




Quantum kernel methods for solving regression problems and differential equationsAnnie E. Paine ^{1,2} Vincent E. Elfving ¹ and Oleksandr Kyriienko ^{1,2}¹*PASQAL SAS, 2 av. Augustin Fresnel, 91120 Palaiseau, France*²*Department of Physics and Astronomy, University of Exeter, Stocker Road, Exeter EX4 4QL, United Kingdom* (Received 4 November 2022; revised 28 February 2023; accepted 7 March 2023; published 31 March 2023)

We propose several 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 automatic 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 minimize 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. Contrary to prevailing hybrid variational approaches for parametrized quantum circuits, we perform training of the weights of the model classically. Under certain conditions this corresponds to a convex optimization problem, which can be solved with provable convergence to global optimum of the model. The proposed approaches also favor hardware implementations, as optimization only uses evaluated Gram matrices, but require a quadratic number of function evaluations. We highlight trade-offs when comparing our methods to those based on variational quantum circuits such as the recently proposed differentiable quantum circuits approach. The proposed methods offer potential quantum enhancement through the rich kernel representations using the power of quantum feature maps, and start the quest towards provably trainable quantum DE solvers.

DOI: [10.1103/PhysRevA.107.032428](https://doi.org/10.1103/PhysRevA.107.032428)**I. INTRODUCTION**

Solvers of differential equations (DEs) are essential for all areas of science [1,2]. These include fluid dynamics, ecology, finance, medical science, and many more. While some simple instances of differential equations can be solved analytically, in majority of cases numerical solvers are required. Existing numerical methods heavily rely on finite differencing methods on finely discretized grids [3]. Other classical methods include global spectral methods which effectively fit a function basis set to the differential equation problem considered [4]. Classical numerical solvers often suffer from instabilities that emerge in highly nonlinear systems. Another problem is the curse of dimensionality caused by an increase of grid for multidimensional systems. Thus, developing new techniques to solve DEs remains a hot area of contemporary research [5] and increasingly requires new computational architectures [6,7].

Quantum computing offers advantages in performing certain computational tasks [8–10]. Enabled by quantum principles, the use of superposition and entanglement can lead to fundamentally different scaling for quantum algorithms

as compared to classical approaches [11]. One example is for solving linear systems of equations [9], and quantum speed-up of matrix-vector multiplication [12]. When using finite-differencing, this translates directly to associated systems of DEs [13–18]. However, previously described techniques rely on large-scale implementation of quantum phase estimation, and require resource overheads that render their implementation infeasible in foreseeable future [19]. This prompts the development of different approaches that can potentially help solving nonlinear DEs with near-term devices.

Recently, rapid improvement of quantum computing hardware has called for algorithms that can operate in the noisy regime [20]. In this case the hybrid quantum-classical workflow is often used. One possibility is to formulate a problem such that solution can be searched variationally [21], where parameterized quantum circuits play a role similar to deep neural networks in classical machine learning (ML). This approach was coined as a quantum machine learning (QML) [22–24] and has triggered the development of various ML protocols for quantum hardware [25–34]. Variational approaches were also used for describing quantum evolution [35,36] and linear algebra [37–39]. In the field of nonlinear differential equations variational algorithms were used together with amplitude encoding [40], where multiple quantum registers are required for encoding nonlinearity. Alternatively nonlinear systems have been discretized, such that they are linearized and then solved by matrix inversion methods [41]. Another approach was proposed in [42], in which a

Published by the American Physical Society under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

QML-type workflow is used. There a DE solution is represented by a differentiable quantum circuit (DQC), with nonlinear dependence being introduced via feature map encoding [43] and cost function based readout, while function derivatives are introduced with automatic circuit differentiation [25,44]. Similar solutions were developed for continuous variable QML [45], stochastic differential equations [46], and generative modeling [47,48].

Another facet of quantum machine learning was revealed when formulating models in terms of kernel functions—similarity functions that define a distance between two data points [24,49]. The core concept of kernel methods is the so-called “kernel trick” that maps data into a high-dimensional space [50]. Kernel methods are frequently used in classical machine learning, and aim to rewrite the ML task as a convex optimization problem [51]. In the quantum domain, kernel functions are conveniently defined as overlaps between parametrized quantum states that represent data [24,49] or any similar measure [52]. It was conjectured that many supervised QML models can be considered as kernel methods that are well suited to near-term devices [53]. Currently these methods have mainly been considered for classification purposes [54,55].

In this paper, we propose to use quantum kernel methods for regression problems, including solvers of nonlinear differential equations [56]. In classical ML kernel methods are used for support vector regression (SVR) [57], where the kernel trick and convex optimization lead to expressive and provably trainable models. Kernel methods can also be applied to solve differential equations [58–60], while being limited by the expressivity of classical kernels. We describe two approaches that express solutions of DEs as quantum kernel-based models, and describe the rules for their automatic differentiation. We refer to the two as mixed model regression (MMR) and support vector regression (SVR) protocols. The protocols are applied to test problems of regression on quantum data, linear DEs, and nonlinear DEs in the form of the Duffing equation [61]. We discuss the cases where the proposed workflow for DEs may provide advantage over existing methods.

II. QUANTUM KERNEL METHODS

We start by introducing the concept of a quantum kernel function. A kernel function is a conjugate-symmetric positive definite function κ mapping two variables $x, y \in \mathcal{X}$ to the complex space, $\kappa: \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{C}$. Quantum kernel function refers to a function which fulfils the requirements of a kernel function and can be evaluated by a quantum computer. An important result concerning kernel functions is known as the kernel trick. The kernel trick relies on 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. 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 $f(s, t) = \sum_i \phi_i(s)\phi_i(t)$ [62]. An example of a quantum kernel function is an overlap $\kappa(x, y) = \langle \psi(x) | \psi(y) \rangle$ where $|\psi(x)\rangle$ denotes a state encoded by the variable x . This is an inner product which

fulfils the requirements of a kernel function. Later we will consider other forms of the quantum kernel function, showing how to encode the variable into the state and how to evaluate the kernel.

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). 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 $f_i = f(x_i)$. We consider how both MMR and SVR approach the described problem.

A. MMR

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

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

where $\mathbf{y} = \{y_i\}_i$ is a set of evaluation points, $|\mathbf{y}|$ denotes the size of set \mathbf{y} , and $\alpha = \{\alpha_i\}_i$ 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 optimized with respect to α and b the corresponding f_α solves the 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 α —the variable which is adjusted during optimization. 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 optimization procedure. Any suitable optimization method may be used to optimize $\mathcal{L}(\alpha)$. We can also see that the considered loss function is convex.

We consider the general case $\mathcal{L}(\alpha) = \sum_{i=1}^{|\mathbf{x}|} L(x_i; \alpha)^2$ with L being a linear function of α , 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 $\alpha_j \in \alpha$. We can write the second-order derivatives as

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

$$= \sum_{i=1}^{|\mathbf{x}|} 2 \left(\frac{\partial L}{\partial \alpha_j} \right)^2 \geq 0, \quad (3)$$

where passing from Eq. (2) to Eq. (3) 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 optimization methods [63].

The workflow to solve an MMR problem is as follows:

- (1) Choose setup for training, including the kernel function, optimizer, \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) Optimize 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 .

B. SVR

For support vector regression we represent a trial function as $f(x) = \mathbf{w}^\dagger \boldsymbol{\varphi}(x) + b$, where \mathbf{w} and b are tunable parameters, and $\boldsymbol{\varphi}(x)$ is a set of functions we later use the kernel trick upon. The first step is to write the problem as a primal (original) optimization model. This reads

$$\min_{\mathbf{w}, b} \{ \mathbf{w}^T \mathbf{w} + \gamma \mathbf{e}^T \mathbf{e} \}, \quad (4)$$

$$\text{subject to } f_i = \mathbf{w}^T \boldsymbol{\varphi}(x_i) + b + e_i, \quad i = 1:N, \quad (5)$$

where \mathbf{e} is the set of error variables which relax the constraints $f_i = \mathbf{w}^T \boldsymbol{\varphi}(x_i) + b + e_i$, and γ is a tunable hyperparameter that changes the emphasis on minimizing 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 [64]. 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 $\varphi(x_i)^T \varphi(x_j)$, 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

$$\left[\begin{array}{c|c} \hat{\Omega} + \hat{f}/\gamma & \mathbf{1} \\ \hline \mathbf{1}^T & 0 \end{array} \right] \begin{bmatrix} \boldsymbol{\alpha} \\ b \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix}, \quad (6)$$

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 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 $f(x; \boldsymbol{\alpha}) = \mathbf{w}^\dagger \boldsymbol{\varphi}(x) + b$, 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

$$f(x; \boldsymbol{\alpha}) = \sum_{i=1}^{|\mathbf{x}|} \alpha_i \kappa(x, x_i) + b. \quad (7)$$

Although we started considering $\boldsymbol{\varphi}$ as our fitting functions, the resulting function of this process is based on kernel evaluations, and we never need knowledge of what $\boldsymbol{\varphi}$ are or to directly evaluate them.

