Joakim Lunde

Solving Laplace's equation with Physics-Informed Neural Networks

TBM4900 - Civil and Environmental Engineering, Master's Thesis
Supervisor: Ivan Depina
June 2025

Norwegian University of Science and Technology
Faculty of Engineering Science
Department of Civil and Environmental Engineering

NTNU

Preface

Takk til Amalie som har støttet meg med masse kjærlighet

Abstract

The `ntnuthesis` document class is a customised version of the standard LaTeX `report` document class. It can be used for theses at all levels – bachelor, master and PhD – and is available in English (British and American) and Norwegian (Bokmål and Nynorsk). This document is ment to serve (i) as a description of the document class, (ii) as an example of how to use it, and (iii) as a thesis template.

Sammendrag

Dokumentklassen `ntnuthesis` er en tilpasset versjon av LaTeX' standard `report`-klasse. Den er tilrettelagt for avhandlinger på alle nivåer – bachelor, master og PhD – og er tilgjengelig på både norsk (bokmål og nynorsk) og engelsk (britisk og amerikansk). Dette dokumentet er ment å tjene (i) som en beskrivelse av dokumentklassen, (ii) som et eksempel på bruken av den, og (iii) som en mal for avhandlingen.

Zhang et al., 2022

Contents

Contents viii

List of Figures ix

List of Tables xi

Glossary xiii

1 Introduction 1
- 1.1 Background and motivation . . . 1
- 1.2 Research question/objective . . . 2
- 1.3 Scope and limitations . . . 2
- 1.4 Thesis structure . . . 2

2 Preliminaries 3
- 2.1 Uncertainty Quantification . . . 3
- 2.2 PINNs Historical Overview . . . 4

3 Literature study 7
- 3.1 Physics-informed neural networks for consolidation of soils . . . 7
- 3.2 Probabilistic physics-informed neural network for seismic petrophysical inversion . . . 8
- 3.3 Machine Learning-Aided Monte Carlo Simulation and Subset Simulation . . . 8
- 3.4 Bayesian neural networks . . . 8
  - 3.4.1 Simple idea . . . 8
  - 3.4.2 In Practice . . . 9
- 3.5 Weight Uncertainty in Neural Networks . . . 10
- 3.6 from chatGPT . . . 10

4 Theoretical Framework 13

5 Methodology 15

6 Results 17
- 6.1 Time Normalisation Challenges . . . 17

6.1.1 Pitfalls of Time Normalisation in Physics–Informed Neural Networks . . . 17

7 Discussion 21

7.1 Limitations . . . 21

8 Conclusion 23

Bibliography 25

List of Figures

3.1 An analogy of $p(D) = \int p(D \mid w)p(w)dw$ explained by ChatGPT. . 11

6.1 **Training loss for three configurations.** (i) Unnormalised time and parameters (blue); (ii) normalised time, but only v_0 z–scored (orange); (iii) normalised time and three parameters m, μ, v_0 z–scored (green). The third setting stalls after $\approx 10^3$ epochs. [Insert figure from experiment] 18

List of Tables

6.1 Hyper–parameters used in the experiments. 19
6.2 Relative $\mathcal{L}_2$ error on an extreme test case ($m_z = \mu_z = v_{0,z} = 5$). . . 19

Glossary

ΔT temperaturendring. 4

Chapter 1

Introduction

1.1 Background and motivation

The rice of deep learning has been a game changer in the field of machine learning. Deep learning has been used to solve a wide range of problems, such as image recognition, speech recognition, and natural language processing.

However, deep learning models are often considered as black boxes, as they are difficult to interpret. This is a problem, as it is important to understand how a model makes predictions in order to trust it. In addition, deep learning models are often overconfident in their predictions, which can lead to catastrophic failures in safety-critical applications. Therefore, there is a need for methods that can provide uncertainty estimates for deep learning models.

