Assignment_7

April 13, 2023

1 Imports

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn
import os
from scipy.io import loadmat

plt.rcParams['figure.figsize'] = (10.0, 7.0)
plt.rcParams["font.size"] = 16
plt.rcParams["font.family"] = "Serif"
plt.rcParams["grid.linestyle"] = "--"
plt.rcParams["grid.linewidth"] = 0.5
```

[ ]:
```
DATA_DIR = 'data'
PLOTS_DIR = 'plots'
```

2 Problem Statement

The goal is to solve the 2D boundary value problem of linear elasiticity using neural networks. The PDE is defined as follows:

$$G\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right]+G\left(\frac{1+v}{1-v}\right)\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 v}{\partial yx}\right]+\sin(2\pi x)\sin(2\pi y)=0$$
$$G\left[\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right]+G\left(\frac{1+v}{1-v}\right)\left[\frac{\partial^2 v}{\partial y^2}+\frac{\partial^2 u}{\partial xy}\right]+\sin(\pi x)+\sin(2\pi y)=0$$

with boundary conditions \$ u, v = 0\$ and

$$G=\frac{E}{2(1+\nu)}$$

3 Solution

3.1 The Neural Network

We will start by defining the neural network. The network will be a simple feed forward network with 5 hidden layers and 30 neurons each layer. The input to the network will be the coordinates

(x, y) and the output will be the displacement u and v. `tanh` activation function will be used for the hidden layers and `linear` activation function will be used for the output layer.

[ ]:
```python
class Displacements(nn.Module):
    def __init__(self, ns=30):
        super(Displacements, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(2, ns),
            nn.Tanh(),
            nn.Linear(ns, ns),
            nn.Tanh(),
            nn.Linear(ns, ns),
            nn.Tanh(),
            nn.Linear(ns, ns),
            nn.Tanh(),
            nn.Linear(ns, ns),
            nn.Tanh(),
            nn.Linear(ns, 2),
        )

    def forward(self, x):
        return self.net(x)

