Solving linear differential equations quantumly

Quantum vs classical methods by example of Poisson's equation

Jonas Broeckmann

9th August 2024

Abstract

Differential equations are used to model a wide range of physical phenomena in areas such as computational fluid dynamics, quantum mechanics and electromagnetism, but also in finance, chemistry and many other fields. Many of these applications already have specifically designed quantum algorithms, some of which show speedups over classical algorithms. This paper will focus on QLSS based approaches for solving linear differential equations by example of Poisson's equation and discuss how the quantum algorithms compare to classical alternatives. An overview of possible approaches and their benefits and costs will be given, as well as a summary of the current state of research in the field.

1 Introduction

Many mathematical descriptions of real-world phenomena are formulated as differential equations. They are equations that describe functions based on their derivatives and are used to model a wide range of physical phenomena in areas such as computational fluid dynamics, quantum mechanics and electromagnetism, but also in finance, chemistry, biology and many other fields [8]. Examples include the heat equation, wave equations and Schrödinger's equation in physics, Black-Scholes equation in finance, and the reaction-diffusion equation in chemistry.

As they are such a widely used tool, it is of interest to investigate how quantum algorithms can be used to solve differential equations and whether they can provide speedups over classical methods. We will start by a brief look at linear differential equations and specifically Poisson's equation, as well as their discretization into a system of linear equations, before introducing the quantum linear system solver (QLSS) and comparing it to classical approaches.

2 Linear differential equations

Differential equations can take many forms, some of which require more advanced methods to solve than others. In this paper, we will focus on linear differential equations, which are a class of differential equations that can be written in the form of a linear combination of derivatives. They are of particular interest because they have well-understood properties and can be solved using a variety of methods. Quantum computing also fits them well, as quantum mechanics is inherently linear [20]. Linear differential equations can further be divided into linear ordinary differential equations (ODEs) and linear partial differential equations (PDEs). Linear PDEs are equations of the form:

$$Lu = f(x_1, ..., x_d) \tag{1}$$

Here L is a linear differential operator, which is a linear combination of partial derivatives with d variables. The function u is the function of interest and f is a given function. Linear ordinary differential equations (ODEs) may be considered a special case of linear PDEs where $d = 1$ and can be written as

$$L = a_0(x) + a_1(x)\frac{\partial}{\partial x} + a_2(x)\frac{\partial^2}{\partial x^2} + ... + a_n(x)\frac{\partial^n}{\partial x^n} \tag{2}$$

where $a_0, ..., a_n$ are functions of x.

In many cases, x represents time, such as in the case of the simple harmonic oscillator. PDEs with more than one independent variable are often used to model phenomena that depend on multiple spatial dimensions, and possibly time as well. Popular examples of linear PDEs are the Schrödinger equation in quantum mechanics and Poisson's equation, which will be discussed later in more detail. Most of the problems that follow from nature are confined to at most three spatial dimensions and one time dimension, but cases with more dimensions also exist, such as in quantum chemistry or quantum many-body problems, in which the dimensionality depends on the number of quantum particles [8, 16].

Analytical solutions to differential equations often provide the most insight into the behavior of the system that is under consideration, because they obtain the exact solutions. While analytical solvers exist for many simple types of differential equations, others are too complex or do not have analytical solutions at all [10, 18]. In these cases, numerical methods are used to approximate the solution computationally. There exist a wide variety of numerical approaches such as algorithmic differentiation [21], spectral methods, the family of Runge-Kutta methods and, related to those, also finite difference methods, which will be the main focus of this paper.

2.1 Poisson's equation

The example used throughout this paper will be Poisson's equation, as it is a simple second-order linear PDE that is used in many areas of physics and engineering to model concepts such as the electric potential or the gravitational field [12]. It was first formulated by the French mathematician Siméon Denis Poisson in the early 19th century and is given by the linear differential operator

$$L = \Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{3}$$

for two-dimensional cartesian coordinates (x, y), but can be generalized to any number of dimensions. The example of Poisson's equation in a quantum algorithmic setting will follow closely[1] to the work of Cao et al. 2013 in the case for $d = 2$ dimensions:

$$\Delta u(x, y) = f(x, y), \quad x, y \in (0, 1) \tag{4}$$

Here f is some given function, often specifying the geometry of the modeled system. We limited this example to the unit square with edges fixed at 0 resulting in the boundary conditions $u(x_0, y) = u(x, y_0) = 0$ with $x_0, y_0 \in \{0, 1\}$.

