Simulating Hamiltonian Dynamics Using Product Formulas

Omar Alsaeed

oalsaeed@middlebury.edu

Thesis Report for PHYS 0705 [†]

May 20, 2021

Simulating the dynamics of quantum systems classically requires resources that scale exponentially with the system size and evolution time. Quantum computers provide an efficient alternative. Algorithms running on quantum computers offer dramatic speedups that, in theory, allow us to consider systems, timescales and resolutions that are larger than anything computable using classical computers. Here we detail a quantum algorithm for simulating Hamiltonian dynamics using the Trotter-Suzuki product formulas [1]. We analyze error scaling and discuss methods to optimize it. We explore randomization techniques to improve the algorithm's performance. We present examples using product formulas in Python on the evolution of spin-$1/2$ particles.

Date Accepted: _____

[2]Department of Physics, Middlebury College, Middlebury, VT

Department of Physics and Astronomy, Dartmouth College, Hanover, NH

Contents

1 Introduction

The objective of this thesis is to explore the Hamiltonian simulation problem, and the effectiveness of the Trotter-Suzuki decomposition method in solving it. We provide a simple overview of the algorithm with a recipe for writing an explicit quantum circuit to approximate the time evolution operator appropriate for undergraduates interested in quantum simulations.

Here we review a prominent quantum algorithm for simulating the dynamics of quantum systems due to a Hamiltonian that is a sum of terms. We first show the importance of the time evolution operator in quantum simulations and the need for approximation methods in Sect. 2. Then we describe the process of constructing an algorithm to obtain the approximants, decomposing the evolution of the sum into a product of evolutions using the Trotter-Suzuki decomposition in Sect. 3 [2]. We first review constructing first-order approximants, and the segmentation of the evolution, then constructing higher order approximants using a recursive way, namely the fractal decomposition [1]. We analyze the effectiveness of the algorithm in Sect. 4 using examples of time evolution in quantum dynamics: the precession of the angular momentum or "spin" of a spin-$1/2$ particle in the presence of a constant magnetic field, and the evolution of a two-qubit Ising system in the presence of a uniform magnetic field. Lastly, we explore the effect of randomizing the order in which the exponentials product is carried out. We finally discuss the results in Sect. 5 and show that the overall findings suggest that the Trotter-Suzuki decomposition allow for the efficient simulation of quantum dynamics and requires time resources that scale sub-linearly in time which is consistent with the widely accepted query complexity [3].

2 Background

The exponential operator is a matrix function on square matrices analogous to the ordinary exponential function. For a square matrix M of dimensions $n \times n$, the exponential operator of M is defined in

terms of its Taylor series

$$e^M = \sum_{k=0}^{\infty} \frac{M^k}{k!}, \tag{1}$$

where M^0 is defined to be the identity matrix I with the same dimensions as M ($n \times n$). The exponential operator is used to solve systems of linear differential equations of the form

$$\frac{\partial}{\partial t} f(t) = Mf(t), \tag{2}$$

where f is a function or a vector and M is an operator -denoted with a hat- or a matrix. The equation of this form we are interested in is the Schrödinger equation

$$i\hbar \frac{\partial}{\partial t} |\psi(t)\rangle = \hat{H} |\psi(t)\rangle. \tag{3}$$

This is the fundamental equation of motion governing quantum dynamics. It determines how the state of a quantum system evolves in time in quantum mechanics. Thus, it describes the dynamics of elementary, subatomic, atomic, and molecular particles. Here we have introduced the use of the bra-ket notation where a ket $|\phi\rangle$ is a column vector representing the state of the wave function ϕ of a system: known as a state vector, and a bra $\langle\phi|$ is the conjugate transpose of the ket vector: a row vector. One way the equation can be derived is by starting with the assumption that the time evolution operator $\hat{U}(t)$, which translates the state of a system forward in time, must be unitary: the unitarity of the time evolution operator is a fundamental postulate in quantum mechanics. The unitarity of the time evolution operator dictates that it, therefore, must be generated by a Hermitian operator (an observable): the energy operator known as the Hamiltonian. Thus, the Hamiltonian operator $\hat{H}$ is the generator of time translations and is a mathematical description of a system's energy. If the Hamiltonian is time-independent $\hat{H}(t) = \hat{H}$ we can obtain a closed-form expression from an infinite series of infinitesimal time translations

$$\hat{U}(t) = \lim_{N \to \infty} [1 - \frac{i}{\hbar} \hat{H} \frac{t}{N}]^N, \tag{4}$$

If compare this Taylor series expansion to that of Eq. 1 (taking $M = \frac{i}{\hbar}\hat{H}t$) we recognize that they are equal. Thus we can write the time evolution operator in the form

$$\hat{U}(t) = e^{-i\hat{H}t/\hbar}. \tag{5}$$

This is the time evolution operator which simulates a quantum system's evolution in time t under a time-independent Hamiltonian $\hat{H}$.

Thus, the solution of the Schrödinger equation for a time-independent Hamiltonian is simply

$$|\Psi(t)\rangle = \hat{U}(t)|\Psi(0)\rangle. \tag{6}$$

This means that in order to solve the the Schrödinger equation when the Hamiltonian is time independent $\hat{H}$, all we need to know is two things: i) the initial state of the system $|\Psi(0)\rangle$, and ii) to be able to work out the action of the operator (6) on this state.

Consider that the size of the Hamiltonian for a n-particle system of spin-$1/2$ particles would have the dimensions $2^n \times 2^n$. For example, a 300-particle system would have a Hamiltonian that has the dimensionality $\sim 2 \times 10^{90}$ which is more than the number of atoms in the observable universe (Eddington's number: $\sim 10^{80}$). Since the exponentiation of matrices is essential for studying quantum dynamics, as well as solving linear differential equations in general, approximation methods are necessary. However, the resources - such as time and memory space - that classical numerical methods require grow exponentially to the system size and evolution time on classical computers. Therefore, to offer an efficient alternative, quantum algorithms were developed [4].Thus, being able to efficiently exponentiate matrices on quantum computers can open up the door for calculations that were once thought impossible. The first quantum algorithm developed for this purpose is based on a mathematical tool known as the Trotter-Suzuki product formula or decomposition, together with its higher order generalizations.

In the present thesis, we review the process of using the Trotter-Suzuki decomposition, or the exponential product formula or simply the product formula (PF), to break down the difficult task

of simulating quantum dynamics into manageable pieces. That will allow us to simulate quantum dynamics more efficiently, amongst the various applications of this decomposition. The simplest Trotter-Suzuki decomposition is given by

$$e^{x(A+B)} = e^{xA}e^{xB} + O(x^2), \tag{7}$$

which follows from Trotter's formula

$$e^{x(A+B)} = \lim_{n\to 0} (e^{xA/n}e^{xB/n})^n, \tag{8}$$

where x is a parameter and A and B are arbitrary bounded operators with some commutation relation $[A,B] \neq 0$ [2]. Here the product of the exponential operators on the right-hand side of Eq. 7 is regarded as an approximate decomposition of the exponential operator on the left-hand side with correction terms of the second order of x. Thus, we call the product of the exponential operators on the right-hand side a first-order approximant $\hat{U}_1$ in the sense that it is correct up to the first-order of x. The higher-order corrections take different forms. We here review the symmetrized higher order recursive decomposition of Suzuki [1] of the form

$$e^{x(A+B)} = e^{p_1 xA}e^{p_2 xB}e^{p_3 xA}e^{p_4 xB}...e^{p_M xB} + O(x^{k+1}), \tag{9}$$

for some integer real number k denoting the order of the approximant $\hat{U}_k$ which the product of the exponential operators on the right-hand side of Eq. 9. We explore the set of parameters $\{p_1, p_2, ..., p_M\}$ so that the correction term may be of the order of x^{k+1}.

Lastly, let us discuss one more key reason that makes the Trotter-Suzuki decomposition very useful for simulating physical systems. To understand this we need to return to the Taylor series of the exponential operator and compare it the Trotter-Suzuki decomposition. The approximant (7) and the generalized one (9), in fact, have a remarkable advantage over other approximants such as

$$e^{x(A+B)} = I + x(A+B) + O(x^2), \tag{10}$$

which is the Taylor series expansion of the exponential function (1) truncated at $k = 1$. The approximant of the form (9) conserves an important symmetry of the system in problems of quantum dynamics.

As mentioned earlier, the unitarity of the time evolution operator $e^{-i\hat{H}t}$ is a postulate of quantum mechanics. The unitarity of the time evolution operator leads to the preservation of the wave function probability $||\Psi||^2$; hence the norm of the wave function $||\Psi||$ is preserved. One way the preservation of the norm of the wave function. That can be understood as a consequence of the conservation of charge; which is a consequence of symmetry through Noether's theorem: the gauge invariance of the electromagnetic field. In quantum mechanics, that means that a phase shift in the wave function of a charged particle is unobservable. It is important to emphasize that the exponential product (1) is unitary. However, the perturbational approximant (10) is not. In fact, the norm typically increases monotonically as the time evolves as we demonstrate in Sect. 4.

In summary, the Lie-Trotter-Suzuki product formulas allow for the simulation of the dynamics of a Hamiltonian that is a sum of terms while requiring polynomial resources in evolution time, at worst. In other words, it allows for faster quantum simulations on classical and quantum computers. In the classical case, it brings down the exponential scaling in resources to some polynomial scaling. In the quantum case, it provides an algorithm for simulating the evolution of quantum systems of larger sizes and for longer times, by breaking down such hard-to-construct unitary gates into a product of smaller, say 1-qubit and 2-qubit, elementary gates that can be executed sequentially to simulate the exact evolution up to some error. In the next subsections, we take a step backwards and remind ourselves of the motivation for quantum simulations and the advantage of using quantum computers to perform the task, and finish by introducing the Hamiltonian simulation problem. Readers that are familiar with quantum mechanics, quantum computing and the Hamiltonian simulation problem can skip through to Sec. 3 glancing over the plots and the relevant equations in our notation.

2.1 Motivation

First of all, let us discuss the power of physical simulations and why we need quantum computers to perform this task that proves difficult even for the best of our good old companion of modern civilization: the classical computer. The universe we live in, some might say, is a grand quantum simulation. If we wish to understand the world around us we better be able to simulate its subsystems efficiently so we can gain a deeper understanding of their dynamics in time. We ought to let nature run the quantum symphony her way rather than try and come up with our own smart methods of simulating that quantum behavior classically.

Quantum simulations are necessary to understand the world around us and to be able to reach a future that is more technologically advanced than ever thought possible before. Simulating physics on a computer is a process that is indispensable in in a growing number of disciplines. To motivate the problem considered in this thesis, let us consider the work physicists do on a fundamental level. There are several ways of thinking about it, one of which is that physicists are trying to find a theory that accurately explains how the universe works. Laplace once predicted that humans would be able to simulate the future perfectly if they had access to the (classical) Hamiltonian of the system, its initial conditions, and unbounded computational power [5]. Laplace's words resonate with the quantum picture of the world. If we know the Hamiltonian (operator) of a quantum system, an accurate description of an initial state of the system, and had access to unlimited computational power, we ought to be able to predict any future state of the system! The equation that tells us how to do that is the fundamental equation of motion: the Schrödinger equation. We saw how the exponential operator appears in the formal solution of the Schrödinger equation, amongst various other similarly important differential equations governing the physics of the world around us. In this thesis, we restrict ourselves to time-independent Hamiltonians.

In this case, the solution to the Schrödinger equation is simply Eq. 5 where H is the Hamiltonian ($2^n \times 2^n$ Hermitian matrix), $|\Psi(0)\rangle$ and $|\Psi(t)\rangle$ are 2^n-dimensional vectors representing the initial and

final states of a system of n particles respectively, and t is the time difference between the two states. In other words, to obtain the state of a quantum system at any subsequent time to the given initial state on a classical computer, one inputs the time over which they wish to evolve the system and the Hamiltonian governing the dynamics of the system as time evolves. A classical computer can and will simulate the evolution of the system under these physical laws. It is known that the classical description is inaccurate in a wide range of systems and applications. Quantum mechanics, however, provides an accurate picture of nature. Moreover, quantum mechanics is responsible for the rise of numerous phenomena that classical physics cannot predict. Therefore, quantum simulations are extremely valuable and are essential to predicting the dynamics of such systems of interest, where quantum mechanics dominates.

However, the problem with simulating quantum systems on classical computers is the computational complexity. Computational complexity is the amount of resources an algorithm needs based on various aspects of the input, mainly the size of the input and the evolution time. Consequently, the obstacle standing between humanity and a simulation of the very same quantum world we live in is the computational complexity. Inherent scaling limits of classical algorithms hinders our ability to simulate complex systems [6]. The resource requirements of classical algorithms simulating quantum dynamics increase exponentially with problem size, as is the case when simulating the time evolution of quantum systems. For example, the time a classical algorithm requires to make a meaningful computation on the evolution of a quantum system of ~ 300 particles over a few seconds might be on the order of the age of the universe ($\sim 10^{17}$ seconds). In other words, the complex space containing all the possible states of a quantum system, known as the Hilbert space, gains one extra dimension for each extra particle considered, giving 2^n possible states for a n-qubit system. Thus, a classical computer requires exponential resources to simulate quantum states. No exact classical algorithm can be developed to overcome this inherent "wall" of complexity for simulating quantum systems. The meaning of the previous sentence might not be clear, since we have not formally defined what it

means to simulate a quantum system, yet. If the output is to be a description of the quantum state, even approximately, this would require exponential space and time to write down [7]. Now the claim is sound: all known classical algorithms for the Hamiltonian simulation problem run in exponential time. Moreover, if this problem had an efficient classical algorithm, then quantum computers, which are quantum systems, would have efficient classical simulations. For example, this would mean that integer factorization has an efficient classical algorithm [8].

Indeed, it was the exponential time complexity of simulating quantum systems on a classical computer that led Feynman, amongst others, to propose the idea of quantum computation to efficiently simulate quantum systems: Use controllable quantum systems in lab (quantum computers) to simulate the dynamics of other quantum systems we wish to study [9]. An ideal implementation of such a computer would have enough qubits ($\sim$ hundreds) that can be maintained in coherence long enough ($\sim$ seconds) to perform a meaningful computation. For example, that few-particles system simulation should take such a quantum computer time that is on the order of seconds. To put it differently, a machine that can probe quantum mechanical systems while maintaining their coherence long enough for meaningful computations to be performed would harness the powers of quantum mechanics to solve quantum mechanical problems. Thus, the prevalence of quantum computing might allow humanity to see straight through this *apparent* wall of complexity, opening up new horizons for innovation and discovery. A solution to the Hamiltonian simulation problem has wide implications for ares of science such as chemistry and condensed matter, and is of industrial relevance in the engineering of new pharmaceuticals, catalysts, and materials [10]. As a result, there is now a cross-industry race toward quantum applications. Within five years, it is possible quantum computing will be used extensively by new categories of professionals and developers to solve problems once considered unsolvable [11].

2.2 Quantum computing

Quantum mechanics is what governs the behavior of the small; describing the properties of nature on an atomic scale. Mathematically, quantum mechanics has various formalisms based on a few fairly simple axioms. In one of them, the wave function Ψ is a mathematical description (complex-valued function) of the quantum state of an isolated quantum system. Its modulus square provides information about the probability density of the energy, momentum, and other physical properties of the system. Moreover, a quantum state, usually denoted $|\Psi\rangle$ in bra-ket notation, can be represented as a state vector in a multi-dimensional Hilbert space. The action a quantum mechanical operator has on a quantum system can be represented as a linear transformation on the state vectors of the system.

Quantum computation is a model of computation based on quantum mechanics. It studies the application of quantum phenomena, that are beyond the scope of classical approximation, for information processing and communication. Various models of quantum computation exist, however the most popular models incorporate the concepts of qubits and quantum gates. In that model, the most fundamental piece of information is a quantum bit or a "qubit". Just like a bit, a qubit is a two-level system representing the state of some other two-level system (e.g. representing the spin of an electron being in the down/up state) in the computational basis $\{|0\rangle, |1\rangle\}$. The only difference is that this two-level system is quantum mechanical, just like the system it is simulating. This chiefly means two things: the state of the system may be in a *superposition* of all its eigenstates, and that the system can be *entangled* with other qubits, including itself, where a qubit's properties are affected by what happens to the other qubits with which it is entangled. The power of superposition allows the creation of an immense number of parallel computational branches, while entanglement utilizes a novel computational layer of reality realizing higher correlations than ever possible classically. These two quantum mechanical phenomena are the fundamental principles quantum computers utilize to achieve the speedups they exhibit over classical computers.

2.3 Hamiltonian simulation

Quantum simulations are computational simulations of quantum systems. They provide valuable information about the Hamiltonian dynamics of quantum systems of interest. The Hamiltonian simulation problem is essentially solving the Schrödinger equation. We are given a Hamiltonian operator that governs the dynamics of some quantum system and we are asked to find the time evolution operator of the quantum state over some time t and apply it on the initial state to obtain the final state of the system, see Fig. 1. The generality of the Hamiltonian simulation problem allows for the use

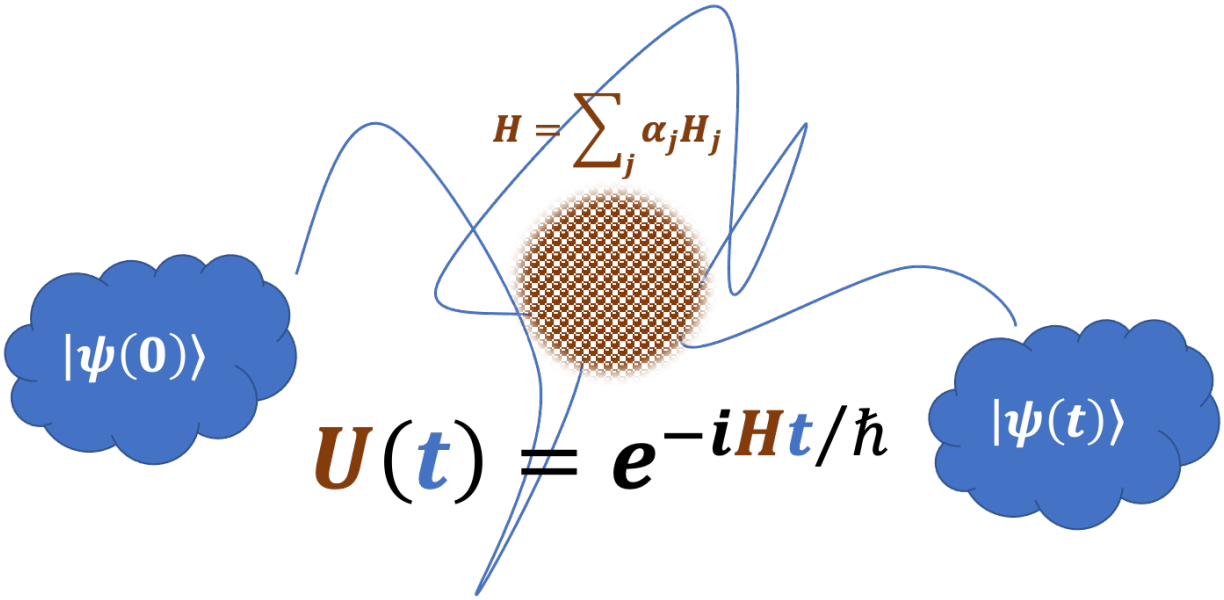


Figure (1) Depiction of the simulation of the time evolution operator of some arbitrary quantum system (6) governed by a Hamiltonian that is a sum of terms to find the final state of the system.

of Hamiltonian simulation algorithms not only to simulate the dynamics of physical systems, but also as a subroutine in numerous quantum algorithms of various applications. Formally, a quantum simulation problem occurs when we have a quantum system that we are interested in knowing something about one of its future states $|\psi(t)\rangle$. For that, one needs a description of the initial state $|\psi(0)\rangle$ of the system, the evolution time t, and the Hamiltonian $\hat{H}$ governing its dynamics

Given the initial state of a quantum system $|\psi(0)\rangle$, **approximate** the evolution operator $e^{-i\hat{H}t/\hbar}$ for Hamiltonian $\hat{H}$ over time t to find the final state of the system $|\psi(t)\rangle$.

Nevertheless, as discussed earlier, the simulation of Hamiltonian evolution in time is almost impossible to solve analytically for relatively large systems. Similarly trouble-some, performing the simulation numerically on classical computers can quickly require enormous resources and become practically intractable. On the other hand, applying an exact evolution gate of a Hamiltonian that is a sum of terms on all the qubits of the system it describes can be unattainable with near-term quantum computing. Thus, mathematical methods to find approximations were developed. That allows to bring down the exponential scaling of simulating quantum dynamics on classical computers to a polynomial scaling, and enables us to break down that one quantum gate representing an exact time evolution of a system of qubits into a sequential product of easier-to-preform gates on subsets of the qubits. The Troter-Suzuki product formulas are one of the prominent examples. However, as is the case with all approximations when taken out of the ideal limits they assume, using the product formulas to approximate the evolution of certain systems of interest introduces error into the time evolution operator. The error $\varepsilon > 0$ the time evolution operator incurs for an approximate evolution $\tilde{U}(t)$ shall be defined as

$$\varepsilon \geq \|\hat{U}(t) - \hat{\tilde{U}}(t)\|_2 \tag{11}$$

where $\|\|.\|\|_2$ is the matrix 2-norm of the difference between the exact evolution operator $\hat{U}(t)$ and approximate one $\hat{\tilde{U}}(t)$. Refer to the Primer for the matrix 2-norm definition 8.2. The matrix norm $\|\|.\|\|_2$ returns the largest singular value; it describes the deviation between an exact evolution and an approximation of it. This metric provides an appropriate indicator of the error between our target matrix and our approximation; it will be used in this paper as an error function of time. A diagram depicting the Hamiltonian simulation problem, incorporating the approximate nature of the obtained solutions, can be found in Fig. 2.

