Physics Informed Neural Network for 2D Diffusion Equation

Eyob Ghebreiesus*

Department of Mechanical, Materials and Aerospace Engineering
Illinois Institute of Technology[b], Armour College of Engineering[c]
10 W 35th St Chicago, IL 60616

Abstract

This project implements a physics-informed neural network (PINN) for solving the 2D diffusion equation with varying diffusion coefficients. The Green's function to the 2D Diffusion equation was solved using inverse Fourier transform first. The model was then trained on synthetic data generated by the exact solution of the diffusion equation, and its accuracy was assessed by comparing predicted data with test data using contour plots and normal graphs. In addition, the PINN model was applied to simulate Brownian motion of a particle in a 2D domain. The results showed that the PINN model was capable of accurately predicting the solution of the 2D diffusion equation and simulating spatial diffusion problems. This approach can potentially be extended to more complex systems and can provide a useful tool for modeling physical phenomena in data science and artificial intelligence.

Keywords: *PINNs, PDE, 2D Diffusion, Green's Functions, Heat Flux, Flow, Tensor, Keras, Data*

1 Introduction

The development of Physics-Informed Neural Networks (PINNs) has recently gained significant attention due to its potential to solve complex physical problems. Using PINNs to test the diffusion 2D equation for data visualization involves training a neural network to approximate the solution while incorporating physical constraints and data, and then using the trained network to predict the solution and visualize the results. PINNs is a technique that combines deep learning methods with partial differential equations (PDEs) to solve inverse problems or to learn the dynamics of physical systems [1]. In general, a neural network can be defined as a type of machine learning algorithm inspired by the structure and function of the human brain. Neural networks can be used for a wide range of tasks, including classification, regression, and pattern recognition, and they have been applied in areas such as computer vision, natural language processing, and speech recognition [2]. In order to use PINNs to test the diffusion 2D equation for data[d] visualization a proper set up of the full PDE equation is needed.

The fundamental solution to the 2D-Diffusion equation in polar coordinates is given by the Green function described below as:

$$\Theta(\vec{\mathbf{r}}, \mathbf{t}) \cdot \left(\frac{1}{4\pi \mathbf{D}\mathbf{t}}\right) \cdot e^{\frac{-\mathbf{r}^2}{4\mathbf{D}\mathbf{t}}}$$

$$\mathcal{L} = \partial_{\mathbf{t}} - \mathbf{D} \cdot \nabla^2$$

where:

- Θ is the spatial function of interest in polar (r^2) coordinate. It is the concentration or density of the diffusing species in cartesian (x, y) coordinates and time t
- $\mathcal{L}$ is the linear differential operator [3][4].
- D is the diffusivity constant (a.k.a κ) in $\frac{m^2}{s}$.

In PINNs, neural networks are used to approximate the solutions to PDEs, and the physical knowledge is encoded in the form of constraints on the network. These constraints are usually derived from the governing equations of the physical system being modeled [5], and they ensure that the neural network satisfies the physics of the system. By using PINNs, in this project we will combine the strengths of both neural networks and physics-based modeling to solve complex problems in using the 2D diffusion equation. The proposed model will be tested for a successful prediction by comparing the loss function and results of the plots with the actual equation. The model should reasonably predict and asses the given data subject to an acceptable error in the realm of data science.

2 Method

This methodology involves generating a capable PINN model for the 2D-Difussion equation using a numerical method, and tensor flows. To begin, the diffusion equation is a partial differential equation that describes the spread of a quantity over time, given its initial distribution. In 2D, the diffusion equation is derived using the Green's functions

*Correspondence Author: *eghebreiesus@hawk.iit.edu*
[b]Illinois Tech main page: *https://www.iit.edu* ©2023
[c]Department of Aerospace Engineering: *engineering@iit.eedu*
[d]*https://github.com/eyobghiday/PINNs-in-Fluid-Mechanics*

[6] and linear operator PDE's. Guaranteed boundary conditions have to be specified in order to solve the diffusion equation. For this project both the Dirichlet's and Neumann's boundary conditions will be utilised [7]. A neural network architecture is then chosen to approximate the solution to the diffusion equation. The neural network takes in the input variables (x, y, t) and outputs the predicted value of $u(x, y, t)$ as an example. The architecture in Figure 1 is designed based on the complexity of the problem and the available data. Following this we can use the PINNs technique to train the neural network. The PINN technique involves minimizing the difference between the predicted and actual values of the quantity of interest, as well as satisfying the differential equation and boundary conditions. This can be done using optimization algorithms such as stochastic gradient descent, sequential or adam [1]. Finally, after the neural network is trained, we can use it to predict the distribution of $u(x, y, t)$ function over time. We can then use data visualization techniques such as contour plots or surface plots to visualize the predicted distribution.

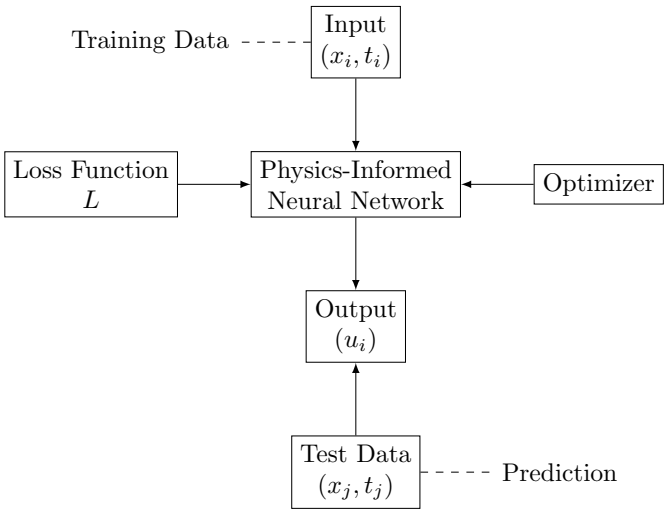

Figure 1: PINNs Architecture

During training, the PINN model should satisfy the 2D Diffusion equations as a constraint, and the generated data set should be used to compute the loss function. As a result the performance of the PINN model will be evaluated in predicting the diffusivity using various metrics, such as mean absolute error, root mean square error, and or the correlation coefficient.