The workflow to prepare an SVR problem is as follows:

- (1) Write model with minimization function and constraints.
- (2) Write out the Lagrangian.
- (3) Find the KKT optimality conditions.
- (4) Eliminate subset of original optimization variables.

(5) Use the kernel trick to realize problem in terms of kernel functions.

(6) Write out remaining relationships as a system of equations.

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

In Appendix A this process is worked through in more detail for a specific (DE) example. 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 also note that the SVR method results in a form of problem that can still be considered as an optimization problem to be solved with an optimizer. 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.

Comparing the MMR and the SVR methods we note that the solving workflows 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. (1) and Eq. (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. (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 deterministic solution. 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.

III. QUANTUM KERNEL FUNCTION EVALUATION

We now look further into the specifics of quantum kernel functions. In particular, we consider their structure, where feature maps encode dependence on a variable into a state. We also consider how to evaluate them and their derivatives. 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

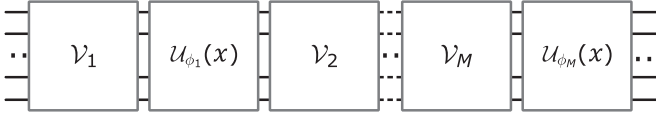


FIG. 1. Circuit diagram showing a general form of function encoding circuit $\mathcal{U}(x)$ used to implement quantum kernel. This is formed by layers of static circuits \mathcal{V}_i and data re-uploading circuits $\mathcal{U}_{\phi_i}(x)$ parametrized by x .

$\kappa(x, y) = |\langle \psi(x) | \psi(y) \rangle|^2$ as an absolute value squared of the overlap. This also corresponds to a valid kernel function [53]. We consider this kernel function as it is real valued—an advantage when expressing real-valued functions.

A. Encoding

The kernel functions we consider contain states $|x\rangle$, which are encoded by a classical variable x . To create such states we use feature map encoding, where x is embedded into the state by parametrizing gates preparing the state, $|\psi(x)\rangle = \mathcal{U}(x)|0\rangle$. A simple example of $\mathcal{U}(x)$ is the product feature map $\mathcal{U}_{\phi}(x) = \bigotimes_{j=1}^N R_{\alpha,j}[\phi(x)]$, where $R_{\alpha,j}[\phi(x)]$ is rotation on qubit j of angle $\phi(x)$ about a Pauli operator α . Other more complicated feature map encodings can be considered. The generalization may include the re-uploading technique [65] where action of feature maps can be layered with (nonvariational) entangling circuits, $\mathcal{U}(x) = \mathcal{U}_{\phi_M}(x)\mathcal{V}_M \cdots \mathcal{V}_2\mathcal{U}_{\phi_1}(x)\mathcal{V}_1$ (see Fig. 1). This layered form terminates with a circuit encoded by a variable as a final entangling circuit cancels out for kernels based on $\mathcal{U}^\dagger(x)\mathcal{U}(y)$. 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 while also being trainable [66].

B. Evaluation

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 [67,68]. 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. Using Hadamards gates and controlled operations an ancillary qubit can then be measured to read $|\langle \psi(x) | \psi(y) \rangle|^2$. The circuit diagram is shown in Fig. 2(b).

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

$$|\langle \psi(x) | \psi(y) \rangle|^2 = \langle 0 | \mathcal{U}^\dagger(y)\mathcal{U}(x) | 0 \rangle \langle 0 | \mathcal{U}^\dagger(x)\mathcal{U}(y) | 0 \rangle. \quad (8)$$

The measurement in Eq. (8) can be implemented naively by the circuit in Fig. 2(a). The circuit is initialized in the zero state. Then $\mathcal{U}(y)$ is applied, followed by $\mathcal{U}^\dagger(x)$. The probability of remaining in the zero state and consequently the kernel function value is then calculated by measuring all qubits and finding the ratio of times $|0\rangle$ is measured.

Another possible implementation is two evaluations of the Hadamard test with $N + 1$ qubits as shown in Fig. 2(c) [69]. This can be used to evaluate the real and imaginary parts of $\langle 0 | \mathcal{U}^\dagger(x)\mathcal{U}(y) | 0 \rangle$ which can then be used to evaluate the kernel as $\text{Re}(\langle 0 | \mathcal{U}^\dagger(x)\mathcal{U}(y) | 0 \rangle) + \text{Im}(\langle 0 | \mathcal{U}^\dagger(x)\mathcal{U}(y) | 0 \rangle)$.

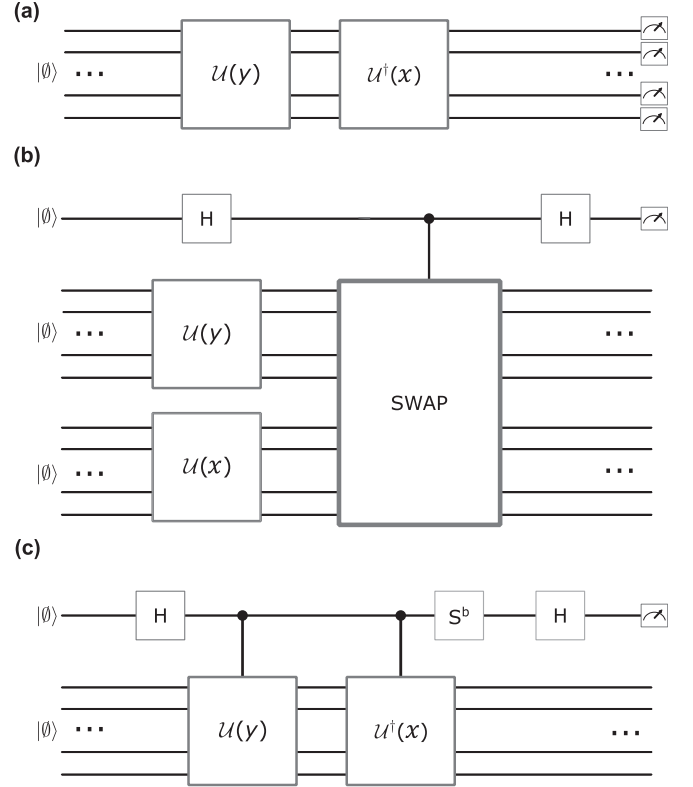


FIG. 2. Circuit diagrams for evaluating the kernel $\kappa(x, y) = |\langle \psi(x) | \psi(y) \rangle|^2$, where in all circuits \mathcal{U} and H represent the kernel feature map and the Hadamard gate, respectively. (a) Naive kernel evaluation based on consecutive application of \mathcal{U} circuits, 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 $\text{Re}(\langle 0 | \mathcal{U}^\dagger(x)\mathcal{U}(y) | 0 \rangle)$ and $\text{Im}(\langle 0 | \mathcal{U}^\dagger(x)\mathcal{U}(y) | 0 \rangle)$ for $b = 0$ and $b = 1$, respectively. S denotes the phase gate, $\exp(-i\pi Z/4)$.

