Lecture 3: Introduction to PINNS and the DRM

Chris Budd and Aengus Roberts[1]

[1]University of Bath

Ai4Sci, December 2025

Some papers to look at

- Raissi, M., P. Perdikaris, and G. E. Karniadakis. 2019. "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations." Journal of Computational Physics 378: 686–707. https://doi.org/https://doi.org/10.1016/j.jcp.2018.10.045.
- Grossman et. al. *Can PINNS beat the FE method?*
- Shin et. al. *On the convergence of PINNS for linear elliptic and parabolic PDEs*
- Wang et. al. *When and why PINNS fail to train: a neural tangent kernel perspective*
- Kiyani *Which Optimizer Works Best for Physics-Informed Neural Networks and Kolmogorov-Arnold Networks?*
- Chen et. al. *Sidecar: A Structure-Preserving Framework for Solving Partial Differential Equations with Neural Networks*
- Russell et. al. *Two-point boundary value problems*

Motivation: Solving ODEs and PDEs

Seek to solve ODE/PDE problems of the form

$$\mathbf{u}_t = F(\mathbf{x}, u, \nabla u, \nabla^2 u) \quad \text{with BC}$$

eg. ODE (IVP or BVP)

$$\frac{du}{dt} = u^2, u(0) = 1, \quad -\epsilon^2 \frac{d^2 u}{dx^2} + u = 1, \quad u(0) = u(1) = 0.$$

eg. PDE

$$-\Delta u = f(x), \quad iu_t + \Delta u + u|u|^2 = 0, \quad u_t = \Delta u + f(x, u) \quad x \in R^n$$

Classical Method 1. Finite Differences

Work with a set of point values for a function never with a function directly
IVP/BVP:

$$U_j^n \approx u(n\Delta t, j\Delta x), \quad \Delta t, \Delta x \ll 1$$

eg.

$$u_t = u_{xx} + u^3$$

IMEX Crank-Nicholson method:

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} =$$

$$\frac{1}{2\Delta x^2}\left(U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1} + U_{j+1}^n - 2U_j^n + U_{j-1}^n\right) + \left(U_j^n\right)^3.$$

- Have error estimates of the form

$$\|U_j^n - u(n\Delta t, j\Delta x)\| < C(u)\Delta t^p \Delta x^q, \quad \Delta t, \Delta x \to 0$$

- Can reduce $C(u)$ and increase p, q using an adaptive approach.
- Lots of software
- Have to recover the function from the point values
- Awkward in higher dimensions!

Classical Method 2: Finite Elements

Express $u(x,t)$ as a Galerkin approximation:

$$u(t,x) \approx U(t,x) = \sum_{i=0}^{N} U_i(t)\, \phi_i(x)$$

with $\phi_i(x)$ **Not** globally differentiable *locally supported, piece-wise polynomial spline functions*

- Work with a function $U(x) \in H^1$
- Require U to satisfy a *weak form* of the PDE (see Lecture 4)
- Have guaranteed error estimates of the form

$$\|u - U\|_{H^1} < C(u) N^{-\alpha}$$

- Lots of software (see Lecture 2b)
- Can reduce $C(u)$ and increase α using an adaptive approach.
- Awkward in higher dimensions!

Classical Method 3: Collocation

Express $u(x,t)$ as a function approximation:

$$u(t,x) \approx U(t,x) = \sum_{i=0}^{N} U_i(t)\, \phi_i(x)$$

with $\phi_i(x)$:

Locally differentiable locally supported, piece-wise polynomial spline functions such as Lagrange polynomial interpolants

- Work with a function $U(x) \in C^n_{loc}$
- Require U to satisfy the PDE exactly at a carefully chosen set of **collocation points**

- Have guaranteed error estimates of the form

$$\|u - U\|_{C_2} < C(u)N^{-\alpha}$$

- Can reduce $C(u)$ and increase α using VERY carefully chosen collocation points
- Implemented in `python` as $solve_b vp$, Colsys
- Impossible in higher dimensions!