2.1 Solution to the 2D Diffusion Equation

The 2D diffusion equation allows us to talk about the statistical movements of randomly moving particles in two dimensions. The movement of each individual particle does not follow the equation, but many identical particles each obeying the same boundary and initial conditions share some statistical properties. In this derivation of the diffusion PDE we well use function P instead of U to easily set apart the derivation function from the function used in the code section even though they mean the same. In an ideal world the diffusion function is just the probability distribution $P(x, y, t)$ which provides the probability of finding a perfectly average particle in the small vicinity of the point (x, y) at a given time t. The Brownian motion is a special case of diffusion 2D where the particles are subject to random forces that cause them to move in a random pattern [8]. The movement of particles in both diffusion 2D and Brownian motion is affected by the diffusion coefficient D, which represents the degree of randomness in the movement of the particles. The evolution of some systems does follow the equation outright but as a group they exhibit the smooth, well-behaved statistical features of the diffusion equation.

Now that with all the relative background knowledge, let's start with the 2D diffusion equation in cartesian coordinates as:

$$\begin{aligned} \nabla^2 - \frac{1}{D}\frac{\partial P}{\partial t} &= 0 \\ \frac{\partial^2 P}{\partial^2 x^2} + \frac{\partial^2 P}{\partial^2 y^2} - \frac{1}{D}\frac{\partial P}{\partial t} &= 0 \end{aligned} \tag{1}$$

Equation (1) is what we're interested in, but in order to solve it we will need to define the boundary conditions as follows:

$$\begin{aligned} \left.\frac{\partial^2 P}{\partial x^2}\right|_{x=\pm-\infty} &= \left.\frac{\partial^2 P}{\partial y^2}\right|_{y=\pm\infty} = 0 \\ \left.\frac{\partial P}{\partial x}\right|_{x=\pm\infty} &= \left.\frac{\partial P}{\partial y}\right|_{y=\pm\infty} = 0 \end{aligned} \tag{B.C}$$

Since the function P is a linear partial differential equation and separable function, we can apply an inverse 2D Fourier transform to solve the solution. Consider the following integral relations that define the 2D FT in Cartesian coordinates. We will call the function $\hat{P}$ the FT of our original function P:

$$\begin{aligned} \hat{P}(k_x, k_y, t) &= \iint_s e^{-i2\pi(k_x \cdot x + k_y \cdot y)} P(x, y, t) dx dy \\ P(x, y, t) &= \iint_s e^{-i2\pi(k_x \cdot x + k_y \cdot y)} \hat{P}(k_x, k_y, t) dx dy \end{aligned} \tag{2}$$

Notice the symmetry in going forward and backward in the transform Equation (2). This is because switching between the normal form of the problem and what we call Fourier Space, where the problem exists after the FT, are physically identical. Let's examine the spatial derivatives of the diffusion equation, where we consider the second derivative to be the function of interest. We can integrate these second derivatives by parts, using u and v as follows:

$$\int_a^b u dv = uv\Big|_a^b - \int_a^b v du$$

$$\text{if } u = e^{-i2\pi(k_x\cdot x+k_y\cdot y)} \text{ and } v = \frac{\partial P}{\partial x}$$
$$\text{then } du = \frac{\partial}{\partial x}e^{-i2\pi(k_x\cdot x+k_y\cdot y)} \text{ and } dv = \frac{\partial^2 P}{\partial x^2}dx$$

Using the substitution of integration by parts we can set up the whole integral as follows.

$$\begin{aligned}
\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial^2 P}{\partial x^2}dxdy &= e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial P}{\partial x}\Big|_{-\infty(x,y)}^{\infty} - \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{\partial}{\partial x}e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\cdot\frac{\partial P}{\partial x}dxdy \\
&= e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial P}{\partial x}\Big|_{-\infty(x,y)}^{\infty} + i2\pi k_x\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial P}{\partial x} \\
&= \cancel{e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial P}{\partial x}\Big|_{-\infty(x,y)}^{\infty}} + i2\pi k_x\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\cdot\frac{\partial P}{\partial x}dxdy \\
&= i2\pi k_x\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial P}{\partial x}
\end{aligned} \tag{3}$$

Looking the last term Equation (3), we effectively transferred one derivative off P and put it on the exponential of the FT, but since the exponential is an explicit function we can just perform the derivative, giving us the constant on the right most integral of the second line. We can apply similar inverse 1D Fourier transform Equation (3) again to get:

$$\begin{aligned}
\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial P}{\partial x}dxdy &= P\cdot e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\Big|_{-\infty(x,y)}^{\infty} - \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\frac{\partial}{\partial x}e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\cdot P\cdot dxdy \\
&= \cancel{P\cdot e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\Big|_{-\infty(x,y)}^{\infty}} + i2\pi k_x\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P\cdot e^{-i2\pi(k_x\cdot x+k_y\cdot y)}dxdy \\
&= i2\pi k_x\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}dxdy
\end{aligned} \tag{4}$$

Equating all the L.H.S with the R.H.S Equation (4) we get:

$$\begin{aligned}
\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial^2 P}{\partial x^2}dxdy &= (i2\pi k_x)^2\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}dxdy \\
\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-i2\pi(k_x\cdot x+k_y\cdot y)}\frac{\partial^2 P}{\partial x^2}dxdy &= (i2\pi k_x)^2\hat{P}
\end{aligned} \tag{5}$$

Since we took the spatial FT (i.e. dealing with x and y), keep in mind, the derivative in time does not change under a FT [9]. Henceforth we can write the 2D Equation (5) as follows, that enable us to use method of separation variables for an Eigen value problem in resulting Equation (6):

$$\begin{aligned}
\frac{\partial^n}{x^n} &= (i2\pi k_x)^n\hat{P} \\
(2\pi)^2\hat{P}\cdot((k_x)^2+(k_y)^2) + \frac{1}{D}\frac{\partial P}{\partial t} &= 0 \\
\hat{P} &= \lambda e^{-D(2\pi)^2(k_x^2+k_y^2)\cdot t}
\end{aligned} \tag{6}$$

