Running and optimising a heisenberg model hamiltonian for three interacting atoms on a quantum computer

Contents

1 Introduction

IBM put forward the problem of simulating a Heisenberg model Hamiltonian for three interacting atoms with maximal accuracy on IBM Quantum's 7-qubit Jakarta system in their annual open science prize[1]. The included material for the contest contained an example of a fairly inaccurate solution. The aim of this investigation is to address the challenge by improving this example solution by using the same gate decomposition and finding an optimal value for the number of trotterisation[1, 2] steps to be performed. Trotterisation and gate decomposition are concepts that will be introduced later in more detail

. While investigating the topic I also came accross an alternative way to achieve a solution to the problem, by using a different gate decomposition, this alternative method is then also optimsed and compared with the example method. The majority of this investigation will be about explaining the relevant concepts and methodology to the reader in a way which can be understood without any background in quantum information theory and without using quantum specific concepts, but instead describing the entire procedure in terms of vectors and matrices.

The problem of simulating quantum systems with quantum computers and finding efficient and accurate ways to do so is relevant since this is an area in which classical computers are ineffective as the computation time with classical computers grows exponentially with the input size. Hence, this is an area for which even quantum computers of the near future are probably going to be effective in. In general simulating quantum systems are important in many fields of science, mainly in chemistry where it can offer new insights and improve the effectiveness of designing new materials and technologies.

The reason I chose this problem to tackle in my mathematics IA is because quantum computation is a field that I want to continue to study more in university and I thought this investigation would offer me an opportunity to delve into the field before university. As background I had already taken introductory course into quantum computation, but this investigation allowed me to gain a more rigorous understanding of one area of quantum computation.

Research problem Finding the most accurate way to simulate a three particle system with a quantum computer

Methodology Finding an optimum value for the number of trotterisation steps for the example decomposition offered in the material of IBMs competition [4]. There is also a different type of decomposition based on the findings of [5] which is then optimised and compared to the other decomposition to find the more accurate way to simulate the quantum system.

[1] https://ibmquantumawards.bemyapp.com/#/event

2 Background

2.1 Quantum states as vectors and ket notation

Quantum states can be represented by vectors and for the length of this investigation they may be considered as such. Quantum states evolve in time by state transformations, which can be represented by matrices. Hence, for the length of this investigation quantum states can be considered as vectors which evolve in time by matrix-vector multiplication. The specific name for the matrix which defines how a quantum state will evolve is a hamiltonian. So any mention of a hamiltonian the reader can simply regard as a matrix which changes a vector by matrix-vector multiplication.

Quantum computers work similarly, they have an initial state which can be expressed as a vector and this vector can be changed by applying operations/gates to it, which can be represented as multiplying the vector with some matrix.

With this information, simulation of a quantum state by quantum computers can be expressed more clearly in mathematical terms. There is a vector which describes a quantum system. It has some initial state and it evolves in time in a way which can be expressed as it being multiplied by some matrix which is dependant upon time, t. The goal is to imitate this time-dependent matrix called a hamiltonian with a quantum computer. However, quantum computers can not apply any arbitrary matrix to initial state, but instead they only have certain matrices they can use and the goal is to construct the hamiltonian by using the matrices available for the quantum computer.

The term qubit is also used in this investigation. A qubit is the fundamental unit of information in quantum computing. However, in this investigation one qubit of the quantum computer corresponds to one of the particles that is being simulated, so the reader can simply regard the qubit as a part of the quantum computer's state that corresponds to one of the simulated particles.

2.2 Tensor product

It is important to note that there are two types of matrix operations used in this investigation, traditional matrix multiplication and the tensor product. In the construction of a single operation or hamiltonian from submatrices, the tensor product is used, but when referring to applying some operations one after another, matrix multiplication is used. This deliniation will become more clear in later sections.

The tensor product is represented by $\otimes$ and the numerical calculation can be seen below.

$$\begin{bmatrix} a_{1,1} & a_{1,2} \\ a_{2,1} & a_{2,2} \end{bmatrix} \otimes \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{1,1}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{1,2}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \\ a_{2,1}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} & a_{2,2}\begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} a_{1,1}b_{1,1} & a_{1,1}b_{1,2} & a_{1,2}b_{1,1} & a_{1,2}b_{1,2} \\ a_{1,1}b_{2,1} & a_{1,1}b_{2,2} & a_{1,2}b_{2,1} & a_{1,2}b_{2,2} \\ a_{2,1}b_{1,1} & a_{2,1}b_{1,2} & a_{2,2}b_{1,1} & a_{2,2}b_{1,2} \\ a_{2,1}b_{2,1} & a_{2,1}b_{2,2} & a_{2,2}b_{2,1} & a_{2,2}b_{2,2} \end{bmatrix}$$

As can be seen the tensor product doubles the number of rows and columns in the matrix. This is important, because a quantum state is represented by a vector of dimensionality 2^n where n is the number of particles represented by the vector. Similarly one two by two matrix represents a change in the state of one of the particles and if we have two of the particles changing at the same time, this change be represented by the tensor product of the two 2×2 matrices corresponding to the changes of each of the particles. Hence, the number of rows and columns for a matrix representing the change in a quantum state with n particles is 2^n. This makes sense, since the changes are represented by matrix-vector multiplication and for matrix-vector multiplication the matrix must have as many columns as the amount of dimensions in the vector. [6]

In the case of constructing a matrix state transformation that doesn't affect all the particles in the system, the identity matrix is used in the place of the non-changing particles.

It is of note that not all possible transformations can be described as tensors of matrices acting on single particles, since the particles can also interact with each other. Hence, the above explanation was merely meant to give an idea as to why the tensor product is used in this investigation.

Furthermore, strictly speaking this numerical operation for matrices is called the kronecker product and the tensor product is an operation on linear maps between vector spaces. However, this distinction is not important for this investigation.

