计算练习：物理信息神经网络 (PINNs)

姜早

2025 年 5 月 8 日

1 概述

在本练习中，将使用前馈网络求解非线性扩散-对流-反应方程。该练习旨在帮助理解和应用物理信息神经网络 (PINNs) 解决实际问题。

2 问题描述

求解以下偏微分方程：

$$Lu \equiv u''(x) - Pe \cdot u'(x) + Da \cdot u(1-u) = 0, \quad x \in (0,1) \tag{1}$$

边界条件为：

$$u(0) = 0, \quad u(1) = 1 \tag{2}$$

其中，Pe 和 Da 是两个无量纲化的物理参数：

- Pe 是描述对流相对扩散强度的 Peclet 数
- Da 是描述反应相对扩散强度的 Damkohler 数

3 方法实现

为了方便调试参数和代码复用，我将整个训练网络分成了几个部分，分别为超参数设置，模型搭建，模型训练，结果绘制四个模块，最终由主函数调用它们来完成我们需要的全部功能。这些模块的具体实现分别为 config.py，models.py, training.py, graphplot.py 和 main.py，接下来我们逐个介绍。

3.1 初始参数设置

首先是超参数设置模块。在 config.py 中，我们使用 python 自带的字典结构来存储设备，网络，训练以及物理问题相关的超参数，方便之后的调试和修改。以下是它的伪代码，具体 python 代码见附件。

Algorithm 1 存储超参数

变量 device 指示计算设备为 `cuda`，若不可用则指向 `cpu`

网络参数：

输入维度 $= 1$

输出维度 $= 1$

隐藏层宽度 $= 18$

网络深度（隐藏层数）$= 6$

激活函数 $=$ tanh

正态分布方差 $= 1.0$

L2 正则化系数 $= 1 \times 10^{-7}$

训练参数：

学习率 $= 1 \times 10^{-4}$

最大训练轮数 $= 40000$

边界损失权重集合 $\lambda_b = \{10, 300, 10000\}$

每个区域采样点数 $= 100$

问题物理参数：

Case 1: Pe $= 0.01$, Da $= 0.01$ （扩散主导）

Case 2: Pe $= 20.0$, Da $= 0.01$ （对流主导）

Case 3: Pe $= 0.01$, Da $= 60.0$ （反应主导）

3.2 神经网络模型

第二步构建网络，该模块实现基本的 MLP 模型和初始化。它具有 6 层全连接网络，其中隐藏层宽度为 18。并且该神经网络满足：

1. 使用标准正态分布初始化参数；
2. 除了输出层以外，使用 tanh 激活函数；
3. 使用 L2 正则化，系数为 1e-7。

需要说明的是，除了标准正态分布以外，我后来在求解的过程中尝试了些别的初始化分布，比如常数点分布，和均匀分布。另外，对于 L2 正则化的实现，因为 pytorch 的优化器如“optim.Adam”方法中设置了 weight_decay 参数，传入参数给它可以帮助我们直接对各层进行对应参数的 L2 正则化，所以我没有显式地将正则化的 loss 加入到总的 loss 计算中；

作为 PDE 求解器，该网络通过自动微分将物理方程约束嵌入损失函数，实现无需监督数据的物理规律学习。代码支持 GPU 加速并保留计算图以进行高阶导数计算。算法的代码结构如下所示：

Algorithm 2 PINN 神经网络模型初始化 (models.py)

输入：接受 config.py 文件，得到输入维度 d_{in}，输出维度 d_{out}，隐藏层宽度 w，深度 L，激活函数 σ，正态分布初始化的标准差 σ_w

网络层列表 layers 初始化为空列表

添加输入层：nn.Linear(d_{in}, w)

for $\ell = 1$ to $L - 1$ **do**

    添加隐藏层：nn.Linear(w, w)

end for

添加输出层：nn.Linear(w, d_{out})

for 每一层 layer **do**

    权重初始化：nn.init.Normal_(layer.weight, mean = 0, std =σ_w)(可以换成 uniform_)

