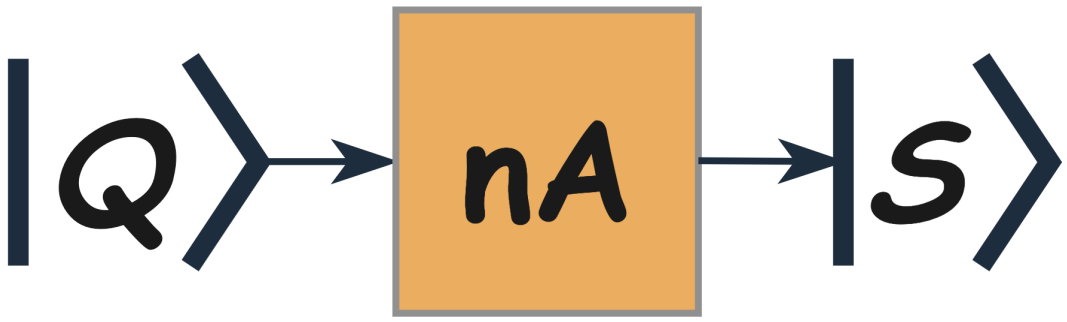


QnAS - Quantum noisy Algorithm Simulator

A tool for simulating noisy quantum algorithms in Python

Ett simuleringsverktyg för kvantalgoritmer med störningar i Python

Kandidatarbete i Teknisk fysik

Axel Blom, Albin Edenmyr, Edvin Martinson, Ludvig Nordqvist, Didrik Palmqvist & Isak Wikman

Institutionen för mikroteknologi och nanovetenskap

Chalmers tekniska högskola
Göteborg, Sverige 2022
www.chalmers.se

Kandidatarbete 2022

QnAS - Quantum noisy Algorithm Simulator

A tool for simulating noisy quantum algorithms in Python
Ett simuleringsverktyg för kvantalgoritmer med störningar i Python

Axel Blom
Albin Edenmyr
Edvin Martinson
Ludvig Nordqvist
Didrik Palmqvist
Isak Wikman

CHALMERS

Institutionen för mikroteknologi och nanovetenskap
WACQT
Chalmers tekniska högskola
Göteborg, Sverige 2022

QnAS - Quantum noisy Algorithm Simulator
Ett simuleringsverktyg för kvantalgoritmer med störningar i Python

Handledare: Anton Frisk Kockum & Kamanasish Debnath, institutionen för mikroteknologi och nanovetenskap
Examinator: Per Lundgren, institutionen för mikroteknologi och nanovetenskap

Kandidatarbete 2022
Institutionen för mikroteknologi och nanovetenskap
WACQT
Chalmers tekniska högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Omslag: Logotyp för QnAS.

Typsatt i LaTeX
Göteborg, Sverige 2022

Sammandrag

Kvantdatorer spås i framtiden kraftigt kunna överträffa klassiska datorer för vissa beräkningar och är därför ett av vår tids mest omtalade forskningsområden. I dagsläget begränsas kvantdatorer dock kraftigt av fysikaliska störningar i hårdvaran. I det här projektet utvecklades ett program vid namn Quantum noisy Algorithm Simulator (QnAS) för att kunna simulera kvantalgoritmer som påverkas av relaxation, urfasning, excitation och oönskad interferens. Målet var att programmet skulle kunna simulera kvantalgoritmer med upp till 15 kvantbitar, vara effektivt och enkelt att använda. Projektet utfördes på uppdrag av Wallenberg Centre for Quantum Technology och har därför deras kvantdatorimplementering i åtanke, men programmet kan även generaliseras för andra kvantdatorer. Programmet är skrivet i Python och bygger på Python-paketet QuTiP. Simuleringarna görs med den stokastiska metoden Monte Carlo-vågfunktionen för att kunna simulera så många kvantbitar som möjligt. Resultatet blev ett Python-paket som enkelt går att installera med pip. Programmet verifierades genom att simulera en enkel kvantalgoritm och jämföra resultatet med experimentell data. I rapporten presenteras även resultat som demonstrerar programmets funktionalitet och prestanda. Antalet kvantbitar som går att simulera i programmet begränsas av datorns prestanda och det anses rimligt att simulera upp till 13 kvantbitar på en persondator medan ett datorkluster borde kunna simulera fler än 15. Förhoppningen är att programmet ska kunna bidra till den fortsatta utvecklingen av kvantdatorer genom att exempelvis ge information om hur störningståliga algoritmer är, indikera vilka hårdvaruparametrar som behöver förbättras samt hjälpa till vid felsökning av experimentella resultat. Vidare diskuteras potential för vidareutveckling för att utöka QnAS användbarhet.

Nyckelord: qnas, kvantdator, kvantalgoritm, kvantbit, python, algoritm, qutip.

Abstract

Quantum computers are predicted to vastly outperform classical computers for certain calculations in the future and are therefore one of the most talked about research fields of our time. For the time being, quantum computers are however severly limited by the physical noises in the hardware. In this project a program called Quantum noisy Algorithm Simulator (QnAS) was developed in order to simulate quantum algorithms affected by relaxation, dephasing, excitation and unwanted interference between qubits. The aim was for the program to be able to simulate up to 15 qubits, be efficient and easy to use. The project was conducted on behalf of Wallenberg Centre for Quantum Technology and therefore had their specific implementation in mind, but the program could be generalised for other implementations of quantum computers as well. The program is written in Python and is based on the package QuTiP. The simulations are performed by the Monte Carlo wave function method in order to simulate as many qubits as possible. The result was a Python package that can be installed via pip. The program was verified by simulating a simple quantum algorithm and comparing the result with experimental data. Also presented in the report are results that demonstrate the functionality and performance of the program. The number of qubits that can be simulated is limited by the performance of the computer and it is considered reasonable to simulate up to 13 qubits on a personal computer whilst more than 15 qubits could be simulated on a computer cluster. The ambition is for the program to be used in the further development of quantum computers by for example giving information about noise resistance of algorithms, indicating which hardware parameters need to be improved and for troubleshooting experimental results. Some potential further developments that could increase the usability of QnAS are also discussed.

Keywords: qnas, quantum computer, quantum algorithm, qubit, python, algorithm, qutip.

Förord

Vi vill tacka våra handledare Anton och KD som möjliggjort projektet och bjudit på både skratt och lärdomar. Projektet har varit otroligt intressant, utmanande och lärorikt, både vad det gäller kvantfysik och programutveckling. När problem har uppstått har vi alltid kunnat lita på handledarnas vägledning och engagemang.

Axel Blom, Albin Edenmyr, Edvin Martinson, Ludvig Nordqvist, Didrik Palmqvist och Isak Wikman
Göteborg, maj 2022

Förkortningar

DRAG	-	Derivative Removal by Adiabatic Gate
MCWF	-	Monte Carlo Wave Function
NISQ	-	Noisy Intermediate Scale Quantum
pip	-	Package Installer for Python
QHO	-	Quantum Harmonic Oscillator
QnAS	-	Quantum noisy Algorithm Simulator
QuTiP	-	Quantum Toolbox in Python
VQA	-	Variational Quantum Algorithm
WACQT	-	Wallenberg Centre for Quantum Technology

Grindbeteckningar

CZ	-	Kontroll-Z
HD	-	Hadamard
$iSWAP$	-	Imaginary swap
PX	-	Pauli-X
PY	-	Pauli-Y
VPZ	-	Virtuell Pauli-Z

Innehåll

1 Inledning

I projektet konstruerades ett program vid namn QnAS (*Quantum noisy Algorithm Simulator*), för att kunna simulera kvantalgoritmer som påverkas av ett antal störningar. Här introduceras och motiveras projektet genom att ge en bakgrund till kvantdatorers potential och de utmaningar forskingsfältet står inför. Utifrån bakgrunden formuleras projektets syfte, vilket även konkretiseras genom att förklara vilka funktioner programmet ska ha.

1.1 Bakgrund

Utvecklingen av kvantdatorer är ett av vår tids stora forskningsområden och drivs på av flera stora globala aktörer såsom företag i form av Google, IBM och Microsoft, universitet som MIT och Oxford, samt en mängd mindre företag vilka är dedikerade enbart åt kvantdatorutveckling. Under de senaste åren har de 10 stater som investerat mest tillsammans investerat hela 21 miljarder US-dollar i utvecklingen av kvantdatorer [1]. Här på Chalmers leds utvecklingen av WACQT (*Wallenberg Centre for Quantum Technology*), ett tolvårigt forskningsprojekt inom kvantteknologi med ett forskningsanslag på 1 miljard svenska kronor [2].

Anledningen till guldrushen inom kvantteknologi, kvantdatorer i synnerhet, är den enorma potential som spås för kvantdatorer. Kvantfysikaliska fenomen såsom superposition och sammanflätning gör att enbart $\log_2(N)$ kvantbitar (eng. *qubits*) kan hålla lika mycket information som N klassiska bitar [3], vilket innebär att kvantdatorer har potential att öka beräkningshastigheten exponentiellt jämfört med klassiska datorer. Google demonstrerade 2019 denna kvantöverlägsenhet (eng. *quantum supremacy*) genom att låta sin 53 kvantbitar stora kvantdator utföra en specifik beräkning på ungefär 200 sekunder, vilken uppskattades ta 10 000 år på den bästa tillgängliga superdatorn [4].

Lösningen till det specifika problemet är i sig inte användbart, men demonstrationen av kvantdatorns beräkningskraft är en milstolpe som förstärker tron på att kvantdatorer i framtiden kan göra nytta för mänskligheten [5]. Universellt programmerbara kvantdatorer [6] skulle möjliggöra bland annat simulering av större kvantmekaniska system, vilket var syftet som gav R. Feynman idén om kvantdatorer i början av 1980-talet [7]. Att simulera molekylers beteenden kan vara revolutionerande inom utveckling av exempelvis läkemedel eller material [8]. Primtalsfaktorisering [9] och stora optimeringsuppgifter är andra problem som kvantdatorer på sikt tros kunna lösa [10].

Ett av de stora hindren för att realisera universellt programmerbara kvantdatorer är den icke-ideala, störningsbenägna hårdvaran vilken försämrar kvantalgoritmernas snabbhet och precision [10]. På sikt ska kvantdatorer kunna använda sig av så kallad kvantfelkorrigering (eng. *quantum error correction*) [11], men eftersom det kräver stora resurser med många kvantbitar dedikerade enbart för felkorrige-

ring kan universellt programmerbara kvantdatorer vara flera decennier bort [12]. I den nuvarande NISQ-eran [8] (*noisy intermediate scale quantum*-eran) är det möjligt att kvantdatorer för första gången blir användbara. Om och hur kvantdatorer faktiskt kommer göra nytta under NISQ-eran är ovisst, men simulering av sammanflätade kvantpartiklar är en möjlig tillämpning [8]. För att kvantdatorerna ska ge tillförlitliga resultat krävs dock utveckling av störningståliga kvantalgoritmer [8].

WACQT:s övergripande mål är att utveckla en kvantdator som kan lösa problem som en superdator inte klarar av [2]. För att klara målet redan under NISQ-eran behövs algoritmer som är tillförlitliga under hårdvaruförusättningarna [10]. Körning av algoritmerna på riktiga kvantdatorer kräver dock förberedelser och finjusteringar [13]. Det finns därför i dagsläget skäl till att avlasta kvantdatorerna de uppgifter som kan utföras av klassiska datorer. En metod för att avlasta experimentalister och kvantdatorer onödigt arbete kan vara att låta mindre kvantalgoritmer testas på klassiska datorer, genom att modelleras och simuleras med hänsyn tagen till verkliga störningsparametrar.

1.2 Syfte och uppgift

Syftet med projektet är att ta fram ett program som kan hjälpa till i utvecklingen av störningsrobusta kvantalgoritmer. Programmet ska helt köras på klassiska datorer där kvantalgoritmer simuleras, i syfte att ge information om hur störningskänslig algoritmerna är givet de fysikaliska parametrar som programmets användare anger. Resultatet från simuleringarna förväntas vara underlag för beslut kring huruvida algoritmen ska implementeras på en riktig kvantdator. Simuleringarnas resultat kan även användas för att diagnostisera felaktiga experimentella resultat samt ge information om vilka experimentella parametrar som behöver förbättras för att det ska vara givande att köra kvantalgoritmen på en riktig kvantdator.

Projektet är genomfört på uppdrag av WACQT och det är därför naturligt att projektet genomförs med deras kvantdatorutveckling i åtanke. Däremot inskränks inte slutproduktens tilltänkta användarbas till medarbetarna på WACQT, då programmet ska bestå av öppen källkod i Python samt vara universellt lämplig för kretsar med supraledande kvantbitar. Givetvis måste simuleringarna vara fysikaliskt korrekta, varför programmet baseras på mjukvarupaketet QuTiP (*Quantum Toolbox in Python*) vilket är en fri programvara [14].

Resultatet av projektet är ett Python-paket som kan läsa in algoritmer som Qiskit-kretsar [15], simulera algoritmerna givet parametrar för valda störningstyper (se avsnitt 1.3) samt återge kompletta tillståndsbeskrivningar för diskreta tidpunkter då algoritmerna körs. Programmet returnerar de slutliga tillståndsbeskrivningarna för alla simulerade kvanttrajektorior (eng. *quantum trajectories*), vars antal kan väljas av användaren. Ett valbart alternativ är att även en tidslista returneras, tillsammans med väntevärdet i varje tidpunkt för de observabler som användaren önskar. En fullständig kravspecifikation återges i appendix A.1.

