Solving Differential Equations using Quantum Algorithms

A concise overview presentation

IOP Publishing Mach. Learn.: Sci. Technol. 6 (2025) 015002 https://doi.org/10.1088/2632-2153/ada3ab

MACHINE LEARNING
Science and Technology

PAPER

Self-adaptive physics-informed quantum machine learning for solving differential equations

Abhishek Setty[1,2,3,*], **Rasul Abdusalamov**[1] and **Felix Motzoi**[2,3]

[1] Department of Continuum Mechanics, RWTH Aachen University, Aachen 52062, Germany
[2] Forschungszentrum Jülich, Institute of Quantum Control (PGI-8), Jülich D-52425, Germany
[3] Institute for Theoretical Physics, University of Cologne, Cologne D-50937, Germany
[*] Author to whom any correspondence should be addressed.

E-mail: a.setty@fz-juelich.de

Keywords: quantum machine learning, physics-informed neural networks, variational quantum algorithms, differential equations

OPEN ACCESS

RECEIVED
22 October 2024

REVISED
28 November 2024

ACCEPTED FOR PUBLICATION
27 December 2024

PUBLISHED
13 January 2025

Abstract

Chebyshev polynomials have shown significant promise as an efficient tool for both classical and quantum neural networks to solve linear and nonlinear differential equations (DEs). In this work, we adapt and generalize this framework in a quantum machine learning setting for a variety of problems, including the 2D Poisson's equation, second-order linear DE, system of DEs, nonlinear Duffing and Riccati equation. In particular, we propose in the quantum setting a modified Self-Adaptive Physics-Informed Neural Network approach, where self-adaptive weights are applied to problems with multi-objective loss functions. We further explore capturing correlations in our loss function using a quantum-correlated measurement, resulting in improved accuracy for initial value problems. We analyse also the use of entangling layers and their impact on the solution accuracy for second-order DEs. The results indicate a promising approach to the near-term evaluation of DEs on quantum devices.

Overview

Differential Equations

↓

PINNs

↓

Quantum PINNs

↓

VQC Training

↓

Results

Why Differential Equations?

Differential Equations: The language of Physical Systems

Physical System	Example
Classical Mechanics	Planetary motion, pendulums
Heat Transfer	Diffusion of heat in materials
Fluid Dynamics	Weather prediction, aerodynamics
Electromagnetism	Wireless communication, optics
Quantum Mechanics	Atoms, molecules, quantum devices

Understanding and predicting physical phenomena often reduces to solving differential equations.

How Do We Solve Differential Equations?

Given a differential equation, how can we obtain its solution?

Analytical Solutions

Idea: Find a closed-form expression satisfying the equation.

$$\frac{dy}{dx} = ky$$

$$y(x) = Ce^{kx}$$

Advantages

Exact solution

Limitations

Available only for a limited class of problems

Numerical Methods

Idea: Discretize the domain and approximate derivatives.

Examples:

- Finite Difference Method (FDM)
- Finite Element Method (FEM)
- Spectral Methods

Advantages

Applicable to complex systems

Limitations

Computational cost grows rapidly with dimension

Can Machine Learning Help?

Instead of solving the differential equation directly, can we learn the solution?