There is a class of Hamiltonians that occur widely in nature and algorithmic applications: k-local

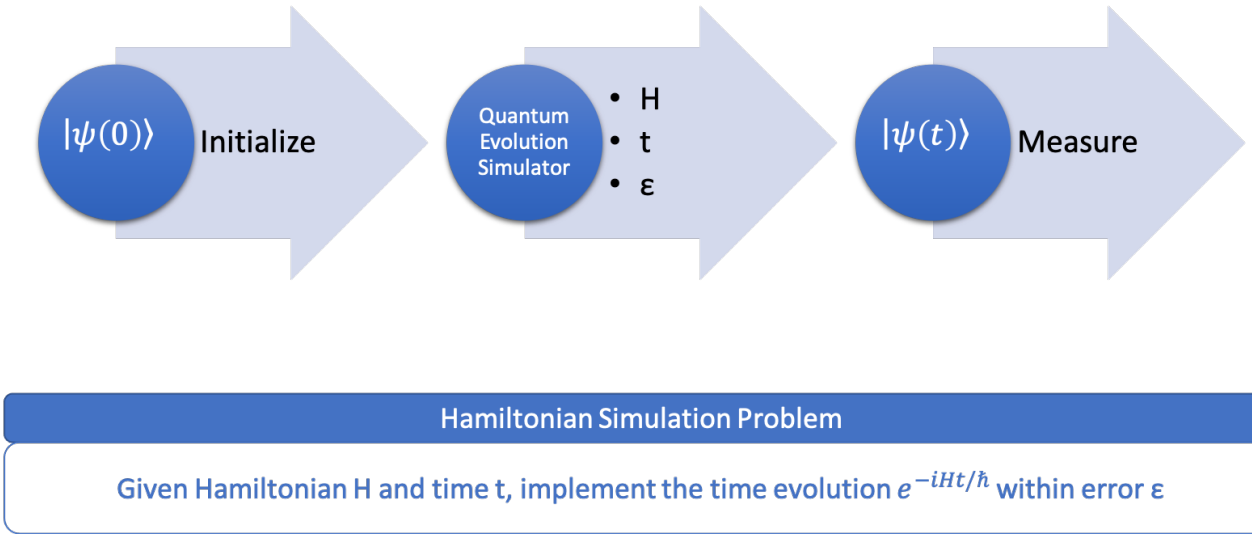


Figure (2) Depiction of the Hamiltonian simulation problem. To simulate the Hamiltonian dynamics of a quantum system, one needs a description of an initial state of the system $|\psi(0)\rangle$ and a description of the Hamiltonian H, evolution time t and the allowed error ε. Then, the quantum simulator evolves the Hamiltonian under the given conditions, incurring at most error ε as defined in Eq. 11, and applies the evolution $e^{-i\hat{H}t/\hbar}$ on the initial state of the system outputting the final state $|\psi(t)\rangle$.

Hamiltonians. One should not confuse this k with the k referring to the order of a product formula approximant. A k-local Hamiltonian is a Hamiltonian that is a linear combination of m k-local terms

$$\hat{H} = \sum_{j=1}^{m} \alpha_j \hat{H}_j, \tag{12}$$

each weighted by its coefficient α_j, where each term acts on at most k particles in the system, or qubits. Local Hamiltonians arise commonly in studying systems of nature, namely those whose Hamiltonian is a sum of local interactions acting nontrivially on a small number of subsystems of limited dimension [4], such as two-body systems. Generally, any local Hamiltonian can be decomposed into a sum of sparse terms. Thus, another parameter that characterizes a Hamiltonian is its sparsity d where a d-sparse Hamiltonian has at least d non-zero elements in any row/column of its matrix representation. The norms of the Hamiltonian $||H||$ set the energy scale for the time evolution of the system, where

the spectral norm of the Hamiltonian determines the maximum energy driving the evolution in time. Lastly, the dimensionality N of the Hamiltonian sets the size of the system's Hilbert space.

Quantum algorithms developed for simulating Hamiltonian dynamics - see Fig. 3 fall in two general categories: divide and conquer algorithms, and quantum walk algorithms [12]. The division step of the first class of algorithms decomposes the Hamiltonian into a sum of sparse terms. The decomposition of a local Hamiltonian is "easy" and generates $O(d)$ terms for a d-sparse local Hamiltonian. The decomposition of a sparse Hamiltonian, however, is not as easy and generates $O(d^2)$ 1-sparse terms for a d-sparse Hamiltonian using graph coloring techniques [3]. The second step, to conquer, after the Hamiltonian is decomposed into simpler terms, is to evolve each term and recombine the evolution. The use of product formulas was the sole efficient method to evolve Hamiltonians, first explicitly achieved by Lloyd in 1996 for k-local Hamiltonians [4], until the recent surge of new efficient Hamiltonian simulation algorithms. Subsequent work to that of Lloyd's considered the broader class of d-sparse Hamiltonians [13, 14, 15]. This report, however, is concerned with the first class of algorithms; the product formulas in particular.

3 Method

3.1 Problem statement

Here we formulate the Hamiltonian simulation problem statement abstractly, to captures the agility of the product formulas in tackling sparse, and thus local, Hamiltonians:

> **Given** a Hamiltonian $\hat{H}$ acting on n qubits in total as a sum of local terms, evolution time t and error $\varepsilon > 0$, **approximate** the time evolution operator $e^{-i\hat{H}t/\hbar}$ up to error ε.

In other words, given the initial state of a quantum system $|\Psi(0)\rangle$ with a time-independent Hamiltonian given as a sum of local terms as defined in Eq. 12, each of which act non-trivially on at most k qubits, implement the unitary time evolution operator $\hat{U}(t) = e^{-i\hat{H}t/\hbar}$, allowing error ε as defined in

Eq. 11. The problem statement suggests the framework depicted in Fig. 3.

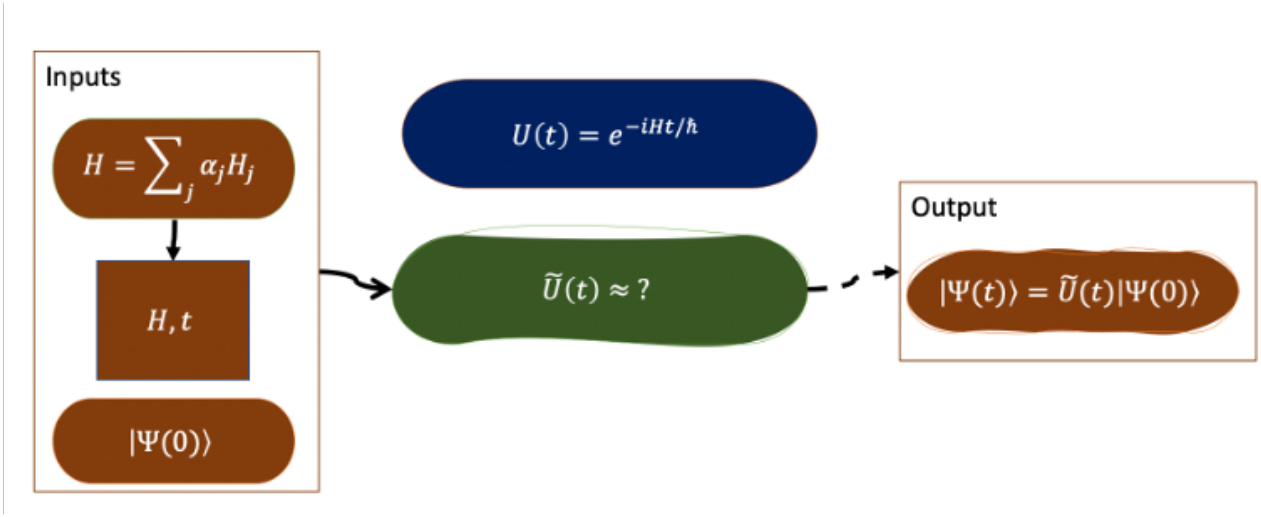


Figure (3) Diagram of the algorithmic framework for simulating the evolution of a quantum system $|\Psi(0)\rangle$ under a Hamiltonian $\hat{H} = \sum_j \alpha_j \hat{H}_j$. For the inputs: initial state, evolution time, and the Hamiltonian we need to find an algorithm that can efficiently approximate the output: the final state of the system, up to a high degree of accuracy.

3.2 Algorithm

The Trotter-Suzuki formula provides a direct approach to decomposing the exponential of a sum of operators. That makes it, with its higher-order generalizations, useful in computing numerical solutions [16]. Most importantly, product formulas have become a key tool for quantum simulation [4, 13, 3]. Hamiltonian simulation using product formulas has numerous applications in quantum information processing, including simulating quantum mechanics [4], and implementing continuous-time quantum algorithms [17].

To approximate the evolution of a quantum system using the product formulas, one can follow these three steps to construct a quantum circuit that solves the problem:

1. Write the Hamiltonian as a normalized sum of Pauli spin operator products acting on different qubits. This is assumed to be readily accessible.

2. Convert each of the operators defined in step 1 into unitary gates such that their sequential execution on a quantum computer recovers an approximation to the original unitary evolution propagator, $e^{-i\hat{H}t/\hbar}$.

3. Apply the product formula algorithm to approximate the full evolution of the Hamiltonian using the Trotter-Suzuki formula to decompose the time evolution operator.

The construction is applied to examples of the evolution of spin-half particles in Section 4 to demonstrate the third step using Python code. The next subsection provides a basic review of the fundamental steps of the product formula algorithm. A brief comment on the second step can be found in subsection C. We use atomic units throughout the thesis (such that $\hbar = m = e = 1$) in the electron's world.

3.3 Product formulas

The present section continues by describing a systematic method to perform step (3). That is to use the Trotter-Suzuki decomposition to decompose the exponential of a sum of non-commuting terms into a product of exponentials. Preserving locality is an essential propriety of the product formulas, and is largely responsible for their utility in the simulation of physical systems. Thus, product formulas provide what is perhaps the most "natural" way of simulating Hamiltonian dynamics: simulating the evolution as local interactions.

Thus, using this approximation, the construction of the time evolution operator can be efficiently carried out on a quantum computer provided that the Hamiltonian can be decomposed into a sum of terms [4], where each Hamiltonian in the sum is acting on a constant number of qubits $k = O(1)$. Decomposing the Hamiltonian into a sum of terms each acting on a constant number of qubits means each term can be implemented directly using a constant number of elementary quantum gates [18]. That allows for the possibility of simulating quantum dynamics of complex systems by breaking down the overall hard-to-perform evolution into smaller problems which can be solved with a constant

number of gates. We first demonstrate the method for the order $k = 1$ yielding, the most efficient formula, the first-order Trotter-Suzuki decomposition for a Hamiltonian of the form (12)

$$e^{-i\hat{H}t/\hbar} = e^{-i\sum_{j=1}^{m}\hat{H}_j t/\hbar} \tag{13}$$

$$= \prod_{j=1}^{m} e^{-i\hat{H}_j t/\hbar} + O(m^2 t^2), \tag{14}$$

where m is the number of terms in the sum.

The expression in Eq. 13 tells us that if all the terms making up the sum commuted, then the exponential of the sum can be expressed as a product of exponentials (without an error term). If each of the terms making up the sum is simple, say a Pauli operator, then we can implement each of these terms directly using a constant number of quantum gates and apply them sequentially to obtain the original exponential of the sum. It is important to note that this property of producing exact approximations to Hamiltonians that are a sum of commuting terms is a special feature that seems to be unique to this method. The product formula decomposition can be dramatically efficient in such cases, and that makes it a valuable asset for near-term quantum computing. Of course, in practice, terms usually do not commute, and thus error occurs. The basic tactic behind the minimal error product formulas incur on the time evolution operator, for the case when it is a complex exponential of a sum of non-commuting terms, is to use a stroboscopic approach. The evolution is broken into a series of time steps, and so making these time steps smaller makes the commutators' error smaller and smaller, leading to an error of zero in the limit the time step goes to zero. The process is repeated for a sufficiently small time step Δt such that each step $\hat{U}(\Delta t)$ is simulated with error $\varepsilon/\Delta t$. The product of the evolutions $\hat{U}(\Delta t)^{t/\Delta t}$ provides an approximation that is within error ε of the exponentiation of the sum.

Thus, to optimize error, we break the evolution into Δt short slices to control the error scaling as

a function of evolution time by varying the length of the segment Δt

$$\hat{U}(\Delta t) = e^{-i\hat{H}\Delta t/\hbar} \tag{15}$$

$$= \prod_{j=1}^{m} e^{-i\hat{H}_j\Delta t/\hbar} + O(m^2(\Delta t)^2), \tag{16}$$

We then repeat the entire product of sub-evolutions $t/\Delta t$ times to retain the full evolution of the system over time t

$$\hat{U}(t) = e^{-i\hat{H}t/\hbar} \tag{17}$$

$$= \hat{U}(\Delta t)^{t/\Delta t} \tag{18}$$

$$= (\prod_{j=1}^{m} e^{-i\hat{H}_j\Delta t/\hbar})^{t/\Delta t} + O(m^2 t\Delta t), \tag{19}$$

where $O(m^2 t\Delta t)$ is the Trotter error [1].

In other words, the expression in Eq. 17 suggests that the larger is the so-called Trotter number $Q = \frac{t}{\Delta t}$, the smaller is the error incurred due to the evolution decomposition. The Trotter number is an integer that represents the number of time steps we wish to break the evolution to or, equally, the number of times we repeat the sub-evolutions. Thus, to efficiently simulate the evolution of a Hamiltonian of non-commuting terms one must decrease the length of the Trotter step Δt, thus increasing the Trotter number Q. The same logic applies, naturally, to decreasing the evolution time while keeping the length of the Trotter step constant. Thus, the approximation is exact in the limit the length of the segments goes to zero

$$\hat{U}(t) = \lim_{\Delta t \to 0} (\hat{U}(\Delta t))^{\frac{t}{\Delta t}}, \tag{20}$$

sending the Trotter error term to zero. This is simply the Trotter formula (8) for an arbitrary sum of m terms. The general flow of the simulation of quantum systems using the first-order product formulas, incorporating the segmentation of the evolution time, is depicted in Fig. 4.

Lastly, we observe that error $O(t\Delta t)$ scaling is polynomial for a given time t and time step Δt. That is remarkable. Physically, that means one can efficiently approximate the evolution of a sparse

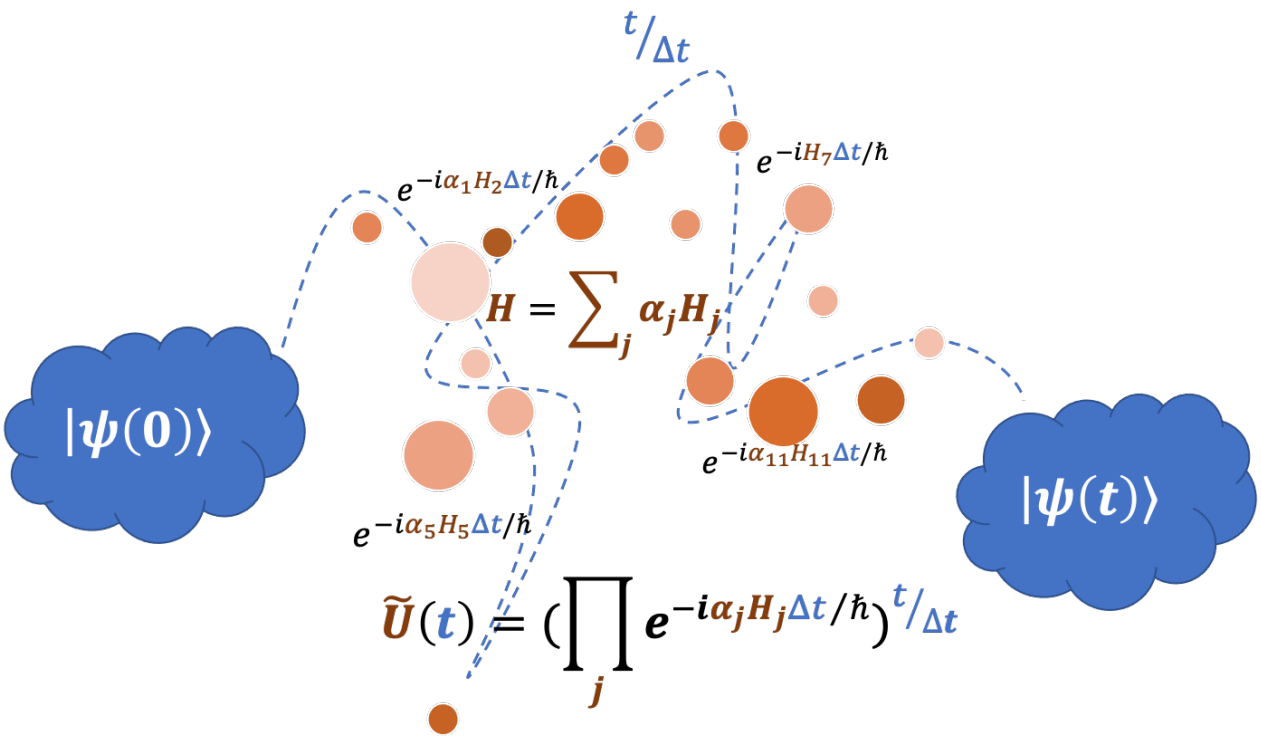


Figure (4) Depiction of the product formula algorithm simulating the evolution of Eq. 6 of a quantum system by decomposing the Hamiltonian into a sum of terms, evolving each subsystem over a small time step Δt, then reconstructing the evolution as a product of sub-evolutions to approximate the final state of the system using the first-order formula of Eq. 7 .

Hamiltonian, provided it is represented as a sum of terms, by taking the product of the evolution of each of the terms. Thus, the approximate evolution, $\hat{\tilde{U}}(t) = (e^{-i\hat{H}_1\Delta t/\hbar} e^{-i\hat{H}_2\Delta t/\hbar})^{\frac{t}{\Delta t}}$ in Eq. 13, can be applied on a given initial state of a system $|\Psi(0)\rangle$ to produce an approximate final state $|\Psi(t)\rangle$ that is at most $t\Delta t$ away from the exact final state $\varepsilon \geq t\Delta t$. In essence, this formula provides a road map for writing an algorithm that approximates the evolution of any sparse Hamiltonian represented as a sum of terms. Based on the three steps model, we develop an algorithm that performs the third step in four stages: (1) decompose the evolution of a sparse Hamiltonian into a product of the evolution of each term (2) evolve each term individually over a sufficiently small Trotter step Δt (3) recombine the sub-evolutions, and (4) repeat the process Trotter number $Q = t/\Delta t$ times to obtain solution incurring at most error $t\Delta t$. The algorithm this method suggests is depicted using its key components in Fig. 5.

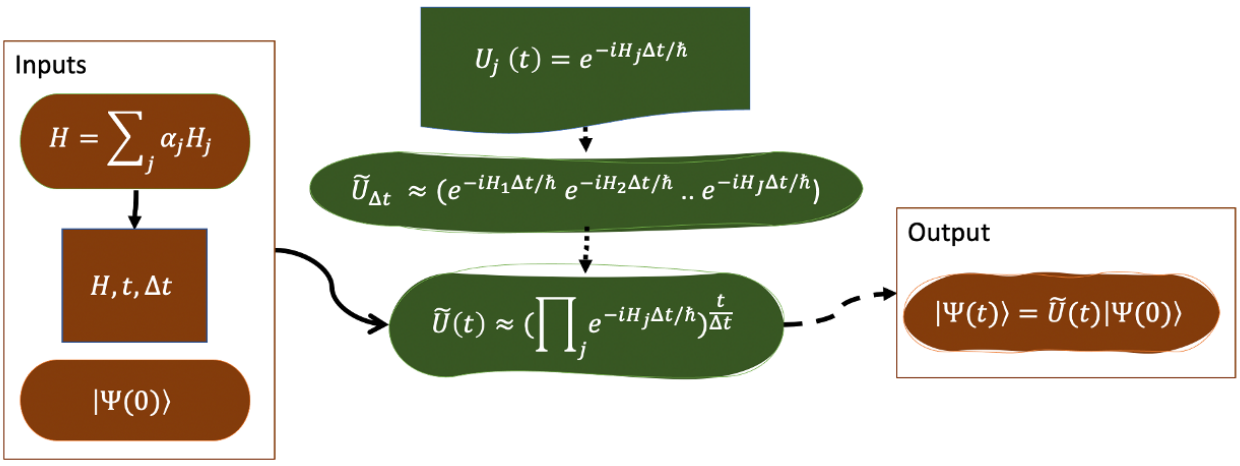


Figure (5) Diagram of the algorithmic steps of the first-order product formula method for simulating the evolution of a quantum system $|\Psi(0)\rangle$ incorporating the time segmentation: divide the evolution of each of the j terms in the Hamiltonian $\hat{H} = \sum_j \alpha_j \hat{H}_j$ into small Δt Trotter steps, each step simulated with error $\varepsilon/\Delta t$. For sufficiently large Trotter number $Q = \frac{t}{\Delta t}$, the full simulation has at most error ε.

