Physics-Informed Deep Learning and its Application in Computational Solid and Fluid Mechanics

Presented by: Alexandros Papados (AMSC)
Advisor: Professor Balakumar Balachandran (ENME)

University of Maryland, College Park:
Applied Mathematics, Applied Statistics, & Scientific Computing

May 4, 2021

Table of Contents

Table of Contents

Project Proposal Recap

- Investigate PINNs and their ability to solve forward and inverse problems in solid and fluid mechanics
- Compare to classical numerical methods such FVM, FEM, and NLS
- **Problems in question:**
  - Conservation Laws - Burgers equation, Euler equations for compressible flow [1] – Fluid Mechanics
  - Plane stress linear elasticity boundary value problem [2] – Solid Mechanics

Why PINNs?

Advantages:

- Simplistic implementation to solve PDEs compared to FVM and FEM
- Parameter estimation requires less data and is faster than standard parameter estimation methods
- Meshless method
- Purpose is to "solve supervised learning tasks while respecting any given law of physics described by a general nonlinear partial differential equation" (Karniadakis et al.)

Drawbacks:

- Forward problem is slower than classical PDE solvers at times
- Weak theoretical grounding

PINNs Universal Approximation Theorem

Theorem (Pinkus, 1999):

Let $\boldsymbol{m}^i \in \mathbb{Z}_+^d$, $i = 1, ..., s$,and set $m = \max_{i=1,\ldots,s} |\boldsymbol{m}^i$. Assume $\sigma \in C^m(\mathbb{R})$ and is not a polynomial. Then the space of single hidden layer neural nets:

$$\mathcal{M}(\sigma) = span\{\sigma(\boldsymbol{w} \cdot \boldsymbol{x} + b) : \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

is dense in $C^{\boldsymbol{m}^1,\ldots,\boldsymbol{m}^s}(\mathbb{R}^d)$. In other words, for any $f \in C^{\boldsymbol{m}^1,\ldots,\boldsymbol{m}^s}(\mathbb{R}^d)$, any compact $K \subset \mathbb{R}^d$, and any $\epsilon > 0$, there exists a $g \in \mathcal{M}(\sigma)$ satisfying

$$\max_{\boldsymbol{x}\in K} \left| D^{\boldsymbol{k}} f(\boldsymbol{x}) - D^{\boldsymbol{k}} g(\boldsymbol{x}) \right| < \epsilon$$

for all $\boldsymbol{k} \in \mathbb{Z}_+^d$ for which $\boldsymbol{k} \leq \boldsymbol{m}^i$.

Project Accomplishments

- Created the first PIDL solver which solves a general hydrodynamic shock-tube problems, W-PINNs-DE
  - Bypasses theoretical and computational limitations faced by original PINNs
  - Solves shock-tube problems to higher accuracy in comparison to other PINNs and finite volume methods
- Demonstrated W-PINNs ability to solve inverse hydrodynamic shock-tube problems
- Used W-PINNs to solve plane stress linear elasticity boundary value problems (LEBVP)

Table of Contents

Table of Contents

LEBVP

- **Motivation:** Solid and Structural Mechanics
- The material matrix for an isotropic material in an elasticity boundary value problem consisting of two parameters, E - Young's Modulus, and ν - Poisson Ratio.
- Let $M_{E\nu} = \frac{E}{(1+\nu)(1-2\nu)}$. Then the material matrix is defined by:

$$C = M_{E\nu} \begin{pmatrix} 1-\nu & 0 & 0 & 0 & \nu & 0 & 0 & 0 & \nu \\ 0 & 1-2\nu & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1-2\nu & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1-2\nu & 0 & 0 & 0 & 0 & 0 \\ \nu & 0 & 0 & 0 & 1-\nu & 0 & 0 & 0 & \nu \\ 0 & 0 & 0 & 0 & 0 & 1-2\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1-2\nu & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1-2\nu & 0 \\ \nu & 0 & 0 & 0 & \nu & 0 & 0 & 0 & 1-\nu \end{pmatrix}$$

- Solve for the amount of deformation a material undergoes under prescribed body force, $\boldsymbol{f}$, and surface force, $\boldsymbol{g}$

LEBVP

- The deformation tensor is defined as

$$\boldsymbol{u} = \left(u_1, u_2, u_3\right)^T$$

- u_i corresponds to the deformation in the x, y, and z direction, and $u_i : \mathbb{R}^3 \rightarrow \mathbb{R}$.