model = Displacements(ns=30)
print(model)
```

```
Displacements(
  (net): Sequential(
    (0): Linear(in_features=2, out_features=30, bias=True)
    (1): Tanh()
    (2): Linear(in_features=30, out_features=30, bias=True)
    (3): Tanh()
    (4): Linear(in_features=30, out_features=30, bias=True)
    (5): Tanh()
    (6): Linear(in_features=30, out_features=30, bias=True)
    (7): Tanh()
    (8): Linear(in_features=30, out_features=30, bias=True)
    (9): Tanh()
    (10): Linear(in_features=30, out_features=2, bias=True)
  )
)
```

This gives us the required network architecture. Next, we load the boundary and interior points.

3.2 Loading the data

[ ]:
```
boundary_points = loadmat(os.path.join(DATA_DIR, 'boundary_points.mat'))
x_boundary = boundary_points['x_bdry']
y_boundary = boundary_points['y_bdry']
assert len(x_boundary) == len(y_boundary), 'x and y boundary points must have␣
↪the same length'
BOUNDARY_POINTS = len(x_boundary)
print(f'Number of boundary points: {BOUNDARY_POINTS}')
```

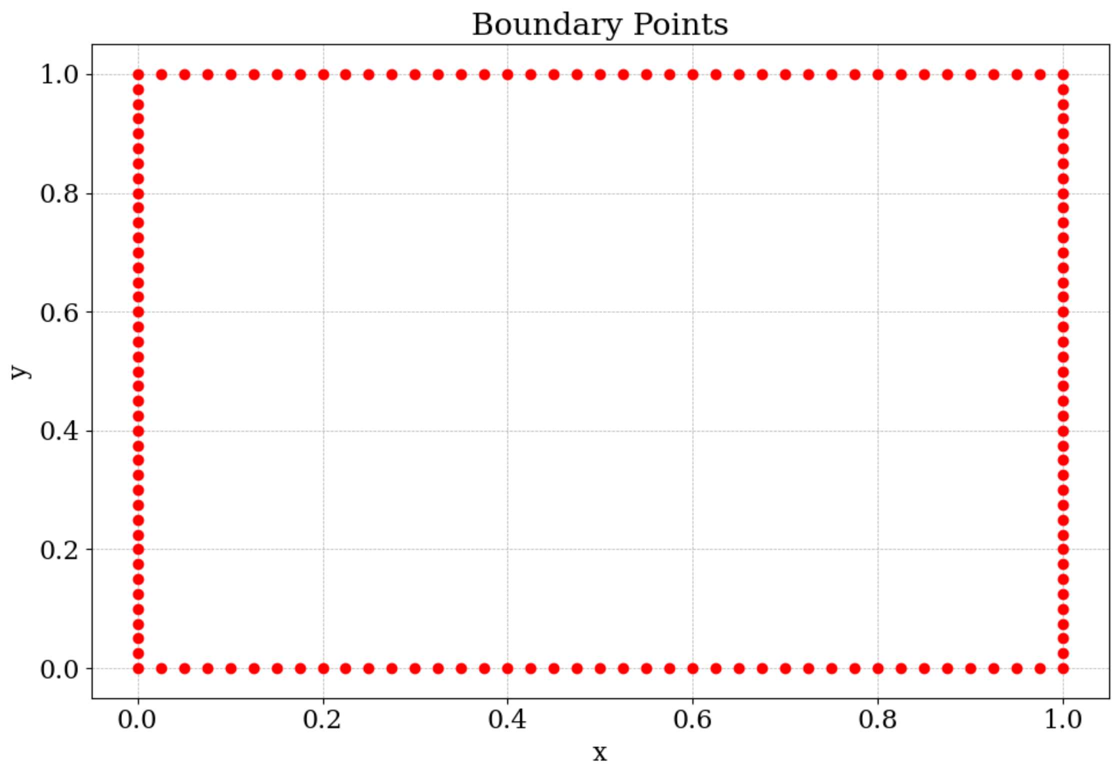
```
Number of boundary points: 160
```

[ ]:
```
interior_points = loadmat(os.path.join(DATA_DIR, 'interior_points.mat'))
x_interior = interior_points['x']
y_interior = interior_points['y']
assert len(x_interior) == len(y_interior), 'x and y interior points must have␣
↪the same length'
INTERIOR_POINTS = len(x_interior)
print(f'Number of interior points: {INTERIOR_POINTS}')
```

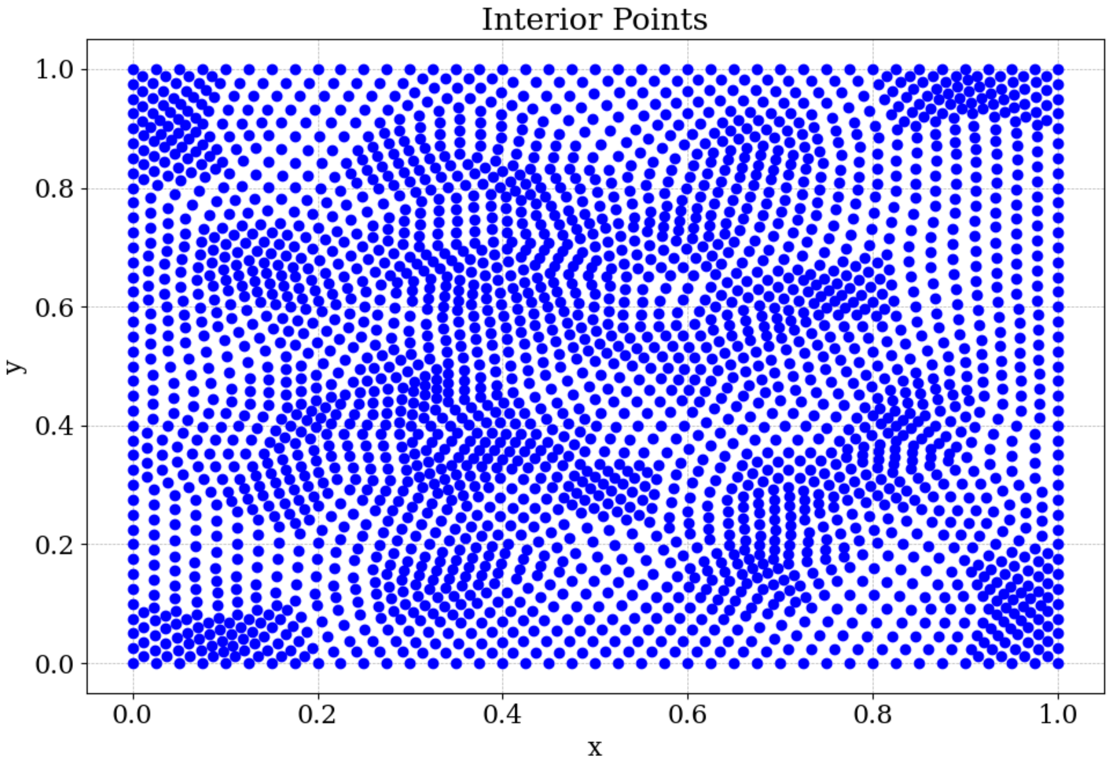
```
Number of interior points: 2705
```

Let's have a look at how the data looks like.

[ ]:
```
plt.plot(x_boundary, y_boundary, 'or')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Boundary Points')
plt.grid()
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, '0101.png'))
plt.show()
```



[ ]:
```
plt.plot(x_interior, y_interior, 'ob')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Interior Points')
plt.grid()
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, '0102.png'))
plt.show()
```



We'll need to concatenate the boundary and interior points to get the complete data. As the boundary condition is $u, v = 0$, we will also create a target array with all zeros for the boundary points.