In the field of engineering, uncertainty estimation is important for decision-making under uncertainty. For example, geotechnical structure design and analysis are affected by measurement uncertainty, statistical uncertainty, and transformation model uncertainty. These geotechnical uncertainties are not capable of being taken into account by conventional deterministic analytical methods. First of all, there are many different causes of uncertainty, including variations in soil characteristics, building techniques, and environmental factors. Therefore, there is a need for methods that can provide uncertainty estimates for geotechnical structure design and analysis.

There has been very little research on uncertainty estimation for deep learning models. One approach is to use Bayesian neural networks, which can provide uncertainty estimates by placing a distribution over the weights of a neural network. However, Bayesian neural networks are computationally expensive and difficult to train. Another approach is to use ensemble methods, which can provide uncertainty estimates by training multiple models on different subsets of the data. However, ensemble methods are also computationally expensive and difficult to train.

Physics infromed neural networks (PINNs) are a class of deep learning models that can be used to solve partial differential equations (PDEs). PINNs have been used to solve a wide range of PDEs, including the heat equation, the wave equation, and the Navier-Stokes equations. However, PINNs are often overconfident in their predictions, which can lead to catastrophic failures in safety-critical applications.

Therefore, there is a need for methods that can provide uncertainty estimates for PINNs.

1.2 Research question/objective

In this thesis, we use a metamodel-based importance sampling (MAIS) with a PINN as our metamodel. The objective of this thesis is to develop a method for uncertainty estimation for PINNs using MAIS. The research question is: Can we use MAIS with a PINN as our metamodel to provide uncertainty estimates for PINNs?

1.3 Scope and limitations

The thesis is limited to the use of MAIS with a PINN as our metamodel for uncertainty estimation. We do not consider other methods for uncertainty estimation, such as Bayesian neural networks or ensemble methods. We also do not consider other types of metamodels, such as Gaussian processes or radial basis functions. Furthermore, we focus on the use of MAIS with a PINN as our metamodel for uncertainty estimation. The partial differential equations considered are SO and SO

Due to

1.4 Thesis structure

Chapter 2

Preliminaries

2.1 Uncertainty Quantification

In the domain reliability analysis, the primary goal is to estimate the probability of failure, or probability of exceeding a certain threshold. Theoretically, the probability of failure is defined as the probability that the limit state function is less than zero. The limit state function is a function of the random variables and the model parameters. The random variables are the input variables to the model, and the model parameters are the parameters of the model. The limit state function is defined as $g(\mathbf{X}, \mathbf{Z})$, where $\mathbf{X}$ is the random variables and $\mathbf{Z}$ is the model parameters. The probability of failure is defined as the integral of the joint probability density function of the random variables and the model parameters over the region where the limit state function is less than zero. The probability of failure is given by the following equation:

$$P_f = \int_{\Omega} f_{\mathbf{X},\mathbf{Z}}(\mathbf{x}, \mathbf{z})\, d\mathbf{x}\, d\mathbf{z} \tag{2.1}$$

where $\Omega = \{(\mathbf{x}, \mathbf{z}) \mid g(\mathbf{x}, \mathbf{z}) < 0\}$.

This integral is often intractable and must be approximated using numerical methods. The most common method for approximating the probability of failure is the Monte Carlo simulation. The Monte Carlo simulation is a statistical method that uses random sampling to estimate the probability of failure. The Monte Carlo simulation works by generating random samples of the random variables and the model parameters, evaluating the limit state function for each sample, and counting the number of samples where the limit state function is less than zero. The probability of failure is then estimated as the ratio of the number of samples where the limit state function is less than zero to the total number of samples. The Monte Carlo simulation is a powerful and versatile method for estimating the probability of failure, but the convergence rate of MCS in number of samples is very slow.