1.3 Avgränsningar

De fysikaliska parametrarna som kommer att implementeras i programmet är störningstyperna urfasning (eng. *dephasing*), relaxation, termisk excitation och oönskad interferens mellan kvantbitar (eng. *ZZ-coupling*). Störningarna har valts eftersom de är de viktigaste störningstyperna för kvantalgoritmer på supraledande kretsar [16–18].

Initialt sattes ett mål om att programmet skulle klara av att simulera upp till 15 kvantbitar under en rimlig tid på en klassisk dator. Att uppnå målet bedömdes genomförbart med hänsyn till projektets omfång samt hårdvaruförutsättningar.

De algoritmer som huvudsakligen testats under programmets körning är så kallade VQA:er (*variational quantum algorithms*), vilka använder en klassisk optimeringsalgoritm vid sidan av kvantdatorn. Styrkorna i VQA:er är att de, genom att avlastas av klassiska datorer, är designade för att fungera på mindre kvantdatorer och är naturligt robusta mot störningar. Det är den här sortens kvantalgoritmer som tros kunna användas i NISQ-eran och är därför de mest relevanta [10].

2 Teori

Teoriavsnittet syftar till att introducera de begrepp och principer som krävs för att förstå hur QnAS har konstruerats, samt att ge en kortare fysikalisk koppling till kvantbitar och de störningar som verkar på dem. Avsnittet inleds med en beskrivning av kvantbitar för att sedan gå vidare med hur de kan beskrivas i öppna kvantsystem. Slutligen ges en beskrivning av kvantalgoritmer och hur deras grindar appliceras.

2.1 Kvantbitar

I en klassisk dator används (klassiska) bitar som minsta informationsbärande enhet. Informationen beskrivs med ett binärt system, där varje bit kan vara i något av tillstånden 0 eller 1. I en kvantdator är motsvarigheten kvantbitar. Här följer en introduktion till hur dessa beskrivs matematiskt och fysikaliskt.

2.1.1 Matematisk beskrivning

En kvantbit är ett kvantmekaniskt tvånivåsystem med tillstånden $|0\rangle$ och $|1\rangle$. Dessa två tillstånd kan skrivas som spinorer [19] i ett tvådimensionellt vektorrum:

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

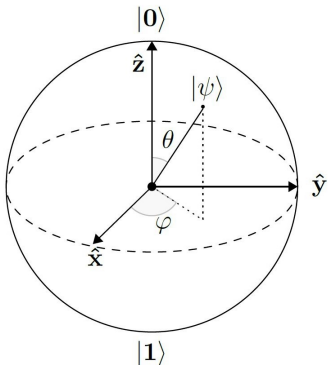
Enligt superpositionsprincipen är även kombinationer av dessa två tillstånd giltiga tillstånd, vilket gör att en kvantbits tillstånd kan skrivas som

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \begin{bmatrix}\alpha\\ \beta\end{bmatrix}. \tag{2}$$

Det krävs alltså två amplituder $\alpha, \beta \in \mathbb{C} : |\alpha|^2 + |\beta|^2 = 1$ för att beskriva en kvantbits tillstånd, vilket är mer information än vad som kan lagras i klassiska bitens nollor och ettor [3]. Ett kvantbitstillstånd kan representeras geometriskt som en vektor på *Blochsfären*, se figur 1, genom parametrisering i sfäriska koordinater enligt

$$|\psi\rangle = \cos\frac{\theta}{2}\,|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}\,|1\rangle\,, \tag{3}$$

där θ är polvinkeln och φ är azimutalvinkeln.



Figur 1: Blochsfären med $|0\rangle$ och $|1\rangle$ markerade på respektive pol. Kvantmekaniska tillstånd representeras av vektorer från origo till sfärens rand

Med fler än en kvantbit i systemet beskrivs det samlade tillståndet med en tensorprodukt av de ingående kvantbitarna. För två kvantbitar $|\psi_1\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ och $|\psi_2\rangle = \gamma\,|0\rangle + \delta\,|1\rangle$ i separabelt tillstånd beskrivs systemet alltså av

$$\begin{aligned}|\psi\rangle &= |\psi_1\rangle \otimes |\psi_2\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle)\\ &= \alpha\gamma|0\rangle \otimes |0\rangle + \alpha\delta|0\rangle \otimes |1\rangle + \beta\gamma|1\rangle \otimes |0\rangle + \beta\delta|1\rangle \otimes |1\rangle\\ &= \alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle.\end{aligned} \tag{4}$$

Tillståndet är nu en superposition av $2^2 = 4$ möjliga bastillstånd, vilket kan beskrivas som en vektor med fyra element. Generaliseringen för N kvantbitar är alltså att det behövs 2^N amplituder för att beskriva systemets tillstånd, vilket är betydligt fler än i det klassiska fallet med N bitar [3].

2.1.2 Fysikalisk beskrivning

Det finns flera olika kvantmekaniska system som kan användas till den fysikaliska implementeringen av kvantbitar. Däribland finns jonfällor (eng. *trapped ion*), elektronspinn i kisel, kvantpunkter (eng. *quantum dots*) eller ultrakalla atomer [3, 20]. Den här beskrivningen kommer dock att utgå från supraledande kvantbitar eftersom det är sådana som används av WACQT [13] och således de som modelleras i QnAS.

Supraledande kvantbitar lagrar information i nanotillverkade anharmoniska oscillatorer, vilka är harmoniska oscillatorer med förskjutna energinivåer. En kvantmekanisk harmonisk oscillator (QHO) kan realiseras med en LC-krets med oändligt antal

energinivåer med samma energigap mellan nivåerna. För en kvantbit defineras dock beräkningsunderrummet (eng. *computational subspace*) som de två lägsta energinivåerna $|0\rangle, |1\rangle$ och högre energinivåer är oönskade. Om energiskillnaden $\Delta E_{|0\rangle\to|1\rangle}$ är lika stor som $\Delta E_{|1\rangle\to|2\rangle}$ riskerar en kvantbit att drivas utanför beräkningsunderrummet. Problemet löses genom att byta ut den linjära induktorn i LC-kretsen med en Josephsonövergång (eng. *Josephson junction*) vilken inför en anharmonicitet U, det vill säga en förskjutning i energigapet så att $\Delta E_{|0\rangle\to|1\rangle} = \Delta E_{|1\rangle\to|2\rangle} + U$ [18]. Hamiltonoperatorn för en kvantbit blir då samma som för en QHO, plus en anharmonicitetsterm. I frekvensenheter ($\hbar = 1$) fås då

$$\hat{H} = \omega_q \hat{a}^\dagger \hat{a} + \frac{U}{2} \hat{a}^\dagger \hat{a}^\dagger \hat{a} \hat{a}, \tag{5}$$

där ω_q är kvantbitsfrekvensen, $\hat{a}$ är förintelseoperatorn (eng. *annihilation operator*) och $\hat{a}^\dagger$ är skapelseoperatorn (eng. *creation operator*) [18].

I verkligheten finns det alltså fler än två kvanttillstånd som kan ockuperas. De högre nivåerna bör beaktas i mer fysikaliskt korrekta simuleringar då det finns risk att kvantbiten lämnar beräkningsunderrummet ifall U är för liten. Om det andra exciterade tillståndet tas med i simuleringen skalas tillståndsrummets dimension med 3^N istället för 2^N.

2.2 Öppna kvantsystem

Ett slutet eller *idealt* kvantsystem växelverkar inte med sin omgivning och lyder under Schrödingerekvationen. I verkligheten finns dock alltid kopplingar till omgivningen och systemet kallas *öppet*. Öppna kvantsystem beskrivs generellt som ett litet system som växelverkar med en stor omgivning, vilket leder till att dynamiken hos det intressanta systemet förändras och andra matematiska verktyg krävs för att beskriva det [21].

2.2.1 Rena och mixade tillstånd

Kvantmekaniskt kan tillstånd beskrivas med vågfunktioner i ett Hilbertrum med bas $\{|i\rangle\}$ enligt

$$|\psi\rangle = \sum_i \alpha_i |i\rangle, \tag{6}$$

där α_i är sannolikhetsamplituder. Denna beskrivning utgår från att det finns ett definitivt (känt) tillstånd för vågfunktionen, klassiskt eller superponerat. Sådana tillstånd kallas rena [11].

I öppna kvantsystem gör kopplingen till omgivningen det omöjligt att veta exakt vilket tillstånd systemet befinner sig i, varför rena tillstånd ger en otillräcklig beskrivning. Istället kan systemet beskrivas som en ensemble av tillstånd, med

en viss sannolikhet att vara i ett visst tillstånd $|\psi_i\rangle$. Dessa tillstånd kallas *mixade* [11, 19, 22]. Den matematiska beskrivningen av denna ensemble kallas täthetsmatrisen

$$\rho = \sum_{i,j} \rho_{i,j} \left|\psi_i\right\rangle \left\langle\psi_j\right| . \tag{7}$$

Koefficienterna $\rho_{i,i}$ motsvarar sannolikheten att vara i tillstånd $|\psi_i\rangle$. Täthetsmatrisen kan genom att välja rätt bas alltid diagonaliseras [23], men även i den valda basen kommer element utanför diagonalen bli nollskilda vid tidsutveckling.

2.2.2 Masterekvationen

Ekvationen som beskriver tidsutvecklingen för ett öppet kvantsystem kallas masterekvationen. Härledningen utelämnas här men återfinns i [22, 24]. En viktig del av härledningen är dock att medelvärdesbilda över omgivningens tillstånd för att få en effektiv beskrivning av omgivningens inverkan och samtidigt endast arbeta med det intressanta systemets frihetsgrader. Masterekvationen skrivs på så kallad *Lindbladform* som

$$\frac{d}{dt}\rho = -i[\hat{H}_{\text{sys}}, \rho] + \sum_j \Gamma_j \mathcal{D}[\hat{X}_j]\rho, \tag{8}$$

där första termen är den för ett slutet kvantsystem och summeringen sker över en rad kollapsoperatorer $\hat{X}_j$ (eng. *collapse operators*) vilka beskriver den dissipativa dynamiken hos systemet med en karakteristisk takt Γ_j [21]. Superoperatorn $\mathcal{D}$ är på Lindbladform definerad som [22, 24]

$$\mathcal{D}[\hat{X}]\rho = \hat{X}\rho\hat{X}^\dagger - \frac{1}{2}\hat{X}^\dagger\hat{X}\rho - \frac{1}{2}\rho\hat{X}^\dagger\hat{X}. \tag{9}$$

2.2.3 Störningar och kollapsoperatorer

Störningarna urfasning, relaxation och excitation är alla stokastiska störningar som uppstår ur kopplingen mellan systemet och omgivningen. De kan därför modelleras med kollapsoperatorer. Var och en av dessa störningar agerar på varje enskild kvantbit med en karakteristisk störningstakt Γ, så det krävs en kollapsoperator för varje störning och kvantbit för att beskriva systemet.

Relaxation och excitation uppstår från ett rent energiutbyte med omgivningen vilket minskar ($|n\rangle \rightarrow |n-1\rangle$) respektive ökar ($|n\rangle \rightarrow |n+1\rangle$) energitillståndet [18]. Deras kollapsoperatorer är därför helt enkelt stegoperatorerna för en QHO, $\hat{X}_{\text{relax}} = \hat{a}$ och $\hat{X}_{\text{excit}} = \hat{a}^\dagger$ [22]. Termisk excitation uppstår då temperaturen i systemet $T > 0\,\text{K}$. Excitationseffekten kan dock oftast försummas vid tillräckligt låga temperaturer ($T \lesssim 40\,\text{mK}$) och tillräckligt hög övergångsfrekvens ($\omega_q/2\pi \approx 4-5$ GHz) då $\Gamma_{\text{excit}} \propto \exp\left(-\frac{\hbar\omega_q}{k_B T}\right)$, där k_B är Boltzmanns konstant och $\hbar$ är Plancks reducerade konstant [18].

Urfasning är en mer subtil process och uppstår ur att kvantbitsfrekvensen ω_q fluktuerar. Det leder till att tillståndets rotationshastighet runt Blochsfärens z-axel varierar och ger en osäkerhet i ackumulerad fas, vilket minskar koherensen [18]. Kollapsoperatorn för urfasning är $\hat{X}_{\text{urfas}} = \hat{a}^\dagger\hat{a} - \hat{I}/2$, vilken roterar tillståndet kring z-axeln.

Oönskad interferens, så kallad *ZZ-koppling*, är en störning som uppstår från interaktion mellan kvantbitar. ZZ-kopplingen är en intrinsisk del av de flerkvantbitsmodeller som används, och är till skillnad från andra störningar som behandlas i detta arbete inte stokastisk, utan bidrar med en konstant term i Hamiltonianen för varje kvantbitspar [25].

