1D WAVE EQUATION USING PHYSICS-INFORMED NEURAL NETWORKS (PINNS)

1. Load the Required Packages

```julia
using NeuralPDE, Lux, Optimization, OptimizationOptimJL
using ModelingToolkit: Interval
```

- **NeuralPDE.jl**: Main package to define and solve PDEs using neural networks.
- **Lux.jl**: Lightweight neural network library.
- **Optimization.jl**: General interface for optimization problems.
- **OptimizationOptimJL.jl**: Optimization backend using classic algorithms like (Broyden–Fletcher–Goldfarb–Shanno) BFGS.
- **ModelingToolkit.Interval**: For specifying the spatial and temporal domains.

2. Define the PDE and Derivatives

```julia
@parameters t, x
@variables u(..)
Dxx = Differential(x)^2
Dtt = Differential(t)^2
Dt = Differential(t)
```

- `@parameters t, x`: Declare time and space as symbolic parameters.
- `@variables u(..)`: Define `u(t, x)` as the unknown solution.
- `Dxx`, `Dtt`: Symbolic second derivatives

$$\frac{\partial^2 u}{\partial x^2} \text{ and } \frac{\partial^2 u}{\partial t^2}$$

- `Dt`: First derivative in time (used for initial velocity condition).

3. Define the 1D Wave Equation

```julia
# PDE
C = 1
eq = Dtt(u(t, x)) ~ C^2 * Dxx(u(t, x))
```

This represents the **1D wave equation**:

$$\frac{\partial^2 u(x,t)}{\partial t^2} = c^2 \times \frac{\partial^2 u(x,t)}{\partial x^2}$$

Where C=1 is the speed of the wave propagation.

4. Initial and Boundary Conditions

```
# Dirichlet initial and boundary conditions
bcs = [u(t, 0) ~ 0.0, # for all t > 0 Dirichlet BC at x = 0
    u(t, 1) ~ 0.0, # for all t > 0  Dirichlet BC at x = 1
    u(0, x) ~ x * (1.0 - x), #for all 0 < x < 1 Initial displacement
    Dt(u(0, x)) ~ 0.0] #for all  0 < x < 1] initial velocity
```

- **Boundary Conditions**: Fixed ends at x=0,and x= 1 with zero displacement.
- **Initial Condition**: A parabola-shaped displacement profile.
- **Initial Velocity**: Zero (i.e., wave starts at rest).

5. Define Domain

```
# Space and time domains
domains = [t ∈ Interval(0.0, 1.0),
    x ∈ Interval(0.0, 1.0)]
```

This snip defines the time (t) and space (x) domain: t∈[0,1], x∈[0,1]

6. Neural Network (PINN) Setup

```
# Discretization
dx = 0.1

# Neural network
chain = Chain(Dense(2, 16, σ), Dense(16, 16, σ), Dense(16, 1))
discretization = PhysicsInformedNN(chain, GridTraining(dx))
```

- **`dx = 0.1`**: Grid resolution for training points as specified
- **`Chain`**: Neural network with:
  - Input layer: 2 inputs (`t`, `x`)
  - Hidden layers: 2 layers of size 16 with `σ` (sigmoid) activation
  - Output layer: 1 neuron (predicting `u`)

- `PhysicsInformedNN()`: Combines the network with knowledge of physics.
- `GridTraining(dx)`: Uniform training grid.

7. Define the PDE System and Discretize

```
@named pde_system = PDESystem(eq, bcs, domains, [t, x], [u(t, x)])
prob = discretize(pde_system, discretization)
```

- The macro @named assigns a name to the PDESystem, which helps when tracking and working with components internally (like optimization, solving, and differentiating). It sets pde_system.name = :pde_system
- ‘prob’ constructs a loss function based on the residuals of the PDE, the boundary and initial conditions.
- It creates a training dataset from the domain using the grid defined by `dx`.
- Then it prepares the **optimization problem** (i.e., tuning the neural network weights to minimize the PDE residual loss).

8. Training with Optimizer

```
callback = function (p, l)
    println("Current loss is: $l")
    return false
end

# optimizer
opt = OptimizationOptimJL.BFGS()
res = Optimization.solve(prob, opt; callback, maxiters = 250)
phi = discretization.phi
```

- `callback`: Logs the loss at every iteration.
- `BFGS`: Gradient-based optimizer for training.
- `phi`: Trained neural network function that approximates `u(t,x)`.