The constant λ is nothing but the normalization factor that can be used for the conservation of linear momentum [8]. Therefore our original function p is then given by:

$$P = \lambda \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-D(2\pi)^2(k_x^2+k_y^2)t} \cdot e^{-i2\pi(k_x\cdot x + k_y\cdot y)} dk_x dk_y \tag{7}$$

We can further compute Equation (7) by method of separation of spatial variable as:

$$\begin{aligned} P &= \lambda \int_{-\infty}^{\infty} e^{i2\pi k_x\cdot x - D(2\pi k_x)^2 t} dk_x \cdot \int_{-\infty}^{\infty} e^{-i2\pi k_y\cdot y - D(2\pi k_y)^2 t} dk_y \\ P &= \lambda \left\{ \int_{-\infty}^{\infty} e^{i2\pi k_x\cdot x - D(2\pi k_x)^2 t} dk_x \right\} \cdot \left\{ \int_{-\infty}^{\infty} e^{-i2\pi k_y\cdot y - D(2\pi k_y)^2 t} dk_y \right\} \end{aligned} \tag{8}$$

Solving Equation (8) by separation of integral requires completing the square of the exponent, re-scaling the integration variable, changing to polar coordinates and then substituting back the Cartesian values. We will look at the integrand part of each individual section first.

$$P = \lambda \int_{-\infty}^{\infty} e^{\underbrace{i2\pi k_x\cdot x - D(2\pi k_x)^2 t}_{1}\, dk_x} \cdot \int_{-\infty}^{\infty} e^{\underbrace{i2\pi k_y\cdot y - D(2\pi k_y)^2 t}_{2}\, dk_y}$$

$$\begin{aligned} \underbrace{i2\pi k_x\cdot x - D(2\pi k_x)^2 t}_{\text{completing square for 1}} &= -4\pi^2 Dt\left(k_x - \frac{ix}{2\pi Dt}\right)^2 - \frac{x^2}{4Dt} \\ \text{Let } u_x &= k_x - \frac{ix}{2\pi Dt} \text{ then} \\ &= -4\pi^2 Dt u_x^2 - \frac{x^2}{4Dt} \\ \int_{-\infty}^{\infty} e^{\underbrace{i2\pi k_x\cdot x - D(2\pi k_x)^2 t}_{1}\, dk_x} &= e^{\frac{-x^2}{4Dt}} \int_{-\infty}^{\infty} e^{4\pi^2 Dt\cdot u_x^2 du_x} \end{aligned} \tag{9}$$

Similarly for the second section of the equation we have:

$$\begin{aligned} \underbrace{i2\pi k_y\cdot y - D(2\pi k_y)^2 t}_{\text{completing square for 2}} &= -4\pi^2 Dt\left(k_y - \frac{iy}{2\pi Dt}\right)^2 - \frac{y^2}{4Dt} \\ \text{Let } v_x &= k_y - \frac{iy}{2\pi Dt} \text{then} \\ &= -4\pi^2 Dt v_x^2 - \frac{y^2}{4Dt} \\ \int_{-\infty}^{\infty} e^{\underbrace{i2\pi k_y\cdot x - D(2\pi k_y)^2 t}_{2}\, dk_y} &= e^{\frac{-y^2}{4Dt}} \int_{-\infty}^{\infty} e^{4\pi^2 Dt\cdot v_x^2 dv_x} \end{aligned} \tag{10}$$

After dividing the equation in to two sections, we will attempt to solve each one individually beginning with the first section obtained in Equation (9).

Notice, Equation (9) is equivalent to the first integral in the original expression, up to a constant factor of $e^{-x^2/(4Dt)}$ with a non elemental integral on the integrand [10]. To solve this equation we will need to use substitution and then changing to polar coordinates.

Let

$$v = 2\pi\sqrt{Dt} \cdot u_x$$

then:

$$\int_{-\infty}^{\infty} e^{4\pi^2 Dt \cdot u_x^2} dU_x = \int_{-\infty}^{\infty} e^{\frac{-v^2}{4}} dv \tag{11}$$

The integral in Equation (11) can be evaluated using the standard result and method of re-scaling as:

$$\int_{-\infty}^{\infty} e^{-x^2/2} dx = \sqrt{2\pi}$$

To see why, we can consider the square of this integral and evaluate it in polar coordinates:

$$\left.\begin{aligned} x &= r\cos\theta \\ y &= r\sin\theta \\ x^2 + y^2 &= r^2 \\ dxdy &= rdrd\theta \end{aligned}\right\} \text{polar transformation}$$

$$\begin{aligned}
\left(\int_{-\infty}^{\infty} e^{-x^2/2} dx\right)^2 &= \int_{-\infty}^{\infty} e^{-x^2/2} dx \cdot \int_{-\infty}^{\infty} e^{-y^2/2} dy \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-x^2/2} \cdot e^{-y^2/2} dxdy \\
&= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-\frac{x^2+y^2}{2}} dxdy \\
&= \int_0^{2\pi}\int_0^{\infty} e^{-r^2/2} \cdot rdrd\theta \\
&= 2\pi \int_0^{\infty} e^{-r^2/2} rdr \\
&= -2\pi \left[e^{-r^2/2}\right]_0^{\infty} \\
&= \left(\sqrt{2\pi}\right)^2
\end{aligned} \tag{12}$$

Therefore from Equation (12) we have:

$$\begin{aligned}
\int_{-\infty}^{\infty} e^{-4\pi^2 Dtu_x^2} du &= \int_{-\infty}^{\infty} e^{-v^2/4} dv \\
&= 2\int_0^{\infty} e^{-v^2/4} dv \\
&= 2\sqrt{\pi}
\end{aligned}$$

Substituting v from Equation (11) back to in Equation (9) and multiplying both by $e^{-x^2/(4Dt)}$, we obtain:

$$e^{-\frac{x^2}{4Dt}} \int_{-\infty}^{\infty} e^{-4\pi^2 Dtu^2} du = \sqrt{\frac{\pi}{4Dt}}\, e^{-\frac{x^2}{4Dt}}$$

