Physics-informed neural networks

Case Study in Scientific Computing

1 Motivation

Partial differential equations (PDEs) model phenomena across the physical sciences, from fluid dynamics to quantum mechanics. Classical mesh-based solvers are well-established but face fundamental difficulties: discretisation cost grows exponentially with dimension, and complex geometries or multiscale solutions demand prohibitively fine meshes. Physics-informed neural networks (PINNs) [1], [2] offer a mesh-free alternative by parametrising the unknown solution as a neural network trained to minimise the PDE residual at collocation points, with boundary and initial conditions enforced through additional penalty terms or hard constraints. The neural network can then be evaluated at any point rather than only at discrete mesh points.

This report investigates two failure modes of vanilla PINNs. The first is *spectral bias* [3], [4], whereby gradient-based training learns low-frequency solution components far faster than high-frequency ones. We address this with random Fourier feature (RFF) embeddings [5] and compare standard and RFF-PINNs on the simple harmonic and van der Pol oscillators, supplementing the latter with curriculum learning when direct training fails. The second concerns boundary enforcement: soft penalty methods are sensitive to the penalty weight, and hard constraints are geometry-specific. We recast this as a constrained optimisation problem and apply the augmented Lagrangian method [6] to the two-dimensional Helmholtz equation, showing it outperforms other soft enforcement strategies while keeping the conditoin number bounded.

2 An introduction to neural networks

Artificial neural networks (ANNs) are models inspired by the function of biological neural networks in the brain. They consist of a series of nodes, connected together by edges. The nodes represent affine functions, and the edges represent activation functions. Figure 1 is a typical diagram of a *multi-layer perceptron (MLP)*, since each node in each layer is connected to every node in neighbouring layers, and there are no feedback loops.

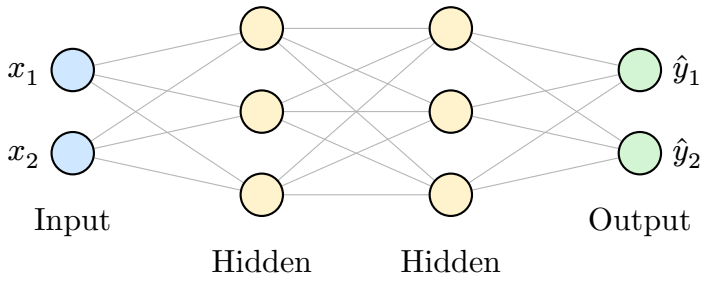

Figure 1: A fully-connected neural network with one input layer, two hidden layers, and one output layer. Each node in a layer is connected to every node in the adjacent layers.

The mathematical description of a MLP is as the composition of affine functions with non-linear activation functions. Let σ_i be an activation function. The technical condition for these to be useful is that they are non-polynomial [7]. Some examples are tanh, the sigmoid function, or the rectified linear unit (ReLU).

Let the input to $\mathbf{x} \in \mathbb{R}^d$. Then, a network with L total layers computes

$$\mathbf{h}_0 := \mathbf{x}$$

for each $\ell = 1, 2, ..., L-1$:

$$\begin{aligned} \mathbf{z}_\ell &= W_\ell \mathbf{h}_{\ell-1} + \mathbf{b}_\ell \\ \mathbf{h}_\ell &= \sigma_\ell(\mathbf{z}_\ell) \end{aligned} \tag{1}$$

where $W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weight matrix, $\mathbf{b}_\ell \in \mathbb{R}^{d_\ell}$ is the bias vector, and σ_ℓ is the activation function, applied element-wise to $\mathbf{z}_\ell$. The output is then $\mathbf{z}_L := W_L \mathbf{h}_{L-1} + \mathbf{b}_L$.

If we let $\mathbf{f}_\ell(\mathbf{z}) := W_\ell \mathbf{z} + \mathbf{b}_\ell$, the full neural network $N : \mathbb{R}^{d_0} \to \mathbb{R}^{d_L}$ can be written as

$$N = \mathbf{f}_L \circ \sigma_{L-1} \circ ... \circ \mathbf{f}_2 \circ \sigma_1 \circ \mathbf{f}_1 \tag{2}$$

where $\circ$ denotes function composition, so that $(f \circ g)(x) := f(g(x))$. The quantity d_ℓ is called the *size*, *width*, or *dimension* of layer ℓ. We usually refer to the layers with $\ell = 2, ..., L-1$ as the hidden layers, and the outer layers as the input and output layers. The values d_0 and d_L are the input and output dimensions, respectively. We collect all trainable parameters (the entries of each W_ℓ and $\mathbf{b}_\ell$) into a single parameter vector $\boldsymbol{\theta}$, which completely determines the neural network given fixed depth and widths.

Neural networks have achieved such widespread use due to two features: the universal approximation theorem, and backpropagation.

2.1 The universal approximation theorem

Informally, the universal approximation theorems (UATs) state that there always exists an MLP which can approximate many classes of functions to any desired degree of accuracy. Formally, we must define what these classes are, and what the accuracy metric is.

The function spaces L^p, C^r (r-times continuously differentiable functions), and H^k (Sobolev space of order k) are all endowed with canonical norms, which also induce metrics. Several papers, including [7], [8], [9], have introduced various universal approximation theorems for these spaces. The general form of a UAT is

Theorem 1 (General UAT): Let F be a function space with a metric ρ. Let Σ be a set of neural networks. Then, Σ is dense in F with respect to the metric ρ.

F can be L^p, C^r, H^k, M (Borel measurable functions), or some other useful function space. The metric varies, from the uniform norm on C^0: $\rho(f,g) \coloneqq \max_x |f(x) - g(x)|$ to the Ky-Fan metric for M defined in [10]. Σ often takes the form of an MLP with at least one hidden layer. The activation functions allowed in these theorems vary, including strictly sigmoid functions [8], squashing functions [7], and general non-polynomial functions [11].

In almost every physical and engineering scenario, we assume some sort of regularity condition on the solutions to our PDEs. Up to some quite pathological exceptions – for example the HJB equation with non-Borel-measurable reward functions [12] – the solution we want to approximate using a PINN is Borel measurable. For more precise details on the metrics used and the exact types of function approximators, see [7].

It is important to note that UATs are existence theorems, hence make no claims on how to find such an approximator. The process of finding a neural network which approximates a solution is called *training*.

2.2 Training

The aim of training a neural network is to find a parameter vector $\boldsymbol{\theta}$ such that the network output $N_{\boldsymbol{\theta}}(x)$ closely matches the desired output y across a set of training data $\{(x_i, y_i)\}_{i=1}^n$. This is formalised by defining a loss function

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \|N_{\boldsymbol{\theta}}(x_i) - y_i\|^2,$$

which measures the mean squared discrepancy between the network's predictions and the true outputs. Training then amounts to the optimisation problem $\boldsymbol{\theta}^* = \operatorname{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta})$, solved iteratively using gradient descent. Starting from an initial guess $\boldsymbol{\theta}_0$, the update rule at step k is usually of the form

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_k),$$

where $\eta_k > 0$ is the *learning rate* at step k. The practical challenge is computing $\nabla_{\boldsymbol{\theta}} \mathcal{L}$ efficiently, since $\boldsymbol{\theta}$ contains all entries of every W_ℓ and b_ℓ, so a naive finite-difference approximation would be prohibitively expensive, and would incur huge finite difference errors.