C. 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:

$$\nabla_n^m \kappa(x, y) = \frac{\partial^{m+n} \kappa(x, y)}{\partial x^n \partial y^m}. \quad (9)$$

To implement derivative evaluation, one way is to consider the kernel as written in Eq. (8) and the parameter shift rule [25,44]. While different variations of the parameter shift rule exist, in a simple form quantum model and its derivative can be written as

$$f(x) = \langle 0 | \mathcal{U}^\dagger(x) C \mathcal{U}(x) | 0 \rangle, \quad (10)$$

$$\partial_x f(x) = \sum_{j=1}^n [f(x + \pi/2_j) - f(x - \pi/2_j)]/2, \quad (11)$$

where C is a measurement operator, n is the number of x -dependent gates in $\mathcal{U}(x)$, and $f(x \pm \pi/2_j)$ denotes $f(x)$ being evaluated with the j th gate parametrized by x shifted by

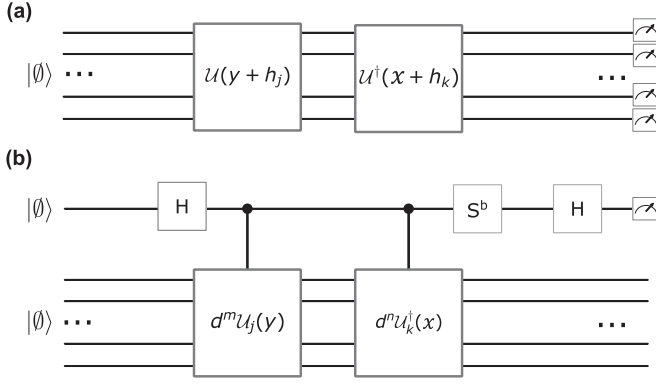


FIG. 3. Circuit diagrams used for evaluation of derivatives. (a) Generic circuit for differentiating kernels shown in Fig. 2(a) where a parameter shift rule is used. Depending on which derivative is calculated, gates parametrized by x and/or y have their parameters shifted up and down (shown by h_j and h_k). Contributions from all parametrized gates are then summed for overall derivative. (b) Using the Hadamard test for evaluation of the overlap $\langle 0|d^n/dx^n U_k^\dagger(x)d^m/dy^m U_j(y)|0\rangle$, 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 $\langle 0|d^n/dx^n U^\dagger(x)d^m/dy^m U(y)|0\rangle$ can be evaluated. These overlaps can then be used to evaluate kernel derivatives.

$\pm\pi/2$. The parameter shift rule can be implemented multiple times to calculate higher order derivatives.

With this method we take the kernel evaluation method as in Fig. 2(a) but shift x and y up and down depending on what derivative is being calculated in each gate that they parametrize. For example, for the first-order derivative with respect to x the number of evaluations of Fig. 3(a) is $2n$. Using the parameter shift rule means we calculate the analytic derivative though it does place some requirements on the gates parametrized by x and y such as being involutory. Generalized parameter shift rules are possible, where such requirements are relaxed [70–74].

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. (8) as

$$\begin{aligned} \frac{\partial}{\partial x} \kappa(x, y) &= \langle 0|U^\dagger(y)d/dx[U(x)]|0\rangle \langle 0|U^\dagger(x)U(y)|0\rangle \\ &+ \langle 0|U^\dagger(y)U(x)|0\rangle \langle 0|d/dx[U^\dagger(x)]U(y)|0\rangle, \end{aligned} \quad (12)$$

and thus we can evaluate this derivative by evaluating $\langle 0|U^\dagger(y)d/dx[U(x)]|0\rangle$ and $\langle 0|U^\dagger(x)U(y)|0\rangle$ (real and imaginary parts). The second term can be evaluated the same as for evaluating the kernel shown in Fig. 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 generalized layered form of kernel encoding $U(x) = U_{\phi_M}(x)\mathcal{V}_M \cdots \mathcal{V}_2 U_{\phi_1}(x)\mathcal{V}_1$ with each feature map being $U_{\phi_j}(x) = \exp[-i\mathcal{G}_j \phi_j(x)]$. For this case the derivative reads $U'(x) = \sum_{j=1}^M U_{M:j+1} U_{\phi_j}(-i\mathcal{G}_j) \phi_j'(x) \mathcal{V}_j U_{j-1:1}$ with

$U_{j:k} = U_{\phi_j}(x)\mathcal{V}_j U_{\phi_{j-1}}(x) \cdots U_{\phi_k}(x)\mathcal{V}_k$. We can now assume that the generators \mathcal{G}_j are unitary. When \mathcal{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 [44].

Once the procedure for evaluating derivatives has been established, we generalize to higher-order derivatives. By using the product rule in Eq. (8) whatever derivative is required, one can express it as sums of products of overlaps with $U(x)$ and $U(y)$ differentiated to different orders. These overlaps can be calculated with two (when generators are unitary) overlap tests for each gate with x and/or y as a parameter (see Fig. 3). These overlap evaluations can be reused for calculating different derivatives where the same overlap occurs.

IV. SOLVING DIFFERENTIAL EQUATIONS

In the following section, 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

$$\text{DE}(x, f, df/dx) = \frac{df}{dx} - g(x, f) = 0, \quad (13)$$

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 the following subsections.

A. MMR

When solving DEs of the type (13) via MMR, we choose a loss function of the form $\mathcal{L}(\alpha) = \sum_{i=1}^{|\mathcal{X}|} \{\text{DE}[x_i, f_\alpha(x_i), df_\alpha/dx(x_i)]\}^2 + (f_\alpha(x_0) - f_0)^2$. We remind that the trial function reads $f_\alpha(x) = b + \sum_{i=1}^{|\mathcal{Y}|} \alpha_i \kappa(x, y_i)$. Therefore, kernels κ and their derivatives $\nabla_{\mathcal{X}}^0 \kappa$ 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 need to be evaluated only once at the start, then being reused throughout optimization. The loss function can then be optimized via any appropriate optimizer 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 generalization bounds.

When the differential equation is linear [i.e., g is linear in f in Eq. (13)] 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. (3). 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 nonconvex.

B. SVR

When considering solving DEs with support vector regression, the formulation of the problem changes depending on the form of differential equation considered [58–60]. The steps for the problem formulation, however, remain the same: state a model, write out the Lagrangian, find KKT optimality conditions, eliminate the subset of prime variables by using the KKT conditions, use the kernel trick, and finally write out remaining equations in matrix form.

We follow the SVR formulation procedure for problems of the form $\text{DE}(x, f, df/dx) = df/dx - g(x, f) = 0$ with initial condition $f(x_0) = f_0$. We provide further details in Appendix A, and here provide the resulting set of equations in the matrix form:

$$\begin{bmatrix} \tilde{\Omega}_1^1 & \Omega_0^1 & \mathbf{h}_0^1 & \mathbf{0} & \hat{0} \\ \Omega_1^0 & \tilde{\Omega}_0^0 & \mathbf{h}_0^0 & \mathbf{1} & -\hat{f} \\ \mathbf{h}_1^{T0} & \mathbf{h}_0^{T0} & \tilde{h} & 1 & \mathbf{0}^T \\ \mathbf{0}^T & \mathbf{1}^T & 1 & 0 & \mathbf{0}^T \\ \hat{D} & \hat{f} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \alpha \\ \eta \\ \beta \\ b \\ \mathbf{y} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{g}} \\ \mathbf{0} \\ f_0 \\ 0 \\ \hat{0} \end{bmatrix}. \quad (14)$$

Here we introduced dummy variables y_i . η and β are dual variables introduced along with α corresponding to the dummy variable constraint and the initial variable constraint, respectively. The remaining notation is as follows:

$$[\Omega_n^m]_{i,j} = \nabla_n^m \kappa(x_j, x_i), \quad (15)$$

$$\tilde{\Omega}_n^m = \Omega_n^m + \hat{f}/\gamma, \quad (16)$$

$$[\mathbf{h}_n^m]_i = \nabla_n^m \kappa(x_0, x_i), \quad (17)$$

$$\tilde{h} = \kappa(x_0, x_0), \quad (18)$$

$$\hat{D} = \text{diag}\left(\left\{\frac{\partial g}{\partial f}(x_i, y_i)\right\}_i\right), \quad (19)$$

$$[\tilde{\mathbf{g}}]_i = g(x_i, y_i). \quad (20)$$

We now have a set of generally nonlinear equations, which can be solved for finding a vector of optimized weights. By substituting the relevant KKT optimality conditions into $f(x; \alpha) = \mathbf{w}^\dagger \boldsymbol{\varphi}(x) + b$ and employing the kernel trick, we get an expression for f in terms of kernel functions

$$f(x) = \sum_{i=1}^{|\mathcal{X}|} \alpha_i \nabla_1^0 \kappa(x_i, x) + \sum_{i=1}^{|\mathcal{X}|} \eta_i \kappa(x_i, x) + \beta \kappa(x_0, x) + b, \quad (21)$$

where optimized variables (weights) are used. Note that if the differential equation is linear, the dummy variable constraints of y_i are not required. This leads to a system of linear equations with lower dimension.