The runtimes of classical solvers are bounded from below by at least $\Omega\left(M^d\right)$ where M is a measure for the numerical precision, which will be discussed later. As the time complexity scales exponentially with the number of dimensions d, in literature this is often referenced as the "curse of dimensionality" [6]. It is therefore of interest to investigate whether quantum algorithms can provide speedups in this regard.

3 Discretization

In order to solve linear differential equations numerically, they are typically discretized. This means that the continuous domain of the differential equation is divided into a finite number of (grid) points, and the differential equation is approximated by a system of algebraic equations that can be solved using numerical methods. The most common discretization methods are finite difference methods and finite element methods.

Finite difference methods approximate the derivatives in the differential equation by differences of function values at neighboring grid points. That is, by choosing a grid spacing h, the derivative of a function $u(x)$ can be approximated by

$$\frac{\partial u}{\partial x}(x) = \frac{u(x+h) - u(x)}{h} + \epsilon \tag{5}$$

at a point x with $\epsilon \in \mathcal{O}(h)$. For more accurate approximations, central differences can be used, which approximate the derivative by

$$\frac{\partial u}{\partial x}(x) = \frac{u(x+h) - u(x-h)}{2h} + \epsilon \tag{6}$$

at a point x with $\epsilon \in \mathcal{O}(h^2)$ [14]. These methods are also known as Euler's methods, which are a special case of the more general family of Runge-Kutta methods, all of which may be used for discretization.

While linear PDEs and linear ODEs of higher-order can be transformed into systems of first-order differential equations [1], all of these methods are also applicable to

[1]Negative sign ommited for simplicity

higher-order derivatives by iterative constructions. After adjusting the grid points accordingly, we get for second-order derivatives

$$\begin{aligned}\frac{\partial^2 u}{\partial x^2}(x) &\approx \frac{\partial}{\partial x}\frac{u(x+\frac{h}{2})-u(x-\frac{h}{2})}{h}\\ &\approx \frac{u(x+h)-2u(x)+u(x-h)}{h^2}\end{aligned} \tag{7}$$

Notice how for linear PDEs this always results in a linear combination of function values at neighboring grid points, making it possible to delegate further computations to linear algebra solvers [20].

We can now apply these approximations to Poisson's equation (4)

$$\begin{aligned}f(x,y) &= \frac{\partial^2 u}{\partial x^2}(x,y)+\frac{\partial^2 u}{\partial y^2}(x,y)\\ &\approx \frac{u(x+h,y)-2u(x,y)+u(x-h,y)}{h^2}+\frac{u(x,y+h)-2u(x,y)+u(x,y-h)}{h^2}\\ &\approx \frac{1}{h^2}(u(x,y-h)+u(x-h,y)-4u(x,y)+u(x+h,y)+u(x,y+h))\end{aligned} \tag{8}$$

As we want to form a system of linear equations, we then discretize the domain that is the unit square $(0,1)\times(0,1)$ into a square grid with $(M+1)^2$ grid points spaced by $h=\frac{1}{M}$ and defined via

$$x_j = jh \quad \text{and} \quad y_k = kh$$

$$\begin{aligned}f_{j,k} &= f(x_j,y_k)\\ u_{j,k} &= u(x_j,y_k)\end{aligned} \tag{9}$$

with $j,k\in\{0,...,M\}$. This results in a series of linear equations

$$f_{j,k} = \frac{1}{h^2}(\tilde{u}_{j+1,k}-2\tilde{u}_{j,k}+\tilde{u}_{j-1,k}+\tilde{u}_{j,k+1}-2\tilde{u}_{j,k}+\tilde{u}_{j,k-1}) \tag{10}$$

with two-dimensional indices. This form would be sufficient for the next step. But to more easily see how this transforms into matrix form, we will reindex the grid points to a single index l first, such that $j = l \mod (M+1)$ and $k=\lfloor l/(M+1)\rfloor$ or equivalently $l=(j \mod (M+1))+k(M+1)$. This can be understood as numbering the points row-wise instead of by coordinates and will result in

$$f_l = \frac{1}{h^2}(\tilde{u}_{l-(M+1)}+\tilde{u}_{l-1}-4\tilde{u}_l+\tilde{u}_{l+1}+\tilde{u}_{l+(M+1)}) \tag{11}$$