Automatic differentiation (AD) is a technique for evaluating exact derivatives of a composite function by applying the chain rule to each elementary operation. It produces derivatives efficiently and to machine precision, without approximation by finite differences. AD applied to neural networks is termed backpropagation, or sometimes reverse-mode AD. For a historical overview and debates on who should truly be cited for these methods, see [13].

Recalling (1), define the error at layer ℓ as

$$\boldsymbol{\delta}_\ell \coloneqq \nabla_{\mathbf{z}_\ell} \mathcal{L} \in \mathbb{R}^{d_\ell}.$$

The forward pass computes $\mathbf{h}_0, \mathbf{z}_1, \mathbf{h}_1, ..., \mathbf{z}_L, \mathbf{h}_L$ in the usual way according to (1). The backward pass then propagates errors from the output layer back to the input:

$$\begin{aligned}
\boldsymbol{\delta}_L &= \nabla_{\mathbf{z}_L} \mathcal{L}, \\
\boldsymbol{\delta}_\ell &= \left(W_{\ell+1}^T \boldsymbol{\delta}_{\ell+1}\right) \odot \sigma_\ell'(\mathbf{z}_\ell), \quad \ell = L-1, ..., 1,
\end{aligned}$$

where $\odot$ denotes the element-wise (Hadamard) product and σ_ℓ' is the derivative of the activation function applied element-wise. σ_ℓ' and $\nabla_{\mathbf{z}_L} \mathcal{L}$ are given explicitly, or can be calculated from their component parts using AD. The gradients with respect to the parameters of layer ℓ are then

$$\nabla_{W_\ell} \mathcal{L} = \boldsymbol{\delta}_\ell \mathbf{h}_{\ell-1}^{\mathrm{T}}, \quad \nabla_{\mathbf{b}_\ell} \mathcal{L} = \boldsymbol{\delta}_\ell.$$

Crucially, this computes all gradients in a single backward pass at a cost comparable to one forward pass, making training feasible even for networks with millions of parameters.

In practice, vanilla gradient descent is rarely used. Most methods use stochastic gradient descent, where the loss is only computed with respect to a subset of the data. The Adam optimiser [14], which maintains per-parameter adaptive learning rates using estimates of the first and second moments of the gradient, is one of the standard SGD algorithms for training neural networks.

3 PINNs

The term physics-informed neural network (PINN) refers to the application of neural networks for solving differential equations.

Suppose we seek to solve the general non-linear differential equation

$$\mathcal{N}[u] = f(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega. \tag{3}$$

with some sort of initial, boundary, or observational data. Collocation points are the discrete set of points in Ω at which the PDE residual and boundary conditions are evaluated during training, providing the pointwise constraints through which the network learns to satisfy the underlying equations. We write these collocation points as $\overline{\Omega} \subsetneq \Omega$, with $N_r := |\overline{\Omega}|$.

We denote by $\hat{u}_{\boldsymbol{\theta}}$ the approximation to u that we aim to learn. It is parametrised by the parameter vector $\boldsymbol{\theta}$. Then, we define the residual loss as

$$\mathcal{L}_r(\boldsymbol{\theta}) := \frac{1}{N_r} \sum_{i=1}^{N_r} \{\mathcal{N}[\hat{u}_{\boldsymbol{\theta}}](\mathbf{x}_i) - f(\mathbf{x}_i)\}^2.$$

The basic idea behind PINNs is to learn an approximation to (3) indirectly by minimizing the residual loss $\mathcal{L}_r$ since if $\hat{u}_{\boldsymbol{\theta}} = u$, then $\mathcal{L}_r(\boldsymbol{\theta}) = 0$, although the converse does not generally hold except at the discrete collocation points. A key advantage of PINNs is that arbitrary derivatives of the network output can be computed to machine precision using automatic differentiation, so the residual loss $\mathcal{L}_r$ is evaluated without finite-difference approximation error and is therefore exact up to machine precision.

3.1 Boundary and initial conditions

Beyond the PDE residual, the problem (3) is typically supplemented by boundary and initial conditions. These can (broadly) be enforced in one of two ways

The first is to give each condition an additional L^2 loss evaluated on its own set of collocation points. For a constraint of the form $\mathcal{B}[u] = g$ on $\partial\Omega$, with boundary collocation points $\overline{\partial\Omega} \subset \partial\Omega$, $N_{\text{BC}} \coloneqq \left|\overline{\partial\Omega}\right|$, the corresponding loss is

$$\mathcal{L}_{\text{BC}}(\boldsymbol{\theta}) \coloneqq \frac{1}{N_{\text{BC}}} \sum_{j=1}^{N_{\text{BC}}} \left\{\mathcal{B}[\hat{u}_{\boldsymbol{\theta}}](\mathbf{x}_j) - g(\mathbf{x}_j)\right\}^2.$$

Initial conditions are handled identically, with collocation points drawn from the initial time slice. Writing $\{\mathcal{L}_i\}_{i=1}^n$ for all non-residual losses, the total training objective is

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_r(\boldsymbol{\theta}) + \sum_{i=1}^{n} \lambda_i \mathcal{L}_i(\boldsymbol{\theta}), \tag{4}$$

where $\{\lambda_i\}_{i=1}^n > 0$ are weights chosen prior to training. Each $\mathcal{L}_i$ vanishes when $\hat{u}_{\boldsymbol{\theta}}$ exactly satisfies the corresponding constraint, so minimising $\mathcal{L}$ should drive $\hat{u}_{\boldsymbol{\theta}}$ closer to u in the L^2 norm.

The second method for enforcing boundary conditions is by transforming the output of the neural network, which we refer to as "hard" constraints. This is generally only doable on simple geometries. For example, suppose we have the 1-D IVP

$$\begin{aligned} u''(x) + u(x) &= 0 \\ u(0) = 1, \quad u'(0) &= 0 \end{aligned} \tag{5}$$

We would enforce the boundary conditions by setting $\hat{u}_{\boldsymbol{\theta}}(x) = 1 + x(N_{\boldsymbol{\theta}}(x) - N_{\boldsymbol{\theta}}(0))$ where $N_{\boldsymbol{\theta}} : \mathbb{R} \to \mathbb{R}$ is an unconstrained neural network. For all neural networks $N_{\boldsymbol{\theta}}$, $\hat{u}'_{\boldsymbol{\theta}}(0) = 0$ and $\hat{u}_{\boldsymbol{\theta}}(0) = 1$, so the final loss in this case is just the residual loss $\mathcal{L}_r$.

4 Spectral bias and random Fourier features

A well-documented failure mode of gradient-based training is *spectral bias* [3], [4], where neural networks learn low-frequency components of a function much faster than high-frequency ones. For PDE solutions that are highly oscillatory, this can

prevent a PINN from converging to an accurate approximation in a reasonable amount of time.