```
[ ]: X_boundary = torch.tensor(np.concatenate((x_boundary, y_boundary), axis=1),␣
      ↪dtype=torch.float32, requires_grad=True)
     X_interior = torch.tensor(np.concatenate((x_interior, y_interior), axis=1),␣
      ↪dtype=torch.float32, requires_grad=True)

     X = torch.cat((X_boundary, X_interior), dim=0)
     u_boundary = torch.tensor(np.zeros((BOUNDARY_POINTS, 1)), dtype=torch.float32,␣
      ↪requires_grad=True)
     v_boundary = torch.tensor(np.zeros((BOUNDARY_POINTS, 1)), dtype=torch.float32,␣
      ↪requires_grad=True)
     U_boundary = torch.cat((u_boundary, v_boundary), dim=1)
     print(X.shape, U_boundary.shape)
```

```
torch.Size([2865, 2]) torch.Size([160, 2])
```

This makes the data ready for training. Next, we'll create the loss function.

3.3 Loss Function

The loss function is made up of two parts: 1. The PDE loss 2. The boundary loss

3.3.1 Boundary Loss

The boundary loss will be a simple RMSE loss. Here is the code for the bounadry loss.

[ ]:
```
def boundary_loss(U_pred_b, U_b, regularization = 1):
    """Calculate the loss for the boundary points."""
    return regularization*torch.mean((U_pred_b - U_b)**2)
```

3.3.2 PDE Loss

The PDE loss is complicated. We first need to determine the derivatives of u and v with respect to x and y. Let's see how we can do that. But first, let's define G. which is used in the PDE:

[ ]:
```
E = 1.0
nu = 0.3
G = E / (2 * (1 + nu))
```

Here, we will define the pde loss which is given by equation 1 in the problem statement.

[ ]:
```
def pde_loss(X_i, model):
    """Calculate the loss for the PDE.

    Parameters
    ----------
    X_i : torch.Tensor
        The interior points.
    model : torch.nn.Module
        The model. It predicts the displacements for the interior points.

    Returns
    -------
    torch.Tensor
        The loss.
    """
    #extract x, y, u and v
    x, y = X_i[:, 0], X_i[:, 1]
    U_i = model(X_i)
    u = U_i[:, 0]
    v = U_i[:, 1]

    #Calculate the derivatives
    dudx, dudy = torch.autograd.grad(u.sum(), X_i, create_graph=True,␣
↪retain_graph=True)[0].T
    dvdx, dvdy = torch.autograd.grad(v.sum(), X_i, create_graph=True,␣
↪retain_graph=True)[0].T

    du2dx2, du2dxdy = torch.autograd.grad(dudx.sum(), X_i, create_graph=True,␣
↪retain_graph=True)[0].T
```

```
    du2dydx, du2dy2 = torch.autograd.grad(dudy.sum(), X_i, create_graph=True,␣
↪retain_graph=True)[0].T

    dv2dx2, dv2dxdy = torch.autograd.grad(dvdx.sum(), X_i, create_graph=True,␣
↪retain_graph=True)[0].T
    dv2dydx, dv2dy2 = torch.autograd.grad(dvdy.sum(), X_i, create_graph=True,␣
↪retain_graph=True)[0].T

    #Calculate the first PDE loss
    t1 = G*(du2dx2 + du2dy2)
    t2 = G*((1+v)/(1-v))*(du2dx2 + dv2dydx)
    t3 = torch.sin(2*torch.pi*x)*torch.sin(2*torch.pi*y)
    loss_1 = t1 + t2 + t3

    #Calculate the second PDE loss
    t1 = G*(dv2dx2 + dv2dy2)
    t2 = G*((1+v)/(1-v))*(du2dxdy + dv2dy2)
    t3 = torch.sin(torch.pi*x) + torch.sin(2*torch.pi*y)
    loss_2 = t1 + t2 + t3

    #total pde loss (minimizing both individual losses)
    loss_pde = torch.mean(loss_1**2) + torch.mean(loss_2**2)
    return loss_pde
```

3.3.3 Total Loss

Now, the total loss:

```
[ ]: pde_losses = []
boundary_losses = []
total_losses = []



def loss(model, epoch, verbosity):
    """Calculate the total loss.

    Parameters
    ----------
    X : torch.Tensor
        The points.
    model : torch.nn.Module
        The model. It predicts the displacements for the points.

    Returns
    -------
    torch.Tensor
        The loss.
```

```
    """
    loss_pde = pde_loss(X_interior, model)
    loss_b = boundary_loss(model(X_boundary), U_boundary, 1000)
    total_loss = loss_pde +  loss_b

    pde_losses.append(loss_pde.item())
    boundary_losses.append(loss_b.item())
    total_losses.append(total_loss.item())
    if verbosity == 0:
        return total_loss
    if (epoch + 1) % verbosity == 0:
        print(
            f"Epoch {epoch+1:>4d} => PDE Loss: {loss_pde.item():.6f} | ␣
↪Boundary Loss: {loss_b.item():.6f} | Total Loss: {total_loss.item():.6f}"
        )
    return total_loss
```

Excellent! We have the loss function ready. Next, we'll create the optimizer and train the model.

3.4 Training

[ ]:
```
model = Displacements()
```

[ ]:
```
pde_losses = []
boundary_losses = []
total_losses = []
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
for epoch in range(2000):
    optimizer.zero_grad()
    l = loss(model, epoch, 100)
    l.backward()
    optimizer.step()
```

