基于 Python 对云地间闪电先导模型的模拟

2022 级电磁学小论文

作者　何业天　PB22000210[1]

[1] *School of the Gifted Young ,USTC*

Submitted: 2023 年 6 月 23 日

摘要

闪电是生活中奇妙的物理现象，其看似变幻莫测但实则有迹可循。在观察到闪电的照片，不难发现其大多呈现的是枝状，本文便是试图利用物理知识与统计知识来对先导闪电形状的形成进行分析，发掘其背后的机制与影响因素。为了更好地观察闪电生成机理，我利用了介电击穿模型（Dielectric Breakdown Model，DBM）作为计算机模拟的理论基础，使用 Python 语言进行编码，通过实时更新的像素界面展现闪电一步步的生成过程。为了提高算法效率，通过运用超松弛迭代法来优化偏微分方程的计算。后续进一步利用深度学习 PINN 模型来对偏微分方程求解进行探究，并与传统数值方法进行比较。

关键字： 枝状闪电，DBM 模型，Python，神经网络

目录

1 引言

1.1 对枝状闪电的模拟背景

从古至今，闪电都是一个令人惊叹而又生畏的自然现象, 中国龙的形体也与闪电类似，象征着古人对巨大自然力量的崇拜。闪电其实也是有分类的，在有记录的闪电中有百分之七十五是云间放电（云对云闪电），百分之二十五为云地放电（云对地闪电）。在本篇论文中，我将选择云地间闪电作为研究对象，主要是因为它们对人们的生命财产有极大的威胁性，且相比云间放电来说需考虑的影响因素较少。

枝状闪电是闪电最常见的形式, 其蕴含的分形结构很好的对应了 DBM 模型。DBM 模型是基于 DLA 模型的电化学拓展模型——拉普拉斯生长 (Laplace growth) 模型, 由 Niemeyer, Pietronero 和 Weismann 提出的宏观数学模型 [1]。DBM 模型利用特定的计算公式和拉普拉斯方程来决定下一个击穿（生长）点的概率，相比于由随机游走进行模拟的扩散受限凝聚模型（Diffusion-limited Aggregation,DLA），是更加适用于电学问题的模型。

云地间的闪电无疑会给人们的人身与财产安全带来潜在的危害，为了将本次论文与现实问题挂钩，我于是在计算机模拟中加入了简易的避雷针模型来观测其对吸引闪电的作用，并在电磁学方面给出解释。

1.2 计算机模拟的硬件软件背景

正如标题所示，本次实验中的计算机模拟主要由 Python 实现，使用 Python 主要有一下三个原因:

1. Python 的 Class 类结构利于去构造一个会被很多函数引用的数据类型，在实验中，像素的数据、像素的生成都需要类中的数据共享，可以简便参数传递。

2. Python 具有很丰富的库来引用。比如在本实验中引用的 numpy,random,Tkinter,matplotlib 和 torch 等库，都提供了便于使用的函数来操作。其中 Tkinter 是经典的建造交互式界面的库，为了更好的体现出一步步的变化过程，采用了其的每次循坏的刷新机制。

3. 为了更好地构建神经网络, 这里采用了 Pytorch 作为构建平台。Pytorch 基于 Python 编译，是目前深度学习运用最多也是最基础的库之一，里面的神经网络相关函数很全并且漏洞很少，故采用。并且其还可以与 cuda 关联，从而实现 GPU 并行计算的加速。

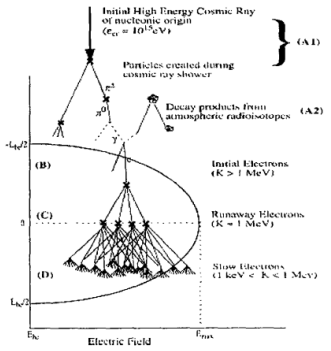

图 1: 逃逸击穿机制示意图

最后说一下我的软件版本信息与电脑信息。CPU 型号：Intel i7；GPU 型号：RTX3060；编译平台：Visual Studio Code；Python 版本：3.11.1；Pytorch 版本：'2.0.1+cu118'；CUDA 版本：12.1

