和，并最后除以数量 R，得到平均值。

这个公式常用于统计性质的计算，通过对多次实验或不同的参数进行重复运算，得到平均结果以减小随机误差或获得更可靠的结果。

在特定的研究中，这个公式可能被用于计算量子态的平均值，从而得到关于量子系统的一般性特征或行为。具体的含义和应用需要根据具体的研究背景和上下文来理解。

B.2.9. 公式 (7)：量子态的权重

$$\psi_{exact} = \sum_{i}^{2^n} i, iei \tag{B.12}$$

在这个公式中，ψ_{exact} 表示一个量子态，它是由一组系数 σ_i 和相位因子 e_i 构成的加权和。这个公式描述了量子态的构成方式。

具体而言，σ_i 是第 i 个状态的系数，表示该状态在量子态中的权重。而 e_i 是与该状态相关的相位因子，用于描述相位角度的贡献。

公式的右侧是一个求和符号，表示对所有可能的 i 进行求和。对于每个 i，我们将 σ_i 乘以相应的相位因子 e_i，然后将它们相加得到总和。

最后，将所有加权的状态相加，得到量子态 ψ_{exact}。这个量子态由基本状态的概率幅度和相位因子构成。

这个公式可能用于描述量子态的精确构建方式，其中的系数和相位因子是根据特定的算法或理论计算得到的。它可以用于比较精确构建的量子态与其他近似方法得到的量子态之间的差异，或者用于研究特定的量子态构建技术。

请注意，具体的含义和应用需要根据具体的研究背景和上下文来理解。

B.2.10. 公式 (8)：评估量子模拟的准确性

$$F(\psi_{avg}, \psi_{exact}) = |\langle\sqrt{\psi_{avg}}, \sqrt{\psi_{exact}}\rangle|^2 \tag{B.13}$$

这个公式描述了两个量子态 ψ_{avg} 和 ψ_{exact} 之间的相似性度量。

在这个公式中，$F(\psi_{avg}, \psi_{exact})$ 表示量子态 ψ_{avg} 和 ψ_{exact} 之间的相似性度量，也被称为 Fidelity。

具体而言，公式中的符号表示了两个量子态之间的内积和模的平方运算。$\langle\sqrt{\psi_{avg}}, \sqrt{\psi_{exact}}\rangle$ 表示将两个量子态分别开根号后进行内积运算。

内积运算用于度量两个量子态之间的相似性。将结果取模的平方 $|\langle\sqrt{\psi_{avg}}, \sqrt{\psi_{exact}}\rangle|^2$ 得到了相似性的度量值。

这个度量值可以用于比较平均量子态 ψ_{avg} 和精确量子态 ψ_{exact} 之间的相似程度。如果结果接近 1，则表示两个量子态非常相似；如果结果接近 0，则表示两个量子态差异很大。

在特定的研究中，这个公式可能被用于评估量子态重构或量子模拟的准确性，或者用于比较不同算法或方法得到的量子态的质量。

请注意，具体的含义和应用需要根据具体的研究背景和上下文来理解。

B.2.11. 量子电路的设计 (1)：三量子比特 Heisenberg 哈密顿量的量子电路

三个量子比特的 Heisenberg 哈密顿量的量子电路，原文图 1，本文图B.2.11。

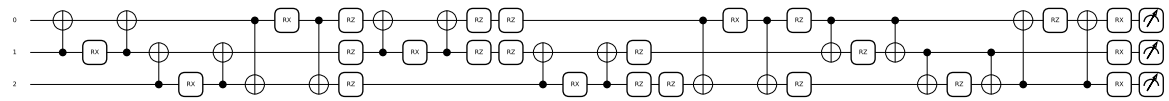


图 B.4: 一个用于模拟三个量子比特的 Heisenberg 哈密顿量的量子电路：一个用于对三个量子比特应用 Heisenberg 哈密顿量进行一次 Trotter 步骤的无噪声模拟的量子电路。该电路与方程式 2 不同，因为我们在 X 方向上使用了一个磁场，而不是 Z 方向上的磁场。

B.2.12. 量子电路的设计 (2)：基本 VQE 算法量子电路

研究涉及的基本 VQE 算法量子电路，原文图 5，本文图B.2.12。

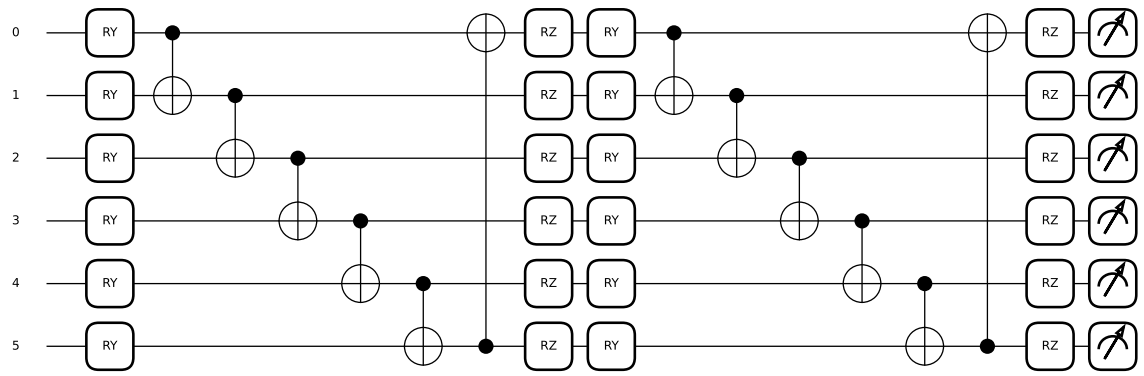


图 B.5: 研究涉及的基本 VQE 算法量子电路：一个用于 VQE 的 6 比特电路的无噪声编码模型。该模型使用两层编码，每层包括一层参数化的 Ry 门、CNOT 门和参数化的 Rz 门。

图B.2.12显示了一个用于 VQE 的 6 比特电路的无噪声编码模型。该模型使用两层编码，每层包括一层参数化的 Ry 门、CNOT 门和参数化的 Rz 门。

通过这种编码方式，可以将输入的信息进行量子态的表示，以便进行后续的计算和优化。在这种特定的电路结构中，通过一系列的 Ry 门、CNOT 门和 Rz 门的组合，将输入的信息编码到 6 比特的量子系统中。

这种编码模型可以用于 VQE 算法，该算法旨在通过变分量子电路来寻找能量期望值的最小值，从而解决量子化学和材料科学等领域的问题。

B.2.13. 量子电路的设计 (3)：带有噪声参数的基本 VQE 算法量子电路

研究涉及带有 0.2 的噪声参数 p 的基本 VQE 算法量子电路，原文图 6，本文图**??**。

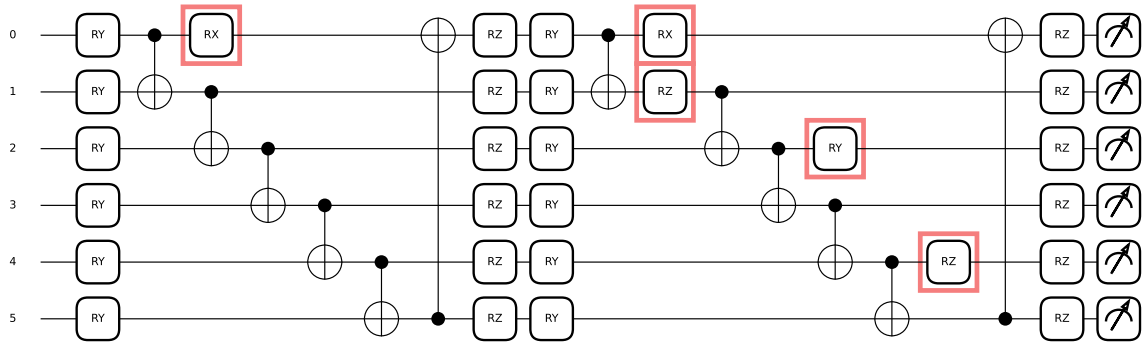
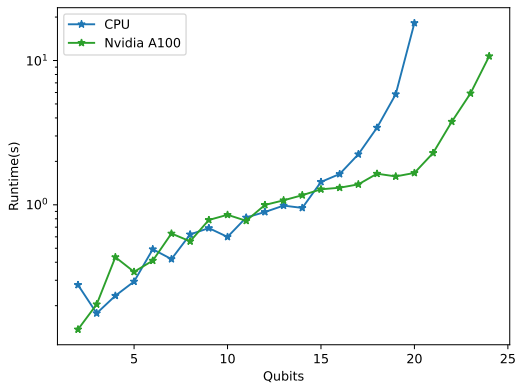
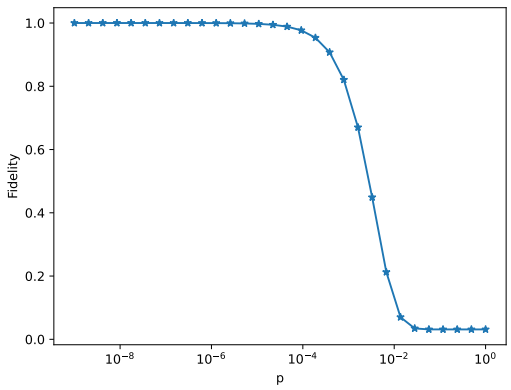


图 B.6: 研究涉及的基本 VQE 算法量子电路：与B.2.12相同的编码模型，但带有 0.2 的噪声参数 p。考虑实际量子计算机的噪声情况下，模拟该编码模型的结果。

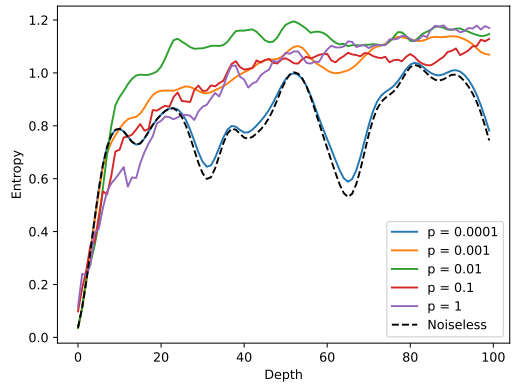
B.2.14. 原图 2，图 3，图 4 和图 7



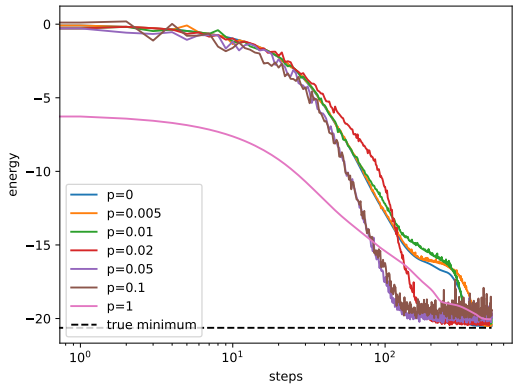
(a) 使用状态向量模拟器对比量子比特数量的扩展性。（原文图 2）



(b) 研究中对噪声强度对准确度的影响进行的分析结果。（原文图 3）



(c) 研究中通过不同噪声强度进行时间演化时的熵变化情况。（原文图 4）



(d) VQE 方法进行优化的横向场海森堡模型收敛速度。（原文图 7）

图 B.7: 研究中使用的图 1、图 3、图 4 和图 7

原文图 2，本文图B.8子图 a，生成程序B.1。

在图中可以看到，当量子比特数量达到约 20 比特时，CPU 的计算需求开始呈指数级增长，而 GPU 可以推进到大约 25 比特。这表明在进行更大规模的状态向量模拟时，使用 GPU 是合理的，尽管指数级增长的情况仍然会出现。

这个结果进一步支持了在大规模状态向量模拟中使用 GPU 的选择。

图 2 代码：

```
import sys
sys.path.append('..\..')
# print(sys.path)
import matplotlib.pyplot as plt
import pickle
import os
from src.tests import calculate_heisenberg_runtime_vs_qubits
from src.plots import plot_runtimes_vs_qubits
```

```
file_name="runtimes_CPU.pickle"
open_file = open(file_name, "rb")
```

```
CPU_times = pickle.load(open_file)
open_file.close()

file_name="runtimes_GPU.pickle"
open_file = open(file_name, "rb")
GPU_times = pickle.load(open_file)
open_file.close()
```

原文图 3，本文图B.7b，程序B.2。

在不同噪声强度下，与理论上无噪声算法相比，准确度的变化情况。通过调整噪声强度参数 p，并计算噪声结果与 PennyLane-Lightning 中的 default.mixed 密度矩阵模拟器之间的准确度，研究了噪声对算法准确度的影响。

原文图 4，本文图B.7c。

在研究中，通过在不同噪声强度下进行时间演化，计算了冯·诺依曼熵（Von Neumann entropy）。这在经典模拟中是很重要的，特别是在考虑使用张量网络等方法时，其中键合维度的增长与量子态的纠缠增长相关 [189]。

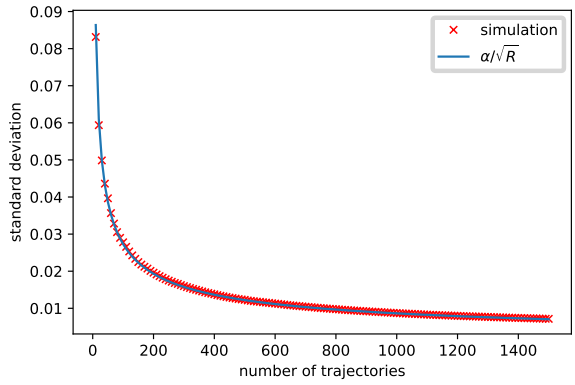
图中的结果展示了不同噪声强度下随时间的演化，对应的熵变化情况。熵的变化可以反映系统的纠缠性质随时间的演化，提供了关于噪声对量子态的影响程度的信息。

原文图 7，本文图B.7d。

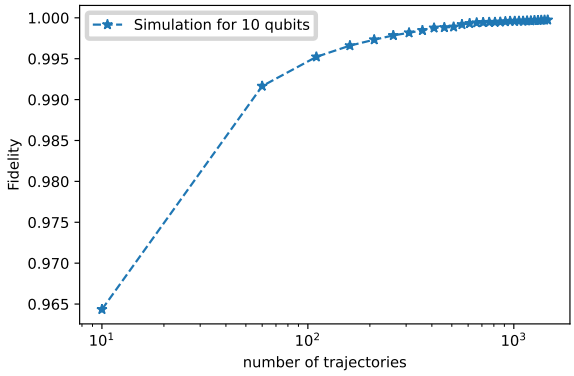
研究通过使用 PennyLane 的 AdamOptimizer 对模型进行优化，直到它收敛或达到最大优化循环次数。在此过程中，使用 $s = 0.2$ 的噪声强度和多个不同的噪声概率 p。为了处理更大的 10 比特系统，使用了 Cyxtera/run:ai 集群上的 NVIDIA A100 GPU 和 lightning.qubit 后端。图中展示了这些结果。

无噪声模型能够逼近真实的本征态能量（通过精确对角化计算）。但有趣的是，尽管高水平的噪声导致不稳定的行为，中等水平的噪声，特别是 $p = 0.05$，在与无噪声运行相比，可以更快且更准确地收敛到基态。此外，对于 $p = 1$ 的模型，噪声被应用在每个 CNOT 门上，因此本质上描述了一个完全不同的模型。这表明某些类型的噪声可以有效地被用作改进 VQE 算法的资源。

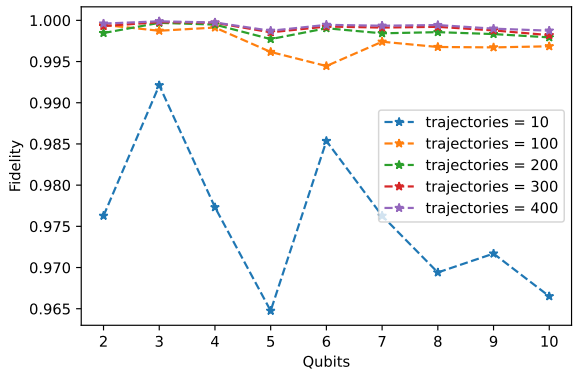
B.2.15. 原图 8，图 9，图 10 和图 11



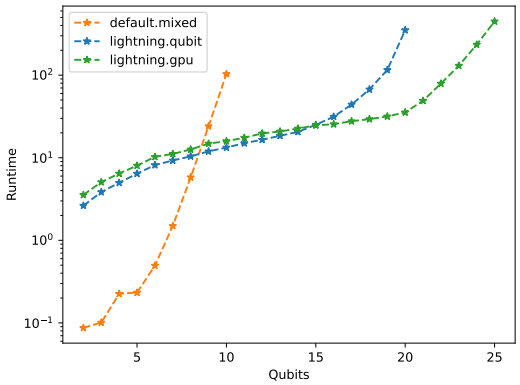
(a) 显示了在一个 **10** 比特系统中，随着轨迹数量的增加，随机噪声方法和精确密度矩阵模拟之间的保真度的标准差。（原文图 8）



(b) 随着轨迹数量的增加，随机噪声算法相对于精确密度矩阵模拟的保真度。（原文图 9）



(c) 在不同轨迹数量下，对高达 **10** 比特系统尺寸的保真度进行比较。（原文图 10）



(d) 在不同量子比特数下，使用不同后端进行模拟的运行时的表现。（原文图 11）

图 B.8: 描述矢量的方法

原文图 8，本文图B.8a。

为了验证方法的有效性，研究了算法收敛性与轨迹数量的关系。由于随机噪声方法本质上是一种蒙特卡洛方法，因此使用了标准差的相关分析 [13]。图中显示的 10 比特系统的结果表明，我们的算法按照预期的 $1/\sqrt{R}$ 的方式收敛，其中 R 是轨迹数量。