3.4 Higher order formulas

The first-order decomposition we examined in the previous subsection performs well in the sense that it provides an approximant that is accurate up to first order in the evolution time $O(t)$. Thus the leading error term in the approximant is generally second-order $O(t^2)$. But what if want to be able to find even more accurate solutions, e.g. accurate up to second or sixth order? There are different answers to this question: there exist different formulas for this purpose. In this paper we will mostly focus on the constructions made by Suzuki and follow his presentation in the paper referenced: A set of symmetrized higher-order exponential formulas defined recursively to provide approximants with a leading error term of $O(t^{2k+1})$ for the $2k^{th}$ approximant [1]. But before we discuss higher order Trotter-Suzuki decompositions, let us remember the first-order approximant in Eq. 13 and compare it

with Suzuki's starting point for a Hamiltonian of the form (12) in this symmetrized approximant

$$e^{-i\hat{H}t/\hbar} \approx \hat{S}_2(\hat{H},t) \tag{21}$$

$$= \prod_{j=1}^{m} e^{-i\hat{H}_j t/2\hbar} \prod_{j=m}^{1} e^{-i\hat{H}_j t/2\hbar}. \tag{22}$$

We notice how the right hand side of Eq. 22 has two products running in opposite order through the indices of the terms of the Hamiltonian on the left hand side. That is because the evolution time in both is divided in half. To understand why is that, let us illustrate with a quantum system that is governed by a Hamiltonian that is a sum of two terms, thus the symmetrized approximant (22) simplifies to

$$e^{-i\hat{H}t/\hbar} \approx e^{-i\hat{H}_1 t/2\hbar} e^{-i\hat{H}_2 t/2\hbar} e^{-i\hat{H}_2 t/2\hbar} e^{-i\hat{H}_1 t/2\hbar}, \tag{23}$$

recognizing the middle two terms as one full evolution under the second Hamiltonian $\hat{H}_2$, and comparing with the Taylor series expansion we get

$$e^{-i\hat{H}t/\hbar} \approx \hat{S}_2(\hat{H},t) \tag{24}$$

$$= e^{-i\hat{H}_1 t/2\hbar} e^{-i\hat{H}_2 t/\hbar} e^{-i\hat{H}_1 t/2\hbar} \tag{25}$$

$$= e^{-it(\hat{H}_1+\hat{H}_2)/\hbar + (-it/\hbar)^3 \hat{R}_3 + (-it/\hbar)^5 \hat{R}_5 + \ldots}, \tag{26}$$

where $\{\hat{R}_3, \hat{R}_5, ..\}$ are terms arising due to the non-commutativity of $\hat{H}_1$ with $\hat{H}_2$. Here we first, evolve the system in time under the first Hamiltonian $\hat{H}_1$ half an evolution, then a full evolution under the second Hamiltonian $\hat{H}_2$, followed by half an evolution under the first Hamiltonian $\hat{H}_1$. In other words, the product of these sub-evolutions approximates the full evolution of the system by dividing it into a symmetrized evolution under the two Hamiltonians: half an evolution under the first followed by the second, then repeating with the order of the Hamiltonians exchanged. Thus, symmetric with respect to the order in which it evolves the system in time under the two Hamiltonians. This symmetry in the system's evolution is compared to the asymmetric evolution of Eq. 13 in Sec. 4. It treats both Hamiltonians equally, in the sense that it does not discriminate in the order in which

it evolves the system under either. It simulates in opposite orders half an evolution under each to symmetrieze the full evolution around that optimal midpoint in the evolution. A symmetric evolution is more accurate than an asymmetric one since it takes into account the non-commutativity of the two operators $[A,B] = AB - BA \neq 0$. This method of symmetrization relies on what's known as the Baker–Campbell–Hausdorff (BCH) formula which is the solution for Z in the equation

$$e^X e^Y = e^Z. \tag{27}$$

for possibly non-commuting X and Y There are various ways to write the solution for Z, all ultimately yield an expression as a formal series (not necessarily convergent) in X and Y

$$Z(X,Y) = log(e^X e^Y) = X + Y + \frac{1}{2}[X,Y] + \frac{1}{12}[X,[X,Y]] - \frac{1}{12}[Y,[X,Y]] + ... \tag{28}$$

We recognize the third term on the right hand side as the piece we need to fully understand the nature of the symmetrized approximant in Eq. 26 and the source of error in the Trotter-Suzuki decomposition. Note that if X and Y commute, the right hand side of 28 would collapse to $X+Y$. This explains the exact approximations of the Trotter-Suzuki decomposition for commuting terms.

Let us first confirm that the first order approximant of Eq. 7 is indeed first-order by expanding both sides of the equation using their Taylor series. Let us start with the left hand side of equation (7)

$$e^{x(A+B)} = I + x(A+B) + \frac{1}{2}(x(A+B))^2 + O(x^3) \tag{29}$$

$$= I + x(A+B) + \frac{x^2}{2}(A^2 + AB + BA + B^2) + O(x^3), \tag{30}$$

and compare it to the expansion of the left hand side of equation (7)

$$e^{xA}e^{xB} = (I + xA + \frac{1}{2}(xA)^2 + O(x^3))(I + xB + \frac{1}{2}(xB)^2 + O(x^3)) \tag{31}$$

$$= I + x(A+B) + \frac{x^2}{2}(A^2 + 2AB + B^2) + O(x^3). \tag{32}$$

We notice that the third term in the expansion of the left hand side (30) has $AB - BA$ in place on the $2AB$ in the third term of the expansion of the right hand side (32). Mainly, the operator A always

comes on the left of the operator B in Eq. (32) while Eq. (30) treats both equally. Hence we obtain

$$e^{xA}e^{xB} = e^{(I+x(A+B)+\frac{x^2}{2}[A,B]+O(x^3))} \tag{33}$$

Thus, indeed the difference between the two will be second-order in x as claimed earlier. On the other hand, now we should be able to see what we mean by calling the approximant (22) symmetric. It symmetrizes the product of the sub-evolutions of the system under the Hamiltonians in the sum.

We find that the symmetrized approximant has the property

$$\hat{S}_2(\hat{H},t)\hat{S}_2(\hat{H},-t) = e^{-i\hat{H}_1 t/2\hbar}e^{-i\hat{H}_2 t/\hbar}e^{-i\hat{H}_1 t/2\hbar}e^{+i\hat{H}_1 t/2\hbar}e^{+i\hat{H}_2 t/\hbar}e^{+i\hat{H}_1 t/2\hbar} = I \tag{34}$$

because of which the even-order terms vanish in the exponent of the right-hand side of Eq. 26. In doing so, it promotes the approximant (22) to a second-order approximant since error is third-order - explaining why we denoted it $\hat{S}_2$.

$$e^{-i\hat{H}t/\hbar} = e^{-i\hat{H}_1 t/2\hbar}e^{-i\hat{H}_2 t/\hbar}e^{-i\hat{H}_1 t/2\hbar} + O(t^3) \tag{35}$$

$$= \hat{S}_2(\hat{H},t) + O(t^3). \tag{36}$$

Now we introduce the method for constructing the symmetrized fourth-order approximant from the symmetrized second-order approximant (22). Consider a product of (22) in the form

$$\hat{S}(\hat{H},t) \equiv \hat{S}_2(\hat{H},st)\hat{S}_2(\hat{H},(1-2s)t)\hat{S}_2(\hat{H},st) \tag{37}$$

$$= e^{-i\hat{H}_1 st/2\hbar}e^{-i\hat{H}_2 st/\hbar}e^{-i\hat{H}_1(1-s)t/2\hbar}e^{-i\hat{H}_2(1-s)t/\hbar}e^{-i\hat{H}_1(1-s)t/2\hbar}e^{-i\hat{H}_2 st/\hbar}e^{-i\hat{H}_1 st/2\hbar} \tag{38}$$

where s is an arbitrary real number for now. The expression (22) is followed by

$$\hat{S}(\hat{H},t) = \hat{S}_2(\hat{H},st)\hat{S}_2(\hat{H},[1-2s]t)\hat{S}_2(\hat{H},st) \tag{39}$$

$$= e^{-it(\hat{H}_1+\hat{H}_2)/\hbar+(-it/\hbar)^3(2s^3+(1-2s)^3)\hat{R}_3+O(t^5)} \tag{40}$$

Note that Suzuki arranged the parameters in the form $s, 1-2s, s$ in Eq. (38) so that i) the first-order term in the exponent of the last line of Eq. (40) should become $-i(\hat{H}_1+\hat{H}_2)t$ and ii) the whole product $\hat{S}(\hat{H},t)$ should be symmetrized, or should satisfy $\hat{S}(\hat{H},t)\hat{S}(\hat{H},-t) = I$. Because of the second

property, the even-order corrections vanish in the exponent of the last line of Eq. (40). That makes the parameter s the solution of the equation

$$0 = 2s^3 + (1-2s)^3 \tag{41}$$

$$s = \frac{1}{2-2^{1/3}} = 1.351207191959657..., \tag{42}$$

Thus, the approximant (38) is fourth-order.

Following the same logic, Suzuki considers another fourth order approximant of the form

$$\hat{S}_4(\hat{H},t) \equiv \hat{S}_2(\hat{H},s_2 t)^2 \hat{S}_2(\hat{H},(1-4s_2)t)\hat{S}_2(\hat{H},s_2 t)^2, \tag{43}$$

with s_2 being the solution to the equation

$$0 = 4s_2^3 + (1-4s_2)^3 \tag{44}$$

$$s_2 = \frac{1}{4-4^{1/3}} = 0.414490771794375... \tag{45}$$

Both approximants are fourth-order, however, the second one is more appropriate for general use. The first one might introduce problems since it crosses into "the past" $t < 0$ which might be problematic for certain problems.

Constructing higher-order approximants can be done recursively by following the method the fourth-order approximant (43) was constructed from the second-order approximant (22). Following this process systematically, we can construct higher-order approximants recursively using the second-order approximant (22) and the general form of the higher order approximant U_{2k}

$$\hat{U}_2(t) = \prod_{j=1}^{m} e^{-i\hat{H}_j t/2\hbar} \prod_{j=m}^{1} e^{-i\hat{H}_j t/2\hbar} \tag{46}$$

$$\hat{U}_{2k}(t) = \hat{U}_{2k-2}^2(s_{2k-2}t)\hat{U}_{2k-2}([1-4s_{2k-2}]t)\hat{U}_{2k-2}^2(s_{2k-2}t) \tag{47}$$

$$= e^{-i\hat{H}t/\hbar} + O(m^2 t^{2k+1}) \tag{48}$$

. Where the real number s_{2k} for the approximant of order $2k+2$ is defined as

$$s_{2k} = \frac{1}{4-4^{1/2k+1}}. \tag{49}$$

Now let us combine this result with the earlier technique of dividing the full evolution of the system into segmented evolutions of length Δt

$$\hat{U}_2(t) = (\prod_{j=1}^{m} e^{-i\hat{H}_j\Delta t/2\hbar} \prod_{j=m}^{1} e^{-i\hat{H}_j\Delta t/2\hbar})^{t/\Delta t} \tag{50}$$

$$\hat{U}_{2k}(t) = [\hat{U}^2_{2k-2}(s_{2k-2}\Delta t)\hat{U}_{2k-2}([1-4s_{2k-2}]\Delta t)\hat{U}^2_{2k-2}(s_{2k-2}\Delta t)]^{t/\Delta t} \tag{51}$$

$$= e^{-i\hat{H}t/\hbar} + O(m^2 t(t\Delta t)^{1/2k}). \tag{52}$$

This is a recipe for a systematic method of constructing $2k^{th}$ order symmetrized approximants recursively starting from the second-order symmetrized approximant. The algorithm this systematic recursive method suggests is depicted in Fig. 6.

Now that we have the full mathematical foundation necessary to construct the full algorithm we can test the performance of these formulas using code. We implement the higher-order approximants following Suzuki's explicit constructions in the paper referenced earlier.

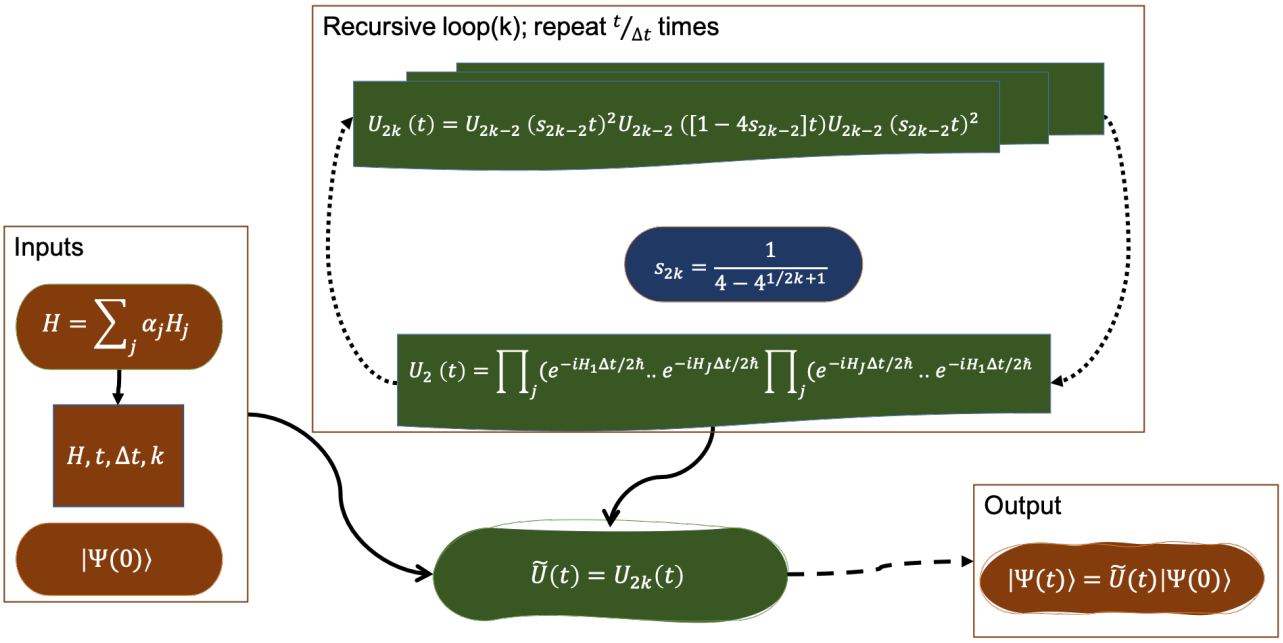


Figure (6) Diagram of the algorithmic steps of the $2k$-order product formula method for simulating the evolution of a local quantum system $|\Psi(0)\rangle$ by dividing the evolution of each of the j terms in the Hamiltonian $\hat{H} = \sum_j \alpha_j \hat{H}_j$ into short evolutions, the length of which is governed by value of the corresponding s_{2k-2} value - and the chosen Δt value. For a $2k$ order approximant, the full simulation $\hat{\tilde{U}}_{2k}(t)$ has error on the order of $\varepsilon = O(t(t\Delta t)^{1/2k})$.

4 Demo

4.1 First-order product formula and time segmentation

Here we detail an implementation of both the initial and full algorithms to show how one would perform the required steps to evolve a quantum system of interest based on the third stage in the full "recipe" presented earlier. First, we demonstrate that using the first order Trotter-Suzuki decomposition on the simplest quantum system of all: a two-level system. We show how the product formula method produces exact solutions when the Hamiltonian is made up of one term as is the case in the example below or is a sum of commuting terms as in Fig. 7. Then, we explore the algorithm's error scaling for a variety of Trotter step over a range of evolution time.

Two-level systems are ubiquitous in nature. A two-level system exists in a superposition of two distinguishable quantum states, meaning the Hilbert space describing such a system is two-dimensional. Therefore, a complete basis spanning the space will consist of two independent states. Any such system can be modeled as one qubit, where the qubit has one of two values of whatever that system represents: whether it be the polarization state of a photon, the state of the lepton family number oscillations of two neutrinos, or the magnetic nuclear spin state of a trapped ion. Here we demonstrate the algorithm on one qubit representing the spin state of a spin-half particle in the presence of a constant magnetic field. Analytical solutions for these two-level systems are easily attainable, see Appendix E, and so we use it to keep the demo rich in its simplicity. As mentioned in the background, considering the spin of a few more particles and adding a few orders of magnitude to the evolution time would vastly increase the resources needed to perform the computation.

Suppose we have a spin-$\frac{1}{2}$ particle revolving in the presence of a constant magnetic field (pointing in the z-direction or "out of the page"). The particle's angular momentum or spin revolves due to the Lorentz force the constant magnetic field generates. That means the quantum Hamiltonian operator representing the constant magnetic field B_0, ignoring the usual kinetic energy term and the standard potential energy term, is time-independent. Thus, the Hamiltonian $\hat{H}$ of a constant magnetic field $\vec{B} = B_0\hat{z}$, for a particle of charge equal to that of the electron $q = -e$, can be re-expressed in terms of the intrinsic spin operator $\hat{\vec{S}} = \hat{S}_x\hat{x} + \hat{S}_y\hat{y} + \hat{S}_z\hat{z}$ and simplified to the spin operator along the z-axis $\hat{S}_z$,

$$\hat{H} = -\hat{\vec{\mu}} \cdot \vec{B} = -\frac{gq}{2mc}\hat{\vec{S}} \cdot \vec{B} = \frac{ge}{2mc}\hat{S}_z B_0 = \omega_0 \hat{S}_z, \tag{53}$$

where we define $\omega_0 = \frac{geB_0}{2mc}$, and

$$\hat{S}_z = \frac{\hbar}{2}\hat{\sigma}_z = \frac{\hbar}{2}\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \tag{54}$$

for the magnetic moment due to the particle's spin $\hat{\vec{\mu}} = -\frac{e}{2mc}\hat{\vec{S}}$. Now we can find the eigenenergies of the Hamiltonian $\hat{H}$ by applying the $\hat{S}_Z$ operator on any initial quantum state of the particle $|\Psi(0)\rangle$. That we know how to do easily because any state $|\Psi(t)\rangle$ is a superposition of the two eigenstates $|+z\rangle$

and $|-z\rangle$ with eigenenergies E_+ and E_- respectively. Thus, we define the two states for the spin of the particle, which we define as $|0\rangle = |+z\rangle$ and $|1\rangle = |-z\rangle$ for spin-up and spin-down along the z-direction, as the computational basis of this two-level system. Now we can observe how the time evolution operator $\hat{U}(t) = e^{-i\hat{H}t/\hbar}$ acts on one of these two states, or of any normalized superposition of the two. Thus, exponentiating the Hamiltonian simply becomes applying the exponential of a superposition of the two eigenstates' corresponding eigenenergies,

$$\hat{H}|\pm z\rangle \quad = \tfrac{\hbar\omega_0}{2}\hat{\sigma}_z|\pm z\rangle = \pm\tfrac{\hbar\omega_0}{2}|\pm z\rangle = E_\pm|\pm z\rangle. \tag{55}$$

The particle's evolved state for initial state $|\Psi(0)\rangle = |+x\rangle$ is then,

$$\begin{aligned} |\Psi(t)\rangle &= \hat{U}(t)|\Psi(0)\rangle & (56)\\ &= e^{-i\hat{H}t/\hbar}\left(\frac{1}{\sqrt{2}}|+z\rangle + \frac{1}{\sqrt{2}}|-z\rangle\right) & (57)\\ &= \frac{1}{\sqrt{2}}e^{-iE_{+z}t}|+z\rangle + \frac{1}{\sqrt{2}}e^{-iE_{-z}t}|-z\rangle. & (58) \end{aligned}$$

So, to observe quantum dynamics at work, consider the particle with a state initialized on the x-y plane, that is a time-dependent state, $|\Psi(0)\rangle = |+x\rangle$ representing the particle initialized with a spin pointing in the positive x-axis. We can predict that the probability of the particle being in such a state, and placed in a magnetic field in the z-direction, rotates the spin of the particle about the z-axis as time progresses, with a period $T = 2\pi/\omega_0$. Classically, we say that the particle's spin is precessing in a magnetic field about the z-axis. However, one should be careful not to carry over too completely the classical picture for the Heisenberg uncertainty principle [19] tells us that the angular momentum -and hence the magnetic moment- of the particle cannot actually be pointing in a specific direction with arbitrary precision [20]. It turns out that the best that one can do is to simultaneously measure both the angular momentum vector's magnitude and its component along one axis. Back to the simulation, and to observe how the probabilities of being in certain spin states evolve in time, we solve for the

probability amplitude of the state being in the $|+x\rangle$ state,

$$\langle +x|\Psi(t)\rangle = (\frac{1}{\sqrt{2}}\langle +z| + \frac{1}{\sqrt{2}}\langle -z|)(\frac{1}{\sqrt{2}}e^{-iE_{+z}t}|+z\rangle + \frac{1}{\sqrt{2}}e^{-iE_{-z}t}|-z\rangle) \tag{59}$$

$$= \cos\frac{\omega_0 t}{2}, \tag{60}$$

giving an oscillatory probability of,

$$P_+ = |\langle +x|\Psi(t)\rangle|^2 = \cos^2\frac{\omega_0 t}{2}, \tag{61}$$

and similarly for the $|\Psi(0)\rangle = |-x\rangle$ case,

$$P_- = |\langle -x|\Psi(t)\rangle|^2 = \sin^2\frac{\omega_0 t}{2}. \tag{62}$$

Thus, equations 61 and 62 are the analytical solutions for the probabilities P_+ and P_- of the particle being in the $|+x\rangle$ or $|-x\rangle$ states respectively as a function of time. The sum of the probabilities for being spin-up or spin-down along the x-axis is one for all times, since these two states $|+x\rangle$ and $|-x\rangle$ form a complete set and probability is conserved.