2 云地间枝状分型闪电

2.1 闪电的形成

我们先来分析一下闪电的产生因素。在云层之间的上升气流把小水滴抬升，使它们产生过冷至摄氏-10 度到-20 度。这些过冷的小水珠会和冰晶相碰撞产生柔软的冰水混合物—软雹，这些碰撞会使冰晶带有正电、软雹带有负电。此时上升气流会继续将较轻的冰晶（带正电）抬升，软雹则因较重而落至云的中下层，进而造成云层上半部带正电，下半部带负电的电荷分离现象。此电荷分离过程使云间的电位差不断增加，直到足以释放而形成闪电。掉落中的冰晶和小水珠通过地球的自然电场会产生电极化的现象。碰撞中的冰粒会因静电感应而带电。[2]

近些年产生了新的起电机制理论——逃逸击穿。[3] 一些科学家 (McCarthy et al. 1992,GureVich et al. 1992, 1999) 发现在大气的各个地方会随机出现一些由宇宙射线以及地面放射性物质辐射所产生的高能粒子 (能量大于 1Mev), 这种高能粒子在一定的电场强度下被加速, 并与大气分子发生碰撞, 当还境电场超过某一临界值时 E_{be} (breakeven electric field), 碰撞后所产生的新粒子将不会存在能量消耗, 从而产生更多的高能粒子并继续与周围的大气分子发生碰撞, 直到发生电子雪崩为止, 从而使得闪电能够在低于常规击穿的电场环境下被激发 (见图1)。

2.2 闪电放电机制

对于我们主要将研究的云地间枝状闪电，其闪击总共有两步，分别是先导和回击，这两步的详细说明如下:

1. 阶梯先导：云里的电荷分布是这样的：在底部是较少的正电荷，在中下是较多的负电荷，在上部是较多的正电荷。此时，云地间电势显然更高，闪电由底部和中下部的放电开始。电子从上往下移动，这一放电由上向下呈阶梯状进行，每级阶梯的长度约为 50 米。两级阶梯间约有 50 微秒的时间间隔。平均速率为 1.5×10^5 米/秒，约为光速的两千分之一，半径约在 1 到 10 米，将传递约五库仑的电量至地面。

2. 回击：阶梯先导很接近地面时，就像接通了一根导线，强大的电流以极快的速度由地面沿着阶梯先导流至云层，这一个过程称为回击，约需 70 微秒的时间，约为光速的三分之一至十分之一。这个过程是闪电现形的主要过程，也是我们将在后续模拟中模拟的对象。倘若经过这样一次闪击后，云层还有足够电量，则会重复第二次阶梯先导。

而我们将要研究的便是闪电的先导路径，其的轨迹则主要由后续介绍的 DBM 模型预测。

3 DBM 模型背景

3.1 Monte Carlo 方法

Monte Carlo 方法起源于二战时期的原子弹设计研究，其是采用随机数于各种物理计算和模拟实验，以类似于赌博中掷骰子的方法来随机决定其中某个单独事件的结果，但最终的结果一定是符合某种数学性质或物理特征的 [4]。后续我们将研究的 DBM 便是基于 Monte Carlo 方法的。

3.2 DLA 模型描述

3.2.1 非平衡生长

生长是生物运动的一个重要形式，人类为了将这些生长模型进行归类与分析，于是建立起了归纳了几个重要因素的生长模型，来简化自然系中的生长机制，进行计算机模拟。大部的生长模型都属于非平衡生长，因为非局域模型中，已形成集团的整体都会对该处的生长概率有影响。无论哪类模型，最本质的特点是生长与历史有关，即最终的集团依赖于以前的集团位型，这意味着某个给定时间的位形与此前的位形有关。对于这种过程，不可能像平衡态那样简单地写出配分函数。[5]

3.2.2 DLA 模型基本概念

由 T.A. Witten 和 L.M. Sander 提出的 DLA 模型 [6] 是最经典最基础的非平衡生长模型之一，其概念

图 2: 3600 个粒子的聚集模型

简单，易于用计算机进行模拟，也是本文主要运用的 DBM 模型的基础。DLA 模型的模拟规则其实就是其名字中三个名词代表的意思：

1. 扩散：首先确定一个中心粒子作为种子，然后一个个放入自由粒子，自由粒子做 Brown 运动，也就是用随机行走问题进行模拟。

2. 受限：在距原点足够远的圆周界处释放的粒子，到大于起始圆的更远处（如 2-3 倍的半径处）或干脆走到点阵边界，这时认为粒子走了一条无用的轨迹，取消该粒子，把它重新放回区域。