```
Epoch  100 => PDE Loss: 1.220348 |  Boundary Loss: 0.000802 | Total Loss:
1.221150
Epoch  200 => PDE Loss: 0.961237 |  Boundary Loss: 0.023160 | Total Loss:
0.984397
Epoch  300 => PDE Loss: 0.600202 |  Boundary Loss: 0.134925 | Total Loss:
0.735127
Epoch  400 => PDE Loss: 0.481341 |  Boundary Loss: 0.123553 | Total Loss:
0.604894
Epoch  500 => PDE Loss: 0.471866 |  Boundary Loss: 0.105441 | Total Loss:
0.577307
Epoch  600 => PDE Loss: 0.453727 |  Boundary Loss: 0.101648 | Total Loss:
0.555375
Epoch  700 => PDE Loss: 0.419336 |  Boundary Loss: 0.096757 | Total Loss:
0.516092
```

```
Epoch  800 => PDE Loss: 0.393134 |  Boundary Loss: 0.086332 | Total Loss:
0.479466
Epoch  900 => PDE Loss: 0.373466 |  Boundary Loss: 0.084987 | Total Loss:
0.458453
Epoch 1000 => PDE Loss: 0.318551 |  Boundary Loss: 0.090182 | Total Loss:
0.408734
Epoch 1100 => PDE Loss: 0.251952 |  Boundary Loss: 0.077333 | Total Loss:
0.329285
Epoch 1200 => PDE Loss: 0.200594 |  Boundary Loss: 0.060307 | Total Loss:
0.260901
Epoch 1300 => PDE Loss: 0.138601 |  Boundary Loss: 0.054153 | Total Loss:
0.192754
Epoch 1400 => PDE Loss: 0.112646 |  Boundary Loss: 0.043940 | Total Loss:
0.156587
Epoch 1500 => PDE Loss: 0.104543 |  Boundary Loss: 0.039561 | Total Loss:
0.144104
Epoch 1600 => PDE Loss: 0.098899 |  Boundary Loss: 0.048736 | Total Loss:
0.147635
Epoch 1700 => PDE Loss: 0.092614 |  Boundary Loss: 0.037408 | Total Loss:
0.130022
Epoch 1800 => PDE Loss: 0.088509 |  Boundary Loss: 0.041126 | Total Loss:
0.129635
Epoch 1900 => PDE Loss: 0.082002 |  Boundary Loss: 0.045362 | Total Loss:
0.127365
Epoch 2000 => PDE Loss: 0.076745 |  Boundary Loss: 0.035364 | Total Loss:
0.112109
```

Let's save the model:

```
[ ]: torch.save(model.state_dict(), 'solutions_model.pt')
```

3.5 Results

Now, we'll plot the results. We will use 300 points for plotting the results. First, we need to create a meshgrid:

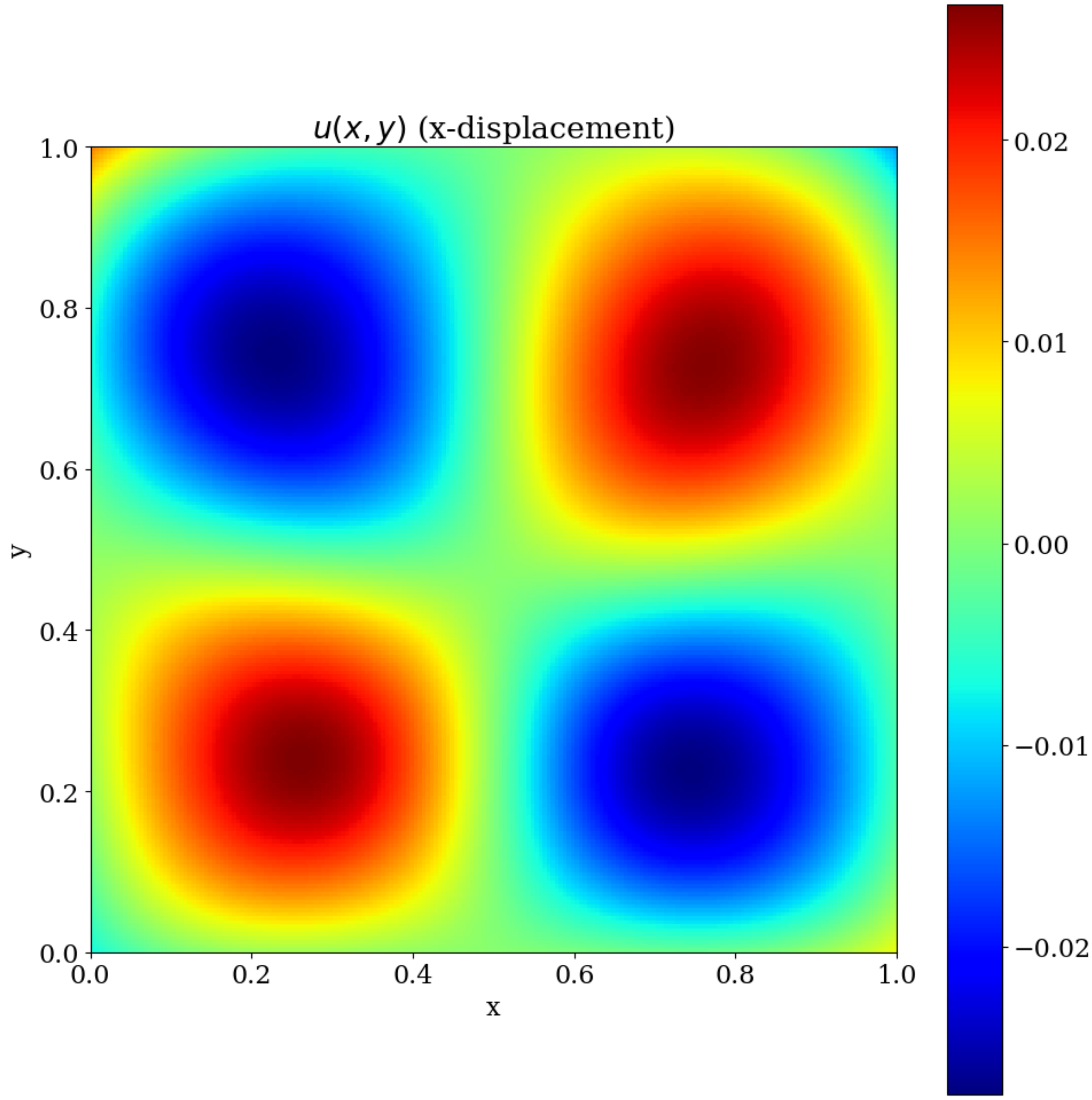
```
[ ]: x_mesh, y_mesh = np.meshgrid(np.linspace(0, 1, 200), np.linspace(0, 1, 200))
     X_to_predict = torch.tensor(np.concatenate((x_mesh.reshape(-1, 1), y_mesh.
      ↪reshape(-1, 1)), axis=1), dtype=torch.float32, requires_grad=True)
```

Next, predict the values for the meshgrid anf separate the values for u and v.

