FRIEDRICH-ALEXANDER-UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT • DEPARTMENT INFORMATIK

Lehrstuhl für Informatik 10 (Systemsimulation)

Lösen von Poissongleichungen auf einem simulierten Quantencomputer

Michael Holzmann

Bachelorarbeit

Lösen von Poissongleichungen auf einem simulierten Quantencomputer

Michael Holzmann

Bachelorarbeit

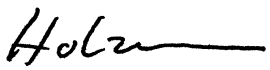
Aufgabensteller:	Prof. Dr. H. Köstler
Betreuer:	Prof. Dr. H. Köstler
Bearbeitungszeitraum:	08.11.2018 – 08.04.2019

Erklärung:

Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Der Universität Erlangen-Nürnberg, vertreten durch den Lehrstuhl für Systemsimulation (Informatik 10), wird für Zwecke der Forschung und Lehre ein einfaches, kostenloses, zeitlich und örtlich unbeschränktes Nutzungsrecht an den Arbeitsergebnissen der Bachelorarbeit einschließlich etwaiger Schutzrechte und Urheberrechte eingeräumt.

Erlangen, den 2. April 2019



Inhaltsverzeichnis

1 Einführung

Im Jahr 1941 stellte Konrad Zuse mit der Zuse Z3 die erste turingmächtige Maschine vor. Der Computer revolutinierte die Gesellschaft und ist heute noch kaum wegzudenken. Während der Entwicklung galt stets das Mooresche Gesetz, welches besagt, dass sich etwa alle 2 Jahre die Anzahl an Transistoren verdoppelt. Dieses Gesetz scheint jedoch seit 2016 langsam an die Grenze zu stoßen, weil die Transistoren so klein werden, dass physikalische Probleme auftauchen. Oft werden CPUs nur noch an der Anzahl der Kerne und weniger an der Frequenz gemessen. Der größte Supercomputer ist „Summit“ mit 2282544 Kernen und 122300 TeraFLOPS (Stand November 2018) [1]. Auch wenn die Parallelität von Computern gesteigert werden kann, so sind einige Aufgaben wie die Primfaktorzerlegung auf einem klassischen Rechner nicht effizient lösbar. Ablöse könnte eine neue Art von Rechenwerk, der Quantencomputer, sein. Quantencomputer basieren auf sogenannten Qubits und nutzen quantenmechanische E□ekte aus. Das Prinzip der Superposition und der Verschränkung erlaubt es spezielle Probleme schneller zu lösen als klassische Rechner. Es ist noch unbekannt, ob alle NP-Probleme damit e□zient gelöst werden können. Beispielsweise kann man bei der Berechnung von NP-vollständigen Problemen wie 3SAT mit Hilfe von Grover-Iterationen eine quadratische Laufzeitbeschleunigung erreichen. Probleme wie die Primfaktorzerlegung liegen in der Klasse BQP, welche von einem Quantencomputer polynomiell mit einer maximaler Fehlerwahrscheinlichkeit von 33% gelöst werden können. Auch das Lösen von linearen Gleichungssystemen kann mit einem Quantencomputer beschleunigt werden. Grundlage dafür ist der Algortihmus HHL09 [3] von Harrow, Hassidim und Seth Loyd, der bei linearen Gleichungen mit hermitischen Matrizen einen exponentiellen Speedup erreichen kann. Basis dieser Idee kann dafür genutzt werden, um Poissongleichungen zu lösen. In dem wissenschaftlichen Artikel „Quantum algorithm and circuit design solving the Poisson equation (Yudong Cao1, Anargyros Papageorgiou, Iasonas Petras, Joseph Traub and Sabre Kais)“ [2] wird gezeigt wie ein „ Quantenchip“ für beliebige Poissongleichungen aussehen könnte. Ziel der Bachelorarbeit ist es, konkrete Quantenschaltpläne zu entwickeln und diese anhand einer Quantensimulation auf einem klassischen Rechner nach zu vollziehen. Als Laufzeitumgebung für die Simulation des Quantencomputers wird das „Quantum Development Kit“ von Microsoft verwendet.

2 Quantum Computing - Rechenmodell und Algorithmen

2.1 Qubits und Hilbertraum

In einem klassischen Computer basieren alle logischen Operationen auf Spannungen und Stromflüsse innerhalb von Transistoren. In einem Quantencomputer ist die Ba-

[1] Top 500 Liste - https://www.top500.org/lists/2018/11/

siseinheit das Qubit. Die Realisierung des Qubits kann dabei sehr unterschiedlich erfolgen. Während das klassische Bit entweder im Zustand 0 oder 1 sein kann, kann das Qubit in einer Überlagerung von Basiszuständen sein. Dieser Zustand basiert auf dem quantenmechanischen E□ekt der Superposition. Der Zustand eines Qubits ist definiert durch:

$$|\Psi\rangle = c_1 |0\rangle + c_2 |1\rangle \;\; c_1, c_2 \in \mathbb{C}, \;\; |c_1|^2 + |c_2|^2 = 1 \tag{1}$$

Alternative:

$$|\Psi\rangle = \cos\frac{\theta}{2} e^{i\phi_1} |0\rangle + \cos\frac{\theta}{2} e^{i\phi_2} |1\rangle \;\; \phi_1, \phi_2 \in [0; 2\pi], \theta \in [0; \pi] \tag{2}$$

Eine Messung des Qubits führt mit einer Wahrscheinlichkeit von $|c_1|^2$ zum klassischen Zustand 0 und mit $|c_2|^2$ zu 1. In der Spin-$\frac{1}{2}$ Algebra ist bekannt, dass die globale Phase $\omega \in [0; 1]$ eines Quantenzustandes $e^{2\pi i\omega} |\Psi\rangle$ keine Rolle spielt.

$$|\Psi\rangle = e^{-i\phi_1} |\Psi\rangle = \cos\frac{\theta}{2} |0\rangle + \cos\frac{\theta}{2} e^{i(\phi_2 - \phi_1)} |1\rangle \tag{3}$$

$\phi_2 - \phi_1$ entspricht der zur $|0\rangle$ relativen Phase ϕ. Dies ermöglicht es das Qubit mit Hilfe zweier Polarkoordinaten als Blochkugel darzustellen. Der Zustandsraum zweier

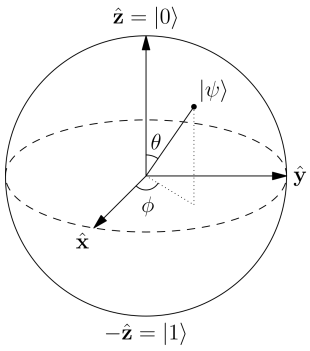


Abbildung 1: Darstellung eines Qubits als Bloch Kugel [2]

Qubits, die sich in den Zuständen $|\Psi_1\rangle$ und $|\Psi_2\rangle$ befinden ist definiert durch:

$$|\Psi\rangle = |\Psi_1\rangle \otimes |\Psi_2\rangle \tag{4}$$

$$= (c_1 |0\rangle + c_2 |1\rangle) \otimes (c_3 |0\rangle + c_4 |1\rangle) \;\; c_i \in \mathbb{C}$$

$$= c_1 c_3 |00\rangle + c_1 c_4 |01\rangle + c_2 c_3 |10\rangle + c_2 c_4 |11\rangle = \begin{pmatrix} c_1 c_3 \\ c_1 c_4 \\ c_2 c_3 \\ c_2 c_4 \end{pmatrix}$$

[2] Bildquelle:https://upload.wikimedia.org/wikipedia/commons/thumb/f/f4/Bloch_Sphere.svg/800px-Bloch_Sphere.svg.png

Ein Quantenregister aus n Qubits besitzt als Zustand einen 2^n dimensionalen Hilbertraum. Während ein Register eines klassischen Computers sich maximal in einem Zustand befinden kann, kann das Qubitregister 2^n Zuständen gleichzeitig annehmen. Operationen, die mit einem Quantenregister arbeiten, werden durch das Prinzip der Quantenparallelität gleichzeitig für alle klassischen Zustände durchgeführt.

2.2 Operationen mit einem Quantencomputer und Verschränkung von Qubits

Neben Messungen eines Quantensystems kann ein Quantencomputer ausschließlich unitäre Operationen durchführen. Das heißt, ein Quantensystem kann mit Hilfe von Messungen projezieren und durch unitäre Operationen umkehrbare Rotationen bzw. Spiegelungen durchführen. Dabei kann jede beliebige unitäre 2×2 Matrix durch Rotationen über drei verschiedene Achsen der Blochkugel erfolgen:

$$R_x(\phi) = \begin{pmatrix} \cos\frac{\phi}{2} & -i\sin\frac{\phi}{2} \\ -i\sin\frac{\phi}{2} & \cos\frac{\phi}{2} \end{pmatrix} R_y(\phi) = \begin{pmatrix} \cos\frac{\phi}{2} & -\sin\frac{\phi}{2} \\ \sin\frac{\phi}{2} & \cos\frac{\phi}{2} \end{pmatrix} R_z(\phi) = \begin{pmatrix} e^{-i\frac{\phi}{2}} & 0 \\ 0 & e^{i\frac{\phi}{2}} \end{pmatrix} \quad (5)$$

Zu den wichtigsten Operationen gehören:

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \; X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \; Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \; Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \; R_\phi = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\phi} \end{pmatrix}$$

Die parallele Ausführung von unitären Operationen U_i auf unterschiedliche Quantenzustände Ψ_i können dabei mit Hilfe des Kroneckerproduktes zur einer Operation auf den ganzen Hilbertraum umgeformt werden:

$$(U_1 |\Psi_1\rangle) \otimes (U_2 |\Psi_2\rangle) = (U_1 \otimes U_2)(|\Psi_1\rangle \otimes |\Psi_2\rangle) \quad (6)$$

Quantencomputer können mit Hilfe der Quantenverschränkung kontrollierte Operationen durchführen. Dabei werden sie nur dann ausgeführt, falls ein oder mehrere Kontrollqubits jeweils den Zustand $|1\rangle$ haben. Mit Hilfe dieser Art von Transformationen können beliebige Logikgatter und Logikschaltkreise aus der klassischen Informatik implementiert werden. Abbildung 3 zeigt folgenden Quantenzustand:

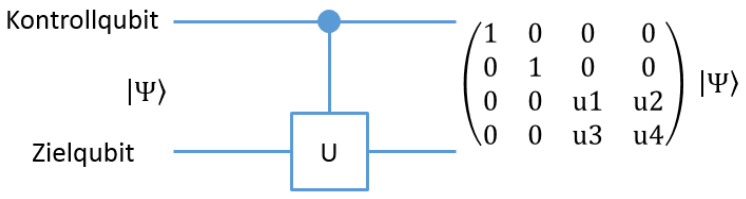


Abbildung 2: Kontrollierte Operation U auf den Zustand Ψ

$$(H |0\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) \xrightarrow{\text{C - NOT}} \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

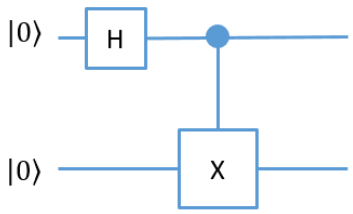


Abbildung 3: Quantenverschränkung mit Hilfe des C-NOT Gatters

Eine Messung eines Qubits würde mit einer Wahrscheinlichkeit von 50% zu $|0\rangle$ resultieren und den Zustand des anderen Qubits ebenso auf $|0\rangle$ setzen. Die Messung des einen Qubits beeinflusst also den Zustand des anderen. Die Qubits sind verschränkt zueinander und teilen sich beiden einen nicht separierbaren Zustand.

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \neq |\Psi_1\rangle \otimes |\Psi_2\rangle \;\; \Psi_1, \Psi_2 \; beliebig$$

Dabei ist zu beachten, dass es sich hierbei nicht um eine Zustandskopie handelt. In Quantensystemen ist es nach dem „No Cloning Theorem“ nicht möglich Quantenzustände zu kopieren. Das „No Cloning Theorem“ ist daher die Basis für die moderne Quantenkryptografie. Für einen Quantensimulator wie das Quantum Development Kit ist es wichtig zu wissen, ob ein System separierbar ist. Ist es komplett verschränkt zueinander, so muss der ganze Hilbertraum gespeichert werden, was zu einem exponentiellen Arbeitsspeicheraufwand führt.

2.3 Quanten Fourier Transformation

Mit einem Quantencomputer können auch komplexe Funktionen wie die diskrete Fouriertransformation implementiert werden[1]. Die Quantenfouriertransformation (QFT) die auf n = $\log_2 N$ angewendet wird, überführt einen N dimensionalen Hilbertraum in einen Quantenzustand, dessen Amplituden den Fourierkoe□zenten entsprechen:

$$\sum_{j=0}^{N-1} f_j |j\rangle \xrightarrow{\text{QFT}} \sum_{k=0}^{N-1} g_k |k\rangle \quad mit \; g_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} e^{2\pi i j k/N} f_j \tag{7}$$

R_i in Abbildung 4 entspricht einem Phasenverschiebungsgatter R_ϕ, bei dem um die relative Phase $e^{\frac{2\pi j}{2^i}}$ mit $j = \sqrt{-1}$ gedreht wird. Durch die Umkehrung des Schaltkreises mit adjungierten Rotationen $R_i^\dagger$ kann die inverse Fourier Transformation $QFT^\dagger$ implementiert werden. QFT selbst ist ein wichtiger Teil vieler Algorithmen und kann genutzt werden, um Periodizitäten diskreter Quantenorakel herauszufinden.