- We solve for the deformation of a material undergoing loading by solving the equilibrium equation:

$$\begin{cases} -\nabla \cdot \boldsymbol{\sigma} = \boldsymbol{f}, & x \in \Omega \subset \mathbb{R}^3 \\ \boldsymbol{u} = 0, & x \in \Gamma_D \\ \boldsymbol{\sigma} \cdot \boldsymbol{\nu} = \boldsymbol{g}, & x \in \Gamma_N \end{cases} \tag{1}$$

where,

$$\boldsymbol{\sigma} = C\boldsymbol{\epsilon}, \quad \epsilon_{ij} = \frac{1}{2}\left[\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right] + \frac{1}{2}\sum_{k=1}^{3} \frac{\partial u_i}{\partial x_k}\frac{\partial u_j}{\partial x_k}, \quad i, j = 1, 2, 3$$

LEBVP

Since we are considering a LEBVP, the parabolic terms vanish, hence

$$\begin{aligned}\boldsymbol{\epsilon} &= \frac{1}{2}\left[\nabla \boldsymbol{u} + \nabla \boldsymbol{u}^T\right] \\ &= A\nabla \boldsymbol{u} \\ &= \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \frac{\partial u_1}{\partial x_1} \\ \frac{\partial u_1}{\partial x_2} \\ \frac{\partial u_1}{\partial x_3} \\ \frac{\partial u_2}{\partial x_1} \\ \frac{\partial u_2}{\partial x_2} \\ \frac{\partial u_2}{\partial x_3} \\ \frac{\partial u_3}{\partial x_1} \\ \frac{\partial u_3}{\partial x_2} \\ \frac{\partial u_3}{\partial x_3} \end{pmatrix}\end{aligned}$$

Plane Stress

A material undergoes plane stress provided the stress vector is zero in a specific plane. Here we chose to have zero stresses in the $z - direction$, hence,

$$\sigma_{3j} = \sigma_{i3} = 0, \quad \text{for } i, j = 1, 2, 3$$

Then the stress tensor in the $xy - direction$ is defined by:

$$\begin{aligned} \boldsymbol{\sigma} &= C_{E\nu}\boldsymbol{\epsilon} \\ &= \frac{E}{(1-\nu^2)} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & (1-\nu)/2 \end{pmatrix} \begin{pmatrix} \epsilon_{11} \\ \\ \epsilon_{22} \\ \\ \gamma_{12} \end{pmatrix} \end{aligned}$$

where $\gamma_{12} = \left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1}\right)$

Forward Problem

LEBVP

$$G\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right]+G\left(\frac{1+\nu}{1-\nu}\right)\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 v}{\partial y \partial x}\right]=\sin(2\pi x)\sin(2\pi y)$$

$$G\left[\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right]+G\left(\frac{1+\nu}{1-\nu}\right)\left[\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 u}{\partial x \partial y}\right]=\sin(\pi x)+\sin(2\pi y)$$

where $G=\frac{E}{2(1+\nu)}$, $E=1$ GPa is the Young's modulus, and $\nu=0.3$ is the Poisson ratio of the material. The problem has fixed boundary conditions.

Table of Contents

Issues Using PINNs

- PINNs have much difficulty approximating simple boundary conditions
- Immense error at the boundary
- Proposed rectification methods [19] do not generalize to solving LEBVP

Algorithm 1: W-PINNs Algorithm

1 Generate weights $\theta \in \mathbb{R}^k$ and a deep neural network (DNN), $\tilde{\boldsymbol{U}}(x, y, \theta)$, where (x, y) are inputs to the network, and $\tilde{\boldsymbol{U}} = [\tilde{u}, \tilde{v}]$ are the outputs. The number of layers, neurons per layer, and activation functions for each layer are prescribed by the user.

2 Sample points (x_n, y_n) from Ω and w_n from $\partial\Omega$. Let N_f, N_{BC} correspond to the number of points sampled from the interior and boundary, respectively.

3 Generate $G(\theta)$:

$$G(\theta) = \frac{1}{N_f} \left|\left| \nabla \cdot \tilde{\boldsymbol{\sigma}}(x, y, \theta) + \boldsymbol{f} \right|\right|_{\Omega}^2 + \frac{\omega_{BC}}{N_{BC}} \left|\left| \tilde{\boldsymbol{U}}(x, y, \theta) - \boldsymbol{U}(x, y) \right|\right|_{\partial\Omega}^2$$

where $\omega_{BC} = 10,000$

4 Update θ by performing stochastic gradient descent:

$$\theta = \theta - \eta \nabla_\theta G(\theta)$$

where η is the learning rate.

