Physics-informed neural networks (PINNs) versus finite elements for solving PDEs in continuum mechanics of solids

Eric M. Stewart
Final Project Report for MIT Course 18.337

May 15, 2023

Abstract

In the last three years in the continuum mechanics of solids literature, physics-informed neural networks (PINNs) which use neural networks for numerical solution of partial differential equations (PDEs) have mounted a challenge to the de facto ruler of PDE solvers in that field: the finite element method (FEM). Such PINNs have intrinsic advantages over FEM in that (i) they are mesh-free and (ii) they do not have an explicit time discretization scheme; however PINNs as a numerical method for solving PDEs are currently in their infancy and are notoriously difficult to work with. In this project, I use the modern language Julia to construct two PINNs for example problems in (i) linear elasticity and (ii) finite deformation hyperelasticity, documenting my challenges and strategies developed along the way. I compare the PINN results to FEM results both for the purposes of validation and of performance comparisons between the two methods. The PINN and FEM codes for this project are publicly available on GitHub, along with a list of tips and tricks for constructing PINNs that I accumulated during my project development.

Code and tips repository

`https://github.com/ericstewart36/pinnsforsolids`

Contents

1 Introduction **2**

2 A brief explanation of PINNs **2**

3 A PINN for linear elasticity **4**
3.1 Theory . . . 4
3.2 Validation problem: Quasi-static simple shear . . . 5
3.3 Naïve approach PINN results . . . 6
3.4 "Mixed formulation" PINN results . . . 7
3.5 Some remarks on the performance of the PINN versus FEM . . . 9

4 A PINN for hyperelasticity **9**
4.1 Theory . . . 10
4.2 Example problem: quasi-static large extension of a soft material . . . 10
4.3 Hyperelasticity PINN results . . . 12
4.4 Attempts at a PINN for dynamic hyperelasticity . . . 13

5 Source codes and general tips for PINNs **14**

6 Concluding remarks **14**

1 Introduction

The Finite Element Method (FEM) reigns supreme among PDE solvers in computational engineering design and analysis, having racked up over 80 years of applications in solid and fluid mechanics, heat transport, biomechanics, electromagnetics, and more (cf. e.g. Liu et al., 2022). The practices and algorithms underlying FEM trace their roots back to seminal works in continuum mechanics of solids, most notably

- Hrennikoff (1941) which first introduced the concept of a "mesh" discretization for elasticity problems, and
- Courant (1943) which introduced a variational method for solving mechanical equilibrium and vibration problems.

Since these two papers, developments tied to FEM have "fundamentally revolutionized the way we do scientific modeling and engineering design, ranging from automobiles, aircraft, marine structures, bridges, highways, and high-rise buildings" (Liu et al., 2022) and ultimately led to the very formation of my field of research: computational solid mechanics.

Numerous other methods exist for numerical solution of PDEs beyond FEM, e.g. finite difference methods, finite volume methods, boundary element methods, and most recently physics-informed neural networks (PINNs). All these methods have individual strengths and weaknesses, making them optimal in some scenarios and not in others; FEM's ease of application to complicated geometries has been central to its success in engineering design and analysis. However in the last three years, coupled to the rise of scientific machine learning, PINNs in particular have mounted a challenge to FEM in the solid mechanics literature. Applications of PINNs in mechanics include phase-field fracture (Goswami et al., 2020), small-strain elastodynamics (Rao et al., 2021), finite deformation hyperelasticity (Fuhg and Bouklas, 2022), and finite deformation plasticity (Niu et al., 2023). None of these applications has used Julia, instead using TensorFlow or PyTorch.

The primary goal of my project is to construct PINNs in Julia for solving continuum mechanics problems in (i) linear elasticity and (ii) finite deformation hyperelasticity, and to verify the results of the PINNs against predictions of FEM programs solving the same problems. Throughout this development and validation process I have documented my learnings as well as the benefits and shortfalls of PINNs relative to FEM — which will be very useful for my own research purposes going forward. PINNs are a young technology and are notoriously difficult to set up, run, and debug and so I have also constructed a list of PINN tips and tricks to share publicly on GitHub so that others can benefit from the learnings in my work.

2 A brief explanation of PINNs

There are four main ingredients in a PINN. Here I list each ingredient with a brief explanation, as well as the relevant Julia (`*.jl`) module(s). The excellent Julia module `NeuralPDE.jl` (Zubov et al., 2021) forms the backbone of the PINN code, and greatly facilitates tying each ingredient together.

1. **Loss function:** As with all PDE solution methods, we must enforce the governing PDE as well as any boundary and initial conditions (collectively, the "BCs"). In the context of the PINN, these are "weakly" enforced by minimizing the loss function

$$\begin{aligned}\mathcal{L}(\mathbf{u}) &\stackrel{\text{def}}{=} \mathcal{L}_{\text{PDE}}(\mathbf{u}) + \mathcal{L}_{\text{BCs}}(\mathbf{u}), \quad \text{where} \\ \mathcal{L}_{\text{PDE}}(\mathbf{u}) &= \int_B f\left(u_i, \frac{\partial u_i}{\partial x_i}, \frac{\partial u_i}{\partial x_j}, \frac{\partial^2 u_i}{\partial x_i^2}, \frac{\partial^2 u_i}{\partial x_i\, \partial x_j}, \text{etc.}\right) dv, \quad \text{and} \\ \mathcal{L}_{\text{BCs}}(\mathbf{u}) &= \int_{\partial B} (\text{ error in BCs }) \, da.\end{aligned} \tag{2.1}$$

   Here

   - we have introduced the vector degree of freedom $\mathbf{u} \in \mathbb{R}^2$.

- the function $f\left(u_i, \frac{\partial u_i}{\partial x_i}, \frac{\partial u_i}{\partial x_j}, \text{etc.}\right)$ in $\mathcal{L}_{\text{PDE}}(\mathbf{u})$ is the governing PDE with all non-zero terms moved to one side so that minimizing it enforces the PDE.
- the 2-D body is denoted as B and its 1-D boundary is denoted ∂B, as in Fig. 1.

The loss function (2.1) is constructed symbolically in my code using `ModelingToolkit.jl`. This loss function encodes the physical information of the problem being solved — it is the physics information or **PI** in "**PI**NN".

2. **Differentiable Neural Networks:** The degrees of freedom (or DOFs — here, the components of the vector $\mathbf{u}$) are expressed using a neural network as shown in Fig. 1. These neural network degrees of freedom are the **NN** in "PI**NN**".

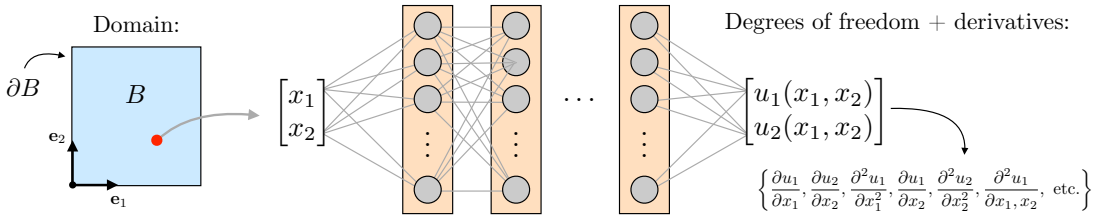


Figure 1: A neural network which maps $\{x_1, x_2\} \to \mathbf{u}(x_1, x_2)$. Spatial derivatives of this neural network are used to enforce the governing PDE in (2.1).