9. Plot Analytic vs PINN Solutions

```
#PLOTS OF THE RESULTS
using Plots
ts, xs = [infimum(d.domain):dx:supremum(d.domain) for d in domains]
function analytic_sol_func(t, x)
    sum([(8 / (k^3 * pi^3)) * sin(k * pi * x) * cos(C * k * pi * t) for k
in 1:2:50000])
end
```

- `ts`, `xs`: Time and space grid for evaluation.
- `infimum(d.domain)`: Gets the lower bound of the domain (0)
- `supremum(d.domain)`: Gets the upper bound of the domain (1)
- `analytic_sol_func`: Fourier series solution to the wave equation.

$$analytic\_sol\_func(t,x) = \sum_{k=1,3,5...}^{50000} (\frac{8}{k^3\pi^3}) \sin(k\pi x)\cos(Ck\pi t).$$

- `For k in 1:2:50000`: Uses only odd numbers (1, 3, 5, ...) for the Fourier terms. Usually, fourier series converges slowly so higher the terms more closer to the true value, however we cannot do infinitely.

```
u_predict = reshape([first(phi([t, x], res.u)) for t in ts for x in xs],
    (length(ts), length(xs)))
u_real = reshape([analytic_sol_func(t, x) for t in ts for x in xs],
    (length(ts), length(xs)))
```

- Evaluate the PINN and analytic solution on the grid.
- Compute the absolute error.

```
p1 = plot(ts, xs, u_real, linetype = :contourf, title = "analytic");
p2 = plot(ts, xs, u_predict, linetype = :contourf, title = "predict");
p3 = plot(ts, xs, diff_u, linetype = :contourf, title = "error");
plot(p1, p2, p3)
```

- Contour plots of true solution, predicted solution, and error.

10. Save Plots and Results

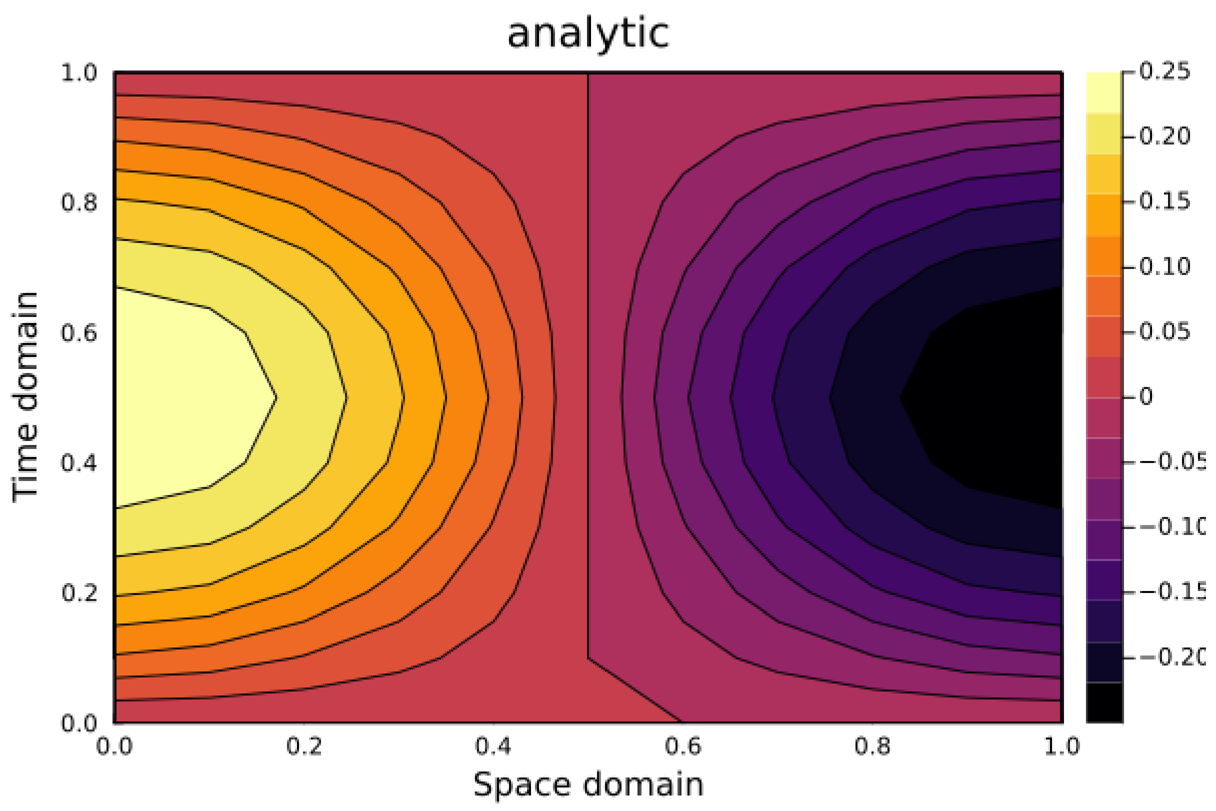
```
savefig(p1, "analytic.png")
savefig(p2, "predict.png")
savefig(p3, "error.png")
```

