IOP Publishing Quantum Sci. Technol. 8 (2023) 015021 https://doi.org/10.1088/2058-9565/acaa51

Quantum Science and Technology

OPEN ACCESS

RECEIVED
7 July 2022

REVISED
3 October 2022

ACCEPTED FOR PUBLICATION
9 December 2022

PUBLISHED
21 December 2022

PAPER

Qade: solving differential equations on quantum annealers

Juan Carlos Criado[1,2,*] and Michael Spannowsky[1,2]

[1] Institute for Particle Physics Phenomenology, Durham University, Durham DH1 3LE, United Kingdom
[2] Department of Physics, Durham University, Durham DH1 3LE, United Kingdom
[*] Author to whom any correspondence should be addressed.

E-mail: juan.c.criado@durham.ac.uk

Keywords: quantum annealing, differential equations, quantum adiabatic computation

Abstract

We present a general method, called Qade, for solving differential equations using a quantum annealer. One of the main advantages of this method is its flexibility and reliability. On current devices, Qade can solve systems of coupled partial differential equations that depend linearly on the solution and its derivatives, with non-linear variable coefficients and arbitrary inhomogeneous terms. We test this through several examples that we implement in state-of-the-art quantum annealers. The examples include a partial differential equation and a system of coupled equations. This is the first time that equations of these types have been solved in such devices. We find that the solution can be obtained accurately for problems requiring a small enough function basis. We provide a Python package implementing the method at gitlab.com/jccriado/qade.

1. Introduction

The development of methods for the solution of differential equations is fundamental for the mathematical modeling of most systems in nature. Methods based on machine learning techniques, first proposed in references [1–4], have gained increasing attention for their versatility in recent years [5–17]. At their core, these methods reformulate the task of solving differential equations as an optimization problem, for which the existing machine learning frameworks are designed [18–27].

Quantum annealing devices [28–40] are particularly well-suited for optimization tasks, as the computation they perform is directly the minimization of their Hamiltonian, which the user can specify. It has been shown that quantum annealers can find the global minimum of the target function more efficiently than classical alternatives in some problems [41–44].

Most quantum annealers are based on a continuous-time evolution using a transverse Ising model Hamiltonian. The specific form of such Hamiltonian requires a dedicated encoding strategy for the problem at hand, i.e. here for the solution of differential equations. A general method for approximately encoding arbitrary target functions with the compact domain as the Hamiltonian of a quantum annealer has been introduced in [44]. The size of the problems that can be solved in the current devices is limited by the number of available qubits and connections between them. To overcome this limitation, one can solve a series of coarse-grained approximations to the problem, each of which require a lower number of qubits, while iteratively improving the precision of the solution, as proposed in [45].

We apply these quantum optimization techniques to the solution of differential equations in the machine learning-oriented formulation described in references [10, 27]. Previous implementations of other methods in quantum computers have been applied successfully to solving differential equations in references [46–49]. In references [46, 49] a variational quantum circuit is used to parametrize the potential solutions. This provides an expressive non-linear ansatz for the solutions. The parameters of this ansatz are then found using a classical method. In references [47, 48], the solution is found using a purely quantum algorithm instead, using either digital quantum computers or quantum annealers.

In this work, we focus on this second possibility; that is, that the solution to the differential equations is directly provided by the quantum computation, with no classical algorithm involved. The main advantage of

our method, which we call *Qade*, is its generality: it does not make assumptions about the equations or boundary conditions beyond the requirement that they depend linearly on the solutions and their derivatives, which is required in order for the corresponding Hamiltonian to be that of an Ising model, and thus embeddable in publicly accessible quantum annealing devices, e.g. devices from D-Wave. This means that systems of coupled linear partial differential equations of any order, with variable coefficients and arbitrary inhomogeneous terms, can be handled by this approach. This allows it to go beyond the method proposed in [47], which deals with functions of a single variable; and the one in [48], which requires that the equations arise as the Euler–Lagrange equations of some functions, and that the basis functions satisfy the boundary conditions. Some examples of equations that fit in the Qade framework are: any of the Sturm–Liouville equations, Maxwell's equations, the Schrödinger equation or the inhomogeneous wave or heat equations.

We provide a Python package implementing Qade in full generality. This package contains the tools for obtaining the quantum annealing formulation of linear differential equations. Furthermore, users with access to the cloud interface to the D-Wave quantum annealers can also perform the necessary annealing runs and decode the results into the final solution.

The rest of this paper is organized as follows. In section 2, we briefly introduce the quantum annealing framework. In section 3, we present the Qade method, which reformulates the task of solving differential equations as a problem directly solvable by quantum annealers. Examples of application of Qade to 3 differential equations are shown in section 4. We summarize our conclusions in section 5. Finally, the accompanying Python package is introduced in appendix.

2. Quantum annealing

In the quantum annealing paradigm, computations are encoded as finding the ground state of an Ising model Hamiltonian

$$H(\sigma) = \sum_{ij} \sigma_i J_{ij} \sigma_j + \sum_i h_i \sigma_i, \tag{1}$$

for a collection of N spin variables $\sigma_i = \pm 1$. To perform a quantum annealing calculation, one must then find a way to reduce the problem at hand to the minimization of H. In section 3, we describe how to obtain the J_{ij} and h_i parameters corresponding to any system of linear partial differential equations.

We now review how H is minimized in a quantum annealing device. Internally, the device has access to a quantum system that it partially controlled. The system is described by a Hilbert space constructed as the tensor product of N 1-qubit spaces $\mathbb{C}^2$ and a Hamiltonian

$$\begin{aligned} H_{\text{quantum}} = A(s) \sum_i \sigma_{ix} \\ + B(s) \left[\sum_{ij} \sigma_{iz} J_{ij} \sigma_{jz} + \sum_i h_i \sigma_{iz} \right], \end{aligned} \tag{2}$$

with σ_{ia} the ath Pauli matrix applied to the ith qubit, and $A(1) = B(0) = 0$. The annealer can set the values of J_{ij} and h_i, prepare the system in the ground state of H_{quantum} at $s = 0$, change the value of s continuously, and measure the observable $\bigotimes_i \sigma_{iz}$ at the end of the annealing process, when $s = 1$.