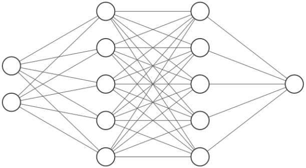
Issues with classical methods

- Well tested and good convergence theories
- Accuracy of the computation depends crucially on the choice and shape of the mesh points
- Generally require **time and practice** to implement (although FireDrake etc. makes this easier)
- Two-fold curse of dimensionality

PINNS

Physics Informed Neural Networks for solving PDEs: advertised as "Mesh free methods".
Use a Deep Neural Net of width W and depth L to give a nonlinear functional approximation to $u(\mathbf{x})$ with input x.

$$y(\mathbf{x}) = DNN(\mathbf{x})$$



$y(\mathbf{x})$ is constructed via a combination of linear transformations and nonlinear/semi-linear activation functions.

Example: Shallow 1D neural net

$$y(\mathbf{x}) = \sum_{i=0}^{W-1} c_i \sigma(a_i \mathbf{x} + \mathbf{b}_i)$$

Often take

$$\sigma(z) = \mathrm{ReLU}^3(z) \equiv z_+^3, \quad \text{or} \quad \sigma(z) = \tanh(z)$$

As a result we can define y_{zz} for every point z

Operation of a 'traditional' PINN

Want to solve a ODE/PDE in $\mathbf{x}$

- PINNS are similar to collocation methods
- Assume that $y(\mathbf{x})$ has strong regularity eg. C^2
- Differentiate $y(\mathbf{x})$ exactly using the chain rule
- Evaluate the PDE residual at N_r collocation points $\mathbf{X}_i$, :chosen to be uniformly spaced, or samples from a random distribution
- Train the neural net to minimise a **loss function** L combining the PDE residual and boundary and initial conditions. May use Epochs linked to the random training samples.

Eg 1. Solution of regular two-point BVPs by PINNs

Consider the two-point BVP with Dirichlet boundary conditions:

$$-u_{xx} = f(x, u, u_x),\ x \in [0, 1] \quad u(0) = a,\ u(1) = b.$$

Define output of the PINN by $y(x)$ and residual $r(x) := y_{xx} + f(x, y, y_x)$. Train the coefficients θ of the NN by minimising the loss function

$$L = \frac{1}{N_r} \sum_{i}^{N_r} |r(X_i^r)|^2 + \frac{\beta}{2} \left(|y(0) - a|^2 + |y(1) - b|^2 \right),$$

where $\{X_i^r\}_i^{N_r}$ are the collocation points (possibly randomly) placed in $(0, 1)$.

Eg 2. Solution of regular second order IVPs by PINNs

Consider the second order IVP with initial conditions:

$$-u_{xx} = f(x, u, u_x), \; x \in [0,1] \quad u(0) = a, \; u'(0) = b.$$

Define output of the PINN by $y(x)$ and residual $r(x) := y_{xx} + f(x, y, y_x)$. Train the coefficients θ of the NN by minimising the loss function

$$L = \frac{1}{N_r} \sum_{i}^{N_r} |r(X_i^r)|^2 + \frac{\beta}{2} \left(|y(0) - a|^2 + |y'(0) - b|^2 \right),$$

where $\{X_i^r\}_i^{N_r}$ are the collocation points (possibly randomly) placed in $(0, 1)$.

