Time Simulation of an Anisotropic Heisenberg Chain in a Superconducting Quantum Processor

Undergraduate Thesis

Diego Alejandro Herrera Rojas

Submitted in partial fulfillment of the requirements for the
Degree of Bachelor of Science in Physics
from Universidad Nacional de Colombia.

Physics Department
Universidad Nacional de Colombia
Bogotá D. C., Colombia
2022

Contents

Abstract

A digital quantum algorithm for simulating time evolution of a completely anisotropic Heisenberg Hamiltonian on an arbitrary lattice is proposed. This consists on a second order Trotter scheme that profits commutation properties of spin-spin interaction. The implementation is tailored for publicly available quantum processors provided by **IBM Quantum Experience**. Recent techniques for implementing high fidelity cross resonance gates are used to reduce execution time on real quantum backends, thus taking more advantage of the finite coherence time of current quantum devices. The proposed algorithm is compared to direct implementations based on direct evolution of Pauli components of the Hamiltonian, and basis sets used by IBM Quantum backends. This work also discusses the limitations of Trotterization error using QASM simulators, and the inherent errors of noisy hardware implementation.

Acknowledgements

The author wishes to thank Professor Karen. M. Fonseca R. for her advice and assistance. Special thanks to the author's parents for their support, as well as Cristian Galvis for his openness to discussion and debate on quantum computing and Qiskit SDK. He would also like to thank IBM Quantum for providing access to pulse enabled backends through the Open Science Prize 2021 Challenge.

Chapter 1

General Introduction

Quantum Computing has become a reality now that the field is entering the so called NISQ (Noisy Intermediate Scale Quantum) era. The digital quantum circuit model is one of the most extended and discussed models of quantum computing. It is based on the concept of qubit, which is an abstract two-level quantum system. By profiting from linear superposition and entanglement of many-qubit systems, quantum algorithms are thought to be more resource-efficient than standard classical algorithms in areas like machine learning and mathematical finance. The present work studies a more fundamental, yet quite versatile, application of quantum computation: simulation of quantum physical systems. This perspective was introduced in 1982 by Richard Feynman [9]. He suggested that using one quantum system to simulate another can reduce the exponential overhead that occurs by incrementing the number of components. In fact, Lloyd [14] has proved that this is the case, and Trotter integration schemes can be used to efficiently simulate quantum systems that can be modeled by local interactions on any number of component.

In particular, this work is concerned with the task of devising a quantum time simulator capable of performing evolution of parametric Heisenberg anisotropic models. A digital quantum algorithm for such tasks, based on Trotter decomposition, is devised using Qiskit Software Development Kit (SDK). At the moment of presentation of this dissertation, Las Heras et. al. [10, 3] and Y. Salathé et. al. [20] have proposed simple digital quantum algorithms for simulating time evolution of small Heisenberg systems. The includes employed by those groups, however, where specifically optimized for superconducting chips whose architecture is not publicly available.

Furthermore, collaborations with IBM Quantum research teams [23, 8, 22] have produced novel techniques for implementing two-qubit gates using efficient microwave pulse schedules based on the cross resonance interaction. This control technique is also used to simulate time evolution, for instance, in the context of Quantum Approximate Optimization Algorithm (QAOA) algorithms [8]. This work extends those insights and adapts them to the architecture available publicly by IBM Quantum through Qiskit SDK. Using commutation properties of local Hamiltonians, and direct pulse control via cross resonance gates, evolution of anisotropic Heisenberg Hamiltonians is simulated on quantum devices.

This dissertation is structured as follows. After a review of the elements of digital quantum time simulation, three simulation algorithms are discussed and compared in terms of probability density fidelity, state fidelity, and measurement of common observables. These correspond to direct trotterization using basis gates, trotterization using commutation properties, and trotterization with direct pulse control. A three-spin Heisenberg Hamiltonian is used as a benchmark model; however, simulation is readily extensible to more complex interactions. Since the three algorithms use the same integration scheme, a discussion of the inherent discretization error is performed using Quantum Assembly (QASM) simulators. Experimental results are obtained using *IBMQ Jakarta* backend. Metrics on fidelity and measured observables mentioned before are finally compared and the utility of cross resonance pulses is made explicit.

Chapter 2

Elements of Digital Quantum Simulation

This chapter introduces the elements of quantum computing and digital time simulation. First, the qubit as a two-level system and a quantum register as a *multi-qubit* system are presented. After a review of the elements of quantum computing, from single qubit operations to universal quantum computing, digital integration schemes for solving Schrödinger equation are presented. With this background, previous work on simulation of spin and fermionic systems is introduced. Finally, recent techniques for implementing hardware efficient two qubit gates are introduced. The present chapter is expected to give a broad overview of the concepts and techniques used in subsequent chapters for implementing a quantum time evolution algorithm for anisotropic Heisenberg-like Hamiltonians.

2.1 Quantum circuit model elements

Nowadays, it is familiar to understand information processing tasks in terms of *bits*. A bit is a binary variable that can be said to have either of two states on a set (commonly denoted by 0, 1). As a result, a bit is nothing more than a variable s whose value is in the set $\{0,1\}$. In modern computers, such an entity can be encoded physically by means of electric current or voltage. Most operations that can act upon a bit are mappings from one of the possible state values to the other. These corresponds to two fundamental operations: the NOT gate, which flips the bit value, or the IDENTITY gate, whose action is actually inaction.

In contrast, a quantum bit or *qubit*, corresponds physically to a two-level quantum system, whose Hilbert space can be spanned by a *computational basis set* $\{|0\rangle, |1\rangle\}$. It is reasonable to assume that this basis set is orthonormal

$$\langle s|s'\rangle = \delta_{ss'} \text{ for } s,s' \in \{0,1\}, \tag{2.1}$$

The canonical representation of a qubit is as a vector in the *Bloch sphere*. This representation maps a generic qubit superposition state

$$\cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle, \tag{2.2}$$

$$\theta,\phi \in \mathcal{R} \tag{2.3}$$

to a vector on the unitary 3-sphere

$$\hat{\mathbf{n}} = \sin(\theta)\cos(\phi)\hat{\mathbf{x}} + \sin(\theta)\sin(\phi)\hat{\mathbf{y}} + \cos(\theta)\hat{\mathbf{z}}. \tag{2.4}$$

In addition to the computational basis, other important states are the *sign basis*

$$|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, \tag{2.5}$$

$$|-\rangle = \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle, \tag{2.6}$$

and the *y basis*

$$|+\mathrm{i}\rangle = \frac{1}{\sqrt{2}}|0\rangle + \mathrm{i}\frac{1}{\sqrt{2}}|1\rangle, \tag{2.7}$$

$$|-\mathrm{i}\rangle = \frac{1}{\sqrt{2}}|0\rangle - \mathrm{i}\frac{1}{\sqrt{2}}|1\rangle. \tag{2.8}$$

Those states actually lie in the poles of the Bloch sphere, and each pair constitutes a basis for a qubit's Hilbert space that has interesting properties regarding measurement.

In contrast to classical bits, a quantum bit can be transformed by an infinite number of operations, corresponding to all possible operators defined on its Hilbert space. Of those, two classes of operators are of huge importance in quantum computing: unitary operators and measurement operators. Unitary operations are used mainly to process quantum information and perform a computation. These are commonly known as *quantum gates*. The fundamental quantum gates are the Pauli operators

$$\hat{X} = |+\rangle\langle+| - |-\rangle\langle-|, \tag{2.9}$$

$$\hat{Y} = |+\mathrm{i}\rangle\langle+\mathrm{i}| - |-\mathrm{i}\rangle\langle-\mathrm{i}|, \tag{2.10}$$

$$\hat{Z} = |0\rangle\langle0| - |1\rangle\langle1|. \tag{2.11}$$

For a single qubit, it is known that any unitary operation can be represented as rotation operator of the form

$$\hat{R}_{\hat{n},\theta} = \cos\left(\frac{\theta}{2}\right) - \mathrm{i}\sin\left(\frac{\theta}{2}\right)\hat{n}\cdot\sigma, \tag{2.12}$$

$$\hat{n}\cdot\sigma = n_x\hat{X} + n_y\hat{Y} + n_z\hat{Z}, \tag{2.13}$$

$$||\hat{n}||^2 = 1. \tag{2.14}$$

In the Bloch sphere representation, quantum gates, therefore, correspond to rotations of the quantum state's associated vector (eq. 2.4). A quite important rotation is the so called *Hadamard gate*, whose representation in the computational basis is

$$\hat{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.15}$$

These rotations allow computations, but the actual process of reading the outcome of an algorithm implies measurement. In current digital quantum computing implementations, measurement operators are projectors onto the computational basis

$$\hat{P}_0 = |0\rangle\langle0|, \tag{2.16}$$

$$\hat{P}_1 = |1\rangle\langle1|. \tag{2.17}$$

These operators allow measurement of the expected value of $\hat{Z}$ operator. By performing rotations to the sign and y basis, it is possible to measure expected values of $\hat{X}$ and $\hat{Y}$, respectively. Since Pauli operators span the space of qubit operators, any qubit observable can be measured by applying suitable rotations and forming linear combinations of expected values of Pauli operators.

2.1.1 Quantum registers and multi-qubit gates

In general, more than one qubit is needed to perform meaningful computations. A system of several qubits is called *quantum register*. A quantum register's Hilbert space is nothing more than the tensor product space of each of its constituent qubits. Thus, a N-qubit register has a 2^N-dimensional Hilbert space. A basis for this space is built by all possible tensor products of computational basis states for each qubit. This would be the *computational basis* of the register. A member of this set may be denoted by

$$|s_{N-1}s_{N-2}\cdots s_0\rangle = |s_{N-1}\rangle \otimes |s_{N-2}\rangle \otimes \cdots \otimes |s_0\rangle, \tag{2.18}$$

$$s_{N-1}, s_{N-2}, \ldots, s_0 \in \{0,1\}. \tag{2.19}$$

There may be other basis of interest in quantum computing. For instance, with $N = 2$, the so called *Bell basis*,

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle), \tag{2.20}$$

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle), \tag{2.21}$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|10\rangle + |01\rangle), \tag{2.22}$$

$$|\Psi^=\rangle = \frac{1}{\sqrt{2}}(|10\rangle - |01\rangle), \tag{2.23}$$

is of capital importance in quantum information theory and quantum communications. It will also prove to be important in this work. Much like single-qubit gates, register gates or multi-qubit gates correspond to unitary operators defined on the register's Hilbert space. An important multi-qubit gate, defined by its action on two qubits is CNOT

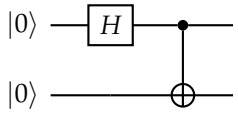
$$\mathrm{CNOT}\,|\psi\rangle_t \otimes |1\rangle_c = \left(\hat{X}\,|\psi\rangle_t\right) \otimes |0\rangle_c, \tag{2.24}$$

$$\mathrm{CNOT}\,|\psi\rangle_t \otimes |0\rangle_c = |\psi\rangle_t \otimes |0\rangle_c, \tag{2.25}$$

where the subscript t denotes the *target* qubit, and the subscript c, the control qubit. Hence, CNOT corresponds to a controlled-$\hat{X}$ operation. This gate can entangle separable two-qubit states. This is of vital importance for universal quantum computing, and can be readily seen from the definition.

2.1.2 Circuit representation

This model of computation can be represented graphically by a *circuit*, in which every wire represents a quantum bit, and a gate corresponds to a unitary operator acting on the register's Hilbert space (possibly a Hilbert subspace). For instance, an algorithm for producing a Bell basis state can be represented as in fig. 2.1



The figure presents a prescription for producing state $|\Phi^+\rangle$. In this circuit representation, the algorithm consists on an application of a Hadamard operator to a control qubit, followed by a CNOT operation, to a two qubit register, initialized on state $|00\rangle$. In general, quantum gates are represented by boxes, labeled properly by the operation that they represent. These boxes cover the subspace over which the corresponding quantum operator acts. For instance, in the case of the algorithm depicted on fig. 2.1, the Hadamard gate acts on a single-qubit subspace, while the CNOT gate acts on the whole two-qubit register's space. Operations are executed from left to right in a quantum algorithm.

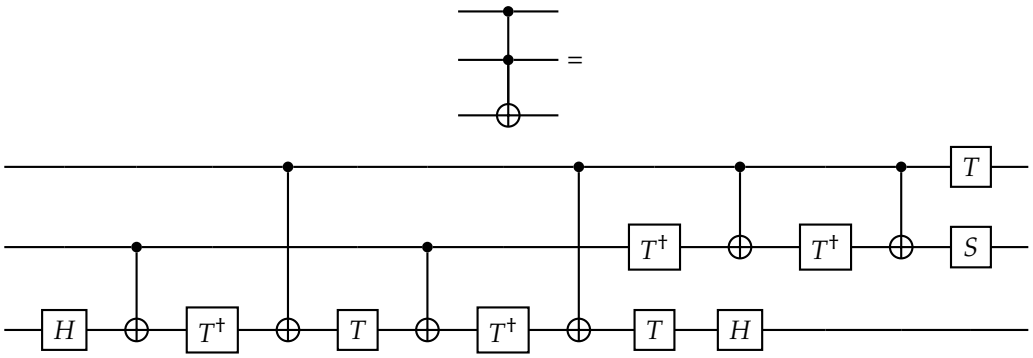
2.1.3 Universal quantum computing

As was stated before, all quantum gates correspond to unitary operations (or measurements), acting on a quantum register. It is desirable to find a set of elementary quantum gates, that act on a small number of qubits, that can generate all possible unitary operations on an N-qubit register. This is the question of universal quantum computing. It can be shown that CNOT and the set of single-qubit unitary operations are enough for producing all possible N-qubit register gates [17]. For instance, it is possible to perform the three-qubit *Controlled CNOT* operation

$$\text{CCNOT}\,|\psi\rangle \otimes |s_t\rangle \otimes |s_t'\rangle = \left(\hat{X}^{s_t s_t'}\,|\psi\rangle\right) \otimes |s_t\rangle \otimes |s_t'\rangle\,, \tag{2.26}$$

$$s_t, s_t' \in \{0,1\}, \tag{2.27}$$

by following algorithm on figure 2.2. In general, however, performing arbitrary single-qubit rotations by hardware operations is impractical. This would require a huge control on the physical qubits that is not yet available for all possible architectures [17]. As a result, most quantum processors available today use a limited set of single-qubit rotations for *approximating* arbitrary rotations.



A commonly used universal set is $\{\hat{H}, \hat{S} = \sqrt{\hat{Z}}, \hat{T} = \sqrt{\hat{S}}\}$. With this set, it is possible to approximate any single qubit rotation to an arbitrary precision, but not exactly, using a finite number of operations. In contrast, IBM Quantum devices use a basis set $\{\hat{S}_x = \sqrt{\hat{X}}, \hat{X}, \hat{R}_{z,\phi}\}$, which can reproduce any single-qubit rotation exactly. As an example, the Hadamard gate can be decomposed as follows

$$\hat{H} = \hat{R}_{z,\pi/2}\hat{S}_x\hat{R}_{z,\pi/2}. \tag{2.28}$$

As may be inferred from the two examples above, emulating arbitrary N-qubit-register operators can cause some computational overhead, that is an increase in the number of elementary operations required to reproduce a quantum computation. Usually, the basis set depends on the architecture of a quantum processor, and thus, due to current limitations of available hardware, it is important to design a quantum algorithm so that the operations involved can be optimally represented by the universal set associated to the device on which it is expected to be implemented.

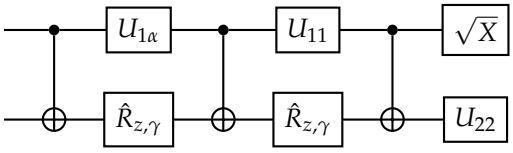
Regarding implementation of two qubit gates, an important advance was made by [25]. This work shows that implementation of unitary gates in the group

$$\hat{U}(\alpha,\beta,\gamma) = \mathrm{e}^{-\mathrm{i}(\alpha\hat{X}\hat{X}+\beta\hat{Y}\hat{Y}+\gamma\hat{Z}\hat{Z})} \tag{2.29}$$

can be performed using only three CNOT gates. The circuit representation of such algorithm can be found on figure 2.3. In that circuit the following single qubit operators are used [8]:

$$\hat{U}_{mn} = \hat{R}_z(m\pi/2)\sqrt{X}\hat{R}_z(n\pi/2), \tag{2.30}$$

$$\hat{U}_{1\alpha} = \hat{R}_z(\pi/2)\sqrt{X}\hat{R}_z(\alpha), \tag{2.31}$$



2.1.4 Quantum computational advantage

It is expected that quantum computing paradigms challenge the strong Church-Turing thesis, which claims that the ultimate reference for computational complexity is the probabilistic Turing Machine Model [17, 5]. In principle, a quantum processor could solve problems that are thought to be hard for probabilistic Turing Machines. Such problems include simulation of chemical systems [16, 6], and multi-particle quantum systems in general [17, 1, 7]. Regarding the quantum circuit model of computation, this usually means that the total gate count of a circuit that implements an algorithm that solves a hard problem is bounded asymptotically by a polynomial function in the size of the input [17, 5]. In general, however, a more accurate measurement of the complexity of a quantum algorithm is the *circuit depth*, which measures the maximum number of operations a qubit has to undergo to complete a computation. One of the goals of this work is to show that it is possible to simulate arbitrarily large spin systems efficiently using quantum digital algorithms, something impossible with the standard methods of computational quantum physics.

2.2 Common Approximation Schemes for Unitary Evolution

Consider a system of N components, whose Hamiltonian can be expressed as a sum of local Hamiltonians (i.e., that model interaction between at most C components) [17, 14]

$$\hat{H} = \sum_{k=1}^{L} \hat{H}_k, \tag{2.32}$$

where L is some polynomial on the number of system components. Such system is said to have local interactions between its constituents. From general quantum theory, its dynamics is governed by the Schrödinger equation

$$\mathrm{i}\frac{\mathrm{d}|\psi\rangle}{\mathrm{d}t} = \hat{H}|\psi\rangle, \tag{2.33}$$

whose solution, for time independent Hamiltonians, is

$$|\psi(t)\rangle = \mathrm{e}^{-\mathrm{i}\hat{H}t}|\psi(0)\rangle. \tag{2.34}$$

In general, $[\hat{H}_i, \hat{H}_j] \neq 0$, and thus

$$\mathrm{e}^{-\mathrm{i}\hat{H}t} \neq \prod_{k=1}^{L} \mathrm{e}^{-\mathrm{i}\hat{H}_k t}. \tag{2.35}$$

Hence, solving Schrödinger equation is a non trivial task. Many systems are described by local interactions; for instance, electrons in a solid material or magnetic moments in a lattice. In several instances, local interaction Hamiltonians are non-commuting, and thus approximation methods are necessary for performing time evolution. In this section, schemes for approximating unitary evolution of a quantum system are discussed.

2.2.1 Trotter Formulas

Consider operators $\hat{H}_1$, $\hat{H}_2$, with $[\hat{H}_1, \hat{H}_2] \neq 0$. By definition

$$e^{-i\hat{H}_1 t} = \sum_{m=0}^{\infty} \frac{(-it)^m}{m!} \hat{H}_1^m, \tag{2.36}$$

$$e^{-i\hat{H}_2 t} = \sum_{l=0}^{\infty} \frac{(-it)^l}{l!} \hat{H}_2^l. \tag{2.37}$$

It is readily shown that

$$e^{-i\hat{H}_1 t} e^{-i\hat{H}_2 t} = \sum_{k=0}^{\infty} \frac{(-it)^k}{k!} \left[\sum_{m=0}^{k} \binom{k}{m} \hat{H}_1^m \hat{H}_2^{k-m} \right]. \tag{2.38}$$

For non-commuting operators, the following identity is true:

$$\sum_{m=0}^{k} \binom{k}{m} \hat{H}_1^m \hat{H}_2^{k-m} = (\hat{H}_1 + \hat{H}_2)^k + f_k(\hat{H}_1, \hat{H}_2), \tag{2.39}$$

where $f_k(\hat{H}_1, \hat{H}_2)$ is a function of the commutator of the operators. Since $f_1(\hat{H}_1, \hat{H}_2) = 0$, it follows that

$$e^{-i\hat{H}_1 t} e^{-i\hat{H}_2 t} = e^{-i(\hat{H}_1 + \hat{H}_2)t} + \mathcal{O}(t^2). \tag{2.40}$$

If $|t| \ll 1$, the product of the exponential operators estimate the evolution operator with an error $\mathcal{O}(t^2)$. In the general case, it must be noted that

$$e^{-i\sum_{k=0}^{L} \hat{H}_k t} = \hat{1} + (-it) \sum_{k=0}^{L} \hat{H}_k + \frac{(-it)^2}{2} \left[\sum_{k=0}^{L} \hat{H}_k^2 + 2 \sum_{jk} \hat{H}_k \hat{H}_j \right] + \mathcal{O}(t^3). \tag{2.41}$$

In consequence, a unitary evolution operator with local interactions may be approximated, to quadratic order, by the exponential product

$$e^{-i\sum_{k=0}^{L} \hat{H}_k t} = \prod_{k=1}^{L} e^{-i\hat{H}_k t} + \mathcal{O}(t^2). \tag{2.42}$$

In some instances, quadratic approximations may be enough. In his seminal paper, Lloyd presents this quadratic approximation for simulation of quantum systems with local interaction [14]. Also, Las Heras et. al. simulate a Hubbard Hamiltonian with up to 4 fermionic modes using second order approximations to unitary evolution [3, 10]. However, in following sections, higher-order approximation schemes are discussed, based upon equation 2.41.

