Quantum Machine Learning and Quantum Protocols for Solving Differential Equations

Submitted by Annie Paine to the University of Exeter as a thesis for the degree of Doctor of Philosophy in Physics, February 2024.

Abstract

Quantum devices are being developed to perform computation in an inherently non-classical way. These devices are fundamentally different from conventional computers and have unique properties due to effects such as superposition and entanglement. At the same time, quantum devices are prone to noise, posing limitations on the depth of calculations and scaling. It remains an important challenge for quantum computing to offer advantage in solving of currently intractable industrial problems.

One rising area of quantum algorithms is quantum machine learning. Building off classical machine learning, quantum machine learning concerns the training of quantum models to learn and recognise relationships in data. Another class is variational quantum algorithms which utilise an optimisation loop to train a trial solution involving quantum evaluations to solve a given problem. These classes of algorithm have possibility of advantage for problems with large amounts of data or large search spaces due to the wide range of functions expressible and data encodable because of the exponential working space.

A possible area of application is differential equations. Differential equations govern many areas of industrial and research interest, from aerodynamics to finance to chemistry, yet many instances remain difficult to solve classically. Throughout my research I have considered solving differential equations with quantum machine learning and variational approaches.

In my thesis I describe four algorithms that I have developed for solving differential equations, each with different strengths and weaknesses, quantum resource requirements and areas of applications. Particular techniques utilised are quantum models representing functions, the parameter shift rule, kernel methods and quantile mechanics. Additionally, I (in collaboration) develop a technique to transform between computational and Chebyshev space. This technique is utilised for developing the algorithm for efficient encoding of physics-informed constraints into quantum models. I conclude this thesis with an outlook into the nascent area of quantum scientific machine learning.

Publications and Manuscripts

1. **A. E. Paine**, V. E. Elfving, and O. Kyriienko, **Physics-informed quantum machine learning: Solving nonlinear differential equations in latent spaces without costly grid evaluations**, Aug. 2023. arXiv:2308.01827 [quant-ph]

2. **A. E. Paine**, V. E. Elfving, and O. Kyriienko, **Quantum quantile mechanics: Solving stochastic differential equations for generating time-series**, Advanced Quantum Technologies, p. 2 300 065, Aug. 2023

3. C. Umeano, **A. E. Paine**, V. E. Elfving, and O. Kyriienko, **What can we learn from quantum convolutional neural networks?**, Aug. 2023. arXiv: 2308.16664 [quant-ph]

4. C. A. Williams*, **A. E. Paine***, H.-Y. Wu, V. E. Elfving, and O. Kyriienko, **Quantum Chebyshev transform: Mapping, embedding, learning and sampling distributions**, Jun. 2023. arXiv: 2306.17026 [quant-ph] (*co-first authors)

5. **A. E. Paine**, V. E. Elfving, and O. Kyriienko, **Quantum kernel methods for solving regression problems and differential equations**, Phys. Rev. A, vol. 107, p. 032 428, 3 Mar. 2023

6. O. Kyriienko, **A. E. Paine**, and V. E. Elfving, **Protocols for trainable and differentiable quantum generative modelling** Feb. 2022. arXiv: 2202.08253 [quant-ph]

7. O. Kyriienko, **A. E. Paine**, and V. E. Elfving, **Solving nonlinear differential equations with differentiable quantum circuits**, Phys. Rev. A, vol. 103, p. 052 416, 5 May 2021

Contents

List of Figures

Definitions

Acronyms

CDF – Cumulative distribution function	
DE – Differential equation	
DQC – Differentiable quantum circuits	
FDM – Finite difference method	
FTQC – Fault tolerant quantum computer	
HEA – Hardware efficient ansatz	
LCU – Linear combination of unitaries	
MMR – Mixed model regression	
NISQ – Noisy intermediate scale quantum	
OU – Ornstein-Uhlenbeck	
PDE – Partial differential equation	
PDF – Probability density function	
PINN – Physics informed neural network	
PSR – Parameter shift rules	
QChT – Quantum Chebyshev transform	
QCL – Quantum circuit learning	
QF – Quantile function	
QFT – Quantum Fourier transform	
QGAN – Quantum generative adversarial networks	
QKM – Quantum kernel methods	

QML – Quantum machine learning

QQM – Quantum quantile mechanics

SciML – Scientific machine learning

SDE – Stochastic differential equation

SVR – Support vector regression

Notation

$|\cdot\rangle$ – A quantum state

$|\cdot\rangle_N$ – An N-qubit state

$|\cdot\rangle\!\rangle$ – A classical linear sum of quantum states

$\hat{\cdot}$ – A unitary operator

$\{\cdot_j\}_j$ – A set of indeterminate size indexed by j

Chapter 1

Introduction

The development and advances in computing have profoundly changed the world. Our daily lives are now closely connected to the widespread use of computers, making tasks that were once considered impossible now achievable. The capability to quickly access vast amounts of information through mobile phones and the Internet, along with the exploration of the universe using space probes and particle accelerators, highlights the significant influence of computing. However, despite these advancements, some problems persist, often due to time and cost limitations stemming from inefficiencies in algorithms.

In recent times, there has been development of quantum devices capable of harnessing quantum effects like superposition and entanglement. These devices mark a fundamental departure from traditional computing approaches. While these current and near-term quantum devices have limitations such as scale and noise, ongoing progress is evident. Consequently, there is a growing interest in determining whether quantum computing can provide advantages and accelerate the resolution of currently challenging problems.

Research has demonstrated that the distinctive quantum properties inherent in quantum computers provide access to a novel set of efficient computational operations when compared to classical computing. Consequently, solving a particular problem may be efficient with quantum computing while inefficient using classical methods, for specific families of problems. Currently, there is active exploration in identifying problem types where quantum advantage can be leveraged, accompanied by the development of corresponding quantum algorithms. Promising areas include linear algebra (attributed to the linearity of quantum mechanics and the rapidly expanding state space), quantum chemistry (owing to the intrinsic quantum nature of the problem), and tasks involving substantial data quantities, such as machine learning (exploiting the potentially exponential model capacity).

A potential domain where quantum computing can offer a significant advantage is in the solution of differential equations (DEs). Differential equations find widespread application across various academic and industrial domains, posing computationally challenging problems in physics, fluid dynamics, geoscience, chemistry, and finance. The computational demands of differential equation solvers currently consume a substantial share of global high-performance computing resources. Many of these equations remain difficult, inefficient, and computationally expensive to solve, often due to properties like high degrees of nonlinearity, stiffness, and multidimensionality. Beyond classical inefficiencies, solving DEs exhibits features that underscore potential quantum advantages. These equations may necessitate the consideration of large datasets, especially in multidimensional problems, and there exists a subset of solvers leading to linear systems of equations.

Quantum algorithms can be sorted into classes based on defining features of their workflow. Throughout my work I focus on two such classes — variational quantum algorithms and quantum machine learning. In particular, I introduce and discuss algorithms which fall into both classes.

Variational quantum algorithms are characterised by their hybrid classical-quantum workflows, where a parametrised trial solution, incorporating quantum evaluations, is optimised based on a success metric associated with a specific problem. These algorithms find application across diverse tasks, including ground state energy problems, decision problems, and regression. Moreover, quantum variational algorithms often yield numerous evaluations of relatively shallow and easily implementable quantum circuits. Consequently, there is optimism that they may encompass valuable algorithms suitable for near-term quantum devices.

Quantum machine learning (QML) draws inspiration from the thriving field of machine learning, characterised by training models to discern and recognise relationships in data, subsequently generalising to previously unseen data. This approach has witnessed substantial success in domains like large language models, speech recognition, and image generation. In quantum machine learning, conventional machine learning is enhanced by incorporating quantum models for training. The underlying intuition is that quantum models, influenced by quantum elements (superposition, entanglement etc), can effectively represent diverse and more complex relationships, potentially providing benefits. Given that many quantum machine learning algorithms employ optimization for model training, these algorithms can straddle both variational quantum algorithms and quantum machine learning classes.

These classes of algorithm are suitable for problems with large volumes of data and offer a highly expressive solution space which are some of the limitation of current classical DE solvers. Therefore, throughout my thesis I focus on the development of variational quantum machine learning algorithms for solving differential equations.

1.1 Aims & Objectives

The overall aim of my studies throughout my PhD was to develop quantum algorithms for the solving of differential equations along with tools to help aid in this endeavour. As discussed in the introduction, the solving of differential equations is a promising application for useful quantum computation. Therefore my aim was to further explore and develop this field, consider new and different approaches and contribute a small step towards the goal of useful quantum computation.

One particular objective was to consider quantum machine learning and variational approaches. During the initial stages of my studies we found this to be an underdeveloped direction for the solving of differential equations. Whilst there was some initial work in the field [1], most focus was on approaches based on discretisation. Variational and quantum machine learning (QML) approaches promise flexibility, high expressivity, and the opportunity for near-term applications at the cost of lacking guarantees for quantum advantage.

Related to the first objective, another objective was to consider near-term suitable approaches. Quantum computers produced now and in the near future have substantial limitations. Therefore any algorithms hoped to be implemented in the near future must consider these limitations. Hybrid algorithms, such as many variational and QML approaches, allow the splitting of tasks between quantum and classical devices. This results in being able to reduce the demands on the quantum device and consequently can lead to near-term suitable algorithms. By finding applications for near-term devices the hope is to hasten the introduction of useful quantum computation.

Another objective was to specifically consider non-linear differential equations and how to deal with the non-linear components. Quantum computers, due to the nature of quantum mechanics, are fundamentally linear. However, the majority of DEs of interest are nonlinear. Therefore, the careful consideration of the introduction of nonlinearity is essential for my aim, the development of quantum DE solvers, to be as widely applicable as possible.

1.2 Thesis Overview

This thesis starts with a review of relevant background such as quantum computing, quantum models, variational quantum algorithms and differential equation solvers. Furthermore, I detail various techniques and subroutines that I will utilise in future chapters.

In the first chapter of my work (chapter 3) I develop an algorithm for the solving of nonlinear PDEs. A quantum model represents a trial solution which can be optimised to solve a given problem. In particular, how to handle boundary and initial conditions is considered. Additionally, how certain choices of the quantum model can result in different fitting functions. Results of using the algorithm are simulated for linear, nonlinear and systems of DEs, including an instance of the Navier-Stokes equations.

In chapter 4, I develop an algorithm to solve stochastic differential equations (SDEs) using quantile mechanics. By using quantile mechanics a given SDE can be converted to a PDE in terms of a quantile function which represents a probability distribution. A quantum model then represents a trial solution of the quantile function which is trained to solve the given SDE. Once trained it can be sampled to generate samples — a form of generative modelling. In particular, the relationship between quantile functions and the function resulting from the training of a well known generative modeling algorithm, QGAN, are explored. Results from the simulation of the algorithm for the Ornstein-Uhlenbeck process are shown.

In chapter 5 the use of quantum kernel methods for the solving of differential equations is considered. A quantum model is proposed in terms of quantum kernel functions with structure such that quantum evaluations are only required pre and post training. Two approaches are then explored to solve a DE — optimisation loop (MMR) and support vector regression (SVR). Results of simulating the algorithm for regression problems and an instance of the Duffing equation are shown.

In chapter 6, instead of developing an algorithm a tool for use in quantum algorithms is developed. This is the quantum Chebyshev transform and corresponding state encoding. This offers the ability to transform between the computational space and the Chebyshev space. Certain actions will be easier in each space so transforming to the appropriate space for each stage of the problem can help greatly. In this chapter the Chebyshev transform and encoding are demonstrated by learning a function in the Chebyshev basis and sampling in the computational. The Chebyshev transform is also utilised in the following chapter.

The last work presented in chapter 7 is another algorithm for nonlinear PDEs. For this algorithm a specific quantum model is chosen such that when solving the DE the independent variable of the DE is not explicitly present. As a result, evaluations do not need to occur at individual grid points during training. How to construct the necessary states for the DE is explained in general and is detailed in terms of Fourier encoding and Chebyshev encoding. Results are shown for various problems including a multidimensional problem.

Concluding remarks and outlook are finally given.

Chapter 2

Background

2.1 Introduction to Quantum Computing

Quantum computing is the use of quantum mechanical systems to perform information processing tasks. The overarching goal of quantum computing research and industry is to solve real-world applicable tasks with a significant speed-up over classical solutions. This requires two different yet related directions of research: the production of quantum computers and the development of the algorithms implementable on them. This goal is known as quantum supremacy or quantum advantage with (usually but not always) supremacy referring to any task and advantage to a useful task. There are many models of quantum computing such as adiabatic quantum computing [2, 3] and measurement based quantum computing [4, 5] all using different methods to implement operations and process quantum information. In my work I use the gate based model [6] and therefore now introduce this model.

2.1.1 Mathematical Framework

The Qubit

In classical computing the unit of information is the bit. It is a binary digit assigned the possible values of 0 and 1. In quantum computing there is the quantum bit known as the *qubit*. The qubit is a two state quantum mechanical system, one of the simplest quantum mechanical systems to express the quantum properties which will differ classical and quantum computing. The qubit resides in a Hilbert space of dimension two, $\mathcal{H}^2$. A basis of this space consists of two states. The default basis is usually the computational basis

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \tag{2.1}$$

with $|0\rangle$ referring to the ground state of the system and $|1\rangle$ the excited state. The labels used are similar to the labels used for the classical bit, 0 and 1. Here we use bra-ket notation, introduced by Paul Dirac, to denote a quantum state [7].

In bra-ket notation $|\cdot\rangle$ is a *ket* and refers to a vector in $\mathcal{H}$. $\langle\cdot|$ is a *bra* and refers to a member of the dual space $\mathcal{H}^*$. The transformation between bra and ket is the conjugate transpose, $|\psi\rangle^\dagger = \langle\psi|$, and vice versa.

The possible states of the qubit are the members of the Hilbert space. This is a vector space and therefore includes the linear sum of the basis states. This is known as a *superposition* — one of the unique properties of quantum versus classical. The sum must be normalised under the Euclidean norm — intuition for this will come later in the measurement section. Therefore, in terms of the computational basis an arbitrary one qubit state $|\psi\rangle$ can be written as

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \quad \alpha, \beta \in \mathbb{C}, \quad \text{s.t. } |\alpha|^2 + |\beta|^2 = 1. \tag{2.2}$$

Multiple Qubits, Mixed States and Entanglement

A one qubit system is very limited, therefore for the majority of algorithms more qubits are going to be needed. A system of N qubits spans a Hilbert space of dimension 2^N, $\mathcal{H}^{2^N}$, as each qubit spans $\mathcal{H}^2$ and a tensor product is taken. This exponential growth of space with respect to qubit number is another feature of quantum computing to utilise for future advantage. An orthonormal basis $\{|\psi_j\rangle\}_j$ of $\mathcal{H}^{2^N}$ can then be found to express an N-qubit system as a linear superposition of basis states — a generalisation of (2.2)

$$|\psi\rangle = \sum_{j=0}^{2^N-1} \alpha_j |\psi_j\rangle, \quad \alpha_j \in \mathbb{C}, \quad \text{s.t.} \sum_{j=0}^{2^N-1} |\alpha_j|^2 = 1. \tag{2.3}$$

So far the states we have considered are what is known as *pure* states. Classical uncertainty of the state results in *mixed* states. These mixed states are a combination of a set of pure states as

$$\rho = \sum_j p_j |\psi_j\rangle\langle\psi_j|, \tag{2.4}$$

where $|\psi_j\rangle$ are pure states and p_j are the probabilities of being the associated state. Therefore each p_j must be positive and $\sum_j p_j = 1$ making (2.4) a convex sum. These type of states generally result from either uncertainty in state preparation or when considering sub-systems.

An interesting question arises when one starts considering multiple qubit states of whether each qubit in the state can be described independently of the other qubits in the state. In other words, whether the multiple qubit state can be written as the tensor product of one qubit states,

$$|\psi\rangle_N \overset{?}{=} \bigotimes_{j=0}^{N} |\psi_j\rangle_1. \tag{2.5}$$

When this is impossible the multi-qubit state is referred to as an *entangled* state. Simply, this means that there is a correlation between the entangled qubits. Entanglement is something that can be utilised in quantum algorithms for speed ups over classical computing. The effect of entanglement can be seen clearly when considering measurements.

Measurement

A classical bit can always be measured for its state and the measurement does not affect the bit. Quantum measurement is different, in quantum mechanics observations affects and changes the system observed. To measure a state an observable O is chosen — this is a Hermitian operator. The eigenvectors of this observable $\{|\phi_j^{m_j}\rangle\}_j$ form an orthonormal basis of the Hilbert space. The associated eigenvalues m_j are the possible outcomes of the measurements.

When the measurement is taken one of the possible outcomes is observed. The state measured then changes to an associated eigenstate of the measurement outcome. This process is known as *collapse*. The probability of a particular outcome m occurring is equal to the square norm of the associated eigenstates amplitudes when the state is considered in terms of the measurement basis. A measurement of $|\psi\rangle$ with observable O resulting in outcome m can be written as

$$|\psi\rangle \rightarrow \frac{M_m|\psi\rangle}{\sqrt{p(m)}}, \quad p(m) = \langle\psi|M_m|\psi\rangle, \quad M_m = \sum_{j \text{ s.t } m_j = m} |\phi_j^{m_j}\rangle\langle\phi_j^{m_j}|, \tag{2.6}$$

where $p(m)$ is the probability of measuring m and therefore of collapsing to $\{|\phi_j^{m_j}\rangle\}_j$ s.t. $m_j = m$.

This measurement process gives us some intuition into the requirement for the norm of a state to be equal to one. The norm is equal to the sum of the probabilities of each measurement outcome which should equal one. We also note that measurement affecting the state means that the exact state of a system cannot be determined with a single measurement and it cannot be measured mid-computation without altering the computation. This is another property which separates classical and quantum computation.

Measurement is a behaviour that strongly differentiates classical from quantum computing. However, this difference sometimes presents itself as an issue to work around. Because measurement changes the state, information on the state can not be gathered and recorded mid computation, unless the state change is desired. Therefore, there is no access to the state throughout computation, just the final results. Additionally, each measurement only extracts partial information of the state. If information about the full state is desired, many measurements would be required, scaling with qubit number. This is known as the data output problem and is discussed further in section 2.5.4.

Operators & Gates

A quantum computer needs to be able to process and manipulate quantum states to be able to carry out algorithms. Classical computing uses logic gates such as AND or NOT to process the bits. Analogously, quantum computers use operators which are often referred to as quantum gates with quantum computing. In classical computing a gate maps bit strings to bit strings. For a quantum gate analogue, basis states are mapped onto superpositions of basis states. Furthermore, the input state can also be a superposition, in which case the gate acts linearly due to the fundamentally linear nature of quantum mechanics.

$$\hat{U} : |i\rangle \rightarrow \sum_{j=0}^{2^N-1} c_{ji}|j\rangle, \tag{2.7}$$

$$\hat{U} : \sum_{i=0}^{2^N-1} \alpha_i|i\rangle \rightarrow \sum_{i,j=0}^{2^N-1} c_{ji}\alpha_i|j\rangle. \tag{2.8}$$

Therefore, an N-qubit quantum gate can always be represented by a matrix of dimension equal to $2^N \times 2^N$. However not all matrices represent valid quantum gates as quantum states need to be normalised before and after they are acted on. Therefore the quantum gate must preserve lengths and so it must be unitary, $\hat{U}^\dagger = \hat{U}^{-1}$. This restriction means that quantum gates are reversible. This can be shown to be the only restriction on the gates and therefore all unitary $2^N \times 2^N$ matrices represent valid N-qubit gates [6].

2.1.2 Quantum Circuits

A quantum circuit carries out information processing and is made up of wires representing qubits connecting gates via their inputs and outputs. Similarly to classical computing when drawing the circuits wires are expressed as lines and

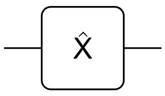
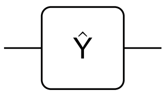
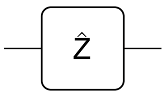
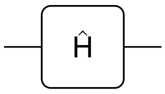
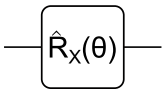
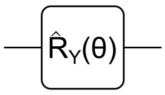
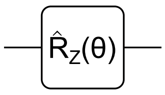
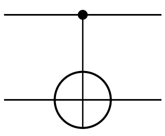
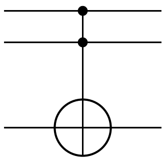
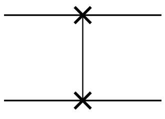
Name	Symbol	Matrix Representation
X		$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$
Y		$\begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$
Z		$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$
H		$\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$
$R_X(\theta)$		$\begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$
$R_Y(\theta)$		$\begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$
$R_Z(\theta)$		$\begin{pmatrix} \exp\left(-i\frac{\theta}{2}\right) & 0 \\ 0 & \exp\left(i\frac{\theta}{2}\right) \end{pmatrix}$
CNOT		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$
Toffoli		$\begin{pmatrix} \hat{I}_{6\times 6} & 0_{6\times 2} \\ 0_{2\times 6} & \hat{X} \end{pmatrix}$
Swap		$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Figure 2.1: **Table of common quantum gates**. Name, symbol and matrix representation given for a set of common quantum gates.

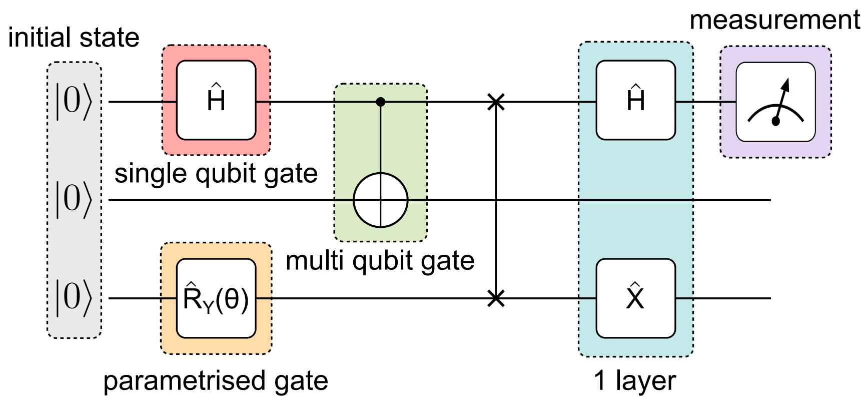


Figure 2.2: **Quantum circuit example.** The circuit begins with the preparation of an initial state — in this case all $|0\rangle$. This state is then acted upon by a mix of single and multi qubit gates, some of which are parametrised. Finally, measurement on one qubit occurs. A layer, referring to a set of gates over multiple qubits implemented at the same step of circuit implementation, is highlighted.

gates as blocks with time flowing from the left to the right. The set of gates implemented over the register of qubits at a given time or step in computation is often referred to as a layer.

Whilst all unitary matrices are valid gates, it is not arbitrarily simple to implement all unitary matrices with physical devices. Therefore algorithms are often decomposed and written in terms of a common set of gates. A non-exhaustive list of these are shown in figure 2.1. The Pauli gates X, Y, Z and rotations in these basis are used to alter states. The Hadamard gate creates superpositions, the controlled NOT and Toffoli gates introduce entanglement, and the SWAP gate swaps the state of two qubits.

A quantum circuit generally consists of a main register containing the input, but often also contains multiple *ancillary* qubits. These qubits are additional qubits used during computation to aid the implementation of certain operations but not for input or output. An example use of ancillas is the implementation of operators which are unitary over the whole space but non-unitary if considering main register alone. An example circuit is shown in figure 2.2.

2.1.3 Quantum Computers

There are multiple different ways to prepare a two level quantum system to represent a qubit and correspondingly there are multiple different types of quantum devices. Example implementations include superconducting circuits [8], trapped ions [9] and neutral atoms [10]. These different quantum devices all have different advantages and disadvantages, making some more useful for certain types of algorithms than others. It is still an open question whether in the future one design will “win” and become the standard architecture or if there will be multiple architectures with different tasks they are best suited to.

One uniting feature of all types of quantum computers is that they are naturally noisy and prone to errors in computation with this noise coming from multiple sources. One source is entanglement between the quantum system and its environment leading to *decoherence*, where the quantum state is altered and information is lost to the environment. Another is gate noise, when the gate is not perfectly implemented. Additionally there is read-out noise. One source of read-out noise is that the full information of the state is not gained in a single measurement and therefore extracting information from the system is generally probabilistic. Additionally, each measurement has its own implicit noise.

For quantum computing to reach its full potential, quantum devices will be needed which have minimal error and can operate with a large number of qubits. This is known as fault-tolerant quantum computing (FTQC) [11]. Whilst quantum devices are currently being developed, these fault tolerant quantum devices are a long-term goal. Current and near term quantum devices remain error prone and are of limited size. Because of these limitations this era of quantum computation is known as noisy, intermediate scale quantum (NISQ) computing [12]. While many theorised quantum algorithms will require fault-tolerant quantum devices for implementation there has also been considerable effort in developing algorithms suitable for near-term quantum devices. It is hoped such algorithms would lead to earlier useful quantum computation.

2.1.4 Potential Use Cases

Quantum computing is not going to offer advantage compared to classical computing for all use cases. Instead applications have to be found with the opportunity for speed-up. Features of problems which can indicate possibility for quantum advantage include being naturally quantum or related to linear algebra or containing large amounts of data.

Considering naturally quantum problems often leads to operations of interest being efficiently implementable due to the same underlying framework of operation and computation. One example is quantum chemistry [13]. Applications include finding the ground states and excited states of systems, simulating the dynamics of molecules and determining other properties. These tasks can currently be hard to calculate depending on the system. Being able to perform these calculations in a time effective manner would give opportunities in many fields such as drug discovery and material design.

Due to the mathematical formulation of quantum mechanics being fundamentally linear and quantum computing natively performing large unitary matrix operations, quantum computing is suitable for linear algebra problems. For example, solving linear systems of equations and inverting matrices. Linear algebra is used in a variety of situations such as differential equations and data science. One of the most well-known quantum algorithms is for linear algebra, the HHL algorithm, named after its authors Harrow, Hassidim and Lloyd [14]. This algorithm solves linear system of equations with a theoretical exponential speed-up over classical algorithms for the same purpose.

Problems which contain large amounts of data are another potential area for quantum speed-up due to the exponential size of the Hilbert space as the number of qubits increase. Therefore, more information can be contained by a smaller number of bits/qubits as compared to the classical case. colorred This means for problems with large amounts of data, this may be able to be processed with fewer (perhaps even exponentially fewer) resources giving an opportunity for quantum speed-up. However, it is important to note that there may be significant overhead for utilising the whole exponential space. For example if an algorithm called for loading information into each amplitude an arbitrary quantum state may have to be prepared, a currently difficult, inefficient task. And then if information needs to be extracted from the full space, full state tomography would need to be applied with many, many measurements. Use cases include machine learning problems such as regression and classification as well as differential equations.

2.2 Variational Quantum Algorithms Overview

Algorithms can be split into classes based on a defining feature of their implementation. One such class is variational algorithms [15]. Variational algorithms are defined by having an optimisation loop which adjusts variational parameters with each iteration of the loop so that a goal is satisfied. Generally these are *hybrid* algorithms – classical and quantum computers being used together – where the quantum computer is used to define a trial solution which depends on variational parameters while the classical computer is in charge of the optimisation loop and update of variational parameters. A typical workflow is shown in figure 2.3.

Because of the split work allocation between classical and quantum computers, these algorithms tend to be more suitable for NISQ devices than many other algorithms. The quantum computer calculates for each epoch separately resulting in shallower circuits. Furthermore the workload is split between the devices such that steps which can be completed efficiently on the classical computer are done so. This leads to a lower workload for the quantum devices. It is important to be aware that this workload sharing does introduce overheads in the measurements and state preparation needed for the communication between classical

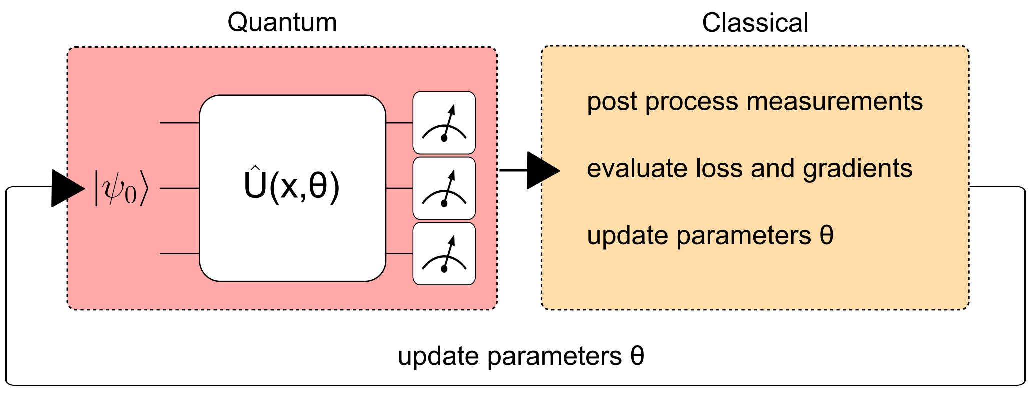


Figure 2.3: **Workflow of variational quantum algorithms.** During typical variational quantum algorithm workflow a set of quantum circuits are evaluated for a given set of parameters $\boldsymbol{\theta}_j$. These measurements are then passed to a classical computer to evaluate a loss function and any relevant derivatives. The loss and loss derivatives are in turn given to an optimiser which suggests new parameters $\boldsymbol{\theta}_{j+1}$. These parameters are passed to the quantum computer for the next set of evaluations. This iterates until an end condition is met.

and quantum devices which can affect (but does not prevent) the availability of quantum benefit.

To implement a variational algorithm first the representation of the trial solution is chosen. Then a measure of how well the trial solution with the current parameters satisfies the problem is needed. This is generally referred to as the loss function and is constructed such that the minimal value represents an optimal solution to a problem. The variational loop now occurs. For the current set of parameters the loss value and/or gradients (dependent on choice of optimiser) are calculated. These are fed to the classical optimiser which suggests new parameters. This is repeated until a stopping criterion is reached such as loss value or maximum number of iterations. The final parameters and the quantum model then give a suggested solution to the problem.

Variational algorithms are not deterministic, convergence to an optimal solution is not guaranteed. For them to be successful first the trial solution must actually be able to express the solution – known as *expressivity*. And then it needs to be possible to find the parameters leading to this solution – known as *trainability*. This is discussed later in section 2.5 and we will see how many hyperparameters affect both of these.

2.3 Classical & Quantum Machine Learning Overview

2.3.1 Classical Machine Learning

Machine learning has become a popular and important tool in computation, allowing a wide variety of tasks to be completed. Fundamentally, machine learning is the training of models to learn and recognise relationships in data [16]. Algorithms start with a trainable model, a set of inputs and a measure of success. This input and measure of success is used to train the model to learn what it can to achieve the given task. Once trained the model can be given more data and return information about it based on what was learnt from training. From classification [17] to generative modelling [18], regression [19], natural language processing [20] and more, machine learning has been applied and leveraged its ability to learn relationships in data to perform tasks intractable or inefficient by other methods.

Generally, machine learning algorithms are split into three broad categories: unsupervised, supervised and and reinforcement learning. These are three different learning paradigms differentiated by what input is given for training and how the model is ”motivated” to improve. Supervised learning begins with the input of sets of data labelled by the properties of interest. Based on these labels the algorithm learns how to map an input to the labels based on the input itself. Generally once trained the purpose is to label new given data. Unsupervised learning is trained on an input of given data but labels are not supplied. Therefore the algorithm is making connections and separations between data by itself and after training applies these to new unseen, data. The input of reinforcement learning is a task to complete and a measure of success. During training the algorithm suggests actions to take and it is then rewarded based on the success of its action. If training is successful the resulting model will be able to achieve

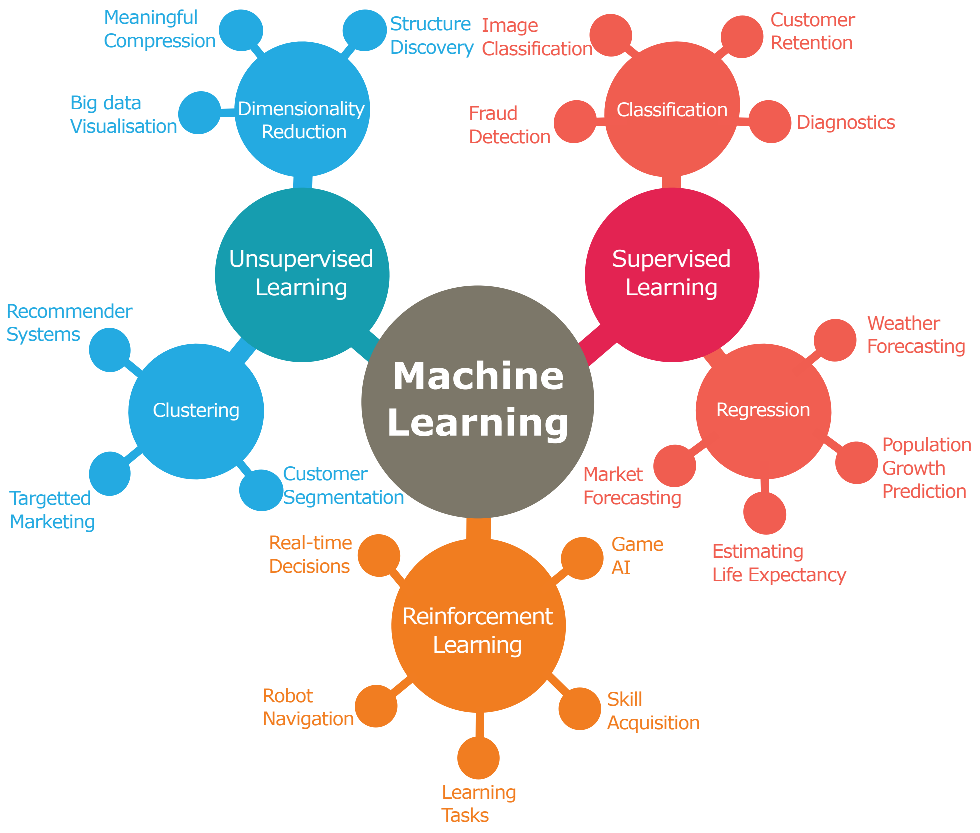


Figure 2.4: **Categories of machine learning and their use cases.** Visual representation of machine learning split into the three most common sub categories: supervised, unsupervised and reinforcement. These are further split into categories with examples of use cases. Figure reproduced from [21].

the desired task successfully. These groups and further sub-classifications and use-cases are shown in Fig. 2.4.

Machine learning algorithms can be grouped by other means as well. For example, what model is used [22] or training method (such as variational, iterative or other). Machine learning algorithms don't tend to be rigid or hard set, they are often very flexible. By changing any of the assumptions such as input, model, training method etc. algorithms can be altered and adapted to generate other approaches to suitable problems. Some difficulty of utilising machine learning comes from first how to choose all these options to improve training success (hyperparameter optimisation/tuning) and then how to be confident enough that the model resulting from training is correct (trust/model confidence). With the development of machine learning in recent times approaches and techniques for addressing these challenges are being developed and constantly improved [23, 24, 25, 26].

2.3.2 Quantum Machine Learning

Quantum machine learning builds upon machine learning by introducing quantum computation within the workflow. Often this is by considering quantum models – models which involve quantum circuit evaluation. Examples of quantum models, specifically to represent functions, are detailed in section 2.4. Currently, as quantum machine learning is still in its initial stages, much progress is in considering quantum analogues of existing classical machine learning algorithms, proposing quantum models and considering the advantages and disadvantages of quantum vs classical for various problems.

The advantages of utilising quantum models are currently expected to include the increased expressivity for a given number of computational resources as well as being able to efficiently represent some relationships that cannot be expressed efficiently classically due to differing behaviour. Another possible avenue for advantage is efficient sampling - as a quantum state effectively defines a probability distribution, each measurement can be understood as a sample from this distribution. This is generally utilised for generative problems and discussed further in sections 4 and 6. However, currently there are also disadvantages such as difficulties training (detailed in section 2.5), some of which even result from the increased expressivity which is hoped to bring advantage.

Proposed quantum machine learning algorithms include classification [27], regression [28] and generative modelling [29]. Some of particular interest for my work are detailed in section 2.6. Additionally, throughout my thesis quantum models representing functions are of use so these are now introduced.

2.4 Quantum Models of Functions

In this section quantum models for functions are introduced. These are functions with definitions including quantum computation which can be trained for a given task. Dependence on independent variable x is encoded either explicitly or implicitly. Generally dependence on a further set of variational parameters $\boldsymbol{\theta}$ is also included. These parameters alter the function and therefore the quantum model defines a family of functions and specifying $\boldsymbol{\theta}$ specifies the function.

2.4.1 Encoding

Dependence upon independent variable $\boldsymbol{x}$ can be encoded in different ways. Generally, all that is required is something in the model evaluation to change depending on the value of $\boldsymbol{x}$. I detail two common methods – *feature map encoding* [28] and *amplitude encoding* [1].

Feature map encoding gives an explicit dependence upon $\boldsymbol{x}$ and consists of including a circuit $\hat{\mathcal{U}}(\boldsymbol{x})$ in the quantum model definition. This circuit includes gates with parameters dependent on $\boldsymbol{x}$. Such a quantum model can be evaluated at any valid value of $\boldsymbol{x}$, leading to a continuous definition of the function. Furthermore, depending on the choice of $\hat{\mathcal{U}}(x)$, the derivatives can be calculated exactly with the *parameter shift rule* (discussed further in section 2.4.4). For certain problems, such as differential equations, this is very useful as it avoids an extra source of numerical imprecision from techniques such as finite difference methods. Most quantum models of this form need to be re-evaluated for each considered $\boldsymbol{x}$ value.

With amplitude encoding the function is defined such that the values at a set of points $\{x_j\}_j$ is equal to a set of measurements of a state.

$$|f\rangle = (f_0, f_1, ..f_{2^N-1}), \quad f(x_j) = |\langle j|f\rangle|^2 = |f_j|^2. \tag{2.9}$$

Variations on this are possible such as overlap instead of measurement or the inclusion of classical post processing. This leads to a function defined at a discrete set of points and is only accessible directly at these points. Interpolation could be used for intermediate values. It also means that 2^N values are stored by an N qubit state and full use of the exponential size of the Hilbert space is made. For derivatives usually some type of finite difference method is used. This can be computed classically after measuring the amplitudes for function values. Alternatively, it can be processed whilst still quantum such as in [1]. Multiple copies of the state are prepared. Some of these states then have their amplitudes cycled via an adder circuit. These states are then added and subtracted as required by the finite difference rule via controlled operations and projective measurements to account for the non-unitary nature of the operation. This prepares a state with the derivatives as amplitudes up to a normalisation factor.

2.4.2 Ansatz

Often the dependence on the variational parameters $\boldsymbol{\theta}$ within the quantum model is encoded via a circuit $\hat{\mathcal{V}}_\theta$ known as a *variational ansatz*. This is a circuit including gates with parameters dependent on $\boldsymbol{\theta}$. There are many different structures of such ansatz.

One category is *hardware efficient ansatz* (HEA) which utilise parameterised and entangling gates easily implemented by the chosen quantum hardware to maximise the entanglement and variation of the circuit whilst minimising quantum resources [30]. It is usually formed of alternating rotational layers and entanglement layers. The rotational layers consist of single qubit gates rotating with angle defined by a variational parameter. Each gate could have unique parameters or a parameter could be used for multiple gates. The entanglement layers consist of multi-qubit gates which entangle qubits and does not depend on any variational parameters. Usually the chosen entanglement operator is a two qubit gate which acts on nearest neighbours because this is often the simplest type of entangle-

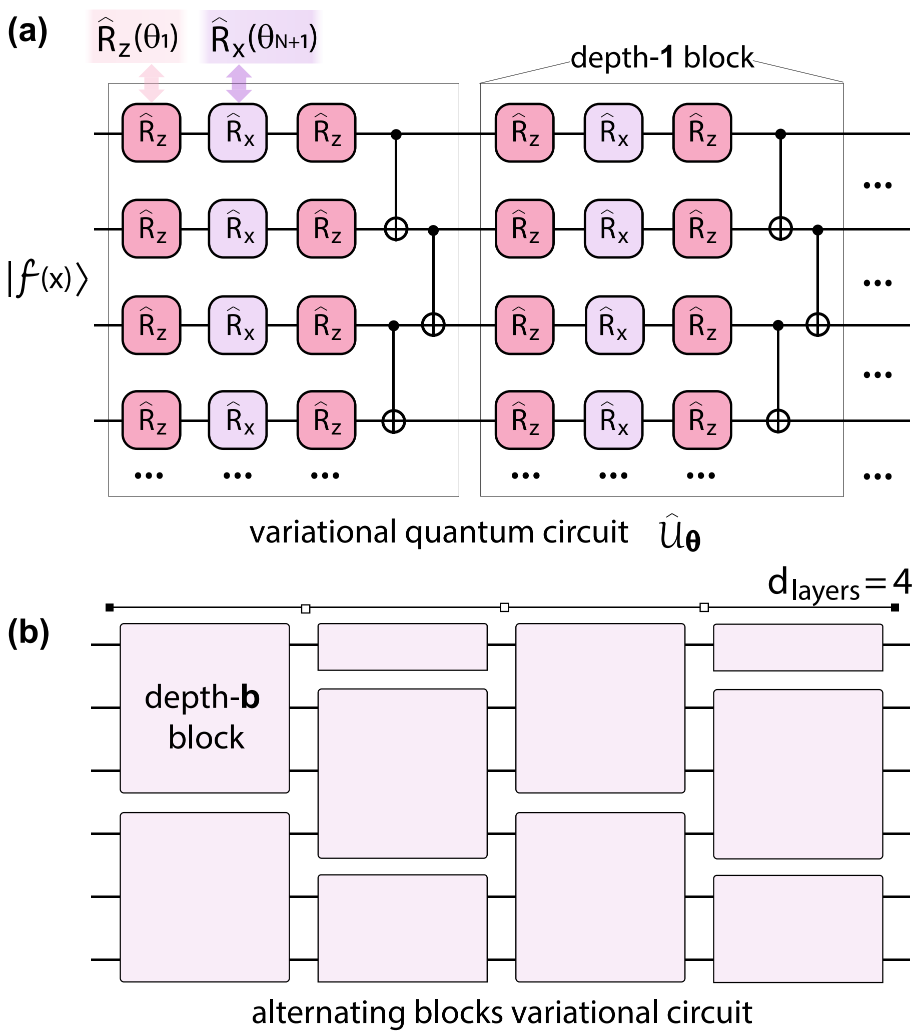


Figure 2.5: **Variational quantum circuits.** (a) A variational ansatz in the hardware-efficient form. It consists of a parametrised rotation layer forming $\hat{R}_z$-$\hat{R}_x$-$\hat{R}_z$ pattern, such that an arbitrary single qubit rotation can be implemented. Variational angles $\boldsymbol{\theta}$ are set for each rotation individually. The rotation layer is then followed by an entangling layer chosen as CNOT operations between nearest neighbours. The blocks of "rotations-plus-entangler" are repeated d times to form the full variational circuit $\hat{\mathcal{U}}_{\boldsymbol{\theta}}$. (b) Alternating blocks ansatz. The variational circuit consists of blocks of width N_b qubits ($N_b/2$ for boundary qubits). Blocks are chosen in the hardware-efficient form shown in (a) with depth of b. The blocks are placed in a checkerboard pattern, and repeated d_{layers} times. The goal is to entangle qubits locally, while avoiding global entangling operations that can result in vanishing gradients during $\boldsymbol{\theta}$ optimisation.

ment operator to implement. An example is shown in Fig. 2.5(a).

A second option is to use the *alternating blocks ansatz*, where instead of global entangling layers separate subblocks are used, interleaved into a checkerboard form [Fig. 2.5(b)]. Each subblock has a hardware efficient form shown in

Fig. 2.5(a) for the specified depth b. The motivation behind this structure is to entangle qubits locally first, and gradually form a correlated state by interleaving subblocks [31].

Another option is using symmetry encoding variational ansatz [32]. This is only possible if a symmetry of the problem is known. Such an ansatz is formed of gates which keep a state in the same symmetry sector no matter the value of the variational parameters. For example, for a chemistry problem where it is known the solution will have a specific number of excited states an ansatz could be found where if the initial state has a number of excited states, post variational ansatz it has the same number. Alternatively, a symmetry could be known that the solution space is degenerate i.e. there is a rule describing equivalent situations. Then an ansatz could be found which is “aware” that they are degenerate and therefore avoids unnecessarily exploring the whole degenerate space. This is known as geometric ansatze [33]. By utilising symmetries the search space is decreased and potentially the problem dimension reduced, often leading to easier training. However, if the space is restricted too far it can lead to problem becoming classically efficient and any possible quantum advantage removed so one must be careful [34].

2.4.3 Structure & Cost Functions

Piecing together the quantum model, how to evaluate the quantum model needs to be defined. Amplitude encoded models are generally evaluated as in (2.9), with $|f\rangle$ dependant on $\boldsymbol{\theta}$ such as $|f\rangle = \hat{\mathcal{V}}_\theta|0\rangle$. For explicit models with feature map embedding this is often an expectation value such as

$$f(x) = \langle\psi_0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{V}}_\theta^\dagger C\hat{\mathcal{V}}_\theta\hat{\mathcal{U}}(x)|\psi_0\rangle. \tag{2.10}$$

The operator C is Hermitian or a sum of Hermitian operators known as the cost function. It is the basis for the expectation value to be measured. The choice of cost function affects what information contained by the state is prioritised. Cost

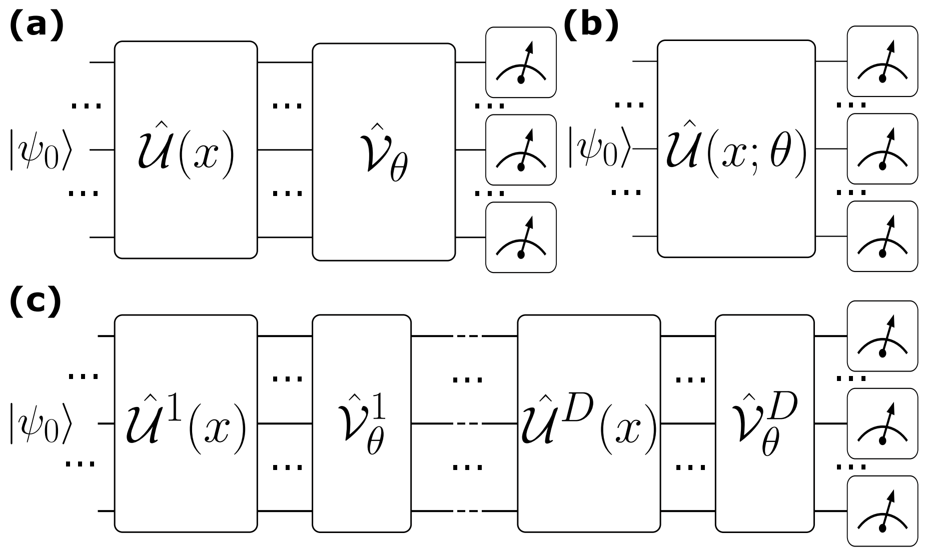


Figure 2.6: **Example Quantum Model Structures.** (a) Circuit for quantum model described in (2.10). Formed of a feature map $\hat{\mathcal{U}}(x)$ and variational ansatz $\hat{\mathcal{V}}_\theta$ applied to an initial state $|\psi_0\rangle$. Measurement is then taken for expectation values. (b) Quantum model without distinct feature map and variational ansatz, instead contributions from encoding variable x and variational parameters θ are intermingled. (c) Quantum model utilising data reuploading of D layers. x is encoded via feature map D times with variational ansatze present between each ”uploading” of x.

functions are split into local and global types, those where each measurement considers one “local” area and those which consider the whole “global” system with one measurement. Local cost operators were shown to have favorable behaviour as compared to global when training [35]. This quantum model is easily restricted to real functions by considering real coefficients for the sum of operators within the cost function. Depending on the problem considered this could be a useful restriction. Classical post-processing can be added such as additional classical only parameters for scaling and shifting. An example circuit for the implementation for this model is shown in Fig. 2.6 (a).

Alternatively the model could be an overlap such as

$$f(x) = \langle 0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{V}}_\theta|0\rangle. \tag{2.11}$$

Generally this is complex, though with suitable choice of feature map and ansatz restriction to a real only function can be implemented. To evaluate an overlap is not as native as measuring an expectation value but it can be done via different techniques such as modified Hadamard tests as described in section 2.7.1.

Many other variations exist. In particular variational ansatz and feature maps can be layered in a technique known as data re-uploading [36], this improves expressivity without increasing qubit number at the expense of depth. An example circuit for this is shown in Fig. 2.6(b). Additionally so far we have treated the dependence on x and θ as separate circuits which follow one another however they can be mixed. An example circuit for this is shown in Fig. 2.6(c). These variations are still being developed and which are optimal is an open question.

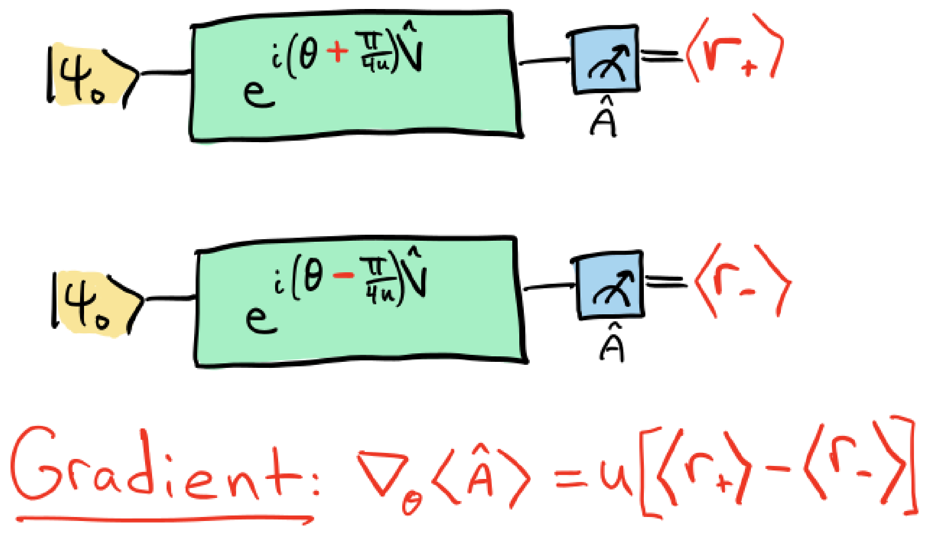
2.4.4 Derivatives

An important property to be able to calculate for a quantum model is its derivatives. These derivatives are split into two classes — feature parameter derivatives and variational parameter derivatives. Derivatives with respect to variational parameters would be required for any variational method where the optimiser uses a form of gradient descent. Whether derivatives with respect to the feature parameter is needed depends on the problem itself. As I consider and develop differential equation solvers this derivative technique will be very important.

There exists many methods for calculating derivatives. One way, which is classical in nature, is finite difference methods (FDM) [37]. These methods give a rule to approximate derivatives in terms of evaluations of the functions with shifted parameter values. One of the simplest and most well known of these methods is the Euler method which states

$$f'(x) \equiv \frac{f(x+h) - f(x)}{h}, \tag{2.12}$$

where $h > 0$ is a chosen value. Smaller h tends to give more accurate approximations up to a point and then issues may be encountered such as floating point accuracy. Other methods in these families differ by number of evaluation points,



$|\psi_0\rangle$
$e^{i(\theta + \frac{\pi}{4u})\hat{V}}$
$\hat{A}$
$\langle r_+ \rangle$
$|\psi_0\rangle$
$e^{i(\theta - \frac{\pi}{4u})\hat{V}}$
$\hat{A}$
$\langle r_- \rangle$
Gradient: $\nabla_\theta \langle \hat{A} \rangle = u[\langle r_+ \rangle - \langle r_- \rangle]$

Figure 2.7: **Parameter shift rule.** Visualisation of using parameter shift rule in (2.14). Two measurements are taken with parameter of interest shifted $\pm\frac{\pi}{4u}$. The difference is then taken with scaling of u to evaluate gradient. Figure from [38].

amount the parameter is shifted by and coefficients to each evaluation. Furthermore rules for higher order derivatives can be found either directly or using successive iterations of a first order derivative rule. These methods do result in a numeric error, with bounds generally able to be found in terms of step sizes h and assumed bounds on analytic higher order derivatives.

Implementing these FDM for quantum models can be done by evaluating each term of the rule separately and then classically post processing. This is common for when the embedding used is a feature map. When the encoding is via amplitude encoding an alternative method also exists. Multiple copies of the state are prepared. The amplitudes of some of these states are then shifted as asked for by the FDM being used. These are then added and subtracted according to the FDM rule with an operator which ends with a projective measurement to enable this non-unitary operation. This will then prepare a state with each amplitude equal to the approximated derivative at that point (possibly with renormalisation). Note that for this, as access to the function is only at a discrete set of points, a rule has to be used which only requires access to function values at those set of points. An example of this is in [1].

There also exists a quantum specific method known as the *parameter shift rule*. This rule is a method for calculating the derivatives of a quantum circuit with respect to a parameter of a gate [39, 28]. Say we have a quantum model

$$f = \langle 0|\hat{\mathcal{U}}^\dagger(\mu) C \hat{\mathcal{U}}(\mu)|0\rangle, \quad \mathcal{U}(\mu) = \prod_j U_j(\mu_j), \tag{2.13}$$

and we want to calculate $\partial_{\mu_k} f$. If the gate U_k of the parameter being differentiated μ_k is generated by a Hermitian operator with two unique eigenvalues then the parameter shift rule states that

$$\frac{\partial f}{\partial \mu_k} = u\left(f\left(\mu_k + \frac{\pi}{4u}\right) - f\left(\mu_k - \frac{\pi}{4u}\right)\right), \tag{2.14}$$

$$f\left(\mu_k \pm \frac{\pi}{4u}\right) = \langle 0|\hat{\mathcal{U}}_\pm^\dagger(\mu) C \hat{\mathcal{U}}_\pm(\mu)|0\rangle, \quad \hat{\mathcal{U}}_\pm(\mu) = \prod_j U_j\left(\mu_j \pm \delta_{j,k}\frac{\pi}{4u}\right), \tag{2.15}$$

where u is the value of the eigenvalues of the generator if they were shifted to $+u$ and $-u$. In particular, if the generator is a Pauli operator multiplied by 0.5, as usual for Pauli rotations, we get $u = 1/2$. A visualisation of this process is shown in Fig. 2.7.

Note that the parameter shift rule calculates the derivative with respect to the parameter of the gate. If the parameter is a function of a variable and the derivative with respect to that variable is wanted the chain rule should be used. Additionally if the derivative with respect to a variable which appears within the parameter of multiple gates is wanted the product rule should be used.

Though this rule looks similar to a FDM it is actually a mathematically exact representation of the derivative, therefore no numerical errors are introduced by this method. However, in practice we remember that the errors intrinsic to quantum computing such as gate noise and read-out error will still be present. Again this method can be applied iteratively for higher order derivatives. There also exists many variations of this rule commonly referred to as generalised parameter shift rules [40, 41].

2.4.5 Quantum Kernel Function

There exists a particular type of quantum model known as a quantum kernel function (QK). These are based on classical kernel functions which are conjugate-symmetric, positive definite functions. The definition of positive definite for complex functions $f(x, y)$ used is

$$\text{For any n real numbers } x_1, x_2, ...x_n \text{ the n by n matrix A with components} \tag{2.16}$$

$$a_{ij} = f(x_i, x_j) \text{ is a positive definite matrix.} \tag{2.17}$$

Here a positive definite matrix M is one such that $z^T Mz$ is greater than zero for all non-zero column vectors z.

These kernel functions are a type of similarity function often used for regression and classification tasks [42]. Due to some unique properties, they are utilised in various algorithms known as kernel methods. Kernel functions and their methods are further introduced in section 2.8.2.

Recently quantum kernel functions have been considered [43, 27]. A quantum kernel function is simply a quantum model which fulfils the requirements of a kernel function.

It can be shown that any quantum models of the form

$$\kappa(x, y) = \langle\psi(x)|\psi(y)\rangle \text{ or } \kappa(x, y) = |\langle\psi(x)|\psi(y)\rangle|^2 \tag{2.18}$$

fulfils the requirement of a quantum kernel function [43] where $|\psi(y)\rangle$ is a y dependent state prepared with an encoding method. This is not the only possible form of quantum kernel with various other structures being proposed [44].

A quantum model for a kernel method typically consists of sums of evaluations of a kernel at different points e.g.

$$f(x) = \sum_j \alpha_j \kappa(x, y_j), \tag{2.19}$$

with $\{\alpha_j\}_j$ being classical variational parameters. Variational parameters can be included as quantum parameters in the preparation of $|\psi(y)\rangle$. Alternatively, variational parameters could remain entirely classical. We will make use of quantum kernels for the solving of differential equations in chapter 5.

2.5 Training (and other) Behaviours

Variational algorithms are not deterministic and do not guarantee convergence to an optimal solution. The training behaviour of the algorithms need to be considered and how that affects the likelihood of receiving the optimal solution. Two important factors are the expressivity and the trainability of the model/algorithm.

The expressivity of a quantum model is the range of functions that the quantum model can represent as the parameters vary. To have any chance of converging to an optimal solutions the quantum model being trained must be able to represent this solution. If no prior knowledge of the solution is known this is usually achieved by trying to create a highly expressive model that can represent a wide range of trial solutions. If there does exist prior knowledge, a quantum model which exhibits this known behaviour can be prepared. This can increase the range of likely solutions expressible for given resources as space is not ”wasted” on expressing functions which cannot possibly be optimal.

Expressivity is solely affected by the quantum model/trial function. Therefore anything that alters the quantum model affects the expressivity. This includes feature map, ansatz, cost operator and any classical pre/post-processing. Expressivity of a given model can be analysed by analytically expanding the given model as a series [45]. E.g. consider the model $f_\theta(x) = \langle\psi(x)|\hat{V}_\theta^\dagger C\hat{V}_\theta|\psi(x)\rangle$. We can then analytically expand $|\psi(x)\rangle$ to consider each amplitude as a function of x

$$|\psi(x)\rangle = \begin{pmatrix} \psi_0(x) \\ \psi_1(x) \\ \ldots \\ \psi_{2^N-1}(x) \end{pmatrix}. \tag{2.20}$$

Thus the choice of encoding can be considered as choosing a set of basis functions. Application of the ansatz then linearly mixes these functions and supplies coefficients

$$\hat{V}_\theta|\psi(x)\rangle = \begin{pmatrix} \sum_j c_j^0(\theta)\psi_j(x) \\ \sum_j c_j^1(\theta)\psi_j(x) \\ \ldots \\ \sum_j c^{2^N-1}(\theta)_j\psi_j(x). \end{pmatrix} \tag{2.21}$$

Finally, the measurement leads to multiplication and summation of terms. The final expression of the model is

$$f_\theta(x) = \sum_{i,j} \tilde{c}_{i,j}(\theta)\psi_i^\dagger(x)\psi_j(x). \tag{2.22}$$

This can then be analysed for more information on expressivity.

Trainability relates to the likelihood of converging to the optimal solution. Even if the set of parameters which represent the problem solution exist, the optimiser could fail to identify them. Trainability is decided by what's known as the loss landscape – how the loss changes with parameters – and the optimiser which explores this landscape. Many hyperparameters affect trainability, from the loss function to the model to the optimiser.

An important consideration is how the expressivity of the quantum model can affect trainability. Whilst the discussion on expressivity may have lead to the

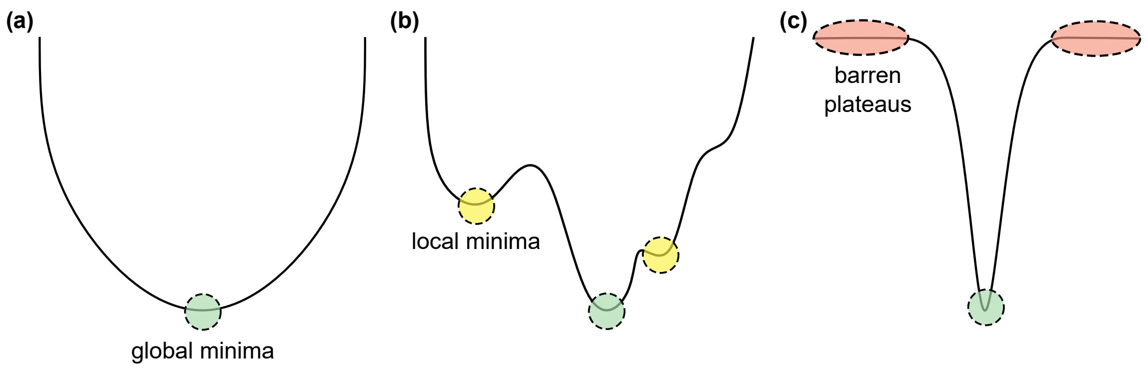


Figure 2.8: **Loss landscape examples with features of interest.** **(a)** A loss landscape with every point decreasing to global minima. This is an example of a convex landscape. **(b)** A loss landscape which contains points with gradient moving away from global minima. Local minima are present. These are points with minimal value in a neighbourhood but not over the whole space. **(c)** A loss landscape which includes areas of exponentially small gradients known as barren plateaus.

thought that the higher the expressivity the better this is not necessarily the case. Whilst higher expressivity means more functions are expressible and this could be interpreted as leading to a higher chance of the optimal solution being expressible, it can also have a negative impact on trainability [46, 47]. This gives increased motive for utilising any prior solution knowledge to design problem specific models with controlled expressivity which are trainable. However, if this is taken too far the resulting model could be efficiently classically simulable [34, 48], removing most chances of quantum advantage. Therefore, a careful balance of expressivity and trainability needs to be found and exactly where this balance is and how to find it remains an open area of research.

2.5.1 Loss Landscape

As mentioned during the discussion of trainability, the main contributing factor is the loss landscape – the change of the landscape as the parameters change. When plotting the loss values with two parameters it can be seen why the term ”landscape” is chosen. As shown in Fig. 2.8 as the parameters vary and the loss increases/decreases features such as peaks, valleys and plateaus appear.