Now remember this is only for the Left (dk_x) section of Equation (9). We still need to follow the same process for the Right (dk_y) section in Equation (10). Following the same process we obtain Equation (15) as:

$$e^{-\frac{y^2}{4Dt}} \int_{-\infty}^{\infty} e^{-4\pi^2 Dtv^2} dv = \sqrt{\frac{\pi}{4Dt}}\, e^{-\frac{y^2}{4Dt}} \tag{15}$$

Finally putting all equations together and multiplying the two sections in Equation (8) we have:

$$\begin{aligned}
P(x,y,t) &= \lambda \frac{e^{\frac{-(r^2)}{4Dt}}}{4\pi D \cdot t} \\
P(x,y,t) &= \lambda \frac{e^{\frac{-(x^2+y^2)}{4D\cdot t}}}{4\pi Dt}
\end{aligned} \tag{16}$$

One last step is to figure out what λ exactly is in Equation (16). For this sample of project the normalization constant λ is just the sum unity function that represents the total sum of the diffusive particles across the 2D surface in the Gaussian distribution. So on larger scales the trainable degrees of freedom behave as Gaussian random variables and on somewhat smaller scales the dynamics is frustrated from the simultaneous maximization of entropy of non-trainable variables and minimization of entropy of trainable variables [2]. These results have some interesting implications for machine learning and physics. So the constant essentially tells us how many non-interacting particles we have in the system (we are considering just one representative particle). Thus, if we apply the boundary condition from $-\infty$ to $+\infty$, the total sum of the function $P(x,y,t)$ adds up to 1. Imposing this condition leads us to Equation (17):

$$\lambda = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} P(x,y,t) dxdy = 1 \tag{17}$$

There for, if $\lambda = 1$ the entirety of the 2D diffusion equation results to Equation (18):

$$\boxed{P(x,y,t) = \frac{e^{\frac{-(x^2+y^2)}{4D\cdot t}}}{4\pi Dt}} \tag{18}$$

A detailed complete PDE derivation of the solution to the diffusion 2D problem can be found here [11].

2.2 Code Setup

The code for this project tries to demonstrate how to use a neural network in order to solve thee diffusion equation. The

main diffusion equation as shown in Equation (18) is defined in the code by the function "diffusion_equation", which takes in the position, time, and diffusion coefficient as inputs and returns the value of the equation at that point as seen in the code below. Here the λ constant is set to 1 and the cartesian form of the function provided in Equation (17) is utilised. The domain and boundary conditions for the problem are defined by creating a grid of x and t values using the NumPy meshgrid function. The model is built using the Keras Sequential API and consists of three fully connected layers, with the output layer having a linear activation function. The model is trained using mean squared error loss and the RMSprop optimizer. Finally, the model is used to predict the diffusion equation for each value of D, and the results are plotted using the Matplotlib library. Full codes to the project can be found on github [12].

```
# This is the main Diffusion equation
def diffusion_equation(x, t, D):
  return 1 / (4 * np.pi * D * t) *
  np.exp(-np.linalg.norm(x) ** 2 / (4*D*t))

# Applying boundary conditions
x = np.linspace(-1, 1, 50)
t = np.linspace(0.01, 1, 20)
X, T = np.meshgrid(x, t)
X_flat = X.flatten()
T_flat = T.flatten()
D_values = [0.1, 0.2, 0.3, 0.4]
N = X_flat.shape[0]
```

Based on the code provided above, one can observe that the model is trained to predict the values of the diffusion equation at different points in space and time, for a range of diffusion coefficients. The training process involved minimizing the difference between the predicted values and the actual values of the diffusion equation using mean squared error loss.

After training, the model is used to predict the values of the diffusion equation for each value of the diffusion coefficient in the range of D_values. The predicted values are then plotted using contour plots, with the x-axis representing position, the y-axis representing time, and the color indicating the value of the diffusion equation.

As stated in the methods section, the homogeneous Neumann's boundary conditions are assumed (refer to the (B.C)). These boundary conditions state that the normal derivative of the solution at the boundary is zero. In this code, the boundaries of the domain are implicitly assumed to be reflective, which means that the diffusion equation at the boundaries is equal to zero. This assumption is necessary to avoid numerical errors due to the finite size of the domain.

3 Results

3.1 Varying Diffusion Coefficients

For the first set up, the neural network is defined using the Keras API, with an input layer of 2 neurons (corresponding to position and time), two hidden layers of 50 neurons each, and a linear output layer. The model is compiled with the mean squared error loss function and the RMSprop optimizer with a learning rate of 0.001. The model is then trained for 200 epochs on the shuffled training data, with a batch size of 256 and a validation split of 0.2. The training loss and validation loss are plotted as a function of the number of epochs. It showed good prediction with minimum loss function shown in Figure (2). This was then followed by a contour plot immediately to test the model.

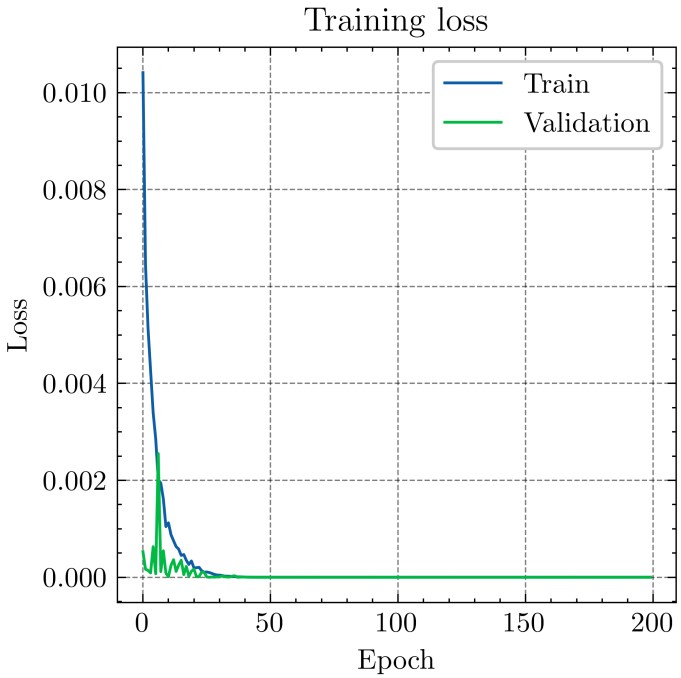

Figure 2: Model training and loss, based on the number of *epochs* = 200 and *batch_size* = 256. It can be observed that there's nearly no loss of data after the 20^{th} epoch

.