With $0 \leq l < (M+1)^2$ this gives us a total of $(M+1)^2$ linear equations which we will transform into sparse matrix form:

$$\begin{pmatrix} f_0 \\ \vdots \\ f_{(M+1)^2-1} \end{pmatrix} = \frac{1}{h^2} \begin{pmatrix}
\begin{matrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{matrix} & \cdots\cdots\cdots\cdots & \begin{matrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{matrix} \\
\vdots & \begin{matrix}
\begin{matrix} 0 & \cdots & 0 \\ \vdots & D & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & I & \vdots \\ 0 & \cdots & 0 \end{matrix} & \ddots \\
\begin{matrix} 0 & \cdots & 0 \\ \vdots & I & \vdots \\ 0 & \cdots & 0 \end{matrix} & \ddots & \begin{matrix} 0 & \cdots & 0 \\ \vdots & I & \vdots \\ 0 & \cdots & 0 \end{matrix} \\
\ddots & \begin{matrix} 0 & \cdots & 0 \\ \vdots & I & \vdots \\ 0 & \cdots & 0 \end{matrix} & \begin{matrix} 0 & \cdots & 0 \\ \vdots & D & \vdots \\ 0 & \cdots & 0 \end{matrix}
\end{matrix} & \vdots \\
\begin{matrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{matrix} & \cdots\cdots\cdots\cdots & \begin{matrix} 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{matrix}
\end{pmatrix} \begin{pmatrix} \tilde{u}_0 \\ \vdots \\ \tilde{u}_{(M+1)^2-1} \end{pmatrix}$$

Here I is the $(M-1) \times (M-1)$ identity matrix and D is the $(M-1) \times (M-1)$ matrix

$$D = \begin{pmatrix} -4 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -4 \end{pmatrix} \tag{12}$$

The many zero rows and columns stem from the boundary condition, i.e. $u_{j,k} = 0$ if $j \in \{0, M\}$ or $k \in \{0, M\}$. We can remove those to further compress the indices and matrix to obtain the form

$$\vec{f} = L\vec{\tilde{u}} = \frac{1}{h^2} \begin{pmatrix} D & I & & 0 \\ I & \ddots & \ddots & \\ & \ddots & \ddots & I \\ 0 & & I & D \end{pmatrix} \vec{\tilde{u}} \tag{13}$$

where L is a $(M-1)^2 \times (M-1)^2$ matrix. This generalizes to higher dimensions d such that L will be an $N \times N$ matrix where $N = (M-1)^d$ [6].

From here we may take one of two approaches in quantum computing: The first is to directly apply a quantum linear system solver as discussed in the next section to solve equation (1) with L derived as above. This is the general approach as most linear PDEs can be transformed into systems of linear equations, but has some drawbacks in this case. The second is much more specific to Poisson's equation, but can take advantage of the specific structure of L. It will be discussed in section 4.3.

4 QLSS

The quantum linear system solver (QLSS) is a general method for solving the classical problem

$$A\vec{x} = \vec{b} \tag{14}$$

on quantum computers. The original version is named HHL after the authors (Harrow, Hassidim and Lloyd) and was published in 2009 [17]. While newer versions exist that improve on HHL further, such as the state-of-the-art method from 2022 [7], we will focus on the original version for simplicity.

4.1 Problem statement

Most of the methods for solving linear PDEs as formulated in equation (1) naturally result in systems of linear equations, which can be solved using linear algebra techniques [14]. The vectors

$$\begin{aligned} \vec{f} &= (f_0, ..., f_{N-1}) \\ \vec{u} &= (u_0, ..., u_{N-1}) \end{aligned} \tag{15}$$

required in these techniques follow from the necessary discretization at some grid points $x_0, ..., x_{N-1}$ of the continuous domain of the differential equation (see section 3).

Solving differential equations in a computational context also means extracting specific properties of the function u under consideration. This property is often the value of the function at a specific point, but may also require further (quantum) computation to obtain [8]. A general quantum solver for differential equations would therefore need to be able to prepare some state $|\tilde{u}\rangle$, that is close to the desired solution state $|u\rangle$ up to some error ϵ:

$$\| |\tilde{u}\rangle - |u\rangle \| \leq \epsilon \tag{16}$$

These two requirements let us then restate the problem for quantum algorithms to solving

$$L |u\rangle = |f\rangle \tag{17}$$

where the states

$$|f\rangle = \sum_{j=0}^{N-1} \frac{f_j}{\|f\|} |j\rangle \quad \text{and} \quad |u\rangle = \sum_{j=0}^{N-1} \frac{u_j}{\|u\|} |j\rangle \tag{18}$$

are normalized quantum states representing the functions f and u after the discretization. Notice how the qubits necessary for the representations of $|f\rangle$ and $|u\rangle$ are exponentially smaller than the grid points necessary for classical representations. The construction of L heavily depends on the chosen method for discretization and can be performed more efficiently for some problems (see subsection 4.3).