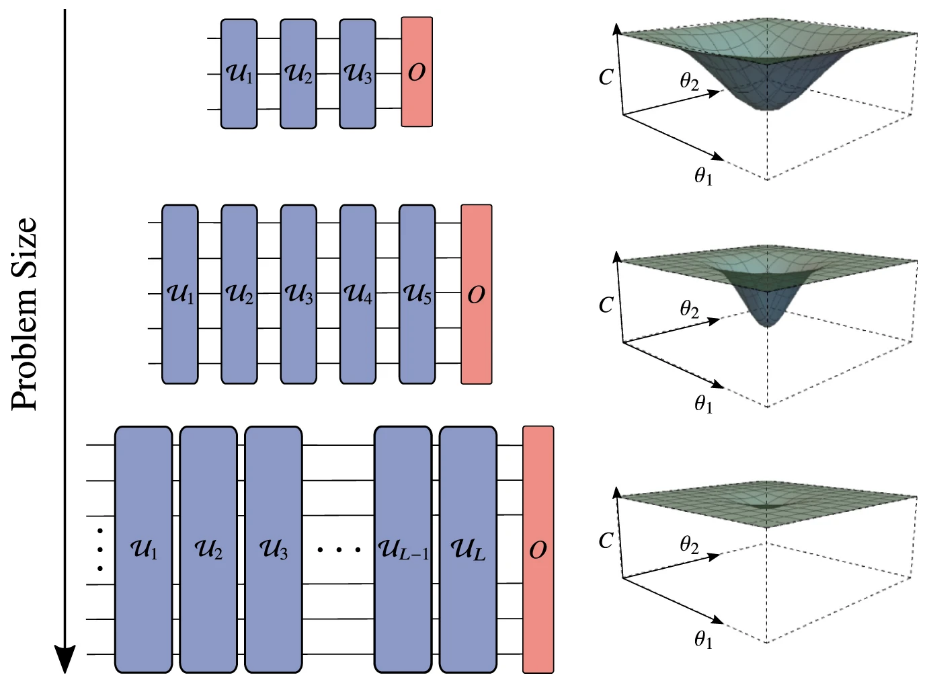


Figure 2.9: **Emergence of barren plateaus with system size.** An example of a property which influences the formation of barren plateaus is system size. As system size increases gradients decrease and barren plateaus form in loss landscape. In these regions training is near impossible as not enough information is available to navigate out of the plateau. Figure from [51]

The loss function is designed such that the global minima relates to the solution. However generally the loss landscape is not a smooth ”valley” with every point smoothly descending to the global minima. Instead there are many ”hills” and ”waves” leading to local minima – a point which is the minimum value for a neighbourhood of that point but is not the true minimum value. Optimisers can struggle to distinguish local minima from global minima and if the optimiser can't ”escape” the loss will stagnate there until training ends. In fact, it has been shown that variational quantum algorithms can lead to exponential local minima [47, 49]. Therefore, either reducing the number of local minima in the landscape or utilising optimisers effective at escaping local minima can improve training. Also, some local minima solutions will be close to the global minima and therefore depending on accuracy required by the problem, training resulting in a local minima is not necessarily a failure. Also, it's important to note that these exponential local minima affect digital variational methods but they are not intrinsic to all quantum optimisations. For example quantum annealing was originally introduced strongly motivated by the idea of using quantum properties to avoid local minima [50].

A training problem unique to variational quantum algorithms is known as barren plateaus [52]. This is a statement that as the system size increases and therefore the Hilbert space size increases, the gradients with respect to the variational parameters exponentially decrease [51]. This leads to the majority of the loss landscape to consist of near flat areas (plateaus barren of features hence the name) which lack enough information for the optimiser to direct to the global minima. This also affects gradient-free optimisers [53]. This phenomena is visualised in Fig. 2.9. Other factors have been shown to affect the occurrence of barren plateaus such as global cost operators [35] and magnitude of the dynamical Lie algebra of the variational ansatz generators [54, 55]. The dynamical Lie algebra of the ansatz generators is the set of operators generated by closing the group under commutation i.e. all commutators between the generators are calculated and included in the group. Then all the commutators between the current group members are calculated and included. This repeats until no new unique operators are found. The number of unique operators in the final group is the magnitude of the dynamical Lie algebra.

2.5.2 Generalisation

Once training finishes, we have access to the trained function. If the training was successful, within error margin, the function will achieve the target of the loss function. E.g. classifying a set of training data, fitting values at training points etc. This can be considered as behaviour over a set of training data. For some algorithms/use cases behaviour away from the training data is important. For example, if we have learnt how to classify a set of training data generally you also want to be able to classify new data correctly. How effective a solution is at this is known as generalisation.

Generalisation is affected by the loss function. The loss function dictates what is prioritised during training and could lead to certain information being over or under emphasised, in turn leading to poor generalisation. Furthermore, the training data considered can have a huge impact. For example, say you want to classify round shapes and square shapes but all the training data has the circles red and the squares blue. This could result in the model learning to classify based on colour instead of the desired criteria of shape and test data of a blue circle would be misclassified as a square.

Additionally, the model affects the generalisation. If the model is overly expressive it is more prone to what is known as overfitting. This is when the model learns to map the training data exactly to the desired output but doesn't learn the relationship between the data and its output. This results in poor generalisation. These problems of generalisation and overfitting are not unique to quantum machine learning and are important considerations for classical machine learning too [56, 57].

2.5.3 Training Aids and Techniques

These difficulties in training and generalisation affect the majority of quantum variational algorithms. Additionally many of them also affect classical variational algorithms (though maybe in slightly altered form). Therefore, there is much interest and research into techniques and approaches to either remove or mitigate these negative behaviours.

Current existing techniques for local minima and barren plateaus generally rely on changing the landscape in some known manner to remove or reduce the likelihood of these negative landscape features to appear. A selection of approaches are now discussed

One technique to reduce local minima is overparametrisation where the number of parameters in the variational ansatz is increased significantly past the number required for expressivity [58]. This results in a landscape with significantly reduced local minima. Informally, this is as the increased number of parameters leads to there always being an escape from points which were previously local minima. The number of parameters and therefore circuit depth required for this relates to the dimension of the lie algebra of the variational ansatz. The increased depth required may cause issues in near-term implementations. As the depth is linked to the magnitude of the lie algebra of the ansatz, ansatze can be constructed which can easily be over-parametrised however this is linked to a lack of expressivity and resulting classical simulability.

Attempts to avoid barren plateaus include limiting the factors known to induce barren plateaus such as entanglement [59], global measurements [35] and reducing ansatz expressivity [55]. Thus models and algorithms which reduce these are proposed [60, 61, 62]. However, none of the proposed approaches so far are considered to have ”solved” the problem of barren plateaus. This is as current approaches tended to either delay the onset of barren plateaus but not prevent it, lead to some other negative behaviour or are only applicable in niche circumstances. How to deal with barren plateaus remains an important open question.

For generalisation, one technique is regularisation [63, 64]. This is a way of controlling the model and/or the loss to control the chance of over-fitting and therefore improve generalisation. One such way is an additional term in the loss function. This term could give motive to the optimiser to reduce coefficients of the higher order/frequency fitting functions. or it could give a term which discourages high magnitude gradients at training points. Alternatively, choosing fitting functions suitable to the problem itself can help improve generalisation as the behaviour between training points is limited to suitable behaviour for the problem.

2.5.4 Data Input & Output Problem

Two non-training specific behaviours to note are the data input and output problems. The data input problem relates to the fact that there is currently no efficient known way to prepare an arbitrary state, it can require exponential circuit depth [65]. This can be an issue for algorithms which require the preparation of a state related to the problem considered.

The data output problem concerns the difficulty in receiving information on a whole unknown state from measurement. As when a state is measured only partial information is recovered - a single eigenvalue relating to a eigenstate of the measurement operator - to receive full information on a state with no additional information known requires full state tomography with a large number of measurements [66]. This causes issues for algorithms where the desired information to extract is encoded across the whole state

This discussion is particularly relevant to the choice of encoding as discussed in section 2.4.1. This is as amplitude encoding is more prone to being affected by these problems. As information is encoded in every amplitude it may be required to prepare an arbitrary state based on the problem and/or reading out every amplitude. Feature map encoding conversely encodes information as parameters of gates avoiding the input problem. And it generally extracts information with a chosen set of measurements avoiding the output problem.

2.6 Selected Variational & Quantum Machine Learning Algorithm Examples

2.6.1 Variational Quantum Eigensolver

One example of a variational quantum algorithm is the variational quantum eigensolver (VQE) [30, 67]. The goal is to find the ground state of a given system described by Hamiltonian $\mathcal{H}$. This is an example of an algorithm for quantum chemistry.

For this a trial state is prepared

$$|\psi_\theta\rangle = \hat{\mathcal{U}}_\theta|0\rangle, \tag{2.23}$$

where $\mathcal{U}_\theta$ is a chosen variational ansatz. The problem is then represented with the loss function

$$\mathcal{L}(\theta) = \langle\psi_\theta|\mathcal{H}|\psi_\theta\rangle. \tag{2.24}$$

This can then be minimised in the standard variational loop to find an estimator for the ground state. For computational ease of implementation generally the Hamiltonian is written as sum of terms $\mathcal{H} = \sum_{j=1}^{M} H_j$ where each term is easy to measure expectation of. Thus the loss will be measured as M expectation values. This algorithm can be generalised for other related purposes such as finding excited states [68]

2.6.2 Quantum Circuit Learning

An algorithm for regression problems is quantum circuit learning (QCL)[28]. This algorithm is both variational and QML. The target of this algorithm is to find a function which fit a set of points $\{(x_\mathrm{j}, f_\mathrm{j})\}_j$.

A quantum model is chosen as

$$f_\theta(x) = \langle 0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{U}}_\theta^\dagger|C|\hat{\mathcal{U}}_\theta\hat{\mathcal{U}}(x)|0\rangle. \tag{2.25}$$

A corresponding loss function for the problem is the sum of the distances between the target values f_j and the current trial values $f(x_j)$

$$\mathcal{L}_\theta(x) = \sum_j L\left(f_\theta(x_j) - f_j\right), \tag{2.26}$$

where L is a distance measure. If $f_\theta(x)$ perfectly fits the target values at the training points this loss is 0, otherwise it is positive. Therefore, an optimiser is used to minimise this loss function to train the trial function to fit the points. If a gradient descent optimiser is used the derivatives with respect to θ are required during training, in particular $\partial f_\theta(x)/\partial\theta_j \ \forall j$. If the variational ansatz is chosen appropri-

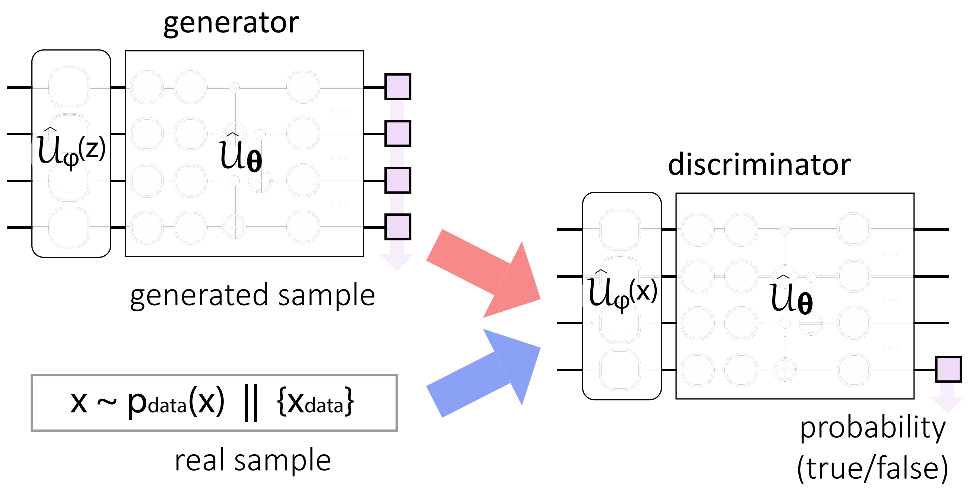


Figure 2.10: **Quantum GAN workflow.** The quantum circuits are used both for generative modelling (generator) and discrimination between real and fake samples (discriminator). Both circuits $\hat{G}_Q(z)$ and $\hat{D}_Q(x)$ are composed of (different) feature maps $\hat{\mathcal{U}}_\varphi(\cdot)$ followed by (different) variational circuits $\hat{\mathcal{U}}_\theta$. They are then trained adversarialy with minimax loss to improve generators ability to produce samples close to target distribution and for discriminator to distinguish between generated and target samples.

ately, the parameter shift rule as introduced in section 2.4.4 can be used.

After training a set of parameters is received θ^* which, when substituted into the trial function, gives a solution function which will fit the training points provided training was successful. This algorithm is inspiration for the algorithm we develop in chapter 3 for solving differential equations.

2.6.3 Quantum Generative Adversarial Networks

Quantum generative adversarial networks (QGAN) are an algorithm approach for solving generative modelling problems [29]. They are heavily based off their classical counterparts GANs – generally replacing the neural network models with quantum models.

The structure of GAN is represented by two neural networks: a generator G_{NN} and a discriminator D_{NN}. The generator takes a random variable $z \sim p_z(z)$ from a latent probability distribution $p_z(z)$. This is typically chosen as a uniform (or normal) distribution for $z \in (-1, 1)$. The generator uses this to prepare a fake sample

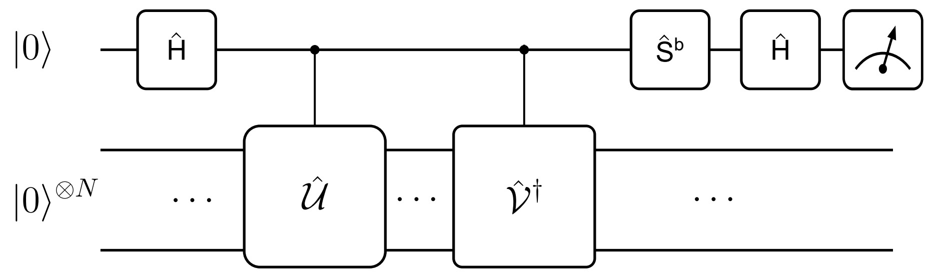


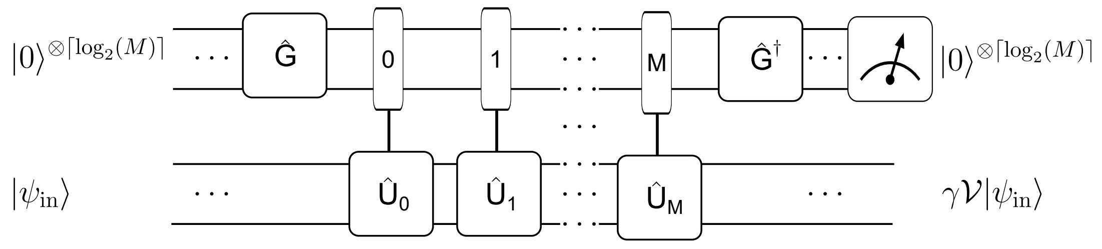
Figure 2.11: **Circuit diagram of Hadamard tests to measure overlaps.** Hadamard test for measuring $\mathrm{Re}\{\langle 0|\hat{\mathcal{V}}^\dagger\hat{\mathcal{U}}|0\rangle\}$ and $\mathrm{Im}\{\langle 0|\hat{\mathcal{V}}^\dagger\hat{\mathcal{U}}|0\rangle\}$ for $b = 0$ and $b = 1$, respectively. $\hat{S}$ denotes the phase gate, defined as $\hat{S} = \exp(-i\pi\hat{Z}/4)$.

$G_{\mathrm{NN}}(z)$ from the generator's probability distribution p_G, $G_{\mathrm{NN}}(z) \sim p_G(G_{\mathrm{NN}}(z))$. The goal is to make samples $\{G_{\mathrm{NN}}(z)\}_{s=1}^{N_\mathrm{s}}$, as close to the training dataset as possible, in terms of their sample distributions. If true samples $x \sim p_{\mathrm{data}}(x)$ are drawn from a (generally unknown) probability distribution $p_{\mathrm{data}}(x)$, our goal is to match $p_G(G_{\mathrm{NN}}(z)) \approx p_{\mathrm{data}}(x)$. This is achieved by training the discriminator network D_{NN} to distinguish true from fake samples, while improving the quality of generated samples $\{G_{\mathrm{NN}}(z)\}_{s=1}^{N_\mathrm{s}}$, optimising a minimax loss

$$\begin{aligned}\min_{G_{\mathrm{NN}}}\max_{D_{\mathrm{NN}}}\mathcal{L}_{\mathrm{GAN}} = \min_{G_{\mathrm{NN}}}\max_{D_{\mathrm{NN}}}\Big\{&\mathbb{E}_{x\sim p_{\mathrm{data}}(x)}[\log D_{\mathrm{NN}}(x)] \\ &+ \mathbb{E}_{z\sim p_z(z)}[\log(1 - D_{\mathrm{NN}}(G_{\mathrm{NN}}(z)))]\Big\},\end{aligned} \tag{2.27}$$

where D_{NN} and G_{NN} are the trainable functions represented by the discriminator and generator, respectively. The first loss term in Eq. (2.27) represents the log-likelihood maximisation that takes a *true* sample from the available dataset, and maximises the probability for producing these samples by adjusting variational parameters. The second term trains G_{NN} to minimise the chance of being caught by the discriminator. The loss corresponds to a minimax game, therefore instead of training to find a minimum we are instead training to find the Nash equilibrium [69].

The generator $\hat{G}_{\mathrm{Q}}(z)$ and discriminator $\hat{D}_{\mathrm{Q}}(z)$ as quantum models for QGAN are shown in Fig. 2.10.



$|0\rangle^{\otimes\lceil\log_2(M)\rceil}$
Ĝ
0
1
M
Ĝ†
$|0\rangle^{\otimes\lceil\log_2(M)\rceil}$
$|\psi_{\text{in}}\rangle$
Û0
Û1
ÛM
$\gamma\mathcal{V}|\psi_{\text{in}}\rangle$

Figure 2.12: **Circuit diagram of LCU.** Non-unitary of interest is first decomposed into sum of unitaries $V = \sum_{j=1}^{M} \alpha_j U_j$. $\lceil\log_2(M)\rceil$ ancillas are prepared as $|0\rangle$. These ancillas are then prepared into a superposition based on $\{\alpha_j\}_j$ by $\hat{G}$. Each term in the sum U_j is then implemented controlled by ancillas in position $|j\rangle$. Finally, the ancillas have $\hat{G}^\dagger$ applied and are measured. If they are measured as $|0\rangle$ the process was successful, otherwise it must be repeated.

2.7 Selected Quantum Circuits & Subroutines

2.7.1 Overlap Measurement

When introducing quantum models in section 2.4 we noted how they could have different structures. Expectations are read as measurements of the observable. Models including overlaps such as $\langle 0|\hat{\mathcal{V}}_\theta^\dagger \hat{\mathcal{U}}_\theta|0\rangle$ are another choice. To measure an overlap one method is to use modified Hadamard tests. The relevant circuit is shown in Fig. 2.11, representing the Hadamard tests with possible addition of the phase gate $\hat{S} = \exp(-i\pi\hat{Z}/4)$ [70]. First, the ancilla register is put into the symmetric superposition state by using Hadamard, followed by the two controlled preparation unitaries (one being reverted). Upon measurement in the Pauli $\hat{X}$ basis (with and without prior phase rotation S), we get the real and imaginary parts of overlaps, correspondingly (Fig. 2.11).

2.7.2 Linear Combination of Unitaries

All natural quantum operators are unitary but sometimes we may wish to implement a non-unitary operator. Linear combination of unitaries (LCU) is a method of implementing a non-unitary matrix which is written as a known sum of unitaries [71].

So say we want to implement $\mathcal{V} = \sum_{j=1}^{M} \alpha_j \hat{U}_j$. To achieve this $\lceil \log_2(M) \rceil$ ancillas are required. The ancillas are then prepared into superpositions which depend on the coefficients $\{\alpha_j\}_j$ as detailed in [71]. If a uniform addition is wanted then Hadamards are chosen. Then each term in the sum is implemented on the main register controlled by a unique ancilla bit string. These implementations are then combined by applying the adjoint of the ancilla preparation operator. Finally, the ancilla is projected onto $|0\rangle$ to effectively ”sum” the contributions. This leaves the main register in the state $\gamma\mathcal{V}|\psi_{\text{in}}\rangle$ where γ is a scaling constant.

By utilising a projective measurement this subroutine is not guaranteed success and is probabilistic. Therefore, if it fails the whole circuit has to be re-implemented adding a run-time overhead. There do exist techniques to mitigate this such as amplitude amplification [72].

2.7.3 Quantum Fourier Transform

The quantum Fourier transform (QFT) is a circuit which implements the transformation between the computational basis and the Fourier basis [6], i.e. the matrix representation of the circuit is the classical discrete Fourier transform. Many algorithms are reliant on this subroutine such as quantum phase estimation – an algorithm to calculate the phase of an eigenvalue for a given unitary [6].

QFT implements the operation

$$|j\rangle \rightarrow \frac{1}{2^{N/2}} \sum_{k=0}^{2^N-1} \exp\left(\frac{2\pi i jk}{2^N}\right) |k\rangle. \tag{2.28}$$

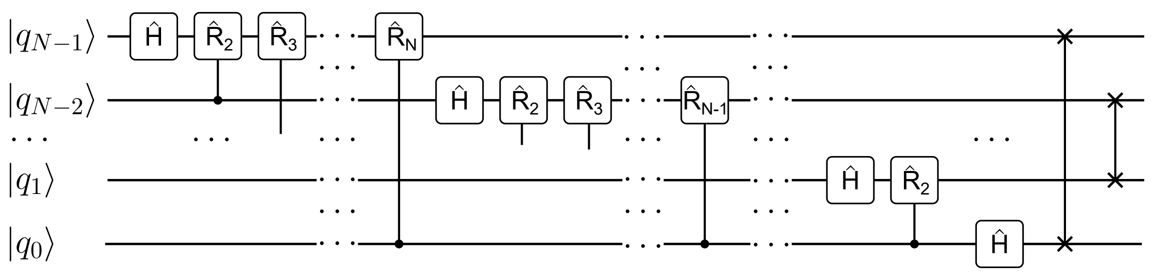


Figure 2.13: **Circuit diagram of quantum Fourier transform (QFT).** The QFT circuit which implements (2.28). It consists of N blocks and then finished by a series of swap gates. Each block is formed of Hadamard and controlled rotations acting on the same qubit. The controlled rotations are of the form $R_l = \mathrm{diagm}\left(1, \exp(2\pi i/2^l\right)$.

This can then be rewritten in a product form with $|j\rangle$ written in its bit string representation as

$$|j_{N-1}..j_1 j_0\rangle \to \frac{\left(|0\rangle + \exp\left[\frac{2\pi i b(j_0)}{2^1}\right]|1\rangle\right)\left(|0\rangle + \exp\left[\frac{2\pi i b(j_{0:1})}{2^2}\right]|1\rangle\right)...\left(|0\rangle + \exp\left[\frac{2\pi i b(j_{0:N-1})}{2^N}\right]|1\rangle\right)}{2^{N/2}}, \tag{2.29}$$

$$b(j_{0:k}) = \sum_{l=0}^{k} 2^l j_l. \tag{2.30}$$

From this representation it can be seen how to write the QFT circuit in terms of blocks of controlled rotations. In particular we note that the last term $|0\rangle + \exp\left[\frac{2\pi i b(j_{0:N-1})}{2^N}\right]|1\rangle$ depends on all qubits, the second term depends on all except the N^{th} qubit etc. until the first term $|0\rangle + \exp\left[\frac{2\pi i b(j_0)}{2^1}\right]|1\rangle$ which only depends on the first qubit. Therefore the n^{th} block (for the n^{th} term) is prepared onto qubit $N-n$ with gates altering only this qubit, controlled by lower magnitude qubits. We note that this prepares the desired state but with qubits in the wrong order. This is corrected by a series of swap gates but these can often be discounted and instead qubits can be relabeled for later operations. This is similar to the bit reversal used in the classical fast Fourier transform implementations of the discrete Fourier transform [73].

The n^{th} block starts with a Hadamard on qubit $N - n$ which acts as $|j_{N-n}\rangle \to \frac{1}{\sqrt{2}}(|0\rangle + \exp(2\pi i j_{N-n}/2)|1\rangle)$. Then follows a series of controlled rotations of $\hat{R}_l =$

$\mathrm{diagm}\left(1, \exp(2\pi i/2^l\right)$ with l depending on term n and control m as $l = N + 1 - n - m$. These rotations leave $|0\rangle$ unaltered and only change $|1\rangle$ if the control qubit $|j_m\rangle$ is in state $|1\rangle$. Together these gates prepare the n^{th} term. The full circuit is shown in Fig. 2.13.

2.7.4 Adder & Subtractor Circuits

We now consider a circuit which can implement the operation:

$$\hat{\mathcal{U}}_{\text{ADD}} : |g\rangle_N |h\rangle_N |0\rangle_{N+1} \to \sum_{j,k} |j\rangle_N |k\rangle_k g_j h_k |j + k\rangle_{N+1}. \tag{2.31}$$

This is effectively adding the N qubit states $|g\rangle$ and $|h\rangle$ in the bit basis. The result is an N+1 qubit state. There are multiple different ways to implement this and I will describe two options.

The first option is via multiple controlled nots and Toffolis [74]. CNOTs and Toffolis are considered due to how they control on $|0\rangle$ and $|1\rangle$ to flip another qubit between $|0\rangle$ and $|1\rangle$, what we want to do with bit addition.

For clarity we consider the case of adding two basis states $|j\rangle$ and $|k\rangle$ with the full result naturally satisfied by linearity. The problem is split into N sub-problems, one for each digit. From lowest to highest magnitude qubit we want to evaluate

$$|j_l\rangle |k_l\rangle |r_{l+1}\rangle |r_l\rangle \to |j_l\rangle\ |k_l\rangle\ |\lfloor (r_l + j_l + k_l)/2 \rfloor\rangle\ |(r_l + j_l + k_l) \bmod 2\ \rangle, \tag{2.32}$$

where $|\cdot_j\rangle$ represents the j^{th} bit of the related state $|\cdot\rangle$. This sums the two given qubits with any carry from previous operation and sets any further carry for the next sub-problem to consider. This can be achieved in many ways as detailed in [74]. One such example is shown in Fig. 2.14 (a). The full adder is then formed of successive implementations of each block.

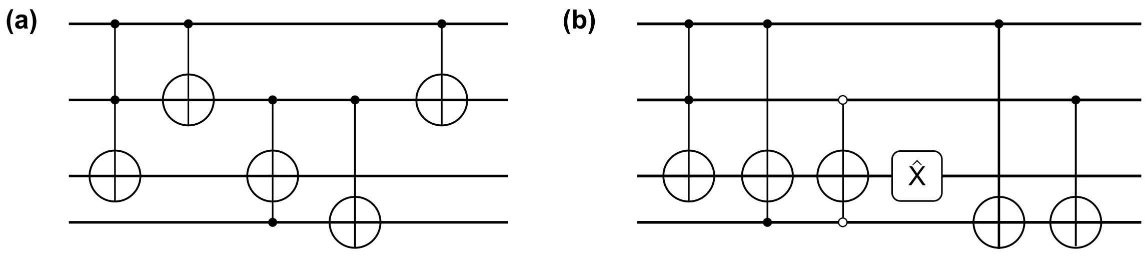


Figure 2.14: **Circuit diagram of Adder and Subtractor blocks.** (a) Block which implements $|A\rangle|B\rangle|0\rangle|C_{\text{in}}\rangle \rightarrow |A\rangle|B\rangle|A+B+C_{\text{in}}\rangle$. Repeated blocks over different qubits will implement full adder (2.31).(b) Block which implements $|A\rangle|B\rangle|0\rangle|C_{\text{in}}\rangle \rightarrow |A\rangle|B\rangle|A-B-C_{\text{in}}\rangle$. Repeated blocks over different qubits will implement full adder (2.36). Note that a controlled operation denoted with a white control node acts when the control register is $|0\rangle$ rather than $|1\rangle$.

The second option is to utilise the quantum Fourier transform [75]. In Fourier space addition relates to multiplication due to $\exp(aG)\exp(bG) = \exp((a+b)G)$ and therefore alters the approach to the problem. This example operates as

$$\hat{\mathcal{U}}_{\text{ADD}} : |g\rangle_N|0\rangle_1|h\rangle_N \rightarrow \sum_{j,k} |j\rangle_N g_j h_k |j+k\rangle_{N+1}. \tag{2.33}$$

Again we consider the case of adding two basis states $|j\rangle$ and $|k\rangle$ which generalises. For this implementation, first the quantum Fourier transform is applied to $N+1$ qubits of $|0\rangle|k\rangle$ for

$$|0\rangle_1|k\rangle_N \rightarrow \frac{1}{2^{N+1/2}} \sum_{l=0}^{2^{N+1}-1} \exp\left(\frac{2\pi i k l}{2^{N+1}}\right)|l\rangle_{N+1}, \tag{2.34}$$

as in (2.28). We also know that the Fourier transform of $|j+k\rangle$ would be

$$|j+k\rangle_{N+1} \rightarrow \frac{1}{2^{N+1/2}} \sum_{l=0}^{2^{N+1}-1} \exp\left(\frac{2\pi i (j+k) l}{2^{N+1}}\right)|l\rangle_{N+1}. \tag{2.35}$$

Therefore we would like to map (2.34) to (2.35) at which point the inverse quantum Fourier transform can be applied for the result.

Using the intuition from how the QFT circuit was built and again considering computational basis states as bit strings, we can find this map as a series of controlled operations. The rotations are of the same form as in the QFT $\hat{R}_j =$

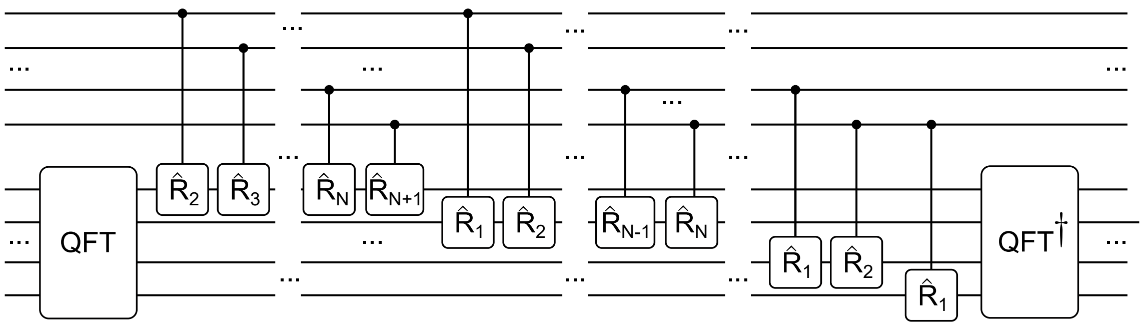


Figure 2.15: **Circuit diagram of Adder via QFT.** Circuit to implement (2.33) using QFT as in Fig. 2.13 as and controlled rotations $R_j = \mathrm{diagm}\left(1, \exp(2\pi/2^j\right)$.

$\mathrm{diagm}(1, \exp\left(2\pi/2^j\right)$. This circuit is shown in Fig. 2.15.

With bit addition implemented, we now consider how to implement bit subtraction as

$$\hat{\mathcal{U}}_{\mathrm{SUB}} : |g\rangle|h\rangle|0\rangle \to \sum_{j,k} |j\rangle|k\rangle g_j h_k |(j-k) \bmod 2^N\rangle. \tag{2.36}$$

As subtraction is the inverse operation from addition, the implementation is deeply linked to the implementation of addition. For the CNOT and toffoli gate implementation the block considering borrowing from higher magnitude qubits is shown in Fig. 2.14(b). For the QFT implementation the alteration is as simple as the altering of the signs in the rotations (resulting in $|k - j \bmod 2^N\rangle$. This is because the Fourier transform of the subtracted state as compared to (2.35) simply has $(j + k)$ altered to $(k - j)$.

2.8 Selected Machine Learning Techniques

2.8.1 SciML & PINNs

There exists a qualitatively distinct family of protocols where DE problems are reformulated as machine learning (ML) tasks. The corresponding field is referred to as scientific machine learning (SciML) [76, 77].

SciML approaches are often based on physics-informed neural networks (PINNs) [78, 79] — deep neural networks that include differential constraints introduced with automatic differentiation. PINN-based solvers have been applied to many use-cases [80, 81]. Although still remaining relatively niche [82], their flexibility allows them to compete with FEM/SEM solvers in cases involving data-based constraints [81], optimisation loops [83], and can excel when dealing with stiff and multidimensional problems [77]. On one hand, the training grid for PINN models that generalise well can use relatively fewer points training points than similar methods [83]. However, PINN-based DE solvers require a costly training process that leads to large overheads as compared to conventional methods such as FEM, often needing $> 10,000$ epochs with many function and gradient evaluations. While the required function evaluations on the grid may be parallelised on GPUs [77], in practice the cost is often not considered worth it for many industrial applications, despite the potential of the ML setting and what the flexibility can offer in terms of utility in use cases. Reducing the grid evaluation cost for each epoch, and removing L_c scaling, is crucial for making PINNs industrially practical.

2.8.2 Kernel Methods

One class of algorithms typically used for classification and regression is kernel methods [42]. For these, the model being trained is written in term of kernel functions – a type of similarity function. A kernel function is a conjugate-symmetric positive definite function κ mapping two variables $x, y \in \mathcal{X}$ to the complex space, $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{C}$. A model is then constructed in terms of these – for example

$$f(x) = \sum_{j=1}^{M} \alpha_j \kappa(x, y_j) + \beta. \tag{2.37}$$

Any suitable algorithm can then be used with this model to find the solution to a given problem.

A unique property of kernel functions which is used in many kernel methods is the kernel trick. The kernel trick is rewriting a problem using the fact that any kernel function can be written as an inner product in a potentially high dimensional feature space, $\kappa(x, y) = \varphi^\dagger(x)\varphi(y)$. Conversely, $\varphi^\dagger(x)\varphi(y)$ always represents a valid kernel function. This is a consequence of Mercers theorem [84]. Informally, it corresponds to the statement that for any symmetric positive definite function $f(s, t)$ there exists a countable set of functions $\{\phi_i\}_i$ such that $f(s, t)$ can be expressed as $f(s, t) = \sum_i \phi_i(s)\phi_i(t)$. By using the kernel trick the model can be expressed in terms of a function defined over a high dimensional space but it does not need to be evaluated in that space.

As mentioned kernel functions can be utilised in many different ways depending on the problem considered [42]. Here I explore the use of a method known as support vector regression (SVR) for regression and differential equation problems. This method utilises the kernel trick to rewrite a given problem as a system of equations to solve. We focus on this application as we will build upon this in a quantum computing setting in Chapter 5.

Support Vector Regression

In this section SVR for regression is explained [85]. By following a set series of steps a regression or classification problem can be written as system of linear equations. The resulting problem is convex, allowing for all the known positive qualities of solving convex problems [86].

To begin, a trial function is represented as $f(x) = \mathbf{w}^\dagger\varphi(x) + b$, where $\mathbf{w}$ and b are tunable parameters, and $\varphi(x)$ is a set of functions we later use the kernel trick upon. Note at this stage we do not have to choose specific $\varphi(x)$ and can leave this is as a statement that such $\varphi(x)$ would exist for our model. The first step is to write the problem as a primal (original) optimisation model. This reads

$$\min_{w,b}\{\mathbf{w}^T\mathbf{w} + \gamma\mathbf{e}^T\mathbf{e}\}, \tag{2.38}$$

$$\text{subject to } f_i = \mathbf{w}^T\varphi(x_i) + b + e_i, \quad i = 1 : N, \tag{2.39}$$

where $\mathbf{e}$ is the set of error variables which relax the constraints $f_i = \mathbf{w}^T\varphi(x_i)+b+e_i$, and γ is a tunable hyperparameter that changes the emphasis on minimising the error.

The process that follows is to write the model in its Lagrangian form, introducing a set of variables (known as dual variables) to implement each constraint. The Karush-Kuhn-Tucker (KKT) optimality conditions are then found, which emerge from equating the first derivative of the Lagrangian with respect to each of the primal and dual variables to zero [87]. These conditions are then used to eliminate a subset of the primal variables. This can intuitively be understood as turning the variable into a constraint. This leads to a system of equations which have terms of $\varphi(x_i)^\dagger\varphi(x_j)$, and by using the kernel trick these terms can be changed to $\kappa(x_i, x_j)$. Now the problem is written in a dual form as a system of equations to solve with coefficients involving kernel evaluations. Similar to the MMR method, these have to be evaluated once at the start. The resulting system of equations is

$$\left[\begin{array}{c|c} \hat{\Omega}+\hat{I}/\gamma & \mathbf{1} \\ \hline \mathbf{1}^T & 0 \end{array}\right]\left[\begin{array}{c} \boldsymbol{\alpha} \\ \hline b \end{array}\right] = \left[\begin{array}{c} \mathbf{f} \\ \hline 0 \end{array}\right], \tag{2.40}$$

where $\Omega_{i,j} = \kappa(x_i, x_j)$, $\hat{\Omega} = \{\Omega_{i,j}\}_{i,j}$, and $\boldsymbol{\alpha}$ are a set of introduced dual variables. The system of equations can now be solved with any available method to solve such a problem. Once solved the relevant KKT conditions can be substituted into the expression $f(x;\alpha) = \mathbf{w}^\dagger\varphi(x) + b$, and the kernel trick applied to get an expression for $f(x)$ in terms of the dual variables, which have been solved for and kernel evaluations. We thus write our model as

$$f(x;\alpha) = \sum_{i=1}^{|\mathbf{x}|} \alpha_i\kappa(x, x_i) + b. \tag{2.41}$$

Although we started considering φ as our fitting functions, the resulting function of this process is based on kernel evaluations and we never need knowledge of what φ are or to directly evaluate them. We can choose the kernel functions to use and know simply that appropriate φ exist due to Mercers theorem/the kernel trick.

The workflow to prepare an SVR problem is as follows:

1. Write model with minimisation function and constraints.
2. Write out Lagrangian.
3. Find the KKT optimality conditions.
4. Eliminate subset of original optimisation variables.
5. Use the kernel trick to realise problem in terms of kernels.
6. Write out remaining relationships as system of equations.
7. Use KKT conditions and kernel trick to express function in terms of kernel functions.

The prepared SVR model can then be used for any problem of the form assumed in preparing the original model. The resulting system of equations can be solved with any suitable method. This workflow and problem set up can then be generalised for differential equation problems. The work flow remains the same with the constraints now being formed of the differential equation at a set of points along with the initial value/boundary condition. As we will make use of this later the full process for an example DE is detailed in appendix A

2.8.3 Quantile Mechanics

Quantile mechanics is a method of formulating stochastic differential equations (SDEs) to solve [88]. In simplistic terms, an SDE is a DE which contains a stochastic/random noise term [89]. Consequently, the solution of the SDE is a probability distribution. With quantile mechanics the SDE is solved in terms of a quantile function (QF) rather than directly as a probability distribution.

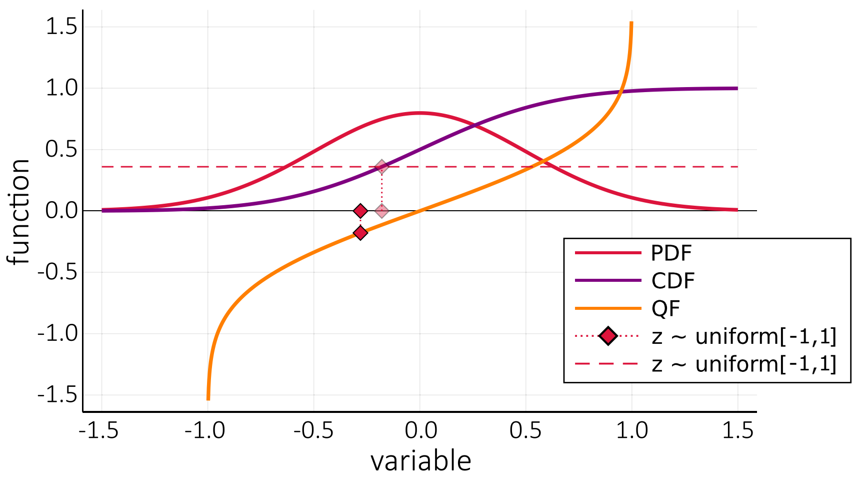


Figure 2.16: **Sampling of a quantile function.** We plot a normal probability density function as a red curve (PDF), being the Gaussian function with $(\mu, \sigma) = (0, 1/2)$. The domain is chosen as $\mathcal{X} = [-1.5, 1.5]$. The corresponding cumulative distribution function is presented by the purple solid curve (CDF). The quantile function, rescaled to $z \in [-1, 1]$, for the corresponding CDF is shown in orange (QF). The red diamonds correspond to a randomly drawn latent variable z, and associated probability for the sample value, connected by dotted lines.

All probability distributions $p(x)$ have an associated quantile function $Q(z)$. This quantile function is the inverse of the cumulative distribution function (CDF) $F(z)$

$$Q(z) \equiv F^{-1}(z). \tag{2.42}$$

This definition is such that if $Q(z)$ is sampled by z following the uniform distribution $Z \sim U(0, 1)$ the resulting samples X follow the associated pdf $X \sim p(x)$. To see this first consider the CDF defined as $F_X(x) = \int_{-\infty}^{x} p(x')dx'$. This maps possible sample values x to a value lying in $[0, 1]$ range on the ordinate axis (see Fig. 2.16 for an illustration). Therefore the inverse function $F^{-1}(z)$ maps $[0, 1]$ to sample values. We note z can easily be rescaled with a popular choice of $[-1, 1]$.

Often finding the associated quantile functions for a given PDF is difficult, finding $F^{-1}(z)$ generally leading to a transcendental problem without a closed-form solution. This poses computational challenges and requires graphical solution methods [90]. Alternatively, they can be obtained by solving nonlinear partial differential equations derived from an SDE. A general system of stochastic differential equations can be written as [89]

$$d\boldsymbol{X}_t = f(\boldsymbol{X}_t, t)dt + g(\boldsymbol{X}_t, t)dW_t, \tag{2.43}$$

where $\boldsymbol{X}_t$ is a vector of stochastic variables parameterised by time t (or other parameters). Deterministic functions f and g correspond to the drift and diffusion processes, respectively. W_t corresponds to the stochastic Wiener process. The stochastic component makes SDEs distinct from other types of partial differential equations, adding a non-differentiable contribution and leading to a solution as a probability distribution.

An SDE can be rewritten to be in terms of a quantile function, the resulting DE known as quantilised Fokker-Planck equation. For the general form in (2.43) this is (see full derivation in Ref. [88])

$$\begin{aligned}\frac{\partial Q(z,t)}{\partial t} =& f(Q,t) - \frac{1}{2}\frac{\partial g^2(Q,t)}{\partial Q} \\ &+ \frac{g^2(Q,t)}{2}\left(\frac{\partial Q}{\partial z}\right)^{-2}\frac{\partial^2 Q}{\partial z^2},\end{aligned} \tag{2.44}$$

where $f(Q,t)$ and $g(Q,t)$ are the drift and diffusion terms familiar from Eq. (2.43). Eq. (2.44) is solved as a function of latent variable z and time t. Once $Q(z,t)$ is known, evaluating it at random uniform z's as t progresses we can get full time series (trajectories) obeying the stochastic differential equation (2.43). This approach is called *quantile mechanics*.

2.8.4 Fitting Bases

When trying to fit functions to a problem – such as regression or DEs – one method is to choose a set of basis functions $\{T_j(x)\}_j$ and then the algorithm's goal is to find the best approximation of the problem as a linear combination of these basis functions.

One popular choice is Fourier functions where

$$T_j(x) = \exp(-i2\pi jx/J) \quad j = 0 : J-1. \tag{2.45}$$

Therefore the fitting functions are trigonometric and periodic and give insight into the frequencies of the function. There are known accuracies for the fitting of Fourier series and convergence as basis set size increases [91].

Additionally, say we took $v(x) = (T_0(x), T_1(x), ...T_J(x))^T$ and evaluated at integers 0:J. The resulting set of vectors forms an orthonormal base. The map from this to identity is then the Discrete Fourier transform – as discussed in section 2.7.3.

An alternative to Fourier functions is to use Chebyshev Polynomials of the first kind $T_j(x)$ and/or second kind $U_j(x)$. These are of the form

$$T_j(x) = \cos(j \operatorname{acos}(x)) \quad j = 0 : J-1, \tag{2.46}$$

$$U_j(x) = \sin((j+1) \operatorname{acos}(x))/\sqrt{1-x^2} \quad j = 0 : J-1. \tag{2.47}$$

With x restricted to $[-1, 1]$ these can be written as polynomials, are not periodic but are restricted to the range given. Similar to Fourier series these have known behaviours for fitting [92]. Associated with the Chebyshev polynomials of the first kind are the Chebyshev nodes

$$x_j = \cos((2j+1)\pi/2J) \quad j = 0 : J-1. \tag{2.48}$$

These are the zeros of the J^{th} first kind Chebyshev polynomial and are often used as training points as their use minimises errors [92] – in particular unwanted oscillatory behaviour at the interval edges.

Chebyshev polynomials also have other properties such as chaining, nesting, and simple differentiation rules. The chaining properties for polynomials of the first and second kind read as $2T_m(x)T_n(x) = T_{m+n}(x) + T_{|m-n|}(x)$ and $U_m(x)U_n(x) =$

$\sum_{k=0}^{n} U_{m-n+2k}(x)$, respectively. Derivatives can be obtained as $dT_n(x)/dx = nU_{n-1}(x)$. Nesting corresponds to the relation $T_n(T_m(x)) \equiv T_{nm}(x)$. Finally, polynomials of different kinds can be converted between as $U_n(x) = 2\sum_{j\text{ even}}^{n} T_j(x)$ when n is even, and $U_n(x) = 2\sum_{j\text{ odd}}^{n}[T_j(x) - 1]$ when n is odd.

There are also of course many of other choices of basis functions but these are two that are commonly used and will be used later in my work. Choice of basis function set often comes down to what most suits the problem itself – i.e. Fourier functions would be more appropriate than Chebyshev for a trigonometric problem.

2.9 Overview of Differential Equations & Solvers

Differential equations govern many areas of industrial and research interest, from aerodynamics to finance to chemistry, yet many instances remain difficult to solve classically. A differential equation (DE) is defined as an equation of unknown functions which includes at least one derivative [93, 94]. The aim of solving a (DE) is to find the functions such that the equations are satisfied. Throughout this thesis we tend to write differential equations as functionals in their residue form e.g.

$$DE[\mathbf{f}, \partial\mathbf{f}, \mathbf{x}] = \mathbf{0}. \tag{2.49}$$

Here $\mathbf{f}$ denotes the unknown functions, $\partial\mathbf{f}$ the derivatives of interest and $\mathbf{x}$ the independent variables. Solving a DE (when solvable) tends to lead to a family of functions with degrees of freedom. Therefore, generally initial conditions or boundary values are specified along with the problem to solve, such as

$$f(x_0) = f_0 \;\text{ or }\; f(\boldsymbol{x}) = g(\boldsymbol{x})\; \boldsymbol{x} \in B. \tag{2.50}$$

By specifying these values a particular solution is set.

An example of a simple form of differential equation is

$$\frac{df}{dx} - g(x)f - h(x) = 0. \tag{2.51}$$

This is a linear DE with a single unknown function f, first order derivatives and single independent variable x. Most DEs of interest, particularly those hard to solve, include other features and properties which generally increase complexity. One property is the inclusion of higher order derivatives. Another is multidimensional functions $f(\boldsymbol{x})$ where there a multiple independent variables. Additionally, non-linear problems can be considered where terms of the DE include products of unknown functions and their derivatives. Also some DEs are instead systems of DEs, including multiple DEs describing multiple unknown functions $\boldsymbol{f}$ to be solved simultaneously. A given DE problem could have any subset of these features.

As previously mentioned, it is accepted that for quantum computing to find advantage, problems which are hard to solve classically should be considered. The field of DEs include such problems and we consider features which can lead to classical inefficiency. One possible type of hard problem is those with large numbers of dimensions. This is due to how the amount of data that needs to be considered whilst solving scales with dimensionality and is sometimes known as the ”curse of dimensionality” [95]. Another feature often leading to classically hard to solve DEs, is what is known as stiffness and relates to the degree of precision needed to be considered during solving due to numerical instability [96]. Furthermore, some DEs can be chaotic meaning any deviation in initial value leads to wildly different solutions [97]. These properties and others lead to many DEs of interest which are currently hard to solve classically and are therefore possible areas to search for quantum advantage.

However, these are just possible areas for advantage, being classically hard is not enough to guarantee quantum advantage, the problem could be hard both classically and for quantum computation. Therefore, when developing quantum algorithms we generally want to focus our attention on areas where we also have intuition for possible quantum advantage. For DEs this often becomes problems which are large scale. Then it is hoped that the exponential computational space could be used to represent and process the large amounts of data and/or represent complicated functions (i.e. expressivity). I now give an overview of current DE solvers, both classical and quantum.

2.9.1 Differential Equation Solvers

Classical

As differential equations are such an area of interest there exists many methods to solve them. First of all is solving them analytically – finding the exact expression for the solution [98]. This gives a perfect solution and most DE applications would ideally be solved by receiving such an expression. However, many DEs of interest cannot be solved so. Instead numerical methods are utilised. These numerical methods are further divided into local and global methods.

Local approaches rely on discretisation of the space of variables, with derivatives being approximated with numerical differentiation techniques (such as finite difference methods). There are different ways this can be used to solve DEs, such as writing as linear system equations to solve or to iteratively step forward from the initial condition point by point. When using this type of method generally no assumption is preemptively hard-coded into the solution – it is solving for a set of values over the discretised space and interpolation is dealt with later if needed – giving useful flexibility. However, often a fine grid for multivariable functions is required to represent a solution qualitatively and quantitatively [99], leading to increasing computational cost.

Global methods represent the solution in terms of a suitable basis set [90]. This recasts the problem to finding optimal coefficients for the polynomial approximation (e.g. Fourier or Chebyshev) of the sought function. By choosing the set of basis functions the type of functions easily representable is set, therefore choosing a suitable basis set for the problem is important. When such a basis set is chosen these methods tend to be good at ensuring that desired behaviour in the solution is shown. When suitable basis functions have been chosen for a problem, convergence generally outperforms that of local/discretised methods and can be exponentially fast [100]. Additionally interpolation is not required as the space has not been discretised. However, finding spectral solutions for complex problems may require ever-increasing basis sets to achieve high accuracy, enlarging the global differential operator, and introduces difficulties when dealing deterministically with boundary conditions. What can be considered as a particular type of global method is PINNs as introduced in section 2.8.1

Quantum

Because differential equations of interest can be hard to solve classically, the solving of DEs has been highlighted as an area of potential use for quantum computing. Referring back to section 2.1.4 there are two main ways that quantum computers are hoped to potentially get a speed-up. One is when the differential equation is discretised as a system of equations, and therefore the naturalness of linear algebra to quantum computing can be used[101, 102]. The other is that certain DEs can have huge amount of information – i.e. high dimension or a large system. Thus the exponential large space could be utilised.

Quantum algorithms for DEs have been developed and is still an area of interest. Many fall under the class of local methods and utilise the discretised space and amplitude encoding to encode and consider an exponential number of points. In particular the DE is reformulated as linear system of equations which can then be solved with an algorithm such as HHL [101, 102]. Alternatively, iterative step methods [103] and time marching techniques [104] have been considered. These methods tend to be effective when considered for linear differential equations with provable guarantees given efficient preparation of arbitrary states, efficient reading of final state and circuit representation of the relevant matrix. Nonlinear versions are being considered [105] however they remain harder to deal with due to the fundamentally linear nature of quantum mechanics. This class of methods tends to utilise amplitude encoding with the solution being encoded over all the amplitudes of a state, therefore requiring the full state to be measured to retrieve the solution. Additionally, these algorithms often require the preparation of an arbitrary state based on the right hand side of the discretised DE. Therefore the data input and output problems are as described in section 2.5.4 often need to be considered.

A different approach for local methods is presented in [1]. This is a hybrid variational approach and considers nonlinear differential equation. Nonlinearity is encoded via *quantum nonlinear processing units* realised by ancillary quantum registers and controlled-multiqubit operations. The approach uses amplitude encoding, offering memory saving while potentially facing the data input and output problems.

There has been comparatively less work thus far for global methods. Considering such an approach to solving differential equations is the work [106]. Here Chebyshev polynomials are used as a basis set. By using the derivative properties and multiplication properties of the Chebyshev polynomials the differential equation can be written as a system of equations – solved this times for coefficients rather than discretised function values. For regression a variational global method has been considered as quantum circuit learning [28] as discussed in section 2.6.2 but not for differential equations. We therefore consider how such a global variational approach can be utilised for the solving of differential equations.

So far my focus has been on digital approaches. Alternatively, there are quantum annealing approaches. Quantum annealing encodes the solution of a problem as a ground state of a quantum system. This system is then prepared on the quantum device and evolved to find the ground state. This type of computation could be used to solve DEs with approaches such as in [107].

2.10 Classical Simulation of Quantum Algorithms

During the development of quantum algorithms, often one wants to implement and test how it runs. However, current quantum devices are limited and expensive enough that this is generally infeasible during development. Instead, during development the algorithm is often simulated on classical devices.

When classically simulating a quantum algorithm it is coded such that the classical computer mimics performing the quantum operations by utilising the mathematical framework of quantum mechanics. This is often done with what is known as full state simulation. The classical computer performs linear algebra where the full state is known as a vector and operators are implemented via their associated matrices. The size of these operations increase exponentially with qubit number and therefore these simulations are inefficient. This is expected, if it could be simulated efficiently there would be no advantage for quantum computation. Therefore these classical simulations are often of limited qubit scale.

Another difference is that classical simulation does not include any intrinsic noise. This allows easy checking of how the algorithm would run in a fully idealised situation. However, significant noise is a for now unavoidable part of quantum computation and therefore it's often desired to know how the algorithm responds to the presence of noise. For this noise models can be included which adds in noise modeled on the noise of devices.

It is possible to encode these simulations entirely from scratch with just linear algebra packages. However, for efficiency, generally a package specifically for classical simulation of quantum algorithm is utilised. There are a variety available such as Qiskit [108], PennyLane [109], Yao [110] and Qadence [111]. These packages tend to have predefined operations and functions to aid the implementation of simulations. Additionally, they often have some built in optimisation in the application of operators than naive matrix application. Some packages are developed related to particular devices so may have functions or noise models specific to that device. Choosing a package to use depends on necessary functions, what programming language you want to use and potentially desired device.

2.10.1 My Implementations

Throughout my thesis I develop algorithms and test run them on example problems. For this I classically simulate my results. I utilise the package Yao [110] used with the programming language Julia. This package was chosen for its built in efficient derivative calculations which are important for DEs and gradient based optimisers.

As my focus is on QML and variational methods there are some similarities between my simulations. Notably they start with a trial function being defined as a quantum model. A loss function is then defined which when minimised results in the trial function solving the given problem. We then have to decide what optimiser to use. These are then used together to minimise the loss function and find a set of theta. If training was successful these theta will result in the quantum model representing the solution to the problem. Pseudocode of this process follows below:

Algorithm 1 General Workflow

```
1: def trial function $f_\theta(x;\theta)$
2:     using number of qubits, variational ansatz, feature map
3: def loss function $\mathcal{L}(\theta)$
4:     using training grid in x, trial function, trial function derivatives
5: def optimiser
6:     using learn rate, maximum iteration number
7: def initial parameters
8: minimise $\mathcal{L}(\theta)$ w.r.t $\theta$ loop
9:     evaluate $\mathcal{L}(\theta)$ and derivatives at current $\theta$
10:    provide to optimiser for new suggested $\theta$
11:    update $\theta$
12:    return to $9$ until end condition satisfied
13: receive optimised parameters $\theta^*$
14: if test problem with known solution evaluate accuracy of resulting function
15:     using $f_\theta(x;\theta^*)$, known solution, measure of success, plotting
```

This is a simplified view of my methodology for implementing classical simulations. The quantum model and loss function changes significantly based on problem and algorithm and provides a lot of complexity in the algorithm and its implementation. Additionally, there are variations on this simplified workflow where significant pre or post calculations are required. Other variations include inclusion of any training techniques such as regularisation and initialisation.

As a specific example, code for my implementation of QCL from [28] as discussed in section 2.6.2 is included in appendix B. This follows closely the workflow as presented in the pseudocode for the task of function learning regression.

Chapter 3

Solving Nonlinear Differential Equations with Differentiable Quantum Circuits

3.1 Declaration of Contribution

The work presented in this chapter is published within [112]. My contributions to this work involve the implementation and running of all simulations, the formulation of the boundary handling and shared contributions for writing of the paper.

3.2 Introduction

In this chapter we develop a variational quantum algorithm for the solving of nonlinear PDEs. We propose a quantum model representing a trial solution of a given DE problem which can then be optimised for the solution by minimising the corresponding loss function. The core of this idea relies on the differentiation of the encoding circuit, the quantum feature map, that allows the representation of an analytical function derivative in a quantum form and the ability to search for the solution in the exponentially large Hilbert space. We show how the method can be applied to industrially relevant problems, and consider a particular example in fluid dynamics described by Navier-Stokes equations. In this example, we compute density, temperature and velocity profiles for the fluid flow in the convergent-divergent nozzle and find good convergence to the correct solutions.

Notable features of this algorithm include the use of quantum feature map encoding allowing for exact derivatives, access to the function value for continuous x and simpler boundary handling. Furthermore, the structure of our quantum model helps the data input-output problem to be avoided. Also the suitable solution search takes place in a exponential space of fitting polynomials with respect to number of qubits. This can lead to high expressivity for limited physical resources. These points and the variational workflow leads to an algorithm suitable for near-term quantum devices. Devices with increased scale and reduced noise would be able to run the algorithm with a wider variety of quantum model circuit choices and parallel training leading to wider variety of expressivity and potentially improved training behaviour.

3.3 Method

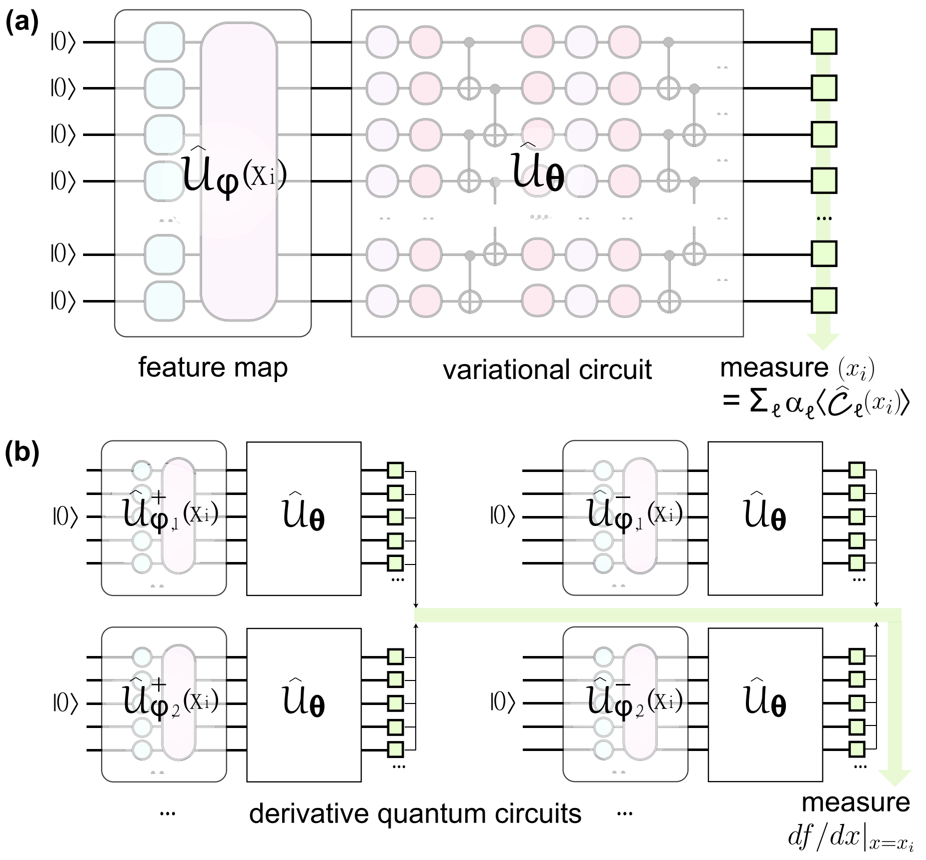
3.3.1 Algorithm

3.3.2 Quantum Model

When solving a DE the target is to find the function(s) which solve the given problem. To achieve this we represent a trial function $f_\theta(x)$ as a quantum model and train it to solve the given DE. How to form a quantum model of a function was discussed in section 2.4. We choose to use the quantum model

$$f_\theta(x) = \langle 0|\hat{\mathcal{U}}_\varphi^\dagger(x)\hat{\mathcal{U}}_\theta^\dagger|C|\hat{\mathcal{U}}_\theta\hat{\mathcal{U}}_\varphi(x)|0\rangle, \tag{3.1}$$

where $\hat{\mathcal{U}}_\varphi(x)$ is a feature map, $\hat{\mathcal{U}}_\theta$ the variational ansatz and C the cost function. The circuit diagram for this model is shown in Fig. 3.1(a). Our algorithm is still valid for alternative quantum models provided that the parameter shift rule (section 2.4.4) is valid to apply or there is an alternative method for calculating



(a)
|0⟩
$\hat{U}_\varphi(x_i)$
$\hat{U}_\theta$
feature map
variational circuit
measure (x_i)
$= \Sigma_\ell \alpha_\ell \langle \hat{\mathcal{C}}_\ell(x_i) \rangle$
(b)
$\hat{U}^+_{\varphi,1}(x_i)$
$\hat{U}^-_{\varphi,1}(x_i)$
$\hat{U}^+_{\varphi,2}(x_i)$
$\hat{U}^-_{\varphi,2}(x_i)$
derivative quantum circuits
measure
$df/dx|_{x=x_i}$

Figure 3.1: **Differentiable quantum circuits.** (a) Quantum circuit used for encoding the value of a function at a specific value of the variable $x = x_i$. The circuit consists of a feature map $\hat{U}_\varphi$ that encodes the x-dependence, followed by variational ansatz $\hat{U}_{\boldsymbol{\theta}}$, and an observable-based readout for the set of operators $\hat{C}_\ell$. The measurement result is classically post-processed to provide a quantum function representation $f_\theta(x)$ as a sum of expectations. To compose the loss function circuit measurements for different points of optimisation grid $\{x_j\}_j$ are required. (b) Derivative of the sought function $f_\theta(x)$ evaluated at specific point $x = x_i$ is estimated as a sum of expectations for derivative quantum circuits. The full structure follows from the feature map differentiation described in the text and shown for example in Fig. 3.2.

derivatives. The choice however can greatly affect performance and choosing suitable set-up for a given problem is of great importance. We now discuss some of the options available in choosing the quantum model.

Feature Maps

As described in section 2.4.1 a quantum feature map is a way to embed dependence on an independent variable x into a quantum model. It consists of a circuit $\hat{\mathcal{U}}_\varphi(x)$ and x is a parameter of that circuit. The choice of feature map effectively chooses the set of fitting basis available. In our algorithm any could be used

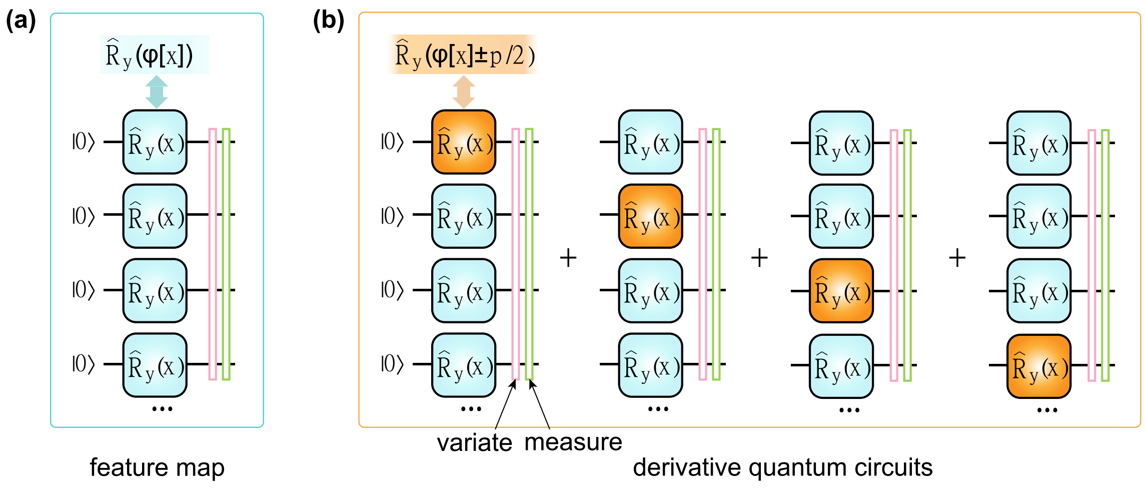


Figure 3.2: **Product feature map and its derivative.** (a) Quantum feature map of a product type, where single qubit rotations (here chosen as $\hat{R}_y$) act at each qubit individually and are parametrised by a function of variable x. Specifically, the expectation value of the circuit is shown, with thin pink and green blocks depicting the variational ansatz and the cost function measurement respectively. For the nonlinear feature encoding the nonlinear function $\varphi(x)$ is used as an angle of rotation. The product feature map can be further generalised to several layers, and different functions $\{\varphi\}$. Several feature maps can be concatenated to represent a multivariable function. (b) shows the derivative quantum circuit for the product feature map. Differentiation over variable x follows the chain rule, with the expectation value of the derivative written as a sum of separate expectations with shifted phases, repeated for each x-dependent rotation.