2.2.4 Monte Carlo-vågfunktionen

För att beskriva kvantbitar som påverkas av störningar i form av kollapsoperatorer, måste täthetsmatrisen tidsutvecklas. Att använda masterekvationen till det medför dock en stor beräkningsmängd redan för små kvantsystem eftersom täthetsmatrisen ρ är en $(M \times M)$-matris som alltså beskrivs av M^2 matriselement för ett M-dimensionellt tillståndsrum ($M = 2^N$ för N tvånivåkvantbitar) [21]. För att komma runt problemet utvecklades på 90-talet en alternativ stokastisk metod, känd som Monte Carlo-vågfunktionen (eng. *Quantum Trajectories* eller *MCWF*) [21].

Grundidén bakom metoden är att istället för att tidsutveckla hela täthetsmatrisen utveckla ett stort antal, N_{traj}, rena kvanttillstånd längs med olika utifrån störningar stokastiskt bestämda kvanttrajektorior. Tidsutvecklingen kan bestämmas ur ett medelvärde från de N_{traj} kvanttillstånden, vilket kan liknas vid att genomföra N_{traj} experiment och ta medelvärdet av mätningarna. Beräkningsfördelen fås ur att ett kvanttillstånd i M dimensioner beskrivs av M vektorelement istället för de M^2 matriselementen som beskriver ρ. Visserligen behöver N_{traj} hållas relativt högt för att få en tillförlitlig tidsutveckling, men på grund av den kvadratiska skalningen av ρ ger MCWF snabbt en kraftig beräkningsfördel även för stora värden på N_{traj}, eftersom M skalar exponentiellt med ökat antal kvantbitar [21].

2.2.4.1 MCWF-algoritmen

För att förklara algoritmen mer i detalj defineras först den icke-hermiteska, effektiva Hamiltonoperatorn $\hat{H}_{\text{eff}}$ enligt

$$\hat{H}_{\text{eff}} = \hat{H} - \frac{i}{2}\sum_j \hat{X}_j^\dagger \hat{X}_j, \tag{10}$$

där $\hat{X}_j$ som innan är kollapsoperatorer. Det här är den Hamiltonian som kommer agera om inget *hopp* sker. Ett hopp är när någon av kollapsoperatorerna tillåts agera på systemet, vilket alltså sker med en viss sannolikhet korresponderande

mot en viss karakteristisk takt Γ, som beskrivet i 2.2.3 [21]. Anledningen till att det krävs en effektiv Hamiltonian är att sannolikheten för att det sker ett hopp i nästa tidsintervall bör öka för varje gång det inte skedde ett hopp i det förra tidsintervallet. Detta för att få den karakteristiska takten Γ.

MCWF-algoritmens steg för ett antal trajektorior N_{traj}, ett diskret tidsintervall $[t_0, t_{\max}]$ med steglängd δt och ett antal kollapsoperatorer $\{\hat{X}_j\}_j$ presenteras i algoritm 1. Algoritmen involverar två stokastiska Monte Carlo-processer. Den första på rad 6 bestämmer om det sker något hopp och den andra på rad 10 bestämmer vilket hopp som i så fall sker [21].

Algoritm 1: MCWF-algoritmen

1 **for** $i \in \{1, 2, \ldots, N_{traj}\}$
2 **for** $t \in \{t_0, t_0 + \delta t, \ldots, t_{max} - \delta t\}$
3 Välj ett slumpmässigt tal $\epsilon \in [0,1]$, där intervallet är likformigt fördelat
4 Beräkna sannolikheten för att inget hopp sker, δp_0, (tidsutveckling under $\hat{H}_{\text{eff}}$).

$$\delta p_0 = \left| e^{-\frac{i}{\hbar}\hat{H}_{\text{eff}}\delta t} |\psi(t)\rangle \right|^2$$

5 Beräkna sannolikheten för att något hopp sker $\delta p = \sum_j \delta p_j = 1 - \delta p_0$
6 **if** $\epsilon < \delta p$
7 Beräkna sannolikheten för varje möjligt hopp δp_j, ($\sim \hat{X}_j$-hoppet)

$$\delta p_j = |\hat{X}_j|\psi(t)\rangle|^2 \delta t = \langle\psi(t)|\hat{X}_j^\dagger \hat{X}_j|\psi(t)\rangle \delta t$$

8 Skapa ett intervall $[0, \delta p]$ indelat i delintervall för varje δp_j

$$[0, \delta p] = [0, \delta p_1] \cup [\delta p_1, \delta p_2] \cup \cdots \cup [\delta p_{j_{max}-1}, \delta p]$$

9 Välj ett slumpmässigt tal $\xi \in [0, \delta p]$
10 **if** $\xi \in [\delta p_{j-1}, \delta p_j]$
11 $\hat{X}_j$-hoppet sker. Beräkna det nya kvanttillståndet från det j:te hoppet

$$|\psi_i(t + \delta t)\rangle = \frac{\hat{X}_j|\psi(t)\rangle}{\|\hat{X}_j|\psi(t)\rangle\|} = \frac{\hat{X}_j|\psi(t)\rangle}{\sqrt{\delta p_j / \delta t}}$$

12 **else**
13 Inget hopp sker. Beräkna nytt kvanttillstånd enligt

$$|\psi_i(t + \delta t)\rangle = \frac{e^{-\frac{i}{\hbar}\hat{H}_{\text{eff}}\delta t}|\psi(t)\rangle}{\|e^{-\frac{i}{\hbar}\hat{H}_{\text{eff}}\delta t}|\psi(t)\rangle\|} = \frac{e^{-\frac{i}{\hbar}\hat{H}_{\text{eff}}\delta t}|\psi(t)\rangle}{\sqrt{\delta p_0}}$$

Ur algoritmen erhålls N_{traj} tillstånd för varje tidpunkt och den genomsnittliga

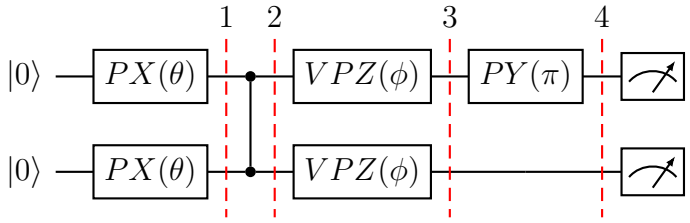
tidsutvecklingen för täthetsmatrisen kan beräknas ur

$$\rho(t) = \frac{1}{N_{\text{traj}}} \sum_{i}^{N_{\text{traj}}} |\psi_i(t)\rangle\langle\psi_i(t)| \tag{11}$$

för alla tidpunkter i tidsintervallet. Alternativt kan väntevärden beräknas direkt ur medelvärdet från de N_{traj} tillstånden vid varje tidpunkt. Tack vare de stora talens lag erhålls en tillförlitlig tidsutveckling med ett tillräckligt stort N_{traj} [21].

2.3 Kvantalgoritmer och grindar

En kvantalgoritms initialtillstånd förbereds generellt till att vara grundtillståndet $|0\cdots00\rangle$ och en kvantalgoritm kan beskrivas som att applicera olika grindar (eng. *gates*) på specifika kvantbitar, i en specifik ordning för att manipulera tillståndet. En tabell med matrisdefinitioner för de grindar som är implementerade i QnAS finns i appendix A.2. I figur 2 visas representationen av en simpel kvantalgoritm för två kvantbitar. Algoritmen kan delas upp i fyra steg där det i varje steg maximalt kan verka en grind på varje kvantbit. I slutet av algoritmen utförs en mätning vilket leder till att tillståndet kollapsar.



Figur 2: Representation av en enkel kvantalgoritm för två kvantbitar i fyra steg, varav det tredje är virtuellt (se avsnitt 2.3.3). Vinklarna $\theta, \phi \in [0,2\pi]$.

Enkvantbitgrindar representeras av rektanglarna i figur 2 och förflyttar kvanttillståndet mellan olika punkter på Blochsfären. $PX(\theta)$ roterar tillståndet med vinkeln $\theta \in [0,2\pi]$ runt x-axeln, medan $PY(\pi)$ roterar tillståndet med vinkeln π runt y-axeln. Enkvantbitgrindar kan således beskrivas av en $m \times m$-matris där m är antalet kvantbitnivåer (normalt två eller tre) [18]. $VPZ(\phi)$ är en virtuell grind som roterar tillståndet med vinkeln ϕ runt z-axeln och förklaras närmare i 2.3.3. Tvåkvantbitgrindar beskrivs istället av $m^2 \times m^2$-matriser och appliceras på två kvantbitar samtidigt, se steg 2 i figur 2 där en kontroll-Z-grind (CZ) representeras av två punkter och ett streck mellan de två kvantbitarna.

Kvantgrindar av högre ordning kan beskrivas av större matriser ($m^3 \times m^3$ för trekvantbitgrindar). Det visar sig dock att alla grindar för fler än två kvantbitar kan brytas ner till ett antal en- och tvåkvantbitgrindar då dessa kan bilda en *universal* mängd av grindar. I experiment används därför nästan alltid bara en- och

tvåkvantbitgrindar. Villkoret för universalitet är att grindarna i mängden ska kunna approximera en godtycklig unitär transformation på ett valfritt antal kvantbitar med önskad noggrannhet [3].

2.3.1 Tensorprodukt av operatorer

För att se till att grindarna verkar på rätt kvantbit i det m^N-dimensionella tillståndsrummet, där N är antalet kvantbitar, används på motsvarande sätt som för kvanttillståndet en tensorprodukt. Om $\hat{A} = \hat{A}_1 \otimes \hat{A}_2$ agerar på det separabla tillståndet i (4) erhålls

$$\hat{A}|\psi\rangle = (\hat{A}_1 \otimes \hat{A}_2)(|\psi_1\rangle \otimes |\psi_2\rangle) = \hat{A}_1|\psi_1\rangle \otimes \hat{A}_2|\psi_2\rangle. \tag{12}$$

$\hat{A}_1$ agerar alltså på $|\psi_1\rangle$ och $\hat{A}_2$ på $|\psi_2\rangle$, så positionen av grindarna i tensorprodukten avgör vilken kvantbit de agerar på. En sådan tensorprodukt kan därför användas för att agera med en specifik grind, $\hat{A}_1$, på en specifik kvantbit, $|\psi_1\rangle$, utan att påverka den andra kvantbitens tillstånd genom att låta den andra operatorn, $\hat{A}_2$, vara identitetsoperatorn $\hat{I}$. Steg 4 i figur 2 beskrivs alltså av $PY \otimes \hat{I}$. För ickeseparabla tillstånd agerar operatorn på hela tillståndet och kan inte längre sägas påverka endast en kvantbit. För fler kvantbitar och större tillståndsrum läggs bara fler faktorer till i tensorprodukten så att dimensionerna stämmer överens [26].

2.3.2 Att driva kvantbitar och roterande referensram

För att applicera en grind, vilken beskrivs av operatorn $\hat{A}$, på en kvantbit läggs en driv-Hamiltonian till kvantbits-Hamiltonianen i (5) enligt

$$\hat{H} = \omega_q \hat{a}^\dagger \hat{a} + \frac{U}{2}\hat{a}^\dagger \hat{a}^\dagger \hat{a}\hat{a} + E(t)\beta \cos(\omega_d t)\hat{A} \tag{13}$$

där ω_d är drivfrekvensen, β är drivstyrkan, $E(t)$ är en pulsfunktion (eng. *envelope function*) och $\hat{A}$ kan skrivas i termer av $\hat{a}$ och $\hat{a}^\dagger$ ($\hat{A} = \alpha\hat{a} + \gamma\hat{a}^\dagger$, $\alpha, \gamma \in \mathbb{C}$). För att förenkla uttrycket används en roterande referensram (eng. *rotating frame*) i vilken $\hat{H}$ roteras med vinkelfrekvensen ω_d genom att applicera en unitär rotationsoperator $\hat{U} = e^{i\omega_d \hat{a}^\dagger \hat{a} t}$ så att $\hat{H} \to \hat{H}_{\text{rot}}$. För unitära transformationer gäller

$$\hat{H}_{\text{rot}} = \hat{U}\hat{H}\hat{U}^\dagger + i\dot{\hat{U}}\hat{U}^\dagger, \tag{14}$$

där $i\dot{\hat{U}}\hat{U}^\dagger = -\omega_d \hat{a}^\dagger \hat{a}$. Vidare kommuterar de två första termerna i $\hat{H}$ med $\hat{U}$ då $[\hat{U}, \hat{a}^\dagger\hat{a}] = 0$ och $[\hat{a}, \hat{a}^\dagger] = 1$, samtidigt som $\hat{U}\hat{a}\hat{U}^\dagger = \hat{a}e^{-i\omega_d t}$ och $\hat{U}\hat{a}^\dagger\hat{U}^\dagger = \hat{a}^\dagger e^{i\omega_d t}$ [22]. Hamiltonianen i den roterande referensramen blir då

$$\hat{H}_{\text{rot}} = (\omega_q - \omega_d)\hat{a}^\dagger\hat{a} + U\hat{a}^\dagger\hat{a}^\dagger\hat{a}\hat{a} + E(t)\frac{\beta}{2}(\alpha\hat{a} + \gamma\hat{a}^\dagger)(e^{i(\omega_d - \omega_d)t} + e^{i(\omega_d + \omega_d)t}), \tag{15}$$

varpå två antaganden görs. Först antas att drivfrekvensen är resonant med kvantbitsfrekvensen, $\omega_q = \omega_d$, vilket är ett rimligt antagande då drivfrekvensen oftast

väljs till att vara resonant [18]. Sen görs roterande våg-approximationen (eng. *rotating wave approximation*) som bygger på att frekvensen $\omega_d + \omega_d$ i den andra exponentialtermen är mycket större än β [22]. Den andra termen kommer då att rotera väldigt snabbt runt origo jämfört med den första och kan approximeras som sitt medelvärde, $e^{i(\omega_d+\omega_d)t} \approx 0$, så länge drivstyrkan β inte är för hög [22]. Slutligen fås [27]

$$\hat{H}_{\text{rot}} = U\hat{a}^\dagger\hat{a}^\dagger\hat{a}\hat{a} + E(t)\frac{\beta}{2}\hat{A}. \tag{16}$$

2.3.2.1 Pulsfunktion

Pulsfunktionen $E(t)$ modelleras så att dess integral över drivtiden, t_d, motsvarar den vinkel θ som önskas i algoritmen enligt

$$\frac{\beta}{2}\int_0^{t_d} E(t)\,\mathrm{d}t = \theta. \tag{17}$$

Drivstyrkan β bestäms utifrån att den maximala vinkeln $\theta_{\max}$ (vilken är lika med π för enkvantbitgrindar) ska erhållas vid en maximal drivtid, $t_{\max}$, och hålls sedan konstant. För att få en mindre vinkel $\theta < \pi$ justeras alltså endast drivtiden och inte β.