To see how QLSS is able to prepare the target state $|\tilde{u}\rangle$, we rephrase the original problem (17) as the matrix inversion problem

$$|u\rangle = L^{-1}|f\rangle = L^{-1}\sum_{j=0}^{N-1}\beta_j|j\rangle \tag{19}$$

where $\beta_j = \frac{f_j}{\|f\|}$. Without loss of generality we assume that L is Hermitian, then there exists an eigendecomposition of the form

$$\begin{aligned} L|j\rangle &= \lambda_j|j\rangle \\ \Rightarrow \quad L^{-1}|j\rangle &= \frac{1}{\lambda_j}|j\rangle \end{aligned} \tag{20}$$

where λ_j are the eigenvalues and $|j\rangle$ the eigenstates of L. Inserting (20) into (19) reduces the problem to finding the inverse of the eigenvalues of L:

$$|u\rangle = \sum_{j=0}^{N-1}\frac{\beta_j}{\lambda_j}|j\rangle \tag{21}$$

Given the problem stated in this form, we are now able to construct the QLSS algorithm.

4.2 Algorithm

We assume that $|f\rangle$ is prepared and that we have access to L via quantum random access memory (QRAM) giving us the unitary $U_L = e^{i\alpha L}$. As show in Figure 1, the algorithm consists of three main steps:

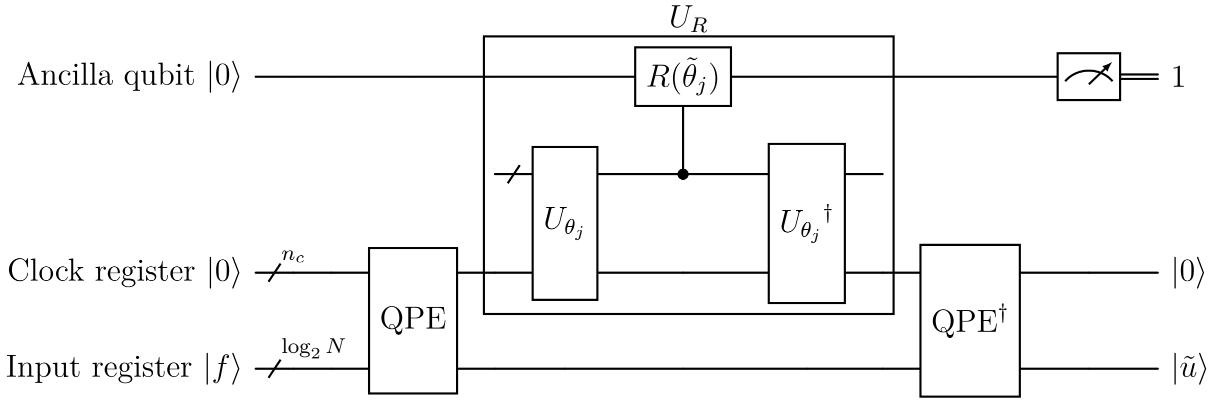


Figure 1: Circuit of the QLSS algorithm

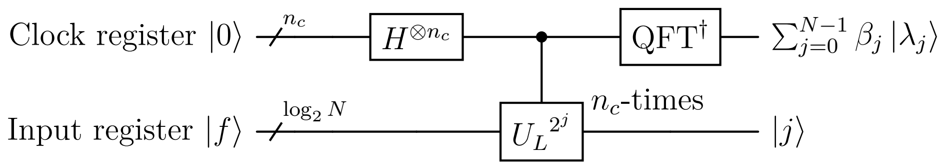
(0.) Prepare the input register.

Although this may not be trivial, we assume that the input register is prepared in the state

$$|f\rangle = \sum_{j=0}^{N-1} \beta_j |j\rangle \tag{22}$$

as defined above.