```
[ ]: U_pred = model(X_to_predict)
     u_pred = U_pred[:, 0].detach().numpy()
     u_pred = u_pred.reshape(x_mesh.shape)
     v_pred = U_pred[:, 1].detach().numpy()
     v_pred = v_pred.reshape(x_mesh.shape)
```

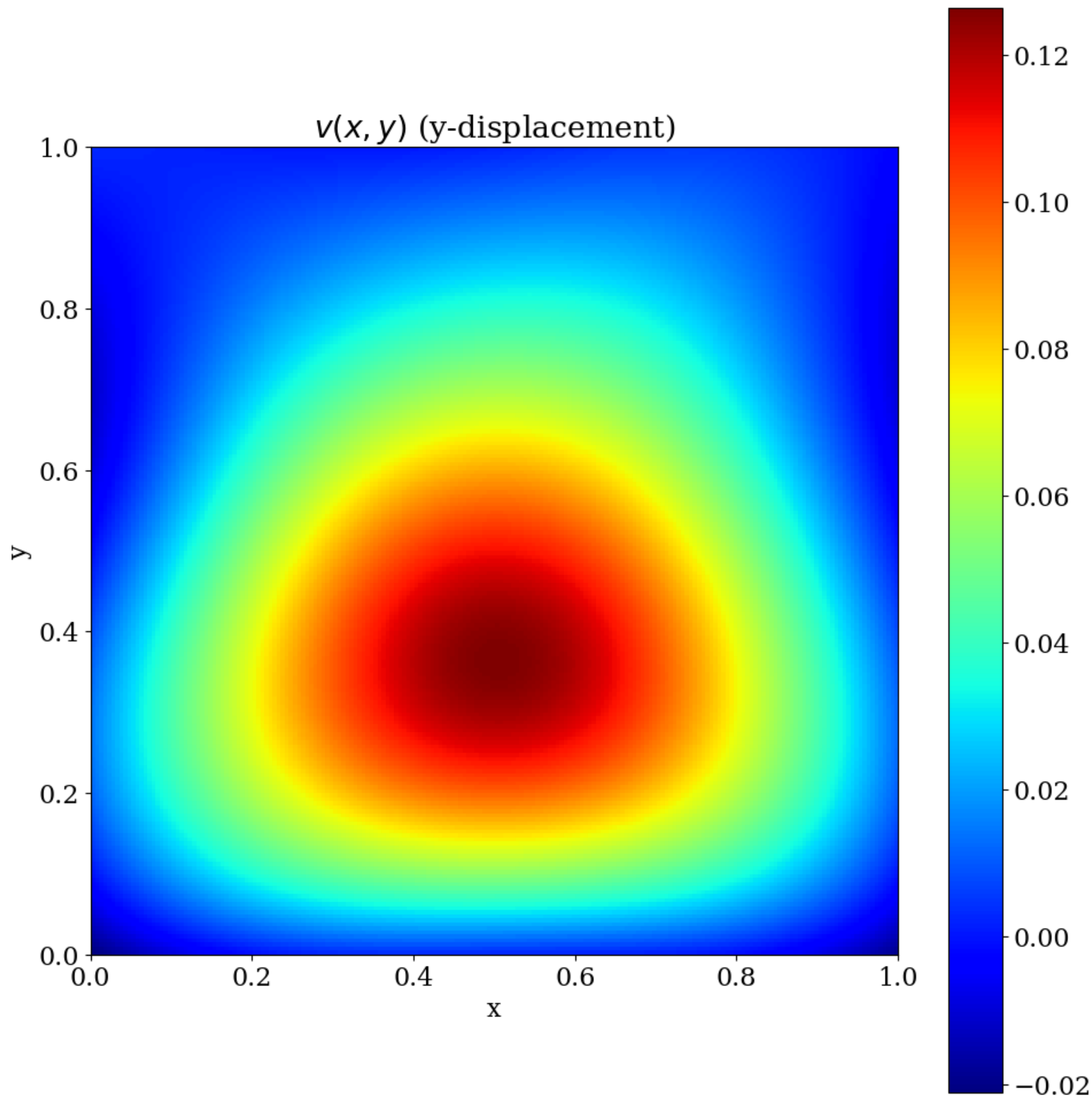
Now, we can plot the displacements:

[ ]:
```
#plot image
plt.figure(figsize=(10, 10))