    偏置初始化：nn.init.Normal_(layer.bias, mean = 0, std = σ_w)(可以换成 uniform_)

end for

激活函数设为 σ （当前仅支持 tanh）

function FORWARD(x)

    for 除输出层以外，每一层 layer do **do**

        $x \leftarrow \sigma(\text{layer}(x))$

    end for

    输出层不加激活：$x \leftarrow \text{layer}(x)$

    return x

end function

3.3 模型训练和损失函数计算

接下来是主要的训练环节。首先需要给出计算 loss 的函数，我们定义损失函数为

$$\mathcal{L} = \mathcal{L}_{int} + \lambda_b \mathcal{L}_{bc} \tag{3}$$

其中：

$$\mathcal{L}_{int} = \frac{1}{N} \sum_{i=1}^{N} |u''(x_i) - Pe \cdot u'(x_i) + Da \cdot u(x_i)(1 - u(x_i))|^2 \tag{4}$$

$$\mathcal{L}_{bc} = |u(0)|^2 + |u(1) - 1|^2 \tag{5}$$

对应 loss 函数的计算我们就不给出伪代码，因为作业的文档中已经给出了提示。计算完 loss 函数后就可以进行正常的反向传播优化了，此时唯一需要注意的事情是在梯度下降之前记录内部的损失 $\mathcal{L}_{int}$ 和边界的损失 $\mathcal{L}_{bc}$，用以在训练结束后进行绘图。因为训练结束就代表已经得到了绘制 loss 下降过程的全部信息了，所以我将用于 loss 曲线绘制的 plot_losses 函数写在这里 training.py 中，在所有的训练 epoch 结束后直接使用保存的数据进行绘图就行，然后保存到给定文件夹中，注意我们绘制的是 y 轴为 log scale 的图像。伪代码如下：

Algorithm 3 训练 PINN 网络的主流程（train_model 函数）

Require: 输入参数 Pe, Da, 边界损失权重 λ_b

从 config.py 文件获得训练的最大 epoch 数，和训练时的采样点数 N。(epochs=40000, N=100)

实例化 PINN 模型 $\mathcal{N}_\theta$，并转移至设备（GPU/CPU）

实例化 Adam 优化器，设置学习率和 L2 正则化系数（1e-7）

生成训练点 $x_i \in [0,1]$ 共 N 个

初始化损失记录列表：R_{int}, R_{bc}, $\mathcal{L}_{\text{total}}$

for epoch = 1 **to** 最大的 epoch 数 **do**

清除优化器梯度

计算损失 $\mathcal{L}_{\text{total}} = R_{\text{int}} + \lambda_b R_{\text{bc}}$

反向传播

优化器执行一步更新

记录当前的 $R_{\text{int}}, R_{\text{bc}}, \mathcal{L}_{\text{total}}$

if epoch 可被 1000 整除 **then**

打印当前训练进度与损失值

end if

end for

if 提供了保存路径 **then**

保存损失曲线图像至指定目录

end if

return 训练后的模型和损失历史用于在 main 函数中保存

3.4 结果图绘制

该部分通过传入模型，模型采样点，对应的物理方程中的参数和图像保存路径，实现求解结果的绘图和保存图像的功能，我们省略它的伪代码，具体实现代码见附件 graphplot.py。

4 实验结果和初步分析

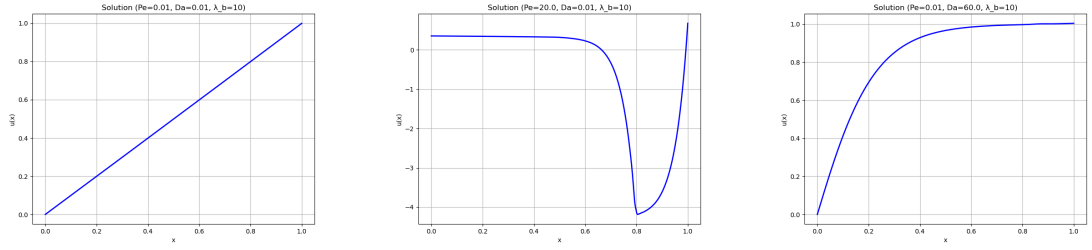
训练求解了三组不同参数值对应的方程，参数分别为

- $Pe = 0.01, Da = 0.01$（扩散占主导）
- $Pe = 20, Da = 0.01$（对流占主导）
- $Pe = 0.01, Da = 60$（反应占主导）.