Eg 3. Solution of regular parabolic PDEs by time-stepping PINNs

Consider the semilinear parabolic PDE with Dirichlet boundary conditions:

$$u_t = u_{xx} + f(x, u, u_x), \ x \in [0, 1] \quad u(0) = a, \ u(1) = b$$

and initial condition

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

Implicit time-stepping scheme $U^n(x) \approx u(n\Delta t, x)$.

$$\frac{U^{n+1}(x) - U^n(x)}{\Delta t} = U_{xx}^{n+1} + f\left(x, U^{n+1}, U_x^{n+1}\right) \equiv F(U^{n+1}).$$

- Start with $U^0(x) = u_0(x)$
- For $n > 0$: Define output of the PINN by $y(x)$ and residual

$$r(x) := U^n(x) + \Delta t F(y).$$

- The PINN is trained by minimising the loss function

$$L = \frac{1}{N_r} \sum_{i}^{N_r} |r(X_i^r)|^2 + \frac{1}{2} \left(|y(0) - a|^2 + |y(1) - b|^2 \right),$$

where $\{X_i^r\}_i^{N_r}$ are the collocation points placed in $(0, 1)$.
- Set $U^{n+1} = y(x)$ and repeat.

Eg 4. Solution of regular parabolic PDEs by a full PINN

Consider the same semilinear parabolic PDE

$$u_t = u_{xx} + f(x, u, u_x),\ x \in [0,1] \quad u(0) = a,\ u(1) = b.$$

Bold approach!

- Have NN with $y(t, x)$ a function of t and x
- Take N_r collocation points Z_i in space and time
- Residual $r = u_t - u_{xx} - f(x, u, u_x)$
- Minimise

$$L = \frac{1}{N_r} \sum_{i}^{N_r} |r(Z_i^r)|^2 + \frac{\beta}{2} \left(|y(0) - a|^2 + |y(1) - b|^2 \right)$$

$$+\gamma \| y(0, x) - u_0(x) \|^2.$$

Problematic as space and time play very different roles in the PDE

Numerical results for: $-u'' = \pi^2 sin(\pi x)$.

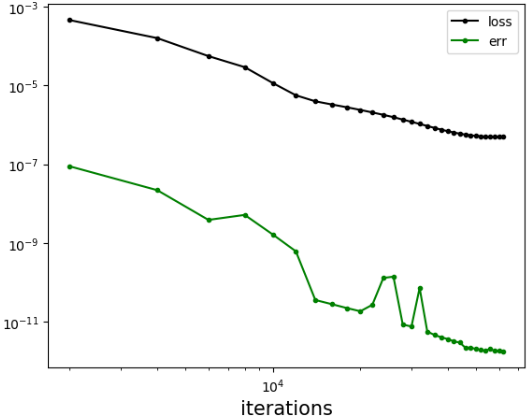


Figure: Residual based Loss and L^2 error of the PINN solution for $N_r = 100$

Numerical results: $u(x) = sin(\pi x)$

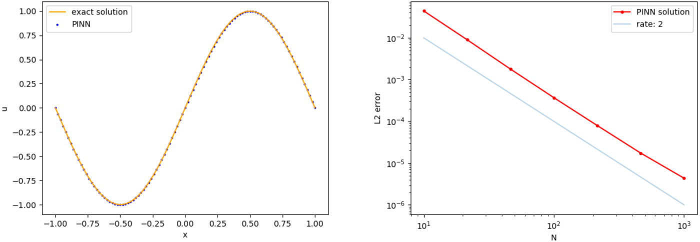


Figure: (Left) PINN with depth $L = 2$ and width $W = 30$. $N_r = 100$ uniformly distributed quadrature points, activation function: tanh, optimizer: Adam with $lr = 1e - 3$ (Right) Convergence rate for 1st order interpolant