The dependence $s(t)$ of the s parameter with time t is referred to as an schedule. In the D-Wave devices we use in the examples in this work, a typical schedule is a monotonically increasing from function $s = 0$ at the initial time to $s = 1$ at the final time, which, depending on the application, can vary between a few μs to about 1 ms. A pause in the increase of $s(t)$ for some time or an increase in its slope towards the end of the run is commonly used. When an appropriate schedule is selected, the final measurement of the annealer is expected to return the ground state of H. Specifically, the adiabatic theorem ensures that the ground state is found with a high probability if process is sufficiently slow [50].

In real-world devices, the number of couplings J_{ij} that can be set to a non-zero value is relatively low, since each qubit is physically connected only to a few others. Abstract systems with a higher degree of connectivity can be embedded in the physical device by chaining qubits with a large negative coupling, so that they are forced to take the same value. Each spin in the abstract system is then represented by a qubit chain in the device, which can have a larger number of non-vanishing connections. The calculation of the embedding corresponding to a given set of values of J_{ij} and h_i is computationally expensive. Thus, once it has been obtained, it is commonly re-used in several runs of the annealing process, which are performed to reduce noise. The final state with the minimal energy is selected as the final solution.

3. Method

We now present Qade, our quantum annealing-based method for solving differential equations. We denote the equations to be solved as

$$E_i(x)[f]|_{x\in\mathcal{X}_i} = 0, \tag{3}$$

for a function $f : \mathbb{R}^{n_{\text{in}}} \to \mathbb{R}^{n_{\text{out}}}$, where the $E_i(x)$ are local functionals of f (i.e. they only depend on the value of f and its derivatives at x), and the $\mathcal{X}_i$ are the domains in which the equations must be satisfied. Initial and boundary conditions can be viewed as a particular case of these equations, in which $\mathcal{X}_i$ is the initial or boundary set of x values. We impose that all the equations are linear functions of f and its derivatives:

$$E_i(x)[f] = \sum_{kn} C^{(k)}_{\text{in}}(x) \cdot (\partial^k f_n(x)) + B_i(x), \tag{4}$$

where the $B_i(x)$ are the inhomogeneous terms, while, the $C^{(k)}_{\text{in}}(x)$ are the variable coefficients of the derivatives, and $\partial^k f_n$ is a vector containing all the partial derivatives of order k of f_n.

As explained in section 2, to solve equation (4) in a quantum annealer, it has to be encoded as the ground state of an Ising model Hamiltonian. We first reformulate it as a minimization problem. Following the machine learning-oriented methods described in references [10, 27], we discretize the domains into finite subsets of sample points $X_i \subset \mathcal{X}_i$, and define the loss function

$$L[f] = \sum_i \sum_{x\in X_i} (E_i(x)[f])^2. \tag{5}$$

There is a certain amount of arbitrariness associated with the definition of $L[f]$: instead of the equations $E_i(x)[f]$, one can use any complete and linearly-independent set of combinations of them. A particular case of this is the weighting of each equation $E_i(x)[f]$ by a different constant factor. Although the system of equations to be solved is equivalent, such a weighting can be useful in practical applications. For example, one can increase the weight for boundary conditions when a first attempt at the solution violates them. We will use this in the example in section 4.3.

The global minimum $L[f_{\text{sol}}] = 0$ is attained if and only if all the equations are satisfied at all the sample points in the X_i sets. Now, we parametrize the function f as a linear combination of a finite set of 'basis' functions Φ_m, as

$$f_n(x) = \sum_m w_{nm}\Phi_m(x). \tag{6}$$

Then, the equations can be written as linear functions of a finite set of parameters, the weights w_{nm}:

$$E_i(x, w) = \sum_{nm} H_{\text{in}}(x)[\Phi_m]\, w_{nm} + B_i(x), \tag{7}$$

$$H_{\text{in}}(x)[\Phi] = \sum_k C^{(k)}_{\text{in}}(x) \cdot (\partial^k \Phi(x)), \tag{8}$$

and the L becomes a quadratic function of them:

$$L(w) = \sum_{nmpq} w_{nm} J_{nm,pq}\, w_{pq} + \sum_{nm} h_{nm}\, w_{nm}, \tag{9}$$

where

$$J_{nm,pq} = \sum_i \sum_{x\in X_i} H_{\text{in}}(x)[\Phi_m]\, H_{ip}(x)[\Phi_q], \tag{10}$$

$$h_{nm} = 2\sum_i \sum_{x\in X_i} H_{\text{in}}(x)[\Phi_m] B_i(x). \tag{11}$$

The final step in converting L into an Ising model Hamiltonian is the binary encoding of each weight in terms of spin variables $\hat{w}^{(\alpha)}_{nm} = \pm 1$, as

$$w_{nm} = c_{nm} + s_{nm} \sum_{\alpha=1}^{n_{\text{spins}}} \frac{\hat{w}^{(\alpha)}_{nm}}{2^\alpha}, \tag{12}$$

Table 1. Hyperparameters of the method presented in section 3, with example values.