1. Apply quantum phase estimation (QPE) to the input and clock register.



Clock register $|0\rangle$
n_c
$H^{\otimes n_c}$
QFT†
$\sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle$
Input register $|f\rangle$
$\log_2 N$
$U_L{}^{2^j}$
n_c-times
$|j\rangle$

Figure 2: QPE algorithm used in Figure 1

The QPE algorithm (Figure 2) is used to estimate the eigenvalues of L. For this it utilises the unitary $U_L = e^{i\alpha L}$ which in the general case is realized via the use of queries to quantum random access memory. The application of QPE results in the state

$$\text{QPE}\left(|0\rangle^{\otimes n_c} \otimes \sum_{j=0}^{N-1} \beta_j |j\rangle\right) = \sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle \otimes |j\rangle \tag{23}$$

Here $n_c \in \mathcal{O}(\log(1/\epsilon))$ controls the precision of the resulting eigenvalues.

2. Apply controlled rotations on the ancilla qubit.

   a) Calculate $\tilde{\theta}_j = \sin^{-1}(C/\lambda_j)$ using an implementation of the inverse and trigonometric function in U_{θ_j} [6]. The result is stored in an intermediate register the size of which controls the precision of the calculation.

   b) Apply a controlled R_y rotation for each calculated angle $\tilde{\theta}_j$ to the ancilla qubit.

   c) Uncompute 2a.

This results in the top wires being in the state

$$U_R\left(|0\rangle \otimes \sum_{j=0}^{N-1} \beta_j |\lambda_j\rangle\right) = \sum_{j=0}^{N-1} \beta_j \left(\sqrt{1 - \frac{C^2}{\tilde{\lambda}_j^2}}\, |0\rangle + \frac{C}{\tilde{\lambda}_j} |1\rangle\right) \otimes |\lambda_j\rangle \tag{24}$$

where C is a normalization constant.

3. Apply the inverse QPE to reset the clock register.

This results in the final state

$$\sum_{j=0}^{N-1} \beta_j \left(\sqrt{1 - \frac{C^2}{\tilde{\lambda}_j^2}}\, |0\rangle + \frac{C}{\tilde{\lambda}_j} |1\rangle\right) \otimes |0\rangle^{\otimes n_c} \otimes |j\rangle \tag{25}$$

Measuring the ancilla qubit as 1 collapses this state to

$$\frac{C}{\|\tilde{u}\|}\sum_{j=0}^{N-1}\frac{\beta_j}{\tilde{\lambda}_j}|j\rangle = |\tilde{u}\rangle \tag{26}$$

which matches the desired state from equation (21).

The main runtime of this circuit is determined by the complexity of one query to QRAM and number of those queries in steps 1 and 3. The QRAM query complexity can in general be estimated as $\mathcal{O}(\text{polylog}(N))$. The term $N = (M-1)^d$ is the size of $|\tilde{u}\rangle$ and for the purpose of runtime analysis can be assumed to scale in $\mathcal{O}\Big((1/\epsilon)^d\Big)$. The query complexity is therefore only linear in d. The number of those queries depends on the sparsity s (we will assume $s \in \mathcal{O}(1)$) and the condition number $\kappa = N^{2/d} \in \mathcal{O}(1/\epsilon)$ of L [5, 6]. Over time the number of queries has reduced as shown in Table 1 resulting in a state-of-the-art query complexity of $\mathcal{O}(1/\epsilon\ \log(1/\epsilon))$.

Year	Reference	Number of queries
2008	Harrow, Hassidim and Lloyd [17]	$\mathcal{O}(\kappa^2/\epsilon)$
2012	Ambainis [2]	$\mathcal{O}\Big(\kappa((\log\kappa)/\epsilon)^3\Big)$
2019	An and Lin [3]	$\mathcal{O}(\kappa\,\text{polylog}(1/\epsilon))$
2019	Lin and Tong [23]	$\mathcal{O}(\kappa\log(\kappa/\epsilon))$
2022	Costa et al. [7]	$\mathcal{O}(\kappa\log(1/\epsilon))$

Table 1: QLSS query complexity over time [7] assuming sparsity $s \in \mathcal{O}(1)$

For now there seems to appear no curse of dimensionality. However, most classical PDE problems require a classical representation of the solution, which requires some readout of $|\tilde{u}\rangle$. If only a specific property is required for which amplitude amplification can be employed, then the readout can be realized in $\Omega(1/\epsilon)$. Otherwise, if the full state is needed, then using quantum tomography a factor of N is introduced, resulting in a readout of $\mathcal{O}(N/\epsilon)$ [4, 15, 8] and discarding any advantage gained from QLSS.