2.3 Taylor series

The taylor series is a way to estimate any function by a polynomial function by creating a polynomial expansion where the constants in the polynomial are chosen such that the derivatives of the polynomial match the derivatives of the function being estimated. If the polynomial expansion is continued into an infinite series, the numerical estimation of the value of the function being estimated can be fully accurate. The specifics of this procedure aren't important for this investigation. All the reader needs to know is that this procedure is used to create an infinithve series which can numerically estimate the value of an exponential function at a point x. The expansion is as follows

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ...$$

2.4 Pauli operators

In quantum mechanics the pauli operators correspond to observables of spin-1/2 systems. In this investigation the model by which the atoms interact is made up of pauli operators. As mentioned previously these operators can be described by matrices, so all one needs to know about the pauli operators are

their corresponding matrices, which can be seen below.

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$$

$$\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

2.5 Schrödinger equation

The Schrödinger equation (1) describes the time evolution of a quantum state. [6]

$$i\hbar \frac{d|\Psi(t)\rangle}{dt} = H|\Psi(t)\rangle \tag{1}$$

In this equation $|\Psi(t)\rangle$ simply represents a quantum state at time t. $\hbar$ is the planck's constant and to simplify we regard $\hbar = 1$ for the rest of this investigation. The schrodinger equation then states that the change in a quantum state at time t is determined by a hamiltonian H which the reader can regard as a matrix. This means that the schrodinger equation is reduced to a simple differential equation which evaluates as follows

$$i\frac{d|\Psi(t)\rangle}{dt} = H|\Psi(t)\rangle \Rightarrow |\Psi(t)\rangle = e^{-iHt}|\Psi(0)\rangle$$

because

$$\begin{aligned} i\frac{d|\Psi(t)\rangle}{dt} &= H|\Psi(t)\rangle \\ \int \frac{1}{|\Psi(t)\rangle} d|\Psi(t)\rangle &= \int \frac{H}{i} 1 dt \\ \ln(|\Psi(t)\rangle) &= \frac{H}{i}t + C \\ e^{\ln(|\Psi(t)\rangle)} &= e^{\frac{H}{i}t+C} \\ |\Psi(t)\rangle &= Ae^{\frac{H}{i}t} \\ |\Psi(t)\rangle &= Ae^{\frac{tH}{i}} \end{aligned}$$

It is true that $\frac{1}{i} = \frac{i}{i^2} = \frac{i}{-1} = -i$, hence $Ae^{\frac{Ht}{i}} = Ae^{-iHt}$. Furthermore, at $t = 0$, $|\Psi(0)\rangle = Ae^{-iH\cdot 0} = Ae^0 = A = |\Psi(0)\rangle$, so we get the solution

$$|\Psi(t)\rangle = e^{-iHt}|\Psi(0)\rangle$$

2.6 Hamiltonian model and classical computation

In the simulation of a spin-1/2 system, the first step is to choose a model with which to describe how the particles in the sytsem interact with each other and how the system changes with time. The chosen model will then give a hamiltonian operator which when applied to the initial conditions of the quantum state, describes how the system changes with time as per the schrödinger equation. The model used in this simulation is the XXX-Heisenberg model as this is the model used in the competition. The XXX-Heisenberg spin model is defined below.[6]

$$H = \sum_{\langle ij \rangle}^{N} J(\sigma_x^{(i)}\sigma_x^{(j)} + \sigma_y^{(i)}\sigma_y^{(j)} + \sigma_z^{(i)}\sigma_z^{(j)})$$

In this sum J is the interaction strength, which we can set to $J = 1$ simplifying the sum. $\sigma_\alpha|\alpha \in \{x, y, z\}$ are simply the pauli matrixes introduced earlier. Furthermore, in the above sum adjacent pauli matrices are tensored together and N is the number of particles being simulated. The notation $\langle ij \rangle$ means we are only summing over nearest neighbours, so in this case the sum would evaluate the sum of $i = 0, j = 1$ and $i = 1, j = 2$. In terms of the model this means only adjacent particles interact with each other. Hence, writing out the entire sum, including the identitiy matrices, would be as follows

$$H = \sigma_x^{(0)}\otimes\sigma_x^{(1)}\otimes I^{(2)} + I^{(0)}\otimes\sigma_x^{(1)}\otimes\sigma_x^{(2)} + \sigma_y^{(0)}\otimes\sigma_y^{(1)}\otimes I^{(2)} + I^{(0)}\otimes\sigma_y^{(1)}\otimes\sigma_y^{(2)} + \sigma_z^{(0)}\otimes\sigma_z^{(1)}\otimes I^{(2)} + I^{(0)}\otimes\sigma_z^{(1)}\otimes\sigma_z^{(2)} \tag{2}$$

Where $\otimes$ is the tensor product and I is the identity matrix.

Now that the hamiltonian for the time evolution of the system is known, the state of the system at time t can be modelled by simply evaluating the exponential of the hamilonian and multiplying that with the initial state of the system as seen from the schrodinger equation. So, that is $|\Psi(t)\rangle = e^{-iHt}|\Psi(0)\rangle$. To simplify the notation we define a function

$$U(t) = e^{-itH} = exp(-itH)$$

Then $|\Psi(t)\rangle = U(t)|\Psi(0)\rangle$. With the hamiltonian being an 8×8 matrix, $U(t)$ can in this case be evaluated with a classical computer using the taylor series. However, for larger systems this calculation on a classical computer soon becomes intractible since the size of the martix grows exponentially with the number of particles being simulated. However, the runtime on a quantum computer grows only polynomially. This is because quantum computers themselves deal with quantum states and the state of a quantum computer can be described by exactly the same vector as the 3-particle quantum system. The only issue is that a quantum computer can not create any arbitrary matrix transformation on its state, but instead is limited to the transformation matrixes that can be built from the set of native operations it can run.