	Example value	Description	Definition
annealing	$n_{\text{reads}} = 200$	number of reads	section 2
	$s(t) = t/(200\mu s)$	quantum annealing schedule	
encoding	$n_{\text{spins}} = 3$	number of spins per weight	equation (12)
	$c_{mn} = 0$	(initial) central values of the weights	
	$s_{mn} = 1$	(initial) scales of the weights	
general	$\Phi_m(x) = x^m$	basis of functions	equation (6)
	$n_{\text{epochs}} = 10$	number of epochs in the iterative procedure	equation (17)
	$S = 1/2$	scale factor to update s_{nm} in each epoch	

with the free parameter c_{mn} and s_{nm} being the center values of the w_{nm}, and the scales by which the can change within the encoding, respectively. Replacing this expression into equation (9), we finally get the Ising model

$$H(\hat{w}) := L(w) = \sum_{nmpq\alpha\beta} \hat{w}_{nm}^{(\alpha)} \hat{J}_{nm,pq}^{(\alpha\beta)} \hat{w}_{pq}^{(\beta)} + \sum_{nm\alpha} \hat{h}_{nm}^{(\alpha)} \hat{w}_{nm}^{(\alpha)}, \tag{13}$$

where

$$\hat{J}_{nm,pq}^{(\alpha\beta)} = 2^{-(\alpha+\beta)} s_{nm} s_{pq} J_{nm,pq}, \tag{14}$$

$$\hat{h}_{nm}^{(\alpha)} = 2^{-\alpha} s_{nm} (h_{nm} + 2c_{pq} J_{nm,pq}). \tag{15}$$

The original problem can then be solved by minimizing H in a quantum annealing device. The solution is recovered by decoding the weights using equation (12), and substituting them in equation (6). The total number of spins in the Ising model is controlled number of spins per weight n_{spins}, the number n_{basis} of functions in the basis Φ_m, and the number n_{out} of functions f_i to be solved for:

$$N = n_{\text{spins}} \times n_{\text{basis}} \times n_{\text{out}}. \tag{16}$$

The size of an Ising model embedded in a current quantum annealer is limited, both in the allowed number of spins N and the number of connections between them. This means that not many spins per weight n_{spin} can be currently used, which implies that each weight can only be determined up to a low precision $2^{-n_{\text{spin}}}$ in a single quantum annealing run. To improve the accuracy of the results, we use a version of the iterative algorithm proposed in [45]. In each iteration I, which we call an *epoch*, the annealer is run for the model defined by setting the centers to the values of the weights obtained in the previous iteration, while all the s_{nm} are scaled by a factor $0 < S \leqslant 1$:

$$c_{nm}^I = w_{nm}^{I-1}, \quad s_{nm}^I = S s_{nm}^{I-1}, \quad I = 0, \ldots, n_{\text{epochs}}. \tag{17}$$

We remark that the use of $n_{\text{epochs}} > 1$ is due to the limited number of qubits and connections available in the physical device being used. A potential problem when $n_{\text{epochs}} > 1$ is that, if the wrong values of the parameters are chosen in the first few epochs, this cannot be corrected in later epochs, when the exponentially-decaying scales s_{mn} have decreased significantly. This can be mitigated by choosing a scale factor S closer to 1, so that s_{mn} decreases slowly, and successive epochs have the chance of correcting errors in the previous ones. In future annealers with a larger size, one might be able to set a larger n_{spins} and $n_{\text{epochs}} = 1$, so that the solution is obtained in one annealing step, eliminating these problems and taking full advantage of the quantum computation.

The method we have presented contains several *hyperparameters* that need to be adjusted to suitable values before application to a concrete problem. We collect them in table 1, together with examples of the typical values they might take to solve differential equations with current quantum annealing devices. To choose a correct set of values for the hyperparameters, one might use domain knowledge about the problem to be solved, such as which basis of functions is most suited or what is the typical size that the corresponding weights might have. When this knowledge is not available, the process involves some trial and error, using the value of loss function $L(w)$ as a measure of the goodness of the solution.

4. Results and discussion

This section illustrates how to use the Qade method proposed in this paper to solve different kinds of differential equations. First, we solve equations whose solutions are known analytically to be able to compare them to the numerical results: the Laguerre equation, as an example of a single ordinary differential equation with variable coefficients; the wave equation, as an example of a partial differential equation; and an example of a first-order system of coupled differential equations. The code for these examples is available at gitlab.com/jccriado/qade/-/tree/main/examples.

4.1. Laguerre equation

The Laguerre equation is given by

$$xy'' + (1-x)y' + ny = 0. \tag{18}$$

Its solutions are the Laguerre polynomials $L_n(x)$. We thus impose the boundary conditions

$$y(0) = 1, \qquad y(1) = L_n(1). \tag{19}$$

We look for a solution of the form $y = w_m x^m$, with $0 \leqslant and m < 4$. We employ the Ising model formulation outlined in section 3 to find the weights w_m using D-Wave's `Advantage_system4.1`. Since the weights w_m are expected to grow for increasing n, we pick the scales $s_m = n - 2$. We also find that setting a high $n_{\text{reads}} = 500$ gives more consistent results. The rest of the hyperparameters are set to the values in table 1. The corresponding Ising model has $N = 12$ spins.

We show our results for $n = 3, 4, 5, 6$ in figure 1, together with the analytical solution. The loss function for all of them is below 2×10^{-2}. The distribution of the energies for the final states of the 500 reads performed in each epoch is presented in the left panel of figure 2. The graph shows that most of the final states concentrate on the bottom part of the distribution. The iterative procedure produces distributions that concentrate on a smaller range of energies for larger epochs.