2.3.3 Virtuell Pauli-Z

Ett undantag till ovanstående är den virtuella Pauli-Z-grinden, vilken till skillnad från verkliga grindar inte appliceras med en egen drivpuls. Grinden kallas virtuell av två anledningar. Dels för att en rotation runt z-axeln, vilket *VPZ* ska ge, bara ger en imaginär fasförskjutning, vilken alltså inte kan mätas och dels för att fasförskjutningen i hårdvaran implementeras som en del i de verkliga grindarnas drivpulser. Detta innebär att *VPZ*-grinden sker momentant (utan tidsåtgång) [18].

2.3.4 Kontroll-Z-grind

Kontroll-Z-grinden (*CZ*) är en tvåkvantbitsgrind som byter tecken på tillståndet $|11\rangle$ men lämnar övriga tillstånd oförändrade [18] enligt

$$CZ|11\rangle = -|11\rangle\,. \tag{18}$$

Den fysikaliska implementeringen av *CZ* hos WACQT bygger på att driva den ena kvantbiten till det andra exciterade tillståndet ($|02\rangle$ eller $|20\rangle$) och sedan tillbaka till $|11\rangle$, vilket innebär en full rotation ($\theta_{\max} = 2\pi$) [13]. Här utnyttjas alltså tillstånd utanför beräkningsunderrummet för att åstadkomma teckenförändringen. För att kompensera för anharmoniciteten drivs systemet på en frekvens som aktiverar övergången mellan $|11\rangle$ och $|02\rangle$ för de berörda kvantbitarna, vilka nu måste modelleras med åtminstone tre energinivåer. Eventuellt kan även en fjärde energinivå krävas för att få med den potentiella termiska excitation som skulle kunna ske i det högre energitillståndet, exempelvis $|02\rangle \rightarrow |03\rangle$.

3 Metod

I det här avsnittet beskrivs hur QnAS-paketet är designat och vilka tester som har genomförts för att verifiera att det fungerar som det ska.

3.1 Programutveckling

För att förklara hur programmet är uppbyggt och fungerar inleds beskrivningen med vad en användare av QnAS behöver mata in, för att sedan redogöra för vad programmet gör med den informationen. En övergripande bild av programmets struktur presenteras i appendix B.

3.1.1 Inmatning

Alla parametrar för QnAS förs in till en funktion vid namn `solve`, vilken agerar som en bas för hela programmet. I `solve` finns också felhantering för att kontrollera alla inmatade parametrar och förhindra oväntade felmeddelanden vid körning. Felhantering finns till viss del även i andra delar av programmet för ökad robusthet.

För att underlätta konstruktionen av kvantkretsar som ska simuleras i QnAS utvecklades en tolk mellan OpenQASM 2.0-formatet, ett standardformat för beskrivningen av kvantkretsar, och QnAS inbyggda kvantkretsformat. För att designa en kvantkrets som ska simulueras i QnAS kan en användare antingen använda sig av Python-paketet Qiskit som utvecklats av IBM eller av annan mjukvara. Objekten som skickas in i tolken ska vara av typen `QuantumCircuit`. Om en krets konstrueras i Python med hjälp av Qiskit kan denna direkt skickas in i tolken utan att modifieras. Om användaren däremot konstruerar en krets med hjälp av annan mjukvara till en OpenQASM-fil behöver denna konverteras till rätt typ av objekt. Det görs med kommandot `from_qasm_file` som finns tillgängligt i Qiskit. I de kvantkretsar som konstrueras kan enbart de grindar som finns tillgängliga på WACQT (förutom $iSWAP$) användas, se appendix A.2, samt en identitetsgrind som kan användas för att säkerställa att alla grindar appliceras i rätt steg av kretsen.

Användaren behöver också definiera en fil där parametrar för kvantbitarna specificeras. Detta görs i form av en .csv-fil där varje rad, förutom den första som är en rad med rubriker, beskriver parametrar för en kvantbit. I första kolumnen numreras kvantbitarna för att förtydliga vilken kvantbit raden tillhör. Kolumn två, tre och fyra består av värden för relaxationstakt, urfasningstakt och excitationstakt i respektive kolumn angivna i MHz. Kolumn fem består av värden för anharmonicitet för kvantbitarna, även det angivet i MHz. Den sista kolumnen beskiver antalet energinivåer för varje kvantbit. Alla värden separeras med semikolon.

Till sist finns även ett par valfria parametrar som en användare kan specificera, men som inte är nödvändiga. Den första är antalet trajektorior, N_{traj}, vilket har ett standardvärde på 500. Användaren kan också specificera maxtider för både

enkvantbitgrindar och tvåkvantbitgrindar. Dessa har standardvärden på 20 respektive 200 ns. För att få med oönskad interferens behöver användaren definiera en $N \times N$ matris som beskriver kopplingen mellan de N kvantbitarna, där elementen anges i Hz. Om användaren vill spara tidsutvecklingen under algoritmen kan den göra det genom att sätta den booleska variabeln `storeTimeDynamics` till `True` och specificera en eller flera väntevärdesoperatorer tillsammans med addreserade kvantbitar, vilket görs med hjälp av QuTiP:s `Qobj`. Antalet kvantbitar som QnAS räknar med fås från kvantkretsen som användaren designat med hjälp av Qiskit eller annan mjukvara.

3.1.2 Grundsystem

Programmets viktigaste komponenter ligger i Python-paketet QuTiP och dess inbyggda funktioner. För att kunna utnyttja dem på bästa sätt krävs dock en robust hantering av alla parametrar för kvantbitar och kvantgrindar. För vardera av dessa har två klasser skapats, `Qubit` respektive `AlgStep`. En kvantbit skapas som ett `Qubit`-objekt och håller information om antal energinivåer, störningstakter för de olika störningarna och anharmonicitet som användaren specificerat. Om kvantalgoritmen använder sig av flera kvantbitar sparas de olika `Qubit`-objekten i en lista.

Delprogrammet som används för att konvertera en kvantkrets från Qiskit till QnAS-format delar upp kretsen i lika många steg som den är djup. Exempelvis delas kretsen i figur 2 upp i 4 nivåer med grindar. För varje nivå skapas ett `Algstep`-objekt som sparas i en lista. Resultatet blir en lista som är lika lång som kvantalgoritmen är djup där varje element i listan motsvarar precis en nivå med grindar. `AlgStep`-objekten håller information om vilka grindar som ska tillämpas, på vilka kvantbitar och med vilka vinklar. Programmet byggdes på detta sätt för att på bästa sätt kunna hantera virtuella kvantgrindar som appliceras momentant och inte ska påverkas av störningar.

Kvantgrindarna definieras med hjälp av klassen `Qobj`, QuTiP:s inbyggda klass för att representera operatorer och tillstånd i antingen matrisform eller vektorform. För att enkelt kunna skapa specifika grindar har ett grindbibliotek, `gateLib`, konstruerats. Detta innehåller funktioner för att skapa varje kvantgrind i appendix A.2 genom tensorprodukter mellan identitesoperatorer och olika sammansättningar av `Qobj`-operatorer för att agera på önskad kvantbit, se avsnitt 2.3.1.

Innan programmet kan lösa dynamiken för systemet måste dessutom ett initialtillstånd och kollapsoperatorer för störningarna skapas. Initialtillståndet skapas som tensorprodukter av tillståndsvektorer för varje kvantbit i sitt grundtillstånd. Kollapsoperatorerna för urfasning, relaxation och excitation läggs till i en lista, där varje element i listan är roten ur störningstakten multiplicerat med `Qobj`-operatorn för motsvarande störning som agerar på en specifik kvantbit. Det blir alltså en lista som innehåller alla kollapsoperatorer för alla kvantbitar.

Nästa steg i programmet är själva lösningsalgoritmen (`mainAlgorithm`), där QuTiP:s inbyggda funktion för MCWF-algoritmen, `mcsolve`, spelar en viktig roll. Inmatningen till `mainAlgorithm` är i princip allt som programmet eller användaren gjort hittills, alltså en lista av `AlgStep`-objekt som håller alla steg i kvantalgoritmen, en lista med `Qubit`-objekt, en lista med kollapsoperatorer, en matris som beskriver ZZ-kopplingen mellan kvantbitarna, ett initialtillstånd, maxtider för grindar, antal trajektorior samt ett booleskt värde (med eventuella väntevärdesoperatorer) som avgör om programmet ska spara tidsutvecklingen.

I `mainAlgorithm` skapas inledningsvis en lista med N_{traj} identiska kopior av initialtillståndet, samt de konstanta termerna för anharmonicitet och oönskad interferens i Hamiltonianen. Efter detta går programmet igenom kvantalgoritmen steg för steg, där den för varje steg skapar tidsberoende operatorer för de verkliga grindarna samt icke-tidsberoende virtuella operatorer. Det skapas även en lista med tider, som är olika lång beroende på vilka grindar som tillämpas, vilken senare talar om för `mcsolve` hur långt tidsspann den ska lösa systemet för. De tidsberoende operatorerna skapas genom att samla operatorerna tillsammans med en tidsfunktion, som finns i en del av programmet vid namn `envelopeFunction`, i ett `QobjEvo`-objekt. Det gör att operatorerna kan verka under olika lång tid och, i detta fall, med en styrka som varierar proportionellt mot en period av en $\sin^2$-formad funktion

$$E(t) = \sin^2\left(\frac{t\pi}{t_d}\right), \tag{19}$$

där t_d är drivtiden för grinden. Som beskrivet i avsnitt 2.3.2.1 kommer integralen av pulsfunktionen motsvara vinkeln som grinden roterar kvantsystemet med.

För varje steg i kvantalgoritmen används QuTiP:s funktion `parfor`, vilket är en slags parallellbearbetning av en `for`-loop på en dators olika kärnor, för att parallellt kunna utföra flera `mcsolve` samtidigt med olika initialtillstånd. Det utförs N_{traj} parallella `mcsolve`, där varje körs med endast en trajektoria. I det första steget i kvantalgoritmen har alltså alla `mcsolve` samma initialtillstånd. Från de parallellbearbetade `mcsolve` fås en lista med N_{traj} olika sluttillstånd, på vilka eventuella virtuella grindar appliceras parallellt genom matrismultiplikation. Till slut sätts sluttillstånden som de nya initialtillstånden inför nästa steg i kvantalgoritmen. `mainAlgorithm` fortsätter tills alla steg i kvantalgoritmen är klara och skickar sedan tillbaka en lista med N_{traj} olika sluttillstånd. Om användaren valt att spara tidsutvecklingen körs en variant av detta som även skickar tillbaka en lista med väntevärden för de valda väntevärdesoperatorerna samt en lista med tider för respektive väntevärde.

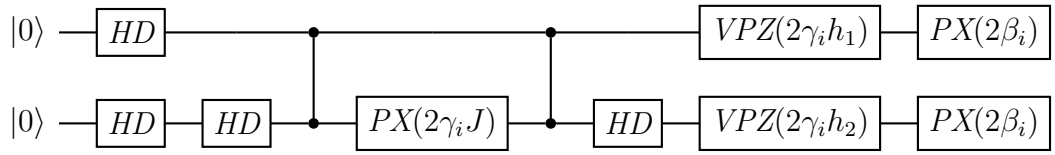
3.2 Programverifiering

Två viktiga aspekter av programmets funktionalitet är dess förmåga att korrekt simulera de fysikaliska processer som förekommer vid körningen av en kvantalgoritm, samt att detta sker inom rimlig tid. Den tid det tar för programmet att köras

på en specifik dator påverkas framförallt av tre faktorer, förutom hur bra datorns hårdvara är. Faktorerna är antalet kvantbitar, antalet trajektorior samt djupet hos den simulerade kvantkretsen.