> Based on the discussion above, it is safe to conclude that the primary bottleneck of all the reliability analysis techniques is the need for running simulation to evaluate the limit-state function. Often, the

> limit-state function are in form of complex nonlinear ordinary/partial differential equations (ODE/PDE) and solving it repeatedly can make the process computationally expensive. (Chakraborty, 2020)
>
> For real-world mechanics and engineering systems, the physical and environmental factors, including material properties, structure size, and boundary conditions, are inherently stochastic [1–5]. Considering this feature, many reliability analysis methods have been suggested for structural design [6–10] and safety assessment. There are many popular reliability analysis methods, including sampling methods [11,12] (e.g. Monte Carlo simulation (MCS) [13–15]), most probable point (MPP) based methods (e.g. first-order reliability method (FORM) [16,17] and second-order reliability method (SORM) [18,19]), surrogate methods [20], expansion methods [21], and approximated integration methods [22,23]. Among them, FORM exhibits superiority in terms of its simplicity and efficiency, and thus it is widely utilized in various engineering problems [24,25]. (Meng et al., 2023)

2.2 PINNs Historical Overview

PINNs were developed by Raissi et al., 2019 in 2019. The use areas are diverse, including but not limited to... The main work of using PINNs concerns the soling of ordinary and partial differential equations (PDEs) and inverse problems governed by PDEs. The main idea is to use the PDE as a constraint in the loss function of the neural network. This is done by enforcing the PDE at a set of collocation points. The collocation points are sampled from the domain of interest. The loss function is minimized by using gradient-based optimization algorithms, as explained in 4. By relying on a PDE as a constraint, the need for labelled data is eliminated. This is a significant advantage of PINNs compared to traditional machine learning methods.

The first work applying PINNs to reliability analysis was done by Chakraborty (2020).

The probabilistic model with metamodel-based importance sampling was proposed by and when....

Uncertainty quantifications in PINNs are at what stage?...

> *The current work on the application of PINN to reliability analysis is still in the preliminary stage and mainly based on the first approach, i.e., physics-informed loss function. Chakraborty (2020) firstly applied PINN to reliability analysis. In his work, the LSF is expressed in the form of linear or nonlinear PDEs, and PINN is used to solve PDEs and obtain the failure probability. Based on the work in Chakraborty (2020), Zhang and Shafieezadeh (2022) introduced an active learning approach with the dual objective of training PINN for solving PDEs and characterizing the limit state. Zhou et al. (2023) applied PINN to the dynamic reliability analysis of multi-state systems. All these works transformed the reliability analysis problems into the solution*

> *of PDEs. However, the PDEs are not always available for reliability problems especially for static reliability, and PINN proposed in Raissi et al. (2019) is not available for such reliability problems that cannot be described by PDEs. For such problems, a novel NN approach combined with physical information needs to be constructed for reliability analysis.*
> Bai and Song, 2023

The fact that PINNs are not always available for reliability problems that cannot be described by PDEs is a significant limitation. This is a motivation for the current work. For static reliability problems, the system behaviour is typically modeled using limit state functions like g(X), where X is the random variables. This is an algebraic equation, with no derivatives.

B-PINNs and E-PINNs and P-PINNs all exist..

The combination of PINNs and metamodel-based importance sampling is novel and has not been explored before.

Chapter 3

Literature study

3.1 Physics-informed neural networks for consolidation of soils

`file:///C:/Users/jolu2/AppData/Local/Microsoft/Windows/INetCache/IE/79SHIDH0/Physics-informed_neural_networ[1].pdf`

For the forward problem, it is difficult to obtain analytical solutions for most of the models related to consolidation.

Researchers have revealed that PINNs possess the following advantages compared with the conventional mesh-based numerical methods in tackling the forward problem. First, PINNs is capable of solving the inverse problem with the only minor change of the code that is used in a forward problem (Liu and Wang, 2019). Secondly, neural network-based methods with mesh-free features can reduce the tedious work of mesh generation (Basir and Senocak, 2022). Thirdly, PINNs can obtain remarkably accurate solutions and reliable parameter estimations with fewer data and average-quality data, to reduce the dependence on the need for large training datasets (Zhang et al., 2021). Fourthly, PINNs can produce results at any point in the domain once it has been trained (Basir and Senocak, 2022).

Uses Tanh

The values of wf and wb are commonly assumed to be 1 in the case that all the loss values are of the same order of magnitude. It should be noted that the choice of these two coefficients is still an open problem needing further investigations (Wang et al., 2021).

