湖南大學

HUNAN UNIVERSITY

课 程 论 文

论 文 题 目 ： PINN在弹性平面应变问题上的应用

学 生 姓 名 ： 雷锐

学 生 学 号 ： S230200246

专 业 班 级 ： 机械2311班

学 院 名 称 ： 机械与运载工程学院

指 导 老 师 ： 王琥

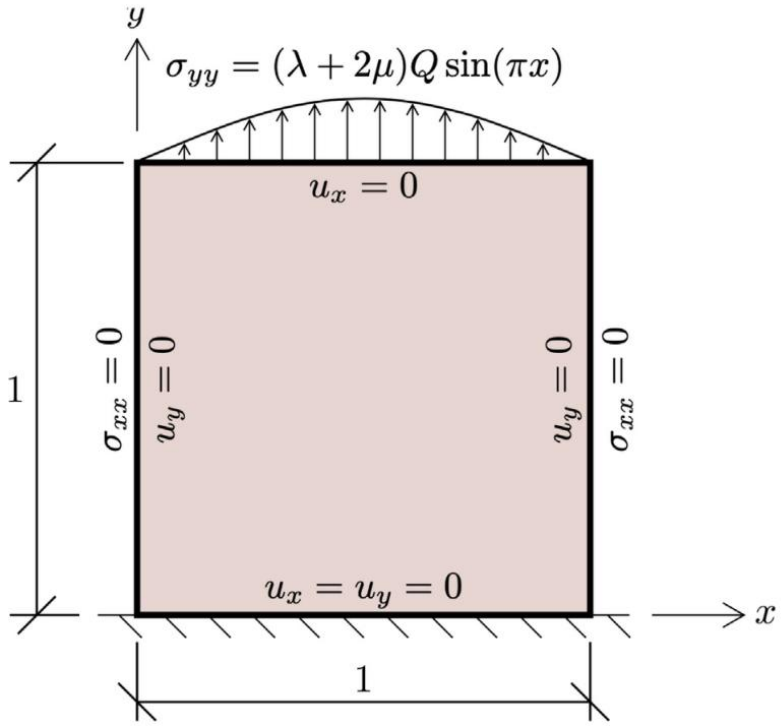
2024年8月13日

目 录

第 1 章 问题描述

在老师给定的研究范围中，我选择 PINN 为此次课程作业的研究主题。

此次课程论文参考论文，对其中用 PINN 求解弹性平面问题的部分进行代码复现。该力学问题的设置如图所示，计算域已进行了正则化处理，已知 1×1 单位正方形各边界上的边界条件及体力，要根据弹性问题的基本定律求解该单元正方形上的位移。



其中，该单位正方形上的体力表达式为：

$$
\begin{aligned}
f_x &= \lambda\left[4\pi^2\cos(2\pi x)\sin(\pi y) - \pi\cos(\pi x)Qy^3\right] \\
&\quad + \mu\left[9\pi^2\cos(2\pi x)\sin(\pi y) - \pi\cos(\pi x)Qy^3\right] \\
f_y &= \lambda\left[-3\sin(\pi x)Qy^2 + 2\pi^2\sin(2\pi x)\cos(\pi y)\right] \\
&\quad + \mu\left[-6\sin(\pi x)Qy^2 + 2\pi^2\sin(2\pi x)\cos(\pi y) + \pi^2\sin(\pi x)Qy^4/4\right].
\end{aligned}
$$

该弹性平面应变问题中，重要参数的取值为：

$$\lambda = 1,\ \ \mu = 0.5, Q = 4$$

求解该单位正方形中位移的解。

位移的精确解析解如下，作为参考。

$$
\begin{aligned}
u_x(x, y) &= \cos(2\pi x)\sin(\pi y), \\
u_y(x, y) &= \sin(\pi x)Qy^4/4.
\end{aligned}
$$

第 2 章 求解思路

首先，该问题是一个简单的弹性力学问题，求解该问题的基础是二维线弹性的三大基本方程，即动量平衡、本够模型和运动学关系三大方程。

$$\sigma_{ij,j} + f_i = 0,$$

$$\sigma_{ij} = \lambda\delta_{ij}\varepsilon_{kk} + 2\mu\varepsilon_{ij},$$

$$\varepsilon_{ij} = \frac{1}{2}\left(u_{i,j} + u_{j,i}\right).$$

将动量平衡式子在 x,y 方向上分解：

$$\sigma_{\mathrm{xx},x} + \sigma_{\mathrm{xy},y} + f_x = 0$$

$$\sigma_{\mathrm{xy},x} + \sigma_{\mathrm{yy},y} + f_y = 0$$

将本构模型展开：

$$①i = j = x$$

$$\sigma_{xx} = \lambda\delta_{xx}\varepsilon_{kk} + 2\mu\varepsilon_{xx} = \lambda\varepsilon_{kk} + 2\mu\varepsilon_{xx} = \lambda(\varepsilon_{xx} + \varepsilon_{yy}) + 2\mu\varepsilon_{xx} = (\lambda + 2\mu)\varepsilon_{xx} + \lambda\varepsilon_{yy}$$

$$②i = x, j = y$$

$$\sigma_{xy} = \lambda\delta_{xy}\varepsilon_{kk} + 2\mu\varepsilon_{xy} = 2\mu\varepsilon_{xy}$$