3.2.1 Fysikalisk verifiering

För att testa programmets trogenhet mot verkligheten har ett experiment återskapats i simulering med QnAS. De experimentella resultaten som reproducerades kommer från [13]. Denna artikel beskriver hur exakta täcknings-problemet (eng. *exact cover problem*) har lösts med en kvantprocessor bestående av två transmonkvantbitar. Problemet kan översättas till att hitta grundtillståndet hos Isinghamiltonianen $\hat{C} = h_1\hat{\sigma}_1^z + h_2\hat{\sigma}_2^z + J\hat{\sigma}_1^z\hat{\sigma}_2^z$ där h_1, h_2 och J är konstanter med värden enligt [13, tab. 1] som normerar kostnadsfunktionen $F = \langle\hat{C}\rangle$ till $[-1,1]$. Vid testning definierades $\hat{\sigma}_i^z$ som $2\hat{a}_i^\dagger\hat{a}_i - \hat{I}$, där $i \in \{1,2\}$ betecknar vilken kvantbit operatorn verkar på. För att hitta grundtillståndet till $\hat{C}$ kan kretsen från figur 3 tillämpas. Hamiltonianen för kretsen beror av två parametrar γ och β motsvarande vinklar för reella rotationer. Det innebär att kostnadsfunktionen kan plottas som ett landskap mot dessa två parametrar. De fyra problemen i experimentet, (a)-(d), med olika värden på J, h_1 och h_2, simulerades i QnAS med de störningstakter och anharmoniciteter som specificeras i [13, tab. 3].



Figur 3: Kvantalgoritmen för experimentet i [13] där h_1, h_2 och J är faktorer i kostnadsfunktionen för de olika problemen och $\gamma_i, \beta_i \in [0,\pi]$ är de vinklar som varieras i algoritmen. *HD* är Hadamardgrinden, se appendix A.2.

3.2.2 Prestandatestning

Eftersom simuleringstiden i programmet är kraftigt beroende av testdatorns hårdvara (främst antal kärnor och arbetsminne) är det mer intressant för den här studien att undersöka hur simuleringstiden varierar beroende på simuleringens parametrar än den specifika körtiden. För att testa prestandan av programmet gjordes därför olika tester då programmets körtid analyserades beroende på antalet kvantbitar, antalet trajektorior samt kvantalgoritmens djup. Alla tester för samma parameter gjordes på samma dator (se appendix C.2 för datorspecifikationer) för att utesluta olika datorers påverkan på körtiden. Då antalet kvantbitar varierades hölls djupet på kvantalgoritmen och antalet trajektorior konstant. Kretsen, som visas i appendix C.1, konstruerades i syfte att sammanfläta alla kvantbitar till ett ickeseparabelt tillstånd då separabla tillstånd eventuellt är enklare att simu-

lera. Sammanflätningen uppnås genom att parvis applicera *CZ*-grindar och resten av algoritmens steg består av enkvantbitgrindar.

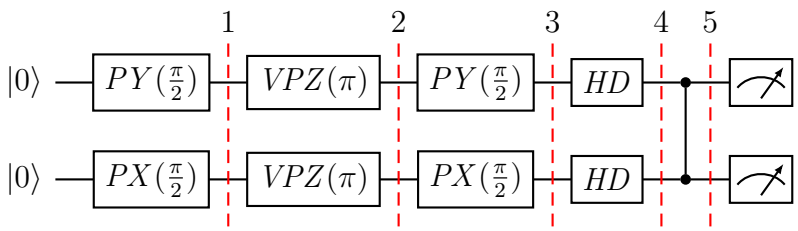
En liknande princip användes då djupet på kvantalgoritmen varierades och antal kvantbitar och trajektorior hölls konstant. För att variera djupet upprepades de sista sju stegen i figur 3 olika antal gånger med grindarnas argument satta till π. Totalt gjordes 20 iterationer och således var kretsens djup i den sista iterationen 141. När djupet varierades gjordes även ett test på hur algoritmens fidelitet berodde av djupet. Fideliteten definierades som en pseudometrik, d, mellan det störningspåverkade tillståndet $|\psi_n\rangle$ och det störningsfria tillståndet $|\psi_{id}\rangle$ enligt [11]

$$d = \frac{1}{2}\mathrm{tr}\big|\,|\psi_n\rangle\langle\psi_n| - |\psi_{id}\rangle\langle\psi_{id}|\,\big|. \tag{20}$$

Störningsparametrarna som användes valdes efter kommunikation med handledare (K Debnath maj 2022) och var $\Gamma_{\text{relax}} = 33\,\text{kHz}, \Gamma_{\text{urfas}} = 25\,\text{kHz}, \Gamma_{\text{excit}} = 0$ och en ZZ-koppling på 100 kHz mellan kvantbitarna. En simulering med halverade störningstakter genomfördes också för att se att fideliteten beror på störningstakterna. Till sist varierades även antalet trajektorior då samma kvantalgoritm med lika många kvantbitar simulerades.

3.2.3 Tidsutveckling

För att visa programmets förmåga att spara tidsutvecklingen gjordes ett test som samtidigt demonstrerade grindarnas funktionalitet. Kretsen i figur 4 utformades för att koncist demonstrera de viktigaste egenskaperna hos grindarna *PX*, *PY*, *VPZ*, *HD* och *CZ*. De tre första stegen i figur 4 förflyttar kvantbitarna på sina respektive Blochsfärer och återställer dem sedan till initialtillståndet $|00\rangle$. Väntevärdena som mättes i de första tre stegen var kvantbitarnas individuella sannolikheter att befinna sig i det första exciterade tillståndet $|1\rangle$, samt deras projektioner på x- eller y-axeln i Blochsfären. Syftet med väntevärdena var att demonstera ockupationssannolikheter och fasegenskaper. I de två avslutande stegen sattes båda kvantbitarna i superposition av *HD*-grindar, varpå *CZ* skapade sammanflätning mellan kvantbitarna i syfte att demonstrera teorin i avsnitt 2.3.4. De väntevärden som mättes var ockupationssannolikheterna för systemets tillstånd $\frac{|00\rangle+|01\rangle+|10\rangle+|11\rangle}{2}$, $\frac{|00\rangle+|01\rangle+|10\rangle-|11\rangle}{2}$ och $|02\rangle$.



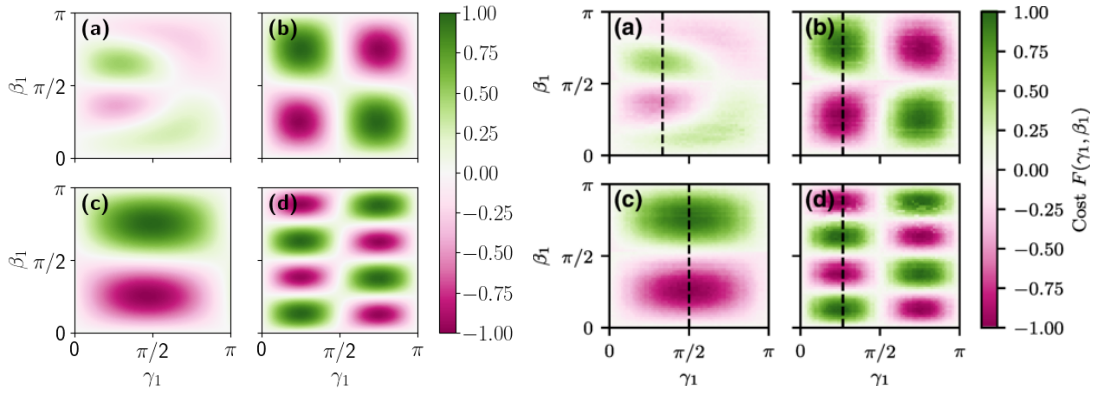
Figur 4: Algoritmen som används för att demonstrera tidsutvecklingen för ett system av två kvantbitar.

4 Resultat

Resultatet av projektet är Python-paketet QnAS. Det finns tillgängligt via pip, *package installer for Python*, och importeras in i ett Python-script med kodraden `import qnas`. Härifrån kan en användare främst kalla funktionerna `solve`, som beskrevs i 3.1.1, och `help`, som ger instruktioner om hur programmet används. Detta är allt som krävs för att kunna simulera en kvantkrets, med vilka parametrar som helst. För den mer avancerade användaren finns även möjlighet att kalla på alla funktioner och klasser i QnAS grundsystem, för att kunna skräddarsy sin simulering. Programmets källkod finns även tillgänglig på GitHub för vidare utveckling. I det här avsnittet presenteras även resultaten för den fysikaliska verifieringen, prestandatestningen och tidsutvecklingen.

4.1 Fysikalisk verifiering

QnAS testades mot en algoritm från ett experiment på två kvantbitar på Chalmers kvantdator [13], som beskrivet i avsnitt 3.2.1. I figur 5 visas en funktionsyta av kostnadsfunktionen, $F = \langle \hat{C} \rangle$, för olika värden på vinkelparametrarna β_1 och γ_1 för både simuleringen i QnAS och resultatet från experimentet.

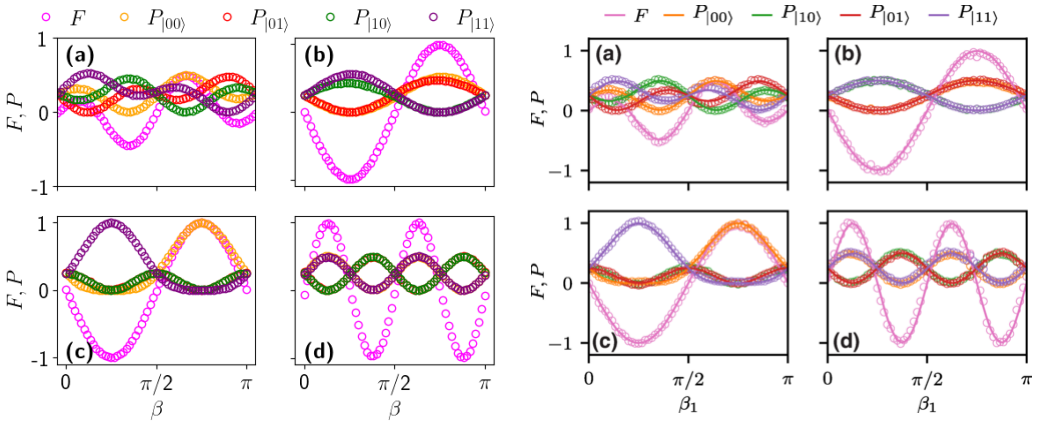


Figur 5: Kostnadsfunktionen från Isinghamiltonianen för problem (a)-(d) i [13], från simulering i QnAS till vänster och experimentet till höger. Bilderna följer samma färgkodning, vilken motsvarar kostnadsfunktionens amplitud.

Numeriska jämförelser mellan experimentell och simulerad data visas i tabell 1. Tabellen visar de optimala vinkelargumenten β, γ, experimentella och simulerade minima samt avvikelser i minima för de olika problemen. Även den genomsnittliga avvikelsen mellan varje datapunkt presenteras. I figur 6 visas kostnadsfunktionen längs de sträckade linjerna i figur 5 i magenta, samt sannolikheten för att tillstånden $|00\rangle, |10\rangle, |01\rangle$ och $|11\rangle$ är ockuperade.

Tabell 1: Jämförelser av experimentell och simulerad data för problemen (a)-(d).

	β		γ		Minima		Avvikelse	Genomsnittlig
	exp.	sim.	exp.	sim.	exp.	sim.	i minima	avvikelse
(a)	1,15	1,15	0,89	0,89	−0,57	−0,45	21 %	9 %
(b)	0,84	0,79	0,84	0,79	−1,05	−0,99	6 %	5 %
(c)	0,73	0,79	1,52	1,52	−1,05	−1,00	5 %	4 %
(d)	2,67	2,78	0,79	0,79	−1,09	−0,99	9 %	6 %



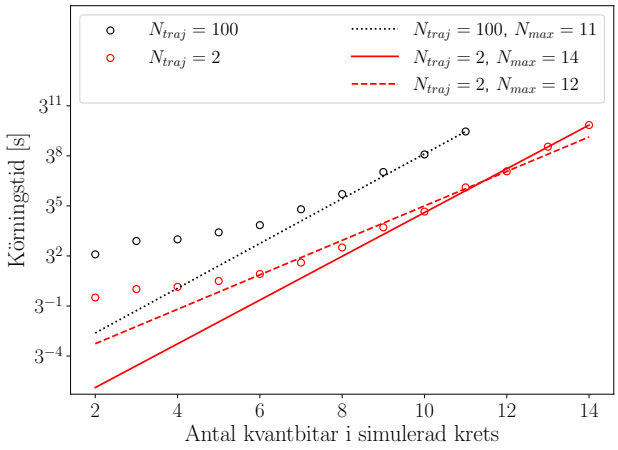
Figur 6: Kostnadsfunktionen och sannolikheter för att vissa tillstånd är ockuperade längs de fixerade värdena på γ_1 markerade med vertikala linjer till höger i figur 5. Simulering i QnAS till vänster och från experimentet [13] till höger.

4.2 Prestandatestning

Resultatet för de två prestandatesterna redovisas nedan i figur 7 och figur 8. I figur 7 visas tidsskalningen som funktion av antal kvantbitar, N. Testet utfördes för varierande antal trajektorior vilket motsvarar de olika mätserierna i figuren. Till dessa mätserier anpassades även tidsfunktioner enligt $t(N) = C \cdot B^N$ där C och B är konstanter, vilka visas i tabell 2. Anpassningarna utfördes med minstakvadratmetoden.