Here we take an overview of our demo code where we present an algorithmic implementation in Python of the four stages discussed earlier. There, we define two Hamiltonians: 1) the Hamiltonian $\hat{H}$ as defined in Eq. 53 2) a prop Hamiltonian whose terms anti-commute defined as a sum of Pauli terms

$$\hat{H}_{prop} = \frac{\hbar\omega_0}{2}(\hat{\sigma}_x + \hat{\sigma}_z), \tag{63}$$

as defined in Primer 8.2, and the two states of interest $|+x\rangle$ and $|-x\rangle$ in terms of the two eigenstates $|+z\rangle$ and $|-z\rangle$ (representing our complete computational basis $|0\rangle, |1\rangle$). Then, we plot the analytical solutions to the spin-up and spin-down probabilities P_+ of Eq. 61 and P_- of Eq. 62 of the half-spin particle problem as seen in Appendix D for reference.

Then, we construct a numerical solver that given the initial state of a quantum system $|\Psi(0)\rangle$, the Hamiltonian governing its dynamics $\hat{H}$, and the time we wish to evolve the state of the system over t,

it outputs the final state $|\Psi(t)\rangle$ by numerical implementation of the exact time evolution operator. The numerical solver finds an exact evolution operator of a given Hamiltonian by direct exponentiation using a method supplied by the Python library "scipy" [21]. We, also, construct an approximate Product Formula (PF) solver that when given an initial state $|\Psi(0)\rangle$, a sparse Hamiltonian $\hat{H}$ provided as a sum of local terms $\hat{H} = \sum_j \alpha_j \hat{H}_j$, evolution time t, and Trotter number Q, it outputs the approximate final state $|\Psi(t)\rangle$. The PF solver finds an approximate evolution operator of a given sparse Hamiltonian presented as a sum of local terms by evolving each term separately by direct exponentiation using the same method used in the numerical solver, from the "scipy" Python library, over the given Trotter step Δt, which is computed from the given evolution time t and Trotter number $Q = t/\Delta t$. For a sufficiently small Trotter step Δt the approximate solver would compute an approximate time evolution operator that is at most ε off the exact evolution operator.

We, thereafter, plot the analytical solution to the probability P_+ of Eq. 61, and the numerical solution against the PF solution for an arbitrary $\Delta t = 1$ to verify the correctness of our method and show how the PF solver perfectly simulates the evolution for a Hamiltonian made up of commuting terms. See Fig. 7.

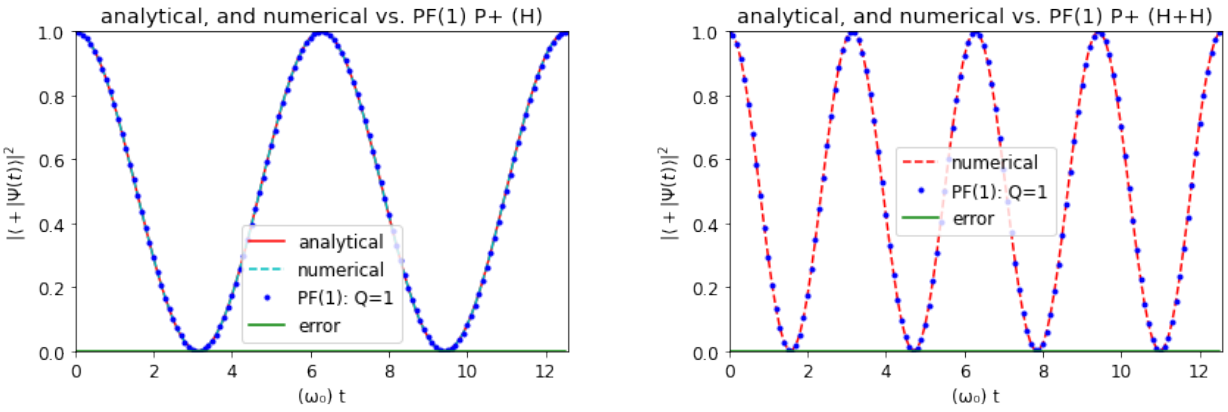


Figure (7) Left panel: the analytical solution to the spin-up probability P_+ found in Eq. 61, the exact solution of the numerical solver, the approximate solution of the PF solver with Trotter number $Q = 1$ and their difference. Right panel: the numerical solution to the spin-up probability P_+, the approximate solution with Trotter number $Q = 1$ and their difference. As time progresses, the PF method produces an exact solution since our Hamiltonian $\hat{H}$ of Eq. 53 is made up of one term and $\hat{H} + \hat{H}$ as of (53) are two commuting terms.

Then, to observe a nontrivial approximation, we considered the transverse Hamiltonian field $\hat{H}_{prop}$. As a test, we plot the PF solution to the spin-up probability P_+ under the Hamiltonian $\hat{H}_{prop}$ with Trotter number $Q = 2, 4, 5$ against the numerical solution and found their difference over time in Fig. 8.

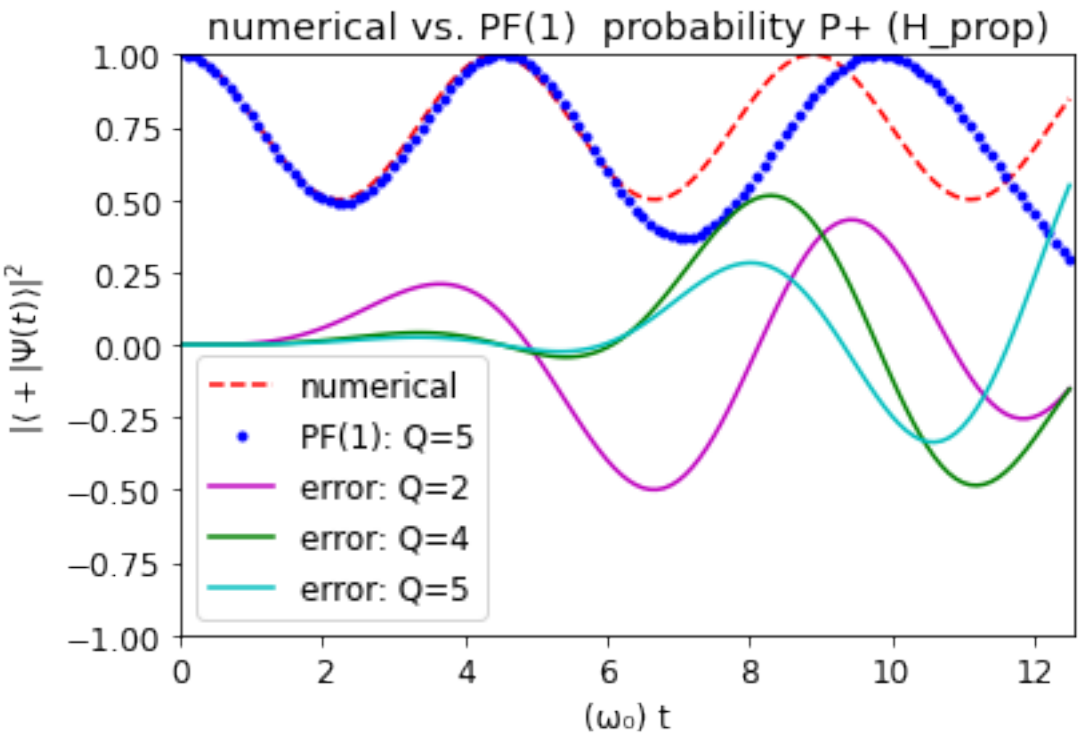


Figure (8) The numerical solution to the spin-up probability P_+ against the PF solution with Trotter number $Q = 5$ and the difference of $Q = 2, 4, 5$. The Hamiltonian $\hat{H}_{prop}$ as defined in Eq. 63 was used to generate the evolutions. As time progresses, the difference of the probabilities increases indicating the decrease in the PF solution accuracy as the evolution time increases.

To further observe the effect of increasing the Trotter number Q on the accuracy of the approximation as time progresses, we compute the PF solution to the spin-up probability P_+ under the Hamiltonian $\hat{H}_{prop}$ with different Trotter numbers $Q = 30, 60, 125, 1000$ against the numerical solution and plot their difference over a range of evolution time as seen in Fig. 9.

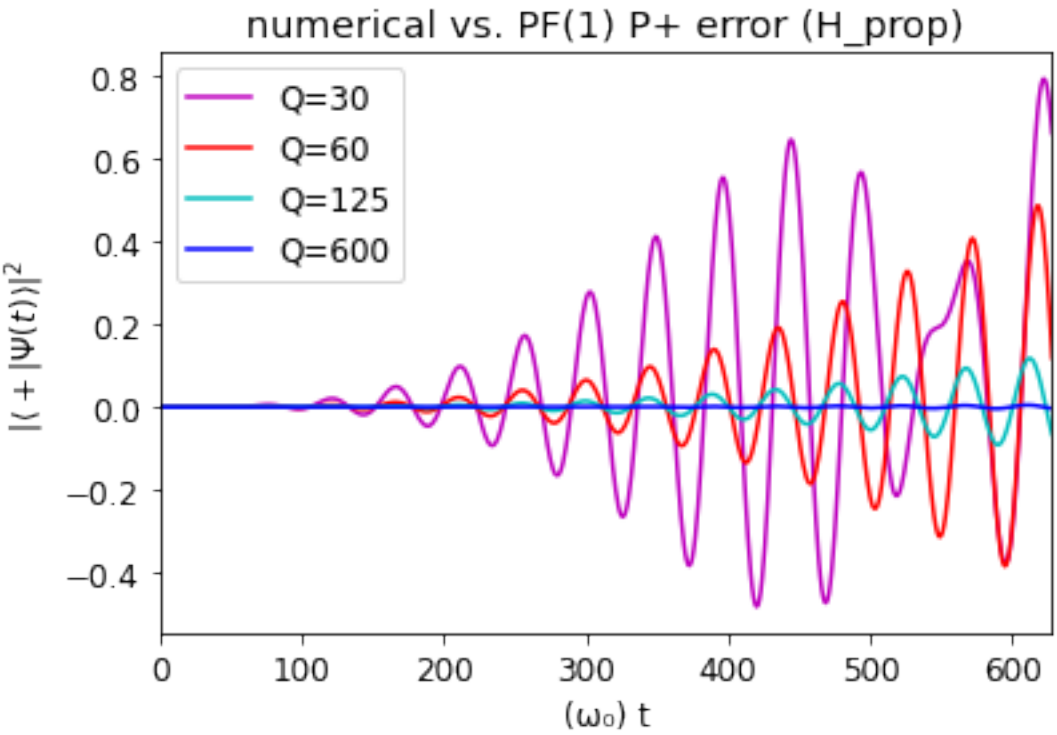


Figure (9) The probability difference between the numerical solution and the PF solution to the spin-up probability P_{+x} under the Hamiltonian $\hat{H}_{prop}$ as defined in Eq. 63 with Trotter numbers $Q = 30, 60, 125, 600$. As time progresses, the solutions with a sufficiently large Trotter number consistently produce more accurate solutions as the evolution time increases.

Lastly, we define an error solver that given a Hamiltonian provided as a sum of local terms $\hat{H} = \sum_j \alpha_j \hat{H}_j$, evolution time t, and Trotter number Q, it outputs the error ε defined, as discussed earlier, as the two-norm of the difference between the exact (numerical) operator $\hat{U}(t)$ and the approximate (PF) operator $\hat{\tilde{U}}(t)$. To observe how error scales in time, we plot the error to the spin-up probability P_+ under the Hamiltonian $\hat{H}_{prop}$ using increasing Trotter numbers $Q = 300, 3000, 30,000$ over a range of evolution time as seen in Fig. 10.

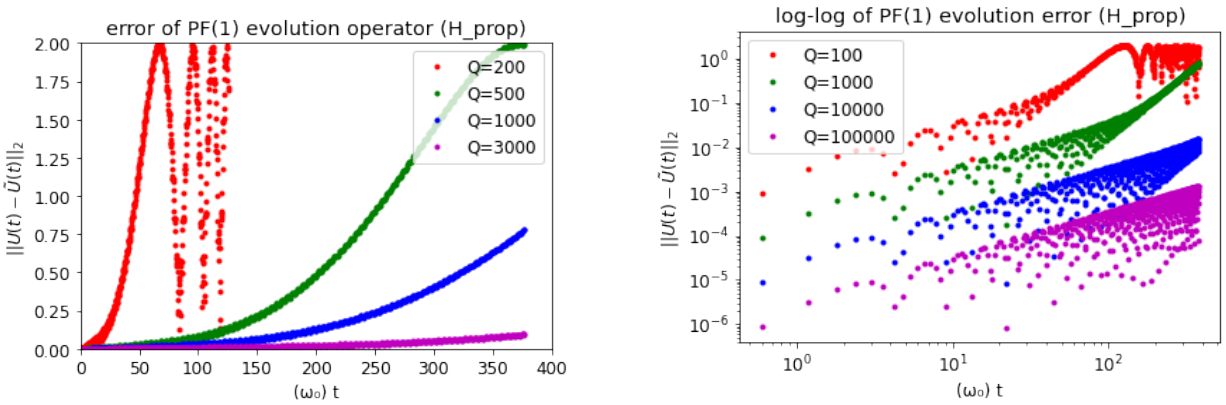


Figure (10) Left panel: error of the PF evolution for the Hamiltonian $\hat{H}_{prop}$ as defined in Eq. 63, and Trotter numbers $Q = 200, 500, 1000, 3000$ over time. Right panel: log-log plot of error of the PF evolution for the Hamiltonian $\hat{H}_{prop}$, and Trotter numbers $Q = 100, 1000, 10000, 100000$ over time. This shows that for a given Trotter number $Q = t/\Delta t$ and as the evolution time t increases, the Trotter step Δt length increases, driving error to scale up in what appears to be a polynomial growth as a function of evolution time and Trotter step $O(t\Delta t)$.

4.2 Higher-order formulas

We saw how the general behavior of the segmentation of the full time evolution of a quantum system generated by a Hamiltonian that is a sum of non-commuting terms is consistent with our claim earlier that doing this process controls error scaling as desired: the smaller the time step is relative to the evolution time, the smaller is the error.

Now, let us consider the order of the approximants starting with the first-order formula we were using in the previous part and move up the ladder through the higher-order symmetrized Suzuki constructions up to the eighth-order approximant. We test these approximations while introducing a new system of interest, the second simplest system of all: a two-body system of spin-$1/2$ particles. The approximants we apply here can be thought of as a 2-qubit gates. Being able to apply two-qubit gates allow us, in theory, to simulate systems that follow the nearest-neighbor interaction models, which is common in magnetism quantum models such as the Ising, the XY, and the Heisenberg models. We test whether the prescribed formulas provide approximants to the order they claim. We test the formu-

las of the first few orders starting with perturbational approximant (10), and moving through the first order formula (17), the symmetrized second-order approximant (22), the symmetrized fourth-order approximant (43), and up to the eighth-order approximant. We explore the effect of having error introduced in the value of the real number s_{2k} over a range of orders of magnitude of relative error. We also explore the difference in the eigenvalues of the ideal time-evolution operator and the symmetrized Trotter-Suzuki approximants. Lastly, we explore the effect of changing the order in which the product of matrices is carried out in the approximants on error.

Two-body systems are ubiquitous in nature, as well. Here, we will focus our attention to one simple example of a two-particle system. The example is of the interaction between the magnetic moments of two spin-$1/2$ particles. Such a two-particle spin-$1/2$ system would exist in any superposition of all four distinguishable quantum states the system can be in, meaning the Hilbert space describing such a system has 2^2 dimensions, or 4-dimensional. Therefore, a complete basis spanning the space will consist of four independent states, defined as $\{|00\rangle = |0\rangle \otimes |0\rangle, |01\rangle = |0\rangle \otimes |1\rangle, |10\rangle = |1\rangle \otimes |0\rangle, |11\rangle = |1\rangle \otimes |1\rangle\}$ in the computational basis. Any such system can be modeled as a two-qubit system, where each qubit has one of the two values each of the two particles can be in. Here we demonstrate the algorithm on two qubits representing the spin state of the two identical spin-half particles with a constant interaction strength $J : \hbar/J = 1$ in the presence of a constant magnetic field of energy $h : \hbar/J = 1$ pointing in the x-direction. Among other consequences, the exchange interaction is responsible for ferromagnetism and the volume of matter. Ferromagnetism arises when a collection of atomic spins align such that their associated magnetic moments all point in the same direction, yielding a net magnetic moment which is macroscopic in size. The simplest theoretical description of ferromagnetism is called the Ising model.

Let us consider the time evolution of the 2-qubit system under the Ising-model Hamiltonian

$$\hat{H}_2 = -J(\hat{\sigma}_z\hat{\sigma}_z) - h(\hat{\sigma}_{x1} + \hat{\sigma}_{x2}), \tag{64}$$

where $\hat{\sigma}_z\hat{\sigma}_z = \hat{\sigma}_z \otimes \hat{\sigma}_z$, $\hat{\sigma}_{x1} = \hat{\sigma}_x \otimes \hat{I}$, and $\hat{\sigma}_{x2} = \hat{I} \otimes \hat{\sigma}_x$. The first term represents the coupling term

(where J is known as the exchange energy) between the two qubits, and the second term represents the action of the constant magnetic field of energy h.

We plot the error in the product formula approximation to the unitary time evolution operator provided by the first-order approximant (13) and compare it with the the perturbational approximant (10); see Fig. 11. We note that the Taylor series solution provides a zero-order approximant in the sense that the leading error term is first-order. We can see the effect we wish to understand by keeping only the first two terms of the series.

We notice the periodic nature of the solution provided by the Trotter-Suzuki decomposition, in contrast to the monotonic increasing nature of the Taylor series approximation. The exact dynamics should conserve unitarity in time. Fig. 11 shows the deviation in the norm of the difference between the exact and approximate evolution due to the approximations. The error in the Trotter approximation (17) oscillates periodically in Fig. 11 (a). We can understand this as follows: when time is evolved one cycle of precession the spin comes back to the original position, it comes back accurately to the initial state because of the unitarity of the Trotter approximation, and hence the oscillation. In contrast, error grows monotonically in the case of the perturbational approximant as is shown in Fig. 11 (b). This is because the norm of the time evolution operator increases by the factor

$$||1 - i\Delta t\hat{H}/\hbar|| \approx 1 + \frac{\Delta t}{\hbar}||\hat{H}|| > 1, \tag{65}$$

The remarkable difference between Fig. 11(a) and Fig. 11(b) thus comes from the fact that the Trotter approximant is unitary.