We use the Adam combined with L-BFGS optimizers

Furthermore, we employ a commonly used Glorot normal scheme (Glorot and Bengio, 2010) as the p

To deal with the stochastic nature of the training procedure, we calculated the results as an average over 5 realizations as suggested by Kadeethum et al. (2020)

Found that more with a lot of domain points, increasing the boundary points help, but not the other way around necessarily.- overfitting?

It is clear from Figure 7a that the performance of PINNs is found to be optimal at learning rates of 10–2 and 10–3

3.2 Probabilistic physics-informed neural network for seismic petrophysical inversion

`https://library.seg.org/doi/10.1190/geo2023-0214.1?utm_source=chatgpt.com`

We propose a probabilistic physics-informed neural network (P-PINN) algorithm for seismic petrophysical inversion with the goal of estimating the most likely model of petrophysical properties, the unknown hyperparameters of the rock-physics model, and the uncertainty of the model predictions.

3.3 Machine Learning-Aided Monte Carlo Simulation and Subset Simulation

`https://journals.sagepub.com/doi/10.1177/03611981241248166?utm_source=chatgpt.com#body-ref-bibr14-03611981241248166`

(...) geotechnical structure design and analysis are also affected by measurement uncertainty, statistical uncertainty, and transformation model uncertainty. These geotechnical uncertainties are not capable of being taken into account by conventional deterministic analytical methods. First of all, there are many different causes of uncertainty, including variations in soil characteristics, building techniques, and environmental factors (13).

3.4 Bayesian neural networks

3.4.1 Simple idea

The difference between a normal neural network and a Bayesian neural network is that the latter has a distribution of weights, rather than a single value. This allows for the model to express uncertainty in the predictions.

We have a KL term as well as the likelihood term in the loss function. The KL term is a measure of how much the posterior distribution differs from the prior distribution. The likelihood term is the negative log likelihood of the data given the weights. Posterior distribution, P(W|D), is the distribution of the weights given the data, and the prior distribution, P(W), is the distribution of the weights before seeing the data.

$$P(W|D) = \frac{P(D|W)P(W)}{P(D)} \tag{3.1}$$

We approximate $P(W|D)$ with a Gaussian distribution, and the KL divergence describes the difference between this distribution and the posterior distribution.

Another approach is to use Markov Chain Monte Carlo. Then we use a Monte Carlo instead of a simple distribution as replacement. Less efficient.

3.4.2 In Practice

Each weight have a distribution $W \sim \mathcal{N}(\mu, \sigma^2)$
Than compute the likelihood of

A numerical example:
NN(x;w,b) = w*x + b
$x_1 = 1$, $y_1 = 1.8$ $x_2 = 3$, $y_2 = 2.7$
We have a Gaussian prior distribution of the weights, $w = b = \mathcal{N}(0, 1^2)$
The data likelihood for a single data point is

$$p(y_i|w, b) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(y_i - \mathrm{NN}(x_i; w, b))^2\right) \tag{3.2}$$

We use variational inference to approximate the posterior distribution. q(w,b) = q(w)q(b). We use a Gaussian distribution for each weight.

$$q(w) = \mathcal{N}(\mu_w, \sigma_w^2) \tag{3.3}$$
$$q(b) = \mathcal{N}(\mu_b, \sigma_b^2) \tag{3.4}$$

Evidence lower bound (ELBO) is the objective function we want to maximize. It is written as: $ELBO = E[\log p(D|w, b)] + \ln(p(w, b)) - \ln(q(w, b))$, where the KL divergence is the difference between the prior and the posterior: $KL(q(w, b)||p(w, b)) = -\ln(q(w, b)) + \ln(p(w, b))$. The prior is written as $p(w, b) = p(w)p(b)$.