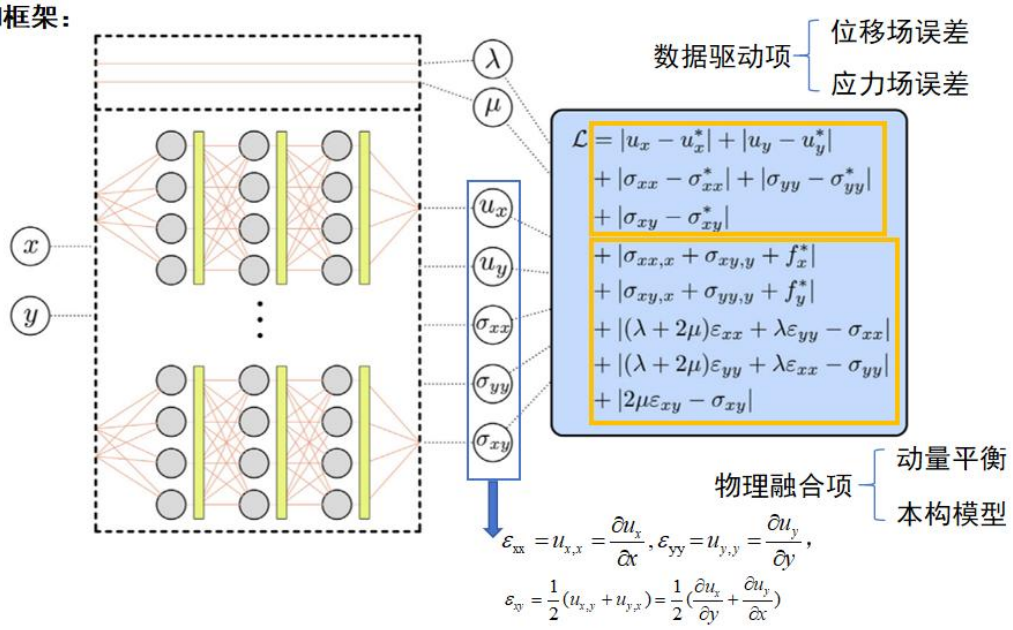
$$③i = j = y$$

$$\sigma_{yy} = \lambda\delta_{yy}\varepsilon_{kk} + 2\mu\varepsilon_{yy} = \lambda\varepsilon_{kk} + 2\mu\varepsilon_{yy} = \lambda(\varepsilon_{xx} + \varepsilon_{yy}) + 2\mu\varepsilon_{yy} = (\lambda + 2\mu)\varepsilon_{yy} + \lambda\varepsilon_{xx}$$

因此，根据上述的边界条件限制和三大基本方程作为约束进行区域内位移的求解。

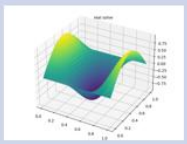
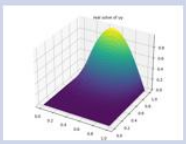
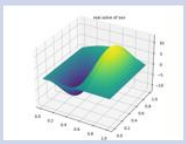
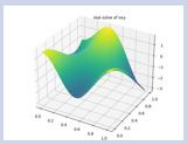
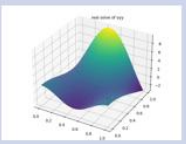
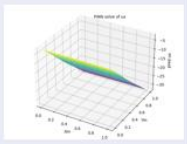
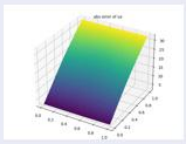
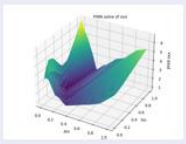
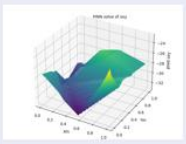
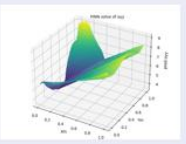
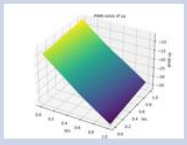
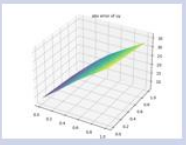
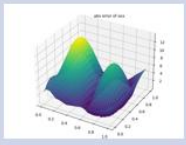
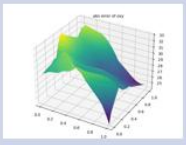
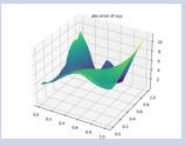
此次选用的求解工具是 PINN，物理信息神经网络（Physics-Informed Neural Networks，简称 PINN）是一种深度学习模型，它将物理定律融入到神经网络的训练过程中。与传统神经网络相比，PINN 能够利用较少的数据量，在噪声较大的环境下也能训练出具有良好泛化能力的模型。通过在损失函数中加入物理信息项，PINN 确保了预测结果不仅拟合数据，而且符合物理规律，特别适用于解决偏微分方程等科学和工程问题。本文所采用的 PINN 框架如下：

