Objective:

Solve the 2D wave equation using PINNs.

Workflow of the Code:

The code solves the 2D wave equation using a Physics-Informed Neural Network (PINN).

1. Initialization and Imports

- Import required libraries (torch, numpy, matplotlib, etc.).
- Configure the computational device (CPU/GPU).
- Set seeds for reproducibility across runs.

2. Define the Wave Equation

- Solve the 2D wave equation: $\partial^2 T/\partial t^2 - c^2(\partial^2 T/\partial x^2 + \partial^2 T/\partial y^2) = 0$

3. Apply Boundary and Initial Conditions

- Boundary Conditions:

At $x=0$: $\partial T/\partial x - c\, \partial T/\partial t = 0$

At $x=L$: $\partial T/\partial x + c\, \partial T/\partial t = 0$

At $y=0$: $\partial T/\partial y + c\, \partial T/\partial t = 0$

At $y=D$: $\partial T/\partial y - c\, \partial T/\partial t = 0$

- Initial Conditions

At $t=0$: $T(x, y, 0) = 0$ and $\partial T/\partial t(x, y, 0) = 0$

4. Center Point Ripple

- Introduce a sinusoidal disturbance at the center of the domain

$T(x, y, t) = \sin(\pi x/L) \sin(\pi y/D) \cos(\omega t)$

5. Loss Function Components:

1. Wave Equation Residual:

Wave Loss = $\partial^2 T/\partial t^2 - c^2(\partial^2 T/\partial x^2 + \partial^2 T/\partial y^2)$

2. Boundary Condition Loss:

Boundary Loss = MSE(Boundary Condition 1) + MSE(Boundary Condition 2)

3. Initial Condition Loss:

Initial Condition Loss = MSE(Initial Condition)

4. Center Point Ripple Loss:

Center Ripple Loss = MSE(Ripple at center)

6. Neural Network Model

For a feed-forward neural network with 5 hidden layers, using the tanh activation function and Xavier initialization:

```
# Model definition
class PINN(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(PINN, self).__init__()
        self.fc1 = nn.Linear(input_dim, 50)
        self.fc2 = nn.Linear(50, 50)
        self.fc3 = nn.Linear(50, 50)
        self.fc4 = nn.Linear(50, 50)
        self.fc5 = nn.Linear(50, 50)
        self.fc6 = nn.Linear(50, output_dim)

        # Xavier initialization
        torch.nn.init.xavier_normal_(self.fc1.weight)
        torch.nn.init.xavier_normal_(self.fc2.weight)
        torch.nn.init.xavier_normal_(self.fc3.weight)
        torch.nn.init.xavier_normal_(self.fc4.weight)
        torch.nn.init.xavier_normal_(self.fc5.weight)
        torch.nn.init.xavier_normal_(self.fc6.weight)

    def forward(self, x):
        x = torch.tanh(self.fc1(x))
        x = torch.tanh(self.fc2(x))
        x = torch.tanh(self.fc3(x))
        x = torch.tanh(self.fc4(x))
        x = torch.tanh(self.fc5(x))
        x = self.fc6(x)
        return x
```

7. Adam Optimizer

The Adam optimiser with a learning rate of 1×10 ^−4 is used for training:

```
# Adam optimizer setup
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
```

8. Training Loop

```
# Training loop
for epoch in range(20001):
    optimizer.zero_grad()
    loss_value = loss_fn(train_points)
    loss_value.backward()
    optimizer.step()
    if epoch % 100 == 0:
        print(f"{epoch} {loss_value.item()}")
```

9. Visualization:

Generate a scatter plot to visualize the predicted wave field T(x,y,t):

```
# Visualization code
test_points_t = torch.tensor(test_points_t, dtype=torch.float32).to(device)
predict = model(test_points_t).detach().cpu().numpy()

# Plotting
plt.scatter(x_te, y_te, c=predict[:, 0], cmap='jet', s=1, edgecolor='none', alpha=1)
plt.colorbar(orientation='vertical')
plt.show()
```