4.2. Wave equation

The wave equation is

$$\frac{\partial^2 \phi}{\partial x^2} - \frac{\partial^2 \phi}{\partial t^2} = 0. \tag{20}$$

For the initial and boundary conditions, we pick

$$\phi(x, 0) = \sin(2\pi x), \qquad \left.\frac{\partial \phi}{\partial t}\right|_{t=0} = \pi \cos(2\pi x), \tag{21}$$

$$\phi(0, t) = \phi(1, t) = \sin(2\pi t)/2. \tag{22}$$

We use the natural choice of basis for the solution of this problem, which is the Fourier basis. Since the input space is two-dimensional ($n_{\text{in}} = 2$), the basis is given as all products $\phi_{m_1}(x)\phi_{m_2}(t)$, with $\phi_0(z) = 1$, $\phi_1(z) = \sin(2\pi z)$, etc. We set the number of ϕ_{m_i} functions per input component to $d = 3$, so the total dimension of the basis is 9. In order to reduce the number of qubits required for the encoding, we pick $n_{\text{spins}} = 2$, with the rest of hyperparameters set to the values in table 1. We get a total of $N = 18$ spins in the Ising model.

We obtain a solution with a loss of $L = 2 \times 10^{-1}$. We present it in figure 1, together with the analytical solution, which is given by

$$\begin{aligned}\phi_{\text{true}}(x, t) = &\cos(2\pi x)\sin(2\pi t) \\ &+ \sin(2\pi x)\cos(2\pi t)/2.\end{aligned} \tag{23}$$

The quartiles for the energies of the final states from the 100 reads from each epoch are shown in the center panel of figure 2. As compared to the distribution of energies for the Laguerre equation, we find that they are even more concentrated towards low energies. The minimum of the distribution decreases significantly for the first few epochs. We attribute both effects to the fact that we have used $n_{\text{spins}} = 2$, which leads to a lower precision in the results. This might make it easier for the annealer to find the optimal solution, but the low precision will limit the goodness of the solution in the early epochs.

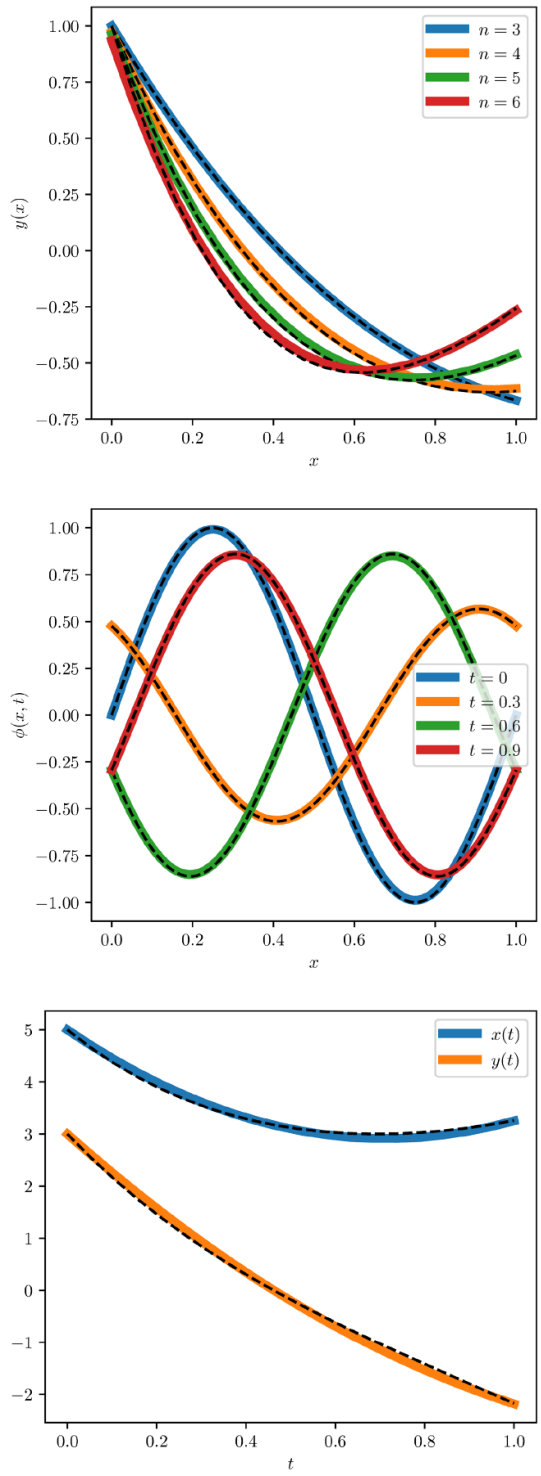

Figure 1. Solutions obtained with `Qade` to the Laguerre equation (top, see section 4.1), wave equation (center, see section 4.2) and a system of first-order coupled equations (bottom, see section 4.3). The dashed lines show the analytical solutions.

4.3. Coupled first-order equations

As an example of a system of coupled equations, we solve

$$2x' + x + 3y = 0, \quad 2y' + 3x + y = 0, \tag{24}$$

$$x(0) = 5, \quad y(0) = 3. \tag{25}$$

In the $t \in [0, 1]$ interval. We use a monomial basis $x(t) = w_{0m}t^m$, $y(t) = w_{1m}t^m$. We pick the value $s_{nm} = 4$ for the scales of the binary encoding of the w_{nm} weights, to account for the relatively large values of x and y. We set all the other hyperparameters to the values in table 1. The resulting Ising model has $N = 18$ spins.