PINN框架：


数据驱动项
位移场误差
应力场误差
物理融合项
动量平衡
本构模型
$\mathcal{L} = |u_x - u_x^*| + |u_y - u_y^*|$
$+ |\sigma_{xx} - \sigma_{xx}^*| + |\sigma_{yy} - \sigma_{yy}^*|$
$+ |\sigma_{xy} - \sigma_{xy}^*|$
$+ |\sigma_{xx,x} + \sigma_{xy,y} + f_x^*|$
$+ |\sigma_{xy,x} + \sigma_{yy,y} + f_y^*|$
$+ |(\lambda + 2\mu)\varepsilon_{xx} + \lambda\varepsilon_{yy} - \sigma_{xx}|$
$+ |(\lambda + 2\mu)\varepsilon_{yy} + \lambda\varepsilon_{xx} - \sigma_{yy}|$
$+ |2\mu\varepsilon_{xy} - \sigma_{xy}|$
$\varepsilon_{xx} = u_{x,x} = \frac{\partial u_x}{\partial x}, \varepsilon_{yy} = u_{y,y} = \frac{\partial u_y}{\partial y},$
$\varepsilon_{xy} = \frac{1}{2}(u_{x,y} + u_{y,x}) = \frac{1}{2}(\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x})$

第 3 章 编写代码

根据所要实现的功能初步进行了代码编写，其运行结果如下，并不理想，无论是趋势还是数值都存在较大差距。

	u_x	u_y	σ_{xx}	σ_{xy}	σ_{yy}
真实解					
PINN解					
误差绝对值					

因此，在第一版的代码基础上进行了修改：从单个神经网络输出五个参数(single)改为五个神经网络输出五个参数(independent)，调整所用 MLP 的层数和参数，在损失函数中加入数据驱动项并调整各项权重以及修正表达式，在修改和调试后，代码能实现该平面弹性应变问题的求解并保有一定精度。

最终的代码的框架如下：

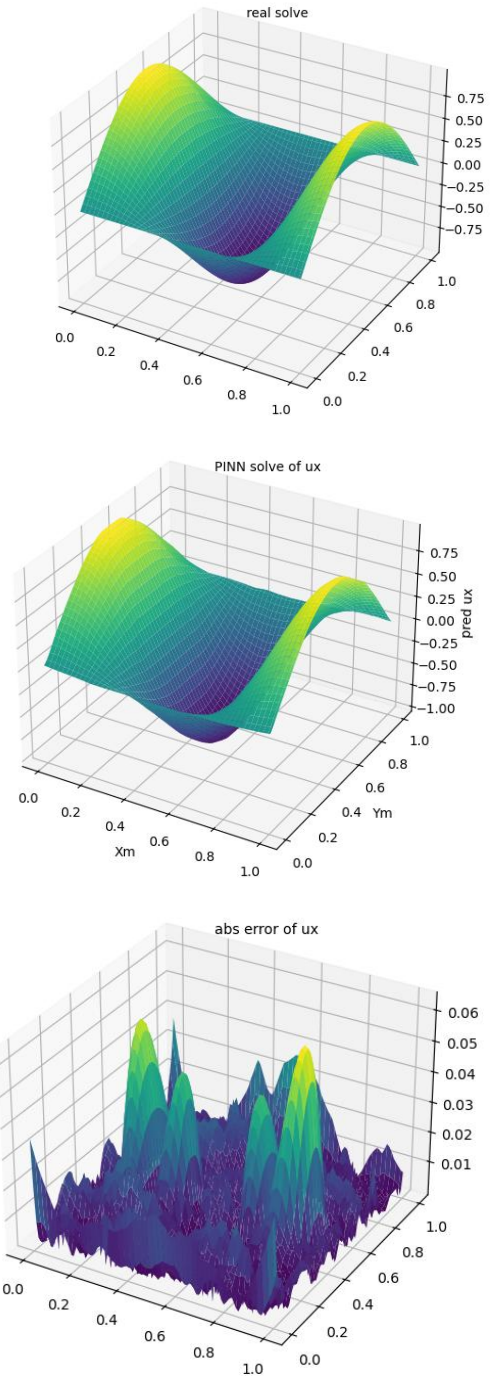
1.导入所需库，进行参数的初始化，设置随机种子；2.定义域并进行采样；3.编写 MLP 神经网络；4.定义 PINN 模型；5.创建 PINN 模型实例，定义损失函数；6、训练，求解并绘图。