2.2.2 Some Cubic Order Schemes

The first cubic order approximation discussed is the so called Baker-Haussdorf formulae [17]. By series expansion, it can be shown that

$$\begin{aligned} e^{-i\hat{H}_1 t}e^{-i\hat{H}_2 t}e^{-i[\hat{H}_1,\hat{H}_2]t^2} =& \hat{1} + (-it)(\hat{H}_1 + \hat{H}_2) \\ &+ \frac{(-it)^2}{2}(\hat{H}_1^2 + \hat{H}_2^2 + \hat{H}_1\hat{H}_2 + \hat{H}_2\hat{H}_1) + \mathcal{O}(t^3) \\ =& e^{-i(\hat{H}_1+\hat{H}_1)t} + \mathcal{O}(t^3). \end{aligned}$$

Although useful in case of operators that constitute a Lie algebra, the formulae above may not be enough in other instances. A more general approximation formulae is

$$e^{-it\sum_{l=0}^{L-1}\hat{H}_l} = \left(\prod_{l=0}^{L-1} e^{-i\hat{H}_l\frac{t}{2}}\right)\left(\prod_{l=L-1}^{0} e^{-i\hat{H}_l\frac{t}{2}}\right) + \mathcal{O}(t^3). \tag{2.43}$$

This formulae can be deduced directly from the identity

$$e^{-i\sum_{l=0}^{2L-1}\hat{K}_l t} = \hat{1} + (-it)\sum_{l=0}^{2L-1}\hat{K}_l + \frac{(-it)^2}{2}\left[\sum_{l=0}^{2L-1}\hat{K}_l^2 + 2\sum_{jl}\hat{K}_l\hat{K}_j\right] + \mathcal{O}(t^3), \tag{2.44}$$

with the identifications

$$\hat{K}_l = \hat{K}_{2L-1-l} = \frac{\hat{H}_l}{2}, \tag{2.45}$$

and the following observations:

$$\begin{aligned} \sum_{l=0}^{2L-1}\hat{K}_l^2 &= \frac{1}{2}\sum_{l=0}^{L-1}\hat{H}_l^2, \\ 2\sum_{l'l}\hat{K}_l\hat{K}_{l'} &= \frac{1}{2}\sum_{l=0}^{L-1}\hat{H}_l^2 + \sum_{l'l}\left(\hat{H}_l\hat{H}_{l'} + \hat{H}_{l'}\hat{H}_l\right), \\ \left(\sum_{l=0}^{L-1}\hat{H}_l\right)^2 &= \sum_{l=0}^{L-1}\hat{H}_l^2 + \sum_{l'l}\left(\hat{H}_l\hat{H}_{l'} + \hat{H}_{l'}\hat{H}_l\right). \end{aligned} \tag{2.46}$$

2.2.3 Suzuki - Trotter Scheme

Suzuki-Trotter scheme is a higher order approximation to an evolution operator, that works iteratively on top of the approximation given by equation 2.43. Set [24, 7]

$$\hat{S}_2(\lambda) = \prod_{l=0}^{L-1} e^{\frac{\lambda}{2}\hat{H}_l}\prod_{l=L-1}^{0} e^{\frac{\lambda}{2}\hat{H}_l}. \tag{2.47}$$

Then, for k 1, the following recursion relation is defined

$$\hat{S}_{2k}(\lambda) = [\hat{S}_{2k-2}(p_k\lambda)]^2\hat{S}_{2k-2}((1-4p_k)\lambda)[\hat{S}_{2k-2}(p_k\lambda)]^2, \tag{2.48}$$

where coefficients p_k are given by

$$p_k = \frac{1}{4 - 4^{\frac{1}{2k-1}}}. \tag{2.49}$$

It has been shown by Suzuki that [24]

$$e^{-it\sum_{l=0}^{L-1}\hat{H}_l} = \hat{S}_{2k}(-it) + \mathcal{O}(t^{2k+1}). \tag{2.50}$$

On [7], Barry et. al. show that Suzuki-Trotter schemes can efficiently simulate sparse Hamiltonians, such as the ones considered in this work. As a matter of fact, they have shown that the number of exponentials (N_{exp}) required to simulate time evolution of a system during a time interval $t\ 0$, with error bounded by $\epsilon\ 0$, is such that

$$N_{\text{exp}} \leq 2L5^{2k}(L\tau)^{1+1/2k}/\epsilon^{1/2k}, \tag{2.51}$$

where $\tau = \|\hat{H}\|\, t$, a norm that measures the maximum expectation value, $2k$ is the order of a Suzuki-Trotter iteration, and $\epsilon \leq 1 \leq 2L5^{2k}$. Notice that by choosing a sufficiently high order, a Suzuki-Trotter scheme can emulate time evolution with almost linear complexity in time.

In most cases, simulation of a quantum system by a digital quantum computer requires mapping its Hilbert space to a 2^M-dimensional Hilbert space. In that case, the number of qubits required for simulation would be M. Clearly, all quantum operators in the simulated system's space should be mapped to operators on an M-qubit space, which would then be approximated by a universal set of gates. In the following section, demonstrations of this approach to the study of the Hubbard Model and the Electronic structure problem are introduced.

2.3 Time Simulation of Spin 1/2 Models

This section introduces examples of simulation of Hamiltonians such as

$$\hat{H} = \sum_{\langle i,j\rangle} J_{ij}^{(x)}\hat{X}_i\hat{X}_j + J_{ij}^{(Y)}\hat{Y}_i\hat{Y}_j + J_{ij}^{(Z)}\hat{Z}_i\hat{Z}_j + \sum_i h_i^{(X)}\hat{X}_i + h_i^{(Y)}\hat{Y}_i + h_i^{(Z)}\hat{Z}_i, \tag{2.52}$$

defined over an arbitrary spin graph, and considering nearest neighbor interaction. Most of them are related to work from Las Heras et. al. [3, 10] and Y. Salathé [20]. In this section, the importance of simulating such systems is discussed. Furthermore, the examples considered here are further generalized in following chapters to simulate any system whose Hamiltonian is of the shape of eq. 2.52, and its limitations are exemplified.

2.3.1 Digital simulation of two-spin models

As a first example, simulation of two-spin models carried out by Y. Salathé et. al. [20] are presented. In their work, a superconducting chip with two trasmon qubits is used to simulate two-spin interaction described by the Hamiltonian

$$\hat{H}_{1,2}^{x,y} = \frac{J}{2}\left(\hat{X}_1\hat{X}_2 + \hat{Y}_1\hat{Y}_2\right). \tag{2.53}$$

This means that they where able to evolve the qubits' state during time t, using microwave pulses on the chip, with unitary dynamics governed by Hamiltonian 2.53. It is easy to see that by performing single qubit rotations, it is possible to emulate more complicated dynamics, for instance, an isotropic XYZ interaction

$$\hat{H}_{1,2}^{x,y,z} = J\left(\hat{X}_1\hat{X}_2 + \hat{Y}_1\hat{Y}_2 + \hat{Z}_1\hat{Z}_2\right). \tag{2.54}$$

An algorithm for simulating time dynamics under Hamiltonian 2.54 is presented on fig. 2.4a. The reported state fidelities of the evolution process are above 82%. It must be noted that, since all terms of the Hamiltonian commute, the evolution of the two qubit state is exact, and only limited

by hardware. In addition to that, an algorithm for simulating the Ising model with transverse homogeneous magnetic field was proposed (see fig. 2.4b). The target Hamiltonian is

$$\hat{H}_I = J\hat{X}_1\hat{X}_2 + \frac{B}{2}\left(\hat{Z}_1 + \hat{Z}_2\right). \tag{2.55}$$

Notice that this Hamiltonian is composed of local parts that do not commute with one another (spin interaction and field interaction), thus an approximate evolution scheme is needed to simulate time evolution. In their work, Salathé et. al. [20] used a second order trotterization (see eq. 2.42). Furthermore, a noise model was proposed to take into account the fact that decoherence and gate errors limit the expected precision of a trotterization scheme. The results show that the main source of error is the infidelity of the two-qubit gate implementation of the XY interaction (eq. 2.53).

2.3.2 Digital simulation of Hubbard models

As a second example, the work of las Heras et. al. is considered[10]. The authors simulated instances of the quintessential Hubbard Hamiltonian

$$\hat{H}_H = -V\sum_{\langle i,j\rangle}(\hat{b}_i^\dagger\hat{b}_j + \hat{b}_j^\dagger\hat{b}_i) + U\sum_i \hat{n}_{i\uparrow}\hat{n}_{i\downarrow}, \tag{2.56}$$

where $\hat{b}_i$ represent fermionic-mode annihilation operators (for spin-up or spin-down particles), and $\hat{n}_{i\uparrow}$,$\hat{n}_{i\uparrow}$, fermionic-mode occupation number operators. This was done using the Jordan-Wigner mapping, which transforms fermionic operators to qubit operators. The rule is that each occupation mode is mapped directly to the state of a qubit, so that its state encodes the occupation number of the mode in the computational basis. Fermionic annihilation operators are mapped directly following the rule [18]

$$\hat{b}_i = \hat{\sigma}_i^+ \otimes \left(\bigotimes_{k=0}^{N}\hat{Z}_k\right), \tag{2.57}$$

where it is assumed that a linear indexing of the modes is used, even for different spin values. The authors considered models with up to four modes. In particular, they showed that the mapping of a two-mode Hubbard Hamiltonian like eq. 2.56 leads to a qubit Hamiltonian

$$\hat{H}_H = \frac{V}{2}(\hat{X}_1\hat{X}_2 + \hat{Y}_1\hat{Y}_2) + \frac{U}{4}(\hat{Z}_1\hat{Z}_2 + \hat{Z}_1 + \hat{Z}_2). \tag{2.58}$$

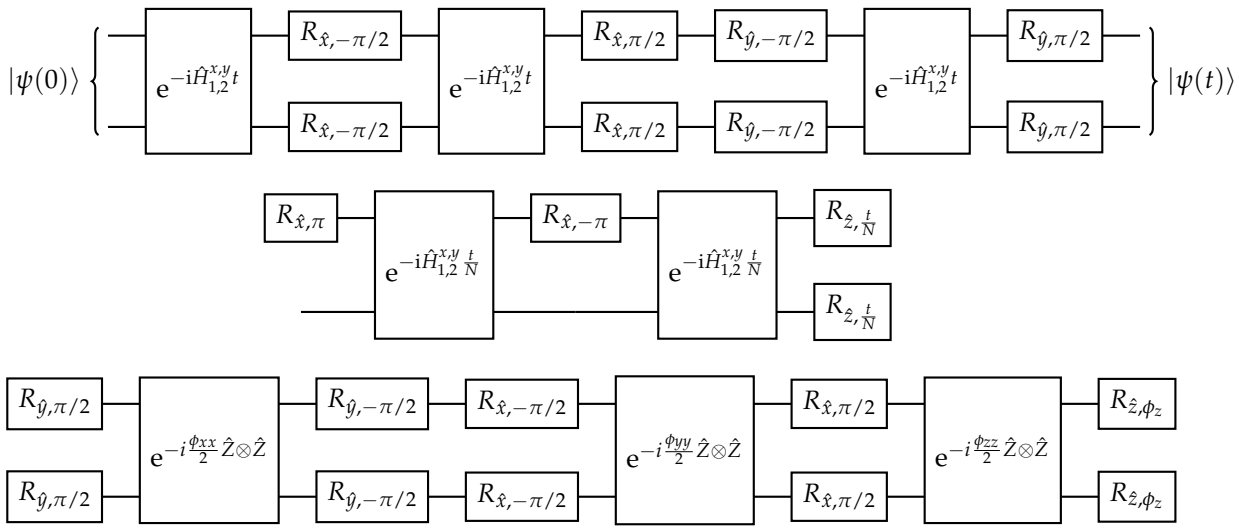
The authors used a superconducting chip for simulating time dynamics governed by Hamiltonian 2.58. The quantum algorithm is represented on figure 2.4c. It was used to simulate time evolution with an initial state

$$|\psi_0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle$$

and $V = U = 1$, for a time $t = 5$. The results show that process errors in the implementation of a Trotter step leads to a linear decrease in state fidelity with the number of steps. This is an important factor to consider when implementing Suzuki-Trotter schemes for simulating time evolution. However, their simulations were able to capture the overall dynamics, obtaining fidelities near 90% with a small number of Trotter steps. Three and four mode anisotropic models were simulated as well, obtaining similar results [10].

It has been shown by Reiner [18] that a one dimensional Hubbard model (eq. 2.56) can be mapped to a qubit Hamiltonian of the type 2.52, by means of the Jordan Wigner mapping. Thus, by generalizing the quantum algorithms presented in these examples, it is possible to study correlation phenomena on metallic solids efficiently, in contrast to current classical simulation methods [18, 12]. In following chapters, general routines for simulating evolution under Hamiltonian 2.52 are

presented, and quantum time evolution is showcased as a tool for studying magnetic properties of solids. Moreover, a discussion of the influence of decoherence and errors in the implementation is carried out, thus presenting the limitations of Suzuki-Trotter schemes for simulating many-body systems.


$|\psi(0)\rangle$
$e^{-i\hat{H}^{x,y}_{1,2}t}$
$R_{\hat{x},-\pi/2}$
$R_{\hat{x},-\pi/2}$
$e^{-i\hat{H}^{x,y}_{1,2}t}$
$R_{\hat{x},\pi/2}$
$R_{\hat{y},-\pi/2}$
$R_{\hat{x},\pi/2}$
$R_{\hat{y},-\pi/2}$
$e^{-i\hat{H}^{x,y}_{1,2}t}$
$R_{\hat{y},\pi/2}$
$R_{\hat{y},\pi/2}$
$|\psi(t)\rangle$
$R_{\hat{x},\pi}$
$e^{-i\hat{H}^{x,y}_{1,2}\frac{t}{N}}$
$R_{\hat{x},-\pi}$
$e^{-i\hat{H}^{x,y}_{1,2}\frac{t}{N}}$
$R_{\hat{z},\frac{t}{N}}$
$R_{\hat{z},\frac{t}{N}}$
$R_{\hat{y},\pi/2}$
$R_{\hat{y},\pi/2}$
$e^{-i\frac{\phi_{xx}}{2}\hat{Z}\otimes\hat{Z}}$
$R_{\hat{y},-\pi/2}$
$R_{\hat{y},-\pi/2}$
$R_{\hat{x},-\pi/2}$
$R_{\hat{x},-\pi/2}$
$e^{-i\frac{\phi_{yy}}{2}\hat{Z}\otimes\hat{Z}}$
$R_{\hat{x},\pi/2}$
$R_{\hat{x},\pi/2}$
$e^{-i\frac{\phi_{zz}}{2}\hat{Z}\otimes\hat{Z}}$
$R_{\hat{z},\phi_z}$
$R_{\hat{z},\phi_z}$

2.4 Pulse Efficient Circuit Implementations

In general, current quantum processors are very sensitive to noise. Undesired (and some uncontrollable) interactions may occur between qubits and the environment, or with control and measurement equipment, that lead to a loss of quantum information and coherence of a device's state [13]. In cases of weakly coupled noise sources, decoherence is modeled by qubit energy relaxation and transverse relaxation. The former is related to ground state decay, or depolarization along the $\hat{z}$ axis of the Bloch sphere. The later, is related to decoherence of superposition states, or depolarization on the plane $\hat{x} - \hat{y}$ of this sphere. Those errors are described by characteristic times T_1 and T_2, respectively. In the presence waekly coupled sources of noise, the Bloch-Redfield density matrix of a qubit with initial state

$$|\phi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle \tag{2.59}$$

evolves in time like [13]

$$\hat{\rho}_{BR} = \begin{bmatrix} 1-(1-|\alpha|^2)e^{-t/T_1} & \beta^*\alpha e^{i\delta t}e^{-t/T_2} \\ \alpha^*\beta e^{-i\delta t}e^{-t/T_2} & |\beta|^2 e^{-t/T_1} \end{bmatrix}, \tag{2.60}$$

where the constant δ is associated to precession of the Bloch vector. This models both longitudinal and transverse relaxation.

Among other sources of error, quantum devices experience thermal relaxation due to interaction with the environment, charge and magnetic flux fluctuations, and decoherence due to photon number fluctuations in the system [13].

IBM Quantum devices have characteristic times of tens of microseconds, while common two-qubit gates have implementation times of hundreds of nanoseconds. As a result, most theorized algorithms are unable to run on current devices, due to decoherence [22, 8, 6, 16]. Direct transpilation of standard quantum algorithms to the basis set of a quantum device, such as the Quantum Fourier Transform or Quantum Phase Estimation [17], even with optimization techniques, require execution times large enough to exceed the coherence times of the qubits.

In this section, some techniques for quantum control of superconducting qubits are introduced, as well as novel techniques for efficient hardware implementation of quantum algorithms based on the cross resonance interaction [8, 21]. Those techniques aim towards optimal usage of the finite coherence times of superconducting qubits on current quantum devices. The insights here presented are the foundations of the time simulation algorithms discussed in following chapters.

2.4.1 Transmon qubits and single qubit gates

IBM Quantum devices consist on a set of coupled superconducting qubits known as transmons [2, 13]. A transmon consists on a Josephson junction shunted by a capacitance. Such devices can be described by the quantum variables of charge ($\hat{Q}$) and magnetic flux ($\hat{\phi}$). This due to the collective motion of Cooper pairs in certain superconducting materials. Those are quantum conjugate variables that satisfy the commutation relation

$$[\hat{Q}, \hat{\phi}] = i\hbar \tag{2.61}$$

The effective Hamiltonian of a transmon is [13]

$$\hat{H}_T = 4E_C\hat{n} - E_J\cos\hat{\phi}, \tag{2.62}$$

where $\hat{n} = \hat{Q}/2e$ is the reduced charge of the transmon. The constants E_C and E_J represent the capacitive and junction energy, respectively. In the transmon regime, the junction energy is much larger than the capacitive energy, and the Hamiltonian 2.62 can be simplified to a Duffing oscillator Hamiltonian [13]. The key insight is that the energy levels of the system are separated by different amounts from their immediate neighboring levels. Hence, engineering techniques can be devised

for constraining the transmon to the subspace spanned by its first two levels, where the effective Hamiltonian is simply

$$\hat{H}_{efe} = -\frac{\omega_0}{2}\hat{Z}. \tag{2.63}$$

Single qubit gates are implemented on hardware by a capacitive coupling to a microwave signal generator. The process is known as *Rabi driving*, and is carefully summarized on [13]. The advantage of this approach is that rotations around the $\hat{x}$ and $\hat{y}$ axes of the Bloch sphere can be controlled by the phase, amplitude and frequency of the microwave driving (which usually occurs at the resonance frequency of the transmon). Moreover, it is possible to implement *virtual rotations* around the $\hat{z}$ axis by controlling the phase of the drive as illustrated on [13]. Those gates employ zero execution time, and have maximum fidelity. As explained on that reference, by introducing appropriate phases between the driving pulses, effective $\hat{z}$ rotations occur when implementing consecutive $\hat{x}$ or $\hat{y}$ rotations. Given that any single qubit unitary has a decomposition

$$\hat{U}(\theta,\phi,\lambda) = \begin{bmatrix} \cos\frac{\theta}{2} & -e^{i\lambda}\sin\frac{\theta}{2} \\ e^{i\phi}\sin\frac{\theta}{2} & e^{i(\phi+\lambda)}\cos\frac{\theta}{2} \end{bmatrix} = \hat{R}_{z,\phi-\pi/2}\sqrt{X}\hat{R}_{z,\pi-\theta}\sqrt{X}\hat{R}_{z,\lambda-\pi/2}, \tag{2.64}$$

it is possible to implement arbitrary single qubit rotations with only two microwave pulses at most. This is very similar to the way IBM Quantum backends implement this type of rotations on hardware [2]. The main sources of error on digital quantum computation are entangling interactions [20]. In this section, the fundamental notions regarding two qubit gate implementation and novel advances in recent years are presented.

2.4.2 Cross resonance interaction

Transmon qubit pairs usually are coupled to one another by a quantum bus whose frequency is different to the resonance frequency of either qubit [13, 21]. In most practical applications, the frequency difference of the two qubits is much larger than the qubit-bus coupling strength. Mathematical modelling of this system uses a Jaynes-Cummings coupling to the bus electromagnetic modes. The details of the study of circuit QED for modelling qubit-qubit interaction can be found on [15]. In general, entangling operations are performed using so called *cross resonance pulses*, in which a target qubit is driven at the *dressed* [1] resonance frequency of the other qubit. For the purpose of this work, it is sufficient to know that the effective interaction between the qubits can be modelled by the **cross resonance Hamiltonian** [23, 21]

$$\hat{H}_{CR}(\Omega) = \nu_{ZX}\frac{\hat{Z}_1\hat{X}_2}{2} + \nu_{IZ}\frac{\hat{Z}_2}{2} + \nu_{ZI}\frac{\hat{Z}_1}{2} + \nu_{IX}\frac{\hat{X}_2}{2} + \nu_{ZZ}\frac{\hat{Z}_1\hat{Z}_2}{2}, \tag{2.65}$$

where 1 denotes the control qubit, 2, the target, and the coefficients depend on the driving pulse amplitude Ω. The cross resonance interaction is used on fixed-frequency transmons to implement entangling gates; specifically, highly calibrated CNOT gates. Ideally, cross resonance interaction would only have a ZX interaction, thus generating a controlled rotation gate

$$CR(\theta) = e^{-i\frac{\theta}{2}\hat{Z}_1\hat{X}_2}. \tag{2.66}$$

In that case, it would be possible to implement CNOT gates using the decomposition [13]

$$\text{CNOT} = \hat{R}_{z_1,\pi/2} \otimes \hat{R}_{x_2,\pi/2} \cdot CR(\theta). \tag{2.67}$$

However, experiments show that, in general, all coefficients on Hamiltonian 2.65 are non zero [23, 21]. As a result, undesired terms must be suppressed so that the evolution Hamiltonian is consistent with the target ZX interaction that generates CNOT gates.