为了理解我们的模拟结果并验证其正确性，使用状态向量和精确密度矩阵方法计算了噪声电路的最终状态的保真度。这个计算涉及不同比特数 n，一个包含 10 个 Trotter 步骤的电路，归一化时间 $T = 1$，以及在上述提到的耦合参数下进行。如上所述，噪声概率 p = 0.0033。为了计算保真度，首先取最终状态向量的绝对值的平方，转换为概率向量 ψ^r。

原文图 9，本文图B.8b。

随着轨迹数量的增加，随机噪声算法相对于精确密度矩阵模拟的保真度。在这里，我们采用概率向量的平方根作为内积 $\langle , \rangle$ 的定义方式。图中展示的保真度曲线对应于轨迹数量，这提供了关于达到所需精度所需的独立运行次数的信息。

通过观察图中的保真度曲线，可以得出关于随机噪声算法的收敛性和准确性的见解。随着轨迹数量的增加，保真度逐渐提高，说明增加独立运行次数可以提高算法的准确性。这个曲线可以帮

助确定达到所需精度所需的独立运行次数，从而指导算法的参数选择和优化过程。

原文图 10，本文图B.8c。

通过对不同数量的轨迹进行比较，评估了随机噪声算法的保真度（fidelity）与精确密度矩阵模拟之间的关系。他们希望找到一个适当的轨迹数量，既能够实现高准确度，又能在一系列系统尺寸上提高计算速度，以便进行基于状态向量的方法和密度矩阵方法的运行时比较。

图 10 展示了在不同轨迹数量下，对高达 10 比特系统尺寸的保真度进行比较。通过这个比较，作者决定在运行时基准测试中使用 200 个轨迹。

具体而言，作者使用了一种基于随机噪声的算法来模拟量子系统。该算法使用多个随机轨迹来近似描述量子系统的演化。通过增加轨迹数量，可以提高算法的准确度，但同时也会增加计算的时间复杂度。

在图 10 中，可以看到随着轨迹数量的增加，保真度逐渐提高。对于每个系统尺寸，随着轨迹数量的增加，保真度曲线趋于稳定。作者根据这些比较结果选择了 200 个轨迹作为运行时基准测试的数量，因为在这个数量下，保真度已经达到了一个较高的水平，并且在一定范围内的系统尺寸上可以实现计算速度的提升。

通过选择适当的轨迹数量，作者能够在保证准确度的同时，实现了基于状态向量的方法和密度矩阵方法的运行时比较。这些运行时比较可以帮助评估不同方法在处理量子系统时的计算效率和准确性。

原文图 11，本文图B.8d。

团队对 Heisenberg 时间演化量子电路进行了运行时（runtime）的比较，使用了 Pennylane 提供的三种不同后端。状态向量模拟（statevector simulations）使用了 200 个轨迹。所有的模拟都使用了 10 个 Trotter 步骤，标准化时间为 T=1，耦合参数如第 III 节中所述，并且噪声概率为 p=0.0033。

图B.8d展示了在不同量子比特数下，使用不同后端进行模拟的运行时。对于小型系统（大约小于 9 个量子比特），密度矩阵方法的运行时更好。然而，对于超过 10 个量子比特的系统，密度矩阵方法的指数级增长变得明显，运行时达到了约 102 秒的量级。另一方面，状态向量模拟器由于计算每个轨迹的开销，最初表现较差。然而，对于较大的系统，无论是 statevector simulations 还是，都比密度矩阵模拟器表现更好。

此外，使用 GPU 加速的模拟在性能上优于基于 CPU 的模拟，这使得能够在与密度矩阵方法模拟 10 个量子比特所需的时间相同的时间内模拟最多 25 个量子比特的时间演化。

图 11 代码：首先导入了一些必要的模块和函数，为后续的代码执行和绘图操作提供了所需的依赖和工具。

```
import sys
sys.path.append('..\..')
# print(sys.path)
import matplotlib.pyplot as plt
import pickle
import os
from src.tests import calculate_heisenberg_runtime_vs_qubits
from src.plots import plot_runtime_sampling
```

这段代码的解释：

- `import sys`：这行代码导入了名为 `sys` 的 Python 模块，该模块提供了与 Python 解释器和系统交互的功能。

- `sys.path.append('..:')`：这行代码将相对路径 `'..:'` 添加到系统路径列表中。这样做是为了将上级目录添加到模块搜索路径中，以便导入其他模块或包。
- `import matplotlib.pyplot as plt`：这行代码导入了名为 `matplotlib.pyplot` 的模块，并将其命名为 `plt`。`matplotlib.pyplot` 是 Matplotlib 库中用于绘图的模块，通过引入它，可以使用其中的绘图功能。
- `import pickle`：这行代码导入了名为 `pickle` 的模块，该模块提供了 Python 对象的序列化和反序列化功能，用于将对象转换为字节流或从字节流中恢复对象。
- `import os`：这行代码导入了名为 `os` 的模块，该模块提供了访问操作系统功能的方法，例如文件和目录操作。
- `from src.tests import calculate_heisenberg_runtime_vs_qubits`:这行代码从名为 `src.tests` 的模块中导入了 `calculate_heisenberg_runtime_vs_qubits` 函数。这个函数是定义在 `src.tests` 模块中的，用于计算海森堡模型在不同量子比特数上的运行时间。
- `from src.plots import plot_runtime_sampling`：这行代码从名为 `src.plots` 的模块中导入了 `plot_runtime_sampling` 函数。这个函数是定义在 `src.plots` 模块中的，用于绘制运行时间的抽样图。

```
file_name="times_dm"
open_file = open(file_name, "rb")
times_dm = pickle.load(open_file)
open_file.close()

file_name="times_q_cpu"
open_file = open(file_name, "rb")
times_q_cpu = pickle.load(open_file)
open_file.close()

file_name="times_q_GPU"
open_file = open(file_name, "rb")
times_q_GPU = pickle.load(open_file)
open_file.close()
```

这段代码的解释：

- `file_name="times_dm"`：这行代码将字符串 `"times_dm"` 赋值给变量 `file_name`。该变量用于指定要打开的文件的文件名。
- `open_file = open(file_name, "rb")`：这行代码使用文件名 `file_name` 创建一个文件对象，并将其赋值给变量 `open_file`。通过指定模式 `"rb"`，文件以二进制模式只读打开。
- `times_dm = pickle.load(open_file)`：这行代码使用 `pickle` 模块的 `load` 函数从打开的文件对象 `open_file` 中加载数据，并将其赋值给变量 `times_dm`。`pickle.load()` 函数用于从文件中读取序列化的对象。
- `open_file.close()`：这行代码关闭文件对象 `open_file`，以释放资源并确保文件的正确关闭。

以上过程重复了三次，分别针对不同的文件名和变量名，实现了从文件中加载数据的操作。通过使用 `pickle` 模块，可以方便地将 Python 对象序列化为字节流并保存到文件中，然后再从文件中加载并反序列化为原始对象。这在这段代码中用于加载名为 `"times_dm"`、`"times_q_cpu"` 和

"times_q_GPU" 的文件中的数据，并分别将它们保存到 times_dm、times_q_cpu 和 times_q_GPU 变量中。

```
plot_runtime_sampling(times_dm, times_q_cpu, times_q_GPU)
```

这段代码的解释：

plot_runtime_sampling(times_dm, times_q_cpu, times_q_GPU) 是调用了名为 plot_runtime_sampling 的函数，并传入了三个参数：times_dm、times_q_cpu 和 times_q_GPU。这个函数的作用是绘制运行时间的抽样图。

根据代码的上下文来看，plot_runtime_sampling 函数应该是一个自定义的函数，可能定义在之前导入的某个模块中。它接受三个运行时间数据作为参数，并根据这些数据生成相应的抽样图。具体的实现细节需要查看 plot_runtime_sampling 函数的定义或者相关文档。

总结起来，这段代码调用了一个用于绘制运行时间抽样图的函数，并传入了三个运行时间数据作为参数。通过这个函数的执行，可以生成对比不同运行时间数据的抽样图，从而直观地比较它们之间的差异或趋势。

B.2.16. 程序部分 (1)：门的设计

首先倒入所需库

```
import pennylane as qml
import pennylane.numpy as np

from .noise_models import apply_depolarizing
```

对代码的解释：

- import pennylane as qml：这行代码导入了名为 pennylane 的 Python 库，并将其命名为 qml。PennyLane 是一个用于量子机器学习和量子计算的开源库，提供了量子计算的工具和接口。
- import pennylane.numpy as np：这行代码导入了名为 pennylane.numpy 的模块，并将其命名为 np。该模块是 PennyLane 中对 NumPy 的封装，提供了与 NumPy 类似的数组和数学函数，用于在量子计算中进行数值计算。
- from .noise_models import apply_depolarizing：这行代码从当前目录下的 noise_models 模块中导入了 apply_depolarizing 函数。这个函数是定义在 noise_models 模块中的，用于在量子比特上应用退化噪声模型。使用 . 表示相对导入，表示该模块与当前脚本文件在同一目录下。

这段代码导入了 PennyLane 库和 NumPy 模块，并从本地的 noise_models 模块中导入了 apply_depolarizing 函数。这些导入语句为后续的量子门函数提供了必要的依赖和工具。

在本项研究中，团队设计了 4 种门 (以函数) 来验证应用退化噪声模型。分别是：

- 函数 R_XX(angle, wire, num_wires, p, backend)：

```
def R_XX(angle, wire, num_wires, p, backend):
    qml.CNOT([wire,(wire+1)%num_wires][::-1])
    apply_depolarizing(wire, p, backend)
    qml.RX(angle, wires=(wire+1)%num_wires)
    qml.CNOT([wire,(wire+1)%num_wires][::-1])
    apply_depolarizing(wire, p, backend)
```

该函数实现了一个 R_XX 门。R_XX 门是一种双量子比特门，作用于两个量子比特的 X 基础上。它通过在两个量子比特之间添加一对 CNOT 门和 RZ 旋转门来实现。该门用于在量子计算中进行相互作用。

- 函数 R_YY(angle, wire, num_wires, p, backend):

```
def R_YY(angle, wire, num_wires, p, backend):
    qml.RZ(np.pi/2, wires=wire)
    qml.RZ(np.pi/2, wires=(wire+1)%num_wires)
    qml.CNOT([wire,(wire+1)%num_wires][::-1])
    apply_depolarizing(wire, p, backend)
    qml.RX(angle, wires=(wire+1)%num_wires)
    qml.CNOT([wire,(wire+1)%num_wires][::-1])
    apply_depolarizing(wire, p, backend)
    qml.RZ(-np.pi/2, wires=wire)
    qml.RZ(-np.pi/2, wires=(wire+1)%num_wires)
```

该函数实现了一个 RYY 门。RYY 门是一种双量子比特门，作用于两个量子比特的 Y 基础上。它通过在两个量子比特之间添加一对 CNOT 门、RZ 旋转门和 RX 旋转门来实现。该门用于在量子计算中进行相互作用。

- 函数 R_ZZ(angle, wire, num_wires, p, backend):

```
def R_ZZ(angle, wire, num_wires, p, backend):
    qml.CNOT([wire,(wire+1)%num_wires])
    apply_depolarizing(wire, p, backend)
    qml.RZ(angle, wires=(wire+1)%num_wires)
    qml.CNOT([wire,(wire+1)%num_wires])
    apply_depolarizing(wire, p, backend)
```

该函数实现了一个 RZZ 门。RZZ 门是一种双量子比特门，作用于两个量子比特的 Z 基础上。它通过在两个量子比特之间添加一对 CNOT 门和 RZ 旋转门来实现。该门用于在量子计算中进行相互作用。

- 函数 R_X(angle, wire):

```
def R_X(angle, wire):
    qml.RX(angle, wires=wire)
```

该函数实现了一个 RX 门。RX 门是一种单量子比特门，作用于单个量子比特的 X 基础上。它通过在指定的量子比特上应用 RX 旋转门来实现。该门用于在量子计算中对单个量子比特进行旋转操作。

除了以上门操作，代码中还调用了名为 apply_depolarizing 的函数，该函数用于在量子比特上应用退化噪声模型。

B.2.17. 程序部分 (2)：模拟海森堡模型的演化和构建用于优化的 VQE 变分态模型

```
from .gates import R_XX, R_YY, R_ZZ, R_X
import pennylane as qml


def simulate_heisenberg_model(num_wires, couplings, T, depth, p=0, backend='lightning.qubit'):
    """该QNode在海森堡哈密顿的Trotter近似下，返回自旋链在时间t演化后的最终态。

```

```
参数:
couplings (列表[float]):
一个长度为4的数组, 按顺序包含耦合常数和磁场强度[J_x, J_y, J_z, h]。
p (浮点数): 每个CNOT门后的去极化概率。
depth (整数): Trotter化深度。
time (浮点数): 状态演化的时间。

返回:
 (numpy.tensor): 演化后的量子态。
    """
    angle_XX = -2*couplings['J_xx']*T/depth
    angle_YY = -2*couplings['J_yy']*T/depth
    angle_ZZ = -2*couplings['J_zz']*T/depth
    angle_X = -2*couplings['h']*T/depth

    for _ in range(depth):
        for wire in range(num_wires):
            R_XX(angle_XX, wire, num_wires, p, backend)
        for wire in range(num_wires):
            R_YY(angle_YY, wire, num_wires, p, backend)
        for wire in range(num_wires):
            R_ZZ(angle_ZZ, wire, num_wires, p, backend)
        for wire in range(num_wires):
            R_X(angle_X, wire)


def simulate_heisenberg_model_single_timestep(num_wires, couplings, dt, p, backend):
    """该QNode在演化时间t下, 根据海森堡哈密顿的Trotter近似, 返回自旋链的最终状态。

参数:
couplings (列表[float]):
一个长度为4的数组, 按顺序包含耦合常数和磁场强度[J_x, J_y, J_z, h]。
p (浮点数): 每个CNOT门后的去极化概率。
depth (整数): Trotter近似的深度。
time (浮点数): 状态演化的时间。

返回:
 (numpy.tensor): 演化后的量子态。
    """
    angle_XX = -2*couplings['J_xx']*dt
    angle_YY = -2*couplings['J_yy']*dt
    angle_ZZ = -2*couplings['J_zz']*dt
    angle_X = -2*couplings['h']*dt

    for wire in range(num_wires):
        R_XX(angle_XX, wire, num_wires, p, backend)
    for wire in range(num_wires):
        R_YY(angle_YY, wire, num_wires, p, backend)
    for wire in range(num_wires):
        R_ZZ(angle_ZZ, wire, num_wires, p, backend)
    for wire in range(num_wires):
        R_X(angle_X, wire)


def create_vqe_ansatz(params, wires):
    """
 “VQE优化的变分电路
```

```
注意: 参数params的长度为4*wires
参数:
params (numpy.array): 用于变分电路的参数
H (qml.Hamiltonian): 用于计算期望值的哈密顿量

返回:
 (float): 相对于哈密顿量H的期望值"
    """
    for i in range(wires):
        qml.RY(params[i], wires=i)

    for i in range(wires):
        qml.CNOT([i,(i+1)%wires])
    for i in range(wires, 2*wires):
        qml.RZ(params[i+4], wires=i)

    for i in range(2*wires, 3*wires):
        qml.RY(params[i+8], wires=i)

    for i in range(wires):
        qml.CNOT([i,(i+1)%wires])

    for i in range(3*wires, 4*wires):
        qml.RZ(params[i+12], wires=i)
```

这段代码是一个用于模拟海森堡模型的量子计算程序。它使用了 PennyLane 库来实现量子计算。

首先,代码中导入了一些量子门操作,包括 `R_XX`、`R_YY`、`R_ZZ` 和 `R_X`。这些操作是通过从 `.gates` 模块中导入的。

接下来，代码定义了两个函数：`simulate_heisenberg_model` 和 `simulate_heisenberg_model_single_timestep`。

`simulate_heisenberg_model` 函数用于在 `T` 时间内使用 Trotter 近似计算海森堡哈密顿量的指数演化。函数接受一些参数，包括量子比特数目 `num_wires`、耦合常数 `couplings`、深度 `depth`、概率 `p` 和后端 `backend` 等。函数首先计算了各个角度 `angle` 的值，然后使用嵌套的循环进行演化计算。对于每个深度 `depth`，对于每个量子比特 `wire`，依次应用 `R_XX`、`R_YY`、`R_ZZ` 和 `R_X` 门。

`simulate_heisenberg_model_single_timestep` 函数与 `simulate_heisenberg_model` 类似，但是它只模拟了一个时间步长 `dt` 的演化。函数的参数和内部操作类似于 `simulate_heisenberg_model`，只是没有深度的循环。

最后，代码定义了一个 `create_vqe_ansatz` 函数，用于创建用于优化的 VQE（Variational Quantum Eigensolver）变分态。该函数接受一组参数 `params` 和量子比特数目 `wires`，然后构建了一个变分量子电路。变分量子电路使用了 `RY`、`CNOT` 和 `RZ` 门，其中 `RY` 门作用于前 `wires` 个量子比特，`CNOT` 门用于连接量子比特，`RZ` 门作用于第 `wires` 到 `2*wires` 个量子比特，`RY` 门作用于第 `2*wires` 到 `3*wires` 个量子比特，`CNOT` 门再次用于连接量子比特，最后 `RZ` 门作用于第 `3*wires` 到 `4*wires` 个量子比特。