Motivated by the same problem in kernel methods [15], random Fourier features (RFFs) address this by replacing the raw network input with a fixed random embedding that explicitly encodes a prescribed range of frequencies. Given input $\mathbf{x} \in \mathbb{R}^d$, the RFF embedding of dimension $2m$ is

$$\varphi(\mathbf{x}) = (\cos(\omega_1^T \mathbf{x}), \sin(\omega_1^T \mathbf{x}), ..., \cos(\omega_m^T \mathbf{x}), \sin(\omega_m^T \mathbf{x})) \in \mathbb{R}^{2m},$$

where the frequency vectors $\omega_1, ..., \omega_m$ are sampled i.i.d. from a distribution p and held fixed throughout training. The network then takes $\varphi(\mathbf{x})$ as its input in place of $\mathbf{x}$.

The choice of p determines which frequencies the embedding emphasises. In practice, $\omega_j \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$ is a common choice. The hyperparameter σ can be viewed as the bandwidth of the Fourier features. A wider bandwidth allows the network to learn more oscillatory functions, but may also introduce noise if the frequencies are too high [5].

We now present some computational experiments using random Fourier features.

4.1 Simple harmonic oscillator

We consider the initial value problem

$$u''(t) + (\omega\pi)^2 u(x) = 0, \quad x \in [0, 1], \quad u(0) = 1, \quad u'(0) = 0, \tag{6}$$

whose unique solution is $u(t) = \cos(\omega\pi x)$. We compare a standard PINN against an RFF-PINN across several values of ω.

Architecture. Both models use an MLP with two hidden layers of width 64, tanh activation functions throughout, and Xavier initialisation [16]. The RFF-PINN prepends a fixed random Fourier feature layer with $p = 32$ frequencies and bandwidth $\sigma = 2$ to the same MLP, replacing the first linear layer. Both initial conditions are enforced exactly via the ansatz

$$\hat{u}_{\boldsymbol{\theta}}(x) \coloneqq 1 + t(N_{\boldsymbol{\theta}}(x) - N_{\boldsymbol{\theta}}(0)), \tag{7}$$

where $N_{\boldsymbol{\theta}}$ is the raw network output, so that $\hat{u}_{\boldsymbol{\theta}}(0) = 1$ and $\hat{u}'_{\boldsymbol{\theta}}(0) = 0$ for all $\boldsymbol{\theta}$. Therefore, no boundary penalty terms are required.

Training. The sole loss is the PDE residual

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N}\left[\hat{u}''_{\boldsymbol{\theta}}(x_i) + (\omega\pi)^2\hat{u}_{\boldsymbol{\theta}}(x_i)\right]^2, \tag{8}$$

evaluated on $N = 1000$ uniformly-spaced collocation points. Parameters are updated with Adam ($\eta = 10^{-3}$). A cosine-annealing schedule [17] is used, which we found to greatly shorten training times uniformly for both regular and RFF PINNs, therefore not impacting the comparative results.

Setup and results. First, a single RFF-PINN is trained with $\omega = 4$ to tolerance $\mathcal{L}(\boldsymbol{\theta}) < 5 \times 10^{-2}$. The learned solution is compared pointwise against $\cos(4\pi t)$ in Figure 2, showing that the solution is learned well.

Next, the we vary $\omega \in \{1, 2, 3, 4\}$ at tolerance $\mathcal{L}(\boldsymbol{\theta}) < 5 \times 10^{-2}$, recording iterations to convergence for each method. Results are shown in Figure 3. We see that the RFF layer is detrimental for $\omega = 1, 2$. This is likely because large frequencies introduced by the RFF layer are not needed, so "tuning them out" requires more training time than learning the function of interest. As ω increases, the training time for the regular PINN increases significantly (note the logarithmic scaling), while the RFF-PINN scales much better, and even turns downwards between $\omega = 3$ and $\omega = 4$, demonstrating the effectiveness of RFF layers in learning highly oscillatory functions.

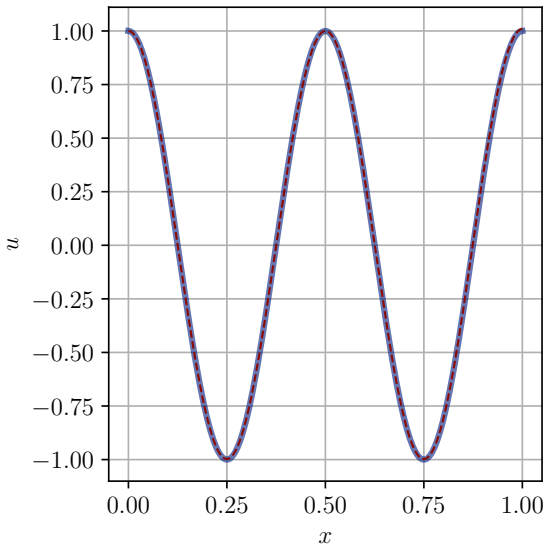

Figure 2: Example solution of (6) with RFF-PINN solution overlay for $\omega = 4$. Blue solid: Exact solution $\cos(4\pi x)$; Red dashed: RFF-PINN solution.

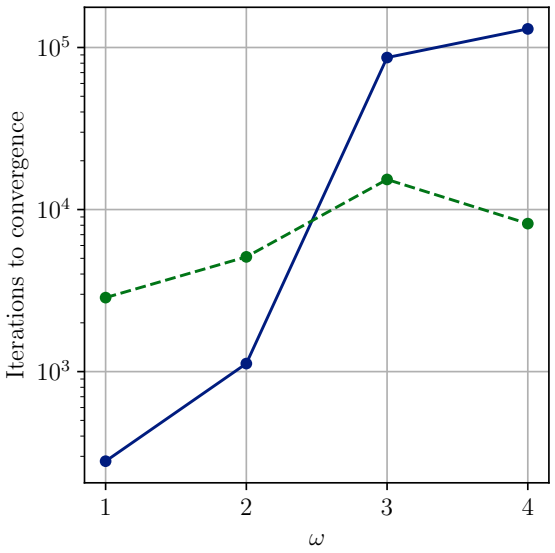

Figure 3: Number of iterations till convergence for a regular PINN and a RFF-PINN for varying values of ω. Blue solid: Regular PINN; Green dashed: RFF-PINN.

4.2 Van der Pol oscillator

We next apply the same approach to the van der Pol (VDP) oscillator

$$u''(t) = \mu(1 - u(t)^2)u'(t) - u(t), \quad t \in [0, 10], \quad u(0) = 2, \quad u'(0) = 0, \tag{9}$$

where $\mu \geq 0$ controls nonlinear damping. At $\mu = 0$ (9) reduces to the simple harmonic oscillator. For large μ the solution exhibits sharp fronts that are difficult for gradient-based methods to resolve. Since (9) admits no closed-form solution, a fourth-order Runge–Kutta integrator with 20,000 steps serves as the reference. Figure 4 shows phase portraits of the van der Pol oscillator for varying values of μ, demonstrating the sharp changes in gradient as μ grows.