In the code, the neural network can be built using `Flux.jl` or `Lux.jl` and acts as a continuous mapping of the coordinates of the computational domain $\{x_1, x_2\}$ to values of the degrees of freedom $\mathbf{u}(x_1, x_2)$. Importantly, we can relatively easily differentiate neural network DOFs to compute derivatives which are needed to evaluate the loss function (2.1) — `NeuralPDE.jl` uses `Zygote.jl` for this purpose.

3. **Integral estimation:** The loss function $\mathcal{L}(\mathbf{u})$ in (2.1) is expressed in terms of integrals which must be estimated numerically. The `NeuralPDE.jl` package provides three primary methods for estimating these integrals: (i) grid methods, (ii) quasi-random methods, and (iii) Gaussian quadrature methods using the `Integrals.jl` package. These methods are illustrated in Fig. 2.

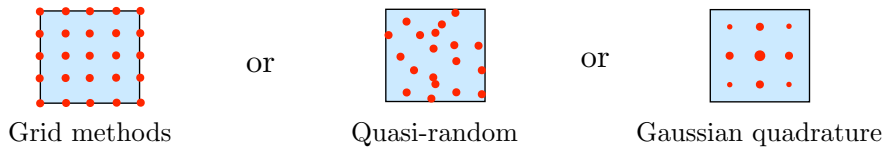


Figure 2: Illustrations of three methods for estimating the loss integral $\mathcal{L}(\mathbf{u})$.

4. Finally, we use the `Optimization.jl` package to iteratively minimize the loss function (2.1) using any of a wide array of available algorithms. Popular choices include Stochastic Gradient Descent, Adam, or BFGS / L-BFGS.

Remark. A fundamental assumption of PINNs is that the neural network which minimizes (2.1) best satisfies the governing PDE(s) and BC(s), and is therefore the solution to the boundary/initial value

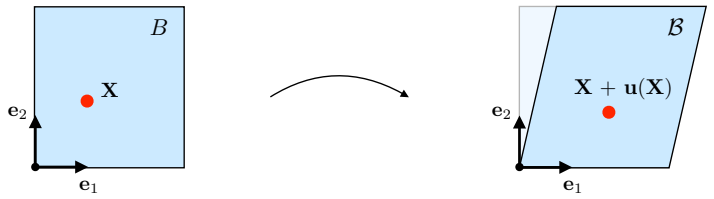
problem under study. We will see in this project that this assumption is not always true; perhaps it's more correct to say that the neural network which minimizes (2.1) is the solution which is the "least wrong" within the particular constraints of the neural network architecture and training regiment we construct.

---

3 A PINN for linear elasticity

The continuum theory of linear elasticity is only valid for materials undergoing a small amount of strain, at most 1-5% — however, it is significantly simpler mathematically than nonlinear "finite deformation" theories. We therefore use a problem in linear elasticity as a starting point for our PINN implementation, and will move to more complicated theory after carefully validating the case of linear elasticity.

3.1 Theory



Reference coordinates $\mathbf{X}$ Deformed coordinates $\mathbf{X} + \mathbf{u}(\mathbf{X})$

Figure 3: The reference and deformed configurations, related by the displacement field $\mathbf{u}(\mathbf{X})$.

We identify a macroscopically-homogeneous body B with the region of space it occupies in a fixed reference configuration, and denote by $\mathbf{X}$ an arbitrary material point of B. A motion of B to the deformed body $\mathcal{B}$ is then a smooth one-to-one mapping $\mathbf{x} = \mathbf{X} + \mathbf{u}(\mathbf{X})$, with accompanying small strain tensor[1]

$$\boldsymbol{\varepsilon} = \frac{1}{2}\left[\nabla\mathbf{u} + (\nabla\mathbf{u})^{\top}\right]. \tag{3.1}$$

The constitutive relation which delivers the stress tensor as a function of strain is then

$$\boldsymbol{\sigma} = 2\,G\,\boldsymbol{\varepsilon} + \left(K - \frac{2}{3}G\right)\operatorname{tr}(\boldsymbol{\varepsilon})\,\mathbf{1} \tag{3.2}$$

where G and K are the shear and bulk moduli respectively, which both have units of pressure. Finally, the equation of motion is the governing PDE and is given by

$$\operatorname{div}\boldsymbol{\sigma} + \mathbf{b} = \rho\ddot{\mathbf{u}}, \tag{3.3}$$

where $\mathbf{b}$ is a body force per unit volume e.g. gravity, and ρ is the mass density in $\mathrm{kg/m^3}$. The equation (3.3) is analogous to Newton's 2nd law but for a continuum body rather than a particle.

[1]Notation: We use standard notation of modern continuum mechanics (Gurtin et al., 2010). Specifically: ∇ and Div denote the gradient and divergence with respect to the material point $\mathbf{X}$ in the reference configuration, and $\Delta = \operatorname{Div}\nabla$ denotes the referential Laplace operator; grad div, and div grad denote these operators with respect to the point $\mathbf{x} = \boldsymbol{\chi}(\mathbf{X}, t)$ in the deformed body; a superposed dot denotes the material time-derivative. Throughout, we write $\mathbf{F}^{-1} = (\mathbf{F})^{-1}$, $\mathbf{F}^{-\top} = (\mathbf{F})^{-\top}$, etc. We write $\operatorname{tr}\mathbf{A}$, $\operatorname{sym}\mathbf{A}$, $\operatorname{skw}\mathbf{A}$, $\mathbf{A}_0$, and $\operatorname{sym}_0\mathbf{A}$ respectively, for the trace, symmetric, skew, deviatoric, and symmetric-deviatoric parts of a tensor $\mathbf{A}$. Also, the inner product of tensors $\mathbf{A}$ and $\mathbf{B}$ is denoted by $\mathbf{A}:\mathbf{B}$, and the magnitude of $\mathbf{A}$ by $|\mathbf{A}| = \sqrt{\mathbf{A}:\mathbf{A}}$.

3.2 Validation problem: Quasi-static simple shear

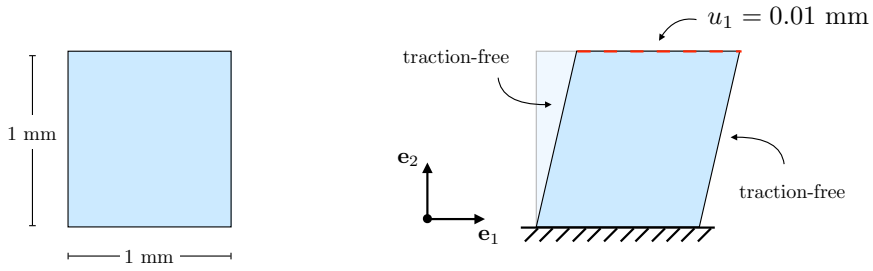


Figure 4: Problem setup for quasi-static simple shear.