[1] In this context, *dressed* reference the effective qubit frequencies when projecting the Hamiltonian to the subspace of zero excitations of the quantum bus [15]

2.4.3 Echoed cross resonance gates

The current approach towards implementing high fidelity cross resonance gates is outlined on [23]. The key insight is that the coefficients of the diagonal terms in the Hamiltonian 2.65 are even functions of the pulse amplitude, while the off-diagonal terms are odd functions of this quantity. As a result, by reversing the sign of the driving pulse amplitude, it is possible to mitigate the undesired diagonal terms by evolving the qubits by the operator

$$\hat{U} = \mathrm{e}^{-\mathrm{i}\hat{H}_{CR}^{efe}(\Omega)t} = \mathrm{e}^{-\mathrm{i}\hat{H}_{CR}(\Omega)t}\mathrm{e}^{-\mathrm{i}\hat{X}_1\hat{H}_{CR}(-\Omega)\hat{X}_1 t}. \tag{2.68}$$

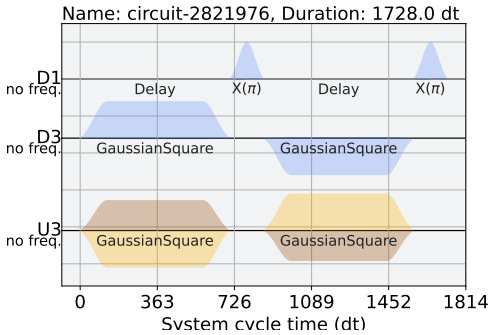
This scheme leads to an effective interaction Hamiltonian [23]

$$\hat{H}_{CR}^{efe}(\Omega) = \bar{\nu}_{ZX}\frac{\hat{Z}_1\hat{X}_2}{2} + \bar{\nu_{IZ}}\frac{\hat{Z}_2}{2} + \bar{\nu_{IY}}\frac{\hat{Y}_2}{2}. \tag{2.69}$$

Undesired unitary error terms can be mitigated by applying appropriate pulses on the target qubit. The general scheme for implementing cross resonance gates on IBM Quantum systems can be seen on figure 2.5. The cross resonance drive is carried out by gaussian square (flat top) pulses, and the $\hat{X}$ pulses are implemented using a Gaussian drive as discussed before in this section. Those pulses are defined by the parameters: width (w), amplitude (A), standard deviation of the rise/fall (σ), and number of standard deviations in the rise/fall (n_σ). The area of a gaussian flat top pulse is

$$\alpha = |A|(w + \sigma\sqrt{2\pi}\,\mathrm{erf}\, n_\sigma), \tag{2.70}$$

where erf $\cdot$ is the error function gaussian integral. Experiments are carried out for finding the optimal value of the parameters for implementing a CNOT gate. In general, if a rotation angle of θ is desired, the area must be scaled in direct proportion to this angle. More mathematical details and experimental demonstrations can be found on [23, 21, 22].



2.4.4 Cross-resonance based transpilation of quantum includes

As mentioned before, IBM Quantum exposes a highly calibrated CNOT gates as entangling operations. However, in some instances, CNOT based circuit transpilation to microwave pulse schedules lead to execution times that are impossible to execute with high fidelity on current quantum devices. An example is time simulation of Majorana fermionic systems [22]. A key insight towards performing pulse efficient implementation of quantum algorithms is to use the highly calibrated CNOT schedule exposed by IBM Quantum backends to perform cross resonance gates discussed on the previous section [22, 8].

The details on the gaussian pulse scaling can be found on [22]. The idea is to scale the area of the pulses in direct proportion to the angle of rotation of the cross resonance gate

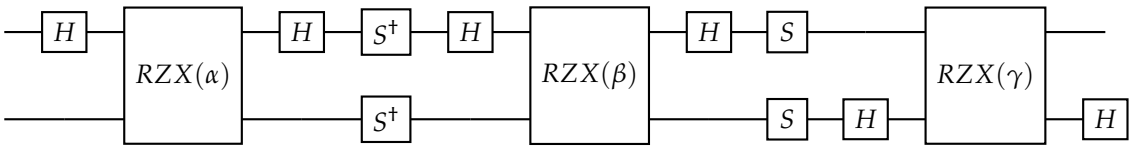
$$\alpha(\theta) = \frac{\theta}{\pi/2}\alpha^*, \tag{2.71}$$

where α^* is the optimal area calibrated for a CNOT gate. Then, adjust the width, amplitude, and duration of the pulse according to equation 2.70. Since the technique uses CNOT calibration, it is more efficient than running calibration experiments each time an algorithm wants to be executed [22].

Through Qiskit SDK, it is possible to perform this calibration as discussed in the appendix of [8].

As pointed out on [8], with calibrated cross resonance gates, it is possible to implement two qubit gates of the group

$$\hat{U}(\alpha,\beta,\gamma) = \mathrm{e}^{-\mathrm{i}(\alpha\hat{X}\hat{X}+\beta\hat{Y}\hat{Y}+\gamma\hat{Z}\hat{Z})}. \tag{2.72}$$



As shown on [8] and [22], this type of algorithm transpilation leads to a significant relative error reduction for shorter pulse schedules, allowing time simulation of interesting physical systems. Benchmark tests were made in comparison to the three CNOT decomposition depicted on figure 2.3. It was shown experimentally that the relative error can be decreased up to 50% by implementing pulse efficient algorithms on current quantum devices [8].

Chapter 3

Time Simulation of Anisotropic Spin Chains

This chapter introduces time simulation algorithms for Hamiltonian 2.52. First a trotterization scheme is introduced, and its advantages are discussed for simulating certain types of graphs or lattices. Relations between discretization step, evolution time and evolution error are computed and discussed in order to establish disadvantages of execution on current (noisy) quantum devices. Then, three possible implementation strategies are introduced: 1) a direct transpilation of includes proposed by Las Heras et. al. [10], 2) a basis-efficient transpilation relying on commutation properties of local Hamiltonians [1], and 3) a pulse-efficient transpilation based on cross-resonance interaction. Finally, the three algorithms are tested using a three-qubit Hamiltonian. Single-qubit Pauli expected value evolution and probability density evolution as qualitative indicators, whereas probability density fidelity and state fidelity are used as quantitative indicators.

3.1 Trotterization And Time Evolution

Consider the multiple spin Hamiltonian

$$\hat{H} = \sum_{\langle i,j \rangle} J_{ij}^{(X)} \hat{X}_i \hat{X}_j + J_{ij}^{(Y)} \hat{Y}_i \hat{Y}_j + J_{ij}^{(Z)} \hat{Z}_i \hat{Z}_j + \sum_i h_i^{(X)} \hat{X}_i + h_i^{(Y)} \hat{Y}_i + h_i^{(Z)} \hat{Z}_i, \tag{3.1}$$

defined over an arbitrary graph. The shape already suggests that the Hamiltonian above can be decomposed in local interactions of the shape

$$\hat{H}_{ij} = J_{ij}^{(X)} \hat{X}_i \hat{X}_j + J_{ij}^{(Y)} \hat{Y}_i \hat{Y}_j + J_{ij}^{(Z)} \hat{Z}_i \hat{Z}_j, \tag{3.2}$$

$$\hat{H}_i = h_i^{(X)} \hat{X}_i + h_i^{(Y)} \hat{Y}_i + h_i^{(Z)} \hat{Z}_i, \tag{3.3}$$

such that

$$\hat{H} = \sum_{\langle i,j \rangle} \hat{H}_{ij} + \sum_i \hat{H}_i. \tag{3.4}$$

This leads to a direct second order trotterization of the shape

[1] The work was performed at early stages independently of that on [25]. However, the insights are identical and thus the author is compelled to refer to this previous work. It is clear, however, that the mentioned reference considers the problem of universal two qubit gates which, albeit related to the specific problem of the present dissertation, is a quite different approach. This dissertation uses the results derived to solve a specific time evolution problem with potential application to specific areas of solid state physics and physical chemistry.

$$e^{-i\hat{H}\Delta t} \approx \prod_{\langle i,j \rangle} e^{-i\hat{H}_{i,j}\Delta t} \prod_{i} e^{-i\hat{H}_i \Delta t} + \mathcal{O}(\Delta t^2). \tag{3.5}$$

In general, interactions associated to disjoint edges commute, and thus can be simulated simultaneously. Hence, an advantage of this approach to time evolution is that by partitioning the graph on sets of mutually disjoint sets, several terms can be implemented in parallel on actual quantum devices. This approach is implemented in the present work, and discussed further on the appendix. This parallelism is illustrated for a spin chain in figure 3.1. It can be noticed that the circuit depth of the trotter step of three or more spins with chain topology is independent of the number of spins. As a result, time complexity only increases with the desired time discretization, which correlates with the error in the time simulation approximation.

Another interesting feature, as shown in figure 3.1, is that it is possible to perform third order time evolution using the first iteration of Suzuki-Trotter scheme with roughly the sme time complexity as the second order trotterization. This, clearly, in the case where no external local fields are present in the model Hamiltonian, and the graph corresponds to a chain. To illustrate the point more precisely, consider the Hamiltonian

$$\hat{H} = \sum_{i=0}^{N-2} \hat{H}_{i,i+1}, \tag{3.6}$$

where $\hat{H}_{i,j}$ is defined as on equation 3.3. It is quite straightforward to see the that the second order evolution corresponds to the approximation

$$e^{-i\hat{H}\Delta t} \approx \prod_{i \text{ even}} e^{-i\hat{H}_{i,i+1}t} \prod_{i \text{ odd}} e^{-i\hat{H}_{i,i+1}t} + \mathcal{O}(\Delta t^2). \tag{3.7}$$

For this particular system, denote

$$\hat{A}(\Delta t) = \prod_{i \text{ even}} e^{-i\hat{H}_{i,i+1}t} \tag{3.8}$$

$$\hat{B}(\Delta t) = \prod_{i \text{ odd}} e^{-i\hat{H}_{i,i+1}t} \tag{3.9}$$

such that

$$e^{-i\hat{H}\Delta t} \approx \hat{A}(\Delta t)\hat{B}(\Delta t) + \mathcal{O}(\Delta t^2) \tag{3.10}$$

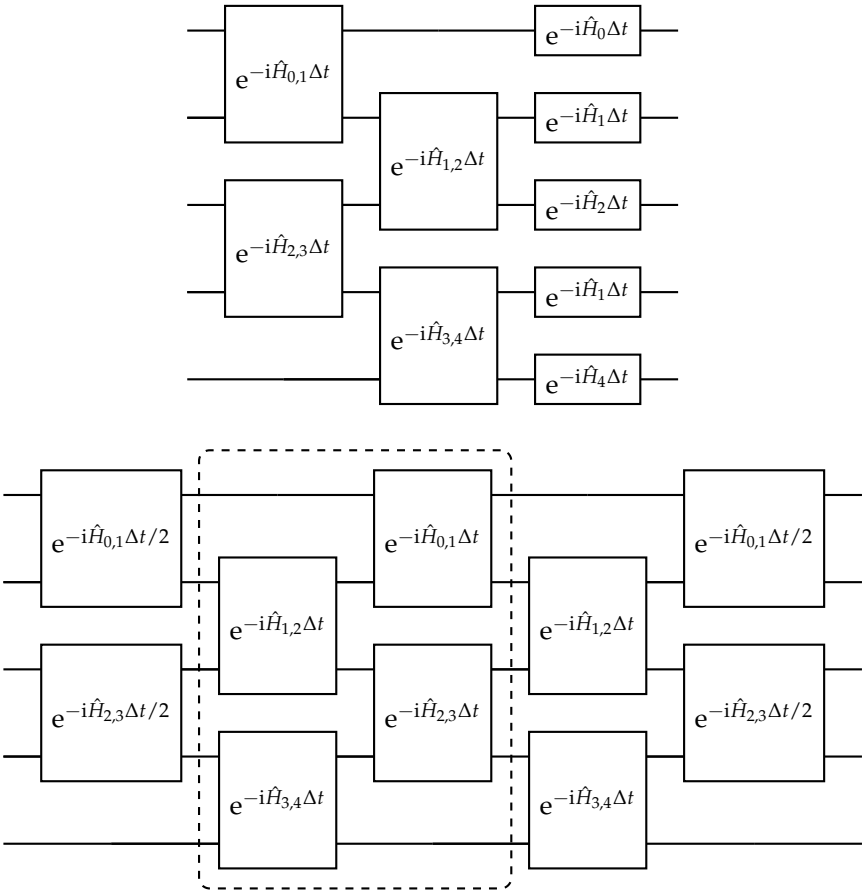
Simulation over a time $t = M\Delta t$ yields

$$e^{-i\hat{H}t} \approx \left(\hat{A}(\Delta t)\hat{B}(\Delta t)\right)^M. \tag{3.11}$$

The third order scheme may be implemented using Suzuki-Trotter zeroth order evolution (eq. 2.43), yielding the following finite time approximation

$$\begin{aligned} e^{-i\hat{H}t} \approx & \left(\hat{A}(\Delta t/2)\hat{B}(\Delta t)\hat{A}(\Delta t/2)\right)^M \\ = & \hat{A}(\Delta t/2)\left(\hat{B}(\Delta t)\hat{A}(\Delta t)\right)^{M-1}\hat{B}(\Delta t)\hat{A}(\Delta t/2) \\ = & \hat{A}(\Delta t/2)\hat{B}(\Delta t)\left(\hat{A}(\Delta t)\hat{B}(\Delta t)\right)^{M-1}\hat{A}(\Delta t/2). \end{aligned} \tag{3.12}$$

It can be seen that the *power* operator (the one with a power in the approximation unitary) on each scheme can be implemented in the same fashion on a quantum circuit. Hence, both second order and third order schemes have roughly the same time complexity when implemented on quantum devices. This optimization is taken into account during implementation on IBM Quantum backends.


$e^{-i\hat{H}_{0,1}\Delta t}$
$e^{-i\hat{H}_{1,2}\Delta t}$
$e^{-i\hat{H}_{2,3}\Delta t}$
$e^{-i\hat{H}_{3,4}\Delta t}$
$e^{-i\hat{H}_0\Delta t}$
$e^{-i\hat{H}_1\Delta t}$
$e^{-i\hat{H}_2\Delta t}$
$e^{-i\hat{H}_1\Delta t}$
$e^{-i\hat{H}_4\Delta t}$
$e^{-i\hat{H}_{0,1}\Delta t/2}$
$e^{-i\hat{H}_{2,3}\Delta t/2}$
$e^{-i\hat{H}_{1,2}\Delta t}$
$e^{-i\hat{H}_{3,4}\Delta t}$
$e^{-i\hat{H}_{0,1}\Delta t}$
$e^{-i\hat{H}_{2,3}\Delta t}$
$e^{-i\hat{H}_{1,2}\Delta t}$
$e^{-i\hat{H}_{3,4}\Delta t}$
$e^{-i\hat{H}_{0,1}\Delta t/2}$
$e^{-i\hat{H}_{2,3}\Delta t/2}$

3.1.1 Limitations of The Scheme for Time Evolution

There are some intrinsic limitations to quantum time simulation using Trotter schemes. To exemplify this, consider an isotropic N-spin chain Hamiltonian, with only two-spin interaction given by

$$\hat{H}_{ij} = \frac{1}{2}\hat{X}_i\hat{X}_j + \hat{Y}_i\hat{Y}_j + \frac{1}{4}\hat{Z}_i\hat{Z}_j. \tag{3.13}$$

Consider an initial state $|\psi_0\rangle = |11\rangle \otimes |0\rangle^{\otimes N-2}$. Evolution over a time t would yield an output (target) state

$$|\psi_t\rangle = \mathrm{e}^{-\mathrm{i}\hat{H}t}|\psi_0\rangle\,. \tag{3.14}$$

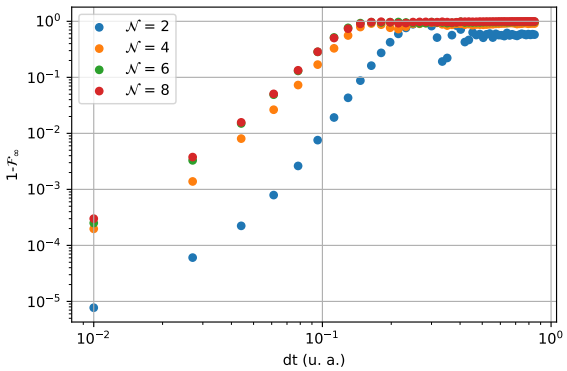
On the other hand, this target state may be approximated by the unitary evolution

$$|\psi_t(\Delta t)\rangle = \hat{U}(\Delta t)^{\lfloor \frac{t}{\Delta t} \rfloor}|0\rangle\,. \tag{3.15}$$

The long time average state fidelity $\mathcal{F}_\infty$ can be defined as follows:

$$\mathcal{F}_\infty(\Delta t) = \lim_{M\to\infty}\frac{1}{M}\sum_{m=1}^{M}|\,\langle\psi_{M\Delta t}|\psi_{M\Delta t}(\Delta t)\rangle\,|^2. \tag{3.16}$$

In figure 3.2, a plot of long time average state infidelity $1-\mathcal{F}_\infty$ can be seen, for different system sizes. It can be seen that there are two clearly separated regimes. For large integration time step Δt, the average fidelity is almost zero, which means that the evolution does not converge in general. However, below certain threshold Δt_c, the average fidelity increases deterministically, following a power law. The results suggest that there may be a limit of large systems (i.e., a possible thermodynamic limit). This results have been observed on previous work by [11]. In that work, it is claimed that the low fidelity regime experience a quantum chaotic dynamics, while the deterministic fidelity regime experiences quantum localization of the system's state. Evidence is provided for the transverse field Ising model. However, due to time constraints, those claims are not demonstrated in this work for the example interaction 3.13.



3.2 Circuit implementations

On chapter 2, some networks for simulating time evolution of the two-qubit Hamiltonian were introduced (see fig. 2.4). In this section, three possible transpiled includes are introduced. Those implement the single qubit and two qubit operators on equation 3.3. The single qubit operators are implemented in the same way on all three alternatives. Each circuit differs from the others on the implementation of the two spin operators. The first alternative is a direct basis transpilation, based on the controlled phase gate. The second alternative is one that takes advantage of the commutation

properties of the local two spin Hamiltonian. This alternative was derived mostly independently from [25] at early stages of the present work. However,a thorough discussion of the insights required to derive this circuit is included. The last option is a cross resonance based implementation as discussed on chapter 2.

3.2.1 Simulation of field interaction

To simulate evolution under Hamiltonian 3.3, a direct approach would be to us te definition of single qubit rotations, and implement a second order trotterization scheme as illustrated on figure 3.3a. However, exact simulation of this model is possible by rotating the Bloch sphere main axes so that the external field points to the $\hat{z}$ direction. This can be done by the operator

$$\hat{U}_{\theta,\phi} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) & \sin\left(\frac{\theta}{2}\right) \\ e^{i\phi}\sin\left(\frac{\theta}{2}\right) & -e^{i\phi}\cos\left(\frac{\theta}{2}\right), \end{bmatrix} \tag{3.17}$$

where θ and ϕ are defined by the spherical representation of the external field (see eq. 3.21):

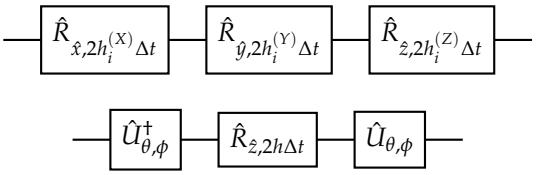
$$h^2 = (h_i^{(x)})^2 + (h_i^{(y)})^2 + (h_i^{(z)})^2, \tag{3.18}$$

$$h_i^{(x)} = h\sin(\theta)\cos(\phi), \tag{3.19}$$

$$h_i^{(y)} = h\sin(\theta)\sin(\phi), \tag{3.20}$$

$$h_i^{(z)} = h\cos(\theta). \tag{3.21}$$

This approach is illustrated on figure 3.3b. Therefore, at the same computational cost, this interaction can be simulated exactly by the former algorithm.



3.2.2 Simulation of Two-spin Interaction

The simpler spin-spin Hamiltonian

$$\hat{H}_{ij} = J_{ij}^{(X)}\hat{X}_i\hat{X}_j + J_{ij}^{(Y)}\hat{Y}_i\hat{Y}_j + J_{ij}^{(Z)}\hat{Z}_i\hat{Z}_j \tag{3.22}$$

is simulated using IBM Quantum device's universal set, or cross resonance pulses. This is the most computationally expensive part of the evolution scheme. As may be seen on figure 2.5, the most time consuming processes are simulating two qubit interactions. In consequence, reducing the number and duration of CNOT gates or cross resonance pulses on the evolution algorithm is crucial for obtaining high fidelity results. Here, a first network that performs direct transpilation of circuit 2.4c is presented. It will be used as a control case, since non-optimized transpilations would yield this network for simulating the Hamiltonian [2]. A basis efficient circuit is introduced, and its mathematical and physical insights are discussed [25]. Finally, the pulse efficient network proposed on [8] is revisited.