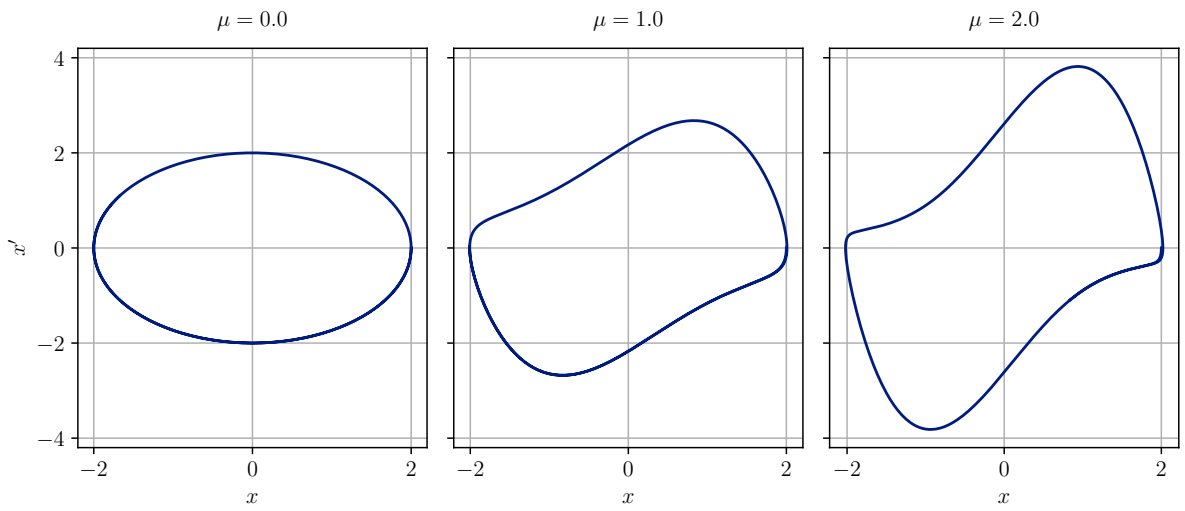

Figure 4: Phase portraits of the van der Pol oscillator for varying values of μ.

Architecture. The models are identical to Section 4.1 except that the RFF bandwidth is $\sigma = 1$ and the hard-enforcement ansatz is

$$\hat{u}_{\boldsymbol{\theta}}(t) := 2 + t(N_{\boldsymbol{\theta}}(t) - N_{\boldsymbol{\theta}}(0)) \tag{10}$$

so that $\hat{u}_{\boldsymbol{\theta}}(0) = 2$ and $\hat{u}'_{\boldsymbol{\theta}}(0) = 0$ for any $\boldsymbol{\theta}$.

Training. The residual loss is

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \left[\hat{u}''_{\boldsymbol{\theta}}(t_i) - \mu\left(1 - \hat{u}_{\boldsymbol{\theta}}(t_i)^2\right) \hat{u}'_{\boldsymbol{\theta}}(t_i) + \hat{u}_{\boldsymbol{\theta}}(t_i) \right]^2, \tag{11}$$

evaluated on $N = 2000$ uniformly-spaced points on $[0, 10]$. The optimiser settings are unchanged (Adam, $\eta = 10^{-3}$, cosine annealing), with tolerance 5×10^{-3}.

Setup and results. In Figure 5, a single RFF-PINN is trained at $\mu = 2$, and its time series and phase portrait are compared against the RK4 reference. While are not an exact visual match, the approximation is clearly close enough that further training would converge, and stopping this early meant that experiments could be completed in a reasonable amount of time.

Figure 6 shows the loss $\mathcal{L}$ and the L^2 error as the RFF-PINN trains for varying values of μ. We see that as μ increases, the loss curves become less and less steep, and have more pronounced cliff-like behaviour, which is especially visible in the $\mu = 1.5$ case. Cliff-like behavior in the training loss is consistent with the optimiser encountering sharp, non-convex regions of the loss landscape rather than a smooth

descent path. While such behavior may indicate that the optimiser is stuck in a local minimum, it can also arise from saddle regions or ill-conditioned valleys, which are known features of neural-network loss landscapes [18], [19].

Figure 7 compares the number of training iterations required for convergence by the standard PINN and the RFF-PINN across varying values of μ. The RFF-PINN consistently converges in fewer iterations than the standard PINN, and its iteration count grows more slowly as μ increases. This improvement is likely because larger values of μ induce larger gradients, corresponding to higher-frequency components in the solution spectrum, which the Fourier feature embedding is better able to represent.

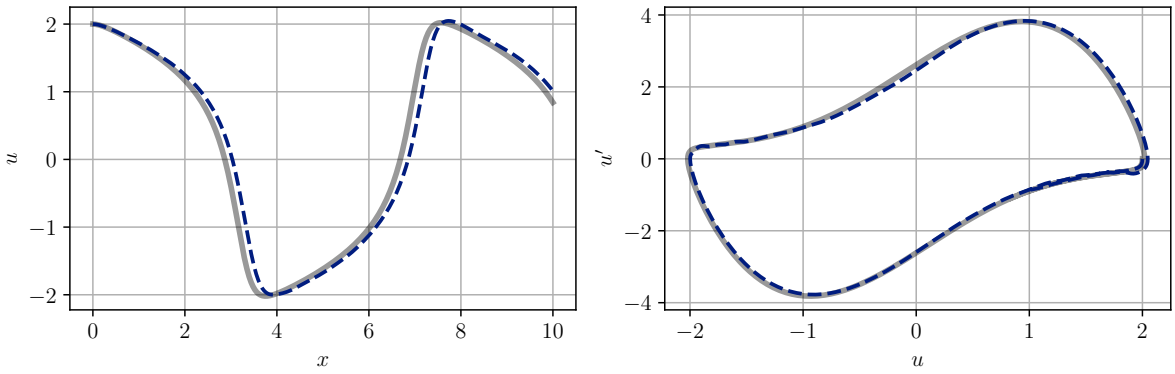

Figure 5: Example solution of the van der Pol oscillator with $\mu = 2$. Grey solid: RK4 solution; Blue dashed: RFF-PINN solution.

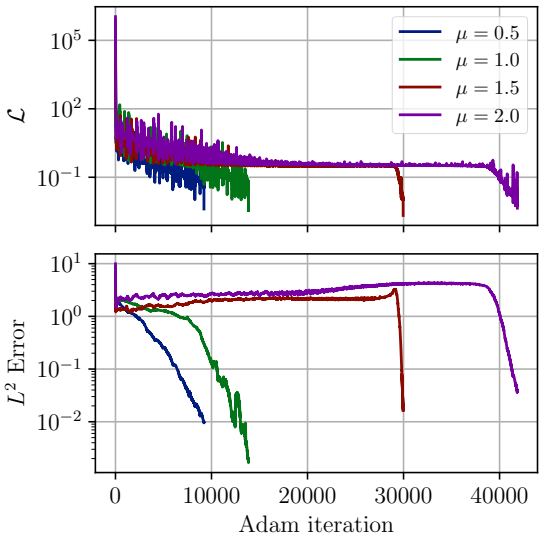

Figure 6: *Top*: Loss $\mathcal{L}$ as per (11); *Bottom*: L^2 error between the RFF-PINN solution and RK4 solution.