具体代码已上传 github，请参考。

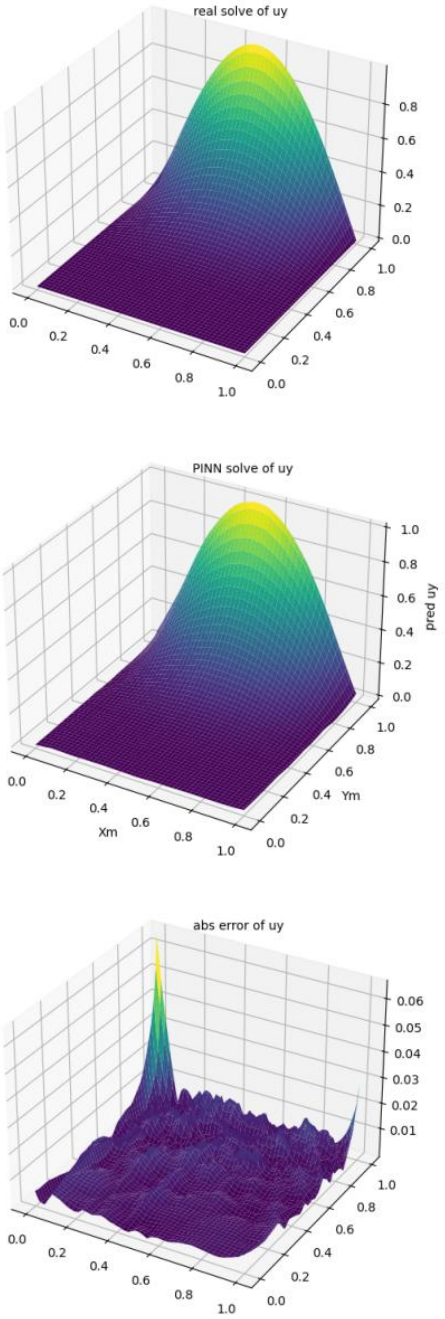
第 4 章 求解结果

运行该 python 文件，整理运算结果如下：

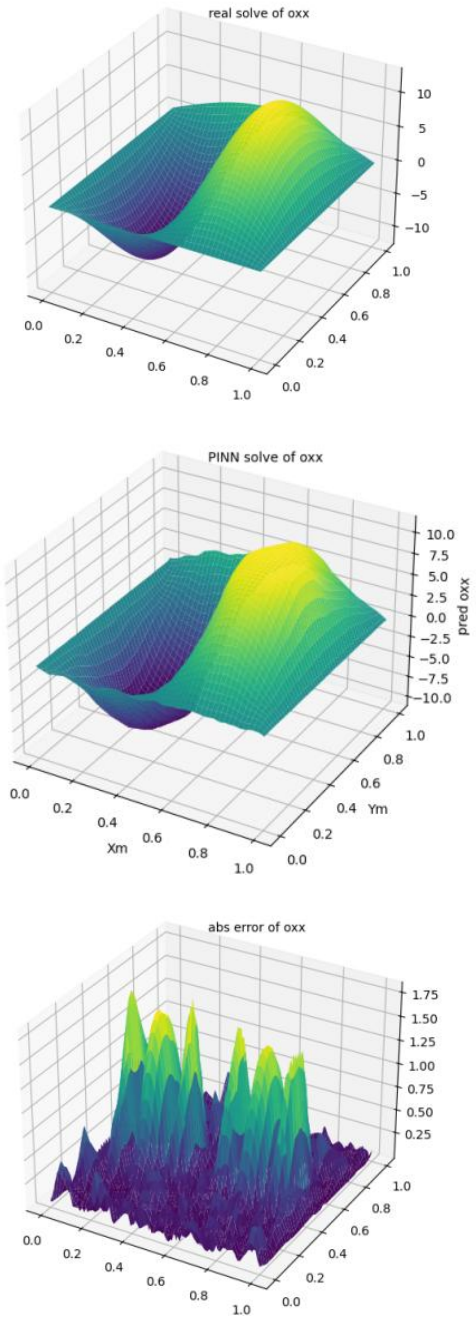
ux 的真实解、PINN 解和误差绝对值。



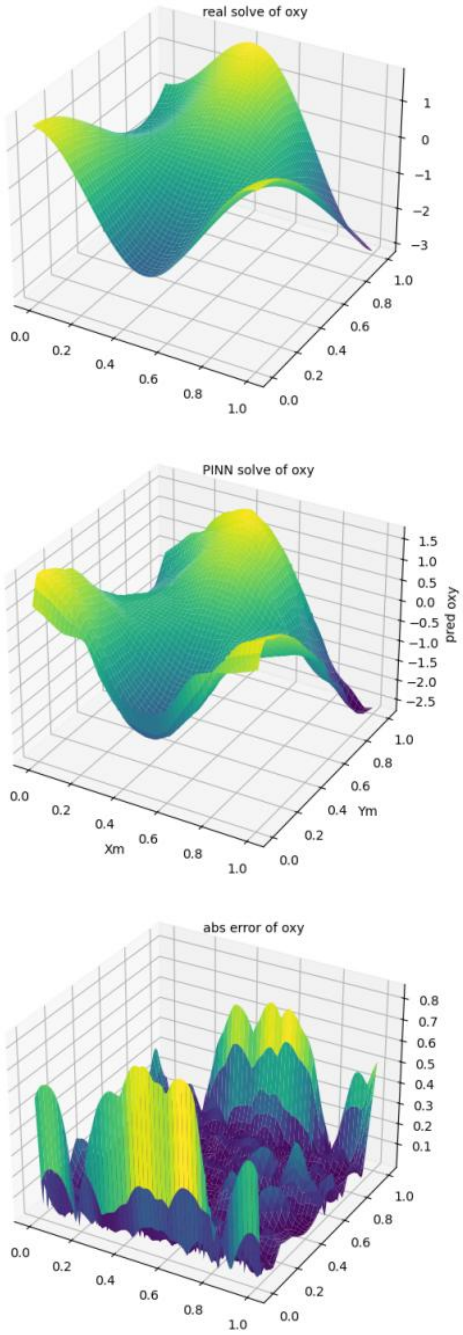
uy 的真实解、PINN 解和误差绝对值。



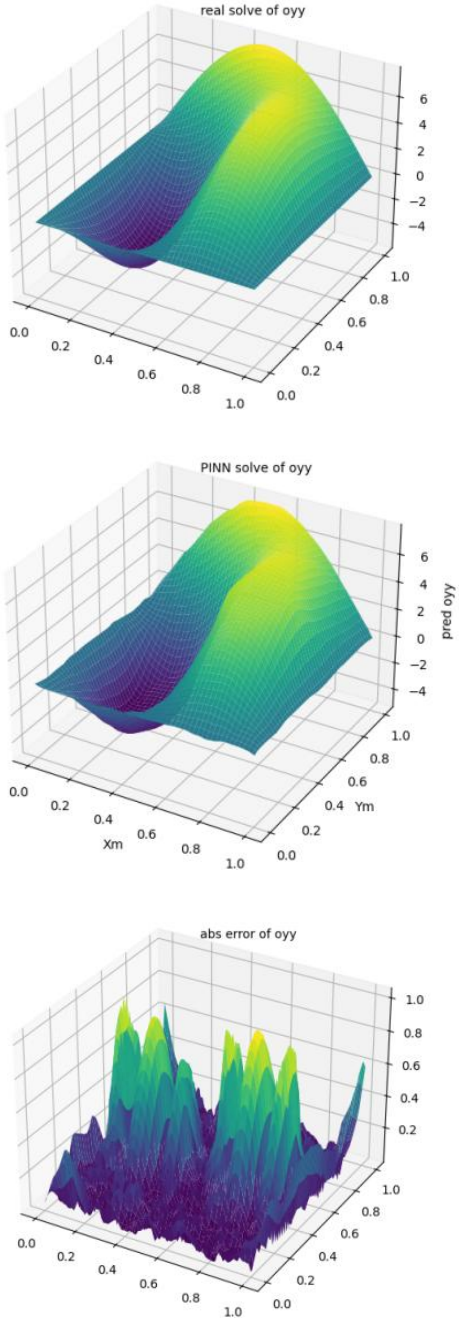
σxx 的真实解、PINN 解和误差绝对值。



σxy 的真实解、PINN 解和误差绝对值。



σyy 的真实解、PINN 解和误差绝对值。



经对比分析可知，PINN 求出的解与真实解在趋势和数值上都很接近，成功实现了 PINN 在该弹性平面应变问题上的复现。

附录 PINN 代码

```
import torch
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
print(torch.__version__)
print(torch.cuda.is_available())
print(torch.version.cuda)


epochs = 10000  # 迭代次数 10000 次
h = 100  # 画图网格密度
N = 5000  # 内点配置点数
N1 = 1250  # 边界点配置点数
N2 = 5000  # 数据点个数


# 这一段的作用是保证实验的可重复性，保证每次运行结果一样
def setup_seed(seed):
    torch.manual_seed(seed)  # seed 就是随机种子的值
    torch.cuda.manual_seed_all(seed)  # 设置所有 CUDA 设备的随机种子，确保在使用 CUDA 加速时（GPU）的随机数生成也是确定性的。
    torch.backends.cudnn.deterministic = True  # cudnn 模式是确定性的


setup_seed(888888)


# Domain and Sampling （定义域并采样）
```

```
def interior(n=N):  # 内点是在边界（计算区域）内的点，下面几个是在边界条件上
    x = torch.rand(n, 1)
    y = torch.rand(n, 1)
    fx = (8.5 * torch.pi**2 * torch.cos(2*torch.pi*x)*torch.sin(torch.pi*y) -
          6 * torch.pi * torch.cos(torch.pi*x) * y**3)  # fx*
    fy = (-24 * torch.sin(torch.pi*x) * y**2 + 3 * torch.pi**2 * torch.cos(torch.pi*y) * torch.sin(2*torch.pi*x) +
          0.5 * torch.pi**2 * torch.sin(torch.pi*x) * y**4)  # fy*
    return x.requires_grad_(True), y.requires_grad_(True), fx, fy  # 标记为需要梯度运算


def down(n=N1):
    x = torch.rand(n, 1)
    y = torch.zeros_like(x)
    return x.requires_grad_(True), y.requires_grad_(True)


def up(n=N1):
    x = torch.rand(n, 1)
    y = torch.ones_like(x)
    return x.requires_grad_(True), y.requires_grad_(True)


def left(n=N1):
    y = torch.rand(n, 1)
    x = torch.zeros_like(y)
    return x.requires_grad_(True), y.requires_grad_(True)
```

```
def right(n=N1):
    y = torch.rand(n, 1)
    x = torch.ones_like(y)
    return x.requires_grad_(True), y.requires_grad_(True)


def data_interior(n=N2):
    x = torch.rand(n, 1)
    y = torch.rand(n, 1)
    ux_solve = torch.cos(2*torch.pi*x)*torch.sin(torch.pi*y)
    uy_solve = torch.sin(torch.pi*x)*y**4
    return x.requires_grad_(True), y.requires_grad_(True), ux_solve, uy_solve


# NN  神经网络
#  神经网络是 2 个输入到 1 个输出，中间有三个隐藏层的神经网络，每层 32 个神经元
# MLP  多层感知机/ANN 人工神经网络/前馈神经网络  是最基础的全连接神经网络
class MLP(torch.nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MLP, self).__init__()
        self.fc1 = torch.nn.Linear(input_size, hidden_size)
        self.fc2 = torch.nn.Linear(hidden_size, hidden_size)
        self.fc3 = torch.nn.Linear(hidden_size, hidden_size)
        self.fc4 = torch.nn.Linear(hidden_size, hidden_size)
        self.fc5 = torch.nn.Linear(hidden_size, output_size)

    def forward(self, x):
```

```
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = torch.relu(self.fc3(x))