Direct transpilation circuit

To adapt circuit 2.4c for IBM Quantum devices, it is helpful to note that

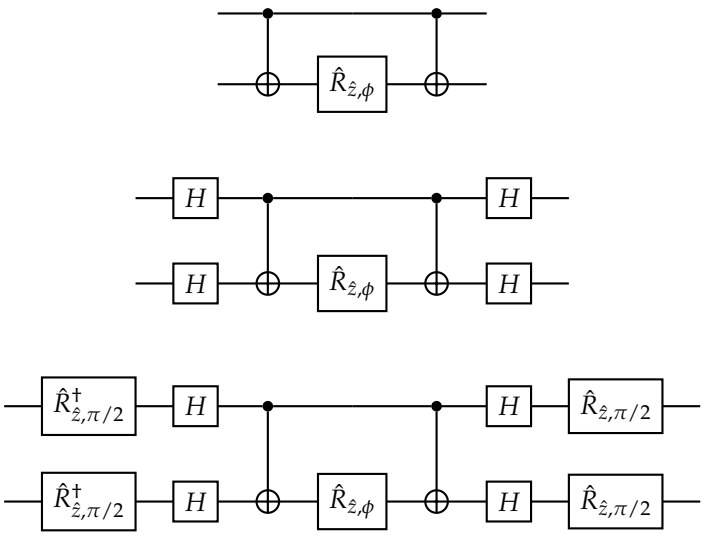
$$e^{-i\phi\hat{Z}_i\otimes\hat{Z}_j} = \cos\left(\frac{\phi}{2}\right) - i\sin\left(\frac{\phi}{2}\right)\hat{Z}_i\otimes\hat{Z}_j = \begin{bmatrix} e^{-i\frac{\phi}{2}} & 0 & 0 & 0 \\ 0 & e^{i\frac{\phi}{2}} & 0 & 0 \\ 0 & 0 & e^{i\frac{\phi}{2}} & 0 \\ 0 & 0 & 0 & e^{-i\frac{\phi}{2}}. \end{bmatrix} \tag{3.23}$$

By the definition of CNOT gate, and single-qubit gates (see chap. 2), it follows that this operator can be implemented by the circuit on figure 3.4a. A direct way to simulate the XX interaction term is to note that

$$\hat{H}^{\otimes 2} e^{-i\phi\hat{Z}_i\otimes\hat{Z}_j}\hat{H}^{\otimes 2} = e^{-i\phi\hat{X}_i\otimes\hat{X}_j}, \tag{3.24}$$

where $\hat{H}$ means the Hadamard gate, not the target Hamiltonian. This follows from the observation that $\hat{H}\hat{Z}\hat{H} = \hat{X}$. In a similar fashion, it is possible to implement the YY interaction by noticing that

$$\left(\hat{R}^{\dagger}_{\hat{z},\pi/2}\hat{H}\right)\hat{Z}\left(\hat{R}^{\dagger}_{\hat{z},\pi/2}\hat{H}\right)^{\dagger} = \hat{Y}. \tag{3.25}$$



This leads to a straightforward algorithm for simulating spin-spin interaction that uses 6 CNOT gates, and 15 single-qubit rotations.

Basis efficient circuit

This gate count can be reduced further by considering the commutation relations between the operators that constitute the spin-spin interaction Hamiltonian

$$[\hat{X}_i\hat{X}_j, \hat{Z}_i\hat{Z}_j] = [\hat{Y}_i\hat{Y}_j, \hat{Z}_i\hat{Z}_j] = 0. \tag{3.26}$$

From basic quantum mechanics, there exists a common basis in which time evolution under hamiltonian 3.22 implies appending global phases via single qubit rotations around $\hat{z}$ axis, and less two-qubit operations. This basis is straightforward to find by noticing that the *total spin* operator

$$\hat{S}^2 = 6 + 2\left(\hat{X}_i\hat{X}_j + \hat{Y}_i\hat{Z}_j + \hat{Z}_j\right) \tag{3.27}$$

commutes with the spin-spin interaction Hamiltonian. From elementary quantum physics, it is known that the eigenstates of such operator are the singlet and triplet states [4]. By mapping quantum bit value to spin value directly, it can be readily seen that the singlet and triplet states correspond exactly to the *Bell states* defined on equations 2.23. By considering that

$$\hat{X}_i\hat{X}_j\left|\Phi^{\pm}\right\rangle = \pm\left|\Phi^{\pm}\right\rangle, \tag{3.28}$$
$$\hat{Y}_i\hat{Y}_j\left|\Phi^{\pm}\right\rangle = \mp\left|\Phi^{\pm}\right\rangle, \tag{3.29}$$
$$\hat{Z}_i\hat{Z}_j\left|\Phi^{\pm}\right\rangle = \left|\Phi^{\pm}\right\rangle, \tag{3.30}$$

$$\hat{X}_i\hat{X}_j\left|\Psi^{\pm}\right\rangle = \pm\left|\Psi^{\pm}\right\rangle, \tag{3.31}$$
$$\hat{Y}_i\hat{Y}_j\left|\Psi^{\pm}\right\rangle = \pm\left|\Psi^{\pm}\right\rangle, \tag{3.32}$$
$$\hat{Z}_i\hat{Z}_j\left|\Psi^{\pm}\right\rangle = -\left|\Psi^{\pm}\right\rangle, \tag{3.33}$$

it is possible to obtain the energies of the Hamiltonian:

$$\hat{H}_{ij}\left|\Psi^{\pm}\right\rangle = \left(-J_{ij}^{(Z)} \pm (J_{ij}^{(X)} + J_{ij}^{(Y)})\right)\left|\Psi^{\pm}\right\rangle, \tag{3.34}$$
$$\hat{H}_{ij}\left|\Phi^{\pm}\right\rangle = \left(J_{ij}^{(Z)} \pm (J_{ij}^{(X)} - J_{ij}^{(Y)})\right)\left|\Phi^{\pm}\right\rangle. \tag{3.35}$$

To each term $J_{ij}^{(X)}$, $J_{ij}^{(Y)}$, $J_{ij}^{(Z)}$, it is possible to assign a phase ϕ_{xx}, ϕ_{yy}, ϕ_{zz}. Those are defined as follows

$$\phi_{xx} = 2J_{ij}^{(X)}\Delta t, \tag{3.36}$$
$$\phi_{yy} = -2J_{ij}^{(Y)}\Delta t, \tag{3.37}$$
$$\phi_{zz} = 2J_{ij}^{(Z)}\Delta t, \tag{3.38}$$

where Δt is the time interval to be simulated. A quantum circuit representing this approach to evolution is presented on figure 3.5. Main stages are separated by slices, which correspond to:

1. Basis change from computational to Bell.
2. Append ϕ_{xx} and ϕ_{zz} phases.
3. Shuffle the basis to append ϕ_{yy} phase.
4. Return to ordered Bell basis

In the first stage, the Bell basis is mapped according to

$$\left|\Phi^{+}\right\rangle \to \left|00\right\rangle, \tag{3.39}$$
$$\left|\Phi^{-}\right\rangle \to \left|01\right\rangle, \tag{3.40}$$
$$\left|\Psi^{+}\right\rangle \to \left|10\right\rangle, \tag{3.41}$$
$$\left|\Psi^{-}\right\rangle \to \left|11\right\rangle. \tag{3.42}$$

From equations 3.35, it may be noted that xx phase correlates to the less significant bit, while zz phase correlates to the most significant bit. Also, yy phase correlates to the parity of the mapped computational basis state, hence the need of a CNOT gate. The last part undoes the CNOT gate

action on the previous step, and returns to the Bell basis. The direct way to perform the last step is illustrated on figure 3.5(a). A more clever approach relies on the following equalities (up to global state phases)

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}}(|{+\mathrm{i}},{-\mathrm{i}}\rangle + |{-\mathrm{i}},{+\mathrm{i}}\rangle), \tag{3.43}$$

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) = \frac{1}{\sqrt{2}}(|{+\mathrm{i}},{+\mathrm{i}}\rangle + |{-\mathrm{i}},{-\mathrm{i}}\rangle), \tag{3.44}$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|{+\mathrm{i}},{+\mathrm{i}}\rangle - |{-\mathrm{i}},{-\mathrm{i}}\rangle), \tag{3.45}$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) = \frac{1}{\sqrt{2}}(|{+\mathrm{i}},{-\mathrm{i}}\rangle - |{-\mathrm{i}},{+\mathrm{i}}\rangle), \tag{3.46}$$

where the single qubit states $\{|+\mathrm{i}\rangle, |+\mathrm{i}\rangle\}$ are defined as on chapter 2, and correspond to the $\hat{Y}$ eigenstates. The shuffling stage that appends yy phase actually permutes states $|\Psi^-\rangle$ and $|\Psi^+\rangle$ of the Bell basis. It is now easy to see that by performing single qubit rotations that are equivalent to the mappings (where the subindex corresponds to qubits 0 and 1, respectively)

$$|0\rangle_0 \to |-\mathrm{i}\rangle_0, \tag{3.47}$$

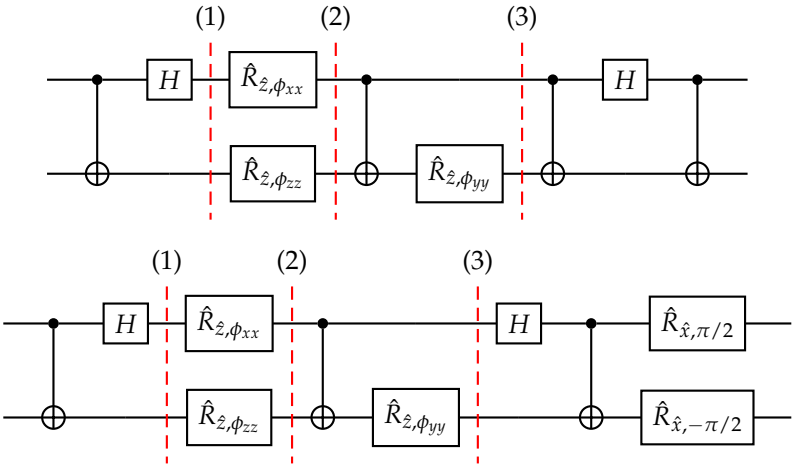
$$|1\rangle_0 \to -\mathrm{i}|+\mathrm{i}\rangle_0, \tag{3.48}$$

$$|0\rangle_1 \to |+\mathrm{i}\rangle_1, \tag{3.49}$$

$$|1\rangle_1 \to \mathrm{i}|-\mathrm{i}\rangle_1, \tag{3.50}$$

$$\tag{3.51}$$

it is possible to perform the desired reordering without using an additional CNOT gate. Such procedure is illustrated on figure 3.5(b). On the last state, the computational basis is mapped back to the Bell basis, and the reordering is performed on the Bell basis using local rotations that correspond to transformation 3.51. As a result, the number of expensive CNOT gates has been halved with respect to the direct transpilation circuit.



3.2.3 Pulse efficient implementation

On subsection 2.4.3, a pulse schedule, introduced in [8], was presented as an efficient alternative for implementing the unitary operators

$$\hat{U}(\alpha,\beta,\gamma) = \mathrm{e}^{-\mathrm{i}(\alpha\hat{X}\hat{X}+\beta\hat{Y}\hat{Y}+\gamma\hat{Z}\hat{Z})}. \tag{3.52}$$

It can be readily seen that the relations

$$\alpha = J_{xx}\Delta t, \tag{3.53}$$
$$\beta = J_{yy}\Delta t, \tag{3.54}$$
$$\gamma = J_{zz}\Delta t, \tag{3.55}$$

yield a unitary that exactly performs time evolution under the two-spin interaction Hamiltonian. The single qubit rotations used on the network representation 2.6 profit the relations between Pauli operators stated on equations 3.24 and 3.25. This algorithm has a slightly different nature than those presented previously. The former two use a particular basis to implement a quantum algorithm that can run on any universal device, regardless of the underlying architecture. The one discussed here, on the other hand, is specifically calibrated for IBM Quantum devices using Qiskit SDK and the pulse scaling technique implemented on [8].

3.3 Comparison and Benchmark: Methodology

In general, time simulation of a quantum system has at least two requirements: 1) that the state after evolution resembles within given error bounds the actual target state, and 2) that expected values of interesting observables can be extracted within given error bounds. In the present work, those two requirements are assessed by considering expected value time evolution, probability density (pdf) time evolution, and target probability density fidelity. In this section, the fundamental methods for implementing experiments on quantum devices, measuring observables and computing simulation fidelities are discussed. A benchmark system with Hamiltonian

$$\hat{H} = \sum_{i=0}^{1} \left(\hat{X}_i \hat{X}_{i+1} + \hat{Y}_i \hat{Y}_{i+1} + \hat{Z}_i \hat{Z}_{i+1} \right) \tag{3.56}$$

is used for simulation on *ibmq jakarta*. The three proposed schemes discussed on previous sections will be benchmarked by computing the metric mentioned above, for this particular system [2]. As a control case, a Qiskit's QASM simulator is used to perform time evolution, since it emulates fault-tolerant quantum computation. This base case will be used for comparing the results obtained by performing the experiment on actual quantum devices and further analysis and discussion.

3.3.1 Measurement of observables

Consider the set of of Pauli tensor product operators defined on a space of N qubits. Each of this operators can be mapped to the set of strings whose characters indicate the Pauli operator that acts on the corresponding qubit. For instance,

$$X\hat{X}I = \hat{I} \otimes \hat{X} \otimes X, \tag{3.57}$$

indicating that the operator acts on a 3-qubit space, and applies and identity gate on the third qubit, while applying an X gate on the first two. It is a well known fact that the set of Pauli tensor product operators constitute a basis for the space of operators defined on an N-qubit system's Hilbert space [17, 5]. Hence, any operator in that space has the expansion

$$\hat{O} = \sum_{P} c_P \hat{P} \tag{3.58}$$

where P is a Pauli tensor product operator. This fact yields a direct approach towards measuring expected value of operators. If the expected value of Pauli tensors can be measured, then linearity

[2] In principle, a larger system could be simulated. However, direct pulse control using the methods proposed on [8] depends on the connectivity if quantum devices. *ibmq jakarta* only supports this type of control on a set of three qubits. Hence the constraints for comparison between the propsed algorithms.

yields the expected value of any N-qubit operator. Now, by default, experiments performed on IBM's quantum devices produce the expected value of the Pauli tensor $\otimes^N \hat{Z}$. As a result, by applying local rotations to the qubits according to equations 3.24 and 3.25, it is possible to measure any Pauli tensor that also includes $\hat{X}$ and $\hat{Y}$ operators. It might be the case that the measured Pauli tensor only acts on a subspace of the register. This is irrelevant, though, since it is possible to trace out the measured subsystem, thus making any action on the complementary system irrelevant [17]. Hence, measuring all the register at once during each experiment, and applying local rotations, is a valid methodology for recovering any N-qubit expected value.

Measuring the probability density is quite straightforward using Qiskit SDK. As mentioned before, the results of the execution of most quantum algorithms produce a single outcome of measuring the register on the computational basis, which can be mapped to a binary string. Approximate reconstruction of the pdf can be carried out by repeating the experiment several times and building an histogram. This is done by Qiskit methods. Once the pdf is reconstructed, the expected value of the operator $\otimes^N \hat{Z}$ can be reconstructed by examining the parity of the outcome string, i.e., by the formula

$$\langle \otimes^N \hat{Z} \rangle = \sum_{s \in \{0,1\}^N} p(s) \prod_{i \in s} (-1)^i, \tag{3.59}$$

where $p(s)$ denots the probability density of the outcomes, and $(-1)^i$ is the parity of the characters in the string. By applying local rotations, it is possible to rotate any state from the eigenbasis of any Pauli tensor operator to the computational basis. As a result, the procedure yields the expected value of any Pauli tensor. Linearity does the rest.

Readout Error Correction

It must be noted that actual quantum device experience readout errors after measurement. This readout errors are typically modeled like random bit flip channels [17, 5]. Suppose that prior to measurement, a quantum state has an associated pdf p_i of measurement on the computational basis. Due to bit flip noise, the actual measured probability is

$$p'_i = M_{ij} p_j, \tag{3.60}$$

where M_{ij} is the *transition matrix* of the process. To correct readout errors, the transition matrix is measured by preparing the states of the computational basis and performing several measurements in order to get an approximate noisy pdf. Then, the corrected pdf is obtained by inverting the transition matrix as follows:

$$p_i = M^{-1}_{ij} p'_j. \tag{3.61}$$

This procedure is applied any time a pdf is computed from a series of experiments on actual quantum devices. The predefined Qiskit interfaces for this task is used on the implementations.

Measuring probability density fidelity

In this work, probability density fidelity is used as a quantitative measure of the precision of the evolution. Consider two states $|\psi\rangle$ and $|\psi'\rangle$, that yield two probability densities of measurement in the computational basis, p_i and p'_i, respectively. The probability density fidelity of those two distributions is defined [10] by the equation

$$F_{p_i, p'_i} = \Big(\sum_i \sqrt{p_i p'_i}\Big)^2. \tag{3.62}$$

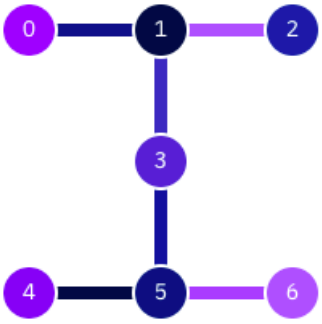
To assess the precision of a time evolution scheme, a target evolution time is fixed, as well as a fixed number of integration steps. The initial state

$$|\psi\rangle = |110\rangle \tag{3.63}$$

is evolved using the evolution scheme (by running experiments on *ibmq jakarta*), thus yielding a pdf (p'_i) as described in this subsection. The exact pdf (p_i) is computed by exact time evolution using numerical diagonalization of the Hamiltonian. The probability density fidelity, for given time and number of integration steps is computed according to equation 3.62.

3.3.2 Device specifications

All experiments are run on *ibmq jakarta*, a backend with a processor Falcon r5.1H, at the time of development of this work. This processor has seven physical qubits. It has a connectivity as depicted on figure 3.6. It supports pulse control as on [8], only on qubits 1, 3, and 5. Like all IBM Quantum devices, its basis set is $\{\hat{S}_x = \sqrt{\hat{X}}, \hat{X}, \hat{R}_{z,\phi}\}$. Those are logical gates, and most of the time, actual execution requires performing swap operations between some qubits due to limited connectivity.



Experiments involving direct transpilation and gate-efficient schemes, are performed using qubits 0, 1, and 2. While experiments involving pulse-efficient schemes are performed on qubits 1, 3, and 5. Quantum devices are calibrated each day, and may vary. However, typical parameter values are reported on table 3.1.

T_1	130 μs
T_2	34 μs
ϵ_{CNOT}	1%
ϵ_{Read}	2%

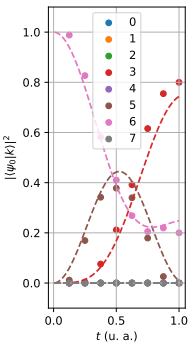
3.3.3 Control case: QASM Simulation Results

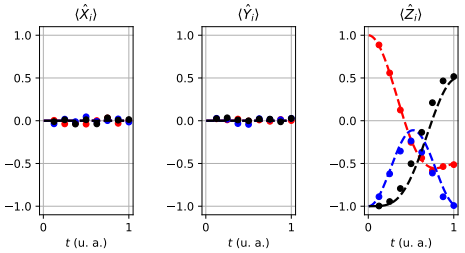
To establish a control case for evaluating results of quantum evolution on real devices, the benchmark system is simulated over a time $t = 1$ using an integration step $dt = 1/8$. In a more quantitative test, the probability density fidelity is computed for times $t = 1,2,3$, with up to 8 integration steps. Units of time are normalized to $\hbar = 1$, so that energy has units of inverse time. These results represent the outcome of executing the direct evolution scheme introduced on section 3.2, on a fault tolerant device. In figures 3.7a and 3.7b, the results of evolution with direct diagonalization of the Hamiltonian are presented as dotted lines.

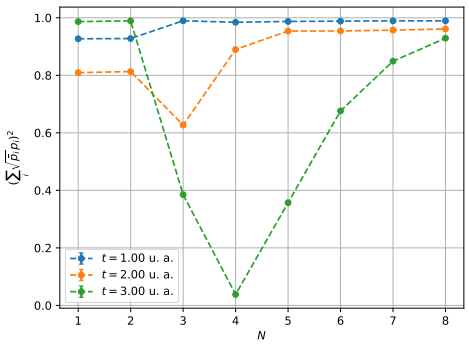
Probability density of measuring a 3-bit string is evolved over time. The convention is that the string is represented by its associated number in decimal system. Expected values of single qubit Pauli operators $\hat{X}_i, \hat{Y}_i, \hat{Z}_i$ are measured for each of the qubits. The first qubit observables correspond to color red data, the second qubit's, to blue data, and the third qubit's, to black data. This convention holds on all graphics of this type. Probaility density fidelity es computed for several number of integration steps N (up to eight), and each point correspond to the average of five sets of 2048 experiment repetitions.