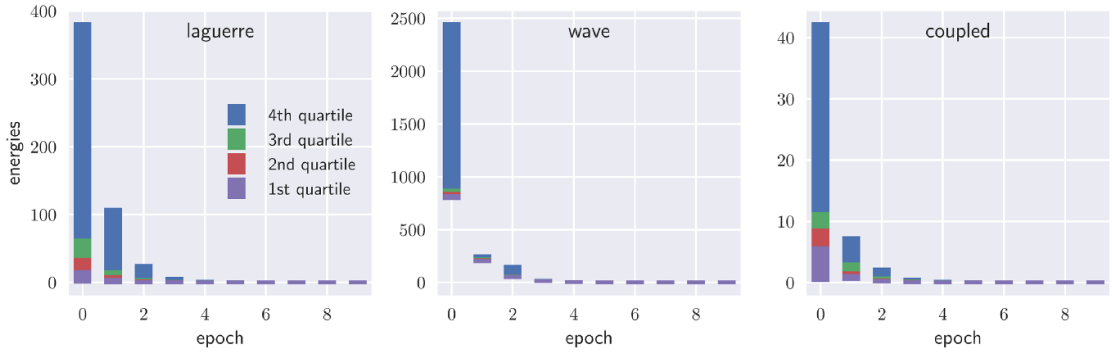

Figure 2. Distribution of energies of the final states from the different reads as a function of the epoch, for the solution of the Laguerre equation (left), the wave equation (center) and the system of coupled equations (right) presented in sections 4.1–4.3, respectively. The four sections in different colors of the block displayed for each epoch show the four quartiles of the energy distribution.

Finally, we find that we need to increase their relative importance in the loss function for the initial conditions to be satisfied. We do so by multiplying the equations (and not the initial conditions) by a factor of $1/10$.

We obtain a solution with a value of the loss of 2×10^{-2}, and present it in figure 1, together with the analytical solution:

$$x_{\text{true}}(t) = \mathrm{e}^{t} + 4\mathrm{e}^{-2t}, \quad y_{\text{true}}(t) = -\mathrm{e}^{t} + 4\mathrm{e}^{-2t}. \tag{26}$$

The distributions of the energies of the final states for the 200 reads in each epoch is displayed in the right panel of figure 2. They are similar to the ones found for the Laguerre equation, with a concentration of states with low energies, and a quick reduction in the range of energies for larger epochs.

4.4. Comparison to classical

In order to compare the performance of Qade against a classical algorithm, we implement the three examples in this section in the state-of-the-art classical solver Elvet [27]. Elvet is based on machine learning methods similar to Qade. A trial function (usually a neural network) with adjustable parameters is proposed. The optimal values of these parameters are obtained by minimizing the loss function equation (5), using an stochastic gradient descent algorithm.

We use a neural network with a single hidden layer to solve each of the examples, with 10, 30 and 5 hidden nodes for the Laguerre, wave and coupled equations, respectively. To find the solution, we train the network using the Adam optimizer [51] with 0.01 learning rate, for 10 000 epochs. An epoch in this context is an step of the gradient descent algorithm, in which all points in the discretized domain are taken into account. The results are shown in figure 3. We show the evolution of the loss function (normalized by the number of points N in the discretized domain) as function of the number of epochs, both for Qade and Elvet. As an alternative measure of the goodness of the solution, we also display the evolution of the mean squared error (MSE) between the approximate solution $f(x)$ and the true one $f_{\text{true}}(x)$:

$$\text{MSE} = \frac{1}{N}\sum_{n,x}\left(f(x) - f_{\text{true}}(x)\right)^2. \tag{27}$$

For the Laguerre equation, Qade gives better results both in terms of loss and MSE for its first 3 epochs, as compared to the first 2000 epochs of Elvet. For longer training times, Qade reaches a reasonably good value of 10^{-3} of the MSE. The limiting factor for improving the solution is its representation in terms of a degree-3 polynomial. Due to its use of a much more expressive neural network, Elvet is capable to significantly reducing the loss and MSE beyond 10^{-3} in this case.

For the wave equation, Qade outperforms Elvet in the MSE and reaches a similar final loss, even while we use a larger neural network (with 30 hidden units) in Elvet for this case. At epoch 8, Qade reaches an MSE of about 10^{-3}. Elvet's final MSE of 10^{-1} is comparable to Qade's third epoch. This is due to the more adequate parametrization used by Qade in terms of a (in this case, periodic) basis of functions, as opposed to a generic, fully connected neural network in Elvet. An even larger and/or more specialized network architecture would be able to overcome this eventually, but then the comparison would cease to be a fair one, given minimality of the examples we consider here for Qade.