2.7 The set of operators that a quantum computer can natively run

The exact set of operators that can be natively run on a quantum computer depends on the quantum computer and often the technology that it is based on. The quantum computer used in this investigation is ibmq_jakarta, which uses one of IBM Quantum's Falcon processors. It is a superconducting quantum computer and has the native set of operations $S = \{\mathrm{R}_x(\theta), \mathrm{R}_y(\theta), \mathrm{R}_z(\theta), \mathrm{CNOT}\}$. The operations $\mathrm{R}_\alpha(\theta)|\alpha \in \{x, y, z\}$ are single qubit rotations and the CNOT is analagous to the classical controlled not operation, because it relates two qubits at once it is a 4×4 matrix. These operations can be reperesented by the following matrices.

$$\mathrm{R}_x(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -i\sin(\frac{\theta}{2}) \\ -i\sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$$

$$\mathrm{R}_y(\theta) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -\sin(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) & \cos(\frac{\theta}{2}) \end{pmatrix}$$

$$\mathrm{R}_z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$$

$$\mathrm{CNOT} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

The goal is then to describe the hamiltonian from section 2.6 with these gates. However, this cannot be done directly so an additional technique is needed, called trotterisation.

2.8 Trotterisation

For real numbers it is true that $e^{a+b} = e^a \cdot e^b | a, b \in \mathbb{R}$. However, because some matrices do not commute, an exponential with two added matrices can not always be split into individual exponentials multiplied together. So, in the case of non-commutative matrices M and N, $e^{M+N} \neq e^M \cdot e^N | M, N \in \mathbb{R}^{n\times n}$ and since the pauli matrices do not commute with each other, this is the case for the hamiltonian from section 2.6. However e^{M+N} can be approximated accurately by $(e^{M/n} \cdot e^{N/n})^n$ with a sufficiently large n[1]. This is called trotterisation. A general form of this rule, used in the context of quantum computing, can be seen below. This is called the suzuki-trotter decomposition[2].

$$e^{-i\sum_l H_l t} = \lim_{n\to\infty} \left(\prod e^{-iH_l t/n}\right)^n$$

With this, given that a hamiltonian, H, is made up of a sum of sub-hamiltonians, H_l, which are executable on a quantum computer, the full hamiltonian can be decomposed into those sub-hamiltonians and given that they are applied in small enough steps, this process can be used to accurately imitate the larger full hamiltonian on a quantum computer.

3 Simulating the hamiltonian on a quantum computer

3.1 Gate decomposition of the hamiltonian

The aim is now to split $U(t)$ into simpler exponentials that can be evaluated with quantum computers and then use trotterisation to estimate the whole hamiltonian. To do this, we can look at the long sum that was shown earlier in equation 2. Now with identities and tensor product sign removed for succinctness.

$$H = \sigma_x^{(0)}\sigma_x^{(1)} + \sigma_x^{(1)}\sigma_x^{(2)} + \sigma_y^{(0)}\sigma_y^{(1)} + \sigma_y^{(1)}\sigma_y^{(2)} + \sigma_z^{(0)}\sigma_z^{(1)} + \sigma_z^{(1)}\sigma_z^{(2)}$$
$$U(t) = \exp\left(-it\left(\sigma_x^{(0)}\sigma_x^{(1)} + \sigma_x^{(1)}\sigma_x^{(2)} + \sigma_y^{(0)}\sigma_y^{(1)} + \sigma_y^{(1)}\sigma_y^{(2)} + \sigma_z^{(0)}\sigma_z^{(1)} + \sigma_z^{(1)}\sigma_z^{(2)}\right)\right)$$

The pairs of pauli operators ($\sigma_x^{(i)}\sigma_x^{(j)}$, $\sigma_y^{(i)}\sigma_y^{(j)}$, $\sigma_y^{(i)}\sigma_y^{(j)}$) evaluate as hamiltonians that can be represented with the available operators on quantum computers. Therefore an appropriate Suzuki-Trotter decomposition would be one which preserves these pairs of pauli exponentials, as follows. This is the simple decomposition which was presented in IBM's material for the competition[4].

$$U(t) \approx \left(\exp\left(\frac{-it}{n}\sigma_x^{(0)}\sigma_x^{(1)}\right)\exp\left(\frac{-it}{n}\sigma_y^{(0)}\sigma_y^{(1)}\right)\exp\left(\frac{-it}{n}\sigma_z^{(0)}\sigma_z^{(1)}\right)\exp\left(\frac{-it}{n}\sigma_x^{(1)}\sigma_x^{(2)}\right)\cdots \right.$$
$$\left. \exp\left(\frac{-it}{n}\sigma_y^{(1)}\sigma_y^{(2)}\right)\exp\left(\frac{-it}{n}\sigma_z^{(1)}\sigma_z^{(2)}\right)\right)^n$$

These pairwise pauli exponentials are commonly notated by $AB(2t) = e^{-it\sigma_\alpha\sigma_\beta}$. So, in this case we'd have $XX(2t) = \exp(-it\sigma_x\sigma_x)$, $YY(2t) = \exp(-it\sigma_x\sigma_x)$ and $ZZ(2t) = \exp(-it\sigma_x\sigma_x)$. These clarify the notation a bit, netting the expression

$$U(t) \approx \left(XX\left(\frac{2t}{n}\right)^{(0,1)} YY\left(\frac{2t}{n}\right)^{(0,1)} ZZ\left(\frac{2t}{n}\right)^{(0,1)} XX\left(\frac{2t}{n}\right)^{(1,2)} YY\left(\frac{2t}{n}\right)^{(1,2)} ZZ\left(\frac{2t}{n}\right)^{(1,2)}\right)^n$$

Now the next step is to figure out how to make these XX, YY and ZZ operators with the available native operations for the quantum computer. Let us begin with the ZZ operations.

$$ZZ(2t) = e^{-it\sigma_z\otimes\sigma_z}$$

let's first evaluate the tensor product of two pauli z operators.