W-PINNs Architecture

Each neural network will have:

- 7 layers
- 30 neurons per layer
- $\tanh(\cdot)$ activation function for nonlinear layers
- learning rate of 0.0005
- No random sampling of computational domain
- $199,350$ epochs

Domain I

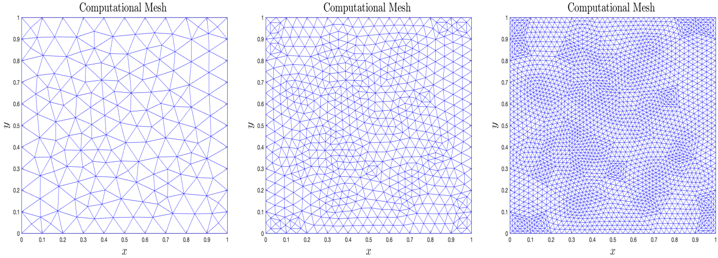


Figure 1 – Mesh I, II, III

Domain I

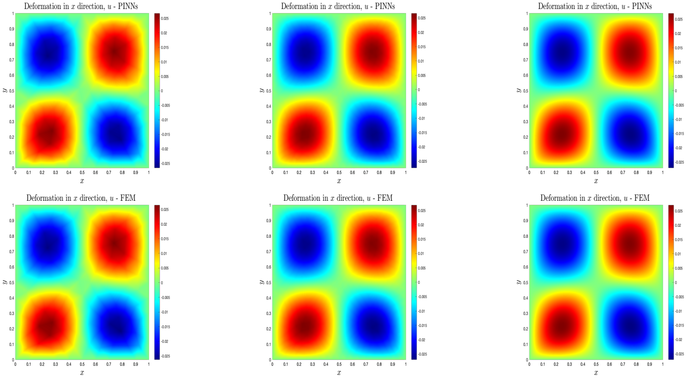


Figure 2 – Top: W-PINNs , Bottom: FEM, Left to Right: Mesh I, II, III

Domain I

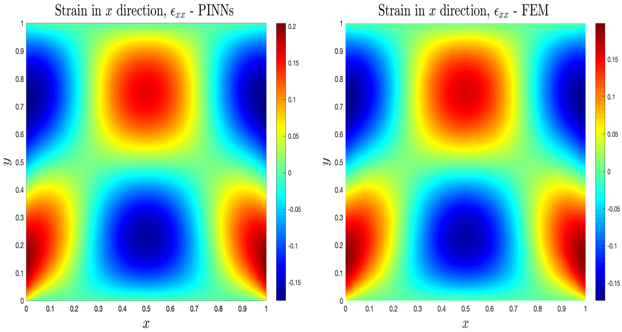


Figure 3 – Strain in x direction, ϵ_{xx} - Mesh III

Domain I

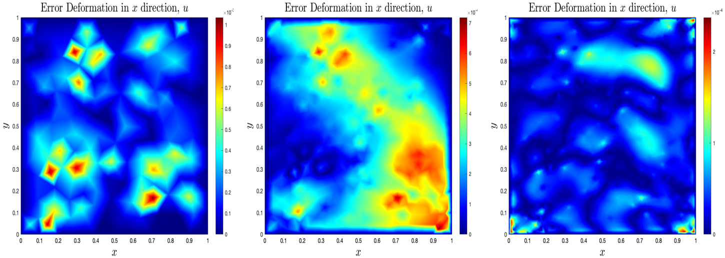


Figure 4 – Absolute Error for deformation in x direction, Left to Right: Mesh I, II, III

Domain I

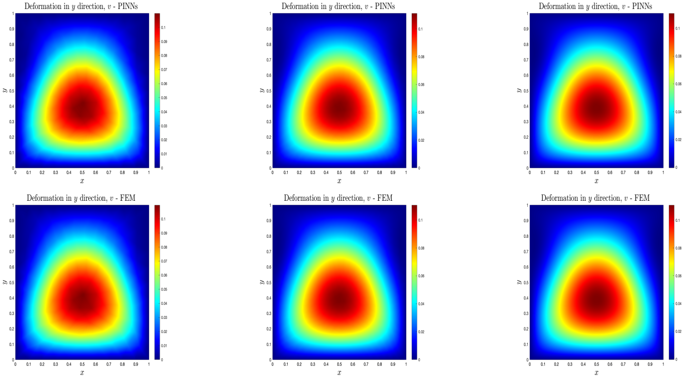


Figure 5 – Top: W-PINNs , Bottom: FEM, Left to Right: Mesh I, II, III

Domain I

- Introduction
- Solid Mechanics
- Linear Elasticity Boundary Value Problems
- W-PINNs
- Software and Coding Languages