Differential Equation

↓

Neural Network

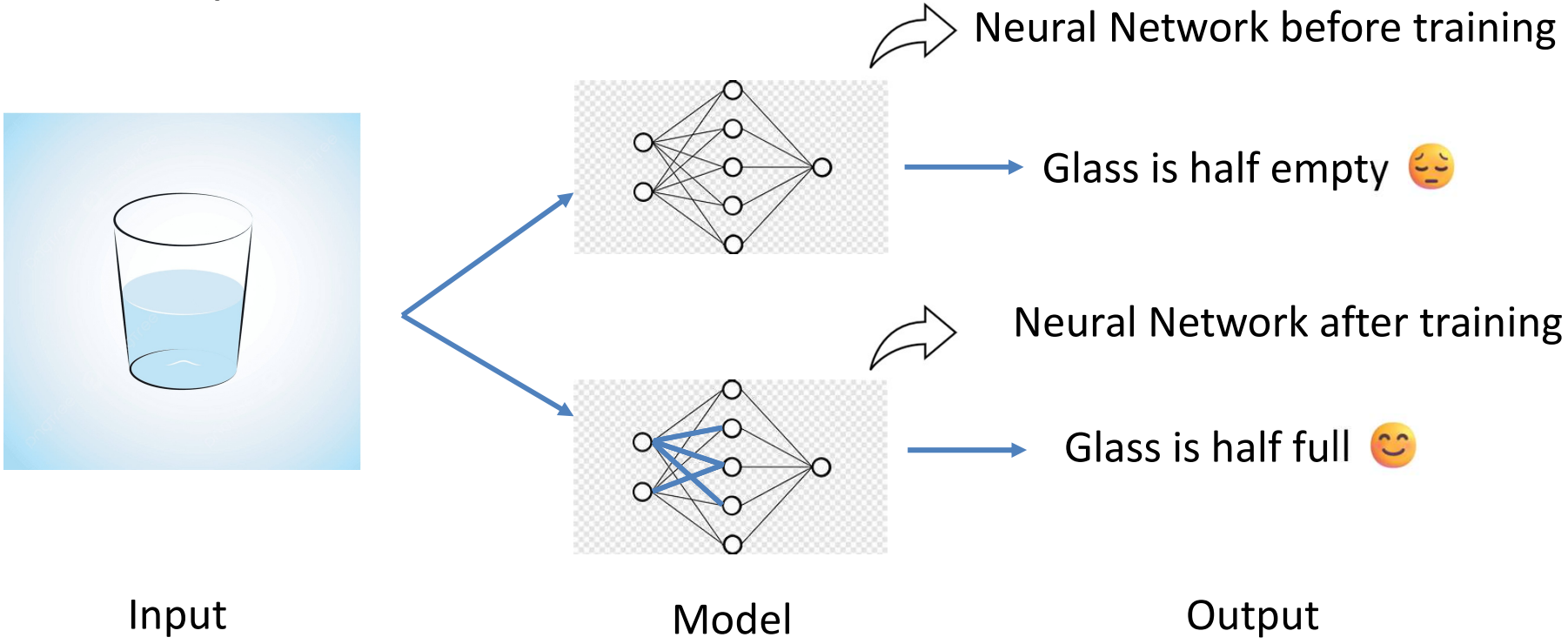
↓

Approximate Solution

Physics-Informed Neural Network

Neural Network as a Function Approximator

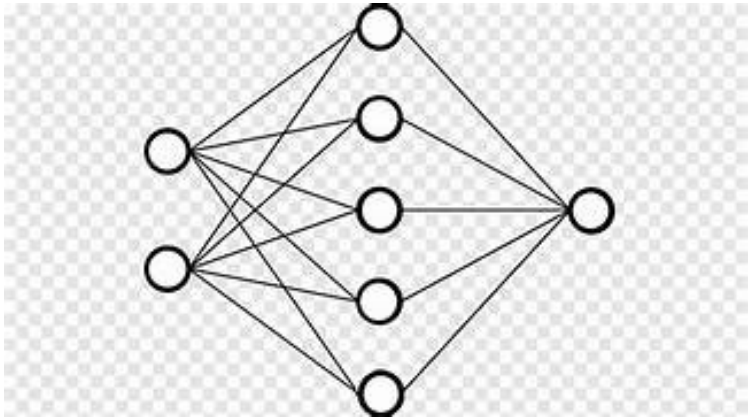
- The Therapist Network



Takeaway: Neural Network is just a trainable function

INPUT

x



OUPUT

$y = NN(x,\theta)$

MODEL

$NN(x,\theta)$

For example

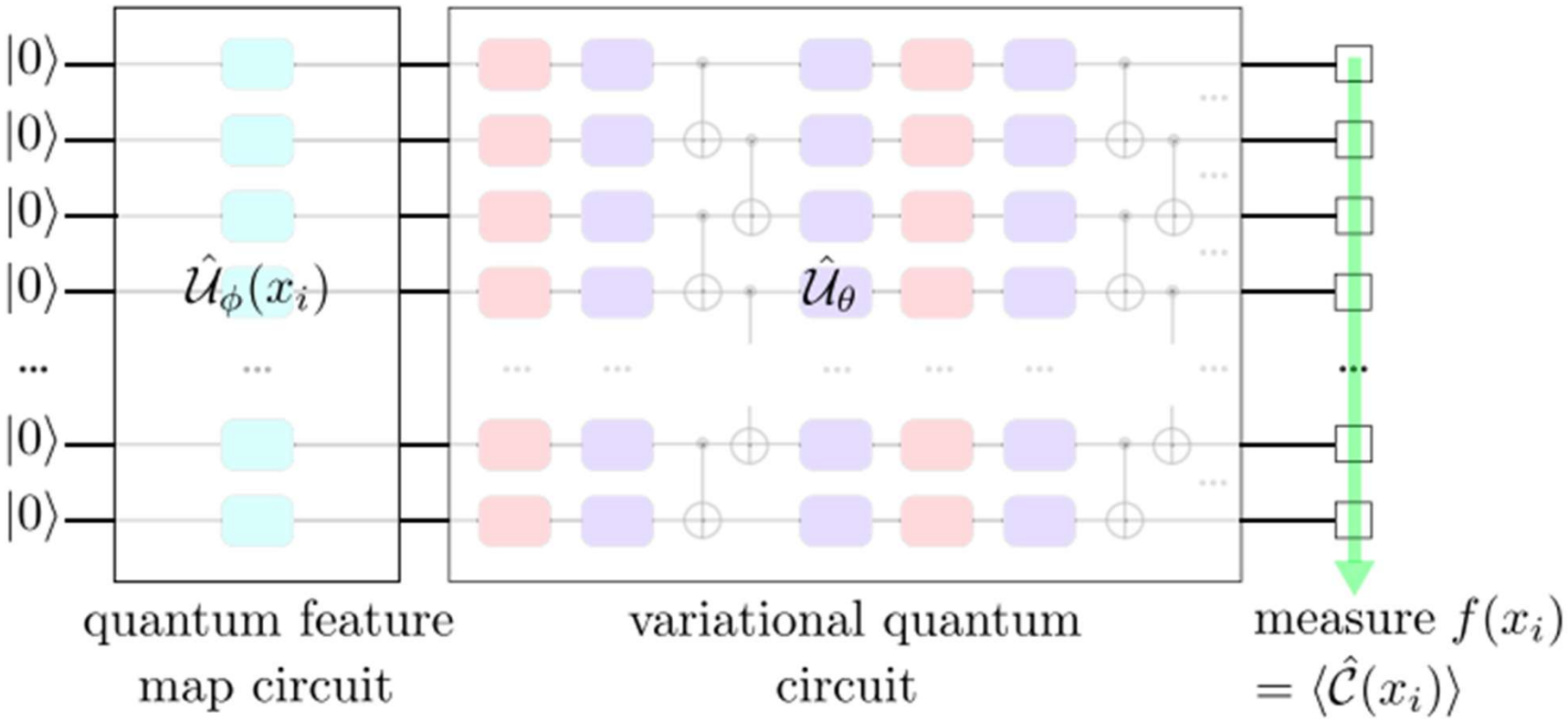
$y = w_2 v(w_1 x + c)$

Trained using:

- Gradient descent approach (stochastic) with an appropriate loss function

Can we have a quantum analog of PINNs?

Quantum Circuit for solving DEs



VQC uses a quantum circuit to represent the function solution, so the Quantum circuit acts as a quantum function approximator.

$U(x) \approx U_\theta(x)$ Comes from

- Qubit
- Quantum gates
- Measurements

Step 1 : Encode the Input

DE variable : x , Needs to be encoded in the quantum circuit.

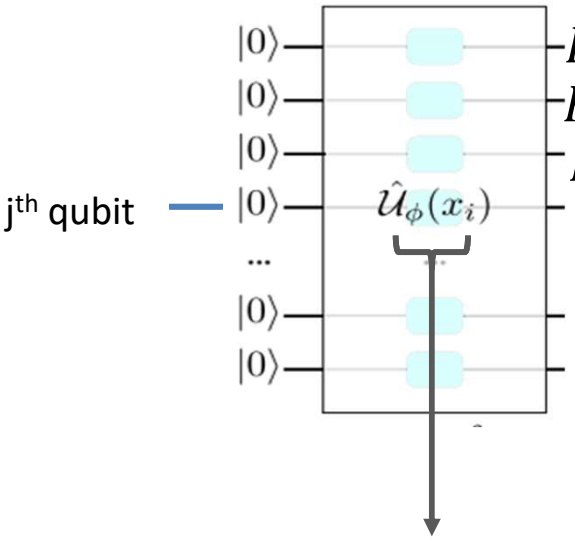
Chebyshev Feature Map encoding

Motivation: Are powerful basis functions for approximating solutions of differential Equations.

Many functions can be approximated efficiently as

$$f(x) \approx \sum_k c_k T_k(x)$$

This is analogous to Fourier expansions.



$$R_y(2\cos^{-1}x)\,|0\rangle = \mathrm{T_1(x)I_1} + \sqrt{1-x^2}\,\mathrm{U_0(x)X_1Z_1}|0\rangle$$
$$R_y(4\cos^{-1}x)\,|0\rangle = \mathrm{T_2(x)I_2} + \sqrt{1-x^2}\,\mathrm{U_1(x)X_2Z_2}|0\rangle$$
$$R_y(6\cos^{-1}x)\,|0\rangle$$

ϕ

$$R_y(2n\cos^{-1}x)\,|0\rangle = \mathrm{T_n(x)I_n} + \sqrt{1-x^2}\,\mathrm{U_{n-1}(x)X_nZ_n}|0\rangle$$

$$\hat{U}_{\phi(x)} = \bigotimes_{j=1}^{N} R_{y,j}(2n[j]\cos^{-1}x)$$

Rotation of the j[th] qubit about the y-axis by angle $2n[j]\cos^{-1}x$.

We now have information stored (via qubits) in basis of Chebyshev's polynomials. These are the building blocks.

$$R_y = \begin{pmatrix} \cos(\frac{\phi}{2}) & -\sin(\frac{\phi}{2}) \\ \sin(\frac{\phi}{2}) & \cos(\frac{\phi}{2}) \end{pmatrix}$$

$$T_n(x) = \cos\left(n\cos^{-1}(x)\right)$$
$$\sin\theta\, U_n(\cos\theta) = \sin(n+1)\theta$$

$$T_0(x) = 1,\quad T_1(x) = x,\quad T_2(x) = 2x^2 - 1,\quad T_3(x) = 4x^3 - 3x$$

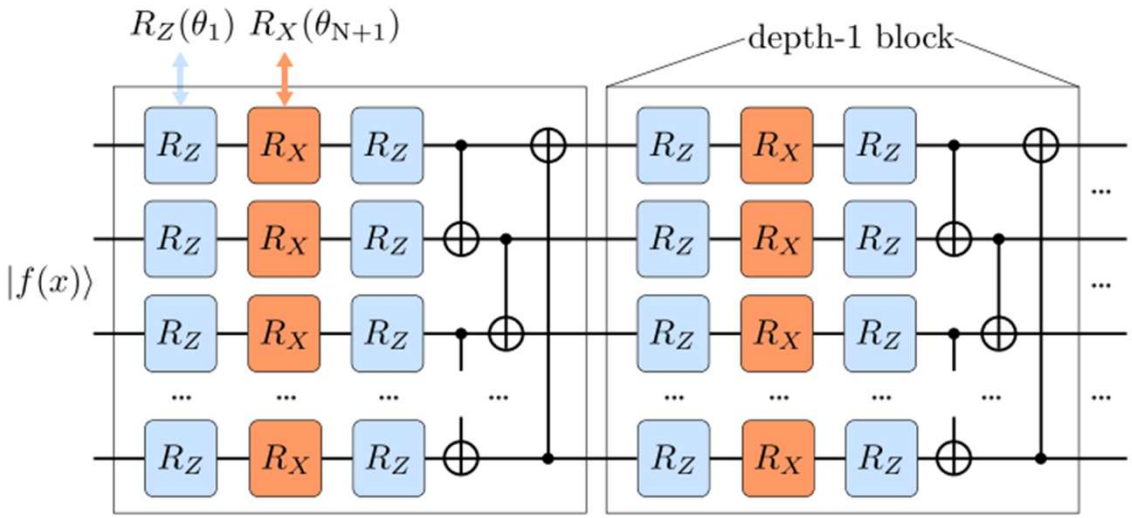
$$U_0(x) = 1,\quad U_1(x) = 2x,\quad U_2(x) = 4x^2 - 1,\quad U_3(x) = 8x^3 - 4x$$

We Have the Features... Now What?

Now we need a recipe that is :

- Expressive enough
- Trainable
- Hardware-friendly

These requirements motivate the use of a Hardware-Efficient Ansatz.



Now imagine the encoded state is a collection of ingredients.

The job is to repeatedly do two things:

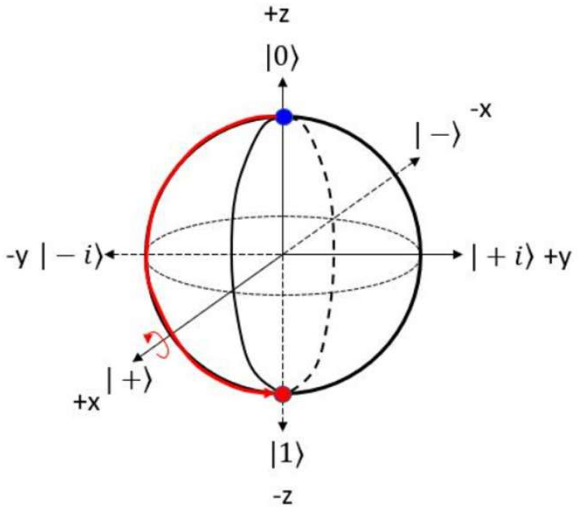
Step 1: Mix each ingredient

This is the rotation layer.

$$R_Z(\theta_1)R_X(\theta_2)R_Z(\theta_3)$$

on every qubit.

Provides Maximum local flexibility for each qubit



Interpretation:

Adjust the contribution of each encoded feature independently.

Step 2: Allow ingredients to interact

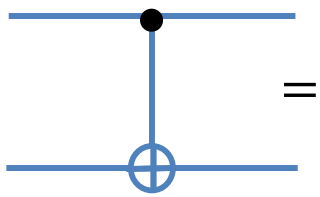
This is the CNOT layer.

Interpretation:

Create correlations between features.

Qubit A

$\alpha_1|0\rangle + \beta_1|1\rangle$



$\alpha_2|0\rangle + \beta_2|1\rangle$

Qubit B

$$= \alpha_1\alpha_2|00\rangle + \alpha_1\beta_2|01\rangle + \alpha_2\beta_1|11\rangle + \beta_1\beta_2|10\rangle$$

Mixture of the amplitudes. Now more complicated structures can emerge.