On the other hand, the fact that the symmetrized product formulas keep the symmetry of the system is one of their remarkable advantages. A symmetric evolution evolves a system's Hamiltonian symmetrically with respect to the local Hamiltonians its made up of. The symmetrization for a Hamiltonian made up of two terms can be thought of as turning the simulator switch "on" for the first term for half an evolution while the other term is "off", then flipping both switches for half an evolution, and repeat the process in reverse order, thus performing a symmetric time evolution. That practically

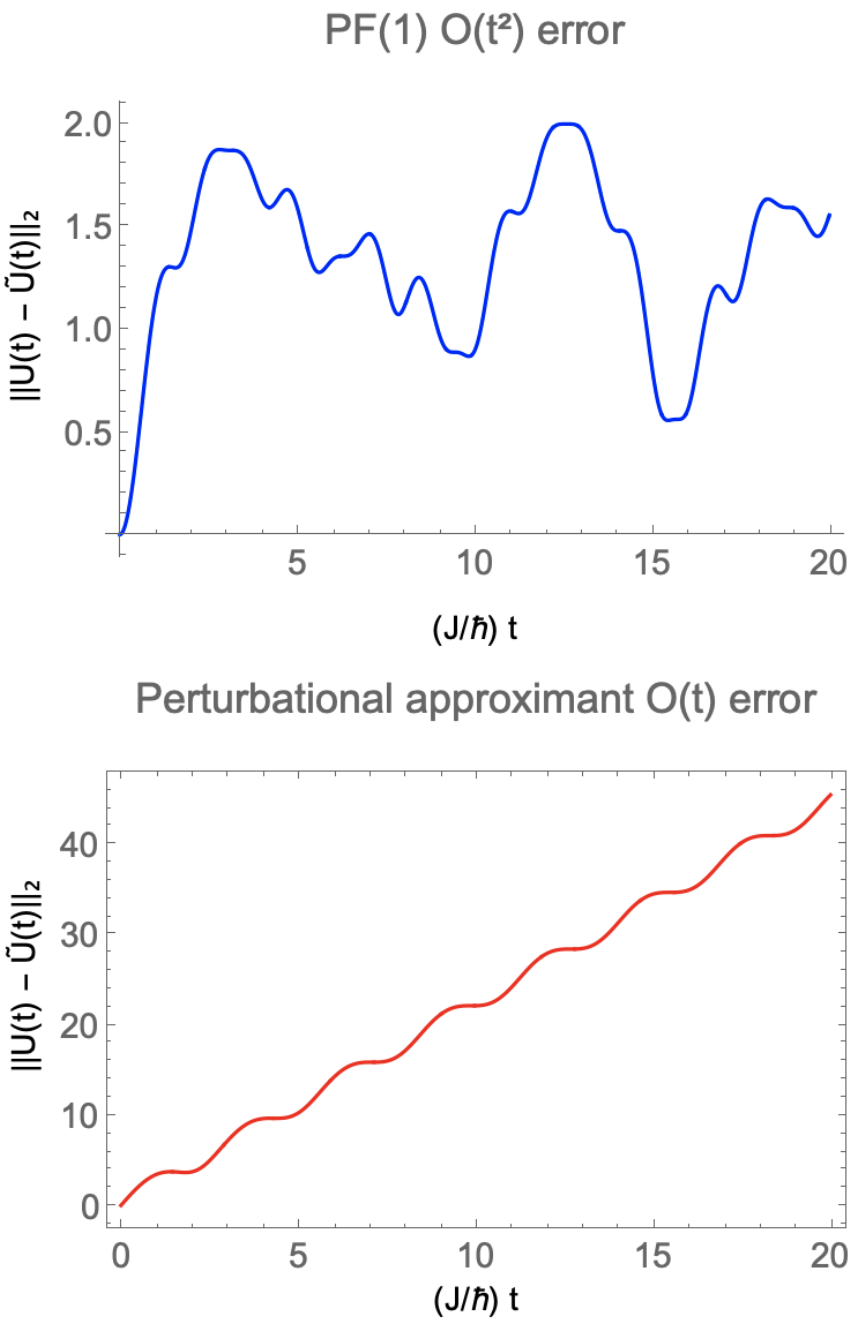


Figure (11) Error of the perturbational approximant (10) using the first two terms of the Taylor series expansion v. the first-order approximant (22) of the evolution operator for the Hamiltonian $\hat{H}_2$ (64). We plot this data on a log-log scale and use a nonlinear fit of the form (ax^b) to obtain values $(a = 4.47214 \times 10^{-4}, b = 1.)$ for the the perturbational approximant and $(a = 2. \times 10^{-8}, b = 2.)$ for the first-order approximant, which confirms the order of the approximants.

means that this symmetrized evolution did not preferentially choose which of the two terms to evolve first; which would make it asymmetric. Here, we demonstrate that this indeed affects numerical accu-

racy strongly using a simple example of quantum dynamics, namely the spin precession of a 2-particle spin-$1/2$ system under the Hamiltonian $\hat{H}_2$ as defined in Eq. 64 initialized in the state $|++\rangle$

$$|\Psi(0)\rangle = |++\rangle = |+\rangle \otimes |+\rangle = 1/2 \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \tag{66}$$

The particle's evolved state is then,

$$|\Psi(t)\rangle = \hat{U}(t)|\Psi(0)\rangle \tag{67}$$

$$= e^{-i\hat{H}_2 t/\hbar}|++\rangle. \tag{68}$$

Now, let us consider the probability that upon a measurement the system is found in its initial state,

$$P_{++} = |\langle\Psi(0)||\Psi(t)\rangle|^2. \tag{69}$$

We plot in Fig. 12 the numerical solution for the 2-qubit system probability P_{++} of Eq. 69 under the Hamiltonian $\hat{H}_2$ (64) as a function of evolution time. We compare it to the first-order approximant of Eq. 17, and the symmetrized second-order approximant (22). We use the Trotter number $Q = 5$ over an appropriate range of evolution time, and include the difference of the two solutions with the numerical solution and call it the error.

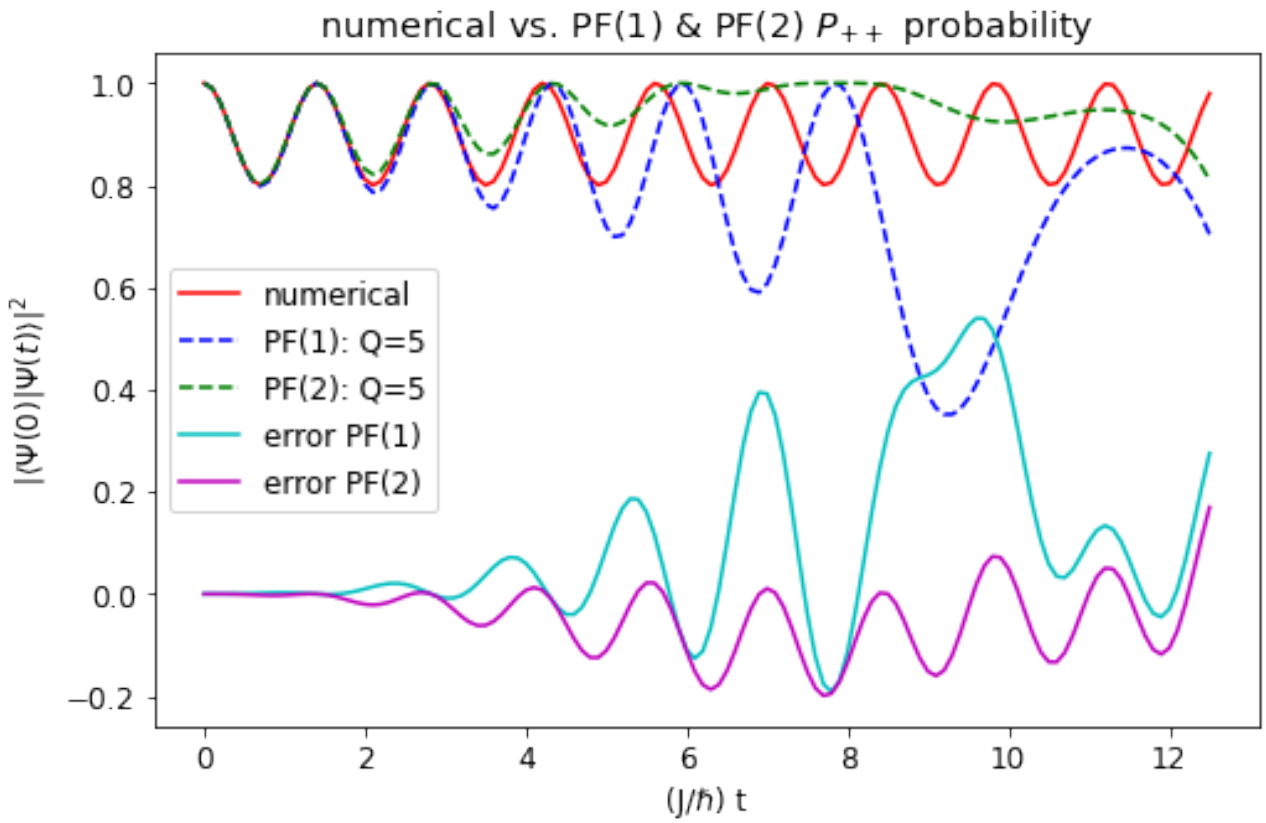


Figure (12) The numerical solution to the 2-qubit system's probability P_{++} (69) under the Hamiltonian $\hat{H}_2$ (64) against the first-order approximant solution (17) and the symmetrized second-order approximant (22) with Trotter number $Q = 5$ and their difference with the numerical solution. As time progresses, the difference of the probabilities of the symmetric approximant increases with a smaller rate compared to the first-order approximant.

Then, we plot the error being the two-norm of the difference between the exact (numerical) operator $\hat{U}(t)$ and the second-order symmetrized approximant operator $\hat{\tilde{U}}(t) = \hat{S}_2(\hat{H}_2, t)$. To observe how error scales in time, we plot the error to the spin-plus plus probability P_{++} under the Hamiltonian $\hat{H}_2$ using increasing Trotter numbers $Q = 10, 100, 1000, 10000, 100000$ over a range of evolution time as seen in Fig. 13.

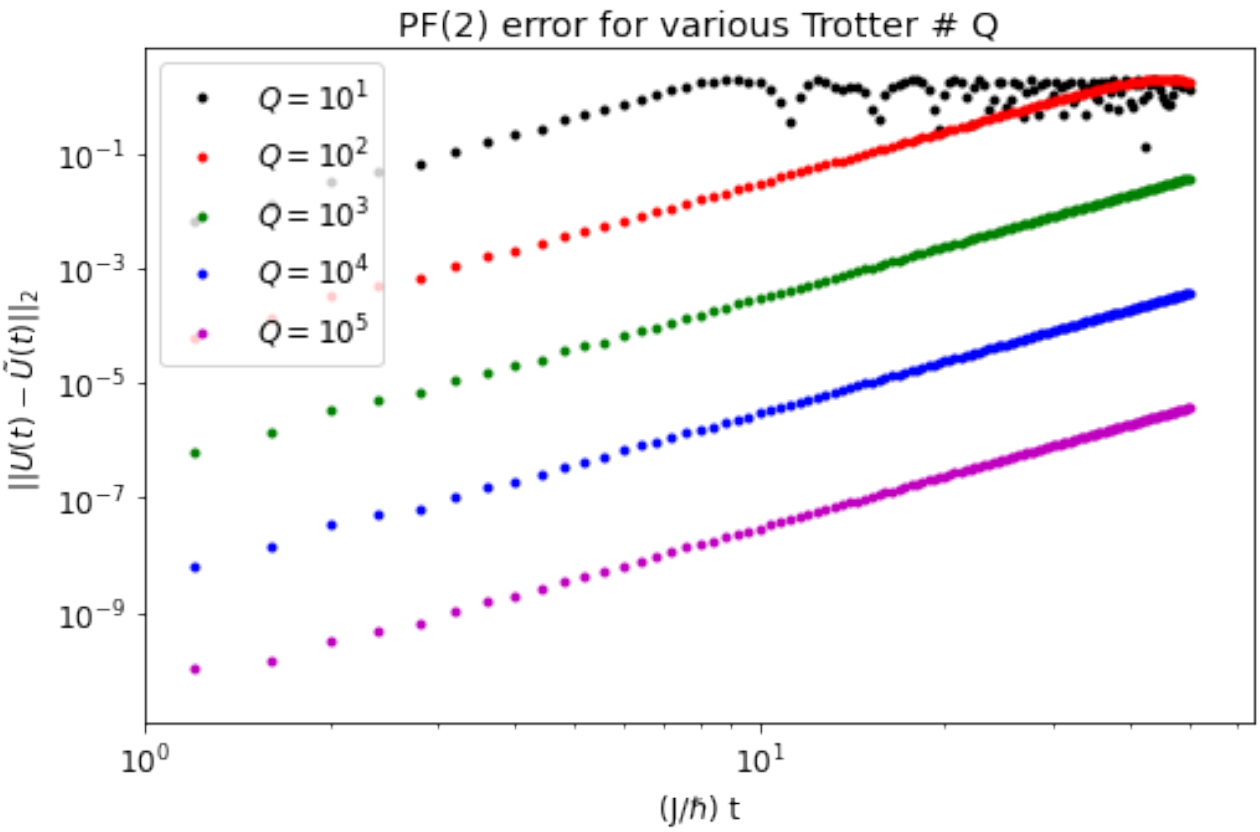


Figure (13) Error of the second-order approximant (22) for the Hamiltonian $\hat{H}_2$ (64), and Trotter numbers $Q = 10, 100, 1000, 10000, 100000$ over evolution time. This shows that for a given Trotter number $Q = t/\Delta t$ as the evolution time t increases, the Trotter step Δt length increases, driving the leading error term to scale up in what appears to follow a power law growth as a function of evolution time and Trotter step $O(t(t\Delta t)^{1/2}$. Thus, data fits will indicate the order of the leading error term in the approximation, which can hint at an appropriate Trotter number Q. Refer to Tab. 1 for the data.

Lastly, we construct the higher-order solutions to this problem choosing a fixed Trotter number of $Q = 1$. We do that to focus solely on the effect of using higher order formulas. We combine the solutions for the first, second, fourth, sixth, and eighth order approximants with the perturbational approximant for comparison and plot them against evolution time; see Fig . 14.

Data table of Fig. 13 fits to power law (ax^b)

$Q = 10^1$	$a = 0.54935526$	$b = 0.25618615$
$Q = 10^2$	$a = 8.66644270 \times 10^{-4}$	$b = 2.01884873$
$Q = 10^3$	$a = 2.98939420 \times 10^{-7}$	$b = 2.99964445$
$Q = 10^4$	$a = 2.99963787 \times 10^{-9}$	$b = 2.99854063$
$Q = 10^5$	$a = 2.99975525 \times 10^{-11}$	$b = 2.99852821$

Table (1) This data for the second order approximant shows a trend that suggests that setting the Trotter number Q on the order of 10^3 for the given evolution time range and using this second-order approximant is sufficiently small, meaning that setting it any higher has no apparent effect on the order of the leading error term.

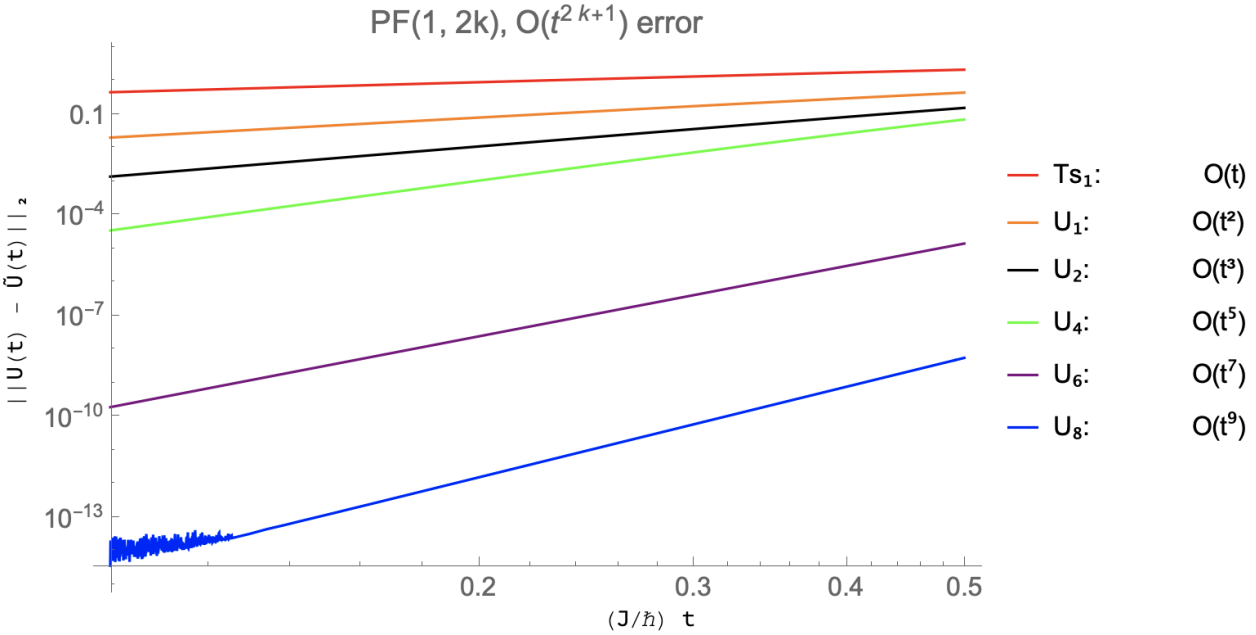


Figure (14) Error of the perturbational, first, second, fourth, sixth and eighth order approximants against evolution time for the Hamiltonian $\hat{H}_2$ (64). This confirms that using higher order formulas will produce more accurate solutions. To verify the effectiveness of the approximants in simulating the evolution of the system up to the correct order in error, we use line fits to produce the data in Tab. 2.

Data table of Fig. 14 fits to power law (ax^b)

$Ts(k=1)$	$a = 0.460209$	$b = 0.944897$
U_1	$a = 0.0225239$	$b = 1.84178$
U_2	$a = 0.00157926$	$b = 2.83983$
U_4	$a = 6.54165 \times 10^{-5}$	$b = 4.32197$
U_6	$a = 2.01291 \times 10^{-10}$	$b = 6.92216$
U_8	$a = 3.18113 \times 10^{-15}$	$b = 8.92021$

Table (2) This data for the perturbational, first, second, fourth, sixth and eighth order approximants show their effectiveness in simulating the evolution of the system with a leading error term that is of the correct order in time, with some showing almost a perfect match.

4.2.1 Varying value of s_{2k}

Here we consider the impact of introducing some error to the value of the number s_{2k-2} in Eq. 49 with various values of relative error Δs varying defined as

$$\Delta s(s, \%) = s + (s \times \%), \tag{70}$$

in orders of magnitude and observe its effect on the leading error term; see Fig. 15. We use the fourth order approximant with its associated value of s_{2k} for $k = 2$

$$s_{2k-2} = \frac{1}{4 - 4^{1/2k+1-2}} \tag{71}$$

$$s_2 = \frac{1}{4 - 4^{1/3}} = 0.4144907717... \tag{72}$$

We notice how a small alteration, for example notice the solution using the 1% error of the original value of s_{2k}, results in a drastic effect on the accuracy of the approximant. We see how this sensitivity to the value of s_{2k} shows up dramatically in the accuracy of the solution. That means it is essential for the approximant to be provided with the correct value of s_{2k} in order to achieve an approximation of the correct order in error. As we see in Fig. 15 for the fourth-order approximant (43), the value s_{2k}

is optimal for the approximant in the sense that alterations in its value (be it a positive or a negative contribution) will worsen the performance of the approximant in error - up to some “tolerable“ error as we shall find out. We notice in the plot that altering the value of s_{2k} up to a certain order seems to only cause an increasing deviation in the value of the multiplicative amplitude but has no effect on the order of the leading error term. To verify that we use a polynomial fit of the form (ax^b) to obtain the values of the multiplicative amplitude a and the power b shown in Tab. 3. What we mean there by a “tolerable“ deviation is that it would only cause an increase in the value of the multiplicative amplitude a, while an “intolerable“ deviation causes an increase in the value of the multiplicative amplitude a of and, more importantly, the order of the leading error term b.

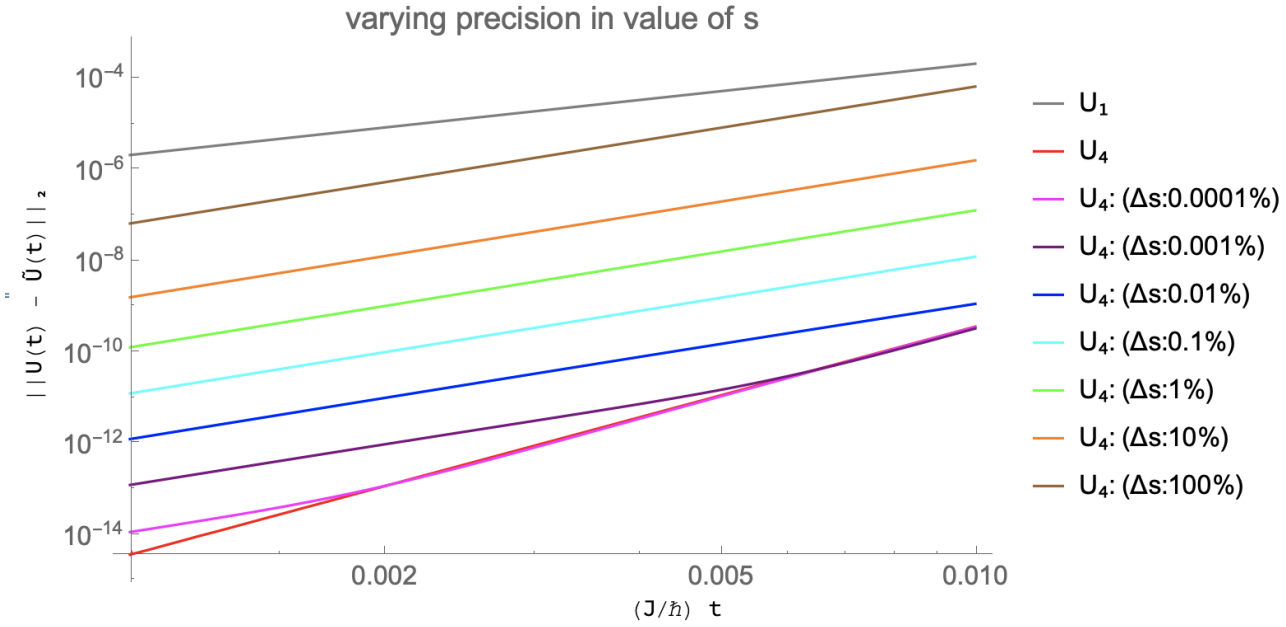


Figure (15) Error for a set of different alterations in the precision of the value s_{2k-2} for the fourth-order approximant (43) s_2 of the evolution operator generated by the Hamiltonian $\hat{H}_2$ (64). Refer to Tab. 3 for the fits data.