(provided derivatives can be found) but in particular we discuss product feature maps and the type of fitting basis they lead to. We also introduce the Chebyshev product basis.

Product feature maps. A product feature map is a type of feature map which consists of a single layer of rotations on each qubit. The angle of rotation is a (possibly non-linear) function of x, $\varphi[x]$.

$$\hat{\mathcal{U}}_{\varphi}(x) = \bigotimes_{j=1}^{N} \hat{R}^{j}_{\alpha}(\varphi[x]), \tag{3.2}$$

where N is the number of qubits. $\hat{R}^{j}_{\alpha}(\varphi[x])$ denotes a rotation on qubit j in basis α (often chosen in Pauli basis but not restricted to) of angle $\varphi[x]$. This represents the feature map choice used in quantum circuit learning [28], and several similar encodings were discussed in [113, 29], and reviewed in [114]. An example with $\varphi(x) = \arcsin x$ and $\alpha = Y$ is shown in Fig. 3.2(a). For this example the unitary operator Eq. (3.2) is expanded as

$$\hat{\mathcal{U}}_{\varphi}(x) = \bigotimes_{j=1}^{N} \exp\left(-i\frac{\arcsin x}{2}\hat{Y}_j\right), \tag{3.3}$$

leading to amplitudes that depend on the encoded variables as $\cos[(\arcsin x)/2]$ and $\sin[(\arcsin x)/2]$. Acting on the initial state $|0\rangle$ this feature map encodes the variable as an N^{th} degree polynomial formed by $\{x, \sqrt{1-x^2}\}$ and their products [28]. Note that these functions come from after the expectation value has been measured, to which the feature map contributes twice. The redundancy from many qubits thus forms a basis set for function fitting [113].

As we are solving differential equations, derivatives with respect to x must be able to be calculated. Here we make use of the parameter shift rule for efficient evaluation of exact gradients as explained in section 2.4.4. Generally in a feature map x is a parameter of more than one gate, therefore the product rule for derivatives is used. The corresponding derivative quantum circuits for the example are shown in Fig. 3.1(b) and Fig. 3.2(b). Alternatively, finite difference methods (FDM) could be used. This would be an approximate derivative but depending on FDM chosen could result in fewer measurements needed.

To generalise the product-type feature map, we can consider cases where the encoding function $\varphi(x)$ depends on qubit number j as $\varphi_j(x)$. This leads to a wider range of basis functions and therefore generally increased expressivity. We refer to this as a tower feature map.

Chebyshev product feature maps. Next, we consider a distinct choice of nonlinear quantum feature map that we name the *Chebyshev product feature map*. Belonging to the product feature map family, it drastically changes the basis set for function representation. As a building block we use a single qubit rotation $\hat{R}^j_y(\varphi_j[x])$ with $\varphi_j(x) = 2n[j]\arccos(x)$ where $n[j]$ is a function mapping qubit number to an integer, such that the encoding circuit reads

$$\mathcal{U}_\varphi(x) = \bigotimes_{j=1}^{N} \hat{R}_y^j(2n[j]\arccos(x)). \tag{3.4}$$

To see the fitting basis this results in we first expand the rotation using Euler's formula, getting

$$\begin{aligned} R_y^j(\varphi_j[x]) &= \exp\left(-i\frac{2n[j]\arccos(x)}{2}\hat{Y}_j\right) \\ &= \cos(n[j]\arccos(x))\mathbb{1}_j - i\sin(n[j]\arccos(x))\hat{Y}_j \\ &= T_{n[j]}(x)\hat{\mathbb{1}}_j + \sqrt{1-x^2}U_{n[j]-1}(x)\hat{X}_j\hat{Z}_j, \end{aligned} \tag{3.5}$$

where $T_n(x)$ and $U_n(x)$ respectively denote degree-n Chebyshev polynomials of first and second kind. The Chebyshev polynomials and their properties were introduced in section 2.8.4. By using this feature map the fitting basis is in terms of Chebyshev polynomials with their fitting properties.

In the present study we consider two types of Chebyshev product feature maps. The first version corresponds to using $n[j] = 1$ for all j defined as

$$\mathcal{U}_\varphi(x) = \bigotimes_{j=1}^{N} R_y^j(2\arccos x). \tag{3.6}$$

When using this with N multiple qubits, the tensor product of the qubits results in multiplications of the Chebyshev polynomials which when using Chebyshev product rules results in fitting basis of Chebyshev polynomials up to degree N.

The second version we consider corresponds to a *Chebyshev tower feature map* defined with $n[j] = j$ as

$$\mathcal{U}_\varphi(x) = \bigotimes_{j=1}^{N} R_y^j(2j\arccos x), \tag{3.7}$$

where the encoded degree grows with the number of qubits, creating a tower-like structure of polynomials with increasing $n = j$. The product rule again results in higher order Chebyshev polynomials but with different order starting basis the highest order now available is the sum of integers up to N which is equal to $N(N+1)/2$. This leads to higher expressivity without increasing system size or number of gates.

Variational Ansatz

Another still open choice for our quantum model is the variational ansatz. This is again open to free choice provided gradients can be calculated. The variational ansatz and some common choices are discussed in section 2.4.2.

Differential equations in general do not have symmetries that can be exploited for symmetry or geometric ansatze. Therefore we consider hardware efficient ansatze as in Fig. 2.5. By considering these types of ansatze the algorithm remains suitable for near-term quantum computing.

Cost Function

The final choice for the quantum model is the cost function C as our quantum model structure is an expectation value. Cost functions are introduced in section 2.4.3. We mainly consider local cost functions due to the delayed onset of barren plateaus when considering them. We also note that when choosing $C = \sum_j \alpha_j C_j$, $\{\alpha_j\}_j$ can be considered static coefficient or a set of variational parameters to be optimised along with θ during the optimisation loop.

3.3.3 Loss Function

As with all variational algorithms training trial solutions to solve a problem we need to proved a measure of how well the current function solves the problem. This measure is the loss function as intoduced in section 2.2. The classical optimiser can then update the variational parameters to reduce this ‘distance’. For this problem our loss is taken as the sum of the ’distance’ between the residue of the DE and 0 at a set of chosen training points. The initial value/boundary can also be considered within the loss (see later section on boundary handling).

We can write a loss function parametrised by variational parameters $\boldsymbol{\theta}$ for a differential equation in the general form

$$\mathcal{L}_{\boldsymbol{\theta}}^{(\mathrm{diff})}[d_x f, f, x] = \frac{1}{M}\sum_{i=1}^{M} L(DE[d_x f(x_i), f(x_i), x_i], 0) \tag{3.8}$$

with $L(a, b)$ being a function describing how the distance between the two arguments a and b is being measured. The loss is estimated on a grid of M points, and is normalised by the grid size. Functional DE corresponds to the differential equation residue written in the form $DE[d_x u, u, x] = 0$. It can be evaluated by combining values of f and $d_x f$ at the training grid points.

The choice of distance definitions L alters the loss landscape and dictates how the optimiser perceives the distance between vectors. This makes the loss distance measure another hyperparameter that when chosen can affect convergence. A standard measure to use is the *mean square error* (MSE) introduced as

$$L(a, b) = (a - b)^2. \tag{3.9}$$

While being simple, we find the choice (3.9) intuitive and performing sufficiently well in numerical simulations. Another common measure is *mean absolute error* (MAE) loss defined with distance $L(a, b) = |a - b|$. Finally, several more complex metrics can be used, including variants of Kullback-Leibler (KL) divergence [115] and Jensen–Shannon divergence [116].

Regularisation

Given that our goal is to find a variational spectral representation of the differential equations solution using large basis sets, the optimisation procedure benefits from having a good initial guess or having access to any knowledge about expected behaviour. This helps the optimiser to avoid local minima. One way to achieve this is regularisation.

There are many methods of regularisation. We consider one of the simplest, bias the optimiser to favour specific behaviour/form by providing an additional regularisation loss term. This is constructed by considering a set of regularisation points $\{x_{\text{reg}}\}_{r=1}^{R}$ and regularisation values $\{u_{\text{reg}}\}_{r=1}^{R}$ as

$$\mathcal{L}_{\boldsymbol{\theta}}^{(\text{reg})}[f, x] = \sum_{r=1}^{R} \zeta(n_j) L\Big(f_{\boldsymbol{\theta}}(x_{\text{reg},r}) - u_{\text{reg},r}\Big). \tag{3.10}$$

L is again a distance measure. This loss term therefore leads the optimiser to try to fit the trial solution to the regularisation terms which describe expected behaviour/shape/any other known information. For example if it is known that the solution should overall increase, regularisation values which increase could be given.

As the regularisation values given are a guide but not the exact value wanted a weighting factor of $\zeta(n_j)$ proceeds the regularisation term where n_j is the epoch number. To prevent fitting to the inexact regularisation values this term should start large for low epoch number then decrease to/near zero as epoch number increases. This gives the optimiser motive to find a solution with the correct general behaviour/shape to start with. Then as the regularisation loss term falls the trial solution should be a good inital guess and the focus goes to the DE term.

Some example choices of regularisation weighting include linearly decreasing regularisation weight, $\zeta(n_j) = 1 - n_j/n_{\text{iter}}$ where n_{iter} is the maximum iteration number. This strategy works for small learning rates and large number of iterations, such that the optimiser has sufficient “time” to adjust to the constantly changing loss landscape. Another choice corresponds to a *reverse sigmoid optimisation schedule*, where a smooth drop of regularisation weight is performed at pre-defined training stages. We parametrise this schedule as

$$\zeta(n_j) = 1 - \tanh\left(\frac{n_j - n_{\text{drop}}}{\delta_j n_{\text{iter}}}\right) \tag{3.11}$$

where n_{drop} denotes the iteration step number at which regularisation weight drops, and δ_j assigns the transition rate. This allows the DQC to initially focus almost entirely on the regularisation optimisation, later switching the focus towards the gradient optimisation.

Variations of this regularisation term exist such as regularising derivatives instead of function value. Alternatively, the term could be set up to avoid certain values/behaviours instead of favouring. E.g. if it was known that the solution should never be 0 a term which diverges if $f_{\boldsymbol{\theta}}(x_{\text{reg},r})$ is 0 would discourage the optimiser from that area.

3.3.4 Boundary Handling

As our goal is to construct a quantum circuit that satisfies a system of differential equations, together with matched derivatives we need to ensure that an initial value or boundary value problem is solved. Generally, this corresponds to fixing the function value at a required initial point or a collection of boundary points, thus resembling the quantum circuit learning tasks considered in Ref. [28]. At the same time, there are several ways how the DQC-based function $f_{\boldsymbol{\theta}}(x)$ can be constructed, leading to varying performance and specific pros/cons when solving particular problems.

Information about the boundary can be included as part of the loss function. A boundary part of the loss can be written in the form

$$\mathcal{L}_{\boldsymbol{\theta}}^{(\text{boundary})}[f, x] = \eta \sum_j L\Big(f_{\boldsymbol{\theta}}(x_0^j) - u_0^j\Big), \tag{3.12}$$

where L is a distance measure, $\{x_0^j\}_j$ represents the set of boundary points (or an initial point), $\{u_0^j\}_j$ is a vector of boundary values, and η is a pinning coefficient which determines the emphasis the optimiser should place on boundary fitting versus DE fitting.

Pinned boundary handling. The first option is to include the information about the boundary in the expectation of the cost function, then trained with the boundary loss term. This corresponds to simply choosing a cost operator $\hat{C}$, and representing the solution in the form

$$f(x) = \langle f_{\varphi,\boldsymbol{\theta}}(x)|\hat{C}|f_{\varphi,\boldsymbol{\theta}}(x)\rangle. \tag{3.13}$$

The initial value u_0 is then matched via the boundary term in the loss function. The strength of the *pinned boundary* handling is in equivalent treatment of boundary and derivative terms, both being encoded in the eigenspectrum of $\hat{C}$. At the same time, the weakness corresponds to the necessity of adjusting the boundary value starting from the one represented by initial $\boldsymbol{\theta}_{\mathrm{init}}$, typically generated randomly. This can be adjusted by shifting $f(x)$ by a constant-times-identity term added to the cost operator, $\hat{C} = \alpha_0 \mathbb{1} + \sum_{j=1}^{M} \alpha_j \hat{C}_j$, where α_0 is set such that for $\boldsymbol{\theta}_{\mathrm{init}} \sim \mathrm{random}[0, 2\pi]$ the function $\langle f_{\varphi,\boldsymbol{\theta}_{\mathrm{init}}}(x)|\hat{C}|f_{\varphi,\boldsymbol{\theta}_{\mathrm{init}}}(x)\rangle$ typically lies close to u_0 value when evaluated at $x = x_0$. Another possible weakness is that the boundary and differential terms in the loss may compete against one another.

Optimised boundary handling. Alternatively, we also propose a boundary handing technique that relies on a classical shift of the solution, but defined by the gradient descent procedure on par with variational angles optimisation. This removes the need to include boundary information in the cost expectation, but information still needs to be included in the loss function. Namely, we seek for the solution in the form

$$f(x) = f_c + \langle f_{\varphi,\boldsymbol{\theta}}(x)|\hat{C}|f_{\varphi,\boldsymbol{\theta}}(x)\rangle, \tag{3.14}$$

where $f_c \in \mathbb{R}$ is a variational parameter alongside the quantum ansatz angles and updated accordingly via the classical optimiser. Therefore, the gradients for f_c have to be calculated additionally when using this boundary handler. One strength of the described method is that, due to the classical shift, even if the random initial angles start such that $\langle f_{\varphi,\boldsymbol{\theta}_{\text{init}}}(x_0)|\hat{C}|f_{\varphi,\boldsymbol{\theta}_{\text{init}}}(x_0)\rangle$ is far from the initial value u_0, the optimiser can quickly and easily update f_c to rectify this. However, this approach could experience the same weakness as the pinned approach and end up with boundary and loss terms competing.

Floating boundary handling. The boundary does not have to be considered within the loss. Instead this choice of the boundary handler corresponds to iteratively shifting the estimated solution based on the boundary or initial point. For this method the boundary information does not require a separate boundary loss term nor is it encoded in the expectation of the cost function. Instead it is set iteratively within the parametrisation of the function. As the function is parametrised to match a specific boundary, information about the boundary is still contained within the function and its derivatives. Therefore boundary information is still present within the loss function despite there not being a separate boundary loss term. We parametrise the function as

$$f(x) = f_{\text{b}} + \langle f_{\varphi,\boldsymbol{\theta}}(x)|\hat{C}|f_{\varphi,\boldsymbol{\theta}}(x)\rangle, \tag{3.15}$$

with $f_{\text{b}} \in \mathbb{R}$ being a parameter adjusted after each iteration step as

$$f_{\text{b}} = u_0 - \langle f_{\varphi,\boldsymbol{\theta}}(x_0)|\hat{C}|f_{\varphi,\boldsymbol{\theta}}(x_0)\rangle. \tag{3.16}$$

This effectively allows the solver to find a function $\langle f_{\varphi,\boldsymbol{\theta}}(x)|\hat{C}|f_{\varphi,\boldsymbol{\theta}}(x)\rangle$ which solves the differential equation shifted to any position, then being shifted to the desired initial condition as shown in Eq. (3.15). This method of boundary handling guarantees exact matching to initial values given and does not require a separate boundary term in the loss function, thus the derivative loss term does not have to compete with the boundary loss. Furthermore, as we allow the cost function to match to the solution shifted by any amount, this simplifies the choice for optimal angles and removes the dependence on initial θ_{init}. However this method does require knowledge of an exact initial value which can be an issue in specific situations.

3.3.5 Generalisation

Most of the discussion so far has used notation/terms so far as if the DE considered is a single, first order, one dimension DE. Many DE of interest will not be of this form and instead of a more general form. Here we discuss how to generalise the algorithm.

Higher Order DEs. A simple generalisation to make. The residue of the differential equation and therefore the loss function will now require evaluation of all the different order of derivatives present in the differential equation. This can be done with succesive applications of the parameter shift rule.

System of DEs. When solving a *system of* differential equations we need to decide how multiple functions should be encoded simultaneously. There are several ways that this can be achieved. The first approach we consider is to use the same quantum register for all functions, thus compressing information about the function vector using the same feature map circuit and variational ansatz. The functions are then defined through the choice of different cost operators at the readout stage. This method is resource-frugal, and is suitable for certain systems. However, the choice of suitable cost operators becomes complicated, as in some cases shared register encoding may exhibit competition between the optimisation of different functions. This concerns the question of expressibility of the set of cost operators, and may potentially be solved using a weighted sum of operators with optimised weights.

A second option is to use separate quantum registers, and correspondingly a different feature map and variational ansatz for each function. This removes the issue of choosing the cost operators, and avoids the related direct competition due to independent parametrisation. While requiring more resources we note that function and derivative evaluation can be done in parallel. However, we note that as a combined loss function is considered care must be taken when timing the circuit runs.

No matter how the multiple functions are encoded the information of the system of DEs is encoded into the optimisation loop via the loss function. Each DE forms its own loss term as in (3.8) which are then summed for the overall differential loss. It is possible for each DE loss term to have a different choice of L and to have different scaling coefficients however generally it is preferred to keep them the same. If they are kept the same the optimiser priorities all DEs in the system the same which is often what's wanted.

Multidimensional DEs. To solve multidimensional DEs the quantum model of the trial solution must contain dependence on all of the variables. This can be done in any way provided all relevant derivatives and values can still be evaluated. How it is done will affect the expressivity and trainability of the model.

One possible choice is to layer separate feature maps for each variable along with variational ansatze – reminiscent of data re-uploading. Or one feature map could be used which uses contributions from all variables in determining its parameters. The second choice has a wider range of possible expressivity – especially if used in conjunction with data re-uploading. The first choice, by keeping contributions from each variable separate, allows for easier control of how each variable is encoded and what behaviour/dependence the trial solution is likely to exhibit when varying that variable. e.g. it would be easier to bias one variable to be periodic and another symmetric.

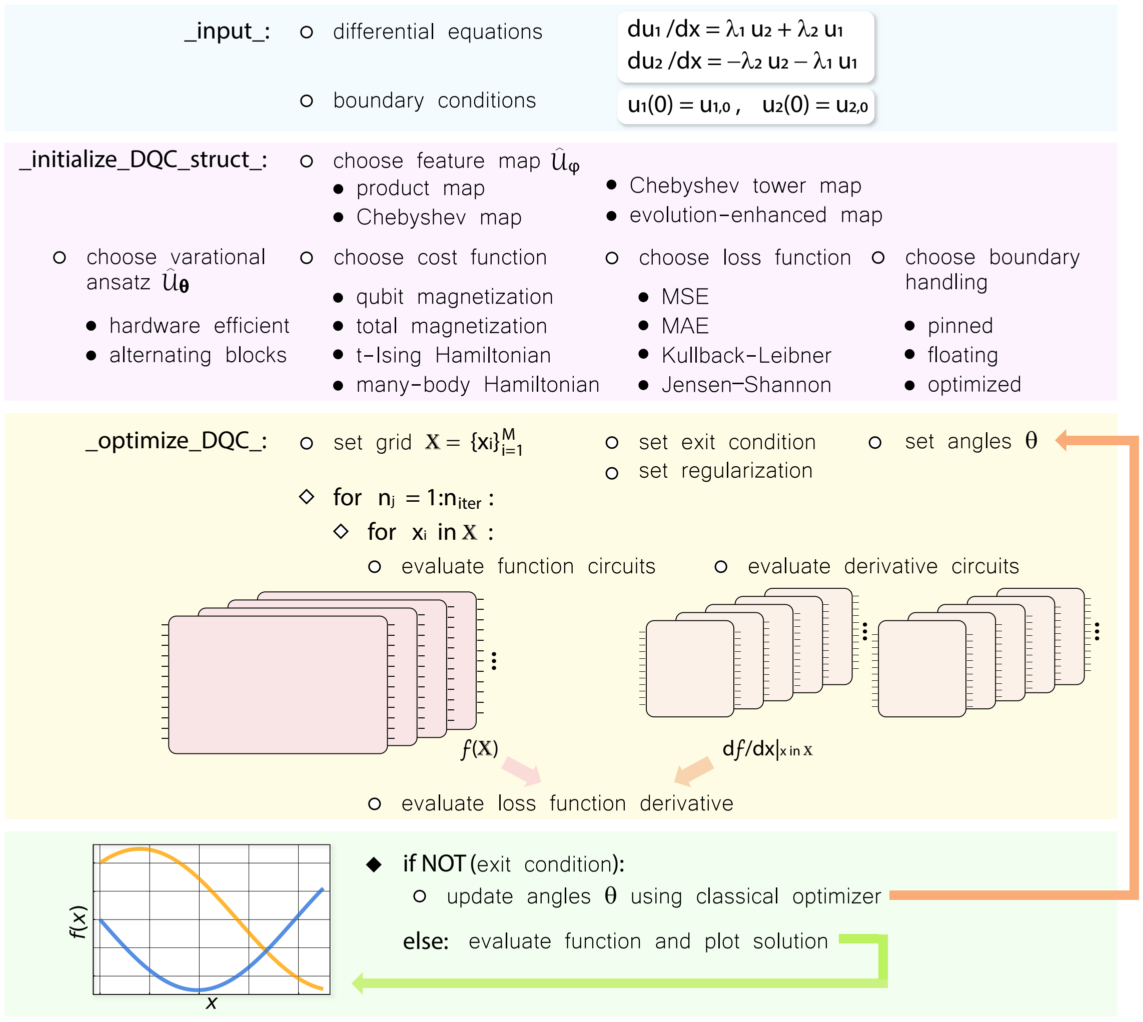


Figure 3.3: **DQC optimisation workflow.** The problem is set as a system of differential equations for functions u, variables x, and specified boundary conditions. The derivative quantum circuit is constructed by choosing the encoding circuit and optimisation schedule. The solution is optimised by evaluating the function and derivative circuits at the defined grid of points $\mathbb{X}$ (which may be multidimensional), and using these values to calculate the loss function derivative. The variational angles are updated in the hybrid quantum-classical loop until a specified exit condition is reached.

Alternatively, not all variables have to be encoded via feature map. A possible technique is, if there is one ”evolution” parameter of importance – generally time – that the other variables are encode via feature map but that the evolution parameter t is included as a dependence of the variational parameters $\theta(t)$. i.e. $\theta(t)$ are no longer static and change as the solution evolves. This comes with more choices, learning a set of theta for each considered point of t and evaluating derivatives from chain rule $d\theta/dt$ via finite difference or by constructing a model for $\theta(t)$ which are trained as well. As well as this approach variables can be encoded within cost function.

As can be seen by this brief overview encoding multidimensional variables is something easily possible but not at all trivial to work out the best way to do it. Research into this is still ongoing.

3.3.6 Workflow

Finally, using the elements and strategies described above, we present a workflow for constructing the differential equation solver based on derivative quantum circuits. This is summarised in Fig. 3.3 showing the flowchart. We start by specifying the input for the solver. This comprises the problem in hand, specified as a set of nonlinear differential equations of various types, together with their respective boundary conditions. Additionally, a set of regularisation points may be added to ensure the optimised solution is chosen in the desired qualitative form. Next, we set up the schedule for derivative quantum circuit optimisation and choose the quantum circuit composition. For this we choose: a) the type of quantum feature map; b) the ansatz of variational quantum circuit, including its depth; c) the cost function type, also choosing if variational weights are considered; d) the type of the loss function; e) the strategy to match the boundary terms and derivatives. We also need to specify the classical optimiser for variational angles and weights (with associated hyperparameters), including the number of iterations and exit conditions. Finally, we specify whether the loss function uses a specific optimisation schedule, where it is changing during the training.

Once the DQC structure and optimisation schedule are defined, we need to specify a set of points $\{x_j\}_j$ for each equation variable. This can be a regular equidistant grid, Chebyshev grid, or a randomly-drawn grid. The variational parameters are set to initial values $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_{\text{init}}$ (e.g. as random angles). The expectation value over variational quantum state $|u_{\varphi,\boldsymbol{\theta}}(x_i)\rangle$ for the cost function is estimated using the quantum hardware for the chosen point x_i. Then a potential solution at this point is constructed, accounting for the boundary handling procedure. Next, the derivative quantum circuits are constructed and their expectation value is estimated for the specified cost function, at point x_i. This is repeated for all x_i in $\mathbb{X}$ where $\mathbb{X}$ is the chosen training grid. We then collect these function values and derivatives, and compose the loss function for the entire grid and system of equations (forming required polynomials and cross-terms by classical post-processing). Regularisation points may be also added, biasing the solution to take specific values at these points. The goal of the loss function is to assign a "score" to how well the potential solution (parametrised by the variational angles $\boldsymbol{\theta}$) satisfies the differential equation, matching derivative terms and the function polynomial to minimise the loss. With the aim to increase the score (and decrease the loss function), we compute the gradient of the loss function with respect to variational parameters $\boldsymbol{\theta}$. Using the gradient descent procedure (or in principle any other classical optimisation procedure) we update the variational angles from iteration n_j into the next one $n_j + 1$, and repeat the steps outlined before until we reach the exit condition. The exit condition may be chosen as: 1) the maximal number of iterations n_{iter} reached; 2) loss function value is smaller than pre-specified value; and 3) loss gradient is smaller than a certain value. Once we exit the classical loop, the solution is chosen as a circuit with angles $\boldsymbol{\theta}_{\mathrm{opt}}$ that minimise the loss. Finally, we extract the full solution by sampling the cost function for optimal angles $\langle u_{\varphi,\boldsymbol{\theta}}(x)|\hat{C}|u_{\varphi,\boldsymbol{\theta}}(x)\rangle$. Notably, this can be done for any point x, as DQC constructs the solution valid also between the points at which loss is evaluated originally.

3.4 Results

3.4.1 Linear Differential Equation

Now let us see how the algorithm performs in practice. For this we choose a differential equation with a known analytical solution, and compare it to the one obtained by the derivative quantum circuit. We choose a single ODE for the initial value problem which reads

$$\frac{du}{dx} + \lambda u \left(\kappa + \tan(\lambda x)\right) = 0, \quad u(0) = u_0, \tag{3.17}$$

where λ and κ are parameters, and u_0 sets the value of the function u at $x = 0$. Eq. (3.17) has a solution in the form of a damped oscillating function,

$$u(x) = \exp(-\kappa\lambda x)\cos(\lambda x) + const, \tag{3.18}$$

where $const$ is determined by the initial condition. While the problem is fairly simple being a single ODE, reproducing the damped oscillating solution requires a rich basis of fitting functions, that needs to include both oscillatoric and increasing/decreasing functions. As λ and κ grow the function starts to oscillate and decay even more rapidly, and the solution becomes harder to express.

To show how the proposed method works, we use differentiable quantum circuits to solve Eq. (3.17) using optimisation of differentiable quantum feature maps. Specifically, we choose two cases with parameters $\lambda = 8$ and $\lambda = 20$, and fixed $\kappa = 0.1$, $u_0 = 1$. These problem parameters are chosen to make DQC construction challenging, with $\lambda = 20$ being a complex case as the resulting solution is highly nonlinear and oscillatoric. We consider an equidistant optimisation grid of 20 points, starting from $x = 0$, with maximal time of 0.9 (dimensionless units are used). This is chosen such that the region with diverging derivative of the nonlinear feature function $d\varphi(x)/dx$ is avoided, and we note that x can be rescaled to match required boundaries. To find the solution we use a quantum register with $N = 6$ qubits, and the cost function is chosen as total magnetisation in the Z direction, $\hat{C} = \sum_{j=1}^{N} \hat{Z}_j$. For the variational circuit, we choose the standard hardware-efficient ansatz described in the Methods section, setting the depth to $d = 5$. To search for optimal angles $\boldsymbol{\theta}_{\mathrm{opt}}$ we perform adaptive stochastic gradient descent using Adam [117] with automatic differentiation enabled by analytical derivatives. Specifically, we code the workflow using Yao.jl package [110, 118, 119] for Julia, which allows fast and efficient implementation. A full quantum state simulator is used in a noiseless setting. In this example we use the *floating boundary* handling.

We search for the circuit-based solution using three different feature maps described in the previous section, and compare their performance. These correspond to the *product feature map* [Eq. (3.3)], the sparse version of the *Chebyshev product feature map* [Eq. (3.6)], and the *tower Chebyshev feature map*, as defined in Eq. (3.7). We label results for these feature maps as Prod, Cheb, and ChebT, respectively. To assess the performance we use several metrics. The first metric is the *full loss* (denoted as L_F). It refers to the loss calculated from the differential equations and any boundary or regularisation terms, in this case $\mathcal{L}_{\boldsymbol{\theta}}[d_x f, f, x] = \mathcal{L}_{\boldsymbol{\theta}}^{(\mathrm{diff})}[d_x f, f, x] + \mathcal{L}_{\boldsymbol{\theta}}^{(\mathrm{boundary})}[f, x] + \mathcal{L}_{\boldsymbol{\theta}}^{(\mathrm{reg})}[f, x]$. The second metric corresponds to *differential loss* (L_D), being a part of the full loss excluding regularisation contribution. Finally, the third metric is the *quality of solution* (L_Q). The quality of the solution is the distance of the current DQC-based solution from the known true solution. This is calculated by evaluating the DQC-based solution and true solution at a set of points and using the MSE loss type, being equal to $L_Q = (1/M)\sum_{i=1}^{M}[f(x_i) - u(x_i)]^2$. Quality of solution gives us a useful way to compare how two different training setups perform, especially if they are training to solve the same differential equations.

The results of DQC training are shown in Fig. 3.4. In the panel Fig. 3.4(a, c) we show the solutions of Eq. (3.18) for $\lambda = 8$ and $\lambda = 20$, respectively, where solid curves (with label 1) represent the analytical solution $u(x)$ in Eq. (3.18). The dashed curves represent the final DQC-based solutions sampled from the cost function at approximation points ($n_{\mathrm{iter}} = 250$ is used). In Fig. 3.4(b,c) we show the relevant training metrics as a function of iteration number, where solid curves denote the *quality of solution* and dashed curves represent the *full loss*.

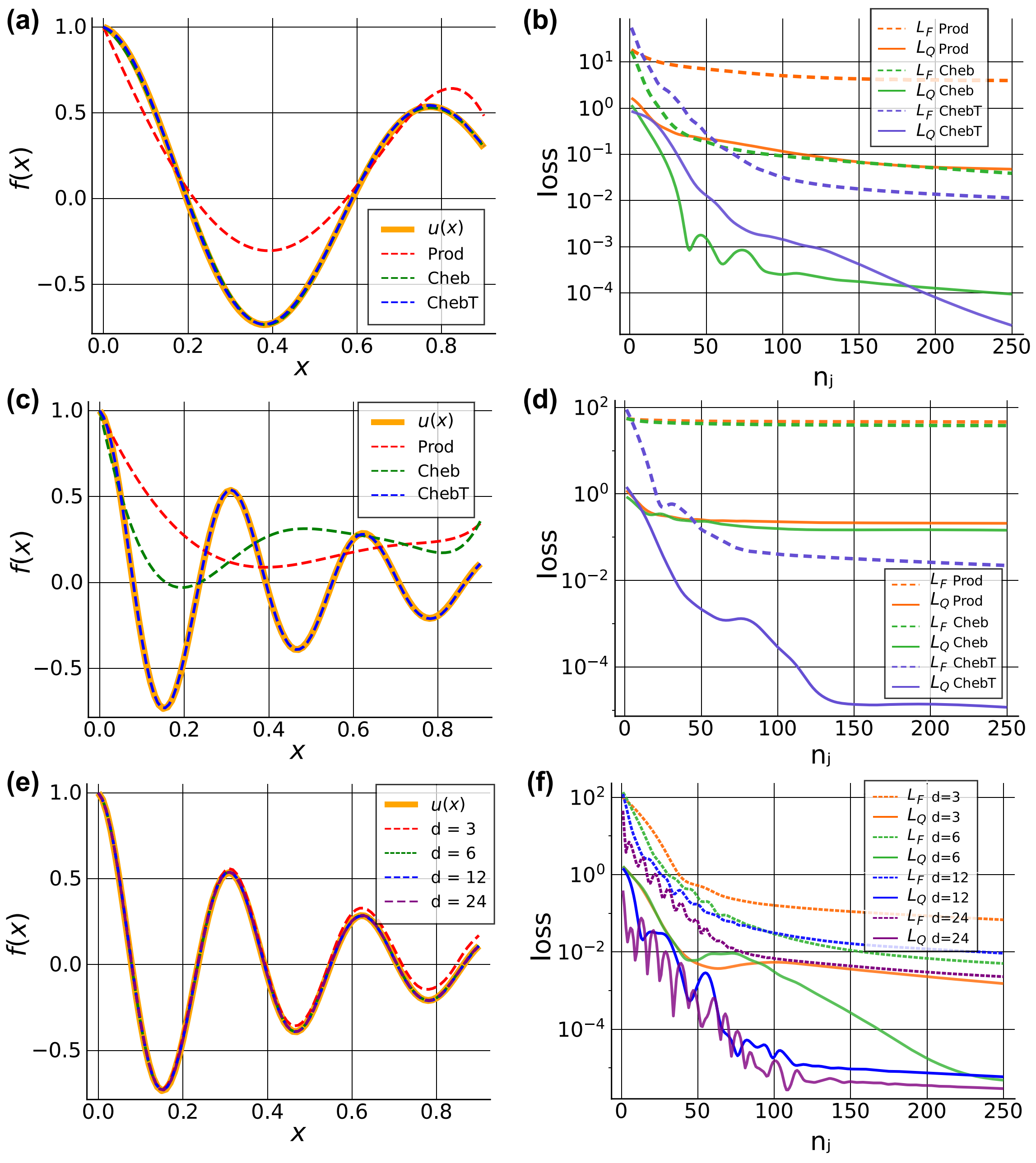


Figure 3.4: **DQC-based solution for a single ODE example.** (a) Results from the circuit trained to solve Eq. (3.17) for $u_0 = 1$, $\lambda = 8$, and $\kappa = 0.1$. We present DQC solutions $f_{\varphi,\theta}(x)$ obtained using three different quantum feature maps $\varphi(x)$ labelled as Prod, Cheb, and ChebT. Corresponding DQC solutions are shown by dashed curves and labelled for clarity (see legend). Analytical solution $u(x)$ is shown by the thick solid curve (label 1). (c) Same as in (a), but for $\lambda = 20$ example. (b, d) *Full loss* L_F (dashed curves) and *quality of solution* L_Q (solid curves) are shown as functions of iteration number n_j for optimisation results displayed in (a) and (c), respectively. (e) DQC-based solution of Eq. (3.17) shown for four different ansatz depths $d = 3, 6, 12, 24$, with the analytical solution $u(x)$ presented by the solid curve. The Chebyshev tower feature map is used. (f) Full loss (dotted curves) and quality of solution (solid curves) shown as a function of iteration number for the solution in (e).

We observe that for $\lambda = 8$ both Chebyshev feature maps converge closely to the true solution [Fig. 3.4(a,d)]. The more expressible *Chebyshev tower feature map* (curves 5, 6) takes longer to converge but reaches a solution closer to the true solution. The less powerful *product feature map* fails to converge with the loss quickly plateauing.

For $\lambda = 20$ the true solution is more oscillatoric and has stronger damping, making the solution harder to represent [Fig. 3.4(b,e)]. The product feature map still fails to converge (curves 1, 2), but now also failing to converge is the simpler Chebyshev feature map (curves 3, 4). The full loss for both cases plateaus rapidly. The more expressible ChebT feature map continues to perform well (curves 5, 6). This supports the hypothesis that choosing a feature map expressible enough for the problem is important, and more simulations with more qubits offers a way to increase the power drastically.

Next, we compare the effect of ansatz depths for the variational circuit $\hat{\mathcal{U}}_{\theta}$. We use $d = 3, 6, 12, 24$, $\lambda = 20$ and the *Chebyshev tower feature map*, and the rest of the training set up remains the same as previously considered. These results are presented in Fig. 3.4(e,f). We observe that for lower depths the solver is slower to converge and does not reach as high accuracy as it does for higher depths. As depth increases more layers of parametrised gates are included in the variational ansatz and so the number of variational angle parameters increase. This causes an increase in the number of gate operations needed in each iteration and how many parameters the classical optimiser needs to update, raising the time taken per iteration. As the depth of the ansatz continues to increase eventually the problem of barren plateaus could be encountered [52]. Then vanishing gradients would cause the solver to struggle to improve the parameters, however at $d = 24$ we had not yet ran into this with over 400 variational parameters. We also note that the alternating blocks ansatz is designed in the way that vanishing gradients can be avoided for certain conditions [120].

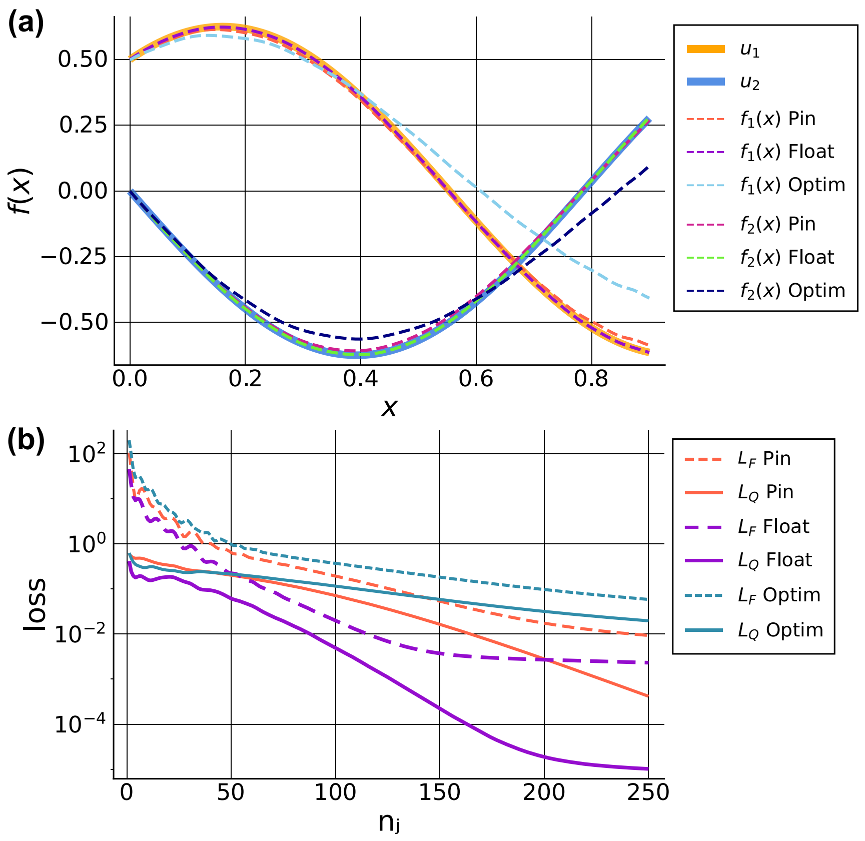


Figure 3.5: **DQC-based solution for a strongly coupled equations example.** (a) Results from the circuit trained to solve the system of differential equations Eq. (3.19) and Eq. (3.20) for $u_{1,0} = 0.5$, $u_{2,0} = 0$, $\lambda_1 = 5$, and $\lambda_2 = 3$. We present DQC solutions $f_1(x)$ and $f_2(x)$ obtained using three different boundary evaluation techniques: pinned boundary (labelled as Pin), floating boundary (Float), and optimised boundary (Optim). Corresponding DQC solutions are shown by dashed curves. Analytical solutions $u_1(x)$ and $u_2(x)$ are shown by thick solid curves. (b) *Full loss* $\mathcal{L}_{\mathrm{F}}$ (dashed curves 1, 3, 5) and *quality of solution* $\mathcal{L}_{\mathrm{Q}}$ (solid curves 2, 4, 6) are shown as functions of iteration number n_j for optimisation results displayed in (a).

3.4.2 Strongly Coupled Equations

Building up on the single ODE example, we proceed to consider a *system* of differential equations, taking two strongly coupled differential equations as an example. This describes the evolution of competing modes $u_1(x)$ and $u_2(x)$ as a function of variable x, which in this case corresponds to time. The associated rate equations read

$$F_1[d_x\boldsymbol{u}, \boldsymbol{u}, x] = \frac{du_1}{dx} - \lambda_1 u_2 - \lambda_2 u_1 = 0, \quad u_1(0) = u_{1,0}, \tag{3.19}$$

$$F_2[d_x\boldsymbol{u}, \boldsymbol{u}, x] = \frac{du_2}{dx} + \lambda_2 u_2 + \lambda_1 u_1 = 0, \quad u_2(0) = u_{2,0}, \tag{3.20}$$

where $\lambda_{1,2}$ are coupling parameters, $u_{1,0}$, $u_{2,0}$ are initial conditions. The larger $|\lambda_1|$ is in comparison to $|\lambda_2|$ the more strongly coupled the two equations will be. This can be intuitively seen considering $|\lambda_1| \geq |\lambda_2|$, leading to larger contribution of u_2 into the equation for du_1/dx derivative than u_1, and vice versa.

To move from considering one differential equation to a system of equations we need to encode multiple functions using quantum registers as described in the Methods. For this specific example we choose a simple parallel encoding, where separate cost functions and ansatze are considered. In this case each function has a separate set of parameters to be optimised. The loss is changed accordingly to include information on separate contributions from two coupled differential equation that are optimised simultaneously.

We encode each function using the differentiable feature map combined with individual variational ansatz parametrised by the set of angles $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ and before deciding on the boundary evaluation type we have

$$f_1(x) = \langle 0|\hat{\mathcal{U}}^{\dagger}_{\phi}(x)\hat{\mathcal{U}}^{\dagger}_{\boldsymbol{\theta}_1}\hat{C}^{(1)}\hat{\mathcal{U}}_{\boldsymbol{\theta}_1}\hat{\mathcal{U}}_{\phi}(x)|0\rangle, \tag{3.21}$$

$$f_2(x) = \langle 0|\hat{\mathcal{U}}^{\dagger}_{\phi}(x)\hat{\mathcal{U}}^{\dagger}_{\boldsymbol{\theta}_2}\hat{C}^{(2)}\hat{\mathcal{U}}_{\boldsymbol{\theta}_2}\hat{\mathcal{U}}_{\phi}(x)|0\rangle, \tag{3.22}$$

where $\hat{C}^{(1,2)}$ are in principle different cost functions for each equation.

For the loss function we consider the sum of the MSE losses for the first and second differential equation. This loss is written as $\mathcal{L}_{\boldsymbol{\theta}}[d_x\boldsymbol{f},\boldsymbol{f},x] = \sum_{i=1}^{M} L(F_1[d_x\boldsymbol{f},\boldsymbol{f},x_i],0)+$ $\sum_{i=1}^{M} L(F_2[d_x\boldsymbol{f},\boldsymbol{f},x_i],0)$ with F_1 and F_2 as written in Eq. (3.19)-(3.20), and L depends on the loss choice as detailed in the Methods section. There can be additional boundary loss terms depending on boundary evaluation method chosen. If present, they contribute to the loss function as a sum of the boundary terms for u_1 and u_2. Note that we consider the loss as a sum of individual contributions coupled together we are trying to minimise both simultaneously with equal weight. Due to the coupling between the two equations this could lead to competition between the two loss terms (a parameter update which causes a loss decrease for one DE's loss may lead to an increase in the other). This may result in increasing the chance of converging to a local minimum rather than the global minimum; however, this effect can be mitigated if loss contributions are weighted in some way. Another solution is to use quantum kernel methods, where loss corresponds to the overlap between quantum feature states. Choosing a suitable loss function in this case is an important point to consider in the future.

We define the problem setting parameters to $\lambda_1 = 5,\ \lambda_2 = 3$ and initial conditions to $u_{1,0} = 0.5,\ u_{2,0} = 0$. We set up the training scheme as in section 3.4.1 using a six-qubit register, cost choice of total magnetisation in the Z direction for both u_1 and u_2, hardware-efficient variational ansatz with depth $d = 5$, Adam optimiser with learning rate 0.02, and feature map choice of the *Chebyshev tower* feature map. We test the performance for the three boundary evaluation types: *pinned boundary*, *floating boundary*, and *optimised boundary*.

The results are shown in Fig. 3.5. The pinned and optimised boundary handlers perform similarly, slowly converging to the analytical solution $u_{1,2}(x)$ within 250 iterations [Fig. 3.5(a)]. The two approaches have similar convergence in terms of the *full loss* [Fig. 3.5(b)], but differ in terms of *quality of solution* [Fig. 3.5(b)]. When using floating boundary type a function close to the true solution is obtained [with L_Q value of approximately 10^{-5}, see Fig. 3.5(b)]. This difference in convergence rate is a result of boundary information having an impact on the loss, demanding the matching for pinned and optimised boundary handlers, whereas the floating boundary automatically matches the initial condition and no loss boundary term is needed. The consequence of competing terms in the loss function can be seen in the early oscillations of the *full loss* in Fig. 3.5(b).

3.4.3 Fluid Dynamics Applications

An area where solvers for complex differential equations are much required is fluid dynamics [121]. In this case several outstanding models are hard to tackle due to their nonlinear nature. Examples include Burger's equation and Navier-Stokes equations. We concentrate on the latter and show how one can approach it with the DQC solver.

Navier-Stokes equations describe a flow of viscous fluids. This highly nonlinear set of partial differential equations is used to model fluids, magnetoplasma, turbulence etc. It is heavily used in the aerospace industry and weather forecasting. It can be derived from general principles. Namely, we consider fluid motion that obeys Newton's law and we simply track the fluid mass passing through an (infinitesimal) volume.

The general form of the Navier-Stokes equations for incompressible fluids can be presented in the form

$$\frac{\partial(\rho v_x)}{\partial t} + \nabla \cdot (\rho v_x \mathbf{V}) = -\frac{\partial p}{\partial x} + \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + \rho f_x, \tag{3.23}$$

$$\frac{\partial(\rho v_y)}{\partial t} + \nabla \cdot (\rho v_y \mathbf{V}) = -\frac{\partial p}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} + \rho f_y, \tag{3.24}$$

$$\frac{\partial(\rho v_z)}{\partial t} + \nabla \cdot (\rho v_z \mathbf{V}) = -\frac{\partial p}{\partial z} + \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} + \rho f_z, \tag{3.25}$$

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{V}) = 0, \tag{3.26}$$

$$\frac{\partial E}{\partial t} + \nabla \cdot (E\mathbf{V}) = -\nabla \cdot (p\mathbf{V}) - \nabla \cdot (\mathbf{q}) + \nabla \cdot (v_x \tau_{x\cdot} + v_y \tau_{y\cdot} + v_z \tau_{z\cdot}) \tag{3.27}$$

where (v_x, v_y, v_z) are instantaneous velocities in the (x, y, z) directions, τ is the stress tensor, f the body force per unit mass acting on the fluid element, ρ the density, p the pressure, and q the heat flux. Here $\mathbf{V}$ is a velocity field and $\tau_{x\cdot}, \tau_{y\cdot}, \tau_{z\cdot}$ denote the sub-vectors of τ formed of the terms with the first index being x, y, z respectively. The penultimate equation is the continuity function and the final the energy conservation equation.

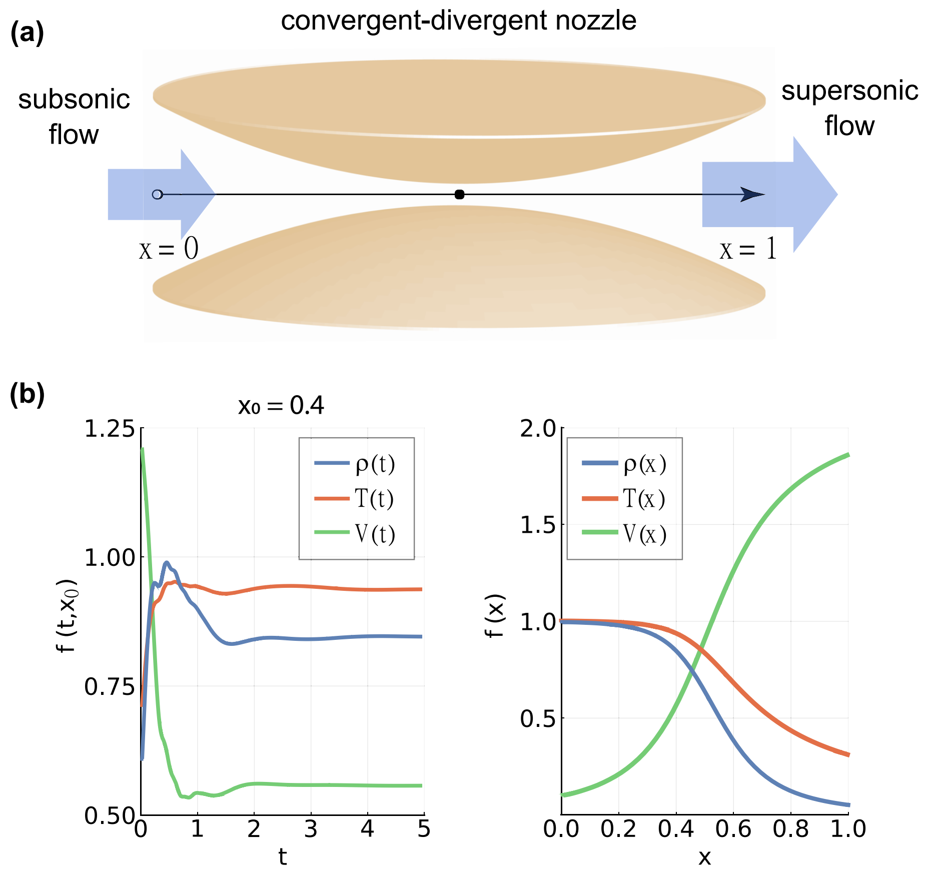


Figure 3.6: **Quasi-1d fluid dynamics.** (a) We consider an example of fluid dynamics problem corresponding to a convergent-divergent nozzle. The air flows from converging part of the nozzle ($x < 0.5$), passes through the throat placed symmetrically in the middle, and exits to diverging part ($x > 0.5$). (b) System variables [density $\rho(t)$, temperature $T(t)$, and velocity $V(t)$] are shown as functions of time, near the centre of the nozzle at $x_0 = 0.4$. For clarity, we label the corresponding by 1, 2, and 3. (c) Steady state solutions are plotted as functions of the spatial dimension x, with the same labelling.

The building blocks for the Navier-Stokes equations described above are the continuity equation for the density ρ subjected to energy conservation and the momentum conservation rules. Finally, as we use energy conservation, we can rewrite equations in terms of one of the thermodynamic state functions, being temperature, or pressure, or enthalpy etc.

While general by itself, Navier-Stokes equations are usually solved in relevant limiting cases. Specifically, this can correspond to space reduction (2D, quasi-1D, 1D), isotropic/anisotropic media properties, and fluid properties (viscous or inviscid flow). The specific example we choose to start with is the flow through a convergent-divergent nozzle, being a paradigmatic task in the aerospace industry

[Fig. 3.6(a)] [122, 121]. The Navier-Stokes equations can be rewritten for the inviscid fluid in quasi-1D approximation. They are now a set of Euler equations and read [121, 122]

$$\frac{\partial \rho}{\partial t} = -\rho \frac{\partial V}{\partial x} - \rho V \frac{\partial (\log A)}{\partial x} - V \frac{\partial \rho}{\partial x}, \tag{3.28}$$

$$\frac{\partial T}{\partial t} = -V \frac{\partial T}{\partial x} - (\gamma - 1) T \left(\frac{\partial V}{\partial x} + V \frac{\partial (\log A)}{\partial x} \right), \tag{3.29}$$

$$\frac{\partial V}{\partial t} = -V \frac{\partial V}{\partial x} - \frac{1}{\gamma} \left(\frac{\partial T}{\partial x} + \frac{T}{\rho} \frac{\partial \rho}{\partial x} \right), \tag{3.30}$$

where Eq. (3.28) corresponds to the continuity equation, Eq. (3.29) describes the energy conservation, and Eq. (3.30) stems from momentum conservation. $A(x)$ corresponds to the spatial shape of the nozzle, and is a function of the lateral coordinate x. γ describes the ratio of specific heat capacities, and is equal to 1.4 for the relevant case of air flow. Here we used nondimensional variables [121].

We set the problem with nozzle shape

$$A(x) = 1 + 4.95(2x - 1)^2, \quad 0 \leq x \leq 1, \tag{3.31}$$

and for simplicity specify boundary conditions as

$$\rho(x = 0) = 1, \quad T(x = 0) = 1, \quad V(x = 0) = 0.1, \tag{3.32}$$

when solving the initial value problem for the steady-state flow. We also need to specify the initial conditions if solving the dynamical problem. These are chosen as [121]

$$\rho(x, t = 0) = 1 - 0.944x, \tag{3.33}$$

$$T(x, t = 0) = 1 - 0.694x, \tag{3.34}$$

$$V(x, t = 0) = (0.1 + 3.27x) T(x, t = 0)^{1/2}. \tag{3.35}$$

We consider an initial value problem and only specify the value of our functions at $t = 0$.

Let us consider the stationary state problem represented by Eqs. (3.28)-(3.30) with conditions above, and equate the time derivatives to zero. Interestingly, when trying to solve the system for the steady state solution using various classical methods, as for example implemented in Mathematica's NDSolve, the calculations do not converge. This proves to be challenging as the system is stiff. Solutions may become unstable depending on initial values, and specifically the input velocity. To understand the problem, it is instructive to rewrite the system of stationary Navier-Stokes equations in the form

$$\frac{d\rho}{dx} = \frac{\rho V^2 d_x(\log A)}{T - V^2}, \tag{3.36}$$

$$\frac{dT}{dx} = \frac{T V^2(\gamma - 1) d_x(\log A)}{T - V^2}, \tag{3.37}$$

$$\frac{dV}{dx} = -\frac{T V d_x(\log A)}{T - V^2}. \tag{3.38}$$

Immediately we observe that each function at the RHS diverges at the point x s.t. $T(x) = V(x)^2$. This leads to singular behaviour and breaks classical solvers, including the ones with stiffness handling. At the same time, full dynamical solution and its $t \to \infty$ extrapolation are possible, shown in Fig. 3.6(b,c) once the *well-suited* initial conditions (3.33)-(3.35) are chosen allowing to avoid instability, as usually done in computational fluid dynamics [121, 122].

DQC solution. We proceed to solve the stationary system of Navier-Stokes equations for the convergent-divergent nozzle by constructing optimised DQC. We consider the case of subsonic-supersonic transition, where flow velocity increases after the center of the nozzle and qualitative behaviour of other state variables (temperature and density drop) is known. However, getting the quantitative results is difficult. We show that derivative quantum circuits can find solutions despite the challenge for the classical solution for the continuous grid.

To construct the solution we employ a two-stage optimisation approach, where the solution is first obtained at smaller x, and later generalised until the end of the nozzle. For the circuit we again consider a six-qubit quantum register with a Chebyshev quantum feature map, keeping in mind that $x \in [0, 1)$. We choose the cost functions as a total magnetisation $\hat{C} = \sum_{j=1}^{N} \hat{Z}_j$. As we consider three functions $\{\rho(x),\ T(x),\ V(x)\}$, three equations contribute to the loss function in the combined manner. We use *floating boundary* handling, where each curve is adjusted according to starting values, chosen as $\rho(0) = 1,\ T(0) = 1,\ V(0) = 0.1$. The variational ansatz is taken in the standard hardware efficient form with $d = 6$ depth. Adam is used as a classical optimiser, and the learning rate is set to 0.01.

At the first stage we train DQC in the region $(x_{\text{min}}, x_{\text{max}}) = (0., 0.4)$, such that the circuit represents the region close to initial point $x = 0$. As expected, in the subsonic region flow velocity grows towards the middle of the nozzle, while temperature and density drop slowly [see Fig. 3.7(a), dashed curves 4, 5, 6]. The training is set for 20 equally distributed points of x, with no prior regularisation and using floating boundary. We optimise DQC for $n_{\text{iter}} = 200$ iterations for Adam with the learning rate of $\alpha = 0.01$. We find a high-quality solution based on the gradient information in the imposed solution region $x < 0.4$ shown in Fig. 3.7(a).

We proceed to search for the full solution that includes the divergent nozzle part for $x > 0.5$ at the second training stage. At this session we choose the grid of 40 points, where two regions of $(0, 0.4)$ and $(0.6, 0.9)$ with 20 points each are used. As we discussed before, the key problem of the convergent-divergent nozzle in the subsonic-supersonic transition case is the divergence around the middle of the nozzle. This causes a major problem to classical solvers, that are unable to find a steady state solution directly. Divergent contributions from this region also impact the loss function, and makes training complicated. However, by excluding the region around the nozzle throat, $(0.4, 0.6)$, in the training we alleviate this problem. Proceeding with the use of the same ansatz and boundary handing, we feed the variational angles from stage 1 as initial parameters for stage 2 training.

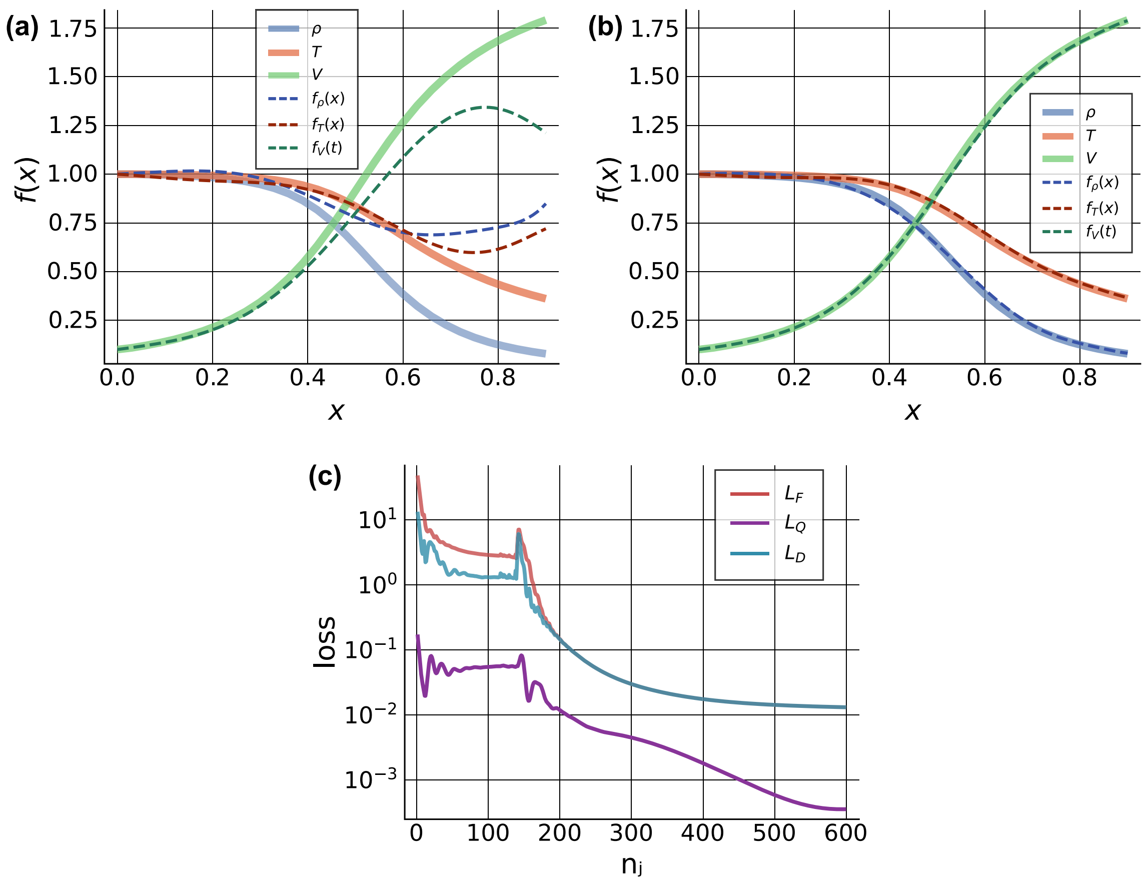


Figure 3.7: **DQC-based solution for the Navier-Stokes convergent-divergent nozzle problem.** (a) Intermediate solution in training where dashed curves correspond to solutions drawn from trained DQC and solid curves to the true solution. Circuits are trained over twenty points in range $(0, 0.4)$ (b) Final solution. Solid curves show the true solutions for the density, temperature, respectively. Dashed curves show the DQC solutions for the density, temperature, respectively. The DQC solution matches the known solution and also what would be physically expected. As the air goes through the nozzle it accelerates and cools down. (c) Full loss (L_F, curve 7), quality of solution (L_Q, curve 8), and differential loss (L_D, curve 9) are shown as functions of the iteration number n_j for the training resulting in solution (b).

We employ weak regularisation to ensure that the required subsonic-supersonic solution is made. Namely, we use 20 points in the $(0, 0.4)$ region benefiting from the previously found solution (in general we have access to as many points as we need), and also add 5 points in the $x \in (0.6, 0.9)$ region representing weak bias towards supersonic solution type. We emphasise we do not provide more boundary data to fit to, instead we provide information to bias towards V increasing, and T and ρ decreasing. The training is performed for $n_{\text{iter}} = 600$, learning rate of $\alpha = 0.005$, and regularisation switch-off function $\zeta(n_j)$ set to be removed smoothly around $n_j = 150$. The full solution is shown in Fig. 3.7(b). It converges to the expected long-time behaviour for the system, where function derivatives from DQC match the nonlinear contributions. The increase of speed for the air flow in the convergent part and decrease of temperature and density is reproduced quantitatively. The details of optimisation run can be inferred from the loss plotted in Fig. 3.7(c). This shows a distinct region in the presence of regularisation ($n_j < 150$), where quality of solution L_Q remains low [Fig. 3.7(c), curve 8], while the full loss L_F improves [Fig. 3.7(c), curve 7] thanks to regularisation contribution and weakly improved derivative contribution L_D [Fig. 3.7(c), curve 9]. For the region with switched off regularisation we observe steady improvement of all metrics, showing that DQCs are efficiently trained to approach true solution, also evidenced by L_Q decrease. Notably, as compared to many methods relying on sparse discretisation of x, we have found the solution along the full nozzle length.