C. Other forms of DEs

Many practical problems are not of the form $\text{DE}(x, f, df/dx) = df/dx - g(x, f) = 0$ as 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 generalize to any other form of DE simply

relying on generalized optimization. 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 $\kappa(\mathbf{x}, \mathbf{y}) = |\langle 0 | \mathcal{U}^\dagger(\mathbf{x}) \mathcal{U}(\mathbf{y}) | 0 \rangle|^2$, where the feature maps now encode a vector of domain variables. The simplest form is $\mathcal{U}(\mathbf{x}) = \mathcal{U}(x_1) \mathcal{U}(x_2) \cdots \mathcal{U}(x_M)$ 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 [58,60] and SVRs for PDEs [59] are possible, as well as their generalizations for systems of differential equations. An example of solving a second-order differential equation is presented within the following results section.

V. RESULTS

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

A. Regression on quantum data

We start by considering the case of regression. We generate a quantum dataset that corresponds to dynamics of total magnetization of a biased honeycomb Kitaev model [75,76]. The Hamiltonian of the system reads

$$\mathcal{H} = J \left(\sum_{(i,j) \in \mathcal{X}} X_i X_j + \sum_{(i,j) \in \mathcal{Y}} Y_i Y_j + \sum_{(i,j) \in \mathcal{Z}} Z_i Z_j \right) + h_z \sum_{j=1}^N Z_j, \quad (22)$$

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 $M_z = \sum_j Z_j/N$ for $N = 12$ qubits on a lattice with periodic boundary conditions [77], starting from the uniform initial state. The choice of a quantum dataset with strong magnetic correlations may be especially suitable for kernel-based regression, given recent advances in learning from experiments [78]. Choosing a subset of evolved magnetization 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 1 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. The classical kernel used is a commonly used radial basis function (RBF) kernel $\kappa(x, y) = \exp[-(x - y)^2/(2\sigma^2)]$, 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 parametrized X rotations, $R_x[\phi(x)]$, 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

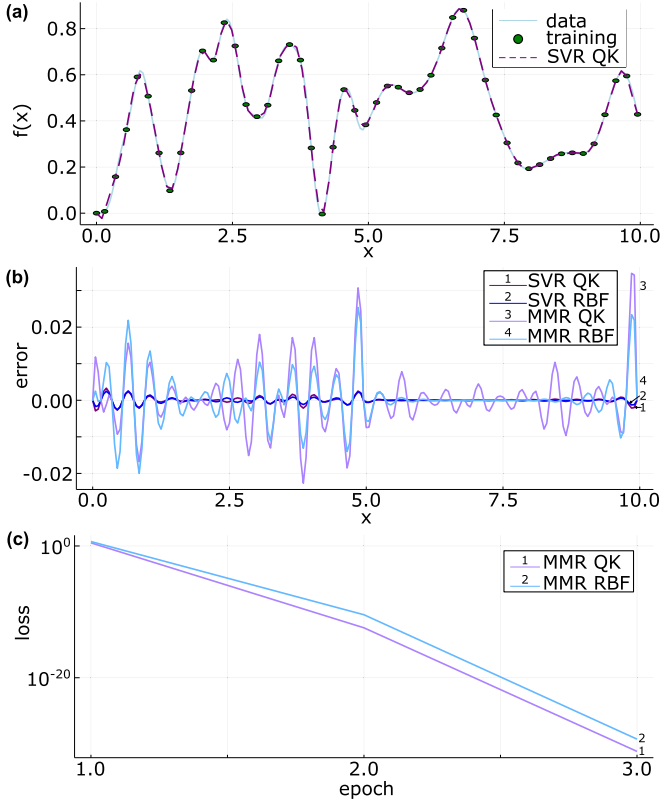


FIG. 4. MMR and SVR used to solve a regression problem. Data to fit are for time-evolved magnetization of a biased honeycomb Kitaev model. (a) Solution via SVR method for quantum kernel with two layers and $N = 8$ shown by dashed purple line. Data plotted as solid light blue curve with points used for training highlighted with green circles. (b) Error between results and underlying data plotted over x as $[f(x) - f_{\text{data}}(x)]$, and we normalize data by the range of magnetization 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 optimizer with just three epochs is used for MMR method. (c) Loss value over epoch number plotted for MMR results shown in (a) and (b).

formulation (see [42] for the details of boundary handling). The loss function for data regression is convex and is optimized 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 [79]. The error of the results of this are shown in Fig. 4(b) with associated loss in Fig. 4(c). As 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 simulation package. The resulting SVR system of equations to solve for this form of problem is as shown in Eq. (6). The results of this with the quantum kernel are shown in Fig. 4(a). The error of the results is shown in Fig. 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.

B. Linear DEs

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

$$\frac{df}{dx} = -\lambda\kappa f - \lambda \exp(-\lambda\kappa x) \sin(\lambda x), \quad (23)$$

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 (23) is $f_{\text{sol}}(x) = \exp(-\lambda\kappa x) \cos(\lambda x)$, a fading oscillatory function.

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 optimized via Newton's method. The error of the results is shown in Fig. 5(b) with corresponding loss in Fig. 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 optimization.

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

$$\begin{bmatrix} \hat{M} & \mathbf{h}_1^0 - \hat{D}\mathbf{h}_0^0 & \mathbf{g} \\ (\mathbf{h}_1^0 - \hat{D}\mathbf{h}_0^0)^T & \tilde{h}_0^0 & 1 \\ \mathbf{g}^T & 1 & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ b \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{r}} \\ \tilde{f}_0 \\ 0 \end{bmatrix}, \quad (24)$$

where the notation is as follows:

$$[\Omega_n^m]_{i,j} = \nabla_n^m \kappa(x_j, x_i), \quad (25)$$

$$\tilde{\Omega}_n^m = \Omega_n^m + I/\gamma, \quad (26)$$

$$[\mathbf{h}_n^m]_i = \nabla_n^m \kappa(x_0, x_i), \quad (27)$$

$$\tilde{h}_n^m = \kappa(x_0, x_0)_n^m, \quad (28)$$

$$\hat{D} = \text{diag}[\{g(x_i)\}_i], \quad (29)$$

$$[\tilde{\mathbf{g}}]_i = g(x_i), \quad (30)$$

$$[\tilde{\mathbf{r}}]_i = r(x_i), \quad (31)$$

$$\hat{M} = \Omega_1^1 - \Omega_1^0 \hat{D} - \hat{D} \Omega_0^1 + \hat{D} \tilde{\Omega}_0^0 \hat{D}. \quad (32)$$

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(b), with the result from using the quantum kernel shown explicitly in Fig. 5(a). Again it can be seen that both kernel types are able to closely approximate the considered function, though not as closely as

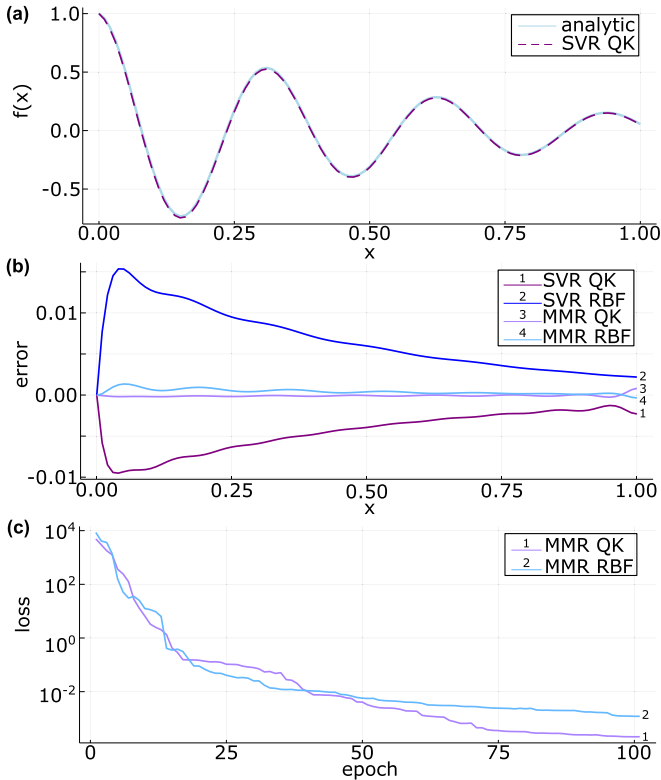


FIG. 5. MMR and SVR used to solve a linear differential equation (23). $\lambda = 20$, $\kappa = 0.1$, and $f(0) = 1$. (a) Solution via SVR method for quantum kernel with two layers and $N = 8$ shown by a dashed purple line. Known analytic solution plotted with a solid light blue line. (b) Error between results and analytic solution plotted over x as $[f(x) - f_{\text{sol}}(x)]/\text{range}(f_{\text{sol}})$. Error plotted for result shown in (a) and SVR method with classical RBF kernel with $\sigma = 0.2$. Plotted results from MMR method with same kernels are also considered. Newton optimizer with 100 epochs used for MMR method. (c) Loss value over epoch number plotted for MMR results shown in (a) and (b).

the MMR method, with the quantum kernel outperforming the classical kernel. In Appendix B we present the comparison of the results of solving this equation with a set of different kernel functions.