We fist sample from the prior distribution, then we compute the likelihood of the data given the weights, and then we compute the KL divergence between the prior and the posterior:

The prior distribution is $w = b = \mathcal{N}(0, 1^2)$. Let's say we draw w' = 0.37 and b' = -0.11 from the prior distribution. Then we compute the likelihood of the data given these weights:

$ELBO = E[\log p(D|w', b')] + \ln(p(w', b')) - \ln(q(w', b'))$
First term is the likelihood of the data given the weights. We have two data points, so we compute the log likelihood for each data point and sum them up:

$$\begin{aligned}
\log p(D|w', b') &= \log p(y_1|w', b') + \log p(y_2|w', b') && (3.5)\\
&= \log\left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(y_1 - \mathrm{NN}(x_1; w', b'))^2\right)\right) && (3.6)\\
&+ \log\left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{1}{2\sigma^2}(y_2 - \mathrm{NN}(x_2; w', b'))^2\right)\right) && (3.7)
\end{aligned}$$

Numerically, if we assume that the observation noise is gaussian with a standard deviation of 0.2, we get:

$$\log p(D|w', b') = \log\left(\frac{1}{\sqrt{2\pi}\cdot 0.2}\exp\left(-\frac{1}{2\cdot 0.2^2}(1.8 - \mathrm{NN}(1; 0.37, -0.11))^2\right)\right) \tag{3.8}$$

$$+ \log\left(\frac{1}{\sqrt{2\pi}\cdot 0.2}\exp\left(-\frac{1}{2\cdot 0.2^2}(2.7 - \mathrm{NN}(3; 0.37, -0.11))^2\right)\right) \tag{3.9}$$

When computed numerically, we get log likelihood = -64,38.

Next term is the log prior, $\ln(p(w', b')) = \ln(p(w')) + \ln(p(b')) = \ln\left(\frac{1}{2\pi}e^{-\frac{0.37^2}{2}}\right) + \ln\left(\frac{1}{2\pi}e^{-\frac{0.11^2}{2}}\right)$

This becomes -0.988 - 0.926 = -1.914.

The last term is the log of the variational distribution, $\ln(q(w', b')) = \ln(q(w')) + \ln(q(b')) = \ln\left(\frac{1}{2\pi}e^{-\frac{\mu_w^2}{2\sigma_w^2}}\right) + \ln\left(\frac{1}{2\pi}e^{-\frac{\mu_b^2}{2\sigma_b^2}}\right)$

3.5 Weight Uncertainty in Neural Networks

`https://arxiv.org/pdf/1505.05424`

Bayes by Backprop is just a type of inference method.

"In general, exact Bayesian inference on the weights of a neural network is intractable as the number of parameters is very large and the functional form of a neural network does not lend itself to exact integration."

In BNN, we write p(y|x,w) as the likelihood instead of the general p(D|W).

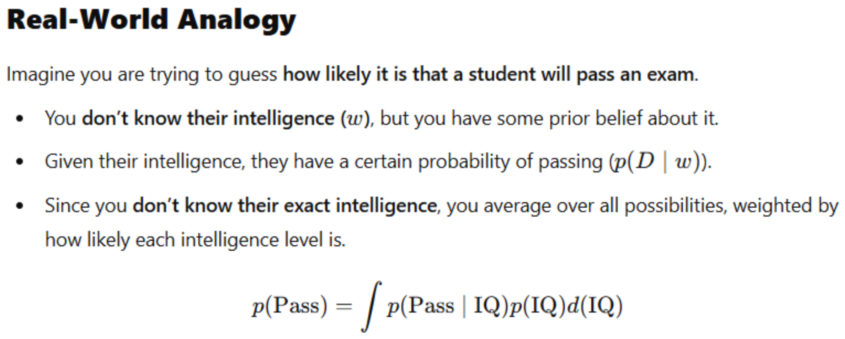
3.6 from chatGPT

An alternative to the variational inference is to use Markov Chain Monte Carlo (MCMC) methods. This is less efficient, but more accurate. VI: Faster, good for large-scale problems, but can suffer from approximation bias due to the factorized assumption. MCMC: More "accurate" in theory, but can be computationally heavy and slow to converge.

ΔT is the temperature change.

From PINN-is "ubsequently, only one IS population around this MPP will be generated. Therefore, the contributions from the failure region far from the selected MPP will be neglected. In addition, it is also difficult to obtain a convergent MPP from the MCS sampling pool for the high-dimensional problem. Thus, the proposed approach is not suitable for multi-MPP or high-dimensional problems."