3.5 Discussion

We presented a general framework for solving general (systems of) differential equations using differentiable quantum circuits on gate-based quantum hardware. The method makes use of quantum feature map circuits to encode function values into a latent space, which allows us to consider spectral decompositions of the trial solutions to differential equations. We showed how our method can accurately represent non-linear solutions, using the high-dimensional Hilbert space of a qubit register. For this we exploit a large spectral basis set of Fourier and Chebyshev functions. We also showed how analytical circuit differentiation can be used to represent function derivatives appearing in differential equations, and constructed loss functions which aim to improve the prepared trial solution. This method opens up a new way of solving general, complex, (non-)linear systems; as an example we presented solutions to the Navier-Stokes equation, but the same method can be applied across all disciplines where differential equations arise.

The method is presented is promising for future advantage but there still are open questions. We did not yet consider the impact of noise on the algorithm performance. Additionally, whilst not observed at the scale simulated, as the method scales barren plateaus as introduced in section 2.5 are likely to occur during training. Current and future efforts on mitigating/solving this variational training behaviour should be viable for this method.

Furthermore the algorithm is heuristic and choices of hyperparameters make a huge difference to performance. For example, the presented quantum feature maps are a good start, but could be improved upon further, where the goal is to find efficient representations of functions. This choice will likely be problem-specific and in some cases problem-motivated, allowing to make optimal use of (quantum) resources. For example, the considered feature maps all provide continuous functions with no sharp transitions. This would cause issues if trying to fit problems with sharp transitions such as the DEs governing fluids with back flows. For such situations the feature map and trial function would have to be carefully engineered with those features in mind and if discontinuities were present further developments made to the algorithm. A good choice of variational ansatz is crucial to loss function convergence success and speed. We presented some examples, but an active area of research is to improve upon these using for example adaptively growing circuits or stronger interplay with intelligent classical optimisation protocols, such as Bayesian optimisation.

In conclusion, we have developed a new way to consider solving differential equations with quantum computers. This method does not currently have a proven avenue for quantum advantage but we hope that with further work, adaptions and refinements a useful application can be found.

Chapter 4

Quantum Quantile Mechanics: Solving Stochastic Differential Equations for Generating Time-Series

4.1 Declaration of Contribution

The work in this chapter is published in [123]. My contributions consist of implementing and running all simulations, the idea behind the relationship of the quantile function and QGAN and suitability for SDEs, and shared contribution to manuscript writing.

4.2 Introduction

Stochastic differential equations (SDEs) describe a broad range of phenomena. They emerge when dealing with Brownian motion and quantum noise [124]. Fields interested in the solving of SDEs include stochastic fluid dynamics [125, 126], population dynamics [127] and financial calculus [89, 128]. Several questions arise when considering an SDE problem: how do we solve them, and what kind of information do we want to get by solving SDEs? These are not trivial questions to answer, because one might be interested in different aspects of SDE-modelled processes. One possible goal for solving an SDE is to to be able to sample from the final/evolved distribution. Therefore, we consider using quantile mechanics for solving SDEs. Quantile mechanics involves representing probability distributions in an easily sampleable form and consequently is well suited to the goal of sampling [88].

We propose a quantum algorithm which represents a probability distribution as quantum quantile function, trains and evolves it to solve a given SDE, then samples from the final distribution. Using a quantum model with a feature map encoding of latent variables, we represent the quantile function for an underlying probability distribution and extract samples as expectation values. Using quantile mechanics we propagate the system in time using a variational workflow, thereby allowing for time-series generation. This algorithm is particularly appropriate for SDEs where the final goal is to sample from the evolved distribution. We also propose an initialisation scheme to aid training. This combined quantum computing and quantile mechanics approach we call *quantum quantile mechanics*. We test the method by simulating the Ornstein-Uhlenbeck process [129] and sampling at times different from the initial point, as required in financial analysis and dataset augmentation.

Additionally, we analyse continuous quantum generative adversarial networks (QGANs), and show that they represent quantile functions with a modified (re-ordered) shape that impedes their efficient time-propagation. Our results shed light on the connection between quantum quantile mechanics (QQM) and QGANs for SDE-based distributions.

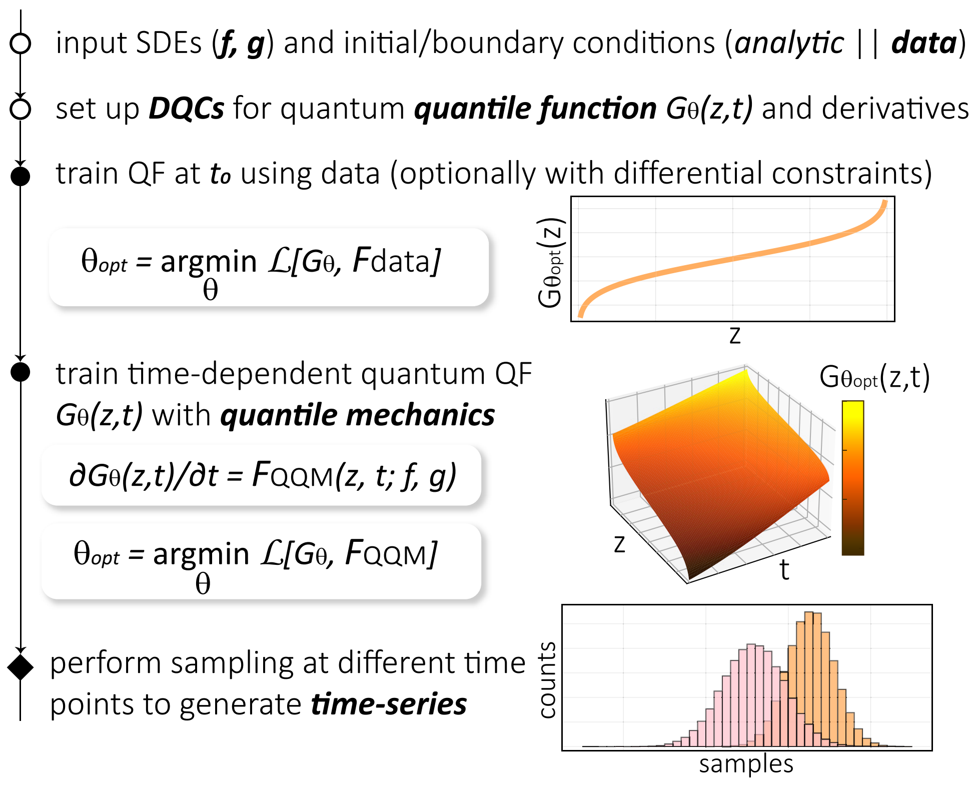

input SDEs (f, g) and initial/boundary conditions (analytic || data)
set up DQCs for quantum quantile function Gθ(z,t) and derivatives
train QF at t₀ using data (optionally with differential constraints)
θopt = argmin ℒ[Gθ, Fdata] θ
Gθopt(z)
z
train time-dependent quantum QF Gθ(z,t) with quantile mechanics
∂Gθ(z,t)/∂t = FQQM(z, t; f, g)
θopt = argmin ℒ[Gθ, FQQM] θ
Gθopt(z,t)
z
t
perform sampling at different time points to generate time-series
counts
samples

Figure 4.1: **Quantum quantile mechanics workflow.** Given a system of stochastic differential equations and initial data, differentiable quantum circuits are trained to represent the corresponding quantile function $G_\theta(z,t)$ as a function of a random latent variable (z) and propagate it in time (t) with quantum mechanics equations. The hybrid quantum-classical loop is used for optimising variational parameters θ through loss function $\mathcal{L}$ minimisation based on data and differential equations for the initial and propagated QF. Evaluating $G_{\theta_{\text{opt}}}(z \sim \mathtt{uniform}(-1,1), t)$ at optimal angles, random values of the latent variable, and different time points t, we generate time series from SDE.

4.3 Method

4.3.1 Algorithm

We have the goal to solve a given SDE which is describing a process of interest. As introduced in section 2.8.3 one method for solving SDEs is quantile mechanics where the SDE is rewritten as a PDE in terms of a quantile function (2.44). The result of this is a quantile function (QF) $Q(z)$ (2.42) which represents the probability distribution solution of the SDE and can be sampled with z following a uniform distribution.

In general, quantile mechanics equations are not easy to solve. Power series solution as function approximation are known [88, 130, 131, 132], as well as some simple examples [133]. To harness the full power of quantile mechanics, we thus propose to use neural representation of QFs. To our knowledge, this is the first application of machine learning methods to use quantile-based sampling, and we envisage that both classical and quantum ML can be used for the universal function approximation [134, 135]. During this chapter, I focus on testing that this algorithm is a valid approach for these problems, in particular using a quantum approach with a quantum model. Further in depth exploration of these approaches performance compared to others is an important step for future work. In particular what is the quantum benefit over the classical version. The intuition is that the use of quantum neural networks offers a potential to reproducing complex functions in the high-dimensional space, including systems where strong correlations are important.

In the following, we combine quantum computing and quantile mechanics to develop the *quantum quantile mechanics* approach. The sketch of the QQM workflow is shown in Fig. 4.1. We represent QFs as quantum models, thus exploiting the large expressivity of quantum-based learning. With this we ensure that we have the ability for solving the problem.

Quantum Model

To represent a trainable quantile function we construct a quantum model using quantum embedding through feature maps $\hat{\mathcal{U}}_{\varphi}(x)$ (with φ labelling a mapping function) [28, 136, 114], followed by variationally-adjustable circuit (ansatz) $\hat{\mathcal{U}}_{\boldsymbol{\theta}}$ parametrised by angles $\boldsymbol{\theta}$ as introduced in section 2.4. The readout is set as a sum of weighted expectation values [135] and the represented function can be differentiated using the parameter shift rule. Following this structure, we assign a generator circuit $G(z, t)$ to represent a function parametrised by t (labels time),

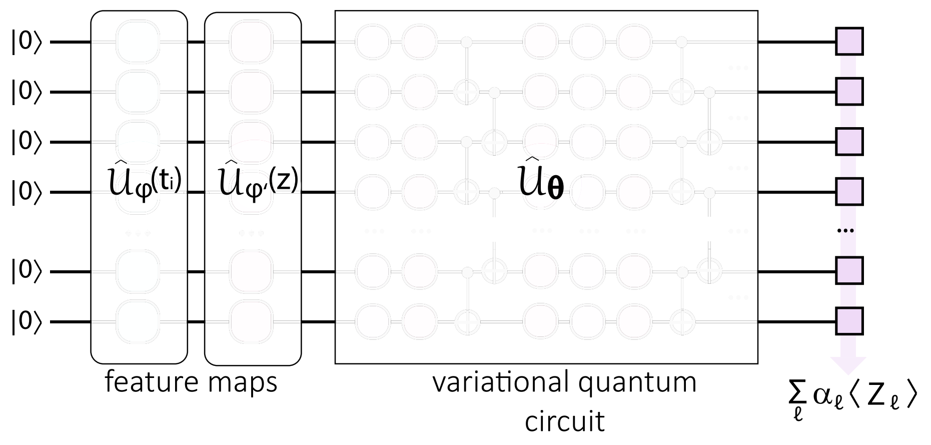


Figure 4.2: **QQM Quantum Model.** State preparation circuit for representing a time-dependent quantile function. Acting on the initial state, two feature maps, with $\hat{R}_y$ using for time-dependence embedding and $\hat{R}_x$ layer for the latent variable embedding. We use HEA for the variational search, and total Z magnetisation as a cost function.

and the embedded latent variable z. The model reads

$$G(z,t) = \langle 0|\hat{\mathcal{U}}_{\phi}(t)^{\dagger}\hat{\mathcal{U}}_{\phi'}(z)^{\dagger}\hat{\mathcal{U}}_{\boldsymbol{\theta}}^{\dagger}\Big(\sum_{\ell=1}^{L}\alpha_\ell C_\ell\Big)\hat{\mathcal{U}}_{\boldsymbol{\theta}}\hat{\mathcal{U}}_{\phi'}(z)\hat{\mathcal{U}}_{\phi}(t)|0\rangle, \tag{4.1}$$

where $\hat{\mathcal{U}}_{\phi}(z)$ and $\hat{\mathcal{U}}_{\phi'}(t)$ are quantum feature maps (possibly different), $\{C_\ell\}_{\ell=1}^{L}$ represent L distinct Hermitian cost function operators, and $\{\alpha_\ell\}_{\ell=1}^{L}$ and $\boldsymbol{\theta}$ are real coefficients that may be adjusted variationally. The overall cost function may be global, where a projector on some specific bitstring is chosen, or local as a sum of Pauli terms. The choice of $\{\alpha_\ell\}_{\ell=1}^{L}$ enables to bias for/against certain operators/qubits information – for example for lower/higher frequencies as in [137]. The circuit structure is shown in Fig. 4.2. We can also work with multiple latent variables and thus multidimensional distributions.

Training

Our next step is developing the training procedure for G such that it represents the QF for an underlying data distribution. Namely, we require that the circuit maps the latent variable $z \in [-1,1]$ to a sample $G(z) = Q(z)$.

The training requires a dataset $\{X_{\mathrm{data}}\}$ generated from a probability distribution for the system we want to study (or measured experimentally), which serves as an initial/boundary condition. Additionally, we know the underlying processes that describe the system and which serve as differential constraints. The problem is specified by SDEs $d\boldsymbol{X}_t = f_{\boldsymbol{\xi}}(\boldsymbol{X}_t, t)dt + g_{\boldsymbol{\xi}}(\boldsymbol{X}_t, t)dW_t$, where we explicitly state that the drift and diffusion functions $f_{\boldsymbol{\xi}}(\boldsymbol{X}_t, t)$ and $g_{\boldsymbol{\xi}}(\boldsymbol{X}_t, t)$ are parametrised by the vector $\boldsymbol{\xi}$ (time-independent). Here, $\boldsymbol{\xi}$ contains the vector of parameters which identify the exact SDE from its class of SDEs and could be found via model discovery [138]. For instance, for the Ornstein-Uhlenbeck model, which we refer to later, we have $\boldsymbol{\xi} = (\mu, \nu, \sigma)$ corresponding to the mean, rate of mean reversion, and volatility, respectively. We consider that the SDE parameters for generating data similar to $\{X_{\mathrm{data}}\}$ are known in the first approximation $\boldsymbol{\xi}^{(0)}$. This can be adjusted during the training to have the best convergence, as used in the equation discovery approach [139, 140].

The loss is constructed as a sum of data-based and SDE-based contributions, $\mathcal{L} = \mathcal{L}_{\mathrm{data}} + \mathcal{L}_{\mathrm{SDE}}$. The first part $\mathcal{L}_{\mathrm{data}}$ is designed such that the data points in $\{X_{\mathrm{data}}\}$ are represented by the trained QF. For this, the data is binned appropriately and collected in ascending order, as expected for any quantile function. Then, we use quantum circuit learning (QCL) as a quantum nonlinear regression method [28] to learn the quantile function $G_0(z)$. $\mathcal{L}_{\mathrm{data}}$ then takes the form $\mathcal{L}_{\mathrm{data}} = \sum_j (G(z_j, 0) - G_0(z_0))^2$ where $\{z_j\}_j$ is a set of training points for the boundary. We note that such an approach represents a data-frugal strategy, where the need for training on *all* data points is alleviated. We also note that this binning only occurs on the boundary/initial date to prepare it for use in training. Binning otherwise is only used for later histogram plotting and is purely visual.

The second loss term $\mathcal{L}_{\mathrm{SDE}}$ is designed such that the learnt quantile function obeys probability models associated to SDEs. Specifically, the generator $G(z, t)$ needs to satisfy the associated quantilised Fokker-Planck equation as introduced in section 2.8.3. The differential loss is introduced using the DQC approach as in chapter 3 [112], and reads

$$\mathcal{L}_{\mathrm{SDE}} = \frac{1}{M} \sum_{z,t \in \mathcal{Z},\mathcal{T}} L\left[\frac{\partial G_{\boldsymbol{\theta}}}{\partial t}, F\left(z, t, f, g, \frac{\partial G_{\boldsymbol{\theta}}}{\partial z}, \frac{\partial^2 G_{\boldsymbol{\theta}}}{\partial z^2}\right)\right], \tag{4.2}$$

where $L[a, b]$ denotes the distance measure for two scalars, and the loss is estimated over the grid of points in sets $(\mathcal{Z}, \mathcal{T})$. Here M is the total number of points and is equal to the number of points in $\mathcal{Z}$ multiplied by the number of points in $\mathcal{T}$. We also introduce the function $F(z, t, f, g, \partial_z G_{\boldsymbol{\theta}}, \partial^2_{zz} G_{\boldsymbol{\theta}})$ denoting the RHS for Eq.2.44, or any other differential constraint. Using a differential equation based constraint is a core principle of physics-informed machine learning and provides different fitting behaviour versus regression due to the requirements on derivative values as well as function values.

Once the training regime is set, the loss is minimised using a hybrid quantum-classical loop where optimal variational parameters $\boldsymbol{\theta}_{\mathrm{opt}}$ (and $\boldsymbol{\alpha}_{\mathrm{opt}}$) are searched using non-convex optimisation methods.

Generalisation of this model is similar to that discussed in chapter 3. Furthermore, as a non-convex variational algorithm, the trainability considerations as discussed in section 2.5 such as expressivity, barren plateaus, local minima etc. are to to be considered.

4.3.2 Initialisation

In this section we describe a method of *parameter initialisation* that we use to provide a good starting function when implementing QCL or DQC. This is achieved by having a circuit structure where the variational circuit can be initialised to the identity operator and two layers of single-qubit rotation gates (which we refer to as the initialisation layers). The parameters of the initialisation layers can then be set to provide a good starting fit of the initial trial function to (an estimate of) the target function.

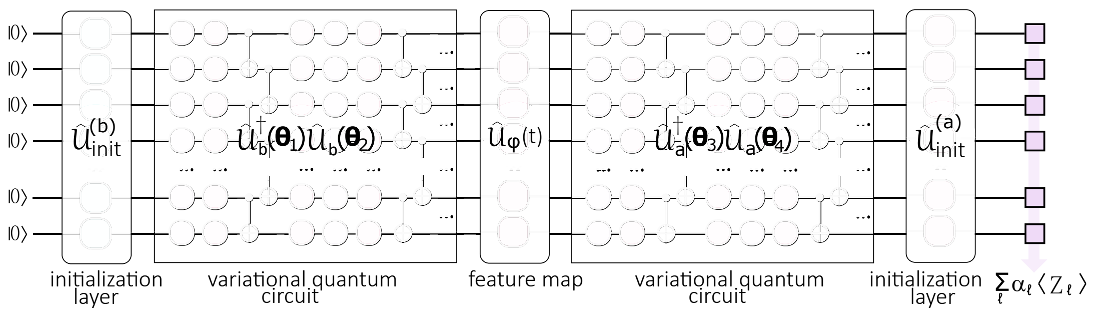

|0⟩
|0⟩
|0⟩
|0⟩
|0⟩
|0⟩
$\hat{U}^{(b)}_{\text{init}}$
$\hat{U}^\dagger_b(\theta_1)\hat{U}_b(\theta_2)$
$\hat{U}_\varphi(t)$
$\hat{U}^\dagger_a(\theta_3)\hat{U}_a(\theta_4)$
$\hat{U}^{(a)}_{\text{init}}$
initialization layer
variational quantum circuit
feature map
variational quantum circuit
initialization layer
$\sum_\ell \alpha_\ell \langle Z_\ell \rangle$

Figure 4.3: **Initialisation circuit.** We show the circuit structure used for when implementing initialisation. It consists of layers of rotations known as the initialisation layers $\hat{\mathcal{U}}_{\text{init}}$, variational quantum circuits made up of $\hat{\mathcal{U}}(\boldsymbol{\theta})$ and the feature map $\hat{\mathcal{U}}_\varphi(t)$. The variational quantum circuits are structured so that they can easily be initialised to the identity operator. Suitable parameters for $\hat{\mathcal{U}}_{\text{init}}$ are chosen by performing classical function fitting. The feature map dictates which function form a basis for the fitting.

The circuit structure that we use is shown in Fig. 4.3. The variational circuits are formed by a parameterised circuit unitary $\hat{\mathcal{U}}_{a/b}(\boldsymbol{\theta}_k)$ followed by the circuit with the adjoint structure but independently tuned set of variational angles, $\hat{\mathcal{U}}^\dagger_{a/b}(\boldsymbol{\theta}_{k'})$. We include variational circuits before and after the feature map to aid expressivity. For initialisation we set $\boldsymbol{\theta}_1 = \boldsymbol{\theta}_2$ and $\boldsymbol{\theta}_3 = \boldsymbol{\theta}_4$. We choose this restriction as it leads to the variational circuits being initialised as identity and consequently the initial trial function is efficiently classically simulable for initialisation. We stress that during training the parameters of these circuits are considered distinct and thus when updated by classical optimiser they do not remain equal.

With the variational circuits initialised to identity the initial trial function is now $f_0(t) = \langle 0|\hat{\mathcal{U}}^{(b)\dagger}_{\text{init}}\hat{\mathcal{U}}^\dagger_\varphi(t)\hat{\mathcal{U}}^{(a)\dagger}_{\text{init}}\mathcal{C}\hat{\mathcal{U}}^{(a)}_{\text{init}}\hat{\mathcal{U}}_\varphi(t)\hat{\mathcal{U}}^{(b)}_{\text{init}}|0\rangle$. By considering initialisation circuits and feature map which act on each qubit individually as a product of Pauli rotations (as is the case with all feature maps discussed in this paper), and cost as the sum of individual qubit measurements, we need to treat only 1-local terms. This means that for initialisation we have a circuit where the qubits are non-interacting and therefore the circuit is efficiently classically tractable. As the number of qubits N increase we can still describe the system with $O(N)$ at the initialisation stage, and we note that in principle k-local terms can also be included as long as the system remains tractable. Alternatively, we can also consider the circuit based on Clifford gates. In this case the circuit remains classically tractable following the Gottesman-Knill theorem. For the remainder of this section we focus on the situation with initialisation and feature map gates acting locally on single qubits.

With the circuit structure as discussed, the general form of the initial trial function can be expressed as

$$f_0(t) = \sum_{j=1}^{N} \alpha_j \, \left(c_j^{(1)}(\theta_{\mathrm{init}}) \, g_j^{(1)}(t) + c_j^{(2)}(\theta_{\mathrm{init}}) \, g_j^{(2)}(t) \right). \tag{4.3}$$

Here $c_j(\theta_{\mathrm{init}})$ are coefficients which depend on the angles parameterising the initialisation layer, α_j are the coefficients of the measurements in the cost function and $g_j(t)$ are functions of the variable encoded within the circuit by the feature map. Because this circuit is classically tractable the exact form of c_j and g_j remain calculable as N increases. To see this note that for the feature maps considered the gate implemented on qubit j is

$$\hat{U}_j(t) = \hat{R}_{\beta,j}(\varphi_j(t)) = \exp\left(-i\hat{P}_j^{\beta}\varphi_j(t)/2\right) \tag{4.4}$$

$$= \cos(\varphi_j(t)/2)\hat{I}_j - i\sin(\varphi_j(t)/2)\hat{P}_j^{\beta} \tag{4.5}$$

where $\beta \in \{x, y, z\}$ are Pauli operators and φ_j are encoding functions. Therefore, the functions $\cos(\varphi_j(t)/2)$ and $\sin(\varphi_j(t)/2)$ are introduced into the state by the feature map. The feature map encoding is thus based on the sets of functions $g^{(1)} = \{\cos(\varphi_j(t))\}_j$ and $g^{(2)} = \{\sin(\varphi_j(t))\}_j$. Note that the coefficients in front depend on initialisation layers, the feature map and the measurement operator chosen. For instance, by choosing specific circuit structures it is possible to select only a specific set of functions to use for initialisation.

Knowing the form of the initial trial function we can choose θ_{init} and α_i such that it is a good starting state. We do this by performing classical regression. The fitting function's coefficient set $\{g_j\}$ are fitted to a limited subset of the target function values. The size of $\{g_j\}$ is at most double the number of qubits and therefore when considering NISQ implementation will be relatively small. Therefore, as a low number of fitting functions are being fitted at a limited number of points, a low-order approximation and its associated coefficients can easily be found. These coefficients are then first used to choose α_j such that they are of suitable scale and the coefficient magnitudes desired are reachable by $\alpha_j c_j$. With α_j set, the expressions $\alpha_j c_j(\theta_{\text{init}})$ are then reversed to find the values which θ_{init} should be initialised to.

Once the initialisation coefficients have been calculated the standard QCL procedure is then commenced. The initialisation parameters will remain fixed whilst the variational ansatz parameters will be updated. As this happens the variational ansatze will no longer remain identity operators and more fitting functions will be introduced, with the set cardinality defined by the feature map. The circuit will no longer be efficiently classically simulable. The increase in the number of fitting functions (along with number of training points) leads to a better fit of the target data being possible. When using initialisation a smaller learning rate is preferred to prevent immediate divergence from the initialised function.

4.4 Results

In the next subsections we present numerical simulations of generative modelling. In the first part we apply the developed quantum quantile mechanics approach for solving a specific SDE, and demonstrate a data-enabled operation.

4.4.1 Ornstein-Uhlenbeck for Financial Forecasting and Trading

To validate the QQM approach and perform time series forecasting, we pick a prototypical test problem. As an example we choose the Ornstein-Uhlenbeck (OU) process. This process appears in many situations such as the motion of particle undergoing Brownian motion. In financial analysis its generalisation is known as the Vasicek model [141]. It describes the evolution of interest rates and bond prices [142]. This stochastic investment model is the time-independent drift version of the Hull–White model [143] widely used for derivatives pricing. We also note that OU describes dynamics of currency exchange rates, and is commonly used in Forex pair trading — a primary example for quantum generative modelling explored to date [144, 145]. Thus, by benchmarking the generative power of QQM for OU we can compare it to other strategies (valid at fixed time point).

The Ornstein-Uhlenbeck process is described by an SDE with an instantaneous diffusion term and linear drift. For a single variable process X_t the OU SDE reads

$$dX_t = \nu(\mu - X_t)dt + \sigma dW_t, \tag{4.6}$$

where the vector of underlying parameters $\boldsymbol{\xi} = (\nu, \mu, \sigma)$ are the speed of reversion ν, the long-term mean level μ, and the degree of volatility σ. The corresponding Fokker-Planck equation for the probability density function $p(x,t)$ reads

$$\frac{\partial p(x,t)}{\partial t} = -\nu \frac{\partial}{\partial x}((\mu - x)p) + \frac{\sigma^2}{2}\frac{\partial^2 p}{\partial x^2}. \tag{4.7}$$

When rewritten in the quantilised form, it becomes a PDE for the quantile mechanics,

$$\frac{\partial Q(z,t)}{\partial t} = \nu[\mu - Q(z,t)] + \frac{\sigma^2}{2}\left(\frac{\partial Q}{\partial z}\right)^{-2}\frac{\partial^2 Q}{\partial z^2}, \tag{4.8}$$

which follows directly from the generic quantile equation as discussed in section 2.8.3. In the following we take the speed of reversion to be positive, $\nu > 0$ and adjust the long-term mean level to zero, $\mu = 0$.

Having established the basics, we train the differentiable quantum circuit to match the OU QF. First, for the starting point of time, we train the circuit to represent a quantile function based on available data (see the workflow chart in Fig. 4.1 and the discussion below). Next, having access to the quantum QF at the starting point, we evolve it in time solving the equation

$$\frac{\partial G(z,t)}{\partial t} = -\nu G(z,t) + \frac{\sigma^2}{2}\left(\frac{\partial G}{\partial z}\right)^{-2}\frac{\partial^2 G}{\partial z^2}, \tag{4.9}$$

as dictated by Eq. (4.8) from utilising quantile mechanics. This is the second training stage in the workflow chart shown in Fig. 4.1.

To check the results, we use the analytically derived PDF valid for the Dirac delta initial distribution $p(x,t_0) = \delta(x - x_0)$ peaked at x_0 that evolves as

$$p(x,t) = \sqrt{\frac{\nu}{\pi\sigma^2(1-\exp[-2\nu(t-t_0)])}} \times \exp\left[-\frac{\nu(x - x_0\exp[-\nu(t-t_0)])^2}{\sigma^2(1-\exp[-2\nu(t-t_0)])}\right], \tag{4.10}$$

and we can write the OU QF evolution as

$$Q(z,t) = x_0\exp\big[-\nu(t-t_0)\big] + \sqrt{\frac{\sigma^2}{\nu}\Big(1-\exp\big[-2\nu(t-t_0)\big]\Big)}\,\mathrm{inverf}(z), \tag{4.11}$$

where $\mathrm{inverf}(x)$ denotes the inverse error function [146]. This provides us a convenient benchmark of a simple case application and allows assessing the solution quality. Additionally we use Euler-Maruyama integration to compare results with the numerical sampling procedure with fixed number of shots.

4.4.2 Ornstein-Uhlenbeck with Analytic Initial Condition

To highlight the generative power of the QQM approach we start by simulating the OU evolution $G(z,t)$ starting from the known initial condition. This is set as $G(z,0) = Q(z,0)$ being the analytic solution [Eq. (4.11)] or can be supplied as a list of known samples associated to latent variable values. To observe a significant change in the statistics and challenge the training, we choose the dimensionless SDE parameters as $\nu = 1$, $\sigma = 0.7$, $x_0 = 4$, and $t_0 = -0.2$ such that we evolve a narrow normal distribution with strongly shifted mean into a broad normal distribution

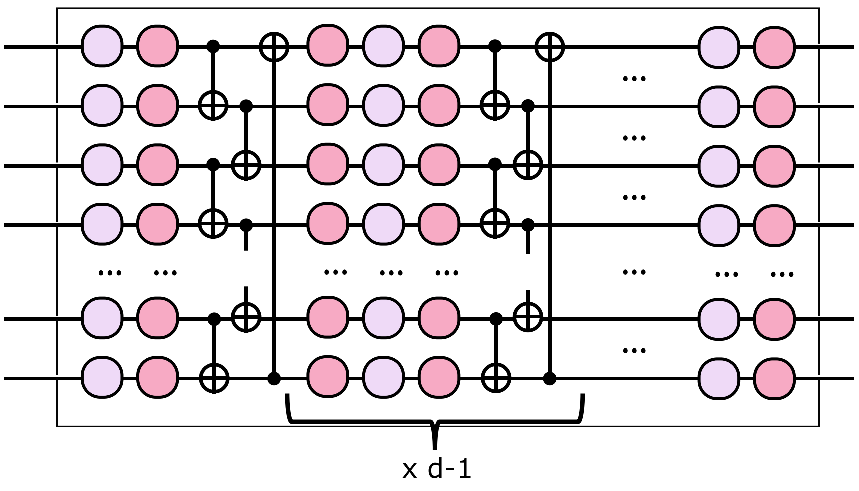


Figure 4.4: **Circuit diagram of variational circuit used in simulations.** Purple/pink boxes represent Pauli X/Y rotation gates. Entangling layer consists of CNOT gates on nearest neighbours and endpoints. Variational circuit of depth d is made up of initial layer of rotations $\hat{R}_x\hat{R}_z$ on each qubit followed by entangling layer composed of CNOTs. Then follows $d-1$ implementations of rotation layer ($\hat{R}_z\hat{R}_x\hat{R}_z$ on each qubit), then entangling layer. The circuit terminates after a final set of two rotations $\hat{R}_z\hat{R}_x$ on each qubit.

at $\mu = 0$.

We use quantum model with $N = 6$ qubits and a single cost operator being the total Z magnetisation, $C = \sum_{j=1}^{N} \hat{Z}_j$. This cost operator was chosen as it collects information from all qubits and it is local (here, the sum of single-qubit operators). Local cost operators were shown to have favorable behaviour as compared to global when considering barren plateaus [35]. To build a model that does not bias certain qubits we choose to have equal coefficients for the operator on each qubit.

For simplicity we train the circuit using a uniformly discretised grid with $\mathcal{Z}$ containing 21 points from -1 to 1, and $\mathcal{T}$ containing 20 values from 0.0 to 0.5, totalling 420 points in the full grid. To encode the function we use the product-type feature maps [28, 112] chosen as $\hat{\mathcal{U}}_{\phi}(t) = \bigotimes_{j=1}^{N} \exp\big[-i \arcsin(t)\hat{Y}_j/2\big]$ and $\hat{\mathcal{U}}_{\phi'}(z) = \bigotimes_{j=1}^{N} \exp\big[-i \arcsin(z)\hat{X}_j/2\big]$. The variational circuit corresponds to HEA with the depth of six layers of generic single-qubit rotations plus nearest-neighbor CNOTs, the circuit diagram of ansatz from is shown in Fig. 4.4. We exploit the floating boundary handling as described in section 3.3.4 [112], and choose a mean squared error (MSE) as the distance measure, $L[a,b] = (a-b)^2$ of our

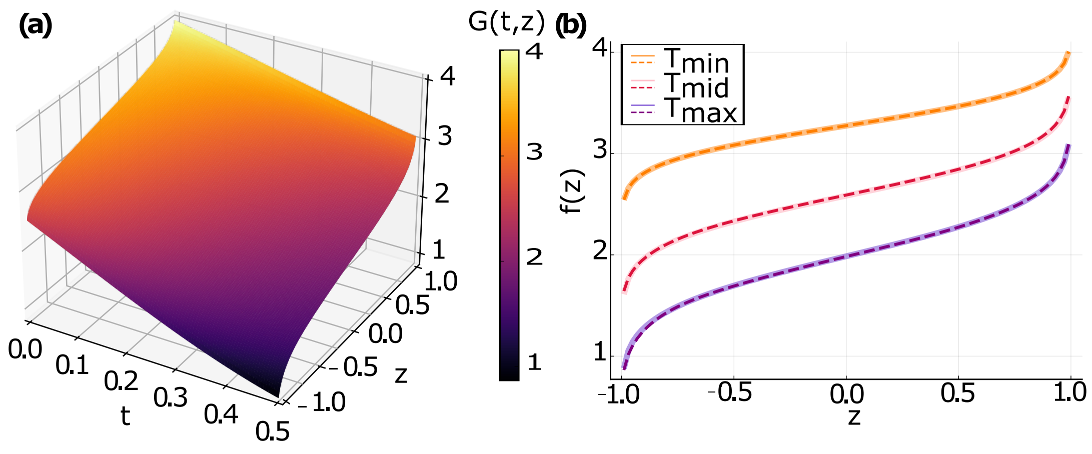


Figure 4.5: **Training time evolution of Ornstein-Uhlenbeck process.** The results are shown for runs with analytic initial condition and parameters chosen as $\nu = 1, \sigma = 0.7, x_0 = 4$. **(a)** Surface plot for the trained DQC-based quantile function $G(z,t)$ that changes in time. **(b)** Slices of the quantum quantile function $G(z,t)$ shown at discrete time points $t = 0$ (labelled as T_{min} hereafter), $t = 0.25$ (T_{mid}), and $t = 0.5$ (T_{max}).

loss in Eq. (4.2). Due to floating boundary handling being used, there is no $\mathcal{L}_{\mathrm{data}}$ term in the loss function. The system is optimised for a fixed number of epochs, 750, and we use the `Adam` optimiser for gradient-based training of variational parameters $\boldsymbol{\theta}$. We implement this workflow with a full quantum state simulator in a noiseless setting. This, as well as the other simulations presented, is realised in `Yao.jl` [110] — a `Julia` package that offers state-of-the-art performance. For measurement, infinite shots are used. The effects of noise is simulated later in section 4.4.6. Gradients calculations are simulated with an in-built function of Yao.jl which utilises automatic differentiation techniques (back propagation) or with application of the parameter shift rule.

We present the results of training in Fig. 4.5. In Fig. 4.5(a) we show the trained quantum QF as a function of time t and the latent variable z. Choosing three characteristic points of time $t = \{0.0, 0.25., 0.50\}$ that we label as $\{T_{\mathrm{min}}, T_{\mathrm{mid}}, T_{\mathrm{max}}\}$, we plot the corresponding quantile functions at these times [Fig. 4.5]. The dashed curves from the DQC training closely follow ideal QFs shown by solid curves. Additionally, we note we observed the training loss smoothly and rapidly converging in 250 epochs as the circuit is expressive enough to represent changes of initial

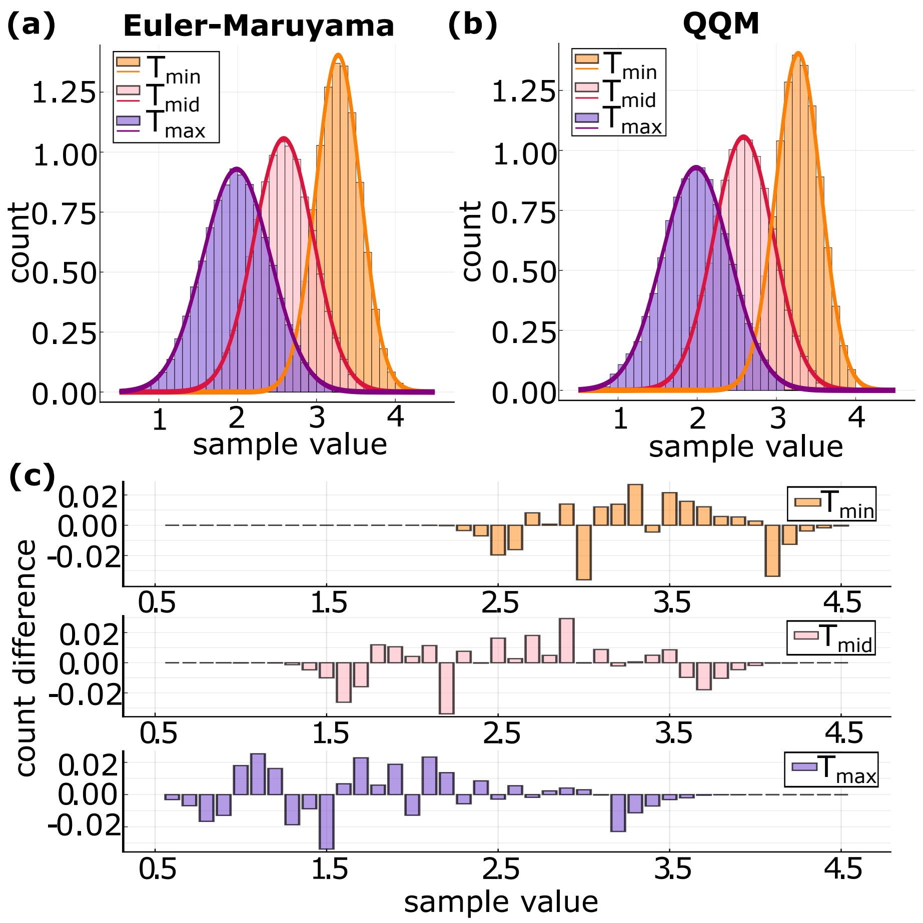


Figure 4.6: **Comparison of histograms from the numerical SDE integration and QQM training. (a)** Distribution of samples from the Euler-Maruyama SDE solver (binned counts divided by the total number of samples N_s), shown against the analytic PDF at three time points $T_{\rm min}$ ($t = 0$), $T_{\rm mid}$ ($t = 0.25$), and $T_{\rm max}$ ($t = 0.5$). $N_s = 100{,}000$ samples are taken, and parameters are the same ($\nu = 1$, $\sigma = 0.7$, $x_0 = 4$). **(b)**. Distribution of samples generated from the DQC-based quantile function, for the training described in Fig. 4.5. **(c)** Bar height difference between **(a)** and **(b)** shown for $T_{\rm min}$ (top), $T_{\rm mid}$ (middle), and $T_{\rm max}$ (bottom).

QF at increasing time, and thus providing us with evolved $G(z,t)$.

Next, we perform sampling and compare the histograms coming from the Euler-Maruyama integration of OU SDE [147, 148] and the QQM training presented above. The results are shown in Fig. 4.6 for the same parameters as Fig. 4.5. In Fig. 4.6(a) we show the three time slices of Euler-Maruyama trajectories, built with $N_s = 100{,}000$ samples to see distributions in full. The counts are binned and normalised such that the total area of the histogram (bin width multiplied by bin height summed) is equal to one, and naturally show excellent correspondence with analytical results. The sampling from trained quantile is performed by draw-

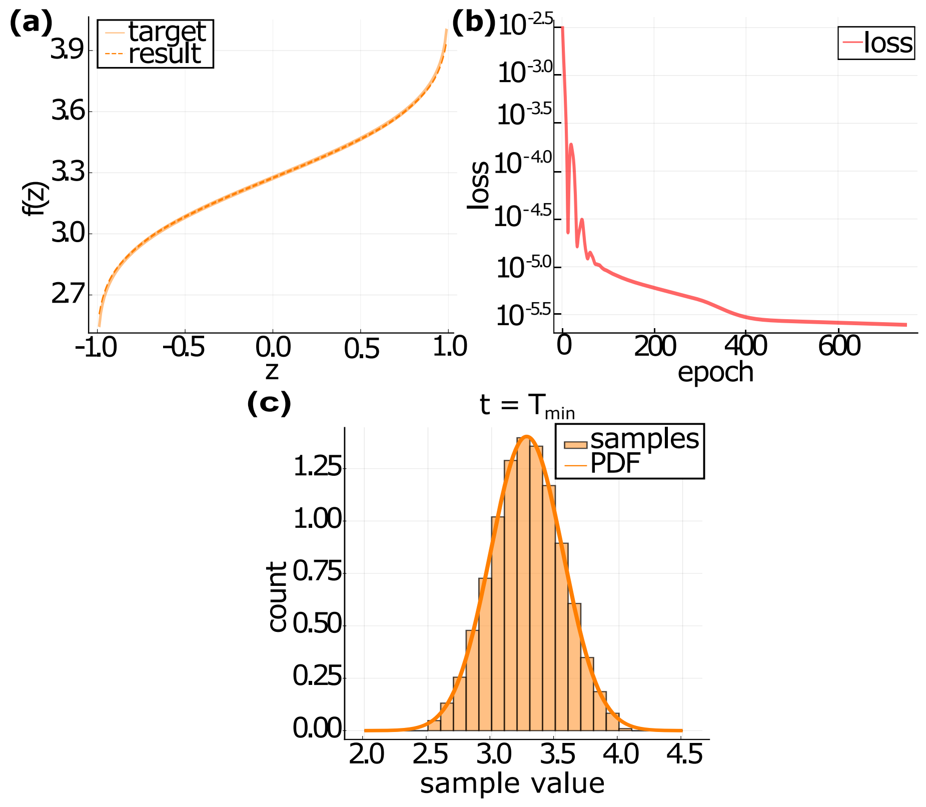


Figure 4.7: **Quantile function trained on initial data.** **(a)** Trained QF for the Ornstein-Uhlenbeck process at $t = 0$ (dashed curve labeled as `result`), plotted together with the known true quantile (solid line labeled as `target`, results overlay). The parameters are the same as for Fig. 4.5. **(b)** Training loss at different epochs, with the final epoch producing the QF in **(a)**. **(c)** Normalised histogram of samples from the data-trained QF, plotted against the analytic distribution (PDF). $N_s = 100{,}000$ random samples are drawn and bin counts are normalised for a total area of one as before.

ing random $z \sim \texttt{uniform}(-1, 1)$ for the same number of samples. In Fig. 4.6(b) we observe that QQM matches well the expected distributions. Importantly, the training correctly reproduces the widening of the distribution and the mean reversion, avoiding the mode collapse that hampers adversarial training [149, 150]. To further corroborate our findings, we plot the difference between two histograms (Euler-Maruyama and QQM) in Fig. 4.6(c), and observe that the count difference remains low at different time points.

4.4.3 Ornstein-Uhlenbeck with Data-inferred Initial Condition

Next, we demonstrate the power of quantile function training from the available data (observations, measurements) corresponding to the Ornstein-Uhlenbeck pro-

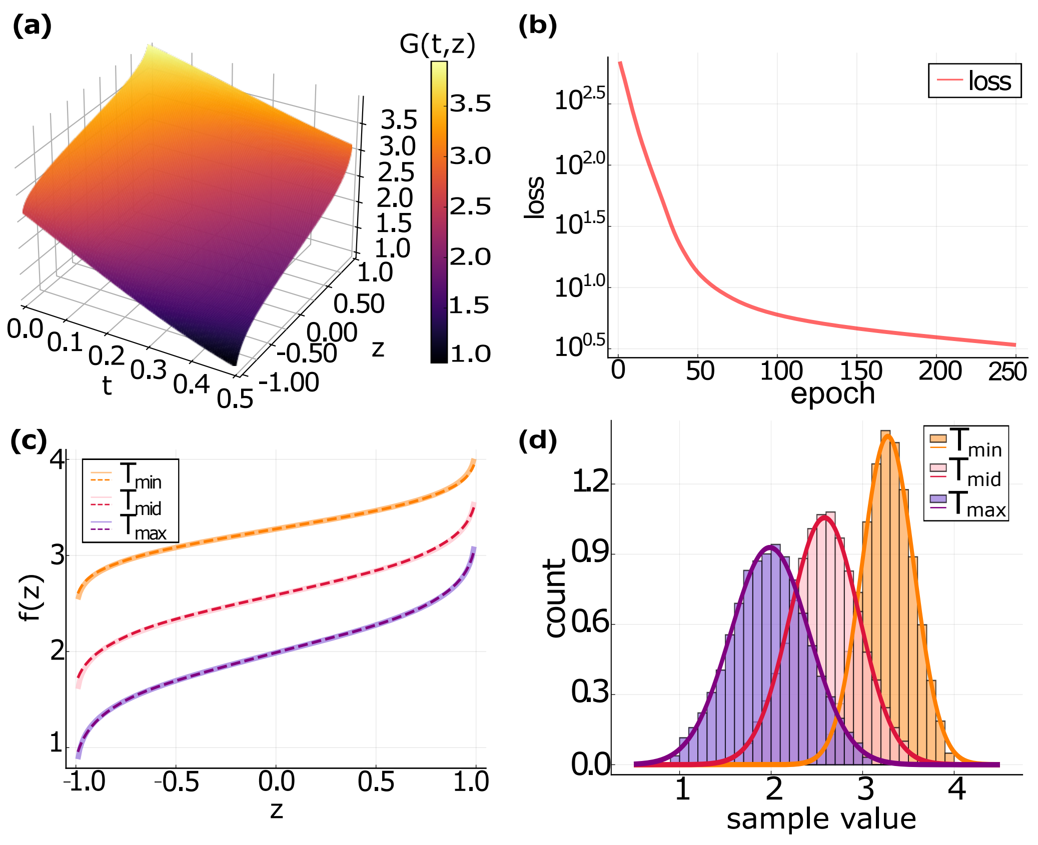


Figure 4.8: **Time-evolution of Ornstein-Uhlenbeck process with data-inferred initial condition.** The initial conditions is taken from Fig. 4.7 and we use the same OU model parameters. **(a)** Surface plot for the quantile function evolved in time. **(b)** Loss as a function of epoch from training in **(a)**.**(c)** Quantile functions shown at three time points. **(d)** Histograms for the data-trained QF evolved with quantum quantile mechanics, shown at same three points of time.

cess. Note that compared to the propagation of a known solution that is simplified by the boundary handling procedure, for this task we learn both the surface $G(z,t)$ and the initial quantile function $G(z,T_{\min})$. To learn the initial QF (same parameters as for Figs. 4.5 and 4.6) we use QCL trained on observations. The observations are $100{,}000$ samples from the normal distribution with mean $\mu_0 = x_0 \exp[\nu t_0]$ and standard deviation $\sigma_0 = \sqrt{\frac{\sigma^2}{\nu}\left(1 - \exp[2\nu t_0]\right)}$, where $x_0 = 4, \nu = 1, t_0 = -0.2, \sigma = 0.7$ (all dimensionless units). The samples in the initial dataset are collected into bins and sorted in the ascending order as required by QF properties. From the original $N_s = 100{,}000$ that are ordered we obtain an interpolated curve. We get target values for QCL training choosing $N_{\text{points}} = 43$ points in $\mathcal{Z}$ between -1 and 1. We note that the training set is significantly reduced, and such data-frugal training holds as long as the QF structure is captured (monotonic increase). The

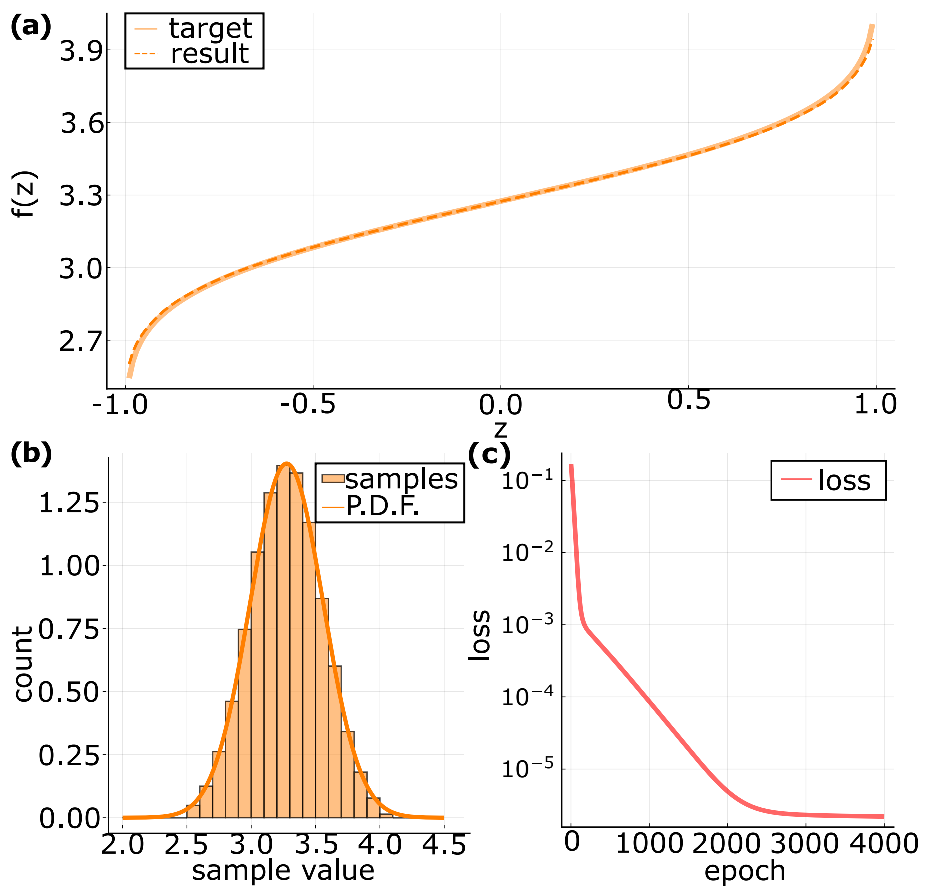


Figure 4.9: **Quantile function trained on initial data — no initialisation. (a)** Trained QF for the Ornstein-Uhlenbeck process at $t = 0$ (dashed curve labeled as `result`), plotted together with the known true quantile (solid line labeled as `target`, results overlay). **(b)** Normalised histogram of samples from the data-trained QF, plotted against the analytic distribution (PDF). $N_s = 100{,}000$ random samples are drawn and bin counts are normalised such that total area of the histogram is one. **(c)** Training loss at different epochs, with the final epoch producing the QF in **(a)**.

training points are in the Chebyshev grid arrangement as $\cos[(2n-1)\pi/(2N_{\text{points}})]$ ($n = 1, 2, ..., N_{\text{points}}$), this puts slight emphasis on training the distribution tails around $|z| \approx 1$. To make the feature map expressive enough that it captures full z-dependence for the trained initial QF, we use a tower-type product feature maps defined as $\hat{\mathcal{U}}_{\phi'}(z) = \bigotimes_{j=1}^{N} \exp[-i \arcsin(z) j \hat{Z}_j/2]$, where rotation angles depend on the qubit number j. For the training we again use a six-qubit register, and follow the same variational strategy as in the previous subsection. We initialise the circuit using an initialisation procedure as detailed earlier. We observe that a high-quality solution with a loss of $\sim 10^{-6}$ for $G(z, 0)$ can be obtained at the number of epochs increased to few thousands, and we find that pre-training with product states allows reducing this number to hundreds with identical quality.

The results are shown in Fig. 4.7, with the QF trained from data shown in Fig. 4.7(a) by the dashed curve that overlays the target QF. The circuit converges to 10^{-6} loss level [Fig. 4.7(b)], being close to the model expressivity limit of the feature map itself. Performing sample generation at the initial time, in Fig. 4.7(c) we observe good correspondence with the expected PDF (sampling procedure is the same as in Fig. 4.6). At the same time we note that small deviations in trained QF (and its derivative) lead to significant deviations for statistics, stressing the importance of expressive circuits and stable training. Using the trained QF as the initial condition, we evolve the system as before. Results are shown in Fig. 4.8. We see the same quality of propagation as the analytic case. We perform generative modelling at later points of time T_{mid} and T_{max}. The histograms in Fig. 4.8(d) confirm the high quality of sampling, and show that the approach is suitable for time series generation.

4.4.4 Initialisation

We make use of initialisation when training the quantile function on initial data as in Fig. 4.7. We note however that using initialisation is not a requirement for convergence. In Fig. 4.9 the results of training the quantile function on initial data without initialisation is shown. We can see that the loss value magnitude and the fit reached is similar to that achieved when training with initialisation. The difference is seen in the number of epochs — without initialisation more epochs are required to reach the same accuracy.

4.4.5 Qubit Number

Qubit number is an important factor of model performance with number required depending on other factors such as the feature map, data reuploading (how many layers of feature maps) and the problem considered. In general expressivity of the model scales with qubit number. Too few qubits can lead to

Qubit Number	Loss	KS
2	9.40	0.276
4	0.88	0.046
6	1.08	0.025
8	0.94	0.022
10	1.52	0.046

Table 4.1: **Table of convergence as qubit number increases.** Ornstein-Uhlenbeck process solved as in Fig. 4.5 for varying number of qubits. The resulting loss and the Kolmogorov-Smirnov measure between result and known solution are shown for each respective qubit number.

low-expressivity which in turn leads to underfitting. Too many qubits can lead to overfitting and therefore generalise poorly. Furthermore more qubits leads to more quantum resource required. While there are generic guidances based on structural risk minimisation [151], in practice the choice requires heuristics and understanding of possible implicit biases [152] (an ongoing research direction in QML).

To see the performance of our proposed method as the qubit number scales we run the same calculation as used in Fig. 4.5 but with varying number of qubits. As well as noting the final loss value from training we calculate the Kolmogorov–Smirnov (KS) test value for an out-of-training measure of model performance. The resulting loss and KS values are presented below.

We observe from the KS measure of performance that N = 6 and N = 8-qubit models have the best performance for our model, while N = 2, 4 and 10 experience underfitting or overfitting. Furthermore the model remains trainable as qubit number increases. However, more iterations are required. For instance, 250 iterations are used for $N = 6$ and 500 for $N = 10$. Thus, for the given problem and chosen product feature map the middle ground is using the $N = 6$ embedding. In Fig. 4.10 we show results as sampling histograms for the trained quantile function, at three time points when ten qubits are used. The best performing qubit number being six is a property of the problem and the hyperparameters used. For larger scale problems the qubit numbers required for preferable performance will

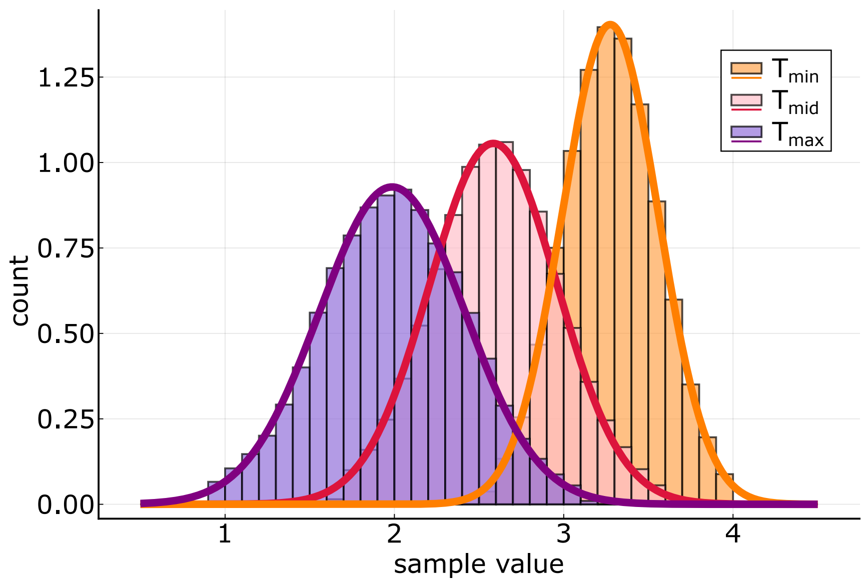


Figure 4.10: **Time-evolution of Ornstein-Uhlenbeck process with ten qubits.** Problem set up same as in Fig. 4.5 but with ten qubits and 500 iterations. Shown is the normalised histogram of samples from the data-trained QF, plotted against the analytic distribution (PDF). 100,000 random samples are drawn and bin counts are normalised such that the total area of each histogram is 1.

generally be much larger, at scales which classical computers cannot simulate easily.

4.4.6 Simulated Noise

Our results presented in the main text are simulated for an infinite number of shots. However, in physical implementations a limited number of shots is used leading to statistical noise in the read out. This typically follows a normal distribution in the limit of large number of shots. To check if our algorithm is robust in the presence of shot noise we perform an additional simulation, where noise is added to the function evaluation and its derivatives. Here we use the set up shown in Fig. 4.5. We add noise to function evaluations following $\mathcal{N}(\mu = 0, \sigma = 0.05)$. This is added to each circuit evaluation so terms made up of multiple circuit evaluations (such as derivatives) would have as many contributions of noise as circuit evaluations required in the calculation. We check that this is equivalent to the noise coming from function evaluations with 2000–3000 shots. No noise is added for measurements used for plotting as at this stage we are concerned with the training. We check separately that the quantum circuit is successfully trained to

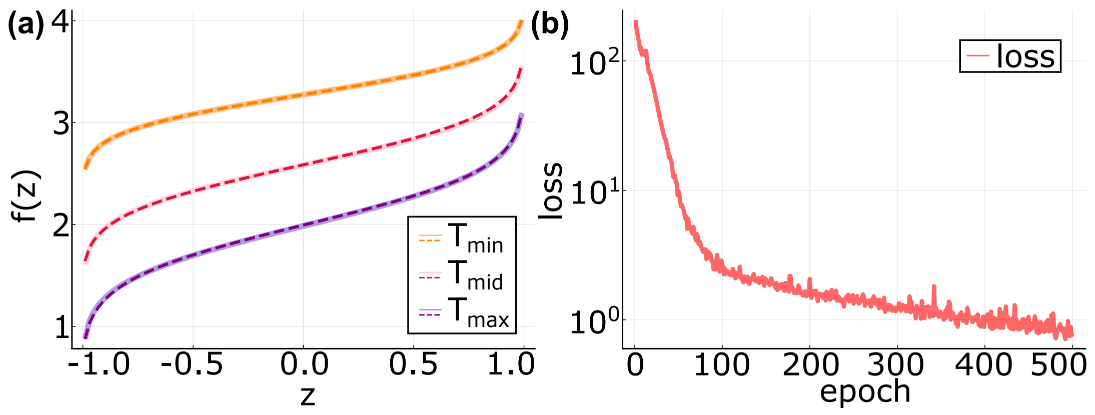


Figure 4.11: **Time-evolution of Ornstein-Uhlenbeck process with simulated limited shots.** Problem set up same as in Fig. 4.5 but with limited number of shots simulated by noise following $\mathcal{N}(\mu = 0, \sigma = 0.05)$ added to function evaluations during training. No noise added for final result plotting. (a) Slice of resulting surface for quantile functions at three time points (same as in main text). (b) Loss as a function of epoch from training resulting in (a).

represent the solution using the same training procedures as previously.

The results of this are shown in Fig. 4.11. As can be seen a result with a similar level of accuracy is achieved, as compared to the case of no noise. Here more epochs are used (500 vs. 250). Despite the loss being noisy (non-smooth) as function of epoch, we can recover a high quality solution.

4.5 QGAN and Quantile Functions

In this section, we analyse continuous quantum generative adversarial networks (QGANs), and show that they represent quantile functions with a modified (reordered) shape that impedes their efficient time-propagation. We investigate the connection between quantum quantile mechanics (QQM) and QGANs for SDE-based distributions, and whether QGAN structure could be used for time evolution and solving of SDEs.

We introduced QGANs in section 2.6.3 and now for an example look at specific application to an Ornstein-Uhlenbeck process. We follow the training strategy from Ref. [29]. We try to model the normal distribution with zero mean and standard deviation of 0.2. Both the discriminator and generator use $N = 6$ registers with the expressive Chebyshev tower feature map [112] followed by $d = 6$ HEA ansatz of the form as in Fig. 4.4. The readout for the generator uses the cost operator $\langle \hat{Z}_1 \rangle$, and the discriminator uses the same cost operator with post processing such that we readout $(\langle \hat{Z}_1 \rangle + 1)/2 \in [0, 1]$ modeling the probability. As before, we use `Adam` and train the QGAN for 2000 epochs using the loss function (2.27). Due to the minimax nature of the training, the loss oscillates and instead of reaching (global) optimum QGAN tries to reach the Nash equilibrium. Unlike QQM training, we cannot simply use variational parameters for the final epoch, and instead test the quality throughout. To get the highest quality generator we test how close together are the discriminator (L_{D}) and generator (L_{G}) loss terms by measuring the mean square distance between the two values. If they are within $\epsilon = 0.1$ distance we perform the Kolmogorov-Smirnov (KS) test [153] and check the distance between the currently generated samples and the training dataset. The result with minimal KS is chosen. We stress that KS is not used for training, and is exclusively for choosing the best result.