C. Solving nonlinear DEs

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 [61]

$$\frac{d^2 f(x)}{dx^2} = c \cos(ex) - af - bf^3, \quad (33)$$

where $f(x)$ is a solution in one dimension, a , b , c and e are constants. We note that as the problem in Eq. (33) is nonlinear, it is not guaranteed that the MMR method stated as an optimization problem is going 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 optimization methods. First, we consider $a = 0.5$, $c = 3$, $e = 3\pi$ and have b , which controls the

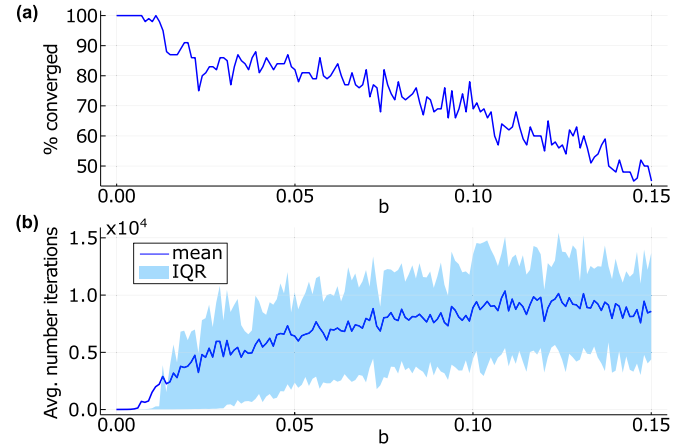


FIG. 6. Convergence behavior of learning solution to the undamped Duffing equation problem, $\frac{d^2 f}{dx^2} = 3 \cos(3\pi x) - 0.5f - bf^3$, with increasing nonlinearity controlled by b . In (a) we plot of percentage of runs which result in convergence (loss less than 10^{-10}) as a function of b . 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 (IQR) is shown with light blue shading.

nonlinear contribution, increasing from 0 to 0.15. This allows testing the convergence for optimization as the nonlinearity of the problem increases. We increase b in increments of 0.001. For each value we use an SVR method repeated 100 times to solve the problem with a different random set of initial variables.

We detail the hyperparameters and setup 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 13 uniformly separated points between zero and one with initial value condition of $y = 1$ and $dy/dx = 1$ at $x = 0$. The Newton method is chosen for optimization and γ is set as 10^5 . The simulation is implemented the same as for the linear case.

These results are shown in Fig. 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}_{tot} the total number of trials. We observe that as the degree of nonlinearity increases, the percentage of initialization states which result in convergence decreases, changing from 100% at $b = 0$ 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 optimization, and that additional training does not require extra quantum resources.

Figure 6 demonstrates one particular situation, and the convergence may be specific to the choice of setup. However, we find that the observed behavior of gradually decreasing convergence and increasing number of epochs at increasing nonlinearity is the general trend. Unless specific optimization routines are developed, kernel methods shall be more suited for problems with limited nonlinearity. We also stress that

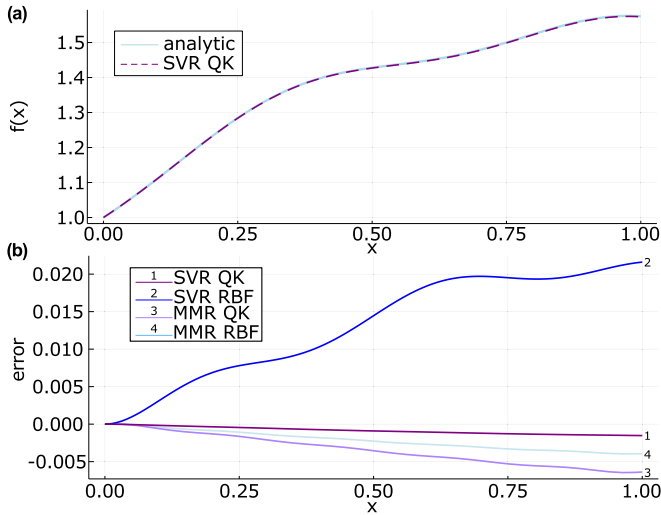


FIG. 7. Kernel-based solution and error for solving the instance of nonlinear Duffing equation [Eq. (33)]. We set $a = 0.5, b = 0.15, c = 3, e = 3\pi$. (a) Solution via SVR method for quantum kernel with two layers, $\phi(x) = qx/2$ and $N = 5$ shown by a dashed purple line. Known analytic solution plotted with a solid light blue line. (b) Error between results and analytic solution plotted over x as $[f(x) - f_{sol}(x)]/\text{range}(f_{sol})$. 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 optimizer used for MMR and SVR methods.

the procedure does lead to a close to perfect solution (we see abrupt decrease of loss), justifying the cost for rerunning the optimization. In the future we plan to investigate optimizers 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. 7. The quantum kernel used is as used in Fig. 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 13 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 optimization. The solution for the SVR method is shown in Fig. 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. 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 behavior, we consider that further work for optimal choice of kernel functions and hyperparameters is required.

VI. DISCUSSION AND CONCLUSION

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 optimized such that for many problems the optimization is convex, leading to fast convergence to the potential solution. Specifically, we propose two approaches, mixed model regression (MMR) and support vector regression (SVR), where optimization 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 optimization becomes nonconvex, thus requiring largely increased number of epochs. At the same time, we note that by kernelizing quantum models we modify the landscape of optimization. This raises the question of convergence difference between parameterized quantum circuits [80,81] and kernel models.

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 nonlinear equations are guaranteed to result in a convex loss landscape is another open question. Making use of a high-dimensional feature space without full tomography of the quantum wave function allows quantum kernel methods to potentially provide tangible advantage beyond classification.

We note that 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 [82]. 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. [82] 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.

Finally, let us discuss the comparison of quantum kernel-based approaches to solving DEs as compared to those based on differentiable quantum circuits [42], which in many ways reflect the difference between classical kernel methods and deep learning [53]. When considering the training landscape, kernel methods for regression and linear differential equation problems have a convex landscape and therefore the associated training guarantees, differentiable quantum circuit based methods do not have this feature. When considering the training stage, kernel methods require evaluating a Gram matrix of kernels with $O(|x|^2)$ points, increasing measurement

budget with the grid size $|\mathbf{x}|$ as compared to $O(|\mathbf{x}|)$ scaling of DQC evaluations for the loss function. At the same time, kernel methods do not rely on additional function evaluations, and optimization is straightforward for both linear and nonlinear problems. Deep learning instead needs to evaluate gradients at each iteration, leading to large overhead if the convergence is slow (for instance, when dealing with barren plateaus). This is a known trade-off for variational vs non-variational methods for ground state search [83]. However, for trainable QML circuits it may be beneficial to do the iterative training, if the number of points in a dataset $|\mathbf{x}|$ is large. When considering model evaluation (reading out solution of DEs), kernel methods in principle require evaluating Gram matrix once again for a different grid, adding another $O(|\mathbf{x}|^2)$ computational steps. At the same time, deep learning and DQC only require evaluating the trained model at points of interest. Finding the optimum strategy between the two methods thus becomes crucially dependent on the problem and available quantum hardware. We expect that future studies will shed light on cases where one or another is preferred, both contributing to the emergent field of quantum DE solvers.

APPENDIX A: FORMULATING AN SVR PROBLEM

Here we show how to formulate an SVR problem, providing more details for the specific example. We consider the case $\text{DE}(x, f, df/dx) = df/dx - g(x, f) = 0$ with initial condition $f(x_0) = f_0$. We use the model formulated as $f(x) = \mathbf{w}^\dagger \boldsymbol{\varphi}(x) + b$.