接下来我将在这一节中展示我的实验结果。首先我给出按照作业要求的参数画出来的结果。

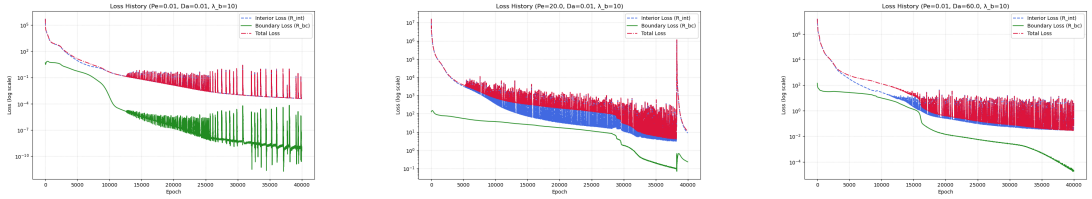
考虑到训练的随机性，我对每组方程参数，保持相同的训练超参数的情况下训练了两次，仅展示训练得比较好的一次。

4.1 第一组训练结果:



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)　(b) 对流主导 ($Pe = 20, Da = 0.01$)　(c) 反应主导 ($Pe = 0.01, Da = 60$)

图 1: 不同参数下的解曲线 ($\lambda_b = 10$)

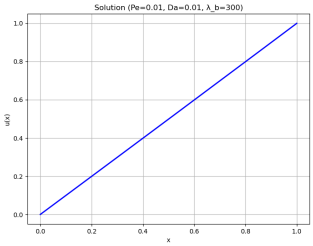


(a) 扩散主导　(b) 对流主导　(c) 反应主导

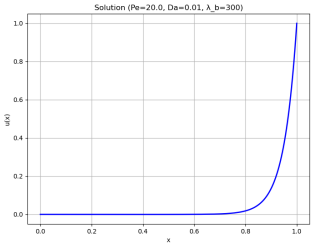
图 2: 不同参数下的 loss 下降曲线 ($\lambda_b = 10$)

可以看到, 在这种情况下, 对流主导的方程的 loss 函数难以下降到足够小的值, 函数图像并不可信。此时它真正的解函数图像还待我们进一步测试。接下来增大 λ_b，我们来观察会发生什么事情。

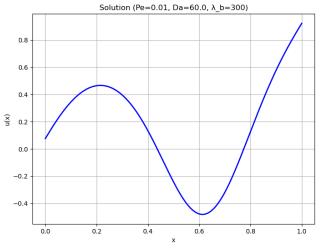
4.2 第二组训练结果：



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)



(b) 对流主导 ($Pe = 20, Da = 0.01$)

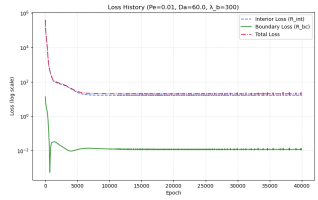


(c) 反应主导 ($Pe = 0.01, Da = 60$)

图 3: 不同参数下的解曲线 ($\lambda_b = 300$)

(a) 扩散主导

(b) 对流主导

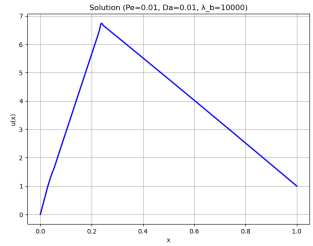


(c) 反应主导

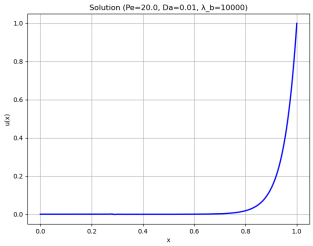
图 4: 不同参数下的 loss 下降曲线 ($\lambda_b = 300$)

可以看到此时对流主导的方程的求解得到了很好的收敛，但是反应主导的方程却产生了麻烦。我们再增大 $\lambda = 10000$ 看看发生了什么。

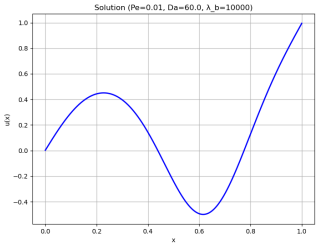
4.3 第三组训练结果：



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)

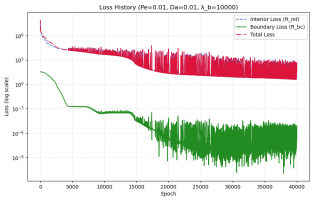


(b) 对流主导 ($Pe = 20, Da = 0.01$)

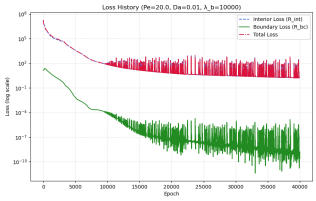


(c) 反应主导 ($Pe = 0.01, Da = 60$)