Data table for fits of Fig. 15 to power law (ax^b)

U_1	$a = 2.00018 \times 10^{-6}$	$b = 1.99994$
$U_4 : \Delta s = 0\%$	$a = 3.47031 \times 10^{-15}$	$b = 4.99972$
$U_4 : \Delta s = 10^{-4}\%$	$a = 3.14613 \times 10^{-15}$	$b = 5.0363$
$U_4 : \Delta s = 10^{-3}\%$	$a = 4.11054 \times 10^{-15}$	$b = 4.88398$
$U_4 : \Delta s = 10^{-2}\%$	$a = 1.34973 \times 10^{-12}$	$b = 2.90987$
$U_4 : \Delta s = 10^{-1}\%$	$a = 1.22706 \times 10^{-11}$	$b = 2.98451$
$U_4 : \Delta s = 10^{0}\%$	$a = 1.22951 \times 10^{-10}$	$b = 2.9982$
$U_4 : \Delta s = 10^{1}\%$	$a = 1.52549 \times 10^{-9}$	$b = 2.99958$
$U_4 : \Delta s = 10^{2}\%$	$a = 6.33163 \times 10^{-8}$	$b = 2.99893$

Table (3) This data for the fourth order approximant shows a trend that suggests that deviations up to order 10^{-3} in the value of s_{2k} are "tolerable" while larger deviations, of order 10^{-2} and higher, are "intolerable".

4.2.2 Randomizing the order of terms

Finally, we explored whether the order in which the product is carried out in an approximant matters in terms of error. Here we change the order in which the product of the exponentials in a the second-order approximant is taken and explore its effects on error scaling.

Consider a 2-dimensional Hamiltonian that is a sum of three terms

$$\hat{H}_3 = (\hat{\sigma}_x + \hat{\sigma}_y) + \hat{\sigma}_y + \hat{\sigma}_z, \tag{73}$$

and call the terms as follows

$$\hat{H}_3 = \hat{H}_i + \hat{H}_j + \hat{H}_k, \tag{74}$$

for simplicity. The second-order approximant of Eq. 22 becomes

$$\hat{S}_2(\hat{H}, t) = e^{-i\hat{H}_i t/2\hbar} e^{-i\hat{H}_j t/2\hbar} e^{-i\hat{H}_k t/\hbar} e^{-i\hat{H}_j t/2\hbar} e^{-i\hat{H}_i t/2\hbar}. \tag{75}$$

Now let us explore what happens if reorder the terms in on the right hand side by giving each of the three terms in the Hamiltonian an index letter of the set $\{i, j, k\}$ and compare the product of different permutations of the index list in terms of error; see Fig. 16.

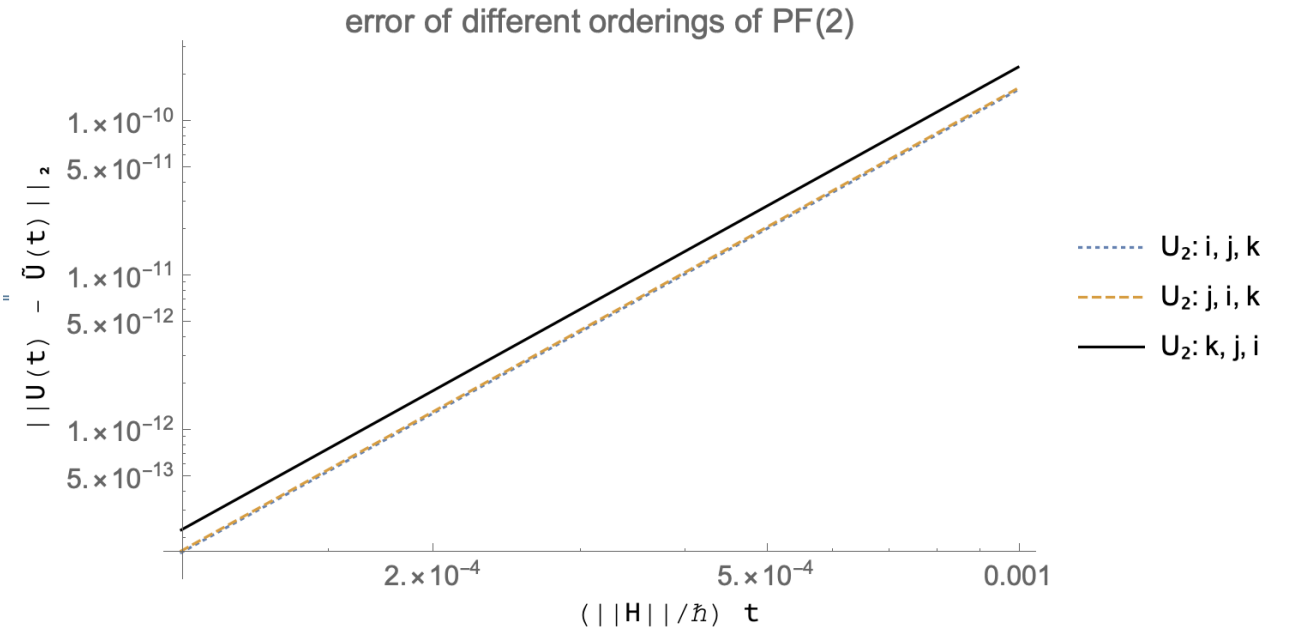


Figure (16) Error for a set of different permutations of the i, j, k indices for the fourth-order approximant (43) of the evolution operator for the Hamiltonian $\hat{H}_3$ (73) showing a slight difference between one of the three permutations with respect to the other two. Refer to Tab. 4 for the fits data.

We notice that the third permutation produces a slightly worse performance in error than the first two. We obtain the parameters for the data fits to observe if there was any effect on the order of the leading error term; see Tab. 4. We notice that the first two permutations had the same exact multiplicative amplitude and order for the leading error term, while the last permutation produced an approximation with a leading error term that has a slightly higher multiplicative amplitude than the other two. This suggests that re-ordering the terms of the Hamiltonian can have a positive/negative impact on the leading error term. To verify that more concretely, different data sets need to be tested using approximants of different orders. That might reveal that arranging the product in a certain way can increase (or decrease) the order of the leading error term.

Data table for fits of Fig. 16 to power law (ax^b)

i,j,k	$a = 2.25347 \times 10^{-13}$	$b = 3.$
j,i,k	$a = 2.25347 \times 10^{-13}$	$b = 3.$
k,j,i	$a = 3.80173 \times 10^{-13}$	$b = 3.$

Table (4) This data for the second order approximant shows that the order in which the exponential product is carried out seems to only affect the multiplicative amplitude of the leading error term, without any considerable effect on its order.

5 Discussion

Here we briefly discuss some of the results of the simulations from the previous section. Code for the simulation of the first example of a 1-particle system can be found in Appendix D.

First, we verified that the PF solver obtains exact solutions for commuting terms, as seen in Fig. 7. We considered a Hamiltonian whose local terms do not commute, $[\hat{H}_1, \hat{H}_2] \neq 0$, $\hat{H}_{prop}$ (63). That would make the approximation difficult, since the non-commuting nature of the Hamiltonians means that the order at which they are carried out will affect the result, revealing the nature of the error this technique induces. The time evolution operator in this case can be expressed using Eq. 13. We observed how the first order product formula solution deviates from the numerical solution for the Hamiltonian H_{prop} of non-commuting terms, as is the case for almost any "real-world" Hamiltonian simulation problem, as seen in Fig. 8. Then, we saw the error scaling for a range of Trotter step; as seen in Fig. 10.

There, we found that the product formula solution matches the numerical solution up to a high degree of accuracy during the first few units of time. Then, as expected, the solution becomes increasingly less accurate as the evolution time increases. This is a general trend in product formulas since errors accumulate, as a result of the non-commuting relation between the terms making up the Hamiltonian. In other words, having more non-commuting terms in the Hamiltonian is bound to in-

crease the overall error. To soften that domino effect of error carrying on in the construction of the solution, choosing a sufficiently large Trotter number, and thus a shorter Trotter step, is necessary. As discussed in Section 3.3, taking a smaller length for the time step Δt, ensures that less error is incurred by each individual evolution and therefore can lead to higher order approximation for the full evolution as a function of evolution time t. This provides a reasonable method for us to minimize error in the simulation of the time evolution algorithmically. It is important to re-emphasize that although a numerical solver is "easily" attainable, the time resources it requires balloons quickly for larger inputs. Thus, it is of extreme importance to be able to efficiently approximate the evolution of a quantum system, in contrast to the exact yet computationally-expensive numerical method.

Refer to Fig. 9 where we plot the difference between the numerical solution to the spin-up probability P_+ and the PF solutions of increasing Trotter numbers, thus smaller Trotter steps, are plotted as a function of evolution time. The (two) terms of the Hamiltonian do not commute, forcing the approximation to err.

We can see that varying the length of the time step the evolution of each term is broken into affects the magnitude of error dramatically. Compare the difference between the $Q = 30$ and $Q = 125$ solutions for example. The $Q = 30$ solution starts noticeably deviating from the numerical solution shortly after the first 100 time units have passed. That is, once the evolution time exceeds the Trotter number, making the Trotter steps larger than one $\Delta t \geq 1$, error incurred from each sub-evolution step the solver takes starts becoming apparent. While the $Q = 600$ solution approximates the exact solution well after 600 time units have passed. That is because even when the sub-evolutions are carried over the largest Trotter steps, occurring towards the end of the range at evolution times $t \leq 600$, the Trotter steps are still well below one $\Delta t < 1$. That demonstrates the effect of decreasing the length of the Trotter step on minimizing the error.

Another method to observe how the algorithm performs in terms of error is to abandon quantum states and focus solely on the time evolution unitary operator. That allows us to analyze the error

incurred without a consideration of the state of the system being studied. As discussed earlier the error in the product formula approximation occurs due to the commutators' error, not the size or the norm of the Hamiltonian. Moreover, when $\hat{H}$ is sparse, most of its matrix norms have comparable values, so the complexity of simulating $\hat{H}$ is not very sensitive to how its size is quantified. Here, our Hamiltonian H_{prop} is made up of 1-sparse Pauli terms. In other words, we can ignore the effect of the dimensionality of the Hamiltonian and extract information about the error solely based on the commutators of the input Hamiltonian terms and the time step we use. Thus, we can focus our attention to the effect of varying the length of the time steps the sub-evolutions are broken into on minimizing the commutators' error.

We plot the results for different Trotter numbers, and include a plot with a semi-log scale while increasing the Trotter number by orders of magnitude, over a range of evolution time, as seen in Fig. 10. First, we observe that error does not scale exponentially with evolution time. We note that error grows in a polynomial-fashion as a function of the Trotter step Δt length and evolution time t. This demonstrates how having a sufficiently small Trotter step Δt to shorten the time each sub-evolution is carried over is necessary for a polynomial scaling in error. Thus, the use of product formulas to devise an algorithm for the decomposition of the evolution of a quantum systems allows for the time, and thus gate, complexity of the algorithm to scale up polynomially as a function of time and time step $O(t\Delta t)$.

Then, we turned our attention to the higher order approximants provided by Suzuki's recursive constructions. We found that the solutions provided by the $2k$ order approximant do in fact have a leading error term of order $2k+1$, as seen in Fig. 14. We verified that with power law fits on the data plotted on a log-log scale, as seen in Tab. 2. Thus, the product formulas allow for the decomposition of the evolution of a quantum systems allows for the time complexity of the algorithm to scale up polynomially as a function of time, time step, and approximant order $O(t(t\Delta t)^{1/2k})$. Following that, we tested the sensitivity of these higher order approximants to the value of s_{2k}, as seen in Fig. 15. We

found that for a given approximant there exists a range over which deviations in value of the number s_{2k} are "tolerable" in the sense that they introduce deviations only in the multiplicative amplitude of the leading error term, leaving its order unchanged; as seen in Tab. 3. Lastly, we explored the effect of randomly reordering the terms in the exponential product of the product formulas, specifically the second-order approximant; as seen in Fig. 16. We found that the simulation for one of the permutations produced a slightly less accurate solution than the other two; as seen in Tab. 4. That suggests that the order in which the product is carried out can be optimized in a certain way. Moreover, we expect that reoredering the terms can have a more dramatic impact on the leading error term (increasing its order) under certain circumstances. Recent "creative" approaches to the product formulas have shown that error can be made to cancel out in an advantageous way, leading to quadratic scaling in some cases [22], as has been explored in the works cited [23] [22].

Thus, we found that the product formulas can provide an algorithm for simulating quantum dynamics that is more computationally efficient than directly simulating the evolution of quantum system, on both classical and quantum computers. When compared to other known algorithms, the product formulas theoretically perform worse in terms of the computational complexity as a function of time (or system size). However, there are works suggesting that the empirical performance of the algorithm on certain Hamiltonian models is considerably better than the theoretical bound, and can even prevail over any other known algorithm [22]. There exists algorithms that have better scaling in terms of sparsity of the Hamiltonian, and logarithmic dependence on the inverse of error, rather than polynomial [24]. However, another key advantage the product formula algorithm has over all the other known algorithms is that it does not require ancillary qubits. Ancillary qubits are additional qubits that are necessary to run an algorithm, on top of the qubits needed to represent the problem. That means methods based on the product formulas have a resource advantage over all the other known quantum algorithms, making it more suitable for near-term quantum computing.

5.1 Future work

We aim to further explore the randomization of the orderings of the exponentials in the exponential product of a range of approximants of different orders. We aim to test that on various Hamiltonian models and for a range of system sizes. Also, we aim to detail how one would perform the second stage in the three stages "recipe", and demonstrate it with an explicit quantum circuit that solves the spin problem we tackled pragmatically. We wish to build and test such an explicit circuit on IBM Quantum Experience. We aim to implement other known algorithms and test their performance, as well.

This problem is difficult as it has long been open in the case of exponentials of sums and that leaves us with plenty of space for future work. Another goal might be to study algorithms that seek improved performance as a function of the order of the approximant $2k$, which would be useful since the number of exponentials needed for a high-order approximation for large k can be prohibitively expensive using the approach we follow [18]. We hope this work motivates further studies of the product formula method, which has been de-emphasized in recent years in favor of more advanced simulation algorithms that are easier to analyze but harder to implement. Indeed, despite the sophistication of these "post-PF methods" and their optimality in certain general models, they can be provably outperformed by the product formulas for simulating many quantum systems [16]. This gap between the theoretical bound for the performance of the product formulas and the empirical performance on certain Hamiltonian models is not yet fully understood and needs more investigation in the years to come.

6 Summary

The problem of simulating quantum dynamics was the original motivation for quantum computers and remains their major near-term application. The simulation of atoms, molecules and biochemical systems is a computationally-expensive problem that has deep implications for a wide range of areas of science and engineering. In this thesis, we reviewed a quantum algorithm for simulating Hamiltonian dynamics based on the Lie-Trotter-Suzuki product formulas. Examples generated in Python and Mathematica were presented demonstrating the performance of the algorithm. Our results for the evolution time, and time step dependence of $O(m^2 t(t\Delta t)^{\frac{1}{2k}})$ for the $2k^{th}$ order formula are consistent with the widely accepted query complexity of $O(d^3 t(\frac{\Delta t}{\epsilon})^{\frac{1}{2k}})$ [3]. We find that although the product formula method requires more resources when compared to the newer techniques, it is extremely simple and can sometimes - somehow - outperform them empirically. Product formulas provide useful mathematical tools that can be utilized in the field of quantum computing as a fairly simple recipe to develop quantum algorithms for simulating quantum systems efficiently. That, in theory, allows for computations of longer times, larger scales, and higher resolutions to be achieved.

7 Acknowledgements

In this thesis, I am studying under the tutelage of Professor Chris Herdman and Professor James Whitfield. I would like to thank you both for making this project possible through your support as my senior project and thesis supervisors. I would like to acknowledge Professor Whitfield for motivating this project after I joined his research group at Dartmouth College in 2019.

References

[1] Naomichi Hatano and Masuo Suzuki. Finding exponential product formulas of higher orders. *Lecture Notes in Physics*, page 37–68, Nov 2005.

[2] Masuo Suzuki. Generalized trotter's formula and systematic approximants of exponential operators and inner derivations with applications to many-body problems. *Commun. Math. Phys.*, 51:183, 1976.

[3] Andrew M. Childs and Robin Kothari. Simulating sparse hamiltonians with star decompositions. *Lecture Notes in Computer Science*, page 94–103, 2011.

[4] Seth Lloyd. Universal quantum simulators. *Science*, 273(5278):1073–8, 1996.

[5] P.-S. Laplace. *Philosophical Essay on Probabilities.* Springer-Verlag, 1825.

[6] Ashley Montanaro. "quantum algorithms: an overview. *Nature*, 2:15023, 2016.

[7] Kothari, Robin. Efficient simulation of hamiltonians, 2010.

[8] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Phys. Rev. A*, 52:R2493–R2496, 1995.

[9] R.P. Feynman. Simulating physics with computers. *Int. J. Theor. Phys.*, 21(467–488), 1982.

[10] Ryan Babbush, Nathan Wiebe, Jarrod McClean, James McClain, Hartmut Neven, and Garnet Kin-Lic Chan. Low-depth quantum simulation of materials. *Phys. Rev. X*, 8:011044, 2018.

[11] J. Welser. 5in5: Five innovations that will change our lives within five years. *IBM*, 2019.

[12] R. Kothari. Quantum algorithms for hamiltonian simulation: Recent results and open problems. *IBM Research*, 2017.

[13] Dorit Aharonov and Amnon Ta-Shma. Adiabatic quantum state generation and statistical zero knowledge. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03, page 20–29, 2003.

[14] Dominic W. Berry, Graeme Ahokas, Richard Cleve, and Barry C. Sanders. Efficient quantum algorithms for simulating sparse hamiltonians. *Commun. Math. Phys*, 270:359–371, 2007.

[15] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Exponential improvement in precision for simulating sparse hamiltonians. *Proceedings of the 46th Annual ACM Symposium on Theory of Computing - STOC '14*, 2014.

[16] Andrew M. Childs, Yuan Su, Minh C. Tran, Nathan Wiebe, and Shuchen Zhu. A theory of trotter error, 2020.

[17] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103(15), 2009.

[18] Andrew M. Childs and Nathan Wiebe. Product formulas for exponentials of commutators. *Journal of Mathematical Physics*, 54:062202, 2013.

[19] Werner Heisenberg. Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik. *Z. Phys.*, 43(172–198), 1927.

[20] Robert B. Griffiths. Hilbert space quantum mechanics, 2014.

[21] P. Virtanen, R. Gommers, Travis E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, Stéfan J. van der Walt, M. Brett, J. Wilson, K. Jarrod Millman, N. Mayorov, Andrew R. J. Nelson, E. Jones, R. Kern, E. Larson, C J Carey, İlhan Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa,

P. van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[22] Andrew M. Childs, Aaron Ostrander, and Yuan Su. Faster quantum simulation by randomization. *Quantum*, 3:182, 2019.

[23] Earl Campbell. Random compiler for fast hamiltonian simulation. *Phys. Rev. Lett.*, 123:070503, 2019.

[24] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Simulating hamiltonian dynamics with a truncated taylor series. *Phys. Rev. Lett.*, 114(9), 2015.

[25] Guang Hao Low and Isaac L. Chuang. Hamiltonian simulation by qubitization. *Quantum*, 2016.

[26] Dominic W. Berry and Andrew M. Childs. Black-box hamiltonian simulation and unitary implementation. *Quantum Information and Computation*, 12(1-2):29–62, 2012.

[27] Andrew M. Childs. On the relationship between continuous- and discrete-time quantum walk. *Communications in Mathematical Physics*, 294(2):581–603, 2009.

[28] Dominic W. Berry, Andrew M. Childs, and Robin Kothari. Hamiltonian simulation with nearly optimal dependence on all parameters. *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, 2015.

[29] Guang Hao Low and Isaac L. Chuang. Optimal hamiltonian simulation by quantum signal processing. *Phys. Rev. Lett.*, 118:010501, 2017.