Test test test test og dette funker kanskje bra?



Real-World Analogy

Imagine you are trying to guess **how likely it is that a student will pass an exam**.

- You **don't know their intelligence** (w), but you have some prior belief about it.
- Given their intelligence, they have a certain probability of passing ($p(D \mid w)$).
- Since you **don't know their exact intelligence**, you average over all possibilities, weighted by how likely each intelligence level is.

$$p(\text{Pass}) = \int p(\text{Pass} \mid \text{IQ}) p(\text{IQ}) d(\text{IQ})$$

Figure 3.1: An analogy of $p(D) = \int p(D \mid w)p(w)dw$ explained by ChatGPT.

Chapter 4

Theoretical Framework

1. Why Monte Carlo Can Struggle With Rare Events When you draw samples from the original (or “nominal”) distribution—e.g., $N(\mu, \sigma^2)$ for each parameter—the probability of seeing “extreme” samples that cause failure (like crossing $y < -1$) may be very small. Hence, most samples do not contribute to the failure count, and the probability estimate can have a high variance (you need a huge number of samples to accurately capture rare failures).

Importance Sampling (IS) addresses this by sampling from a modified distribution that focuses on the region more likely to produce failure (i.e., “importance region”). Then each sample is re-weighted to maintain an unbiased estimate of the true probability.

You want to model:

$p(y, t)$ where the randomness comes from $(m, \mu, k) \sim$ known distribution p(y,t) where the randomness comes from $(m, \mu, k) \sim$ known distribution This is called a parametric uncertainty propagation problem. You have two main options:

Chapter 5

Methodology

Intuition behind multiplying $\pi(u)$ and $f(u)$:

The algorithm you're using is essentially combining both *prior knowledge* (from the CDF) and *likelihood* (from the PDF) to form a *combined target function.* The product $\pi(u) \cdot f(u)$ gives you a distribution that is **both cumulative and likelihood-based**.

- $\pi(u)$ captures the idea of how likely we are to be in a region up to u. In the context of failure, if we think of $g(\mathbf{u})$ as the performance measure, $\pi(u)$ reflects the likelihood of reaching or exceeding the failure threshold, based on the *cumulative distribution.*
- $f(u)$ reflects how likely it is to actually land on a specific value of u (from the *probability density*).

Together, multiplying the *CDF* and the *PDF* allows you to combine:

- The likelihood of being in the *failure region* (via $\pi(u)$),
- And the likelihood of specific configurations (via $f(u)$).

This combined term $\hat{h}(\mathbf{u})$ is what you're sampling from in the Metropolis-Hastings algorithm.

Why multiply, not add?

- *Multiplying* the two terms gives you a **joint distribution** where both the *likelihood* of the parameters and the *probability of failure* are considered together.
- If you were to *add* $\pi(u)$ and $f(u)$, you would be combining two very different types of quantities (one representing cumulative probability, the other representing point likelihood), which would not give you a valid combined distribution.

Conclusion:

It is correct and *intuitive* to multiply the CDF ($\pi(u)$) and the PDF ($f(u)$) in this case, because:

- You are combining the *probability of being in the failure region* (captured by $\pi(u)$),
- And the *likelihood of a particular sample* (captured by $f(u)$).

This forms a valid joint distribution that you can use in the Metropolis-Hastings algorithm.

Chapter 6

Results

I have tried to have time linspaced and shufled, and also heavytailed both ways, but nothing is better then just having it ranomdly distributed.

6.1 Time Normalisation Challenges

6.1.1 Pitfalls of Time Normalisation in Physics–Informed Neural Networks

6.1.1.0.1 Motivation. Normalising the temporal domain to $\tau \in [0,1]$ is attractive because it reduces the dynamic range of the inputs fed into the network and allows the same collocation grid to be reused for different physical problems. In a PINN, however, temporal rescaling alters the size of every time derivative that appears in the loss function. When this interacts with feature normalisation of other parameters (*e.g.,* m, μ, v_0) and with saturating activations such as tanh, the result can be severe degradation of training for large values of the normalised parameters. This subsection formalises the effect and documents the empirical behaviour we observed.