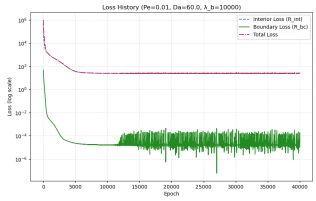
图 5: 不同参数下的解曲线 ($\lambda_b = 10000$)



(a) 扩散主导



(b) 对流主导



(c) 反应主导

图 6: 不同参数下的 loss 下降曲线 ($\lambda_b = 10000$)

我们可以看到，此时对流主导的方程得到了很好的收敛，但是扩散主导和反应主导的方程却都不收敛了。由此我们初步断定，也许在边界权重 λ_b 较大时，对流主导的方程的求解更容易收敛，在边界权重较小时，扩散主导的方程求解更容易收敛，而反应主导的方程可能是最难收敛的，因为它在两种 λ_b 的取值下都没有收敛。

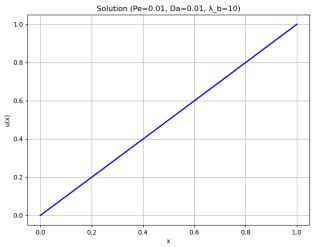
5 进一步的分析与讨论

我们初步得到该方程的解的函数图像的大该样子，扩散主导的方程的解形如线性函数，对流方程的解形如指数函数，而反应主导的方程的解形如对数函数。并且还有一些结论，如下

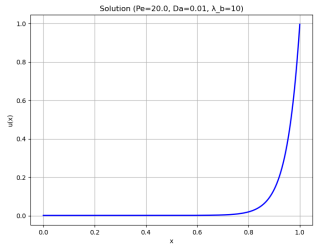
- 反应主导情况 ($Da = 60$) 最难求解
- 边界条件权重 λ_b 需要根据不同情况调整:
  - 扩散主导: $\lambda_b = 10$ 足够
  - 对流主导: 需要更高 λ_b（如 300）

但是我最开始写代码的时候，忘记了加上 l2 正则化的条件，阴差阳错得到了一个较差的结果，但是它也印证了我们反应主导的方程 (Pe=0.01, Da=60) 是最难求解的猜想。它的结果为

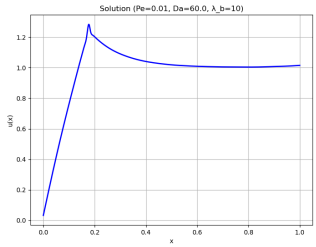
5.1 第一组训练结果:



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)

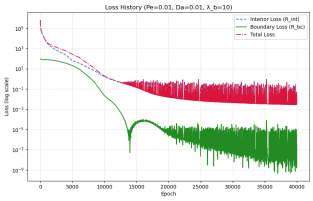


(b) 对流主导 ($Pe = 20, Da = 0.01$)

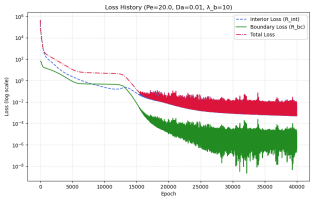


(c) 反应主导 ($Pe = 0.01, Da = 60$)

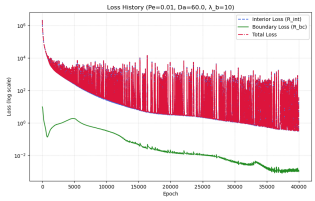
图 7: 不同参数下的解曲线 ($\lambda_b = 10$)



(a) 扩散主导



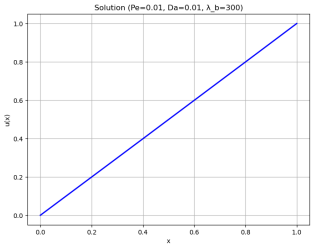
(b) 对流主导



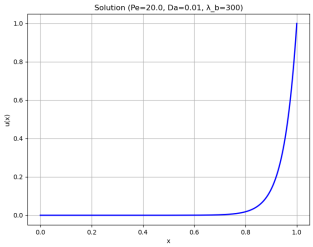
(c) 反应主导

图 8: 不同参数下的 loss 下降曲线 ($\lambda_b = 10$)

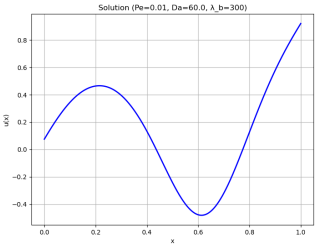
5.2 第二组训练结果：



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)