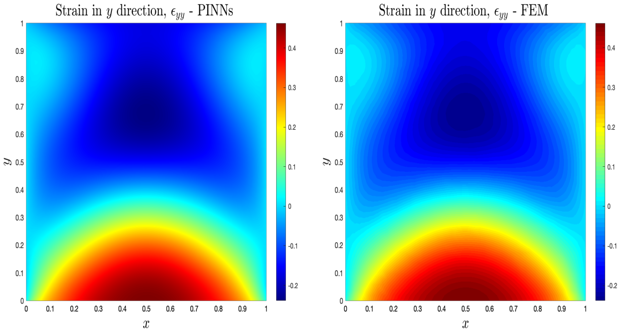


Figure 6 – Strain in y direction, ϵ_{yy} - Mesh III

Domain I

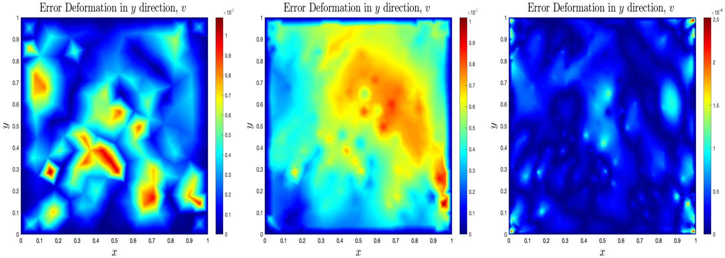
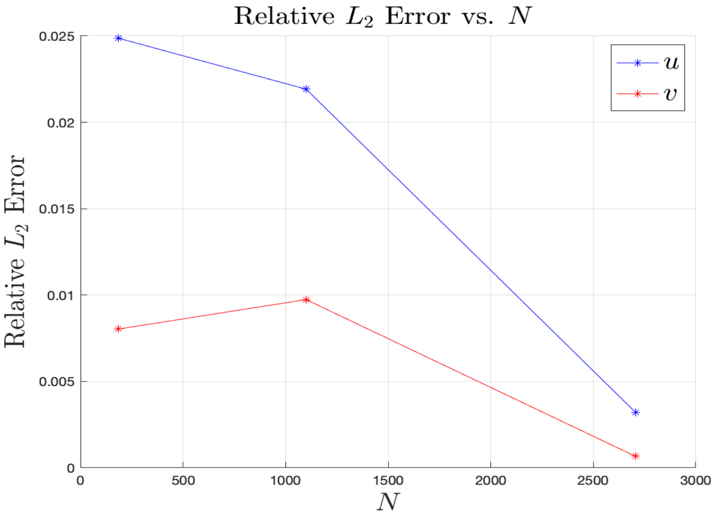


Figure 7 – Absolute Error for deformation in x direction, Left to Right: Mesh I, II, III

Domain I



Relative L_2 Error vs. N
Relative L_2 Error
N
0.025
0.02
0.015
0.01
0.005
0
0
500
1000
1500
2000
2500
3000
u
v

Domain I

Domain	Mesh I	Mesh II	Mesh III
$\frac{\|\|u_{approx}-u_{exact}\|\|_2}{\|\|u_{exact}\|\|_2}$	$2.4e-02$	$2.3e-02$	$9.4e-04$
$\frac{\|\|v_{approx}-v_{exact}\|\|_2}{\|\|v_{exact}\|\|_2}$	$7.3e-03$	$9.0e-03$	$4.7e-04$