3. 聚集：粒子走到种子的最近邻位置与种子相碰，这时让粒子粘结到种子上不再运动，进行如是不断的粘黏，从而形成一个聚集集团。

DLA 集团簇一只是分形研究的重要对象。其分形维数 D 将粒子数 n 与簇的大小 r 联系起来:$n = r^D$。在二维模型下，D 约为 1.71。[7] 然而，需要注意的是 DLA 模型是一张非平衡生长模型，虽然它产生了分形结构，但这种分形结构的性质只在无限维空间中较为稳定，而在小空间内会产生扰动。例如在无限大的二维平面内，DLA 通常有四个或五个大分支，它们或多或少是稳定的；在小的长度尺度上，分支机构将在不稳定的生存的永无止境的恶性循环中竞争。这种 DLA 增长的图景导致了“分支增长模型（branched growth model)”，其中分支之间的竞争——在所有长度尺度上——被表示为一个动力系统。然后，整个集群动力学被表示为同时运行的一大群耦合动力系统。这种方法允许对所有维度的集群动力学和分形属性进行近似但非常详细的解决方案。分支增长模型的 D 结果与数值结果非常一致，尤其是在高维度上。这种方法还允许计算多重分形属性；这些结果也与模拟一致。这种方法特别适用于计算簇的拓扑自相似性，涉及的知识需要较高的科目基础，这里就不过多赘述了。

最后提一下近年的新研究 Hastings-Levitov 方法。[8] 基于 Hele-Shaw 问题拥有的自然共形表示，如图3所

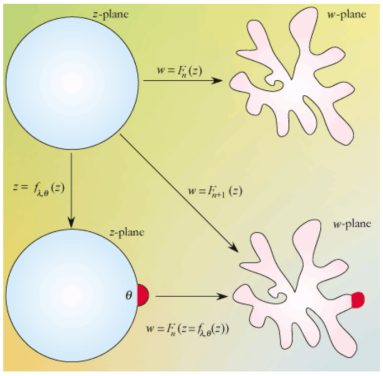

图 3: Hastings-Levitov 迭代共形映射

示，黎曼映射定理带来的一组共形映射，将复 z 平面中的单位圆映射到物理 w 平面中的簇表面，而 z 空间中圆周上两点之间的角距离与物理空间中这两个点的图像连接的弧上的总生长概率成正比。Hastings-Levitov 算法让我们不仅可以重现 DLA，还可以重现更通用的模型。如果随机选择 z 空间（物理空间的原像）中的生长位置 q，我们将得到 DLA。但其他选择也是可能的。

上述的分型结构解释实际上也可以移接到后续的 DBM 模型中，从而体现出枝状闪电的分型性质，但笔者的数学能力有限，故就不再进行下一步的分型分析以及推导了。

3.3 DBM 模型数学背景

3.3.1 有限差分法

在数学中，有限差分法（finite-difference methods，FDM），是一种微分数值方法，是通过有限差分來近似导数，从而寻求微分方程的近似解。[9] 以下是简单的推导过程：由泰勒展开可以得到：

$$f(x_0+h)=f(x_0)+f'(x_0)h+R_1(x) \tag{1}$$

其中 $R_1(x)$ 是一阶近似的余值，由式1，假设 $x_0=a$ 和 $R_1(x)$ 足够小时，便可以忽略掉可以得到:

$$f'(a)\approx\frac{f(a+h)-f(a)}{h} \tag{2}$$

这样的方法虽说有误差，但也是最常用的数值求解微分方程的方式之一，后续对 Laplace 方程的计算便是基于该计算方式。

3.3.2 Laplace 方程

泊松方程是描述电场电势与自由电子关系的偏微分方程，具体如下：

$$\left(\frac{\partial^2}{\partial x^2}+\frac{\partial^2}{\partial y^2}+\frac{\partial^2}{\partial z^2}\right)\varphi(x,y,z)=f(x,y,z) \tag{3}$$