这些函数用于模拟海森堡模型的演化和构建用于优化的 VQE 变分态。

```
import pennylane as qml
import pennylane.numpy as np

def create_ising(h, wires):
```

```
    """
创建Ising模型的哈密顿量

参数:
h (浮点数): 磁场强度
wires (整数): 电路中的线路/量子比特数量

返回:
 (qml.Hamiltonian): Ising模型的哈密顿量
    """
    couplings = [-h]
    ops = [qml.PauliX(wires-1)]

    for i in range(wires-1):
        couplings = [-h] + couplings
        ops = [qml.PauliX(i)] + ops

    for i in range(wires):
        couplings = [-1] + couplings
        ops = [qml.PauliZ(i)@qml.PauliZ((i+1)%wires)] + ops

    return qml.Hamiltonian(couplings,ops)

def create_heisenberg(params, wires):
    """
创建Heisenberg模型的哈密顿量

参数:
couplings (列表[float]):
包含Heisenberg哈密顿量中的J_x、J_y、J_z和h参数的列表
wires (整数): 电路中的线路/量子比特数量

返回:
 (qml.Hamiltonian): Heisenberg模型的哈密顿量
    """

    couplings = [-params[-1]]
    ops = [qml.PauliX(wires-1)]

    for i in range(wires-1):
        couplings = [-params[-1]] + couplings
        ops = [qml.PauliX(i)] + ops

    for i in range(wires):
        couplings = [-params[-2]] + couplings
        ops = [qml.PauliZ(i)@qml.PauliZ((i+1)%wires)] + ops

    for i in range(wires):
        couplings = [-params[-3]] + couplings
        ops = [qml.PauliY(i)@qml.PauliY((i+1)%wires)] + ops

    for i in range(wires):
        couplings = [-params[0]] + couplings
        ops = [qml.PauliX(i)@qml.PauliX((i+1)%wires)] + ops

    return qml.Hamiltonian(couplings,ops)
```

这段代码定义了两个函数：create_ising 和 create_heisenberg，用于创建 Ising 模型和 Heisenberg 模型的哈密顿量。

create_ising 函数用于创建 Ising 模型的哈密顿量。它接受两个参数：磁场强度 h 和电路中的量子比特数量 wires。函数首先初始化了 couplings 列表和 ops 列表。然后，使用循环将 qml.PauliX 和 qml.PauliZ 操作添加到 ops 列表中，并在每次循环中更新 couplings 列表。最后，使用 couplings 和 ops 列表创建一个 qml.Hamiltonian 对象，并将其返回作为函数的结果。

create_heisenberg 函数用于创建 Heisenberg 模型的哈密顿量。它接受两个参数：params 和 wires。params 是一个包含 J_x、J_y、J_z 和 h 参数的列表，用于描述 Heisenberg 哈密顿量。函数首先初始化了 couplings 列表和 ops 列表。然后，使用循环将 qml.PauliX、qml.PauliZ 和 qml.PauliY 操作添加到 ops 列表中，并在每次循环中更新 couplings 列表。最后，使用 couplings 和 ops 列表创建一个 qml.Hamiltonian 对象，并将其返回作为函数的结果。

这些函数使用了 PennyLane 库提供的量子操作和哈密顿量对象。它们用于定义和创建相应模型的哈密顿量，以便在量子计算中使用。

```
import matplotlib.pyplot as plt
import pennylane as qml
import pennylane.numpy as np

def draw_circuit_nice(function:qml.qnode, **kwargs):
    "draw␣quantum␣function␣from␣pennylane␣in␣a␣nice␣visual␣format"
    qml.drawer.use_style("black_white")
    plt.figure(dpi=400)
    fig, ax = qml.draw_mpl(function)(**kwargs)
    plt.savefig(f"./final_plots/circuit.pdf", format="pdf", bbox_inches='tight')
    plt.show()


def calculate_error(approx, exact):
    "approx:␣sv,␣exact:␣dm"
    #sigma_exact = np.sqrt(np.diag(exact))
    #error = 1 - np.abs(np.vdot(approx,sigma_exact))**2

    return  abs(1 - qml.math.fidelity(approx, exact))


def calculate_mean(obs:np.ndarray, vectors:list[np.ndarray]):
    """
计算 F(vectors)=fidelity 的平均值（期望值）。具体计算细节请参见 `pennylane.math.fidelity`。

参数:
obs (np.ndarray): 用于计算平均值和偏差的可观察量。这是从密度矩阵演化得到的真实密度矩阵。

返回值:
可观察量的期望值。
    """
    R = len(vectors)
    m = 0
    for v in vectors:
        m += qml.math.fidelity(obs,v)
    return m/R


def calculate_mean_paper(obs:np.ndarray, vectors:list[np.ndarray]):
```

```
    s = np.zeros_like(vectors[0])
    R = len(vectors)
    for v in vectors:
        s += abs(v)**2
    phi = s/R
    sigma_exact = np.diag(obs)
    f = abs(np.vdot(np.sqrt(phi), np.sqrt(sigma_exact)))**2
    return f


def calculate_avg_dms(vectors:list[np.ndarray]):
    r"calculate␣average␣density␣matrix␣for␣list␣of␣\ket{phi^r}"

    R = len(vectors)
    d = len(vectors[0])
    rho = np.zeros((d,d), dtype=complex)
    print(rho.shape)
    for v in vectors:
        rho += np.outer(v, v.conj())
    rho_avg = (rho/R)
    return rho_avg


def calculate_std(obs:np.ndarray, vectors:list[np.ndarray], means=None, bias:bool=False, mode=str):
    """
计算可观察量 F(vectors)=fidelity 的标准偏差。

参数:
obs (np.ndarray): 用于计算平均值和偏差的可观察量。这是从密度矩阵演化得到的真实密度矩阵。
bias (Optional(bool)): 决定计算中分母的因子。默认为 False -> R(R-1)（无偏），否则为 R（有偏）。

返回值:
可观察量的标准偏差。
    """

    R = len(vectors)
    if not bias:
        if R == 1:
            factor = 1
        else:
            factor = R*(R-1)
    else:
        factor = R

    if not means:
        if mode=='normal':
            mean = calculate_mean(obs=obs, vectors=vectors)
        if mode=='paper':
            mean = calculate_mean_paper(obs=obs, vectors=vectors)
    s = 0
    for v in vectors:
        if mode=='normal':
            s += (qml.math.fidelity(obs,v) - mean)**2
        if mode=='paper':
            phi = abs(v)**2
            rho_exact = np.diag(obs)
            f =  abs(np.vdot(np.sqrt(phi), np.sqrt(rho_exact)))**2
```

```
            s += ( f - mean)**2
    return np.sqrt(s/factor)


def calculate_std_vs_ntraj(obs:np.ndarray, vectors:list[np.ndarray], mode='paper'):
    "states:␣"

    stds = []
    Rmax = len(vectors)
    ntrajs = list(np.arange(0,Rmax+1,10))
    for idx in ntrajs[1:]:
        stds.append(calculate_std(obs=obs,vectors=vectors[:idx], mode=mode))

    exp = 1/np.sqrt(ntrajs[1:])
    scale = exp[1]/stds[1]
    return ntrajs, stds, exp, scale

def calculate_mean_vs_ntraj(obs:np.ndarray, vectors:list[np.ndarray], mode='paper'):
    "states:␣"

    means = []
    Rmax = len(vectors)
    ntrajs = list(np.arange(0,Rmax+1,10))
    for idx in ntrajs[1:]:
        if mode=='paper':
            means.append(calculate_mean_paper(obs=obs,vectors=vectors[:idx]))
        if mode =='normal':
            means.append(calculate_mean(obs=obs,vectors=vectors[:idx]))

    return ntrajs, means
```

这段代码中定义了一些函数，它们主要用于计算量子态矢量和量子态的密度矩阵之间的物理量，并进行可视化展示。这些物理量包括平均值、标准偏差以及它们与轨迹数的关系。

这里首先引入了 `matplotlib.pyplot`、`pennylane` 和 `pennylane.numpy`。

- `matplotlib.pyplot` 是用于创建图形和进行图形可视化的库。
- `pennylane` 是一个用于量子计算的库，它可以与许多流行的机器学习库（如 PyTorch 和 TensorFlow）一起使用。
- `pennylane.numpy` 是 `pennylane` 版本的 `numpy`，它和传统的 `numpy` 类似，但是可以在量子设备上运行。

然后，代码定义了一些函数，这些函数是计算和可视化不同物理量的工具。现在，我将逐一解释这些函数的功能：

1. `draw_circuit_nice`: 这个函数接受一个量子函数，以及可选的关键字参数，并生成这个量子函数的可视化。
2. `calculate_error`: 计算量子态的真实密度矩阵与模拟（或近似）的量子态向量之间的误差。
3. `calculate_mean`: 计算一组量子态向量对于给定的可观察量的平均值（期望值）。
4. `calculate_mean_paper`: 这个函数也计算一组量子态向量对于给定的可观察量的平均值，但采用的方法可能与 `calculate_mean` 函数不同。
5. `calculate_avg_dms`: 计算一组量子态向量的平均密度矩阵。

6. `calculate_std`: 计算一组量子态向量对于给定的可观察量的标准偏差。
7. `calculate_std_vs_ntraj`: 计算一组量子态向量对于给定的可观察量的标准偏差，作为轨迹数量（`ntraj`）的函数。
8. `calculate_mean_vs_ntraj`: 计算一组量子态向量对于给定的可观察量的平均值，作为轨迹数量（`ntraj`）的函数。

这段代码的主要目标是提供一种工具，用于测量和理解量子态向量和密度矩阵之间的相似性和差异。

图

Listing B.1: 绘制运行时间与量子比特数量的关系即原文图二（Scaling of statevector simulator against number of qubits on Cyxtera/run:ai cluster.）、本文中图B.7a的程序代码

```
import matplotlib.pyplot as plt
import os

# 定义一个函数，绘制运行时间与量子比特数量的关系
def plot_runtimes_vs_qubits(num_wires_list, runtimes1, runtimes2):
    script_dir = os.path.dirname(__file__) # 获取脚本的当前目录
    rel_path = "runtime_vs_qubits.pdf" # 指定保存图形的文件名
    abs_file_path = os.path.join(script_dir, rel_path) # 构建文件的完整路径
    plt.plot(range(2, len(runtimes1)+2), runtimes1, label='CPU', marker='*', color='tab:blue') # 绘制CPU的运行时间曲线
    plt.plot(range(2, len(runtimes2)+2), runtimes2, label='Nvidia A100', marker='*', color='tab:green') # 绘制Nvidia A100的运行时间曲线

    plt.xlabel("Qubits") # x轴标签为"Qubits"
    plt.ylabel("Runtime(s)") # y轴标签为"Runtime(s)"
    plt.yscale('log') # y轴使用对数刻度
    plt.legend() # 显示图例
    plt.savefig(abs_file_path, dpi=400) # 保存图形为PDF文件
    plt.show() # 显示图形
```

这段代码定义了一个名为 ‘`plot_runtimes_vs_qubits`‘ 的函数，这个函数主要用于绘制处理不同数量的量子比特（程序中 qubits）所需的运行时间。

这段代码首先导入了 ‘matplotlib.pyplot‘ 和 ‘os‘ 模块：

- ‘`matplotlib.pyplot`‘ 是一个用于生成图形和进行图形可视化的 Python 库。
- ‘os‘ 是一个 Python 模块，提供了一种方便的使用操作系统函数的方式。

‘`plot_runtimes_vs_qubits`‘ 函数接受三个参数：

- ‘`num_wires_list`‘ 参数并未在代码中使用，这可能是代码的一个遗留部分，或者是待使用的部分。
- ‘`runtimes1`‘ 和 ‘`runtimes2`‘ 是两个列表，表示两种不同的计算设备（这里分别是 CPU 和 Nvidia A100 GPU）处理不同数量的量子比特所需的运行时间。

工作流程如下：

1. 首先，该函数获取脚本的当前目录，然后构建一个指向输出文件（这里是 ”`runtime_vs_qubits.pdf`”）的绝对路径。

2. 然后，使用 ‘matplotlib.pyplot‘ 来绘制运行时间与量子比特数量的关系图。这里，x 轴是量子比特的数量（从 2 开始），y 轴是运行时间（以秒为单位）。它分别为 CPU 和 Nvidia A100 GPU 绘制了曲线，并对它们进行了标记和着色。
3. 这个函数设置了图形的标签和图例，并将 y 轴设置为对数尺度。
4. 最后，该函数将图形保存为一个 PDF 文件，然后显示出来。

所以，这个函数主要用于比较 CPU 和 Nvidia A100 GPU 在处理不同数量的量子比特时的性能差异，并利用量化结果生成可视化的图形。

Listing B.2: 绘制保真度与噪声的关系

```
import matplotlib.pyplot as plt  # 导入画图库
import os  # 导入操作系统接口模块

# 定义一个函数，用于绘制保真度与噪声的关系
def plot_fidelities_vs_noise(p_list, fidelities, depth):
    script_dir = os.path.dirname(__file__)  # 获取脚本所在目录
    rel_path = "fidelity_vs_noise.pdf"  # 相对路径
    abs_file_path = os.path.join(script_dir, rel_path)  # 生成绝对路径

    plt.plot(p_list, fidelities, marker='*')  # 绘制折线图
    plt.xscale('log')  # 设置x轴为对数刻度
    plt.xlabel("p")  # x轴标签
    plt.ylabel("Fidelity")  # y轴标签

    plt.savefig(abs_file_path, dpi=400)  # 保存图片
    plt.show()  # 显示图片
```

这段代码定义了一个函数 `plot_fidelities_vs_noise`，这个函数绘制了保真度（fidelity）与噪声参数 p 之间的关系，并将该图像保存为 PDF 文件。具体解释如下：

1. 导入必要的库：`matplotlib.pyplot` 用于绘图，`os` 用于处理文件和目录的路径。
2. 定义函数 `plot_fidelities_vs_noise`，该函数接收三个参数：`p_list`（噪声参数的列表）、`fidelities`（对应的保真度列表）和 `depth`（虽然在这段代码中没有使用）。
3. 在函数内部，首先获取当前 Python 文件的路径（`script_dir`），然后指定相对路径 `rel_path` 为”fidelity_vs_noise.pdf”，最后通过 `os.path.join` 将这两个路径连接起来，生成 PDF 文件的绝对路径 `abs_file_path`。
4. 使用 `matplotlib.pyplot` 的 `plot` 函数绘制保真度与噪声参数的折线图。其中，`p_list` 为 x 轴数据，`fidelities` 为 y 轴数据。图上的数据点以 ‘*‘ 符号表示。
5. 使用 `xscale('log')` 将 x 轴设置为对数刻度。
6. 设置 x 轴标签为”p”，y 轴标签为”Fidelity”。
7. 使用 `savefig` 将绘制的图像保存为 PDF 文件，分辨率设置为 400 dpi。
8. 使用 `show` 显示图像。这通常用于在交互式会话中查看图像，不一定需要在脚本中使用。

注意，代码中 `depth` 参数没有在函数中使用。

工作流程：

这个 Python 脚本定义了一个名为 `plot_fidelities_vs_noise` 的函数，该函数的作用是根据输入的噪声参数列表（`p_list`）和对应的保真度（`fidelities`）来绘制一幅图，图中的 x 轴代

表噪声参数，y 轴代表保真度，然后将这幅图保存为 PDF 文件，并在屏幕上显示出来。下面是详细的工作流程：

1. 首先，代码导入了两个库。matplotlib.pyplot 库是用来绘图的，os 库则是用来处理文件路径的。
2. 然后，定义了一个名为 plot_fidelities_vs_noise 的函数，该函数接受三个参数：p_list，fidelities 和 depth。其中，p_list 是一个列表，包含了所有噪声参数，fidelities 也是一个列表，包含了 p_list 中每一个噪声参数对应的保真度，depth 在这段代码中并没有被用到。
3. 在函数内部，首先获取了当前 Python 文件的目录路径（script_dir），然后设置相对路径（rel_path）为"fidelity_vs_noise.pdf"，接着使用 os.path.join 函数将这两个路径拼接起来，生成了 PDF 文件的绝对路径（abs_file_path）。
4. 然后，调用 plt.plot 函数，根据 p_list 和 fidelities 绘制出一条折线，折线上的每一个点都使用星号 (*) 表示。
5. 接着，使用 plt.xscale 函数将 x 轴的刻度设置为对数刻度，然后分别使用 plt.xlabel 和 plt.ylabel 函数设置 x 轴和 y 轴的标签。
6. 最后，使用 plt.savefig 函数将绘制出的图像保存为 PDF 文件，分辨率为 400 dpi，然后使用 plt.show 函数在屏幕上显示这幅图。

Listing B.3: 绘制保真度与量子位数量的关系

```
import matplotlib.pyplot as plt  # 导入画图库
import os  # 导入操作系统接口模块

# 定义一个函数，用于绘制保真度与量子位数量的关系
def plot_fidelities_vs_qubits(num_wires_list, fidelities, depth):
    script_dir = os.path.dirname(__file__)  # 获取脚本所在目录
    rel_path = "fidelity_vs_wires.pdf"  # 定义相对路径
    abs_file_path = os.path.join(script_dir, rel_path)  # 生成绝对路径

    plt.plot(num_wires_list, fidelities, marker='*')  # 绘制折线图
    plt.xlabel("Wires")  # 设置x轴标签
    plt.ylabel("Fidelity")  # 设置y轴标签

    plt.savefig(abs_file_path, dpi=400)  # 保存图片到指定路径
    plt.show()  # 显示图片
```

这段 Python 代码用于绘制保真度与量子位数量的关系图。以下是对这段代码的详细解释：

1. 首先，我们导入必要的模块：matplotlib.pyplot（用于绘图）和 os（用于处理操作系统相关的操作，比如文件路径）。
2. 然后，我们定义了一个函数 plot_fidelities_vs_qubits，该函数接受三个参数：
   - num_wires_list：表示量子位数量的列表。每个元素都是一个整数，表示一个特定的量子位数量。
   - fidelities：表示对应的保真度的列表。每个元素都是一个浮点数，表示与 num_wires_list 中相应量子位数量对应的保真度。