Table 1 – Relative L_2 errors

Domain II

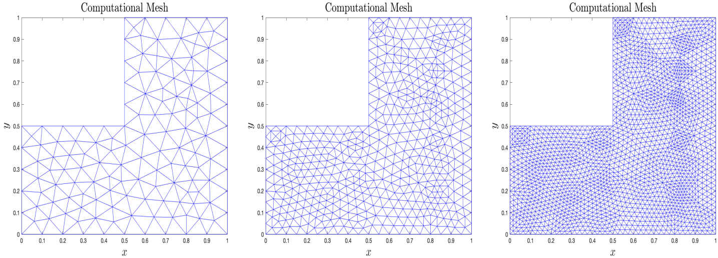


Figure 8 – Computational Mesh IV, V, and VI

Domain II

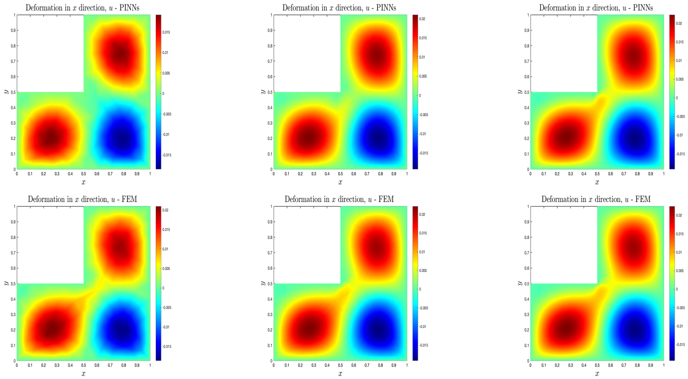


Figure 9 – Top: W-PINNs , Bottom: FEM, Left to Right: Mesh IV, V, VI

Domain II

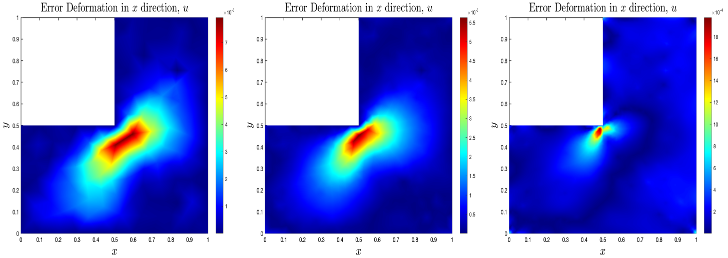


Figure 10 – Absolute Error for deformation in x direction, Left to Right: Mesh IV, V, VI

Domain II

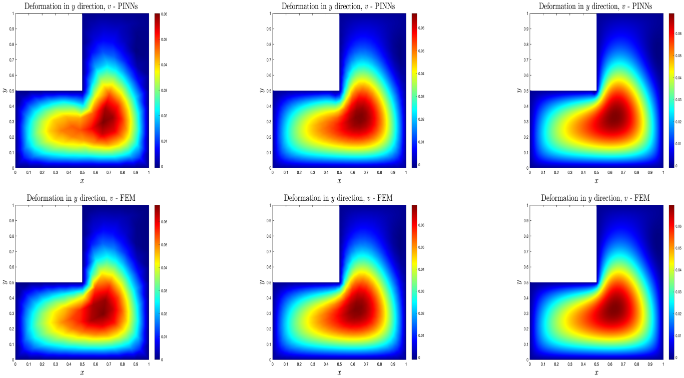


Figure 11 – Top: W-PINNs , Bottom: FEM, Left to Right: Mesh IV, V, VI

Domain II

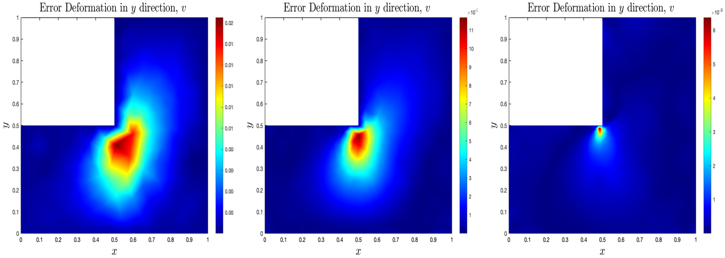
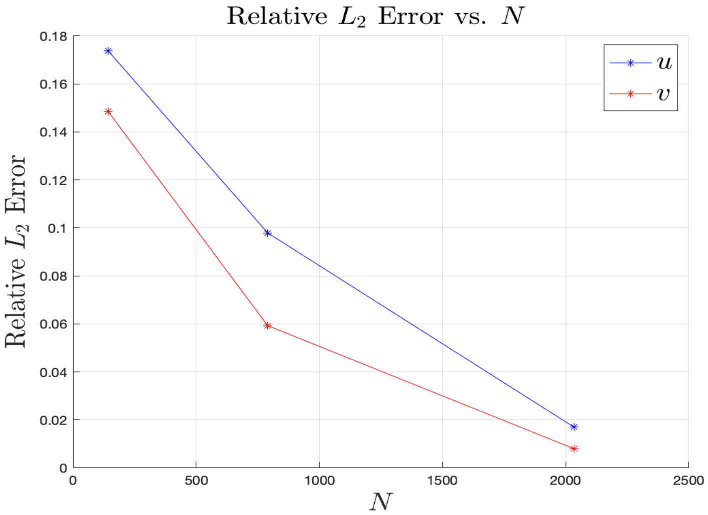