Eg 2. Singular Reaction-Diffusion Equation

Solve $-\varepsilon^2 u_{xx} + u = 1 - x$ on $[0, 1]$ $\quad u(0) = u(1) = 0$

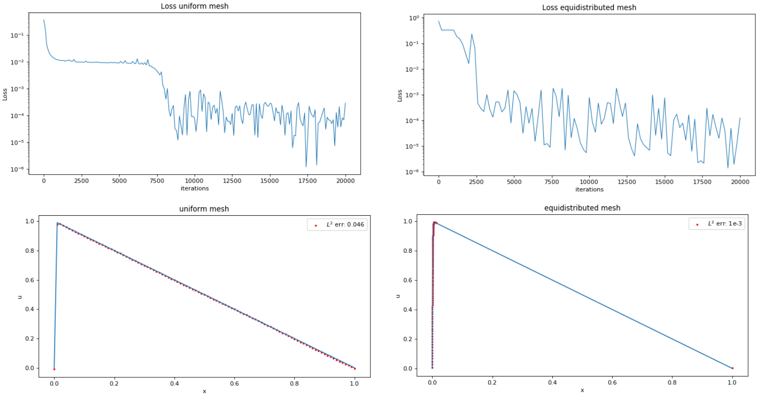


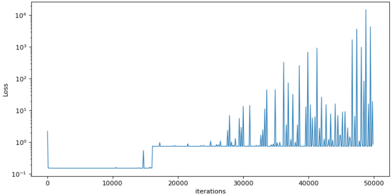
Figure: PINN (tanh) trained for 20000 epochs, $N_r = 101$, Adam optimizer with $lr = 1e-3$. (left) Uniform collocation points (right) Adapted collocation points **much faster training**

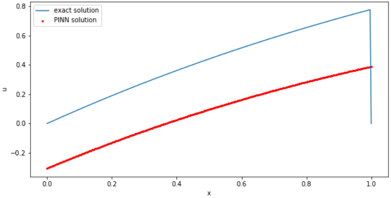
Eg 3. Bad news: Convection-dominated equation

PINNs often fail to train when the solution of the BVP exhibits singular and convective behaviour [Krishnapriyan, Aditi et. al., (2021)]:

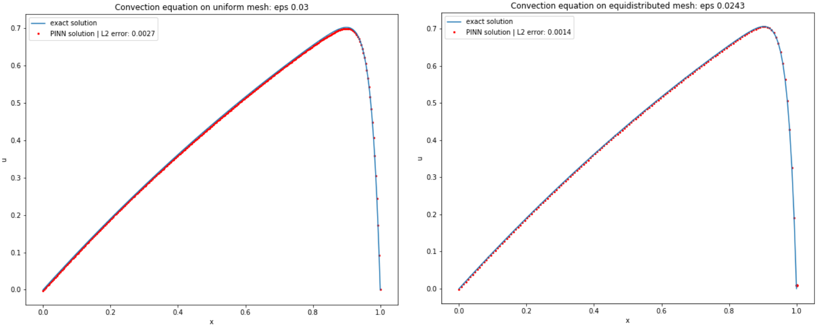
$$-\varepsilon u_{xx} + \left(1 - \frac{\varepsilon}{2}\right) u_x + \frac{1}{4}\left(1 - \frac{1}{4}\varepsilon\right) u = e^{-x/4} \text{ on } [0,1] \quad u(0) = u(1) = 0$$

$$u(x) = exp^{\frac{-x}{4}} \left(x - \frac{exp^{-\frac{1-x}{\varepsilon}} - exp^{-\frac{1}{\varepsilon}}}{1 - exp^{-\frac{1}{\varepsilon}}} \right)$$





Numerical results: Convection Equation



Convergence theory for PINNS

[Shin, Darbon and Kaniardarkis, 2020], [Jiao, Lai, Lo, Wang, Yang, 2024]

- PINN error is a combination of approximation error and training/optimization error
- Show that a PINN (depth L width W) can be constructed with low approximation error which reduces as the complexity of the PINN increases.
- Hope that the optimization error can be reduced to acceptable levels.
- Try things out on simple problems

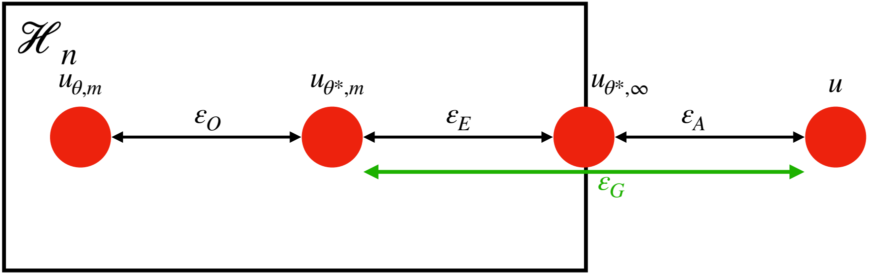
BUT

- **Non convex (no guarantee of uniqueness or convergence of training)**
- **Badly conditioned**
- PINNs are nonlinear function approximators. No easy bound on the solution error.
- Solutions can sometimes have no connection to reality!