The four contour plots in Figure (3) show the predicted solution of the diffusion equation for different values of coefficients D ranging $0.1, 0.2, 0.3$ and 0.4. This is done by evaluating the trained neural network on the entire domain and plotting the results. From the contour plots we can arrive at the conclusion that the predicted values of the diffusion equation are consistent with the expected behavior of diffusion. For example, at early times, the diffusion is localized around the initial position, while at later times, it spreads out and becomes more diffuse. We also see that the diffusion is slower for lower values of the diffusion coefficient, which is consistent with our understanding of diffusion.

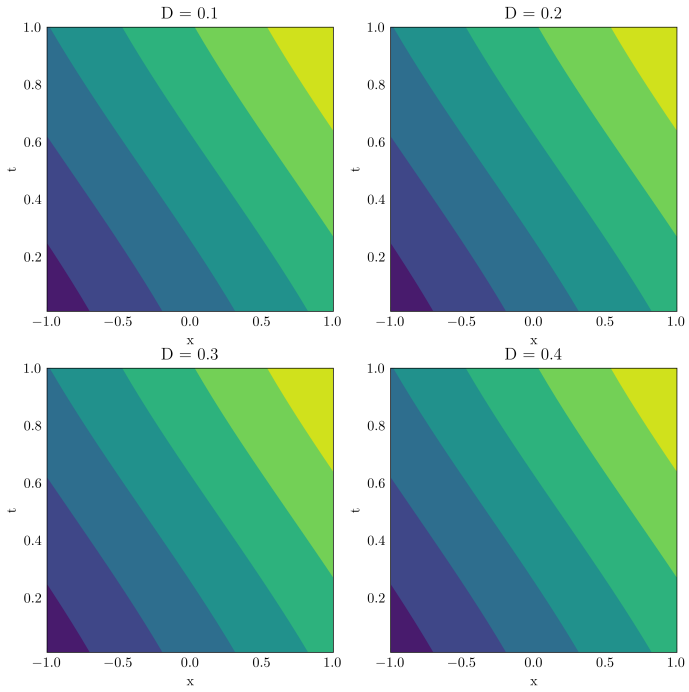

Figure 3: Prediction values for four $D_1 = 0.1$, $D_2 = 0.2$, $D_3 = 0.3$, and $D_4 = 0.4$ coefficients

3.2 Comparing Diffusion Across a Plate

Another section of the code simulates the diffusion of heat through a two-dimensional square plate of size 10 mm x 10 mm with thermal diffusivity of water. The set up is similar except diffusion of heat is governed by the heat equation, which describes how temperature changes over time in a given domain. The heat equation involves the second derivative of temperature with respect to both space and time. Thus, the plate has a circular region in the center with a high temperature and the rest of the plate is at a low temperature. Two graphs are generated using two different methods. The first section of the code (see Figure 4) uses the forward-difference method in time and central-difference method in space to numerically solve the heat diffusion equation. The boundary conditions are set as cold on the bottom and sides and hot on top. The code outputs four figures at different timesteps that show the temperature distribution in the plate.

The color scheme used in the maps represents temperature, with blue indicating the lowest temperature (273.15 K) and red indicating the highest temperature (373.15 K). The color scale is normalized to the initial and final temperatures of the plate, which are set to 273.15 K and 373.15 K, respectively. The temperature distribution on the plate can be observed by looking at the color distribution on the thermal maps.

Figure (5) on the other hand was generated by solving the heat diffusion equation using the Crank-Nicolson method to calculate the next timestep of the temperature distribution. The temperature distribution is initialized with the initial temperature and then evolved in time with the specified boundary conditions until a final time as adapted from Arocha [13]. The temperature distribution at specific time intervals is saved and plotted the matplotlib library.

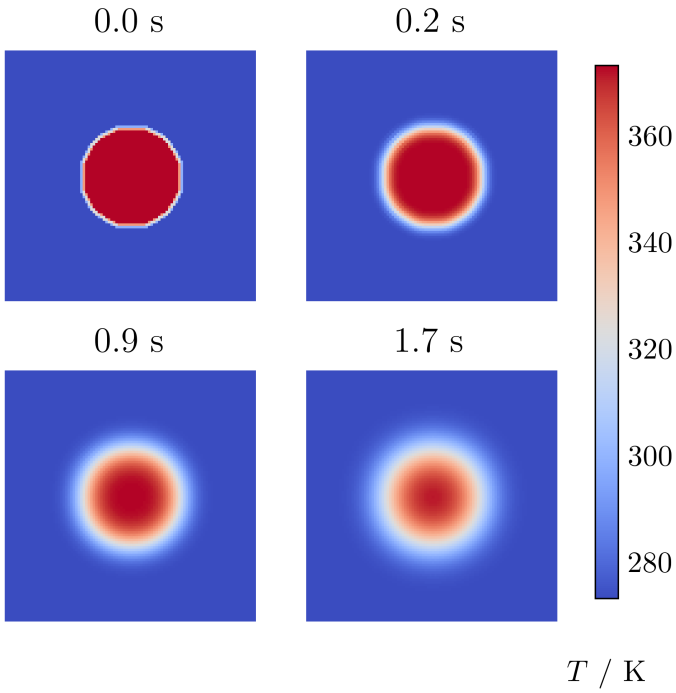

Figure 4: Diffusion rate of water using forward-difference method in time and central-difference method in space. Normalized color scale to the initial and final temperatures of the plate, which are set to 273.15 K and 373.15 K, respectively based on the boundary conditions.