        x = torch.relu(self.fc4(x))
        x = self.fc5(x)
        return x


# 定义 PINN 模型
class PINN(torch.nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(PINN, self).__init__()
        self.nn1 = MLP(input_size, hidden_size, output_size)
        self.nn2 = MLP(input_size, hidden_size, output_size)
        self.nn3 = MLP(input_size, hidden_size, output_size)
        self.nn4 = MLP(input_size, hidden_size, output_size)
        self.nn5 = MLP(input_size, hidden_size, output_size)

    def forward(self, x, y):
        param1 = self.nn1(torch.cat((x, y), dim=1))
        param2 = self.nn2(torch.cat((x, y), dim=1))
        param3 = self.nn3(torch.cat((x, y), dim=1))
        param4 = self.nn4(torch.cat((x, y), dim=1))
        param5 = self.nn5(torch.cat((x, y), dim=1))
        return param1, param2, param3, param4, param5


# 创建 PINN 模型实例
input_size = 2  # 输入特征数（x 和 y）
```

```
hidden_size = 64    # 隐藏层大小
output_size = 1    # 输出参数大小
model = PINN(input_size, hidden_size, output_size)

# loss
loss = torch.nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
loss_values = []
# 损失函数使用 MSE 均方方差：MSE=1/n[∑(y_true,i-y_pred,i)2]


def gradients(w, x, order=1):
    if order == 1:
        return torch.autograd.grad(w, x, grad_outputs=torch.ones_like(w),
                                   create_graph=True,
                                   only_inputs=True,)[0]
    else:
        return gradients(gradients(w, x), x, order=order - 1)


# 损失
def l_dl1():
    x, y, fx, fy = interior()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(gradients(oxx, x, 1) + gradients(oxy, y, 1), -fx)


def l_dl2():
    x, y, fx, fy = interior()
    ux, uy, oxx, oxy, oyy = model(x, y)
```

```
    return loss(gradients(oxy, x, 1) + gradients(oyy, y, 1), -fy)


def l_bg1():
    x, y, fx, fy = interior()
    ux, uy, oxx, oxy, oyy = model(x, y)
    exx = gradients(ux, x, 1)
    eyy = gradients(uy, y, 1)
    exy = (gradients(ux, y, 1) + gradients(uy, x, 1))/2
    return loss(oxx, 2*exx + eyy)


def l_bg2():
    x, y, fx, fy = interior()
    ux, uy, oxx, oxy, oyy = model(x, y)
    exx = gradients(ux, x, 1)
    eyy = gradients(uy, y, 1)
    exy = (gradients(ux, y, 1) + gradients(uy, x, 1))/2
    return loss(oyy, 2*eyy + exx)


def l_bg3():
    x, y, fx, fy = interior()
    ux, uy, oxx, oxy, oyy = model(x, y)
    exx = gradients(ux, x, 1)
    eyy = gradients(uy, y, 1)
    exy = (gradients(ux, y, 1) + gradients(uy, x, 1))/2
    return loss(oxy, exy)
```

```
def l_down1():
    x, y = down()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(ux, torch.zeros_like(ux))


def l_down2():
    x, y = down()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(uy, torch.zeros_like(uy))


def l_up1():
    x, y = up()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(ux, torch.zeros_like(ux))


def l_up2():