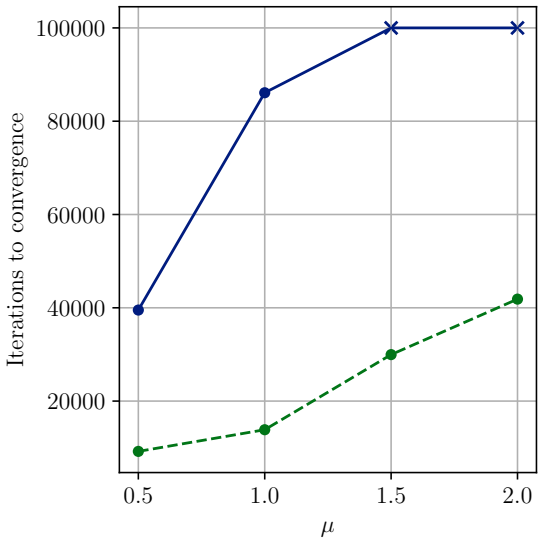

Figure 7: Number of iterations till convergence for a regular PINN and a RFF-PINN applied to the VDP oscillator with varying values of μ. Blue solid: Regular PINN, crosses indicate that the regular PINN did not converge within 10^5 iterations; Green dashed: RFF-PINN.

5 Curriculum learning

For the same van der Pol system (9), standard PINN training on (9) does not converge within 10^5 iterations for $\mu \geq 1.5$, so new approaches are needed for $\mu > 2$. We apply curriculum learning [20], [21] to overcome this, training the network on a sequence of progressively harder problems by incrementally increasing μ.

Architecture. The model is a tanh MLP with four hidden layers of width 64 and Xavier initialisation, with the same hard-enforcement ansatz as (10):

$$\hat{u}_{\boldsymbol{\theta}}(t) := 2 + t(N_{\boldsymbol{\theta}}(t) - N_{\boldsymbol{\theta}}(0)). \tag{12}$$

Training. The residual loss is identical to (11). Parameters are updated with Adam ($\eta = 10^{-3}$) under a cosine-annealing schedule that resets at the start of each curriculum stage. Training uses $N = 1000$ uniformly-spaced collocation points on $[0, 10]$.

Curriculum procedure. Training begins at $\mu = 0$ and proceeds in stages. At each stage the network is trained until the residual loss falls below the threshold

$\tau = 5 \times 10^{-4}$, at which point μ is incremented by 0.1 and training continues from the current parameters $\boldsymbol{\theta}$. This is repeated until the target $\mu = 5$ is reached.

Setup and results. Figure 8 shows the curriculum-trained solution at $\mu = 5$ alongside the RK4 reference, together with the residual loss and L^2 error plotted continuously across all stages. The network successfully learns the van der Pol oscillator up to $\mu = 5$ in approximately 93,800 total epochs. Compared to the over 100,000 epochs required for $\mu = 2$ in Figure 7, curriculum learning greatly speeds up convergence.

Integer milestones are reached at approximately 10,500 epochs ($\mu = 1$), 29,800 epochs ($\mu = 2$), 59,200 epochs ($\mu = 3$), and 76,500 epochs ($\mu = 4$). Interestingly, we see a slight decrease in L^2 error over time even though the loss, by definition, does not decrease as μ increases.

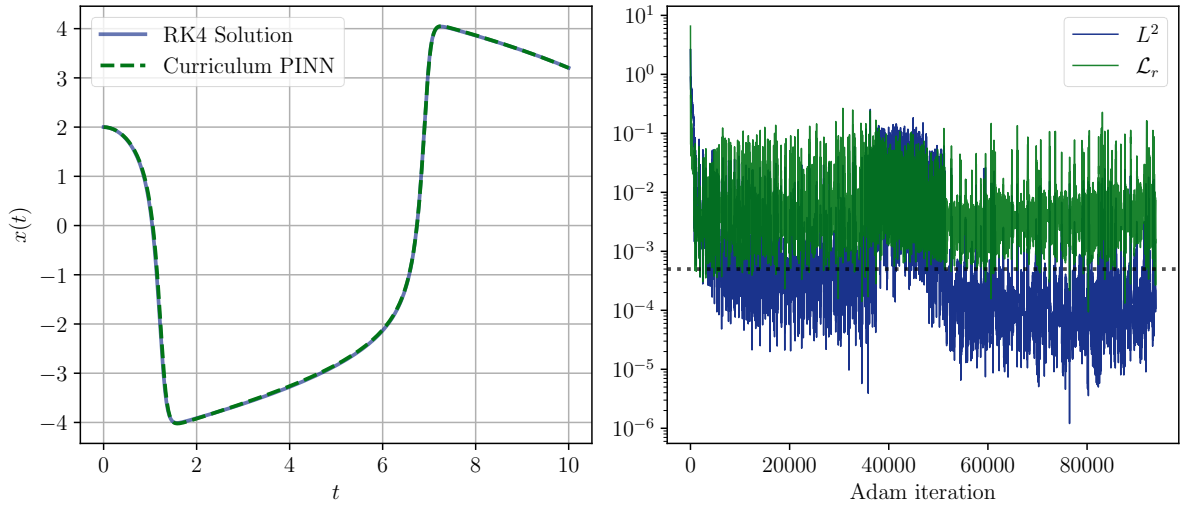

Figure 8: Curriculum learning for the van der Pol equation with $\mu = 5$. *Left*: PINN solution overlaid on the exact solution; *Right*: Residual loss $\mathcal{L}_r$ and L^2 error over training process.

6 Extension: PINNs as a constrained optimisation problem

This is the individual extension portion of the project, and aims to recreate some results from H. Son *et al.* [6]. Recall the issue of enforcing boundary conditions presented in Section 3. While we have looked at this as an unconstrained minimisation problem so far, we could also look at this as a constrained minimisation problem, where the initial/boundary conditions are the constraints.

Formally, for the problem (3) with boundary conditions

$$\mathcal{B}[u] = g \quad \text{on} \quad \partial\Omega$$

and with boundary collocation points $\{x_i\}_{i=1}^{N_B}$, $x_i \in \overline{\partial\Omega}$, consider the constrained optimisation problem

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathcal{L}_r(\boldsymbol{\theta}) \\ &\text{subject to} \\ &\mathcal{B}[\hat{u}_{\boldsymbol{\theta}}](x_i) = g(x_i) \quad \text{for all} \quad i = 1, ..., N_B. \end{aligned} \tag{13}$$

We then write $\mathbf{b}(\boldsymbol{\theta}) \coloneqq \left(b_1(\boldsymbol{\theta}), ..., b_{N_B}(\boldsymbol{\theta})\right)^{\mathrm{T}}$ for $b_i(\boldsymbol{\theta}) \coloneqq \mathcal{B}[\hat{u}_{\boldsymbol{\theta}}](x_i) - g(x_i)$ so that (13) becomes

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathcal{L}_r(\boldsymbol{\theta}) \\ &\text{subject to} \\ &\mathbf{b}(\boldsymbol{\theta}) = \mathbf{0} \end{aligned} \tag{14}$$

This is a constrained optimisation problem with $N_B = \left|\overline{\partial\Omega}\right|$ constraints. This motivates the use of classical constrained optimisation methods.

6.1 Quadratic penalty method

The quadratic penalty (QP) method [22] is a method for solving (14) by constructing the quadratic penalty function (QPF)

$$\Phi_\beta(\boldsymbol{\theta}) \coloneqq \mathcal{L}_r(\boldsymbol{\theta}) + \beta \|\mathbf{b}(\boldsymbol{\theta})\|^2$$

for $\beta > 0$. If we define $\{\beta\}_{n=0}^{\infty}$, $\beta_n \to \infty$, and

$$\boldsymbol{\theta}_n \coloneqq \arg\min_{\boldsymbol{\theta}} \Phi_{\beta_n}(\boldsymbol{\theta}),$$

it has been shown that [22], under suitable conditions, if $\lim_{n\to\infty} \boldsymbol{\theta}_n =: \boldsymbol{\theta}^*$ exists, $\boldsymbol{\theta}^*$ is a Karush–Kuhn–Tucker (KKT) point of (14). The suitable conditions are the linear independence constraint qualification (LICQ), which requires that $\nabla b_i(\boldsymbol{\theta})$ are all linearly independent at the KKT point. This is unlikely to hold becaude the constraints $b_i(\boldsymbol{\theta})$ usually only differ by collocation points. If we have many collocation points, it is likely that nearby constraints are exactly or almost linearly dependent.