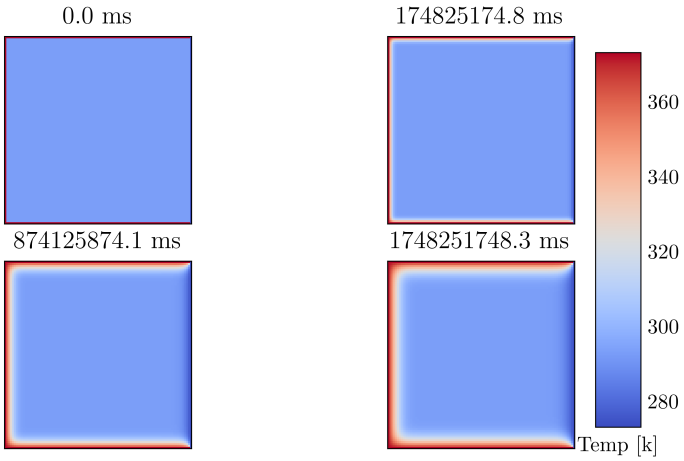

Figure 5: Diffusion rate of water using Crank-Nicolson method, adapted from Arocha, 2018 [13] for comparison. The temperature distribution is initialized first and then evolved in continuous time with the specified boundary conditions until a certain value of time vector.

Compared to the previous model, this one appears to be more specific to a particular physical system (heat diffu-

sion in a plate of water), whereas the previous model was a general machine learning model that could be applied to various problems. Additionally, this model uses numerical methods to solve the differential equations governing the system, whereas the previous model did not explicitly solve differential equations. Both models have their own strengths and weaknesses and are applicable in different contexts.

3.3 Using Finite Difference Method

In Figure (6) the code numerically solves a 2D diffusion equation with given boundary and initial conditions using the finite difference method. The final solution, represented by the array U, is then plotted using a heatmap with the matplotlib library. The resulting plot shows how the initial concentration profile evolves over time due to diffusion, with the red and blue colors representing high and low concentrations, respectively.

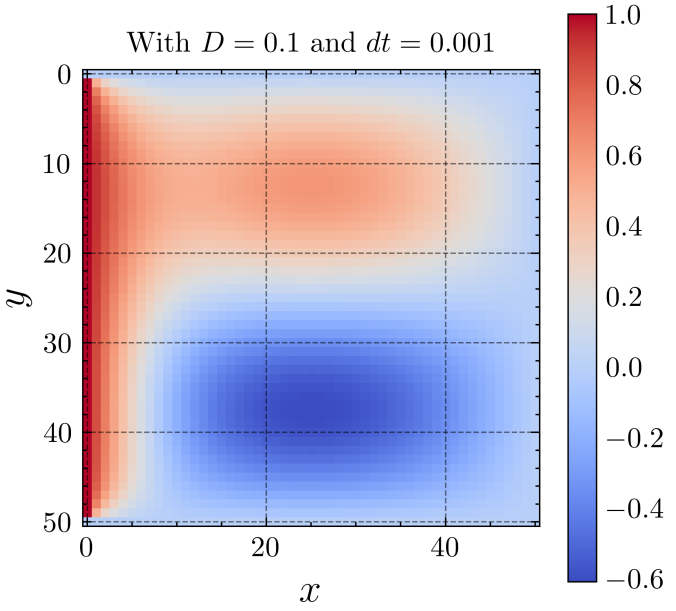

Figure 6: D = 0.1 and T=0.001

This brief python code simulates the diffusion of a substance in a 2D domain. It defines the domain size and grid resolution, the diffusion coefficient, time step, and boundary conditions. It then creates the grid and initializes the solution array with the initial condition and boundary conditions. The code then iteratively computes the diffusion for each time step using a finite difference method and plots the final solution as shown above.

3.4 Heat Flux and Temperature using Diffusion PiNN Model

This section solves the two-dimensional diffusion equation using the PINN model provided in the code. The model is trained to predict the temperature distribution in the domain, given the diffusion coefficient and boundary conditions. The diffusion coefficient is defined as a function of the position, and four different values are used for comparison. The PINN model is defined with several dense layers and the loss function is defined as the mean square error between the predicted and actual temperature distributions. The model is trained using the RMSprop optimizer and the training data is generated by concatenating the x and y coordinates. The x and $y$axes represent the spatial coordinates within the domain, and the colorbar on the right side of the plot indicates the temperature scale. Finally, the temperature distribution is plotted using for each diffusion coefficient.

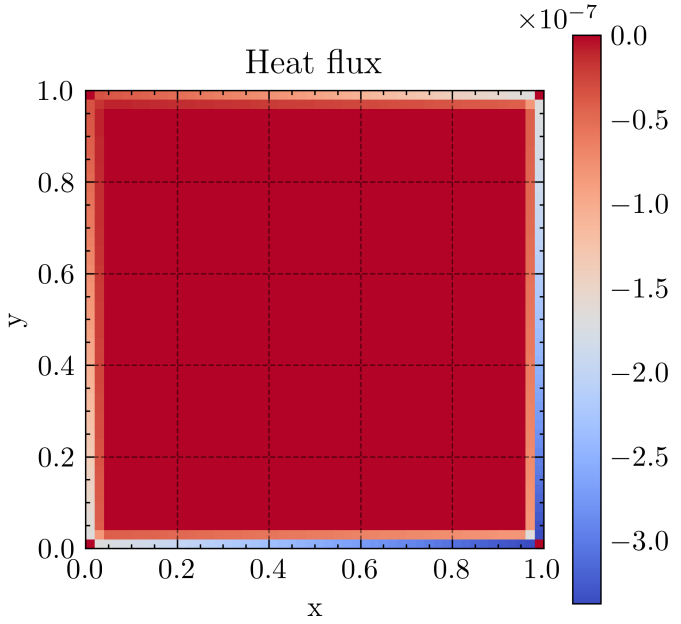

Figure 7: Heat Flux as the gradient of the temperature distribution. A positive value indicates heat flowing in the positive $+x$ or $+y$ direction, while a negative value indicates heat flowing in the negative $-x$ or $-y$ direction

The heat flux plot (Figure (7)) shows the rate of heat transfer per unit area in the system. In this case, the heat flux is calculated as the gradient of the temperature distribution. Positive values of the heat flux indicate heat flowing in the positive x or y direction, while negative values indicate heat flowing in the negative x or y direction. The heat flux plot can provide additional insights into the behavior of the system, such as identifying regions of high or low heat transfer and the direction of heat flow.