As a first step, the problem needs to be written in primal SVR model form,

$$\min_{\mathbf{w}, b} \mathbf{w}^\dagger \mathbf{w} + \gamma \mathbf{e}^T \mathbf{e} + \gamma \boldsymbol{\xi}^T \boldsymbol{\xi}, \quad (\text{A1})$$

$$\text{subject to } \mathbf{w}^T \boldsymbol{\varphi}'(x_i) - g(x_i, y_i) = e_i \quad i = 1:N, \quad (\text{A2})$$

$$\mathbf{w}^T \boldsymbol{\varphi}(x_0) + b = f_0, \quad (\text{A3})$$

$$y_i = \mathbf{w}^T \boldsymbol{\varphi}(x_i) + b + \xi_i \quad i = 1:N. \quad (\text{A4})$$

Here the minimization function is such that the magnitude of \mathbf{w} , \mathbf{e} , and $\boldsymbol{\xi}$ are minimized, with \mathbf{w} being the set of fitting coefficients. \mathbf{e} and $\boldsymbol{\xi}$ are the errors in the constraints. Minimizing 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 minimization function minus each of the constraints, preceded by a variable coefficient:

$$\mathcal{L} = \frac{1}{2} \mathbf{w}^T \mathbf{w} + \frac{\gamma}{2} \mathbf{e}^T \mathbf{e} + \frac{\gamma}{2} \boldsymbol{\xi}^T \boldsymbol{\xi} \quad (\text{A5})$$

$$- \sum_{i=1}^N \alpha_i [\mathbf{w}^T \boldsymbol{\varphi}'(x_i) - g(x_i, y_i) - e_i] \quad (\text{A6})$$

$$- \beta [\mathbf{w}^T \boldsymbol{\varphi}(x_0) + b - f_0] \quad (\text{A7})$$

$$- \sum_{i=1}^N \eta_i [\mathbf{w}^T \boldsymbol{\varphi}(x_i) + b + \xi_i - y_i]. \quad (\text{A8})$$

The introduced variables $\boldsymbol{\alpha}$, \mathbf{v} , and β are referred to as dual variables.

The next step is to calculate the KKT conditions. These are found by equating to zero the derivatives of the Lagrangian with respect to each of its variables, both primal and dual, (\mathbf{w} , b , \mathbf{e} , $\boldsymbol{\xi}$, \mathbf{y} , $\boldsymbol{\alpha}$, β , $\boldsymbol{\eta}$). The derivatives read

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = \mathbf{w} - \sum_i [\alpha_i \boldsymbol{\varphi}'(x_i) + \eta_i \boldsymbol{\varphi}(x_i)] - \beta \boldsymbol{\varphi}(x_0) = 0, \quad (\text{A9})$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\beta - \sum_i \eta_i = 0, \quad (\text{A10})$$

$$\frac{\partial \mathcal{L}}{\partial e_i} = \gamma e_i + \alpha_i = 0, \quad (\text{A11})$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = \gamma \xi_i - \eta_i = 0, \quad (\text{A12})$$

$$\frac{\partial \mathcal{L}}{\partial y_i} = \alpha_i \frac{\partial g}{\partial y}(x_i, y_i) + \eta_i = 0, \quad (\text{A13})$$

$$\frac{\partial \mathcal{L}}{\partial \alpha_i} = -[\mathbf{w}^\dagger \boldsymbol{\varphi}'(x_i) - g(x_i, y_i) - e_i] = 0, \quad (\text{A14})$$

$$\frac{\partial \mathcal{L}}{\partial \beta} = -[\mathbf{w}^\dagger \boldsymbol{\varphi}(x_0) + b - f_0] = 0, \quad (\text{A15})$$

$$\frac{\partial \mathcal{L}}{\partial v_i} = -[y_i - \mathbf{w}^\dagger \boldsymbol{\varphi}(x_i) - b - \xi_i] = 0. \quad (\text{A16})$$

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:

$$\left(\sum_j [\alpha_j \boldsymbol{\varphi}'(x_j) + v_j \boldsymbol{\varphi}(x_j)] + \beta \boldsymbol{\varphi}(x_0) \right)^\dagger \boldsymbol{\varphi}'(x_i) \quad (\text{A17})$$

$$-g(x_i, y_i) + \alpha_i/\gamma = 0, \quad (\text{A18})$$

$$\left(\sum_j [\alpha_j \boldsymbol{\varphi}'(x_j) + v_j \boldsymbol{\varphi}(x_j)] + \beta \boldsymbol{\varphi}(x_0) \right)^\dagger \boldsymbol{\varphi}(x_0) \quad (\text{A19})$$

$$+b - f_0 = 0, \quad (\text{A20})$$

$$- \left(\sum_j [\alpha_j \boldsymbol{\varphi}'(x_j) + v_j \boldsymbol{\varphi}(x_j)] + \beta \boldsymbol{\varphi}(x_0) \right)^\dagger \boldsymbol{\varphi}(x_i) \quad (\text{A21})$$

$$+y_i - b - \eta_i/\gamma = 0, \quad (\text{A22})$$

$$\sum_i \eta_i + \beta = 0, \quad (\text{A23})$$

$$\alpha_i \frac{\partial g}{\partial y}(x_i, y_i) + \eta_i = 0. \quad (\text{A24})$$

For these equations we then expand out the brackets and use the kernel trick, introducing the kernel function κ as $\kappa(x, y) = \boldsymbol{\varphi}^\dagger(x) \boldsymbol{\varphi}(y)$ and corresponding derivatives. We re-

member that this is a consequence of Mercer's 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

$$\begin{bmatrix} \tilde{\Omega}_1^1 & \Omega_0^1 & \mathbf{h}_0^1 & \mathbf{0} & \hat{0} \\ \Omega_1^0 & \tilde{\Omega}_0^0 & \mathbf{h}_0^0 & \mathbf{1} & -\hat{f} \\ \mathbf{h}_1^T & \mathbf{h}_0^T & \tilde{h} & 1 & \mathbf{0}^T \\ \mathbf{0}^T & \mathbf{1}^T & 1 & 0 & \mathbf{0}^T \\ \hat{D} & \hat{f} & \mathbf{0} & \mathbf{0} & \hat{0} \end{bmatrix} \begin{bmatrix} \alpha \\ \eta \\ \beta \\ b \\ y \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{g}} \\ \mathbf{0} \\ \tilde{f}_0 \\ 0 \\ \hat{0} \end{bmatrix}, \quad (\text{A25})$$

where the notation is as follows:

$$[\Omega_n^m]_{i,j} = \nabla_n^m \kappa(x_j, x_i), \quad (\text{A26})$$

$$\tilde{\Omega}_n^m = \Omega_n^m + \hat{f}/\gamma, \quad (\text{A27})$$

$$[\mathbf{h}_n^m]_i = \nabla_n^m \kappa(x_0, x_i), \quad (\text{A28})$$

$$\tilde{h} = \kappa(x_0, x_0), \quad (\text{A29})$$

$$\hat{D} = \text{diag}\left(\left\{\frac{\partial g}{\partial f}(x_i, y_i)\right\}_i\right), \quad (\text{A30})$$

$$[\tilde{\mathbf{g}}]_i = g(x_i, y_i). \quad (\text{A31})$$

We now have a set of nonlinear equations that can be solved for a set of variables, 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 Mercer's 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

$$f(x) = \sum_{i=1}^{|\mathbf{x}|} \alpha_i \nabla_1^0 \kappa(x_i, x) + \sum_{i=1}^{|\mathbf{x}|} \eta_i \kappa(x_i, x) + \beta \kappa(x_0, x) + b, \quad (\text{A32})$$

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.

APPENDIX B: KERNEL COMPARISON

We show how different quantum kernel functions perform in comparison to one another when solving the same problem with the same hyperparameters. We solve the linear differential equation presented in Eq. (23). Below we list the kernel functions we compare by label. The kernel functions are defined over $N = 8$ qubits with varying generators that include the following:

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

(2) 1L-tower: A single layered tower feature map with $|\psi(x)\rangle = \mathcal{U}(x)\mathcal{V}|0\rangle$ where \mathcal{V} is a HEA of depth five with

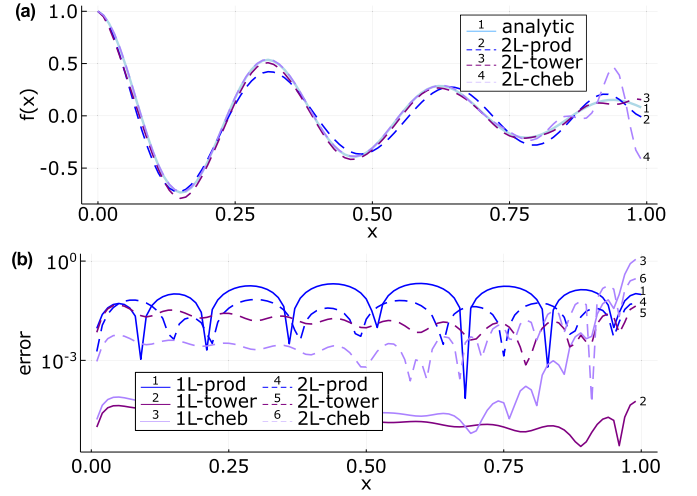


FIG. 8. Comparison of the results of solving (23) with different kernel functions (kernel functions described in above list). MMR method is used with $N = 8$, 21 training and evaluation points uniformly distributed between 0 and 0.99 and the Newton method is used for optimization. (a) Solutions from two layered kernels plotted with dashed lines. The known analytic solution is shown with a solid line. (b) The absolute normalized error of the solutions from different kernel functions plotted against x . The error is calculated as $\text{abs}\{[f(x) - f_{\text{sol}}(x)]/\text{range}(f_{\text{sol}})\}$.

randomized parameters which are set throughout training and $\mathcal{U}(x) = \prod_{j=1}^N R_X^j(jx)$.