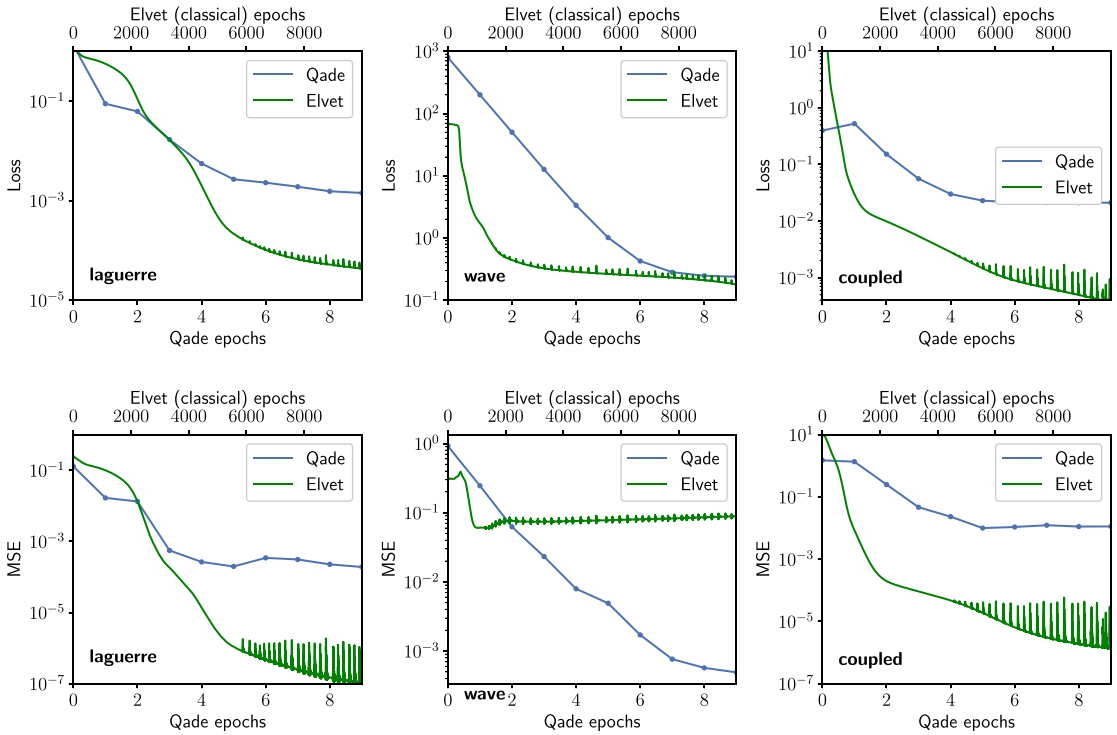

Figure 3. Comparison of the iterative approximation of the solution by Qade and the classical solver Elvet. The loss function (top) and MSE (bottom) are shown as functions of epoch, for the solution of the $n = 4$ Laguerre equation (left), the wave equation (center) and the system of coupled equations (right) presented in sections 4.1–4.3, respectively.

For the system of coupled equations, Qade performs better than Elvet only in its first epoch. While the solution is consistently improved in Qade throughout epochs 1–6, giving good final approximation with an MSE of 10^{-2}, Elvet obtains a near-perfect solution with an MSE of 10^{-7}.

The neural networks used in the Elvet calculations are small enough to be trained more efficiently in a modern laptop CPU than in a specialized graphical processing unit device. In an Apple M1 processor, the training times for the Laguerre, wave and coupled examples are: 2 s, 5 s and 6 s. For Qade, the total annealing times in the D-Wave device for each example are: 1 s, 0.4 s, and 0.4 s.

5. Conclusions

We have presented Qade, a general method for the solution of differential equations using quantum annealing devices. The first step is to re-formulate the equations as an optimization problem, by means of a general procedure which was originally developed for the application of machine learning. Then, this is transformed into a binary quadratic problem, which can directly be solved in a quantum annealer. We find that quantum annealing provides a reliable and fast way of solving highly non-trivial differential equations, encoded as an Ising-model ground-state finding problem.

We have implemented Qade the proposed method in a Python package, described in appendix which provides a user-friendly interface for the calculation of the binary quadratic model corresponding to a set of equations, and for its solution in a D-Wave quantum annealer.

We have applied the method to three examples of differential equations, with features including variable coefficients, partial derivatives, and coupled equations. The current quantum annealing technology only allows to solve problems that require a low number of qubits to encode, but we find that the chosen equations can already be solved reliably. We also run a comparison to a classical solver for these equations. The performance we obtain is comparable to that of Qade, but the lack of limitations in the representation of functions in the classical algorithm leads to a more precise solution in two of the three examples. However, in view of the relatively good performance we achieve with Qade so far, if the current limitations are lifted in future quantum annealers (with a larger number of qubits and a higher degree of connectivity), Qade has the potential to surpass classical methods for the solution of the larger differential equations that arise in real-world applications.

Data availability statement

The data that support the findings of this study are available upon reasonable request from the authors.

Appendix. The qade Python package

We provide an implementation of the Qade method, described in section 3, in the form of the Python package `qade`, which is publicly available in GitLab (gitlab.com/jccriado/qade) and PyPI, from which it can be installed through:

```
> pip install qade
```

For `qade` to send problems to be solved in the D-Wave systems, an installation of the D-Wave Ocean Tools, with the access token configured, is required. If this is not present, `qade` can still be used to compute the Ising model, whose ground state represents the solution to a given set of equations. In the rest of this section, we describe `qade`'s interface in full generality. For examples of use, see gitlab.com/jccriado/qade/-/tree/main/examples.

Defining the problem

The input data for `qade` consists of the sets X_i of sample points for the equations to be solved, together with the values of the vectors coefficients $C_{\text{in}}^{(k)}(x)$ and inhomogeneous terms $B_i(x)$ defined in equation (4), evaluated at all points $x \in X_i$. For easiness of use, an interface allowing for the specification of these parameters through a symbolic expression for an equation is provided. In order to define an equation

$$c_1(x)\frac{\partial^k f_1}{\partial x_1^{k_1}\cdots\partial x_{n_{\text{in}}}^{k_{n_{\text{in}}}}} + c_2(x)\frac{\partial^l f_2}{\partial x_1^{l_1}\cdots\partial x_{n_{\text{in}}}^{l_{n_{\text{in}}}}} + \cdots + b(x) = 0, \tag{A1}$$

the user would write the code:

```
f1, f2, ... = qade.function(n_in, n_out)
eq = qade.Equation(
   c1 * f1[k1, ...] + c2 * f2[l1, ...]
   + ... + b,
   samples,
)
```

where `samples` is an array-like[3] with shape `(n_samples, n_in)` (or just `(n_samples,)` when `n_in == 1`), representing the set of samples X_i; while `c1`, `c2`, …, `b` are either scalars or array-like objects of shape `(n_samples,)`, giving the values of the corresponding parameters in the equation at all the sample points.

Solving the problem

The first step in solving a given set of equations is choosing an adequate basis of functions. The function

```
basis = qade.basis(
   name, size_per_dim, n_in = 1, scale = 1.0
)
```

provides access to five pre-defined bases, which are listed in table A1. The `size_per_dim` defines the d parameter for the first three bases in the table, and how many grid points per input-space dimension are defined for the last 2. In both cases, the total dimension of the bases is `size_per_grid ** n_in`. The last two arguments, `n_in` and `scale` are only used by the last two bases. `scale` corresponds to λ in the definition of the corresponding $\phi(r)$ functions.