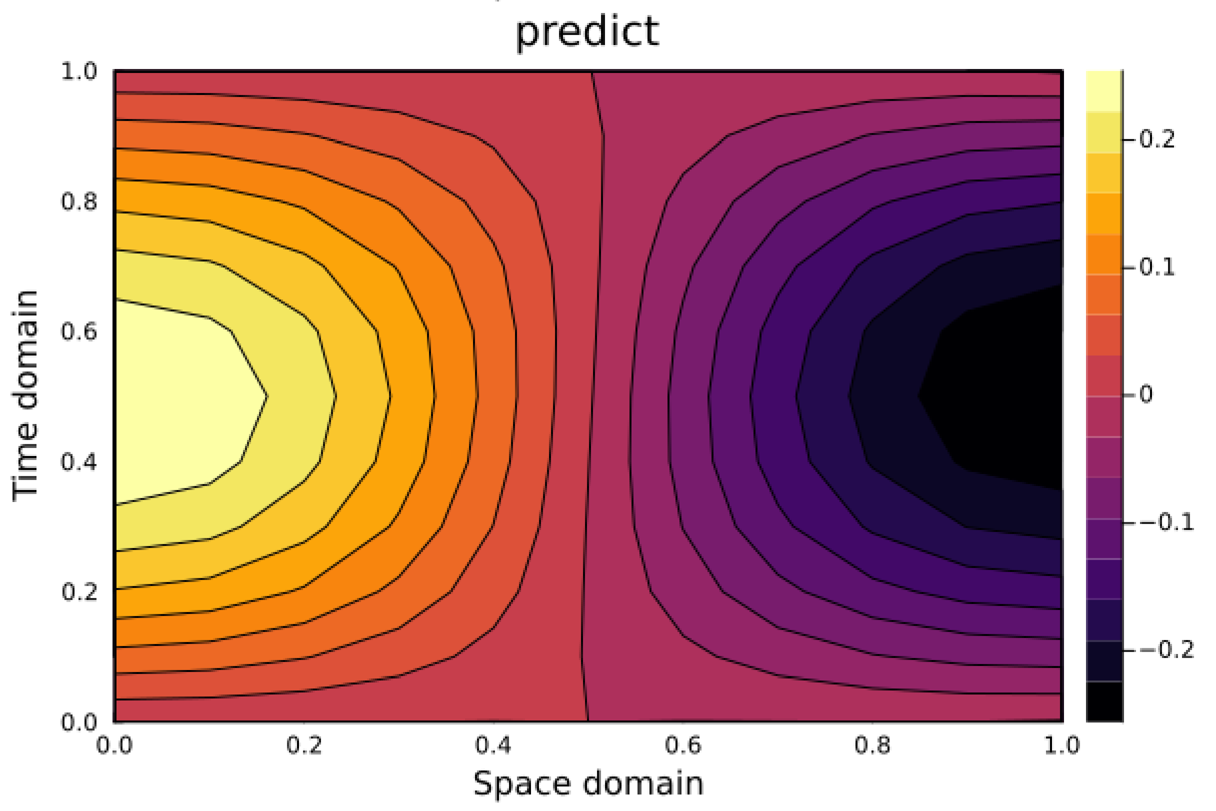
- Saves the plots as PNG files.