    x, y = up()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(oyy, 8 * torch.sin(torch.pi * x))


def l_left1():
    x, y = left()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(oxx, torch.zeros_like(oxx))
```

```
def l_left2():
    x, y = left()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(uy, torch.zeros_like(uy))


def l_right1():
    x, y = right()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(uy, torch.zeros_like(uy))


def l_right2():
    x, y = right()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(oxx, torch.zeros_like(oxx))


def l_data1():
    x, y, ux_solve, uy_solve = data_interior()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(ux_solve, ux)


def l_data2():
    x, y, ux_solve, uy_solve = data_interior()
    ux, uy, oxx, oxy, oyy = model(x, y)
    return loss(uy_solve, uy)
```

```
xc = torch.linspace(0, 1, h)  # 在 0 到 1 之间等间距撒 h-2 个点，加上两端一共 h 个点，间距为（1-0）/(h-1)
xm, ym = torch.meshgrid(xc, xc, indexing='ij')  # 以等间距采的点创建二维网格,xm 每一行都是 xc，ym 每一列都是 xc
xx = xm.reshape(-1, 1)  # 把现在的数组改写成 i 行 j 列，这里 i=-1 表示未指定值，可以是任何正整数，j=1 表示改写成 1 列的
yy = ym.reshape(-1, 1)
xy = torch.cat([xx, yy], dim=1)  # h=10 的话，xy 大小是（10000,2），每一行就是一个点的 xy 坐标

# Training
opt = torch.optim.Adam(params=model.parameters())
for i in range(epochs):
    opt.zero_grad()
    # 计算损失
    l_value = (l_bg1()*2 + l_bg2()*2 + l_bg3() + l_down1() + l_down2() + l_up1()/10 + l_up2()/10 + l_left1() +
                    l_left2() + l_right1() + l_right2())*100 + l_dl1()/300 + l_dl2()/50 + (l_data1() + l_data2())*10000
    l_value.backward()
    opt.step()
    loss_values.append(l_value.item())
    if i % 100 == 0:
        print(f"Epoch {i}, Loss: {l_value.item()}")

# Inference
u_pred, uy_pred, oxx_pred, oxy_pred, oyy_pred = model(xx, yy)
# 将元组中的所有张量合并为一个张量，这里假设所有张量形状相同，可以直接相加
u_real = torch.cos(2*torch.pi*xx)*torch.sin(torch.pi*yy)
```

```
uy_real = torch.sin(torch.pi*xx) * (yy**4)
exx_real = -2 * torch.pi * torch.sin(2*torch.pi*xx) * torch.sin(torch.pi*yy)
eyy_real = 4 * torch.sin(torch.pi*xx) * yy**3
exy_real =
(torch.pi*torch.cos(2*torch.pi*xx)*torch.cos(torch.pi*yy)+torch.pi*torch.cos(torch.pi
*xx)*yy**4)/2
oxx_real = 2*exx_real + eyy_real
oxy_real = exy_real
oyy_real = 2*eyy_real+exx_real
# ux
u_error = torch.abs(u_pred-u_real)
u_pred_fig = u_pred.reshape(h, h)
u_real_fig = u_real.reshape(h, h)
u_error_fig = u_error.reshape(h, h)
print("Max abs error of ux is:",
float(torch.max(torch.abs(u_pred-torch.cos(2*torch.pi*xx)*torch.sin(torch.pi*yy)))))

# uy
uy_error = torch.abs(uy_pred-uy_real)
uy_pred_fig = uy_pred.reshape(h, h)
uy_real_fig = uy_real.reshape(h, h)
uy_error_fig = uy_error.reshape(h, h)
print("Max abs error of uy is:",
float(torch.max(torch.abs(uy_pred-torch.sin(torch.pi*xx) * (yy**4)))))

# oxx
oxx_error = torch.abs(oxx_pred-oxx_real)
oxx_pred_fig = oxx_pred.reshape(h, h)
oxx_real_fig = oxx_real.reshape(h, h)