$$\sigma_z\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = Z$$

e^{-itZ} can then be evaluated via the taylor series. For diagonal matrices the exponential is simple evaluates as follows

$$e^{-itZ} = \begin{pmatrix} e^{-it} & 0 & 0 & 0 \\ 0 & e^{it} & 0 & 0 \\ 0 & 0 & e^{it} & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix} = ZZ(2t)$$

A way this operation can be implemented on a quantum computer is with the following series of operations [4]

$$\mathrm{CNOT}^{(0,1)} \cdot \mathrm{R}_z(2t)^{(0)} \otimes I^{(1)} \cdot \mathrm{CNOT}^{(0,1)} = ZZ(2t)$$

To confirm this, the matrix representations are multiplied togeher and the resultant matrix is compared with the above matrix representation for the ZZ operation. Firstly the tensor product of the Rz and identity is evaluated as below.

$$\mathrm{R}_z(2t)^{(0)} \otimes I^{(1)} = \begin{pmatrix} e^{-it} & 0 \\ 0 & e^{it} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} e^{-it} & 0 & 0 & 0 \\ 0 & e^{it} & 0 & 0 \\ 0 & 0 & e^{-it} & 0 \\ 0 & 0 & 0 & e^{it} \end{pmatrix}$$

This multiplied with the CNOT operators then evaluates to the desired matrix as expected.

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} e^{-it} & 0 & 0 & 0 \\ 0 & e^{it} & 0 & 0 \\ 0 & 0 & e^{-it} & 0 \\ 0 & 0 & 0 & e^{it} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} e^{-it} & 0 & 0 & 0 \\ 0 & e^{it} & 0 & 0 \\ 0 & 0 & e^{it} & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix} = ZZ(2t)$$

XX and YY operations can then be constructed by applying the single qubit rotations Ry and Rx around the ZZ operation as follows. [4]

$$YY(2t) = \mathrm{R}_x(\frac{\pi}{2})^{(0)} \otimes \mathrm{R}_x(\frac{\pi}{2})^{(1)} \cdot ZZ(2t) \cdot \mathrm{R}_x(-\frac{\pi}{2})^{(0)} \otimes \mathrm{R}_x(-\frac{\pi}{2})^{(1)}$$
$$XX(2t) = \mathrm{R}_y(\frac{\pi}{2})^{(0)} \otimes \mathrm{R}_y(\frac{\pi}{2})^{(1)} \cdot ZZ(2t) \cdot \mathrm{R}_y(-\frac{\pi}{2})^{(0)} \otimes \mathrm{R}_y(-\frac{\pi}{2})^{(1)}$$

These can be proved via working out the required matrix multiplications, which are written out in section 4.2 of the appendix and this is the easiest way to prove that these sets of native operations do in fact correspond to the XX and YY operations. There is also a more algebraic working out on how to arrive at this conclusion included in section 4.2.2 of the appendix, which represents a slightly more natural way to discover the correct set of native operations that correspond to the XX and YY operations.

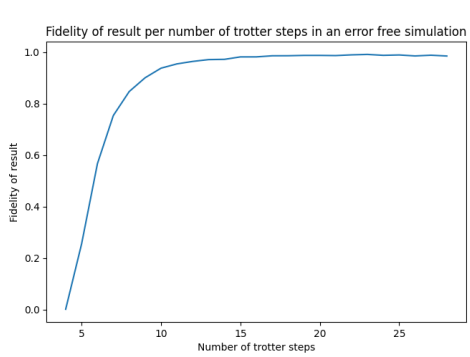
3.2 Optimisation

Now that the set of operations that need to be porfermod on the quantum computer are known, we need to execute it. However, there remains the question of how many trotter steps should be performed. In theory, the more steps are used, the more accurate the solution will be. To test this the trotterised program with different number of trotter steps can be run on a simulator of a quantum computer. However, a measure of the accuracy of the end state of the quantum system is needed.

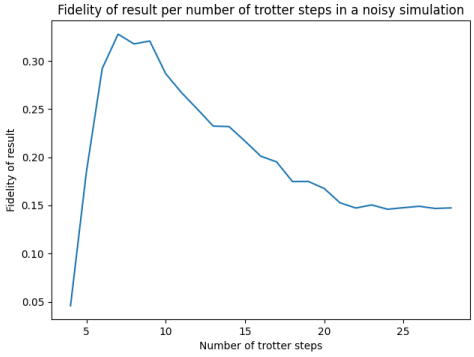
In order to quantify the accuracy of the simulation of the interacting particles the state-fidelity of the output state with the correct state, calculated classically with the full hamiltonian is evaluated. The specifics of this fidelity calculation are not important, the main idea is simply that the state fidelity gives a number between 0 and 1 where a higher value means that the states are more similar. More information is included in section 4.3 in the appendix.

The results about the accuracy of the simulation at different numbers of trotter steps can be seen in figure 1a, which shows that increasing the number of trtotter steps increases the accuracy of the simulation, achieving over 0.95 state-fidelity with each number of trotter steps more than 11.

However, an issue in actually achieving such high fidelity on current real quantum computers is that they contain a lot of error in each operation and as we increase the number of trotter steps we also increase the number of operations performed on the quantum computer, which means an increase in the amount of total error. If we run the program on a simulator which contains the expected amount of error in each operation as would occur on an actual quantum computer, specifically IBM's jakarta system, we gain results as seen in figure 1b. From these results it can be seen that the fidelity peaks and then declines as the error begins to build up and cancel the effects of the increased precision from the increased number of trotter steps. Looking at the raw data, found in section 4.1 of the appendix, we can see that the number of trotter steps that has the highest fidelity is seven. Hence, with this particular simulation model and on this particular quantum computer the optimal number of trotter steps is seven.



(a) Fidelity results on an ideal quantum computer



(b) Fidelity results on a simulator which incorporates noise

Figure 1: Fidelity results of simulations

3.3 Simpler trotter decomposition

However the particular trotter decomposition used above isn't the only one possible. According to the results of [5] if we consider the effect of the XX, YY and ZZ acting on two qubits globally, a procedure can be designed for a quantum computer with half the amount of the two-qubit CNOT operations. Two qubit operations create a significant amount more error than operations that act on a single qubit at a time. Hence, this decomposition of the hamiltonian has a possibility of offering a significant impromenent in terms of fidelity.