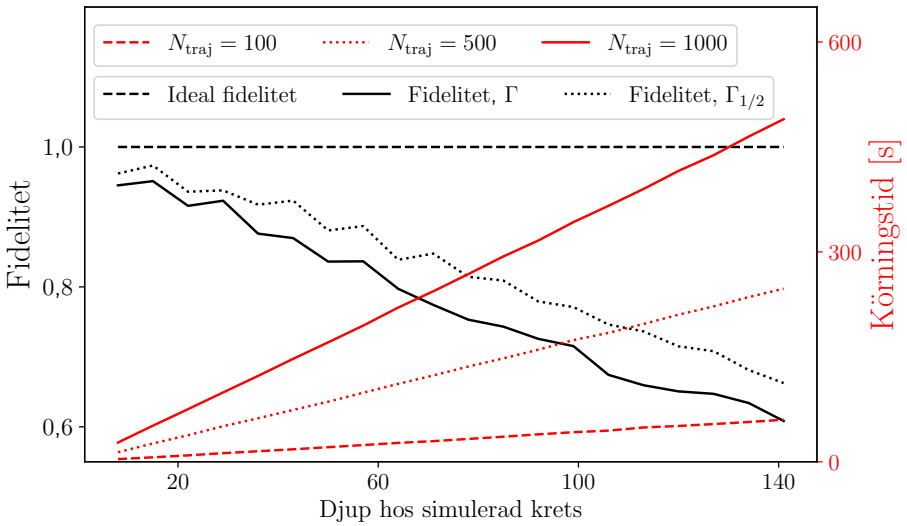
Tabell 2: Parametrar för kurvanpassningarna av tiddsskalningen mot antalet kvantbitar där $t(N) = C \cdot B^N$ för N kvantbitar där $N_{\max}$ är antalet kvantbitar i den största simulerade kretsen.

N_{traj}	C [s]	B	$N_{\max}$
100	$2{,}95{\cdot}10^{-3}$	4,37	11
2	$8{,}66{\cdot}10^{-5}$	4,22	14
100	$4{,}76{\cdot}10^{-2}$	3,29	10
2	$2{,}87{\cdot}10^{-3}$	3,11	12



Figur 7: Simuleringstid mot antalet kvantbitar för några olika N_{traj}, där $N_{\max}$ är största antalet simulerade kvantbitar i mätserien. Cirklar representerar mätdata och linjer några av de kurvanpassningar som specificeras i tabell 2.

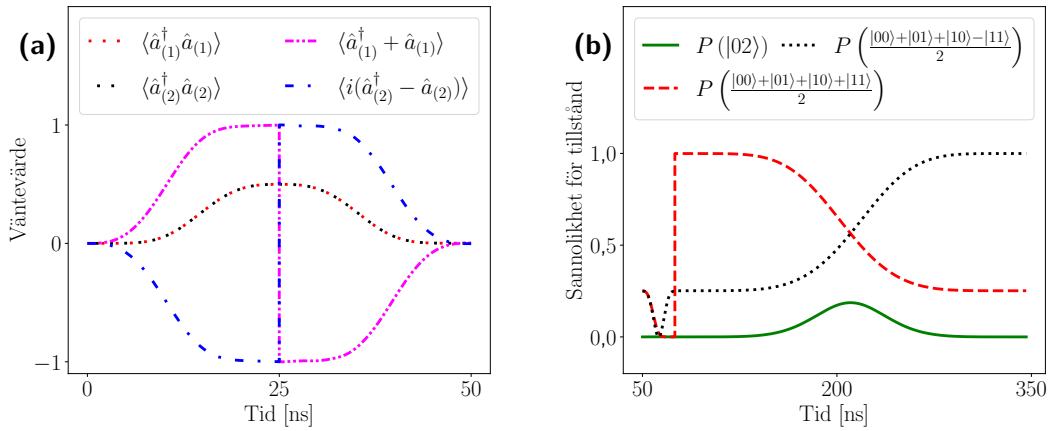
I figur 8 presenteras hur simuleringstiden och fideliteten påverkas av kretsdjupet. Ett tydligt linjärt beroende för både fideliteten och simuleringstiden konstateras. I figuren ses också att lutningen på simuleringstidens linjära beroende ökar med antalet trajektorior. Fideliteten beror på störningstakterna vilket även det ses i figur 8 där halverade störningstakter ger en högre fidelitet.



Figur 8: Fidelitet (vänster) och simuleringstid (höger) mot algoritmdjup för ett antal olika värden på N_{traj} och olika störningstakter. För fideliteten beror resultatet inte av N_{traj}, eftersom medelvärdet konvergerar för höga N_{traj}, varför endast $N_{\text{traj}} = 1000$ visas utöver det störningsfria fallet. Störningstakterna för Γ specificeras i avsnitt 3.2.2, där $\Gamma_{1/2}$ innebär halverade takter.

4.3 Tidsutveckling

Här presenteras väntevärden enligt beskrivningen i avsnitt 3.2.3. Eftersom *CZ*-grindens långa drivtid på 271 ns tog upp stora delar av algoritmens körtid delades visualiseringen in i två delalgoritmer. I figur 9(a) visas resultatet för algoritmens tre första steg. Fasinformation om systemet erhålls från väntevärdena för operatorerna $\hat{a}^\dagger_{(1)} + \hat{a}_{(1)}$, vilken ger första kvantbitens projektion på x-axeln i Blochsfären, samt $i(\hat{a}^\dagger_{(2)} - \hat{a}_{(2)})$, som ger andra kvantbitens projektion på y-axeln. Med kännedom om algoritmen i figur 4 inses det att PX och PY skickar en kvantbit i tillståndet $|0\rangle$ till olika positioner på Blochsfären. Den momentana fasändringen vid 25 ns beror på att *VPZ*-grinden appliceras. I samma graf visas även väntevärden för operatorerna $\hat{a}^\dagger_{(j)}\hat{a}_{(j)},\ j = 1{,}2$, vilka ger kvantbitarnas respektive sannolikheter att mätas i första exciterade tillståndet $|1\rangle$. Tidsutveckingen för algoritmens två avslutande steg visas i figur 9(b). Det kan utläsas att *HD*-grindarna tar systemet från initialtillståndet $|00\rangle$ till det mixade tillståndet $\frac{|00\rangle+|01\rangle+|10\rangle+|11\rangle}{2}$. När *CZ*-grinden appliceras illustrerar figuren att $|02\rangle$-tillståndet används för att byta tecken på tillståndets $|11\rangle$-term, och därmed nå det sammanflätade tillståndet $\frac{|00\rangle+|01\rangle+|10\rangle-|11\rangle}{2}$.



Figur 9: Tidsutveckling av algoritmen i figur 4. I (a) illustreras kvantalgoritmens tre första steg där de två kvantbitarnas excitationssannolikhet och projektioner på x- respektive y-axeln visas. I (b) visas de avslutande två stegen där sannolikheten för olika tillstånd ritats ut.

5 Diskussion

Diskussionen syftar till att diskutera huruvida programmet utifrån resultaten uppfyller kraven i appendix A.1 samt att kommentera de övergripande metodval som gjorts i dess utveckling. Även framtida utvecklingsmöjligheter diskuteras.

5.1 Kravuppfyllande

Det mest centrala kravet för QnAS, att kunna simulera kvantalgoritmer med upp till 15 kvantbitar i närvaro av olika sorters störningar, är till stor del uppnått. Tekniskt sett är QnAS inte heller begränsat till 15 kvantbitar, men körtiden på en vanlig dator för mer än 13 kvantbitar är längre än vad som kan anses vara rimlig. Programmet tar hänsyn till relaxation, urfasning, excitation, oönskad interferens och anharmonicitet hos kvantbitar, vilket var en del av detta krav ihop med att programmet är generaliserat för olika antal energinivåer hos kvantbitarna. Att störningarna påverkar kvantalgoritmen visas i figur 8. Fler störningstyper kan enkelt läggas till genom att lägga till fler kollapsoperatorer och utöka antalet kolumner med störningstakter i inmatningsfilen.

Nästa mål för QnAS var att kunna simulera de kvantgrindar som används av WACQT, vilket till stor del har uppnåtts. Alla grindar, förutom *iSWAP* som inte implementerades i tid, finns tillgängliga i QnAS. För vidare utveckling skulle olika sorters bibliotek för grindar kunna konstrueras och därmed utöka användarkretsen för QnAS. Till exempel skulle utöver det nuvarande biblioteket, med grindar från WACQT, ett bibliotek med grindar för IBM:s kvantdatorer kunna konstrueras. På så sätt skulle en användare ha möjlighet att välja olika grindar beroende på vilken kvantdator som är aktuell. För att konstruera nya bibliotek krävs dock information om den fysikaliska implementeringen av grindarna på respektive kvantdator.

Förutom testet som beskrivs i avsnitt 3.2.3 har grindar endast testats, under utvecklingen, för diverse olika initialtillstånd genom att kontrollera sluttillståndet. Ett bättre sätt att kontrollera grindarnas funktionalitet hade varit med hjälp av så kallad kvanttillståndstomografi (eng. *quantum state tomography*). Det innebär att sluttillståndet mäts för många olika initialtillstånd i flera olika baser och på så sätt fås ett bättre mått på om grinden fungerar eller inte.

För att kontrollera QnAS användbarhet var ett mål med programmet att kunna efterlikna experimentella resultat. Detta är något som gjordes framgångsrikt i avsnitt 4.1. QnAS hittade minima vid ungefär rätt vinklar för varje problem med en genomsnittlig avvikelse på 0,03 radianer. Minimivärdena från simuleringen avviker dock desto mer och uppvisar konsekvent ett högre värde än i experimentet. Störst avvikelse uppstår i a)-problemet vilket också är det svåraste. En anledning till avvikelsen i de andra problemen är att kostnadsfunktionen går under -1 (utanför sin värdemängd $F \in [-1,1]$) i den experimentella datamängden, vilket den aldrig gör i simuleringen.

Anledningen till det är förmodligen statistiska störningar på den experimentella datamängden från att den genomgått feldämpning (eng. *error mitigation*) för att förhindra fel vid förberedelse och mätning av tillstånd. De experimentella fluktuationerna kan då leda till att väntevärdena överestimeras och därmed att F går utanför sin värdemängd. Eftersom den experimentella datamängden avviker mycket från kostnadsfunktionens värdemängd och därmed har en relativt stor fel-

marginal anses en genomsnittlig avvikelse i QnAS på under 10% vara godkänd. Sammantaget tyder resultatet på att en simulering i QnAS skulle kunna ligga till grund för beslut kring huruvida en algoritm ska implementeras på en riktig kvantdator. För att kunna dra en mer välgrundad slutsats borde dock fler experiment simuleras och jämföras numeriskt.

En funktion av QnAS som eftersträvades var att ha möjligheten att spara tidsutvecklingen hos ett system för någon av användaren specificerad väntevärdesoperator. Den här funktionen skulle kunna vara till hjälp vid felsökning av experimentella resultat och är helt implementerad i programmet, men har utrymme för förbättringar. I nuläget kan endast en- och tvåkvantbitoperatorer tillämpas som väntevärdesoperatorer med det vanliga inmatningsformatet. Mer generella operatorer kan dock matas in till QnAS ifall användaren själv ser till att väntevärdesoperatorn har korrekta dimensioner. Programmet skulle kunna generaliseras till att kunna använda vilken typ av operator som helst för att spara tidsutvecklingen även med det vanliga inmatningsformatet.

För att göra QnAS användarvänligt sattes krav på hur och vilka parametrar en användare behöver specificera och hur programmet bör kunna användas. Detta mål anses vara uppnått, då lösningen beskriven i avsnitt 3.1.1 ihop med tillgängligheten av QnAS, se avsnitt 4, är relativt användarvänligt. Det finns dessutom en användarmanual med instruktioner och exempel för paketet på pip vilket underlättar för en användare. I framtiden skulle enhetstester (eng. *unit testing*) kunna implementeras. Detta skulle förenkla vidareutveckling då programfel kan förebyggas.

Funktioner för att göra QnAS mer användarvänligt är svårt att konkretisera, eftersom vad som anses användarvänligt är subjektivt. För att få en bättre insikt i fortsatta utvecklingsmöjligheter hade en användarundersökning kunnat utföras där olika personer, förslagsvis medarbetare på WACQT med goda ämneskunskaper, hade fått testa QnAS och lämna omdömen. En sådan användarundersökning har inte genomförts under utvecklingen av QnAS med anledning av tidsbrist, då det inte var planerat för i projektet.