Figure 12 – Absolute Error for deformation in y direction, Left to Right: Mesh IV, V, VI

Domain II

Introduction
Solid Mechanics
Linear Elasticity Boundary Value Problems
W-PINNs
Software and Coding Languages



Domain II

Domain	Mesh IV	Mesh V	Mesh VI
$\frac{\|\|u_{approx} - u_{exact}\|\|_2}{\|\|u_{exact}\|\|_2}$	$1.7e-01$	$9.0e-02$	$1.4e-02$
$\frac{\|\|v_{approx} - v_{exact}\|\|_2}{\|\|v_{exact}\|\|_2}$	$1.5e-01$	$5.9e-02$	$9.0e-03$

Table 2 – Relative L_2 errors

Additional Mesh Refinement

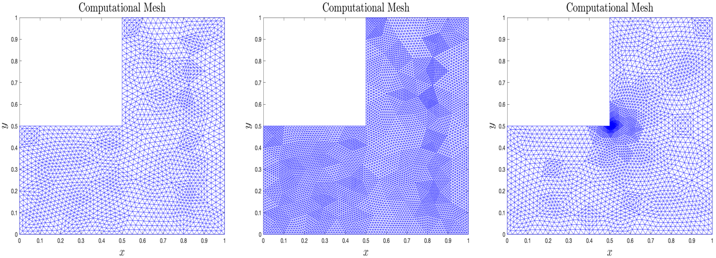


Figure 13 – Left: Mesh VI, Middle: Refined Mesh, Right: Locally Refined Mesh

Absolute Errors

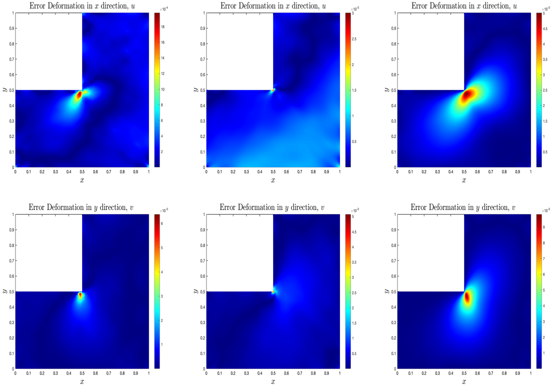


Figure 14 – Top: Absolute error of deformation in x direction for each mesh. Bottom: Absolute error of deformation in y direction for each mesh. Left: Mesh VI, Middle: Refined Mesh, Right: Locally Refined Mesh

Refinement Errors

Domain	Mesh VI	Refined	Locally Refined
$\frac{\|\|u_{approx} - u_{exact}\|\|_2}{\|\|u_{exact}\|\|_2}$	$1.4e-02$	$4.9e-02$	$7.8e-02$
$\frac{\|\|v_{approx} - v_{exact}\|\|_2}{\|\|v_{exact}\|\|_2}$	$9.0e-03$	$2.0e-02$	$4.8e-02$