- `depth`：虽然在这段代码中，这个参数并没有被使用，但在其他上下文中，这个参数可能表示量子电路的深度。

3. 在 `plot_fidelities_vs_qubits` 函数内部，我们首先获取脚本文件所在的目录，然后定义一个相对路径`"fidelity_vs_wires.pdf"`，表示我们将要保存的图像文件的名字。我们使用 `os.path.join` 函数将脚本文件所在的目录和相对路径拼接起来，得到图像文件的绝对路径。
4. 然后，我们使用 `matplotlib.pyplot.plot` 函数绘制保真度与量子位数量的关系图。这个函数的第一个参数是 x 轴的值（即量子位数量），第二个参数是 y 轴的值（即对应的保真度）。我们还设置了一个参数 `marker='*'`，表示在每个数据点处绘制一个星号。
5. 我们使用 `matplotlib.pyplot.xlabel` 和 `matplotlib.pyplot.ylabel` 函数设置了 x 轴和 y 轴的标签。
6. 我们使用 `matplotlib.pyplot.savefig` 函数将绘制的图像保存到之前定义的路径。参数 `dpi=400` 表示图像的分辨率为 400 点每英寸。
7. 最后，我们使用 `matplotlib.pyplot.show` 函数显示绘制的图像。

这段代码的主要功能是根据输入的数据绘制保真度与量子位数量的关系图，并保存和显示这个图像。

这段 Python 代码主要是定义一个函数，用于绘制保真度与量子位数量的关系，并将结果保存为 PDF 文件。这个函数的名称是‘`plot_fidelities_vs_qubits`‘，接收三个参数：一个表示量子位数量的列表（‘`num_wires_list`‘）、一个对应的保真度列表（‘`fidelities`‘）、以及一个名为‘`depth`‘的参数（在这个函数中未使用）。这段代码的工作流程如下：

1. 首先，程序导入了需要的库：‘`matplotlib.pyplot`‘和‘`os`‘。‘`matplotlib.pyplot`‘是用于数据可视化的库，‘`os`‘是用于处理操作系统相关任务的库，如文件和目录路径操作。
2. 程序定义了一个函数‘`plot_fidelities_vs_qubits`‘。在函数内部，首先通过‘`os.path.dirname(__file__)`‘获取了当前 Python 脚本文件的路径（‘`script_dir`‘）。
3. 然后，定义了一个相对路径‘`rel_path`‘，它指向最终要保存的 PDF 文件。通过‘`os.path.join`‘，函数将‘`script_dir`‘与‘`rel_path`‘连接起来，形成一个绝对文件路径‘`abs_file_path`‘。
4. 接下来，函数使用‘`matplotlib.pyplot.plot`‘来绘制保真度与量子位数量的关系图。这里，‘`num_wires_list`‘是 x 轴的数据（量子位数量），‘`fidelities`‘是 y 轴的数据（保真度），每个数据点在图上用星号标记。
5. 函数然后用‘`plt.xlabel`‘和‘`plt.ylabel`‘设置了图像的 x 轴和 y 轴标签。
6. 然后，使用‘`matplotlib.pyplot.savefig`‘函数，将绘制的图像保存到先前定义的路径，保存格式为 PDF，分辨率设为 400 dpi。
7. 最后，函数使用‘`matplotlib.pyplot.show`‘命令来在屏幕上显示图像。

Listing B.4: 绘制错误与量子位数量的关系

```
def plot_error_vs_qubits(num_wires_list, errors, depth, p=0, samples=1, save=True):
    # 绘制错误与量子位数量的关系图
    # 参数:
    #   num_wires_list (list): 量子位数量的列表
    #   errors (list): 对应的错误列表
    #   depth (int): 电路的深度
```

```
    #   p (float, optional): 噪声参数, 默认为0
    #   samples (int, optional): 采样次数, 默认为1
    #   save (bool, optional): 是否保存图像, 默认为True
    #
    # 返回值:
    #   None
    plt.figure(figsize=[10, 8])
    plt.plot(num_wires_list, errors, marker='*', linestyle='--')
    plt.title(f"Runtime vs. Wires (Heisenberg Model, Depth={depth}, p={p}, samples={samples})")
    plt.xlabel("# wires")
    plt.ylabel("Error")
    if save:
        plt.savefig(f"error_p{p}_samples{samples}.pdf", format="pdf", bbox_inches='tight')
    plt.show()
```

这段程序主要用于绘制错误与量子位数量的关系图。以下是对程序的详细解释：

1. 定义了一个函数 ‘plot_error_vs_qubits‘，用于绘制错误与量子位数量的关系图。
2. 参数 ‘num_wires_list‘是一个列表，包含了量子位数量的值。
3. 参数 ‘errors‘是一个列表，包含了对应的错误值。
4. 参数 ‘depth‘是一个整数，表示电路的深度。
5. 参数 ‘p‘是一个浮点数，表示噪声参数，默认值为 0。
6. 参数 ‘samples‘是一个整数，表示采样次数，默认值为 1。
7. 参数 ‘save‘是一个布尔值，表示是否保存图像，默认为 True。
8. 函数内部，首先创建一个图形对象，并设置图形的大小为 10x8 英寸。
9. 使用 ‘plt.plot‘函数绘制错误与量子位数量的关系图。传入 ‘num_wires_list‘作为 x 轴数据，‘errors‘作为 y 轴数据，使用星号作为标记，并使用虚线连接数据点。
10. 使用 ‘plt.title‘函数设置图形的标题，标题中包含了模型名称（Heisenberg Model）、电路深度（depth）、噪声参数（p）和采样次数（samples）的信息。
11. 使用 ‘plt.xlabel‘函数设置 x 轴的标签为” wires”，表示量子位数量。
12. 使用 ‘plt.ylabel‘函数设置 y 轴的标签为”Error”，表示错误的大小。
13. 如果 ‘save‘参数为 True，则使用 ‘plt.savefig‘函数保存绘制的图形为 PDF 格式，并设置文件名为”error_p{p}_samples{samples}.pdf”。使用参数 ‘format="pdf"‘指定保存为 PDF 格式，‘bbox_inches='tight'‘用于修剪图形边界。
14. 最后，使用 ‘plt.show‘函数显示绘制的图形。

这段代码的功能是绘制错误与量子位数量的关系图，并根据给定的参数保存图像。通过图形的标题和标签，可以清晰地表达图形所表示的信息。

详细工作流程：

1. 函数 plot_error_vs_qubits 被定义，接受多个参数：num_wires_list、errors、depth、p、samples 和 save。
2. 函数使用文档字符串（docstring）提供了关于函数功能和参数的说明。
3. 调用 plt.figure(figsize=[10, 8]) 创建一个大小为 10x8 的图像。

4. 使用 `plt.plot(num_wires_list, errors, marker='*', linestyle='--')` 函数绘制错误与量子位数量之间的关系曲线。`num_wires_list` 是量子位数量的列表，`errors` 是对应的错误列表。`marker='*'` 表示使用星号作为数据点的标记，`linestyle='--'` 表示使用虚线作为曲线的样式。
5. 调用

   =depth, p=p, samples=samples)")`plt.title(f"Runtime vs. Wires (Heisenberg Model, Depth`

   `=depth, p=p, samples=samples)")` 设置图像的标题。标题使用了格式化字符串，将 `depth`、`p` 和 `samples` 的值插入到字符串中。
6. 调用 `plt.xlabel("# wires")` 设置 x 轴的标签为`"# wires"`。
7. 调用 `plt.ylabel("Error")` 设置 y 轴的标签为`"Error"`。
8. 使用条件语句 `if save:` 检查 `save` 参数的值。如果为 `True`，则执行以下操作：
   - 调用 `plt.savefig(f"error_pp_samplessamples.pdf", format="pdf", bbox_inches='tight')` 保存图像为 PDF 格式。文件名使用了格式化字符串，将 `p` 和 `samples` 的值插入到字符串中。
9. 调用 `plt.show()` 显示图像。

请注意，在代码中使用 `plt` 之前，需要导入 `matplotlib.pyplot` 模块。你需要确保在代码中添加了 `import matplotlib.pyplot as plt` 语句以便正常工作。

Listing B.5: 绘制保真度与噪声变化关系图

```
def plot_fidelities_vs_noise_changing_depth(p_list, fidelities, depth_list):
    for i, depth in enumerate(depth_list):
        plt.loglog(p_list, fidelities[i], label="Depth␣=␣"+str(depth))  # 使用对数-对数坐标轴绘制保真度与噪声的曲线，标签为"Depth = 深度值"

    plt.title("Fidelity␣vs.␣Noise␣(Heisenberg␣Model)")  # 设置图表标题为"Fidelity vs. Noise (Heisenberg Model)"
    plt.xlabel("p")  # 设置x轴的标签为"p"
    plt.ylabel("Fidelity")  # 设置y轴的标签为"Fidelity"
    plt.legend()  # 显示图例
    plt.show()  # 显示图表
```

这个程序定义了一个名为 ‘`plot_fidelities_vs_noise_changing_depth`‘的函数，该函数用于绘制保真度与噪声变化关系的图表。

函数接受三个参数：‘`p_list`‘，‘`fidelities`‘和 ‘`depth_list`‘。这些参数的含义如下：

- ‘`p_list`‘是一个包含噪声水平的列表。这些噪声水平将被用作 x 轴的值。
- ‘`fidelities`‘是一个二维列表，每个子列表包含了相应噪声水平下的一组保真度值。列表的长度与 ‘`depth_list`‘相同，即每个噪声水平对应一个保真度值列表。
- ‘`depth_list`‘是一个包含不同深度值的列表。这些深度值将用作图例中各曲线的标签。

函数的工作流程如下：

1. 使用 ‘`enumerate()`‘函数迭代 ‘`depth_list`‘中的深度值和其索引。在每次迭代中，深度值被赋值给变量 ‘`depth`‘，索引被赋值给变量 ‘`i`‘。

2. 对于每个深度值，使用‘plt.loglog()‘函数将相应噪声水平(‘p_list‘)和保真度值(‘fidelities[i]‘)绘制在对数-对数坐标轴上。每条曲线的标签为”Depth = 深度值”。
3. 设置图表标题为”Fidelity vs. Noise (Heisenberg Model)”。
4. 设置 x 轴的标签为”p”。
5. 设置 y 轴的标签为”Fidelity”。
6. 调用 ‘plt.legend()‘函数以显示图例。
7. 调用 ‘plt.show()‘函数以显示图表。

这个函数使用 ‘matplotlib‘库来进行绘图。它通过在对数-对数坐标轴上绘制多条曲线来展示不同深度下的保真度与噪声之间的关系。每条曲线对应一个深度值，曲线的形状和趋势可以通过观察图表来了解。图表的标题、轴标签和图例信息都被设置为便于理解和解释。最后，图表被显示出来供用户查看和分析。

Listing B.6: 绘制熵与时间关系图

```
def plot_entropies_vs_time(entropies_list, p_list):
    script_dir = os.path.dirname(__file__)  # 获取当前脚本所在目录的路径
    rel_path = "entropy_vs_time.pdf"  # 设置保存图像的相对路径
    abs_file_path = os.path.join(script_dir, rel_path)  # 获取保存图像的绝对路径
    for i, p in enumerate(p_list):
        if p == 0:
            continue
        plt.plot(entropies_list[i], label="p = "+str(p))  # 绘制熵与时间的曲线，标签为"p = p值"

    plt.plot(entropies_list[0], linestyle='--', color='black', label="Noiseless")  # 绘制无噪声数据
        的曲线，使用虚线样式和黑色颜色
    plt.xlabel("Depth")  # 设置x轴的标签为"Depth"
    plt.ylabel("Entropy")  #设置y轴的标签为"熵"
    plt.legend()  #显示图例
    plt.savefig(abs_file_path, dpi=400)  #保存图像为PDF格式
    plt.show()  #显示图像
```

这段程序是用于绘制熵与时间关系图的函数。下面是它的详细解释：

1. 定义了一个名为 plot_entropies_vs_time 的函数，它接受两个参数：entropies_list 和 p_list，分别表示不同条件下的熵值和参数列表。
2. 使用 script_dir = os.path.dirname(__file__) 获取当前脚本所在目录的路径。
3. 设置变量 rel_path 为”entropy_vs_time.pdf”，表示保存图像的相对路径。
4. 使用 abs_file_path = os.path.join(script_dir, rel_path) 获取保存图像的绝对路径。
5. 使用 for 循环遍历参数列表 p_list 的元素，通过 enumerate 函数获取索引和对应的值。对于 p 值不为 0 的情况：
   - 使用 plt.plot(entropies_list[i], label="p = "+str(p)) 绘制熵与时间的曲线，标签为”p = p 值”。
6. 使用 plt.plot(entropies_list[0], linestyle='--', color='black', label="Noiseless") 绘制无噪声数据的曲线，使用虚线样式和黑色颜色。
7. 使用 plt.xlabel("Depth") 设置 x 轴的标签为”Depth”。
8. 使用 plt.ylabel("Entropy") 设置 y 轴的标签为” 熵”。

9. 使用 `plt.legend()` 显示图例。
10. 使用 `plt.savefig(abs_file_path, dpi=400)` 将图像保存为 PDF 格式，并使用指定的 dpi（每英寸点数）。
11. 使用 `plt.show()` 显示图像。

该函数根据传入的熵值和参数列表绘制出熵与时间之间的关系图像，并保存为 PDF 文件。这段程序的工作流程如下：

1. 定义了一个名为 `plot_entropies_vs_time` 的函数，它接受两个参数：`entropies_list` 和 `p_list`，分别表示不同条件下的熵值列表和参数列表。
2. 使用 `script_dir = os.path.dirname(__file__)` 获取当前脚本所在目录的路径。
3. 设置变量 `rel_path` 为"entropy_vs_time.pdf"，表示保存图像的相对路径。
4. 使用 `abs_file_path = os.path.join(script_dir, rel_path)` 获取保存图像的绝对路径。
5. 使用 `for` 循环遍历参数列表 `p_list` 的元素，通过 `enumerate` 函数获取索引和对应的值。对于 `p` 值不为 0 的情况：
   - 使用 `plt.plot(entropies_list[i], label="p = "+str(p))` 绘制熵与时间的曲线，标签为"p = p 值"。
6. 使用 `plt.plot(entropies_list[0], linestyle='--', color='black', label="Noiseless")` 绘制无噪声数据的曲线，使用虚线样式和黑色颜色。
7. 使用 `plt.xlabel("Depth")` 设置 x 轴的标签为"Depth"。
8. 使用 `plt.ylabel("Entropy")` 设置 y 轴的标签为" 熵"。
9. 使用 `plt.legend()` 显示图例。
10. 使用 `plt.savefig(abs_file_path, dpi=400)` 将图像保存为 PDF 格式，并使用指定的 dpi（每英寸点数）。
11. 使用 `plt.show()` 显示图像。

该函数根据传入的熵值列表和参数列表，绘制出熵与时间之间的关系图像，并将图像保存为 PDF 文件。在图像中，每个参数值对应一条曲线，以及一条无噪声数据的曲线。图像的 x 轴表示深度（Depth），y 轴表示熵（Entropy）。图例显示了每个参数对应的标签，以便进行识别和解释。最后，绘制的图像将以指定的 dpi 保存为 PDF 文件，并在函数执行完成后显示在屏幕上。

Listing B.7: 绘制运行时间与抽样数的图像

```
def plot_runtime_sampling(times_dm, times_q_cpu, times_q_GPU):
    script_dir = os.path.dirname(__file__)  # 获取当前脚本所在目录的路径
    rel_path = "runtime_sampling.pdf"  # 设置保存图像的相对路径
    abs_file_path = os.path.join(script_dir, rel_path)  # 获取保存图像的绝对路径

    plt.plot(range(2, len(times_dm)+2), times_dm, marker='*', linestyle='--', color='tab:orange',
        label=f'default.mixed')  # 绘制 default.mixed 数据的曲线
    plt.plot(range(2, len(times_q_cpu)+1), times_q_cpu[-1], marker='*', linestyle='--', color='tab:
        blue', label=f'lightning.qubit')  # 绘制 lightning.qubit 数据的曲线
    plt.plot(range(2, len(times_q_GPU)+6), times_q_GPU[-1], marker='*', linestyle='--', color='tab:
        green', label=f'lightning.gpu')  # 绘制 lightning.gpu 数据的曲线
    plt.xlabel("Qubits")  # 设置x轴的标签为"抽样数"
    plt.ylabel("Runtime")  # 设置y轴的标签为"运行时间"
```

```
    plt.yscale('log')  # 使用对数刻度
    plt.legend()  # 显示图例
    plt.savefig(abs_file_path, dpi=400)  # 保存图像为PDF格式

    plt.show()  # 显示图像
```

这段程序是用于绘制运行时间与抽样数之间关系图像的函数。下面是它的详细解释：

1. 定义了一个名为 `plot_runtime_sampling` 的函数，它接受三个参数：`times_dm`、`times_q_cpu` 和 `times_q_GPU`，分别表示不同数据的运行时间。
2. 使用 `script_dir = os.path.dirname(__file__)` 获取当前脚本所在目录的路径。
3. 设置变量 `rel_path` 为"runtime_sampling.pdf"，表示保存图像的相对路径。
4. 使用 `abs_file_path = os.path.join(script_dir, rel_path)` 获取保存图像的绝对路径。
5. 使用 `plt.plot` 函数绘制曲线，分别绘制了 `times_dm`、`times_q_cpu` 和 `times_q_GPU` 的数据曲线。
   - `plt.plot(range(2, len(times_dm)+2), times_dm, marker='*', linestyle='--', color='tab:orange', label=f'default.mixed')` 绘制了 `default.mixed` 数据的曲线，使用星号作为标记，虚线作为样式，橙色作为颜色。
   - linestyle='–', color='tab:blue', label=f'lightning.qubit') 绘制了 `lightning.qubit` 数据的曲线，使用星号作为标记，虚线作为样式，蓝色作为颜色。
   - linestyle='–', color='tab:green', label=f'lightning.gpu') 绘制了 `lightning.gpu` 数据的曲线，使用星号作为标记，虚线作为样式，绿色作为颜色。
6. 使用 `plt.xlabel("Qubits")` 设置 x 轴的标签为" 抽样数"。
7. 使用 `plt.ylabel("Runtime")` 设置 y 轴的标签为" 运行时间"。
8. 使用 `plt.yscale('log')` 设置 y 轴使用对数刻度。
9. 使用 `plt.legend()` 显示图例。
10. 使用 `plt.savefig(abs_file_path, dpi=400)` 将图像保存为 PDF 格式，并使用指定的 dpi（每英寸点数）。
11. 使用 `plt.show()` 显示图像。