6.1.1.1 Mathematical analysis

Let $t\_\mathrm{phys} \in [0,T]$ denote physical time and define the normalised variable

$$\tau \;=\; \frac{t\_\mathrm{phys}}{T} \in [0,1], \qquad T > 0. \tag{6.1}$$

Assume the PINN learns a single scalar state $y(\tau)$, where the physical (state) is related by $y(\tau) = y\_\mathrm{phys}\Big(t\_\mathrm{phys} = T\tau\Big)$. Applying the chain rule yields

$$\frac{\partial y}{\partial \tau} = T\,\frac{\partial y\_\mathrm{phys}}{\partial t\_\mathrm{phys}}, \qquad\qquad \frac{\partial^2 y}{\partial \tau^2} = T^2\,\frac{\partial^2 y\_\mathrm{phys}}{\partial t\_\mathrm{phys}^2}. \tag{6.2}$$

6.1.1.1.1 Boundary–condition scaling. For a damped oscillator we enforce

$$y\_\mathrm{phys}(0) = y_0, \qquad \dot{y}\_\mathrm{phys}(0) = v_0. \tag{6.3}$$

Expressed in τ,

$$y(0) = y_0, \qquad \frac{\partial y}{\partial \tau}(0) = T\, v_0. \tag{6.4}$$

Hence **the derivative target is multiplied by** T. For $T = 5\,\mathrm{s}$ and $v_0 = 5.5\,\mathrm{m\,s^{-1}}$ the network must produce a slope of 27.5 at $\tau = 0$.

6.1.1.1.2 Effect on the network. Consider the first hidden layer

$$z^{(1)} \;=\; W^{(1)}x + b^{(1)}, \qquad x = [\tau, m_z, \mu_z, k_z, y_{0,z}, v_{0,z}]^\top, \quad a^{(1)} = \tanh\!\Big(z^{(1)}\Big). \tag{6.5}$$

All z–scored inputs $m_z, \mu_z, v_{0,z}$ are distributed over $\approx [-5, 5]$ in the training data. With weights of order $\mathcal{O}(1)$ the pre–activations quickly reach $|z^{(1)}| > 3$, pushing tanh into saturation where $\tanh'(z^{(1)}) \approx 0$ and gradients vanish. The large derivative requirement (6.4) forces even larger weights in deeper layers, reinforcing the saturation.

6.1.1.2 Empirical demonstration

Figure 6.1: Training loss for three configurations. (i) Unnormalised time and parameters (blue); (ii) normalised time, but only v_0 z–scored (orange); (iii) normalised time and three parameters m, μ, v_0 z–scored (green). The third setting stalls after $\approx 10^3$ epochs. [Insert figure from experiment]

Figure 6.1 shows the training loss for three set–ups.[1] Configuration (iii) collapses once the joint sample $m_z = \mu_z = v_{0,z} = 5$ is encountered; pre–activation statistics for the first layer become $\min z = -8.7$, $\max z = 7.1$, confirming saturation.

6.1.1.3 Mitigation strategies

(i) **Clip feature range:** restrict each z–score to $|x_z| \leq 3$ so that $|z^{(1)}| \lesssim 3$.

(ii) **Learnable input scaling:** prepend a 6×6 linear layer (no bias) that adapts per–feature gains during training.

(iii) **Switch activation:** replace tanh with SiLU or ReLU to avoid saturation.

(iv) **Stay in physical time:** forego normalising t entirely; then the derivative target remains $\dot{y}(0) = v_0$ and weight magnitudes stay moderate.

Table 6.2 summarises test–error statistics after applying each remedy.

---

[1] Exact hyper–parameters are given in Table 6.1.