As a validation problem, we consider the case of quasi-static simple shear illustrated in Fig. 4. The bottom edge of the 1 mm square is kept fixed, while the top edge is displaced 0.01 mm in the horizontal direction (a shear strain of $\gamma = [0.01\,\text{mm} \,/\, 1.0\,\text{mm}] = 1\%$) and the left and right edges are kept traction-free. We neglect body and inertial forces so that $\mathbf{b} = \ddot{\mathbf{u}} = \mathbf{0}$, and using (3.1) and (3.2) we may write (3.3) purely in terms of displacement vector components as[2]

$$\begin{aligned}
\left(K + \frac{1}{3}G\right)\left(\frac{\partial^2 u_1}{\partial x_1^2} + \frac{\partial^2 u_2}{\partial x_1\,\partial x_2}\right) + G\left(\frac{\partial^2 u_1}{\partial x_1^2} + \frac{\partial^2 u_1}{\partial x_2^2}\right) = 0, \\
\left(K + \frac{1}{3}G\right)\left(\frac{\partial^2 u_1}{\partial x_1\,\partial x_2} + \frac{\partial^2 u_2}{\partial x_2^2}\right) + G\left(\frac{\partial^2 u_2}{\partial x_1^2} + \frac{\partial^2 u_2}{\partial x_2^2}\right) = 0,
\end{aligned} \tag{3.4}$$

and the boundary conditions are[3]

$$\left.\begin{aligned} u_1(x_1, 0) &= 0 \\ u_2(x_1, 0) &= 0 \end{aligned}\right\} \text{Bottom edge fixed}$$

$$\left.\begin{aligned} u_1(x_1, 1) &= 0.01\,\text{mm} \\ u_2(x_1, 1) &= 0 \end{aligned}\right\} \text{Top edge shear}$$

$$\left.\begin{aligned} &\left[\left(K - \tfrac{2}{3}G\right)\left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2}\right) + G\left(\frac{\partial u_1}{\partial x_1}\right)\right]\Bigg|_{(0,x_2)} = 0 \\ &G\left(\frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2}\right)\Bigg|_{(0,x_2)} = 0 \end{aligned}\right\} \text{Left edge traction-free} \tag{3.5}$$

$$\left.\begin{aligned} &\left[\left(K - \tfrac{2}{3}G\right)\left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2}\right) + G\left(\frac{\partial u_1}{\partial x_1}\right)\right]\Bigg|_{(1,x_2)} = 0 \\ &G\left(\frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2}\right)\Bigg|_{(1,x_2)} = 0 \end{aligned}\right\} \text{Right edge traction-free}$$

For simplicity and to keep the different contributions to the loss function unity order of magnitude we use unit material parameters, which in our unit system is $G = 1$ MPa and $K = 1$ MPa.

Remark.

In FEM, traction-free boundary conditions are enforced on a given edge "naturally" by the nature of the weak form, i.e. if one does not apply Dirichlet or Neumann boundary conditions to an edge that edge is naturally traction-free. This is not the case for PINNs, since they do not use a similar "weak form" of the governing PDE but rather the loss function (2.1) — in fact, before I realized I had to explicitly enforce the traction-free conditions $(3.5)_{5-8}$ I observed erroneous results from my PINNs.

---

[2] One of the foundation assumptions of linear elasticity is that deformations are sufficiently small that $\dfrac{\partial}{\partial x_i}(\cdot) \approx \dfrac{\partial}{\partial X_i}(\cdot)$.

[3] The traction-free boundary condition requires that $\boldsymbol{\sigma}\mathbf{n} = \mathbf{0}$, which for the right and left edges means $\sigma_{11} = \sigma_{12} = 0$.

3.3 Naïve approach PINN results

For the initial PINN architecture, I used two separate neural networks for $u_1(x_1, x_2)$ and $u_2(x_1, x_2)$ which each had 2 hidden layers that were 5 nodes wide, with hyperbolic tangent activation functions.[4] I used the quadrature integral estimation method, and training consisted of (i) 2000 steps of Adam with a learning rate of 10^{-3}, then (ii) 2000 steps of Adam with a learning rate of 10^{-4}, and finally (iii) 10,000 steps of LBFGS.[5] The loss during training, contours of the solution variables $u_1(x_1, x_2)$ and $u_2(x_1, x_2)$ on the deformed body, reference FEM solution generated from code I wrote in the open-source finite element program FEniCS (Logg et al., 2012; Alnæs et al., 2015), and contours of relative error between the PINN and FEM are shown in Figure 5. Visualization of the FEniCS results are created in the open-source program ParaView (Ahrens et al., 2005), and visualizations of the PINN results are created using Julia's `Plots.jl` module.

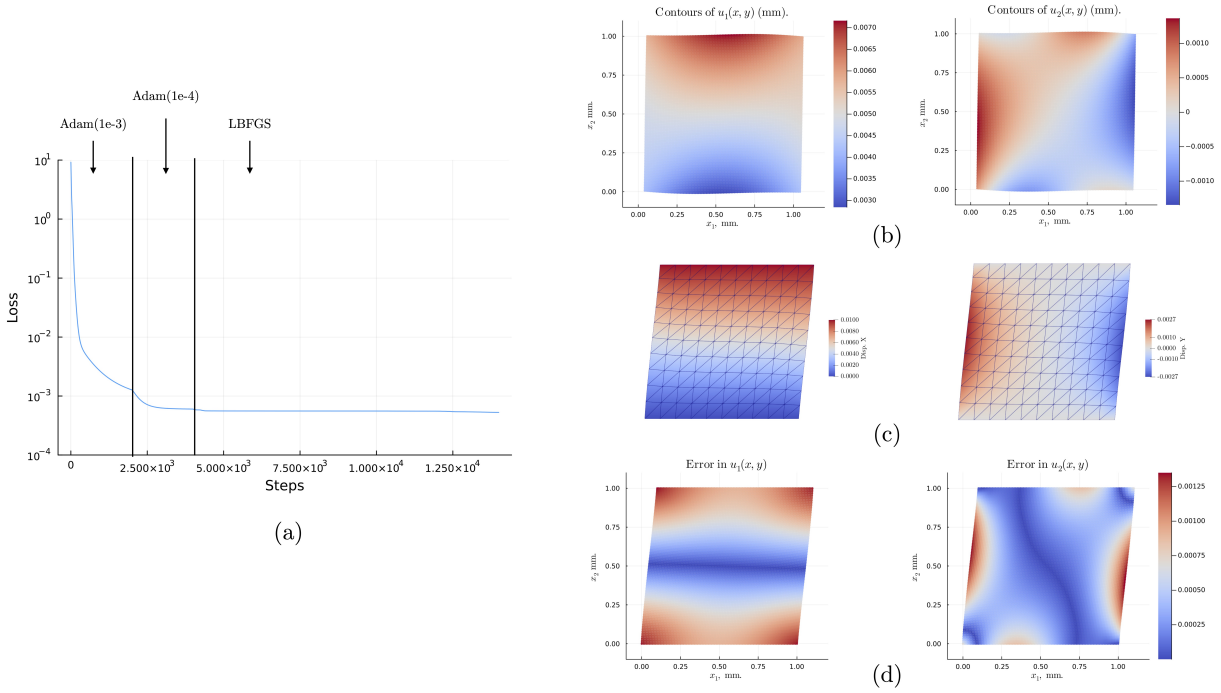


Figure 5: Results from the (u_1, u_2)-based PINN: (a) loss during training, (b) contours of u_1 and u_2 on the deformed body from the PINN, (c) contours of u_1 and u_2 on the deformed body from FEM, and (d) contours of error in u_1 and u_2 for the PINN relative to FEM on the deformed body. Deformed geometries are shown with displacements multiplied by a factor of 10.