该函数可以根据传入的数据绘制出运行时间与抽样数之间的关系图像，并保存为 PDF 文件。

Listing B.8: 运行时间与系统熵之间的关系

```
import pennylane as qml
import pennylane.numpy as np
import matplotlib.pyplot as plt

def simulate(num_wires, couplings, p, time, depth):

    def XX(i):
        qml.CNOT([i,(i+1)%num_wires][::-1])
        qml.RX(-2*couplings[0]*time/depth, wires=(i+1)%num_wires)
        qml.CNOT([i,(i+1)%num_wires][::-1])

    def YY(i):
        #YY
        qml.RZ(np.pi/2, wires=i)
```

```
        qml.RZ(np.pi/2, wires=(i+1)%num_wires)
        qml.CNOT([i,(i+1)%num_wires][::-1])
        qml.RX(-2*couplings[1]*time/depth, wires=(i+1)%num_wires)
        qml.CNOT([i,(i+1)%num_wires][::-1])
        qml.RZ(-np.pi/2, wires=i)
        qml.RZ(-np.pi/2, wires=(i+1)%num_wires)

    def ZZ(i):
        #ZZ
        qml.CNOT([i,(i+1)%num_wires])
        qml.RZ(-2*couplings[2]*time/depth, wires=(i+1)%num_wires)
        qml.CNOT([i,(i+1)%num_wires])

    def magnet(i):
        #magnetic field
        qml.RX(-2*couplings[3]*time/depth, wires=i)

    for j in range(depth):
        for i in range(num_wires):
            XX(i)
        for i in range(num_wires):
            YY(i)
        for i in range(num_wires):
            ZZ(i)
        for i in range(num_wires):
            magnet(i)



num_wires_list = [i for i in range(2,6)]
couplings = [1, 2, 1, 0.3]
p = 0.5
time = 1
depth = 100

print("GPU starts")

entropys = np.zeros((len(num_wires_list), depth))

for w, wires in enumerate(num_wires_list):
    print('qubits:', wires)
    dev = qml.device("lightning.qubit", wires=wires) #lightning.gpu

    @qml.qnode(dev)
    def heisenberg_trotter(init_state, couplings, p, time, depth):
        qml.QubitStateVector(init_state, wires=range(wires))
        simulate(wires, couplings, p, time, depth)
        return qml.state()

    dt = time/depth
    init_state = np.zeros(2**wires)
    init_state[0] = 1
    for n in range(depth):
        state = heisenberg_trotter(init_state, couplings, p, time=dt, depth=1)
        entropy = qml.math.vn_entropy(state, indices=[w for w in range(wires//2)])
        entropys[w,n] = entropy
        init_state = state
```

```
72
73 for w in range(len(num_wires_list)):
74     plt.plot(entropys[w,:], label=f'{num_wires_list[w]}␣qubits')
75 plt.title("Runtime␣vs.␣Wires␣(Heisenberg␣Model,␣depth=%d" % depth)
76 plt.xlabel('time')
77 plt.ylabel('entropy')
78 plt.legend()
79 plt.plot()
```

这段程序使用 Pennylane 和 Matplotlib 库来模拟并绘制 Heisenberg 模型中运行时间与系统熵之间的关系图。

首先，程序导入了所需的库和模块：Pennylane（作为 qml）和 Pennylane 的 NumPy 接口（作为 np），以及 Matplotlib（作为 plt）。

然后，程序定义了一个名为 simulate 的函数，该函数用于模拟 Heisenberg 模型的演化。函数接受以下参数：

- num_wires：系统中的量子比特数量。
- couplings：Heisenberg 模型的耦合参数。
- p：概率参数。
- time：模拟的总时间。
- depth：模拟的深度。

在 simulate 函数内部，定义了一系列辅助函数，每个函数对应模型中的不同部分（XX, YY, ZZ 和磁场）。这些函数使用 Pennylane 的量子门操作和旋转操作来模拟相应的演化。

接下来，使用嵌套的循环来执行模拟。外部循环根据给定的深度重复执行一系列模型演化的步骤。内部循环按照量子比特的顺序依次应用 XX、YY、ZZ 和磁场操作。

在模拟过程中，使用了 Pennylane 的 qml.device 函数选择了计算设备（lightning.qubit），并定义了一个名为 heisenberg_trotter 的量子节点。该节点将初始状态、耦合参数、概率、时间和深度作为输入，并在每个深度上调用 simulate 函数模拟系统的演化。最后，函数返回系统的状态。

接下来，程序定义了一些变量，如量子比特数量列表 num_wires_list、耦合参数 couplings、概率参数 p、总时间 time 和深度 depth。这些变量用于定义模拟的参数。

然后，程序创建一个名为 entropys 的零矩阵，用于存储模拟结果中的熵值。矩阵的大小为 len(num_wires_list) 行，depth 列。

接下来，程序通过循环迭代量子比特数量列表 num_wires_list 中的每个值，进行模拟和计算系统的熵值。对于每个量子比特数量，程序创建一个名为 dev 的 Pennylane 设备，并定义了一个名为 heisenberg_trotter 的量子节点。然后，程序通过循环迭代深度的次数，计算每个深度下的系统状态和熵值，并将熵值存储在 entropys 矩阵中。

最后，程序使用 Matplotlib 绘制图表。通过循环迭代量子比特数量列表，程序绘制了每个量子比特数量下的熵值曲线。图表的标题为"Runtime vs. Wires (Heisenberg Model, depth= 深度值)"，x 轴标签为"time"，y 轴标签为"entropy"，并显示图例。最后，调用 plt.plot() 函数显示图表。

这段程序主要用于模拟 Heisenberg 模型中运行时间与系统熵之间的关系，并通过绘制曲线图来可视化结果。

Listing B.9: 程序主要用于计算噪声和无噪声 Heisenberg 模型的最终状态之间的保真度，并创建一个特定形式的哈密顿量

```
import json
import pennylane as qml
import pennylane.numpy as np

num_wires = 4 #increase this

dev = qml.device("default.mixed", wires=num_wires) #lightning.qubit # lightning.gpu

def create_hamiltonian(params):

        couplings = [-params[-1]]
        ops = [qml.PauliX(3)]

        for i in range(3):

            couplings = [-params[-1]] + couplings
            ops = [qml.PauliX(i)] + ops

        for i in range(4):

            couplings = [-params[-2]] + couplings
            ops = [qml.PauliZ(i)@qml.PauliZ((i+1)%4)] + ops

        for i in range(4):

            couplings = [-params[-3]] + couplings
            ops = [qml.PauliY(i)@qml.PauliY((i+1)%4)] + ops

        for i in range(4):

            couplings = [-params[0]] + couplings
            ops = [qml.PauliX(i)@qml.PauliX((i+1)%4)] + ops

        return qml.Hamiltonian(couplings,ops)

@qml.qnode(dev)
def evolve(params, time, depth):

    qml.ApproxTimeEvolution(create_hamiltonian(params), time, depth)

    return qml.state()

def calculate_fidelity(couplings, p, time, depth):
    """该函数返回有噪声和无噪声Trotterization的Heisenberg模型的最终状态之间的保真度，只使用CNOT门和
        旋转门。

参数:

couplings（列表，浮点数）：包含Heisenberg哈密顿量中J_x、J_y、J_z和h参数的列表，如问题陈述中所定义。
p（浮点数）：退极化门作用在每个CNOT门的目标比特上的退极化概率。
time（浮点数）：Trotterization模拟的时间周期。
depth（整数）：Trotterization的深度。
返回值:

（浮点数）：有噪声和无噪声Trotterization的最终状态之间的保真度。
    """
```

```
    return qml.math.fidelity(heisenberg_trotter(couplings,0,time, depth),heisenberg_trotter(
        couplings,p,time,depth))

qml.math.fidelity(heisenberg_trotter(random_params,0,1,2),evolve(random_params,1,2))
```

这个程序主要用于计算噪声和无噪声 Heisenberg 模型的最终状态之间的保真度，并创建一个特定形式的哈密顿量。

首先，程序导入了必要的库和模块：‘json‘、‘pennylane‘（作为‘qml‘）和 Pennylane 的 NumPy 接口（作为‘np‘）。

接下来，程序定义了一个变量‘num_wires‘，表示量子比特的数量。该变量可以根据需求进行调整。

然后，程序使用 Pennylane 的‘qml.device‘函数创建一个名为‘dev‘的设备，用于模拟量子计算。默认使用的设备是‘default.mixed‘，即混合态设备。你还可以选择其他设备，如‘lightning.qubit‘或‘lightning.gpu‘，根据计算环境的要求来进行调整。

接下来，程序定义了一个名为‘create_hamiltonian‘的函数，用于创建一个特定形式的哈密顿量。该函数接受一个参数‘params‘，该参数是一个包含多个参数值的列表。在函数内部，通过循环和条件语句构建了一个特定形式的哈密顿量，包括 Pauli X、Y 和 Z 算符的乘积。最后，使用这些算符和耦合参数创建了一个‘qml.Hamiltonian‘对象，并将其返回。

接下来，程序定义了一个名为‘evolve‘的量子节点函数，用于模拟哈密顿量的时间演化。该函数接受参数‘params‘（用于哈密顿量的创建）、‘time‘（模拟的总时间）和‘depth‘（模拟的深度）。在函数内部，使用 Pennylane 的‘qml.ApproxTimeEvolution‘函数对哈密顿量进行近似时间演化。最后，函数返回系统的状态。

然后，程序定义了一个名为‘calculate_fidelity‘的函数，用于计算有噪声和无噪声 Heisenberg 模型之间的保真度。该函数接受参数‘couplings‘（耦合参数列表）、‘p‘（退相干门的失效概率）、‘time‘（模拟的总时间）和‘depth‘（模拟的深度）。在函数内部，使用 Pennylane 的‘qml.math.fidelity‘函数计算有噪声和无噪声 Trotterizations 的最终状态之间的保真度，并将其返回。

最后一行代码计算了有噪声和无噪声 Heisenberg 模型的最终状态之间的保真度，即‘qml.math.fidelity(heisenberg_trotter(random_params,0,1,2),evolve(random_params,1,2))‘。

这段程序主要用于模拟和计算 Heisenberg 模型中有噪声和无噪声情况下的保真度，并提供了用于创建特定形式的哈密顿量的函数。

```
import json
from xxlimited import Xxo
import pennylane as qml
import pennylane.numpy as np
from timeit import default_timer as timer

#num_wires = 10 #increase this

#dev = qml.device("default.mixed", wires=num_wires) #lightning.qubit # lightning.gpu

def simulate(dev, couplings, p, time, depth):
    @qml.qnode(dev)
    def heisenberg_trotter(couplings, p, time, depth):
        """这个QNode在使用海森堡哈密顿量的Trotter近似进行演化后，返回自旋链的最终状态。

参数:
- couplings (list(float)): 一个长度为4的数组，按照顺序包含耦合常数和磁场强度[J_x, J_y, J_z, h]。
- p (float): 每个CNOT门后的去极化概率。
```

```
- depth (int): Trotter展开的深度。
- time (float): 系统演化的时间。

返回值:
- (numpy.tensor): 演化后的量子态。
        """

        def XX(i):
            # XX
            qml.RY(0,wires=1)
            print([i,(i+1)%num_wires][::-1])
            qml.CNOT([i,(i+1)%num_wires][::-1])
            if np.random.rand() < p:
                qml.RX(0.01, wires=i)
                qml.RY(0.01, wires=i)
                qml.RZ(0.01, wires=i)
            qml.RX(-2*couplings[0]*time/depth, wires=(i+1)%num_wires)
            qml.CNOT([i,(i+1)%num_wires][::-1])
            if np.random.rand() < p:
                qml.RX(0.01, wires=i)
                qml.RY(0.01, wires=i)
                qml.RZ(0.01, wires=i)


        def YY(i):
            #YY
            qml.RZ(np.pi/2, wires=i)
            qml.RZ(np.pi/2, wires=(i+1)%4)
            qml.CNOT([i,(i+1)%num_wires][::-1])
            if np.random.rand() < p:
                qml.RX(0.01, wires=i)
                qml.RY(0.01, wires=i)
                qml.RZ(0.01, wires=i)
            qml.RX(-2*couplings[1]*time/depth, wires=(i+1)%num_wires)
            qml.CNOT([i,(i+1)%num_wires][::-1])
            if np.random.rand() < p:
                qml.RX(0.01, wires=i)
                qml.RY(0.01, wires=i)
                qml.RZ(0.01, wires=i)
            qml.RZ(-np.pi/2, wires=i)
            qml.RZ(-np.pi/2, wires=(i+1)%num_wires)

        def ZZ(i):
            #ZZ
            qml.CNOT([i,(i+1)%num_wires])
            if np.random.rand() < p:
                qml.RX(0.01, wires=i)
                qml.RY(0.01, wires=i)
                qml.RZ(0.01, wires=i)
            qml.RZ(-2*couplings[2]*time/depth, wires=(i+1)%num_wires)
            qml.CNOT([i,(i+1)%num_wires])
            if np.random.rand() < p:
                qml.RX(0.01, wires=i)
                qml.RY(0.01, wires=i)
                qml.RZ(0.01, wires=i)


```

```
        def magnet(i):
            #magnetic field
            qml.RX(-2*couplings[3]*time/depth, wires=i)
        for j in range(depth):
            for i in num_wires:
            #first the XX, YY, ZZ part:
                XX(i)
            for i in num_wires:
                YY(i)
            for i in num_wires:
                ZZ(i)
            for i in num_wires:
                magnet(i)

        return qml.state()

    return heisenberg_trotter(couplings, p, time, depth)

def our_depolarising_noise(p):
    qml.Identity(wires=i) * np.sqrt(1-p)
    qml.PauliX(wires=i) * np.sqrt(p/3)
    qml.PauliY(wires=i) * np.sqrt(p/3)
    qml.PauliZ(wires=i) * np.sqrt(p/3)



def calculate_fidelity(dev, couplings, p, time, depth):
    """这个函数返回使用仅包含CNOT门和旋转门的嘈杂和无噪声Trotter化的海森堡模型的最终态之间的保真
        度。

参数:
- couplings (list(float)): 包含海森堡哈密顿量中的J_x、J_y、J_z和h参数的列表，如问题陈述中所定义。
- p (float): 每个CNOT门的目标量子比特上的去极化门的去极化概率。
- time (float): Trotter化模拟的时间演化周期。
- depth (int): Trotter化的深度。

返回值:
- (float): 嘈杂和无噪声Trotter化的最终态之间的保真度。     """
    return qml.math.fidelity(simulate(dev, couplings,0,time, depth),simulate(dev, couplings,p,time,
        depth))


num_wires = np.arange(2, 5)

timing = []
for t in num_wires:
    dev = qml.device("default.mixed", wires=num_wires) #lightning.gpu
    start = timer()
    calculated_fidelity_results = calculate_fidelity(dev, [1,2,1,0.3],0.5,2.5,1)
    end = timer()
    print(calculated_fidelity_results)
    timing.append(end - start)

print(qml.numpy.mean(timing))

#print(calculated_fidelity_results)
```

这个程序主要用于模拟和计算 Heisenberg 模型中有噪声和无噪声情况下的保真度，并使用 CNOT 门和旋转门来实现模拟。

首先，程序导入了必要的库和模块：‘json‘、‘xxlimited‘（作为 ‘Xxo‘）、‘pennylanev‘（作为 ‘qml‘）和 Pennylane 的 NumPy 接口（作为 ‘np‘），以及 ‘timeit‘库中的 ‘default_timer‘函数（作为 ‘timer‘）。

然后，程序定义了一个名为 ‘simulate‘的函数，用于模拟 Heisenberg 模型的演化。该函数接受设备（‘dev‘）、耦合参数（‘couplings‘）、概率参数（‘p‘）、模拟的总时间（‘time‘）和电路深度（‘depth‘）作为输入。在函数内部，定义了一个名为 ‘heisenberg_trotter‘的量子节点函数，用于实现 Heisenberg 模型的 Trotterization 演化。该节点函数接受耦合参数、概率参数、模拟的总时间和深度作为输入，并在循环中依次应用 XX、YY、ZZ 和磁场操作。最后，函数返回系统的状态。

接下来，程序定义了一个名为 ‘our_depolarising_noise‘的函数，用于实现极化误差模型。该函数接受概率参数 ‘p‘作为输入，并在每个量子比特上应用退极化门的不同组合，以模拟有噪声的情况。

然后，程序定义了一个名为 ‘calculate_fidelity‘的函数，用于计算有噪声和无噪声 Heisenberg 模型之间的保真度。该函数接受设备、耦合参数、概率参数、模拟的总时间和深度作为输入。在函数内部，调用了 ‘simulate‘函数模拟有噪声和无噪声情况下的系统演化，并使用 Pennylane 的 ‘qml.math.fidelity‘函数计算最终状态之间的保真度。

接下来，程序定义了一个变量 ‘num_wires‘，它是一个整数数组，表示要模拟的量子比特数量。然后，程序创建一个空列表 ‘timing‘，用于存储计算保真度所需的时间。

接下来，通过循环迭代 ‘num_wires‘中的每个值，程序依次创建 Pennylane 的设备（‘dev‘）并计算有噪声和无噪声情况下的保真度。在每次循环中，使用 ‘timer‘函数计算计算保真度所需的时间，并将其添加到 ‘timing‘列表中。

最后，程序打印出计算的保真度结果和计算所需的平均时间。

这段程序主要用于模拟和计算 Heisenberg 模型中有噪声和无噪声情况下的保真度，并评估计算所需的时间。

C

量子算法

C.0.1. Simon 算法

C.0.2. 简介

量子计算作为一种新兴的计算技术，为处理复杂问题和大量数据提供了一种全新的方法。量子计算中的一种著名算法是 Simon 算法，它由 Daniel Simon 于 1994 年提出。该算法在理论上证明了量子计算在某些问题上比经典计算更具优势，为 Shor 的算法等后续量子算法奠定了基础。在本文中，我们将详细介绍 Simon 算法的基本原理、应用和对量子计算发展的重要性。