While we are not guaranteed to converge to a local minimum, it is still worth testing this experimentally. A more practical downside of this method is that as $\beta_n \to \infty$, the Hessian of Φ_β becomes ill-conditioned, with the condition number of the Hessian also tending to ∞.

6.2 Augmented Lagrangian method

The augmented Lagrangian (AL) method for (14) is a modification of the QP method with

$$\Phi(\boldsymbol{\theta}; \boldsymbol{\lambda}, \beta) = \mathcal{L}_r(\boldsymbol{\theta}) + \boldsymbol{\lambda}^{\mathrm{T}} \mathbf{b}(\boldsymbol{\theta}) + \beta \|\mathbf{b}(\boldsymbol{\theta})\|^2.$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{N_B}$ is the vector of Lagrange multipliers. If we define $\{\beta_n, \boldsymbol{\lambda}_n\}_{n=0}^{\infty}$ so that $\beta_n \geq \beta_{\min} > 0$ and $\boldsymbol{\lambda}_n \to \boldsymbol{\lambda}$, where $\boldsymbol{\lambda}$ is the exact vector of Lagrange multipliers of (14), and $\boldsymbol{\theta}_n \to \boldsymbol{\theta}^*$ exists, then assuming the LICQ again, $\boldsymbol{\theta}^*$ is a KKT point. Again, this theorem [22, Thm. 22] does not guarantee convergence in our case for the same reasons as for the QP method. However, it does give us a method for finding $\boldsymbol{\theta}_n$. By [6], (14) becomes the unconstrained problem

$$\max_{\boldsymbol{\lambda}} \min_{\boldsymbol{\theta}} \Phi(\boldsymbol{\theta}; \boldsymbol{\lambda}, \beta).$$

So, we can solve this by simultaneously performing gradient descent on $\boldsymbol{\theta}$ and gradient ascent on $\boldsymbol{\lambda}$. The advantage of the AL method over the QP method is that β can remain bounded (in our case constant), so that the condition number of the Hessian of Φ also remains bounded [22].

6.3 Experiments

We present some experiments comparing the quadratic penalty and augmented Lagrangian methods with methods introduced in Section 3 as applied to the Helmholtz equation.

Helmholtz equation. We study the 2-D Helmholtz equation as in [6]:

$$\begin{aligned} \Delta u + u &= f(x, y), \qquad && (x, y) \in \Omega = (-1, 1)^2 \\ u(x, y) &= 0, && (x, y) \in \partial\Omega \end{aligned} \tag{15}$$

with the forcing function $f(x, y) = \left(1 - 17\pi^2\right) \sin(\pi x) \sin(4\pi y)$ so that $u(x, y) = \sin(\pi x) \sin(4\pi y)$, which is plotted in Figure 9. The high-frequency y-component creates spectral bias difficulties for regular MLPs.

Architecture. All five methods share a 2–256–256–1 tanh MLP with Xavier initialisation. They differ only in how the Dirichlet condition is imposed. The hard method transforms the network output using

$$\hat{u}_{\boldsymbol{\theta}}(x, y) := (1 - x^2)(1 - y^2) N_{\boldsymbol{\theta}}(x, y),$$

which vanishes everywhere on $\partial\Omega$, so no boundary loss is needed. All the other methods use the boundary penalty term

$$\mathcal{L}_{\mathrm{BC}}(\boldsymbol{\theta}) = \frac{1}{N_B} \sum_{j=1}^{N_B} \left[\hat{u}_{\boldsymbol{\theta}}(x_j)\right]^2$$

with either a fixed weight $\beta_n \equiv \beta \in \{1, 500\}$ (as in the soft boundary enforcement of Section 3.1) or an increasing $\beta_n := n \to \infty$ (as in the QP method).

The AL method uses $\beta_n \equiv 500$ together with adaptive multipliers $\boldsymbol{\lambda}_n$. This gives the following losses:

Method	**Loss $\mathcal{L}(\boldsymbol{\theta})$**
Hard	$\mathcal{L}_r(\boldsymbol{\theta})$
Soft	$\mathcal{L}_r(\boldsymbol{\theta}) + \beta \mathcal{L}_{\mathrm{BC}}$
Quadratic penalty (QP)	$\mathcal{L}_r(\boldsymbol{\theta}) + \beta_n \mathcal{L}_{\mathrm{BC}}, \quad \beta_n \to \infty$
Augmented Lagrangian (AL)	$\mathcal{L}_r(\boldsymbol{\theta}) + 500 \mathcal{L}_{\mathrm{BC}} + \frac{1}{N_B} \sum_j \lambda_j \hat{u}_{\boldsymbol{\theta}}(x_j)$

Training. The residual loss is

$$\mathcal{L}_r(\boldsymbol{\theta}) = \frac{1}{N_R} \sum_{i=1}^{N_R} \left[\Delta \hat{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) + \hat{u}_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - f(\boldsymbol{x}_i)\right]^2, \tag{16}$$

evaluated on $N_R = 2500$ uniformly sampled interior points and $N_B = 200$ boundary points. All methods are trained for 2×10^4 Adam steps with $\eta = 10^{-4}$. The AL method also uses the Adam optimiser for gradient ascent on $\boldsymbol{\lambda}$ and a learning rate $\eta_\lambda = 10^{-4}$.

Setup and results. Figure 10 shows absolute error heatmaps for all five methods alongside the exact solution. Soft boundary enforcement with $\beta = 1$ fails to enforce the boundary conditions, leaving visible errors along the edges. Soft with $\beta = 500$ and

the AL method are indistinguishable when compared to the $\beta = 1$ case, with the QP method falling slightly behind on the interior.

Figure 11 shows residual loss $\mathcal{L}_r$ and boundary loss $\mathcal{L}_{\text{BC}}$ throughout training. The AL-PINN suppresses boundary loss as effectively as soft $\beta = 500$ while maintaining comparable residual accuracy. The QP method forces boundary loss to near zero early but at the cost of a stagnating residual. The soft $\beta = 1$ method and the QP method are opposites in some sense, in that they prioritise the residual and the boundary loss respectively. This results in the loss not being prioritised being higher, but also being more stable, which can be seen from the roughness of the graphs. It is notable that the AL method outperforms the soft $\beta = 500$ method in both residual and boundary loss.

Figure 12 plots the L^2 error for the same methods. We see that the soft $\beta = 1$ method underperforms all other methods by an order of magnitude. The QP and soft $\beta = 500$ are next. The AL method seems to have the lowest L^2 loss at the final iteration, but from the left plot in Figure 12 we see that the hard method actually outperforms the AL method, and this is likely a fluke of where training was stopped.