plt.imshow(u_pred, cmap='jet', origin='lower', extent=[0, 1, 0, 1])
plt.colorbar()
plt.xlabel("x")
plt.ylabel("y")
plt.title("$u(x, y)$ (x-displacement)")
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "0201.png"))
```



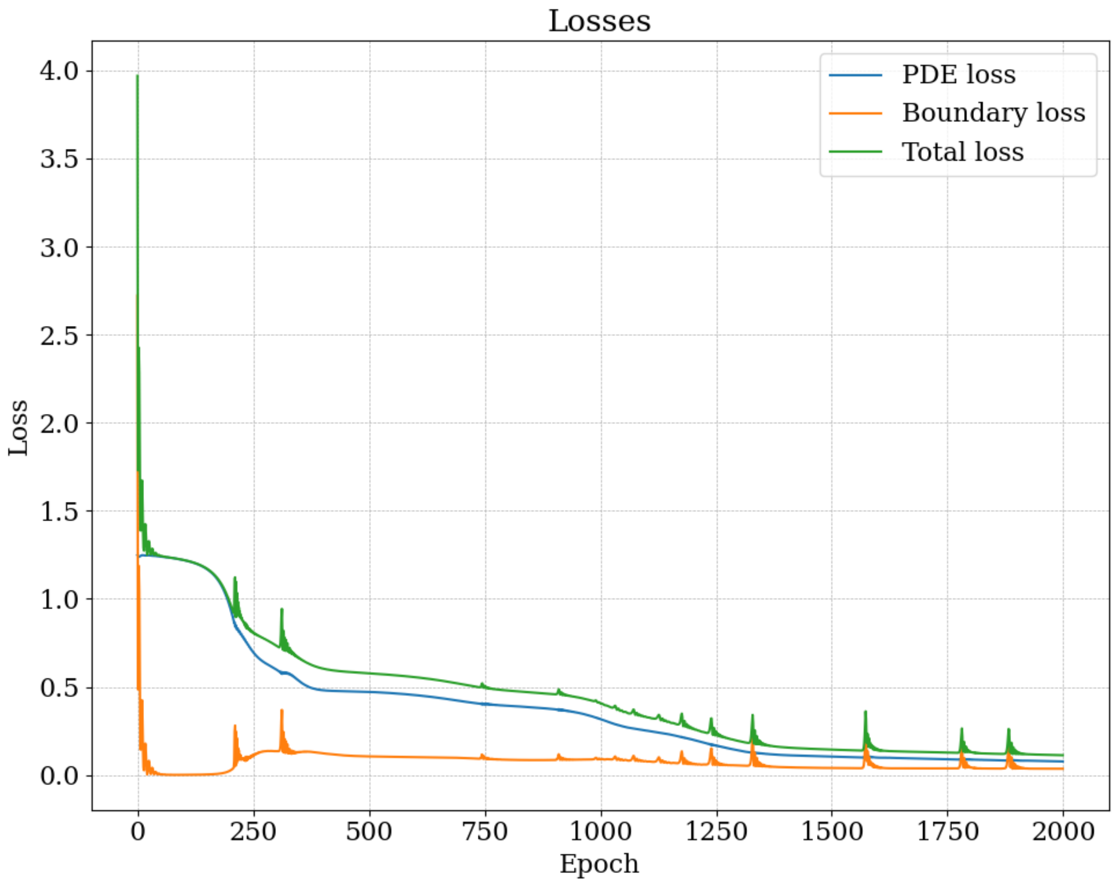
[ ]:
```
plt.figure(figsize=(10, 10))
plt.imshow(v_pred, cmap='jet', origin='lower', extent=[0, 1, 0, 1])
plt.colorbar()
```