Let's call this new procedure $CC(2t)$. As it incorporates $XX(2t)$, $YY(2t)$ and $ZZ(2t)$, the new trotter decomposition for $U(t)$ would be as follows

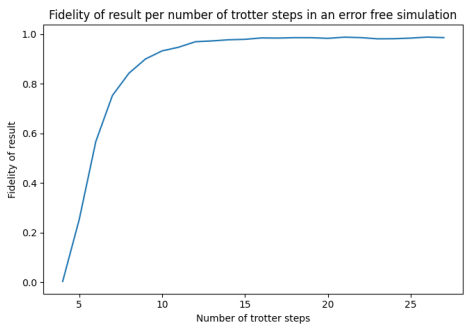
$$U(t) \approx \left(CC\left(\frac{2t}{n}\right)^{(0,1)} CC\left(\frac{2t}{n}\right)^{(1,2)} \right)^n$$

The set of operations that constitute $CC(2t)$ are

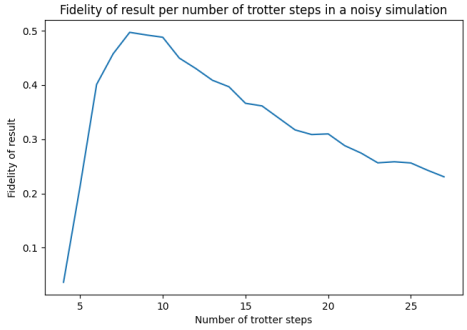
$$\mathrm{CNOT}^{(0,1)} \cdot \mathrm{R}_x^{(0)}(2t - \pi/2) \otimes \mathrm{R}_z(2t) \cdot \mathrm{R}_x^{(0)}(\pi) \cdot \mathrm{R}_y^{(0)}(\pi/2) \cdot \mathrm{CNOT}^{(0,1)} \cdot \mathrm{R}_x^{(0)}(\pi) \cdot \mathrm{R}_y^{(0)}(\pi/2) \otimes \mathrm{R}_z^{\dagger(1)}(2t) \cdot \cdot$$
$$\mathrm{CNOT}^{(0,1)} \cdot \mathrm{R}_x^{(0)}(\pi/2) \otimes \mathrm{R}_x^{\dagger(1)}(\pi/2)$$

The fidelity results for this more advanced trotter decomposition can be found in figure 2. The peak fidelity for this decomposition was at eight trotter steps. It is suprisingly close to the optimal number of trotter steps for the more naive trotter decomposition. Regardless, the performance is better as at the optimal number of trotter steps the fidelity was around 0.5, compared to the 0.33 of the other

decomposition.



(a) Fidelity results on an ideal quantum computer



(b) Fidelity results on a simulator which incorporates noise

Figure 2: Fidelity results for the CC trotter decomposition

Conclusion

To conclude, an answer to the proposed question of finding an optimum number of trotterisation steps for the example decomposition of the problem found on IBM's website has been found, at seven trotter steps where the highest fidelity result of the simulation observed at a fidelity of 0.33. However, utilising a different decomposition of the problem, for which the basic framework can be found from the results of [5] a better fidelity was observed at a fidelity of 0.5 with eight trotterisation steps.

References

[1] H. F. Trotter, Proc. Amer. Math. Soc. 1959, 10, 545 (https://www.jstor.org/stable/2033649)

[2] N. Hatano, M. Suzuki. 2005. Quantum Annealing and Other Optimiza- tion Methods. Berlin, Heidelberg. (Eds. A. Das, B. K. Chakrabarti, pp. 37–68), Springer. https://www.researchgate.net/publication/252247914_Quantum_Annealing_and_Related_Optimization_Methods

[3] https://github.com/qiskit-community/open-science-prize-2021/blob/main/ibmq-qsim-sup-mat.ipynb

[4] https://github.com/qiskit-community/open-science-prize-2021/blob/main/ibmq-qsim-challenge.ipynb

[5] J. Ferrando-Soria, S. A. Magee, A. Chiesa, S. Car- retta, P. Santini, I. J. Vitorica-Yrezabal, F. Tuna, G. F. Whitehead, S. Sproules, K. M. Lancaster, A.-L. Barra, G. A. Timco, E. J. McInnes, R. E. Winpenny, Chem 2016, 1 , 727.

[6] Francesco Tacchino, Alessandro Chiesa, Stefano Carretta, Dario Gerace. 8 Jul 2019. Quantum computers as universal quantum simulators: state-of-art and perspectives. https://arxiv.org/abs/1907.03505

4 Appendix

4.1 Raw data

Table 1 contains the data for the first trotter decomposition and table

Number of trotter steps	Error-free simulation fidelity	Noisy simulation fidelity
4	0.0021031165466949444	0.04580906649633155
5	0.25496446389426364	0.18467260753659914
6	0.5671824090504947	0.2921104895904907
7	0.7538166934529676	0.3278757388955661
8	0.8468847928493274	0.31775360428509175
9	0.9003173879541906	0.32071777535116697
10	0.9374701504939451	0.28689358667412423
11	0.954128717517863	0.2671593969696603
12	0.9638315965018874	0.2500496819773973
13	0.9708784851764108	0.2323853808128836
14	0.9718274058565327	0.2318760756408678
15	0.9813165701603399	0.21685303048368573
16	0.9814188130010469	0.20117113307668869
17	0.9856160011303932	0.19516181334882005
18	0.9857798579724723	0.17479731883625962
19	0.9870723174803319	0.17487187979018698
20	0.9871057358601508	0.16761545322811872
21	0.9864026842068171	0.15269200057625262
22	0.9892969490653968	0.1473019380742204
23	0.9909473172337464	0.1504939637378226
24	0.9874248278543186	0.1460248387336429
25	0.9888538040178718	0.147636598079782
26	0.9850361626700561	0.1490875744164054
27	0.9878768716655355	0.1468416083165825
28	0.9847417474329817	0.1474017526246159