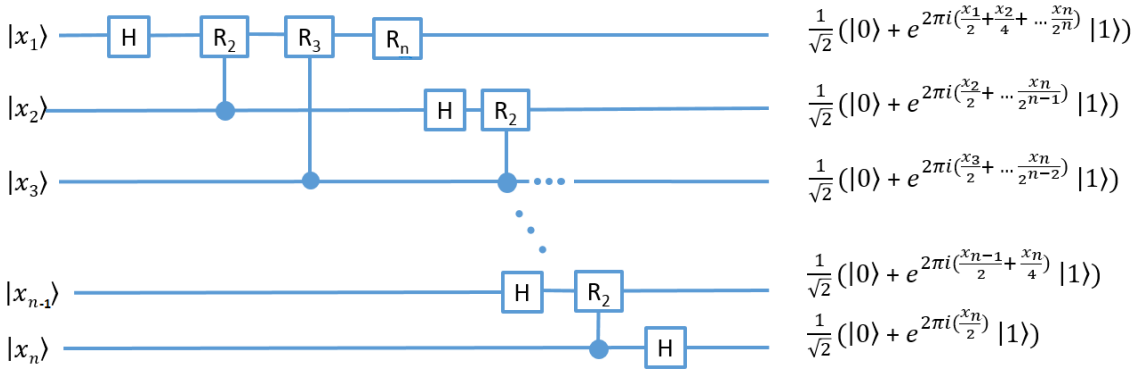


Abbildung 4: Diskrete Quantenfouriertransformation

2.4 Der Phase Kickback und der Phasenbestimmungsalgorithmus

Sei U eine unitäre Operation mit Eigenzustand Ψ und Eigenwert $e^{2\pi i\omega}$, $\omega \in [0;1]$. Die kontrollierte Operation von U hat folgenden E□ekt:

$$\frac{1}{\sqrt{2}}(|0\rangle|\Psi\rangle + |1\rangle|\Psi\rangle) \xrightarrow{\text{C-U}} \frac{1}{\sqrt{2}}(|0\rangle|\Psi\rangle + |1\rangle e^{2\pi i\omega}|\Psi\rangle)$$

$$= \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\omega}|1\rangle) \otimes |\Psi\rangle$$

Wenn sich das Kontrollqubit in einer Superposition befindet und das Zielregister im Eigenzustand von U ist, wird die relative Phase des Kontrollqubits um die Phase $2\pi\omega$ verschoben. Dieses Phänomen wird „Phase Kickback“ genannt und dient als Basis für viele Quantenalgorithmen wie beispielsweise dem Algorithmus von Deutsch und Josza oder dem Phasenbestimmungsalgorithmus. Das Ziel des Phasenbestimmungsalgorithmuses (PEA) ist es, die Phase ω zu bestimmen, um wie viel sich die globale Phase des Eigenzustandes $|\Psi\rangle$ verschiebt, wenn U darauf angewendet wird:

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

PEA kann also dafür genutzt werden, um eine Eigenwertbestimmung durchzuführen. Abbildung 5 zeigt die Auswirkungen des Algorithmuses auf das Quantensystem, welches aus $|\Psi\rangle$und n= $\log_2 N$ Qubits besteht:

$$|0_1\rangle \otimes |0_2\rangle \otimes \ldots |0_n\rangle \otimes |\Psi\rangle \xrightarrow{H_n} \frac{1}{\sqrt{N}}(|0\rangle + |1\rangle)^{\otimes n} \otimes |\Psi\rangle$$

Die Ausführung der kontrollierten Operationen verschieben die Phase der Kontrollqubits:

$$\frac{1}{\sqrt{N}}((|0\rangle + e^{2\pi i 2^{n-1}\omega}|1\rangle) \otimes (|0\rangle + e^{2\pi i 2^{n-2}\omega}|1\rangle) \otimes \ldots (|0\rangle + e^{2\pi i 2^0\omega}|1\rangle)) \otimes |\Psi\rangle$$

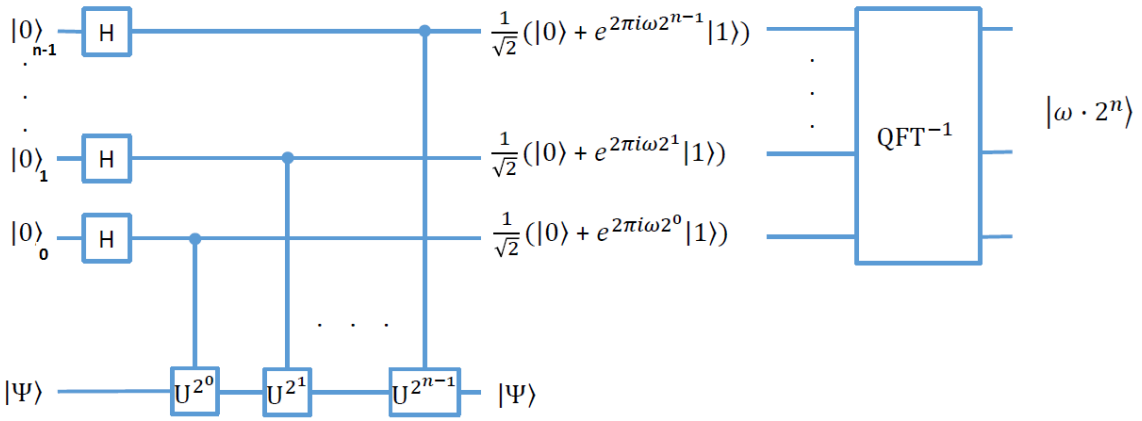


Abbildung 5: Phasenbestimmungsalgorithmus

$$= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i k\omega} |k\rangle \otimes |\Psi\rangle = \mathsf{QFT}(|2^n\omega\rangle) \otimes |\Psi\rangle$$

Wenn nun die inverse Quantenfouriertransformation auf das Kontrollregister angewendet wird, so ergibt sich der Zustand $|2^n\omega\rangle \otimes |\Psi\rangle$. Falls statt $|\Psi\rangle$ ein beliebiger Zustand $|\Phi\rangle$ benutzt wird, so kann $|\Phi\rangle$ als Superposition von Eigenzuständen u_i dargestellt werden:

$$|\Phi\rangle = \sum_j b_j |u_i\rangle \text{ mit } b_j = \langle\Phi|u_j\rangle \tag{8}$$

Die Anwendung des Phasenbestimmungsalgorithmus auf $|\Phi\rangle$ erzeugt folgenden verschränkten Zustand:

$$|00...0_n\rangle |\Phi\rangle \xrightarrow{\text{PEA}} \sum_j b_j |\omega_j 2^n\rangle |u_j\rangle \tag{9}$$

Eine Messung des Kontrollregisters würde mit Wahrscheinlichkeit $|b_j|^2$ zufällig zum Zustand $|\omega_j 2^n\rangle |u_j\rangle$ führen. Diese Superpostion kann benutzt werden, um lineare Gleichungssysteme zu lösen, welches in Kapitel 3 genauer erläutert wird.

2.5 Orakelkonstruktion in Q#

Bevor Quantenschaltkreise auf einem echten Quantencomputer implementiert werden, werden sie auf einem klassichen Computer getestet. Durch Quantensimulationen können Algorithmen validiert werden und können so ein Hilfsmittel für den Entwurf von Quantenorakel sein. Es gibt viele Quantensimulatoren, die einerseits nur für die Simulation von Quantensystemen, andererseits auch für die Integration auf einem echten Quantencomputer geeignet sind. Es gibt Quantensimulatoren wie Quipper, die auf

funktionalen Sprachen basieren, als auch welche, die imperativ arbeiten und syntaktisch modernen Hochsprachen ähneln. Für diese Arbeit wird das Quantum Development Kit benutzt, welches im Dezemeber 2017 von Microsoft veröentlicht worden ist. Der Quantensimulator wird dabei von einer Treiberklasse und einer Operatorklasse definiert. Die Treiberklasse ruft die Simulation auf und kann mit einer .NET Sprache wie C# oder in Python implementiert werden. Die Operationen auf den Qubits selbst werden in der Operatorklasse in der Sprache Q# beschrieben. Q# selbst ist imperativ und unterstützt wie moderne Hochsprache Typkonstrukte, Bedingungen und Schleifen. Operationen können dabei Parameter und Rückgabewerte haben. Innerhalb einer Operation können unitäre Tranformationen und Messungen durchgeführt werden. Um Qubits zu erhalten, muss man sie durch einen using -Block initialisieren. Am Ende eines using-Blocks müssen die Qubits wieder freigegeben werden, sodass sie im Nullzustand sind. Dies kann durch die Operation „ResetAll(Qubit[])“ geschehen. Die globale Phase kann dabei verschoben sein und ignoriert werden. Das Quantum Development Kit verfügt über viele Debug-Möglichkeiten wie Assertionen oder Zustandsmessungen. Zudem automatisiert das System Verfahren wie Verschränkung. Wird in der Operatordefiniton das Schlüsselwort „controlled auto“ verwendet, so baut das Quantum Development System diesen Operator für beliebig viele Kontrollqubits. Beispielsweise kann die kontrollierte Operation U dann einfach durch den Aufruf „(Controlled U)(Kontrollregister, Zielregister) “ verwendet werden. Dies ermöglicht es, Schaltkreise wie AdvancedCNOT zu definieren. AdvancedCNOT ist eine Bitflip-Operation auf das Zielregister, falls das Kontrollregister einer gegebenen Bitmaske entspricht. Damit können beliebige boolische Quantenschaltkreise implementiert werden, indem pro Ein- und Ausgabecodierung ein AdvancedCNOT verwendet wird. Die vierte Zeile von Abbildung 15 wird durch AdvancedCNOT(Kontrollregister, Zielregister, [false;true;false;true], [false;false;true;true;false;false;true;true]) implementiert. Die Ausführung dieser Operation auf ein Zielregister im Nullzustand ist:

$$|0000\rangle\,|00000000\rangle \xrightarrow{\text{AdvCNOT}} |0000\rangle\,|00000000\rangle$$
$$|0101\rangle\,|00000000\rangle \xrightarrow{\text{AdvCNOT}} |0101\rangle\,|00110011\rangle$$

Die Implementierung dieses Gatters wird in Q# wie folgt implementiert:

```
operation advancedCNOT(controlregister:Qubit[], target:Qubit[],
   controlmodifier:Bool[],targetmodifier:Bool[]):(){
    body{
        for(i in 0..Length(controlregister)-1){
            if(!controlmodifier[i]){
                X(controlregister[i]);
            }
        }
        for(i in 0..Length(target)-1){
           if(targetmodifier[i]){
                (Controlled X)(controlregister,target[i]);
            }
        }
        for(i in 0..Length(controlregister)-1){
```

```
            if(!controlmodifier[i]){
                X(controlregister[i]);
            }
        }
    }
        adjoint auto controlled auto adjoint controlled auto
}
```

3 Lösen von Poissongleichungen mit einem simulierten Quantencomputer

3.1 Problemdefinition und Anforderungen der Lösung

Ziel ist es beliebig dimensionale Poissongleichungen mit Dirichlet Randbedingungen zu lösen, die wie folgt definiert sind:

$$\Delta \text{u(x)} = f(x) \ , \ \text{x in } \Omega \ \ \text{f, g beliebig}$$
$$\text{u(x)} = g(x) \ , \ \text{x auf } \delta\Omega \ \ \Omega = (0,1)^d$$

x befindet sich dabei im d - dimensionalen Einheitsraum Ω. Eine Variante, um die Lösung u(x) zu bekommen, ist es den Einheitsraum in M+1 Gitterpunkte zu unterteilen und einen Lösungungsvektor $u^{(m-1)^d}$ mit Hilfe eines linearen Geichungssystems zu lösen. M beschreibt für diese Arbeit einen Exponenten der Basis 2. Das Gleichungssystem mit $h = \frac{1}{M}$ sieht in der ersten Dimension wie folgt aus:

$$\frac{1}{h^2} A_{d=1} \begin{pmatrix} u_1 \\ \\ \vdots \\ \\ u_{m-1} \end{pmatrix} = \frac{1}{h^2} \begin{pmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ 0 & \ddots & \ddots & \\ & & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ \\ \vdots \\ \\ u_{m-1} \end{pmatrix} = \begin{pmatrix} f_1 + \frac{1}{h^2} u_0 \\ \\ \vdots \\ \\ f_n + \frac{1}{h^2} u_m \end{pmatrix} = \vec{b} \qquad (10)$$

Die Poissonmatrix $A_{d=2}$ für die zweite Dimension kann mit Hilfe von $A_{d=1}$ konstruiert werden:

$$A_{d=2} = \begin{pmatrix} A_{d=1} + 2I_{M-1} & -I_{M-1} & & 0 \\ -I_{M-1} & \ddots & \ddots & \\ 0 & \ddots & \ddots & \\ & & -I_{M-1} & A_{d=1} + 2I_{M-1} \end{pmatrix} \text{ mit } I_{M-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ & & \ddots & \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad (11)$$

Für beliebige Dimensionen d lässt sich A_d aus einer Summe von Kroneckerprodukten darstellen[2]:

$$A_d = A_{d=1} \otimes I \otimes ...I + I \otimes A_{d=1} \otimes I \otimes ...I + ... + I \otimes ...A_{d=1} \qquad (12)$$

Diese Zerlegung spielt eine wichtige Rolle bei der Implementierung der Hamiltoniansimulation dieser Matrix.

Um $u^{(m-1)^d}$ mit M+1 Gittepunkten und d Dimensionen zu erhalten, muss ein lineares Gleichungssystem mit einer $(M-1)^d \times (M-1)^d$ Matrix gelöst werden. Klassiche Verfahren würden dieses Problem beispielsweise mit iterativen Verfahren wie dem CG-Verfahren in $\mathcal{O}((M-1)^d)$ lösen. Das Verfahren, welches in dem Artikel "Quantum algorithm and circuit design solving the Poisson equation"[2] vorgestellt wird, kann e□ektiv eine exponentielle Laufzeitbeschleunigung mit Hilfe eines Quantencomputers erzielen. In einem Quantensystem können Zustände nur einen normierten Hilbertraum annehmen. Aus diesem Grund muss zur Eingabecodierung der Eingabevektor $\vec{b}$ und zur Ausgabencodierung der Lösungsvektor $\vec{u}$ die Länge 1 haben. Das bedeutet, dass für beliebig normierte Eingaben $\vec{b}$ der Quantenalgorithmus nur die normierte Lösung $\tilde{u}$ der Gleichung ausgibt. Folglich liegt die Lösung im Raum $\tilde{\vec{u}} \cdot \alpha$ mit α beliebig. Wird als Eingabe ein normierter Eigenvektor der Poissonmatrix eingegeben, so erzeugt der Quantenalgorithmus keine weiteren Informationen, da die Lösung der Eingabe entspricht.