Although the final loss value is reasonably low (around 10^{-3}) the contours of u_1 and u_2 are qualitatively quite different from those of the reference FEM solution. For example, while u_1 in the PINN does increase with x_2, the actual values at the top and bottom edges are quite off from their enforced values of 0.01 and 0 mm respectively, and the values of the degrees of freedom are not uniform along the top and bottom edges as required. The PINN seems to have captured the broad strokes of the problem, but the enforcement of boundary conditions in particular is quite poor and the deformed shape of the body — indicative of how well the governing PDE is being solved — is quite different from that of the FEM solution.

[4] Through much experimentation with neural networks as deep as 6 layers and as wide as 100 nodes per layer, I found that the number of hidden layers did not make much of a difference in the final results, nor did the dimension of the hidden layers so long as that dimension is $\gtrapprox 4$. Therefore I do not dwell on the specific architecture of the neural network chains in this work and simply present results from a single reasonably-sized architecture. Also, the hyperbolic tangent activation function for the hidden layers seemed to yield the best results.

[5] Not much happens during LBFGS training for this initial study, but the LBFGS stage proved essential for future studies.

3.4 "Mixed formulation" PINN results

After much experimentation, I found that the best remedy to the poor results of the PINN in Fig. 5 was to recast the problem as a "mixed formulation" in terms of the five degrees of freedom[6]

$$\{u_1, u_2, \sigma_{11}, \sigma_{22}, \sigma_{12}\}, \tag{3.6}$$

so that all of the governing equations and boundary conditions are exclusively in terms of first derivatives of the degrees of freedom, in contrast to (3.4) which includes second derivatives. Specifically, the equation of motion becomes

$$\begin{aligned} \frac{\partial \sigma_{11}}{\partial x_1} + \frac{\partial \sigma_{12}}{\partial x_2} &= 0, \\ \frac{\partial \sigma_{12}}{\partial x_1} + \frac{\partial \sigma_{22}}{\partial x_2} &= 0. \end{aligned} \tag{3.7}$$

However, we must now enforce the constitutive relation (3.2) as three extra governing equations, viz.

$$\begin{aligned} \sigma_{11} &= \left(K + \frac{1}{3}G\right)\left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2}\right) + 2\,G\,\frac{\partial u_1}{\partial x_1} \\ \sigma_{22} &= \left(K + \frac{1}{3}G\right)\left(\frac{\partial u_1}{\partial x_1} + \frac{\partial u_2}{\partial x_2}\right) + 2\,G\,\frac{\partial u_2}{\partial x_2} \\ \sigma_{11} &= G\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1}\right). \end{aligned} \tag{3.8}$$

Then, the boundary conditions are simply

$$\begin{aligned} &\left.\begin{aligned} u_1(x_1, 0) &= 0 \\ u_2(x_1, 0) &= 0 \end{aligned}\right\} \text{Bottom edge fixed} \\ &\left.\begin{aligned} u_1(x_1, 1) &= 0.01\,\text{mm} \\ u_2(x_1, 1) &= 0 \end{aligned}\right\} \text{Top edge shear} \\ &\left.\begin{aligned} \sigma_{11}\big|_{(0,x_2)} &= 0 \\ \sigma_{12}\big|_{(0,x_2)} &= 0 \end{aligned}\right\} \text{Left edge traction-free} \\ &\left.\begin{aligned} \sigma_{11}\big|_{(1,x_2)} &= 0 \\ \sigma_{12}\big|_{(1,x_2)} &= 0 \end{aligned}\right\} \text{Right edge traction-free} \end{aligned} \tag{3.9}$$

To reiterate, the main difference between this formulation and the previous one is that it involves lower-order derivatives for all quantities — at the cost of introducing three extra degrees of freedom and three extra governing equations for the stress tensor components.

The architecture for the neural networks, integral estimation methods, and training methods are identical to the previous section.[7] The results of the "mixed formulation" PINN are vastly improved over the previous results, suggesting that with all else equal, our PINN has greater accuracy for problems involving lower-order derivatives. The loss during training, contours of the solution variables $u_1(x_1, x_2)$ and $u_2(x_1, x_2)$ on the deformed body, reference FEM solution generated from code I wrote in the open-source finite element program FEniCS, and contours of relative error between the PINN and FEM are shown in Figure 6.

[6] The linear elastic stress tensor is symmetric so that $\sigma_{12} = \sigma_{21}$.

[7] Although the neural network for each degree of freedom is the same size as in the previous problem, the fact that we have 5 degrees of freedom for the mixed formulation means we necessarily have more total parameters. However, the greater expressive power of the model itself does not seem to be the reason for the improved performance, since I ran experiments on 2-degree of freedom models with similar number of parameters which still did not yield satisfactory results.