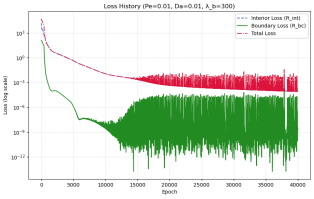


(b) 对流主导 ($Pe = 20, Da = 0.01$)



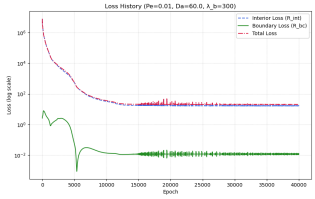
(c) 反应主导 ($Pe = 0.01, Da = 60$)

图 9: 不同参数下的解曲线 ($\lambda_b = 300$)



(a) 扩散主导

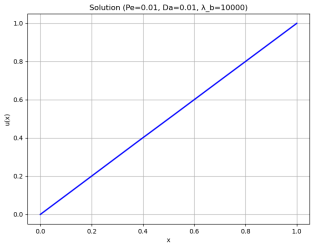
(b) 对流主导



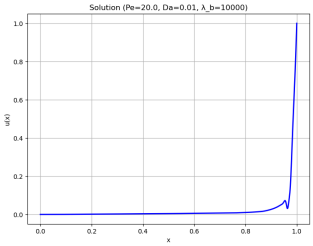
(c) 反应主导

图 10: 不同参数下的 loss 下降曲线 ($\lambda_b = 300$)

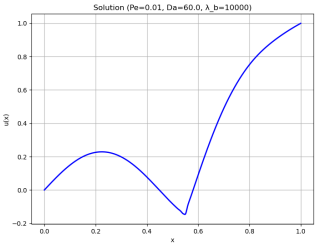
5.3 第三组训练结果：



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)



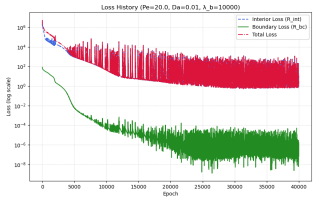
(b) 对流主导 ($Pe = 20, Da = 0.01$)



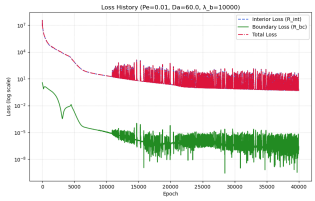
(c) 反应主导 ($Pe = 0.01, Da = 60$)

图 11: 不同参数下的解曲线 ($\lambda_b = 10000$)

(a) 扩散主导



(b) 对流主导

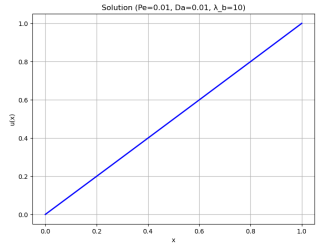


(c) 反应主导

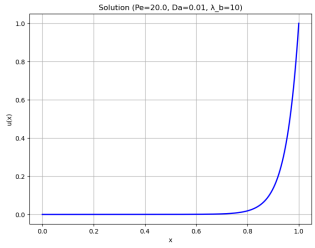
图 12: 不同参数下的 loss 下降曲线 ($\lambda_b = 10000$)

可以看到基本上所有的反应主导的方程都没有正确求解，这也说明了它可能是最难求解的。最后，我想给出一个我自己尝试的，对偏置项和权重项初始化的分布使用均匀分布的结果（仍然对每一层进行 l2 正则化），此时我发现这个方程在不同的参数下，不同的 λ_b 下都得到了很好的收敛。它们的代码和最初的代码唯一的区别仅仅是在 models.py 中将初始化处的 nn.init.normal_ 方法改为使用 nn.init.uniform_ 方法。

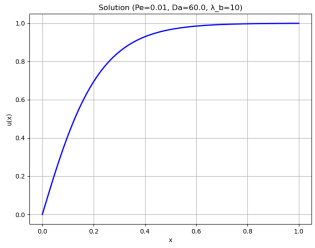
5.4 第一组训练结果:



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)

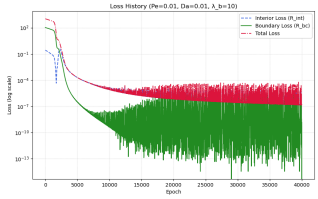


(b) 对流主导 ($Pe = 20, Da = 0.01$)

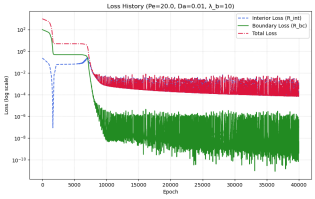


(c) 反应主导 ($Pe = 0.01, Da = 60$)