The results for QGAN training are shown in Fig. 4.12. A total of 10,000 samples from $N(0, 0.2)$ are used as training data. A different random subset of 1000 samples is used at each epoch for the loss evaluation. The trained generator $G_{\mathrm{Q}}(z)$ is shown as a function of the latent variable z. We note that it has a strongly-oscillating nature. The loss as a function of epoch is shown separately for the generator (blue curve) and discriminator (red curve) loss terms. They oscillate around the analytic value for the Nash equilibrium (black dotted line, `NE`), and briefly settle around NE after 1600 epochs where resultant circuit parameters are saved. We sample the QGAN generator using $N_{\mathrm{s}} = 100{,}000$ and plot the normalised histogram in Fig. 4.12(c). We observe that the distribution roughly matches the target (solid curve, PDF), though finer points are missing [cf. Fig. 4.7(c)], including the missing tail at negative values. In Fig. 4.12(b) the loss is shown and exhibits the oscillations that are often seen in QGAN training due to the competition in the minimax loss, and associated mode collapse phenomena [150]. There are further techniques that could be implemented to alter this such

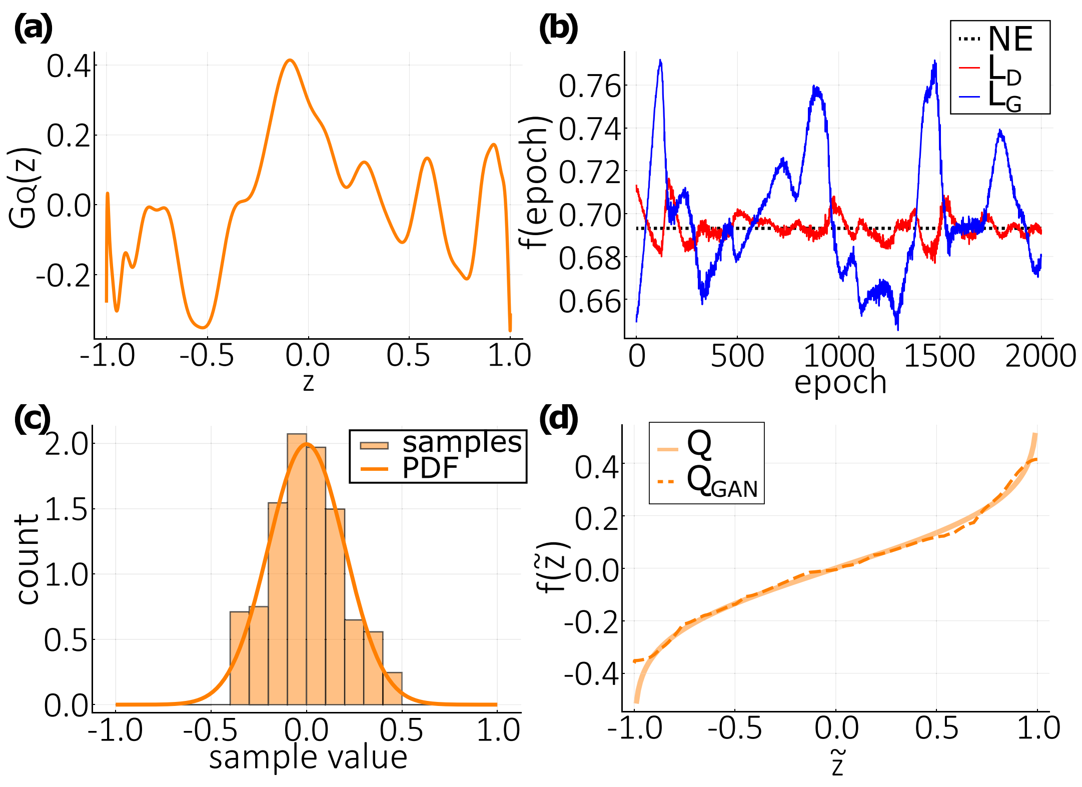


Figure 4.12: **QGAN training and fixed-time sampling. (a)** Generator function is shown for the optimal training angles. **(b)** Generator (L_G, red) and discriminator (L_D, blue) loss terms at different epochs. The Nash equilibrium at $-\ln(1/2)$ is shown by black dotted line (NE). **(c)** Normalised histogram for QGAN sampling ($N_s = 100{,}000$), as compared to the target normal distribution ($\mu = 0$, $\sigma = 0.2$). **(d)** Ordered quantile $Q_{\mathrm{GAN}}(\tilde{z})$ from the resulting QGAN generator shown in **(a)** (dashed curve), as compared to the true QF of the target distribution (solid curve).

as [154, 155]. Naturally, the generator of QGAN $G_{\mathrm{Q}}(z)$ does the same job as the trained quantile function $G(z)$ from previous subsections. We proceed to connect the two explicitly.

4.5.1 Reordered Quantile Functions and their DEs

The main difference between the quantile function and the generator of QGAN is that the true QF is a strictly monotonically increasing function, while the QGAN generator G_{Q} is not. We can connect them by noticing that the QGAN works with the latent variable $z \in \mathcal{Z}$, which we can rearrange into a QF by ordering the observations and assigning them the *ordered* latent variable $\tilde{z} \in \tilde{\mathcal{Z}}$ (both functions produce the same sample distribution). It is convenient to define a mapping $h : \tilde{\mathcal{Z}} \to \mathcal{Z}$ for G_{Q} which rearranges it into increasing form, $Q_{\mathrm{GAN}}(\tilde{z}) = G_{\mathrm{Q}}(h(\tilde{z}))$. In practice, finding $h(\tilde{z})$ requires the evaluation of $G_{\mathrm{Q}}(z)\ \forall\ z \in \mathcal{Z}$ and re-assigning the samples to values of $\tilde{\mathcal{Z}}$ in ascending order. Importantly, both h and its inverse $\mathrm{inv}[h] : \mathcal{Z} \to \tilde{\mathcal{Z}}$ can be defined in this process.

In Fig. 4.12(d) we show the results of reordering for the generator function in Fig. 4.12(a). The reordered quantile Q_{GAN} is plotted (dashed curve), approximately matching the target quantile (solid curve). We observe that the center of the quantile is relatively well approximated but the tails are not (particularly for $\tilde{z} < 0$). This agrees with what is observed in the sampling shown in Fig. 4.12(c). Having established the correspondence for QGAN-based generative modeling and quantile-based modeling we ask the question: can we apply differential equations to the quantile-like function to add differential constraints, and evolve the system in time enabling generative modelling?

The answer to the question above is far from trivial. To use a re-ordered QGAN quantile function for further training and time-series generation we need to account for the mapping when writing differential equations of quantile mechanics. Let us look into a specific case to develop an intuition on the behaviour of re-ordered quantile functions with differential equations. A quantile function $Q(\tilde{z})$ of a normal distribution with mean μ and standard deviation σ satisfies a quantile ODE [88, 130]

$$\frac{d^2Q}{d\tilde{z}^2} - \frac{Q-\mu}{\sigma^2}\left(\frac{dQ}{d\tilde{z}}\right)^2 = 0, \tag{4.12}$$

where we use the tilde notation $\tilde{z}$ to highlight that this is an ordered variable. Assuming perfect training such that $Q_{\mathrm{GAN}}(\tilde{z}) = G_{\mathrm{Q}}(h(\tilde{z}))$ closely matches $Q(\tilde{z})$, we substitute it into Eq. (4.12), and observe that the original QGAN generator obeys

$$\begin{aligned} &\frac{d^2G_{\mathrm{Q}}(z)}{dz^2} - \frac{G_{\mathrm{Q}}(z)-\mu}{\sigma^2}\left(\frac{dG_{\mathrm{Q}}(z)}{dz}\right)^2 \\ &\quad = \frac{\mathrm{inv}[h]''(z)}{\mathrm{inv}[h]'(z)}\frac{dG_{\mathrm{Q}}(z)}{dz}. \end{aligned} \tag{4.13}$$

The left-hand side (LHS) of Eq. (4.13) has the same form as for the true QF [cf. Eq (4.12)], but the right-hand side (RHS) differs from zero and involves derivatives of the inverted mapping function $\mathrm{inv}[h](z)$. This has important implications for training $G_{\mathrm{Q}}(z)$ with differential constraints, as the loss term includes the difference between LHS and RHS. Let us analyze the example of the quantile ODE in Eq. (4.13). In Fig. 4.12(a) we plot the LHS for $G_{\mathrm{Q}}(z)$ coming from the QGAN training. The result is a smooth function, and we expect all relevant terms, including derivatives $dG_{\mathrm{Q}}(z)/dz$ or $d^2G_{\mathrm{Q}}(z)/dz^2$, can be evaluated and trained at all points of the latent space. However, the problem arises when the RHS enters the picture. The additional term strongly depends on the contributions coming from inverse map derivatives $\mathrm{inv}[h]'$ and $\mathrm{inv}[h]''$. At the same time we find that the map from a non-monotonic to a monotonically increasing function is based on a multivalued function. And the inverse of the map (along with the map itself) is continuous but not smooth — it becomes non-differentiable at some points due to $G_{\mathrm{Q}}(z)$ oscillations.

As a toy-box example to see this we start by considering a quantile-like generator $G_{\mathrm{A}}(z)$ and use the mapping h that reorders it into ideal quantile function (QF) for the normal distribution. The mapping reads $h(\tilde{z}) = \pm(\tilde{z} + 1)/2$, and is shown in Fig. 4.13(a) as a multivalued function. It ensures that if we start from the normal QF, we arrive to $G_{\mathrm{A}}(z)$ with a single dip. The corresponding QGAN-like generator $G_{\mathrm{A}}(z)$ with a single dip is shown in Fig. 4.13(b) (we consider $\mu = 0$ and $\sigma = 0.2$). Our motivation is to understand how the presence of nonmonotonicity changes the behaviour of the system.

First, let us check that the reordering into increasing quantile function works as expected. Assigning the values of $G_{\mathrm{A}}(z)$ in the ascending order we get the reordered QF $Q_{\mathrm{A}}(\tilde{z})$. This is plotted in Fig. 4.13(c), matching the ideal $Q(\tilde{z})$ as expected. Once we have established the mapping for ideal re-ordering, let us look at its properties.

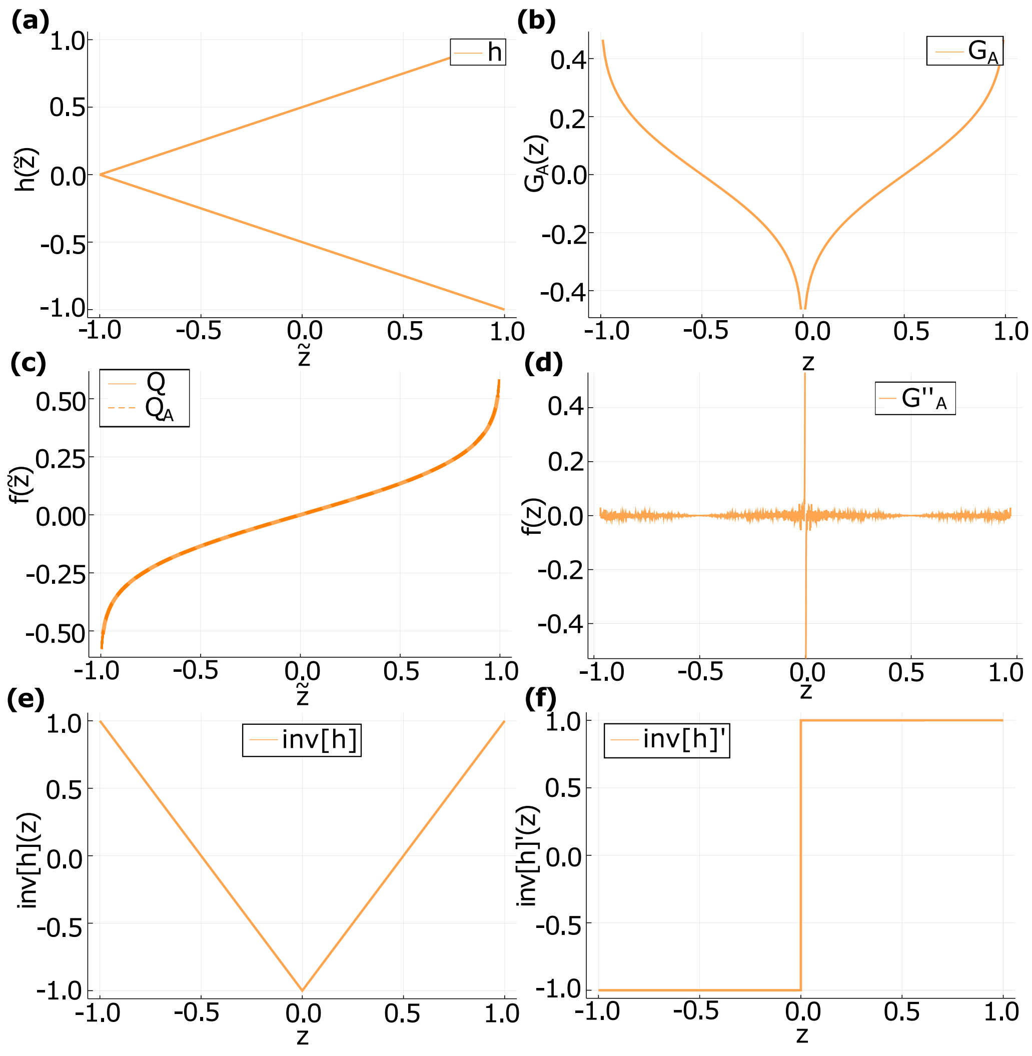


Figure 4.13: **QGAN quantile function reordering example.** **(a)** Mapping function $h(\tilde{z}) = \pm(\tilde{z}+1)/2$. **(b)** Non-ordered quantile function $G_{\mathrm{A}}(z) = Q(\mathrm{inv}[h](z))$ corresponding to mapping in **(a)** and normal quantile function. **(c)** Known quantile function (solid curve, Q) compared to numerical ordering of G_A (dashed line, Q_A). **(d)** Second derivative $\frac{d^2G(z)}{dz^2}$ plotted for G_{A} shown in **(b)**. Derivatives of $\mathrm{inv}[h]$ are calculated using finite differencing. **(e)** Inverse mapping function $\mathrm{inv}[h]$ plotted for different values of the latent variable z. We note the point of non-differentiability at $z = 0$. **(f)** Analytic first derivative of $\mathrm{inv}[h]$ that has a discontinuity at $z = 0$.

We discussed how the reordered quantile function from QGAN training matches the appropriate quantile function. We can perform the same check for the simple re-ordering presented above. Evaluating the difference between the RHS and LHS of Eq. (4.13) (akin to loss term) for $G_{\mathrm{A}}(z)$ and known mapping $h(\tilde{z})$, we observe that the difference remains zero everywhere (we have a perfect solution), apart from the middle point $z = 0$ where it diverges [Fig. 4.13(d)]. For calculating the derivatives of $\mathrm{inv}[h]$ we use finite difference (first-order forward Euler's

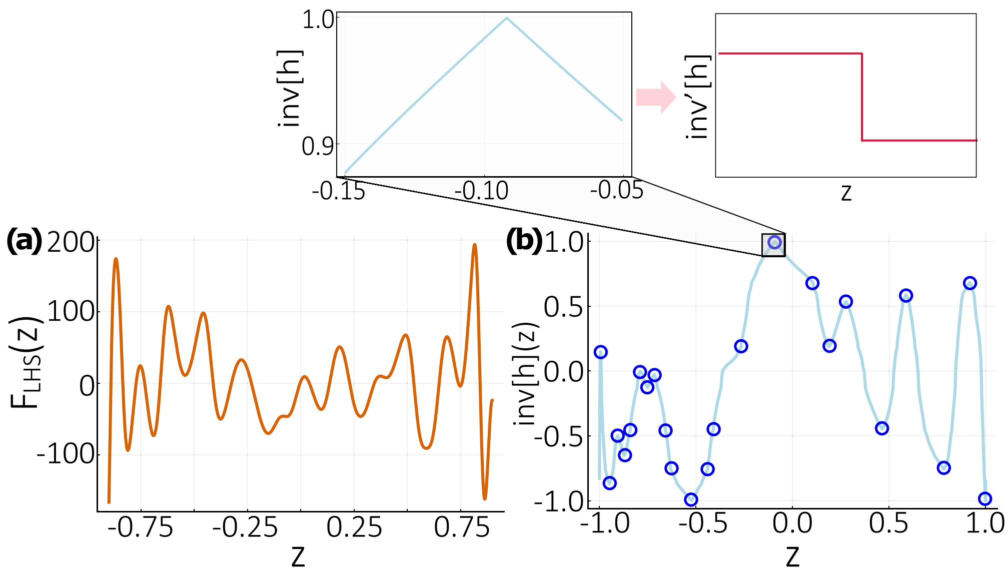


Figure 4.14: **QGAN quantile function analysis.** In **(a)** we plot the LHS of Eq. (4.13) for the trained QGAN generator $G_{\mathrm{Q}}(z)$ shown in Fig. 4.12(a). The resulting function is oscillatoric but smooth. **(b)** Inverse mapping function $\mathrm{inv}[h]$ shown as a function of z. It transforms $G_{\mathrm{Q}}(z)$ into the increasing quantile function $Q_{\mathrm{GAN}}(\tilde{z})$ that is plotted in Fig. 4.12(d). We highlight the non-differentiable points by blue circles, and zoom in on the characteristic behaviour in the inset above. The derivative is not defined at the discontinuity (top right inset).

method with a step of 10^{-5}), similarly for the QGAN case, thus observing small noise coming from numerical differentiation. The noise could be further reduced by considering alternative numerical differentiation schemes such as higher-order schemes or those more robust to step size such as the complex step rule [156]. The reason behind the unfavorable loss term behaviour can be tracked to the properties of the mapping function. We show the inverse mapping $\mathrm{inv}[h]$ plotted in Fig. 4.13(e), and its derivative $\mathrm{inv}[h]'$ is presented in Fig. 4.13(f). We see that $\mathrm{inv}[h]$ has a point of non-differentiability at $z = 0$ and $\mathrm{inv}[h]'$ is ill defined there. This provides the intuition behind the divergence. When training the DE-based loss for Eq. (4.13) with $z = 0$ included the loss becomes non-trainable. We stress that the same is observed for the non-ideal G_{Q}, where multiple non-differentiable points appear that we do not know in advance.

We see this behaviour in practice for the results from Fig. 4.12. As can be seen in Fig. 4.14, for these results the evaluation of the LHS of (4.13) results in a smooth function. For the RHS derivatives of $\mathrm{inv}[h]$ are involved. Fig. 4.14(b) shows $\mathrm{inv}[h]$ from training in Fig. 4.12, highlighting the points with non-analytic behaviour blue circles. The inset for Fig. 4.14(b) clearly shows the discontinuity. This translates to the absence of $\mathrm{inv}[h]'(z)$ at a set of points, which unlike zero derivatives cannot be removed by reshuffling the terms in the loss function. The discovered unlikely property of the mapping puts in jeopardy the attempts to use differential-based learning for QGAN generators. However, we do not currently think that it will be impossible to use QGAN for differential-based learning, it may be possible to engineer techniques to alleviate the found problems such as ways to exclude the set of "problem" points to yield stable training or to bias the QGAN to learn monotonic functions. While more studies are needed to estimate the severity of discontinuities and if techniques could be found, our interim conclusion is that quantile functions in the canonical increasing form are more suitable for evolution and time series generation.

4.6 Discussion

We proposed a distinct quantum algorithm for generative modelling from stochastic differential equations. Summarising the findings, we have developed the understanding of generative modelling from stochastic differential equations based on the concept of quantile functions. We proposed to represent the quantile function with a trainable (neural) representation, which may be classical- or quantum-based. In particular, we focused on parameterising the trainable quantile function with a differentiable quantum circuit that can learn from data and evolve in time as governed by quantile mechanics equations. Using Ornstein-Uhlenbeck as an example, we benchmark our approach and show that it gives a robust strategy for generative modelling in the NISQ setting. Furthermore, we notice that adversarial schemes as continuous QGAN lead to modified quantile-like function that potentially have intrinsic obstacles for evolving them in time.

For future implementation of this algorithm for more complex problems further work on choosing suitable feature maps that can accommodate the problem whilst remaining trainable is needed. One such approach could be encoding implicit biases [152] and utilising symmetries via geometric machine learning techniques [157, 33]. This is particularly suitable for quantile functions as they are monotonically increasing functions where derivatives can be constrained. Additionally, whilst we have shown some obstacles for QGAN type models in time, whether these could be mitigated/avoided or if they are fundamentally blocking is still to be answered.

We conclude by saying that the strategy we propose uses the large expressive power of quantum neural networks, and we expect that elements of the approach can be used for other architectures.

Chapter 5

Quantum Kernel Methods for Solving Regression Problems and Differential Equations

5.1 Declaration of Contribution

The work presented in this chapter is published in [158]. My contributions consist of the initial idea, the development and running of simulations and majority of manuscript writing.

5.2 Introduction

We now consider another approach to solving differential equations with a quantum computer. This is based on kernel methods which were discussed in section 2.8.2. Quantum kernel methods have previously been considered for various classification tasks. Classically, kernel methods are also applicable to regression and differential equations [85, 159, 160]. Therefore we work on creating and simulating quantum kernel methods for differential equations.

We propose two approaches for solving regression problems and differential equations (DEs) with quantum kernel methods. We compose quantum models as weighted sums of kernel functions, where variables are encoded using feature maps and model derivatives are represented using differentiation of quantum circuits. While previously quantum kernel methods primarily targeted classification tasks, here we consider their applicability to regression tasks, based on available data and differential constraints. We use two strategies to approach these problems. First, we devise a mixed model regression with a trial solution represented by kernel-based functions, which is trained to minimise a loss for specific differential constraints or datasets. Second, we use support vector regression that accounts for the structure of differential equations. The developed methods are capable of solving both linear and nonlinear systems. These two approaches for regression and differential equations will be introduced and compared. Simulated results are then shown.

Contrary to prevailing hybrid variational approaches for parametrised quantum circuits, we perform training of the weights of the model classically. This results in an approach utilising the expected increased expressivity of quantum models yet does not require quantum evaluation throughout training - only pre and post training. We also note that this can easily be adapted to include quantum parameters and consequently quantum evaluation during training but we focus on the first case. Under certain conditions this corresponds to a convex optimisation problem, which can be solved with provable convergence to global optimum of the model. The proposed approaches also favor hardware implementations, as optimisation only uses evaluated Gram matrices, but require quadratic number of function evaluations. We highlight trade-offs when comparing our methods to those based on variational quantum circuits such as the proposed differentiable quantum circuits (DQC) approach. The proposed methods offer potential quantum enhancement through the expected rich kernel representations using the power of quantum feature maps and their expressivity.

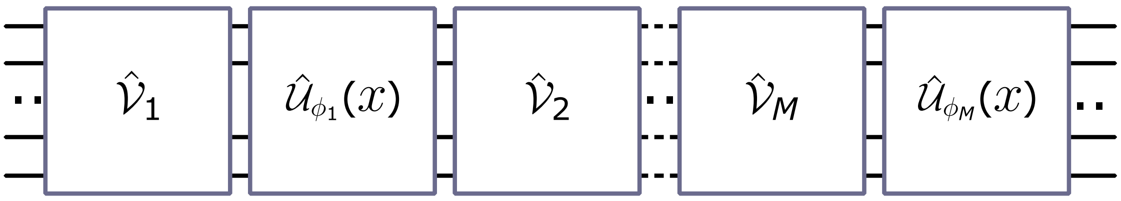


Figure 5.1: **General encoding circuit for kernel function.**Circuit diagram showing a general form of function encoding circuit $\mathcal{U}(x)$ used to implement quantum kernel. This is formed by layers of static circuits $\mathcal{V}_i$ and data re-uploading circuits $\mathcal{U}_{\phi_i}(x)$ parametrised by x.

5.3 Method

5.3.1 Quantum Model

Similar to our previous variational algorithms one of the first steps is to consider what quantum model we will use and how it forms a trial solution for the problem. We will find that the exact form of the trial solution depends on the problem and method considered but all are formed of quantum kernel functions (QK).

Earlier we mentioned a quantum kernel function of the form $\kappa(x, y) = \langle\psi(x)|\psi(y)\rangle$, being an inner product that is generally complex for quantum states. In the following, we consider $\kappa(x, y) = |\langle\psi(x)|\psi(y)\rangle|^2$ as an absolute value square of the overlap. This also corresponds to a valid kernel function [43]. We consider this kernel function as it is real valued — an advantage when expressing real valued functions.

The kernel functions we consider contain states $|x\rangle$, which are encoded by a classical variable x. To create such states we choose to use feature map encoding, where x is embedded into the state by parametrising gates preparing the state, $|\psi(x)\rangle = \mathcal{U}(x)|0\rangle$ as introduced in section 2.4. Other more complicated feature map encodings can be considered. A useful generalisation includes the re-uploading technique [36] where action of feature maps can be layered with (non-variational) entangling circuits, $\hat{\mathcal{U}}(x) = \hat{\mathcal{U}}_{\phi_M}(x)\hat{\mathcal{V}}_M...\hat{\mathcal{V}}_2\hat{\mathcal{U}}_{\phi_1}(x)\hat{\mathcal{V}}_1$ (see Fig. 5.1). This layered form terminates with a circuit encoded by a variable, as a

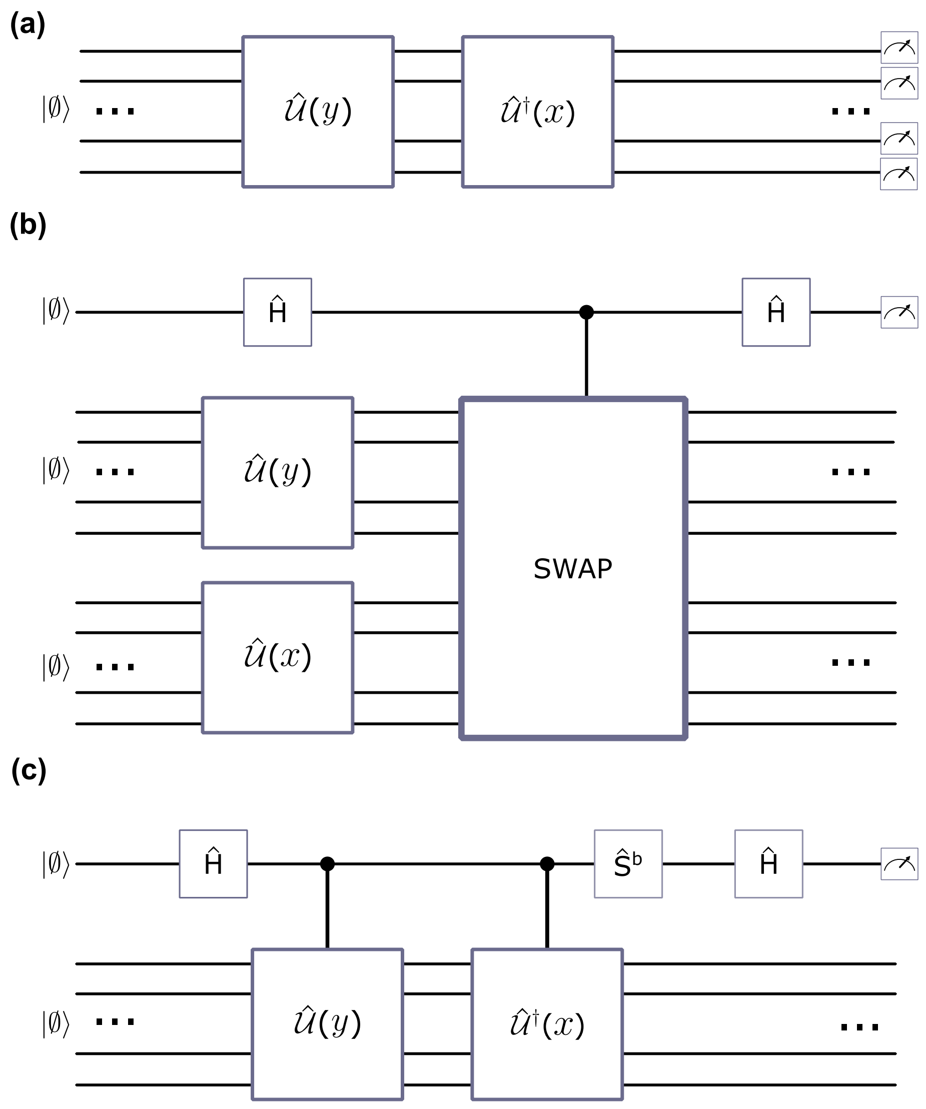


Figure 5.2: **Circuit diagrams for kernel function evaluation.** Circuit diagrams for evaluating the kernel $\kappa(x, y) = |\langle\psi(x)|\psi(y)\rangle|^2$, where in all circuits $\mathcal{U}$ and H represent the kernel feature map and the Hadamard gate, respectively. (a) Naive kernel evaluation based on consecutive application of $\mathcal{U}$ circuits, followed by measuring each qubit. The kernel value is inferred from a probability of returning to the initial state. (b) SWAP test measuring $|\langle\psi(x)|\psi(y)\rangle|^2$. The controlled SWAP onto the size $2N$ register is composed of qubitwise controlled SWAP on the n^{th} qubit pair, repeated for $n \in 1 : N$. (c) Hadamard test measuring Re($\langle 0|\mathcal{U}^\dagger(x)\mathcal{U}(y)|0\rangle$) and Im($\langle 0|\mathcal{U}^\dagger(x)\mathcal{U}(y)|0\rangle$) for $b = 0$ and $b = 1$, respectively. S denotes the phase gate, $\exp(-\pi Z/4)$.

final entangling circuit would cancel itself out out for kernels based on $\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{U}}(y)$. As with many variational algorithms, when choosing feature maps it is important to have a map expressible enough to represent the solution to the problem whilst also being trainable [151].

We now discuss how to implement the quantum kernel function $\kappa(x, y) = |\langle\psi(x)|\psi(y)\rangle|^2$. One way is to use the coherent SWAP test [161, 162]. This test requires $2N + 1$ qubits, where N is the number of qubits used to express $|\psi(x)\rangle$. $|\psi(x)\rangle$ and $|\psi(y)\rangle$ are both prepared on separate registers then via Hadamard gates and controlled operations an ancillary qubit can then be measured to read $|\langle\psi(x)|\psi(y)\rangle|^2$. With this approach only one qubit has to be read and the controlled operations is limited to the SWAP gate. The circuit diagram is shown in Fig. 5.2(b).

We can also employ other methods. For this, we use the fact that the kernel evaluation can be written as

$$|\langle\psi(x)|\psi(y)\rangle|^2 = \langle 0|\hat{\mathcal{U}}^\dagger(y)\hat{\mathcal{U}}(x)|0\rangle\langle 0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{U}}(y)|0\rangle. \tag{5.1}$$

The measurement in Eq. (5.1) can be implemented naively by the circuit in Fig. 5.2(a). The circuit is initialised in the zero state. Then $\hat{\mathcal{U}}(y)$ is applied, followed by $\hat{\mathcal{U}}^\dagger(x)$. The probability of remaining in the zero state and consequently the kernel function value is then calculated by measuring all qubits and finding the fraction of times $|0\rangle$ is measured.

Another possible implementation is two evaluations of the Hadamard test with $N + 1$ qubits as shown in Fig. 5.2(c) [70] as was introduced in section 2.7.1 to measure overlaps. This can be used to evaluate the real and imaginary parts of $\langle 0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{U}}(y)|0\rangle$ which can then be used to evaluate the kernel as $\mathrm{Re}(\langle 0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{U}}(y)|0\rangle)^2 + \mathrm{Im}(\langle 0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{U}}(y)|0\rangle)^2$.

Derivatives

As our goal is to solve differential equations, we need to be able to evaluate derivatives of the kernel function. We introduce notation for the derivatives as follows,

$$\nabla_n^m \kappa(x, y) = \frac{\partial^{m+n}\kappa(x, y)}{\partial x^n \partial y^m}. \tag{5.2}$$

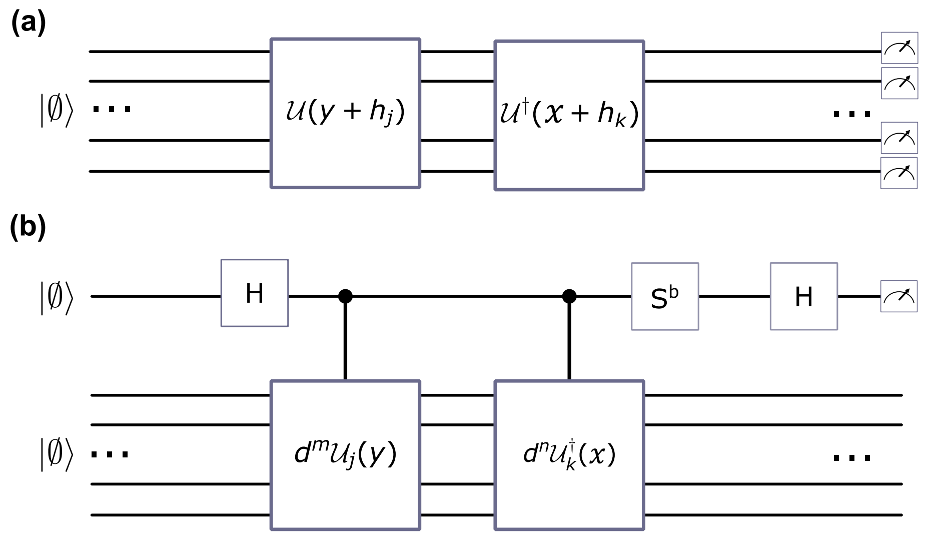


Figure 5.3: **Circuit diagrams for evaluation of derivatives of kernel functions**. (a) Generic circuit for differentiating kernels shown in Fig. 5.2(a) where a parameter shift rule is used. Depending on which derivative is calculated, gates parametrised by x and/or y have their parameters shifted up and down (shown by h_j and h_k). Contributions from all parametrised gates are then summed for overall derivative. (b) Using Hadamard test for evaluation of the overlap $\langle 0|d^n/dx^n\mathcal{U}_k^\dagger(x)d^m/dy^m\mathcal{U}_j(y)|0\rangle$, where k and j index over which gates with x and/or y as parameters are differentiated. When $b=0$ and $b=1$ are used the real and imaginary part is evaluated. By summing over j,k the full overlap $\langle 0|d^n/dx^n\mathcal{U}^\dagger(x)d^m/dy^m\mathcal{U}(y)|0\rangle$ can be evaluated. These overlaps can then be used to evaluate kernel derivatives.

To implement derivative evaluation, one way is to consider the kernel as written in Eq. (5.1) and the parameter shift rule [39, 28]. The parameter shift rule was introduced in section 2.4.4. With this method, we take the kernel evaluation method as in Fig. 5.2(a) but shift x and y up and down depending on what derivative is being calculated in each gate that they parametrise. For example for the first order derivative with respect to x the number of evaluations of Fig. 5.3(a) is $2k$ with k being the number of gates parametrised by x. Using the parameter shift rule means we calculate the analytic derivative though it does place some requirements on the gates parametrised by x and y such as the generator being involutory. Generalised parameter shift rules are possible, where such requirements are relaxed [40, 41, 163, 164, 165].

We can also implement derivatives via the Hadamard test. First, we note the form of the first-order derivative of the kernel in x by using the product rule in

Eq. (5.1) as

$$\frac{\partial}{\partial x}\kappa(x,y) = \langle 0|\hat{\mathcal{U}}^\dagger(y) d/dx(\hat{\mathcal{U}}(x))|0\rangle\langle 0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{U}}(y)|0\rangle + \langle 0|\hat{\mathcal{U}}^\dagger(y)\hat{\mathcal{U}}(x)|0\rangle\langle 0|d/dx(\hat{\mathcal{U}}^\dagger(x))\hat{\mathcal{U}}(y)|0\rangle, \tag{5.3}$$

and thus we can evaluate this derivative by evaluating $\langle 0|\hat{\mathcal{U}}^\dagger(y)d/dx(\hat{\mathcal{U}}(x))|0\rangle$ and $\langle 0|\hat{\mathcal{U}}^\dagger(x)\hat{\mathcal{U}}(y)|0\rangle$ (real and imaginary parts). The second term can be evaluated the same as for evaluating the kernel shown in Fig. 5.2(b), and calculations can be reused because the derivatives are evaluated over the same set of points as the kernel itself. To calculate the first term a modified Hadamard test can be used.

For such a modification we consider the generalised layered form of kernel encoding $\hat{\mathcal{U}}(x) = \hat{\mathcal{U}}_{\phi_M}(x)\hat{\mathcal{V}}_M ... \hat{\mathcal{V}}_2\hat{\mathcal{U}}_{\phi_1}(x)\hat{\mathcal{V}}_1$ with each feature map being $\hat{\mathcal{U}}_{\phi_j}(x) = \exp(-iG_j\phi_j(x))$. The derivative then reads $\mathcal{U}'(x) = \sum_{j=1}^{M} \hat{\mathcal{U}}_{M:j+1}\hat{\mathcal{U}}_{\phi_j}(-iG_j)\phi_j'(x)\hat{\mathcal{V}}_j\hat{\mathcal{U}}_{j-1:1}$ with $\hat{\mathcal{U}}_{j:k} = \hat{\mathcal{U}}_{\phi_j}(x)\hat{\mathcal{V}}_j\hat{\mathcal{U}}_{\phi_{j-1}}(x)...\hat{\mathcal{U}}_{\phi_k}(x)\hat{\mathcal{V}}_k$. We can now assume that the generators G_j are unitary. When G_j are unitary we can calculate each overlap term in the derivative expansion with two Hadamard tests. However, if this is not the case, one can decompose them into sums of unitary terms and evaluate them separately with increased number of Hadamard tests [39].

Once the procedure for evaluating derivatives being set up, we generalise to higher-order derivatives. By using the product rule in Eq. (5.1) whatever derivative is required, one can express it as sums of products of overlaps with $\hat{\mathcal{U}}(x)$ and $\hat{\mathcal{U}}(y)$ differentiated to different orders. These overlaps can be calculated with two (when generators unitary) overlap tests for each gate with x and/or y as a parameter (see Fig. 5.3). These overlap evaluations can be reused for calculating different derivatives where the same overlap occurs.

5.3.2 Regression

Our goal is to use the quantum kernel functions to solve regression problems and differential equations. We consider two main methods — mixed model regression (MMR) and support vector regression (SVR)[85]. Let us first consider using these methods to solve data-driven regression problems. This is a simpler case than solving differential equations yet still requires representing a solution function via quantum kernel functions, and can be built upon to solve differential equations. For this regression problem we have a set of values $\{x_i, f_i\}_i$ and we want to find a function $f(x)$ that fits these points such that $f_i = f(x_i)$. These approaches are very similar to the classical version with quantum contributions entering through the quantum kernel function as discussed earlier. We consider how both MMR and SVR approach the described problem.

Mixed Model Regression

When using the mixed model regression we represent a trial function as

$$f_\alpha(x) = b + \sum_{i=1}^{|\mathbf{y}|} \alpha_i \kappa(x, y_i), \tag{5.4}$$

where $\mathbf{y} = \{y_i\}_i$ is a set of evaluation points, $|\mathbf{y}|$ denotes the size of set $\mathbf{y}$, and $\boldsymbol{\alpha} = \{\alpha_i\}_i$ and b are tunable, classical coefficients. We then choose a loss function corresponding to the problem such as $\mathcal{L}(\alpha) = \sum_{i=1}^{|\mathbf{x}|} (f_\alpha(x_i) - f_i)^2$. The loss function is chosen such that when optimised with respect to α and b the corresponding $f_\alpha(x)$ fit the regression problem. There are multiple different valid loss functions for a given problem, we have used mean square error (MSE) for our loss function.

The loss function requires the evaluation of $\{f_\alpha(x_i)\}_i$, which in turn requires the evaluation of $\{\kappa(x_i, y_j)\}_{i,j}$. These evaluations are independent of $\boldsymbol{\alpha}$ — the variable which is adjusted during optimisation. This means that the kernel function will only need to be evaluated once for each point in $\{x_i, y_j\}_{i,j}$ at the start of the optimisation procedure. Any suitable optimisation method may be used to optimise $\mathcal{L}(\boldsymbol{\alpha})$. We can also see that the considered loss function for regression is convex.

We consider the general case $\mathcal{L}(\boldsymbol{\alpha}) = \sum_{i=1}^{|\mathbf{x}|} L(x_i; \boldsymbol{\alpha})^2$ with L being a linear function of $\boldsymbol{\alpha}$, and represents a measure of how well the current trial function solves the problem at a training point. A sufficient condition for convexity of the loss function is $\partial^2\mathcal{L}/\partial\alpha_j^2 \geq 0$ everywhere for all $\alpha_j \in \boldsymbol{\alpha}$. We can write the second-order derivatives as

$$\frac{\partial^2\mathcal{L}}{\partial\alpha_j^2} = \sum_{i=1}^{|\mathbf{x}|}\left[2\left(\frac{\partial L}{\partial\alpha_j}\right)^2 + L\frac{\partial^2 L}{\partial\alpha_j^2}\right] = \sum_{i=1}^{|\mathbf{x}|} 2\left(\frac{\partial L}{\partial\alpha_j}\right)^2 \geq 0, \tag{5.5}$$

where passing from second to third expression we use the linearity of L in α. When a loss function is convex its minimum is global, and there are bounds on convergence for various optimisation methods [86].

The workflow to solve an MMR problem is as follows:

1. Choose setup for training, including the kernel function, optimiser, $\mathbf{x}$, $\mathbf{y}$.
2. Identify the loss function for problem considered.
3. Calculate set of kernel function evaluated over $\mathbf{x} \otimes \mathbf{y}$.
4. Optimise the loss function.

Once the model is trained, we can also evaluate it at a grid of points different from the training grid, learning the solution in the full domain of x.

Support Vector Regression

The method of how to format a regression problem for SVR was introduced in section 2.8.2. As a recap, the resulting system of equations is

$$\left[\begin{array}{c|c} \hat{\Omega} + \hat{I}/\gamma & \mathbf{1} \\ \hline \mathbf{1}^T & 0 \end{array}\right]\left[\begin{array}{c} \boldsymbol{\alpha} \\ \hline b \end{array}\right] = \left[\begin{array}{c} \mathbf{f} \\ \hline 0 \end{array}\right], \tag{5.6}$$

where $\Omega_{i,j} = \kappa(x_i, x_j)$, $\hat{\Omega} = \{\Omega_{i,j}\}_{i,j}$, and $\boldsymbol{\alpha}$ are a set of introduced dual variables. The associated model of the solution is

$$f(x;\alpha) = \sum_{i=1}^{|\mathbf{x}|} \alpha_i \kappa(x, x_i) + b. \tag{5.7}$$

This follows from the workflow to prepare an SVR problem as follows:

1. Write model with minimisation function and constraints.

2. Write out Lagrangian.

3. Find the KKT optimality conditions.

4. Eliminate subset of original optimisation variables.

5. Use the kernel trick to realise problem in terms of kernels.

6. Write out remaining relationships as system of equations.

7. Use KKT conditions and kernel trick to express function in terms of kernel functions.

The prepared SVR model can then be used for any problem of the form assumed in preparing the original model. The workflow for solving an SVR problem is as follows:

1. Choose setup for training, including the kernel function, system of equations solver, $\mathbf{x}$, γ.

2. Identify suitable SVR model for the problem considered.

3. Calculate set of kernel function evaluated over $\mathbf{x} \otimes \mathbf{x}$.

4. Solve system of equations.

We note that the SVR method results in a form of problem that can still be considered as an optimisation problem to be solved with an optimiser. The system of equations $Ax = b$ can be translated into the loss function $\mathcal{L}(x) = \sum_i [(Ax)_i - b_i]^2$. Here we use MSE loss but other forms can be employed. This formulation can be especially useful when considering problems resulting in nonlinear systems of equations.

Comparison

Comparing the MMR and the SVR methods we note that the solving workflow for the two are similar. Namely, we choose a setup, identify what to solve based on method and problem, calculate the set of kernel function evaluations, and solve the model identified in step two. However, identifying the model for the SVR method is a more involved process.

Both MMR and SVR result in a function approximation to the solution of the problem considered. For regression this is Eq. (5.4) and Eq. (5.7), respectively. These two functions look very similar with the difference being the kernel evaluation at y_i for MMR versus x_i for SVR. This is a consequence of using the kernel trick when formulating the SVR model, which necessarily results in $\mathbf{y} = \mathbf{x}$. Also to be highlighted is that the form of Eq. (5.7) depends on the problem considered. For example, later we see that when solving differential equations, evaluations of the kernel derivative are involved in the function expression. However, for MMR the form of the model remains the same no matter what problem considered.

One benefit of using the MMR model is the simpler identifying of the model to solve. Another is the convexity when considering certain problems. The benefit of SVR is that when linear the resulting system of equations to solve is also linear and thus has a deterministic solution obtainable ina single step. Furthermore, the initial trial function is in terms of φ which can be of higher dimensionality than the kernel function yet never needs to be evaluated directly.

5.3.3 Solving Differential Equations

In the following text, we collect the described tools for model and derivative evaluations, and apply them to solve differential equations. While there are many possible choices, we start by considering a simple class given by the differential constraint

$$\mathrm{DE}(x, f, df/dx) = \frac{df}{dx} - g(x, f) = 0, \tag{5.8}$$

with initial condition $f(x_0) = f_0$, and g a smooth function of x and f which in general can be nonlinear in either of those arguments. We now use both MMR and SVR to solve this type of DEs in next subsections.

5.3.4 Mixed Model Regression

When solving DEs of the type (5.8) via MMR, we choose a loss function in the form $\mathcal{L}(\alpha) = \sum_{i=1}^{|x|} [\mathrm{DE}(x_i, f_\alpha(x_i), df_\alpha/dx(x_i))]^2 + (f_\alpha(x_0) - f_0)^2$. We remind that the trial function reads $f_\alpha(x) = b + \sum_{i=1}^{|y|} \alpha_i \kappa(x, y_i)$. Therefore, kernels κ and their derivatives $\nabla_1^0 \kappa$ are evaluated over $\{x_i, y_j\}_{i,j}$, leading to corresponding f_α and df_α/dx evaluations. These values are independent of α, and only need to be evaluated once at the start, then being reused throughout optimisation. The loss function can then be optimised via any appropriate optimiser for getting optimal weights α_{opt}. The resulting function is then a suitable approximation to the solution of the differential equation, mainly being limited by expressivity of the model and generalisation bounds.

When the differential equation is linear [i.e., g is linear in f in Eq. (5.8)] the considered loss function is convex. This is true when the differential equation is linear and f_α (and consequently f'_α) is linear in α, meaning we are in the situation as described by Eq. (5.5). When the differential equation is nonlinear this is *not necessarily* the case. In order to determine that one needs to check for the convexity of the loss function. One possibility is a numerical check by sampling the second derivatives of the loss with respect to the optimizable parameters at many locations. If this value is ever negative then the problem is non-convex.

5.3.5 Support Vector Regression

In appendix A how to form an SVR problem for a given differential equation is explained. The overall process is the same as regression: state a model, write out Lagrangian, find KKT optimality conditions, eliminate subset of prime variables by using the KKT conditions, use the kernel trick, and finally write out remaining equations in matrix form. But some specifics do change based on type of DE [159, 160, 166].

As an example the SVR formulation procedure for problems of the form $\mathrm{DE}(x, f, df/dx) = df/dx - g(x, f) = 0$ with initial condition $f(x_0) = f_0$ is detailed in appendix A and results in

$$\left[\begin{array}{c|c|c|c|c} \tilde{\Omega}_1^1 & \Omega_0^1 & \mathbf{h}_0^1 & \mathbf{0} & \hat{0} \\ \hline \Omega_1^0 & \tilde{\Omega}_0^0 & \mathbf{h}_0^0 & \mathbf{1} & -\hat{I} \\ \hline {\mathbf{h}^T}_1^0 & {\mathbf{h}^T}_0^0 & \tilde{h} & 1 & \mathbf{0}^T \\ \hline \mathbf{0}^T & \mathbf{1}^T & 1 & 0 & \mathbf{0}^T \\ \hline \hat{D} & \hat{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right] \left[\begin{array}{c} \boldsymbol{\alpha} \\ \hline \boldsymbol{\eta} \\ \hline \beta \\ \hline b \\ \hline \mathbf{y} \end{array}\right] = \left[\begin{array}{c} \tilde{\mathbf{g}} \\ \hline \mathbf{0} \\ \hline f_0 \\ \hline 0 \\ \hline \hat{0} \end{array}\right]. \tag{5.9}$$

Here, we introduced dummy variables y_i. $\boldsymbol{\eta}$ and β are dual variables introduced along with $\boldsymbol{\alpha}$ corresponding to the dummy variable constraint and the initial variable constraint, respectively. The remaining notation is as follows

$$[\Omega_n^m]_{i,j} = \nabla_n^m \kappa(x_j, x_i), \tag{5.10}$$

$$\tilde{\Omega}_n^m = \Omega_n^m + \hat{I}/\gamma, \tag{5.11}$$

$$[\mathbf{h}_n^m]_i = \nabla_n^m \kappa(x_0, x_i), \tag{5.12}$$

$$\tilde{h} = \kappa(x_0, x_0), \tag{5.13}$$

$$\hat{D} = \mathrm{diag}\left(\left\{\frac{\partial g}{\partial f}(x_i, y_i)\right\}_i\right), \tag{5.14}$$

$$[\tilde{\mathbf{g}}]_i = g(x_i, y_i). \tag{5.15}$$

We now have a set of generally nonlinear equations, which can be solved for finding a vector of optimised weights. Any suitable method for solving a system of nonlinear equations can be utilised, in our later examples we optimise for the solution. By substituting the relevant KKT optimality conditions into $f(x) = b + \sum_{i=1}^{|y|} \alpha_i \kappa(x, y_i)$ and employing the kernel trick, we get an expression for f in terms of kernel functions

$$f(x) = \sum_{i=1}^{|\mathbf{x}|} \alpha_i \nabla_1^0 \kappa(x_i, x) + \sum_{i=1}^{|\mathbf{x}|} \eta_i \kappa(x_i, x) + \beta\kappa(x_0, x) + b, \tag{5.16}$$

where optimised variables (weights) are used. Note that if the differential equation is linear, y_i will not appear in both the matrix and the vector to be solved for. This leads to a system of linear equations with lower dimension. This, similar to regression is then solved with any suitable method. We focus on variational approach.

5.3.6 Other Forms of Differential Equations

Many practical problems are not of the form $\mathrm{DE}(x, f, df/dx) = df/dx - g(x, f) = 0$ considered above. For instance, they may include terms of higher order, higher dimension, or indeed many other different variations. When considering the MMR method, one can readily generalise to any other form of DE simply relying on generalised optimisation. For this, a suitable loss function needs to be formulated for the chosen equation. Additionally, we shall be able to evaluate each term of the differential equation. For systems of DEs the overall loss becomes the sum of the loss of each individual differential equation within the system. For PDEs with domains of more than one dimension, the kernel function can be considered as $\kappa(\mathbf{x}, \mathbf{y}) = |\langle 0|\hat{\mathcal{U}}^\dagger(\mathbf{x})\hat{\mathcal{U}}(\mathbf{y})|0\rangle|^2$, where the feature maps now encode a vector of domain variables. The simplest form is $\hat{\mathcal{U}}(\mathbf{x}) = \hat{\mathcal{U}}(x_1)\hat{\mathcal{U}}(x_2)...\hat{\mathcal{U}}(x_M)$ with $M = |\mathbf{x}|$.

When the SVR method is used, the considered problem needs to be formulated into the SVR form, resulting in a different form of matrix equation. Higher order derivative SVRs [159, 166] and SVRs for PDEs [160] are possible, as well as their generalisations for systems of differential equations. An example of solving a second order differential equation and the resulting SVR formulation is presented within the following results section.

5.4 Results

Having established quantum kernel approaches for solving DEs and learning from data, we apply them to specific problems and show the results.

5.4.1 Regression on Quantum Data

We start by considering the case of regression. We generate a quantum dataset that corresponds to dynamics of total magnetisation of a biased honeycomb Kitaev model [167, 168]. The Hamiltonian of the system reads

$$\mathcal{H} = J\left(\sum_{\langle i,j\rangle\in\mathcal{X}} \hat{X}_i\hat{X}_j + \sum_{\langle i,j\rangle\in\mathcal{Y}} \hat{Y}_i\hat{Y}_j + \sum_{\langle i,j\rangle\in\mathcal{Z}} \hat{Z}_i\hat{Z}_j\right) + h_z\sum_{j=1}^{N}\hat{Z}_j, \tag{5.17}$$

where $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$ are sets of bonds. We choose antiferromagnetic coupling and set $h_z/J = 0.2$. Specifically, we simulate nonequilibrium effects by performing time evolution of $M_z = \sum_j \hat{Z}_j/N$ for $N = 12$ qubits on a lattice with periodic boundary conditions [169], starting from the uniform initial state. The choice of quantum dataset with strong magnetic correlations may be especially suitable for kernel-based regression, given recent advances in learning from experiments [170]. Choosing a subset of evolved magnetisation values labeled by x values (here corresponding to time), we proceed to perform MMR.

When implementing the MMR method we consider $\mathbf{x}$ with 51 values of x between 0 and 10, associated to the data, and $\mathbf{y} = \mathbf{x}$. We use and compare the results from a classical kernel and a quantum kernel function. The classical kernel used is a commonly used radial basis function (RBF) kernel $\kappa(x, y) = \exp\left[(x - y)^2/(2\sigma^2)\right]$, with σ being a hyperparameter that describes a width of the kernel. In calculations we choose $\sigma = 0.2$ as that shows favorable performance. For the quantum kernel, we use layers of depth-five hardware-efficient ansatz (HEA) and feature maps based on parametrised $\hat{X}$ rotations, $\hat{R}_X(\phi(x))$, acting on each qubit. We set $\phi(x) = qx/2$, where q is the qubit index, and consider a register of eight qubits. For the loss function MSE is used with a pinned boundary formulation (see [112] for the details of boundary handling). The loss function for data regression is convex and is optimised via Newton's method. In this case just three epochs is enough for converging to low loss values. We model the system with full state simulation using the `Julia`'s package `Yao.jl` [110]. The error of the results of this are shown in Fig. 5.4(b) with associated loss in Fig. 5.4(a). As

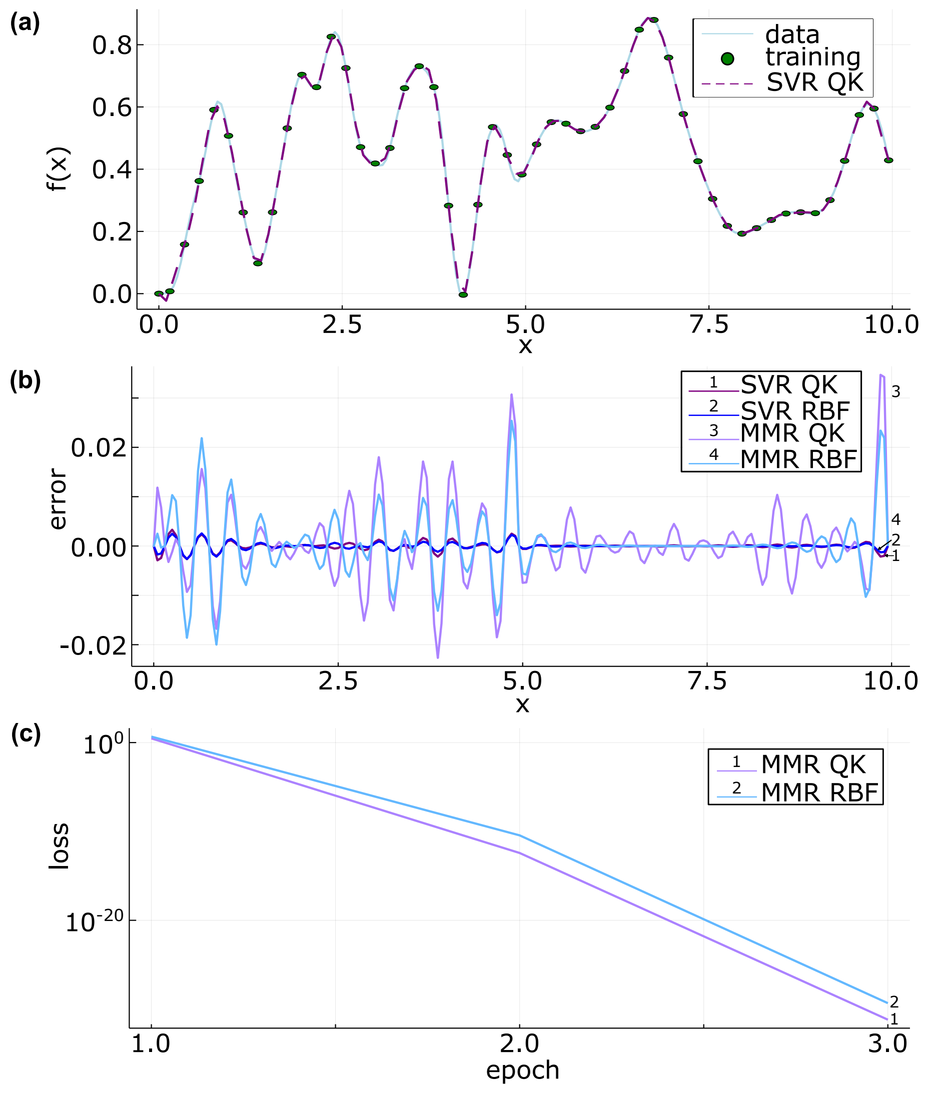


Figure 5.4: **MMR and SVR used to solve a regression problem. Data to fit is for time-evolved magnetisation of a biased honeycomb Kitaev model.** (a) Solution via SVR method for quantum kernel function with two layers and $N = 8$ shown by dashed purple line. Data plotted as solid light blue curve with points used for training highlighted with green circles. (b) Error between results and underlying data plotted over x as $[f(x) - f_{\text{data}}(x)]$, and we additionally normalise data by the range of magnetisation values. Error plotted for result shown in (a) and SVR method with classical RBF kernel with $\sigma = 0.2$. Also plotted results from MMR method with same kernels considered. Newton optimiser with just 3 epochs is used for MMR method. (c) Loss value over epoch number plotted for MMR results shown in (a) and (b).

can be seen, both kernel types are able to closely approximate the considered function. Moreover, we note that for complicated quantum data coming from spin-spin correlation one can benefit from specifically-designed quantum kernels that account for the structure of the problem.

When implementing the SVR method, we use the same points $\mathbf{x}$, kernel functions and the simulation package. The resulting SVR system of equations to solve for this form of problem are as shown in Eq. (5.6). The results of this with the quantum kernel are shown in Fig. 5.4(a). The error of the results is shown in Fig. 5.4(b). It can be seen that both kernel types are able to closely approximate the considered function, and that SVR outperforms the MMR method which we believe to be an affect of the different methods of solving optimisation for MMR versus ”exact” linear algebra computations for SVR.

5.4.2 Solving Linear Differential Equations

Next, we consider solvers of linear differential equations. In particular, we solve the equation

$$\frac{df}{dx} = -\lambda\kappa f - \lambda\exp(-\lambda\kappa x)\sin(\lambda x), \tag{5.18}$$

where parameters are chosen as $\lambda = 20$ and $\kappa = 0.1$, along with initial condition $f(0) = 1$. The analytic solution to the differential equation (5.18) is $f_{\text{sol}}(x) = \exp(-\lambda\kappa x)\cos(\lambda x)$, being a fading oscillatory dependence.

When implementing the MMR method we consider $\mathbf{x}$ and $\mathbf{y}$ of 21 points uniformly spaced over $[0, 1]$. We use and compare the results from a classical RBF kernel with $\sigma = 0.2$ and a quantum kernel with two layers of HEA circuits (depth equal to five) followed by feature map of $R_x(\phi(x))$ on each qubit with $\phi(x) = qx/2$, where q is the qubit index. We consider eight qubits in the register. For the loss function MSE is used with a pinned boundary. This loss function is convex, as the DE is linear, and is optimised via Newtons method. The error of the results are shown in Fig. 5.5(b) with corresponding loss in Fig. 5.5(c). As can be seen both kernel types are able to closely approximate the considered function with the error less than 0.002 in magnitude, the quantum kernel slightly outperforms the classical kernel although we did not further explore hyperparameter optimisation.

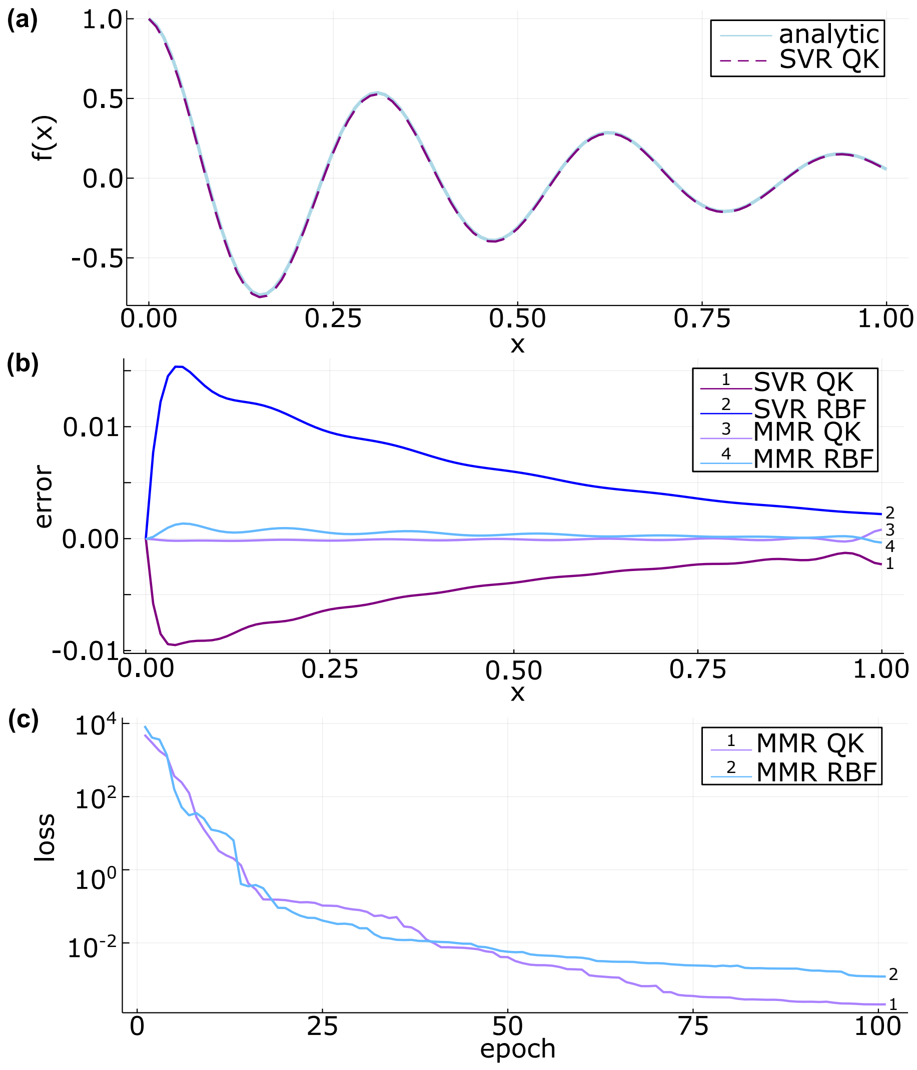


Figure 5.5: **MMR and SVR used to solve a linear differential equation** (5.18)**.** $\lambda = 20$**,** $\kappa = 0.1$ **and** $f(0) = 1$**.** (a) Solution via SVR method for quantum kernel with two layers and $N = 8$ shown by dashed purple line. Known analytic solution plotted with solid light blue line. (b) Error between results and analytic solution plotted over x as $(f(x) - f_{\rm sol}(x))/{\rm range}(f_{\rm sol})$. Error plotted for result shown in (a) and SVR method with classical RBF kernel with $\sigma = 0.2$. Also plotted results from MMR method with same kernels considered. Newton optimiser with 100 epochs used for MMR method. (c) Loss value over epoch number plotted for MMR results shown in (a) and (b).