C.0.3. 背景

Simon 问题源于数学和计算机科学中的黑箱问题，即给定一个未知函数 f，我们希望建立一个算法来获取 f 的某些性质。在 Simon 问题中，函数 f 满足以下条件：

$$f : 0, 1^n - > 0, 1^n \tag{C.1}$$

存在一个未知的比特串 $s(0 <= s < 2^n)$，使得对于所有 x, y，当且仅当 $y = x \oplus s$ 时，$f(x) = f(y)$ 其中，$\oplus$ 表示异或操作。问题的目标是找到未知的比特串 s。

C.0.4. 与经典计算的对比

在解决 Simon 问题时，经典计算方法需要进行大约 $2^(n/2)$ 次查询才能找到 s，而量子计算方法只需要 $O(n)$ 次查询。这意味着量子计算在解决这类问题时具有指数级的加速效果。

C.0.5. 原理

Simon 算法基于量子计算的基本原理，包括叠加态、测量、量子傅里叶变换等。算法的关键步骤如下：

以下是对 Simon 算法量子电路示意图的解释：

- **初始化**：电路中的每个量子比特都初始化为 |0□ 状态。前 n 个量子比特用于存储输入值 x，后 n 个量子比特用于存储黑箱函数 $f(x)$ 的输出。

Algorithm 6 Simon's Algorithm

procedure SimonsAlgorithm(f)
    准备 n 个量子比特处于状态 $|\psi_0\rangle = |0\rangle^{\otimes n}$
    对所有量子比特应用 Hadamard 变换 $H^{\otimes n}$，得到状态 $|\psi_1\rangle = H^{\otimes n}|\psi_0\rangle$
    查询黑箱函数 U_f，得到状态 $|\psi_2\rangle = U_f|\psi_1\rangle$
    对前 n 个量子比特应用 Hadamard 变换 $H^{\otimes n}$，得到状态 $|\psi_3\rangle = (H^{\otimes n} \otimes I)|\psi_2\rangle$
    测量前 n 个量子比特，得到结果 y
    求解线性方程组 $s \cdot y = 0 \pmod 2$，其中 s 是待求的秘密字符串
    重复直到找到一个非平凡解 s
end procedure

- **Hadamard 变换**：对前 n 个量子比特应用 Hadamard 门 H。这将使得这些量子比特处于一个均匀的叠加态。
- **受控非门**：这些门表示一个重要的黑箱操作。前 n 个量子比特中的每个量子比特都与后 n 个量子比特中的一个量子比特相连接。这些受控非门负责实现黑箱函数 U_f，将量子态 $|x\rangle|y\rangle$ 映射到量子态 $|x\rangle|y\rangle f(x)\rangle$。换句话说，在这个阶段，我们将输入 x 与黑箱函数的输出 $f(x)$ 进行模 2 加法。
- **再次应用 Hadamard 变换**：对前 n 个量子比特再次应用 Hadamard 门 H。这将使得这些量子比特的状态发生改变。
- **测量**：对前 n 个量子比特进行测量。测量结果是一个向量 y，它将用于求解线性方程组 $s\cdot y = 0(mod 2)$，其中 s 是我们要找的秘密字符串。
- **后处理**：在经典计算机上重复电路并收集多次测量结果。然后，根据这些结果解析线性方程组，以找到秘密字符串 s。

C.0.6. 算法实现

Listing C.1: Simon's 算法代码

```
mport numpy as np
from qiskit import QuantumCircuit, Aer, transpile, assemble
from qiskit.providers.aer import QasmSimulator
from qiskit.visualization import plot_histogram

# 定义黑箱函数
def black_box_f(bitstring):
    # 此处定义你的黑箱函数
    return bitstring

def simon_circuit(s):
    n = len(s)
    qc = QuantumCircuit(2 * n, n)

    # 应用Hadamard门到前n个量子比特
    qc.h(range(n))

    # 添加黑箱函数
    for i in range(n):
```

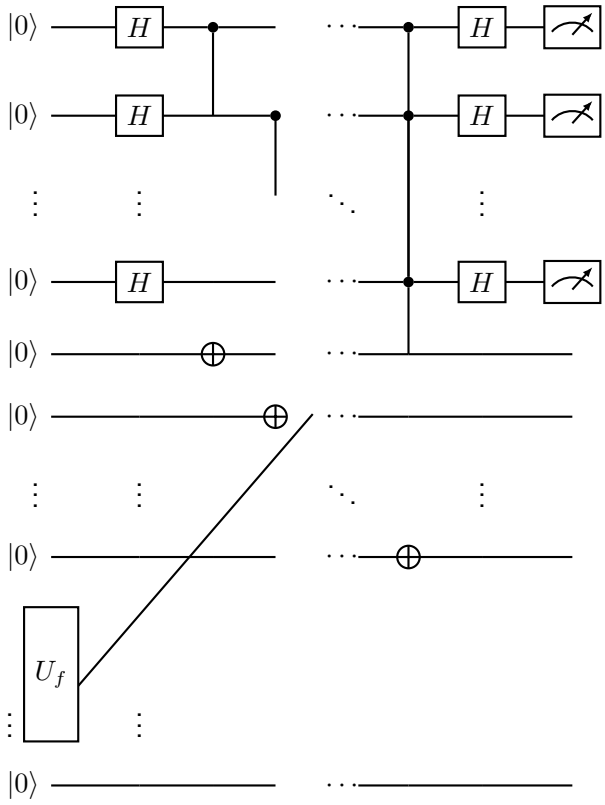


图 **C.1:** Simon 算法的量子电路示意图

```
        if s[i] == '1':
            qc.cx(i, n + i)

    # 再次应用Hadamard门到前n个量子比特
    qc.h(range(n))

    # 测量前n个量子比特
    qc.measure(range(n), range(n))

    return qc

s = "110"  # 秘密字符串
simon_circ = simon_circuit(s)
simon_circ.draw('mpl')
```

C.0.7. Simon 算法的应用

尽管 Simon 算法在某些特定问题上展示了量子计算的优越性，但目前它并没有直接的实际应用。然而，Simon 算法在量子计算领域的理论价值不容忽视。首先，它证明了在某些问题上，量子计算能够比经典计算更高效地解决问题。其次，它为后续的量子算法奠定了基础，如 Shor 的算法和 Grover 的算法。

Shor 的算法是一种用于寻找大整数因子的量子算法，它对密码学产生了深远的影响。Shor 的算法在解决整数因子分解问题时，具有指数级的速度优势。这意味着目前广泛应用的 RSA 加密算法可能会在量子计算机面前变得不再安全。Shor 的算法受到 Simon 算法的启发，其核心思想是利

用量子傅里叶变换来提取数学结构，从而实现指数级的加速。

Grover 的算法是另一个著名的量子搜索算法，它可以在无序数据库中以平方根的速度进行搜索。这使得 Grover 的算法在搜索大型无序数据库时具有显著的优势。与 Simon 算法类似，Grover 的算法也采用了量子计算的基本原理，例如叠加态和测量。

C.0.8. 总结

Simon 算法是量子计算中的一个重要算法，它在理论上证明了量子计算在某些问题上具有显著的优势。尽管 Simon 算法本身没有直接的实际应用，但它对后续的量子算法产生了深远的影响。Shor 的算法C.1和 Grover 的算法等著名量子算法都受到了 Simon 算法的启发。正是由于 Simon 算法等早期量子算法的贡献，量子计算领域得以快速发展，为解决各种复杂问题提供了一种全新的方法。随着量子计算技术的不断发展和完善，未来可能会出现更多的量子算法和应用，从而实现计算机科学的革新。

C.1. Deutsch's Algorithm

C.1.1. 简介

量子计算在计算机科学和信息技术领域取得了显著进展。量子计算机利用量子力学的原理，以量子比特（qubit）为基本单位，实现了在经典计算机上难以实现的高效算法。Deutsch’s Algorithm（Deutsch 算法）是量子计算中的一种重要算法，它由大卫 · 德意志（David Deutsch）在 1985 年首次提出。这个算法在理论上证明了量子计算机在某些问题上比经典计算机更有优势，为量子计算领域奠定了基础。

C.1.2. 背景与原理

在量子计算中，一个量子比特可以处于 0 和 1 的叠加态，即同时表示 0 和 1。这使得量子计算机可以处理大量信息，从而在某些任务上具有明显优势。Deutsch 算法展示了如何利用这种并行性解决一个特定的问题。这个问题涉及一个黑箱函数（oracle），它接收一个输入值，并输出一个相应的结果。Deutsch 的问题旨在判断该黑箱函数是常量还是平衡的。常量函数指的是对所有输入都输出相同值的函数，而平衡函数则是对一半输入输出 0，另一半输出 1 的函数。

C.1.3. 基本过程

其中，U_f 是一个两比特门，其对于所有的 $x \in 0,1$ 和 $y \in 0,1$ 都满足 $U_f \left|x,y\right\rangle = \left|x, y \oplus f(x)\right\rangle$，其中 $\oplus$ 是模 2 加法运算。在实际应用中，U_f 可以通过构造一个经典函数 f 的量子电路来实现。

Deutsch 算法通过以下几个步骤来解决这个问题：

（1）准备工作：首先，将两个量子比特分别初始化为 |0> 和 |1> 状态，表示为 |0>|1>。

（2）Hadamard 变换：接着，对每个量子比特执行 Hadamard 变换（H）。Hadamard 变换将基态 |0> 和 |1> 映射到叠加态 (|0>+|1>) 和 (|0>-|1>)。这样，量子比特的初始状态变为 (|0>+|1>)(|0>-|1>)。

（3）Oracle 操作：将第一步中的黑箱函数作用在当前的量子比特状态上。这个操作将根据输入值改变第一个量子比特的状态，而第二个量子比特的状态将不受影响。

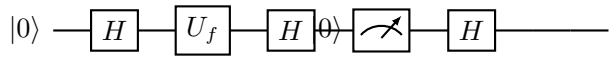
（4）第二次 Hadamard 变换：再次对第一个量子比特执行 Hadamard 变换。此时，第一个量子比特的状态将依赖于黑箱函数的性质（常量或平衡）。

（5）测量与判断：对第一个量子比特进行测量。如果测量结果为 0，那么黑箱函数为常量函数；

Algorithm 7 Deutsch's Algorithm

```
procedure Deutsch(f)
    Initialize two qubits q_1 and q_2 to |0⟩
    Apply Hadamard gate to q_1 to get |+⟩
    Apply Hadamard gate to q_2 to get |+⟩
    Apply U_f gate to the two qubits
    Apply Hadamard gate to q_1
    Measure q_1
    if q_1 is |0⟩ then
        return f is constant
    else
        return f is balanced
    end if
end procedure
```

如果测量结果为 1，那么黑箱函数为平衡函数。



C.1.4. 量子优势

Deutsch 算法的关键在于利用量子叠加态的并行性。在经典计算法中，要解决类似的问题，需要至少两次查询黑箱函数来确定它是常量还是平衡的。然而，通过使用 Deutsch 算法，量子计算机仅需一次查询就可以解决这个问题。这是因为量子比特的叠加态允许算法在一个步骤中同时处理多个可能的输入。因此，Deutsch 算法证明了量子计算机在特定问题上具有显著的速度优势。

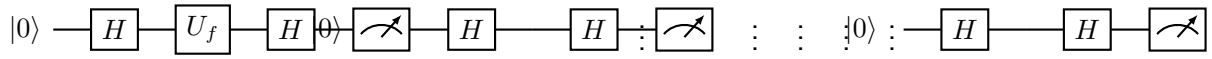
C.1.5. Deutsch-Jozsa 算法

Algorithm 8 Deutsch-Jozsa Algorithm

```
procedure Deutsch-Jozsa(f)
    Initialize n qubits q_1, ..., q_n to |0⟩
    Apply Hadamard gate to each qubit to get |+⟩^{⊗n}
    Apply U_f gate to the n qubits
    Apply Hadamard gate to each qubit
    Measure all qubits
    if the measurement result is 0 then
        return f is constant
    else
        return f is balanced
    end if
end procedure
```

其中，U_f 是一个 n 比特门，其对于所有的 $x \in 0, 1^n$ 都满足 $U_f |x, y\rangle = |x, y \oplus f(x)\rangle$，其中 $\oplus$

是模 2 加法运算。在实际应用中，U_f 可以通过构造一个经典函数 f 的量子电路来实现。Deutsch 算法的一个更广泛的应用是 Deutsch-Jozsa 算法，由大卫 · 德意志和理查德 · 约兹在 1992 年共同提出。



其中，U_f 门的作用为 $U_f|x,y\rangle = |x, y \oplus f(x)\rangle$，其中 $x \in 0,1^n$，$y \in 0,1$，可以通过构造一个经典函数 f 的量子电路来实现。量子电路的输入为 n 个初始值为 $|0\rangle$ 的量子比特，输出为 $n+1$ 个测量结果。

Deutsch-Jozsa 算法是 Deutsch 算法的扩展，可以处理多量子比特的情况。在这种情况下，黑箱函数可以接收 n 个输入量子比特，输出一个结果。与 Deutsch 算法类似，Deutsch-Jozsa 算法的目标是判断这个黑箱函数是常量还是平衡的。通过使用类似的量子技巧，Deutsch-Jozsa 算法可以在一次查询中解决这个问题，而经典计算机需要 $2^(n-1)+1$□□□□□□□□

C.1.6. 影响

Deutsch 算法和 Deutsch-Jozsa 算法在量子计算领域具有重要意义。它们证明了量子计算机在某些问题上具有显著优势，并为后续更复杂的量子算法奠定了基础，如 Shor 的算法和 Grover 的搜索算法等。这些算法在加密、搜索和优化等领域展示了量子计算的潜力，激发了对量子计算的广泛研究和发展。

C.1.7. 总结

总之，Deutsch 算法是量子计算领域的一个里程碑式算法，它展示了量子计算机在某些问题上的速度优势。通过利用量子力学原理，Deutsch 算法在一次查询中就能解决一个在经典计算机上需要多次查询的问题。这个算法为后续更复杂的量子算法奠定了基础，并为量子计算领域的未来发展提供了强大的动力。

C.2. Deutsch-Jozsa Algorithm

以下我们解释量子计算机如何将计算步骤应用于其量子比特寄存器。这里有两种模型：量子图灵机 [?, ?] 和量子电路模型 [?, ?]。这两种模型是等价的，意味着它们可以在多项式时间内模拟彼此，假设电路是适当的“均匀”。在这里，我们仅解释更受研究者欢迎的量子电路模型。

在经典复杂性理论中，*布尔电路* 是一种包含与、或和非门的有限有向无环图。这种图有 n 个输入节点，包含 n 个输入位（$n \geq 0$）。内部节点是与、或和非门，还有一个或多个指定的输出节点。根据电路设计，初始输入位进入与、或和非门，最终输出节点得到某个值。我们说一个电路*计算*某个布尔函数 $f : 1^n \to 1^m$，如果对于每个输入 $x \in 1^n$，输出节点得到正确的值 $f(x)$。

电路族 是一组电路 $\mathcal{C} = C_n$，每个输入大小 n 对应一个。每个电路有一个输出位。这样的电路族可以*识别*或*判断*一个*语言* $L \subseteq 1^* = \cup_{n\geq 0} 1^n$，即对于每个 n 和每个输入 $x \in 1^n$，当 $x \in L$ 时，电路 C_n 输出 1，否则输出 0。[1]如果存在一个确定性图灵机，给定输入 n，可以用对数级空间输出 C_n，那么这样的电路族就是*一致多项式*。[2]需要注意的是，电路 C_n 的大小（门的数量）随着 n 的增长最多是多项式的。已知一致多项式电路族的能力等于多项式时间确定性图灵机：当且仅当 $L \in$ 时，语言 L 可以由一致多项式电路族判断 [?, Theorem 11.5],，其中 是由多项式时间图灵机可判

[1]我们可以将语言 L 视为一系列布尔函数 $f_n : 1^n \to 1$，其中 f_n 在 L 中的 n 位字符串上取值为 1。然后电路 C_n 计算函数 f_n。

[2]对数空间意味着时间最多为 n 的多项式，因为这样的机器只有 (n) 个不同的内部状态，所以它在 (n) 步之后要么停止要么永远循环。

断的语言类。

类似地，我们可以考虑随机电路。除了 n 个输入位之外，这些电路还接收一些随机位（“硬币翻转”）作为输入。如果随机电路对于每个输入 x（概率取决于随机位的值），以至少 $2/3$ 的概率成功输出正确的答案 $f(x)$，则随机电路计算函数 f。随机电路的能力等于随机图灵机的能力：当且仅当 $L \in$ 时，语言 L 可以由一致多项式随机电路族判断，其中 （“有界错误概率多项式时间”）是可以用成功概率至少为 $2/3$ 的随机图灵机有效识别的语言类。由于我们可以有效地降低随机算法的错误概率（参见附录 **??**），这里的特定值 $2/3$ 实际上并不重要，可以用 $(1/2, 1)$ 之间的任何固定常数替换。

D

Task Division Example

If a task division is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.

表 **D.1:** Distribution of the workload

	Task	Student Name(s)
	Summary	
Chapter 1	Introduction	
Chapter 2		
Chapter 3		
Chapter *		
Chapter *	Conclusion	
	Editors	
	CAD and Figures	
	Document Design and Layout	

参考文献

[1] J. D. Hidary and J. D. Hidary, *Quantum computing: an applied approach*. Springer, 2019, vol. 1.

[2] A. Church, “Am turing. on computable numbers, with an application to the entscheidungs problcm. proceedings of the london mathematical society, 2 s. vol. 42 (1936–1937), pp. 230–265.” *The Journal of Symbolic Logic*, vol. 2, no. 1, pp. 42–43, 1937.