Now repeat the block (rotation+ entanglement) d times, for even richer expressivity.

We have a quantum state prepared … Now what?

Now is the time for extraction of it's properties

Measurements

Difference between a quantum and classical algorithm:

In Classical algorithm,

input → computation → output = a number, a vector , direct approximation of the answer etc.

In Quantum algorithm,

input → parameterized circuit → $|\psi(\theta)\rangle$ → measurement → samples → classical estimate.

But what since we know the whole circuit and gates and everything used, so in principle we can just keep following it and end up with the amplitude form of the final state, isn't it?

Umm… not really!

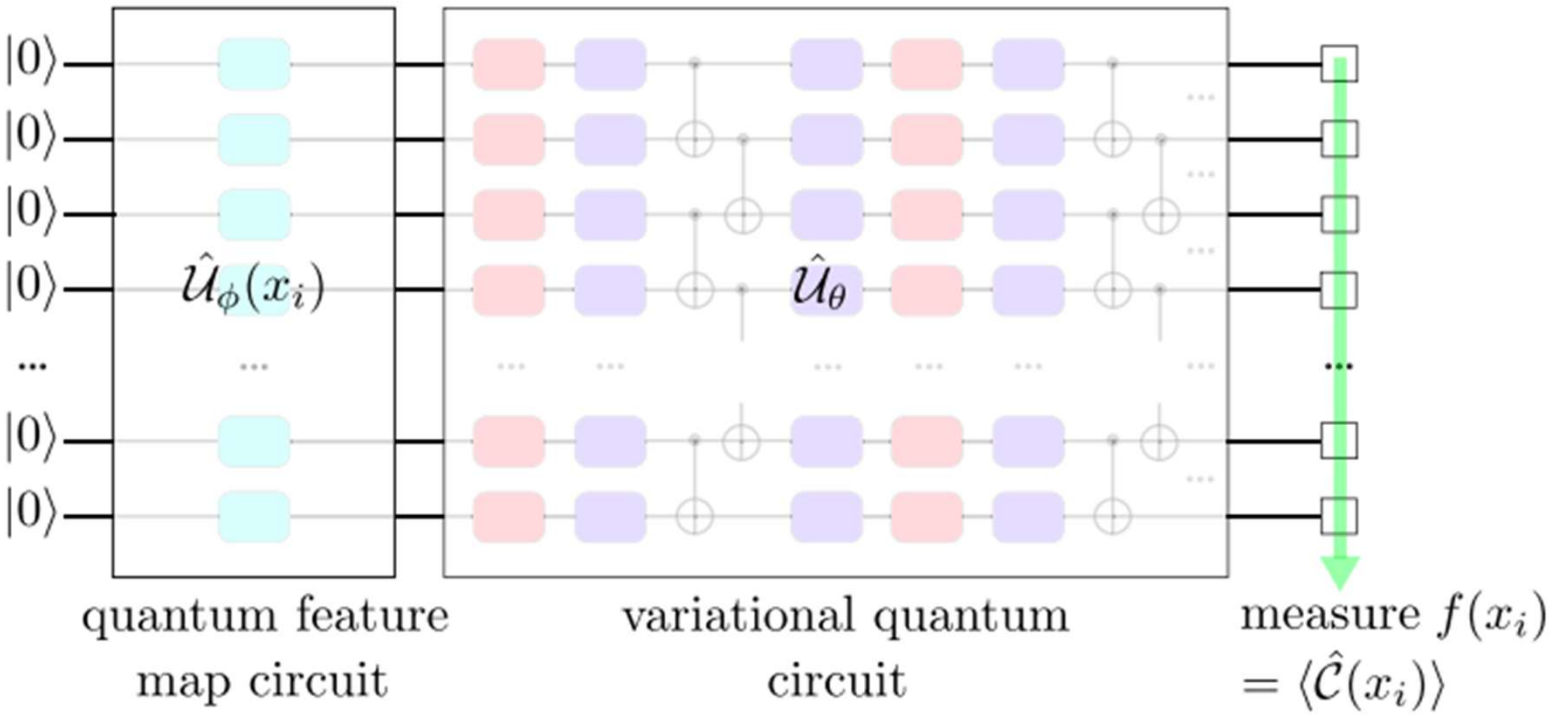
This is where the problem of simulating a quantum system on a classical Machine comes Into the picture

For n qubits, the state vector has 2^n complex amplitudes:

- so, for n=10 : 1024 amplitudes (easy)
- For n= 20 : 1 million amplitudes (ok-ish)
- For n= 50 : 1 quadrillion amplitudes (impossible)
- For n=100 : astronomically impossible !

Classical difficulty arises not only from the size of the quantum state, but also from the structure of entanglement and interference, which makes even sampling and expectation estimation computationally challenging

Recap



Encodes Chebyshev's features with input x

Tunes and mixes the features

Output = $f(x)$

Optimization of f(x)

Loss function

To simulate the system, the quantum circuit is trained using the following loss function:

$$L_\theta[d_x f, f, x] = L_\theta^{(diff)}[d_x f, f, x] + L_\theta^{(boundary)}[f, x]$$

$$L_\theta^{(diff)}[d_x f, f, x] = \frac{1}{M} \sum_{i=1\ to\ M} L(F[d_x f(x_i), f(x_i), x_i], 0)$$

$$L_\theta^{(boundary)}[f, x] = \eta L(f(x_0), u_0)$$

$$L(a, b) = function\ to\ describe\ the\ distance\ between\ a\ and\ b.$$

$$L(a, b) = (a - b)^2$$

$$L(a, b) = |a - b|$$

For example: Harmonic Oscillator:

Consider:

$$\frac{d^2u}{dx^2} + u(x) = 0$$

With boundary conditions,

$u(0) = 0, \quad u\left(\frac{\pi}{2}\right) = 1$

Then loss function would be:

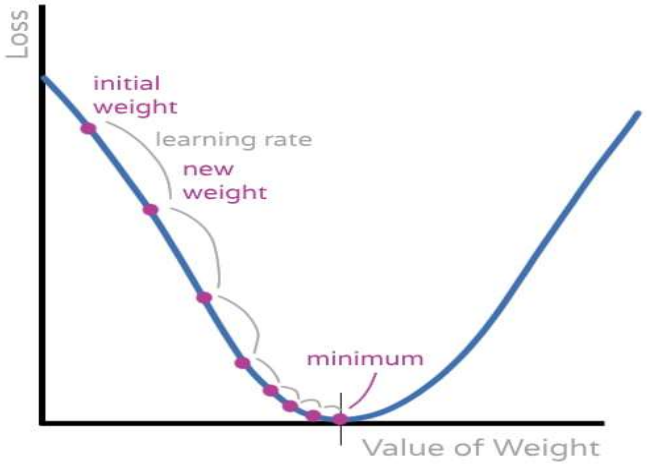
$$L = \frac{1}{N_f}\sum_i \left(\frac{d^2u_{pred}}{dx^2} + u_{pred}(x)\right)^2 + \left(u_{pred}(0) - 0\right)^2 + \left(u_{pred}\left(\frac{\pi}{2}\right) - 1\right)^2$$

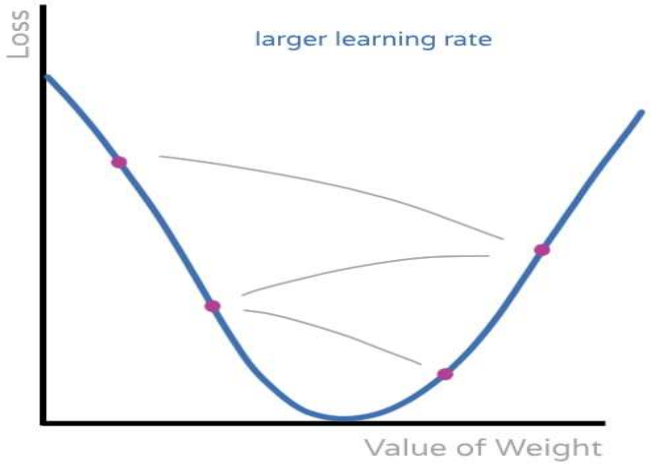
The parameter update rule for gradient descent approach :

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \eta \frac{\partial L}{\partial \theta_i}$$

η: learning rate

$\frac{\partial L}{\partial \theta_i}$: gradient of the loss with respect to parameter θ_i





Now how do we compute the derivatives of the classical function f(x)?

$$\frac{df}{dx} = \sum_j \frac{\partial U_\phi}{\partial \phi_j} \frac{d\phi_j}{dx}$$

As the function f is not given analytically. It is produced through quantum measurements, Therefore we need to differentiate the encoder circuit $\widehat{U}_\phi$.