3.2 Der Quantenalgorithmus

Abbildung 6 zeigt den Schaltkreis für den Quantenalgorithmus, der in "Quantum algorithm and circuit design solving the Poisson equation"gezeigt wird. Der Algorithmus

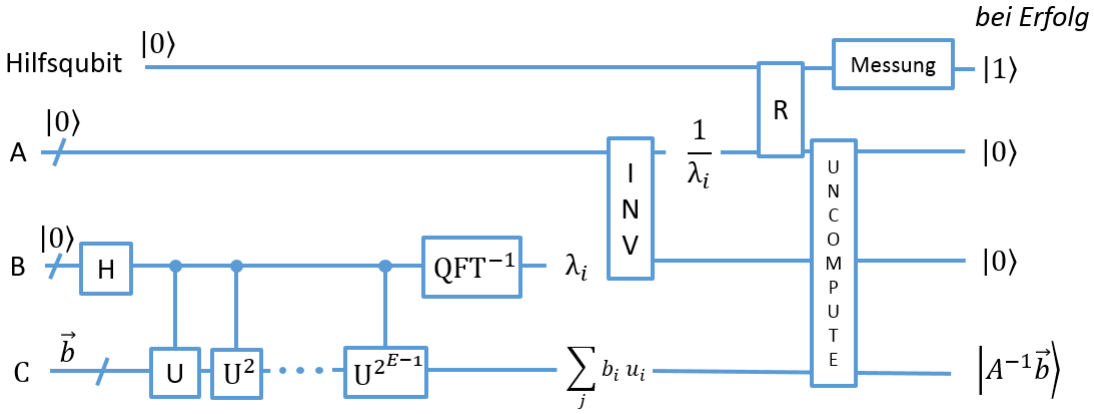


Abbildung 6: Schaltkreis für das Lösen der Poissongleichung

besteht dabei aus folgenden Schritten:

- Konstruiere in Register C den Zustand $\sum_j \beta_j |j\rangle$. Dabei entsprechen β_j den Elementen des normierten Eingabevektors $\vec{b}$.
- Wende den Phasenbestimmungsalgorithmus (PEA) mit Hilfe eines Kontrollregisters B und der Hamiltoniansimulation U = $e^{\frac{1}{h^2} A t}$ mit t = $2\pi i \frac{1}{2^E}$ auf das

Register C an. E entspricht dabei die Anzahl der Qubits von Register B. Register C und B befinden sich dann im verschränkten Zustand $\sum_j b_j |\sim \lambda_j\rangle |\vec{u_j}\rangle$ mit $b_j = \langle\vec{b}|\vec{u_j}\rangle$. λ_j ist der zum Eigenvektor $\vec{u_j}$ gehörende Eigenwert.

- Berechne den Kehrwert $\frac{1}{\lambda_j}$in Register A. Das System befindet sich dadurch in $\sum_j b_j |\frac{1}{\sim\lambda_j}\rangle |\sim \lambda_j\rangle |\vec{u_j}\rangle$
- Versetzte das System mit Hilfe kontrollierter Rotationen auf ein Hilfsqubit in den Zustand: $\sum_j b_j |\frac{1}{k_j}\rangle |k_j\rangle |\vec{u_j}\rangle (\sqrt{1-\frac{\alpha^2}{k_j^2}}) |0\rangle + \frac{\alpha}{k_j} |1\rangle)$ mit $k_j =\sim \lambda_j$ und dem Amplitudenfaktor α, der benutzt werden kann, um die Erfolgswahrscheinlichkeit des Algorithmuses zu verbessern.
- Kehre mit Hilfe adjungierter Operationen alle Berechnungen um, die in Register B und C statt gefunden haben. Das heißt die Kehrwertberechnung und PEA werden rückgängig gemacht, wodurch die Qubits in Register C und B im Nullzustand sind. Das Quantensystem lässt sich dann wie folgt beschreiben: $\sum_j b_j |0\rangle |0\rangle |\vec{u_j}\rangle (\sqrt{1-\frac{\alpha^2}{k_j^2}}) |0\rangle + \frac{\alpha}{k_j} |1\rangle)$
- Führe eine Messung an dem Hilfsqubit durch. Ist es in dem Zustand $|1\rangle$, so hat der Algorithmus Erfolg, weil der Zustand in den Raum der normierten Lösung projeziert worden ist: C$\sum_j b_j \frac{\alpha}{k_j} |\vec{u_j}\rangle |1\rangle = CA^{-1}\vec{b}|1\rangle$. C beschreibt eine Normalisierungskonstante und sorgt für die Normiertheit des Systems. Ist der Zustand des gemessenen Qubits $|0\rangle$, so ist der Algorithmus fehl geschlagen und muss von Vorne gestartet werden.

3.3 Die Konstruktion des Eingabevektors $\vec{b}$

Die Konstruktion eines beliebig normierten Eingabevektors $\vec{b}^{(M-1)^d}$ zu $\sum_{j=1}^{(M-1)^d} \beta_j |j\rangle$ verlangt $\log_2(M^d)$ Qubits. Falls die eindimensionale Poissongleichung gelöst werden soll, so hat Register C $\log_2(M)$ Qubits und kann M verschiedene klassische Zustände (gleichzeitig) annehmen. Hier ist zu beachten, dass das Quantenregister einen Zustand mehr zur Verfügung hat, da nur M-1 Zustände codiert werden müssen. Deshalb dient Zustand $|1\rangle$ und nicht $|0\rangle$ als Eingabe- und Ausgabezustand für den Wert β_1. Zustand $|0\rangle$ kommt daher nicht vor und hat somit die Amplitude 0. Die Kontruktion von $|0..0_{\log_2 M}\rangle$ zu $\sum_{j=1}^{M-1} \beta_j |j\rangle$ kann dabei mit Hilfe eines Quantenorakels erfolgen. In dem Artikel „E□cient state preparation for a register of quantum bits“[8] wird beschrieben, dass $\vec{b}$ mit einer polynomiellen Gatterlaufzeit bezüglich Qubitanzahl erzeugt werden kann, solange die Amplituden β_i e□zient berechnet werden können. In dieser Arbeit werden Eingaben verwendet, die sehr einfach mit Hilfe weniger Gatter zu realisieren sind. Beispielsweise kann $\vec{b} = \begin{pmatrix}1\\0\\0\end{pmatrix}$ so erzeugt werden:

```
(using b = Qubits[2]){
    X(b[0]);
}
```

oder $\vec{b} = \frac{1}{2}\begin{pmatrix}1 & 0 & 1 & 0 & 1 & 0 & 1\end{pmatrix}^T$

```
(using b = Qubits[3]){
    X(b[0]);
    H(b[1]);
    H(b[2]);
}
```

Um kompliziertere Eingaben zu konstruieren können über das Quantum Development Kit mit der Operation „PrepareArbitraryState(coe□cients: ComplexPolar[], qubits : BigEndian)“ die Amplituden direkt gesetzt werden. Bei Poissongleichungen mit d Dimensionen sind $\mathcal{O}(\log_2(M)d)$ Qubits für die Eingabe nötig. Dabei ist jeder klassische Zustand $|j\rangle$ nur dann valide, falls er folgende Eigenschaft erfüllt:

$$|j\rangle = |\Psi_1\rangle \otimes |\Psi_2\rangle \otimes ... \otimes |\Psi_d\rangle \text{ mit } |\Psi_i\rangle \neq |0\rangle$$

$|\Psi_i\rangle$ beschreibt den Zustand für einen $\log_2 M$ Qubitblock. Der Grund für diese Bedingung ist die parallele Ausführung der Hamiltoniansimulation der eindimensionalen Poissongleichung, welches in Kapitel 3.4.2 genauer erläutert wird. Da Zustand $|0\rangle$ stets die Amplitude null hat, darf auch bei einer validen Konstruktion einer multidimensionalen Eingabe $|\Psi_i\rangle$ nicht $|0\rangle$ sein. Somit sind alle Zustände invalide, deren binäre Repräsentation einen $\log_2(M)$ Qubit Block besitzen, der im Zustand $|0\rangle$ ist. Bei M=4 und d=2 wird β_1 nicht durch $|0001\rangle = |1\rangle$, sondern durch $|0101\rangle = |5\rangle$ codiert, da $|1\rangle$ aus $|00\rangle \otimes |01\rangle$ besteht und die Eigenschaft nicht erfüllt. Der Algorithmus ersetzt bei Erfolg das Eingaberegister mit der Lösung $\vec{u}$, weshalb diese Zustandscodierung ebenfalls für die Ausgabe zutri□t. Abbildung 18 zeigt die Codierungstabelle für die Ein- und Ausgabe in Register C bei M=4 und d=2 ($(M-1)^d$ Zustände).

i	Zustand im Register C
1	$\|5\rangle$
2	$\|6\rangle$
3	$\|7\rangle$
4	$\|9\rangle$
5	$\|10\rangle$
6	$\|11\rangle$
7	$\|13\rangle$
8	$\|14\rangle$
9	$\|15\rangle$

Abbildung 7: Ein- und Ausgabecodierung für M=4 und d=2

3.4 Phasenbestimmung einer Hamiltoniansimulation

Ziel dieses Kapitels ist es, den Eingabezustand $\vec{b}$ mit Hilfe des Phasenbestimmungsalgorithmuses in eine Superposition von verschränkten Zuständen $\sum_j b_j \left|\sim \lambda_j\right\rangle \left|\vec{u_j}\right\rangle$ zu bringen. Wichtig dabei ist es, die eindimensionale Poissonmatrix $\frac{1}{h^2}A_1$ zu simulieren. Da diese Matrix nicht unitär ist, muss diese Operation durch eine Hamiltoniansimulation $e^{\frac{1}{h^2}A_1 t}$ durchgeführt werden. Hamiltonian Simulationen spielen bei der Simulation von Quantensystemen eine wichtige Rolle. Mit Hilfe von Hamiltoniansimulationen können Dynamiken von Molekülsystemen simuliert werden, weshalb dies das wichtigste Anwendungsgebiet in der „Quantum Chemistry“ ist. Es kann gezeigt werden[7], dass dünnbesezte und hermitische Matrizen A e□zient simuliert werden können, solange A durch eine Linearkombination von einfach implementierbaren Matrizen darstellbar ist. Seien $\lambda_1, \lambda_2, ...\lambda_{M-1}$ die Eigenwerte von $\frac{1}{h^2}A_{d=1}$ mit $\lambda_i \leq \lambda_{i+1}$, so sind $e^{t\lambda_i}$ die Eigenwerte der Hamiltoniansimulation. Wird für t = $2\pi i k$ und $k = \frac{1}{2^E}$ (mit Register B aus E Qubits) gewählt, so entspricht die Phase von $e^{2\pi i k \lambda_i}$ den $\frac{1}{2^E}$ -fachen Eigenwerte von $\frac{1}{h^2}A_1$. Die Phasenbestimmung der Hamiltoniansimulation erzeugt aus $\vec{b}$ den folgenden Zustand:

$$\left|0...0_E\right\rangle \otimes \left|\vec{b}\right\rangle \xrightarrow{\text{PEA}} \sum_{j=1}^{M-1} b_j \left|\sim \lambda_j k \cdot 2^E\right\rangle \left|\vec{u_j}\right\rangle = \sum_{j=1}^{M-1} b_j \left|\sim \lambda_j\right\rangle \left|\vec{u_j}\right\rangle \tag{13}$$

3.4.1 Hamiltoniansimulation einer eindimensionalen Poissonmatrix

Die e□ziente Implementierung der Hamiltoniansimulation gehört zu den schwierigsten Problemen des Algorithmuses. Für das Matrixexponential $e^{\frac{t}{h^2}A_1}$ kann der Spektralsatz verwendet werden [2]:

$$e^{\frac{t}{h^2}A_1} = S e^{t\Delta} S \text{ mit } \Delta = \begin{pmatrix} \lambda_1 & 0 & 0 & 0 \\ 0 & \lambda_2 & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_{M-1} \end{pmatrix} \tag{14}$$

S beschreibt dabei die Matrix der diskreten Sinustransformation mit Größe $(M-1) \times (M-1)$. $S_{i,j} = \sqrt{\frac{2}{M}} \sin(\frac{\pi i j}{2M})$ $i, j = 1, ..., M-1$. Die Implementierung dieser Matrix geschieht mit Hilfe der Quantenfouriertransformation. Folgender Zusammenhang

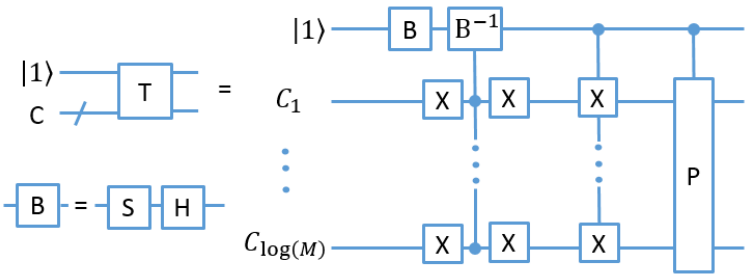


Abbildung 8: Schaltkreis für die Operation T

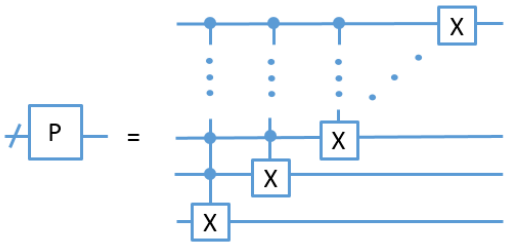


Abbildung 9: Schaltkreis für die Operation P. P inkrementiert das Eingaberegister um dem Wert eins.

existiert zwischen DFT und DST[4]:

$$T^{\dagger}\ QFT\ T = C \otimes -iS = \begin{pmatrix} C & 0 \\ 0 & -iS \end{pmatrix} \text{ mit } T = \begin{pmatrix} 1 & & & & & & \\ & \frac{1}{\sqrt{2}} & & & & \frac{i}{\sqrt{2}} & \\ & & \ddots & & \ddots & & \\ & & & \frac{1}{\sqrt{2}} & & & \frac{i}{\sqrt{2}} \\ & & & & 1 & & \\ & & & \frac{1}{\sqrt{2}} & & & \frac{-i}{\sqrt{2}} \\ & & \iddots & & & \iddots & \\ & \frac{1}{\sqrt{2}} & & & & \frac{-i}{\sqrt{2}} & \end{pmatrix} \tag{15}$$

C beschreibt dabei eine diskrete $(M+1) \times (M+1)$ Cosinustransformation und QFT die diskrete Quantenfouriertransformation der Größe $2M \times 2M$ (durch Hinzugabe eines Hilfsqubits).Abbildung 8 zeigt die Implementierung der unitären Matrix T.

```
//Operation T in Q#
operation T(qubits: Qubit[], ancilla:Qubit): () {
    body {
        B(ancilla);
        for (i in 0..Length(qubits) - 1) {
            X(qubits[i]);
        }
```

```
        (Controlled Adjoint B)(qubits[0..Length(qubits) - 1],
            ancilla);
        for (i in 0..Length(qubits) - 1) {
            X(qubits[i]);
            CNOT(ancilla, qubits[i]);
        }
         (Controlled P)([ancilla], qubits[0..Length(qubits) - 1]);
    }
    adjoint auto controlled auto adjoint controlled auto
}
```