The space complexity as well as comparisons to classical methods will be discussed further in section 5.

4.3 Special case for Poisson's equation

As proven in [9] it is possible to rewrite A in the d-dimensional case as

$$L = \underbrace{B \otimes I \otimes \cdots \otimes I}_{d \text{ times}} + I \otimes B \otimes I \otimes \cdots \otimes I \; + \; \cdots \; + \; I \otimes \cdots \otimes I \otimes B \tag{27}$$

$$= \underbrace{B \oplus \cdots \oplus B}_{d \text{ times}} \tag{28}$$

$$B = D + 2I = \begin{pmatrix} -2 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & -2 \end{pmatrix} \tag{29}$$

where $\oplus$ denotes the Kronecker sum[2] and $\otimes$ the Kronecker product. From this follows that L may be written in terms of matrix exponentials

$$e^{i\alpha L} = e^{i\alpha B} \otimes \cdots \otimes e^{i\alpha B} \tag{30}$$

where α is some abitrary factor [26]. This allows for the use of Hamiltonian simulation to directly compute the U_L required in the QPE step of QLSS [6]. By skipping QRAM in this manner we will observe a reduction in the total number of qubits necessary.

5 Summary of costs

To outline the costs of solving Poisson's equation using quantum algorithms, we will compare the time and space complexities with common classical methods. Both depend heavily on the size after discretization $N = (M-1)^d \in \mathcal{O}\Big((1/\epsilon)^d\Big)$ which in turn depends exponentially on the number of dimensions d as well as the chosen discretization fidelity which is related to the error ϵ. It is therefore important for solving methods to minimize the factor of N as much as possible.

Table 2 and Table 3 compare the time and space complexities with three classical methods. The first represents general but naive methods for solving systems of linear equations such as Cholesky decomposition. The second and third are the best known methods for solving Poisson's equation. While direct methods are typically more accurate, iterative methods tend to be faster. But both achieve minimal costs possible for classical methods [9]. It should be noted that in comparison to the quantum methods, the classical methods are often parallelized on modern computers, which can drastically improve the stated runtimes [9]. Nevertheless, the costs remain exponential in d.

Now considering the quantum methods, the general QLSS method is able to solve PDEs exponentially faster (only linear in d) than classical methods if readout of the solution is ignored. But for most PDE problems the full solution is required,

[2] $A \oplus B \coloneqq A \otimes I_b + I_a \otimes B$

Method		Time
Classical [9]	General (e.g. Cholesky decomp.)	$\mathcal{O}(N^3)$
	Best direct (FFT, Block cyclic reduction)	$\mathcal{O}(N \log N)$
	Best iterative (Multigrid)	$\mathcal{O}(N)$ or* $\mathcal{O}(N \log N)$
Quantum	General QLSS [8]	$\mathcal{O}\Big(\text{polylog}(N) \quad sN^{\frac{2}{d}} \log\Big(\frac{1}{\epsilon}\Big) \quad \frac{1}{\epsilon}N^{**}\Big)$
	– for Poisson's eq. with QRAM	$\mathcal{O}\Big(\text{poly}(d)\text{polylog}\Big(\frac{1}{\epsilon}\Big)\frac{1}{\epsilon} \quad \frac{1}{\epsilon}N^{**}\Big)$
	– Special case [6]	$\mathcal{O}\Big(\Big(d\log^3\Big(\frac{1}{\epsilon}\Big) + \log^4\Big(\frac{1}{\epsilon}\Big)\Big) \quad \frac{1}{\epsilon}N^{**}\Big)$

Table 2: Time costs of solving Poisson's equation in terms of d dimensions, ϵ error and $N = \mathcal{O}\Big(\epsilon^{-d}\Big)$. *with comparable error to direct methods **factor for full readout

Method		Space
Classical [9]	General (e.g. Cholesky decomp.)	$\mathcal{O}(N^2)$
	Best direct (FFT, Block cyclic reduction)	$\mathcal{O}(N)$
	Best iterative(Multigrid)	$\mathcal{O}(N)$
Quantum	General QLSS (with QRAM) [8]	$\mathcal{O}(N^2)^*$
	– with Hamiltonian sim. [6]	$\mathcal{O}\Big(d\log^2\Big(\frac{1}{\epsilon}\Big) + \log^3\Big(\frac{1}{\epsilon}\Big)\Big)$

Table 3: Space costs of solving Poisson's equation in terms of d dimensions, ϵ error and $N = O(\epsilon^{-d})$. *for dense matrices, more efficient for sparse matrices

discarding any quantum advantage gained. This holds for all listed quantum methods, including the special case for Poisson's equation.