When implementing the SVR method, we use the same $\mathbf{x}$ and kernel functions. The corresponding SVR system of equations to solve for a problem of form $df/dx + g(x)f + r(x) = 0$ reads

$$\left[\begin{array}{c|c|c} \hat{M} & \mathbf{h}_1^0 - \hat{D}\mathbf{h}_0^0 & \mathbf{g} \\ \hline (\mathbf{h}_1^0 - \hat{D}\mathbf{h}_0^0)^T & \tilde{h}_0^0 & 1 \\ \hline \mathbf{g}^T & 1 & 0 \end{array}\right] \left[\begin{array}{c} \boldsymbol{\alpha} \\ \hline \beta \\ \hline b \end{array}\right] = \left[\begin{array}{c} \tilde{\mathbf{r}} \\ \hline f_0 \\ \hline 0 \end{array}\right], \tag{5.19}$$

where the notation is as follows:

$$[\Omega_n^m]_{i,j} = \nabla_n^m \kappa(x_j, x_i), \tag{5.20}$$

$$\tilde{\Omega}_n^m = \Omega_n^m + I/\gamma, \tag{5.21}$$

$$[\mathbf{h}_n^m]_i = \nabla_n^m \kappa(x_0, x_i), \tag{5.22}$$

$$\tilde{h}_n^m = \kappa(x_0, x_0)_n^m, \tag{5.23}$$

$$\hat{D} = \mathrm{diag}\left(\{g(x_i)\}_i\right), \tag{5.24}$$

$$[\tilde{\mathbf{g}}]_i = g(x_i), \tag{5.25}$$

$$[\tilde{\mathbf{r}}]_i = r(x_i), \tag{5.26}$$

$$\hat{M} = \Omega_1^1 - \Omega_1^0\hat{D} - \hat{D}\Omega_0^1 + \hat{D}\tilde{\Omega}_0^0\hat{D}. \tag{5.27}$$

We choose $\gamma = 10^5$ and this system is then solved with `Julia`'s built-in matrix-defined linear equation solver. The error of these results is shown in Fig. 5.5(b), with the result from using the quantum kernel shown explicitly in Fig. 5.5(a). Again it can be seen that both kernel types are able to closely approximate the considered function, with some variation likely from optimisation versus solving linear system of equations, with quantum kernel outperforming the classical kernel.

5.4.3 Solving Nonlinear Differential Equations

We now move on to consider solving *nonlinear* differential equations. As an example we choose the Duffing equation in the absence of damping term given by [171]

$$\frac{d^2 f(x)}{dx^2} = c\cos(ex) - af - bf^3, \tag{5.28}$$

where $f(x)$ is a solution in one dimension, a, b, c and e are constants. The resulting SVR formulation from following the procedure described earlier for a DE of the form $\frac{d^2 f(x)}{dx^2} = g(x, f)$ with $f(x_0) = f_0$ and $f'(x_0) = f_0'$ is

$$\left[\begin{array}{c|c|c|c|c|c} \tilde{\Omega}_2^2 & \Omega_0^2 & \mathbf{h}_0^2 & \mathbf{h}_1^2 & \mathbf{0} & \hat{0} \\ \hline \Omega_2^0 & \tilde{\Omega}_0^0 & \mathbf{h}_0^0 & \mathbf{h}_1^0 & \mathbf{1} & -\hat{I} \\ \hline \mathbf{h}^T{}_2^0 & \mathbf{h}^T{}_0^0 & \tilde{h}_0^0 & \tilde{h}_1^0 & 1 & \mathbf{0}^T \\ \hline \mathbf{h}^T{}_2^1 & \mathbf{h}^T{}_0^1 & \tilde{h}_0^1 & \tilde{h}_1^1 & 0 & \mathbf{0}^T \\ \hline \mathbf{0}^T & \mathbf{1}^T & 1 & 0 & 0 & \mathbf{0}^T \\ \hline \hat{D} & \hat{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \hat{0} \end{array}\right] \left[\begin{array}{c} \boldsymbol{\alpha} \\ \hline \boldsymbol{\eta} \\ \hline \beta_1 \\ \hline \beta_2 \\ \hline b \\ \hline \mathbf{y} \end{array}\right] = \left[\begin{array}{c} \tilde{\mathbf{g}} \\ \hline \mathbf{0} \\ \hline f_0 \\ \hline f_0' \\ \hline 0 \\ \hline \hat{0} \end{array}\right]. \tag{5.29}$$

With notation as follows

$$[\Omega_n^m]_{i,j} = \nabla_n^m \kappa(x_j, x_i), \tag{5.30}$$

$$\tilde{\Omega}_n^m = \Omega_n^m + \hat{I}/\gamma, \tag{5.31}$$

$$[\mathbf{h}_n^m]_i = \nabla_n^m \kappa(x_0, x_i), \tag{5.32}$$

$$\tilde{h}_n^m = \nabla_n^m \kappa(x_0, x_0), \tag{5.33}$$

$$\hat{D} = \mathrm{diag}\left(\left\{\frac{\partial g}{\partial f}(x_i, y_i)\right\}_i\right), \tag{5.34}$$

$$[\tilde{\mathbf{g}}]_i = g(x_i, y_i). \tag{5.35}$$

Therefore, this is the formulation we use for this problem with $g(x, f) = c\cos(ex) - af - bf^3$.

We note that as the problem in Eq. (5.28) is non-linear, it is not guaranteed that the MMR method stated as an optimisation problem here is going to lead to a convex problem. This may affect the ease of convergence for the MMR method. Furthermore the SVR method does not result in a deterministic linear problem; instead it results in a nonlinear problem, which can be solved with optimisation methods. First, we consider $a = 0.5$, $c = 3$, $e = 3\pi$ and have b, which controls the nonlinear contribution, increasing from 0 to 0.15. This allows testing the convergence for optimisation as the non-linearity of the problem increases. We increase b in increments of 0.001. For each value we use an SVR method repeated a hundred times to solve the problem with a different random set of initial variables.

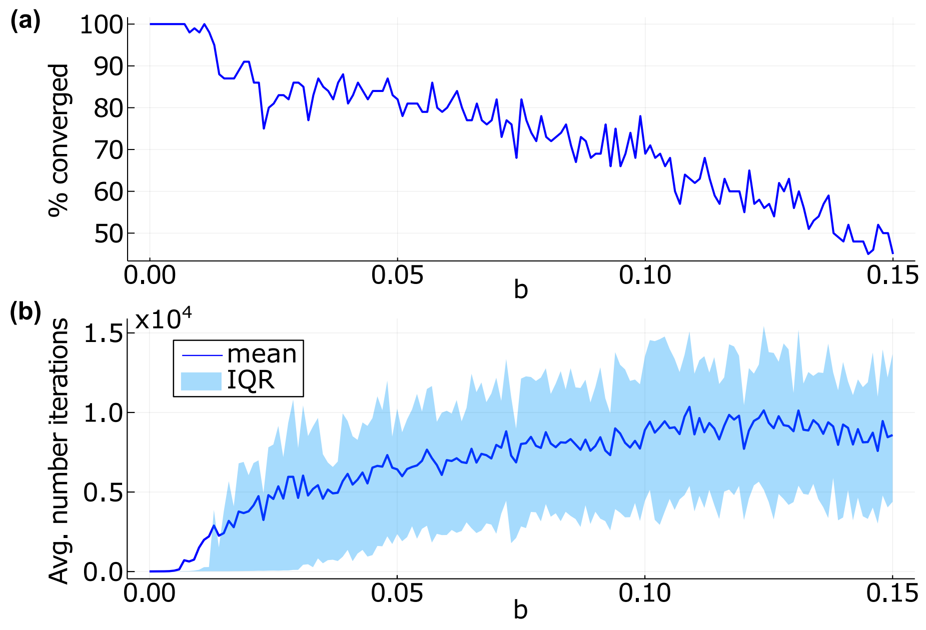


Figure 5.6: **Convergence behaviour of learning solution to the undamped Duffing equation problem, $\frac{d^2f}{dx^2} = 3\cos(3\pi x) - 0.5f - bf^3$, with increasing non-linearity controlled by b.** In (a) we plot of percentage of runs which result in convergence (loss less than 10^{-10}) as a function of b. In (b) we show an average number of iterations of the runs that converged in (a) vs value of b. Mean is shown with solid dark blue line, the interquartile range is shown with light blue shading.

We detail the hyper-parameters and set-up of the simulation as follows. A quantum kernel over five qubits is used of form of two layers of HEA depth five followed by feature map based on $R_x(\phi(x))$ on each qubit with $\phi(x) = qx/2$, where q is the qubit index. Training occurs over thirteen uniformly separated points between zero and one with initial value condition of $f = 1$ and $df/dx = 1$ at $x = 0$. The Newton method is chosen for optimisation and γ is set as 10^5. The simulation is implemented the same as for the linear case.

These results are shown in Fig. 5.6, where we label a trial as converged if the error for the kernel-based solution is close to machine precision (loss less than 10^{-10}). For each value b we plot the percentage of convergence as $100\mathcal{N}_{\text{conv}}/\mathcal{N}_{\text{tot}}$ with $\mathcal{N}_{\text{conv}}$ the number of converged trials and $\mathcal{N}_{\text{tot}}$ the total number of trials. We observe that as the degree of non-linearity increases, the percentage of initialisation states which result in convergence decreases, changing from 100% at $b = 0$

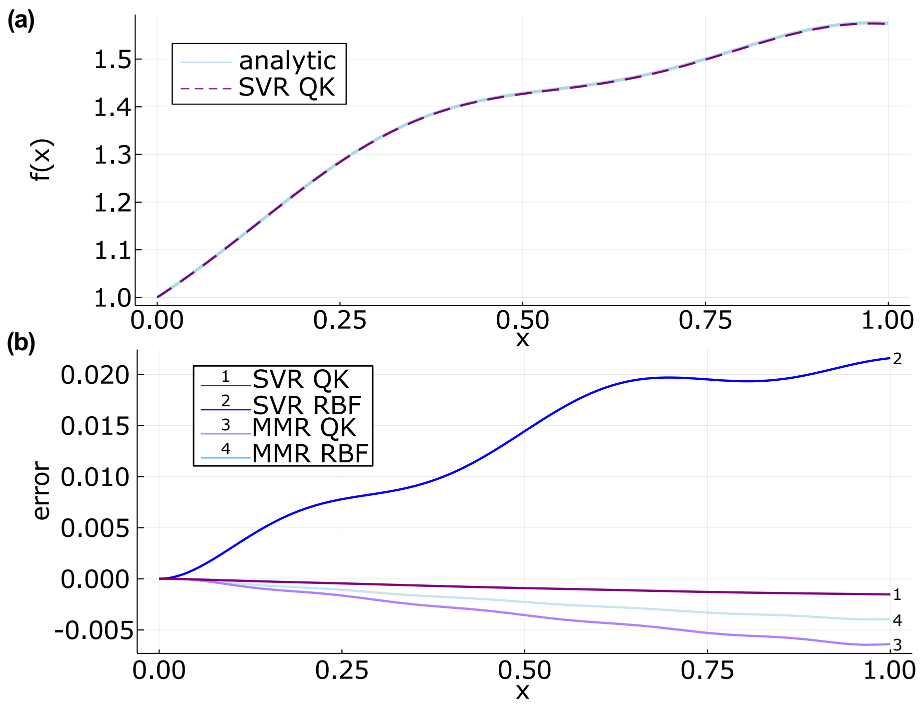


Figure 5.7: **Kernel-based solution and error for solving the instance of nonlinear Duffing equation [Eq.** (5.28)**].** We set $a = 0.5, b = 0.15, c = 3, e = 3\pi$. (a) Solution via SVR method for quantum kernel with two layers, $\phi(x) = qx/2$ and $N = 5$ shown by dashed purple line. Known analytic solution plotted with solid light blue line. (b) Error between results and analytic solution plotted over x as $(f(x) - f_{\mathrm{sol}}(x))/\mathrm{range}(f_{\mathrm{sol}})$. Error plotted for result shown in (a) and SVR method with classical RBF kernel with $\sigma = 0.8$. Also plotted results from MMR method with same form of kernels considered with $\sigma = 0.8$, $\phi(x) = 2qx/5$ and $N = 8$. Newton optimiser used for MMR and SVR methods.

to around 45% at $b = 0.15$. Furthermore the number of epochs required for convergence also increases as b increases from a mean of 5 to around 9000. We stress that the cost here is on classical optimisation, and that additional training does not require extra quantum resources.

Fig. 5.6 demonstrates one particular situation, and the convergence may be specific to the choice of setup. However, we find that the observed behaviour of gradually decreasing convergence and increasing number of epochs at increasing nonlinearity is the general trend. Unless specific optimisation routines are developed, kernel methods shall be more suited for problems with limited nonlinearity. We also stress that the procedure does lead to a close to perfect solution (we see abrupt decrease of loss), justifying the cost for re-running the optimisation. In the future we plan to investigate optimisers and kernel forms that may exploit this feature.

We also show the results from applying SVR and MMR with both quantum kernel and classical RBF kernel with $b = 0.15$ in Fig. 5.7. The quantum kernel used is as used in Fig. 5.6, though with an alteration to $\phi(x) = 2qx/5$ and eight qubits for the MMR case. The classical kernel used is RBF with $\sigma = 0.2$ for MMR and $\sigma = 0.8$ for SVR. For all methods training occurs over thirteen uniformly separated points between zero and one with initial value condition of $y = 1$ and $dy/dx = 1$ at $x = 0$. Newton's method is used for optimisation. The solution for the SVR method is shown in Fig. 5.7(a), closely matching the exact $f(x)$. We can also see that for the nonlinear Duffing equation example for all methods the error is minimal at $x = 0$ [see Fig. 5.7(b)] where the initial value is set. The magnitude of error increases with x. We see that the combination of a quantum kernel with classical SVR processing shows the best performance. However, we note that this required the most iterations for convergence, and one to one comparison of budgets is not straightforward. To have optimal behaviour, we consider that further work for optimal choice of kernel functions and hyper-parameters is required.

5.4.4 Kernel Comparison

We show how different quantum kernel functions perform in comparison to one another when solving the same problem with the same hyper-parameters. We solve the linear differential equation presented in Eq. (5.18). Below we list the kernel functions we compare by label. The kernel functions are defined over $N = 8$ qubits with varying generators that include:

- 1L-prod: A single layered product feature map with $|\psi(x)\rangle = \hat{\mathcal{U}}(x)\hat{\mathcal{V}}|0\rangle$ where $\hat{\mathcal{V}}$ is a HEA of depth five with randomised parameters which are set throughout training and $\hat{\mathcal{U}}(x) = \Pi_{j=1}^{N}\hat{R}_X^j(x)$.
- 1L-tower: A single layered tower feature map with $|\psi(x)\rangle = \hat{\mathcal{U}}(x)\hat{\mathcal{V}}|0\rangle$ where $\hat{\mathcal{V}}$ is a HEA of depth five with randomised parameters which are set throughout training and $\hat{\mathcal{U}}(x) = \Pi_{j=1}^{N}\hat{R}_X^j(jx)$.

- 1L-cheb: A single layered Chebyshev product feature map with $|\psi(x)\rangle = \hat{\mathcal{U}}(x)\hat{\mathcal{V}}|0\rangle$ where $\hat{\mathcal{V}}$ is a HEA of depth five with randomised parameters which are set throughout training and $\hat{\mathcal{U}}(x) = \Pi_{j=1}^{N}\hat{R}_X^j(j\text{arccos}(x))$.
- 2L-prod: A two-layer product feature map with $|\psi(x)\rangle = \hat{\mathcal{U}}(x)\hat{\mathcal{V}}_2\hat{\mathcal{U}}(x)\hat{\mathcal{V}}_1|0\rangle$ where $\hat{\mathcal{V}}_{1,2}$ are HEAs of depth five with randomised parameters which are set throughout training and $\hat{\mathcal{U}}(x) = \Pi_{j=1}^{N}\hat{R}_X^j(x)$.
- 2L-tower: A two-layer tower feature map with $|\psi(x)\rangle = \hat{\mathcal{U}}(x)\hat{\mathcal{V}}_2\hat{\mathcal{U}}(x)\hat{\mathcal{V}}_1|0\rangle$ where $\hat{\mathcal{V}}_{1,2}$ are HEAs of depth five with randomised parameters which are set throughout training and $\hat{\mathcal{U}}(x) = \Pi_{j=1}^{N}\hat{R}_X^j(jx)$.
- 2L-cheb: A two-layer Chebyshev product feature map with $|\psi(x)\rangle = \hat{\mathcal{U}}(x)\hat{\mathcal{V}}_2\hat{\mathcal{U}}(x)\hat{\mathcal{V}}_1|0\rangle$ where $\hat{\mathcal{V}}_{1/2}$ are HEAs of depth five with randomised parameters which are set throughout training and $\hat{\mathcal{U}}(x) = \Pi_{j=1}^{N}\hat{R}_X{}^j(j\ \text{arccos}(x))$.

For describing the feature maps, we use terminology as introduced in Ref. [112], with a product feature map referring to one where the same gate is applied to each qubit for the generator, and a tower where rotational gates have a dependence on qubit number.

First, we solve this differential equation with the MMR method with 21 training and evaluation points uniformly distributed between 0 and 0.99. For the MSE loss function, used with a boundary loss term, the kernel functions used are described as above. The Newton method is used for optimisation. The results are shown in Fig. 5.8. In Fig. 5.8(a) we can see that of the 2L kernel functions the Chebyshev kernel function performed the best over the majority of the region however oscillated near the boundary with one. This is a known behaviour of Chebyshev functions and can be ameliorated by transforming the training region to avoid the boundary or with use of Chebyshev training nodes. 2L-tower performs well throughout whilst 2L-prod captures the general shape but does not quite match the peaks and troughs. This potentially corresponds to the reduced expressivity, in comparison to other feature maps. In Fig. 5.8(b) we see the normalised absolute error of the results. We can see that the 1L kernel functions contain the best

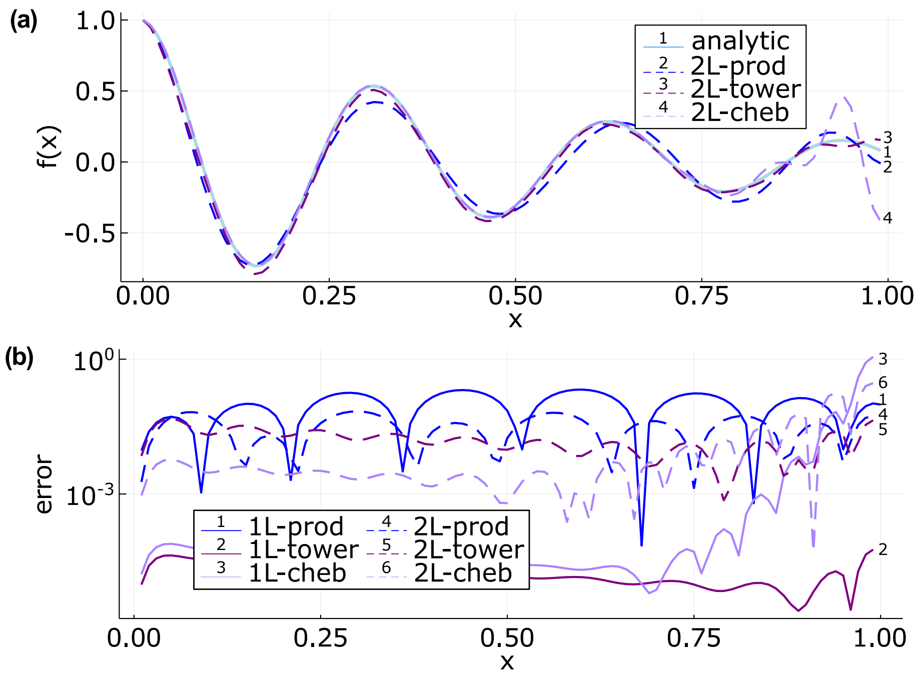


Figure 5.8: **Comparison of the results of solving** (5.18) **with different kernel functions with MMR.** Kernel functions used are described in above list. MMR method is used with $N = 8$, 21 training and evaluation points uniformly distributed between 0 and 0.99 and the Newton method is used for optimisation. (a) Solutions from two layered kernels plotted with dashed lines. The known analytic solution is shown with a solid line. (b) The absolute normalised error of the solutions from different kernel functions plotted against x. The error is calculated as $\text{abs}[(f(x) - f_{\text{sol}}(x))/\text{range}(f_{\text{sol}})]$.

and worst performing of the set with 1L-tower and 1L-prod respectively. At the same time, the 2L kernel functions show a comparatively similar behaviour.

We then solve the same problem with the same set of kernel functions with the SVR method. The training points are uniformly distributed between 0 and 0.99, and we use $\gamma = 10^9$. The results are shown in Fig. 5.9. In Fig. 5.9(a) we can see that both 2L-tower and 2L-cheb performed well and again 2L-prod captured the overall shape but does not exactly match the known solution. Similarly to the MMR case, Fig. 5.9(b) shows that the 2L kernels again contain the best and worst performing kernel functions and that Chebyshev kernel again exhibits strong oscillations near the boundary with one.

We highlight that these results are one example of the methods being applied to one problem with a particular set up, and therefore guaranteed conclusions

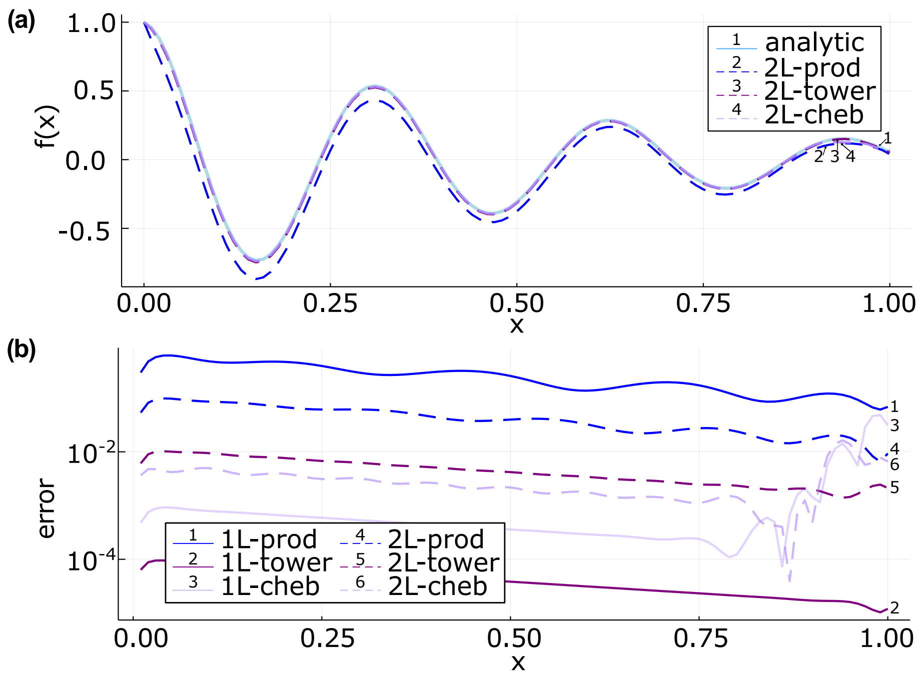


Figure 5.9: **Comparison of the results of solving** (5.18) **with different kernel functions with SVR.** Kernel functions used are described in above list. SVR method is used with $N = 8$, 21 training points uniformly distributed between 0 and 0.99 and $\gamma = 10^9$. (a) Solutions from two layered kernels plotted with dashed lines. The known analytic solution is shown with a solid line. (b) The absolute normalised error of the solutions from different kernel functions plotted against x. The error is calculated as $\text{abs}((f(x) - f_{\text{sol}}(x))/\text{range}(f_{\text{sol}}))$.

can not be drawn from them. However, in general we observe that the choice of kernel functions is important to receive good results and that methods to find kernel functions suitable for a chosen problem will be important for the future of kernel methods. We also observe that many kernel functions are likely to have similar performance — sufficiently good results can be obtained without having to find the "perfect" kernel function. Choosing a suitable kernel function is an important step for classical applications of kernel methods too, some inspiration for how to approach quantum kernel function choice may be able to be found from classical approaches [172].

5.5 Discussion

In this work, we proposed quantum protocols for solving differential equation with kernel methods. We represent potential solutions as quantum models that are based on weighted sums of kernel functions, corresponding to overlaps of quantum states. The adjustable weights are optimised such that for many problems the optimisation is convex, leading to fast convergence to the potential solution. Specifically, we propose two approaches, being mixed model regression (MMR) and support vector regression (SVR), where optimisation workflow is different. An important element of our approach is the automatic differentiation of quantum kernels with respect to encoded feature variables using quantum circuit differentiation. We applied both MMR and SVR for several toy problems. First, we presented regression for a quantum dataset, corresponding to nonequilibrium dynamics of quantum spin liquids. In this case, the use of quantum kernels may offer advantage, as native quantum operations are used. Second, we solve linear DEs, showing that nontrivial solutions can be routinely found with few epochs. Finally, applying our approaches to some nonlinear problems, the optimisation becomes non-convex, thus requiring largely increased number of epochs. At the same time, we note that by kernelising quantum models we modify the landscape of optimisation. This raises the question of convergence difference between parameterised quantum circuits [58, 173] and kernel models.

As the algorithms developed in this and the previous chapter are for the same purpose, solving DEs, the question arises of how do they compare? Both algorithms prepare a trial function and train it to solve a given DE. The MMR kernel approach and the DQC approach to training are very similar, with loss functions to optimise. The SVR approach has some differences - it is only applicable for kernel functions and it results in a system of equations which can be represented via a loss function to minimise or can be solved with other methods for such systems. The trial functions themselves are a significant difference - DQC relies on expectation measurement and kernel methods use overlaps. An expectation method is easier for a quantum device to compute. However, DQC requires remeasurement throughout training whilst the kernel functions, as considered, only require evaluation pre and post training. Therefore, both algorithms are hoped to be near-term amenable, with DQC requiring many ”easier” measurements and kernel methods requiring fewer ”harder” measurements. We do note alternative kernel functions could be considered which do require evaluation during training. A more direct comparison can be seen in the use of these algorithms for solving the linear DE of a damped oscillator in Fig. 3.4 and Fig. 5.5. As this is just one example with one set of hyperparameters nothing too conclusive can be drawn from this comparison. But as an illustrative example it shows that both methods are able to provide good fit to such a problem with a number of iterations of magnitude 100.

While this work presents a first step towards quantum kernel-based differential equation solving, many aspects are left unexplored. We have observed that as expected the choice of kernel function is important to receive accurate results but is not unique. Thus quantum feature map design is one such aspect, covering how to choose kernel functions appropriately and could potentially be problem-motivated for each specific case. Finding conditions for which non-linear equations are guaranteed to result in a convex loss landscape is another open question. Also left unconfirmed is any available quantum advantage. Whilst there is intuition that quantum benefit could come from the expressivity and range of kernel functions expressible by quantum devices it is not guaranteed. Therefore, a topic for future work would be to either confirm or refute this current intuition,

We note the errors may limit the performance of near-term devices, often requiring an increase of the number of evaluations of kernel functions and therefore the circuit measurement budget. In particular we consider the recent work on the topic of exponential concentration of quantum kernel methods and its effect on trainability [174]. This work highlights that quantum kernels do have to be chosen and used carefully. If care is not taken the chosen quantum kernel can concentrate around a specific value leading to a model which cannot be efficiently evaluated due to an exponential number of measurements being required. The authors in Ref. [174] propose guidelines on how to avoid such issues, and further work in the field is likely to build upon this in the future due to the effect this has on all kernel function based methods.

Chapter 6

Quantum Chebyshev Transform

6.1 Declaration of Contribution

The work in this chapter is released as a manuscript [175] and is currently under consideration for publication. My contributions for this work include joint development of the feature map and transformation (specifically including simplification process of the transform circuit and derivative consideration of the feature map), initial development of simulation code and shared contributions for writing.

6.2 Introduction

A vital component to many quantum algorithms is the quantum Fourier transform as described in 2.7.3. This circuit transforms between the Fourier space and computational space. In different spaces certain tasks are easier or harder therefore transforming to an appropriate space is an important part of an algorithm. Fourier space is not unique for this.

We develop a paradigm for building quantum models in the orthonormal space of scaled Chebyshev polynomials. We show how to encode data into quantum states with amplitudes being Chebyshev polynomials with degree growing exponentially in the system size. Similar to the quantum Fourier transform, we describe the quantum circuit for the mapping between computational and Cheby-

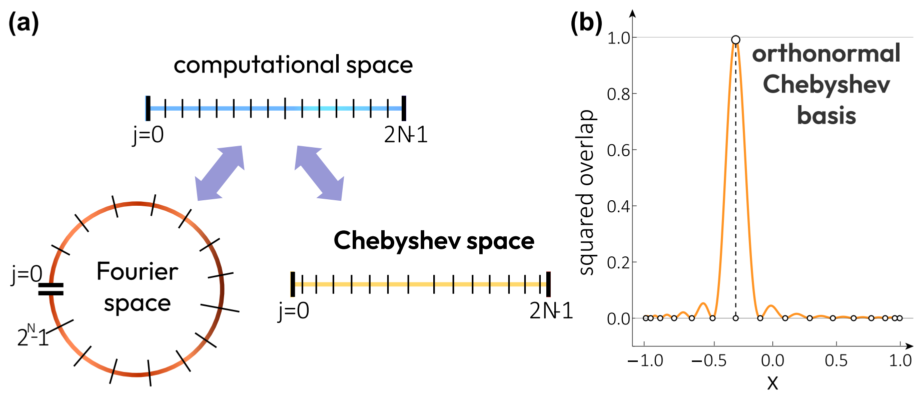


Figure 6.1: **Visualisation Chebyshev Space.** **(a)** Visualisation for mapping bitstrings from the computational basis (labelled by integers $j \in [0, 2^N - 1]$) into the Chebyshev space with non-equidistant nodes given by $x_j^{\mathrm{Ch}} = \cos\left(\pi(2j+1)/2^{N+1}\right)$, or the Fourier space with an equidistant grid of phases (sectors on a circular open-ended loop with $2\pi j/2^N$ arcs). **(b)** Squared overlap between two Chebyshev states $|\langle\tau(x')|\tau(x)\rangle|^2$, where we choose $x' = x_7^{\mathrm{Ch}}$, showing that the basis is orthonormal at the nodes. Notably, the Chebyshev overlaps are real, unlike the complex overlaps between Fourier states, but lead to similar mid-point behaviour.

shev spaces. We propose an embedding circuit for generating the orthonormal Chebyshev basis of exponential capacity, represented by a continuously-parameterised circuit with a single ancilla and one projective measurement. This enables automatic quantum model differentiation, and opens a route to solving stochastic differential equations. We apply the developed paradigm to generative modeling from physically- and financially-motivated distributions, and use the quantum Chebyshev transform for efficient sampling of these distributions in extended computational basis.

6.3 Method

Our goal is designing models based on feature maps that can enable representation in a chosen space (in our case Chebyshev) and the computational space. The computational basis corresponds to the set of orthonormal states $\{|x_j\rangle\}_{j=0}^{2^N-1}$ such that overlaps equate to the Kronecker delta function, $\langle x_j|x_{j'}\rangle = \delta_{j,j'}$. One can easily generate states $|x_j\rangle$ by applying Pauli $\hat{X}$ operators to the computational zero, sometimes referred to as the basis encoding [43]. However, building models in this space is difficult, as it assumes adjusting amplitudes (probabilities) for the entire domain. By transforming to another space this difficulty could be avoided.

6.3.1 Chebyshev Encoding

Specifically, we want to generate an (unnormalised) state $|\tau(x)\rangle$ with amplitudes given by Chebyshev polynomials $T_k(x) \equiv \cos(k \arccos(x))$ of the first kind and degree k,

$$|\tau(x)\rangle = \frac{1}{2^{N/2}} T_0(x)|0\rangle + \frac{1}{2^{(N-1)/2}} \sum_{k=1}^{2^N-1} T_k(x)|k\rangle, \tag{6.1}$$

where we note the distinct amplitude for the computational zero state weighted by $T_0(x) \equiv 1$. Given the properties of polynomials $T_k(x)$ and their orthogonality conditions,

$$\sum_{j=0}^{2^N-1} T_k(x_j^{\mathrm{Ch}}) T_\ell(x_j^{\mathrm{Ch}}) = \begin{cases} 0, & k \neq \ell, \\ 2^N & k = \ell = 0, \\ 2^{N-1} & k = \ell \neq 0, \end{cases} \tag{6.2}$$

the states in Eq. (6.1) are orthonormal at the Chebyshev nodes $x_j^{\mathrm{Ch}} := \cos\left(\pi(2j+1)/2^{N+1}\right)$, defined at zeros of Chebyshev polynomials. Namely, the set of Chebyshev states $\{|\tau(x_j^{\mathrm{Ch}})\rangle\}_{j=0}^{2^N-1}$ are such that $\langle \tau(x_j^{\mathrm{Ch}})|\tau(x_{j'}^{\mathrm{Ch}})\rangle = \delta_{j,j'}$. We note that the Chebyshev nodes $x_j^{\mathrm{Ch}} \in (-1, 1)$ form a non-equidistant grid, unlike the standard computational basis and Fourier basis associated to equidistant "ruler"-type and "clock"-type grids (see Fig. 6.1(a) for the illustration). We also highlight that outside of the Chebyshev nodes the states $|\tau(x)\rangle$ are not orthogonal. Their squared overlap is plotted in Fig. 6.1(b), and can be derived analytically for one of the variables x' set to one of the Chebyshev nodes, such that

$$|\langle \tau(x')|\tau(x)\rangle|^2 = \left(\frac{T_{2^N}(x')T_{2^N-1}(x) - T_{2^N-1}(x')T_{2^N}(x)}{2^N(x'-x)} - \frac{1}{2^N} \right)^2, \tag{6.3}$$

which can be derived from the Christofel-Darboux formula for Chebyshev polynomials [176]. Note for $x = x'$ the limit has to be calculated.

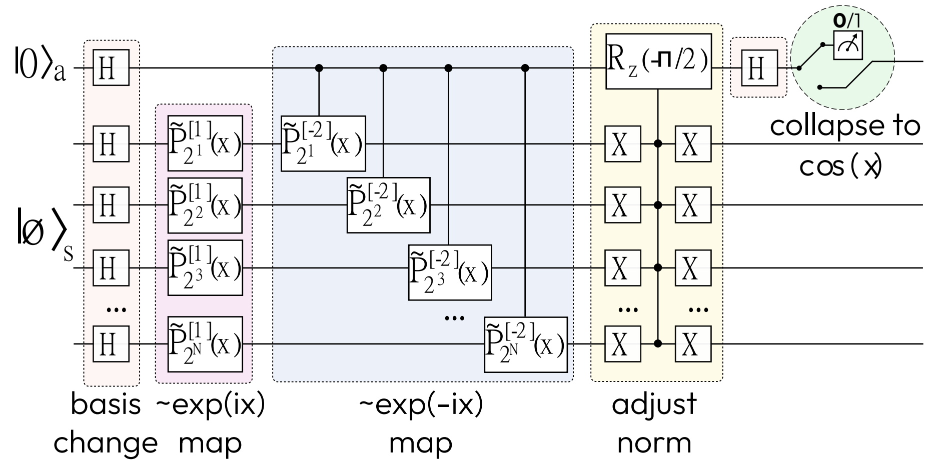


Figure 6.2: **Quantum Chebyshev feature map.** Quantum Chebyshev feature map that creates a Chebyshev state via x-parametrised circuit with single projective meausurement — sequence of phase feature maps that embed complex exponents, controlled rotation to adjust the norm of the zero frequency term, and post-selecting the sum term with ancilla measured in 0. Scaled single-qubit phase shift gates in the feature map circuit are defined as $\tilde{\mathrm{P}}_l^{[s]}(x) = \mathrm{diag}\{1, \exp(is2^N \arccos(x)/l)\}$, where l grows exponentially as 2^j, with j being the qubit number, and s takes values of 1 and -2. Here, H and X are Hadamard and Pauli $\hat{X}$ gates.

Therefore, though at the Chebyshev nodes an orthonormal basis is prepared, preparing Chebyshev states for arbitrary $x \in (-1, 1)$ is a non-unitary process. We can use a unitary quantum circuit with an ancillary qubit, leading to normalised states by definition. We denote such normalised states as $|\tilde{\tau}(x)\rangle := |\tau(x)\rangle / \sqrt{\langle\tau(x)|\tau(x)\rangle}$, which coincide with $|\tau(x)\rangle$ for the Chebyshev nodes, and converge in the large N limit.

To construct the orthonormal Chebyshev feature map $\hat{\mathcal{U}}_\tau(x)$ as a circuit which prepares $|\tilde{\tau}(x)\rangle$ state, we observe that Chebyshev polynomials are represented by cosines evaluated at the specific grid. Thus, they can be prepared using a combination of exponents for some scaled variable x, $\cos(x) = \{\exp(ix) + \exp(-ix)\}/2$, where each amplitude is embedded via the phase feature map [137]. The two can be combined using the linear combination of unitaries (LCU) approach [177], where maps are conditioned on the state of the (top) ancilla qubit, and effectively interferred, choosing the ancilla collapsed to 0 outcome. LCU is further detailed in section 2.7.2. Since $\exp(-ix) = \exp(ix)\exp(-i2x)$, we can condition only one of the phase feature maps [see corresponding blocks in Fig. 6.2 labeled as $\exp(\pm ix)$ map]. With this, equal-weight combinations of exponents are prepared as amplitudes for variables scaled such that $T_k(x)$ are recovered. Finally, we need to adjust the weight of the constant term $T_0(x)$. For this we use a round of Grover iterate circuit [6], rotating the state around $|0\rangle$ for the fixed angle. As the measurement operation on the ancilla commutes with rotation, we push it to the end and conclude the circuit.

We highlight that the Chebyshev-based product feature maps introduced in previous studies [112, 123] and chapters 3 and 4 of the thesis are fundamentally different. These feature maps prepare states where the amplitudes are related to the Chebyshev polynomials and provides them as a fitting basis. However those feature maps do not prepare a state where each amplitude is a Chebyshev polynomial. In particular they do not prepare an orthogonal basis when evaluated over a grid. This prevents their conversion to amplitude encoding (standard in quantum information processing), and building generative models in particular.

6.3.2 Chebyshev Transform

Next, we need to develop a map between Chebyshev states and the computational basis states (and reverse). Note that once we have prepared the states forming an orthonormal basis, there exists a bijection and corresponding transformation between this basis and any different orthonormal basis. Here, we introduce the Chebyshev transform as $\hat{\mathcal{U}}_{\mathrm{QChT}} = \sum_{j=0}^{2^N-1} |\tau(x_j^{\mathrm{Ch}})\rangle\langle x_j|$. We show the corresponding circuit in Fig. 6.3, and explain the reasoning below.

First, we note that the Chebyshev transform can be understood as a specific version of the cosine transform [178]. Namely, the vector of amplitudes for state $|\tau(x_j^{\mathrm{Ch}})\rangle$ corresponds to the $(j+1)^{\mathrm{th}}$ column of the type-II discrete cosine transform

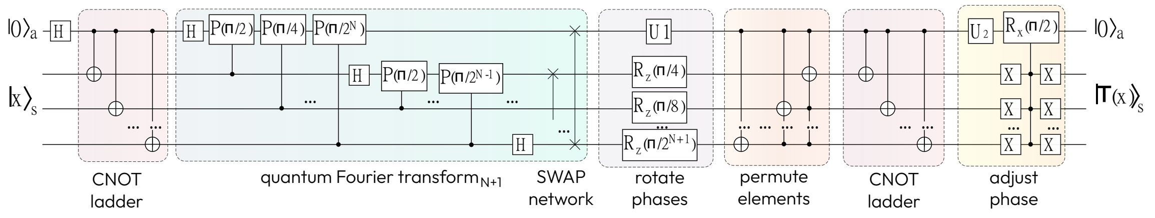


Figure 6.3: **Quantum Chebyshev Transform.** Quantum Chebyshev transform ($\hat{\mathcal{U}}_{\mathrm{QChT}}$) circuit which maps computational basis states $\{|x_j\rangle\}_{j=0}^{2^N-1}$ into Chebyshev states $\{|\tau(x_j^{\mathrm{Ch}})\rangle\}_{j=0}^{2^N-1}$. The transform involves a QFT circuit applied to $N+1$ qubits (N-qubit system plus one ancillary qubit), followed by phase-adjusting and permutation circuits. Here, we use a standard phase shift gate defined as $\mathrm{P}(\phi) = \mathrm{diag}\{1, \exp(i\phi)\}$. We note that local phase rotations $\mathrm{U}_1 = \mathrm{P}(-\pi/2^{N+1})\mathrm{R_Z}(-\pi(2^N - 1)/2^{N+1})$ and $\mathrm{U}_2 = \mathrm{P}(-\pi/2)\mathrm{R_Y}(-\pi/2)$ acting on the ancilla can be combined into a single gate.

matrix, $\mathrm{DCT}_N^{\mathrm{II}}$, defined as

$$\mathrm{DCT}_N^{\mathrm{II}} := 2^{-\frac{N-1}{2}} \Big\{ c_k \cos[k(j+1/2)\pi/2^N] \Big\}_{k,j=0\ldots 2^N-1}, \tag{6.4}$$

where $c_0 = 1/\sqrt{2}$, and $c_k = 1$ for $k \neq 0$. We note that this matrix is strongly related to Fourier transform matrix, but requires interfering and mixing its elements, thus suggesting the use of an extended QFT circuit. The circuit starts with a Hadamard gate on the ancilla, being the most significant bit, followed by a CNOT ladder that is typically used for a cat state preparation. This is followed by a $N+1$ QFT circuit, which is later converted into blocks of purely real and imaginary components through a series of unitary gates. Namely, the single qubit gates U_1 and $\mathrm{R_Z}$ are introduced to adjust the relative phases for states split as $|0\rangle_a|\Phi\rangle$ and $|1\rangle_a|\Phi\rangle$, for any N-qubit intermediate state denoted by $|\Phi\rangle$. The permutation circuit is used to reorder amplitudes of the conditioned states, followed by another CNOT ladder. The circuit is concluded with the constant phase $\hat{\mathrm{U}}_2$ and multi-controlled $\hat{\mathrm{R}}_{\mathrm{X}}$ gates to fix weights and to ensure that the amplitudes of $|0\rangle_a|\Phi\rangle$ ($|1\rangle_a|\Phi\rangle$) are purely real (imaginary) for any input states. We note that the ancilla starts and ends in $|0\rangle$ state ("clean" run). Finally, concatenating the described embedding and the map, we get $\hat{\mathcal{U}}_f(x) = \hat{\mathcal{U}}_{\mathrm{QChT}}^{\dagger}\hat{\mathcal{U}}_{\tau}(x)$.

6.3.3 Chebyshev Map Derivative

Next, we briefly discuss how to differentiate Chebyshev models. This can be approached in several ways. First, we can use the parameter shift rule [39, 28, 40], which is valid in the training stage when cost function corresponds to projection on zero state (including ancilla qubit). In this case we need to decompose controlled phase rotations using CNOT conjugation, and apply two shifts per individually reuploaded parameter x. In total, the orthonormal Chebyshev map can be differentiated with $4N$ shifts. Alternatively, one can see this as differentiation in presence of mid-circuit measurements [179]. Another option is to differentiate the feature map formally, extracting an effective generator $\hat{\mathcal{G}}_{\text{eff}}$, and taking derivatives as a linear combination of unitaries [39, 28, 158]. This method of derivatives will be used in the following chapter.

6.4 Results

6.4.1 Generative Modelling

Next, we demonstrate examples of applying quantum Chebyshev transform to generative modelling from relevant distributions. We solve the stochastic differential equation $dS_t = \mu S_t dt + \sigma S_t dW_t$ with constant drift μ, constant volatility σ and stochastic Wiener process W_t. This is an instance of the Black-Scholes equations [180] which have uses in finances for modelling asset and option processes. The known solution of this SDE is $\ln(S_t) = \ln S_0 + (\mu - \sigma^2/2)t + \sigma W_t$, a lognormal probability density function with form

$$\mathbb{P}(S_t) = \frac{1}{S_t \sigma \sqrt{2\pi t}} \exp\left\{ \frac{-[\ln(S_t/S_0) + (\mu - \sigma^2/2)t]^2}{2\sigma^2 t} \right\}. \tag{6.5}$$

We choose $p_{\text{tar}} = \mathbb{P}(S_{t_0})$ as the target function, we use the Chebyshev model to learn this distribution. To do this, we utilise the differentiable quantum generative models (DQGM) framework which I now briefly introduce with full details in [137].

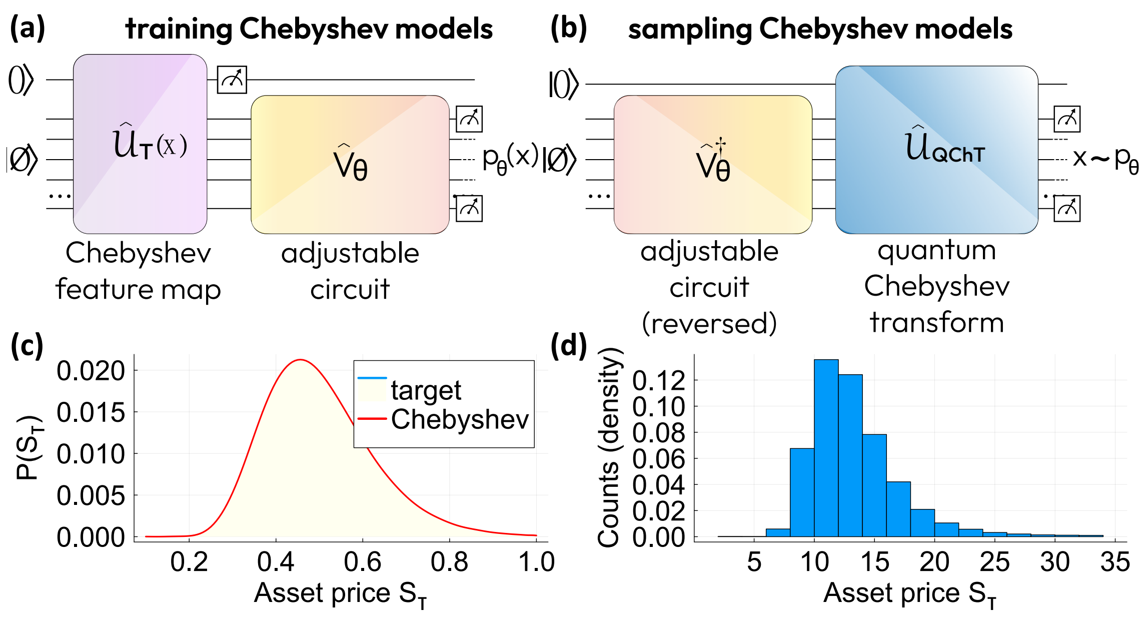


Figure 6.4: **Learning and sampling distributions with quantum Chebyshev models.** **(a)** Circuit used to train the model in latent space. **(b)** Circuit used to sample the model in the computational basis. **(c)** Training a lognormal distribution with parameters $\mu = 0$, $\sigma = 0.25$, $S_0 = 0.5$, and $t_0 = 1$. **(d)** Sampling from the trained lognormal distribution, generated with 10^6 shots (binned).

Differentiable Quantum Generative Model Algorithm

The task of this algorithm is generative modelling - to prepare a way to sample from a desired distribution based on some knowledge of the distribution. This knowledge could be a set of data or a differential equation.

To achieve this we consider a trial function

$$p_\theta(x) = \langle 0|\hat{\mathcal{U}}_\tau^\dagger(x)\hat{V}_\theta^\dagger|0\rangle\langle 0|\hat{V}_\theta\hat{\mathcal{U}}_\tau(x)|0\rangle, \tag{6.6}$$

where $\hat{V}_\theta$ is a variational ansatz and $\hat{\mathcal{U}}_\tau(x)$ a feature map. For this method there are limitations on the choice of feature map. It must be such that there exists a transformation $\hat{\mathcal{U}}_T$ between the basis induced by the feature map and the computational basis. For example the Chebyshev basis introduced in this chapter and the Fourier basis as discussed.

This trial function represents a probability density function (pdf) with x being the possible values of the distribution and $p_\theta(x)$ the probability of measuring x. We do note that this is not normalised in terms of area under curve but instead at the discrete nodes of the basis (i.e. Chebyshev nodes and Fourier nodes). This becomes clearer once we introduce the use of the transform. We see that this trial function is similar to that of QCL and DQC and therefore is able to be trained to fit a set of data points or DE in that manner. The use of the continuous feature map allows for continuous fitting of the desired distribution, access to exact derivatives and avoidance of data input and output problems during training.

After training we now have a function $p_\theta(x)$ which represents the distribution we want to sample from, but with the current form of $p_\theta(x)$ this is not a simple task. Therefore we consider the transformation associated to the chosen basis and include it as $I = \hat{\mathcal{U}}^\dagger_{QChT}\hat{\mathcal{U}}_{QChT}$ for

$$p_\theta(x) = \langle 0|\hat{\mathcal{U}}^\dagger_\tau(x)\hat{\mathcal{U}}^\dagger_{QChT}\hat{\mathcal{U}}_{QChT}\hat{V}^\dagger_\theta|0\rangle\langle 0|\hat{V}_\theta\hat{\mathcal{U}}^\dagger_{QChT}\hat{\mathcal{U}}_{QChT}\hat{\mathcal{U}}_\tau(x)|0\rangle, \tag{6.7}$$

$$p_\theta(x) = \langle x|\hat{\mathcal{U}}_{QChT}\hat{V}^\dagger_\theta|0\rangle\langle 0|\hat{V}_\theta\hat{\mathcal{U}}^\dagger_{QChT}|x\rangle, \tag{6.8}$$

where $|x\rangle$ is in the computational basis i.e. the j^{th} Chebyshev node x^{Ch}_j is represented by the j^{th} computational basis state $|j\rangle$. By considering this form of the trial function we can see that the probability of measuring x^{Ch}_j is the same as sampling $|j\rangle$ from $\hat{\mathcal{U}}_{QChT}\hat{V}^\dagger_\theta|0\rangle$. Therefore, we can now sample from the distribution represented by $p_\theta(x)$ by preparing $\hat{\mathcal{U}}_{QChT}\hat{V}^\dagger_\theta|0\rangle$ and measuring it in the computational basis.

This method trains a representation of a probability distribution in one space and then samples it in another space. This makes use of the different properties of the spaces and utilises them for the tasks they are suited to.

Application

We now apply this method to the problem considered. We begin by variationally training the trial function in the latent space with the Chebyshev feature map $\hat{\mathcal{U}}_{\tau}(x)$ shown in Fig. 6.4(a), and then sample from the distribution in the computational basis using the Chebyshev transform circuit $\hat{\mathcal{U}}_{\text{QChT}}$ shown in Fig. 6.4(b). The result in Fig. 6.4(c) shows the trained lognormal distribution using $N = 5$ qubits with a variational hardware efficient ansatz $\hat{V}_{\theta}$ of depth 14 [123]. We employ a mean squared error loss with a low learning rate of 0.005, simulated over five thousand epochs using Julia's Yao package [110]. The training grid consists of positive Chebyshev nodes $\{x_j^{\text{Ch}}\}\ \forall\ j \in [0, 2^{N-1}]$ and additional points between these nodes $\{x_{j/2}^{\text{Ch}}\}$. The histogram in Fig. 6.4(d) shows the resulting sampled distribution. Like previous variational algorithms many of the choices of hyperparameters such as ansatz and feature map greatly affect the chance of success of training.

6.4.2 Comparison with Fourier Encoding

As a further example, we consider a linear distribution with probability density function $\mathbb{P}(x) = x$. In this case, we highlight the importance of the Chebyshev basis as compared to the Fourier basis of similar expressivity. Using only $N = 2$ qubits for the embedding, we learn the linear distribution with the orthonormal Chebyshev, and compare it to the Fourier model built with the phase map [137]. The results are shown in Fig. 6.5(a), where hyperparameters are the same as before. We note that while Chebyshev model follows $\mathbb{P}(x)$ closely, the Fourier model experiences oscillations around the linear trend. Moreover, once we evaluate the derivatives for the respective generative models, we observe that while Chebyshev's derivative comes close to one, the Fourier derivatives are largely off [Fig. 6.5(b)]. This is an important consideration for solving differential equations, as large deviation in derivative-based loss terms lead to poor convergence overall. The plot in Fig. 6.5(c) shows the sampled distribution of the linear quantum Chebyshev model, where we use the optimised $\hat{V}_{\theta^*}$ for 2 qubits, and map it to an extended register of $N = 8$ qubits with QChT. We remind that samples are shown

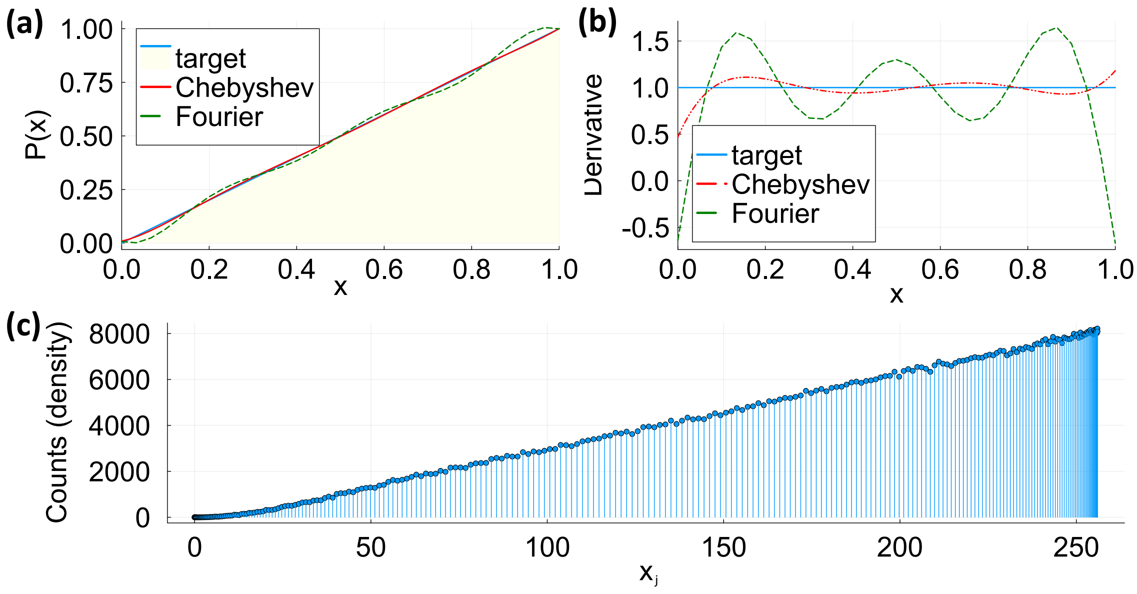


Figure 6.5: **Comparison between the quantum Chebyshev model and the quantum Fourier model for generative modelling using the DQGM framework.** **(a)** Training a linear distribution with ansatz depth of 6 layers. **(b)** Derivatives of the trained linear models. **(c)** Sampling from the trained linear distribution on an extended register of 8 qubits, shown for integer labels of Chebyshev nodes x_j.

for the Chebyshev grid.

6.5 Discussion

In this chapter we have highlighted the idea that changing to a suitable basis for a given problem can be of great use – demonstrated by how this is currently used with the Fourier transform. We look at what are the requirements on a space with associated state encoding for a different transformation to be considered - namely that there exists a set of 2^N points where the states evaluated within the space encoding are orthonormal.

In particular we consider the Chebyshev space which consists of polynomials useful for certain function fitting tasks. We show the encoding that prepares a state with these polynomials as amplitudes. Due to orthogonality constraints on this space we note the requirement for an ancillary qubit. The transformation is then shown by relating to discrete cosine transform and consequently the Fourier transform. We show a use-case where this transformation is useful – function fitting in the Chebyshev latent space and then sampling in the computational space.

We particularly want to emphasise that using the applicable space for your problem is of great importance – there is not a one space solves all. We have mentioned the existing quantum Fourier transform and introduced the quantum Chebyshev transform but finding further types of transforms suitable for other problems is an area of study for future work.

Chapter 7

Physics-Informed Quantum Machine Learning: Solving nonlinear differential equations in latent spaces without costly grid evaluations

7.1 Declaration of Contribution

The work presented in this chapter is available as a manuscript [181] and is currently under consideration for publication. My contributions consist of the initial idea and development (with supervision), implementation and running of simulations and being the main writer.

7.2 Introduction

Our aim is to solve differential equations by encoding their solutions into quantum models and, broadly speaking, utilising a quantum parallelism [6]. The protocol employs physics-informed constraints for learning, and efficiently evaluates loss terms on an entire computational grid (specifically, for the full set of independent basis functions). By avoiding direct grid point evaluations a source of inefficiency in many DE solvers is circumvented, particularly for problems with significant enough number of grid points that evaluating at each is a costly procedure such as multidimensional problems.

We propose a physics-informed quantum algorithm to solve nonlinear and multidimensional differential equations (DEs) in a quantum latent space. We suggest a strategy for building quantum models as state overlaps, where exponentially large sets of independent basis functions are used for implicitly representing solutions. By measuring the overlaps between states which are representations of DE terms, we construct a loss that does not require independent sequential function evaluations on grid points. In this sense, the solver evaluates the loss in an intrinsically parallel way, utilising a global type of the model. When the loss is trained variationally, our approach can be related to the differentiable quantum circuit protocol, but does not scale with the training grid size.

Specifically, using the proposed model definition and feature map encoding, we represent function- and derivative-based terms of a differential equation as corresponding quantum states. Importantly, we propose an efficient way for encoding nonlinearity, for some bases requiring only an additive linear increase of the system size $O(N + p)$ in the degree of nonlinearity p. By utilising basis mapping, we show how the proposed model can be evaluated explicitly. This allows to implement arbitrary functions of independent variables, treat problems with various initial and boundary conditions, and include data and regularization terms in the physics-informed machine learning setting. On the technical side, we present toolboxes for exponential Chebyshev and Fourier basis sets, developing tools for automatic differentiation and multiplication, implementing nonlinearity, and describing multivariate extensions. The approach is compatible with, and tested on, a range of problems including linear, nonlinear and multidimensional differential equations.

7.3 Method

7.3.1 Informal Overview

To start with an informal overview, imagine that you need to solve a differential equation for some function $f(x)$ of a scalar variable x. The equation involves derivatives (df/dx, d^2f/dx^2 etc.), and it also may include the function itself, its square, plus other functions that depend on x. This can be readily extended to multidimensional problems that include other independent variables (y etc.). Additionally, the problem of solving differential equations assumes specifying initial or boundary terms. Say, we want to solve $df/dx - f(x) = 0$. How can we check if the derivative indeed matches the function itself? We posit that this can be addressed as a global evaluation of compatibility of the two DE terms in a *latent space*.

Let us imagine a collection of DE terms as the problem space (Fig. 7.1, ground layer). Our next step is to associate each term with a quantum state living in an N-qubit Hilbert space that spans 2^N states. We refer to this as a the latent space, where DE terms are elevated to (Fig. 7.1, middle layer). For this, we develop a recipe (mapping $f \to |f\rangle$, $df/dx \to |f'\rangle$, $x \to |x\rangle$ etc) such that each term in the problem space has its own quantum representation. Here, we make sure that different DE terms can be projected in the same basis (for instance, shown as a set of Chebyshev polynomials in Fig. 7.1), and the corresponding functions can be read out as overlaps with states representing variables (i.e. $|x\rangle$). Next, we need to check if the constraints hold. Returning back to the simple example above, we compare the terms df/dx and $f(x)$ in the same basis by measuring the effective distance between latent space representations, corresponding to the overlap of quantum states $|f'\rangle$ and $|f\rangle$. This is repeated for all terms in the problem, and our goal is to learn the solution that satisfies constraints and initial values (or some data). Here, we can write a total loss as a sum of individual contributions L_i (Fig. 7.1, top layer) and learn the solution variationally for an adjustable model

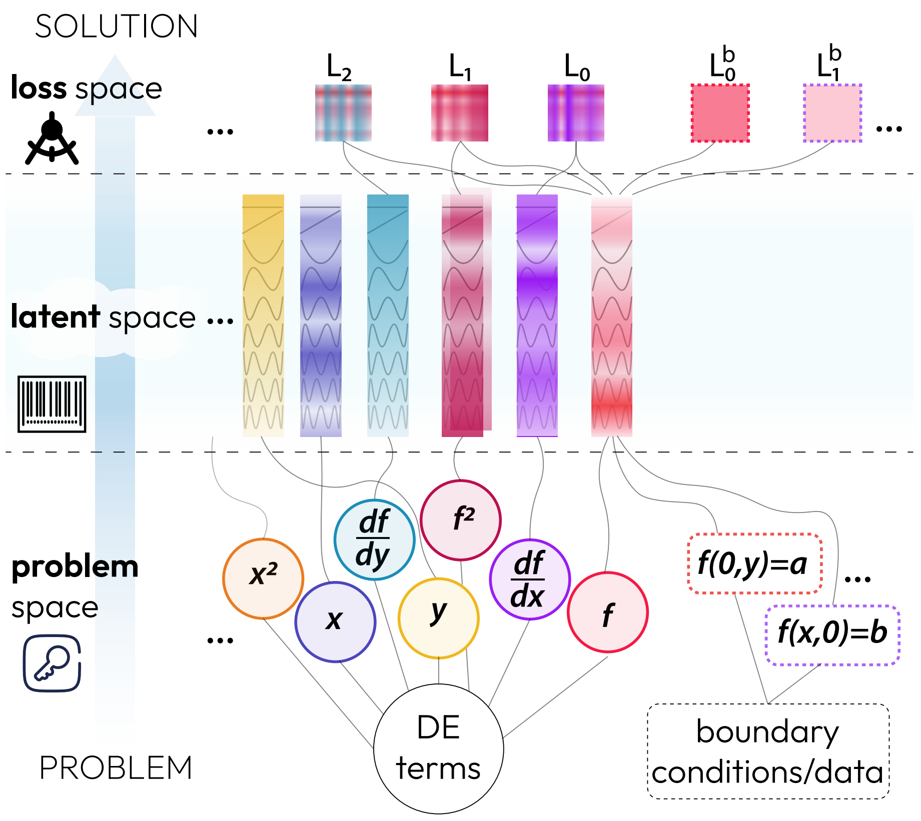


Figure 7.1: **A conceptual visualisation of the proposed physics-informed approach to solving nonlinear differential equations.** We start from the problem space, where differential equation (DE) terms, their relations, and the boundary terms are provided. Next, these terms are elevated to the latent space via feature map that prepares corresponding quantum states, with amplitudes associated to components of independent basis functions. These states are then compared via pairwise overlaps, forming corresponding distances in the loss space, and boundary terms are evaluated explicitly via basis mappings. The total loss can be minimised to learn a model that provides a solution to the problem.

$f_\theta(x)$, similarly to classical physics-informed neural network approach and quantum approaches for derivative quantum circuits [112]. Alternatively, we can use the evaluated constraints to form a system of equations and solve it using quantum matrix inversion techniques.

7.3.2 Technical Overview

We proceed with a formal definition of the problem. The goal of the protocol is to solve a system of D differential equations,

$$\mathrm{DE}_j(\boldsymbol{x}, \boldsymbol{f}, d\boldsymbol{f}/d\boldsymbol{x}, d^2\boldsymbol{f}/d\boldsymbol{x}^2, ...) = 0 \quad \text{for } j = 1 : D, \tag{7.1}$$

where $\mathrm{DE}_j(\boldsymbol{v})$ denotes a sum of terms $\boldsymbol{v}$ in j-th differential equation that equates to zero. Additionally, the initial or boundary conditions have to be specified and satisfied. We want to obtain a solution of Eq. (7.1) as functions $\boldsymbol{f}(\boldsymbol{x})$ of variables $\boldsymbol{x}$. For this, we use a quantum model $\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x})$ parametrised by a vector of weights $\boldsymbol{\theta}$, and reformulate the problem as learning an optimal model $\boldsymbol{f}_{\theta^*}(\boldsymbol{x})$ to represent $\boldsymbol{f}(\boldsymbol{x})$. Specific to our protocol, we set up the model in the form that can be evaluated at any value of variable $\boldsymbol{x}$, while also admitting the latent space representation such that the learning process does not require explicit evaluation of the quantum model at the grid of training points. To keep the discussion simple, initially we consider the case with $D = 1$, omitting the sub-indices ($\mathrm{DE}_{j=1} \equiv \mathrm{DE}$), keeping first-order derivatives only, and a single dimension for both dependent and independent variables, i.e. $\mathrm{DE}(x, f, df/dx) = 0$. The details of generalisation are described in the last subsection of the protocol section.