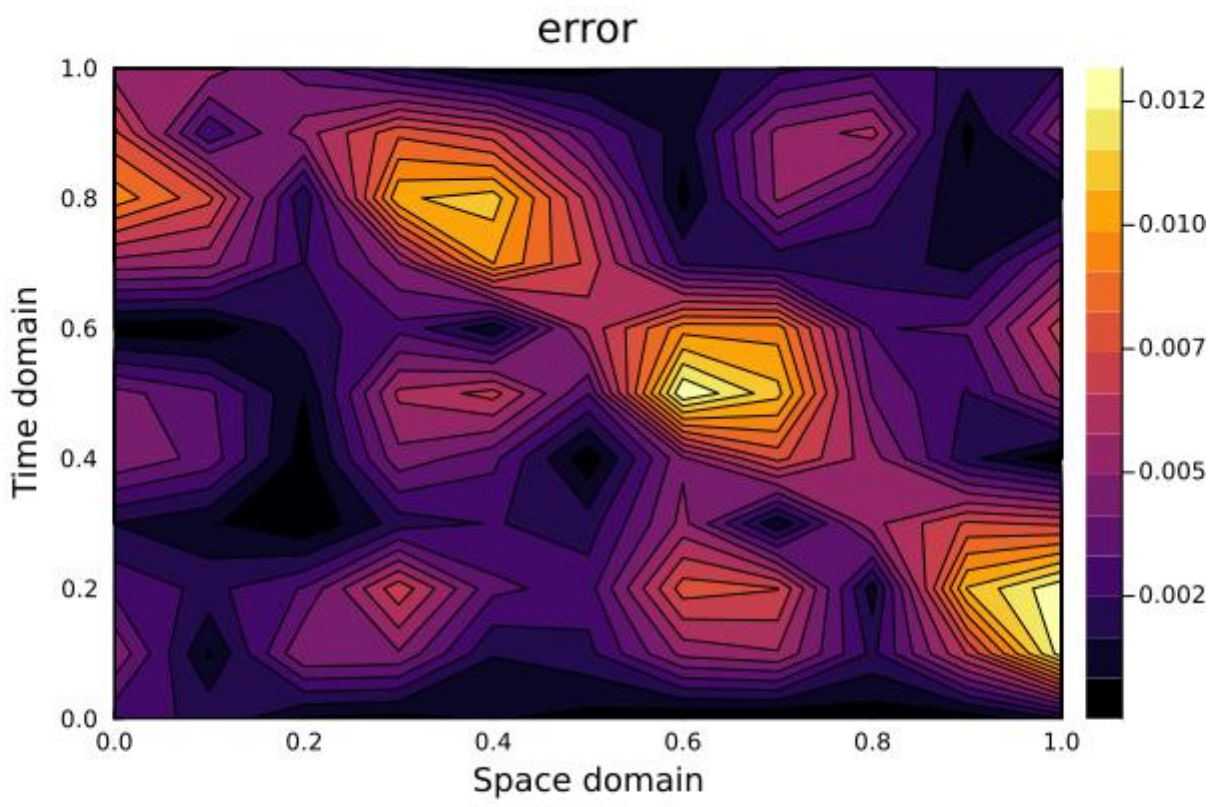
```
#save the results in a csv file
using CSV
using DataFrames

#convert the analytic solution into list and save the analytic solution in
csv file
df_analytic= DataFrame(t = repeat(ts, inner = length(xs)), x = repeat(xs,
outer = length(ts)), u_real = vec(u_real))
CSV.write("analytic.csv", df_analytic)
#convert the prediction into list and save the predicted results in csv
file
df_predict= DataFrame(t = repeat(ts, inner = length(xs)), x = repeat(xs,
outer = length(ts)), u_predict = vec(u_predict))
CSV.write("predict.csv", df_predict)
```

- Saves all values to CSV for external analysis.

Results and Discussion







X-axis is the space domain, and Y-axis is the time domain (0 to 1)

The absolute error is 0.0125 maximum, which is very low. The highest error appears in the localized region may be due to sharp changes which was difficult to capture.

Overall, the PINN model predicts the analytical values well. Further it can be finetuned by increasing the iterations, deepening the network, use multi-level refinements like Adam+ BFGS, and modifying the learning rate.