In terms of space complexity, the structure of the problem is crucial. For dense matrices in the general case, the space complexity of QLSS is $\mathcal{O}(N^2)$ due to the use of QRAM, which is worse than the best classical methods. But structures of sparse matrices can be exploited to reduce the space complexity [8]. As such, the special case for Poisson's equation is able to achieve a space complexity that is exponentially better than classical methods.

6 Current state of research

While we have discussed methods for solving Poisson's equation using quantum algorithms, it is important to note that the research still evolves rapidly and also includes broader problems than the ones presented here. This section will give a brief overview in this regard.

Nonlinear PDEs This paper has only looked at solving linear PDEs with quantum algorithms. However, many real world problems are more closely modelled by nonlinear PDEs, so we provide Figure 3 as a brief and non-exhaustive map of current approaches for solving these equations. Algorithms either represent them as linear PDEs [20, 19] or use domain specific methods to solve them. Another promising new approach which also applies to linear PDEs is to use time-marching methods to utilize Hamiltonian simulations [13, 22]. This is related to the special case for Poisson's equation discussed in subsection 4.3.

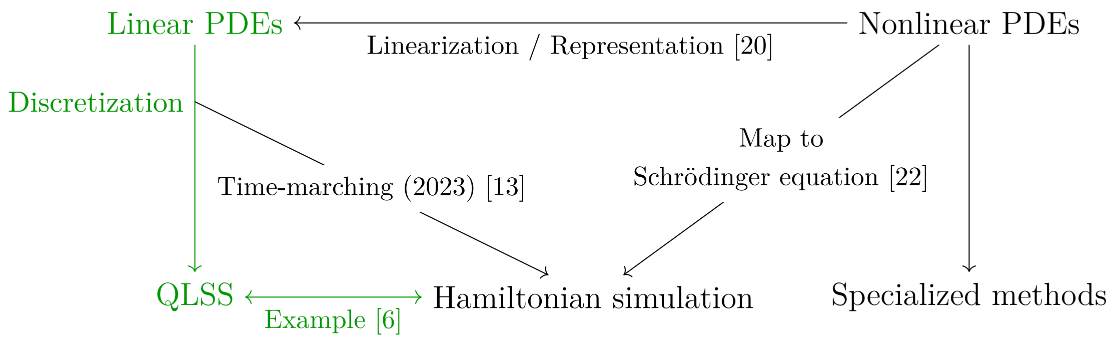


Figure 3: Map for solving nonlinear PDEs with quantum algorithms. Green indicates the parts covered in this paper.

Time and space costs Large time costs remain a significant challenge in the discussed algorithms. The quantum speedup often vanishes during the readout phase, which looses the overall speedup gained over classical algorithms. This issue arises from the one-to-one translation of classical differential equation problems, which require solutions to be present classically. To improve on this, future problems might be restated to require results to be present as quantum states, which could potentially mitigate this issue. Other promising approaches not based on QLSS as suggested by Fang, Lin and Tong [13] also show at least polynomial speedup.

In addition to time costs, memory costs for the general QLSS approach can also be substantial due to QRAM. However, these costs can be more efficient for problems with special structures, such as the one discussed in this paper or other sparse problems. For those, the curse of dimensionality (in space) can be broken. Alternatives to QRAM are also being explored, with Wang, McArdle and Berta [25] proposing the use of classical data structures and requiring only $\mathcal{O}(\log N)$ qubits.

High-dimensional PDEs The quantum algorithms show the most potential for solving high-dimensional PDEs, as their main advantage is to minimize the dimension dependency. In quantum chemistry for instance, some problems require one dimension per simulated particle, which can be resource intensive with classical methods [8, 16].

The Black-Scholes equation is another example, although more efficient Monte Carlo based methods exist for this particular equation [11].

Conclusion While there are still significant challenges to overcome, quantum algorithms for solving differential equation problems continue to show potential, particularly for problems with special structures and high-dimensional PDEs. While some argue that only polynomial speedup is possible for most practical applications [24], there is still much ongoing research in the field, with improvements being made at each step.