A small number of integration steps is used since the finite coherence times of the qubits in the device limit the depth of the circuit that can be executed on real devices. If the gate execution times

approach the coherence times, the state coherence is irreversibly lost, and thus the final state does not resemble at all the target final state. As may be seen on figure 3.7c, there is a sort of transient stage during which the Trotter approximation does not converge. As seen before, there exists a critical integration time step Δt, below which the state fidelity has a predictable approximation error. It may be also noted that high fidelities can be achieved by using a relatively small number of integration steps. Furthermore, observables resemble quite well the theoretical values. This indicates that as long as the integration time step $\Delta t = t/N$ is below the convergence threshold, the trotter error can be controlled.







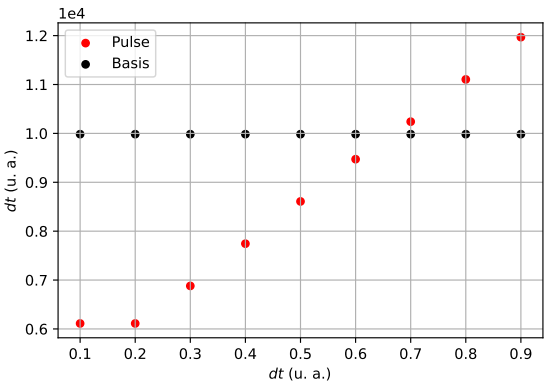
3.4 Comparison and Benchmark: Results

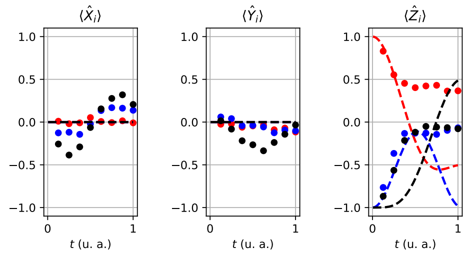
Now that the main metrics, and the way they are measured using *ibmq jakarta*, results of simulation using the schemes proposed on section 3.2 are presented. Results are compared to fault-tolerant simulation outcome in figure 3.7.

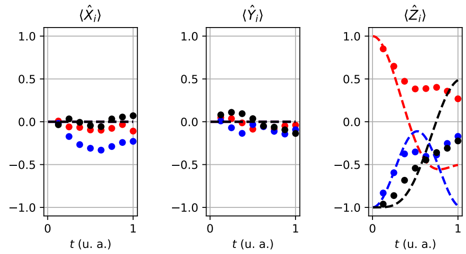
Expected values of single qubit observables is measured following conventions established on section 3.3.3. From figure 3.9, it can be seen that the scheme that produces the largest resemblance to the ideal values (dotted lines) is the pulse efficient implementation, followed by the basis efficient and the direct transpilation. Qualitatively, the difference in performance is overwhelming. While the base efficient and direct transpilation approaches struggle to converge for four steps, the pulse efficient scheme is able to converge surprisingly well for the entire set of eight steps. Since the calibration of entangling gates is the same for all cases (except for a parameter scaling on pulse efficient experiments) it can be inferred that the use of shorter entangling pulses, which improves the usage of the finite coherence times of the system, is the most determinant factor when simulating this kind of Hamiltonians.

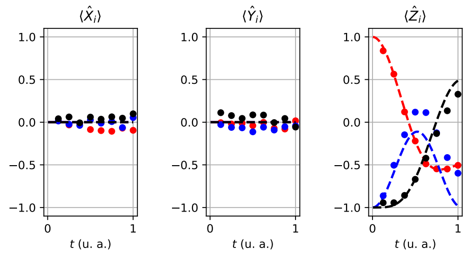
Those features can be seen more clearly on figure 3.10. From the ideal results on figure 3.7, it can be seen that the probability density only presents oscillations for states $|3\rangle$, $|6\rangle$, and $|5\rangle$. Figures 3.10a and 3.10b suggest that the state coherence is lost very fast (qubits depolarize fairly quickly), and the probability density fidelity decays very rapidly as the circuit depth of simulation increases, using both direct and basis efficient schemes. On the other hand, pulse efficient schemes seem to better maintain state coherence since the execution time of each trotter step is shorter. As a result, the state fidelity is larger.

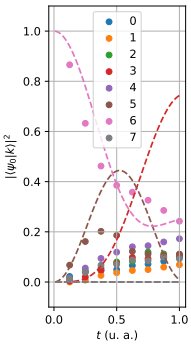
On figure 3.11, this observations are made more quantitative. After the transient stage in which the Trotter approximation does not converge in general, it can be seen that the probability density fidelity decays consistently for direct transpilation and basis efficient schemes, as the number of integration steps increase. On the other hand, pulse efficient schemes have an increasing fidelity with the number of integration steps. While the first two schemes produce fidelities near 30% - 40%, direct pulse control yields fidelities near 70% in some cases, having actual execution times about 40% shorter. There is a caveat, though. As the integration time step increases, the benefits of pulse schemes is reduced, since the actual execution time increases proportionally (see figure 3.8). This can be seen on figure 3.11c. As the target simulation time increases, the fidelity remains below 50%. Since the circuit depth does not change with respect to shorter times, it can be concluded that this reduction in the fidelity is due to the larger execution time of the experiments (see figure 3.8). In consequence, simulating anisotropic Heisenberg Hamiltonians on quantum computers imply optimizing the tradeoff between short Trotter steps with pulse control, and a large number of repetitions required to simulate large times. As may be inferred from figure 3.2, it must be procured that the integration time step Δt is below the threshold for predictable Trotter error control, though.

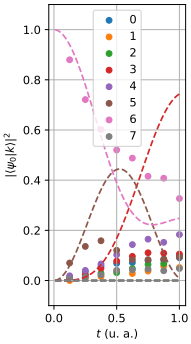


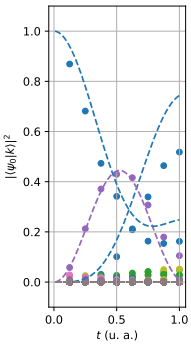

⟨X̂_i⟩
⟨Ŷ_i⟩
⟨Ẑ_i⟩
t (u. a.)
t (u. a.)
t (u. a.)

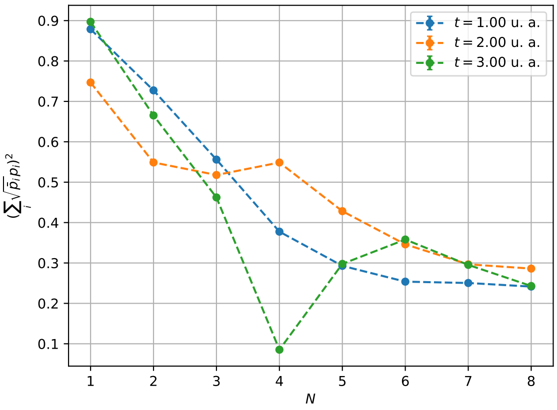

⟨X̂_i⟩
⟨Ŷ_i⟩
⟨Ẑ_i⟩
t (u. a.)
t (u. a.)
t (u. a.)

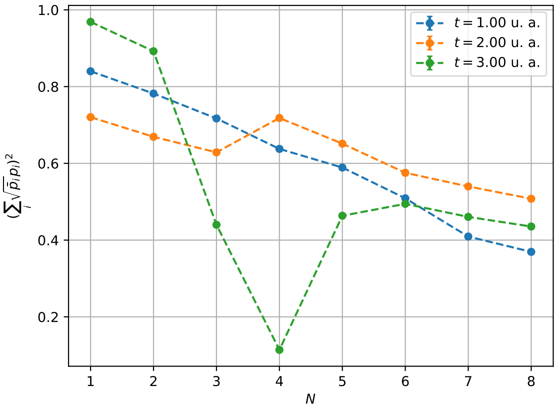

⟨X̂_i⟩
⟨Ŷ_i⟩
⟨Ẑ_i⟩
t (u. a.)
t (u. a.)
t (u. a.)

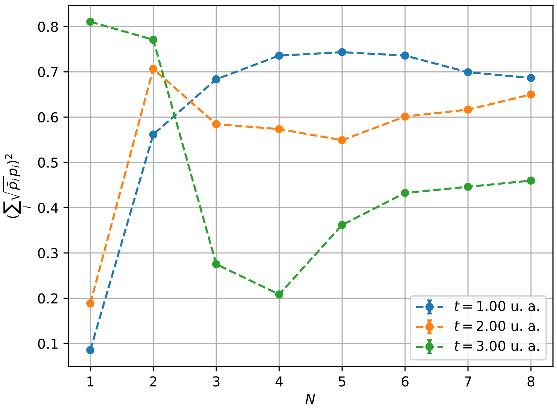

0
1
2
3
4
5
6
7
1.0
0.8
0.6
0.4
0.2
0.0
$|\langle\psi_0|k\rangle|^2$
0.0
0.5
1.0
t (u. a.)


0
1
2
3
4
5
6
7
1.0
0.8
0.6
0.4
0.2
0.0
$|\langle\psi_0|k\rangle|^2$
0.0
0.5
1.0
t (u. a.)


1.0
0.8
0.6
0.4
0.2
0.0
$|\langle\psi_0|k\rangle|^2$
0.0
0.5
1.0
t (u. a.)


t = 1.00 u. a.
t = 2.00 u. a.
t = 3.00 u. a.
N


t = 1.00 u. a.
t = 2.00 u. a.
t = 3.00 u. a.
N


t = 1.00 u. a.
t = 2.00 u. a.
t = 3.00 u. a.
N

Chapter 4

Concluding remarks

In this work, quantum algorithms based on Trotter schemes have been discussed and implemented on real quantum devices. Results from previous work [25] have been rediscovered and used for time simulation of anisotropic Heisenberg Hamiltonians. Also, pulse efficient transpilation, following methods in [8], has been implemented for the same purpose. Simulation results suggest that the best technique for simulating spin chains on real quantum devices is cross resonance pulse scaling. It has been shown that the main leverage of this method with respect to basis efficient transpilation is the reduced execution time of the control sequences on IBM Quantum devices. Since this method is quite hardware dependent, it is conjectured, though, that the basis efficient scheme discussed in this work might be a good alternative for simulating spin-1/2 Hamiltonians on CNOT-based quantum hardware. Since it has been shown by [25] that this transpilation is optimal for two-qubit interaction, the transpilation discussed in section 3.2.2 may be optimal for the Trotterization scheme used, on this type of hardware. It has also been shown that simulation of quantum systems using Trotterization requires balancing a tradeoff between trotter error and noise error due to decoherence. The integration time step Δt must be chosen wisely so that the system evolves in the controlled error regime (fig. 3.2), but also the execution time of the algorithm is short enough to make efficient use of the finite coherence time of quantum devices. Although the algorithms seem to reproduce the physics of the system (for instance, expected value of observables) reasonably well for small times, faithful simulation of quantum systems using current devices seems to be impractical.

Although time simulation of quantum systems is impractical on current quantum devices, it has a wide range of applications to quantum computing. As a concluding remark, an application of quantum simulation to study ground state properties of Hamiltonians is discussed. For instance, the ground state energy of a quantum Hamiltonian may be computed using the adiabatic theorem [19]. Consider time independent Hamiltonians $\hat{H}_0$ and $\hat{H}_m$, and an initial state $|\psi_m\rangle$ such that

$$\hat{H}_m |\psi_m\rangle = \epsilon_m |\psi_m\rangle , \tag{4.1}$$

defined on a Hilbert space. Now, evolve a system in that initial state, under time-dependent Hamiltonian

$$\hat{\mathcal{H}}(t) = \left(1 - \frac{t}{T}\right)\hat{H}_m + \frac{t}{T}\hat{H}_0, \tag{4.2}$$

from $t = 0$ to $t = T$. In the limit $T \to \infty$, the adiabatic theorem states that the final state of the system, $|\psi\rangle_0$, is such that

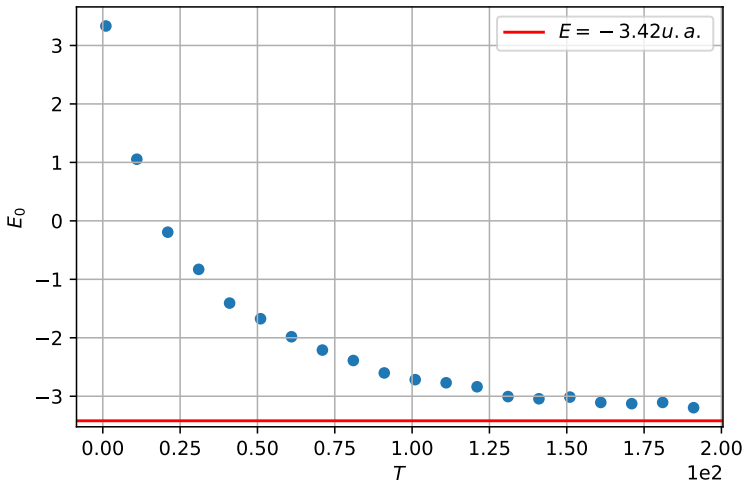
$$\hat{H}_0 |\psi_0\rangle = \epsilon_0 |\psi_0\rangle , \tag{4.3}$$

That is, the final state is a corresponding eigenstate of $\hat{H}_0$. In particular, if the initial state 4.3 is a ground state of the Hamiltonian $\hat{H}_m$, the final state is also a ground state of the Hamiltonian $\hat{H}_0$. By choosing a particularly simple Hamiltonian $\hat{H}_m$, whose ground state is easy to prepare on a

quantum device, this type of evolution can be used to compute the ground state of a non trivial Hamiltonian. As an illustration, ground state energy of the three-qubit Hamiltonian

$$\hat{H}_0 = \frac{1}{2}\sum_{i=0}^{1}\left(\hat{X}_i\hat{X}_{i+1} + 2\hat{Y}_i\hat{Y}_{i+1} + \frac{1}{2}\hat{Z}_i\hat{Z}_{i+1}\right) + \sum_{i=0}^{2}\hat{Z}_i \tag{4.4}$$

is computed. In figure 4.1, the expected value of $\hat{H}_0$ after evolution for $t = T$ is plotted as a function of T. However, as may be seen, the evolution needs to be quite slow for convergence. This prevents practical usage of this tool on current quantum devices as seen in the present work. Albeit interesting, more work needs to be done from the area of quantum control to reduce gate execution times further, and improve quantum coherence of current quantum devices, so that applications of time simulation, such as the one briefly introduced here, can be implemented on actual quantum devices.



Appendices

Appendix A

Basic implementation of circuits with Qiskit

The simulations were carried out using Qiskit SDK, whose main interface can be found on [2]. The main structure is a *Heisenberg graph* object that may be instantiated as follows

```
testGraph = HeisenbergGraph(
    spinInteractions={
        (0, 1): [1, 1, 1],
    },
    externalField={
        0: [0.0, 0.0, 0.0],
        1: [0.0, 0.0, 0.0],
    },
    localSimulation=False,
    backendName='ibmq_jakarta',
    noisySimulation=False,
    initialState=1/np.sqrt(2) * np.array([1, 0, 1, 0])
)
```

The *spinInteractions* parameter describes a graph (in the form of a python dictionary), indexed by tuples that represent the index of the interacting spins; and whose value is an array with the values of the parameters in the two-spin interaction Hamiltonian. The *externalField* parameter corresponds to the single qubit Hamiltonian parameters (in the form of a dictionary), indexed by integers that represent qubit indices; and whose value is an array with the parameters of the particular single-qubit Hamiltonian. The other parameters are somewhat self explanatory. All experiments are carried out using this class' functions. Those include computation of expected values, pdf evolution, readout error mitigation, etc. The source code can be found on this GitHub repository.

From this parent class, three child classes are derived, that only differ on how the two-qubit interaction is implemented, and the experiments, executed. Those are

1. *HeisenbergGraph*: For basis efficient scheme.
2. *DirectSpinGraph*: For direct transpilation scheme.
3. *PulseSpinGraph*: For pulse efficient scheme.

Two spin interaction is implemented as a circuit on a method *edgeCircuit*, which is overloaded on each class. Experiment execution is implemented on a method *execute* that is also overloaded. Those are the only difference between the classes. Other methods are the same, since they relate to auxiliary function for data processing or plotting. All optimizations are set to minimum (*optimization level zero*). In the case of pulse implementations the *RZXCalibrationBuilderNoEcho* pass manager is used to use CNOT calibration for cross resonance (*rzx*) gate implementation as described on the main text. On qubit Hamiltonian evolution is performed the same on all classes, as follows

```
def vertexCircuit(self, vertex, spinChain):
    '''
    Function for building circuit that
    performs vertex Hamiltonian evolution
    '''
    Hx = vertex['externalField'][0]
    Hy = vertex['externalField'][1]
    Hz = vertex['externalField'][2]
    H = np.sqrt(Hx**2 + Hy**2 + Hz**2)
    # Parameter values for Qiskit
    PHI = np.arctan2(Hy, Hx) + 2*np.pi if H > 0 else 0
    THETA = np.arccos(Hz/H) if H > 0 else 0
    LAMBDA = np.pi if H > 0 else 0
    # Align to field main axis
    qcVertex = QuantumCircuit(spinChain)
    qcVertex.u(-THETA, -LAMBDA, -PHI, spinChain[vertex.index])
    qcVertex.rz(vertex['paramExternalField'], spinChain[vertex.index])
    qcVertex.u(THETA, PHI, LAMBDA, spinChain[vertex.index])
    return qcVertex
```

where the u gate corresponds to the gates discussed on equation 2.64.

A.0.1 *HeisenbergGraph* implementation

```
    def edgeCircuit(self, edge, spinChain):
        '''
        Function for building circuit that
        performs edge Hamiltonian evolution
        '''
        start, end = edge.tuple
        J = edge['paramExchangeIntegrals']
        qcEdge = QuantumCircuit(spinChain)

        # See thesis document for more information
        # on the nature of the implementation

        # Go to computational basis
        qcEdge.cx(spinChain[start], spinChain[end])
        qcEdge.h(spinChain[start])
        # Append x and z phases
        qcEdge.rz(J[0], spinChain[start])
        qcEdge.rz(J[2], spinChain[end])
        # Shuffle and append y phase
        qcEdge.cx(spinChain[start], spinChain[end])
        qcEdge.rz(-J[1], spinChain[end])
        # Return to ordered Bell basis
        qcEdge.cx(spinChain[start], spinChain[end])
        qcEdge.h(spinChain[start])
        qcEdge.cx(spinChain[start], spinChain[end])

    def execute(self, circuits, backend, shots=2048):
        '''
        Function for executing the
        experiments according to
        backend
        '''
        return execute(circuits, backend, shots=shots, optimization_level=0)
```

A.0.2 *DirectSpinGraph* implementation

```
def edgeCircuit(self, edge, spinChain):
    '''
    Function for building circuit that
    performs edge Hamiltonian evolution
    '''
    start, end = edge.tuple
    J = edge['paramExchangeIntegrals']
    qcEdge = QuantumCircuit(spinChain)
    # Compute J1 phase
    qcEdge.h([spinChain[start], spinChain[end]])
    qcEdge.cx(spinChain[start], spinChain[end])
    qcEdge.rz(J[0], spinChain[end])
    qcEdge.cx(spinChain[start], spinChain[end])
    qcEdge.h([spinChain[start], spinChain[end]])
    # Compute J3 phase
    qcEdge.sdg([spinChain[start], spinChain[end]])
    qcEdge.h([spinChain[start], spinChain[end]])
    qcEdge.cx(spinChain[start], spinChain[end])
    qcEdge.rz(J[1], spinChain[end])
    qcEdge.cx(spinChain[start], spinChain[end])
    qcEdge.h([spinChain[start], spinChain[end]])
    qcEdge.s([spinChain[start], spinChain[end]])
    # Compute J3 phase
    qcEdge.cx(spinChain[start], spinChain[end])
    qcEdge.rz(J[2], spinChain[end])
    qcEdge.cx(spinChain[start], spinChain[end])
    return qcEdge

def execute(self, circuits, backend, shots=2048):
    '''
    Function for executing the
    experiments according to
    backend
    '''
    return execute(circuits, backend, shots=shots, optimization_level=0)
```

A.0.3 *PulseSpinGraph* implementation

```
def edgeCircuit(self, edge, spinChain):
    '''
    Function for building circuit that
    performs edge Hamiltonian evolution
    '''
    start, end = edge.tuple
    try:
        dt = kwargs['dt']
        J = 2*dt*np.array(edge['exchangeIntegrals'])
    except KeyError:
        J = edge['paramExchangeIntegrals']
    qcEdge = QuantumCircuit(spinChain)
    if np.abs(edge['exchangeIntegrals'][0]) > 1e-3:
        # Compute J0 phase
        # Rotate start qubit to X
        qcEdge.h(spinChain[start])
        # Duplicate pulse to cancel
        # undesired terms on CR
        qcEdge.barrier()
```

```
            qcEdge.rzx(J[0]/2, spinChain[start], spinChain[end])
            qcEdge.x(spinChain[start])
            qcEdge.rzx(-J[0]/2, spinChain[start], spinChain[end])
            qcEdge.x(spinChain[start])
            qcEdge.barrier()
            # Rotate start qubit to X
            qcEdge.h(spinChain[start])
        if np.abs(edge['exchangeIntegrals'][1]) > 1e-3:
            # Compute J1 phase
            # Rotate start qubit to X -> Y
            qcEdge.sdg(spinChain[start])
            qcEdge.h(spinChain[start])
            # Rotate end qubit to Y
            qcEdge.sdg(spinChain[end])
            # Duplicate pulse to cancel
            # undesired terms on CR
            qcEdge.barrier()
            qcEdge.rzx(J[1]/2, spinChain[start], spinChain[end])
            qcEdge.x(spinChain[start])
            qcEdge.rzx(-J[1]/2, spinChain[start], spinChain[end])
            qcEdge.x(spinChain[start])
            qcEdge.barrier()
            # Rotate start qubit to X
            qcEdge.h(spinChain[start])
            qcEdge.s(spinChain[start])
            # Rotate end qubit to Y
            qcEdge.s(spinChain[end])
        if np.abs(edge['exchangeIntegrals'][2]) > 1e-3:
            # Compute J2 phase
            # Rotate end qubit to Z
            qcEdge.h(spinChain[end])
            # Duplicate pulse to cancel
            # undesired terms on CR
            qcEdge.barrier()
            qcEdge.rzx(J[2]/2, spinChain[start], spinChain[end])
            qcEdge.x(spinChain[start])
            qcEdge.rzx(-J[2]/2, spinChain[start], spinChain[end])
            qcEdge.x(spinChain[start])
            qcEdge.barrier()
            # Rotate end qubit to Z
            qcEdge.h(spinChain[end])
        return qcEdge.to_instruction()

    def execute(self, circuits, backend, shots=2048):
        '''
        Function for executing a
        pulse schedule representing
        the algorithm
        '''
        if self.localSimulation:
            return execute(circuits, backend, shots=2048, optimization_level=0)
        else:
            transpiledCircuits = transpile(
                circuits,
                basis_gates=['x', 'sx', 'rz', 'rzx'],
                optimization_level=1
            )
            pm = PassManager([RZXCalibrationBuilderNoEcho(backend)])
            passCircuits = pm.run(transpiledCircuits)
```

```
    return execute(
        passCircuits,
        backend=backend,
        shots=shots,
    )
```

Appendix B

Jupyter notebook for analysis

B.1 Algorithm Benchmarking with QASM Simulations

The purpose of this notebook is to evaluate the performance of the implemented quantum time evolution algorithm. The common evaluated criteria are circuit depth and state fidelity. For QASM simulations, the metrics to be evaluated are

1. **Circuit depth**
2. **Trace distance**

In this notebooks, data is generated using Qiskit, and analyzed to find an expected relation between *evolution time*, *expected fidelity* and *number of integration steps*. So first, It is wise to start with a little summary about the general results of quantum time evolution as presented by [1], then proceed to demonstrate how to use the devised routines for simulating a generic spin graph, and finally, produce some plots that show the actual performance of the programmed algorithm.