Table 3 – Relative L_2 errors

Domain III

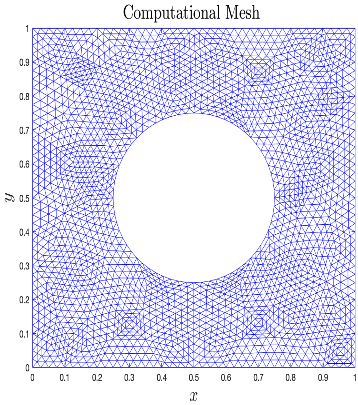


Figure 15 – Mesh VII, $N = 2,320$

Domain III

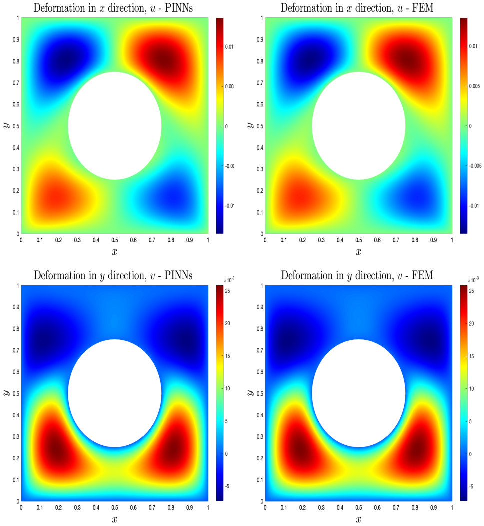


Figure 16 – Deformation in x and y direction

Domain III

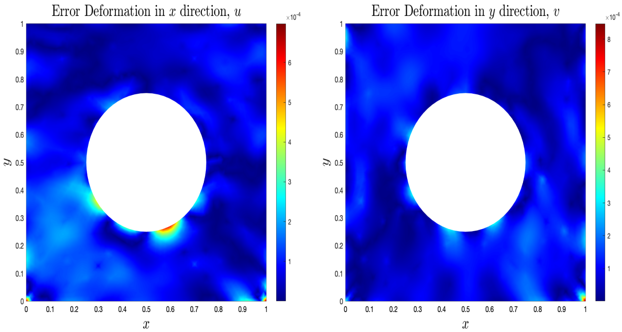


Figure 17 – Absolute Error

$\frac{\|\|u_{approx} - u_{exact}\|\|_2}{\|\|u_{exact}\|\|_2}$	$\frac{\|\|v_{approx} - v_{exact}\|\|_2}{\|\|v_{exact}\|\|_2}$
$9.9e-03$	$9.8e-03$

Table 4 – Relative L_2 errors

Domain IV

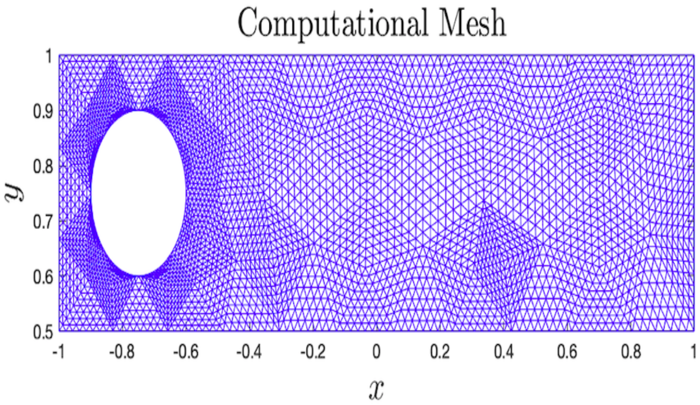


Figure 18 – Mesh VIII, $N = 3,600$

Domain IV

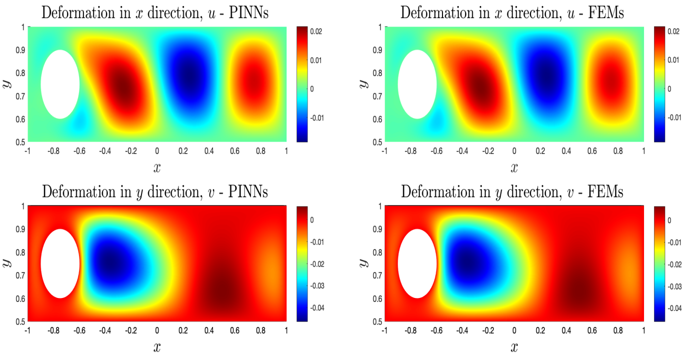


Figure 19 – Deformation in x and y direction

Domain IV

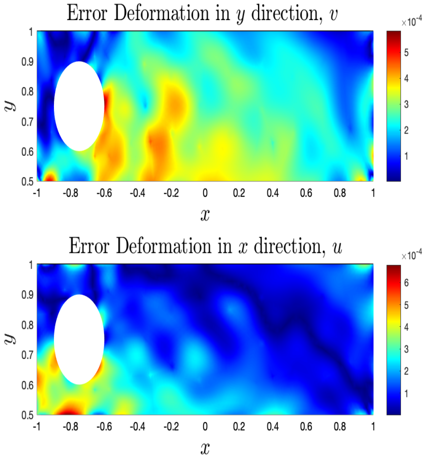


Figure 20 – Absolute Error

Domain IV

$\frac{\lvert\lvert u_{approx} - u_{exact} \rvert\rvert_2}{\lvert\lvert u_{exact} \rvert\rvert_2}$	$\frac{\lvert\lvert v_{approx} - v_{exact} \rvert\rvert_2}{\lvert\lvert v_{exact} \rvert\rvert_2}$
$3.6e-03$	$2.5e-03$