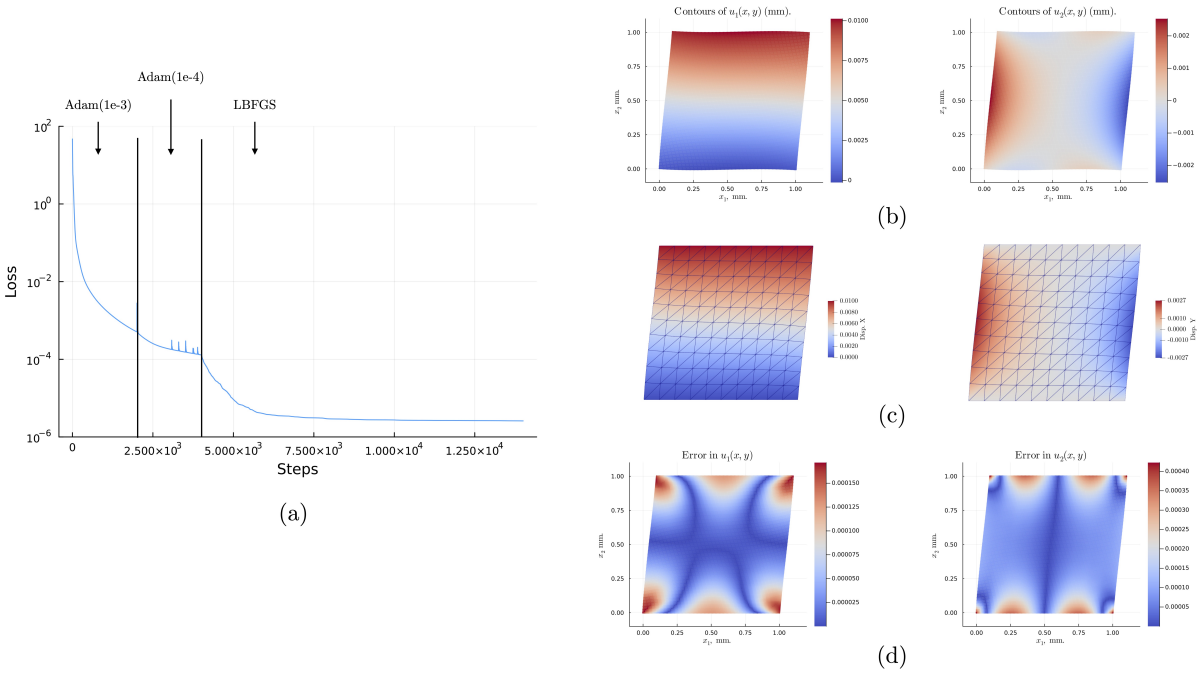


Figure 6: Results from the $(u_1, u_2, \sigma_{11}, \sigma_{22}, \sigma_{12})$-based PINN: (a) loss during training, (b) contours of u_1 and u_2 on the deformed body from the PINN, (c) contours of u_1 and u_2 on the deformed body from FEM, and (d) contours of error in u_1 and u_2 for the PINN relative to FEM on the deformed body. Deformed geometries are shown with displacements multiplied by a factor of 10.

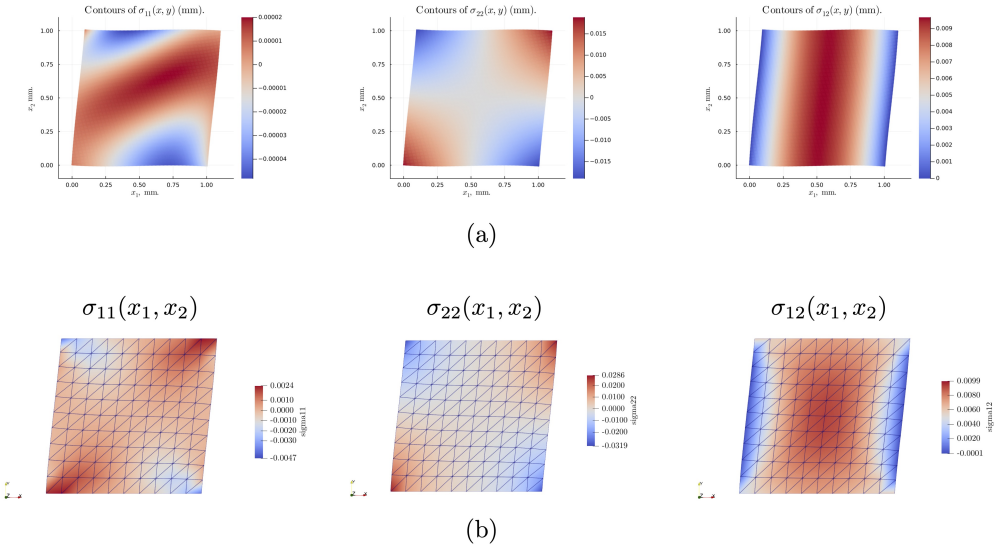


Figure 7: Contours of the components of the stress tensor predicted by (a) the mixed formulation PINN and (b) FEM. Deformed geometries are shown with displacements multiplied by a factor of 10.

The contours of u_1 from the mixed formulation PINN are pretty much spot-on with the FEM solution — the RMS error value between the PINN and FEM is 5.962e-5 mm or ≈ 60 nm. The RMS error for u_2 is similarly low, 1.217e-4 mm or ≈ 120 nm. Additionally, the deformed shape of the square agrees quite well with that of the FEM, indicating good satisfaction of the governing PDEs. In Fig. 7 we also plot contours

of the components of the stress tensor predicted by the PINN versus those from FEM, and can see that all are in reasonably good agreement. Most importantly, the value of the shear stress σ_{12} in the middle of the domain goes to 0.01 MPa, as required by the analytical expression

$$\sigma_{12} = G\,\gamma\,, \tag{3.10}$$

where here $G = 1$ MPa and $\gamma = 0.01$. We take the good quantitative agreement in displacement and stress fields between the PINN and FEM as validation of our PINN implementation of linear elasticity.

3.5 Some remarks on the performance of the PINN versus FEM

In general, the PINNs in my work have significantly worse performance (higher runtime) than my FEM code which solves an identical problem. This is due to several factors, but in my view the two biggest are:

- FEM algorithms have had the benefit of several decades of performance improvements while PINNs are really still in their infancy, and
- my PINNs rely on the most resource-intensive part of neural network development — the training — to come up with a solution to a PDE.

As a benchmark, the time to generate a solution to the quasi-static simple shear problem for the PINN, FEniCS-based FEM (written in Python), and Abaqus-based FEM (written in FORTRAN) are shown in Table 1. Here, the timing was measured against the longest step in the solution procedure — for the PINN this was the training, and for the FEM codes this was the Newton-Raphson solver. My machine is a 2019 Macbook Pro with a 2.4 GHz Quad-Core Intel Core i5 processor and 16 GB of RAM.

Table 1: Time to solve the quasi-static simple shear problem for the PINN versus two FEM softwares.

PINN	FEniCS FEM (Python)	Abaqus FEM (FORTRAN)
23 m 29.64 s	0.538 s	0.200 s

Clearly, there is a huge gap in performance between the PINN and both FEM softwares. Since `Flux.jl` and `Lux.jl` support GPU-based training, I attempted close this gap by by training the PINN neural network on a GPU. I was eventually able to run the neural network training on a GPU on JuliaHub, but this ultimately did not actually speed up the training process for the PINN. I was unable to find out why the GPU did not speed up the training process within the time constraints of the project, so stuck with my CPU for the results shown.

4 A PINN for hyperelasticity

Having carefully validated our PINN for linear elasticity, we move on to more complicated theory – finite deformation hyperelasticity. The theory in this section is much more general than that of the previous section, and is not limited to small deformations. Such a theory is typically applicable to soft materials, especially elastomers and polymer gels. For simplicity, however, we still use unit material parameter values of $G = 1$ MPa and $K = 1$ MPa.