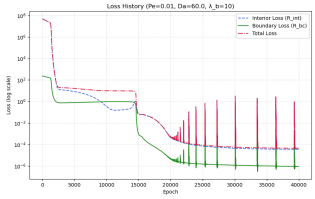
图 13: 不同参数下的解曲线 ($\lambda_b = 10$)



(a) 扩散主导



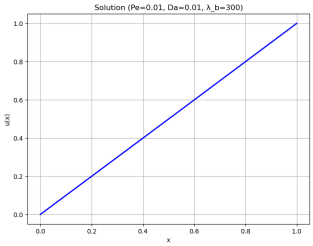
(b) 对流主导



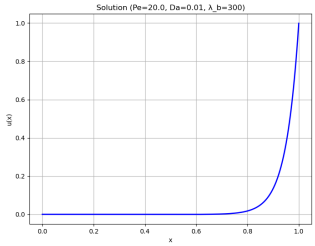
(c) 反应主导

图 14: 不同参数下的 loss 下降曲线 ($\lambda_b = 10$)

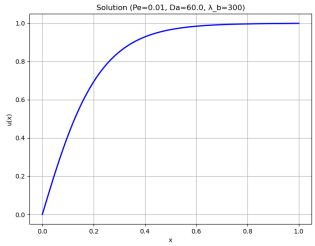
5.5 第二组训练结果：



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)

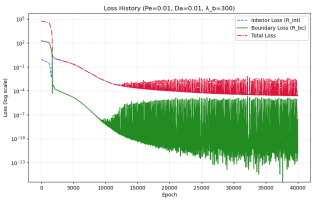


(b) 对流主导 ($Pe = 20, Da = 0.01$)



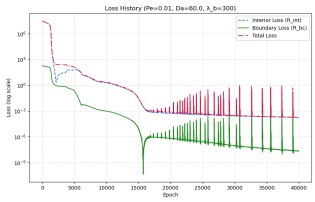
(c) 反应主导 ($Pe = 0.01, Da = 60$)

图 15: 不同参数下的解曲线 ($\lambda_b = 300$)



(a) 扩散主导

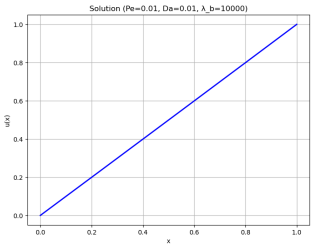
(b) 对流主导



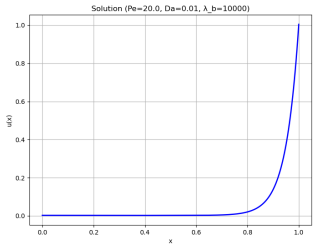
(c) 反应主导

图 16: 不同参数下的 loss 下降曲线 ($\lambda_b = 300$)

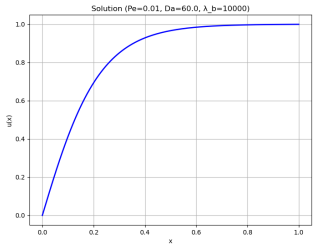
5.6 第三组训练结果：



(a) 扩散主导 ($Pe = 0.01, Da = 0.01$)

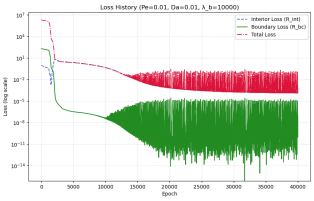


(b) 对流主导 ($Pe = 20, Da = 0.01$)



(c) 反应主导 ($Pe = 0.01, Da = 60$)

图 17: 不同参数下的解曲线 ($\lambda_b = 10000$)



(a) 扩散主导

(b) 对流主导

(c) 反应主导

图 18: 不同参数下的 loss 下降曲线 ($\lambda_b = 10000$)

可以看到所有情况下都得到了很好的收敛，甚至最终 loss 函数的数量级到达了 1e-7 的程度，这相比于使用正态分布进行神经网络参数初始化的收敛性，提升十分明显。

6 总结

我们使用 PINN 网络对不同参数的给定方程进行了求解，得出了使用正态分布初始化的神经网络学习该方程的解的一系列过程，最后我们成功构建了求解复杂 PDE 的 PINN 模型，验证了不同物理参数对解的影响，掌握了超参数调整对边界条件满足的重要性，并且发现改变不同的初始化分布对神经网络的训练可能会有显著的效果。