References

[1] Ravi P Agarwal, Simona Hodis and Donal O'Regan. *500 examples and problems of applied differential equations*. Springer, 2019.

[2] Andris Ambainis. 'Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations'. In: *arXiv preprint arXiv:1010.4458* (2010).

[3] Dong An and Lin Lin. 'Quantum linear system solver based on time-optimal adiabatic quantum computing and quantum approximate optimization algorithm'. In: *ACM Transactions on Quantum Computing* 3.2 (2022), pp. 1–28.

[4] Anurag Anshu and Srinivasan Arunachalam. 'A survey on the complexity of learning quantum states'. In: *Nature Reviews Physics* 6.1 (2024), pp. 59–69.

[5] Susanne C Brenner. *The mathematical theory of finite element methods*. Springer, 2008.

[6] Yudong Cao et al. 'Quantum algorithm and circuit design solving the Poisson equation'. In: *New Journal of Physics* 15.1 (2013), p. 013021.

[7] Pedro CS Costa et al. 'Optimal scaling quantum linear-systems solver via discrete adiabatic theorem'. In: *PRX quantum* 3.4 (2022), p. 040303.

[8] Alexander M Dalzell et al. 'Quantum algorithms: A survey of applications and end-to-end complexities'. In: *arXiv preprint arXiv:2310.03011* (2023).

[9] James W Demmel. *Applied numerical linear algebra*. SIAM, 1997.

[10] Byakatonda Denis. 'An overview of numerical and analytical methods for solving ordinary differential equations'. In: *arXiv preprint arXiv:2012.07558* (2020).

[11] Erik Ekedahl, Eric Hansander and Erik Lehto. 'Dimension reduction for the black-scholes equation'. In: *Department of Information Technology, Uppsala University* (2007).

[12] Lawrence C Evans. *Partial differential equations*. Vol. 19. American Mathematical Society, 2022.

[13] Di Fang, Lin Lin and Yu Tong. 'Time-marching based quantum solvers for time-dependent linear differential equations'. In: *Quantum* 7 (2023), p. 955.

[14] David Francis Griffiths and Desmond J Higham. *Numerical methods for ordinary differential equations: initial value problems*. Vol. 5. Springer, 2010.

[15] Jeongwan Haah et al. 'Sample-optimal tomography of quantum states'. In: *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*. 2016, pp. 913–925.

[16] Jiequn Han, Arnulf Jentzen and Weinan E. 'Solving high-dimensional partial differential equations using deep learning'. In: *Proceedings of the National Academy of Sciences* 115.34 (2018), pp. 8505–8510.

[17] Aram W Harrow, Avinatan Hassidim and Seth Lloyd. 'Quantum algorithm for linear systems of equations'. In: *Physical review letters* 103.15 (2009), p. 150502.

[18] James Higgins. *Differential Equations ODE Flow Chart.* Apr. 2016. URL: https://www.pdx.edu/learning-center/sites/learningcenter.web.wdt.pdx.edu/files/2020-07/Differential_Equations_ODE_Flow_Chart.pdf (visited on 16/07/2024).

[19] Shi Jin and Nana Liu. 'Quantum algorithms for computing observables of nonlinear partial differential equations'. In: *arXiv preprint arXiv:2202.07834* (2022).

[20] Shi Jin, Nana Liu and Yue Yu. 'Time complexity analysis of quantum algorithms via linear representations for nonlinear ordinary and partial differential equations'. In: *Journal of Computational Physics* 487 (2023), p. 112149.

[21] Nadav Levi. 'Numerical integration of ODEs with Automatic differentiation'. PhD thesis. 2021.

[22] Sarah K Leyton and Tobias J Osborne. 'A quantum algorithm to solve nonlinear differential equations'. In: *arXiv preprint arXiv:0812.4423* (2008).

[23] Lin Lin and Yu Tong. 'Optimal polynomial based quantum eigenstate filtering with application to solving quantum linear systems'. In: *Quantum* 4 (2020), p. 361.

[24] Ashley Montanaro and Sam Pallister. 'Quantum algorithms and the finite element method'. In: *Physical Review A* 93.3 (2016), p. 032324.

[25] Samson Wang, Sam McArdle and Mario Berta. 'Qubit-efficient randomized quantum algorithms for linear algebra'. In: *PRX Quantum* 5.2 (Apr. 2024), p. 020324.

[26] EW Weisstein. *Kronecker Sum. From MathworldA Wolfram Web Resource.* 2021.