Ett viktigt krav på ett program som QnAS är att simuleringar kan utföras inom en rimlig tidsram. Det är omöjligt att undvika minst en exponentiell tidskomplexitet då antalet kvantbitar ökar. Idealt hade dock en tidskomplexitet $\propto 3^N$ erhållits för trenivåkvantbitar istället för de som presenteras i tabell 2. Hur körningstiden skalas mot antalet kvantbitar är starkt kopplat till datorns hårdvaruspecifikationer, mer specifikt arbetsminnet. Detta åskådliggörs i tabell 2 där parametrarna för kurvanpassningen varierar kraftigt beroende på antalet kvantbitar i den största simulerade kretsen i mätserien. Anledningen tros vara att vid för stort antal kvantbitar nås gränsen för vad datorns arbetsminne klarar av och simuleringstiden ökar kraftigt. En annan anledning till diskrepansen från $\propto 3^N$ är de kringliggande processer som görs i QnAS för att uppnå de övriga kraven på programmet. Här finns det eventuellt en förbättringspotential som skulle kunna undersökas i en vi-

dareutveckling av programmet. Att körtiden växer linjärt med antalet trajektorior och djup i kvantalgoritmen är en godkänd tidskomplexitet som inte har mycket rum för förbättring.

5.2 Programutveckling

För att uppnå kravet om en fysikalisk implementering av *VPZ*-grinden krävs det att QnAS kan hantera virtuella grindar, vilket löstes genom att dela upp kvantalgoritmen i steg. I QuTiP:s funktion `mcsolve` kan en operator inte appliceras momentant, det vill säga genom matrismultiplikation. Uppdelningen av kvantalgoritmen möjliggör matrismultiplikationer på tillstånden var som helst i kvantalgoritmen. Metoden att bryta `mcsolve` efter varje algoritmsteg innebar dock att den inbyggda parallellbearbetningen i `mcsolve` inte kunde användas.

Parallellbearbetningen av `mcsolve` är viktig ur ett tekniskt perspektiv men framför allt ur ett prestandaperspektiv. I QuTiP:s kod kan `mcsolve` bara ta ett initialtillstånd och ger tillbaka N_{traj} sluttillstånd. Det skulle innebära problem om kvantalgoritmen delas upp och programmet för varje steg skulle kört en `mcsolve` med N_{traj} trajektorior. Efter första steget skulle ett initialtillstånd utvecklas till N_{traj} sluttillstånd, vilka hade använts som initialtillstånd nästa steg och därmed utvecklats till N_{traj}^2 sluttillstånd. Problemen ligger alltså dels i att antalet tillstånd hade utvecklats till N_{traj}^n stycken, där n är djupet på kvantalgoritmen, men också i att `mcsolve` endast kan ta ett initialtillstånd som inmatning. Genom att vända på N_{traj} och initialtillstånden, och alltså köra N_{traj} stycken `mcsolve` med en trajektoria var, hålls antalet tillstånd konstant till N_{traj}. För att ändå kunna bearbeta trajektoriorna parallellt användes istället QuTiP:s parallellbearbetningsverktyg `parfor` utanför `mcsolve`. Utan denna hade QnAS variant av `mcsolve` blivit flera gånger långsammare (förutsatt att datorn inte har endast en kärna).

5.2.1 Tidstestning av olika lösningar

Vid den fysikaliska verifieringen uppstod oro över programmets snabbhet och därmed dess förmåga att möta kravet om att programmet ska vara snabbt och effektivt att använda. För att utreda om en alternativ uppbyggnad av programmet skulle vara snabbare utvecklades en alternativ lösning där hela algoritmen kördes i en stor `mcsolve`, istället för att dela upp den i en `mcsolve` per algoritmsteg. De båda varianterna testades utförligt med en rad olika tester, i vilka samma simulering genomfördes med de två olika metoderna på samma hårdvara och simuleringstiden jämfördes.

Resultatet från simuleringarna presenteras i tabell 5 i appendix D och visar att metoden med en stor `mcsolve` var något snabbare för mindre kretsar, men betydligt långsammare för mer avancerade kretsar med flera kvantbitar eller många steg. Ursprungslösningen har dessutom en mer fysikaliskt korrekt implementering av *VPZ*, som där kan appliceras utan drivpuls. I lösningen med en stor `mcsolve`

skulle den istället behöva modelleras med en väldigt kort och hög puls för att nå rätt vinkel under en väldigt kort tid, se ekvation (17) där arean under pulsen är proportionerlig mot vinkeln. När *VPZ* modellerades på det här sättet, med en 1 ns lång puls, ökade dessutom simuleringstiden kraftigt då högre tidsupplösning krävdes för att inte `mcsolve` skulle missa den korta pulsen. Effekten av det här återges i första testet i tabell 5.

Eftersom programmet ska vara snabbt även för avancerade kretsar och måste ha en välfungerande implementering av *VPZ* beslutades det att behålla ursprungslösningen. En potentiell vidareutveckling vore dock att undersöka effekten av att dela in algoritmen i något större steg än de som görs i den nuvarande versionen av QnAS, för att se om det skulle påverka hastigheten. Exempelvis vore det möjligt att bara bryta `mcsolve` vid de steg som innehåller virtuella grindar. Effekten av detta skulle dock troligtvis vara liten, då en stor `mcsolve` inte var mycket snabbare än ursprungslösningen ens för mindre kretsar. En annan möjlighet vore att skriva om själva `mcsolve`-funktionen till att acceptera N_{traj} initialtillstånd och returnera lika många sluttillstånd. Eftersom QnAS använder sig av samma `parfor`-funktion som `mcsolve` borde detta dock inte göra någon större skillnad mot den nuvarande lösningen. Ytterligare en möjlighet vore att undersöka ifall det finns andra bättre parallellbearbetare som skulle kunna snabba på beräkningarna. Eventuellt skulle även vissa delar av programmet kunna skrivas i snabbare språk såsom C.

5.2.2 Pulsfunktion

Pulsfunktionen som används i QnAS, se ekvation (19), valdes då den är lättintegrerad och därmed enkel att använda för att beräkna drivtider för specifika vinklar. I verkligheten används dock så kallade DRAG-pulser (*Derivative Removal by Adiabatic Gate*) [13], vilka ger en mer fyrkantig pulsfunktion. I brist på tid implementerades inte DRAG-pulser i QnAS men det är en relativt enkel vidareutveckling som kan göras genom att ändra pulsfunktionen i `envelopeFunction.py`.

6 Slutsatser

I det här projektet skapades programmet QnAS i syfte att kunna simulera kvantalgoritmer med olika typer av störningar. Målet var att kunna simulera upp mot 15 kvantbitar på en klassisk dator. QnAS bedöms kunna vara till hjälp i situationer både där kvantalgoritmers genomförbarhet ska uppskattas och där felaktiga experimentella resultat ska undersökas. Det skulle också kunna ge en inblick i vilka hårdvaruparametrar som behöver ändras för ett bättre resultat. I avsnitt 3.2.1 framkom att programmet kan simulera kvantalgoritmer med en mindre felmarginal.

Även om kvantalgoritmer med upp mot 15 kvantbitar inte går att simulera på en vanlig dator inom en rimlig tid med QnAS är det fullt möjligt att göra detta,

även med fler än 15 kvantbitar, på en större server eller datorkluster. Användare av programmet på WACQT eller i andra forskningsprojekt har med stor sannolikhet tillgång till sådana resurser. Det finns dessutom andra situationer där det finns ett användarvärde även utan tillgång till kraftfullare datorer. Exempelvis för kvantalgoritmer med upp till 13 kvantbitar, samt för kvantalgoritmer som kan generaliseras för färre antal kvantbitar, kan QnAS vara användbart.

För att vara till nytta för WACQT bedöms DRAG-pulser och $iSWAP$-grinden vara de viktigaste vidareutvecklingarna. Fler störningstyper hade också kunnat införas. De här vidareutvecklingarna bedöms vara relativt enkla att implementera på grund av bra struktur och dokumentation av QnAS. För att på ett rigoröst sätt felsöka programmet hade fler numeriska jämförelser mellan experiment och simulering kunnat studeras. Kvanttillståndstomografi av de olika grindarna skulle också vara ett bra steg för att verifiera programmet. Ytterliggare en förbättring vore att implementera enhetstester som kan hitta eventuella existerande programfel samt undvika att nya programfel uppstår i takt med att programmet vidareutvecklas.

En annan potentiell förbättring av QnAS är minskad tidskomplexitet för parametrar som antal kvantbitar, trajektorior och kretsdjup, vilket skulle öka användbarheten. Det bedöms dock vara ett större jobb att undersöka om betydande förbättring är möjlig då tidskomplexiteten redan är relativt nära de ideala fallen.

Sammanfattningsvis verkar QnAS vara ett bra verktyg med utvecklingspotential som förhoppningsvis kommer att användas i WACQT:s fortsatta utveckling av kvantdatorer.

Referenser

[1] Fein R. Follow the Money... the Quantum Computing Goldrush; 2021. [citerad 2022-01-25]. Hämtad från: https://medium.com/@russfein/follow-the-money-the-quantum-computing-goldrush-407a47f261fc.

[2] WACQT | Chalmers; 2021. [citerad 2022-01-25]. Hämtad från: https://www.chalmers.se/en/centres/wacqt/Pages/default.aspx.

[3] Ferrini G, Frisk Kockum A, Vikstål P. Quantum Computing - Lecture Notes. Chalmers University of Technology; 2021. Hämtad från: https://www.chalmers.se/en/centres/wacqt/graduate%20school/aqa/Pages/default.aspx.

[4] Arute F, Arya K, Babbush R, Bacon D, Bardin JC, Barends R, et al. Quantum supremacy using a programmable superconducting processor. Nature. 2019 10;574:505-10. Hämtad från: https://www.nature.com/articles/s41586-019-1666-5.

[5] Gibney E. Hello quantum world! Google publishes landmark quantum supremacy claim. Nature. 23 okt 2019. [citerad 2022-05-10]. Hämtad från: https://www.nature.com/articles/d41586-019-03213-z.

[6] Deutsch D. Quantum Computational Networks. Proceedings of the Royal Society of London Series A, Mathematical and Physical Sciences. 1989;425(1868):73-90. Hämtad från: http://www.jstor.org/stable/2398494.

[7] Feynman RP. Simulating physics with computers. International Journal of Theoretical Physics. 1982:467–488. Hämtad från: https://doi.org/10.1007/BF02650179.

[8] Preskill J. Quantum Computing in the NISQ era and beyond. Quantum. 2018 8;2:79. Hämtad från: https://doi.org/10.48550/arXiv.1801.00862.

[9] Shor PW. Algorithms for quantum computation: discrete logarithms and factoring. I: Proceedings 35th Annual Symposium on Foundations of Computer Science; 1994. s. 124-34. Hämtad från: https://ieeexplore.ieee.org/document/365700.

[10] Cerezo M, Arrasmith A, Babbush R, et al. Variational quantum algorithms. Nature Reviews Physics. 2021;3:625-44. Hämtad från: https://doi.org/10.1038/s42254-021-00348-9.

[11] Nielsen MA, Chuang IL. Quantum Computation and Quantum Information (10th Anniversary edition). Cambridge University Press; 2000.

[12] Brooks M. Beyond quantum supremacy: the hunt for useful quantum computers. Nature. 2019;574:19-21. Hämtad från: https://doi.org/10.1038/d41586-019-02936-3.

[13] Bengtsson A, Vikstål P, Warren C, Svensson M, Gu X, Kockum AF, et al. Improved Success Probability with Greater Circuit Depth for the Quantum Approximate Optimization Algorithm. Physical Review Applied. 2020 Sep;14:034010. Hämtad från: https://link.aps.org/doi/10.1103/PhysRevApplied.14.034010.

[14] Johansson JR, Nation PD, Nori F. QuTiP 2: A Python framework for the dynamics of open quantum systems. Computer Physics Communications. 2013 4;184:1234-40. Hämtad från: https://doi.org/10.1016/j.cpc.2012.11.019.

[15] Qiskit org [internet]. Open-Source Quantum Development. Armonk, NY: IBM Research; 2020. [citerad 2022-05-04]. Hämtad från: https://qiskit.org/.

[16] Ganzhorn M, Salis G, Egger DJ, Fuhrer A, Mergenthaler M, Müller C, et al. Benchmarking the noise sensitivity of different parametric two-qubit gates in a single superconducting quantum computing platform. Physical Review Research. 2020 Sep;2:033447. Hämtad från: https://link.aps.org/doi/10.1103/PhysRevResearch.2.033447.

[17] Kosen S, Li HX, Rommel M, Shiri D, Warren C, Grönberg L, et al.. Building Blocks of a Flip-Chip Integrated Superconducting Quantum Processor; 2021. Hämtad från: https://arxiv.org/abs/2112.02717.

[18] Krantz P, Kjaergaard M, Yan F, Orlando TP, Gustavsson S, Oliver WD. A quantum engineer's guide to superconducting qubits. Applied Physics Reviews. 2019 6;6:021318. Hämtad från: https://doi.org/10.1063/1.5089550.

[19] Griffiths DJ, Schroeter DF. Introduction to Quantum Mechanics. 3:e Uppl. Cambridge University Press; 2004.

[20] Ladd TD, Jelezko F, Laflamme R, Nakamura Y, Monroe C, O'Brien JL. Quantum computers. Nature. 2010;464(7285):45-53. Hämtad från: https://www.nature.com/articles/nature08812.

[21] Daley AJ. Quantum trajectories and open many-body quantum systems. Advances in Physics. 2014;63. Hämtad från: https://doi.org/10.1080/00018732.2014.933502.