6.1.1.4 Summary

Normalising the temporal axis in a PINN scales every time derivative by T. If several additional inputs are z–scored into a wide range (*e.g.,* $\pm 5\sigma$) and activations with bounded output (tanh) are used, the combined effect is to drive early–layer pre–activations far outside the linear region, causing vanishing gradients and failure to fit boundary conditions for large parameter values. Mitigations include clipping the input range, letting the network learn its own per–feature scale, or replacing tanh by a non–saturating activation function. non–saturating activation.

Table 6.1: Hyper–parameters used in the experiments.

Parameter	Value	Notes
Hidden layers	3	20 neurons each
Activation	`tanh`	unless stated otherwise
Optimiser	Adam	$\eta = 10^{-3}$
Training samples	500	per epoch

Table 6.2: Relative $\mathcal{L}_2$ error on an extreme test case ($m_z = \mu_z = v_{0,z} = 5$).

Configuration	Test error	Converged?
Baseline (norm. time, 3 z–scored features)	3.1e−1	no
Clipped inputs ($\pm 3\sigma$)	[fill]	yes
SiLU activation	[fill]	yes
Learnable scaler + tanh	[fill]	yes
Physical time (no t norm.)	[fill]	yes

Chapter 7

Discussion

7.1 Limitations

Only done it for normally non-correlated distributed input paramters

Have to assume an initial error for the PINN model when calcuating pi function.

Effect of the model error σ_e on MCMC performance

During the implementation of the Metropolis-Hastings algorithm, it was observed that the sampler failed to effectively explore the region of interest—specifically, the failure region where the limit state function $G(u) < 0$. This behavior was traced back to the choice of the model error σ_e, which appears in the probability of failure expression:

$$\pi(u) = \Phi\left(-\frac{G(u)}{\sigma_e}\right),$$

where Φ denotes the standard normal cumulative distribution function. When σ_e is set too high, the function $\pi(u)$ becomes overly smooth and flat. As a result, even input samples u with significantly negative values of $G(u)$ (which indicate strong failures) do not achieve noticeably higher values of $\pi(u)$ compared to safe samples. This flattens the distribution and causes the acceptance probability α in the Metropolis-Hastings step,

$$\alpha = \frac{\pi(u_{\text{prop}})f(u_{\text{prop}})}{\pi(u_{\text{curr}})f(u_{\text{curr}})},$$

to be dominated by the prior density $f(u)$. Consequently, the sampler essentially behaves as though it is drawing from the prior, rather than being guided by the structure of the failure region.

To address this issue, a smaller value of σ_e was chosen. This adjustment increased the sensitivity of $\pi(u)$ with respect to $G(u)$, thereby sharpening the probability landscape. In effect, samples with lower values of $G(u)$ received significantly higher weights, which improved the algorithm's ability to identify and explore the failure region.

This tuning of σ_e was essential for the Metropolis-Hastings sampler to focus more effectively on the regions of high failure probability.

Chapter 8

Conclusion

Bibliography

Bai, Zhiwei and Shufang Song (Nov. 2023). “Structural reliability analysis based on neural networks with physics-informed training samples”. In: Engineering Applications of Artificial Intelligence 126, p. 107157. DOI: `https://doi.org/10.1016/j.engappai.2023.107157`.

Chakraborty, Souvik (2020). Simulation free reliability analysis: A physics-informed deep learning based approach. arXiv: `2005.01302 [stat.ML]`. URL: `https://arxiv.org/abs/2005.01302`.

Meng, Zeng et al. (Sept. 2023). “PINN-FORM: a new physics-informed neural network for reliability analysis with partial differential equation”. In: Computer Methods in Applied Mechanics and Engineering 414, p. 116172. DOI: `https://doi.org/10.1016/j.cma.2023.116172`.

Raissi, Maziar, Paris Perdikaris, and George E Karniadakis (Feb. 2019). “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations”. In: Journal of Computational Physics 378, pp. 686–707. DOI: `https://doi.org/10.1016/j.jcp.2018.10.045`.

Zhang, Sheng et al. (June 2022). “Physics-informed neural networks for consolidation of soils”. In: Engineering Computations 39.7, pp. 2845–2865. DOI: `https://doi.org/10.1108/EC-08-2021-0492`.