The key observation so far is that, for the same $\beta = 500$ value, the addition of the Lagrange multiplier maximisation causes the AL method to significantly outperform the soft method in both loss minimisation and L^2 error. This result is consistent with the findings of [6].

Figure 13 shows the condition number for the same five PINNs over iterations 4000-20000. The first 4000 iterations are omitted due to very large condition numbers, which make the late-time behaviour harder to see. We see that the condition number grows with β, and that the AL method only very slightly increases the condition number as compared to the soft $\beta = 500$ case. As expected, the only method with a growing κ is the QP method, where we expect the condition number to grow without bound with β. This also shows that the asymptotics discussed in [22] are relevant to neural net applications, since this behaviour begins after roughly 4000 iterations.

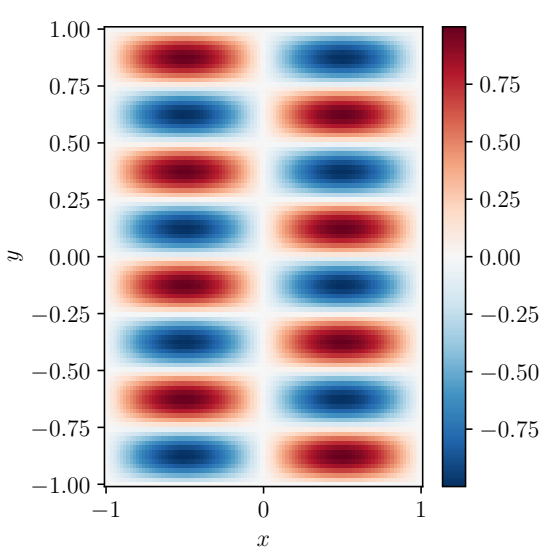

Figure 9: Heatmap of the exact solution of the Helmholtz equation (15).

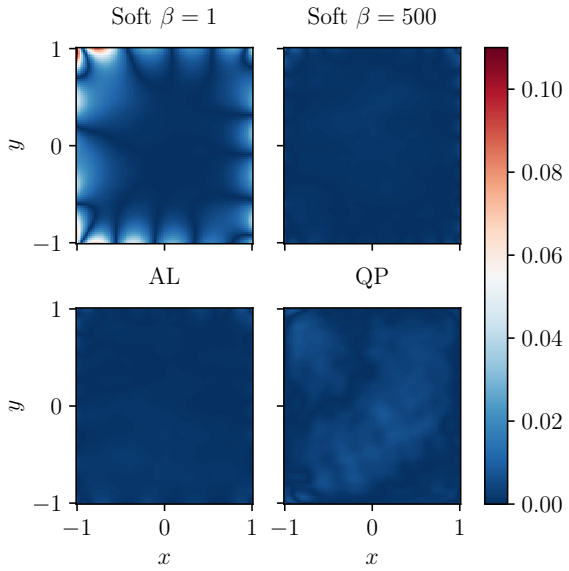

Figure 10: Heatmap of the error of four PINNs.

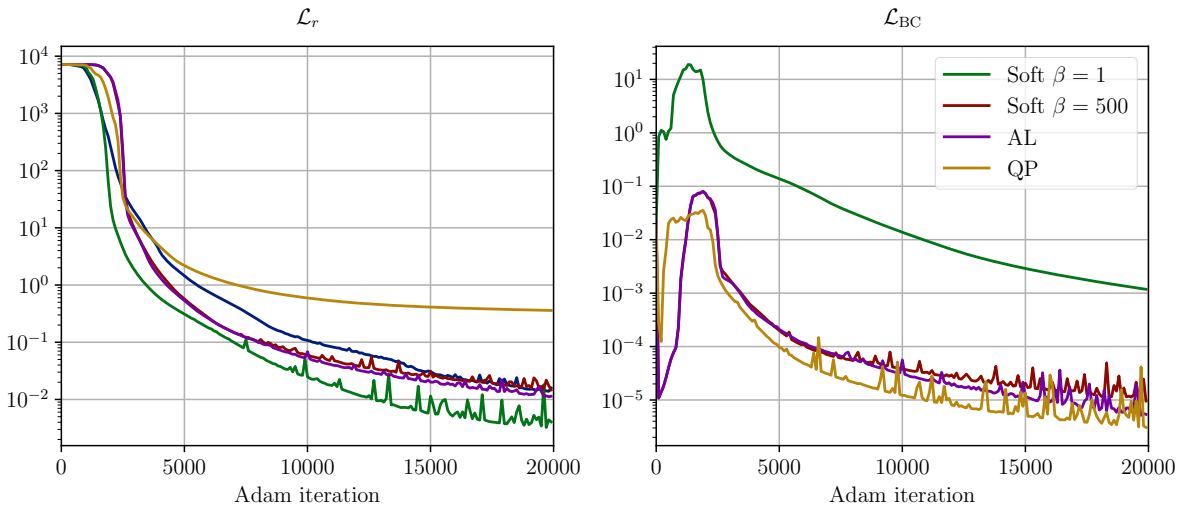

Figure 11: Losses over 20000 Adam iterations for five different PINNs. *Left*: residual loss; *Right*: boundary loss.

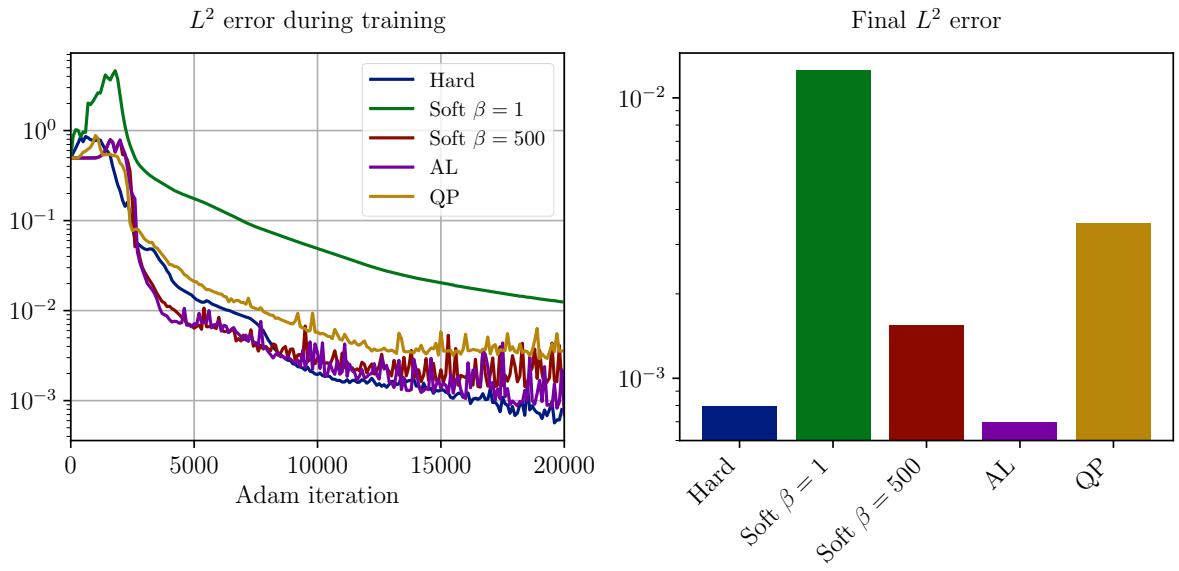

Figure 12: *Left*: L^2 error over 20000 Adam iterations for five different PINNs. *Right*: Final L^2 error.