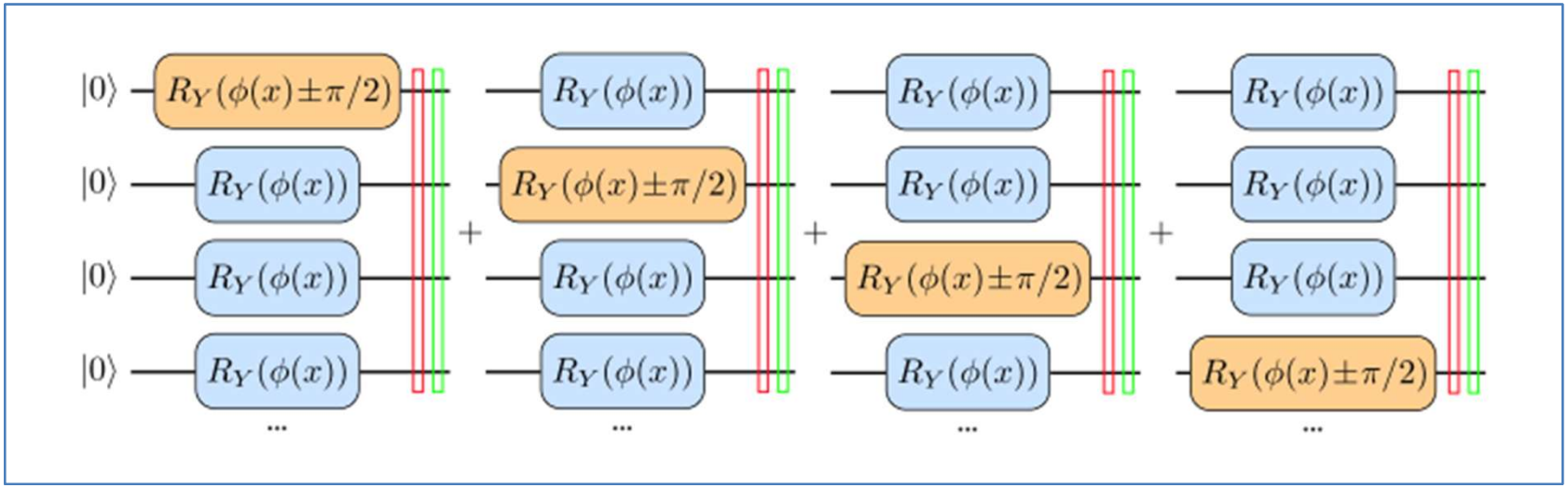
<u>Parameter shift rule for the derivative of the encoder circuit:</u>

For string of Pauli operators or involutory operators in the encoder circuit, Then the derivative are written as,

$$\frac{\partial U_\phi}{\partial \phi_j} = \langle \psi^+_{d\phi,j,\theta}(x) | \hat{C} | \psi^+_{d\phi,j,\theta}(x) \rangle - \langle \psi^-_{d\phi,j,\theta}(x) | \hat{C} | \psi^-_{d\phi,j,\theta}(\mathrm{x}) \rangle$$

State prepared with a phase shift in j^{th} qubit angle $\phi(x)\ to\ \phi(x) + \frac{\pi}{2}$

State prepared with a phase shift in j^{th} qubit angle $\phi(x)\ to\ \phi(x) - \frac{\pi}{2}$



$$\frac{d^2f}{dx^2} = \frac{d}{dx}\left(\frac{df}{dx}\right) = \frac{d}{dx}\left(\sum_j \frac{dU_{\phi_j}}{d\phi_j}\frac{d\phi_j}{dx}\right)$$

$$= \sum_{j,k} \frac{d^2U_\phi}{d\phi_j d\phi_k}\left[\frac{4jk}{1-x^2}\right] + \sum_j \frac{dU_\phi}{d\phi_j}\left[-\frac{2jx}{(1-x^2)^{\frac{1}{2}}}\right]$$

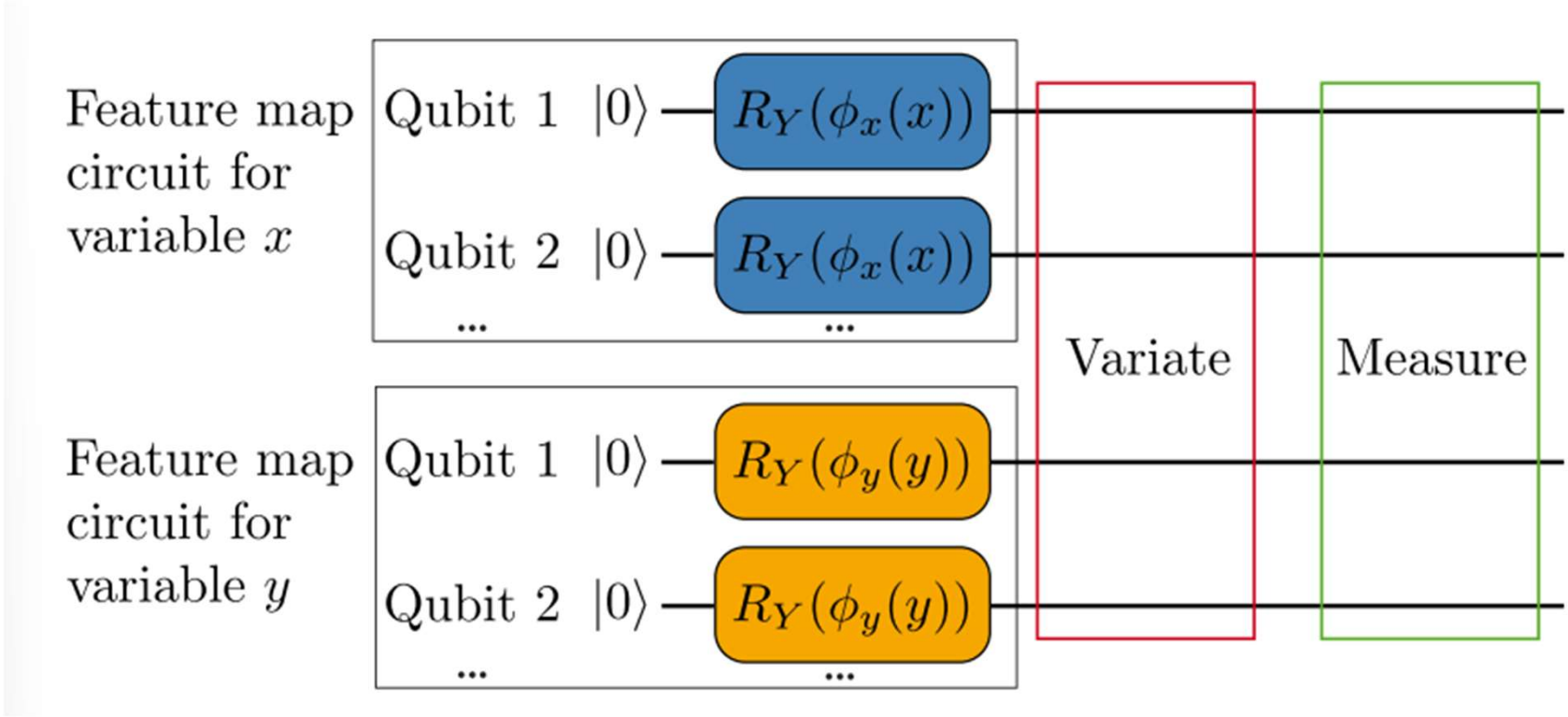
Four shifted terms per qubit

Two shifted terms per qubit

What about the PDE? How can we encode other independent variables?

We define two unitary operators $\widehat{U}_{\phi_x}(x)$ and $\widehat{U}_{\phi_y}(y)$.

$$\widehat{U}_{\phi(x,y)}(x,y) = \widehat{U}_{\phi_x}(x) \otimes \widehat{U}_{\phi_y}(y)$$



The calculation of the first and second-order derivatives of quantum circuits concerning all qubits are described by equations,

$$\begin{bmatrix} \frac{\partial u}{\partial x} \\ \frac{\partial u}{\partial y} \end{bmatrix} = \begin{bmatrix} \frac{\partial u_{\text{pred}}}{\partial \phi_{xj}\left(x_f^i\right)} \\ \frac{\partial u_{\text{pred}}}{\partial \phi_{ym}\left(y_f^i\right)} \end{bmatrix},$$

$$\begin{bmatrix} \frac{\partial^2 u}{\partial x^2} & \frac{\partial^2 u}{\partial x \partial y} \\ \frac{\partial^2 u}{\partial y \partial x} & \frac{\partial^2 u}{\partial y^2} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 u_{\text{pred}}}{\partial \phi_{xj}\left(x_f^i\right) \partial \phi_{xk}\left(x_f^i\right)} & \frac{\partial^2 u_{\text{pred}}}{\partial \phi_{xj}\left(x_f^i\right) \partial \phi_{yn}\left(y_f^i\right)} \\ \frac{\partial^2 u_{\text{pred}}}{\partial \phi_{ym}\left(y_f^i\right) \partial \phi_{xk}\left(x_f^i\right)} & \frac{\partial^2 u_{\text{pred}}}{\partial \phi_{ym}\left(y_f^i\right) \partial \phi_{yn}\left(y_f^i\right)} \end{bmatrix}$$

Self-adaptive weights for multi-objective loss function

$$\mathcal{L}\left(\boldsymbol{\theta}, \boldsymbol{\lambda}_f, \boldsymbol{\lambda}_b\right) = \mathcal{L}_f\left(\boldsymbol{\theta}, \boldsymbol{\lambda}_f\right) + \mathcal{L}_b\left(\boldsymbol{\theta}, \boldsymbol{\lambda}_b\right)$$

$$\mathcal{L}_f\left(\boldsymbol{\theta}, \boldsymbol{\lambda}_f\right) = \frac{1}{N_f}\sum_{i=1}^{N_f} m\left(\lambda_f^i\right)\left(f\left(x_f^i; \boldsymbol{\theta}\right)\right)^2,$$

$$\mathcal{L}_b\left(\boldsymbol{\theta}, \boldsymbol{\lambda}_b\right) = \frac{1}{N_b}\sum_{i=1}^{N_b} m\left(\lambda_b^i\right)\left(u_{\text{pred}}\left(x_b^i; \boldsymbol{\theta}\right) - u_b\left(x_b^i\right)\right)^2$$

Mask functions → Strictly increasing function of λ.