[3] Y. I. Manin, “Vychislimoe i nevychislimoe (computable and noncomputable), moscow: Sov,” 1980.

[4] ——, “Classical computing, quantum computing, and shor’s factoring algorithm,” *arXiv preprint quant-ph/9903008*, 1999.

[5] R. P. Feynman, “Simulating physics with computers,” in *Feynman and computation*. CRC Press, 2018, pp. 133–153.

[6] ——, “Quantum mechanical computers,” *Optics news*, vol. 11, no. 2, pp. 11–20, 1985.

[7] P. Benioff, “Quantum mechanical hamiltonian models of turing machines,” *Journal of Statistical Physics*, vol. 29, pp. 515–546, 1982.

[8] D. Deutsch, “Quantum theory, the church–turing principle and the universal quantum computer,” *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, vol. 400, no. 1818, pp. 97–117, 1985.

[9] J. Preskill, “Lecture notes for physics 229: Quantum information and computation,” *California Institute of Technology*, vol. 16, no. 1, pp. 1–8, 1998.

[10] ——, “Quantum computing 40 years later,” *arXiv preprint arXiv:2106.10522*, 2021.

[11] D. Deutsch and R. Jozsa, “Rapid solution of problems by quantum computation,” *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[12] D. R. Simon, “On the power of quantum computation,” *SIAM journal on computing*, vol. 26, no. 5, pp. 1474–1483, 1997.

[13] E. Bernstein and U. Vazirani, “Quantum complexity theory,” in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 11–20.

[14] P. W. Shor, “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[15] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[16] R. Rivest, “Cryptography. in van leeuwen, j,” *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, pp. 719–756, 1994.

[17] C. H. Bennett and G. Brassard, “Quantum cryptography: Public key distribution and coin tossing,” *arXiv preprint arXiv:2003.06557*, 2020.

[18] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, R. Cleve, and I. L. Chuang, “Experimental realization of an order-finding algorithm with an nmr quantum computer,” *Physical Review Letters*, vol. 85, no. 25, p. 5452, 2000.

[19] A. Einstein, B. Podolsky, and N. Rosen, “Can quantum-mechanical description of physical reality be considered complete?” *Physical review*, vol. 47, no. 10, p. 777, 1935.

[20] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, p. 79, 2018.

[21] E. Schrödinger, “Discussion of probability relations between separated systems,” in *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 31, no. 4. Cambridge University Press, 1935, pp. 555–563.

[22] R. Landauer, “Irreversibility and heat generation in the computing process,” *IBM journal of research and development*, vol. 44, no. 1/2, p. 261, 2000.

[23] M. A. Nielsen and I. Chuang, “Quantum computation and quantum information,” 2002.

[24] P. Benioff, “The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines,” *Journal of statistical physics*, vol. 22, pp. 563–591, 1980.

[25] Y. Manin, “Computable and uncomputable,” *Sovetskoye Radio, Moscow*, vol. 128, 1980.

[26] S. Lloyd, “A potentially realizable quantum computer,” *Science*, vol. 261, no. 5128, pp. 1569–1571, 1993.

[27] M. Hayashi and H. Zhu, “Secure uniform random-number extraction via incoherent strategies,” *Physical Review A*, vol. 97, no. 1, p. 012302, 2018.

[28] S. Aaronson and S.-H. Hung, “Certified randomness from quantum supremacy,” *arXiv preprint arXiv:2303.01625*, 2023.

[29] P. W. Shor, “Algorithms for quantum computation: discrete logarithms and factoring,” in *Proceedings 35th annual symposium on foundations of computer science*. Ieee, 1994, pp. 124–134.

[30] L. M. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, and I. L. Chuang, “Experimental realization of shor’s quantum factoring algorithm using nuclear magnetic resonance,” *Nature*, vol. 414, no. 6866, pp. 883–887, 2001.

[31] L. K. Grover, “Quantum mechanics helps in searching for a needle in a haystack,” *Physical review letters*, vol. 79, no. 2, p. 325, 1997.

[32] E. Farhi and S. Gutmann, “Analog analogue of a digital quantum computation,” *Physical Review A*, vol. 57, no. 4, p. 2403, 1998.

[33] Y. Nakamura, Y. A. Pashkin, and J. Tsai, “Coherent control of macroscopic quantum states in a single-cooper-pair box,” *nature*, vol. 398, no. 6730, pp. 786–788, 1999.

[34] Y. Nakamura, Y. A. Pashkin, and J. S. Tsai, “Rabi oscillations in a josephson-junction charge two-level system,” *Physical Review Letters*, vol. 87, no. 24, p. 246601, 2001.

[35] J. Cirac and P. Zoller, “Quantum computations with cold trapped ions,” *Phys. Rev. Lett*, vol. 74, p. 20.

[36] M. Veldhorst, H. Eenink, C.-H. Yang, and A. S. Dzurak, “Silicon cmos architecture for a spin-based quantum computer,” *Nature communications*, vol. 8, no. 1, p. 1766, 2017.

[37] J. W. Silverstone, J. Wang, D. Bonneau, P. Sibson, R. Santagati, C. Erven, J. O’Brien, and M. Thompson, “Silicon quantum photonics,” in *2016 International Conference on Optical MEMS and Nanophotonics (OMN)*. IEEE, 2016, pp. 1–2.

[38] M. Verbin, O. Zilberberg, Y. E. Kraus, Y. Lahini, and Y. Silberberg, “Observation of topological phase transitions in photonic quasicrystals,” *Physical review letters*, vol. 110, no. 7, p. 076403, 2013.

[39] D. P. DiVincenzo, “Topics in quantum computers,” *Mesoscopic electron transport*, pp. 657–677, 1997.

[40] L. B. Nguyen, G. Koolstra, Y. Kim, A. Morvan, T. Chistolini, S. Singh, K. N. Nesterov, C. Jünger, L. Chen, Z. Pedramrazi *et al.*, “Scalable high-performance fluxonium quantum processor,” *arXiv preprint arXiv:2201.09374*, 2022.

[41] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, “Elementary gates for quantum computation,” *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[42] T. Toffoli, “Reversible computing,” in *Automata, Languages and Programming: Seventh Colloquium Noordwijkerhout, the Netherlands July 14–18, 1980 7*. Springer, 1980, pp. 632–644.

[43] E. Fredkin and T. Toffoli, “Conservative logic,” *International Journal of theoretical physics*, vol. 21, no. 3-4, pp. 219–253, 1982.

[44] H. M. Sheffer, “A set of five independent postulates for boolean algebras, with application to logical constants,” *Transactions of the American mathematical society*, vol. 14, no. 4, pp. 481–488, 1913.

[45] Y. Shi, “Both toffoli and controlled-not need little help to do universal quantum computation,” *arXiv preprint quant-ph/0205115*, 2002.

[46] P. O. Boykin, T. Mor, M. Pulver, V. Roychowdhury, and F. Vatan, “On universal and fault-tolerant quantum computing,” *arXiv preprint quant-ph/9906054*, 1999.

[47] M. Ozols, “Clifford group,” *Essays at University of Waterloo, Spring*, 2008.

[48] C. M. Dawson and M. A. Nielsen, “The solovay-kitaev algorithm,” *arXiv preprint quant-ph/0505030*, 2005.

[49] G. H. Hardy and J. E. Littlewood, “Some problems of diophantine approximation,” *Acta math*, vol. 37, no. 1, pp. 193–239, 1914.

[50] D. E. Knuth, “Big omicron and big omega and big theta,” *ACM Sigact News*, vol. 8, no. 2, pp. 18–24, 1976.

[51] R. Solovay and V. Strassen, “A fast monte-carlo test for primality,” *SIAM journal on Computing*, vol. 6, no. 1, pp. 84–85, 1977.

[52] M. Agrawal, N. Kayal, and N. Saxena, “Primes is in p,” *Annals of mathematics*, pp. 781–793, 2004.

[53] T. D. Ladd, F. Jelezko, R. Laflamme, Y. Nakamura, C. Monroe, and J. L. O’Brien, “Quantum computers,” *nature*, vol. 464, no. 7285, pp. 45–53, 2010.

[54] Y. Wang, X. Zhang, T. A. Corcovilos, A. Kumar, and D. S. Weiss, “Coherent addressing of individual neutral atoms in a 3d optical lattice,” *Physical review letters*, vol. 115, no. 4, p. 043003, 2015.

[55] H. Pichler, S.-T. Wang, L. Zhou, S. Choi, and M. D. Lukin, “Quantum optimization for maximum independent set using rydberg atom arrays,” *arXiv preprint arXiv:1808.10816*, 2018.

[56] H. Levine, A. Keesling, A. Omran, H. Bernien, S. Schwartz, A. S. Zibrov, M. Endres, M. Greiner, V. Vuletić, and M. D. Lukin, “High-fidelity control and entanglement of rydberg-atom qubits,” *Physical review letters*, vol. 121, no. 12, p. 123603, 2018.

[57] T.-Y. Wu, A. Kumar, F. Giraldo, and D. S. Weiss, “Stern–gerlach detection of neutral-atom qubits in a state-dependent optical lattice,” *Nature Physics*, vol. 15, no. 6, pp. 538–542, 2019.

[58] M. Saffman, T. G. Walker, and K. Mølmer, “Quantum information with rydberg atoms,” *Reviews of modern physics*, vol. 82, no. 3, p. 2313, 2010.

[59] D. S. Weiss and M. Saffman, “Quantum computing with neutral atoms,” *Physics Today*, vol. 70, no. 7, 2017.

[60] H. Bernien, S. Schwartz, A. Keesling, H. Levine, A. Omran, H. Pichler, S. Choi, A. S. Zibrov, M. Endres, M. Greiner *et al.*, “Probing many-body dynamics on a 51-atom quantum simulator,” *Nature*, vol. 551, no. 7682, pp. 579–584, 2017.

[61] C. Schreyvogel, V. Polyakov, R. Wunderlich, J. Meijer, and C. Nebel, “Active charge state control of single nv centres in diamond by in-plane al-schottky junctions,” *Scientific reports*, vol. 5, no. 1, pp. 1–12, 2015.

[62] S. B. van Dam, M. Walsh, M. J. Degen, E. Bersin, S. L. Mouradian, A. Galiullin, M. Ruf, M. IJspeert, T. H. Taminiau, R. Hanson *et al.*, “Optical coherence of diamond nitrogen-vacancy centers formed by ion implantation and annealing,” *Physical Review B*, vol. 99, no. 16, p. 161203, 2019.

[63] C. Adami and N. J. Cerf, “Quantum computation with linear optics,” 1998.

[64] E. Knill, R. Laflamme, and G. J. Milburn, “A scheme for efficient quantum computation with linear optics,” *nature*, vol. 409, no. 6816, pp. 46–52, 2001.

[65] P. Kok, W. J. Munro, K. Nemoto, T. C. Ralph, J. P. Dowling, and G. J. Milburn, “Linear optical quantum computing with photonic qubits,” *Reviews of modern physics*, vol. 79, no. 1, p. 135, 2007.

[66] R. Raussendorf and H. J. Briegel, “A one-way quantum computer,” *Physical review letters*, vol. 86, no. 22, p. 5188, 2001.

[67] R. Raussendorf and H. Briegel, “Computational model underlying the one-way quantum computer,” *arXiv preprint quant-ph/0108067*, 2001.

[68] R. Raussendorf, D. Browne, and H. Briegel, “The one-way quantum computer–a non-network model of quantum computation,” *journal of modern optics*, vol. 49, no. 8, pp. 1299–1306, 2002.

[69] R. Raussendorf, D. E. Browne, and H. J. Briegel, “Measurement-based quantum computation on cluster states,” *Physical review A*, vol. 68, no. 2, p. 022312, 2003.

[70] D. Hanneke, J. Home, J. D. Jost, J. M. Amini, D. Leibfried, and D. J. Wineland, “Realization of a programmable two-qubit quantum processor,” *Nature Physics*, vol. 6, no. 1, pp. 13–16, 2010.

[71] E. Martin-Lopez, A. Laing, T. Lawson, R. Alvarez, X.-Q. Zhou, and J. L. O’brien, “Experimental realization of shor’s quantum factoring algorithm using qubit recycling,” *Nature photonics*, vol. 6, no. 11, pp. 773–776, 2012.

[72] J. Carolan, C. Harrold, C. Sparrow, E. Martín-López, N. J. Russell, J. W. Silverstone, P. J. Shadbolt, N. Matsuda, M. Oguma, M. Itoh *et al.*, “Universal linear optics,” *Science*, vol. 349, no. 6249, pp. 711–716, 2015.

[73] X. Qiang, X. Zhou, J. Wang, C. M. Wilkes, T. Loke, S. O’Gara, L. Kling, G. D. Marshall, R. Santagati, T. C. Ralph *et al.*, “Large-scale silicon quantum photonics implementing arbitrary two-qubit processing,” *Nature photonics*, vol. 12, no. 9, pp. 534–539, 2018.

[74] S. Sun, H. Kim, Z. Luo, G. S. Solomon, and E. Waks, “A single-photon switch and transistor enabled by a solid-state quantum memory,” *Science*, vol. 361, no. 6397, pp. 57–60, 2018.

[75] L.-M. Duan and H. Kimble, “Scalable photonic quantum computation through cavity-assisted interactions,” *Physical review letters*, vol. 92, no. 12, p. 127902, 2004.

[76] M. Z. Hasan and C. L. Kane, “Colloquium: topological insulators,” *Reviews of modern physics*, vol. 82, no. 4, p. 3045, 2010.

[77] X.-L. Qi and S.-C. Zhang, “Topological insulators and superconductors,” *Reviews of Modern Physics*, vol. 83, no. 4, p. 1057, 2011.

[78] L. Lu, J. D. Joannopoulos, and M. Soljačić, “Topological photonics,” *Nature photonics*, vol. 8, no. 11, pp. 821–829, 2014.

[79] T. Ozawa, H. M. Price, A. Amo, N. Goldman, M. Hafezi, L. Lu, M. C. Rechtsman, D. Schuster, J. Simon, O. Zilberberg *et al.*, “Topological photonics,” *Reviews of Modern Physics*, vol. 91, no. 1, p. 015006, 2019.

[80] J.-S. Xu, K. Sun, Y.-J. Han, C.-F. Li, J. K. Pachos, and G.-C. Guo, “Simulating the exchange of majorana zero modes with a photonic system,” *Nature communications*, vol. 7, no. 1, p. 13194, 2016.

[81] Z. Wang, Y. Chong, J. D. Joannopoulos, and M. Soljačić, “Observation of unidirectional backscattering-immune topological electromagnetic states,” *Nature*, vol. 461, no. 7265, pp. 772–775, 2009.

[82] M. Hafezi, E. A. Demler, M. D. Lukin, and J. M. Taylor, “Robust optical delay lines with topological protection,” *Nature Physics*, vol. 7, no. 11, pp. 907–912, 2011.

[83] M. C. Rechtsman, J. M. Zeuner, Y. Plotnik, Y. Lumer, D. Podolsky, F. Dreisow, S. Nolte, M. Segev, and A. Szameit, “Photonic floquet topological insulators,” *Nature*, vol. 496, no. 7444, pp. 196–200, 2013.

[84] M. Hafezi, S. Mittal, J. Fan, A. Migdall, and J. Taylor, “Imaging topological edge states in silicon photonics,” *Nature Photonics*, vol. 7, no. 12, pp. 1001–1005, 2013.

[85] A. Blanco-Redondo, I. Andonegui, M. J. Collins, G. Harari, Y. Lumer, M. C. Rechtsman, B. J. Eggleton, and M. Segev, “Topological optical waveguiding in silicon and the transition between topological and trivial defect states,” *Physical review letters*, vol. 116, no. 16, p. 163901, 2016.

[86] S. Mittal, S. Ganeshan, J. Fan, A. Vaezi, and M. Hafezi, “Measurement of topological invariants in a 2d photonic system,” *Nature Photonics*, vol. 10, no. 3, pp. 180–183, 2016.

[87] L. Xiao, X. Zhan, Z. Bian, K. Wang, X. Zhang, X. Wang, J. Li, K. Mochizuki, D. Kim, N. Kawakami *et al.*, “Observation of topological edge states in parity–time-symmetric quantum walks,” *Nature Physics*, vol. 13, no. 11, pp. 1117–1123, 2017.

[88] W.-J. Chen, M. Xiao, and C. T. Chan, “Photonic crystals possessing multiple weyl points and the experimental observation of robust surface states,” *Nature communications*, vol. 7, no. 1, p. 13038, 2016.

[89] J. Noh, S. Huang, D. Leykam, Y. Chong, K. Chen, and M. Rechtsman, “Experimental observation of optical weyl points,” in *European Quantum Electronics Conference*. Optica Publishing Group, 2017, p. JSIV_2_1.

[90] F. Li, X. Huang, J. Lu, J. Ma, and Z. Liu, “Weyl points and fermi arcs in a chiral phononic crystal,” *Nature Physics*, vol. 14, no. 1, pp. 30–34, 2018.

[91] Y. E. Kraus, Y. Lahini, Z. Ringel, M. Verbin, and O. Zilberberg, “Topological states and adiabatic pumping in quasicrystals,” *Physical review letters*, vol. 109, no. 10, p. 106402, 2012.

[92] M. Verbin, O. Zilberberg, Y. Lahini, Y. E. Kraus, and Y. Silberberg, “Topological pumping over a photonic fibonacci quasicrystal,” *Physical Review B*, vol. 91, no. 6, p. 064201, 2015.

[93] O. Zilberberg, S. Huang, J. Guglielmon, M. Wang, K. P. Chen, Y. E. Kraus, and M. C. Rechtsman, “Photonic topological boundary pumping as a probe of 4d quantum hall physics,” *Nature*, vol. 553, no. 7686, pp. 59–62, 2018.

[94] A. Blanco-Redondo, B. Bell, M. Segev, and B. Eggleton, “Photonic quantum walks with symmetry protected topological phases,” in *AIP Conference Proceedings*, vol. 1874, no. 1. AIP Publishing LLC, 2017, p. 020001.