Appendix A Perturbational approximation: Taylor series

Product formulas preserve unitarity, and thus probabilities and norms, as we discussed in the Background and demonstrated Sec. 4. This makes the Trotter-Suzuki decomposition extremely useful. We saw how using the perturbational approximation using the Taylor series expansion of the time evolution operator truncated at some arbitrary order k is not unitary. Consider perturbational approximants using the Taylor series expansion of the exponential function (1) for the Hamiltonian $\hat{H}_2$. We plot the error in the solutions provided by the sum of the first few terms. Note the increase in error over time is happening at a faster rate the higher is the order of truncation. More importantly, notice the monotonic nature of the increase in error over time which is the result of the non-unitarity of the approximants. See Fig. 17

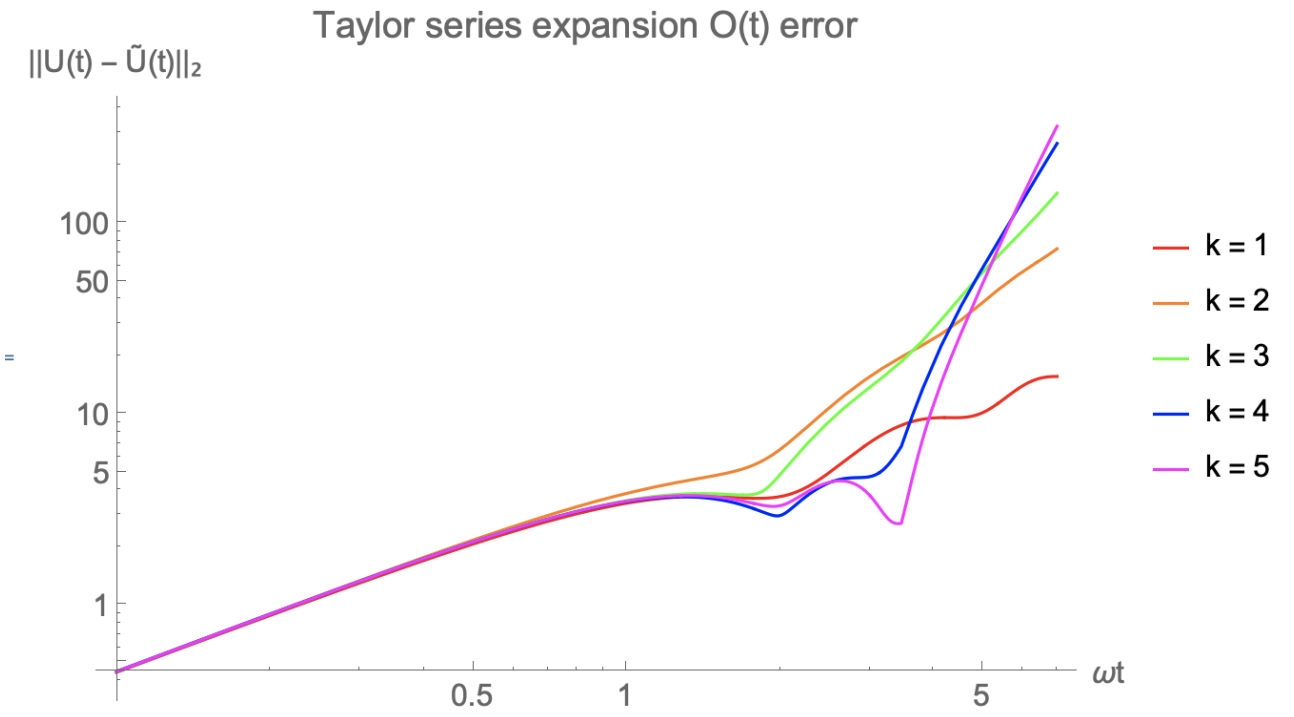


Figure (17) log-log plot of error of the first few terms of the Taylor series of the evolution operator for the Hamiltonian $\hat{H}_2$.

Appendix B Local v. sparse Hamiltonian

There are two classes of Hamiltonians that occur widely in nature and algorithmic applications: k-local and d-sparse Hamiltonians. A k-local Hamiltonian is a Hamiltonian that is a linear combination of m k-local terms

$$H = \sum_{j=1}^{m} \alpha_j H_j, \tag{76}$$

each weighted by its coefficient α_j, where each term acts on at most k particles in the system, or qubits. Local Hamiltonians arise commonly in studying systems of nature, namely those whose Hamiltonian is a sum of local interactions acting nontrivially on a small number of subsystems of limited dimension [4], such as two-body systems. On the other hand, a d-sparse Hamiltonian is a Hamiltonian that has at most d nonzero entries in any given row or column. This can be related to the model of local Hamiltonians when the number of the nonzero entries d is proportional to the number of terms k and exponential in the dimensionality $d = \text{Order}(\text{polylog}N)$ where N is the number of dimensions in the system [25]. The class of sparse Hamiltonians is a richer class of Hamiltonians than the class of local Hamiltonians and has more broad applications than representing physical systems. Generally, any local Hamiltonian can be decomposed into a sum of sparse terms. An important assumption made here is that sparse Hamiltonians are structured, too. That means that there exists a black-box function or an "oracle" that can efficiently - using a constant number of operations $O(1)$- determine the location and value of the j^{th} nonzero element in the i^{th} row [12].

Appendix C Other known algorithms

Quantum algorithms developed for simulating Hamiltonian dynamics - see Fig. 3 fall in two general categories: divide and conquer algorithms, and quantum walk algorithms [12]. The division step of the first class of algorithms decomposes the Hamiltonian into a sum of sparse terms. The decomposition of a local Hamiltonian is "easy" and generates $O(d)$ terms for a d-sparse local Hamiltonian. The decomposition of a sparse Hamiltonian, however, is not as easy and generates $O(d^2)$ 1-sparse terms for a d-sparse Hamiltonian using graph coloring techniques [3]. This causes all the known algorithms in this class to scale up as d^2 at best [12]. The second step, to conquer, after the Hamiltonian is decomposed into simpler terms, is to evolve each term and recombine the evolution. The use of product formulas was the sole efficient method to evolve Hamiltonians, first explicitly achieved by Lloyd in 1996 for k-local Hamiltonians [4], until the recent surge of new efficient Hamiltonian simulation algorithms. Subsequent work to that of Lloyd's considered the broader class of d-sparse Hamiltonians [13, 14, 15]. Some of the methods in the first class are based on techniques of representing the evolution as a Linear Combination of Unitaries (LCU) and applying Oblivious Amplitude Amplification (OAA) to recombine the evolution [24], and Quantum Signal Processing (QSP) [25]. The first quantum walk algorithms used a method of phase estimation, followed by algorithms based on LCU+OAA and QSP as well. The state of the art in algorithmic realizations of quantum simulations is represented by algorithms based on techniques of representing the time evolution as a Linear Combination of Unitaries (LCU) and applying Oblivious Amplitude Amplification (OAA) to approximate the evolution [24], alternate methods of use for product formulas [3], and quantum signal processing [25].

The product formula algorithm have nearly optimal scaling with ε, especially when considering algorithms that use higher-order formulas [14, 3]. Table 5 compares six different quantum algorithms developed recently to solve the Hamiltonian simulation problem based on their query complexity. The table demonstrates an interesting trend of increased speed-ups over the past decade with new

algorithms promising lower bounds on the order $\log 1/\varepsilon$. Quantum signal processing algorithms seem to be a strong contestant where the logarithmic dependence on the error is an additive term rather than a multiplicative one as is the case in the LCU methods. Although quantum walk algorithms perform better in terms of evolution time times the max-norm of the Hamiltonian $t\|\hat{H}\|_{max}$, and sparsity d, their dependence on the error ε is significantly worse [26].

Class of algorithm	Algorithm	Query complexity
Divide & conquer	Product formula [14, 3]	$O(d^3 t(\Delta t/\varepsilon)^{1/2k})$
Quantum walks	Phase estimation [27, 26]	$O(dt/\varepsilon)$
Divide & conquer	Fractional queries [15] or Taylor series [24]	$O(d^2 t\frac{\log d^2 t/\varepsilon}{\log\log d^2 t/\varepsilon})$
Quantum walks	Linear combination of quantum walks [28]	$O(dt\frac{\log dt/\varepsilon}{\log\log dt/\varepsilon})$
Divide & conquer	Quantum signal processing [25]	$O(dt+\log 1/\varepsilon)$
Quantum walks	Qubitization/quantum signal processing [29]	$O(d^2 t+\log 1/\varepsilon)$

Table (5) Table compares query complexity of a number of Hamiltonian simulation algorithms. Here it is assumed that $\|H\|_{max}=1$.

Appendix D Code

Here we review the code we used for the evolution of the 1-qubit system (precession of a spin-$1/2$ particle) to illustrate how one could code up the algorithm.

We start by importing libraries and defining two auxiliary functions.

Precession of a spin-1/2 particle in a magentic field

Atomic units | hbar = 1

```
import math as math
import numpy as np
import scipy.linalg as spla
from matplotlib import pyplot as plt

j = complex(0,1)  #imaginary number i
evolutionTime = 1.0 #testing prop
```

```
[3] def findProbability(psi, phi):
      #find probability of state psi being in the phi state P = || |psi⟩⟨phi| ||^2, returns probability as float
      probabPsiPhi =  np.abs(np.vdot(psi, phi))**2 #Probability of two states 'matching'
      return probabPsiPhi

    def diffProb(probA, probB):
      #function finds difference of probabilities
      difference = probA - probB
      return difference
```

We define the Hamiltonian represented as a Pauli matrix. Then, we define the computational basis of the one-qubit system as the two eigenstates of the Hamiltonian, $|0\rangle$ and $|1\rangle$, and the two states of interest, $|+x\rangle$ and $|-x\rangle$, as a superposition of the eigenstates.

Represent Hamiltonian **H** of the particle's spin due to a constant magnetic field B_0 solely in the z-direction as a z-spin operator **S_z**

H = **w_0 S_z** | w_0 = (g e B_0)/(2m c)

```
#define pauli matrices σ_x, σ_y, σ_z
pauliX = np.array([[0,1],[1,0]])
pauliY = np.array([[0,-1j],[1j,0]])
pauliZ = np.array([[1,0],[0,-1]])

#define the Hamiltonian
ħ = 1 #constant
w0 = 1 #constant
spinZ = pauliZ /2 # S_z = 1/2 σ_z
hamiltonian = w0 * ħ * spinZ

propH1 = w0 * ħ * (pauliX+pauliZ)/2
propH2 = [w0 * ħ * pauliX/2,w0 * ħ * pauliZ/2]
```

```
[5] #define ket and bra of up/down z state vectors as the two logical states making up the cmputaional basis
    ketPlusZ = np.array([[1],[0]])    #up-state     |+z⟩
    braPlusZ = ketPlusZ.transpose()
    ketMinusZ = np.array([[0],[1]])   #down-state   |-z⟩
    braMinusZ = ketMinusZ.transpose()

    #define ket and bra of up/down x state vectors
    ketPlusX = 1/np.sqrt(2) * (ketPlusZ + ketMinusZ)   #up      |+x⟩ = 1/√2 [|+z⟩ + |-z⟩]
    braPlusX = ketPlusX.transpose()
    ketMinusX = 1/np.sqrt(2) * (ketPlusZ - ketMinusZ)  #down    |-x⟩ = 1/√2 [|+z⟩ - |-z⟩]
    braMinusX = ketMinusX.transpose()
```

```
[6] np.dot(hamiltonian,ketMinusZ)   #sanity check: H|-z⟩ = eigenMinus |-z⟩ = -1/2 |-z⟩

    array([[ 0. ],
           [-0.5]])
```

Then, we plot the analytical solution to the spin-up and spin-down probabilities P_{+x} and P_{+x} as defined in Eq. 61 and Eq. 62 for reference.

Example

Consider states of Ψ(t) where state of Ψ(0) are initizlized on x-y plane (up/down x states)

Example: Ψ(0) = |+x⟩

Ψ(t) = 1/√2 (exp(-i w0 t/2) |+z⟩ + exp(+i w0 t/2) |-z⟩)

Analytical Solution

Plot the anayltytical probabilities of being in up/down spin states along x-axis (|+x⟩, |-x⟩)

P_i = |c_i|^2

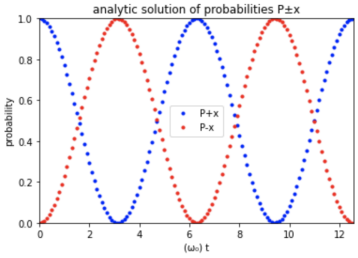
Example: P_+x = |c_+x|^2 = |⟨+x|Ψ(t)⟩|^2

```
[7] #analytical probabilities P±x
    time = np.arange(0,4*np.pi,0.1) #time interval
    probEigenPlus = [np.cos(((w0*k)/2))**2 for k in time] #spin-up probability P+x
    probEigenMinus = [np.sin(((w0*q)/2))**2 for q in time] #spin-down probability P-x
```

```
[8] #plot the analytical probabilities P±x
    plt.plot(time, probEigenPlus, 'b.', time, probEigenMinus, 'r.')
    plt.ylim(0, 1)
    plt.xlim(0,4*np.pi)
    plt.xlabel('(ω₀) t')
    plt.ylabel('probability')
    plt.legend(['P+x','P-x'])
    plt.title("analytic solution of probabilities P±x")
    plt.rcParams.update({'font.size': 12})
    plt.show()
```

Here we define a numerical solver that given an initial state $|\Psi(0)\rangle$, a Hamiltonian H, and evolution time t outputs the evolved state $|\Psi(t)\rangle$ by exponentiating the Hamiltonian matrix using the "scipy.linalg.expm" function.

[8]



Numerical Solution

The fucntion evolvePsiExpm(hamiltonian, psi0, time) evolves an initial state over a given Hamiltonian and time.

```
[9] def evolvePsiExpm(hamiltonian, psi0, time):
      #function evolves Psi0 in time under the evolution operator the hamiltonian generates using spla.expm, returns evolved state Psi(t)
      unitary = spla.expm(-j * hamiltonian * time) #time evolution unitary opertor

      psiT = np.dot(unitary, psi0) # Psi(t) = U(t) Psi(0)

      return psiT
```

```
[10] round(findProbability(ketPlusZ, evolvePsiExpm(hamiltonian,ketPlusZ,evolutionTime)),15) #sanity check: U(t)|+z⟩ = +1      Round to 15 decimals

     1.0
```

Here we define the PF solver for the approximate evolved state. The function takes the same inputs as the numerical solver, in addition to the Trotter number $Q = t/\Delta t$.

PF Solution

The fucntion evolvePsiPF(hamiltonian, psi0, time, timeStep) evolves an initial state over a given Hamiltonian, time and time step.

```
def evolvePsiPF(hamiltonian, psi0, time, TrotterNumber):
  #function evolves psi0 in time under the hamiltonian (list) generated evolution operator using first-order PF, returns evolved state psi(t)
  unitaryPF = spla.expm(-j * hamiltonian[0] * time/TrotterNumber) #initialize product of unitary evolution operators

  k = len(hamiltonian) #order of k-local hamiltonian
  if k > 1: #check: one term H
    for i in range(1, k): #k-1 terms in k range, start at 1 : unitaryPF was intialized with 1st term
      unitaryPF = np.matmul(spla.expm(- j * hamiltonian[i] * time/TrotterNumber), unitaryPF) #multiply product terms using matrix multiplication

  unitaryPF = np.linalg.matrix_power(unitaryPF, TrotterNumber) #repeat Trotter number since broken into Trotter steps

  psiT = np.dot(unitaryPF, psi0) #initialize Psi(t) | Psi(t) = unitaryPF Psi(0)

  return psiT
```

```
[12] round(findProbability(ketPlusZ, evolvePsiPF([hamiltonian],ketPlusZ,evolutionTime,1)), 15) #sanity check: U(t)|+z⟩ = +1       Round to 15 decimals

     1.0
```

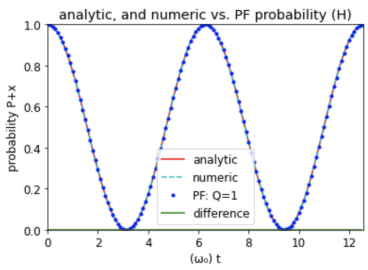
Then, we plot the PF solution to the probability P_{+x} against the analytic and numerical solutions over an appropriate evolution time and Trotter number for the Hamiltonian H defined in Eq. 53 and the Hamiltonian $H + H$.

Plot numerical vs. PF spin-up probability P+x

```
[13] #numeric and analytic probability P+x
     npTime = np.linspace(0, len(time)-1,len(time),dtype=int)

     probNum = [findProbability(ketPlusX, evolvePsiExpm(hamiltonian,ketPlusX,q)) for q in time] #numeric: hamiltonian, spin-up probability P+x
     probAnl = [np.cos(((w0*k)/2))**2 for k in time] #analytic: spin-up probability P+x
```

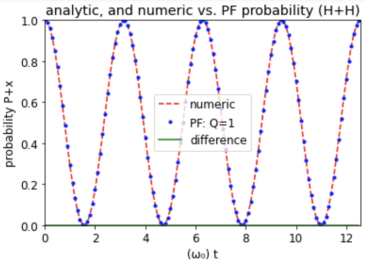
```
#PF spin-up probability P+x, one Hamiltonian term: hamiltonian
probPF = [findProbability(ketPlusX, evolvePsiPF([hamiltonian],ketPlusX,q,1)) for q in time] #PF: hamiltonian , Q = 1, probability P+x

diffProbPF = [diffProb(probNum[q], probPF[q]) for q in npTime]

#plot the numeric vs. PF and difference of probability P+x
plt.plot(time, probAnl, 'r-', time, probNum, 'c--', time, probPF, 'b.', time, diffProbPF, 'g')
plt.ylim(0, 1)
plt.xlim(0,4*np.pi)
plt.xlabel('(ω₀) t')
plt.ylabel('probability P+x')
plt.legend(['analytic','numeric', 'PF: Q=1', 'difference'])
plt.title("analytic, and numeric vs. PF probability (H)")
plt.show()
```

Here we find the difference between the numerical and the PF solutions to the probability P_{+x} for different Trotter numbers and evolution time, and plot them.




```
[15] #numeric vs. PF spin-up probability P+x, one Hamiltonian term: H2 = hamiltonian + hamiltonian
     probNum1 = [findProbability(ketPlusX, evolvePsiExpm(hamiltonian+hamiltonian,ketPlusX,q)) for q in time] #numeric: H2, up probability P+x
     probPF1 = [findProbability(ketPlusX, evolvePsiPF([hamiltonian,hamiltonian],ketPlusX,q,1)) for q in time] #PF: H2 , Q = 1, probability P+x

     diffProbPF1 = [diffProb(probNum1[q], probPF1[q]) for q in npTime]

     #plot numeric vs. PF and difference of probability P+x
     plt.plot(time, probNum1, 'r--', time, probPF1, 'b.', time, diffProbPF1, 'g')
     plt.ylim(0, 1)
     plt.xlim(0,4*np.pi)
     plt.xlabel('(ω₀) t')
     plt.ylabel('probability P+x')
     plt.legend(['numeric', 'PF: Q=1', 'difference'])
     plt.title("analytic, and numeric vs. PF probability (H+H)")
     plt.show()
```

[15]

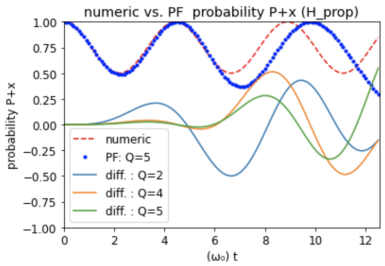



```
#numeric vs. PF probability P+x, anticommuting terms
probNumAnti = [findProbability(ketPlusX, evolvePsiExpm(propH1,ketPlusX,q)) for q in time] #num: propH1 , up probability P+x
probPFAnti = [findProbability(ketPlusX, evolvePsiPF(propH2,ketPlusX,q,2)) for q in time] #PF: propH2 , Q = 2, up probability P+x
probPFAnti1 = [findProbability(ketPlusX, evolvePsiPF(propH2,ketPlusX,q,4)) for q in time] #PF: propH2 , Q = 4, up probability P+x
probPFAnti2 = [findProbability(ketPlusX, evolvePsiPF(propH2,ketPlusX,q,5)) for q in time] #PF: propH2 , Q = 5, up probability P+x

diffProbPFAnti = [diffProb(probNumAnti[q], probPFAnti[q]) for q in npTime]
diffProbPFAnti1 = [diffProb(probNumAnti[q], probPFAnti1[q]) for q in npTime]
diffProbPFAnti2 = [diffProb(probNumAnti[q], probPFAnti2[q]) for q in npTime]

#plot numeric vs. PF and difference for probability P+x , Q = 1
plt.plot(time, probNumAnti, 'r--', time, probPFAnti2, 'b.', time, diffProbPFAnti, time, diffProbPFAnti1, time, diffProbPFAnti2)
plt.ylim(-1,1)
plt.xlim(0,4*np.pi)
plt.xlabel('(ω₀) t')
plt.ylabel('probability P+x')
plt.legend(['numeric','PF: Q=5','diff. : Q=2','diff. : Q=4','diff. : Q=5'],loc='lower left')
plt.title("numeric vs. PF  probability P+x (H_prop)")
plt.show()
```

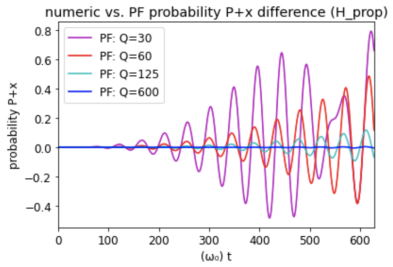
Here we find the probability of the numerical and PF solutions to the probability P_{+x} over an appropriate range of Trotter numbers and plot them.

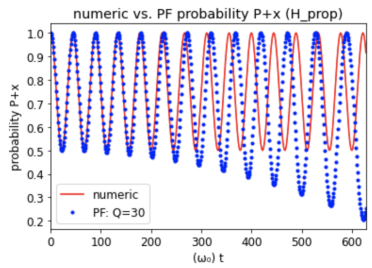
Here we find the difference between the numerical and the PF solution to the probability P_{+x} over a range of Trotter number and plot them.