B.1.1 General Quantum Time Simulation Theory

In this project, a second order Trotter scheme is proposed to estimate evolution of a spin system. Consider a Hamiltonian that can be written as the sum of *local Hamiltonians* (acting on a small subsystem at a time).

$$\hat{H} = \sum_{i=1}^{l} \hat{H}_i$$

By direct computation, it is possible to show that

$$\mathrm{e}^{-\mathrm{i}t\hat{H}} = \prod_{i=1}^{l} \mathrm{e}^{-\mathrm{i}t\hat{H}_i} + \mathcal{O}(t^2)$$

Now, if evolution is carried out over a small enough period of time, then the difference between the right hand side and the left hand side is small. This leads to an integration scheme that relies on the implementation of local evolution operators, that is capable of simulate complex correlation in a quantum system, as long as it arises from small-subsystem interactions.

Efficiency of Quantum Time Simulation Algorithms

Berry et. atl. [1] have already considered the general error theory of Suzuki-Trotter schemes, and te minimal amount of resources required to simulate a quantum system. First, let's discuss the expected number of operations required to obtain a desired error bound, for a given simulation time, and then, the minimal amount of resources required to simulate a given quantum system.

In their paper, Berry et. al. demonstrate that the expected number of steps (as measured by the number of exponentials required to implement a k-th order Suzuki-Trotter scheme) is bounded by a power law

$$N \leq A_k \frac{t^{1+1/2k}}{\epsilon^{1/2k}}$$

It is interesting, as they point out, that there is an optimal order for the integration scheme. However, the main takeaway here, at least by now, is that

> There exists a power law that relates the number of steps, integration time and error bound, or a given quantum system.

And so, this is what will be sought when benchmarking the proposed algorithm.

Finally, Berry et. al. point out that any Hamiltonian (at least of practical interest) may be simulated by a quantum algorithm using at least a linear amount of steps. Now, the remarkable part, at least for the purpose of this project is that

> If a simulation algorithm uses an amount of steps that scales linearly with time for given precision, then it is almost optimal. At least within a constant factor that can be optimized.

B.1.2 Structure of the Notebook

This notebook is structured as follows. First, results obtained by Salathé et. al. and Las Heras et. al. are reproduced to show the correctness of the algorithms proposed. Simulations are carried out on `ibmq_jakarta`, and measurement error mitigation is implemented as well. After that, basis efficient and pulse efficient implementation of the routines are compared in terms of: 1) expected value of single qubit observables and 2) pdf evolution. Then pulse schedules are revised as well. Also, an study of the inherent limitations of Trotterization is carried out. Finally, an annihilation process s carried out to illustrate applications of quantum time evolution.

Let's start by importing the necessary packages

[1]:
```
from PyHeisenberg import HeisenbergGraph, DataAnalyzer, PulseSpinGraph,␣
↪DirectSpinGraph
import numpy as np
import matplotlib.pyplot as plt
import warnings
from operator import itemgetter
warnings.filterwarnings('ignore')
plt.style.use('FigureStyle.mplstyle')
```

B.1.3 Reproduction of results obtained by Salathé et. al.

In this section, a two-spin isotropic Heisenberg model is simulated using the networks proposed in the main document. This Hamiltonian corresponds to

$$\hat{H} = J(\hat{X}_1\hat{X}_2 + \hat{Y}_1\hat{Y}_2 + \hat{Z}_1\hat{Z}_2)$$

Following the group's work, the initial state

$$|\psi_0\rangle = |+\rangle \otimes |0\rangle$$

Is evolved for a time $t = 3\pi/4$, when $J = 1$. Time series of the expected values of single-qubit Pauli operators are generated and compared to results obtained by direct diagonalization of the Hamiltonian. The time interval is partitioned on 12 equally spaced intervals.

Definition of the Graph Object

As mentioned on the main text, the Hamiltonian gives rise to a graph that is used to store the information of the spin-spin and field-spin interactions. More parameters such as the execution backend, initial state and noise activation are included as well.

[10]:
```
testGraph = HeisenbergGraph(
    spinInteractions={
        (0, 1): [1, 1, 1],
    },
    externalField={
        0: [0.0, 0.0, 0.0],
        1: [0.0, 0.0, 0.0],
    },
    localSimulation=False,
    backendName='ibmq_jakarta',
    noisySimulation=False,
    initialState=1/np.sqrt(2) * np.array([1, 0, 1, 0])
)
testAnalyzer = DataAnalyzer(spinGraph=testGraph)
```

Computation of Exact Evolution Time Series

These correspond to direct diagonalization using Numpy. By default, 200 steps are used for producing a somewhat smooth theoretical curve of the evolution of expected values of a Pauli string operator.

[22]:
```
pauliStrings = ['XI', 'IX', 'YI', 'IY', 'ZI', 'IZ']
timesEx, resultSeriesExact = testGraph.exactPauliExpValSeries(
    pauliStrings,
    t = 3*np.pi/4
)
ExX1, ExX2, ExY1, ExY2, ExZ1, ExZ2 = itemgetter(*pauliStrings)(resultSeriesExact)
```

Computation of Experimental Evolution Time Series

These correspond to quantum time simulation using algorithms descried on the main text. By default, 200 steps are used for producing a somewhat smooth experimental curve of the evolution of expected values of a Pauli string operator. Due to decoherence and gate errors, it is suggested that no more than 20 steps for simulating time evolution on real quantum devices.

To mitigate readout errors, a special technique is used. It uses a markovian model for correcting possible bit filp errors on readout. It is implemented on this notebook using `qiskit.ignis`.

Measurement error mitigation To apply measurement error mitigation, generate a sequence of quantum circuits to measure the readout errors when preparing states of the computational basis. Then fit an evolution matrix for assessing the transition probabilities, and define a filter for correction results of future algorithm's execution. This is done by `qiskit.ignis`.

[8]:
```
measurementFitter = testGraph.getCalibrationFitter()
```

```
Job Status: job has successfully run
```

Computation of Time Series on IBM Quantum Due to current limitations of cloud-based quantum computing, the `HeisenbergGraph` class optimizes the process of running quantum circuits on real quantum deices. It is recommended that the following routine be used for computing time series of expected value of Pauli string observables. The function returns `ndarrays`, which can be linearly combined to compute time series of generic quantum operators.

[19]:
```
times, resultSeries = testGraph.pauliExpValSeries(
    pauliStrings,
    MAX_STEPS = 12,
    t = 3*np.pi/4,
    measurementFitter= measurementFitter
)
X1, X2, Y1, Y2, Z1, Z2 = itemgetter(*pauliStrings)(resultSeries)
```

```
Job Status: job has successfully run
```

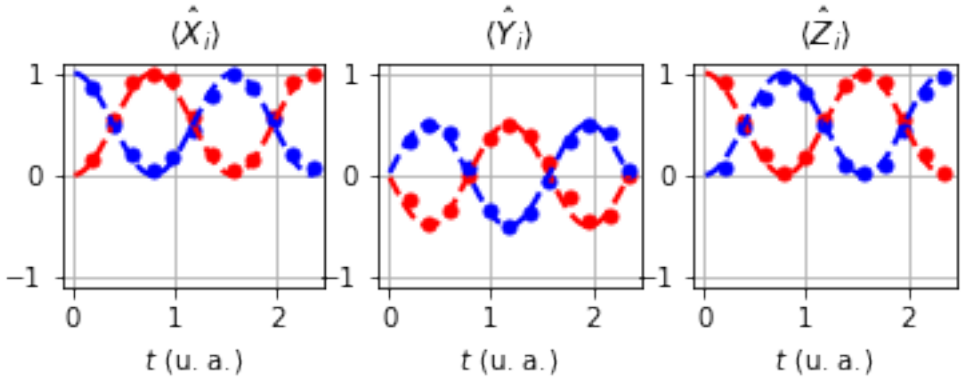
What follows is to plot the experimental results. A comparative plot of theoretical values an experimental values is presented on this cell. Since time evolution of this system is exact, the experimental points (scatter plots) resemble quite well the theoretical curves (dashed lines). Red datasets corresponds to the second spin's observables, whereas the blu datasets corresponds to the first spin's observables.

Important: During simulations, the specifications of the QPU were

T_1	T_2	CNOT err.	Readout err.
103.34 us	124.79 us	1.931e-2	3.070e-2

Measurement error mitigation is optional. However, it is highly recommended. Since `Qiskit Runtime` is still on beta, this project doesn't rely on that architecture.

[20]:
```
fig, (axX, axY, axZ) = plt.subplots(1,3)
axX.plot(times, X1, 'ro', times, X2, 'bo')
axX.plot(timesEx, ExX1, color='red', linewidth=2, linestyle='dashed')
axX.plot(timesEx, ExX2, color='blue', linewidth=2, linestyle='dashed')
axX.set_title(r"$\langle\hat{X}_i\rangle$")
axX.set_xlabel(r"$t$ (u. a.)")
axX.set_aspect('equal')
axX.set_ylim([-1.1, 1.1])
axY.plot(times, Y1, 'ro', times, Y2, 'bo')
axY.plot(timesEx, ExY1, color='red', linewidth=2, linestyle='dashed')
axY.plot(timesEx, ExY2, color='blue', linewidth=2, linestyle='dashed')
axY.set_title(r"$\langle\hat{Y}_i\rangle$")
axY.set_xlabel(r"$t$ (u. a.)")
axY.set_aspect('equal')
axY.set_ylim([-1.1, 1.1])
axZ.plot(times, Z1, 'ro', times, Z2, 'bo')
axZ.plot(timesEx, ExZ1, color='red', linewidth=2, linestyle='dashed')
axZ.plot(timesEx, ExZ2, color='blue', linewidth=2, linestyle='dashed')
axZ.set_title(r"$\langle\hat{Z}_i\rangle$")
axZ.set_xlabel(r"$t$ (u. a.)")
axZ.set_aspect('equal')
axZ.set_ylim([-1.1, 1.1])
fig.savefig('XYZModel_HerasEtAl.pdf')
plt.show()
```



B.1.4 Reproduction of results of Las Heras et. al. and Salathé et. al.

In this section, the two-spin transverse field Ising model,

$$\hat{H} = \frac{V}{2}(\hat{X}_1\hat{X}_2 + \hat{Y}_1\hat{Y}_2 + \hat{Z}_1\hat{Z}_2) + \frac{U}{4}(\hat{Z}_1 + \hat{Z}_2)$$

with $V = U = 1$, is simulated over a time $\tau = 5$. The initial state is set to

$$|\psi_0\rangle = |+\rangle \otimes |0\rangle$$

Simulations are carried out using `ibmq_quito`, and error correction. First, a graph object is instantiated with the desired parameter values ($J = 1$). Spin interactions are defined as a dictionary with a tuple key that points the index of the involved sites, and a value that has the interaction parameters as defined on the main document. Filed interactions are described y a dictionary whose key is the site index, and its value is an iterable that has the cartesian components of the local field, as defined on the main text.

A comparative evolution plot is generated using the `dataAnalyzer` object. The number of Trotter steps is taken to be $N = 8$. However, this can be changed at will.

[2]:
```
isingGraph = HeisenbergGraph(
    spinInteractions={
        (0, 1): [0.5, 0.5, 0.5],
    },
    externalField={
        0: [0.0, 0.0, 0.25],
        1: [0.0, 0.0, 0.25],
    },
    localSimulation=False,
    backendName='ibmq_jakarta',
    noisySimulation=False,
    initialState=1/np.sqrt(2) * np.array([1, 0, 1, 0])
)
isingAnalyzer = DataAnalyzer(spinGraph=isingGraph)
```

Evolution of occupation number

Remember that each computational basis state can be mapped to an integer. This integer is shown on the plot legends. An excited fermionic mode is mapped to the ground state of a qubit $|1\rangle$, and the ground fermionic mode to the excited state of a qubit $|0\rangle$. Simulations were carried out on `ibmq_bogota`. Notice that, in this case, the occupation numbers are not so close to the theoretical

values. This is due to many sources of errors. This will be addressed bellow. By now, those are stated

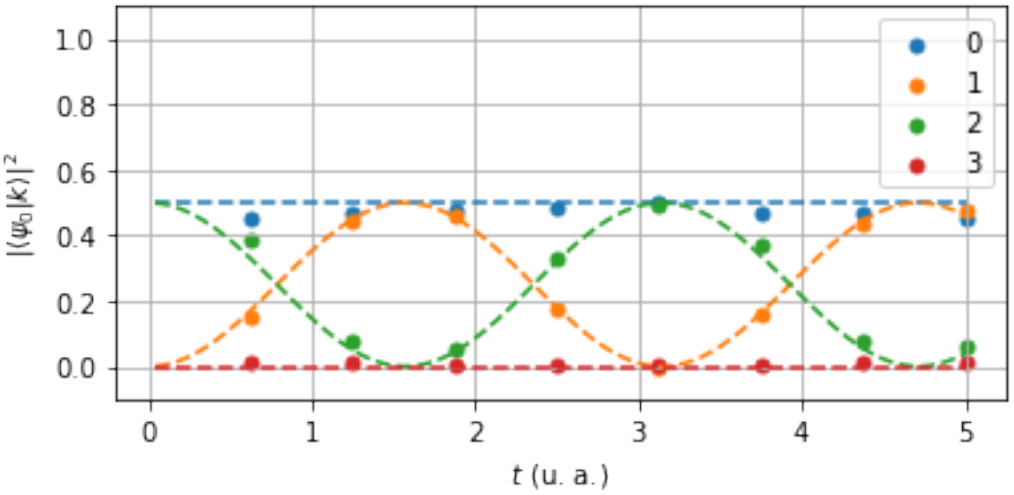
1. Readout error
2. Gate error (CNOT and $\pi/2$-pulses)
3. Qubit relaxation
4. Decoherence nd thermal relaxation

Important: During simulations, the specifications of the QPU were

T_1	T_2	CNOT err.	Readout err.
103.34 us	124.79 us	1.931e-2	3.070e-2

[9]:
```
isingAnalyzer.comparativeEvolution(
    STEPS=8,
    t=5.0,
    measurementFitter=measurementFitter,
    figureFile='HubbardTwoMode_LasHeras.pdf'
)
```

```
Job Status: job has successfully run
```



B.1.5 Simulation of a 3-Spin XXX model

As another way to demonstrate the utility of the trotterization scheme proposed, a three spin Hamiltonian

$$\hat{H} = \sum_{i=0}^{1} \left(\hat{X}^{(i)}\hat{X}^{(i+1)} + \hat{Y}^{(i)}\hat{Y}^{(i+1)} + \hat{Z}^{(i)}\hat{Z}^{(i+1)} \right)$$

Which is a generalization of the Hamiltonian simulated by Salathé et. al. Pretty much the same observables are measured. As before, a graph is instantiated, with the desired two-qubit interactions. SIMulations are carried out on `ibmq_jakarta`.

[6]:
```
xGraph = HeisenbergGraph(
    spinInteractions={
        (0, 1): [1, 1, 1],
        (1, 2): [1, 1, 1],
```

```
    },
    externalField={
        0: [0.0, 0.0, 0.0],
        1: [0.0, 0.0, 0.0],
        2: [0.0, 0.0, 0.0],
    },
    localSimulation=True,
    backendName='qasm_simulator',
    noisySimulation=False,
    initializeList=[1, 2]
)
xAnalyzer = DataAnalyzer(spinGraph=xGraph)
```

After, the exact evolution of the expected value of single qubit Pauli operators is computed

[3]:
```
pauliStrings = [
    'XII', 'IXI', 'IIX',
    'YII', 'IYI', 'IIY',
    'ZII', 'IZI', 'IIZ',
]
timesEx, resultSeriesExact = xGraph.exactPauliExpValSeries(
    pauliStrings,
    t=1
)
ExX1, ExX2, ExX3, ExY1, ExY2, ExY3, ExZ1, ExZ2, ExZ3 = itemgetter(
    *pauliStrings)(resultSeriesExact)
```

And a measurement error mitigation fitter is devised suited to the conditions of the backend.

[4]:
```
measurementFitter = xGraph.getCalibrationFitter()
```

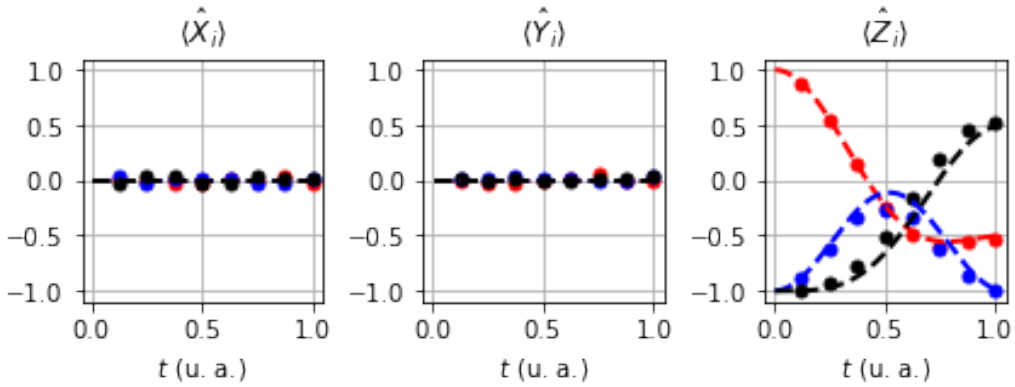
```
Job Status: job has successfully run
```

Finally, experimental data is captured on IBM Quantum's backend.