Compare with the PINN formulation with collocation

- PINN is a nonlinear approximation
- It may have better expressivity
- The collocation equations become an ill conditioned optimisation problem
- We may not find a good optimum duce to the lack of convexity and the ill conditioning
- Pre conditioning and the right choice of optimiser are essential

Detailed PINN Convergence Theory 1



Detailed PINN Convergence Theory 2

Underlying solution: $u(x)$

H_n: Class of functions approximated by NN with n degrees of freedom

- $u_{\theta,m}$: Approximation found in practice after some training. Is less accurate than
- $u_{\theta_m^*}$: Approximation obtained by perfect optimisation with finite collocation points. Is less accurate than
- $u_{\theta^*,\infty}$: Approximation obtained by perfect optimisation with infinite collocation points.

Measured error is $\|u_{\theta,m} - u\|$

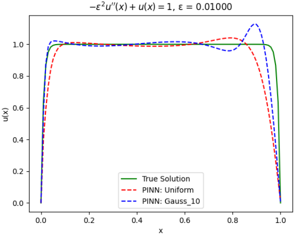
- $\|u_{\theta,m} - u_{\theta^*,m}\|$: Optimisation error $\mathcal{E}_O$: Hard to contol and depends on the optimiser and the initialisation:
- $\|u_{\theta^*,m} - u_{\theta^*,\infty}\|$: Estimation error $\mathcal{E}_E$: Main results on this
- $\|u_{\theta^*,\infty} - u^*\|$: Approximation error $\mathcal{E}_A$:

Results on a mildly singular BVP

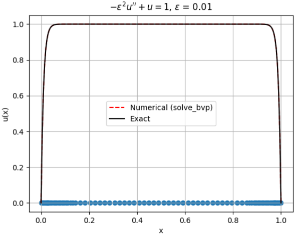
Compare standard collocation and a PINN for the ODE using ADAM

$$-\epsilon^2 u_{xx} + u = 1, \quad u(0) = u(1) = 0, \quad 0 < \epsilon \ll 1$$

This has a boundary layer at $x = 0, x = 1$ of width ϵ.

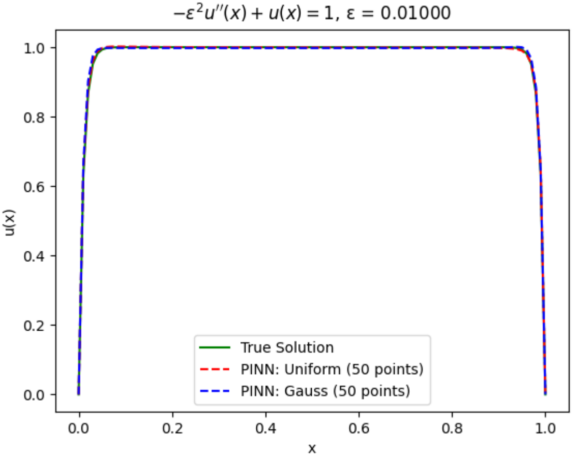


PINN



Collocation

Using the LBFGS optimiser



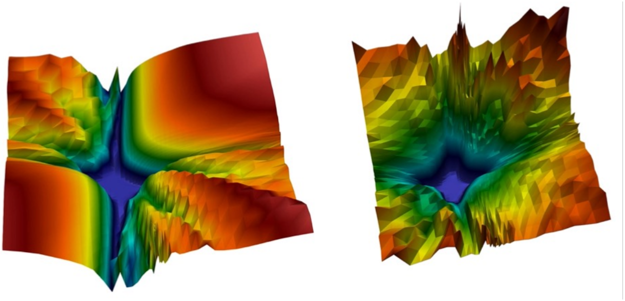
How and why do PINNS struggle to train .. and how can we fix this?