(3) 1L-cheb: A single layered Chebyshev feature map with $|\psi(x)\rangle = \mathcal{U}(x)\mathcal{V}|0\rangle$ where \mathcal{V} is a HEA of depth five with randomized parameters which are set throughout training and $\mathcal{U}(x) = \prod_{j=1}^N R_X^j[\text{jarcos}(x)]$.

(4) 2L-prod: A two-layer product feature map with $|\psi(x)\rangle = \mathcal{U}(x)\mathcal{V}_2\mathcal{U}(x)\mathcal{V}_1|0\rangle$ where $\mathcal{V}_{1,2}$ are HEAs of depth five with randomized parameters which are set throughout training and $\mathcal{U}(x) = \prod_{j=1}^N R_X^j(x)$.

(5) 2L-tower: A two-layer tower feature map with $|\psi(x)\rangle = \mathcal{U}(x)\mathcal{V}_2\mathcal{U}(x)\mathcal{V}_1|0\rangle$ where $\mathcal{V}_{1,2}$ are HEAs of depth five with randomized parameters which are set throughout training and $\mathcal{U}(x) = \prod_{j=1}^N R_X^j(jx)$.

(6) 2L-cheb: A two-layer Chebyshev feature map with $|\psi(x)\rangle = \mathcal{U}(x)\mathcal{V}_2\mathcal{U}(x)\mathcal{V}_1|0\rangle$ where $\mathcal{V}_{1/2}$ are HEAs of depth five with randomized parameters which are set throughout training and $\mathcal{U}(x) = \prod_{j=1}^N R_X^j[\text{jarcos}(x)]$.

For describing the feature maps, we use terminology as is introduced in Ref. [42], 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. An MSE loss function is used with a boundary loss term, and the kernel functions used are described as above. The Newton method is used for optimization. The results are shown in Fig. 8. In Fig. 8(a) we can see that of the 2L kernel functions the Chebyshev kernel function performed the best over the majority of the region but oscillated near the boundary with one. This is a known

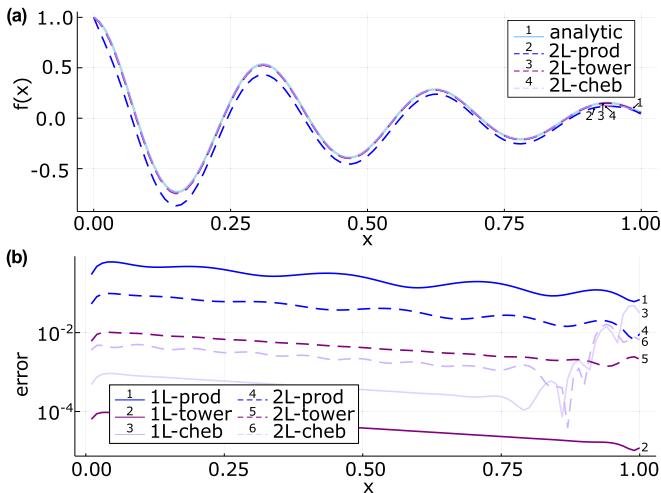


FIG. 9. Comparison of the results of solving (23) with different kernel functions (kernel functions described in above list). MVR method is used with $N = 8$, 21 training points uniformly distributed between 0 and 0.99 and $\gamma = 10^9$. (a) Solutions from two layered kernels plotted with dashed lines. The known analytic solution is shown with a solid line. (b) The absolute normalized error of the solutions from different kernel functions plotted against x . The error is calculated as $\text{abs}[\{f(x) - f_{\text{sol}}(x)\}/\text{range}(f_{\text{sol}})]$.

behavior 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. 8(b) we see the normalized absolute error of the results. We can see that the 1L kernel functions contain the best 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 behavior.

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. 9. In Fig. 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. 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 cannot 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.

- [1] G. F. Simmons, *Differential Equations with Applications and Historical Notes* (CRC Press, New York, 2016).
- [2] E. C. Zachmanoglou and D. W. Thoe, *Introduction to Partial Differential Equations with Applications* (Courier Corporation, University of Michigan, 1986).
- [3] G. D. Smith, *Numerical Solution of Partial Differential Equations: Finite Difference Methods* (Oxford University Press, Oxford, 1985).
- [4] J. P. Boyd, *Chebyshev and Fourier Spectral Methods* (Courier Corporation, Heidelberg, 2001).
- [5] C. Rackauckas and Q. Nie, *J. Open Res. Softw.* **5**, 15 (2017).
- [6] C. Rackauckas, M. Innes, Y. Ma, J. Bettencourt, L. White, and V. Dixit, [arXiv:1902.02376](https://arxiv.org/abs/1902.02376).
- [7] S. Cai, Z. Mao, Z. Wang, M. Yin, and G. E. Karniadakis, *Acta Mech. Sinica* **37**, 1727 (2021).
- [8] P. W. Shor, in *Proceedings 35th Annual Symposium on Foundations of Computer Science* (IEEE, 1994), pp. 124–134.
- [9] A. W. Harrow, A. Hassidim, and S. Lloyd, *Phys. Rev. Lett.* **103**, 150502 (2009).
- [10] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell *et al.*, *Nature (London)* **574**, 505 (2019).
- [11] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information* (Cambridge University Press, Cambridge, 2000).
- [12] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, *Nature (London)* **549**, 195 (2017).
- [13] S. K. Leyton and T. J. Osborne, [arXiv:0812.4423](https://arxiv.org/abs/0812.4423).
- [14] D. W. Berry, A. M. Childs, A. Ostrander, and G. Wang, *Commun. Math. Phys.* **356**, 1057 (2017).
- [15] S. Lloyd, G. D. Palma, C. Gokler, B. Kiani, Z.-W. Liu, M. Marvian, F. Tennie, and T. Palmer, [arXiv:2011.06571](https://arxiv.org/abs/2011.06571).
- [16] J.-P. Liu, H. Øie Kolden, H. K. Krovi, N. F. Loureiro, K. Trivisa, and A. M. Childs, *Proc. Natl. Acad. Sci. USA* **118**, e2026805118 (2021).
- [17] S. Jin and N. Liu, [arXiv:2202.07834](https://arxiv.org/abs/2202.07834).
- [18] N. Linden, A. Montanaro, and C. Shao, *Commun. Math. Phys.* **395**, 601 (2022).
- [19] A. Scherer, B. Valiron, S.-C. Mau, S. Alexander, E. van den Berg, and T. E. Chapuran, *Quant. Info. Proc.* **16**, 60 (2017).
- [20] K. Bharti, A. Cervera-Lierta, T. H. Kyaw, T. Haug, S. Alperin-Lea, A. Anand, M. Degroote, H. Heimonen, J. S. Kottmann, T. Menke *et al.*, *Rev. Mod. Phys.* **94**, 015004 (2022).
- [21] M. Cerezo, A. Arrasmith, R. Babbush, S. C. Benjamin, S. Endo, K. Fujii, J. R. McClean, K. Mitarai, X. Yuan, L. Cincio *et al.*, *Nat. Rev. Phys.* **3**, 625 (2021).
- [22] M. Benedetti, E. Lloyd, S. Sack, and M. Fiorentini, *Quantum Sci. Technol.* **4**, 043001 (2019).
- [23] A. Perdomo-Ortiz, M. Benedetti, J. Realpe-Gómez, and R. Biswas, *Quantum Sci. Technol.* **3**, 030502 (2018).
- [24] M. Schuld and N. Killoran, *Phys. Rev. Lett.* **122**, 040504 (2019).
- [25] K. Mitarai, M. Negoro, M. Kitagawa, and K. Fujii, *Phys. Rev. A* **98**, 032309 (2018).