[22] Frisk Kockum A. Quantum optics with artifcial atoms [Doktorsavhandling]. Chalmers University of Technology; 2014. Hämtad från: https://publications.lib.chalmers.se/records/fulltext/206197/206197.pdf.

[23] Verlinde H. PHY305: Notes on Entanglement and the Density Matrix;. Hämtad från: https://wwwphy.princeton.edu//~verlinde/PHY305/density2.pdf.

[24] Lindblad G. On the generators of quantum dynamical semigroups. Communications in Mathematical Physics. 1976 6;48:119-30. Hämtad från: https://doi.org/10.1007/BF01608499.

[25] Fors SP. The static ZZ coupling in superconducting qubits [Masteruppsats]. Chalmers University of Technology; 2022. Hämtad från: https://hdl.handle.net/20.500.12380/304540.

[26] Quantiki - Quantum Information Portal and Wiki [internet]. Tensor product. Bratislava: Research Center for Quantum Information, Institute of Physics, Slovak Academy of Sciences; 2015. [citerad 2022-04-18]. Hämtad från: https://www.quantiki.org/wiki/tensor-product.

[27] Qiskit org [internet]. Qubit Drive and the Rotating Wave Approximation. Armonk, NY: IBM Research; 2021. [citerad 2022-04-18]. Hämtad från: https://qiskit.org/textbook/ch-quantum-hardware/transmon-physics.html#6.-Qubit-Drive-and-the-Rotating-Wave-Approximation-.

A Programspecifikation

A.1 Kravspecifikation

Programmet ska:

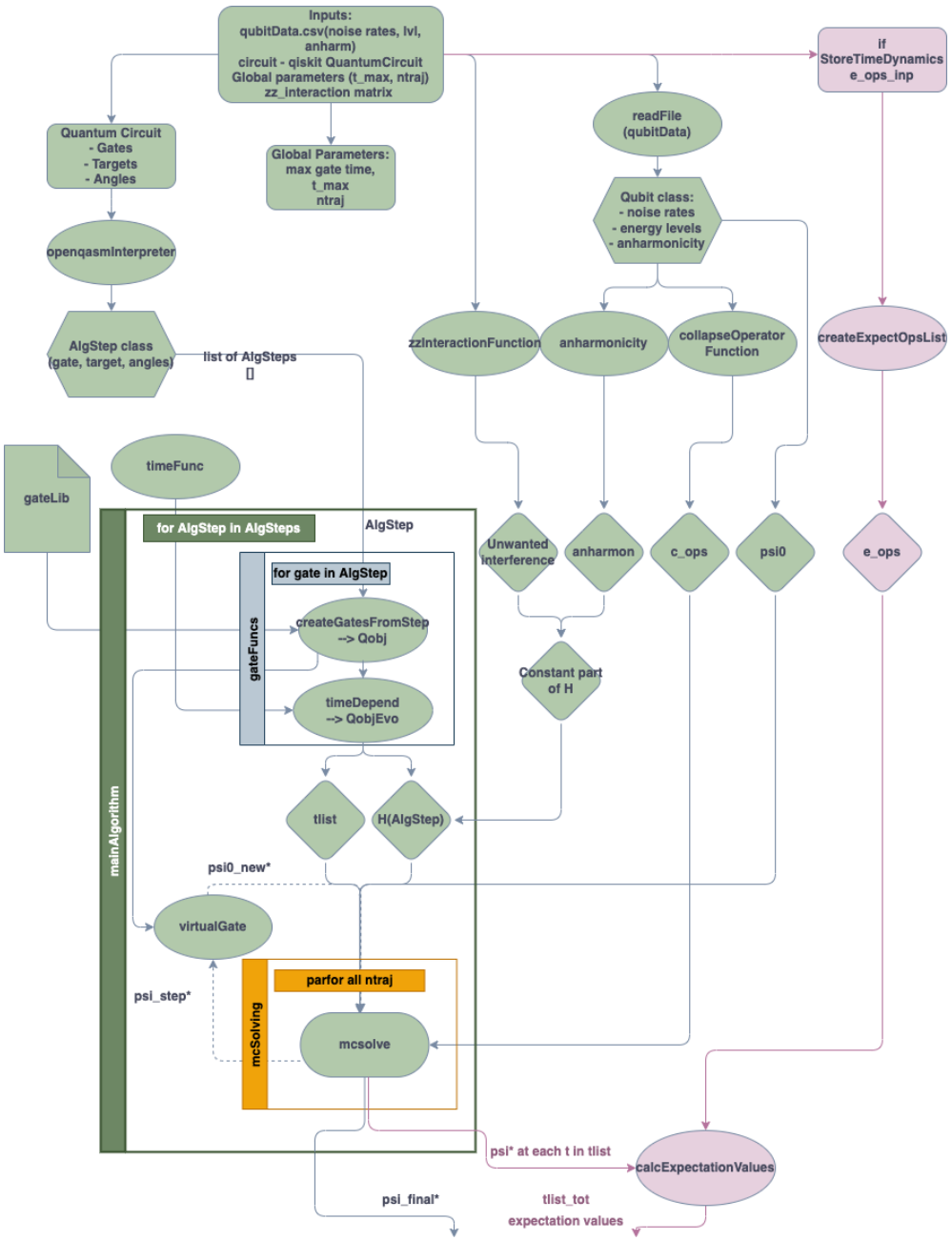
1. Kunna simulera kvantalgoritmer med upp till 15 kvantbitar, vilka ska kunna ha två, tre eller fyra energinivåer och påverkas av störningarna relaxation, excitation, urfasning och oönskad interferens.
2. Kunna simulera de kvantgrindar som används av WACQT, se definitioner i A.2, på ett sätt som efterliknar den fysikaliska implementeringen i WACQT:s kvantkretsar.
3. Kunna efterlikna experimentell data på ett sätt som kan vara till underlag för beslut om kvantalgoritmen ska implementeras på en riktig kvantdator.
4. Kunna återge tidsutvecklingen under algoritmen för ett antal väntevärdesoperatorer speciferade av användaren.
5. Hantera inmatning i `qnas.solve()` på formen:
   - `Qbfile` - .csv-fil med hårdvaruparametrar för varje kvantbit:
     - Relaxationstakt, Γ_{relax} [MHz].
     - Urfasningstakt, Γ_{urfas} [MHz].
     - Excitationstakt, Γ_{excit} [MHz].
     - Anharmonicitet, U [MHz].
     - Antal energinivåer som ska simuleras.
   - `circuit` - `QuantumCircuit`-objekt från Qiskit som specificerar kretsen.
   - `zz_int` - Matris som specifierar ZZ-kopplingen mellan kvantbitarna [Hz].
   - `ntraj` - Antalet trajektorior N_{traj}.
   - `tmax` - Maximala driftstider för en- och tvåkvantbitsgrindar [s].
   - `store_time_dynamics` - Boolesk variabel som anger om tidsutvecklingen ska sparas.
   - `e_ops` - Matris som håller `Qobj` och måltavla för de väntevärden användaren är intresserad av.
6. Ha en felhantering som ger feedback om vad som gått fel och kontrollerar alla parametrar i inmatningen innan simuleringen börjar.
7. Kunna utföra simuleringar inom en rimlig tid på en ordinär dator.
8. Vara lätt att installera och användarvänligt för den tänkta användaren (en medarbetare på WACQT med goda kunskaper om kvantalgoritmer) utifrån användarmanualen i pip.

A.2 Grinddefinitioner

Tabell 3: Definition av de av WACQT:s grindar som går att använda i QnAS, där $\hat{a}$ och $\hat{a}^\dagger$ är förintelse- respektive skapelseoperatorerna.

Grind	Namn i QnAS	2-nivå definition	Definition i QnAS (generell nivå)
Pauli-X, σ_x	*PX*	$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$\hat{a} + \hat{a}^\dagger$
Pauli-Y, σ_y	*PY*	$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$	$i(\hat{a}^\dagger - \hat{a})$
Virtuell Pauli-Z	*VPZ*	$\begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{bmatrix}$	$\begin{bmatrix} e^{-i\frac{\theta}{2}} & 0 & 0 & \dots \\ 0 & e^{i\frac{\theta}{2}} & 0 & \\ 0 & 0 & 1 & \\ \vdots & & & \ddots \end{bmatrix}$
Hadamard	*HD*	$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$	$VPZ_{(\pi)}PY_{(-\frac{\pi}{2})}$
Kontroll-Z	*CZ*	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$	$\|11\rangle\langle 02\| + \|02\rangle\langle 11\|$

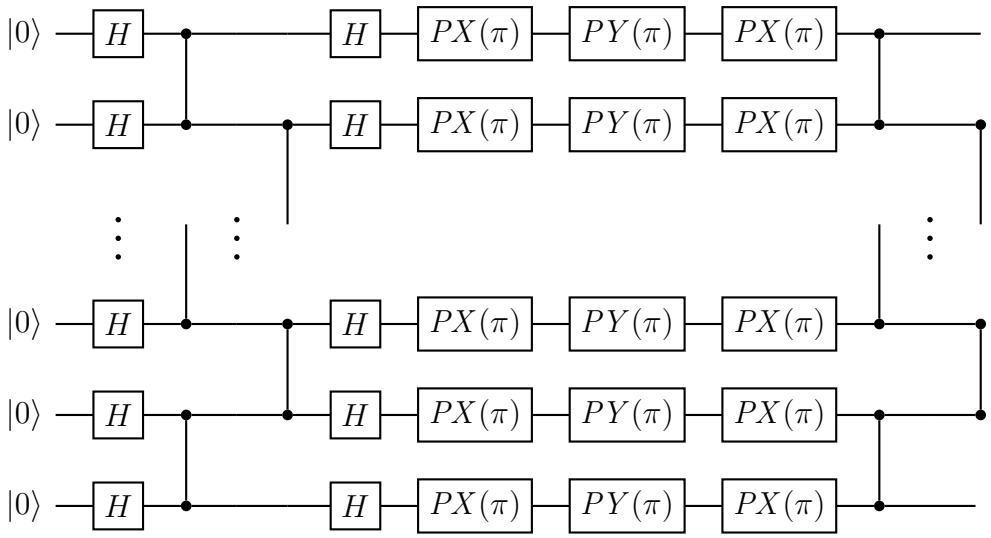
B Övergripande programstruktur



Figur 10: Övergripande programstruktur som visar hur inmatningen hanteras och hur programmets funktioner samverkar. Funktioner representeras av ovaler, objektsklasser av hexagoner, bibliotek av pappersymbolen och diamanterna representerar de steg där indata har omvandlats färdigt till det som används av programmets kärna `mcsolve`. Huvudprogrammet är de gröna delarna medan de rosa visar de extra processer som genomförs om användaren valt att spara tidsutvecklingen. De transparenta fyrkanterna är de olika iterationerna som görs. Ytterst finns `mainAlgorithm` som går igenom alla `AlgSteps`, vari `gateFuncs` skapar Hamiltonianen för varje `AlgStep` och `mcSolving` utför den parallella iterationen av de N_{traj} trajektoriorna. Symbolen * används för att visa att det är N_{traj} tillstånd.

C Prestandatestning

C.1 Krets för prestandatestning av antal kvantbitar



Figur 11: Krets använd för att testa programmets prestanda vid varierande antal kvantbitar.

C.2 Datorspecifikation

Tabell 4: Hårdvaruspecifikationer för de datorer som användes vid prestandatestningen. Liknande specifikationer är vad som avses med ”vanlig dator”.

Tiddsskalning mot:	RAM	Antal Kärnor	Processor
Antal kvantbitar, figur 7	8 GB	2	Intel Core i5 3,1 GHz
Algoritmens djup, figur 8	16 GB	8	AMD Ryzen 7 3700U with Radeon Vega Mobile Gfx 2.30 GHz

D Resultat från tidstestning av olika lösningar

Tabell 5: Urval av resultat från tidstestningen av de olika metoderna. Upplösningen som specificeras är antalet vinklar som testades i avsnitt 4.1. De algoritmsteg som användes byggde på de ifrån figur 3, men eftersom *VPZ* ej var implementerat i fallet "En `mcsolve`" byttes steg som innehöll *HD* ut med motsvarande grindar ($PX_{\pi/2}$) och steg med *VPZ* togs bort. I de fall märkta med * användes dock *HD* och *VPZ* i de metoder det var implementerat, varför antalet steg är 7-8 (*8 med *VPZ*). I fallen där fler än två kvantbitar användes modifierades stegen ytterligare genom att även lägga på grindar som verkade på de extra kvantbitarna. När fler än 7 steg användes återupprepades de 5 stegen med konstanta vinklar (π) från figur 3.

	Testspecifikation						
N_{traj}:	100	100	2000	100	500		100
Upplösning:	20×20	30×30	20×20	20×20	1×1		1×1
Antal kvantbitar:	2	2	2	4	6		2
Antal algoritmsteg:	7-8*	7	7-8*	7	20	50	200
	Simuleringstid [s]						
Ursprungslösning	1250	2826	20999	3683	798	1871	94
En `mcsolve`	905	2259	18247	4234	3928	24907	310
... med 1 ns *VPZ*	6650	-	-	-	-	-	-

Institutionen för mikroteknologi och nanovetenskap
Chalmers tekniska högskola
Göteborg, Sverige
www.chalmers.se

CHALMERS