oxx_error_fig = oxx_error.reshape(h, h)
```

```
print("Max abs error of oxx is:", float(torch.max(torch.abs(oxx_pred-2*(-2 * torch.pi
* torch.sin(2*torch.pi*xx) * torch.sin(torch.pi*yy)) - 4 * torch.sin(torch.pi*xx) *
yy**3))))

# oxy
oxy_error = torch.abs(oxy_pred-oxy_real)
oxy_pred_fig = oxy_pred.reshape(h, h)
oxy_real_fig = oxy_real.reshape(h, h)
oxy_error_fig = oxy_error.reshape(h, h)
print("Max abs error of oxy is:",
float(torch.max(torch.abs(oxy_pred-(torch.pi*torch.cos(2*torch.pi*xx)*torch.cos(torc
h.pi*yy)+torch.pi*torch.cos(torch.pi*xx)*yy**4)/2))))

# oyy
oyy_error = torch.abs(oyy_pred-oyy_real)
oyy_pred_fig = oyy_pred.reshape(h, h)
oyy_real_fig = oyy_real.reshape(h, h)
oyy_error_fig = oyy_error.reshape(h, h)
print("Max abs error of oyy is:", float(torch.max(torch.abs(oyy_pred-2*(4 *
torch.sin(torch.pi*xx) * yy**3)+2 * torch.pi * torch.sin(2*torch.pi*xx) *
torch.sin(torch.pi*yy)))))
print(xx)
print(xy)

# 绘制损失函数随迭代次数变化的图
plt.figure(1)
plt.plot(loss_values, label='Training Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs. Epochs')
```

```
plt.legend()
plt.show()
plt.savefig("loss_vs_epochs.png")

# 作 PINN of ux 数值解图
fig = plt.figure(2)  # 创建图像
ax = Axes3D(fig)  # 创建 3D 坐标轴对象
fig.add_axes(ax)  # 添加坐标轴
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
u_pred_fig.detach().numpy(), cmap='viridis')
# 确定 xyz 三坐标轴绘制 3D 曲面，pytorch 的底层逻辑是 xm,ym,u 都是张量，需要把它们都转化为 numpy 多维数组
ax.text2D(0.5, 0.9, "PINN solve of ux", transform=ax.transAxes)
# 在图里(0.5,0.9)这个比例位置添加“PINN”字样（坐标值解释为坐标轴的比例，而不是数据坐标）（（0.5,0.5）就是整个图的正中间）
# 设置坐标轴标签
ax.set_xlabel('Xm')
ax.set_ylabel('Ym')
ax.set_zlabel('pred ux')
plt.show()  # 出图
fig.savefig("PINN solve of ux.png")  # 存图

# 作 ux 真解图
fig = plt.figure(3)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
u_real_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "real solve", transform=ax.transAxes)
plt.show()
```

```
fig.savefig("real solve of ux.png")

# 作 ux 误差图
fig = plt.figure(4)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
u_error_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "abs error of ux", transform=ax.transAxes)
plt.show()
fig.savefig("abs error of ux.png")

# 作 PINN of uy 数值解图
fig = plt.figure(5)  # 创建图像
ax = Axes3D(fig)  # 创建 3D 坐标轴对象
fig.add_axes(ax)  # 添加坐标轴
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
uy_pred_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "PINN solve of uy", transform=ax.transAxes)
ax.set_xlabel('Xm')
ax.set_ylabel('Ym')
ax.set_zlabel('pred uy')
plt.show()  # 出图
fig.savefig("PINN solve of uy.png")  # 存图

# 作 uy 真解图
fig = plt.figure(6)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
```

```
uy_real_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "real solve of uy", transform=ax.transAxes)
plt.show()
fig.savefig("real solve of uy.png")

# 作 uy 误差图
fig = plt.figure(7)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