[16]




```
#numeric vs. PF probability P+x, anticommuting terms, varying Q
time = np.arange(0,20*np.pi,0.1)
npTime = np.linspace(0, len(time)-1,len(time),dtype=int)

probNumAnti_2 = [findProbability(ketPlusX, evolvePsiExpm(propH1,ketPlusX,k)) for k in time]      #num: propH1 , up probability P+x
probPFAnti_1 = [findProbability(ketPlusX, evolvePsiPF(propH2,ketPlusX,q,30)) for q in time]      #PF: propH2 , Q = 30, up probability P+x
probPFAnti_2 = [findProbability(ketPlusX, evolvePsiPF(propH2,ketPlusX,l,60)) for l in time]     #PF: propH2 , Q = 60, up probability P+x
probPFAnti_3 = [findProbability(ketPlusX, evolvePsiPF(propH2,ketPlusX,m,125)) for m in time]    #PF: propH2 , Q = 125, up probability P+x
probPFAnti_4 = [findProbability(ketPlusX, evolvePsiPF(propH2,ketPlusX,m,1000)) for m in time]   #PF: propH2 , Q = 1000, up probability P+x

#plot numeric vs. PF and difference for probability P+x
plt.plot(npTime, probNumAnti_2, 'r-', npTime, probPFAnti_1, 'b.')
plt.xlim(0, 200*np.pi)
plt.xlabel('(ω₀) t')
plt.ylabel('probability P+x')
plt.legend(['numeric','PF: Q=30'])
plt.title("numeric vs. PF probability P+x (H_prop)")
plt.show()
```





[17]

```
#difference of numeric vs. PF probability P+x, anticommuting terms, varying Q
difProbPFAnti_1 = [diffProb(probNumAnti_2[q], probPFAnti_1[q]) for q in npTime]
difProbPFAnti_2 = [diffProb(probNumAnti_2[q], probPFAnti_2[q]) for q in npTime]
difProbPFAnti_3 = [diffProb(probNumAnti_2[q], probPFAnti_3[q]) for q in npTime]
difProbPFAnti_4 = [diffProb(probNumAnti_2[q], probPFAnti_4[q]) for q in npTime]

#plot numeric vs. PF difference for probability P+x, varying Q
plt.plot(npTime, difProbPFAnti_1, 'm-', npTime, difProbPFAnti_2, 'r-', npTime, difProbPFAnti_3, 'c-', npTime, difProbPFAnti_4, 'b-')
plt.xlim(0, 200*np.pi)
plt.xlabel('(ω₀) t')
plt.ylabel('probability P+x')
plt.legend(['PF: Q=30','PF: Q=60','PF: Q=125','PF: Q=600'],loc='upper left')
plt.title("numeric vs. PF probability P+x difference (H_prop)")
plt.show()
```

We define a function to find the error ε of the PF approximate solution of the time evolution unitary, without reference to the quantum states it applies on. The error is the matrix two-norm of the unitary that is the difference between the numerical solution and the PF one.

Then, we plot error over a range of evolution time for a number of Trotter numbers to observe

Error = || U_numeric - U_PF ||_2

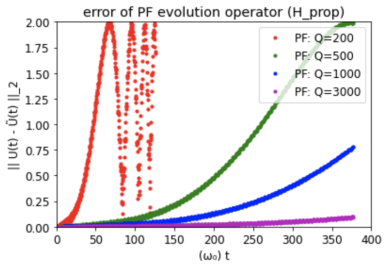
```
[19] def ErrorNormPF(hamiltonian, time, TrotterNumber):
       #function takes hamiltonian (list) and finds norm of difference between time evolution unitaries, PF vs. numeric spla.expm, returns error

       #evolve time under the hamiltonian using spla.expm, returns PF evolution opertaor
       hamiltonianExpm = sum(hamiltonian)
       unitaryExpm = spla.expm(-j * hamiltonianExpm * time) #time evolution opertor

       #evolve time under the hamiltonian using first-order PF, returns PF evolution opertaor
       unitaryPF = spla.expm(-j * hamiltonian[0] * time/TrotterNumber) #initialize product of (unitary evolution operators)

       k = len(hamiltonian) #order of k-local hamiltonian
       if k > 1: #check: one term H
         for i in range(1, k): #k-1 terms in k range, start at 1 : unitaryPF was intialized with 1st term
           unitaryPF = np.matmul(spla.expm(- j * hamiltonian[i] * time/TrotterNumber), unitaryPF) #multiply product terms using matrix multiplication

       unitaryPF = np.linalg.matrix_power(unitaryPF, TrotterNumber) #repeat Trotter number since broken into Trotter steps

       #find norm of difference
       errorNorm = np.linalg.norm(unitaryExpm - unitaryPF, 2)       #returns matrix 2-norm 'largest singular value'

       return errorNorm
```

```
[20] ErrorNormPF([hamiltonian],evolutionTime,1)    #sanity check: || U_expm(hamiltonian) - U_pf(hamiltonian) || = 0

     0.0
```

```
[21] ErrorNormPF(propH2,evolutionTime,1)   #sanity check: || U_expm(pauliX+pauliZ) - U_pf(pauliX+pauliZ) || > 0

     0.23645877516352262
```

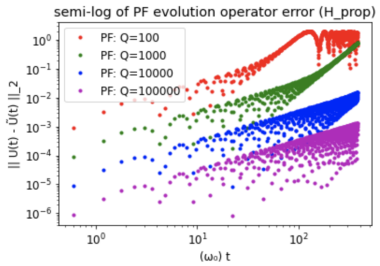
error scaling in time and length of time step.

```
#error of numeric vs. PF time evolution unitary, anticommuting terms, varying Q
timeFactor = 6
errorNorm_1 = [ErrorNormPF(propH2,q, 200) for q in timeFactor*time] #PF: propH2, Q = 200
errorNorm_2 = [ErrorNormPF(propH2,q, 500) for q in timeFactor*time] #PF: propH2, Q = 500
errorNorm_3 = [ErrorNormPF(propH2,q, 1000) for q in timeFactor*time] #PF: propH2, Q = 1000
errorNorm_4 = [ErrorNormPF(propH2,q, 3000) for q in timeFactor*time] #PF: propH2, Q = 5000

#plot error vs. time varying Q
plt.plot(2*time, errorNorm_1, 'r.', timeFactor*time, errorNorm_2, 'g.', timeFactor*time, errorNorm_3, 'b.', timeFactor*time, errorNorm_4, 'm.')
plt.ylim(0,2)
plt.xlim(0,400)
plt.xlabel('(ω₀) t')
plt.ylabel('|| U(t) - Ũ(t) ||_2')
plt.legend(['PF: Q=200','PF: Q=500','PF: Q=1000','PF: Q=3000'],loc='upper right')
plt.title("error of PF evolution operator (H_prop)")
plt.show()
```



Finally, we plot the data on a log-log graph to observe error growth over a range of evolution time for for a number of Trotter numbers to demonstrate that error is scaling in a polynomial-like behavior as a function of evolution time and length of time step.

```
#error of numeric vs. PF time evolution unitary, anticommuting terms, varying Q by orders of magnitude
logErrorNorm_1 = [errorNormPF(propH2,q, 100) for q in timeFactor*time] #PF: propH2, Q = 100
logErrorNorm_2 = [errorNormPF(propH2,q, 1000) for q in timeFactor*time] #PF: propH2, Q = 1000
logErrorNorm_3 = [errorNormPF(propH2,q, 10000) for q in timeFactor*time] #PF: propH2, Q = 10000
logErrorNorm_4 = [errorNormPF(propH2,q, 100000) for q in timeFactor*time] #PF: propH2, Q = 100000

#plot log error vs. time varying Q by orders of magnitude
plt.semilogy(timeFactor*time, logErrorNorm_1, 'r.', timeFactor*time, logErrorNorm_2, 'g.', timeFactor*time, logErrorNorm_3, 'b.', timeFactor*time, logErrorN
#plt.xlim(0,400)
plt.xscale("log")
plt.xlabel('(ω₀) t')
plt.ylabel('|| U(t) - Ũ(t) ||_2')
plt.legend(['PF: Q=100','PF: Q=1000','PF: Q=10000','PF: Q=100000'],loc='upper left')
plt.title("semi-log of PF evolution operator error (H_prop)")
plt.show()
```



This is the end of the code we used to simulate the evolution of the 1-particle system. The Python and Mathematica code for the time evolution simulation of the 2-particle system and the other examples presented throughout the paper can be accessed on GitHub using the link provided here (to be posted..).

Appendix E Analytical solutions

Here we present the analytical solution for the evolution of the 1-qubit system we discussed earlier. First we represent the Hamiltonian as a spin operator,

$$\hat{H} = \omega_0 \hat{S}_z \qquad\qquad \hat{S}_z = \frac{\hbar}{2}\sigma_z = \frac{\hbar}{2}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{77}$$

then define the two eigenstates as the computational basis,

$$|+z\rangle = |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \qquad\qquad |-z\rangle = |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \tag{78}$$

and find the two eigenenergies of the system,

$$\hat{H}|+z\rangle = \frac{\omega_0}{2}\sigma_z|+z\rangle = +\frac{\omega_0}{2}|+z\rangle = E_+|+z\rangle \tag{79}$$

$$\hat{H}|-z\rangle = \frac{\omega_0}{2}\sigma_z|-z\rangle = -\frac{\omega_0}{2}|-z\rangle = E_-|-z\rangle. \tag{80}$$

Then define two states as a superposition of the eigenstates, for our example,

$$|+x\rangle = \frac{1}{\sqrt{2}}[|+z\rangle + |-z\rangle] \qquad\qquad |-x\rangle = \frac{1}{\sqrt{2}}[|+z\rangle - |-z\rangle]. \tag{81}$$

Now, we evolve the $|+x\rangle$ spin state by applying the time evolution operator on the state which is a superposition of the eigenstates,

$$\hat{U}(t)|\Psi(0)\rangle = e^{-i\hat{H}t}|+x\rangle \tag{82}$$

$$= \frac{1}{\sqrt{2}}[e^{-i\hat{H}t}|+z\rangle + e^{-i\hat{H}t}|-z\rangle] \tag{83}$$

$$|\Psi(t)\rangle = \frac{1}{\sqrt{2}}[e^{-i\tilde{E}_+t}|+z\rangle + e^{-i\tilde{E}_-t}|-z\rangle], \tag{84}$$

which is represented as a two-dimensional state vector,

$$|\Psi(0)\rangle = |+x\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} \qquad\qquad |\Psi(t)\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix} e^{-i\tilde{E}_+t} \\ e^{-i\tilde{E}_-t} \end{pmatrix}. \tag{85}$$

Thus, to observe quantum dynamics in motion, we first find the probabilities of the particle being in the $|+z\rangle$ and $|-z\rangle$ spin eigenstate for the evolved state $|\Psi(t)\rangle$, we take the modulus squared of their inner-product.

$$P_+ = |C_+(t)|^2 = |\langle +z|\Psi(t)\rangle|^2 \tag{86}$$

$$= |\begin{pmatrix} 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} e^{-i\tilde{E}_+ t} \\ e^{-i\tilde{E}_- t} \end{pmatrix}|^2 = |\frac{e^{-i\tilde{E}_+ t}}{\sqrt{2}}|^2 = \frac{1}{2} \tag{87}$$

$$P_- = |C_-(t)|^2 = |\langle -z|\Psi(t)\rangle|^2 \tag{88}$$

$$= |\begin{pmatrix} 0 & 1 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} e^{-i\tilde{E}_+ t} \\ e^{-i\tilde{E}_- t} \end{pmatrix}|^2 = |\frac{e^{-i\tilde{E}_- t}}{\sqrt{2}}|^2 = \frac{1}{2}, \tag{89}$$

adding up to a probability of one $P = 1$ as expected since any state the particle can be in is always a superposition of the two spin eigenstates,

$$P_+ + P_- = \frac{1}{2} + \frac{1}{2} = 1. \tag{90}$$

Now, we find the probabilities of the particle being in the $|+x\rangle$ and $|-x\rangle$ spin states for the evolved state $|\Psi(t)\rangle = \hat{U}(t)|+x\rangle$,

$$P_{+x} = |C_{+x}(t)|^2 = |\langle +x|\Psi(t)\rangle|^2 \tag{91}$$

$$= |\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} e^{-i\tilde{E}_+ t} \\ e^{-i\tilde{E}_- t} \end{pmatrix}|^2 = |\frac{1}{2}(e^{-iE_+ z t} + e^{-iE_- z t})|^2 \tag{92}$$

$$= \cos^2 \frac{\omega_0 t}{2}, \tag{93}$$

where we made use of the trigonometric identity $\cos x = \frac{e^{ix}+e^{-ix}}{2}$. Doing a similar calculation for P_{-x} yields,

$$P_{-x} = |C_{-x}(t)|^2 = |\langle -x|\Psi(t)\rangle|^2 = \sin^2 \frac{\omega_0 t}{2}, \tag{94}$$

adding up to a probability of one $P = 1$, too, as expected since the two states are a superposition of the two eigenstates,

$$P_{+x} + P_{-x} = \cos^2 \frac{\omega_0 t}{2} + \sin^2 \frac{\omega_0 t}{2} = 1. \tag{95}$$

8 Primer

8.1 Quantum mechanics

Humans perform classical simulations all the time. Using their biological computers - the brain - people predict where should they place their foot next or when should they shoot to hit a moving ball. These classical simulations on our biological computers tend to be approximate in nature, but are mostly instantaneous and largely accurate nonetheless. The time evolution in these simulations tend to be short lived, short in nature. That in turn allows us to come up with an accurate simulated solution, or to be ε-close to the exact solution where ε is the difference between the exact solution and the brain's approximation. Formally, a physical simulation problem occurs when we have a physical system with a known initial state and the dynamics governing its behavior represented as an energy operator or the "Hamiltonian" of the system, and we are interested in knowing something about one of its future states:

> **Given** the initial state of a physical system $|\psi(0)\rangle$, **evolve** the system under the Hamiltonian H over time t to find its final state $|\psi(t)\rangle$ within error ε.

In other words, to get the state of a physical system at some given time a computer needs a description of the state of the system at some earlier time and the Hamiltonian dictating its evolution.

The state of an isolated quantum system is described mathematically with the **wave function** $\Psi(t)$. It is a complex-valued probability amplitude, and the probabilities for the possible results of measurements made on the system can be derived from it.

Classically, the **Hamiltonian** of a system is the sum of the system's kinetic and potential energy $H = K + P$. In quantum mechanics a Hamiltonian is an operator represented in a complex square Hermitian matrix (a $n \times n$ matrix for an n-body system) corresponding to the sum of energies for all particles in the system. Its spectrum is the set of possible outcomes when one measures the total energy of a system.

An **operator** is a function over a space of physical states to another space of physical states. In QM any *observable*, i.e., any quantity which can be measured in a physical experiment, is associated with a self-adjoint linear operator. Applying the operator over a wave function resembles measuring the physical quantity it resembles of the system described by that wave function.

To describe the state of an isolated quantum system, we solve the **Schrödinger equation**, which is a second order differential equation whose solutions describe the allowed states particles can exist in. If $|\psi(t)\rangle$ is the state of the system at time t, then

$$H|\psi(t)\rangle = i\hbar\frac{\partial}{\partial t}|\psi(t)\rangle.$$

It relates the total energy of a quantum mechanical system to its spatial and temporal state. Solving the equation for a given quantum system reveals the eigenenergies of the system,

$$H\Psi = E\Psi,$$

where for a time-independent Hamiltonian, the closed-form solution is the **time evolution operator**,

$$U = e^{-iHt/\hbar}.$$

Now to simulate the evolution of a quantum system for a time-independent Hamiltonian we apply the time evolution operator on the initial quantum state of the system to produce the final state,

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

.

A **time-independent** function has a static description, one that does not change with time. A time-independent Hamiltonian is a Hamiltonian that describes a system where time does not impact the nature of forces acting on the system. A static potential well or magnetic field are examples of that.

A **quantum state** is a mathematical entity whose norm squared provides a probability distribution for the outcomes of a measurement on the system. A pure quantum state can be represented by a state

vector in a Hilbert space over the complex numbers. Pure states are also known as state vectors or wave functions, the latter term applying particularly when they are represented as functions of position or momentum.

Local Hamiltonians are a sum of terms each acting on $O(1)$ qubits. The matrix representing a local Hamiltonian can be seen as a sum of sub-matrices, each representing the interactions local to a subsystem within the overall picture. **Sparse** Hamiltonians are ones where the number of nonzero entries in any given row/column is sparse. A sparse Hamiltonian usually would have poly(logN) nonzero entries in any given row, though no polynomial size description is needed (can be exponential in logN). Generally, any local Hamiltonian can be decomposed into a linear combination of sparse Hamiltonians.

Due to the principle of **superposition**, the state of a quantum mechanical system is in a superposition of all its possible states. That means that while a classical bit is either in $|0\rangle$ or $|1\rangle$ a qubit is in all the possible linear combinations of the two, preserving normalization. **Entanglement**, on the other hand, is more subtle. One can think of two entangled particles as being one system rather than two, the states in which the two entangled particles exist are correlated.

Coherence is the state at which qubits are maintained in isolation of external interference, that is to be able to read them. When the qubit(s) isolation from external affects is broken **decoherence** occurs, causing the states to get mixed up and the computation to become meaningless. Decoherence is a peculiarly quantum form of noise that has no classical analog. It destroys quantum superpositions and is the most important and ubiquitous form of noise in quantum computers and quantum communication channels.

A **qubit** or quantum bit is the basic unit of quantum information. It is the quantum version of the classical binary bit physically realized with a two-state device, such as a transistor. A qubit is a two-state system, too. The only difference is that it is a quantum-mechanical system, meaning it can be in a coherent superposition of the two states simultaneously and that it can be entangled with other

qubits to realize higher correlations than is possible with classical bits. Systems that can be modeled as one qubit include: the spin of the electron in which the two levels can be taken as spin up and spin down; or the polarization of a single photon in which the two states can be taken to be the vertical polarization and the horizontal polarization.

An **oracle** function is an abstract function that is able to solve to take an input and make a decision on the output in a single black-box operation.

8.2 Linear algebra

A **complex number** is a number that can be expressed in the form $a + bi$, where a and b are real numbers, and i satisfies the equation $i^2 = -1$. Because no real number satisfies this equation, i is called an imaginary number. For the complex number $a + bi$, a is called the real part, and b is called the imaginary part.

A **vector** is an element of a vector space. A Euclidean vector is a line connecting two points in a 3-dimensional Euclidean geometry where it has a magnitude, the length of the line, and a direction.

A **coordinate vector** is a representation of a vector as an ordered list of numbers that describes the vector in terms of a particular ordered basis. Coordinates are always specified relative to an ordered basis.

A **state vector** is a unit vector that describes the state of a system. vectors are specified of a particular basis set.

A **basis** set is a linearly independent spanning set. This means that a set of vectors in a vector space is called a basis, if every element of of the vector space may be written in a unique way as a (finite) linear combination of the basis vectors.

A **vector field** is an assignment of a vector to each point in a subset of space.

A **matrix** is a rectangular array or table of numbers, symbols, or expressions, arranged in rows and columns. A major application of matrices is to represent linear transformations. If the matrix is

square, then it is possible to deduce some of its properties by computing its determinant. For example, a square matrix has an inverse if and only if its determinant is not zero. Insight into the geometry of a linear transformation is obtainable (along with other information) from the matrix's eigenvalues and eigenvectors.

A **norm** is a function from a real or complex vector space to the non-negative real numbers that behaves in certain ways like the distance from the origin: it commutes with scaling, obeys a form of the triangle inequality, and is zero only at the origin. In particular, the Euclidean distance of a vector from the origin is a norm, called the Euclidean norm, or 2**-norm**, which may also be defined as the square root of the inner product of a vector with itself. In other words, the two-norm is a mathematical measure of length given by "the square root of the squares" and is denoted by $\|\cdot\|_2$. A **matrix norm** is a vector norm in a vector space whose elements (vectors) are matrices (of given dimensions). There is one type of matrix norms which we need to know about: matrix norms induced by vector norms. In the case of ℓ_2 norm for vectors, the induced matrix norm is the spectral norm. The spectral norm of a matrix A is the largest singular value of A (i.e., the square root of the largest eigenvalue of the matrix A^*A, where A^*A denotes the conjugate transpose of A:

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^*A)} = \sigma_{\max}(A).$$

.

A **linear transformation** (also called a linear mapping) is a mapping between two vector spaces that preserves the operations of addition and scalar multiplication. If V and W are finite-dimensional vector spaces and a basis is defined for each vector space, then every linear map from V to W can be represented by a matrix.

An **eigenvector** of a linear transformation is a nonzero vector that changes by a scalar factor when that linear transformation is applied to it. The corresponding **eigenvalue**, often denoted by λ , is the factor by which the eigenvector is scaled. Eigenvectors and their corresponding eigenvalues are characteristic of a linear transformation.

a **Hilbert space** generalizes the concept of a Euclidean space. It is an inner product space, extending vector algebra and calculus from two or three dimensional spaces to spaces of finite or infinite number of dimensions. That allows for the introduction of geometrical notions such as: the length of a vector and the angle between two vectors. An inner product naturally induces an associated norm which canonically makes every inner product space into a normed vector space. If this normed space is also a Banach space, a complete normed vector space, then the inner product space is called a Hilbert space. In other words, a Hilbert space is a complete inner product space. When a vector space is an inner product space and is complete the concept of orthogonality can be used. In quantum mechanics, Hilbert spaces are used to represent the state space of quantum systems.

A **Hermitian** matrix is a complex square matrix that is equal to its own conjugate transpose. That is,

$$A_{i,j} = \overline{A_{j,i}}.$$

In physics, that it usually denoted by a dagger:

$$A = A^{\dagger}.$$

A **unitary** matrix is a complex square matrix whose inverse is equal to its conjugate transpose. That is,

$$AA^{\dagger} = A^{\dagger}A = I.$$

in the relevant notation. Unitary matrices are of significant importance in quantum mechanics for they preserve norms, and thus probability amplitudes.

The **Pauli matrices** are a set of three 2×2 complex matrices which are Hermitian and unitary. They occur in the Pauli equation which takes into account the interaction of the spin of a particle with an external electromagnetic field.

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \qquad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{96}$$