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \eta^k \nabla_{\boldsymbol{\theta}} \mathcal{L}\left(\boldsymbol{\theta}^k, \boldsymbol{\lambda}_f^k, \boldsymbol{\lambda}_b^k\right),$$

$$\boldsymbol{\lambda}_f^{k+1} = \boldsymbol{\lambda}_f^k + \rho_f^k \nabla_{\boldsymbol{\lambda}_f} \mathcal{L}\left(\boldsymbol{\theta}^k, \boldsymbol{\lambda}_f^k, \boldsymbol{\lambda}_b^k\right),$$

$$\boldsymbol{\lambda}_b^{k+1} = \boldsymbol{\lambda}_b^k + \rho_b^k \nabla_{\boldsymbol{\lambda}_b} \mathcal{L}\left(\boldsymbol{\theta}^k, \boldsymbol{\lambda}_f^k, \boldsymbol{\lambda}_b^k\right),$$

→ Parameter λ maximizes whereas Parameter θ minimizes.

Results

1. Riccati equation:

Why Study the Riccati Equation?

- Appears in optimal control and state estimation.
- Used in robotics, aerospace, autonomous systems, and quantum control.
- Provides a nonlinear model for many physical processes.

$$\frac{du}{dx} - 4u + 6u^2 - \sin(50x) - u\cos(25x) + \frac{1}{2} = 0,$$

$$u(x = 0) = u_0$$

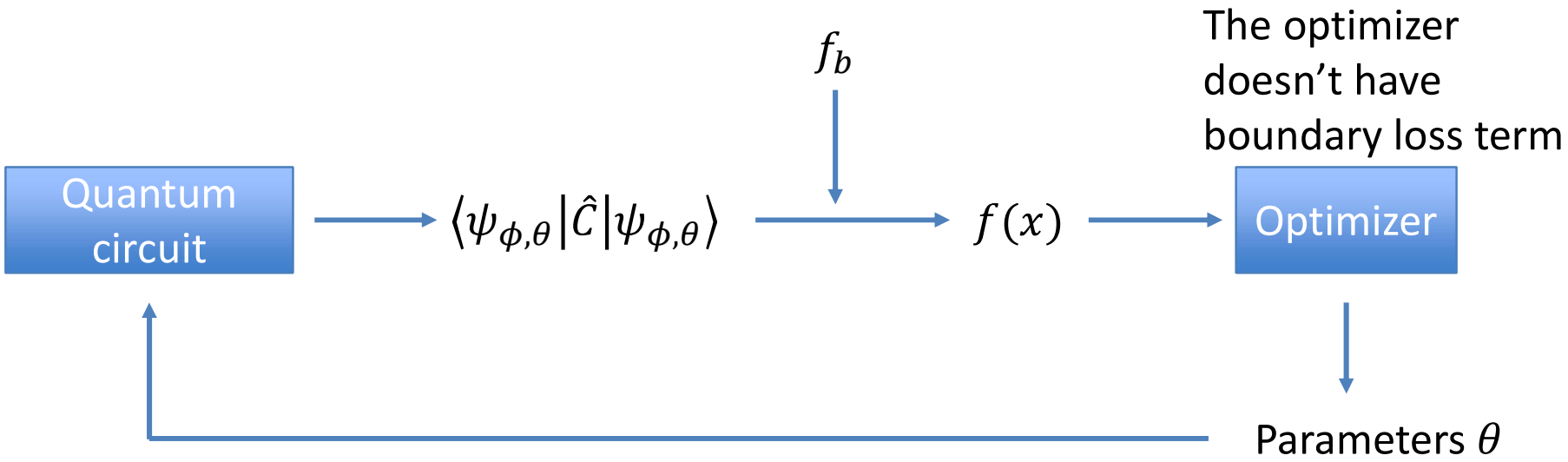
Floating boundary handling

In multiple loss function, trying to optimize one loss function will affect the Performance of the other.

But, if DE has f(x) term in it then there is a possibility to solve the problem In a more efficient way.

$$f(x) = f_b + \langle \psi_{\phi,\theta} | \hat{C} | \psi_{\phi,\theta} \rangle$$

$$f_b = u_0 - \langle \psi_{\theta,\phi}(x_0) | \hat{C} | \psi_{\theta,\phi}(x_0) \rangle$$



Earlier Loss term is given by (here u means f):

$$L = \left(\sum_x \frac{du}{dx} - 4u + 6u^2 - \sin(50x) - u\cos(25x) \right) + \frac{1}{2} + (u(x=0) - u_0)$$

Comes from the differential eqn

Boundary condition

With modified u(x) :

$$u_m(x) = \left(u_0 - u_{pred}(x_0)\right) + u_{pred}(x)$$

The loss term becomes:

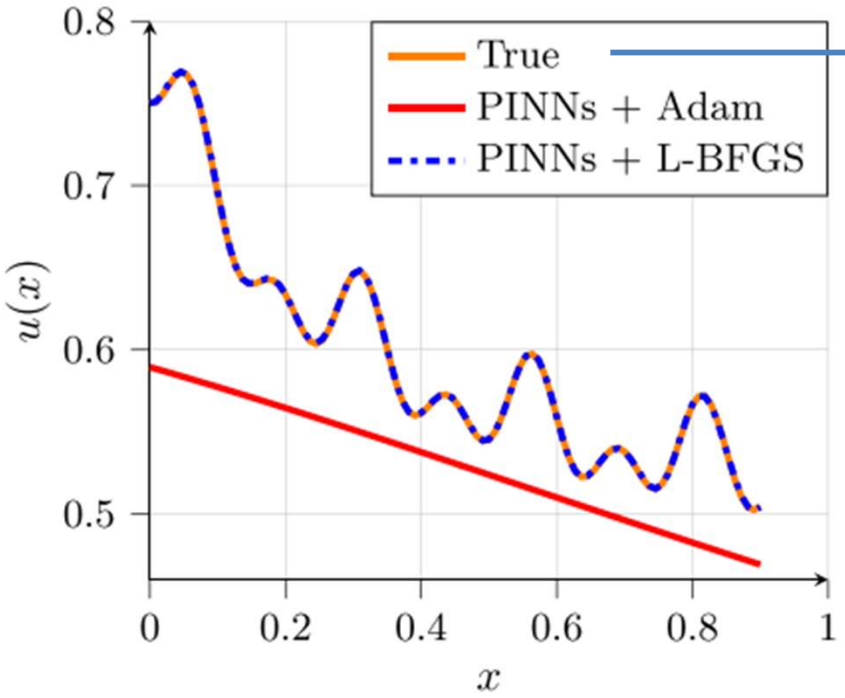
$$L = \sum_x \frac{du_m}{dx} - 4u_m + 6u_m^2 - \sin(50x) - u_m\cos(25x) + \frac{1}{2}$$

Addition:

$$\hat{C} = \sum_j Z_j$$

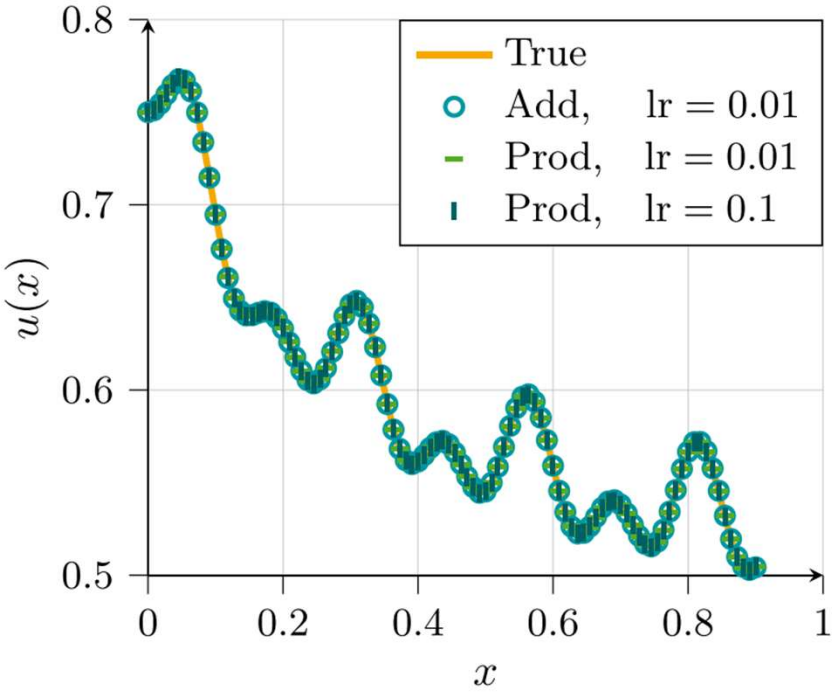
Product:

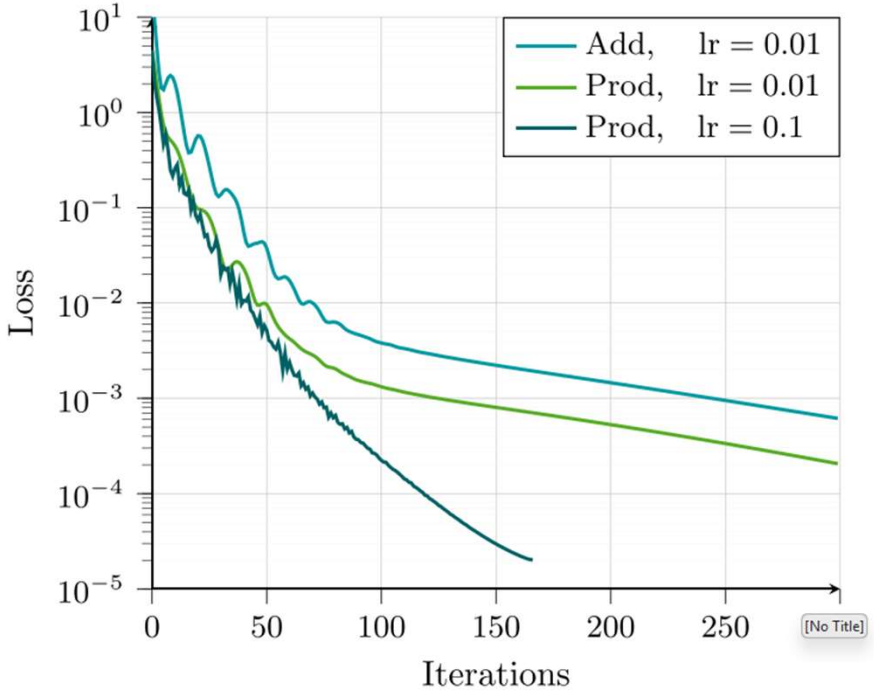
$$\widehat{C} = \bigotimes_{j=1}^{N} Z_j$$



From Runge-kutta third order

8 Qubits and 8 block depth





Key takeaways:

- Classical ML PINNs requires high precision update direction by L-BFGS optimizer
- Choice of observable as the tensor product outperformed the addition choice.

Why so?

Because it can show correlations between the qubits thus expressing Entanglement.

as an example, say for a bell state: $|\phi\rangle = \frac{|00\rangle}{\sqrt{2}} + \frac{|11\rangle}{\sqrt{2}}$

$\langle Z_1 \rangle + \langle Z_2 \rangle = 0 + 0 = 0$

but

$\langle Z_1 Z_2 \rangle = 1$ Meaning qubits are perfectly correlated

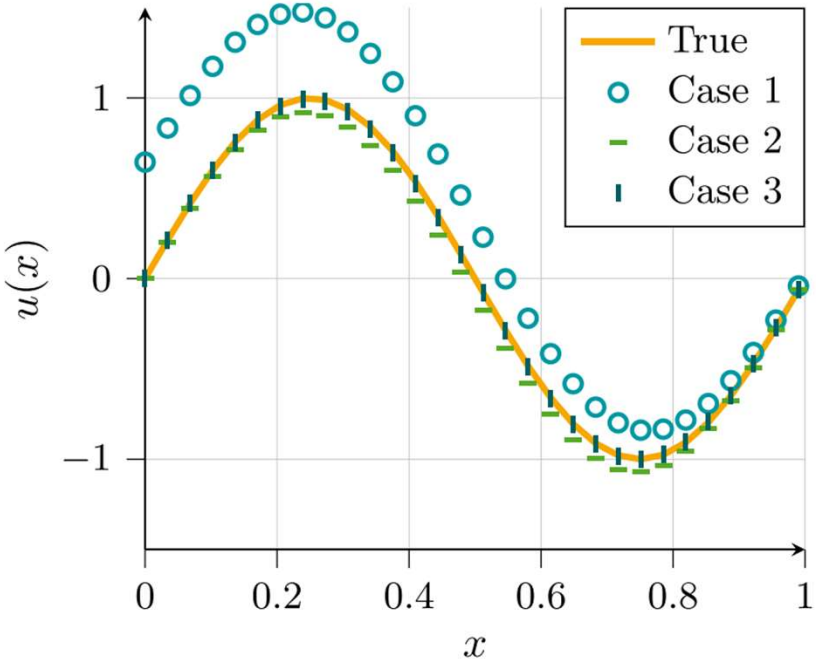
2. Second-order linear DE:

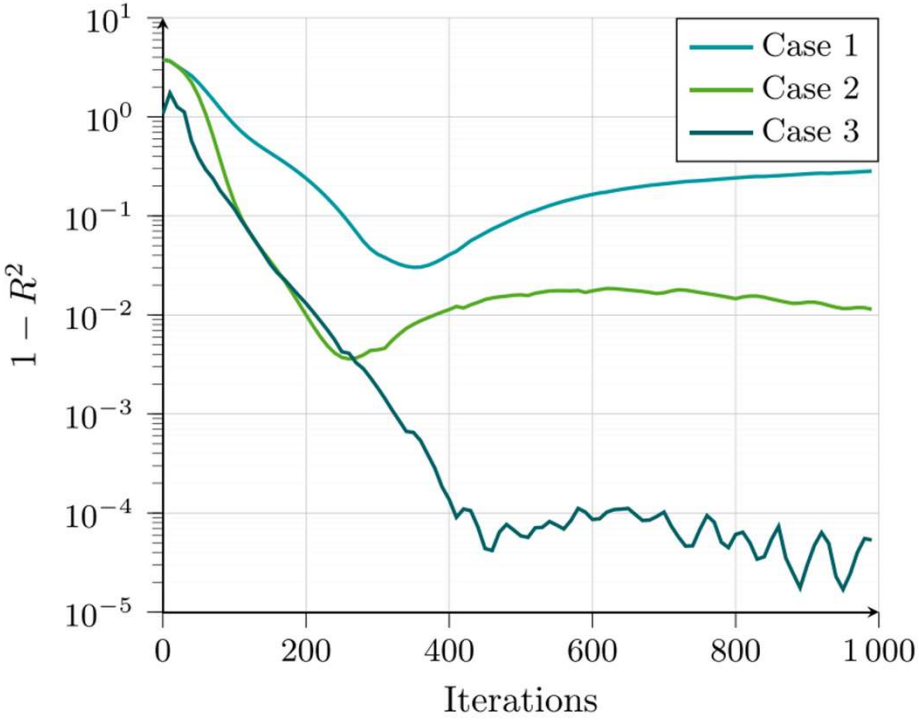
$$\frac{d^2u}{dx^2} + 4\pi^2 \sin(2\pi x) = 0$$

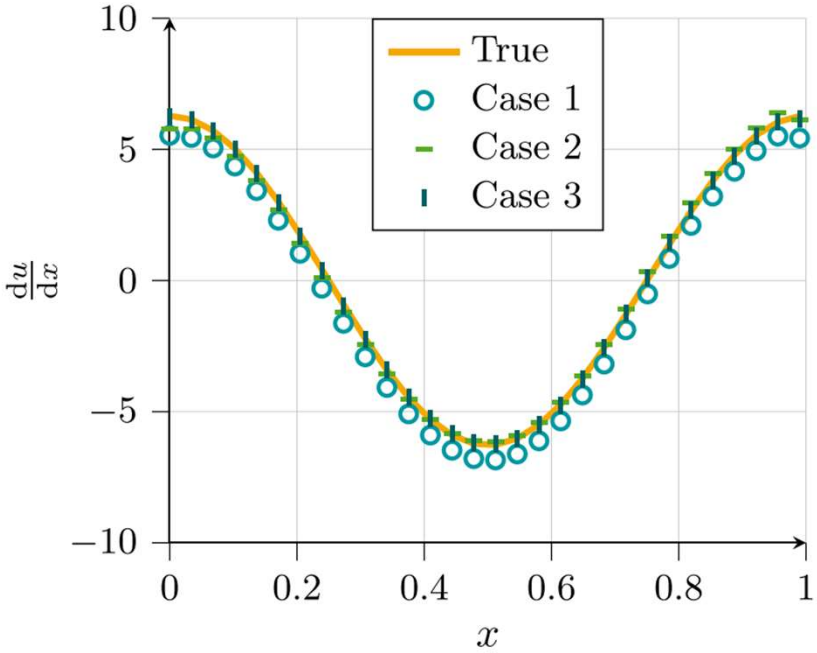
Loss term (here no floating boundary handling):

$$Loss = \frac{\alpha_f}{N_f}\left[\frac{d^2u}{dx^2} + 4\pi^2 \sin(2\pi x)\right]$$

$$+ \quad \alpha_b[(u(x=0)-0)^2 + (u(x=0.99) - sin2\pi 0.99)^2]$$







Case 1:

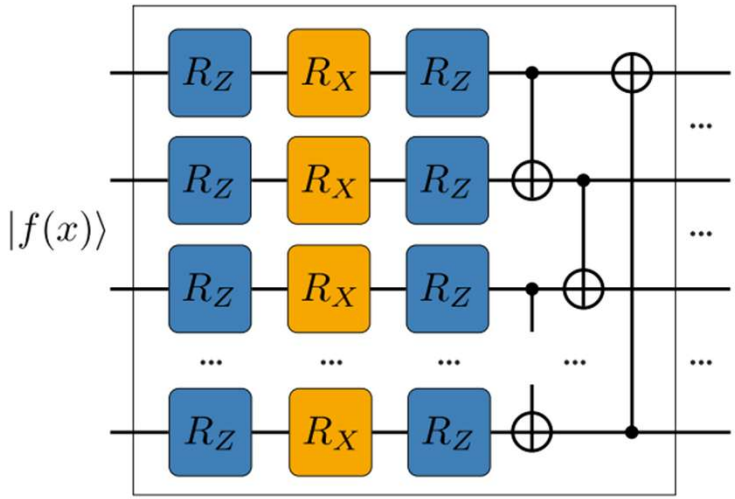
balanced weights of $\alpha_f = \alpha_b = 1$