```
plt.xlabel("x")
plt.ylabel("y")
plt.title("$v(x, y)$ (y-displacement)")
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "0202.png"))
```



[ ]:
```
plt.figure(figsize=(10, 8))
plt.plot(pde_losses, label="PDE loss")
plt.plot(boundary_losses, label="Boundary loss")
plt.plot(total_losses, label="Total loss")
plt.legend()
plt.title("Losses")
plt.xlabel("Epoch")
```

```python
plt.ylabel("Loss")
plt.grid()
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "0301.png"))
```



4 Testing on Custom Points

This section deales with evaluating the model on 2000 interior collocation points and 400 boundary points. For this, I will create 2500 points from a uniform distribution by using `np.meshgrid`. These 2500 points already have the boundary points. I will then use the model to predict the values for these points and then plot the results.

[ ]:
```python
x1 = np.linspace(0, 1, 50, endpoint=True)
y1 = np.linspace(0, 1, 50, endpoint=True)

X_C, Y_C = np.meshgrid(x1, y1)

test_points = torch.tensor(np.concatenate((X_C.reshape(-1, 1), Y_C.reshape(-1,
    1)), axis=1), dtype=torch.float32, requires_grad=True)
```

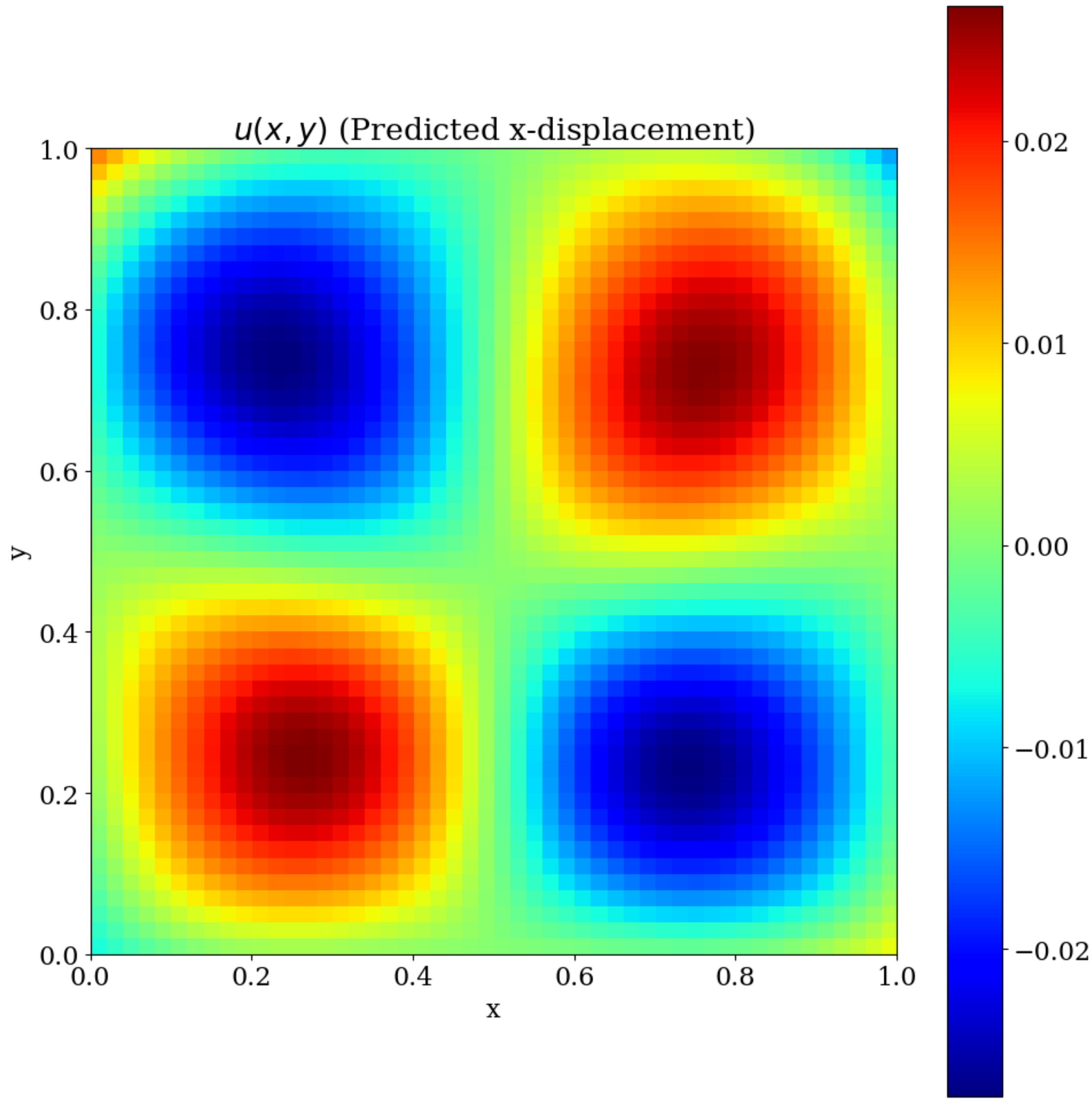
[ ]:
```python
loaded_model = Displacements()
loaded_model.load_state_dict(torch.load('solutions_model.pt'))
```

[ ]:
```
<All keys matched successfully>
```

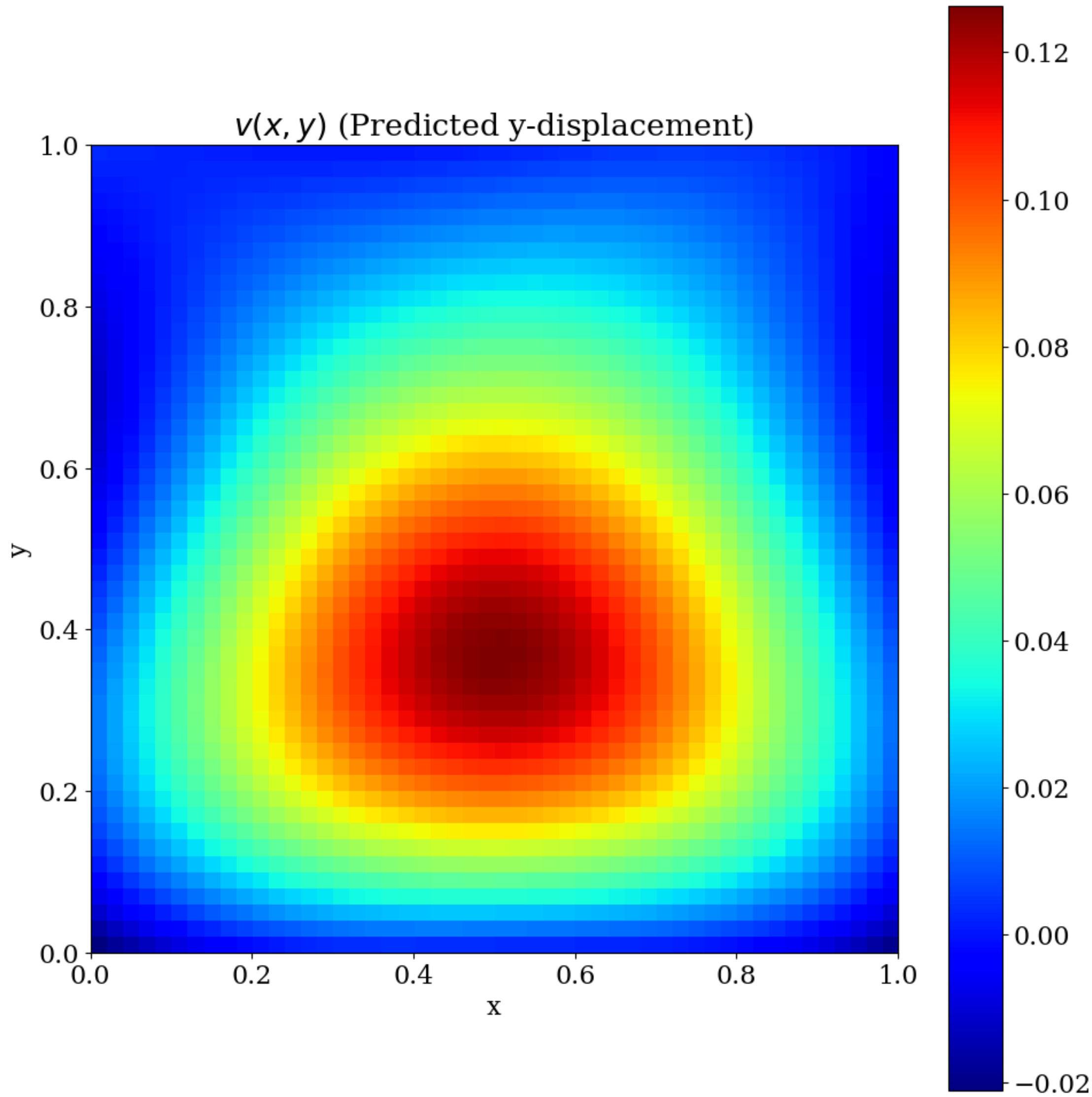
[ ]:
```python
U_pred = loaded_model(test_points)
```

[ ]:
```python
u_pred = U_pred[:, 0].detach().numpy()
u_pred = u_pred.reshape(X_C.shape)
v_pred = U_pred[:, 1].detach().numpy()
v_pred = v_pred.reshape(X_C.shape)
```

[ ]:
```python
plt.figure(figsize=(10, 10))
plt.imshow(u_pred, cmap='jet', origin='lower', extent=[0, 1, 0, 1])
plt.colorbar()
plt.xlabel("x")
plt.ylabel("y")
plt.title("$u(x, y)$ (Predicted x-displacement)")
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "0501.png"))
```



[ ]:
```
plt.figure(figsize=(10, 10))
plt.imshow(v_pred, cmap='jet', origin='lower', extent=[0, 1, 0, 1])
plt.colorbar()
plt.xlabel("x")
plt.ylabel("y")
plt.title("$v(x, y)$ (Predicted y-displacement)")
plt.tight_layout()
plt.savefig(os.path.join(PLOTS_DIR, "0502.png"))
```


$v(x,y)$ (Predicted y-displacement)
1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x
0.12
0.10
0.08
0.06
0.04
0.02
0.00
−0.02