Given a list of equations `equations = [eq1, eq2, ...]` and a basis, `Qade` computes the quadratic loss function using equations (10) and (11). As a simplification, the pairs of indices nm and pq are flattened as $N = nm_{\max} + m$ and $P = pq_{\max} + q$, so that J and h become a matrix J_{PQ} and a vector h_P. They are obtained using the function

```
J, h = qade.loss(equations, basis)
```

The corresponding parameters $\hat{J}$ and $\hat{h}$ for the Ising model Hamiltonian are computed using equations (14) and (15). They are also flattened into a matrix $\hat{J}_{\hat{N}\hat{P}}$ and a vector $\hat{h}_{\hat{N}}$, through $\hat{N} = \alpha N_{\max} + N$

[3] An array-like object is either a `numpy` array or an object that can be converted into one. This includes scalars, lists and tuples.

Table A1. Bases of functions implemented by `Qade`.

Basis functions	Name	Definition
$\Phi_m(x) = \phi_{m_1}(x_1)\phi_{m_2}(x_2)\ldots$ $(0 \leqslant m_i < d, \quad m = m_1 + m_2 d + \ldots)$	`"fourier"`	$\phi_m(x) = \begin{cases} \cos(\pi n x) \\ \sin(\pi (n+1) x) \end{cases}$
	`"monomial"`	$\phi_m(x) = x^m$
	`"trig"`	$\phi_m(x) = \cos^{d-m-1}(x)\sin^m(x)$
$\Phi_m(x) = \phi(\lvert x - z_m \rvert)$ ($z_m \in$ equally-spaced grid in $[0,1]^{n_{\text{in}}}$)	`"gaussian"`	$\phi(r) = -\exp\left[-(r/\lambda)^2\right]$
	`"multiquadric"`	$\phi(r) = \sqrt{r^2 + \lambda^2}$

and $\hat{P} = \beta P_{\max} + P$, so that they can be directly provided to the D-Wave framework to be embedded in a quantum annealer. They are given by the function:

```
J_hat, h_hat = qade.ising(
   equations, basis,
   n_spins, centers, scales,
)
```

The last three arguments are optional. `n_spins` (default: 3) is the number of spins to use per weight w_N. `centers` (default: array of zeros) and `scales` (default: array of ones) are the flattened arrays of center values c_N and scales s_N from the binary encoding in equation (12).

The complete process of finding these parameters, sending them to a D-Wave quantum processing unit (QPU), setting it up and running the annealing process, reading the results, and decoding them back into the w_{nm} matrix of weight is automatized by a single function call:

```
sol = qade.solve(equations, basis, ...)
```

The returned object `sol` is a callable that receives an array `x` of samples and returns the value `sol(x)` of the solution at the sample points. It also contains three attributes:

- `sol.basis`, the basis of functions in which the problem was solved (its name is available through `sol.basis.name`).
- `sol.weights`, the matrix w_{nm} of weights.
- `sol.loss`, the value of the loss function.

ORCID iD

Juan Carlos Criado https://orcid.org/0000-0003-3571-994X

References

[1] Lee H and Kang I S 1990 Neural algorithm for solving differential equations *J. Comput. Phys.* **91** 110
[2] Meade A and Fernandez A 1994 The numerical solution of linear ordinary differential equations by feedforward neural networks *Math. Comput. Modelling* **19** 1
[3] Meade A and Fernandez A 1994 Solution of nonlinear ordinary differential equations by feedforward neural networks *Math. Comput. Modelling* **20** 19
[4] Lagaris I E, Likas A and Fotiadis D I 1997 Artificial neural networks for solving ordinary and partial differential equations (arXiv:physics/9705023)
[5] Raissi M, Perdikaris P and Karniadakis G E 2017 Physics informed deep learning (part i): data-driven solutions of nonlinear partial differential equations (arXiv:1711.10561[cs.AI])
[6] Raissi M, Perdikaris P and Karniadakis G E 2017 Physics informed deep learning (part ii): data-driven discovery of nonlinear partial differential equations (arXiv:171 1.10566[cs.AI])
[7] Raissi M, Perdikaris P and Karniadakis G 2019 Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations *J. Comput. Phys.* **378** 686
[8] Han J, Jentzen A and E W 2018 Solving high-dimensional partial differential equations using deep learning *Proc. Natl Acad. Sci.* **115** 8505
[9] Magill M, Qureshi F, and de Haan H W, 2018 Neural networks trained to solve differential equations learn general representations (arXiv:1807.00042[stat.ML])
[10] Piscopo M L, Spannowsky M and Waite P 2019 Solving differential equations with neural networks: applications to the calculation of cosmological phase transitions *Phys. Rev.* D **100** 016002
[11] Dockhorn T, 2019 A discussion on solving partial differential equations using neural networks (arXiv:1904.07200[cs.LG])
[12] Regazzoni F, Dedè L and Quarteroni A 2019 Machine learning for fast and reliable solution of time-dependent differential equations *J. Comput. Phys.* **397** 108852
[13] Chen R T Q, Rubanova Y, Bettencourt J and Duvenaud D 2019 Neural ordinary differential equations (arXiv:1806.07366[cs.LG])