Wenn das Hilfsqubit als Initialwert den Zustand $|0\rangle$ hat, so wird der obere linke Block der Matrix durchgeführt. Da die diskrete Sinustransformation implementiert werden muss, wird das Hilfsqubit vorher auf $|1\rangle$ gesetzt, wodurch der untere rechte Block und damit die Matrix $\begin{pmatrix} c & 0 \\ 0 & -iS \end{pmatrix}$ durchgeführt wird. c $\in \mathbb{C}$ beschreibt dabei den letzten Eintrag der Matrix C. Weil durch die in Kapitel 3.3 definierte Konstruktionsvorschirft der Zustand $|0\rangle$ nicht existiert, kommt die Ausführung der ersten Zeile nicht vor, wodurch c keine Rolle spielt. Um die Hamiltoniansimulation $e^{\Delta t}$ zu konstruieren, müssen zuerst die Eigenwerte von A_1 e□zient berechnet werden. A_1 ist eine tridiagonale Matrix in der Form einer Toeplitzmatrix und hat somit die Eigenwerte[5]:

$$\lambda_j = 4M^2 \sin^2(\frac{j\pi}{2M}) \text{ mit } j = 1, ..., M-1 \tag{16}$$

Ziel ist es also ein Quantenorakel zu entwerfen, welches $|j\rangle\,|00..0_s\rangle$ zu einen verschränkten Zustand $|j\rangle\,|\lambda_j\rangle$ transformiert. Dabei bestimmt s die Anzahl der Qubits, welche λ_j codieren. In dem Referenzartikel[2] wird beschrieben wie der Sinus mit wiederholten Quadrieren der Taylorentwicklung einer komplexen Zahl approximiert werden kann.

$$\sin(x) = \Im(e^{ix}) = \Im((e^{ix/r})^r) \text{ mit } r = 2^b \; b \in \mathbb{R} \tag{17}$$

$$e^{ix/r} \sim 1 - \frac{ix}{r} + \frac{x^2}{r^2} = W_1 \tag{18}$$

Durch kontrollierte Operationen können beliebige Logikgatter der klassischen Informatik implementiert wird. Der wissenschaftliche Artikel „Quantum Networks for Elementary Arithmetic Operations“[9] zeigt, dass dadurch auch arithmetische Operationen wie Addition und Multiplikation durchgeführt werden können. Im ersten Schritt muss W_1 erzuegt werden. Da diese Zahl komplex ist, werden zwei Register der Größe s benötigt, die den imaginären und reellen Teil darstellen.

$$|j\rangle\,|0\rangle^s\,|0\rangle^s \rightarrow |j\rangle\,|\frac{\pi j}{r2M}\rangle\,|1 - \frac{\pi^2 j^2}{r^2 4M^2}\rangle \tag{19}$$

Die Divisionen selbst müssen nicht arithmetisch implementiert werden. Da r und M Exponenten der Basis 2 sind, kann die Division durch Shiften oder Uminterpretation der Qubitwertigkeit geschehen. W_1 wird dann schrittweise $\log_2(r)$ mal quadriert:

$$|\Im(W^{2^k})\rangle\,|\Re(W^{2^k})\rangle\,|0\rangle^s\,|0\rangle^s \rightarrow |\Im(W^{2^k})\rangle\,|\Re(W^{2^k})\rangle\,|\Im(W^{2^{k+1}})\rangle\,|\Re(W^{2^{k+1}})\rangle \;\; k = 1, ..., \log_2(r) \tag{20}$$

Dabei werden pro Schritt 2 Register aus s Qubits zusätzlich benötigt. Nach diesen Schritten muss das Register $|\Im(W^r)\rangle$ noch mit $\frac{4M^2 2\pi}{2^E} 2^e$ multipliziert werden. Nach

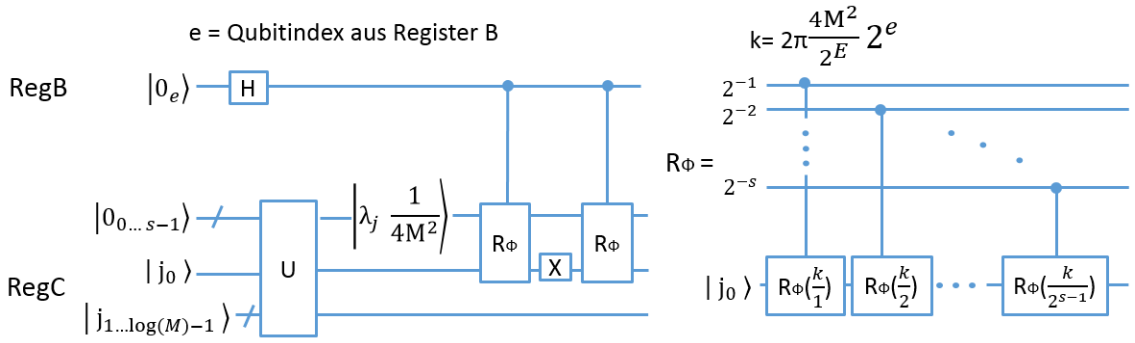


Abbildung 10: Schaltkreis zum Phasekickback der Eigenwerte

dem die Eigenwerte λ_j berechnet wurden, müssen die relative Phasen der Kontrollqubits aus Register B mit Hilfe des „Phase Kickbacks“ um den Eigenwert verschoben werden. Dies kann erreicht werden, indem man eine globale Phasenverschiebung mit den Eigenwerten auf Register C anwendet. Diese Operation kann mit Hilfe von 2s kontrollierten Phasenverschiebungsgattern R_ϕ realisiert werden, welche in Abbildung 10 zu sehen sind. Dabei kann der Faktor k= $\frac{4M^2 2\pi}{2^E} 2^e$ direkt in den Phasenverschiebungsgattern als skalarer Faktor benutzt werden. Der erzeugte Zustand, welchen der Schaltkreis zeigt, befindet sich in:

$$\begin{aligned}
\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |j\rangle\,|0\rangle^s &= \frac{1}{\sqrt{2}}(|0\rangle\,|j\rangle\,|0\rangle^s + |1\rangle\,|j\rangle\,|0\rangle^s) \\
&\xrightarrow{U} \frac{1}{\sqrt{2}}(|0\rangle\,|j\rangle\,|\lambda_j \frac{1}{4M^2}\rangle + |1\rangle\,|j\rangle\,|\lambda_j \frac{1}{4M^2}\rangle) \\
&\xrightarrow{R} \frac{1}{\sqrt{2}}(|0\rangle\,|j\rangle\,|\lambda_j \frac{1}{4M^2}\rangle + |1\rangle\, e^{2\pi i \lambda_j \frac{1}{2^E} 2^e}\,|j\rangle\,|\lambda_j \frac{1}{4M^2}\rangle) \\
&= \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i \lambda_j \frac{1}{2^E} 2^e}\,|1\rangle) \otimes |j\rangle\,|\lambda_j \frac{1}{4M^2}\rangle
\end{aligned}$$

Der Phasenbestimmungsalgorithmus verlangt neben der Ausführung von einer Hamiltoniansimulation U = $e^{\frac{t}{h^2} A_1}$ auch die Ausführung von U^2 bis $U^{2^{E-1}}$.

$$U^{2^e} = S e^{\frac{t}{h^2} \Delta 2^e} S \text{ für } e = 0, \ldots, E-1 \tag{21}$$

Die 2^e fache Hamiltoniansimulation einer Diagonalmatrix kann einfach durch eine 2^e fachen Multiplikation des Skalars k realisiert werden, welches eine 2^e -fache Phasenverschiebung erzeugt. Die Laufzeit von U^2 ist daher dieselbe wie von U^1.

Dieser Algorithmus wurde bei der Simulation auf einem klassischen Rechner nicht benutzt. Bei der Berechnung der Eigenwerte werden pro Schritt 2s Qubits zusätzlich benutzt. Bei $\log_2(r)$ Quadrierungen sind mindestens $\log_2(r) \cdot 2s$ Qubits nötig. Bei r =

8 und s = 6 wird der Arbeitsspeicheraufwand für die Simulation allein für die Berechnung der Eigenwerte um den Faktor 12288 erhöht, welches auf einem Rechner mit 16 Gigabyte Arbeitsspeicher nicht simulierbar ist. Daher wird für die Hamiltoniansimulation der Diagonalmatrix kein Quantenschaltkreis benutzt. Stattdessen wird die Operation „ApplyDiagonalUnitary“ des Quantum Development Kits verwendet:

```
    //Index beschreibt die 2^Index -fache Ausfuehrung der
        Hamiltoniansimulation, qubits: das Register C
operation HamiltonianSimulation(index: Int, qubits: Qubit[],
    c_length:Int,eigenvalues: Double[][], ancilla:Qubit): () {
    body {
            DST(qubits,ancilla);
            ApplyDiagonalUnitary(eigenvalues[index], BigEndian(
                qubits[c_length-1..-1..0]));
            (Adjoint DST)(qubits,ancilla);
    }
}
```

3.4.2 Hamiltoniansimulation von multidimensionalen Poissonmatrizen

Poissonmatrizen mit der Dimension d können durch Zerlegung mit Hilfe von eindimensionalen Poissonmatrizen A_1 realisiert werden. Aus (12) ist bekannt, dass $A_d = A_{d=1} \otimes I \otimes ...I + I \otimes A_{d=1} \otimes I \otimes ...I + ... + I \otimes ...A_{d=1}$. Die Hamiltoniansimulation von $e^{\frac{1}{h^2}tA_d}$ ist:

$$e^{\frac{1}{h^2}tA_d} = e^{\frac{1}{h^2}t(A_{d1}\otimes I\otimes...I+I\otimes A_{d1}\otimes I\otimes...I+...+I\otimes...A_{d1})}$$

$$= e^{\frac{1}{h^2}tA_1} \otimes e^{\frac{1}{h^2}tA_1} \otimes ... \otimes e^{\frac{1}{h^2}tA_1} \tag{22}$$

Die Simulation einer d - dimensionalen Poissonmatrix wird durch die parallele Ausführung von d eindimensionalen Hamiltoniansimulationen durchgeführt. Abbildung 11

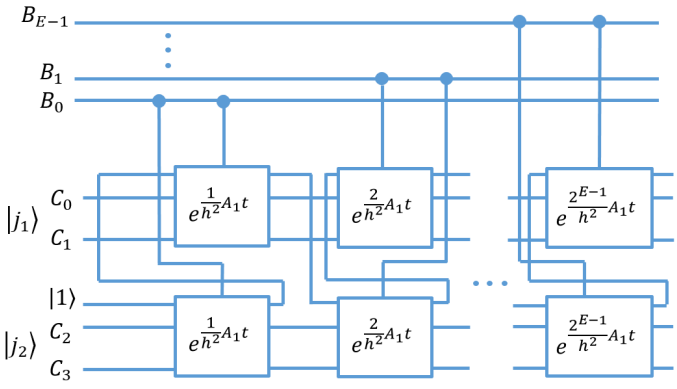


Abbildung 11: Phasenbestimmung einer Hamiltoniansimulation für M=2 und d=2

zeigt den Schaltkreis dieser Simulation. Dabei wird das Eingaberegister C in $\log_2(M)$

Qubit Blöcke unterteilt und jeweils mit $e^{\frac{1}{h^2}A_1}$ simuliert. Das Hilfsqubit, welches zur Auswahl der Sinustransformation genutzt wird, kann für alle Hamiltoniansimulationen wiederverwendet werden, da es stehts im Zustand $|1\rangle$ bleibt. Die Berechnung der Eigenwerte wohingegen müssen pro Hamiltoniansimulation einmal durchgeführt werden. Da die Simulation von $e^{\frac{1}{h^2}A_1}$ mit einer Eingabe arbeitet, die nicht $|0\rangle$ ist, dürfen alle (d viele) $\log_2(M)$ Blöcke sich nicht in einem Nullzustand befinden. Dies erzeugt die Ein- und Ausgabecodierung des Registers C, welche in Kapitel 3.3 beschrieben ist. Seien die Simulationskosten von A_1 $\mathcal{O}(\log_2(M)^2)$, so kostet die Simulation von mehrdimensionalen Problem $\mathcal{O}(d\log_2(M)^2)$. Durch diesen Trick wird e□ektiv der „Fluch der Dimensionen“ gebrochen und eine exponentielle Laufzeitbeschleunigung erzielt.

3.4.3 Register B und Messung der verschränkten Eigenwerten

In diesem Kapitel wird die Auswirkungen des Phasenbestimmungsalgorithmuses auf Register B und C analysiert. Damit PEA den Anfangszustand $|0...0_E\rangle\,|\vec{b}\rangle$ zu $\sum_j b_j\,|\sim\lambda_j\rangle\,|u_j\rangle$ verwandelt, muss Register B groß genug sein, um die Eigenwerte λ_j e□ektiv zu speichern. Die Eigenwerte einer multidimensionalen Poissonmatrix setzten sich aus der Kombination aus Eigenwerten von $\frac{1}{h^2}A_1$ zusammen [2]. So sind die Eigenwerte bei einer zweidimensionalen Poissongleichung:

$$\lambda_{j,k} = 4M^2\sin^2(\frac{j\pi}{2M}) + 4M^2\sin^2(\frac{k\pi}{2M})\ j,k = 1,...,M-1$$

Somit kann die Einschätzung gemacht werden, wie groß die Eigenwerte einer multidemensionalen Poissonmatrix werden:

$$\lambda_{max} = 4dM^2\sin^2(\frac{(M-1)\pi}{2M}) \leq 4dM^2 \tag{23}$$

Damit also Register B die Eigenwerte als Ganzzahl auflösen kann, muss es aus $E = \log_2(4dM^2) = 2 + \log_2(d) + 2\log_2(M)$ Qubits bestehen. Für folgende Messung wird das Beispiel 1 verwendet, welches folgende Eigenwerte und Eigenvektoren hat:

$$\text{Beispiel 1}\ :\ A = \begin{pmatrix} 32 & -16 & 0 \\ -16 & 32 & -16 \\ 0 & -16 & 32 \end{pmatrix}\ \text{mit}\ \vec{b} = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$\lambda_1 = 9{,}37\ \ \lambda_2 = 32\ \ \lambda_3 = 54{,}62\ \ \vec{u_1} = \begin{pmatrix} 0{,}5 \\ \frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix}\ \vec{u_2} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix}\ \vec{u_3} = \begin{pmatrix} 0{,}5 \\ -\frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix}$$

Der Phasenbestimmungsalgorithmus, angewendet auf $\vec{b}$, sollte im Idealfall die Eingabe in eine Superposition von Eigenzuständen und Eigenwerten verwandeln:

$$|0...0_E\rangle\,|\vec{b}\rangle \xrightarrow{\text{PEA}} \frac{1}{2}\,|9{,}37\rangle \begin{pmatrix} 0{,}5 \\ \frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix} + \frac{1}{\sqrt{2}}\,|32\rangle \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} + \frac{1}{2}\,|54{,}62\rangle \begin{pmatrix} 0{,}5 \\ -\frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix} \tag{24}$$