A subject of intensive research! See [Wang et. al.] for a summary: Observe

- PINNS train especially badly when they have high frequency features
- Loss function has multi-scale interactions, leading to ill-conditioning and stiffness in the gradient flow dynamics and a stringent stability requirement on the loss rate
- Standard networks show ill-conditioning and spectral bias and cannot learn high frequencies. This has been see for example in weather forecasting

They advocate the use of Neural Tanjent Kernels in the optimisation process. [Kiyani et. al.] extend this with the development of better optimisers eg. Broyden methods, for PINNS

Loss landscape is highly non-convex, [Kiyani et. al.]



Also Issues with **conditioning** as in the last lecture on approximation.
Can fix for shallow networks by **preconditioning**

Deep Ritz Method: Motivation the Calculus of variations

Seek to solve PDE problems eg. of the form

$$-\epsilon^2 u_{xx} + u = 1, \quad u(0) = u(1) = 0, \quad \Omega = [0,1] \quad (*)$$

A PINN minimises the PDE residual as the loss function

Instead use ideas from the calculus of variations to find a loss function

Functional minimisers: Consider the functional

$$F(u) = \int_\Omega \frac{\epsilon^2 u_x^2}{2} + \frac{u^2}{2} - u \, dx.$$

Then $F(u)$ is minimised when u is the (unique) solution of the PDE (*)

Approach 1: Finite Element Methodology

Express $u(x)$ as a Galerkin approximation:

$$u(x) \approx U(x) = \sum_{i=0}^{N} U_i(t)\, \phi_i(x)$$

with $\phi_i(x)$ **Not** globally differentiable *locally supported, piece-wise polynomial spline functions*

U is in the linear space $\mathcal{V}$ spanned by the functions $\phi_i(x)$.

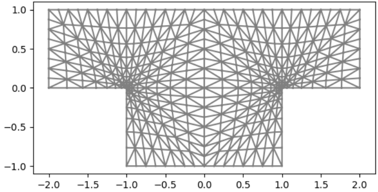
- Minimise F over all such functions U.
- If the PDE is linear, this leads to a linear system for the coefficients U_i of the approximation
- Linear system is symmetric and well conditioned
- Solve this system using (for example) a conjugate-gradient method.

Features of the FE method

- Unique solution (if the PDE is linear)
- Have guaranteed error estimates for all N, due to **Céa's lemma**, of the form
$$\|u - U\|_{H^1} < C(u)N^{-\alpha}$$
- Can reduce $C(u)$ and increase α using an adaptive approach.
- But .. Awkward in higher dimensions!

Meshes

Traditional PDE computations using Finite Element Methods use a computational mesh τ comprising mesh points and a mesh topology with $\phi_i(x)$ defined over the mesh



Meshes can be tricky to construct! Although ML methods can help a lot, see [B, Maierfofer, Rowbottom et. al, ICML 2025]

Approach 2: The Deep Ritz Method [Weinan E and Bing Yu, (2017)]

Lovely Idea!!!

Let $y(x)$ be the output of a parametrised NN

$$y(x) = NN(x)$$

with Y the (nonlinear) set of functions parametrised by θ

Then set

$$y^* = argmin_Y F.$$

Allowing for the boundary conditions

Deep Ritz method for the Poisson equation

The **Deep Ritz Method** (DRM) seeks the solution u satisfying

$$y = \arg\min_{v \in H} \mathcal{F}(v),$$

where H is the set of admissible functions (trial functions) and

$$\mathcal{I}(v) = \int_\Omega \left(\frac{1}{2} |\nabla v(\mathbf{x})|^2 - f(\mathbf{x}) v(\mathbf{x}) \right) d\mathbf{x} + \beta \int_{\partial\Omega} (v(\mathbf{x}) - u_D)^2 d\mathbf{s}$$