Table 1: Fidelity results raw data

Number of trotter steps	Error-free simulation fidelity	Noisy simulation fidelity
4	0.002909086294269409	0.03644010697868956
5	0.25319889404029533	0.21307631511017205
6	0.5675534379308356	0.4008868116390052
7	0.7529624621419745	0.4573556662359596
8	0.8432362183566051	0.4969143062065654
9	0.9006890421961892	0.4920774542534445
10	0.9332005576074343	0.4878318441181584
11	0.9480530331885794	0.4494478859138808
12	0.9700165604553097	0.43018012020482366
13	0.9734138843676051	0.4086284949378489
14	0.9780150449137837	0.3966353770265542
15	0.9797797577036413	0.36617586188349227
16	0.9855394813166638	0.36133917689394507
17	0.9849223116835327	0.33919966812855007
18	0.9865596357024198	0.31710338888315126
19	0.9864015382369319	0.3085714919172886
20	0.9838352626635579	0.3097756524027505
21	0.9885699903462656	0.2881064096991076
22	0.9867026974344956	0.27426976827321164
23	0.9818892899054602	0.25642750603270525
24	0.9823251506368268	0.258445899956959
25	0.9848289023472557	0.25620049099994346
26	0.9888525832064006	0.24286976019158432
27	0.9866106194954485	0.23074486863610202

Table 2: Data for the more advanced trotter decomposition

4.2 More information about ZZ and YY operations

The following calculations utilise Euler's formula, $e^{i\theta} = \cos(\theta) + i\sin(\theta)$

4.2.1 Proofs via matrix calculations

$$
\begin{aligned}
XX(2t) = e^{-it\sigma_x\otimes\sigma_x} &= \begin{pmatrix} \cos(t) & 0 & 0 & -i\sin(t) \\ 0 & \cos(t) & -i\sin(t) & 0 \\ 0 & -i\sin(t) & \cos(t) & 0 \\ -i\sin(t) & 0 & 0 & \cos(t) \end{pmatrix} \\
&= \mathrm{R}_y(\frac{\pi}{2})^{(0)} \otimes \mathrm{R}_y(\frac{\pi}{2})^{(1)} \cdot ZZ(2t) \cdot \mathrm{R}_y(-\frac{\pi}{2})^{(0)} \otimes \mathrm{R}_y(-\frac{\pi}{2})^{(0)} \\
&= \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} e^{-it} & 0 & 0 & 0 \\ 0 & e^{it} & 0 & 0 \\ 0 & 0 & e^{it} & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix} \cdot \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \\
&= \frac{1}{2}\begin{pmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} e^{-it} & 0 & 0 & 0 \\ 0 & e^{it} & 0 & 0 \\ 0 & 0 & e^{it} & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix} \frac{1}{2}\begin{pmatrix} 1 & -1 & -1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos(t) & 0 & 0 & -i\sin(t) \\ 0 & \cos(t) & -i\sin(t) & 0 \\ 0 & -i\sin(t) & \cos(t) & 0 \\ -i\sin(t) & 0 & 0 & \cos(t) \end{pmatrix}
\end{aligned}
$$

$$
\begin{aligned}
YY(2t) = e^{-it\sigma_y \otimes \sigma_y} &= \begin{pmatrix} \cos(t) & 0 & 0 & i\sin(t) \\ 0 & \cos(t) & -i\sin(t) & 0 \\ 0 & -i\sin(t) & \cos(t) & 0 \\ i\sin(t) & 0 & 0 & \cos(t) \end{pmatrix} \\
&= \mathrm{R}_x(\frac{\pi}{2})^{(0)} \otimes \mathrm{R}_x(\frac{\pi}{2})^{(1)} \cdot ZZ(2t) \cdot \mathrm{R}_x(-\frac{\pi}{2})^{(0)} \otimes \mathrm{R}_x(-\frac{\pi}{2})^{(1)} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & -i \\ -i & 1 \end{pmatrix} \cdot \begin{pmatrix} e^{-it} & 0 & 0 & 0 \\ 0 & e^{it} & 0 & 0 \\ 0 & 0 & e^{it} & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix} \cdot \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \otimes \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix} \\
&= \frac{1}{2} \begin{pmatrix} 1 & -i & -i & -1 \\ -i & 1 & -1 & -i \\ -i & -1 & 1 & -i \\ -1 & -i & -i & 1 \end{pmatrix} \begin{pmatrix} e^{-it} & 0 & 0 & 0 \\ 0 & e^{it} & 0 & 0 \\ 0 & 0 & e^{it} & 0 \\ 0 & 0 & 0 & e^{-it} \end{pmatrix} \frac{1}{2} \begin{pmatrix} 1 & i & i & -1 \\ i & 1 & -1 & i \\ i & -1 & 1 & i \\ -1 & i & i & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos(t) & 0 & 0 & i\sin(t) \\ 0 & \cos(t) & -i\sin(t) & 0 \\ 0 & -i\sin(t) & \cos(t) & 0 \\ i\sin(t) & 0 & 0 & \cos(t) \end{pmatrix}
\end{aligned}
$$