4.1 Theory

The kinematics of finite deformations are described by the deformation gradient tensor[8]

$$\mathbf{F} = \mathbf{1} + \nabla \mathbf{u}, \tag{4.2}$$

which encodes all the stretching and rotation information of the body's deformation. Further, the volumetric deformation is measured by

$$J = \det \mathbf{F} > 0, \tag{4.3}$$

so that $\mathbf{F}$ is invertible. The free energy of the deformable body is then a function of $\mathbf{F}$, and here we will use the compressible Neo-Hookean form for the free energy[9]

$$\psi_{\text{R}} = \frac{1}{2} G(\operatorname{tr}(\mathbf{F}^{\top}\mathbf{F}) - 3 - 2\ln J) + \frac{1}{2} K(J-1)^2. \tag{4.4}$$

In finite deformations there are many different stress measures, but here we will use the Piola stress tensor $\mathbf{T}_{\text{R}}$. For thermodynamic consistency, the Piola stress tensor is given by the derivative of the free energy function with respect to $\mathbf{F}$,

$$\mathbf{T}_{\text{R}} = \frac{\partial \psi_{\text{R}}}{\partial \mathbf{F}} = G(\mathbf{F} - \mathbf{F}^{-\top}) + K(J-1)\mathbf{F}^{-\top}. \tag{4.5}$$

Remark. Writing out the components of the expression (4.5) for the Piola stress in terms of u_1 and u_2 is quite lengthy and complex, since it involves the inverse of $\mathbf{F}$ as well as its determinant. However, I was able to use `ModelingToolkit.jl` to evaluate the stress tensor symbolically in the PINN code rather than writing it out in full.

Finally, the referential form of the equation of motion is the governing PDE, given by

$$\operatorname{Div} \mathbf{T}_{\text{R}} + \mathbf{b}_{\text{R}} = \rho_{\text{R}} \ddot{\mathbf{u}}, \tag{4.6}$$

where $\mathbf{b}$ is a body force per unit volume of the reference body e.g. gravity, and ρ_{R} is the referential mass density in kg/m^3.

4.2 Example problem: quasi-static large extension of a soft material

To test out our finite-deformation hyperelasticity PINN, we model an example problem that involves large levels of stretching well beyond the limit of applicability for linear elasticity, shown in Fig. 8. The upper edge of the specimen is extended 1 mm in the vertical direction while the bottom edge is held fixed. During the deformation, the sides of the specimen contract inwards significantly to accommodate the large levels of stretch in the vertical direction. As before, we neglect body forces and assume the deformation is quasi-static so that $\mathbf{b}_{\text{R}} = \ddot{\mathbf{u}} = \mathbf{0}$.

[8] For this project, I assume plane strain conditions prevail so that

$$\mathbf{F} = \begin{bmatrix} 1 + \dfrac{\partial u_1}{\partial X_1} & \dfrac{\partial u_1}{\partial X_2} & 0 \\ \dfrac{\partial u_2}{\partial X_1} & 1 + \dfrac{\partial u_2}{\partial X_2} & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.1}$$

[9] Generally, quantities with the subscript $(\cdot)_{\text{R}}$ indicate a quantity measured in the reference body.

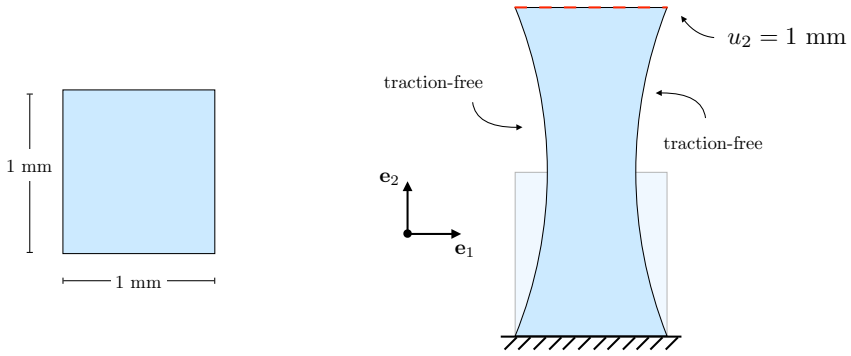


Figure 8: Problem setup for large extension.

We again use a "mixed formulation" for solving the problem in terms of displacements and the components of the stress tensor. Importantly, the Piola stress tensor is not symmetric so we must use six degrees of freedom

$$\{u_1, u_2, T_{\mathrm{R},11}, T_{\mathrm{R},22}, T_{\mathrm{R},12}, T_{\mathrm{R},21}\}. \tag{4.7}$$

Neglecting body and inertial forces, the equation of motion is then

$$\begin{aligned} \frac{\partial T_{\mathrm{R},11}}{\partial X_1} + \frac{\partial T_{\mathrm{R},21}}{\partial X_2} &= 0, \\ \frac{\partial T_{\mathrm{R},12}}{\partial X_1} + \frac{\partial T_{\mathrm{R},22}}{\partial X_2} &= 0, \end{aligned} \tag{4.8}$$

as before, the constitutive relations must be enforced as four extra equations to determine the four independent in-plane components of the Piola stress $\mathbf{T}_{\mathrm{R}}$.[10] Finally, the boundary conditions are

$$\begin{aligned} &\left.\begin{aligned} u_1(x_1, 0) &= 0 \\ u_2(x_1, 0) &= 0 \end{aligned}\right\} \text{Bottom edge fixed} \\ &\left.\begin{aligned} u_1(x_1, 1) &= 0 \\ u_2(x_1, 1) &= 1.0\,\mathrm{mm} \end{aligned}\right\} \text{Top edge pull} \\ &\left.\begin{aligned} T_{\mathrm{R},11}\big|_{(0,x_2)} &= 0 \\ T_{\mathrm{R},21}\big|_{(0,x_2)} &= 0 \end{aligned}\right\} \text{Left edge traction-free} \\ &\left.\begin{aligned} T_{\mathrm{R},11}\big|_{(1,x_2)} &= 0 \\ T_{\mathrm{R},21}\big|_{(1,x_2)} &= 0 \end{aligned}\right\} \text{Right edge traction-free} \end{aligned} \tag{4.9}$$

Remark. To obtain results which better enforce the boundary conditions in the hyperelasticity PINN, we modify the loss function $(2.1)_1$ to the augmented form

$$\mathcal{L}(\mathbf{u}) = \lambda_{\mathrm{PDE}}\, \mathcal{L}_{\mathrm{PDE}}(\mathbf{u}) + \lambda_{\mathrm{BCs}}\, \mathcal{L}_{\mathrm{BCs}}(\mathbf{u}), \tag{4.10}$$

with specific values $\lambda_{\mathrm{PDE}} = 1$ and $\lambda_{\mathrm{BCs}} = 10$. The general strategy (4.10) of re-weighting different loss function contributions has been successfully employed in applications in the literature by e.g. Rao et al. (2021).

---

[10] For brevity, the lengthy expressions for the components of the Piola stress tensor in terms of u_1 and u_2 are omitted here, cf. the Remark on p. 10.

4.3 Hyperelasticity PINN results

The architecture for the neural networks and integral estimation methods are identical to those used for linear elasticity. However for the hyperelasticity problem I found that training took much longer due to the increased complexity of the governing equations and boundary conditions, and that LBFGS was the best-performing optimization algorithm by far. I therefore trained the neural network for only 500 steps of LBFGS, after which the loss seemed to have converged. The loss during training, contours of the solution variables $u_1(x_1, x_2)$ and $u_2(x_1, x_2)$ on the deformed body, reference FEM solution generated from code I wrote in the open-source finite element program FEniCS are shown in Figure 9.