The Deep Ritz method is based of the following assumptions:

- $y(\mathbf{x})$ is NN based approximation of u
- A numerical quadrature rule for the functional using chosen quadrature points eg. random, optimal
- An algorithm for solving the optimization problem eg. SGD on random quadrature points

DRM vs.Finite Element

The DRM is superficially similar to the Finite Element method but has crucial differences as **the NN approximating subspace is nonlinear**, and the problem of finding the weights is non-convex and badly conditoned.

FE: linear

- Limited expressivity, reduced accuracy
- Adaptive only with effort
- Not equivariant
- Need a complex mesh data structure
- Convex with guarantees of uniqueness for many problems (and direct calculation using linear algebra)
- Well conditioned
- Work on saddle-point problems (eg. most problems)
- Good (a-priori and a-posteriori) guaranteed error bounds :

  Cea's Lemma: Bounds solution error by interpolation error on the FE space

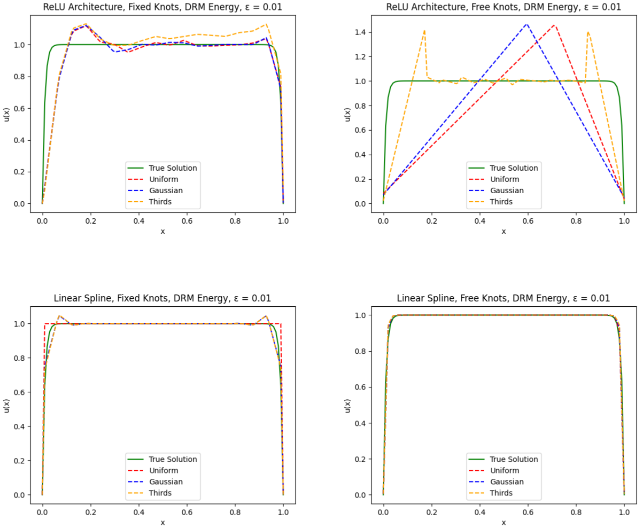
DRM:nonlinear

- Very expressive (potential high accuracy for a small number of degrees of freedom)
- Self adaptive
- Equivariant
- Don't need a complex mesh data structure
- Ill conditioned
- Don't work on saddle-point problems eg. most problems for example:

$$u_{xx} + u = 1, \quad u_x x + u^3 = 0.$$

- Don't have Céas Lemma

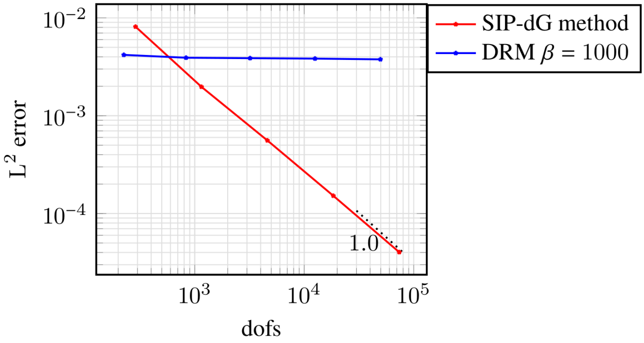
Example 1: $-\epsilon u'' + u = 1, \quad u(0) = u(1) = 0.$



Example 2: Poisson equation in 2D:

DRM method works well for small DOF

dG Finite Element Method is **much** better for more DOF



This convergence pattern is seen in many other examples

Summary

- PINNS and DRMS both show promise as a quick way of solving PDEs but have only really been tested on quite simple problems so far
- PINS not (yet) competitive with FE in like-for-like comparisons
- PINNs need careful meta-parameter tuning and preconditioning to work well
- Long way to go before we understand PINNS orn DRMS completey and have a satisfactory convergence theory for them in the general case.
- Lots of great stuff to do!