Table 5 – Relative L_2 errors

Conclusion

- W-PINNs accurately compute solutions on moderately refined mesh $N < 4,000$
- Over refinement is computationally costly and accumulates higher error
- Local refinement increases error in refinement areas
- 2,000 - 4,000 training points is recommended

Table of Contents

Software and Coding Languages

Coding Languages

- Python
- MATLAB

Libraries

- PyTorch

Reference

[1] Roesner, K. G, Leutloff, D, Srivastava, R. C. (1995). *Computational fluid dynamics: Selected topics*. Berlin: Springer.

[2] Chen, Y, Press, H. H. (2013). *Computational Solid Mechanics Structural Analysis and Algorithms*. Berlin: De Gruyter.

[3] Thomas, J. W. (1999). *Numerical Partial Differential Equations: Conservation Laws and Elliptic Equations*. New York: Springer.

[4] Golsorkhi, N. A, Tehrani, H. A. (2014). *Levenberg-marquardt Method For Solving The Inverse Heat Transfer Problems*. Journal of Mathematics and Computer Science, 13(04), 300-310. doi:10.22436/jmcs.013.04.03

[5] Chen, Z. (2010). *Finite Element Methods and its Applications*. Berlin: Springer.

[6] Mao, Z, Jagtap, A. D, Karniadakis, G. E. (2020). *Physics-informed neural networks for high-speed flows*. Computer Methods in Applied Mechanics and Engineering, 360, 112789. doi:10.1016/j.cma.2019.112789

Reference

[7] Raissi, M, Perdikaris, P, Karniadakis, G. (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. doi:10.1016/j.jcp.2018.10.045

[8] Sirignano, J, Spiliopoulos, K. (2018). *DGM: A deep learning algorithm for solving partial differential equations*. Journal of Computational Physics, 375, 1339-1364. doi:10.1016/j.jcp.2018.08.029

[9] Lu, L, Jagtap, A. D, Karniadakis, G. E. (2019). *DeepXDE: A Deep Learning Library for Solving Differential Equations*. ArXiv.org,arxiv.org/abs/1907.04502.

[10] Cybenko, G. (1989). *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals, and Systems, 2(4), 303-314. doi:10.1007/bf02551274

[11] Mishra, S, Molinaro, R. (2020). *Estimates on the generalization error of Physics Informed Neural Networks (PINNs) for approximating PDEs II: A class of inverse problems*. https://arxiv.org/abs/2007.01138

Reference

[12] Pinkus, A. (1999). *Approximation theory of the MLP model in neural networks*. Acta Numerica, 8, 143-195. doi:10.1017/s0962492900002919

[13] Sod, G. A. (1978). *A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws*. Journal of Computational Physics, 27(1), 1-31. doi:10.1016/0021-9991(78)90023-2

[14] LeVeque, R. J. (2011). *Finite volume methods for hyperbolic problems*. Cambridge: Cambridge Univ. Press.

[15] Michoski, C, Milosavljević, M, Oliver, T, Hatch, D. R. (2020). *Solving differential equations using deep neural networks*. Neurocomputing, 399, 193-212. doi:10.1016/j.neucom.2020.02.015

[16] Patel, R. G, Manickam, I, Trask, N, Wood, M. A. (2020). *Thermodynamically consistent physics-informed neural networks for hyperbolic systems*. doi:https://arxiv.org/abs/2012.05343

References

[17] Kim, J, Kim, A, Lee, S. *Artificial Neural Network-Based Automated Crack Detection and Analysis for the Inspection of Concrete Structures*. Applied Sciences, vol. 10, no. 22, 2020, p. 8105., doi:10.3390/app10228105.

[18] Cazzanti, L, Khan, M, Cerrina, F. *Parameter Extraction with Neural Networks*. Metrology, Inspection, and Process Control for Microlithography XII, 1998, doi:10.1117/12.308780.

[19] Wang, S, Tang, Y, Perdikaris, P. *Understanding and mitigating gradient pathologies in physics-informed neural networks*. 2020.

The End

Thank You! Questions?