The first step is to note that the DE can be written as

$$\mathrm{DE}(x, f, df/dx) = \sum_{k=1}^{T} \mathrm{DE}^{[k]}(x, f, df/dx) = 0, \tag{7.2}$$

where $\{\mathrm{DE}^{[k]}(x, f, df/dx)\}_k$ denote the separate terms of the differential equation formed of products of f, and df/dx as well as arbitrary functions of x. We instantiate a quantum model in the general form

$$f_\theta(x) = \langle x|f_\theta\rangle\!\rangle, \tag{7.3}$$

represented by the overlap between a quantum state $|x\rangle$ and a classical linear sum of quantum states $|f_\theta\rangle\!\rangle$. We define this combination as a classical sum of quantum states weighted with individual coefficients and label it using the double-ket notation, $|\circ\rangle\!\rangle$. For example $|\psi\rangle\!\rangle = \sum_j \alpha_j |\psi_j\rangle$. The coefficients $\{\alpha_j\}_j$ do not need to satisfy any normalisation conditions. This has previously been used in quantum filter diagonalisation [182, 183, 184]), We stress that single-state models are naturally included in this definition, $|f_\theta\rangle \in |f_\theta\rangle\!\rangle$, and can readily be used. We also introduce the state $|x\rangle$ that encodes the variable x, in particular using the computational basis. Importantly, we can generate $|x\rangle = \hat{\mathcal{U}}(x)|0\rangle$ by applying an x-dependent unitary operator to the computational zero state $|0\rangle$ [137, 175]. Here $\hat{\mathcal{U}}(x)$ represents a feature map [28, 136] as introduced in section 2.4.1. Within the proposed quantum model definition, we posit that each term $\mathrm{DE}^{[k]}$ can be expressed as

$$\mathrm{DE}^{[k]}(x, f_\theta, df_\theta/dx) = \langle\tilde{x}|\mathrm{DE}^k_\theta\rangle\!\rangle, \tag{7.4}$$

$$|\mathrm{DE}^k_\theta\rangle\!\rangle = \sum_{\ell=1}^{t_k} \alpha^k_\ell |\Delta^{k,\ell}_\theta\rangle, \tag{7.5}$$

where $|\tilde{x}\rangle$ is a quantum state corresponding to the latent space representation of the variable. We note that $|\tilde{x}\rangle$ is the same for all terms when forming the model. However, it may differ from $|x\rangle$ and imply an alternative basis. This will be specifically required when dealing with product terms later. In Eq. (7.5) we explicitly show that each term of the differential equation $\mathrm{DE}^{[k]}$ can be based on a sum of t_k parametrised quantum states $\{|\Delta^{k,\ell}_\theta\rangle\}_{\ell=1}^{t_k}$, weighted by coefficients $\{\alpha^k_\ell\}_{\ell=1}^{t_k}$. Here, we keep the mixture states $|\mathrm{DE}^k_\theta\rangle\!\rangle$ in the general form, and provide the details for their construction in the following section. Substituting the expression in Eq. (7.4) into Eq. (7.2), we get

$$\mathrm{DE}(x, f_\theta, df_\theta/dx) = \langle\tilde{x}| \sum_{k=1}^{T} |\mathrm{DE}^k_\theta\rangle\!\rangle = 0, \tag{7.6}$$

for all x. Essentially, Eq. (7.6) represents a check of T differential constraints for the model (7.3) based on quantum state overlaps with the same state $|\tilde{x}\rangle$. We additionally restrict the state $|\tilde{x}\rangle$ such that its amplitudes when considered as functions of x (as in section 2.5) form a linearly independent set of functions (discussed in detail later). With this restriction the differential equation is satisfied if and only if $\sum_{k=1}^{T} |\mathrm{DE}^k_\theta\rangle\!\rangle = \mathbf{0}$, where the used notation implies that every element of the differential constraint vector has zero value. If the restriction on $|\tilde{x}\rangle$ is not in place then $\sum_{k=1}^{T} |\mathrm{DE}^k_\theta\rangle\!\rangle = \mathbf{0}$ is sufficient but not a necessary condition. In this case, valid solutions of the problem would not be valid solutions of training for $\sum_{k=1}^{T} |\mathrm{DE}^k_\theta\rangle\!\rangle = \mathbf{0}$. Therefore, by constructing $|\mathrm{DE}^k_\theta\rangle\!\rangle$ for each differential equation term with adjustable parameters the solution can be learnt by solving $\sum_{k=1}^{T} |\mathrm{DE}_\theta^k\rangle\rangle = 0$ in several ways.

Loss-based Variational Optimisation

For solving this problem we consider a variational approach based on minimising the loss function

$$\mathcal{L}_{\mathrm{DE}}(\theta) = \sum_{k=1}^{T} \langle\langle \mathrm{DE}_\theta^k| \sum_{m=1}^{T} |\mathrm{DE}_\theta^m\rangle\rangle = \sum_{k=1}^{T}\sum_{m=1}^{T}\sum_{\ell=1}^{t_k}\sum_{n=1}^{t_m} \alpha_\ell^k \alpha_n^m \langle \Delta_\theta^{k,\ell} | \Delta_\theta^{m,n}\rangle. \tag{7.7}$$

To evaluate the loss function $\mathcal{L}_{\mathrm{DE}}(\theta)$ the overlaps between prepared quantum states are measured and then post-processed classically to account for weights. Similarly, we need to evaluate derivative-based contributions. Once states are prepared, the overlaps are estimated using known methods, for instance the Hadamard test [70] as detailed in section 2.7.1 and other techniques typically used in quantum kernel methods [136, 158] or ground state estimation [184, 183]. We provide a brief summary and circuits for evaluating overlaps in section 2.7.1.

Next, we need to introduce the boundary or initial conditions for solving differential equations. The significance of this step shall not be underestimated, as differential constraints can be satisfied in many ways, and it is the initial value terms that pin the solution. Similarly, one can imagine problems where a set of data is supplied to regularise $f_\theta(x)$, and we need to include this information when learning the solution. We resolve the question of introducing initial value terms by utilising the access to our model via quantum feature map circuits. Namely, the function can be evaluated at some initial point x_0 as $f_\theta(x_0) = \langle 0|\hat{\mathcal{U}}(x_0)|f_\theta\rangle\rangle$, rather than globally. For instance, given the initial value condition $f(x_0) = f_0$, we can introduce a corresponding loss term as a distance measure between $f_\theta(x_0)$ and f_0, which can be additionally weighted by a pre-defined constant η. Using the mean square error (MSE) for defining the distance, this loss term is

$$\mathcal{L}_{\mathrm{init}}(\theta) = \eta \left\{ f_\theta(x_0) - f_0 \right\}^2, \tag{7.8}$$

where η controls the importance of the initial value contribution. Similarly, we can include data dependence for regularising the solution as

$$\mathcal{L}_{\text{data}}(\theta) = \zeta \sum_{x_i \in \mathcal{X}} \{f_\theta(x_i) - f_i\}^2 \, , \tag{7.9}$$

where $\mathcal{X} = \{x_i\}_i$ are grid points in the supplied data set, with corresponding function values $\{f_i\}_i$, and ζ being a weighting parameter.

The total loss is then

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{DE}}(\theta) + \mathcal{L}_{\text{init}}(\theta) + \mathcal{L}_{\text{data}}(\theta), \tag{7.10}$$

which is to be minimised either by non-convex optimisation methods, finding $\theta^* = \text{argmin}_\theta \mathcal{L}(\theta)$, or applying other iterative methods to recover the optimal mixture state $|f_{\theta^*}\rangle\!\rangle$. When choosing the variational optimisation, it is convenient to use a gradient descent and adjust angles θ based on $\nabla_\theta \mathcal{L}(\theta)$. This can be approached by various circuit differentiation techniques, including the parameter shift rule [39, 28] as introduced in section 2.4.4 and generalisations for wider range of circuits [40, 163, 41], or LCU-based derivatives that are specifically fitting the differentiation of kernels [158, 70].

Alternatively a matrix inversion-based workflow that avoids variational approaches and exploits the same problem encoding in the latent space could be used. In this work we focus on the variational approach with this alternative approach left for future development.

Finally, the overall workflow for the protocol (Fig. 7.1) is summarised below.

1. Choose DE with initial value to solve, thus specifying the problem. Split DE into separate product terms.
2. Choose hyperparameters and set-up workflow components such as x encoding, mixture state for building the model, optimisation procedure etc.

3. Run circuits to prepare relevant state(s) associated to each term (thus instantiating the model in the quantum latent space).
4. Compute overlaps between terms (therefore project results back to scalars forming individual loss contributions).
5. Optimise the total loss until differential and boundary constraints are satisfied.

7.3.3 Function Representation

A vital component of the proposed algorithm is being able to prepare $|\mathrm{DE}^k_\theta\rangle\rangle$ for the various terms that can exist in a generic differential equation. This preparation largely depends on how we represent the x-dependence via feature mapping $|x\rangle = \hat{\mathcal{U}}(x)|0\rangle$. The basis functions of the model depend on how we choose $\hat{\mathcal{U}}(x)$ acting on N qubits. Certain limitations on $\hat{\mathcal{U}}(x)$ are imposed such that states associated with derivatives with respect to x, functions of x, and products can be prepared. The first restriction on $\hat{\mathcal{U}}(x)|0\rangle$ is such that the generated basis functions are independent. To illustrate this, let us write explicitly the state representing the variable as a vector with amplitudes τ_j that depend on x,

$$|x\rangle = (\tau_0(x), \tau_1(x), ..., \tau_{2^N-1}(x))^{\mathrm{T}}, \tag{7.11}$$

and choosing 2^N basis functions such that $\{\tau_j\}_j$ are linearly independent, meaning that $\forall\ j \nexists\ \{c_k\}_k$ s.t. $\tau_j(x) = \sum_{k\in\bar{j}} c_k \tau_k(x)$. This ensures that Eq. (7.6) is solved if and only if $\sum_{k=1}^{T} |\mathrm{DE}^k\rangle\rangle = \mathbf{0}$ holds.

Next, as discussed in the previous subsection, our model is built as the overlap $f_\theta(x) = \langle x|f_\theta\rangle\rangle$. Using the same basis as for encoding x, we can express the mixture state as a vector of 2^N components,

$$|f_\theta\rangle\rangle = (f_{\theta,0}, f_{\theta,1}, ..., f_{\theta,2^N-1})^{\mathrm{T}}. \tag{7.12}$$

This leads to the model represented by the sum of the independent basis functions,

$$f_\theta(x) = \langle x | f_\theta \rangle\!\rangle = \sum_{j=0}^{2^N-1} f_{\theta,j} \tau_j^*(x), \tag{7.13}$$

which allows introducing rules for differentiating the model.

In this subsection we further discuss how to prepare $|\mathrm{DE}_\theta^k\rangle\!\rangle$ for the model, its derivatives, other functions of x, and products of these functions for general choice of $\hat{\mathcal{U}}(x)|0\rangle$. Later in the text we will present the specific state preparation strategies for two choices of encoding — orthonormal Fourier and Chebyshev bases [175].

Scaled-and-shifted Function Representation

As long as the conditions on the basis are satisfied, we can further extend the function representation introducing additional scaling factors and additive shifts. This is particularly important when we need to increase or limit the expressivity of our models, also impacting the trainability. Here, the expressivity corresponds to the range of functions it is able to represent, and the trainability defines how easy it is to train the model to represent a particular function that can be expressed. To highlight the need for these considerations, we note that in case of choosing the model as $f_\theta(x) = \langle x|f_\theta\rangle$, it is bound in $0 \leq |f_\theta(x)| \leq 1$, thus being significantly limited in expressivity. One option that is readily available corresponds to introducing a scaling factor for the single-state model, which reads

$$f_\theta(x) = \alpha \langle x|f_\theta\rangle, \tag{7.14}$$

with α being a parameter (fixed or adjustable). Another option corresponds to adding a constant to the model in the form

$$f_\theta^{\mathrm{sh}}(x) = \alpha\langle x|f_\theta\rangle + \alpha_{\mathrm{sh}} \equiv \alpha\langle x|f_\theta\rangle + \alpha_{\mathrm{sh}} g_1(x), \tag{7.15}$$

where we highlight that adding the constant α_{sh} is equivalent to introducing the function $g_1(x) := 1$ (up to a scaling factor). For this we introduce the state $|\psi_1\rangle$ such that $\langle x|\psi_1\rangle = 1$. We note that the shifted model falls into a category of mixture-based models with $|f_\theta\rangle\rangle = \alpha|f_\theta\rangle + \alpha_{\text{sh}}|\psi_1\rangle$, and even more general operators than scaling and shifting may be applied to produce $|f_\theta\rangle\rangle$. In order to be able to use the shifting strategy, we note that the embedding has to be chosen such that the unity function can be expressed by the basis, and that its latent space representation is either analytically or variationally obtainable.

Evaluating Derivatives

To include the differential constraints we need to connect a formal model derivative with the overlap evaluation based on a modified *derivative* (mixture) state. Taking the formal derivative of $f_\theta(x)$ we get

$$f'_\theta(x) = \frac{d}{dx}\langle x|f_\theta\rangle\rangle = \langle 0|\hat{\mathcal{U}}^{\dagger\prime}(x)|f_\theta\rangle\rangle, \tag{7.16}$$

where we need to evaluate the derivative for the conjugate transposed feature map $\hat{\mathcal{U}}^{\dagger\prime}(x)$. We note that for certain x-parametrisation we can associate a derivative of an operator with the product of two operators, $\hat{\mathcal{U}}'(x) = \check{G}\hat{\mathcal{U}}(x)$, where $\check{G}$ is an x-independent operator to be determined. This is the form of embedding we rely upon. As in general the emergent operator (matrix) is not unitary, we use the check mark $\check{\circ}$ to denote this, also marking such operators throughout. In practice, these operations can be readily absorbed into the mixture state definition, or compiled with the linear combination of unitaries (LCU) approach [177], or other decomposition methods (QR, Householder etc [185]). For example, taking a feature map of the form $\exp(-ix\hat{H})$ generated by the Hermitian operator $\hat{H}$ we observe that $\check{G} = -i\hat{H}$ is a skew-Hermitian operator. This can be decomposed into Pauli strings and thus a sum of unitaries. Using the developed notation, the derivative of the model can be expressed as

$$f'_\theta(x) = \langle x|\check{G}^\dagger|f_\theta\rangle\!\rangle \equiv \langle x|f'_\theta\rangle\!\rangle, \tag{7.17}$$

where we have introduced the derivative mixture state $|f'_\theta\rangle\!\rangle := \check{G}^\dagger|f_\theta\rangle\!\rangle$. Crucially, the state $|f'_\theta\rangle\!\rangle$ now becomes our proxy to the derivative term evaluation, and serves as $|\mathrm{DE}_\theta\rangle\!\rangle$ for introducing df_θ/dx into the total loss.

Dealing with Independent Functions

Another type of terms that may arise in differential equations is a function of the independent variable $g(x)$, which we need to include into physics-informed constraints. In this case, we have to exploit the same model structure as for the trial function, and postulate that

$$g(x) = \langle x|g\rangle\!\rangle, \tag{7.18}$$

where $|g\rangle\!\rangle$ is a state that labels the function. Often, it is sufficient to use a single pure state, $|g\rangle\!\rangle = |g\rangle$, with a scale factor.

There are several ways how the function $g(x)$ can be loaded into the system. First, if the x encoding $|\tau(x)\rangle$ is such that at set of 2^N points $\{x_j\}_j$, $\{|\tau(x_j)\rangle\}$ form an orthonormal basis we can use a unitary map $\hat{\mathcal{U}}_\mathrm{T}$ that transforms states between the encoding basis and the computational basis $|\tilde{x}\rangle$ [137, 175]. Using the resolution of identity, we can write the function now as the overlap model in the computational basis, $g(x) = \langle x|\hat{\mathcal{U}}_\mathrm{T}^\dagger\hat{\mathcal{U}}_\mathrm{T}|g\rangle\!\rangle = \langle\tilde{x}|\tilde{g}\rangle\!\rangle$. The state $|\tilde{g}\rangle\!\rangle$ can be identified from the vector of evaluations $\{g(x_j)\}_j$ where $\{x_j\}_j$ are the set of points at which $|x\rangle$ maps to $|\tilde{x}\rangle$. From here we can use quantum state preparation algorithms [186, 187, 188] to create the required component states of $|\tilde{g}\rangle\!\rangle$, sans any prefactors α that can be accounted for separately when evaluating the total loss.

Alternatively, we can also use regression and learn states $|g\rangle\rangle$ for representing functions with another parametrised mixed state $|g_\phi\rangle\rangle$. This can be achieved for instance via quantum circuit learning [28], set up such that $g(x) = \langle x|g_{\phi^*}\rangle\rangle$ for some optimal parameters ϕ^*. Scaling and shifting factors (naturally included in the mixture state representation) would help to express functions with varying magnitude. Another alternative for the case when the basis functions in Eq. (7.11) are known, is to utilise a classical method for spectral function decomposition [189], and load corresponding coefficients with the quantum state preparation algorithms.

Multiplying Functions

Equipped with the knowledge of how to construct various individual functions and derivatives, we proceed to encoding products of functions. Here, the crucial point is that products require: a) loading information about amplitudes of multiplied states; b) keeping track of the basis that must be extended in the nontrivial way. For this, let us consider a minimal example of multiplying two functions $g(x) = \langle x|g\rangle\rangle$ and $h(x) = \langle x|h\rangle\rangle$, based on the corresponding states. We note that the same procedure can be employed to multiply the model $f(x)$ itself, for instance representing powers $f^2(x)$ etc.

As functions $g(x)$ and $h(x)$ both depend on the variable x, we can expand them as $g(x) = \sum_j g_j \tau_j(x)$ and $h(x) = \sum_j h_j \tau_j(x)$, where g_j, h_j are state amplitudes. Therefore, the product of functions corresponds to

$$g(x)h(x) = \sum_{j,k} g_j h_k \tau_j(x)\tau_k(x), \tag{7.19}$$

and we observe that in general it also includes products of basis functions. While previously we specified each function in terms of components for the $\mathcal{A} = \{\tau_j(x)\}_j$ bases, now the emerging basis is $\mathcal{A}^{(2)} = \{\tau_j(x)\tau_k(x)\}_{j,k}$ that may not be unique or independent. To fix this, we consider a new basis set $\mathcal{B} = \{\beta_j(x)\}_j$ which is independent, and choose it such that it contains required products, $\mathcal{A}^{(2)} \subset \mathcal{B}$, being also padded to have a power-of-two cardinality. Next, we demand that there exists a state $|xx\rangle$ that provides such basis, and that the product can be defined via the overlap with some state $|gh\rangle\!\rangle$,

$$g(x)h(x) = \langle xx|gh\rangle\!\rangle = \sum_j (gh)_j \beta_j(x), \tag{7.20}$$

where $(gh)_j$ are amplitudes for this state. We note that the encoding circuit for $|xx\rangle$ can differ from $|x\rangle$, however it is often heavily related to $\hat{\mathcal{U}}(x)$ (see the Model Encoding section). Furthermore when $|xx\rangle$ does differ from $|x\rangle$ we still need to be able to represent every term of the DE with the same x dependence. To achieve this any single function terms (e.g. $h(x)$ alone) can be taken and multiplied with $g(x) = 1$ as a trivial function. All terms are then expressed via $|xx\rangle$ instead of $|x\rangle$.

Knowing that $g(x)h(x)$ can be expressed in terms of $|xx\rangle$ extended variable state, we need to know how to prepare $|gh\rangle\!\rangle$. Specifically, this assumes that we have access to circuit that create $|g\rangle\!\rangle$ and $|h\rangle\!\rangle$. We note that original product bases in $\mathcal{A}^{(2)}$ can be expanded in the extended $\mathcal{B}$ basis as

$$\tau_j(x)\tau_k(x) = \sum_{l=0}^{2^{\tilde{N}}-1} b_l^{j,k} \beta_l(x), \tag{7.21}$$

where $b_l^{j,k}$ are expansion coefficients, and $\tilde{N}$ is the smallest integer that accommodates $\mathcal{B}$ with the specified properties. Substituting this into Eq. (7.19), we observe that

$$g(x)h(x) = \sum_{j=0}^{2^N-1} \sum_{k=0}^{2^N-1} \sum_{l=0}^{2^{\tilde{N}}-1} g_j h_k b_l^{j,k} \beta_l(x) = \sum_{j=0}^{2^{\tilde{N}}-1} (gh)_j \beta_j(x), \tag{7.22}$$

and the linear relation between products of amplitudes $\{g_j h_k\}_{j,k}$ and the amplitudes $\{(gh)_j\}_j$ of the product state can be seen. Therefore, a matrix $\check{M}$ can be constructed such that $|gh\rangle\!\rangle = \check{M}|g\rangle\!\rangle|h\rangle\!\rangle$. Implementation of $\check{M}$ depends on the choice of $\hat{\mathcal{U}}(x)$. Later we describe it for the two specific choices of Fourier and Chebyshev encodings.

Finally, we note that similar strategy can be applied to cases where more that two functions shall be multiplied, where essential steps correspond to defining a suitable variable state in the extended basis, and the corresponding multiply operator.

7.3.4 Generalisations

So far we have described a simple scenario of solving a first order, one-dimensional differential equation. However, many differential equations of interest have more general forms. Next, we briefly discuss how to expand our approach for a wider range of differential equations.

Introducing nonlinearities.—In the preceding subsection we have shown that latent space embedded functions can be multiplied in the extended basis. Similar procedure applies for higher-order terms with powers of $f^M(x)$ arranged for extended registers. For equations that include nonlinear function of the model, e.g. $\cos[f(x)]$, we need to use series expansions (Taylor, Fourier, Chebyshev etc), truncating the degree of nonlinearity at some acceptable level.

Evaluating higher-order derivatives.—Previously we have suggested that derivatives for overlap models with specific encodings $\hat{\mathcal{U}}(x)$ can be evaluated as $f_\theta'(x) = \langle x|\check{G}^\dagger|f_\theta\rangle\!\rangle$. This simply follows from the fact that $\check{G}$ is a generator for our encoding. Higher-order derivatives follow the same strategy, where we repeatedly "lower-down" and concatenate generators, leading to the m^{th}-order derivative

$$f_\theta^{(m)}(x) = \langle x|(\check{G}^\dagger)^m|f_\theta\rangle\!\rangle = \langle x|f_\theta^{(m)}\rangle\!\rangle. \tag{7.23}$$

Solving system of differential equations.—For systems of differential equations, we can treat each separate equation as a part of differential constraints. This leads to the total loss that grows with the number of differential equations, being the sum of contributions,

$$\mathcal{L}_{\text{system}}(\theta) = \sum_{j=1}^{D} \sum_{k=1}^{T} \langle\langle \mathrm{DE}_j^{[k]} | \sum_{l=1}^{T} | \mathrm{DE}_j^{[l]} \rangle\rangle. \tag{7.24}$$

Tackling multidimensional problems.—For differential equations that feature several independent variables, generally we need to change the encoding strategy accommodating a vector $\boldsymbol{x}$. This can be done by developing encoding circuits for each component, and using the tensor product structure for dealing with them. For instance, consider the simplest generalisation to two variables x and y. The two-variable model is then instantiated as

$$f_\theta(x, y) = (\langle x| \otimes \langle y|)|f_\theta\rangle\rangle, \tag{7.25}$$

and we can extend the tensor product to multiple variables $\{x_i\}_i$ with separate registers. When working with these variables we need to employ parallel feature maps, $\bigotimes_i \hat{\mathcal{U}}^i(x_i)$. Correspondingly, the multidimensional transform operator reads $\bigotimes_i \hat{\mathcal{U}}^i_{\mathrm{T}}$, where $\hat{\mathcal{U}}^i_{\mathrm{T}}$ is the transform for the i^{th} encoding (which can be different). Next, to take the derivative with respect to the i^{th} variable one needs to act on the i^{th} register with $\check{G}_i^\dagger$, being the associated derivative operator for the i^{th} encoding. The multiplier circuit is also constructed depending on the choice of encoding. We also note that one can employ an alternative strategy of encoding multiple variables on the same register using serial circuits (with controls), and leave this as a question for future investigations.

7.3.5 Model Encoding

Until this point, we have kept the approach general and valid for any choice of encoding that satisfies requirements for the parallel (or global) evaluation. In this section, we introduce two specific example choices of encoding models for global physics-informed quantum DE solvers. For each choice, we detail the components and circuits required for the full workflow. These include feature map $\hat{\mathcal{U}}(x)$, differentiation rules for $d\hat{\mathcal{U}}(x)/dx$ that involve the generator $\check{G}$, transformation circuit $\hat{\mathcal{U}}_\mathrm{T}$, and the multiplication operator $\check{M}$. The choices are the Chebyshev encoding [175] and the Fourier encoding [137], but we stress that other encodings can be developed in a similar way, for instance the encoding used in recently proposed harmonic neural networks [190].

7.3.6 Chebyshev Encoding

Feature Map

The Chebyshev polynomials represent a high-performing basis for regression, function fitting, and integral evaluation tasks [92]. They are widely used in spectral methods for solving differential equations, financial modelling, and describing many-body systems. In quantum SciML Chebyshev polynomials were used for solving differential equations with derivative quantum circuits [112, 123], though without linear basis independence. Recently, we have proposed a protocol for generating an orthonormal Chebyshev basis set via quantum feature maps [175] which is detailed in chapter 6. Building up on the developed toolbox, we show that this basis type enables building global Chebyshev models in the quantum latent space.

We choose the encoding in such a way that amplitudes of a quantum state correspond to x-dependent Chebyshev polynomials of the first kind $T_k(x) \equiv \cos(k \arccos x)$, where k denotes a polynomial degree. The corresponding variable encoding state reads [175]

$$|x\rangle = \frac{1}{\mathcal{N}_N(x)} \left(\frac{1}{2^{N/2}} T_0(x)|0\rangle + \frac{1}{2^{(N-1)/2}} \sum_{k=1}^{2^N-1} T_k(x)|k\rangle \right), \tag{7.26}$$

where we ensure that the states $\{|x\rangle\}$ are orthogonal on the Chebyshev grid with 2^N points. This N-qubit state is normalised by

$$\mathcal{N}_N(x) = \frac{1}{2^{(N-1)/2}} \left(1/2 + \sum_{k=1}^{2^N-1} T_k^2(x) \right)^{1/2}, \tag{7.27}$$

where we note that in general it is x-dependent when evaluated outside of the Chebyshev grid. We want the Chebyshev polynomials themselves to be the basis functions, not rescaled by $\mathcal{N}_N(x)$. Thus, we introduce the corresponding model written in the form

$$f_\theta(x) = \mathcal{N}_N(x)\langle x|f_\theta\rangle\!\rangle. \tag{7.28}$$

The multiplier $\mathcal{N}_N(x)$ leads to a slightly modified workflow concerning post-processing, where relevant overlaps are effectively evaluated with the state $\mathcal{N}_N(x)\langle x|$. However, the encoding of variable x is still based on independent basis functions, and all steps remain valid.

Derivatives

To encode derivatives for Chebyshev-based models, we utilise the properties of Chebyshev polynomials. Specifically, it is known that derivatives can be expressed as $T'_n(x) = nU_{n-1}(x)$, where $U_n(x)$ are Chebyshev polynomials of the second kind [191]. This can then be further expanded to

$$T'_0(x) = 0, \tag{7.29}$$

$$T'_{2n}(x) = 4n \sum_{m=1}^{n} T_{2m-1}(x), \tag{7.30}$$

$$T'_{2n+1}(x) = (4n+2) \sum_{m=1}^{n} T_{2m}(x) + (2n+1)T_0(x). \tag{7.31}$$

Therefore, for all Chebyshev polynomials we can write the derivative in an analytical form as a sum

$$T'_n(x) = \sum_{j=0}^{2^N-1} w_j^n T_j(x), \tag{7.32}$$

where coefficients w_j^n for each degree are simply collecting the terms from differentiation rules in Eqs. (7.29)–(7.31).

We reiterate that our goal is creating a derivative state, where amplitudes are now differentiated with respect to x. Note that derivatives are exact (no finite differencing is involved), and the procedure can be considered as an automatic quantum differentiation. From this, a derivative operator $\check{G}$ is composed such that

$$\frac{d}{dx}\mathcal{N}_N(x)|x\rangle = \check{G}\mathcal{N}_N(x)|x\rangle, \tag{7.33}$$

with matrix elements of $\check{G}$ corresponding to $G_{i,j} = w^i_j c_j$, where $c_0 = \sqrt{2}$, and $c_j = 1$ for $j \neq 0$. The additional scaling introduced as coefficients $\{c_j\}$ comes from the distinct prefactor of zero degree polynomial $T_0(x)$ (i.e. constant term) compared to other amplitudes of $|x\rangle$, which originates from the Chebyshev orthonormality conditions [175]. Since $\check{G}$ is generally a non-unitary matrix, it can be implemented with the LCU approach, including its ancilla-free versions without increasing the circuit depth, and encoding the action of $\check{G}$ with the help of an ancilla qubit [71, 192]. With this we get the derivative expressed as

$$f'_\theta(x) = \mathcal{N}_N(x)\langle x|\check{G}^\dagger|f_\theta\rangle. \tag{7.34}$$

Transform

At the Chebyshev nodes $\{x_j\}_j,\ x_j = \cos[(2j+1)\pi/2^{N+1}]$, the states prepared by the Chebyshev feature map $\{\mathcal{N}(x_j)|x_j\rangle\}_j$ form an orthogonal basis which we call the Chebyshev basis. Therefore, there exists an operator $\hat{\mathcal{U}}_{\mathrm{QChT}}$ which transforms between the computational basis (real-space grid) and the Chebyshev basis. The corresponding transformation operator is introduced in chapter 6 [175] and allows relating functions values at $\{x_j\}_j$ with amplitudes of a state via

$$g(x) = \mathcal{N}_N(x)\langle x|\hat{\mathcal{U}}_{\mathrm{QChT}}\hat{\mathcal{U}}^\dagger_{\mathrm{QChT}}|g\rangle. \tag{7.35}$$

Multiplier

Next, we discuss how to build the multiplier in the Chebyshev basis and introducing nonlinear terms in differential equations, which is a non-trivial problem.

We begin by recalling the multiplication rules of the Chebyshev polynomials,

$$T_j(x)T_k(x) = \left[T_{j+k}(x) + T_{|j-k|}(x)\right]/2, \tag{7.36}$$

and aim to exploit them for the automatic multiplication. From the relation above we observe that the basis of the product of two functions based on Chebyshev polynomials up to T_{2^N-1} can be expressed by a Chebyshev polynomial expansion up to degree $2(2^N-1)$. As $2(2^N-1) < 2^{N+1}-1$ it can be represented by the $N+1$ qubit Chebyshev basis. Specifically, we can write the product basis as

$$\mathcal{N}_{N+1}(x)|xx\rangle = \frac{1}{2^{(N+1)/2}}T_0(x)|0\rangle + \frac{1}{2^{N/2}}\sum_{k=1}^{2^{N+1}-1} T_k(x)|k\rangle, \tag{7.37}$$

which is simply the Chebyshev encoding for $N+1$ qubits. Using Eq. (7.36), the desired product state $|gh\rangle$ has to be of the form

$$|gh\rangle_l = \frac{1}{2^{N/2}}\sum_{j=\max(0,l-2^N+1)}^{\min(l,2^N-1)} \frac{c_l}{c_j c_{l-j}}|g\rangle_j|h\rangle_{l-j} + \frac{1}{2^{N/2}}\sum_{j=l}^{2^N-1}\frac{1}{c_j c_{j-l} c_l}\left(|g\rangle_j|h\rangle_{j-l} + |g\rangle_{j-l}|h\rangle_j\right), \tag{7.38}$$

where $\boldsymbol{c}$ has values $c_0 = \sqrt{2}$ for $c_j = 1\ j \neq 0$, and $|\cdot\rangle_j$ represents the j^{th} bit of the state $|\cdot\rangle$. $\boldsymbol{c}$ accounts for the different coefficient of $T_0(x)$. We need to implement an operator $\check{M}$ that acts as

$$\check{M} : |g\rangle|h\rangle|0\rangle \rightarrow \frac{1}{2^{N/2}}|a\rangle\sum_{j,k}\frac{c_{j+k}}{c_j c_k}g_j h_k|j+k\rangle + \frac{1}{2^{N/2}}|\tilde{a}\rangle\sum_{j,k}\frac{c_{|j-k|}}{c_j c_k}g_j h_k||j-k|\rangle, \tag{7.39}$$

where $|a\rangle$ and $|\tilde{a}\rangle$ are (combined registers of) ancillary states that can be discarded later. This operator can be split into two parts, where $\check{M}_+$ prepares the summation $|j+k\rangle$ with corresponding amplitude multiplication, and $\check{M}_-$ prepares the summation of $||j-k|\rangle$. For $\check{M}_+$ we use the adder circuit. The adder circuits were introduced in section 2.7.4 and can be implemented in several ways [74, 75]. We consider

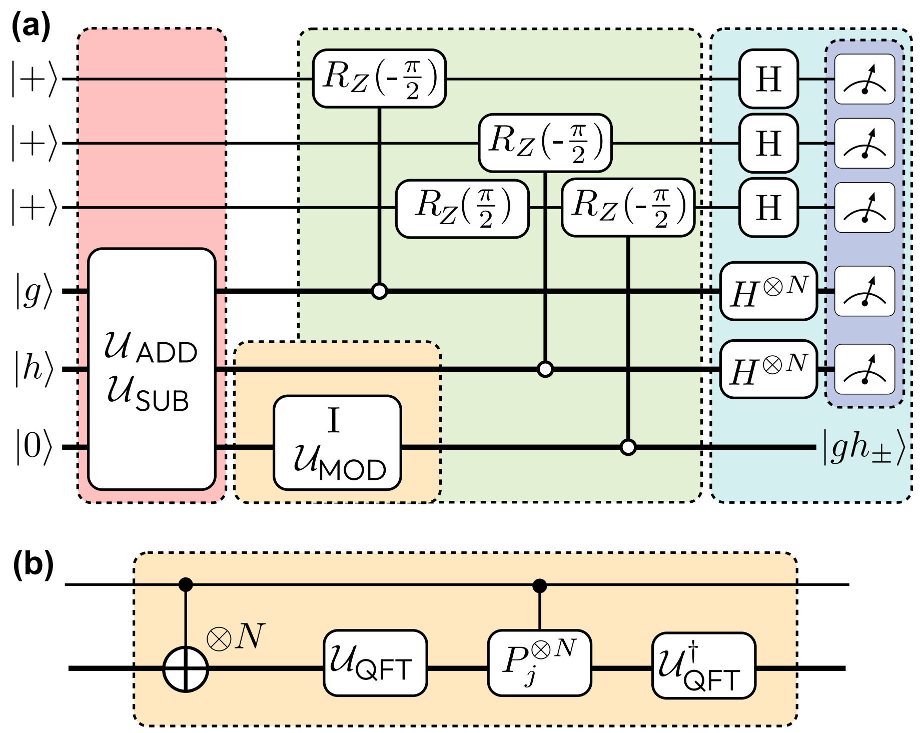


Figure 7.2: **Circuit diagram for multiplying functions in Chebyshev basis. (a)** Circuit diagram of multiplication circuits $\check{M}_+$ and $\check{M}_-$, which together implement the multiplier operator $\check{M}$ for the Chebyshev encoding. First, either $\hat{\mathcal{U}}_{\text{ADD}}$ or $\hat{\mathcal{U}}_{\text{SUB}}$ are applied followed by identity or $\hat{\mathcal{U}}_{\text{MOD}}$, respectively. Details of $\hat{\mathcal{U}}_{\text{MOD}}$ are shown in figure panel (b). Then, the block of rotations $\hat{C}$ is applied to alter the coefficients. Finally, Hadamards on all ancillary qubits are applied before a projective measurement onto $|0\rangle$. The probability of success of the projective measurement is used to estimate the renormalisation factors $\tilde{r}_\pm$. **(b)** Circuit diagram of $\hat{\mathcal{U}}_{\text{MOD}}$. First, a layer of CNOTs are applied controlled by the most significant bit (MSB). Then via the quantum Fourier transform (QFT) implementation of the adder circuit the MSB is added to the remainder of the register. The phase gate $\hat{P}_j$ corresponds to $\mathrm{diag}([1, \exp(2\pi i/2^j)$, where j is a qubit index.

the realisation built from blocks shown in Fig. 2.14(a), which acts as [193, 194]

$$\hat{\mathcal{U}}_{\text{ADD}} : |g\rangle|h\rangle|0\rangle \to \sum_{j,k} |j\rangle|k\rangle g_j h_k |j+k\rangle. \tag{7.40}$$

This is close to our desired operation, and can be used as a base for $\check{M}_+$ (Fig. 7.2a). Next, the effects of varying coefficients c need to be included. After the adder is applied, an operator $\hat{C}$ is needed which acts according to the rules:

1. If $|j\rangle = |0\rangle$ alter shift by $\frac{1}{\sqrt{2}}$.

2. If $|k\rangle = |0\rangle$ alter shift by $\frac{1}{\sqrt{2}}$.

3. If $|j+k\rangle = |0\rangle$ alter shift by $\sqrt{2}$.

To implement this we use controlled $\hat{R}_Z$ rotations acting on ancillas (Fig. 7.2a, three qubits at the top). The ancillas are prepared in the uniform state. Then, one controlled rotation is used for each shift: $\hat{R}_Z(-\pi/2)$ controlled by $|j\rangle = |0\rangle$, $\hat{R}_Z(-\pi/2)$ controlled by $|k\rangle = |0\rangle$, and $\hat{R}_Z(-\pi/2)$ controlled by $|j+k\rangle = |0\rangle$. The last of which if followed by a single-qubit rotation $\hat{R}_Z(\pi/2)$ (not controlled). Hadamards are then applied to the ancilla and a projective measurement onto $|0\rangle$ reduces the operation to

$$\hat{C} : |0\rangle \sum_{j,k} |j\rangle|k\rangle g_j h_k |j+k\rangle \to \frac{1}{r_+}|0\rangle \sum_{j,k} |j\rangle|k\rangle \frac{c_{j+k}}{c_j c_k} g_j h_k |j+k\rangle, \tag{7.41}$$

where r_+ is the renormalisation factor after the measurement and is accounted for later.

In Eq. (7.41) registers $|j\rangle$ and $|k\rangle$ are entangled with $|j+k\rangle$. To discard the loading registers safely, we want to remove this entanglement, and apply a layer of Hadamards to all qubits within $|j\rangle$ and $|k\rangle$ followed by projective measurements onto $|0\rangle$. The corresponding operator $\check{D}$ acts as

$$\check{D} : \frac{1}{r_+}|0\rangle \sum_{j,k} |j\rangle|k\rangle \frac{c_{j+k}}{c_j c_k} g_j h_k |j+k\rangle \to \frac{1}{\tilde{r}_+}|0\rangle|0\rangle|0\rangle \sum_{j,k} \frac{c_{j+k}}{c_j c_k} g_j h_k |j+k\rangle, \tag{7.42}$$

which is our desired state other than the renormalisation factor $\tilde{r}_+$. This constant $\tilde{r}_+$ can be estimated as the square root of the success probability of the projective measurements onto $|0\rangle$. We can now compile $\check{M}_+$ as

$$\check{M}_+ = \tilde{r}_+ \check{D}\hat{C}\hat{\mathcal{U}}_{\mathrm{ADD}}. \tag{7.43}$$

The subtraction part $\check{M}_-$ of the multiplier is prepared in similar way. We start with a subtractor circuit as a base,

$$\hat{\mathcal{U}}_{\mathrm{SUB}} : |g\rangle|h\rangle|0\rangle \to \sum_{j,k} |j\rangle|k\rangle g_j h_k |j-k \bmod 2^{N+1}\rangle. \tag{7.44}$$

A subtractor circuit is generally an adapted adder circuit, as subtraction is the inverse operation of addition. This was introduced alongside the adder in section 2.7.4 with our chosen implementation being built from blocks shown in Fig. 2.14(b). An extra operator $\hat{\mathcal{U}}_{\mathrm{MOD}}$ is now needed (Fig. 7.2b), as we want to prepare $||j-k|\rangle$ but the subtractor prepares $|(j-k) \bmod 2^{N+1}\rangle$. Therefore, when $(j-k)<0$ we need to map states from $|(j-k) \bmod 2^{N+1}\rangle = |2^{N+1}+j-k\rangle$ to $|k-j\rangle$. Since $|j-k| < 2^N$, when using $N+1$ qubits we can separate the cases of $(j-k)<0$ and $(j-k)\geq 0$ by the value of the most significant bit (MSB) in the result register. Namely, we get $|1\rangle$ in MSB if $(j-k)<0$, and $|0\rangle$ otherwise. To implement the mapping, a layer of CNOTs controlled by the MSB is applied to the result register, performing $|2^{N+1}+j-k\rangle \rightarrow |k-j-1\rangle$. Then, controlled again by the MSB in $|1\rangle$, the state is added with the adder circuit to the result register for $||j-k|\rangle$,

$$\hat{\mathcal{U}}_{\mathrm{MOD}} : \sum_{j,k} |j\rangle|k\rangle g_j h_k |j-k \bmod 2^{N+1}\rangle \rightarrow \sum_{j,k} |j\rangle|k\rangle g_j h_k ||j-k|\rangle. \tag{7.45}$$

For the rest of the circuit, we also need to introduce rescaling to account for the differing coefficient of T_0. The operator $\hat{C}$ can be used same way as before. Also similar to the previous case, the other registers need to be disentangled from the result register with Hadamards and projective measurements, with the renormalisation factor $\tilde{r}_-$. This operator $\check{\check{D}}$ slightly differs from $\check{D}$ in that it must also apply to the MSB of the results register. This introduces an extra scale factor of 2. Summarising these steps, we have

$$\check{\check{D}}\hat{C} : \sum_{j,k} |j\rangle|k\rangle g_j h_k ||j-k|\rangle \rightarrow \frac{2}{\tilde{r}_-}|0\rangle|0\rangle \sum_{j,k} \frac{c_{|j-k|}}{c_j c_k} g_j h_k ||j-k|\rangle, \tag{7.46}$$

$$\check{M}_- = \frac{\tilde{r}_-}{2}\check{\check{D}}\hat{C}\hat{\mathcal{U}}_{\mathrm{MOD}}\hat{\mathcal{U}}_{\mathrm{SUB}}. \tag{7.47}$$

The entire multiplier operator can be pulled together as

$$\check{M} = \frac{1}{2^{N/2}}\left(\check{M}_+ + \check{M}_-\right), \tag{7.48}$$

and is shown in Fig. 7.2. This generalises for multiplying registers of different sizes N_1 and N_2 with the prefactor of $2^{-N/2}$ changing to $2^{-\min(N_1,N_2)/2}$.

7.3.7 Fourier Encoding

Feature Map

Another popular basis for building various mathematical models is the Fourier basis. This basis is formed of the functions $T_j^N(x) := \exp(i2\pi jx/2^N)$, where N is the number of qubits. The corresponding feature map with exponentially large capacity is referred to as the phase feature map, introduced in Ref. [112]. The phase feature map is formed by a layer of Hadamards on each qubit followed by a layer of controlled phase gates, $\hat{P}_N(x, j) = \mathrm{diag}\{1, \exp(i\pi x 2^{j-N})\}$, where j is a qubit index we act upon. Applied to the computational zero, the phase feature map prepares the state

$$|x\rangle = \frac{1}{2^{N/2}} \sum_{j=0}^{2^N-1} T_j^N(x)|j\rangle. \tag{7.49}$$

The set of corresponding Fourier states $\{|x_j\rangle\}_j$ evaluated at integer points $\{j\}_{j=0}^{2^N-1}$ is orthonormal.

Derivatives

When taking derivatives we observe that

$$\hat{P}'_N(x, j) = \mathrm{diag}(0, i\pi 2^{j-N})P(x, j) = \check{G}_j \hat{P}_N(x, j), \tag{7.50}$$

$$\check{G}_j = i\pi 2^{j-N-1}\hat{I} - i\pi 2^{j-N-1}\hat{Z}. \tag{7.51}$$

Using the product rule, we repeat the procedure for other gates, leading to $\check{G} = \sum_j \check{G}_j$ as the full derivative operator for the Fourier encoding.

Transform

We have already recalled that at the set of points $\{j\}_{j=0}^{2^N-1}$ the Fourier map prepares an orthonormal basis. This basis can be mapped to the computational basis by a unitary operator (bijection). The corresponding transform operator is the quantum Fourier transform (QFT) [6] which was detailed in section 2.7.3.

Multiplier

When considering a multiplication operator, we again begin by recalling the multiplication rules of the basis functions. In the Fourier case these simply correspond to

$$T_j^N(x)T_k^N(x) = T_{j+k}^N(x). \tag{7.52}$$

The resulting product basis state reads

$$|xx\rangle = \frac{1}{2^{(N+1)/2}} \sum_{j=0}^{2^{N+1}-1} T_j^N(x). \tag{7.53}$$

The form of $|xx\rangle$ is similar to $|x\rangle$, and the same encoding circuit can be used when extended to $N+1$ qubits by acting on the additional qubit with $P_N(x, N+1)$. From this we find the desired $|gh\rangle$ and the multiplication operator $\check{M}$:

$$|gh\rangle_l = \frac{1}{2^{(N-1)/2}} \sum_{j=\max(0,l-2^N+1)}^{\min(l,2^N-1)} |g\rangle_j |h\rangle_{l-j}, \tag{7.54}$$

$$\check{M} : |g\rangle|h\rangle|0\rangle \to \frac{1}{2^{(N-1)/2}} |a\rangle \sum_{j,k} g_j h_k |j+k\rangle. \tag{7.55}$$

Comparing to the Chebyshev encoding case in Eq. (7.39), we see that the Fourier multiplicator is a simplified version of the Chebyshev multiplicator, as it does not require rescaling and subtraction. The Fourier multiplicator reads

$$\check{M} = \frac{1}{2^{(N-1)/2}} \tilde{r}_+ \check{D} \hat{\mathcal{U}}_{\text{ADD}}. \tag{7.56}$$

With this, the necessary building blocks for implementing training with Fourier encoding have been introduced.

7.3.8 Comparing the Encodings

Comparing Chebyshev encoding versus Fourier encoding, we note their most notable differences. The first is that the Chebyshev encoding state is real for all x (i.e. it leads to states with real amplitudes). Therefore a purely real function can easily be enforced by using a variational ansatz which prepares states with real-only amplitudes (for instance, based on $\hat{R}_Y$ rotations and fixed CZ/CNOT gates). As many problems considered are purely real, this can be a useful restriction to be able to enforce. The Fourier encoding state is naturally complex, and therefore such a simple enforcement for purely real functions is not available. Potentially, a specific variational ansatz could be created for this purpose but it is not currently known.

The next point of comparison is the basis functions themselves. Fourier encoding basis functions are periodic in nature and are particularly suited to harmonic problems. Chebyshev encoding basis functions are real polynomials, and are known to offer best-in-class series expansion in terms of L_∞ norm. For each problem an appropriate basis (one of these two or another entirely) would have to be found. Finally, we note that in general the required operator toolbox for implementing the Fourier encoding is slightly simpler than for the Chebyshev encoding.

With two specific encodings considered, we now reconsider the general case. For any encoding chosen the appropriate operator toolbox must be found. Generally, their implementation will vary greatly based on the encoding. The multiplier in particular will vary. However, we believe that many encodings of interest will have multipliers that are able to be implemented with modifications of the adder and subtractor circuits.

7.4 Results

Now, let us apply the developed strategy to several exemplary differential equations.

7.4.1 Linear Differential Equation Example

First, we consider a linear differential equation of the form

$$\frac{df(x)}{dx} + \kappa \exp(-\kappa x) \cos(\lambda x) + \lambda \exp(-\kappa x) \sin(\lambda x) = 0, \tag{7.57}$$

for some real-valued parameters λ, κ, and the initial value $f(0) = 1$. This differential equation has a known analytic solution $f(x) = \exp(-\kappa x) \cos(\lambda x)$, which represents dynamics of a damped oscillator.

We simulate the proposed algorithm, implementing the steps discussed in the previous sections. We use the *Julia* programming language and the *Yao.jl* package [110] for the full statevector simulation. In particular we use the Chebyshev encoding and represent the function using four qubits. The derivative operator, multiplier and transform follow the Model Embedding section. For building the model we prepare $|f_\theta\rangle\!\rangle = \theta_s \hat{\mathcal{U}}_\theta |o\rangle$, where θ_s is a classical variational parameter representing a scale factor, and $\mathcal{U}_\theta$ is an adjustable circuit (variational ansatz). For the variational ansatz we use alternating layers of $\hat{R}_Y$ rotations and layers of CNOTs. Specifically, seven rotational and six entanglement layers are used, initialised randomly. This ansatz prepares states with real-only amplitudes, leading to functions that are guaranteed real-valued.

The differential equation in the latent space is formed of two terms: the function of an independent variable $g(x) := \exp(-\kappa x)\,(\kappa \cos(\lambda x) + \lambda \sin(\lambda x))$ and the function derivative df/dx. The latent representation of df/dx is $|df/dx\rangle\!\rangle := \theta_s \check{G}^\dagger \hat{\mathcal{U}}_\theta |0\rangle$. The representation of $g(x)$ in latent space is obtained from the use of the transform, as described in the previous section. We assume the non-normalised version of this

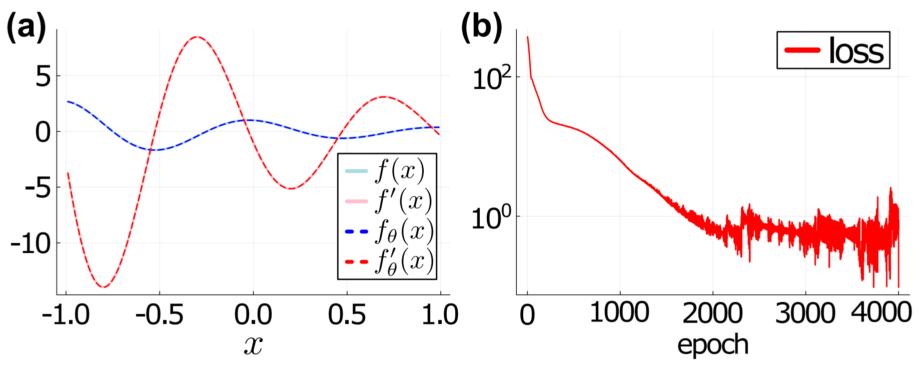


Figure 7.3: **Results of solving a linear differential equation.** Solving equation (7.57), for $\kappa = 1$, $\lambda = 2\pi$, and initial value of $f(0) = 1$. **(a)** Target function and its derivative (solid curves) plotted as a function of x, vs parallelly-learnt function and its derivative evaluated from quantum circuits (dash curves). The functions overlay. **(b)** Value of loss function over epoch (iteration of the Adam optimiser).

wavefunction can be written as a normalised part $|g\rangle$ multiplied with some scalar N_g, to attain the right overall scaling in the latent space representation. Aiming to equate the derivative and the independent function, the differential equation loss reads

$$\mathcal{L}_{\mathrm{DE}}(\theta) = \theta_s^2 \langle 0|\hat{\mathcal{U}}_\theta^\dagger \check{G}\check{G}^\dagger \hat{\mathcal{U}}_\theta|0\rangle + \theta_s N_g \langle g|\check{G}^\dagger \hat{\mathcal{U}}_\theta|0\rangle + \theta_s N_g \langle 0|\hat{\mathcal{U}}_\theta^\dagger \check{G}|g\rangle + N_g^2 \langle g|g\rangle,$$

and the initial value component of the loss is $\mathcal{L}_{\mathrm{init}}(\theta) = (f_\theta(0) - 1)^2$.

The total loss is then taken as $\mathcal{L}(\theta) = (\mathcal{L}_{\mathrm{DE}})^p + \eta \mathcal{L}_{\mathrm{init}}$, where we set the weight factor for the initial value loss $\eta = 10$ to regularise solutions and emphasise the importance of the boundary. This hyperparameter can be chosen heuristically or meta-trained, and furthermore it can be a function of the epoch number [112, 123]. Additionally, we have introduced p as a power for scaling the DE loss, which is chosen as $p = 1/2$. The square root is monotonically increasing over $\mathbb{R}_+$, and does not change the optimum. However, it allows converting the loss from MSE to RMSE (root-mean-square error) loss. This is commonly used in deep learning, being more sensitive to outliers (or initial value pinning as in our case). The loss $\mathcal{L}(\theta)$ is minimised via the Adam optimiser with a small learning rate 0.005 and 4000 epochs. The results are shown in Fig. 7.3. In Fig. 7.3a the dashed curves show functions and derivatives obtained from $f_{\theta^*}(x)$ and $df_{\theta^*}(x)/dx$, evaluated at quasioptimal angles θ^* coming from the optimisation procedure. The corresponding decrease of loss as a function of epoch is shown in Fig. 7.3(b). The loss observed is noisy, this may be able to be reduced with tuning of the optimiser but we found decreasing the learning rate (the first change usually made with noisy training) did not alleviate the behaviour in this situation. We observe that the obtained solutions follow the ground truth for the system (Fig. 7.3(a), solid curves).

7.4.2 Shifted Linear Differential Equation Example

Next, we address an example that requires extending the expressivity of the model, also improving the trainability. Expressivity and trainability of this model mainly depend on choices of the encoding for $|x\rangle$. However, this can be aided by shifting and scaling, such that we use trainable quantum states to reproduce only a nontrivial x dependence. In the Protocol section we introduced the modified models (7.14) and (7.15). Here, we formulate the model $f_\theta^{\text{sh}}(x) = \theta_s \langle x|\hat{\mathcal{U}}_\theta|0\rangle + \theta_{\text{sh}}$ with variationally adjustable scaling and shifting parameters (θ_s and θ_{sh}). This model performs well when the solution value range is significantly smaller than the magnitude of the solution (effectively decreasing the demands for expressivity and boosting trainability).

To showcase the use of shifted models, we simulate solving the differential equation of the form

$$\frac{df(x)}{dx} - f(x) + 15 = 0, \quad f(0) = 16, \tag{7.58}$$

with both types of models (7.14) and (7.15). This problem has an analytic solution $f(x) = \exp(x) + 15$ as a ground truth. We use Chebyshev encoding over four qubits and the real-only ansatz of depth six as used in previous results. With the Chebyshev encoding $g(x) = 1$ is an accessible basis function, therefore the appropriate state representation ψ_1 can be found by inspecting basis functions or using the transform.

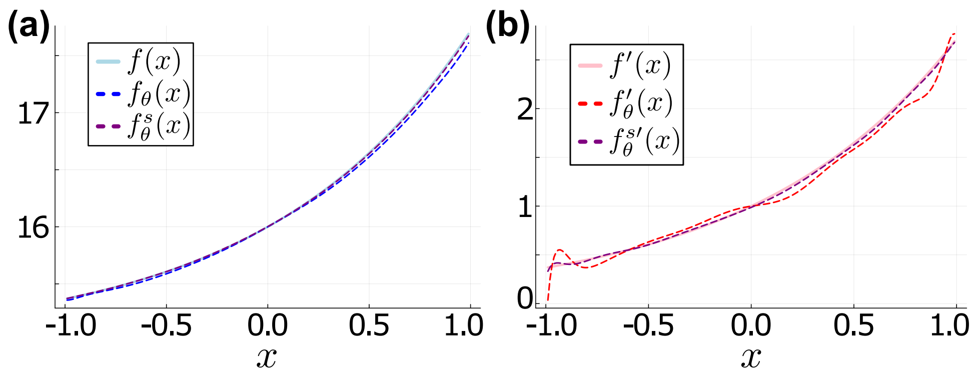


Figure 7.4: **Comparison of solving linear differential equation with shifted and non-shifted model.** Solving equation $df(x)/dx - f(x) + 15 = 0$ with initial value of $f(0) = 16$, approached with two different models [Eqs. (7.14) and (7.15)]. **(a)** Plot of the ground truth function (solid curve) vs trained solution regular [$f_{\theta^*}(x)$] and shifted [$f_{\theta^*}^{\mathrm{sh}}(x)$] models shown by dash curves. **(b)** Derivative of the ground truth (solid curve) vs derivatives of solutions obtained from training the regular and shifted models (dashed curves). The shifted model closely follows the ground truth, while the regular scaled model starts to deviate from the ground truth, which is evident from derivatives.

For this differentiation equation there are three DE terms: df/dx is represented by $|\mathrm{DE}_1\rangle = \theta_s \check{G}^\dagger \hat{\mathcal{U}}_\theta |0\rangle$; f is represented by $|\mathrm{DE}_2\rangle = \theta_s \hat{\mathcal{U}}_\theta |0\rangle$ for the regular scaled model in Eq. (7.14); and $|\mathrm{DE}_2\rangle = \theta_s \hat{\mathcal{U}}_\theta |0\rangle + \theta_{\mathrm{sh}} |\psi_1\rangle$ for the shifted model in Eq. (7.15). The constant function $g(x) = 15$ is represented by $|\mathrm{DE}_3\rangle = 15|\psi_1\rangle$. The differential loss is then formed as in Eq. (7.7). We also include the initial value loss as $\mathcal{L}_{\mathrm{init}}(\theta) = \{f_\theta(0) - 16\}^2$.

We follow the same training workflow as in the previously considered linear case. We rescale and combine loss contributions as $\mathcal{L}(\theta) = (\mathcal{L}_{\mathrm{DE}})^p + \eta \mathcal{L}_{\mathrm{init}}$, with $p = 1/2$ and $\eta = 10$. The loss is minimised with Adam optimiser with the learning rate of 0.005 for both models. Results are shown in Fig. 7.4. We observe that the shifted model performs better (purple dashed curves in Fig. 7.4a,b for function and derivative). We achieve a better performance with shifting, and require fewer epochs roughly reduced by an order of magnitude, signifying the improved trainability.

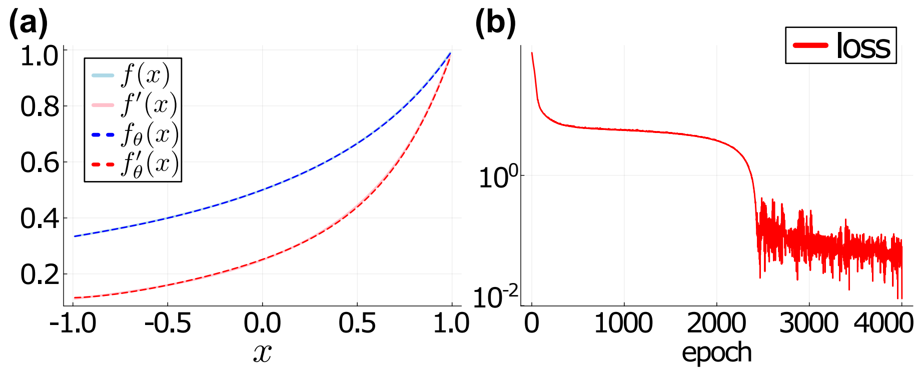


Figure 7.5: **Results of solving nonlinear differential equation.** DE considered $df/dx - f^2 = 0$ with initial value $f(0) = 0.5$. **(a)** Plot of target function and derivative (solid lines) versus resulting function and derivative (dash lines). **(b)** Value of loss function over epoch iteration.

7.4.3 Nonlinear Differential Equation Example

We now move on to consider an example of a nonlinear differential equation, chosen as

$$\frac{df(x)}{dx} + f^2(x) = 0, \quad f(0) = 1/2. \tag{7.59}$$

This DE has an analytic solution $f(x) = 1/(2-x)$, set as the ground truth. Notably, even though Eq. (7.59) looks simple, we should appreciate that solving nonlinear differential equations with amplitude encoding on quantum computing is far away from being easy. In our approach we are able to tackle it with latent space basis multipliers.

We solve the problem following the workflow used in the linear case, while changing the variable state representation and using multiplier circuits $\check{M}$. We employ the Chebyshev encoding on three qubits and the same depth-six variational ansatz. The two terms of this DE, df/dx and f^2, are then mapped into the latent space with $|xx\rangle$ variable encoding. The first term is prepared by multiplying $g(x) = 1$ and df/dx for $|\mathrm{DE}_1\rangle\rangle = \theta_s \check{M}(|g\rangle \otimes \check{G}^\dagger \hat{\mathcal{U}}_\theta|0\rangle)$. As unity is a naturally occurring basis function of the Chebyshev encoding, the relevant $|g\rangle$ can easily be found as $2^{N/2}[1, 0, 0, ...]$. The second term f^2 is represented by $|\mathrm{DE}_2\rangle\rangle = \theta_s^2 \check{M}(\hat{\mathcal{U}}_\theta|0\rangle \otimes \hat{\mathcal{U}}_\theta|0\rangle)$. These states are used to form a differential equation loss in Eq. (7.7), supplemented by initial value loss as $\mathcal{L}_{\rm init}(\theta) = \{f_\theta(0) - 1/2\}^2$. The total loss is the same as for previous problems, $\mathcal{L}(\theta) = (\mathcal{L}_{\rm DE})^p + \eta\mathcal{L}_{\rm init}$, $p = 1/2$, $\eta = 10$. The loss is minimised with Adam optimiser with the learning rate of 0.005. The resulting solutions are shown in Fig. 7.5a, where the trained solution of the nonlinear DE (blue and red dashed curves for the function and its derivative, respectively). This follows nicely the ground truth (solid curves in Fig. 7.5a). The corresponding training is presented in Fig. 7.5b, where we observe that the optimiser explores the landscape of solutions, and ultimately finds the suitable candidate when the loss drops below unity.

7.4.4 Multidimensional Differential Equation Example

Finally, we test the solver for tackling multidimensional problems. We consider a differential equation for a function $f(x,y)$ of two independent variables x and y, written as

$$\frac{df(x,y)}{dy} - 2y - x = 0, \quad f(x,0) = 1. \tag{7.60}$$

This DE has an analytic solution $f(x,y) = y^2 + xy + 1$ as the ground truth.

We encode the two independent variables in parallel registers using the model

$$f_\theta(x,y) = \theta_s(\langle x| \otimes \langle y|)|f_\theta\rangle. \tag{7.61}$$

We choose to encode both $|x\rangle$ and $|y\rangle$ with Chebyshev encoding of two qubits. $|f_\theta\rangle$ is prepared with the same real-only ansatz as in previous examples, applied over four qubits in total, with an ansatz depth of six.