[14] Shen X, Cheng X and Liang K 2020 Deep Euler method: solving odes by approximating the local truncation error of the Euler method (arXiv:2003.09573[math.NA])
[15] Rudd K, Muro G D and Ferrari S 2014 A constrained backpropagation approach for the adaptive solution of partial differential equations *IEEE Trans. Neural Netw. Learn. Syst.* **25** 571
[16] Rudd K and Ferrari S 2015 A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks *Neurocomputing* **155** 277
[17] Sirignano J and Spiliopoulos K 2018 DGM: a deep learning algorithm for solving partial differential equations *J. Comput. Phys.* **375** 1339
[18] Lu L, Meng X, Mao Z and Karniadakis G E 2019 DeepXDE: a deep learning library for solving differential equations (arXiv:1907.04502)
[19] Koryagin A, Khudorozhkov R and Tsimfer S 2019 Pydens: a Python framework for solving differential equations with neural networks (arXiv:1909.11544)
[20] Hennigh O, Narasimhan S, Nabian M A, Subramaniam A, Tangsali K, Rietmann M, del Aguila Ferrandis J, Byeon W, Fang Z and Choudhry S 2020 Nvidia simnet^{T}M: an ai-accelerated multi-physics simulation framework (arXiv:2012.07938[physics.flu-dyn])
[21] Chen F, Sondak D, Protopapas P, Mattheakis M, Liu S, Agarwal D and Giovanni M D 2020 Neurodiffeq: a Python package for solving differential equations with neural networks *J. Open Source Softw.* **5** 1931
[22] Hartmann D, Lessig C, Margenberg N and Richter T 2020 A neural network multigrid solver for the Navier-Stokes equations (arXiv:2008.11520[physics.comp-ph])
[23] Jin X, Cai S, Li H and Karniadakis G E 2021 NSFnets (Navier-Stokes flow nets): physics-informed neural networks for the incompressible Navier-Stokes equations *J. Comput. Phys.* **426** 109951
[24] Li Z, Kovachki N, Azizzadenesheli K, Liu B, Bhattacharya K, Stuart A and Anandkumar A 2020 Fourier neural operator for parametric partial differential equations (arXiv:2010.08895[cs.LG])
[25] Lau L L H and Werth D 2020 Oden: a framework to solve ordinary differential equations using artificial neural networks (arXiv:2005.14090[physics.com p-ph])
[26] Guidetti V, Muia F, Welling Y and Westphal A 2021 dNNsolve: an efficient NN-based PDE solver (arXiv:2103.08662[cs.LG])
[27] Araz J Y, Criado J C and Spannowsky M 2021 Elvet – a neural network-based differential equation and variational problem solver (arXiv:2103.14575[cs.LG])
[28] Finilla A B, Gomez M A, Sebenik C and Doll J D 1994 Quantum annealing: a new method for minimizing multidimensional functions *Chem. Phys. Lett.* **219** 343
[29] Kadowaki T and Nishimori H 1998 Quantum annealing in the transverse Ising model *Phys. Rev.* E **58** 5355
[30] Brooke J, Bitko D, Rosenbaum T F and Aeppli G 1999 Quantum annealing of a disordered magnet *Science* **284** 779
[31] Dickson N G 2013 Thermally assisted quantum annealing of a 16-qubit problem *Nat. Commun.* **4** 1903
[32] Lanting T *et al* 2014 Entanglement in a quantum annealing processor *Phys. Rev.* X **4** 021041
[33] Albash T, Vinci W, Mishra A, Warburton P A and Lidar D A 2015 Consistency tests of classical and quantum models for a quantum annealer *Phys. Rev.* A **91** 042314
[34] Albash T and Lidar D A 2018 Adiabatic quantum computing *Rev. Mod. Phys.* **90** 015002
[35] Boixo S, Smelyanskiy V N, Shabani A, Isakov S V, Dykman M, Denchev V S, Amin M H, Smirnov A Y, Mohseni M and Neven H 2016 Computational multiqubit tunnelling in programmable quantum annealers *Nat. Commun.* **7** 10327
[36] Chancellor N, Szoke S, Vinci W, Aeppli G and Warburton P A 2016 Maximum–entropy inference with a programmable annealer *Sci. Rep.* **6** 22318
[37] Benedetti M, Realpe-Gómez J, Biswas R and Perdomo-Ortiz A 2016 Estimation of effective temperatures in quantum annealers for sampling applications: a case study with possible applications in deep learning *Phys. Rev.* A **94** 022308
[38] Muthukrishnan S, Albash T and Lidar D A 2016 Tunneling and speedup in quantum optimization for permutation-symmetric problems *Phys. Rev.* X **6** 031010
[39] Cervera Lierta A 2018 Exact Ising model simulation on a quantum computer *Quantum* **2** 114
[40] Lanting T 2017 The D-Wave 2000Q processor (presented at AQC 2017)
[41] Farhi E, Goldstone J and Gutmann S 2002 Quantum adiabatic evolution algorithms versus simulated annealing (arXiv:ph/0201031)
[42] Abel S, Chancellor N and Spannowsky M 2021 Quantum computing for quantum tunneling *Phys. Rev.* D **103** 016008
[43] Abel S, Blance A and Spannowsky M 2021 Quantum optimisation of complex systems with a quantum annealer (arXiv:2105.13945[quant-ph])
[44] Abel S, Criado J C and Spannowsky M 2022 Completely quantum neural networks (arXiv:2202.11727[quant-ph])
[45] Zlokapa A, Mott A, Job J, Vlimant J-R, Lidar D and Spiropulu M 2020 Quantum adiabatic machine learning by zooming into a region of the energy surface *Phys. Rev.* A **102** 062405
[46] Lubasch M, Joo J, Moinier P, Kiffner M and Jaksch D 2020 Variational quantum algorithms for nonlinear problems *Phys. Rev.* A **101** 010301
[47] Zanger B, Mendl C B, Schulz M and Schreiber M 2021 Quantum algorithms for solving ordinary differential equations via classical integration methods *Quantum* **5** 502
[48] Srivastava S and Sundararaghavan V 2019 Box algorithm for the solution of differential equations on a quantum annealer *Phys. Rev.* A **99** 052355
[49] Kyriienko O, Paine A E and Elfving V E 2021 Solving nonlinear differential equations with differentiable quantum circuits *Phys. Rev.* A **103** 052416
[50] Born M and Fock V 1928 Beweis des adiabatensatzes *Z. Phy.* **51** 165
[51] Kingma D P and Ba J 2014 Adam: a method for stochastic optimization (arXiv:1412.6980)