Eine Messung des Registers B sollte beispielsweise mit einer Wahrscheinlichkeit von 50% den Eigenwert 32 ergeben und den Zustand des Register C in eine Superposition bringen, welche dem Eigenzustand u_2 gleicht. Abbildung 12 zeigt die Verteilung der

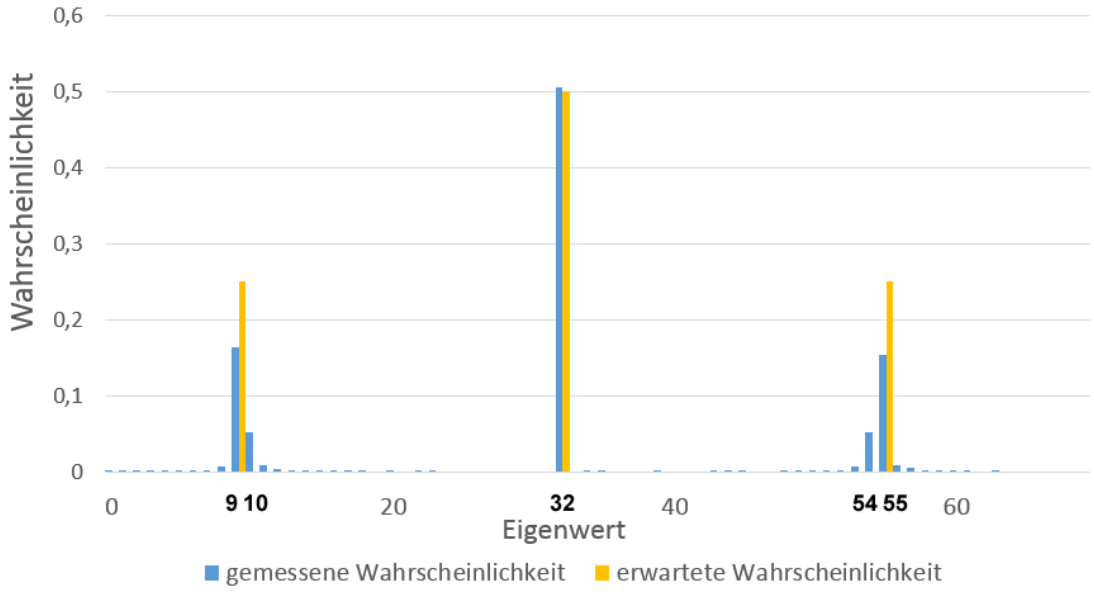


Abbildung 12: Zustandsverteilung des Registers B bei 1000 Messungen

Zustände des Registers B nach 1000 Messungen. Der Phasenbestimmungsalgorithmus erzeugt eine exakte Verteilung um die Eigenwerte, wenn die Eigenwerte selbst von Register B genau aufgelöst werden können. Eigenwert 32 wird exakt mit seiner Amplitude wiedergespiegelt, da er eine Ganzzahl ist. Eigenwert 9,37 und 54,62 können nicht exakt mit einer ganzen Zahl gespeichert werden. PEA erzeugt dabei eine Verteilung um den Eigenwert. In der Messung ist zu sehen, dass die Eigenwerte λ_1 zum größten Teil zu $|9\rangle$ und λ_3 zum Zustand $|55\rangle$ führen. Würde für Register B ein Qubit mehr benutzt werden ($E = E+1$), so können Eigenwerte mit der Auflösung 2^{-1} gespeichert werden. Die Zustandsverteilung um den Eigenwert wäre damit genauer, wodurch letztendlich die Genauigkeit der Lösung verbessert wird.

Nun muss noch gezeigt werden, dass die Eigenwerte mit den Eigenzuständen verschränkt sind. Dies kann validiert werden, indem man Register B misst und die Zustandsverteilung des Registers C analysiert. Eine Superpostion (oder Eigenzustand) kann dabei auch direkt mit Hilfe des Quantum Development Kits ausgelesen werden. Die Funktion „DumpRegister (location : 'T, qubits : Qubit[])“ gibt die Amplituden der Superposition aus, solange die zu messenden Qubits nicht mit anderen Qubits verschränkt sind. Nach der Messung eines Eigenwertes wird Register C zum zugehörigen Eigenzustand gebracht, wobei die Verschränkung zwischen B und C bricht.

Abbildung 13 zeigt drei verschiedene Messungen der Register B und C. Es ist zu

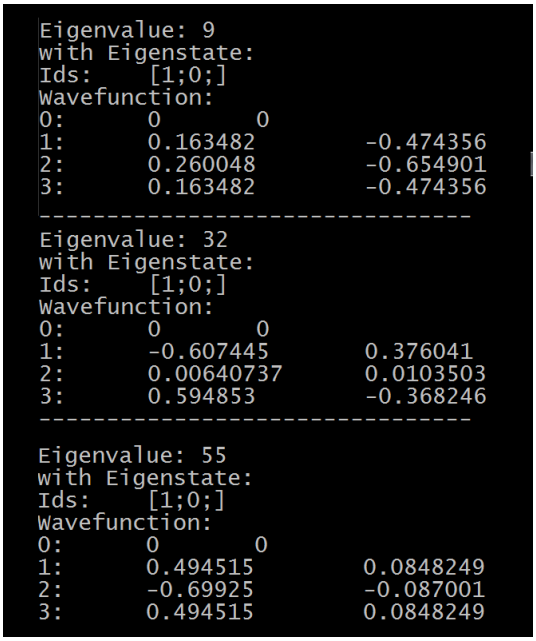


Abbildung 13: Auszug aus DumpRegister(RegC) bei Eigenwerten 32, 55 und 9. Spalte 1 beschreibt den Zustand. Spalte 2 zeigt den reelen Teil und Spalte 3 den imaginären Teil der Amplitude

erkennen, dass die komplexen Amplituden der Zustände genau zu den Eigenvektoren passen, die zu den Eigenwerten gehören. Dabei sind die Eigenzustände um eine globale Phase verschoben. Beispielsweise befindet sich Register C bei Eigenwert 9 in $\sim e^{-2\pi i\, 0{,}19}\vec{u_1}$. Diese Phasenverschiebung entspricht der globalen Phase, welches das Quantum Development Kit zufällig beim Initialisieren erzeugt. Die globale Phase des Systems hat keinen Einfluss auf den Algorithmus und die Lösung des Problems.

3.5 Kehrtwertberechnung der Eigenwerte

Wie Kapitel 3.4 zeigt, transformiert der Phasenbestimmungsalgorithmus die Eingabe $\vec{b}$ in eine Superposition von Eigenzuständen: $\sum_j b_j \vec{u_j}$. Wenn die Amplituden dieser Superposition mit dem Kehrwert der Eigenwerte multipliziert werden, so entspricht das der Lösung $A^{-1}\vec{b}$. In diesem Kapitel wird ein Quantenschaltkreis entwickelt, der den Kehrtwert der Eigenwerte in ein Register speichert. Dabei werden zwei Methoden gezeigt, wie diese mit Hilfe von zwei zusätzlichen Hilfsqubits approximiert werden können. A besteht dabei aus n=$\max(E, 8)$ Qubits.

3.5.1 Methode A: Approximation durch feste Eingabe/Ausgabe

Methode A beschreibt ein hybrides Verfahren. Für λ_j, die kleiner als 16 sind, wird eine feste Ausgabecodierung verwendet. Für größere Eigenwerte wird eine Abschätzung des

Kehrwertes gemacht:

$$\frac{1}{\lambda_j} \approx \frac{1}{2^p} \; p \in \mathbb{R} \text{ mit } |2^p - \lambda_j|_{min} \tag{25}$$

2^p beschreibt dabei einen Exponenten der Basis zwei, welcher dem Eigenwert nahe liegt. Für dieses Verfahren iteriert der Schaltkreis über die Qubits aus Register B und beginnt bei der Wertigkeit 2^{E-1} und geht bis 2^4. Hat die binäre Repräsentation des Eigenwertes an der Stelle zweimal die eins in Folge, so ist $2^p > \lambda_j$. Wenn ein $|10\rangle$ Block gelesen wird, so wird abgerundet. Die Iteration bricht ab, sobald die erste eins gelesen wird. Hilfsqubit$_1$ beschreibt dabei die Abbruchbedingung für die Iteration. Die

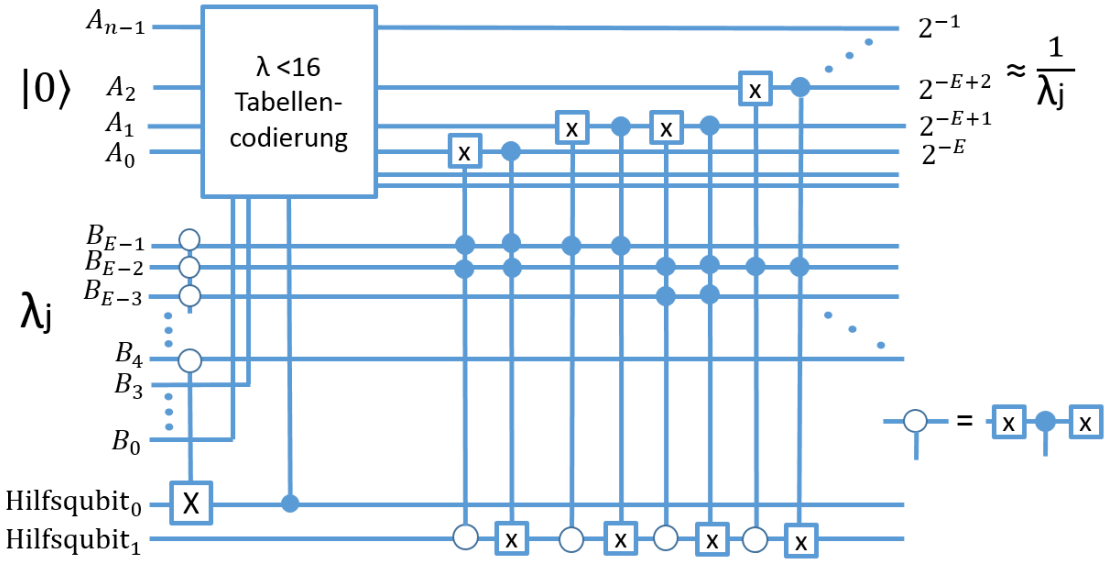


Abbildung 14: Schaltkreis von Methode A zur Kehrwertberechnung

Ausgabecodierung für $\lambda_j < 16$ entspricht dabei Abbildung 15. Zweck dieser direkten Codierung ist die genauere Approximation von größen Kehrwerten. Der absolute Fehler, der durch (25) erzeugt wird, ist bei kleineren Eigenwerten größer, weshalb ein hybrides Verfahren genutzt wird. Die Implementierung der Codierungstabelle wird dabei durch 14 „advancedCNOT“ Gatter, welche in Kapitel 2.5 beschrieben werden, durchgeführt.

$$\text{Sei } |\Psi\rangle = \frac{1}{2}|9\rangle \begin{pmatrix} 0{,}5 \\ \frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix} + \frac{1}{\sqrt{2}}|32\rangle \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} + \frac{1}{2}|55\rangle \begin{pmatrix} 0{,}5 \\ -\frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix}$$

Methode A verschränkt Register A mit Ψ:

$$|\Psi\rangle|0..0_n\rangle \xrightarrow{\text{INV}_A} \frac{1}{2}|9\rangle|\approx \frac{1}{9}\rangle \begin{pmatrix} 0{,}5 \\ \frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix} + \frac{1}{\sqrt{2}}|32\rangle|\frac{1}{32}\rangle \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix} + \frac{1}{2}|55\rangle|\frac{1}{64}\rangle \begin{pmatrix} 0{,}5 \\ -\frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix}$$

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	2^{-6}	2^{-7}	2^{-8}
0	0	1	0	1	0	0	0	0	0	0	0
0	0	1	1	0	1	0	1	0	1	0	1
0	1	0	0	0	1	0	0	0	0	0	0
0	1	0	1	0	0	1	1	0	0	1	1
0	1	1	0	0	0	1	0	1	0	1	0
0	1	1	1	0	0	1	0	0	1	0	0
1	0	0	0	0	0	1	0	0	0	0	0
1	0	0	1	0	0	0	1	1	1	0	0
1	0	1	0	0	0	0	1	1	0	0	1
1	0	1	1	0	0	0	1	1	0	0	1
1	1	0	0	0	0	0	1	0	1	0	1
1	1	0	1	0	0	0	1	0	0	1	1
1	1	1	0	0	0	0	1	0	0	1	0
1	1	1	1	0	0	0	1	0	0	0	1

Abbildung 15: Codierungstabelle für $2 \geq \lambda_j < 16$. Die linke Seite beschreibt die binäre Darstellung von λ_j und die rechte Seite zeigt die Ausgabe des Kehrwertes.