4.2.2 More algebraic methodology

The first working for the XX function is taken from IBMs supplementary material for the open science prize[3]. The working for the YY function was adapted from that by me. The working below uses the fact that $R_y(\pi/2)\sigma_z R_y(-\pi/2) = \sigma_x$ and that $R_x(\pi/2)\sigma_z R_x(-\pi/2) = \sigma_y$

$$
\begin{aligned}
\left(R_y^{(0)}(\pi/2) \otimes R_Y^{(1)}(\pi/2)\right) ZZ(t)^{(0,1)} \left(R_y^{(0)}(-\pi/2) \otimes R_Y^{(1)}(-\pi/2)\right) &= \\
\exp\left[\frac{-i\pi}{4}(\sigma_y^{(0)} + \sigma_y^{(1)})\right] \exp\left[\frac{-it}{2}\sigma_z^{(0)} \otimes \sigma_z^{(1)}\right] \exp\left[\frac{i\pi}{4}(\sigma_y^{(0)} + \sigma_y^{(1)})\right] &= \\
\exp\left[\frac{-i\pi}{4}(\sigma_y^{(0)} + \sigma_y^{(1)})\right] \left[\cos(t/2)(I^{(0)} \otimes I^{(1)}) - i\sin(t/2)(\sigma_z^{(0)} \otimes \sigma_z^{(1)})\right] \exp\left[\frac{i\pi}{4}(\sigma_y^{(0)} + \sigma_y^{(1)})\right] &= \\
\cos(t/2) \exp\left[\frac{-i\pi}{4}(\sigma_y^{(0)} + \sigma_y^{(1)})\right] (I^{(0)} \otimes I^{(1)}) \exp\left[\frac{i\pi}{4}(\sigma_y^{(0)} + \sigma_y^{(1)})\right] - i \exp\left[\frac{-i\pi}{4}(\sigma_y^{(0)} + \sigma_y^{(1)})\right] \sin(t/2)(\sigma_z \otimes \sigma_z) \exp\left[\frac{i\pi}{4}(\sigma_y^{(0)} + \sigma_y^{(1)})\right] &= \\
\cos(t/2)(I^{(0)} \otimes I^{(1)}) - \frac{i\sin(t/2)}{4}\left[\left(I^{(0)} - i\sigma_y^{(0)}\right)\sigma_z^{(0)}\left(I^{(0)} + i\sigma_y^{(0)}\right)\right] \otimes \left[\left(I^{(1)} - i\sigma_y^{(1)}\right)\sigma_z^{(1)}\left(I^{(1)} + i\sigma_y^{(1)}\right)\right] &= \\
\cos(t/2)(I^{(0)} \otimes I^{(1)}) - i\sin(t/2)(\sigma_x^{(0)} \otimes \sigma_x^{(1)}) &= \\
\exp\left[\frac{-it}{2}\sigma_x^{(0)} \otimes \sigma_x^{(1)}\right] &= \\
= XX(t)
\end{aligned}
$$

$$
\begin{aligned}
\left(R_x^{(0)}(\pi/2)\otimes R_x^{(1)}(\pi/2)\right) ZZ(t)^{(0,1)} \left(R_x^{(0)}(-\pi/2)\otimes R_x^{(1)}(-\pi/2)\right) = \\
\exp\left[\frac{-i\pi}{4}(\sigma_x^{(0)}+\sigma_x^{(1)})\right]\exp\left[\frac{-it}{2}\sigma_z^{(0)}\otimes\sigma_z^{(1)}\right]\exp\left[\frac{i\pi}{4}(\sigma_x^{(0)}+\sigma_x^{(1)})\right] = \\
\exp\left[\frac{-i\pi}{4}(\sigma_x^{(0)}+\sigma_x^{(1)})\right]\left[\cos(t/2)(I^{(0)}\otimes I^{(1)}) - i\sin(t/2)(\sigma_z^{(0)}\otimes\sigma_z^{(1)})\right]\exp\left[\frac{i\pi}{4}(\sigma_x^{(0)}+\sigma_x^{(1)})\right] = \\
\cos(t/2)\exp\left[\frac{-i\pi}{4}(\sigma_x^{(0)}+\sigma_x^{(1)})\right](I^{(0)}\otimes I^{(1)})\exp\left[\frac{i\pi}{4}(\sigma_x^{(0)}+\sigma_x^{(1)})\right] - i\exp\left[\frac{-i\pi}{4}(\sigma_x^{(0)}+\sigma_x^{(1)})\right]\sin(t/2)(\sigma_z\otimes\sigma_z)\exp\left[\frac{i\pi}{4}(\sigma_x^{(0)}+\sigma_x^{(1)})\right] = \\
\cos(t/2)(I^{(0)}\otimes I^{(1)}) - \frac{i\sin(t/2)}{4}\left[\left(I^{(0)}-i\sigma_x^{(0)}\right)\sigma_z^{(0)}\left(I^{(0)}+i\sigma_x^{(0)}\right)\right]\otimes\left[\left(I^{(1)}-i\sigma_x^{(1)}\right)\sigma_z^{(1)}\left(I^{(1)}+i\sigma_x^{(1)}\right)\right] = \\
\cos(t/2)(I^{(0)}\otimes I^{(1)}) - i\sin(t/2)(\sigma_y^{(0)}\otimes\sigma_y^{(1)}) = \\
\exp\left[\frac{-it}{2}\sigma_y^{(0)}\otimes\sigma_y^{(1)}\right] = \\
= YY(t)
\end{aligned}
$$

4.3 Fidelity

The exact method for the fidelity analysis would require a deeper understanding of quantum computation than offered in this investigation. In short, when measurements are done on a quantum computer it is impossible to capture all the information about the quantum state, therefore quantum tomography is used. It runs the program many times and measures the state in different ways to get more information about the state. Further information can be found from the documentation for the method used: https://qiskit.org/documentation/tutorials/noise/8_tomography.html