Although the loss only converges to a value on the order of 1, the hyperelasticity PINN still does quite well predicting the deformed shape of the body versus FEM. The PINN result displays the characteristic "hourglass" type shape we'd expect, and matches the values of u_1 through the body very well. The lateral contraction predicted by the PINN is close to that predicted by the FEM, although the PINN has contracted by 0.15 mm on each side in contrast to the 0.13 mm predicted by FEM. Near the corners of the finite element mesh, you can see that the mesh is perhaps not fine enough of a resolution to fully resolve the local deformation and appears piecewise linear — the PINN, not being confined to a spatial mesh, does not suffer from this issue and appears smooth near the corners.[11]

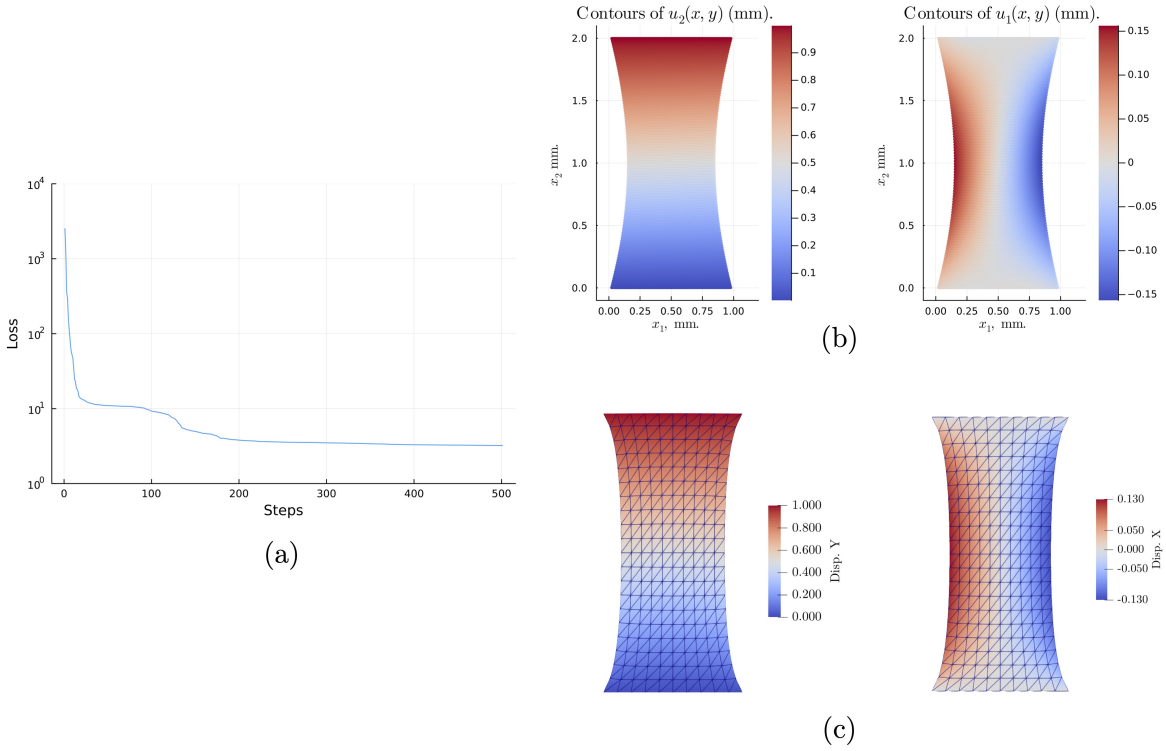
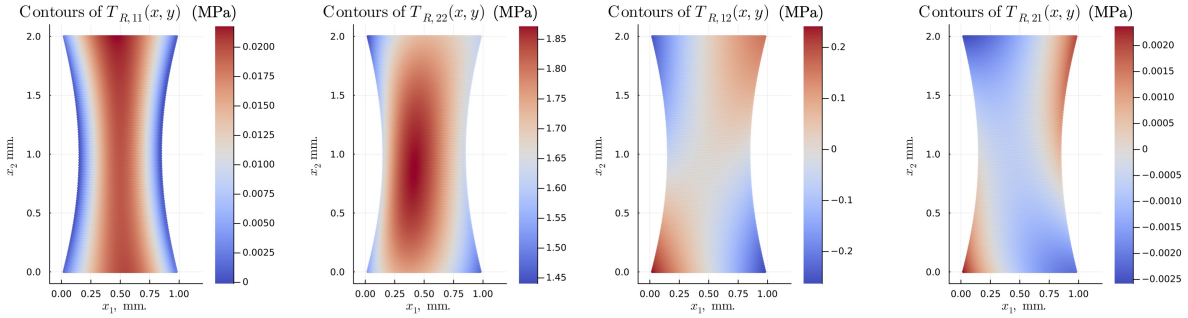


Figure 9: Results from the hyperelastic PINN: (a) loss during training, (b) contours of u_1 and u_2 on the deformed body from the PINN, (c) contours of u_1 and u_2 on the deformed body from FEM. Importantly, the deformed body in all cases is shown at true scale.

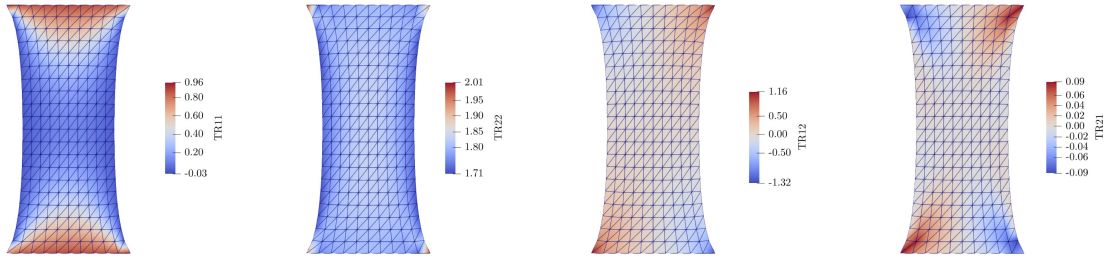
The Piola stress components predicted by the PINN and the corresponding FEM are shown in Fig. 10. Although the scale of the legend is a bit different for some stress components in the PINN versus FEM, in general the distribution of stress values in the deformed body is similar for both solvers. Since we are applying a stretch in the $\mathbf{e}_2$-direction, the normal component of the stress in this direction $T_{\mathrm{R},22}$ is the most important component — and both the PINN and the FEM report a value of $T_{\mathrm{R},22} = 1.85$ MPa in the bulk of the body. Although the quantitative agreement for the other Piola stress tensor components is weaker, the

[11] Probably the best way to resolve which of the PINN or FEM predictions is more correct near the corners is to run an actual physical experiment!

qualitative patterns of each component (where the stresses are locally higher and lower in the body) agree well between the PINN and FEM. I believe even better quantitative agreement in these other Piola stress components could be achieved with greater training time.



(a)



(b)

Figure 10: Contours of the Piola stress components $T_{\mathrm{R},11}, T_{\mathrm{R},22}, T_{\mathrm{R},12}$, and $T_{\mathrm{R},21}$ in the deformed body predicted by (a) the PINN and (b) the FEniCS FEM. The deformed body is shown at true scale.