[5]:
```
times, resultSeries = xGraph.pauliExpValSeries(
    pauliStrings,
    MAX_STEPS=8,
    t=1,
    measurementFitter=measurementFitter
)
X1, X2, X3, Y1, Y2, Y3, Z1, Z2, Z3 = itemgetter(*pauliStrings)(resultSeries)
```

[6]:
```
fig, (axX, axY, axZ) = plt.subplots(1, 3)
fig.tight_layout(pad=1.0)
axX.plot(times, X1, 'ro', times, X2, 'bo', times, X3, 'ko')
axX.plot(timesEx, ExX1, color='red', linewidth=2, linestyle='dashed')
axX.plot(timesEx, ExX2, color='blue', linewidth=2, linestyle='dashed')
axX.plot(timesEx, ExX3, color='black', linewidth=2, linestyle='dashed')
axX.set_title(r"$\langle\hat{X}_i\rangle$")
axX.set_xlabel(r"$t$ (u. a.)")
axX.set_aspect(aspect=0.5)
axX.set_ylim([-1.1, 1.1])
axY.plot(times, Y1, 'ro', times, Y2, 'bo', times, Y3, 'ko')
axY.plot(timesEx, ExY1, color='red', linewidth=2, linestyle='dashed')
axY.plot(timesEx, ExY2, color='blue', linewidth=2, linestyle='dashed')
axY.plot(timesEx, ExY3, color='black', linewidth=2, linestyle='dashed')
axY.set_title(r"$\langle\hat{Y}_i\rangle$")
axY.set_xlabel(r"$t$ (u. a.)")
```

```
axY.set_aspect(aspect=0.5)
axY.set_ylim([-1.1, 1.1])
axZ.plot(times, Z1, 'ro', times, Z2, 'bo', times, Z3, 'ko')
axZ.plot(timesEx, ExZ1, color='red', linewidth=2, linestyle='dashed')
axZ.plot(timesEx, ExZ2, color='blue', linewidth=2, linestyle='dashed')
axZ.plot(timesEx, ExZ3, color='black', linewidth=2, linestyle='dashed')
axZ.set_title(r"$\langle\hat{Z}_i\rangle$")
axZ.set_xlabel(r"$t$ (u. a.)")
axZ.set_aspect(aspect=0.5)
axZ.set_ylim([-1.1, 1.1])
fig.savefig('XYZModel_BasisEfficient.pdf')
plt.show()
```



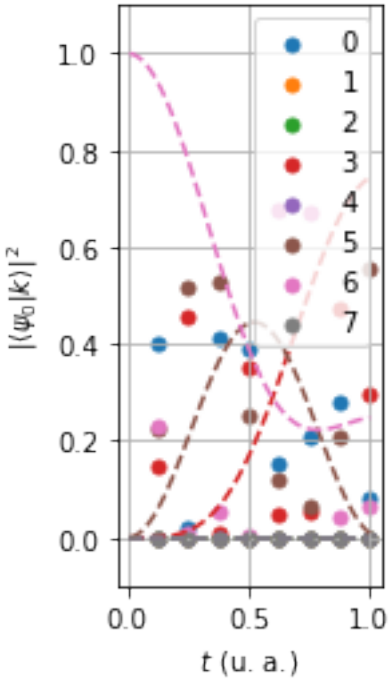
As is readily seen, the results are poor. Specially for a large number of integration steps. The reason for this is decoherence and qubit relaxation. To the lowest level, quantum algorithms are implemented via microwave pulses acting on a superconducting chip. The duration of hose pulses, and its relation to coherence and qubit relaxation times, determines the fidelity of the evolution. The longer the pulse schedule that implements the algorithm, the worse the fidelity. Although the evolution algorithm here implemented is *basis efficient*, i. e. uses a small number of CNOT gates and single qubit rotations per integration step, it is not *pulse efficient*, i. e. the actual sequence of microwave pulses implemented on hardware is simply to long and decoherence and qubit relaxation prevent high fidelities.

A more intelligent approach relies on implementations that not only are simple in the high level circuit model abstraction, but also imply a reduced the duration of its associated pulse schedule. IBM Quantum devices implement standard single qubit rotations with relatively high precision, and the bottleneck is mostly due to two-qubit gates. Such interactions are performed on hardware via **Cross-Resonance gates**

$$\hat{R}_{zx}(\phi) = \mathrm{e}^{-\mathrm{i}\phi\hat{Z}\otimes\hat{X}}$$

Although IBM Quantum devices have a universal set that uses CNOT as entangling gate, on hardware, actual hardware, two-qubit interactions are performed via cross-resonance

[8]:
```
xAnalyzer.comparativeEvolution(
    STEPS=8,
    t=1.0,
    measurementFitter=measurementFitter,
    figureFile='XYZModel_evol_DirectTranspilation.pdf',
    showLegend=True,
    showPlot=True
)
```



Notice that cross-resonance is very similar to the terms in the Hamiltonian, up to single qubit rotations. So, this seems to be the key for simulating spin systems with higher fidelity. This approach will be explored further in the future. By now, the limits of the proposed Trotterization scheme, and its relation with the target dynamics is discussed further.

Simulation using Cross Resonance gates

In this section, an evolution algorithm based on the cross resonance gate described above is tested. This is part of the Open Science Challenge from IBM. The first approach consists on using the `PassManager` object for preventing transpilation of RZX gates to basis gates, thus implementing them directly on hardware using microwave pulses. From trotterization benchmark, I have seen that at least a time step $dt \approx 0.2$ u. a. should be used for simulations to produce decent outcome. This is related to the particular Floquet dynamics at large time step.

[5]:
```
pGraph = PulseSpinGraph(
    spinInteractions={
        (0, 1): [0, 0, 0],
        (1, 2): [0, 0, 0],
        (1, 3): [1, 1, 1],
        (3, 5): [1, 1, 1],
        (4, 5): [0, 0, 0],
        (5, 6): [0, 0, 0],
    },
    externalField={
        0: [0.0, 0.0, 0.0],
        1: [0.0, 0.0, 0.0],
        2: [0.0, 0.0, 0.0],
        3: [0.0, 0.0, 0.0],
        4: [0.0, 0.0, 0.0],
        5: [0.0, 0.0, 0.0],
        6: [0.0, 0.0, 0.0],
    },
    localSimulation=False,
```

```
    backendName='ibmq_jakarta',
    noisySimulation=False,
    initializeList=[3,5]
)
pAnalyzer = DataAnalyzer(spinGraph=pGraph)
```

[20]:
```
pauliStrings = ['IXIIIII', 'IIIXIII', 'IIIIIXI', 'IYIIIII',
                'IIIYIII', 'IIIIIYI', 'IZIIIII', 'IIIZIII', 'IIIIIZI']
timesEx, resultSeriesExact = pGraph.exactPauliExpValSeries(
    pauliStrings,
    t=1
)
ExX1, ExX2, ExX3, ExY1, ExY2, ExY3, ExZ1, ExZ2, ExZ3 = itemgetter(
    *pauliStrings)(resultSeriesExact)
```

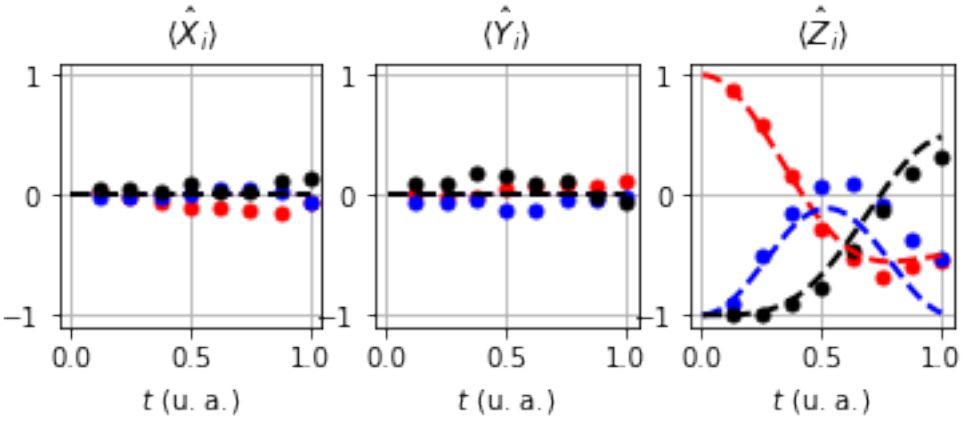
[28]:
```
measurementFitter = pGraph.getCalibrationFitter()
```

```
Job Status: job has successfully run
```

[22]:
```
times, resultSeries = pGraph.pauliExpValSeries(
    pauliStrings,
    MAX_STEPS=8,
    t=1,
    measurementFitter=measurementFitter
)
X1, X2, X3, Y1, Y2, Y3, Z1, Z2, Z3 = itemgetter(*pauliStrings)(resultSeries)
```

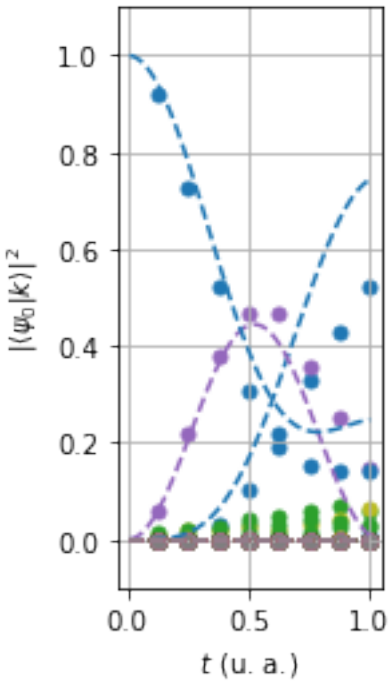
```
Job Status: job has successfully run
```

[24]:
```
fig, (axX, axY, axZ) = plt.subplots(1, 3)
axX.plot(times, X1, 'ro', times, X2, 'bo', times, X3, 'ko')
axX.plot(timesEx, ExX1, color='red', linewidth=2, linestyle='dashed')
axX.plot(timesEx, ExX2, color='blue', linewidth=2, linestyle='dashed')
axX.plot(timesEx, ExX3, color='black', linewidth=2, linestyle='dashed')
axX.set_title(r"$\langle\hat{X}_i\rangle$")
axX.set_xlabel(r"$t$ (u. a.)")
axX.set_aspect(aspect=0.5)
axX.set_ylim([-1.1, 1.1])
axY.plot(times, Y1, 'ro', times, Y2, 'bo', times, Y3, 'ko')
axY.plot(timesEx, ExY1, color='red', linewidth=2, linestyle='dashed')
axY.plot(timesEx, ExY2, color='blue', linewidth=2, linestyle='dashed')
axY.plot(timesEx, ExY3, color='black', linewidth=2, linestyle='dashed')
axY.set_title(r"$\langle\hat{Y}_i\rangle$")
axY.set_xlabel(r"$t$ (u. a.)")
axY.set_aspect(aspect=0.5)
axY.set_ylim([-1.1, 1.1])
axZ.plot(times, Z1, 'ro', times, Z2, 'bo', times, Z3, 'ko')
axZ.plot(timesEx, ExZ1, color='red', linewidth=2, linestyle='dashed')
axZ.plot(timesEx, ExZ2, color='blue', linewidth=2, linestyle='dashed')
axZ.plot(timesEx, ExZ3, color='black', linewidth=2, linestyle='dashed')
axZ.set_title(r"$\langle\hat{Z}_i\rangle$")
axZ.set_xlabel(r"$t$ (u. a.)")
axZ.set_aspect(aspect=0.5)
axZ.set_ylim([-1.1, 1.1])
fig.savefig('XYZModel_PulseEfficient.pdf')
plt.show()
```



[29]:
```
pAnalyzer.comparativeEvolution(
    STEPS=8,
    t=1.0,
    measurementFitter=measurementFitter,
    figureFile='XYZModel_evol_PulseEfficient.pdf',
    showLegend=False,
    showPlot=True
)
```

```
Job Status: job has successfully run
```



B.1.6 Pulse duration comparison

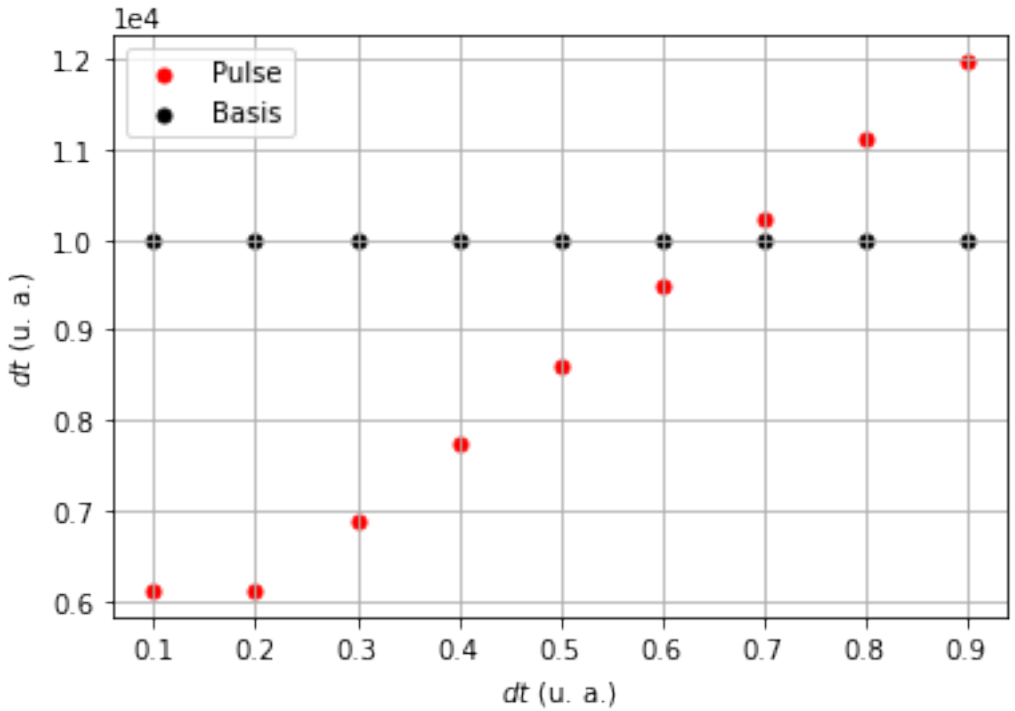
In the following cells, the pulse schedules of each trotter step for the three methods are compared

[12]:
```
from qiskit.transpiler import PassManager
from qiskit.transpiler.passes import RZXCalibrationBuilderNoEcho
from qiskit import schedule, transpile
```

[39]:
```
t = np.arange(0.1, 1.0, step=0.1)
pulseSchedTimes = []
basisSchedTimes = []
```

[41]:
```
pm = PassManager([RZXCalibrationBuilderNoEcho(pGraph.backend)])
for dt in t:
    xGraphStep = xGraph.rawEvolutionCircuit(STEPS=1, t=dt)
    pGraphStep = pGraph.rawEvolutionCircuit(STEPS=1, t=dt)
    pGraphPulseEfficientStep = transpile(pm.run(pGraphStep), pGraph.backend)
    pulseSchedTimes.append(schedule(pGraphPulseEfficientStep, pGraph.backend).
 ↪duration)
    xGraphTranspiledStep = transpile(xGraphStep, pGraph.backend)
    basisSchedTimes.append(schedule(xGraphTranspiledStep, pGraph.backend).
 ↪duration)
```

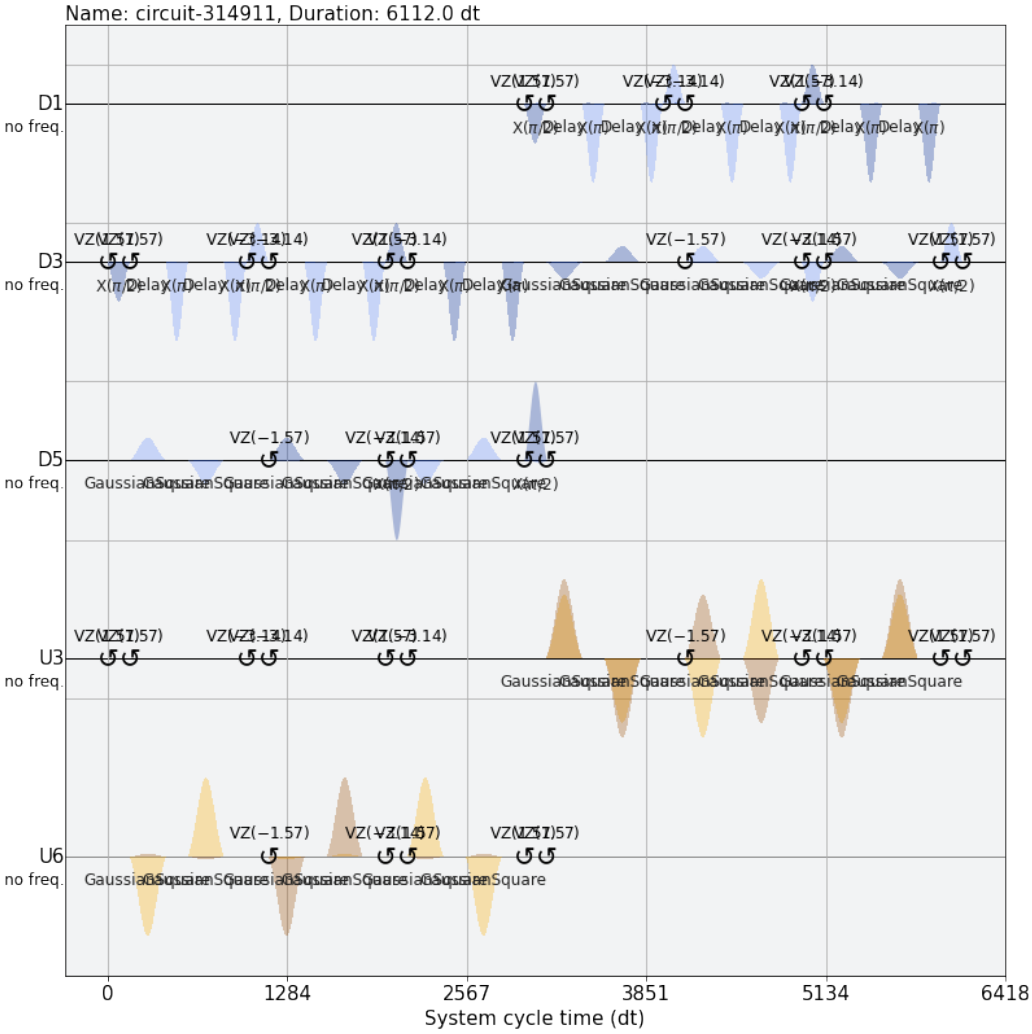
[46]:
```
fig, ax = plt.subplots(1,1)
ax.set_xlabel(r"$dt$ (u. a.)")
ax.set_ylabel(r"$dt$ (u. a.)")
ax.scatter(t, pulseSchedTimes, color='r', label=r"Pulse")
ax.scatter(t, basisSchedTimes, color='k', label=r"Basis")
ax.legend()
fig.savefig("schedTimeDurationPlot.pdf")
plt.show()
```



[47]:
```
xGraphStep = xGraph.rawEvolutionCircuit(STEPS=1, t=1/8)
pGraphStep = pGraph.rawEvolutionCircuit(STEPS=1, t=1/8)
```

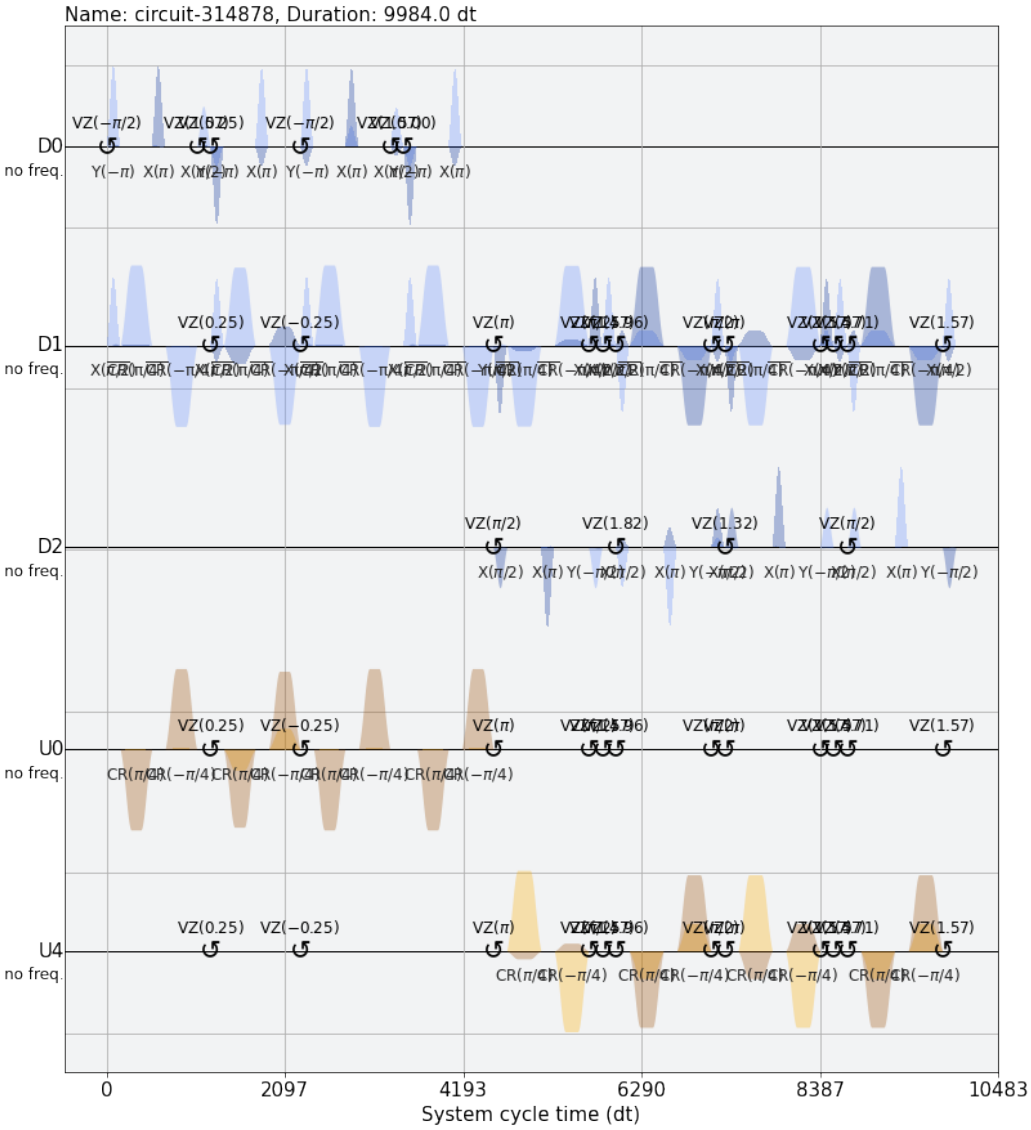
[53]:
```
pGraphPulseEfficientStep = transpile(pm.run(pGraphStep), pGraph.backend)
schedule(pGraphPulseEfficientStep, pGraph.backend).draw(time_unit='ns')
```

[53]:



[54]:
```
xGraphTranspiledStep = transpile(xGraphStep, pGraph.backend)
schedule(xGraphTranspiledStep, pGraph.backend).draw()ca
```

[54]:



B.1.7 Trotterization Benchmark

What follows is a benchmark that is designed to assess the quality of the evolution algorithm. On a first stage, the asymptotic behavior is studied in terms of the actual unitary. In particular, the actual relation between evolution error, total time evolution, and number of integration steps is computed for a representative benchmark Hamiltonian. After that, the dynamics induced by trotterization scheme is compared to the actual dynamics, for a particular initial state, as a function of the integration step $dt = t/N$. Long time evolution averages are used to illustrate possible differences. The size of the system is varied so as to characterize the erformance of the algorithm as a function of this parameter.