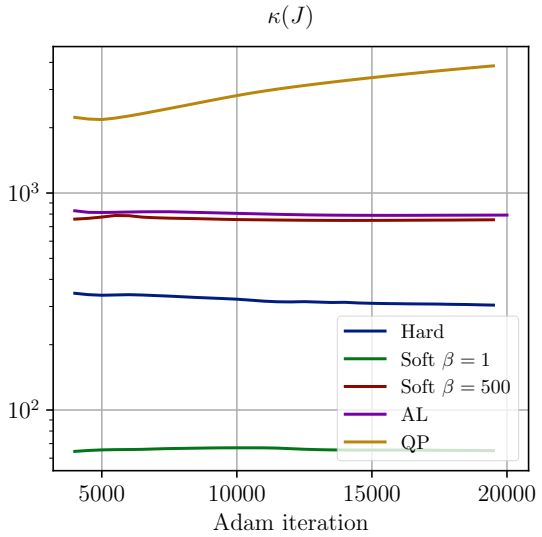

Figure 13: Condition number over 20000 iterations for five different PINNs.

7 Conclusion

This report has examined two failure modes of vanilla PINNs and proposed remedies for each.

For spectral bias, we compared standard PINNs against RFF-PINNs on the simple harmonic and van der Pol oscillators. The RFF embedding consistently reduced iterations to convergence as oscillation frequency or nonlinear damping increased, with the advantage becoming decisive at high frequencies where the standard PINN failed to converge within the iteration budget. The trade-off is that the fixed high-frequency embedding is detrimental at low frequencies, where it introduces unnecessary complexity that the optimiser must tune out. This suggests that RFF-PINNs are most valuable when the dominant frequency of the solution is known in advance.

For the van der Pol oscillator at large μ, both standard and RFF-PINNs failed to converge within 10^5 iterations for $\mu \geq 1.5$. We addressed this with curriculum learning, training the network on a sequence of problems with incrementally increasing μ, starting from $\mu = 0$ and advancing each time the residual fell below a fixed threshold. This successfully reached $\mu = 5$ in approximately 93,800 total epochs, substantially outperforming the direct training for $\mu = 2$, which did not converge within 100,000 iterations.

For boundary enforcement, we applied the quadratic penalty and augmented Lagrangian methods to the two-dimensional Helmholtz equation. Soft enforcement with a small penalty weight ($\beta = 1$) failed to satisfy the boundary conditions, while a large weight ($\beta = 500$) recovered the accuracy. The QP method drove boundary loss to near zero early but stagnated the residual, and its condition number grew without bound as $\beta_n \to \infty$, consistent with theory. The AL method, by contrast, maintained a bounded condition number while achieving boundary accuracy and residual loss comparable to hard enforcement – the best-performing baseline – without requiring a geometry-specific output transformation.

Taken together, the results suggest that RFF embeddings and augmented Lagrangian boundary enforcement are practical, well-motivated improvements over their vanilla counterparts in the regimes studied. Natural directions for future work include adaptive selection of the RFF bandwidth σ during training, and extending the constrained optimisation approach to other methods, such as interior point methods.

Bibliography

[1] I. Lagaris, A. Likas, and D. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, vol. 9, no. 5, pp. 987–1000, Sep. 1998, doi: 10.1109/72.712178.

[2] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019, doi: 10.1016/j.jcp.2018.10.045.

[3] N. Rahaman *et al.*, "On the Spectral Bias of Neural Networks," in *Proceedings of the 36th International Conference on Machine Learning*, PMLR, May 2019, pp. 5301–5310. [Online]. Available: https://proceedings.mlr.press/v97/rahaman19a.html

[4] Z.-Q. J. Xu, L. Zhang, and W. Cai, "On understanding and overcoming spectral biases of deep neural network learning methods for solving PDEs," *Journal of Computational Physics*, vol. 530, p. 113905, Jun. 2025, doi: 10.1016/j.jcp.2025.113905.

[5] M. Tancik *et al.*, "Fourier Features Let Networks Learn High Frequency Functions in Low Dimensional Domains." [Online]. Available: http://arxiv.org/abs/2006.10739

[6] H. Son, S. W. Cho, and H. J. Hwang, "Enhanced physics-informed neural networks with Augmented Lagrangian relaxation method (AL-PINNs)," *Neurocomputing*, vol. 548, p. 126424, Sep. 2023, doi: 10.1016/j.neucom.2023.126424.

[7] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, Jan. 1989, doi: 10.1016/0893-6080(89)90020-8.

[8] K.-I. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, no. 3, pp. 183–192, Jan. 1989, doi: 10.1016/0893-6080(89)90003-8.

[9] K. Hornik, M. Stinchcombe, and H. White, "Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks,"

Neural Networks, vol. 3, no. 5, pp. 551–560, Jan. 1990, doi: 10.1016/0893-6080(90)90005-6.

[10] A. Hofinger, "The Metrics of Prokhorov and Ky Fan for Assessing Uncertainty in Inverse Problems," *Sitzungsberichte und Anzeiger der mathematisch-naturwissenschaftlichen Klasse*, vol. 1, p. SBII–107–SBII–125, 2010, doi: 10.1553/SundA2006sSBII-107.

[11] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Networks*, vol. 6, no. 6, pp. 861–867, Jan. 1993, doi: 10.1016/S0893-6080(05)80131-5.

[12] D. Bertsekas and S. E. Shreve, *Stochastic Optimal Control: The Discrete-Time Case*. Athena Scientific, 1996.

[13] J. Schmidhuber, "Who Invented Backpropagation?." Accessed: May 17, 2026. [Online]. Available: https://people.idsia.ch/~juergen/who-invented-backpropagation.html

[14] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization." [Online]. Available: http://arxiv.org/abs/1412.6980

[15] A. Rahimi and B. Recht, "Random Features for Large-Scale Kernel Machines," in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 2007. [Online]. Available: https://papers.nips.cc/paper_files/paper/2007/hash/013a006f03dbc5392effeb8f18fda755-Abstract.html

[16] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., in Proceedings of Machine Learning Research, vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[17] I. Loshchilov and F. Hutter, "SGDR: Stochastic Gradient Descent with Warm Restarts." [Online]. Available: http://arxiv.org/abs/1608.03983

[18] K. Kawaguchi, "Deep learning without poor local minima," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, in NIPS'16. 2016, pp. 586–594.

[19] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, in NIPS'18. Red Hook, NY, USA: Curran Associates Inc., 2018, pp. 6391–6401.

[20] P. Soviany, R. T. Ionescu, P. Rota, and N. Sebe, "Curriculum Learning: A Survey," *International Journal of Computer Vision*, vol. 130, no. 6, pp. 1526–1565, Jun. 2022, doi: 10.1007/s11263-022-01611-x.

[21] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, in ICML '09. New York, NY, USA: Association for Computing Machinery, Jun. 2009, pp. 41–48. doi: 10.1145/1553374.1553380.

[22] C. Cartis, "C6.2 Continuous Optimisation." [Online]. Available: https://courses.maths.ox.ac.uk/course/view.php?id=6141