In a different section of the code, the Dirichlet boundary condition [6] was used was; where the temperature is specified on the boundaries of the domain. Specifically, the temperature is set to zero on the top, bottom, left, and right boundaries of the square domain. Figure (8) shows the temperature distribution across the various x and y values in the plate. The color of the plot represents the temperature distribution, where red corresponds to high temperatures and blue corresponds to low temperatures. And finally, the heat

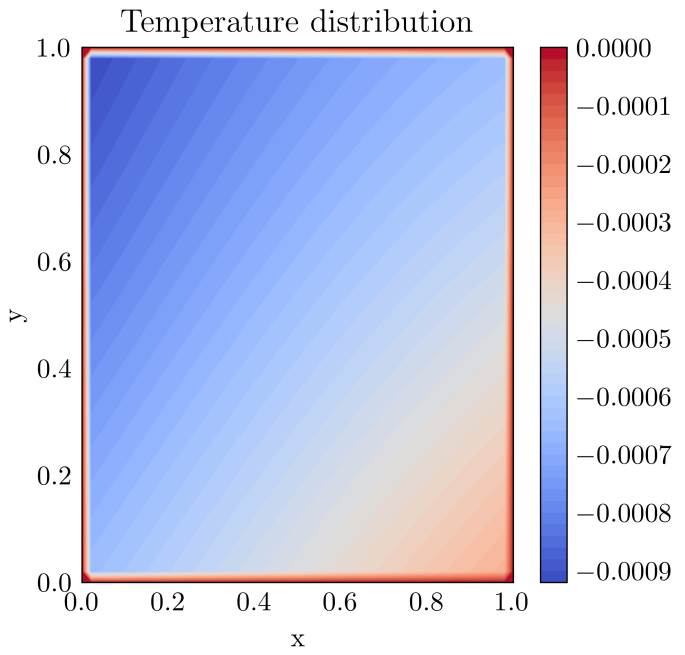

Figure 8: Temperature distribution, across the various x and y values in the plate. Red corresponds to high temperatures while blue corresponds to values at low temperatures

map plot in Figure (9) shows the circular cross-sectional temperature profile at time = 1 second, providing a color heat map of the similar scale.

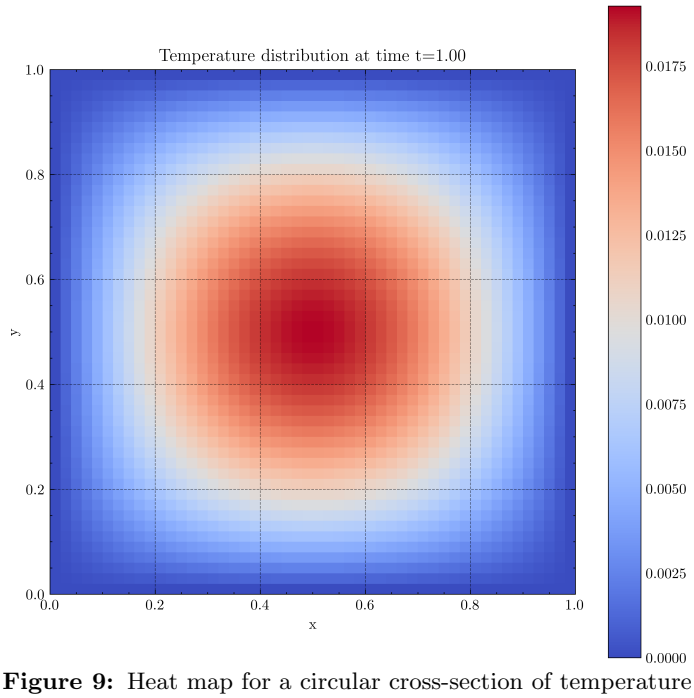

Figure 9: Heat map for a circular cross-section of temperature Profile at $Time = 1$ seconds

4 Discussions

Overall, the results suggest that the neural network is capable of accurately predicting the behavior of the diffusion equation for a range of diffusion coefficients, which could have applications in fields such as physics, chemistry, and engineering. However, it should be noted that the quality of the results may depend on factors such as the complexity of the system being modeled, the quality of the training data, and the architecture and hyperparameters of the neural network.

The predicted temperature distribution plots (seen in Figures 7, 8, and 9) show that the PINN model is able to accurately predict the temperature distribution for different values of the diffusion coefficient. The plots also indicate that the temperature distribution is symmetric along the x and y axes. The heat flux plot in figure (7), shows the flux of heat through the boundaries of the system. The plot indicates that there is a high heat flux at the corners of the system, which is expected due to the non-uniform diffusion coefficient.

Additionally, the contour plot (Figure 8) of the temperature distribution provides a more detailed visualization of the boundaries and gradients of the system. The contour plot shows that the temperature gradient is highest at the corners of the system, which again is expected due to the non-uniform diffusion coefficient.

Thus, all the plots and outputs suggest that the PINN model is able to accurately predict the temperature distribution and behavior of the system as well. Guaranteed, a further analysis and validation would be necessary to fully assess the performance of the model. Other techniques can also be used to determine the optimal parameters of a neural network using an iterative optimization. Such technique is called backpropagation [6]. The popular algorithm used for backpropagation is stochastic gradient descent, which is a stochastic version of the gradient descent algorithm. An important aspect of this procedure is the efficient computation of the gradient of the loss function using automatic differentiation. Another technique is also convolutional neural networks (CNNs). They are a type of deep neural network used for image and video recognition tasks. They use learnable filters to convolve over input data and extract relevant features. The architecture of a CNN typically includes convolutional, pooling, and fully connected layers [14]. CNNs have achieved state-of-the-art performance in image and video recognition tasks, and have also been used in other fields, such as natural language processing and speech recognition.