Case 2:

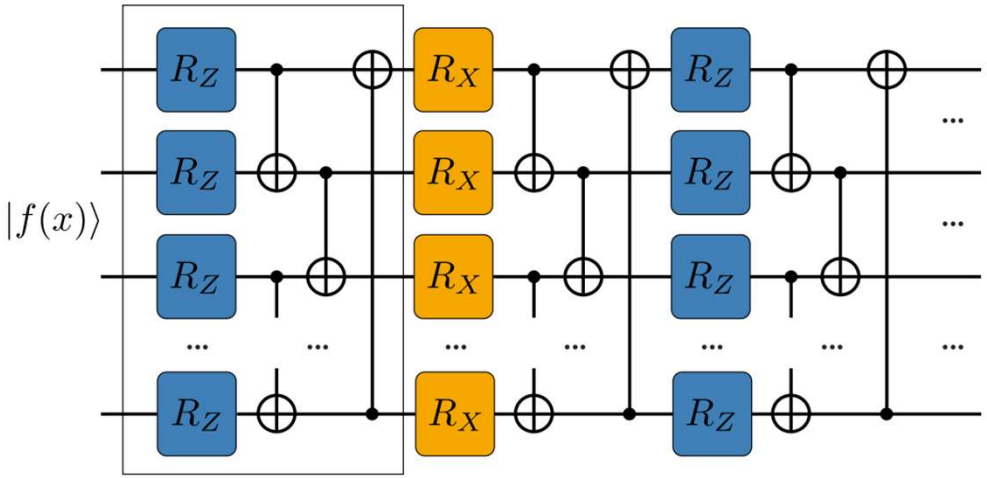
balanced weights of $\alpha_f = 10^{-1}$ and $\alpha_b = 10^3$

(No Self-adaptive yet!)