4.4 Attempts at a PINN for dynamic hyperelasticity

Having established the baseline capabilities for my PINN to model finite deformation hyperelasticity of rubber-like materials, I then spent some time trying to add inertial effects (or dynamics, modeled by the acceleration term $\rho_{\mathrm{R}}\,\ddot{\mathbf{u}}$ in (4.6)). Adding these effects introduces significant complexity to the PINN, specifically

- a new material parameter ρ_{R} for the mass density of the material,
- a new temporal dimension t as input to the PINN, and
- second derivatives of the displacement with respect to time,

and a similar "mixed formulation" necessitates the use of eight total degrees of freedom

$$\{u_1, u_2, \dot{u}_1, \dot{u}_2, T_{\mathrm{R},11}, T_{\mathrm{R},22}, T_{\mathrm{R},12}, T_{\mathrm{R},21}\}. \tag{4.11}$$

I was ultimately unsuccessful in getting reasonable results from the dynamic PINN in time for the project, but I have publicly posted the code for a dynamic hyperelastic PINN and reference FEM code on the project GitHub as a starting point for someone else, or myself in the future to pursue this direction. To the best of my knowledge, such a dynamic hyperelastic PINN has not been previously demonstrated in the literature and so would be a very nice and novel contribution.

5 Source codes and general tips for PINNs

I have constructed a publicly available GitHub repository,

`https://github.com/ericstewart36/pinnsforsolids`

which contains all the Julia PINN codes and Python FEniCS codes used in this work, as well as a list of helpful tips, tricks, and best practices for PINNs that I learned during this project. I now present the tips and tricks from my work here:

- **BC enforcement:** Unlike FEM, PINNs only "weakly" enforce the boundary conditions via the loss function — that is, they permit solutions which do not completely satisfy the applied boundary conditions. If you have trouble with BC enforcement, you can use the augmented loss function scheme (4.10) to improve the enforcement of the BCs by setting $\lambda_{\text{BCs}} > \lambda_{\text{PDE}}$.
  - The paper Rao et al. (2021) presents a scheme for forcible boundary condition enforcement using a composition of three functions: one which satisfies the boundary conditions precisely, one which measures the distance to the nearest boundary condition "edge", and one which represents the solution within the domain. Implementing this approach in `NeuralPDE.jl` could be a nice future project.
- **Low-order derivatives:** I found that reformulating my problem in terms of lower-order derivatives — even at the cost of extra governing equations and degrees of freedom — was *essential* to getting accurate results from my PINN. This observation has been made before in the literature for TensorFlow-based PINNs by Rao et al. (2021).
- **Neural network architecture:** I found that solving PDEs on a unit square does not require very deep or wide neural networks — 2 hidden layers with 5 hidden nodes each and hyperbolic tangent activation was sufficient. In general, the optimization process and problem formulation matter much more than neural network architecture.
- **Optimization algorithms:** Adam and BFGS/LBFGS seem to work the best for training PINNS — Adam for a fast "burn-in" process and BFGS/LBFGS to drill down to finer scales.
- **Material parameters:** I found that using physical values of the material parameters (e.g. steel has $G = 210,000$ MPa and $K = 245,000$ MPa) led to poor convergence behavior since the different components of the loss function had different orders of magnitude. For best convergence in physical problems, it is essential to devise a unit system and normalization scheme which renders the different components of the loss functions similar orders of magnitude.
  - In this work I sidestep the issue of material parameter scaling by simply using unit material parameters rather than physically representative values. However, for PINNs to be useful in engineering they must be able to accommodate physical values of material parameters.

6 Concluding remarks

This project has been a very valuable learning experience for me about PINNs and the Julia language. I heavily leveraged many existing Julia tools — especially `NeuralPDE.jl` — and was still ultimately surprised by how lengthy and difficult it was to set up a PINN for problems in continuum mechanics and get reasonable comparison to FEM results. I constructed a list of useful tips and tricks for PINNs based on my experience and have made it publicly available, so that ideally others don't have to re-learn these items.

In the end, I was able to construct PINNs for (i) linear elasticity and (ii) finite deformation hyperelasticity which yield favorable comparisons to finite element solutions of the same problems. Although PINNs for similar constitutive models and governing equations exist in the literature, these are the first such PINNs written using Julia. In terms of performance, the PINNs are on the order of 1,000 times slower than FEM — but to be fair the technology is still in its infancy in comparison to FEM.

While PINNs are a bit cumbersome, we are only at the beginning of their story and I am optimistic about their future applications in continuum mechanics of solids. Their development is tethered to the development of one of the fastest-growing fields of scientific machine learning — neural networks — and since they are mesh-free and lack any explicit time discretization I believe they will be superior to FEM for select problems in continuum mechanics. Examples of such problems include e.g. resolving the stress singularity at a crack tip or modeling shocks in solids.

References

J. Ahrens, B. Geveci, and C. Law. ParaView: An End-User Tool for Large-Data Visualization. In *Visualization Handbook*, pages 717–731. Butterworth-Heinemann, Jan. 2005.

M. Alnæs, J. Blechta, J. Hake, A. Johansson, B. Kehlet, A. Logg, C. Richardson, J. Ring, M. E. Rognes, and G. N. Wells. The FEniCS Project Version 1.5. *Archive of Numerical Software*, 3(100), Dec. 2015.

R. Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49(1):1–23, 1943.

J. N. Fuhg and N. Bouklas. The mixed Deep Energy Method for resolving concentration features in finite strain hyperelasticity. *Journal of Computational Physics*, 451:110839, Feb. 2022.

S. Goswami, C. Anitescu, S. Chakraborty, and T. Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106: 102447, Apr. 2020.

M. E. Gurtin, E. Fried, and L. Anand. *The mechanics and thermodynamics of continua*. Cambridge University Press, 2010.

A. Hrennikoff. Solution of Problems of Elasticity by the Framework Method. *Journal of Applied Mechanics*, 8(4):A169–A175, 1941.

W. K. Liu, S. Li, and H. S. Park. Eighty Years of the Finite Element Method: Birth, Evolution, and Future. *Archives of Computational Methods in Engineering*, 29(6):4431–4453, Oct. 2022.

A. Logg, K.-A. Mardal, and G. Wells. *Automated Solution of Differential Equations by the Finite Element Method*. Springer, 2012.

S. Niu, E. Zhang, Y. Bazilevs, and V. Srivastava. Modeling finite-strain plasticity using physics-informed neural network and assessment of the network performance. *Journal of the Mechanics and Physics of Solids*, 172:105177, 2023.

C. Rao, H. Sun, and Y. Liu. Physics-Informed Deep Learning for Computational Elastodynamics without Labeled Data. *Journal of Engineering Mechanics*, 147(8):04021043, Aug. 2021.

K. Zubov, Z. McCarthy, Y. Ma, F. Calisto, V. Pagliarino, S. Azeglio, L. Bottero, E. Luján, V. Sulzer, A. Bharambe, N. Vinchhi, K. Balakrishnan, D. Upadhyay, and C. Rackauckas. NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations, 2021. URL `https://arxiv.org/abs/2107.09443`.