[95] S. Mittal, E. A. Goldschmidt, and M. Hafezi, “A topological source of quantum light,” *Nature*, vol. 561, no. 7724, pp. 502–506, 2018.

[96] J.-L. Tambasco, G. Corrielli, R. J. Chapman, A. Crespi, O. Zilberberg, R. Osellame, and A. Peruzzo, “Quantum interference of topological states of light,” *Science advances*, vol. 4, no. 9, p. eaat3187, 2018.

[97] C.-K. Hong, Z.-Y. Ou, and L. Mandel, “Measurement of subpicosecond time intervals between two photons by interference,” *Physical review letters*, vol. 59, no. 18, p. 2044, 1987.

[98] T. Watson, S. Philips, E. Kawakami, D. Ward, P. Scarlino, M. Veldhorst, D. Savage, M. Lagally, M. Friesen, S. Coppersmith *et al.*, “A programmable two-qubit quantum processor in silicon,” *nature*, vol. 555, no. 7698, pp. 633–637, 2018.

[99] V. Bouchiat, D. Vion, P. Joyez, D. Esteve, and M. Devoret, “Quantum coherence with a single cooper pair,” *Physica Scripta*, vol. 1998, no. T76, p. 165, 1998.

[100] E. National Academies of Sciences, Medicine *et al.*, “Quantum computing: progress and prospects,” 2019.

[101] P. J. O’Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, N. Ding *et al.*, “Scalable quantum simulation of molecular energies,” *Physical Review X*, vol. 6, no. 3, p. 031007, 2016.

[102] S. Rosenblum, Y. Y. Gao, P. Reinhold, C. Wang, C. J. Axline, L. Frunzio, S. M. Girvin, L. Jiang, M. Mirrahimi, M. H. Devoret *et al.*, “A cnot gate between multiphoton qubits encoded in two cavities,” *Nature communications*, vol. 9, no. 1, p. 652, 2018.

[103] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, “A quantum engineer’s guide to superconducting qubits,” *Applied physics reviews*, vol. 6, no. 2, p. 021318, 2019.

[104] A. Roy and D. P. DiVincenzo, “Topological quantum computing,” *arXiv preprint arXiv:1701.05052*, 2017.

[105] A. Y. Kitaev, “Fault-tolerant quantum computation by anyons,” *Annals of physics*, vol. 303, no. 1, pp. 2–30, 2003.

[106] M. Freedman, A. Kitaev, M. Larsen, and Z. Wang, “Topological quantum computation,” *Bulletin of the American Mathematical Society*, vol. 40, no. 1, pp. 31–38, 2003.

[107] V. Lahtinen and J. Pachos, “A short introduction to topological quantum computation,” *SciPost Physics*, vol. 3, no. 3, p. 021, 2017.

[108] J. G. Bohnet, B. C. Sawyer, J. W. Britton, M. L. Wall, A. M. Rey, M. Foss-Feig, and J. J. Bollinger, “Quantum spin dynamics and entanglement generation with hundreds of trapped ions,” *Science*, vol. 352, no. 6291, pp. 1297–1301, 2016.

[109] D. Lucas, C. Donald, J. Home, M. McDonnell, A. Ramos, D. Stacey, J.-P. Stacey, A. Steane, and S. Webster, “Oxford ion-trap quantum computing project,” *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 361, no. 1808, pp. 1401–1408, 2003.

[110] N. Friis, O. Marty, C. Maier, C. Hempel, M. Holzäpfel, P. Jurcevic, M. B. Plenio, M. Huber, C. Roos, R. Blatt *et al.*, “Observation of entangled states of a fully controlled 20-qubit system,” *Physical Review X*, vol. 8, no. 2, p. 021012, 2018.

[111] D. R. Leibrandt, J. Labaziewicz, V. Vuletić, and I. L. Chuang, “Cavity sideband cooling of a single trapped ion,” *Physical review letters*, vol. 103, no. 10, p. 103001, 2009.

[112] C. Flühmann, T. L. Nguyen, M. Marinelli, V. Negnevitsky, K. Mehta, and J. Home, “Encoding a qubit in a trapped-ion mechanical oscillator,” *Nature*, vol. 566, no. 7745, pp. 513–517, 2019.

[113] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, “Trapped-ion quantum computing: Progress and challenges,” *Applied Physics Reviews*, vol. 6, no. 2, p. 021314, 2019.

[114] R. LaRose, “Overview and comparison of gate level quantum software platforms,” *Quantum*, vol. 3, p. 130, 2019.

[115] M. Fingerhuth, T. Babej, and P. Wittek, “Open source software in quantum computing,” *PloS one*, vol. 13, no. 12, p. e0208561, 2018.

[116] E. S. Fried, N. P. Sawaya, Y. Cao, I. D. Kivlichan, J. Romero, and A. Aspuru-Guzik, “qtorch: The quantum tensor contraction handler,” *PloS one*, vol. 13, no. 12, p. e0208510, 2018.

[117] M. Smelyanskiy, N. P. Sawaya, and A. Aspuru-Guzik, “qhipster: The quantum high performance software testing environment,” *arXiv preprint arXiv:1601.07195*, 2016.

[118] C. H. Bennett, G. Brassard, C. Crépeau, R. Jozsa, A. Peres, and W. K. Wootters, “Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels,” *Physical review letters*, vol. 70, no. 13, p. 1895, 1993.

[119] D. Boschi, S. Branca, F. De Martini, L. Hardy, and S. Popescu, “Experimental realization of teleporting an unknown pure quantum state via dual classical and einstein-podolsky-rosen channels,” *Physical Review Letters*, vol. 80, no. 6, p. 1121, 1998.

[120] K. S. Chou, J. Z. Blumoff, C. S. Wang, P. C. Reinhold, C. J. Axline, Y. Y. Gao, L. Frunzio, M. Devoret, L. Jiang, and R. Schoelkopf, “Deterministic teleportation of a quantum gate between two logical qubits,” *Nature*, vol. 561, no. 7723, pp. 368–373, 2018.

[121] C. H. Bennett and S. J. Wiesner, “Communication via one-and two-particle operators on einstein-podolsky-rosen states,” *Physical review letters*, vol. 69, no. 20, p. 2881, 1992.

[122] C. Wang, F.-G. Deng, Y.-S. Li, X.-S. Liu, and G. L. Long, “Quantum secure direct communication with high-dimension quantum superdense coding,” *Physical Review A*, vol. 71, no. 4, p. 044305, 2005.

[123] K. Mattle, H. Weinfurter, P. G. Kwiat, and A. Zeilinger, “Dense coding in experimental quantum communication,” *Physical Review Letters*, vol. 76, no. 25, p. 4656, 1996.

[124] S. Bravyi, D. Gosset, and R. König, “Quantum advantage with shallow circuits,” *Science*, vol. 362, no. 6412, pp. 308–311, 2018.

[125] S. Bravyi, D. Gosset, R. Koenig, and M. Tomamichel, “Quantum advantage with noisy shallow circuits,” *Nature Physics*, vol. 16, no. 10, pp. 1040–1045, 2020.

[126] M. Loceff, “A course in quantum computing for the community college. vol. 1,” *Foothill College.—2015*.

[127] E. Bernstein and U. Vazirani, “Proceedings of the 25th annual acm symposium on theory of computing,” *San Diego, CA*, 1993.

[128] N. D. Mermin, *Quantum computer science: an introduction*. Cambridge University Press, 2007.

[129] C. Gidney and M. Ekerå, “How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits,” *Quantum*, vol. 5, p. 433, 2021.

[130] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, “Strengths and weaknesses of quantum computing,” *SIAM journal on Computing*, vol. 26, no. 5, pp. 1510–1523, 1997.

[131] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O’brien, “A variational eigenvalue solver on a photonic quantum processor,” *Nature communications*, vol. 5, no. 1, p. 4213, 2014.

[132] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, “The theory of variational hybrid quantum-classical algorithms,” *New Journal of Physics*, vol. 18, no. 2, p. 023023, 2016.

[133] O. Higgott, D. Wang, and S. Brierley, “Variational quantum computation of excited states,” *Quantum*, vol. 3, p. 156, 2019.

[134] D. Wecker, M. B. Hastings, and M. Troyer, “Progress towards practical quantum variational algorithms,” *Physical Review A*, vol. 92, no. 4, p. 042303, 2015.

[135] D. Wang, O. Higgott, and S. Brierley, “Accelerated variational quantum eigensolver,” *Physical review letters*, vol. 122, no. 14, p. 140504, 2019.

[136] S. Sim, Y. Cao, J. Romero, P. D. Johnson, and A. Aspuru-Guzik, “A framework for algorithm deployment on cloud-based quantum computers,” *arXiv preprint arXiv:1810.10576*, 2018.

[137] J. R. McClean, M. E. Kimchi-Schwartz, J. Carter, and W. A. De Jong, “Hybrid quantum-classical hierarchy for mitigation of decoherence and determination of excited states,” *Physical Review A*, vol. 95, no. 4, p. 042308, 2017.

[138] J. R. McClean, Z. Jiang, N. C. Rubin, R. Babbush, and H. Neven, “Decoding quantum errors with subspace expansions,” *Nature communications*, vol. 11, no. 1, p. 636, 2020.

[139] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, “Barren plateaus in quantum neural network training landscapes,” *Nature communications*, vol. 9, no. 1, p. 4812, 2018.

[140] E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” *arXiv preprint arXiv:1411.4028*, 2014.

[141] M. Born, “Das adiabatenprinzip in der quantenmechanik,” *Zeitschrift für Physik*, vol. 40, no. 3-4, pp. 167–192, 1927.

[142] M. Born and V. Fock, “Beweis des adiabatensatzes,” *Zeitschrift für Physik*, vol. 51, no. 3-4, pp. 165–180, 1928.

[143] O. V. Besov, “Trudy matematicheskogo instituta imeni va steklova,” in *Proc. Steklov Inst. Math*, vol. 284, 2014, pp. 81–96.

[144] Y. Cao, G. G. Guerreschi, and A. Aspuru-Guzik, “Quantum neuron: an elementary building block for machine learning on quantum computers,” *arXiv preprint arXiv:1711.11240*, 2017.

[145] J. Romero and A. Aspuru-Guzik, “Variational quantum generators: Generative adversarial quantum machine learning for continuous distributions,” *Advanced Quantum Technologies*, vol. 4, no. 1, p. 2000003, 2021.

[146] J. S. Otterbach, R. Manenti, N. Alidoust, A. Bestwick, M. Block, B. Bloom, S. Caldwell, N. Didier, E. S. Fried, S. Hong *et al.*, “Unsupervised machine learning on a hybrid quantum computer,” *arXiv preprint arXiv:1712.05771*, 2017.

[147] E. Farhi and H. Neven, “Classification with quantum neural networks on near term processors,” *arXiv preprint arXiv:1802.06002*, 2018.

[148] P. Wittek and C. Gogolin, “Quantum enhanced inference in markov logic networks,” *Scientific reports*, vol. 7, no. 1, pp. 1–8, 2017.

[149] I. Kerenidis and A. Prakash, “Quantum recommendation systems,” *arXiv preprint arXiv:1603.08675*, 2016.

[150] E. Tang, “A quantum-inspired classical algorithm for recommendation systems,” in *Proceedings of the 51st annual ACM SIGACT symposium on theory of computing*, 2019, pp. 217–228.

[151] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, “Supervised learning with quantum-enhanced feature spaces,” *Nature*, vol. 567, no. 7747, pp. 209–212, 2019.

[152] M. Schuld and N. Killoran, “Quantum machine learning in feature hilbert spaces,” *Physical review letters*, vol. 122, no. 4, p. 040504, 2019.

[153] G. Verdon, J. Pye, and M. Broughton, “A universal training algorithm for quantum deep learning,” *arXiv preprint arXiv:1806.09729*, 2018.

[154] M. Mohseni, P. Read, H. Neven, S. Boixo, V. Denchev, R. Babbush, A. Fowler, V. Smelyanskiy, and J. Martinis, “Commercialize quantum technologies in five years,” *Nature*, vol. 543, no. 7644, pp. 171–174, 2017.

[155] M. Reiher, N. Wiebe, K. M. Svore, D. Wecker, and M. Troyer, “Elucidating reaction mechanisms on quantum computers,” *Proceedings of the national academy of sciences*, vol. 114, no. 29, pp. 7555–7560, 2017.

[156] J. Olson, Y. Cao, J. Romero, P. Johnson, P.-L. Dallaire-Demers, N. Sawaya, P. Narang, I. Kivlichan, M. Wasielewski, and A. Aspuru-Guzik, “Quantum information and computation for chemistry,” *arXiv preprint arXiv:1706.05413*, 2017.

[157] F. Verstraete and J. I. Cirac, “Renormalization algorithms for quantum-many body systems in two and higher dimensions,” *arXiv preprint cond-mat/0407066*, 2004.

[158] S. R. White, “Density matrix formulation for quantum renormalization groups,” *Physical review letters*, vol. 69, no. 19, p. 2863, 1992.

[159] U. Schollwöck, “The density-matrix renormalization group,” *Reviews of modern physics*, vol. 77, no. 1, p. 259, 2005.

[160] G. Vidal, “Entanglement renormalization,” *Physical review letters*, vol. 99, no. 22, p. 220405, 2007.

[161] P. Hayden, S. Nezami, X.-L. Qi, N. Thomas, M. Walter, and Z. Yang, “Holographic duality from random tensor networks,” *Journal of High Energy Physics*, vol. 2016, no. 11, pp. 1–56, 2016.

[162] A. Milsted and G. Vidal, “Tensor networks as conformal transformations,” *arXiv preprint arXiv:1805.12524*, 2018.

[163] J. Preskill, “Quantum computing and the entanglement frontier,” *arXiv preprint arXiv:1203.5813*, 2012.

[164] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, “Characterizing quantum supremacy in near-term devices,” *Nature Physics*, vol. 14, no. 6, pp. 595–600, 2018.

[165] A. W. Harrow and A. Montanaro, “Quantum computational supremacy,” *Nature*, vol. 549, no. 7671, pp. 203–209, 2017.

[166] A. Bouland, B. Fefferman, C. Nirkhe, and U. Vazirani, “Quantum supremacy and the complexity of random circuit sampling,” *arXiv preprint arXiv:1803.04402*, 2018.

[167] R. Movassagh, “Efficient unitary paths and quantum computational supremacy: A proof of average-case hardness of random circuit sampling,” *arXiv preprint arXiv:1810.04681*, 2018.

[168] C. Neill, P. Roushan, K. Kechedzhi, S. Boixo, S. V. Isakov, V. Smelyanskiy, A. Megrant, B. Chiaro, A. Dunsworth, K. Arya *et al.*, “A blueprint for demonstrating quantum supremacy with superconducting qubits,” *Science*, vol. 360, no. 6385, pp. 195–199, 2018.

[169] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo, “Quantum supremacy is both closer and farther than it appears,” *arXiv preprint arXiv:1807.10749*, 2018.

[170] S. Aaronson and A. Arkhipov, “The computational complexity of linear optics,” in *Proceedings of the forty-third annual ACM symposium on Theory of computing*, 2011, pp. 333–342.

[171] N. Spagnolo, C. Vitelli, M. Bentivegna, D. J. Brod, A. Crespi, F. Flamini, S. Giacomini, G. Milani, R. Ramponi, P. Mataloni *et al.*, “Experimental validation of photonic boson sampling,” *Nature Photonics*, vol. 8, no. 8, pp. 615–620, 2014.

[172] S. B. Bravyi and A. Y. Kitaev, “Quantum codes on a lattice with boundary,” *arXiv preprint quant-ph/9811052*, 1998.

[173] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, “Topological quantum memory,” *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.

[174] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, “Surface codes: Towards practical large-scale quantum computation,” *Physical Review A*, vol. 86, no. 3, p. 032324, 2012.

[175] P. Baireuther, T. E. O’Brien, B. Tarasinski, and C. W. Beenakker, “Machine-learning-assisted correction of correlated qubit errors in a topological code,” *Quantum*, vol. 2, p. 48, 2018.

[176] N. Ofek, A. Petrenko, R. Heeres, P. Reinhold, Z. Leghtas, B. Vlastakis, Y. Liu, L. Frunzio, S. Girvin, L. Jiang *et al.*, "Extending the lifetime of a quantum bit with error correction in superconducting circuits," *Nature*, vol. 536, no. 7617, pp. 441–445, 2016.

[177] A. Almheiri, X. Dong, and D. Harlow, "Bulk locality and quantum error correction in ads/cft," *Journal of High Energy Physics*, vol. 2015, no. 4, pp. 1–34, 2015.

[178] L. Susskind, "Dear qubitzers, gr= qm," *arXiv preprint arXiv:1708.03040*, 2017.

[179] J. Von Neumann, *Mathematical foundations of quantum mechanics: New edition*. Princeton university press, 2018, vol. 53.

[180] S. Axler, *Linear algebra done right*. Springer Science & Business Media, 1997.

[181] M. Artin, "Algebra (vol. 2.)," 2010.

[182] E. G. Rieffel and W. H. Polak, *Quantum computing: A gentle introduction*. MIT Press, 2011.

[183] J. B. Fraleigh, *A first course in abstract algebra*. Pearson Education India, 2003.

[184] D. S. Dummit and R. M. Foote, *Abstract algebra*. Wiley Hoboken, 2004, vol. 3.

[185] J. J. Rotman, *Advanced modern algebra*. American Mathematical Soc., 2010, vol. 114.

[186] F. W. Lawvere and S. H. Schanuel, *Conceptual mathematics: a first introduction to categories*. Cambridge University Press, 2009.

[187] T.-D. Bradley, "What is applied category theory?" *arXiv preprint arXiv:1809.05923*, 2018.

[188] E. Riehl, *Category theory in context*. Courier Dover Publications, 2017.

[189] F. Verstraete, V. Murg, and J. I. Cirac, "Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems," *Advances in physics*, vol. 57, no. 2, pp. 143–224, 2008.