Some limitations of using the PINNs model include the need for a large amount of data to train the model, as well as the sensitivity of the model to the choice of hyperparameters such as the number of layers and neurons in each layer. In addition, the use of the neural network may lead to overfitting, which can result in poor generalization to new data [5]. The use of Green's functions in Physics-Informed Neural Networks (PINNs) has several limitations. One of the main limitations is that the analytical expression for

the Green's function may not be available for more complex problems. In such cases, numerical or approximate methods may be required to obtain the Green's function, which can be time-consuming and may result in inaccuracies. Another limitation is that the use of Green's functions assumes that the underlying physics of the problem is linear and time-invariant [3]. However, many real-world problems involve non-linear and time-varying physics, which may not be accurately captured by Green's functions.

Furthermore, the PINNs model assumes that the underlying physics is continuous and differentiable, which may not be the case in all scenarios. The implementation of this approach requires knowledge of neural networks, particularly Physics-Informed Neural Networks, and their training. The PiNN model should be trained with the same boundary conditions and initial conditions as the original code. It is essential to select an appropriate architecture for the neural network and perform hyperparameter tuning to optimize the model's performance. Finally, the training of the PINNs model can be computationally expensive, especially for high-dimensional problems, which may limit its practicality in certain applications.

5 Conclusion

The application of Physics informed neural network provides a powerful tool for determining the solution of the 2D diffusion equation. In this study, a PINN model is presented, which is a type of neural network capable of solving partial differential equations (PDEs) with high accuracy, using both supervised and unsupervised learning. The full 2D diffusion partial differential equation is solved using the model, which is followed by a discussion of the training loss function, a measure of how well the model can approximate the solution to the PDE. The model considered two different diffusion coefficients, and a comparison between the trained and predicted data with test data is carried out using contour plots and normal graphs. Despite the challenges encountered, the process of building the model highlights the potential of PINNs to solve complex physical problems, particularly in fields such as fluid dynamics and data science.

An example code utilizing the PINNs model to solve a diffusion equation generates a heatmap to visualize the solution. The results indicate that the PINNs model can accurately approximate the solution to the PDE, with low training loss and high accuracy. Overall, the discussion emphasizes the effectiveness of the PINNs model in solving PDEs, and the significance of the training loss function in measuring the accuracy of the model. The example code provides a clear demonstration of the power and versatility of this approach in solving complex problems in physics and engineering. For further details, a detailed derivation of the 2D equation can be found in the referenced source [11]. Access to the codes and all necessary documentation accompanying this project for future updates is also made accessible through github [12].

List of Figures

References

[1] F. Fernández de la Mata, A. Gijón, M. Molina-Solana, and J. Gómez-Romero, "Physics-informed neural networks for data-driven simulation: Advantages, limitations, and opportunities," *Physica A: Statistical Mechanics and its Applications*, vol. 610, p. 128 415, 2023, ISSN: 03784371. DOI: 10.1016/j.physa.2022.128415. [Online]. Available: https://doi.org/10.1016/j.physa.2022.128415.

[2] M. I. Katsnelson, V. Vanchurin, and T. Westerhout, "Emergent scale invariance in neural networks," *Physica A: Statistical Mechanics and its Applications*, vol. 610, p. 128 401, 2023, ISSN: 03784371. DOI: 10.1016/j.physa.2022.128401. [Online]. Available: https://doi.org/10.1016/j.physa.2022.128401.

[3] D. Skinner, "Green's functions for PDEs," pp. 126–142, [Online]. Available: http://www.damtp.cam.ac.uk/user/dbs26/1BMethods/GreensPDE.pdf.

[4] S. Nair, *Advanced Topics in Applied Mathematics This*. Cambridge University Press, 2011, ISBN: 9788527729833. [Online]. Available: www.cambridge.org/9781107006201.

[5] A. A. Hassan, "Green's Function for the Heat Equation," *Fluid Mechanics: Open Access*, vol. 04, no. 02, pp. 2–7, 2017. DOI: 10.4172/2476-2296.1000152.

[6] N. Sukumar and A. Srivastava, "Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks," *Computer Methods in Applied Mechanics and Engineering*, vol. 389, p. 114 333, 2022, ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2021.114333. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045782521006186.

[7] J. R. Willis, "Polarization approach to the scattering of elastic waves-I. Scattering by a single inclusion," *Journal of the Mechanics and Physics of Solids*, vol. 28, no. 5-6, pp. 287–305, 1980, ISSN: 00225096. DOI: 10.1016/0022-5096(80)90021-6.

[8] T. Ursell, "APh Physics Laboratory," *Physics*, 2005. [Online]. Available: http://www.physics.nyu.edu/grierlab/methods/node11.html.

[9] J. Tang, V. C. Azevedo, G. Cordonnier, and B. Solenthaler, "Neural Green's function for Laplacian systems," *Computers Graphics*, vol. 107, pp. 186–196, 2022, ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2022.07.016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849322001406.

[10] B. Conrad, "Impossibility theorems for elementary integration," *University of Michigan*, 2005. [Online]. Available: https://www.claymath.org/library/academy/LectureNotes05/Conrad.pdf.

[11] G. Eyob, "Full 2d-difussion equation derivation.pdf," *MMAE-502*, pp. 1–16, 2023. [Online]. Available: https://github.com/eyobghiday/PINNs-in-Fluid-Mechanics/blob/main/Eyob_Ghiday_Difussion_Derivation.pdf.

[12] G. Eyob, *Application of PiNNs in 2D Difussion Equation*, 2023. [Online]. Available: https://github.com/eyobghiday/PINNs-in-Fluid-Mechanics.

[13] M. A. Arocha, "Crank nicolson method," 2018. [Online]. Available: https://matlabgeeks.weebly.com/uploads/8/0/4/8/8048228/crank_nicolson_method_presentation-v5.pdf.

[14] D. Finol, Y. Lu, V. Mahadevan, and A. Srivastava, "Deep convolutional neural networks for eigenvalue problems in mechanics," *International Journal for Numerical Methods in Engineering*, vol. 118, no. 5, pp. 258–275, 2019. DOI: https://doi.org/10.1002/nme.6012. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.6012.