Benchmark Hamiltonian

The benchmark Hamiltonian is of the shape

$$\hat{H} = \sum_{i=0}^{\mathcal{N}-2} \left(\frac{1}{2} \hat{X}^{(i)} \hat{X}^{(i+1)} + \hat{Y}^{(i)} \hat{Y}^{(i+1)} + \frac{1}{4} \hat{Z}^{(i)} \hat{Z}^{(i+1)} \right) + \sum_{i=0}^{\mathcal{N}-1} \left(\hat{X}^{(i)} + \frac{1}{4} \hat{Y}^{(i)} + \frac{1}{2} \hat{Z}^{(i)} \right)$$

This corresponds to an XYZ Model, with an external field constant across the chain, and with non-periodic boundary conditions. The size of the system $\mathcal{N}$ may be varied arbitrarily. However, it is suggested that sizes no larger than $\mathcal{N} = 8$ be used on standard machines. Evolution of large systems is qte resource demanding (obviously). Although this exact notebook was used for data generation throughout the project, execution times tend to be quite large, and thus unless they are sought to be reproduced exactly, it is advised that the reader restricts the system's size for simple demos.

B.1.8 Asymptotic Analysis of Evolution Precision

In this section, a benchmark model is used for studying the Trotterization scheme performance at very small step sizes. Previous work predict a power law that relates

1. Evolution unitary error (ϵ)
2. Number of integration steps (N)
3. Total simulation time (t)

The Trotterization scheme proposed in this project produces a unitary $\hat{U}_N(t)$, that approximates the exact evolution unitary

$$\hat{U}(t) = \mathrm{e}^{-\mathrm{i}\hat{H}t}$$

Studying a sample

In this section a 5 spin system is used to evaluate the evolution error as a function of both t and N. Here

$$\epsilon = ||\hat{U}_N(t) - \hat{U}(t)||$$

Where the distance is simply the Frobenius norm. The results are evaluated using QASM simulator. Both time and number of integration steps are varied to determine a power law that predicts the asymptotic error for large times .

[4]:
```
numSpins = 6
benchmarkGraph = HeisenbergGraph(
    spinInteractions={
        (idx, idx+1): [0.5, 1.0, 0.25]
        for idx in range(numSpins-1)
    },
    externalField={
        idx: [1.0, 0.25, 0.5]
        for idx in range(numSpins)
    },
    localSimulation=True,
    backendName='qasm_simulator',
    noisySimulation=False,
    initialState=np.array([1 if idx == 1 else 0 for idx in range(2**numSpins)])
)
benchmarkAnalyzer = DataAnalyzer(spinGraph=benchmarkGraph)
```

Results and dscussion After computing errors, for this sample system, it is noticed that there is in fact a power law that describes the asymptotic error. In fact, linear regressions are performed to determine the exponents. Bellow, a plot confirming this fact is presented.

[ ]:
```
sim = benchmarkAnalyzer.unitaryErrorMixedPlot(
    STEPS = np.array([idx for idx in range(200, 232)]),
    times = [1,2,4,8,16,32]
)
```

In the following cells, linear regressions are performed, and exponents can be seen for different times and number of steps. The data suggests that

$$\epsilon \approx A \frac{t^{\frac{6}{5}}}{N}$$

In this asymptotic regime. This shows that although Berry et. al. produce a relatively tight bound for the Trotterization error, the actual computational complexity is better than the predicted by those computations, at least for the type of systems considered in this work. Since circuit depth scales linearly with the number of integration steps. However, this behavior is only asymptotic. It will be shown in following cells that the dynamics for a small number of integration steps and large times is fairly erratic, and there is a clear convergence radius that marks the onset of the asymptotic behavior.

Note: The following cells contain the results of the regressions.

[7]:
```
from scipy.stats import linregress
import pandas as pd
from IPython.display import display
STEPS = np.array([idx for idx in range(200, 232)])
times = [1, 2, 4, 8, 16, 32]
columns = ["slope", "intercept", "r-value", "p-value", "std-err"]
stepRows = [f"t={t}" for t in times]
timeRows = [f"N={n}" for n in STEPS]
```

[8]:
```
stepsLinregress = [
    [
        item
        for item in linregress(
            np.log(STEPS),
            np.log(sim[idx, :])
        )
    ]
    for idx in range(len(times))
]
stepDf = pd.DataFrame(
    data=stepsLinregress,
    index=stepRows,
    columns=columns
)
display(stepDf)
```

	slope	intercept	r-value	p-value	std-err
t=1	-1.000033	2.415382	-1.000000	8.642650e-189	1.048778e-07
t=2	-0.999982	3.260707	-1.000000	2.989916e-187	1.180216e-07
t=4	-1.000372	3.971356	-1.000000	7.748542e-150	2.085690e-06
t=8	-1.005337	4.742891	-1.000000	2.448726e-113	3.451890e-05
t=16	-1.066622	5.872705	-0.999998	1.547176e-81	4.205103e-04
t=32	-1.364227	8.624357	-0.999990	4.768404e-72	1.114168e-03

[10]:
```
timesLinregress = [
    [
        item
        for item in linregress(
            np.log(times),
            np.log(sim[:, idx])
        )
    ]
    for idx in range(len(STEPS))
```

```
]
timesDf = pd.DataFrame(
    data=timesLinregress,
    index=timeRows,
    columns=columns
)
display(timesDf.head(6))
```

	slope	intercept	r-value	p-value	std-err
N=200	1.192165	-2.935006	0.996322	0.000020	0.051267
N=201	1.191721	-2.939612	0.996346	0.000020	0.051078
N=202	1.191281	-2.944198	0.996370	0.000020	0.050891
N=203	1.190845	-2.948762	0.996394	0.000019	0.050706
N=204	1.190413	-2.953305	0.996417	0.000019	0.050522
N=205	1.189985	-2.957828	0.996440	0.000019	0.050340

Trotter dynamics

As mentioned before, the asymptotic behavior is preceded by a more erratic dynamics. In this section, this is characterized by two metrics: **Long time average fidelity** and **Inverse participation ration**, defined as follows

$$\mathcal{F}_\infty = \lim_{n\to\infty} \sum_{i=1}^{N} |\langle\psi_0|\hat{U}(t)^\dagger \hat{U}_N(t)|\psi_0\rangle|^2$$

$$\text{IPR} = \sum_n |\langle\phi_n|\psi_0\rangle|^4$$

Where $|\psi_0\rangle$ is the initial state of the system, and

$$\hat{U}_N(t)|\phi_n\rangle = \mathrm{e}^{-\mathrm{i}E_n t}|\phi_n\rangle$$

These quantities are studied as a function of the *integration step* $dt = t/N$.

Study of Long Time Average Fidelity Data has been generated using `floquetDataGeneration.py`. The algorithm is based upon diagonalization with NumPy. This may not be the best algorithm, however, this was the best option given my knowledge of scientific computing on python. The datasets are computed for chains of sizes 2, 4, 6, and 8. Data can be found on `../datafiles/FloquetData`.

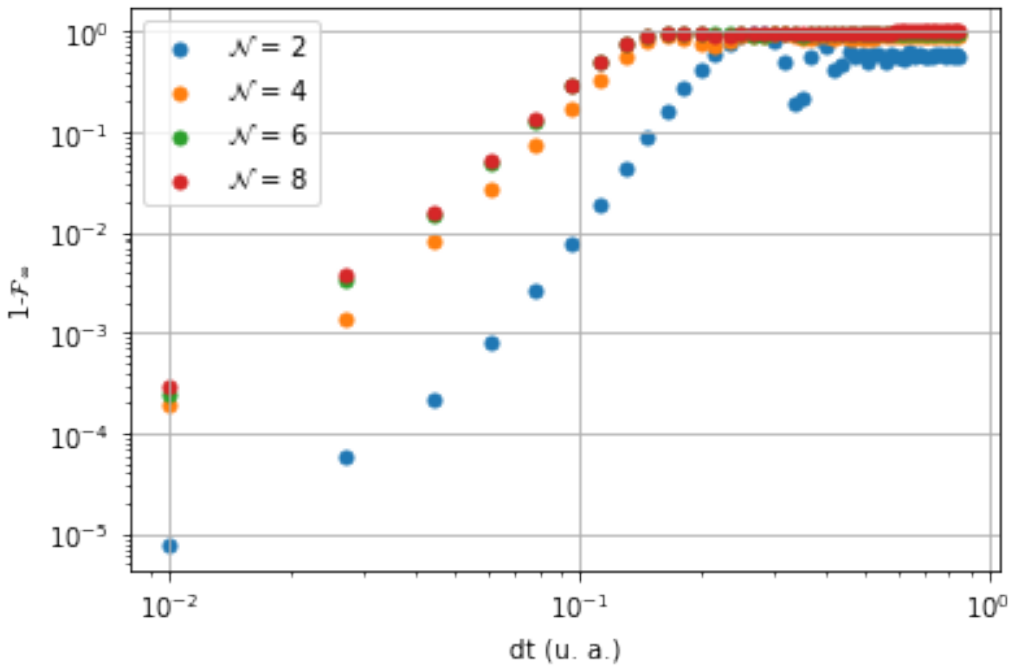
In the following cell, ndarrays are generated with the simulation results

[3]:
```
spinVals = [2, 4, 6, 8]
datasets = [
    np.genfromtxt(
        f"../datafiles/FloquetData/sampleData_{spins}spins.csv",
        delimiter=","
    )
    for spins in spinVals
]
```

Mow, plots for different spin values are generated

[8]:
```
figFloq, axFid = plt.subplots(1,1)
axFid.set_xlabel("dt (u. a.)")
axFid.set_xscale("log")
axFid.set_yscale("log")
# axFid.set_title("Long time average fidelity")
```

```
axFid.set_ylabel(r"1-$\mathcal{F}_{\infty}$")
# axIPR.set_xlabel("$dt$ (u. a.)")
# axIPR.set_xscale("log")
# axIPR.set_yscale("log")
# axIPR.set_title("IPR")
for idx, spin in enumerate(spinVals):
    axFid.scatter(
        datasets[idx][0:50, 0],
        1-datasets[idx][0:50, 1],
        label=r"$\mathcal{N}$" + f" = {spin}"
    )
#     axIPR.scatter(
#         datasets[idx][:, 0],
#         datasets[idx][:, 2],
#         label=r"$\mathcal{N}$" + f" = {spin}"
#     )
# axIPR.legend()
axFid.legend()
figFloq.savefig(
    "../images/Benchmark/_others/longTimeFloquetFidelity.pdf"
)
plt.show()
```



Results and discussion From the $\mathcal{F}_\infty$ plot, it can be seen that there is a sharp change of regime for time steps dt 0.2 approximately. There, the asymptotic regime is quite visible, whereas for larger time steps, the average fidelity is almost zero, which seems to suggest that there is a quite different time evolution below this threshold.

Note: It is important to understand better how to describe the dynamics in terms of quantum chaos. Due to the normalization of the wavefunction, trajectory divergency saturates. Therefore, measures of quantum chaos should be studied carefully and addressed better here.

Also, it is desirable to perform finite size scaling, to determine the large system critical time step

below which deterministic dynamics occur. However, data looks weird, and may need be repeated.

B.1.9 Annihilation with time evolution

One possible application of time evolution is the use of the adiabatic theorem for computing ground state properties of a given (time-independent) Hamiltonian $\hat{H}_0$. Consider a time-dependent T-periodic Hamiltonian

$$\hat{H}_a(t) = (1 - \frac{t}{T})\hat{H}_m + \frac{t}{T}\hat{H}_0$$

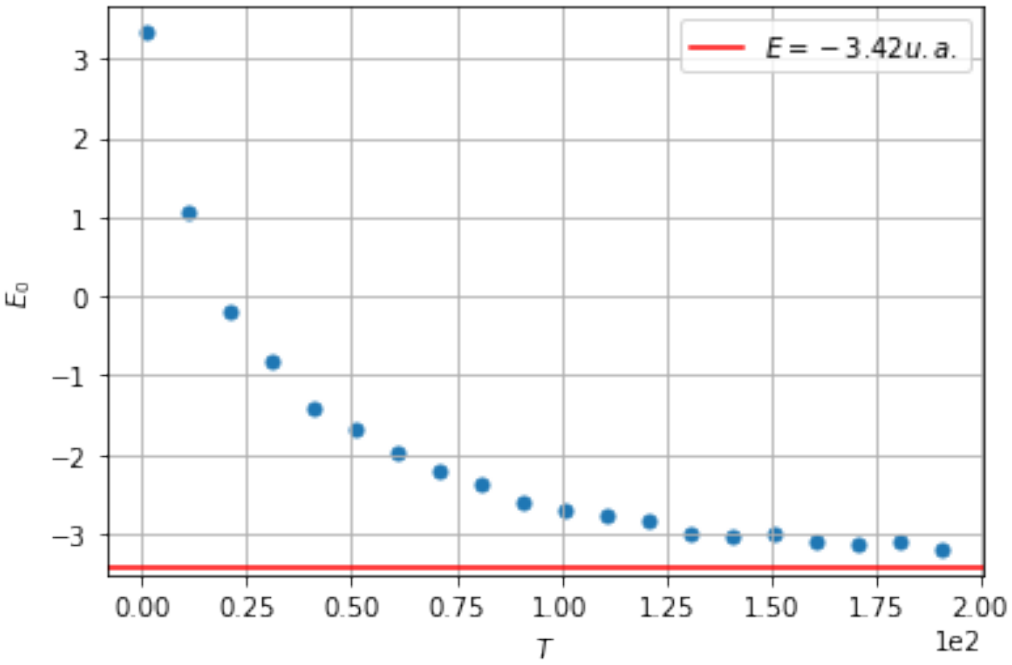
where $\hat{H}_m$ is some time-independent Hamiltonian, which is to be called *mixer Hamiltonian*. Consider an initial state $|\psi_0\rangle$ such that

$$\hat{H}_m|\psi_0\rangle = \epsilon_0|\psi_0\rangle$$

and the energy is the minimum over all the set of eigenstates. The adiabatic theorem states that, if the period $T \to \infty$, evolution of state $|\psi_0\rangle$ over a period yields a final state that is a ground state of the Hamiltonian $\hat{H}_0$. In the following cells, time evolution algorithm is used to find the ground state energy of an anisotropic Heisenberg Hamiltonian.

[2]:
```
aGraph = HeisenbergGraph(
    spinInteractions={
        (0, 1): [1/2, 1, 1/4],
        (1, 2): [1/2, 1, 1/4],
    },
    externalField={
        0: [0.0, 0.0, 1.0],
        1: [0.0, 0.0, 1.0],
        2: [0.0, 0.0, 1.0],
    },
    localSimulation=True,
    backendName='qasm_simulator',
    noisySimulation=False,
    initializeList=[1, 2]
)
aAnalyzer = DataAnalyzer(spinGraph=aGraph)
```

[3]:
```
_ = aAnalyzer.annealingProcessGraph(
    maxPeriod=200,
    periodStep=10,
    STEPS=500
)
```


$E = -3.42 u.a.$
E_0
3
2
1
0
−1
−2
−3
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e2
T

Bibliography

[1] Daniel S. Abrams and Seth Lloyd. "Simulation of Many-Body Fermi Systems on a Universal Quantum Computer". In: *Physical Review Letters* (1997). DOI: 10.1103/physrevlett.79.2586.

[2] MD SAJID ANIS et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2021. DOI: 10.5281/zenodo.2573505.

[3] R. Barends et al. "Digital quantum simulation of fermionic models with a superconducting circuit". In: *Nature communications* (2015). DOI: 10.1038/ncomms8654.

[4] Mark Beck. *Quantum Mechanics. Theory and Experiment*. Cambridge University Press, 2012.

[5] Giuliano Benentia, Giulio Casati, and Giuliano Strini. *Principles of Quantum Computation and Information. Volume I: Basic Concepts*. World Scientific, 2004.

[6] Cao et al. "Quantum Chemistry in the Age of Quantum Computing". In: *Chemical Reviews* (2019). DOI: 10.1021/acs.chemrev.8b00803.

[7] Richard Cleve Dominic. W. Berry Graeme Ahokas and Barry C. Sanders. "Efficient quantum algorithms for simulating sparse Hamiltonians". In: *Communications in Mathematical Physics* (2006). DOI: 10.1007/s00220-006-0150-x.

[8] Nathan Earnest, Caroline Tornow, and Daniel J. Egger. *Pulse-efficient circuit transpilation for quantum applications on cross-resonance-based hardware*. 2021. arXiv: 2105.01063 [quant-ph].

[9] Richard P. Feynman. "Simulating Physics with Computers". In: *International Journal of Theoretical Physics* (1982). DOI: 10.1007/bf02650179.

[10] Urtzi Las Heras et al. "Fermionic models with superconducting includes". In: *EPJ Quantum Technology* (2015). DOI: 10.1140/epjqt/s40507-015-0021-5.

[11] Markus Heyl, Philipp Hauke, and Peter Zoller. "Quantum localization bounds Trotter errors in digital quantum simulation". In: *Science Advances* 5.4 (2019). ISSN: 2375-2548. DOI: 10.1126/sciadv.aau8342. URL: http://dx.doi.org/10.1126/sciadv.aau8342.

[12] J. Hubbard. "Electron Correlations in Narrow Energy Bands". In: *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences* (1963). DOI: 10.2307/2414761.

[13] P. Krantz et al. "A quantum engineer's guide to superconducting qubits". In: *Applied Physics Reviews* 6.2 (2019), p. 021318. ISSN: 1931-9401. DOI: 10.1063/1.5089550. URL: http://dx.doi.org/10.1063/1.5089550.

[14] Seth Lloyd. "Universal Quantum Simulators". In: *Nature* (1996).

[15] Easwar Magesan and Jay M. Gambetta. "Effective Hamiltonian models of the cross-resonance gate". In: *Physical Review A* 101.5 (2020). ISSN: 2469-9934. DOI: 10.1103/physreva.101.052308. URL: http://dx.doi.org/10.1103/PhysRevA.101.052308.

[16] Sam McArdle et al. "Quantum computational chemistry". In: *Physical Review X* (2016). DOI: 10.1103/PhysRevX.6.031007.

[17] M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[18] Jan Michel Reiner. "Emulation of the One-Dimensional Fermi-Hubbard Model Using Superconducting Qubits". MA thesis. Karlsruher Institut für Technologie, 2015.

[19] J. J. Sakurai and Jim Napolitano. *Modern Quantum Mechanics*. Cambridge University Press, 2021.

[20] Y. Salathé et al. “Digital Quantum Simulation of Spin Models with Circuit Quantum Electrodynamics”. In: *Physical Review X* 5.2 (2015). ISSN: 2160-3308. DOI: 10.1103/physrevx.5.021027. URL: http://dx.doi.org/10.1103/PhysRevX.5.021027.

[21] Sarah Sheldon et al. “Procedure for systematically tuning up cross-talk in the cross-resonance gate”. In: *Physical Review A* 93.6 (2016). ISSN: 2469-9934. DOI: 10.1103/physreva.93.060302. URL: http://dx.doi.org/10.1103/PhysRevA.93.060302.

[22] John P. T. Stenger et al. “Simulating the dynamics of braiding of Majorana zero modes using an IBM quantum computer”. In: *Physical Review Research* 3.3 (2021). ISSN: 2643-1564. DOI: 10.1103/physrevresearch.3.033171. URL: http://dx.doi.org/10.1103/PhysRevResearch.3.033171.

[23] Neereja Sundaresan et al. “Reducing Unitary and Spectator Errors in Cross Resonance with Optimized Rotary Echoes”. In: *PRX Quantum* 1.2 (2020). ISSN: 2691-3399. DOI: 10.1103/prxquantum.1.020318. URL: http://dx.doi.org/10.1103/PRXQuantum.1.020318.

[24] Masuo Suzuki. “Fractal decomposition of exponential operators with applications to many-body theories and Monte Carlo simulations”. In: *Physics Letters A* (1990). DOI: 10.1016/0375-9601(90)90962-n.

[25] G. Vidal and C. M. Dawson. “Universal quantum circuit for two-qubit transformations with three controlled-NOT gates”. In: *Physical Review A* 69.1 (2004). ISSN: 1094-1622. DOI: 10.1103/physreva.69.010301. URL: http://dx.doi.org/10.1103/PhysRevA.69.010301.