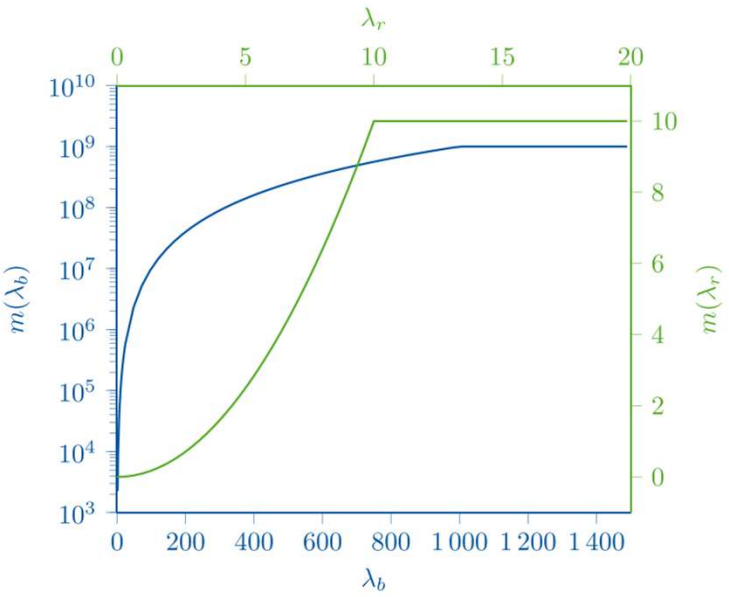
Case 1 and 2

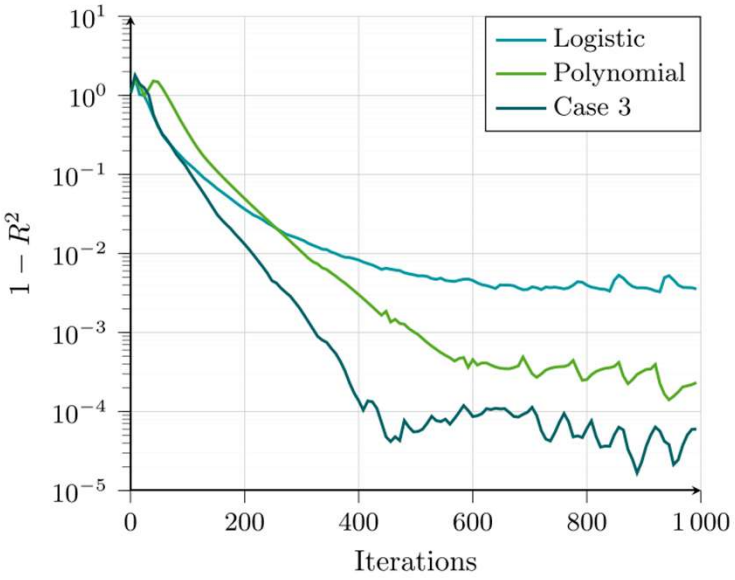


Case 3

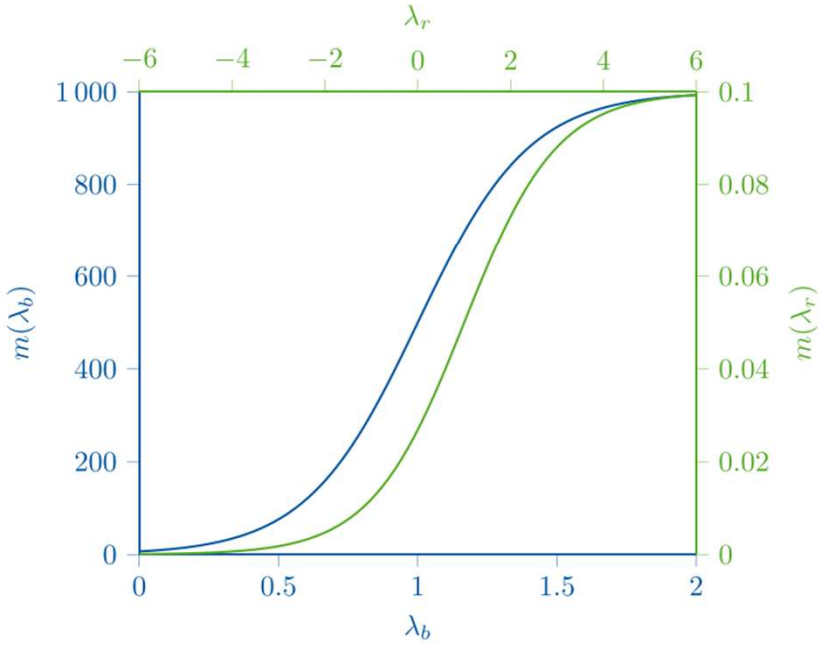


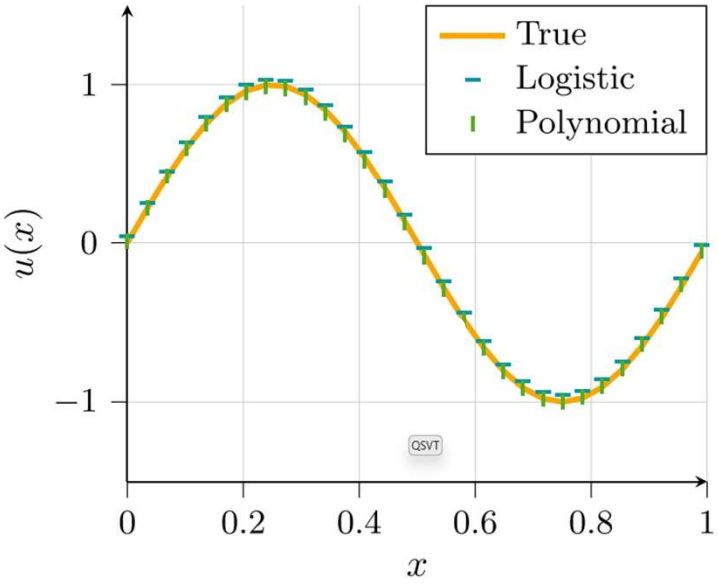
Polynomial Mask fn





Logistic Mask fn

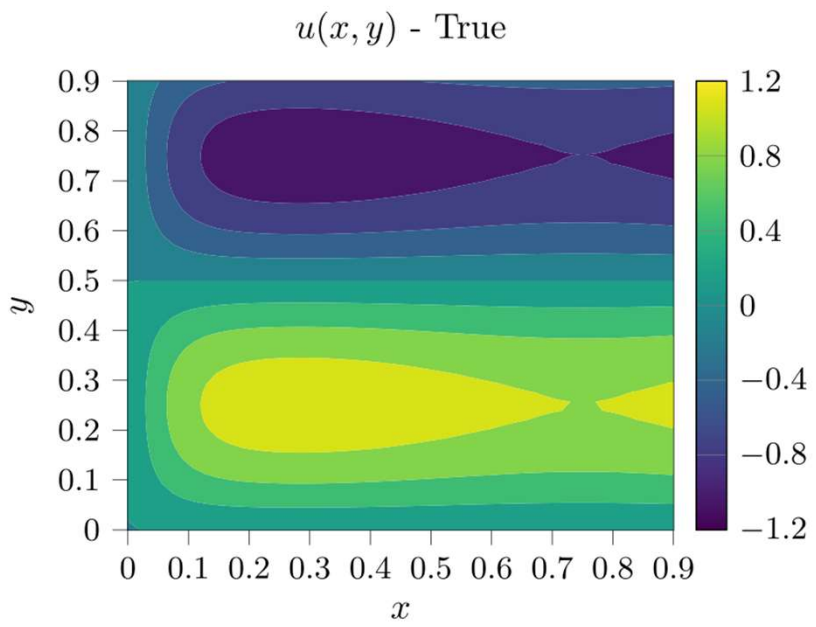


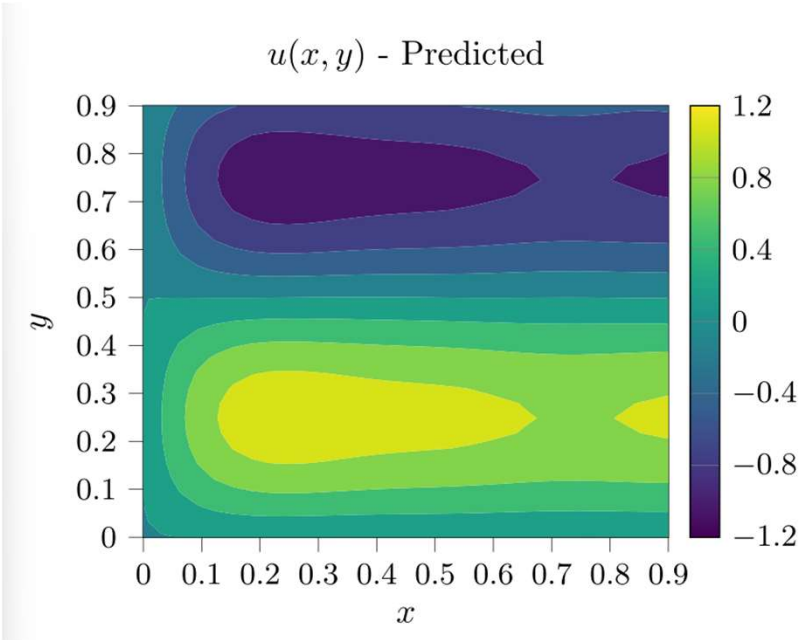


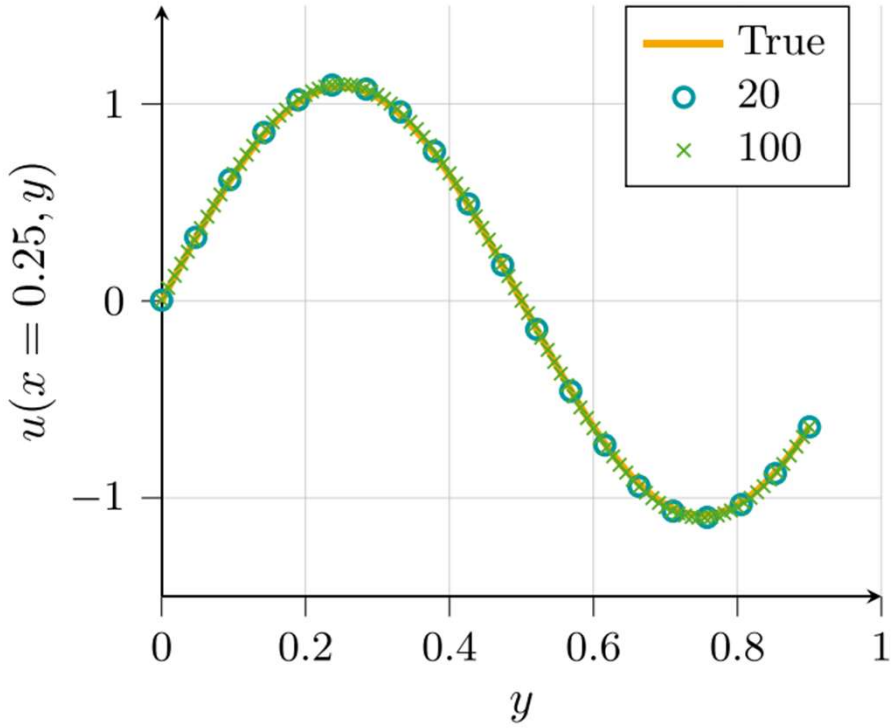
3. Second-order PDE:

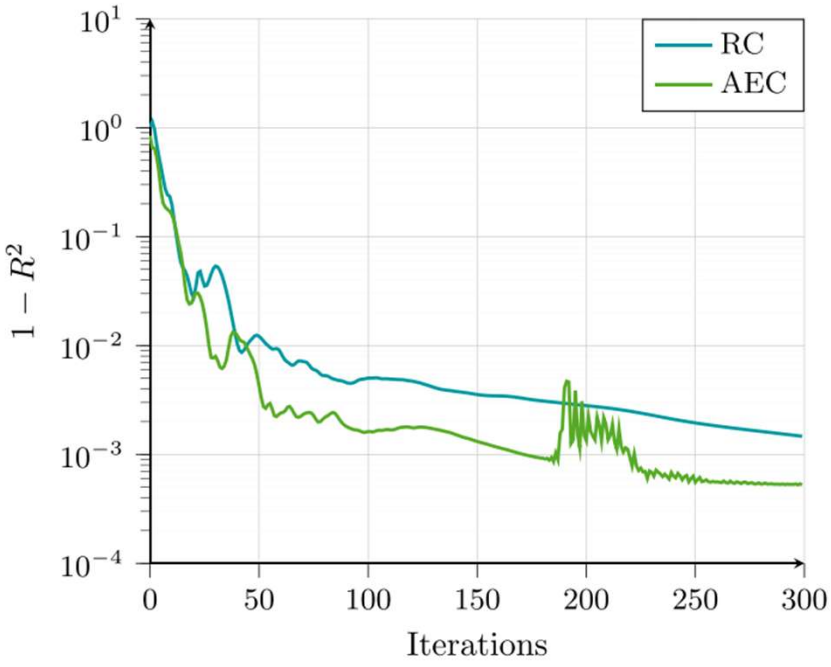
$$\Delta u(x,y) = g(x,y) \qquad \text{with} \quad x,y \in [0,0.9] \times [0,0.9]$$

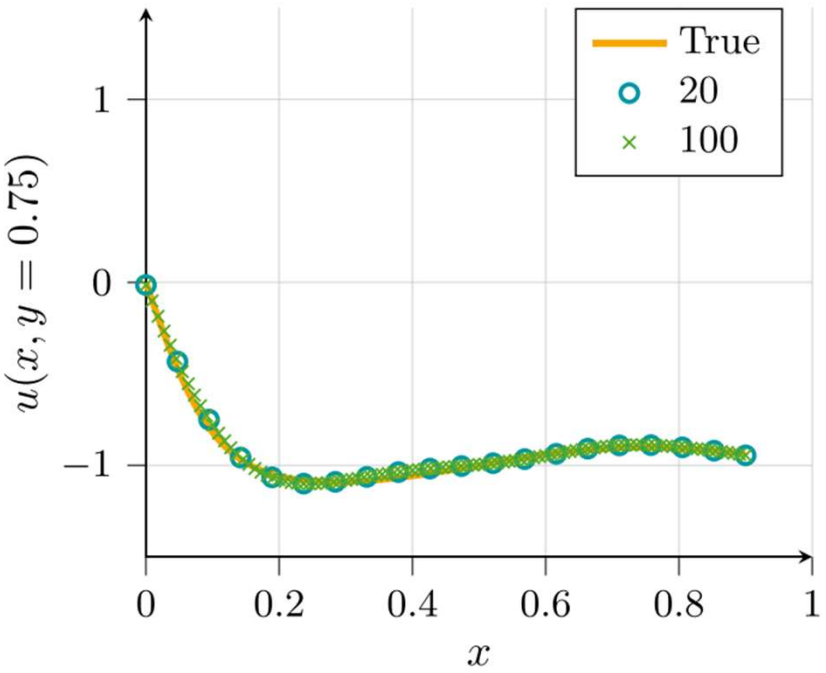
- 900 collocation points(i.e 30×30 equidistant points)
- Dirichlet boundary conditions given by $u_b(x,0), u_b(x,0.9), u_b(0,y), and\ u_b(0.9,y)$.
- $C = \alpha_f MSE_f + \alpha_b MSE_b \quad , \quad \alpha_f = 10^{-1}\ ; \alpha_b = 10^3$
- The circuit architecture for this problem is chosen to be 3 qubits for variable x and 3 qubits for variable y with a depth of 7 blocks in the variational circuit.


$u(x, y)$ - True
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
x
1.2
0.8
0.4
0
−0.4
−0.8
−1.2


$u(x, y)$ - Predicted
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
x
1.2
0.8
0.4
0
−0.4
−0.8
−1.2


True
20
100
$u(x = 0.25, y)$
1
0
−1
0 0.2 0.4 0.6 0.8 1
y


RC
AEC
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
$1 - R^2$
0
50
100
150
200
250
300
Iterations


True
20
100
1
0
−1
$u(x, y = 0.75)$
0
0.2
0.4
0.6
0.8
1
x

Conclusions:

Key Takeaways

-Differential equations are solved by minimizing a physics-informed loss function.

-The VQC acts as a quantum ansatz for the solution $u(x)$.

-Function values are extracted through expectation value measurements.

-PDE residuals require derivatives, obtained through the chain rule and parameter-shift rule.

-Training is performed using a hybrid quantum-classical optimization loop.

Outlook

-More expressive ansatz and feature maps (The number of layers d is directly proportional to the circuit's expressive power. However, as the parameters increase, the optimization will suffer with a problem known as barren plateau)

-Improved trainability and noise resilience.

-A further question that still needs to be addressed is whether a potential quantum advantage can be achieved.

Thank you!