4.4 Code

```
import numpy as np
import matplotlib.pyplot as plt

from qiskit import QuantumCircuit, QuantumRegister, IBMQ, execute,
    transpile
from qiskit.providers.aer import QasmSimulator
from qiskit.ignis.verification.tomography import state_tomography_circuits,
     StateTomographyFitter
from qiskit.quantum_info import state_fidelity
from qiskit.opflow import Zero, One, I, X, Y, Z

import warnings
warnings.filterwarnings('ignore')

#IBMQ.save_account('MY_API_TOKEN')
provider = IBMQ.load_account()

provider = IBMQ.get_provider(hub='ibm-q-community', group='ibmquantumawards
    ', project='open-science-22')
jakarta = provider.get_backend('ibmq_jakarta')

sim_noisy_jakarta = QasmSimulator.from_backend(provider.get_backend('
    ibmq_jakarta'))
sim = QasmSimulator()


def heisenberg_matrix():
    identity = np.eye(2, 2)
    pauli_x = np.array([[0, 1], [1, 0]])
    pauli_y = np.array([[0, -1j], [1j, 0]])
    pauli_z = np.array([[1, 0], [0, -1]])

    xx_gates = np.kron(identity, np.kron(pauli_x, pauli_x)) + np.kron(
        pauli_x, np.kron(pauli_x, identity))
```

```
30      yy_gates = np.kron(identity, np.kron(pauli_y, pauli_y)) + np.kron(
            pauli_y, np.kron(pauli_y, identity))
31      zz_gates = np.kron(identity, np.kron(pauli_z, pauli_z)) + np.kron(
            pauli_z, np.kron(pauli_z, identity))
32
33      hamiltonian = xx_gates + yy_gates + zz_gates
34
35      return hamiltonian
36
37
38  def gen_xx(t):
39      xx_qc = QuantumCircuit(2, name='XX')
40
41      xx_qc.ry(np.pi / 2, [0, 1])
42      xx_qc.cnot(0, 1)
43      xx_qc.rz(2 * t, 1)
44      xx_qc.cnot(0, 1)
45      xx_qc.ry(-np.pi / 2, [0, 1])
46
47      return xx_qc.to_instruction()
48
49
50  def gen_yy(t):
51      yy_qc = QuantumCircuit(2, name='YY')
52
53      yy_qc.rx(np.pi / 2, [0, 1])
54      yy_qc.cnot(0, 1)
55      yy_qc.rz(2 * t, 1)
56      yy_qc.cnot(0, 1)
57      yy_qc.rx(-np.pi / 2, [0, 1])
58
59      return yy_qc.to_instruction()
60
61
62  def gen_zz(t):
63      zz_qc = QuantumCircuit(2, name='ZZ')
64
```

```
65      zz_qc.cnot(0, 1)
66      zz_qc.rz(2 * t, 1)
67      zz_qc.cnot(0, 1)
68
69      return zz_qc.to_instruction()
70
71
72  def gen_cc(t):
73      cc_qc = QuantumCircuit(2, name='CC')
74
75      cc_qc.cnot(0, 1)
76      cc_qc.rx(2 * t - np.pi / 2, 0)
77      cc_qc.rz(2 * t, 1)
78      cc_qc.h(0)
79      cc_qc.cnot(0, 1)
80      cc_qc.h(0)
81      cc_qc.rz(-2 * t, 1)
82      cc_qc.cnot(0, 1)
83      cc_qc.rx(np.pi / 2, 0)
84      cc_qc.rx(-np.pi / 2, 1)
85
86      return cc_qc.to_instruction()
87
88
89  def og_trot_step(t):
90      trot_qc = QuantumCircuit(3, name='Trot')
91
92      for i in range(2):
93          trot_qc.append(gen_xx(t), [i, i + 1])
94          trot_qc.append(gen_yy(t), [i, i + 1])
95          trot_qc.append(gen_zz(t), [i, i + 1])
96
97      return trot_qc.to_instruction()
98
99  def fresh_trot_step(t):
100     trot_qc = QuantumCircuit(3, name='Trot')
101
```

```
102     for i in range(2):
103         trot_qc.append(gen_cc(t), [i, i + 1])
104 
105     return trot_qc.to_instruction()
106 
107 
108 def gen_trot_circ(steps, trot_decomp):
109     t = np.pi/steps
110     final_circ = QuantumCircuit(7)
111 
112     # Prepare |110> for qubits 1, 3, 5
113     final_circ.x([3, 5])
114 
115     for step in range(steps):
116         final_circ.append(trot_decomp(t), [1, 3, 5])
117 
118     return state_tomography_circuits(final_circ, [1, 3, 5])
119 
120 
121 def run_circuits(n_points, backend, shots, trot_decomp):
122     q_jobs = []
123     for t_steps in range(n_points):
124         st_qcs = gen_trot_circ(t_steps + 4, trot_decomp)
125         job = execute(st_qcs, backend, shots=shots)
126         q_jobs.append(job)
127 
128     return q_jobs, st_qcs
129 
130 
131 def tomographies(q_jobs, st_qcs):
132     target_state = (One ^ One ^ Zero).to_matrix()
133     fids = []
134 
135     for job in q_jobs:
136         tomo_fitter = StateTomographyFitter(job.result(), st_qcs)
137         rho_fit = tomo_fitter.fit(method='lstsq')
138         fid = state_fidelity(rho_fit, target_state)
```

```
139             fids.append(fid)
140
141         return fids
142
143
144 def plotter(data, titl, num):
145         xs = []
146         for n in range(len(data)):
147             xs.append(n + 4)
148
149         plt.figure(num)
150         plt.plot(xs, data)
151         plt.xlabel('Number of trotter steps')
152         plt.ylabel('Fidelity of result')
153         plt.title(titl)
154
155
156 #print(heisenberg_matrix())
157
158 n_of_points = 24
159 ideal_q_jobs, i_st_qcs = run_circuits(n_of_points, sim, 1024,
        fresh_trot_step)
160 noisy_q_jobs, n_st_qcs = run_circuits(n_of_points, sim_noisy_jakarta, 1024,
         fresh_trot_step)
161
162 ideal_data = tomographies(ideal_q_jobs, i_st_qcs)
163 noisy_data = tomographies(noisy_q_jobs, n_st_qcs)
164
165 print(ideal_data)
166 print(noisy_data)
167
168 plotter(ideal_data, 'Fidelity of result per number of trotter steps in an
        error free simulation', 1)
169 plotter(noisy_data, 'Fidelity of result per number of trotter steps in a
        noisy simulation', 2)
170 plt.show()
```