3.5.2 Methode B: Approximation durch Fixpunktiteration

Methode B nutzt das Newton-Raphson Divisionsverfahren zur Berechnung des Kehrwertes. Die Nullstelle für die Funktion $f(x) = 1/x - \lambda_j$ ist $\frac{1}{\lambda_j}$. Die Nullstelle wird mit Hilfe eines Newtonverfahrens berechnet:

$$\begin{aligned} x_{i+1} &= x_i - \frac{f(x)}{f(x)'} \\ &= x_i - \frac{\frac{1}{x} - \lambda_j}{-\frac{1}{x^2}} \end{aligned}$$

$$\rightarrow x_{i+1} = 2x_i - \lambda_j x^2 \tag{26}$$

Für den Startwert x_0 wird eine Approximation des Kehrwertes benutzt, der in (25) definiert ist. Methode B berechnet ähnlich wie Methode A eine Näherung des Kehrwertes $\frac{1}{2^p}$ in Register A. Dabei kann Fall A ($2^p \leq \lambda_j$) oder Fall B ($2^p > \lambda_j$) auftreten. Um zu erkennen, welcher Fall eintritt, wird Hilfsqubit$_1$ verschränkt und auf $|1\rangle$ gesetzt, wenn Fall B genutzt wird. Das Newtonverfahren erzeugt in der ersten Iteration für $x_0 = \frac{1}{2^p}$:

$$x_1 = \frac{2}{2^p} - \lambda_j \frac{1}{2^{2p}} = \frac{2^{p+1}}{2^{2p}} - \lambda_j \frac{1}{2^{2p}} = (2^{p+1} - \lambda_j)\frac{1}{2^{2p}} = (2^{p+1} + B2(\lambda_j))\frac{1}{2^{2p}} \tag{27}$$

B2 beschreibt die Zweierkomplimentdarstellung einer binären Zahl. B2X beschreibt die Zweierkomplimentärdarstellung einer Zahl ohne die Invertierung der vorderen 0

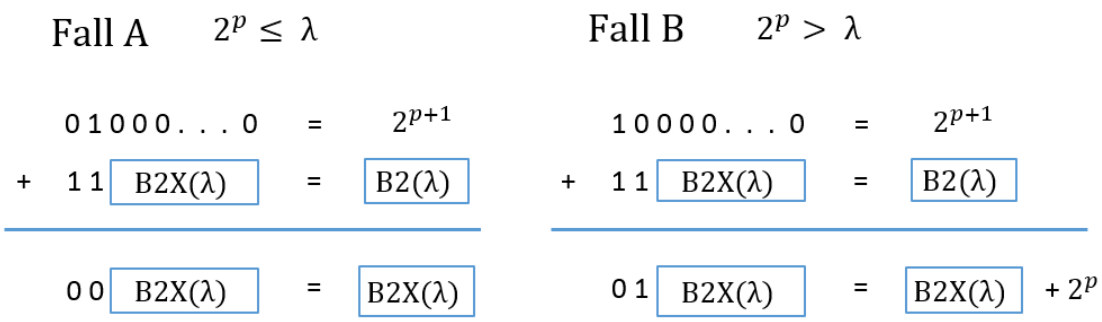


Abbildung 16: Fallunterscheidung für $2^{p+1} + B2(\lambda_j)$

Bits. $2^{p+1}+B2(\lambda_j)$ kann dabei leicht berechnet werden, indem eine Fallunterscheidung gemacht wird:

$$\text{Fall A}: \; 2^{p+1} + B2(\lambda_j) = B2X(\lambda_j) \; \rightarrow \; x_1 = B2X(\lambda_j)\frac{1}{2^{2p}} \tag{28}$$

Sei $\lambda = 9$, so entspricht die Approximation $2^p = 2^3 = 8$. Fall A tritt ein, sodass x_1 wie folgt berechnet wird:

$$x_1 = \frac{2}{2^3} - 9\frac{1}{2^6} = B2X(9)\frac{1}{2^6} = 7\frac{1}{64} = 0{,}109 \approx \frac{1}{9}$$

$$\text{Fall B}: \; 2^{p+1} + B2(\lambda_j) = B2X(\lambda_j) + 2^p \; \rightarrow \; x_1 = B2X(\lambda_j)\frac{1}{2^{2p}} + \frac{1}{2^p} \tag{29}$$

Sei $\lambda = 55$, so entspricht die Approximation $2^p = 2^6 = 64$. Fall B tritt ein und damit entspricht x_1:

$$x_1 = \frac{2}{2^6} - 55\frac{1}{2^{12}} = B2X(55)\frac{1}{2^{12}} + \frac{1}{2^6} = 9\frac{1}{2^{12}} + \frac{1}{2^6} = 0{,}0178 \approx \frac{1}{55}$$

Um x_1 zu konstruieren, muss der Quantenschaltkreis den Wert 2^{-p} und das Zweierkompliment von λ_j speichern. Der Schaltkreis für dieses Verfahren wird in Abbildung 17 gezeigt. Aus Register A kann der Faktor 2^{-2p}, sowie der Summand 2^{-p} ausgelesen werden. Die Ausgabe des Verfahrens beschreibt eine spezielle Form der Quantengleitkommazahl mit Register A als einen Exponenten der Basis 2 und Register B als eine Mantisse. Der Algorithmus tranformiert das Beispiel 1, welches sich in $|\Psi\rangle$ befindet in:

$$|\Psi\rangle\,|0..0_n\rangle \xrightarrow{\text{INV}_B} \underbrace{\frac{1}{2}\,|7\rangle\,|\frac{1}{8}\rangle \begin{pmatrix} 0{,}5 \\ \frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix}}_{\text{Fall A}} + \underbrace{\frac{1}{\sqrt{2}}\,|32\rangle\,|\frac{1}{32}\rangle \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ -\frac{1}{\sqrt{2}} \end{pmatrix}}_{\text{Fall A}} + \underbrace{\frac{1}{2}\,|9\rangle\,|\frac{1}{64}\rangle \begin{pmatrix} 0{,}5 \\ -\frac{\sqrt{2}}{2} \\ 0{,}5 \end{pmatrix}}_{\text{Fall B}}$$

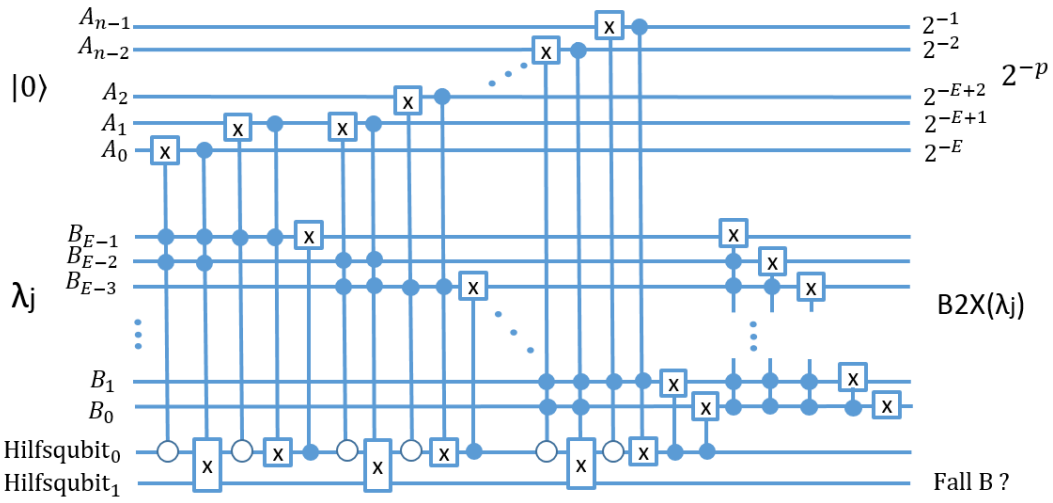


Abbildung 17: Schaltkreis für Methode B zur Kehrwertberechnung

3.5.3 Vergleich der Methoden

Methoden A und B erzeugen unterschiedliche Ausgaben. Während Methode A den Kehrtwert als Fixkommazahl in Register B ausgibt, stellt Methode B den Kehrtwert in Form einer Gleitkommazahl dar. Ist der Eigenwert ein Exponent der Basis 2, so wird der Kehrwert durch die Approximation 2^{-p} von beiden Methoden exakt berechnet. Abbildung 18 zeigt die Kehrwertberechnung beider Methoden.

λ	Methode A	Methode B	λ^{-1}
4	0,25	0,25	0,25
6	0,164	0,125	0,166
9	0,1093	0,1093	0,1111
55	0,0156	0,0178	0,0181
90	0,0156	0,0093	0,0111
180	0,0078	0,0046	0,0055

Abbildung 18: Vergleich der Kehrwertapproximationen von Methode A und B

Für Eigenwerte, die kleiner als 16 sind, approximiert Methode A den Kehrtwert mit Auflösung 2^{-8} und ist damit generell genauer als Methode B. Wenn Poissongleichungen mit mehr Dimensionen oder durch eine höhere Gitterdiskretisierung M gelöst werden müssen, sind auch die Eigenwerte größer. Methode A nutzt dann für $\lambda_j \geq 16$ die grobe Näherung 2^{-p} aus (25). Methode B nutzt die gleiche Approximation als Startwert und führt einen Schritt der Fixpunktiteration zusätzlich aus. Dank der quadratischen Konvergenz des Newton-Verfahrens verbessert sich die Genauigkeit des Kehrwertes um das Doppelte. Methode B berechnet daher für $\lambda_j \geq 16$ eine bessere Näherung als Methode A, welches sich auch in der Lösung der Poissongleichung widerspiegelt.

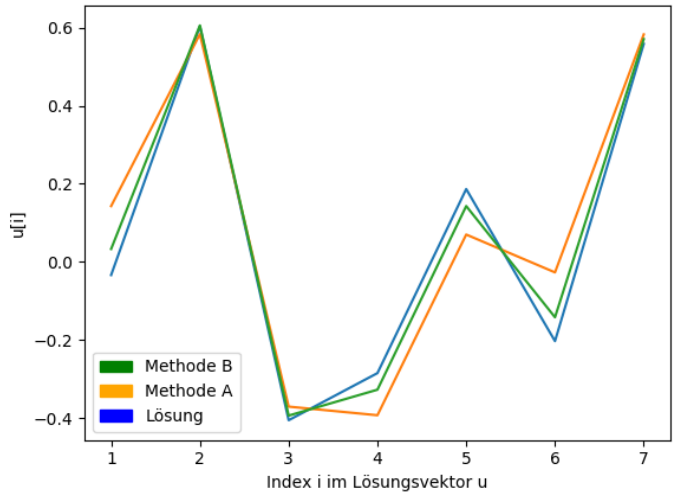


Abbildung 19: Ergebnisse des Algorithmuses mit Methode A und B für $M = 8,\ d = 1,\ \alpha = 30$ und $\vec{b} = (-0{,}23,\ 0{,}57,\ -0{,}39,\ -0{,}12,\ 0{,}29,\ -0{,}39,\ 0{,}45)^T$

3.6 Die kontrollierte Rotation und der Amplitudenfaktor

Dieses Kapitel beschreibt das Verfahren, wie der α-fache Kehrwert $\frac{1}{\lambda_j}$ in die $|1\rangle$ Amplitude eines Hilfsqubits H_r rotiert werden kann ($\alpha \in \mathbb{R}$). Ähnlich wie in dem Schaltkreis von Abbildung 10 geschieht dies durch kontrollierte Rotationen. Hierbei soll nicht die Phase, sondern die Amplitude beeinflusst werden, sodass mit R_y über die y-Achse der Blochkugel rotiert werden muss. Da Methode A und Methode B zwei unterschiedliche Ausgabencodierungen für den Kehrwert haben, muss die kontrollierte Rotation zur Methode angepasst werden. Während die kontrollierte Rotation von Methode A nur Register A benötigt, muss für Methode B Register A, B und ein weiteres Hilfsqubit verwendet werden. Der Quantenschaltkreis für die Rotation mit einer Fixkommazahl wird in Abbildung 20 gezeigt. Sei $k_1 ... k_n$ die binäre Repräsentation des Kehrwer-

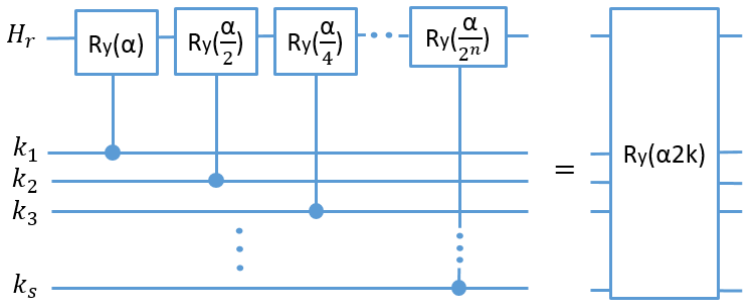


Abbildung 20: Kontrollierte Rotation des Qubits H_r für Methode A.

tes $k \approx \lambda_j^{-1}$ mit Wertigkeitsreihenfolge 2^{-1} bis 2^{-n}. Das Verfahren erzeugt folgende

Rotation auf das Qubit H_r:

$$k_1 R_y(\alpha) \cdot k_2 R_y(\frac{\alpha}{2}) \cdot k_3 R_y(\frac{\alpha}{4}) \cdot ... \cdot k_n R_y(\frac{\alpha}{2^n}) \; k_i \in (0,1) \; \alpha \in \mathbb{R}$$

$$= R_y(k_1\alpha + k_2\frac{\alpha}{2} + k_3\frac{\alpha}{4} + ... + k_n\frac{\alpha}{2^n}) = R_y(\alpha 2k) = \begin{pmatrix} \cos\frac{\alpha 2k}{2} & -\sin\frac{\alpha 2k}{2} \\ \sin\frac{\alpha 2k}{2} & \cos\frac{\alpha 2k}{2} \end{pmatrix}$$

Diese Rotation angewendet auf ein Qubit, welches mit $|0\rangle$ initialisiert ist, ergibt:

$$|k\rangle |0\rangle \xrightarrow{\text{Ry}} |k\rangle (\sqrt{1-\sin^2(\alpha k)}\,|0\rangle + \sin(\alpha k)\,|1\rangle) \tag{30}$$

Um nun e□ektiv die Amplitude von $|1\rangle$ auf den α-fachen Kehrwert zu setzen, muss um den Winkel $\arcsin(\alpha k)$ rotiert werden. Es müsste also vor der Anwendung der kontrollierten Rotation ein $\arcsin$ Modul geben, welches auf Register A angewendet wird. In dieser Arbeit wird auf dieses Modul verzichtet und die Kleinwinkelnäherung genutzt: $\alpha k \approx \sin(\alpha k)$ $\alpha k \in [0; 0.5]$. Für Methode B muss eine Quantengleitkommazahl der Form $B2X(\lambda_j)2^{-2p}$ in die Amplitude rotiert werden. Dies kann erreicht werden, indem eine kontrollierte Rotation der Mantisse $k = B2X(\lambda_j)$ durchgeführt wird. Der binäre Exponent kann dabei wie α direkt in den Rotationsgattern als Faktor implementiert werden. Da 2^{-2p} nur zur Laufzeit bekannt ist und sich je nach Eigenwert unterscheidet, müssen E (= Anzahl der Qubits von Register A) unterschiedliche Exponenten implementiert werden. Dies geschieht durch E Verschränkungen mit Register A. Wie in Kapitel 3.5.2 beschrieben, kann Fall B auftreten, sodass zusätzlich noch um den Wert 2^{-p} rotiert werden muss. Dies wird ebenfalls durch E verschränkte Rotationen durchgeführt, falls Hilfqubit$_1$ (definiert in Abbildung 17) den Zustand $|1\rangle$ hat. Abbildung 21 zeigt die Implementierung dieses Verfahrens. Wird die Rotation