- [26] J.-G. Liu and L. Wang, *Phys. Rev. A* **98**, 062324 (2018).
- [27] C. Zoufal, A. Lucchi, and S. Woerner, *npj Quantum Inf.* **5**, 103 (2019).
- [28] B. Coyle, D. Mills, V. Danos, and E. Kashefi, *npj Quantum Inf.* **6**, 60 (2020).
- [29] A. Abbas, D. Sutter, C. Zoufal, A. Lucchi, A. Figalli, and S. Woerner, *Nat. Comput. Sci.* **1**, 403 (2021).
- [30] Y. Du, M.-H. Hsieh, T. Liu, S. You, and D. Tao, *PRX Quantum* **2**, 040337 (2021).
- [31] H.-L. Huang, Y. Du, M. Gong, Y. Zhao, Y. Wu, C. Wang, S. Li, F. Liang, J. Lin, Y. Xu *et al.*, *Phys. Rev. Appl.* **16**, 024051 (2021).
- [32] S. Y.-C. Chen, C.-H. H. Yang, J. Qi, P.-Y. Chen, X. Ma, and H.-S. Goan, *IEEE Access* **8**, 141007 (2020).
- [33] S. L. Wu, J. Chan, W. Guan, S. Sun, A. Wang, C. Zhou, M. Livny, F. Carminati, A. D. Meglio, A. C. Y. Li *et al.*, *J. Phys. G: Nucl. Part. Phys.* **48**, 125003 (2021).
- [34] S. Y.-C. Chen and S. Yoo, *Entropy* **23**, 460 (2021).
- [35] S. Endo, J. Sun, Y. Li, S. C. Benjamin, and X. Yuan, *Phys. Rev. Lett.* **125**, 010501 (2020).
- [36] C. Cîrstoiu, Z. Holmes, J. Iosue, L. Cincio, P. J. Coles, and A. Sornborger, *npj Quantum Inf.* **6**, 82 (2020).
- [37] X. Xu, J. Sun, S. Endo, Y. Li, S. C. Benjamin, and X. Yuan, *Sci. Bull.* **66**, 2181 (2021).
- [38] C. Bravo-Prieto, R. LaRose, M. Cerezo, Y. Subasi, L. Cincio, and P. J. Coles, *arXiv:1909.05820*.
- [39] C.-C. Chen, S.-Y. Shiau, M.-F. Wu, and Y.-R. Wu, *Sci. Rep.* **9**, 16251 (2019).
- [40] M. Lubasch, J. Joo, P. Moinier, M. Kiffner, and D. Jaksch, *Phys. Rev. A* **101**, 010301(R) (2020).
- [41] H.-L. Liu, Y.-S. Wu, L.-C. Wan, S.-J. Pan, S.-J. Qin, F. Gao, and Q.-Y. Wen, *Phys. Rev. A* **104**, 022418 (2021).
- [42] O. Kyriienko, A. E. Paine, and V. E. Elfving, *Phys. Rev. A* **103**, 052416 (2021).
- [43] M. Schuld, R. Sweke, and J. J. Meyer, *Phys. Rev. A* **103**, 032430 (2021).
- [44] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran, *Phys. Rev. A* **99**, 032331 (2019).
- [45] M. Knudsen and C. B. Mendl, *arXiv:2012.12220*.
- [46] A. E. Paine, V. E. Elfving, and O. Kyriienko, *arXiv:2108.03190*.
- [47] J. Romero and A. Aspuru-Guzik, *Adv. Quantum Technol.* **4**, 2000003 (2021).
- [48] O. Kyriienko, A. E. Paine, and V. E. Elfving, *arXiv:2202.08253*.
- [49] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, *Nature (London)* **567**, 209 (2019).
- [50] N. Ardehshir, C. Sanford, and D. J. Hsu, *Adv. Neural Info. Proc. Syst.* **34**, 4907 (2021).
- [51] S. Y. Kung, *Kernel Methods and Machine Learning* (Cambridge University Press, Cambridge, 2014).
- [52] H.-Y. Huang, M. Broughton, M. Mohseni, R. Babbush, S. Boixo, H. Neven, and J. R. McClean, *Nat. Commun.* **12**, 2631 (2021).
- [53] M. Schuld, *arXiv:2101.11020*.
- [54] R. Mengoni and A. Di Pierro, *Quantum Mach. Intel.* **1**, 65 (2019).
- [55] Z. Li, X. Liu, N. Xu, and J. Du, *Phys. Rev. Lett.* **114**, 140504 (2015).
- [56] A patent application for the method described in this paper has been submitted by PASQAL.
- [57] L. Wang, *Support Vector Machines: Theory and Applications*, Vol. 177, Studies in Fuzziness and Soft Computing (Springer Science & Business Media, 2005).
- [58] S. Mehrkanoon, T. Falck, and J. A. Suykens, *IEEE Trans. Neural Netw. Learn. Syst.* **23**, 1356 (2012).
- [59] S. Mehrkanoon and J. A. Suykens, *Neurocomputing* **159**, 105 (2015).
- [60] Y. Lu, Q. Yin, H. Li, H. Sun, Y. Yang, and M. Hou, *J. Ind. Manage. Optimiz.* **16**, 1481 (2020).
- [61] J. M. T. Thompson and H. B. Stewart, *Nonlinear Dynamics and Chaos*, 2nd ed. (Wiley, 2002).
- [62] J. Mercer, *Phil. Trans. R. Soc. Lond. A* **209**, 415 (1909).
- [63] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization* (Cambridge University Press, Cambridge, 2004).
- [64] H. W. Kuhn and A. W. Tucker, in *Traces and Emergence of Nonlinear Programming* (Springer, 2014), pp. 247–258.
- [65] A. Pérez-Salinas, A. Cervera-Lierta, E. Gil-Fuster, and J. I. Latorre, *Quantum* **4**, 226 (2020).
- [66] M. C. Caro, E. Gil-Fuster, J. J. Meyer, J. Eisert, and R. Sweke, *Quantum* **5**, 582 (2021).
- [67] H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf, *Phys. Rev. Lett.* **87**, 167902 (2001).
- [68] O. Higgott, D. Wang, and S. Brierley, *Quantum* **3**, 156 (2019).
- [69] K. Mitarai and K. Fujii, *Phys. Rev. Res.* **1**, 013006 (2019).
- [70] O. Kyriienko and V. E. Elfving, *Phys. Rev. A* **104**, 052417 (2021).
- [71] D. Wierichs, J. Izaac, C. Wang, and C. Y.-Y. Lin, *Quantum* **6**, 677 (2022).
- [72] A. F. Izmaylov, R. A. Lang, and T.-C. Yen, *Phys. Rev. A* **104**, 062443 (2021).
- [73] J. G. Vidal and D. O. Theis, *arXiv:1812.06323*.
- [74] D. O. Theis, *arXiv:2112.14669*.
- [75] L. Savary and L. Balents, *Rep. Prog. Phys.* **80**, 016502 (2017).
- [76] M. Hermanns, I. Kimchi, and J. Knolle, *Annu. Rev. Condens. Matter Phys.* **9**, 17 (2018).
- [77] T. A. Bespalova and O. Kyriienko, *arXiv:2109.13883*.
- [78] H.-Y. Huang, M. Broughton, J. Cotler, S. Chen, J. Li, M. Mohseni, H. Neven, R. Babbush, R. Kueng, J. Preskill *et al.*, *Science* **376**, 1182 (2022).
- [79] X.-Z. Luo, J.-G. Liu, P. Zhang, and L. Wang, *Quantum* **4**, 341 (2022).
- [80] M. Larocca, N. Ju, D. García-Martín, P. J. Coles, and M. Cerezo, *arXiv:2109.11676*.
- [81] M. Larocca, P. Czarnik, K. Sharma, G. Muraleedharan, P. J. Coles, and M. Cerezo, *Quantum* **6**, 824 (2022).
- [82] S. Thanasilp, S. Wang, M. Cerezo, and Z. Holmes, *arXiv:2208.11060*.
- [83] T. A. Bespalova and O. Kyriienko, *PRX Quantum* **2**, 030318 (2021).