The two terms of the differential equation are df/dy and $g(x,y) = -2y - x$. For the derivative term, as variables are encoded in separate registers, the derivative operator $\check{G}^\dagger$ is applied to the register of the variable being differentiated. The remaining register is left untouched. Therefore, the corresponding state is

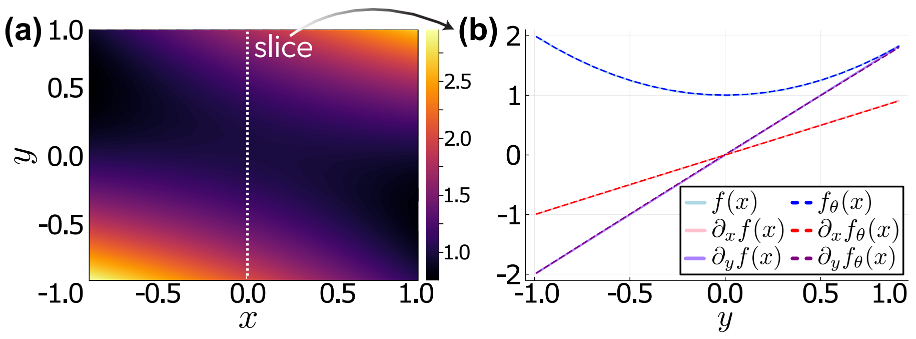


Figure 7.6: **Results of solving a multidimensional differential equation.** Problem considered is $df/dy - 2y - x = 0$ with the boundary condition $f(x, 0) = 1$. **(a)** Result from parallel multidimensional training shown as a density plot of $f_\theta(x, y)$ vs x and y. **(b)** Slice of resulting function and derivatives at $x = 0$. Analytic solutions shown in solid lines. Functions resulting from training shown in dashed lines.

$|\mathrm{DE}_1\rangle\!\rangle = \left(I \otimes \check{G}^\dagger\right)\hat{\mathcal{U}}_\theta|0\rangle$. For the second term, the latent space state is prepared by applying $\hat{\mathcal{U}}^\dagger_{\mathrm{QChT}} \otimes \hat{\mathcal{U}}^\dagger_{\mathrm{QChT}}$ to the state with amplitudes of $g(x, y)$ evaluated at the Chebyshev nodes for $|\mathrm{DE}_2\rangle\!\rangle$. Next, we need to sum the differential term and the boundary condition term, $\mathcal{L}_{\mathrm{BC}}(\theta) = \sum_j \{f_\theta(x_j, 0) - x_j\}^2$, where $\{x_j\}$ are a set of boundary evaluation points. We choose 21 uniformly spaced points for $x \in (-1, 1)$. Again, the total loss is $\mathcal{L}(\theta) = (\mathcal{L}_{\mathrm{DE}})^p + \eta\mathcal{L}_{\mathrm{BC}}$, $p = 1/2$, $\eta = 10$. Minimising this loss via Adam optimiser with the learning rate of 0.005 we get results plotted in Fig. 7.6. The full trained solution is shown as a density plot in Fig. 7.6a, closely following the ground truth (average deviation is in the range of 10^{-3}). We also take a slice at $x = 0$ and plot the solution as a function of y in Fig. 7.6b. We recover the expected behaviour from the multidimensional training.

7.4.5 Fourier Encoding

So far our presented results have made use of the Chebyshev encoding but we have discussed another encoding, the Fourier encoding. This encoding naturally represents periodic solutions. Furthermore, the encoding is naturally complex and therefore we cannot easily restrict the trial function to real only values. We remember that we cannot classical post process the trial solution for a real only result as that would break the properties allowing for the removal of x whilst training and how we compute multiplication. Therefore, to show this alternative

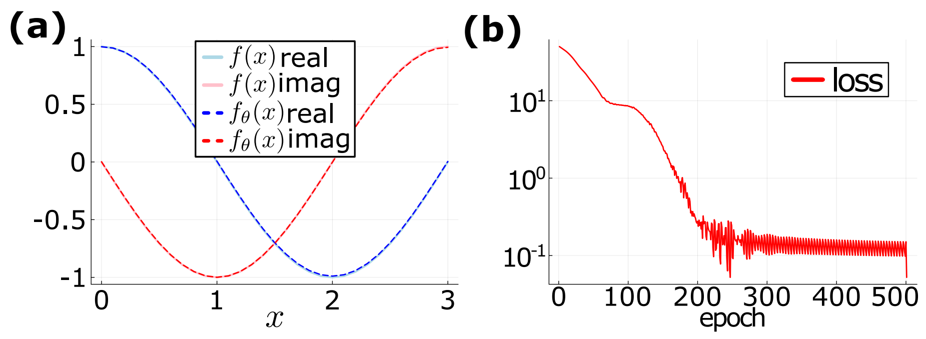


Figure 7.7: **Results of solving a nonlinear differential equation with Fourier encoding.** Problem considered is $f\ df/dx + \pi i/2\exp(-ix\pi) = 0$ with an initial value of $f(0) = 1$. **(a)** Plot of real and imaginary parts of target function (solid lines) compared to real and imaginary part of resulting function (dash lines). **(b)** Value of loss function over epoch iteration.

encoding, and to emphasise the importance of choosing an appropriate encoding for a problem we show results of using the Fourier encoding for one problem it is suited for and another it is not.

For our first problem we consider the nonlinear first order differential equation and initial value

$$f\frac{df}{dx} + \frac{\pi i}{2}\exp^{-ix\pi} = 0, \quad f(0) = 1. \tag{7.62}$$

This differential equation has analytic solution $f(x) = \exp(-ix\pi/2)$ and is periodic and complex. Therefore it is expected that the Fourier encoding would be very well suited to the problem.

To solve it we simulate our algorithm in the same manner as previous results. We utilise the Fourier encoding with two qubits. The variational ansatz used is a hardware efficient ansatz of depth six. An adam optimiser of learn rate 0.01 is used.

For this problem there are two terms in the differential equation, $f\ df/dx$ and $g(x) = i\frac{\pi}{2}\exp^{-ix\pi}$. The first term $f\ df/dx$ is represented by $|DE_1\rangle\!\rangle = \theta_s^2\check{M}(\hat{\mathcal{U}}_\theta|0\rangle \otimes$

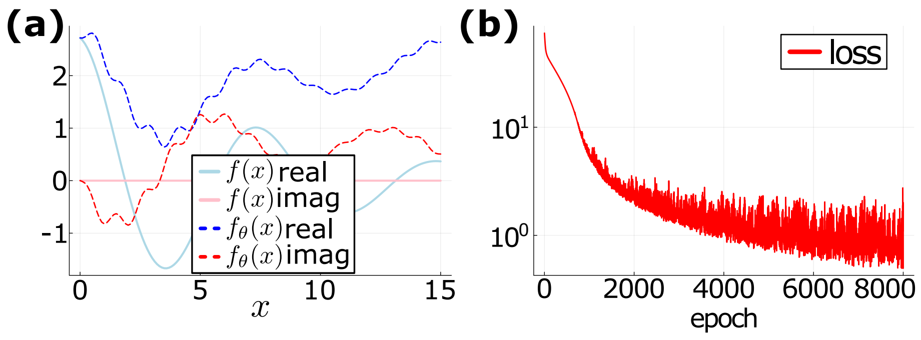


Figure 7.8: **Results of solving a real, linear differential equation with Fourier encoding.** Solving equation (7.57), for $\kappa = 1$, $\lambda = 2\pi$, and initial value of $f(7.5) = 1$. **(a)** Plot of real and imaginary parts of target function (solid lines) compared to real and imaginary part of resulting function (dash lines). **(b)** Value of loss function over epoch iteration.

$\check{G}^\dagger \hat{\mathcal{U}}_\theta |0\rangle$. For the second term, we find that $g(x)$ is naturally expressed in the extended multiplication basis and can be found as $|DE_2\rangle\!\rangle = -i\pi\sqrt{2}[0, 0, 1, 0...0]$. These states are used to form a differential equation loss in Eq. (7.7), supplemented by initial value loss as $\mathcal{L}_{\text{init}}(\theta) = (f_\theta(0) - 1)^2$. The total loss is similar as for previous problems, $\mathcal{L}(\theta) = (\mathcal{L}_{\text{DE}})^p + \eta\mathcal{L}_{\text{init}}$, $p = 1/2$, $\eta = 20$. The results of this are shown in Fig. 7.7. It can be seen a good fit is achieved within a relatively small number of epochs - magnitude of 100.

The second problem we consider is the same as in the linear differential equation for Chebyshev encoding (7.57), rescaled such that the range of the Fourier nodes $[0, 2^N - 1]$ is mapped onto the domain $[-1, 1]$, the range of the Chebyshev nodes. A linear mapping $f(x) = 2x/(2^N - 1) - 1$ is used for this. This function is real only and non-periodic, which would lead to the assumption that naive Fourier basis fitting would not be a optimal choice.

For this problem we used a Fourier encoding over four qubits and a hardware efficient ansatz of depth ten. The terms of the differential equation are prepared the same as the Chebyshev case though with the Fourier encoding equivalents of each circuit. The total loss is also constructed the same and optimised with adam optimiser with learn rate 0.005.

The results of this are shown in Fig. 7.8. As can be seen, a good fit was not found. In particular, it struggles to represent a real function. With a larger number of qubits to contribute more basis functions a better fit likely could be found. But we note that the Chebyshev encoding solution found a good fit with this number of qubits (four) as shown in Fig. 7.3. This highlights the importance of choosing a suitable encoding for a given problem.

7.5 Discussion

Within this chapter we develop a quantum algorithm for solving differential equations without costly grid evaluations. In particular we show that by representing all terms of the DE in terms of an encoding which fulfill a requirement on linear independence the encoding can then effectively be removed from consideration whilst training. We detail how to achieve this with respect to Fourier encoding and Chebyshev encoding (as introduced in chapter 6). The algorithm is most suitable for those with large numbers of points required such as multidimensional problems because this is when the lack of need to individually evaluate at each point will be most advantageous.

This process can be used with any suitable encoding choice provided that the relevant term preparation gates are implementable – i.e. multiplication and derivative. An avenue of future work is the development of additional encoding choices. Additionally we have considered a variational approach and therefore the open research questions of how best to ensure trainable and expressive optimisation is of interest.

On that note we highlight that whilst throughout this chapter the focus has been on a variational approach, the way the terms are encoded can lead to a linear systems approach. This is briefly introduced but not in detail – developing this approach is for future research. Another important point is to highlight that though a hybrid variational algorithm the circuits that require implementation during training are of sufficient complexity that useful NISQ implementation is unlikely. We also don’t believe full fault tolerance would be required, instead ”intermediate scale” quantum devices between NISQ and FTQC will likely be needed.

Chapter 8

Conclusion

8.1 Summary

Throughout the preparation of this thesis I have researched and developed quantum machine learning algorithms for the solving of differential equations along with enabling tools, specifically concerning various transforms and encodings — useful subroutines for building quantum models. By considering a variety of approaches for the solving of differential equations, I have developed a range of algorithms which have different advantages and limitations. These have been applied to different problems, yielding different quantum resource requirements and comparison between algorithms. I have shown that quantum machine learning is a viable approach for introducing differential constraints and searching for solutions of differential equations. Here, there are multiple avenues for achieving a quantum advantage in the future, including increased expressivity of quantum models, parallelism over grid evaluations, utilisation of linear system solving with improved scaling, and performing efficient sampling.

The first algorithm developed, differentiable quantum circuits, as introduced in chapter 3 variationally trains a quantum model to solve a differential equation. As a result it is very generalisable and suitable for wide range of DE problems due to the ease of adapting the quantum model and therefore altering expressivity and trainability. The quantum model can be chosen such that each individual quantum circuit evaluation is easy to implement resulting in a near-term friendly algorithm. However, this algorithm fundamentally involves non-convex optimisation requiring quantum circuit evaluations for every training point at every iteration, potentially resulting in a large number of required quantum circuit evaluations.

In chapter 4 I further developed the previous algorithm whilst considering quantile mechanics to consider solving SDEs. A quantum model is used to represent a quantile function which can be trained and evolved in time to solve an SDE. The resulting model can be sampled to generate data. As a development of the previous algorithm it shares many properties.

The third algorithm introduced in chapter 5 utilises quantum kernel methods to solve nonlinear PDEs. Here, the quantum model is considered in terms of quantum kernel functions. The problem can then be solved as a general optimisation procedure (MMR) which is not specific to kernel methods or via SVR which is specific to kernel methods. When considered as a SVR problem, the problem is then written as a system of equations and can either be solved variationally or any other appropriate algorithm. One useful property of kernel methods is that when the differential equation considered is linear the resulting problem to solve is convex. Additionally, due to the form of the quantum model, quantum circuit evaluations are only required pre and post training. Therefore, whilst the quantum model can be considered more complex (overlap measurement required) many fewer evaluations are required and can still be considered near-term friendly.

Next, the quantum Chebyshev transform is developed in chapter 6. The quantum Chebyshev transform converts between Chebyshev space and computational space. In different spaces certain behaviours/features can be more or less easy to compute or influence — for example Chebyshev space is suitable for function fitting and computational space for sampling. By transforming between spaces the appropriate space for the current task can be used, improving efficiency. This work is utilised in chapter 7. We see that transforms are an important tool and further development of transforms for spaces appropriate for other tasks is open for future work.

The final algorithm was developed in chapter 7. This algorithm constructs a state/sum of states to represent each term of the differential equation in a latent space. By developing multiplication operators in latent space nonlinear DEs can be solved. Chebyshev space as developed in chapter 6 is considered along with Fourier space but any suitable space could be used. This method can result in either a system of equations or an optimisation problem — I have focused on the variational approach. With this method, x-encoding can be removed from consideration during training resulting in no grid evaluations required during training (though quantum circuit evaluations do still occur). Therefore, this algorithm is suitable for problems which result in large number of grid points such as multidimensional problems. However, the circuits implemented are more complex and are unlikely suitable for near-term use, and more likely to require quantum devices from future generations.

Comparison

These developed algorithms are all for the purpose of solving differential equations and therefore there are similarities between them. There are also differences based on the approach for solving DEs, the trial functions and the loss functions.

One comparison is the suitability for near-term implementation. All of the proposed algorithms have many hyperparameters affecting the quantum circuits to be implemented. To consider which algorithms are near-term suitable we consider what quantum circuits need to be evaluated and how suitable they are. For DQC as presented in chapter 3 we find that a feature map and variational ansatz are required with no limit on their form and then a expectation value must be taken. Therefore, by choosing an appropriate feature map, ansatz and measurement basis for the device this algorithm is found to be the most near-term appropriate. The QQM approach in chapter 4 is similar in many respects to DQC but there is a larger focus on multidimensional problems to allow time evolution. Therefore, whilst still near-term possible this included extra complexity puts a few more requirements onto the quantum device as compared to DQC. For the kernel method approaches, what is needed is a feature map and overlap measurement. As overlaps are more complicated to measure, similar to QQM, these approaches are hopefully near-term implementable but with some more complications than the DQC approach. The final developed algorithm has much more stringent requirements on the quantum device. Overlaps are again required and at much greater number than the kernel approaches. Additionally, nonlinearity is processed in a quantum manner which results in multiplier circuits which are deep and projective measurements are required. Therefore this approach, whilst not requiring fault tolerance, is not near-term suitable.

To finish the discussion on near-term implementation it is worth noting that even near-term appropriate does not mean that useful quantum computation is immediately imminent. At first the near term algorithms will be implementable for small scale problems - such as the linear DE toy problems I consider in result sections or even simpler. This will be an important step for the goal of quantum computation but these problems remain easy to solve classically. Then it will be the task to further refine the algorithms and the devices and their interactions together to scale up the problems considered until useful problems are considered. Whilst this is still hoped to be near-term it is important to emphasise that this is not immediate.

Another comparison is the flexibility of the algorithm and what range of problems they are suitable for and what problems it performs best for. DQC is a very flexible approach with it easily able to be adapted to a wide range of problems. For example QQM can be viewed as an adaption of DQC for SDEs. Consequently, DQC itself does not have a particular strong point of application but once adapted it does. The kernel method MMR approach is similarly flexible. For

SVR it depends mostly on the formulation of the problem into the suitable form. Consequently, it strongly favours problems with low degrees of nonlinearity. The algorithm introduced in chapter 7s strong point is not requiring separate grid evaluations during training. Therefore, it performs best for problems with large implicit grid sizes such as multidimensional problems. Additionally, the multiplier circuits for the encodings considered are complex giving a preference for low orders of nonlinearity. However, I do note that all these algorithms are able to solve simpler problems such as the one dimensional damped oscillator in chapters 3, 5 and 7. At such problem scales the differences between the algorithm more relates to near term suitability than performance. Differences in performance would become more apparent at larger problem scales where algorithm performance would likely favour the types of problems just discussed.

A final point to consider is what are the quantum benefits of the proposed algorithms. I first of all acknowledge a lack of any algorithmic proof of advantage, or (more positively) proof of lack of advantage. This is due to the variational nature of the algorithms I have developed. Because convergence is not guaranteed, and even when convergence occurs there are no promises on number of iterations, it is hard to find the complexity of such algorithms in a concrete enough way to declare advantage has been found. Instead we look towards the more nebulous and where we hope to find quantum benefits. This would then be tested heuristically once algorithms are implementable in comparison to other algorithms. So where do we hope to find quantum benefits? A general answer for these type of algorithms is expressivity - utilising quantum computing gives an exponential space with respect to the number of qubits and therefore, depending on trial function, a huge range of functions. However, during my studies it has been shown and discussed that we cannot rely just on expressivity as it can introduce training issues. Therefore we look at more specific avenues for quantum advantage. In the chapter 7 I use quantum parallelism and superposition to avoid the need to evaluate grid points separately. This would then hopefully show advantage for problems with large number of point such as multidimensional problems.

8.2 Aims & Objectives

Looking back at my aims and objectives I consider whether I have succeeded in fulfilling them. The overall goal of developing quantum DE solvers I have achieved with the development of four algorithms in chapters 3, 4, 5 and 7. Additionally in chapter 6 a tool utilised in chapter 7 was developed. All of these developed algorithms utilise variational and/or QML methods. Additionally, the algorithms developed in chapters 3 and 5 are near-term appropriate with suitable hyper-parameter choices such as feature map/kernel function choices.

All proposed algorithms are applicable for nonlinear DE problems. In chapter 3 and 4 nonlinearity is achieved by classical post processing within the loss function. For chapter 5 we have two approaches, one (MMR) also uses classical post processing within loss function whilst the other (SVR) considers it within the classical preparation of the problem and leads to nonlinear system of equations to solve. In Chapter 7 I develop a new way to treat nonlinearity in a quantum manner. By considering basis functions with multiplication rules such as Chebyshev and Fourier, we can encode the multiplication rule in an operator which allows us to implement multiplication and therefore nonlinearity. Thus I conclude I have reached my aims and objectives.

8.3 Outlook

My primary focus has centered on variational quantum machine learning algorithms, which share certain characteristics. These algorithms, for instance, have the potential to generate near-term solutions by choosing appropriate quantum models. Moreover, they exhibit versatility and flexibility, allowing for extensive modifications to the quantum model and optimisation methods, making them adaptable to a wide range of problems. However, this adaptability poses a challenge — the choices made in these algorithms are often heuristic and can significantly impact performance. Consequently, there is ongoing research dedicated to finding more systematic approaches for selecting suitable hyperparameters for specific problems.

Another overarching question for these types of algorithms revolves around the realisation of quantum advantage. While there is intuition regarding potential advantages at a larger scale, the non-convex nature of these variational problems offers no guarantees. Related are concerns about training guarantees and convergence, especially in the context of issues like barren plateaus and local minima in the loss landscape, which persist and drive active research efforts, as these problems impact nearly all variational scenarios. Current efforts include a focus on designing models and choosing hyperparameters to avoid these negative features, with approaches such as overparametrisation, ansatz designs and initialisation proposed, as well as deepening understanding of what causes these behaviour so as to help avoid these situations. Also, to remember, quantum machine learning algorithms are not required to be variational. Non-variational algorithms would naturally avoid variational concerns, though perhaps at the price of near term suitability and quantum resource requirements. In particular some frameworks can be approached with both variational and non-variational approaches, e.g. as mentioned for the algorithms developed in chapters 5 and 7.

When considering the future of quantum machine learning for differential equations there are still many avenues for development and unanswered questions. Many fall under developing and identifying appropriate quantum models for a problem as well as efficient training regimes. Of particular importance is identifying where advantage could be realised and what is still needed to access this. I highlight large, multidimensional problems as a promising area due to classical inefficiency from the many grid points generated, known as the “curse of dimensionality”. By exploiting quantum properties such as superposition and parallelism of quantum computation, separate grid point consideration could be avoided. Additionally, if a continuous encoding is used, often the data input and output problem can be avoided, the trained model can be evaluated continuously, and exact derivative methods such as parameter shift rule can be utilised. We started on this journey in chapter 7 but further development of this approaches scaling and how best to encode multiple dimensions for this framework is open. The development and consideration of alternative approaches for the same goal is also open.

Another important aspect to consider is approaches for considering nonlinear differential equations. Many differential equations of interest are nonlinear but non-linearity is not trivial to implement due to the natural linearity of quantum mechanics, often leading to inefficiency. Approaches considered within this thesis were classical post processing within loss function, and function multiplication operator (specific to choice of basis). Other approaches include quantum non-linear processing units as in [1]. Further development of these approaches as well as new approaches is an open area of research. One particular question left open from this thesis is whether there is a basis suitable for function fitting with a more efficient associated function multiplier operator. By improving the efficiency of representing non-linear differential equations the variety of DE problems with a promise for future advantaged is widened.

In summary, throughout my studies, I have developed algorithms designed to tackle challenges in scientific computing, specifically concerning solving differential equations using quantum computing. These algorithms underscore the suitability of the quantum machine learning paradigm for solving such equations, hinting at the potential for future quantum advantage. There are ample opportunities for refining, adapting, and utilising these algorithms. They can be used as building blocks to progress towards useful quantum computation. I believe the field of quantum scientific machine learning for solving differential equations will continue to evolve, contributing to the long-term goal of realising tangible quantum advantage.

Bibliography

[1] Michael Lubasch et al. “Variational quantum algorithms for nonlinear problems”. In: *Physical Review A* 101.1 (2020), p. 010301.

[2] Edward Farhi et al. “Quantum computation by adiabatic evolution”. In: *arXiv preprint quant-ph/0001106* (2000).

[3] Dorit Aharonov et al. “Adiabatic quantum computation is equivalent to standard quantum computation”. In: *SIAM review* 50.4 (2008), pp. 755–787.

[4] Robert Raussendorf, Daniel E Browne, and Hans J Briegel. “Measurement-based quantum computation on cluster states”. In: *Physical review A* 68.2 (2003), p. 022312.

[5] Hans J Briegel et al. “Measurement-based quantum computation”. In: *Nature Physics* 5.1 (2009), pp. 19–26.

[6] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[7] Paul Adrien Maurice Dirac. “A new notation for quantum mechanics”. In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 35. 3. Cambridge University Press. 1939, pp. 416–418.

[8] Göran Wendin. “Quantum information processing with superconducting circuits: a review”. In: *Reports on Progress in Physics* 80.10 (2017), p. 106001.

[9] Colin D Bruzewicz et al. “Trapped-ion quantum computing: Progress and challenges”. In: *Applied Physics Reviews* 6.2 (2019).

[10] Loïc Henriet et al. “Quantum computing with neutral atoms”. In: *Quantum* 4 (2020), p. 327.

[11] Peter W Shor. “Fault-tolerant quantum computation”. In: *Proceedings of 37th conference on foundations of computer science*. IEEE. 1996, pp. 56–65.

[12] John Preskill. “Quantum computing in the NISQ era and beyond”. In: *Quantum* 2 (2018), p. 79.

[13] Yudong Cao et al. “Quantum chemistry in the age of quantum computing”. In: *Chemical reviews* 119.19 (2019), pp. 10856–10915.

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

[15] Marco Cerezo et al. “Variational quantum algorithms”. In: *Nature Reviews Physics* 3.9 (2021), pp. 625–644.

[16] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2020.

[17] Sotiris B Kotsiantis, Ioannis Zaharakis, P Pintelas, et al. “Supervised machine learning: A review of classification techniques”. In: *Emerging artificial intelligence applications in computer engineering* 160.1 (2007), pp. 3–24.

[18] GM Harshvardhan et al. “A comprehensive survey and analysis of generative models in machine learning”. In: *Computer Science Review* 38 (2020), p. 100285.

[19] Dastan Maulud and Adnan M Abdulazeez. “A review on linear regression comprehensive in machine learning”. In: *Journal of Applied Science and Technology Trends* 1.4 (2020), pp. 140–147.

[20] Humza Naveed et al. “A comprehensive overview of large language models”. In: *arXiv preprint arXiv:2307.06435* (2023).

[21] *Introduction to Machine Learning for Beginners*. URL: https://towardsdatascience.com/introduction-to-machine-learning-for-beginners-eed6024fdb08 (visited on 02/12/2023).

[22] *What Is a Machine Learning Model?* URL: https://www.mathworks.com/discovery/machine-learning-models.html (visited on 02/13/2024).

[23] Rémi Bardenet et al. “Collaborative hyperparameter tuning”. In: *International conference on machine learning*. PMLR. 2013, pp. 199–207.

[24] Li Yang and Abdallah Shami. “On hyperparameter optimization of machine learning algorithms: Theory and practice”. In: *Neurocomputing* 415 (2020), pp. 295–316.

[25] Amy Rechkemmer and Ming Yin. “When confidence meets accuracy: Exploring the effects of multiple performance indicators on trust in machine learning models”. In: *Proceedings of the 2022 chi conference on human factors in computing systems*. 2022, pp. 1–14.

[26] Ming Yin, Jennifer Wortman Vaughan, and Hanna Wallach. “Understanding the effect of accuracy on trust in machine learning models”. In: *Proceedings of the 2019 chi conference on human factors in computing systems*. 2019, pp. 1–12.

[27] Carsten Blank et al. “Quantum classifier with tailored quantum kernel”. In: *npj Quantum Information* 6.1 (2020), p. 41.

[28] Kosuke Mitarai et al. “Quantum circuit learning”. In: *Physical Review A* 98.3 (2018), p. 032309.

[29] Jonathan Romero and Alán Aspuru-Guzik. “Variational quantum generators: Generative adversarial quantum machine learning for continuous distributions”. In: *Advanced Quantum Technologies* 4.1 (2021), p. 2000003.

[30] Abhinav Kandala et al. “Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets”. In: *nature* 549.7671 (2017), pp. 242–246.

[31] Kouhei Nakaji and Naoki Yamamoto. “Expressibility of the alternating layered ansatz for quantum computation”. In: *Quantum* 5 (2021), p. 434.

[32] Scott E Smart and David A Mazziotti. “Efficient two-electron ansatz for benchmarking quantum chemistry on a quantum computer”. In: *Physical Review Research* 2.2 (2020), p. 023048.

[33] Johannes Jakob Meyer et al. “Exploiting symmetry in variational quantum machine learning”. In: *PRX Quantum* 4.1 (2023), p. 010328.

[34] Eric R Anschuetz et al. “Efficient classical algorithms for simulating symmetric quantum systems”. In: *Quantum* 7 (2023), p. 1189.

[35] Marco Cerezo et al. “Cost function dependent barren plateaus in shallow parametrized quantum circuits”. In: *Nature communications* 12.1 (2021), p. 1791.

[36] Adrián Pérez-Salinas et al. “Data re-uploading for a universal quantum classifier”. In: *Quantum* 4 (2020), p. 226.

[37] Gordon D Smith. *Numerical solution of partial differential equations: finite difference methods*. Oxford university press, 1985.

[38] *The stochastic parameter-shift rule*. URL: `https://pennylane.ai/qml/demos/tutorial_stochastic_parameter_shift` (visited on 02/19/2023).

[39] Maria Schuld et al. “Evaluating analytic gradients on quantum hardware”. In: *Physical Review A* 99.3 (2019), p. 032331.

[40] Oleksandr Kyriienko and Vincent E Elfving. “Generalized quantum circuit differentiation rules”. In: *Physical Review A* 104.5 (2021), p. 052417.

[41] David Wierichs et al. “General parameter-shift rules for quantum gradients”. In: *Quantum* 6 (2022), p. 677.

[42] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. “Kernel methods in machine learning”. In: (2008).

[43] Maria Schuld. “Supervised quantum machine learning models are kernel methods”. In: *arXiv preprint arXiv:2101.11020* (2021).

[44] Norihito Shirai et al. “Quantum tangent kernel”. In: *arXiv preprint arXiv:2111.02951* (2021).

[45] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. “Effect of data encoding on the expressive power of variational quantum-machine-learning models”. In: *Physical Review A* 103.3 (2021), p. 032430.

[46] Michael Ragone et al. “A unified theory of barren plateaus for deep parametrized quantum circuits”. In: *arXiv preprint arXiv:2309.09342* (2023).

[47] Xuchen You and Xiaodi Wu. “Exponentially many local minima in quantum neural networks”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 12144–12155.

[48] M Cerezo et al. “Does provable absence of barren plateaus imply classical simulability? Or, why we need to rethink variational quantum computing”. In: *arXiv preprint arXiv:2312.09121* (2023).

[49] Eric R Anschuetz and Bobak T Kiani. “Quantum variational algorithms are swamped with traps”. In: *Nature Communications* 13.1 (2022), p. 7760.

[50] Aleta Berk Finnila et al. “Quantum annealing: A new method for minimizing multidimensional functions”. In: *Chemical physics letters* 219.5-6 (1994), pp. 343–348.

[51] Samson Wang et al. “Noise-induced barren plateaus in variational quantum algorithms”. In: *Nature communications* 12.1 (2021), p. 6961.

[52] Jarrod R McClean et al. “Barren plateaus in quantum neural network training landscapes”. In: *Nature communications* 9.1 (2018), p. 4812.

[53] Andrew Arrasmith et al. “Effect of barren plateaus on gradient-free optimization”. In: *Quantum* 5 (2021), p. 558.

[54] Martin Larocca et al. “Diagnosing barren plateaus with tools from quantum optimal control”. In: *Quantum* 6 (2022), p. 824.

[55] Zoë Holmes et al. “Connecting ansatz expressibility to gradient magnitudes and barren plateaus”. In: *PRX Quantum* 3.1 (2022), p. 010313.

[56] Jindong Wang et al. “Generalizing to unseen domains: A survey on domain generalization”. In: *IEEE transactions on knowledge and data engineering* 35.8 (2022), pp. 8052–8072.

[57] Xue Ying. “An overview of overfitting and its solutions”. In: *Journal of physics: Conference series*. Vol. 1168. IOP Publishing. 2019, p. 022022.

[58] Martin Larocca et al. “Theory of overparametrization in quantum neural networks”. In: *Nature Computational Science* 3.6 (2023), pp. 542–551.

[59] Carlos Ortiz Marrero, Mária Kieferová, and Nathan Wiebe. “Entanglement-induced barren plateaus”. In: *PRX Quantum* 2.4 (2021), p. 040316.

[60] Arthur Pesah et al. “Absence of barren plateaus in quantum convolutional neural networks”. In: *Physical Review X* 11.4 (2021), p. 041011.

[61] Taylor L Patti et al. “Entanglement devised barren plateau mitigation”. In: *Physical Review Research* 3.3 (2021), p. 033090.

[62] Stefan H Sack et al. “Avoiding barren plateaus using classical shadows”. In: *PRX Quantum* 3.2 (2022), p. 020365.

[63] Yingjie Tian and Yuqi Zhang. “A comprehensive survey on regularization strategies in machine learning”. In: *Information Fusion* 80 (2022), pp. 146–166.

[64] Evan Peters and Maria Schuld. “Generalization despite overfitting in quantum machine learning models”. In: *Quantum* 7 (2023), p. 1210.

[65] Martin Plesch and Časlav Brukner. “Quantum-state preparation with universal gate decompositions”. In: *Physical Review A* 83.3 (2011), p. 032302.

[66] K Vogel and H Risken. “Determination of quasiprobability distributions in terms of probability distributions for the rotated quadrature phase”. In: *Physical Review A* 40.5 (1989), p. 2847.

[67] Jules Tilly et al. “The variational quantum eigensolver: a review of methods and best practices”. In: *Physics Reports* 986 (2022), pp. 1–128.

[68] Ken M Nakanishi, Kosuke Mitarai, and Keisuke Fujii. “Subspace-search variational quantum eigensolver for excited states”. In: *Physical Review Research* 1.3 (2019), p. 033062.

[69] Ian Goodfellow et al. “Generative adversarial networks”. In: *Communications of the ACM* 63.11 (2020), pp. 139–144.

[70] Kosuke Mitarai and Keisuke Fujii. “Methodology for replacing indirect measurements with direct measurements”. In: *Physical Review Research* 1.1 (2019), p. 013006.

[71] Andrew M Childs and Nathan Wiebe. “Hamiltonian simulation using linear combinations of unitary operations”. In: *arXiv preprint arXiv:1202.5822* (2012).

[72] Gilles Brassard et al. “Quantum amplitude amplification and estimation”. In: *Contemporary Mathematics* 305 (2002), pp. 53–74.

[73] Pierre Duhamel and Martin Vetterli. “Fast Fourier transforms: a tutorial review and a state of the art”. In: *Signal processing* 19.4 (1990), pp. 259–299.

[74] Francisco Orts et al. “A review on reversible quantum adders”. In: *Journal of Network and Computer Applications* 170 (2020), p. 102810.

[75] Lidia Ruiz-Perez and Juan Carlos Garcia-Escartin. “Quantum arithmetic with the quantum Fourier transform”. In: *Quantum Information Processing* 16 (2017), pp. 1–14.

[76] Chris Rackauckas et al. “Diffeqflux. jl-A julia library for neural differential equations”. In: *arXiv preprint arXiv:1902.02376* (2019).

[77] Christopher Rackauckas et al. “Universal differential equations for scientific machine learning”. In: *arXiv preprint arXiv:2001.04385* (2020).

[78] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: *Journal of Computational physics* 378 (2019), pp. 686–707.

[79] Shengze Cai et al. “Physics-informed neural networks (PINNs) for fluid mechanics: A review”. In: *Acta Mechanica Sinica* 37.12 (2021), pp. 1727–1738.

[80] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. “Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations”. In: *Science* 367.6481 (2020), pp. 1026–1030.

[81] Salvatore Cuomo et al. “Scientific machine learning through physics–informed neural networks: Where we are and what’s next”. In: *Journal of Scientific Computing* 92.3 (2022), p. 88.

[82] Stefano Markidis. “The old and the new: Can physics-informed deep-learning replace traditional linear solvers?” In: *Frontiers in big Data* 4 (2021), p. 669097.

[83] 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.

[84] James Mercer. “Xvi. functions of positive and negative type, and their connection the theory of integral equations”. In: *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character* 209.441-458 (1909), pp. 415–446.

[85] Lipo Wang. *Support vector machines: theory and applications*. Vol. 177. Springer Science & Business Media, 2005.

[86] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[87] Harold W Kuhn and Albert W Tucker. “Nonlinear programming”. In: *Traces and emergence of nonlinear programming*. Springer, 2013, pp. 247–258.

[88] György Steinbrecher and William T Shaw. “Quantile mechanics”. In: *European journal of applied mathematics* 19.2 (2008), pp. 87–112.

[89] Bernt Oksendal. *Stochastic differential equations: an introduction with applications*. Springer Science & Business Media, 2013.

[90] John P Boyd. *Chebyshev and Fourier spectral methods*. Courier Corporation, 2001.

[91] Georgi P Tolstov. *Fourier series*. Courier Corporation, 2012.

[92] Lloyd N Trefethen. *Approximation Theory and Approximation Practice, Extended Edition*. SIAM, 2019.

[93] Philip Hartman. *Ordinary differential equations*. SIAM, 2002.

[94] Michael Renardy and Robert C Rogers. *An introduction to partial differential equations*. Vol. 13. Springer Science & Business Media, 2006.

[95] Martin Hutzenthaler et al. “Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations”. In: *Proceedings of the Royal Society A* 476.2244 (2020), p. 20190630.

[96] Lawrence F Shampine and Charles William Gear. “A user’s view of solving stiff ordinary differential equations”. In: *SIAM review* 21.1 (1979), pp. 1–17.

[97] Morris W Hirsch, Stephen Smale, and Robert L Devaney. *Differential equations, dynamical systems, and an introduction to chaos*. Vol. 60. Academic press, 2004.

[98] Mark S Gockenbach. *Partial differential equations: analytical and numerical methods*. SIAM, 2010.

[99] John Charles Butcher. *Numerical methods for ordinary differential equations*. John Wiley & Sons, 2016.

[100] Zhiping Mao and George Em Karniadakis. “A spectral method (of exponential convergence) for singular solutions of the diffusion equation with general two-sided fractional derivative”. In: *SIAM Journal on Numerical Analysis* 56.1 (2018), pp. 24–49.

[101] Dominic W Berry. “High-order quantum algorithm for solving linear differential equations”. In: *Journal of Physics A: Mathematical and Theoretical* 47.10 (2014), p. 105301.

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

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

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

[105] Seth Lloyd et al. "Quantum algorithm for nonlinear differential equations". In: *arXiv preprint arXiv:2011.06571* (2020).

[106] Andrew M Childs and Jin-Peng Liu. "Quantum spectral methods for differential equations". In: *Communications in Mathematical Physics* 375.2 (2020), pp. 1427–1457.

[107] Benjamin Zanger et al. "Quantum algorithms for solving ordinary differential equations via classical integration methods". In: *Quantum* 5 (2021), p. 502.

[108] Robert Wille, Rod Van Meter, and Yehuda Naveh. "IBM's Qiskit tool chain: Working with and developing for real quantum computers". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 1234–1240.

[109] Ville Bergholm et al. "Pennylane: Automatic differentiation of hybrid quantum-classical computations". In: *arXiv preprint arXiv:1811.04968* (2018).

[110] Xiu-Zhe Luo et al. "Yao. jl: Extensible, efficient framework for quantum algorithm design". In: *Quantum* 4 (2020), p. 341.

[111] Dominik Seitz et al. "Qadence: a differentiable interface for digital-analog programs". In: *arXiv preprint arXiv:2401.09915* (2024).

[112] Oleksandr Kyriienko, Annie E Paine, and Vincent E Elfving. "Solving nonlinear differential equations with differentiable quantum circuits". In: *Physical Review A* 103.5 (2021), p. 052416.

[113] Francisco Javier Gil Vidal and Dirk Oliver Theis. “Input redundancy for parameterized quantum circuits”. In: *Frontiers in Physics* 8 (2020), p. 297.

[114] Marcello Benedetti et al. “Parameterized quantum circuits as machine learning models”. In: *Quantum Science and Technology* 4.4 (2019), p. 043001.

[115] Don H Johnson, Sinan Sinanovic, et al. “Symmetrizing the kullback-leibler distance”. In: *IEEE Transactions on Information Theory* 1.1 (2001), pp. 1–10.

[116] ML Menéndez et al. “The jensen-shannon divergence”. In: *Journal of the Franklin Institute* 334.2 (1997), pp. 307–318.

[117] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[118] Jinfeng Zeng et al. “Learning and inference on generative adversarial quantum circuits”. In: *Physical Review A* 99.5 (2019), p. 052306.

[119] Jin-Guo Liu et al. “Variational quantum eigensolver with fewer qubits”. In: *Physical Review Research* 1.2 (2019), p. 023025.

[120] Kouhei Nakaji and Naoki Yamamoto. “Expressibility of the alternating layered ansatz for quantum computation”. In: *Quantum* 5 (2021), p. 434.

[121] John David Anderson and John Wendt. *Computational fluid dynamics*. Vol. 206. Springer, 1995.

[122] Frank Gaitan. “Finding flows of a Navier–Stokes fluid through quantum computing”. In: *npj Quantum Information* 6.1 (2020), p. 61.

[123] Annie E Paine, Vincent E Elfving, and Oleksandr Kyriienko. “Quantum Quantile Mechanics: Solving Stochastic Differential Equations for Generating Time-Series”. In: *Advanced Quantum Technologies* 6.10 (2023), p. 2300065.

[124] Crispin Gardiner and Peter Zoller. *Quantum noise: a handbook of Markovian and non-Markovian quantum stochastic methods with applications to quantum optics*. Springer Science & Business Media, 2004.

[125] Darryl D Holm. “Variational principles for stochastic fluid dynamics”. In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 471.2176 (2015), p. 20140963.

[126] Carlo Cintolesi and Etienne Mémin. “Stochastic modelling of turbulent flows for numerical simulations”. In: *Fluids* 5.3 (2020), p. 108.

[127] Ulrich Dobramysl et al. “Stochastic population dynamics in spatially extended predator–prey systems”. In: *Journal of Physics A: Mathematical and Theoretical* 51.6 (2018), p. 063001.

[128] Tim Siu-tang Leung and Xin Li. *Optimal mean reversion trading: Mathematical analysis and practical applications*. Vol. 1. World Scientific, 2015.

[129] Daniel T Gillespie. “Exact numerical simulation of the Ornstein-Uhlenbeck process and its integral”. In: *Physical review E* 54.2 (1996), p. 2084.

[130] Jose Antonio Carrillo and Giuseppe Toscani. “Wasserstein metric and large–time asymptotics of nonlinear diffusion equations”. In: *New Trends in Mathematical Physics: In Honour of the Salvatore Rionero 70th Birthday*. World Scientific, 2004, pp. 234–244.

[131] William T Shaw. “Sampling Student’s T distribution-use of the inverse cumulative distribution function”. In: *Journal of Computational Finance* 9.4 (2006), p. 37.

[132] William T Shaw, Thomas Luu, and Nick Brickman. “Quantile mechanics II: changes of variables in Monte Carlo methods and GPU-optimised normal quantiles”. In: *European Journal of Applied Mathematics* 25.2 (2014), pp. 177–212.

[133] Warren Gilchrist. *Statistical modelling with quantile functions*. CRC Press, 2000.

[134] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[135] Takahiro Goto, Quoc Hoan Tran, and Kohei Nakajima. “Universal approximation property of quantum feature map”. In: *arXiv preprint arXiv:2009.00298* (2020).

[136] Maria Schuld and Nathan Killoran. “Quantum machine learning in feature Hilbert spaces”. In: *Physical review letters* 122.4 (2019), p. 040504.

[137] Oleksandr Kyriienko, Annie E Paine, and Vincent E Elfving. “Protocols for trainable and differentiable quantum generative modelling”. In: *arXiv preprint arXiv:2202.08253* (2022).

[138] Niklas Heim et al. “Quantum model-discovery”. In: *arXiv preprint arXiv:2111.06376* (2021).

[139] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. “Discovering governing equations from data by sparse identification of nonlinear dynamical systems”. In: *Proceedings of the national academy of sciences* 113.15 (2016), pp. 3932–3937.

[140] Maziar Raissi. “Deep hidden physics models: Deep learning of nonlinear partial differential equations”. In: *The Journal of Machine Learning Research* 19.1 (2018), pp. 932–955.

[141] Oldrich Vasicek. “An equilibrium characterization of the term structure”. In: *Journal of financial economics* 5.2 (1977), pp. 177–188.

[142] Rogemar S Mamon. “Three ways to solve for bond prices in the Vasicek model”. In: *Advances in Decision Sciences* 8.1 (2004), pp. 1–14.

[143] John Hull and Alan White. “Pricing interest-rate-derivative securities”. In: *The review of financial studies* 3.4 (1990), pp. 573–592.

[144] Brian Coyle et al. “Quantum versus classical generative modelling in finance”. In: *Quantum Science and Technology* 6.2 (2021), p. 024013.

[145] Alex Kondratyev. “Non-Differentiable Leaning of Quantum Circuit Born Machine with Genetic Algorithm”. In: *Wilmott* 2021.114 (2021), pp. 50–61.

[146] Milton Abramowitz, Irene A Stegun, and Robert H Romer. *Handbook of mathematical functions with formulas, graphs, and mathematical tables*. 1988.

[147] Peter E Kloeden et al. *Stochastic differential equations*. Springer, 1992.

[148] Christopher Rackauckas and Qing Nie. "Differentialequations. jl–a performant and feature-rich ecosystem for solving differential equations in julia". In: *Journal of open research software* 5.1 (2017).

[149] Tong Li, Shibin Zhang, and Jinyue Xia. "Quantum generative adversarial network: A survey". In: *Computers, Materials & Continua* 64.1 (2020), pp. 401–438.

[150] Jie Gui et al. "A review on generative adversarial networks: Algorithms, theory, and applications". In: *IEEE transactions on knowledge and data engineering* 35.4 (2021), pp. 3313–3332.

[151] Matthias C Caro et al. "Encoding-dependent generalization bounds for parametrized quantum circuits". In: *Quantum* 5 (2021), p. 582.

[152] Joseph Bowles et al. "Contextuality and inductive bias in quantum machine learning". In: *arXiv preprint arXiv:2302.01365* (2023).

[153] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. "Quantum generative adversarial networks for learning and loading random distributions". In: *npj Quantum Information* 5.1 (2019), p. 103.

[154] Bobak Toussi Kiani et al. "Learning quantum data with the quantum earth mover's distance". In: *Quantum Science and Technology* 7.4 (2022), p. 045002.

[155] Elie Bermot et al. "Quantum generative adversarial networks for anomaly detection in high energy physics". In: *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. Vol. 1. IEEE. 2023, pp. 331–341.

[156] Joaquim RRA Martins, Peter Sturdza, and Juan J Alonso. "The complex-step derivative approximation". In: *ACM Transactions on Mathematical Software (TOMS)* 29.3 (2003), pp. 245–262.

[157] Martín Larocca et al. “Group-invariant quantum machine learning”. In: *PRX Quantum* 3.3 (2022), p. 030341.

[158] Annie E Paine, Vincent E Elfving, and Oleksandr Kyriienko. “Quantum kernel methods for solving regression problems and differential equations”. In: *Physical Review A* 107.3 (2023), p. 032428.

[159] Siamak Mehrkanoon, Tillmann Falck, and Johan AK Suykens. “Approximate solutions to ordinary differential equations using least squares support vector machines”. In: *IEEE transactions on neural networks and learning systems* 23.9 (2012), pp. 1356–1367.

[160] Siamak Mehrkanoon and Johan AK Suykens. “Learning solutions to partial differential equations using LS-SVM”. In: *Neurocomputing* 159 (2015), pp. 105–116.

[161] Harry Buhrman et al. “Quantum fingerprinting”. In: *Physical Review Letters* 87.16 (2001), p. 167902.

[162] Oscar Higgott, Daochen Wang, and Stephen Brierley. “Variational quantum computation of excited states”. In: *Quantum* 3 (2019), p. 156.

[163] Artur F Izmaylov, Robert A Lang, and Tzu-Ching Yen. “Analytic gradients in variational quantum algorithms: Algebraic extensions of the parameter-shift rule to general unitary transformations”. In: *Physical Review A* 104.6 (2021), p. 062443.

[164] Javier Gil Vidal and Dirk Oliver Theis. “Calculus on parameterized quantum circuits”. In: *arXiv preprint arXiv:1812.06323* (2018).

[165] Dirk Oliver Theis. “Optimality of finite-support parameter shift rules for derivatives of variational quantum circuits”. In: *arXiv preprint arXiv:2112.14669* (2021).

[166] Yanfei Lu et al. “Solving higher order nonlinear ordinary differential equations with least squares support vector machines.” In: *Journal of Industrial & Management Optimization* 16.3 (2020).

[167] Lucile Savary and Leon Balents. “Quantum spin liquids: a review”. In: *Reports on Progress in Physics* 80.1 (2016), p. 016502.

[168] Maria Hermanns, Itamar Kimchi, and Johannes Knolle. “Physics of the Kitaev model: fractionalization, dynamic correlations, and material connections”. In: *Annual Review of Condensed Matter Physics* 9 (2018), pp. 17–33.

[169] Tatiana A Bespalova and Oleksandr Kyriienko. “Quantum simulation and ground state preparation for the honeycomb Kitaev model”. In: *arXiv preprint arXiv:2109.13883* (2021).

[170] Hsin-Yuan Huang et al. “Quantum advantage in learning from experiments”. In: *Science* 376.6598 (2022), pp. 1182–1186.

[171] John Michael Tutill Thompson, H Bruce Stewart, and Rick Turner. “Nonlinear dynamics and chaos”. In: *Computers in Physics* 4.5 (1990), pp. 562–563.

[172] Patrick Koch et al. “Tuning and evolution of support vector kernels”. In: *Evolutionary Intelligence* 5 (2012), pp. 153–170.

[173] Martin Larocca et al. “Diagnosing barren plateaus with tools from quantum optimal control”. In: *Quantum* 6 (2022), p. 824.

[174] Supanut Thanasilp et al. “Exponential concentration and untrainability in quantum kernel methods”. In: *arXiv preprint arXiv:2208.11060* (2022).

[175] Chelsea A Williams et al. “Quantum Chebyshev Transform: Mapping, Embedding, Learning and Sampling Distributions”. In: *arXiv preprint arXiv:2306.17026* (2023).

[176] Ismail Mohd et al. “Orthogonal functions based on Chebyshev polynomials”. In: *MATEMATIKA: Malaysian Journal of Industrial and Applied Mathematics* (2011), pp. 97–107.

[177] Andrew M Childs, Robin Kothari, and Rolando D Somma. “Quantum algorithm for systems of linear equations with exponentially improved dependence on precision”. In: *SIAM Journal on Computing* 46.6 (2017), pp. 1920–1950.

[178] Andreas Klappenecker and Martin Rotteler. “Discrete cosine transforms on quantum computers”. In: *ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis. In conjunction with 23rd International Conference on Information Technology Interfaces (IEEE Cat.* IEEE. 2001, pp. 464–468.

[179] Roeland Wiersema et al. “Measurement-induced entanglement phase transitions in variational quantum circuits”. In: *SciPost Physics* 14.6 (2023), p. 147.

[180] Fischer Black and Myron Scholes. “The pricing of options and corporate liabilities”. In: *Journal of political economy* 81.3 (1973), pp. 637–654.

[181] Annie E Paine, Vincent E Elfving, and Oleksandr Kyriienko. “Physics-Informed Quantum Machine Learning: Solving nonlinear differential equations in latent spaces without costly grid evaluations”. In: *arXiv preprint arXiv:2308.01827* (2023).

[182] Robert M Parrish and Peter L McMahon. “Quantum filter diagonalization: Quantum eigendecomposition without full quantum phase estimation”. In: *arXiv preprint arXiv:1909.08925* (2019).

[183] Tatiana A Bespalova and Oleksandr Kyriienko. “Hamiltonian operator approximation for energy measurement and ground-state preparation”. In: *PRX Quantum* 2.3 (2021), p. 030318.

[184] Oleksandr Kyriienko. “Quantum inverse iteration algorithm for programmable quantum simulators”. In: *npj Quantum Information* 6.1 (2020), p. 7.

[185] Emanuel Malvetti, Raban Iten, and Roger Colbeck. “Quantum circuits for sparse isometries”. In: *Quantum* 5 (2021), p. 412.

[186] Arthur G Rattew and Bálint Koczor. “Preparing arbitrary continuous functions in quantum registers with logarithmic complexity”. In: *arXiv preprint arXiv:2205.00519* (2022).

[187] Xiao-Ming Zhang, Tongyang Li, and Xiao Yuan. “Quantum state preparation with optimal circuit depth: Implementations and applications”. In: *Physical Review Letters* 129.23 (2022), p. 230504.

[188] Javier Gonzalez-Conde et al. *Efficient amplitude encoding of polynomial functions into quantum computers*. 2023.

[189] M Yousuff Hussaini, Craig L Streett, and Thomas A Zang. *Spectral methods for partial differential equations*. Tech. rep. 1983.

[190] Atiyo Ghosh et al. “Harmonic (Quantum) Neural Networks”. In: *arXiv preprint arXiv:2212.07462* (2022).

[191] Tingting Wang, Han Zhang, et al. “Some identities involving the derivative of the first kind Chebyshev polynomials”. In: *Mathematical Problems in Engineering* 2015 (2015).

[192] Guang Hao Low and Isaac L Chuang. “Hamiltonian simulation by qubitization”. In: *Quantum* 3 (2019), p. 163.

[193] Feng Wang et al. “Improved quantum ripple-carry addition circuit”. In: *Science China Information Sciences* 59 (2016), pp. 1–8.

[194] Omid Daei, Keivan Navi, and Mariam Zomorodi-Moghadam. “Optimized quantum circuit partitioning”. In: *International Journal of Theoretical Physics* 59.12 (2020), pp. 3804–3820.

Appendix A

Support Vector Regression for Differential Equations

Support vector regression for kernel methods was introduced in section 2.8.2. This workflow and problem set up can then be generalised for differential equation problems. The work flow remains the same with the constraints now being formed of the differential equation at a set of points along with the initial value/boundary condition. Derivatives are now included which we denote

$$\nabla_n^m \kappa(x, y) = \frac{\partial^{m+n}\kappa(x, y)}{\partial x^n \partial y^m}. \tag{A.1}$$

When considering solving DEs with support vector regression, the formulation of the problem changes depending on the form of differential equation considered [159, 160, 166]. I work through an example of preparing and SVR problem for the example class of problems $\mathrm{DE}(x, f, df/dx) = df/dx - g(x, f) = 0$ with initial condition $f(x_0) = f_0$. We use the model formulated as $f(x) = \mathbf{w}^\dagger\varphi(x) + b$.

As a first step, the problem needs to be written in primal SVR model form,

$$\min_{w,b} \mathbf{w}^{\dagger}\mathbf{w} + \gamma \mathbf{e}^T\mathbf{e} + \gamma \boldsymbol{\xi}^T\boldsymbol{\xi}, \tag{A.2}$$

$$\text{subject to } \mathbf{w}^T\varphi'(x_i) - g(x_i, y_i) = e_i \; i = 1 : N, \tag{A.3}$$

$$\mathbf{w}^T\phi(x_0) + b = f_0, \tag{A.4}$$

$$y_i = \mathbf{w}^T\varphi(x_i) + b + \xi_i \; i = 1 : N. \tag{A.5}$$

Here, the minimisation function is such that the magnitude of $\mathbf{w}$, $\mathbf{e}$ and ξ are minimised, with $\mathbf{w}$ being the set of fitting coefficients. $\mathbf{e}$ and ξ are the errors in the constraints. Minimising this function one finds the smallest $\mathbf{w}$ that fulfils the constraints with smallest possible error. Finding the smallest possible $\mathbf{w}$ is a form of regularisation helping prevent overfitting. γ is a tunable hyperparameter which dictates how much emphasis is placed on error reduction.

The constraints correspond to the differential equation at each point x_i, the initial condition and introduced dummy variables $y_i = f(x_i) + \xi_i$, respectively. The dummy variables are introduced to reflect the nonlinearity of the problem.

The second step is to find the Lagrangian of the model. This corresponds to the minimisation function minus each of the constraints, preceded by a variable coefficient,

$$\mathcal{L} = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{\gamma}{2}\mathbf{e}^T\mathbf{e} + \frac{\gamma}{2}\boldsymbol{\xi}^T\xi \tag{A.6}$$

$$- \sum_{i=1}^{N} \alpha_i(\mathbf{w}^T\varphi'(x_i) - g(x_i, y_i) - e_i) \tag{A.7}$$

$$- \beta(\mathbf{w}^T\varphi(x_0) + b - f_0) \tag{A.8}$$

$$- \sum_{i=1}^{N} \eta_i(\mathbf{w}^T\varphi(x_i) + b + \xi_i - y_i). \tag{A.9}$$

The introduced variables $\boldsymbol{\alpha}$, $\boldsymbol{\eta}$ and β are referred to as dual variables.

The next step is to calculate the KKT conditions. These are found by equating to zero derivative of the Lagrangian with respect to each of its variables, both primal and dual, $(\mathbf{w}, b, \mathbf{e}, \boldsymbol{\xi}, \mathbf{y}, \boldsymbol{\alpha}, \beta, \boldsymbol{\eta})$. The derivatives read

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_i (\alpha_i \varphi'(x_i) + \eta_i \varphi(x_i)) - \beta \varphi(x_0) = 0, \tag{A.10}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\beta - \sum_i \eta_i = 0, \tag{A.11}$$

$$\frac{\partial \mathcal{L}}{\partial e_i} = \gamma e_i + \alpha_i = 0, \tag{A.12}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = \gamma \xi_i - \eta_i = 0, \tag{A.13}$$

$$\frac{\partial \mathcal{L}}{\partial y_i} = \alpha_i \frac{\partial g}{\partial y}(x_i, y_i) + \eta_i = 0, \tag{A.14}$$

$$\frac{\partial \mathcal{L}}{\partial \alpha_i} = -\left(\mathbf{w}^\dagger \varphi'(x_i) - g(x_i, y_i) - e_i\right) = 0, \tag{A.15}$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = -\left(\mathbf{w}^\dagger \varphi(x_0) + b - f_0\right) = 0, \tag{A.16}$$

$$\frac{\partial \mathcal{L}}{\partial \eta_i} = -\left(y_i - \mathbf{w}^\dagger \varphi(x_i) - b - \xi_i\right) = 0. \tag{A.17}$$

These are a set of $6|\mathbf{x}| + 2$ equations which necessarily need to be satisfied for optimality.

These conditions are now used to eliminate a subset of the primal variables $\mathbf{w}, \mathbf{e}, \boldsymbol{\xi}$ leaving $3|\mathbf{x}| + 2$ equations:

$$\left(\sum_j [\alpha_j \varphi'(x_j) + \eta_j \varphi(x_j)] + \beta\varphi(x_0)\right)^\dagger \varphi'(x_i) \tag{A.18}$$

$$-g(x_i, y_i) + \alpha_i/\gamma = 0,$$

$$\left(\sum_j [\alpha_j \varphi'(x_j) + \eta_j \varphi(x_j)] + \beta\varphi(x_0)\right)^\dagger \varphi(x_0) \tag{A.19}$$

$$+b - f_0 = 0,$$

$$-\left(\sum_j [\alpha_j \varphi'(x_j) + \eta_j \varphi(x_j)] + \beta\varphi(x_0)\right)^\dagger \varphi(x_i) \tag{A.20}$$

$$+y_i - b - \eta_i/\gamma = 0,$$

$$\sum_i \eta_i + \beta = 0, \tag{A.21}$$

$$\alpha_i \frac{\partial g}{\partial y}(x_i, y_i) + \eta_i = 0. \tag{A.22}$$

For these equations we then expand out the brackets and use the kernel trick, introducing the kernel function κ as $\kappa(x, y) = \varphi^\dagger(x)\varphi(y)$ and corresponding derivatives. We remember that this is a consequence of Mercers theorem, given that $\varphi^\dagger(x)\varphi(y)$ is a kernel for any φ. Now we are able to write the resulting equations in matrix form as

$$\left[\begin{array}{c|c|c|c|c} \tilde{\Omega}_1^1 & \Omega_0^1 & \mathbf{h}_0^1 & \mathbf{0} & \hat{0} \\ \hline \Omega_1^0 & \tilde{\Omega}_0^0 & \mathbf{h}_0^0 & \mathbf{1} & -\hat{I} \\ \hline \mathbf{h}^{T}{}_1^0 & \mathbf{h}^{T}{}_0^0 & \tilde{h} & 1 & \mathbf{0}^T \\ \hline \mathbf{0}^T & \mathbf{1}^T & 1 & 0 & \mathbf{0}^T \\ \hline \hat{D} & \hat{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{array}\right] \left[\begin{array}{c} \boldsymbol{\alpha} \\ \hline \boldsymbol{\eta} \\ \hline \beta \\ \hline b \\ \hline \mathbf{y} \end{array}\right] = \left[\begin{array}{c} \tilde{\mathbf{g}} \\ \hline \mathbf{0} \\ \hline f_0 \\ \hline 0 \\ \hline \hat{0} \end{array}\right], \tag{A.23}$$

where the notation is as follows

$$[\Omega_n^m]_{i,j} = \nabla_n^m \kappa(x_j, x_i), \tag{A.24}$$

$$\tilde{\Omega}_n^m = \Omega_n^m + \hat{I}/\gamma, \tag{A.25}$$

$$[\mathbf{h}_n^m]_i = \nabla_n^m \kappa(x_0, x_i), \tag{A.26}$$

$$\tilde{h} = \kappa(x_0, x_0), \tag{A.27}$$

$$\hat{D} = \mathrm{diag}\left(\left\{\frac{\partial g}{\partial f}(x_i, y_i)\right\}_i\right), \tag{A.28}$$

$$[\tilde{\mathbf{g}}]_i = g(x_i, y_i). \tag{A.29}$$

We now have a set of nonlinear equations that can be solved for a set of variable, representing solution to the original stated problem. These equations are written in terms of κ, and not φ. Also note that these equations are true for any valid kernel function, and we can choose our kernel function freely. We need not know what the corresponding φ are, we simply know from Mercers theorem that such functions exist. Therefore the formulation of these equations (in particular the use of the kernel trick to introduce the kernel) is valid.

The remaining step is to write $f(x) = \mathbf{w}^\dagger \varphi(x) + b$ in a form that is instead dependent on the variables solved for. We find it to be

$$f(x) = \sum_{i=1}^{|\mathbf{x}|} \alpha_i \nabla_1^0 \kappa(x_i, x) + \sum_{i=1}^{|\mathbf{x}|} \eta_i \kappa(x_i, x) + \beta \kappa(x_0, x) + b, \tag{A.30}$$

by using the $\mathbf{w}$ KKT condition and then the kernel trick.

We have now formulated an SVR method for the form of problem considered. If a different form of DE is considered this process would have to be repeated for that form. Once in the SVR form any relevant algorithm can be used. We do note that when nonlinear problems are considered the resulting system of equations is in turn nonlinear. Consequently the guarantee of convexity is lost.

Appendix B

QCL Example Code

```
using Zygote, GalacticFlux, GalacticOptim #packages for optimisation

using Plots

using Statistics

#packages for classical simulation of quantum computing

using Yao, Yao.EasyBuild


n = 4 #number of qubits

d = 5 # depth of variational ansatz


x_min = 0.

x_max = 5.

n_points = 20

x_train = x_min:(x_max-x_min)/(n_points-1):x_max #grid for training

xs = reshape(collect(x_train), 1, :)


f_tar(x) = x^2*cos(2*\pi*x) + 1 # target function


f_m(x) = return chain(n, [put(i=>Ry(i*x)) for i=1:n]) # feature map

cost = sum([chain(n, put(i=>Z)) for i=1:n]) #measurement operator

# variational ansatz - using Yao included HEA

var_ansatz = variational_circuit(n,d)
```

```
num_theta = nparameters(var_ansatz_qcl)+ 2

#trial function
f_train(x::Float64, theta) = theta[end-1]*real(expect(cost, zero_state(n) =>
    chain(n, f_m(x), dispatch(var_ansatz_qcl, theta[1:end-2])))) + theta[end]
f_train(x, theta) = [f_train(xj, theta) for xj in x]

# loss function for regression
# MSE between current trial function values and taret values over training grid
function loss(theta, p=nothing)
    fvals = f_train(vec(xs), theta)
    mean(abs2, fvals .- f_tar.(vec(xs)))
end

#callback to track loss
loss_list = []
function callback(theta, l)
    push!(loss_list, l)
    false
end

using Random
Random.seed!(1234) # if you want reproducible randomness

theta_init = rand(num_theta) # initialise parameters
niter = 1000 # maximum iterations

# set up optimiser - using adam with automatic derivatives
opt_f = OptimizationFunction(loss, GalacticOptim.AutoZygote())
prb = OptimizationProblem(opt_f,\theta_init)
```

```
opt = ADAM(1e-2)


# perform optimisation

res = solve(prb, opt, maxiters=niter, callback = callback)

res_theta = res.minimizer # result from training


# visualise loss over iteration

lp = plot(1:lastindex(loss_list), loss_list, yaxis = :log10)

display(lp)


# visualise result in comparison to target function

xp = x_min:0.01:x_max

p1 = plot(xp, f_tar.(xp), label="Truth", w =3, color = :lightblue)

plot!(p1, xp, f_train(vec(xp), res_theta) |> vec, label="Pred.",

       style =:dash, color = :blue, w = 2)

plot!(p1, x_train, f_tar.(x_train), seriestype =:scatter, label = "Train")

display(p1)
```