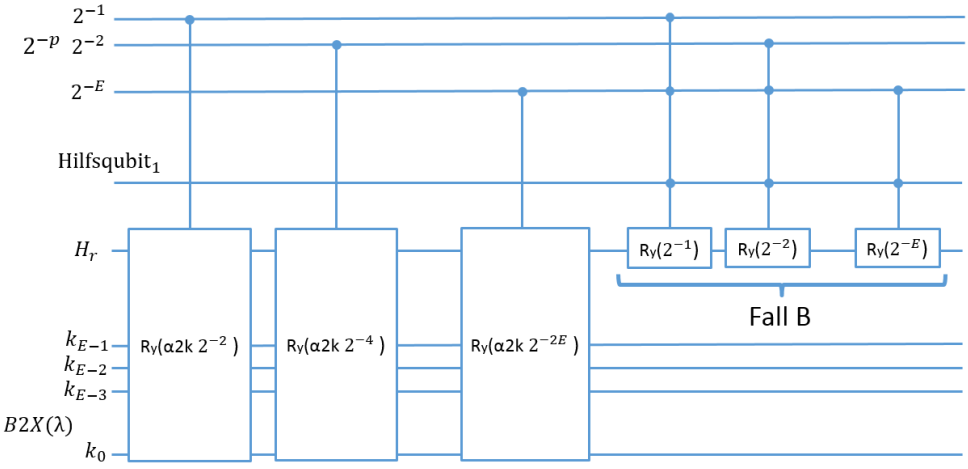


Abbildung 21: Kontrollierte Rotation des Qubits H_r für Methode B.

mit Methode A oder B korrekt ausgeführt, so befindet sich das Quantensystem bei idealer Kehrwertberechnung in:

$$\sum_j b_j |\vec{u_j}\rangle |\lambda_j\rangle |\lambda_j^{-1}\rangle |0\rangle \xrightarrow{R_y} \sum_j b_j |\vec{u_j}\rangle |\lambda_j\rangle |\lambda_j^{-1}\rangle (\sqrt{1-\sin^2(\frac{\alpha}{\lambda_j})}\,|0\rangle + \sin(\frac{\alpha}{\lambda_j})\,|1\rangle)$$

$$\approx \sum_j b_j \left|\vec{u_j}\right\rangle \left|\lambda_j\right\rangle \left|\lambda_j^{-1}\right\rangle \left(\sqrt{1-(\frac{\alpha}{\lambda_j})^2}\left|0\right\rangle + \frac{\alpha}{\lambda_j}\left|1\right\rangle\right) \text{ für } \alpha \leq \frac{\lambda_1}{2} \tag{31}$$

Da der Algorithmus nur dann Erfolg hat, wenn die Messung von $H_r \left|1\right\rangle$ ergibt, ist es wichtig zu wissen, wie hoch die Erfolgswahrscheinlichkeit ist. Da H_r mit allen anderen Qubits verschränkt ist, kann das Qubit nicht direkt mit Hilfe von „DumpRegister(H_r)“ gemessen werden, da das System sich in einem nicht separierbaren Zustand befindet. Es kann jedoch das System aller Qubits mit „DumpMachine()“ ausgegeben werden. Um die Erfolgswahrscheinlicht zu messen, werden die Wahrscheinlichkeiten aller Zustände addiert, bei denen H_r den Zustand $\left|1\right\rangle$ hat. Die Amplitude von $H_r = \left|1\right\rangle$ ist:

$$\sqrt{\sum_j b_j^2 \sin^2(\frac{\alpha}{\lambda_J})}$$

und damit ist die Erfolgswahrscheinlichkeit Ω des Algorithmuses:

$$\Omega = \sum_j b_j^2 \sin^2(\frac{\alpha}{\lambda_J}) \tag{32}$$

Für Beispiel 1 und für einen Amplitudenfaktor $\alpha = 1$ ergibt der Algorithmus eine Erfolgswahrscheinlichkeit von: $\frac{1}{4}\sin^2(\frac{1}{9{,}37}) + \frac{1}{2}\sin^2(\frac{1}{32}) + \frac{1}{4}\sin^2(\frac{1}{54{,}67}) = 0{,}34\,\%$. Man müsste daher durchschnittlich den Algorithmus 300 mal durchführen, um ein korrektes Ergebnis zu erhalten. Um dies zu verbessern, kann der Amplitudenfaktor α erhöht werden. Bei $\alpha = 2$ wird die Ampltiude (unter der Annahme der Kleinwinkelnäherung) verdoppelt und damit die Erfolgswahrscheinlichkeit vervierfacht. Die Implementierung des Faktors α wird direkt in den Rotationsgatter R_y realisiert. Bei einer Verdopplung des Amplitudenfaktors wird um den doppelten Winkel gedreht. Abbildung 22 zeigt die Auswirkungen des Amplitudenfaktors.

Es ist zu erkennen, dass der quadratische Fehler für Beispiel 1 stark ab $\alpha = 8$ ansteigt. Grund dafür ist die Approximation durch die Kleinwinkelnäherung. Die rotierte Amplitude für Eigenwert 9,37 entspricht $\sin(\frac{8}{9.37}) \neq \frac{8}{9.37}$. Für die verschränkten Zustände mit größeren Eigenwerten ist der Kehrwert $\frac{\alpha}{\lambda_j}$ kleiner, wodurch die Kleinwinkelnäherung nicht verletzt wird. Das lineare Verhältnis b_j zwischen den verschränkten Zuständen wird beeinflusst, sodass der Fehler der Ausgabe stark ansteigt. Der Amplitudenfaktor darf also nicht zu groß gewählt werden, damit keine fehlerhaften Ergebnisse ausgegeben werden. Das Produkt $\alpha\frac{1}{\lambda_j}$ darf daher nicht größer als $0{,}5$ sein. Dieses Produkt wird bei dem kleinsten Eigenwert am größten, sodass für den Amplitudenfaktor gilt: $\alpha \leq \frac{\lambda_1}{2}$. Dies ist keine Seltenheit bei Quantenalgorithmen. Oft muss eine gewisse Vorkenntnis gegeben sein, um Quantenalgorithmen e□ektiv durchzuführen. Der kleinste Eigenwert kann dabei so abgeschätzt werden:

$$\lambda_1 = 4dM^2 \sin^2(\frac{\pi}{2M}) \tag{33}$$

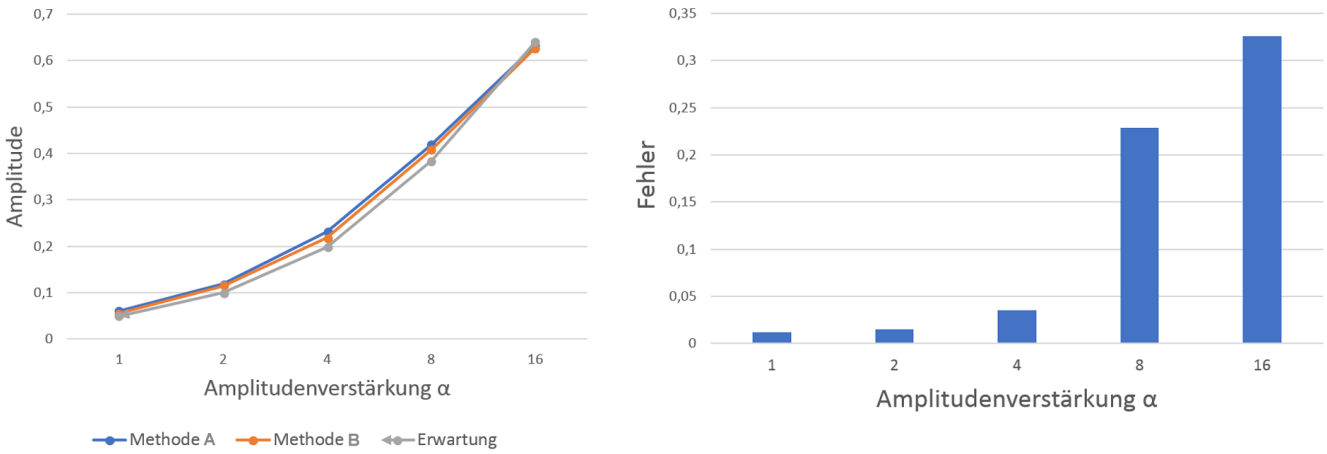


Abbildung 22: Links zeigt für Beispiel 1 die Auswirkungen von α auf die Amplitude von H_r und rechts auf den quadratischen Fehler zur Lösung (bei Verwendung von Methode B)

Um für Beispiel 1 relativ genaue Ergebnisse zu erhalten, sollte ein Amplitudenfaktor von 4 nicht überschritten werden. Im Durchschnitt muss dadurch der Algorithmus etwa 25 mal durchgeführt werden. Wenn jedoch eine höhere Erfolgswahrscheinlichkeit erzielt werden soll, so kann der Amplitudenfaktor schrittweise erhöht werden. Oft ist der Faktor b_1 der Linearkombination für $\vec{b}$ null beziehunsgweise $\vec{u_1}$ ist nicht Teil der Linearkombination $\sum_j b_j \vec{u_j}$. Die Grenze des Amplitudenfaktors wäre damit nicht mehr abhängig von λ_1, sodass ein Amplitudenfaktor $\alpha \leq \frac{\lambda_2}{2}$ gewählt werden kann. Amplitudenfaktor, Erfolgswahrscheinlichkeit und Fehler hängen daher von der Eingabe $\vec{b}$, Diskretisierung und Dimension ab. Da die Linearkombination prinzipiell nicht bekannt ist, wäre ein gutes Vorgehen, den Algorithmus mit kleinen Amplitudenfaktoren durchzuführen und α schrittweise anzuheben, solange ein solides Ergebnis erzielt wird.

3.7 Das Modul UNCOMPUTE

Dieses Modul hat die Funktion, die Verschränkung zwischen Eigenzustand, Eigenwert und dem Kehrwert aufzuheben. Ohne die Ausführung des Moduls würde eine Messung des Qubits H_r mit der Erfolgswahrscheinlichkeit aus (32) zufällig zum Zustand $|\vec{u_j}\rangle\,|\lambda_j\rangle\,|\frac{1}{\lambda_j}\rangle$ führen. Um die normierte Lösung jedoch zu erhalten, müssen Register A und B freigegeben werden. Dies kann durch die inverse Ausführung von INV und PEA erreicht werden. Dabei werden die für die Module verwendeten adjungierten Operationen in umgekehrter Reihenfolge ausgeführt. Kennt man den Schaltplan zur Ausführung unitärer Transformationen, so kann auch die inverse Operation dazu implementiert werden. Das Quantum Development Kit automatisiert das Verfahren. Wird bei der Definition einer Operation der Schlüssel „adjoint auto“ verwendet, so kann über den Aufruf „(Adjoint U)(Parameter)“ die adjungierte Transformation

durchgeführt werden. Das Modul INV wird in Q# daher so realisiert:

```
(Adjoint INV)(RegB,RegA,Ancillas[1..2],methodA);
(Adjoint PEA)(RegB,RegC,Ancillas,dimension,size,eigenvalues);
```

Das Quantensystem befindet sich nach der Ausführung des Moduls in:

$$|\Psi\rangle \xrightarrow{INV^\dagger} \sum_j b_j |\vec{u_j}\rangle |\lambda_j\rangle |0\rangle \left(\sqrt{1-\sin^2(\frac{\alpha}{\lambda_j})}|0\rangle + \sin(\frac{\alpha}{\lambda_j})|1\rangle\right)$$
$$\xrightarrow{PEA^\dagger} \sum_j b_j |\vec{u_j}\rangle |0\rangle |0\rangle \left(\sqrt{1-\sin^2(\frac{\alpha}{\lambda_j})}|0\rangle + \sin(\frac{\alpha}{\lambda_j})|1\rangle\right)$$

Die Eigenzustände sind nachher nur noch mit dem rotierten Qubit H_r verschränkt. Die Messung von Qubit H_r bricht die letzte Verschränkung zwischen H_r und $\vec{u_j}$. Wenn die Messung zum Zustand $|1\rangle$ führt, hat der Algorithmus erfolgreich die Kehrwerte in die Amplituden der Eigenzustände gebracht: $C\sum_j b_j \sin(\frac{\alpha}{\lambda_j})|\vec{u_j}\rangle \otimes |1\rangle \approx C\sum_j b_j \frac{\alpha}{\lambda_j}|\vec{u_j}\rangle \otimes |1\rangle = C\mathsf{A}^{-1}\vec{b} \otimes |1\rangle = \frac{\vec{u}}{||\vec{u}||} \otimes |1\rangle$. $C = (\sqrt{\sum_j (\frac{b_j \alpha}{\lambda_j})^2})^{-1}$ beschreibt eine Normalisierungskonstante und sorgt für die Normierung des Zustandes. Der Zustand in Register C beschreibt eine Superposition, welche der normierten Lösung der Poissongleichung gleicht.

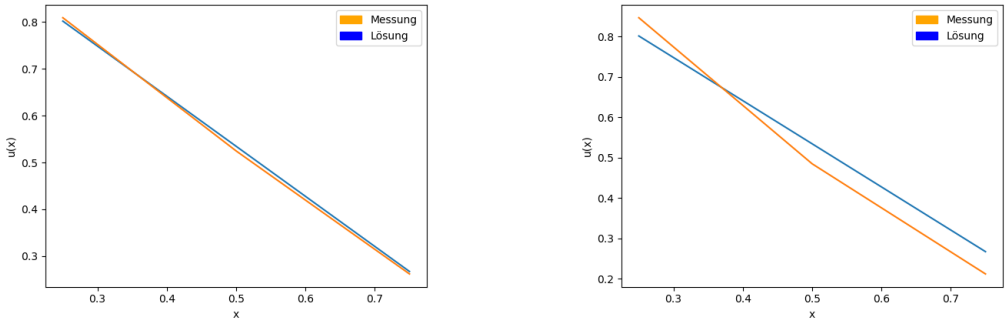
3.8 Ergebnisse und Bewertung des Algorithmuses