uy_error_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "abs error of uy", transform=ax.transAxes)
plt.show()
fig.savefig("abs error of uy.png")

# 作 PINN of oxx 数值解图
fig = plt.figure(8)  # 创建图像
ax = Axes3D(fig)  # 创建 3D 坐标轴对象
fig.add_axes(ax)  # 添加坐标轴
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oxx_pred_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "PINN solve of oxx", transform=ax.transAxes)
ax.set_xlabel('Xm')
ax.set_ylabel('Ym')
ax.set_zlabel('pred oxx')
plt.show()  # 出图
fig.savefig("PINN solve of oxx.png")  # 存图

# 作 oxx 真解图
fig = plt.figure(9)
```

```
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oxx_real_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "real solve of oxx", transform=ax.transAxes)
plt.show()
fig.savefig("real solve of oxx.png")

# 作 oxx 误差图
fig = plt.figure(10)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oxx_error_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "abs error of oxx", transform=ax.transAxes)
plt.show()
fig.savefig("abs error of oxx.png")

# 作 PINN of oxy 数值解图
fig = plt.figure(11)  # 创建图像
ax = Axes3D(fig)  # 创建 3D 坐标轴对象
fig.add_axes(ax)  # 添加坐标轴
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oxy_pred_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "PINN solve of oxy", transform=ax.transAxes)
ax.set_xlabel('Xm')
ax.set_ylabel('Ym')
ax.set_zlabel('pred oxy')
plt.show()  # 出图
```

```
fig.savefig("PINN solve of oxy.png")  # 存图

# 作 oxy 真解图
fig = plt.figure(12)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oxy_real_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "real solve of oxy", transform=ax.transAxes)
plt.show()
fig.savefig("real solve of oxy.png")

# 作 oxy 误差图
fig = plt.figure(13)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oxy_error_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "abs error of oxy", transform=ax.transAxes)
plt.show()
fig.savefig("abs error of oxy.png")

# 作 PINN of oyy 数值解图
fig = plt.figure(14)  # 创建图像
ax = Axes3D(fig)  # 创建 3D 坐标轴对象
fig.add_axes(ax)  # 添加坐标轴
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oyy_pred_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "PINN solve of oyy", transform=ax.transAxes)
ax.set_xlabel('Xm')
```

```
ax.set_ylabel('Ym')
ax.set_zlabel('pred oyy')
plt.show()  # 出图
fig.savefig("PINN solve of oyy.png")  # 存图

# 作 oyy 真解图
fig = plt.figure(15)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oyy_real_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "real solve of oyy", transform=ax.transAxes)
plt.show()
fig.savefig("real solve of oyy.png")

# 作 oyy 误差图
fig = plt.figure(16)
ax = Axes3D(fig)
fig.add_axes(ax)
ax.plot_surface(xm.detach().numpy(), ym.detach().numpy(),
oyy_error_fig.detach().numpy(), cmap='viridis')
ax.text2D(0.5, 0.9, "abs error of oyy", transform=ax.transAxes)
plt.show()
fig.savefig("abs error of oyy.png")
```