In diesem Kapitel wird die Laufzeit des Algorithmuses analysiert und erfolgreiche Ausgaben gezeigt. Der Quantenschaltkreis nutzt pro Dimension $\log_2(M)+2s\log_2(r)$ $s,r \in \mathbb{R}$ Qubits. Für PEA und die Kehrwertberechnung werden $2E$ Qubits benötigt. Nebenbei werden 4 Hilfsqubits benutzt, wodurch der Algorithmus ingesamt $(d(\log_2(M)+2s\log_2(r))+2\log_2(d)+4\log_2(M)+8) \in \mathcal{O}(d\log_2(M))$ Qubits in Anspruch nimmt. Neben der Anzahl der Qubits, ist die Anzahl der Quantengatter beziehungsweise die Gatterlaufzeit ein wichtiger Aspekt. Die Simulation einer eindimensionalen Poissonmatrix geschieht in $\mathcal{O}(\log_2(M)^2)$, da einerseits die Sinustransformation und andererseits bei der Berechnung der Eigenwerte arithmetische Operationen eine quadratische Laufzeit haben. Die Simulation von multidimensionalen Problemen geschieht durch die Simulation von d - eindimensionalen Hamiltoniansimulationen. Durch PEA werden die Simulationen jeweils E (E $\in \mathcal{O}(\log_2 M+\log_2 d)$) mal ausgeführt, wodurch der Phasenbestimmungsalgorithmus eine Laufzeit von $\mathcal{O}(d\log_2(M)^3+d\log_2(d)\log_2(M)^2)$ hat. Die Kehrwertmodule für Methode A und B werden in linearer Laufzeit zu E durchgeführt. Die kontrollierte Rotation für Methode A nutzt E kontrollierte Rotationsgatter, wohingegen Methode B die kontrollierte Rotation der Mantisse zusätzlich E mal öfter anwenden muss als Methode A. Die Laufzeit einer Iteration des Schaltkreises ist für beide Methoden daher durch PEA der Poissonmartrix $\mathcal{O}(d\log_2(M)^3+d\log_2(d)\log_2(M)^2)$ beschrieben, da er der aufwendigste Teil des Algorithmuses ist. Wie in Kapitel 3.6.3 erläutert, hat der Algorithmus eine gewisse Erfolgswahrscheinlichkeit Ω. Daher ist es wichtig, zu wissen, wie hoch diese mindestens ist und wie viele Iterationen maximal

durchgeführt werden müssten, um im Durchschnitt ein Ergebnis zu erhalten. Aus Kapitel 3.6.3 ist bekannt, dass der Amplitudenfaktor α nicht größer als $\frac{\lambda_1}{2}$ sein darf, um große Fehler zu vermeiden. Dabei wäre die Erfolgswahrscheinlichkeit laut (32) am kleinsten, wenn das lineare Verhältnis b_j größten Teils auf Zustände mit größeren Eigenwerten fällt. Die minimale Erfolgswahrscheinlichkeit ist daher:

$$\Omega_{min} = \sin(\frac{\lambda_1}{\lambda_{(M-1)^d}})^2 \approx \frac{\lambda_1^2}{\lambda_{(M-1)^d}^2} = \frac{1}{\kappa_2^2} \tag{34}$$

Der Algorithmus muss also im schlimmste Fall κ_2^2 mal iteriert werden, um ein Ergebnis zu erhalten. Dabei ist zu erwähnen, dass die Konditionszahl κ_2 nicht abhängig von der Dimension ist. Ist M kleiner, so liegen die Eigenwerte näher aneinander, wodurch die Kondition kleiner wird. Der Quantenalgorithmus ist bezüglich der Kondition schlechter als klassiche Verfahren wie das CG-Verfahren, die eine Laufzeit linear zu κ_2 haben. Dafür kann der Algorithmus bezüglich Dimension und Gitterdiskretisierung die Lösung in Form einer Superposition mit exponentieller Laufzeitbeschleunigung berechnen. Die Dimension wird zum linearen Faktor der Laufzeit, wodurch der Fluch der Dimensionen e□ektiv gebrochen wird. Im Folgenden werden Ergebnisse des Quantensimulators mit unterschiedlichen Amplitudenfaktoren gezeigt. Dabei wird bei erfolgreicher Messung das Register C mit „DumpRegister(RegC)“ gemessen. Oft gibt DumpRegister bei Erfolg kein Ergebnis aus, da die Qubits aus Register C noch minimal mit anderen Qubits verschränkt sind. Grund dafür könnte eine nicht exakte Umkehrung der Operation im Modul UNCOMPUTE sein, wodurch Verschränkungen nicht ganz aufgehoben werden. Abbildung 23 zeigt das Ergebnis für Beispiel 1 bei einem starken und schwachen Amplitudenfaktor. Während für $\alpha = 3$ die Erfolgs-



(a) $\alpha = 3$ $\Omega \approx 3\%$ (b) $\alpha = 10$ $\Omega \approx 25\%$

Abbildung 23: Messung des Register C bei Erfolg für Beispiel 1. Dabei wurde Methode B zur Kehrwertberechnung genutzt.

wahrscheinlichkeit bei 3% liegt, liegt sie bei $\alpha = 10$ schon bei 25%, wodurch der Algorithmus im Durchschnitt nur 4 mal ausgeführt werden muss. Nachteilhaft ist dabei die Verletzung der Kleinwinkelnähreung verletzt, sowie die Überrotation des Qubits H_r, sodass bei großen Amplitudenfaktoren größere Fehler zu erkennen sind. Die drei

zu lösenden Unbekannten für Beispiel 1 sind sehr einfach zu berechnen. Wenn hingegen eine größe Diskretiserung M verwendet wird, werden zum einem mehr Qubits benötigt und zum anderen wird die Kondition des Problems schlechter. Mehr Qubits bedeuten, dass mehr Arbeitsspeicher gebraucht wird, um das System zu simulieren. Abbildung 24 zeigt die Simulationsergebnisse für M=16 und d=1. Der Quantensimu-

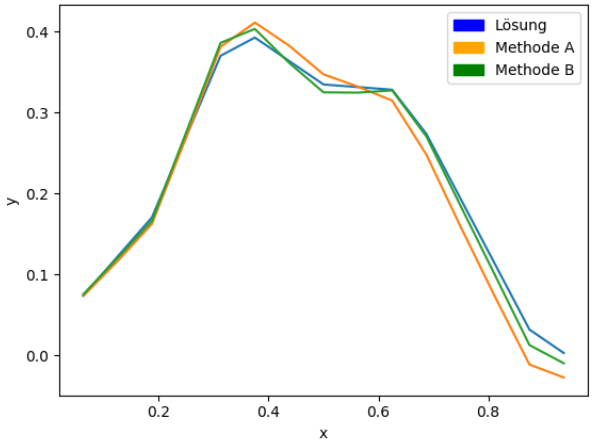
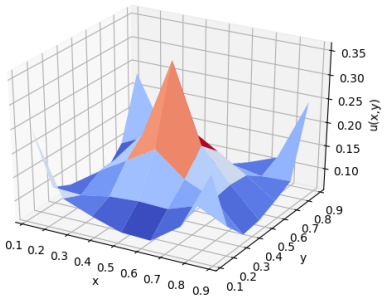
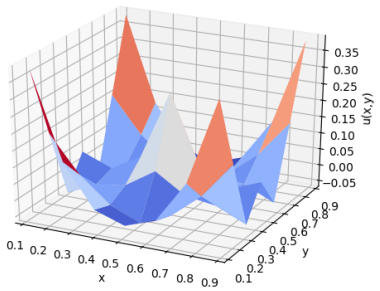


Abbildung 24: Ergebnis für das Problem: M=16, d=1, $\vec{b} = (0{,}18,\ 0,\ -0{,}37,\ 0,\ 0{,}55,\ 0{,}37,\ 0,\ -0{,}18,\ 0,\ 0{,}37\ ,\ 0{,}18,\ 0,\ 0,\ -0{,}37,\ -0{,}18)^T$, $\alpha = 10,\ \Omega \approx 5\%$

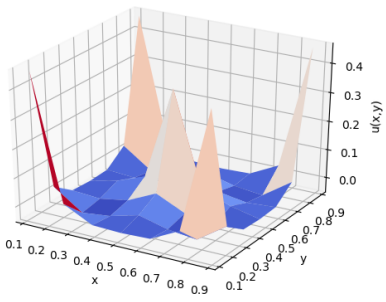
lator nutzte dabei 29 Qubits. Wegen dem exponentiellen Wachstum des Hilbertraums bezüglich der Anzahl an Qubits wurde für dieses Problem etwa 13 Gigabyte Arbeitsspeicher benötigt. Je größer M desto größer die Kondition und desto kleiner die Erfolgswahrscheinlichkeit des Verfahrens. Selbst bei einem Amplitudenfaktor von 8, musste etwa 20 mal iteriert werden, bis ein solides Ergebnis erzielt worden ist. Neben Problemen mit höhere Gitterdiskretisierung können auch mehrdimensionale Probleme gelöst werden. Durch die parallele Ausführung der Hamiltoniansimulation muss bei einem mehrdimensionalen Problem die Eingabe- und Ausgabecodierung aus Kapitel 3.1 beachtet werden. Für M=8 und d=2 wird beispielsweise β_1 durch die Amplitude des Zustandes $|9\rangle$ repräsentiert. Abbildung 25 zeigt das Ergebnis für eine Poissongleichung mit kleinen Potentialen (zB. Wärmeerzeuger) in den Ecken und einem großen Potential in der Mitte des zweidimensionalen Einheitsraumes. Die Simulation erforderte ebenso 29 Qubits und braucht etwa 15 Minuten pro Iteration. Während mit $\alpha = 8$ der Algorithmus nach fünf Stunden ein Ergebnis erzielte, hat es für $\alpha = 300$ nur zwei Iterationen gebraucht. Für eine grobe Approximation der Lösung reicht es also große Amplitudenfaktoren zu nehmen. Für genauere Lösungen muss abhängig von der Kondition mehr iteriert werden. Die Ergebnisse zeigen, dass der Algorithmus grobe Approximationen, sowie genauere Ergebnisse der normierten Lösung ausgeben kann. Es stellt sich die Frage, ob Poissongleichungen auf einen echtem Quantencomputer gelöst werden sollten. Ist man an den diskreten Werten der Lösung interresiert, so kann diese Frage klar mit nein beantwortet werden. Der Algorithmus erzeugt die



(a) normierte Lösung



(b) Messung mit $\alpha = 30\ \Omega \approx 3\%$



(c) Messung mit $\alpha = 300\ \Omega \approx 71\%$

Abbildung 25: Messung des Register C bei Erfolg für M=8, d=2, b=$(0{,}46_1,\ 0_2,\ \ldots,\ 0{,}46_7,\ 0_8,\ \ldots\ 0{,}37_{24},\ 0_{25},\ \ldots\ 0{,}46_{43},\ 0_{44},\ \ldots,\ 0{,}46_{49})^T$. Für die Kehrwertberechnung wurde Methode B genutzt.

normierte Lösung in Form einer Superposition in Register C. Superpositionen können dabei nicht direkt gemessen werden. Man müsste bei einer Lösung aus N Unbekannten, den Algorithmus mehr als $\mathcal{O}(N)$ mal erfolgreich ausführen, um die Amplituden einer Superposition auszulesen. Auch wenn die Lösung der Größe N perfekt konstruiert werden kann, so hat das Auslesen der Unbekannten $\mathcal{O}(N)$. Einen Algorithmus zu entwickeln, der logarithmisch zu N läuft, ergibt also keinen Nutzen, solange das „Read Out Problem“ besteht. Ist man jedoch nicht direkt an den Werten der Lösung, sondern an das Ergebnis eines Operators $\langle\vec{u}|\mathrm{U}|\vec{u}\rangle$ interressiert, so kann es sich lohnen. Beispielsweise kann sehr einfach ein Messoperator auf Register C angewendet werden. Da die diskreten Funktionswerte der Lösung den Amplituden enstsprechen, wird bei einer Messung des Registers mit hoher Wahrscheinlichkeit eine Maximalstelle der Poissonlösung herauskommen. Ist man an einem zufälligen Fourierkoe□zenten der Lösung interressiert, so kann die Quantenfouriertransformation auf $\vec{u}$ angewendet werden. Quantencomputer erzeugen genau dann gegenüber klassische Rechner einen Vorteil, falls man an eine bestimmte Korrelation zwischen diskreten Funkionswerten interressiert ist. Daher kann sich die Berechnung auf einem echten Quantencomputer lohnen, falls man zum Beispiel Maxima/Minima oder Periodizitäten der Lösung berechnen will.

4 Weitere Forschungsmöglichkeiten

Dieses Arbeit kann in vielen Bereichen noch weiterentwickelt werden. Beispielsweise könnte ein arcsin-Modul implementiert werden, um das Problem der Kleinwinkelnäherung zu umgehen. Damit kann der Amplitudenfaktor auf λ_1 gesetzt werden, wodurch zum einem eine Vervierfachung der Mindesterfolgswahrscheinlichkeit Ω_{min} und zum anderen kleinere Fehler erreicht werden. Auch bietet die Realisierung auf einem ech-

ten Quantencomputer neue Herausforderungen. Momentan (Stand März 2019) arbeit IBM an einem 50 Qubit Quantencomputer. 50 Qubits könnten aussreichen, um kleine Poissongleichungen zu lösen. Dabei könnte man versuchen, Probleme zu konstruieren, deren Hamiltoniansimulation durch einfache Anwendung von Pauligattern realisiert werden kann. Das schwierigste und größte Problem bei einem echten Quantencomputer ist die Dekohärenz. Unter Dekohärenz versteht man den zufälligen Verlust der Superposition. Dies kann durch thermische Einflüsse von außen enstehen, wodurch Quantencomputer oft Fehlerraten von bis zu 70% haben. Die Dekohärenzzeit, der Zeitpunkt ab dem die Qubits dekohärieren, bestimmt eine maximale Gattertiefe. Der Algorithmus muss daher bezüglich Gatterlaufzeit optimiert werden. Neben der Dekohärenz können bei Qubits zufällige Phasen- und Bitflips passieren. Es können Quantenfehlerkorrekturcodes wie Shor-Code verwendet werden, um diese Fehler zu erkennen und zu korrigieren[6]. Oft müssen dabei mehrere Qubits zueinander verschränkt werden, um ein Qubit stabil zu halten. Neben IBM forschen viele andere Firmen an der Realisierung eines stabilen Quantencomputers. Quantencomputer selbst können dabei auf unterschiedliche Weise gebaut werden. Ob sich das Mooresche Gesetz dabei auch für Quantencomputer durchsetzen wird, wird sich in vielleicht naher Zukunft zeigen.

Literatur

[1] G. Benenti, G. Casati, D. Rossini, and G. Strini. *Principles of Quantum Computation and Information*. World Scientific, 2019.

[2] Y. Cao, A. Papageorgiou, I. Petras, J. Traub, and S. Kais. Quantum algorithm and circuit design solving the poisson equation. 2012.

[3] A. W. Harrow, A. Hassidim, and S. Lloyd. Quantum algorithm for solving linear systems of equations. 2008.

[4] A. Klappenecker and M. Roetteler. Discrete cosine transforms on quantum computers. 2001.

[5] D. Kulkarni, D. Schmidt, and S.-K. Tsui. Eigenvalues of tridiagonal pseudo-toeplitz matrices. 1997.

[6] M. Nakahara and T. Ohmi. *Quantum Computing - From Linear Algebra to Physical Realizations*. Taylor & Francis Group, 2008.

[7] A. Papageorgiou and C. Zhang. On the e□ciency of quantum algorithms for hamiltonian simulation. 2010.

[8] A. N. Soklakov and R. Schack. E□cient state preparation for a register of quantum bits. 2004.

[9] V. Vedral, A. Barenco, and A. Ekert. Quantum networks for elementary arithmetic operations. 1995.