其中的 $\varphi(x,y,z)$ 是电势分布函数，$f(x,y,z)$ 是自由电子分布函数。而当空间中无自由电子时，Possion 方程便化为了 Laplace 方程，下面我们考虑的是二维平面间的模拟，故采用二维的 Laplace 方程如下：

$$\nabla^2\varphi=0 \tag{4}$$

由有限差分法，我们可以尝试对 Laplace 方程进行数值模拟，可得下面的式子：

$$\begin{aligned}\frac{\partial^2u(x_i,y_j)}{\partial x^2}=&\frac{1}{h_1^2}\left[u(x_{i+1},y_j)-2u(x_i,y_j)+u(x_{i-1},y_j)\right]\\&-\frac{1}{12}h_1^2\frac{\partial^4u(\xi_i,y_j)}{\partial x^4},\xi_i\in(x_{i-1},x_{i+1})\\\frac{\partial^2u(x_i,y_j)}{\partial y^2}=&\frac{1}{h_2^2}\left[u(x_i,y_{j+1})-2u(x_i,y_j)+u(x_i,y_{j-1})\right]\\&-\frac{1}{12}h_2^2\frac{\partial^4u(x_i,\eta_j)}{\partial y^4},\eta_j\in(y_{j-1},y_{j+1})\end{aligned} \tag{5}$$

考虑离散化的网格，我们不妨令步长为 1，加上边界条件后可化简得：

$$4u_{ij}-\left(u_{(i+1)j}+u_{(i-1)j}+u_{i(j+1)}+u_{i(j-1)}\right)=0 \tag{6}$$

这便是后续我们求解方程时的核心公式。

3.3.3 第一类边界条件

对于 Laplace 方程的第一类边界条件又名 Dirichlet 边界条件，只有当边界条件存在时方程才有唯一解，这也是后续进行神经网络模拟的关键之一。具体的概念如下：

在空间 R^n 中的某一区域 Ω 的边界 $\partial\Omega$ 上给定一个连续函数 g，求解这样的一个函数 $u=u(x_1,\cdots,x_n)$ 使得它在 Ω 内满足方程 (1.1)(或 (1.2))，在 $\bar{\Omega}$ 上连续，并且在 $\partial\Omega$ 上满足:

$$u|_{\partial\Omega}=g \tag{7}$$

3.3.4 Laplace 方程求解

对于差分法处理过的 Laplace 方程，其实际变成了一个求解线性方程的问题。对于网格上的全部 n 个点我们可以列出一组 n 项的线性方程组，该线性方程组的增广矩阵设为 A，其中的元素设为 a_ij。该增广矩阵有这样的性质：当 i=j，$a_ij=4$；当是该点旁边的元素时，$a_ij=-1$；其他情况，$a_ij=0$。在求得线性方程的解后，我们就得到了电势分布。然而已知增广矩阵后如何去求解是一个问题，直接用 Guass 消元法的话过于笨重，速度过慢，因此我们需要用一种全新的方法进行改善。

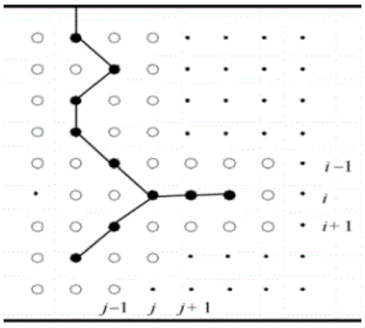

图 4: 平板放电时的 DBM 模拟示意图

上述的线性放程是一个很大的稀疏矩阵，如果采用 Guass 消元法，会消耗大量的算力和时间。因此这里采用了 Gauss-Seidel 迭代法来提高效率：

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) \quad (8)$$

进一步的，基于 Gauss-Seidel 迭代法的超松弛迭代法 (Successive Over Relaxation，SOR) [10] 可以更好的提高效率。这里利用并行计算原理有以下公式：

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) \quad (9)$$

其中 ω 为加速收敛因子。其核心思想为将 Gauss-Seidel 迭代求解出的新的解向量元素与上一个解向量中相对应的元素，做加权平均组合成一个全新的解向量的元素，这个全新的解向量即为超松弛迭代法的解。

4 DBM 模型的计算机模拟

4.1 DBM 机制介绍

如在引言中说的，DBM 是一种与电学息息相关的生长模型，与 DLA 模型一样在大型方格网上进行模拟。其比 DLA 更有指向性，虽然仍依赖于随机算法，但不再需要随机行走来聚集粒子，而是采用电击穿理论决定下一步击穿点的概率，从而进行生长。

在模拟场中，各点的电位关系满足 Possion 方程。然而在 DBM 的模型中，在每一步击穿过程间每个点格的电子并非自由电子，可视为整个空间中并无自由电子，于是 Possion 方程转化为 Laplace 方程4。接下来再分析一下击穿过程。基于 DBM 模型，我们有一个概率分布的公式如下：

$$p(i,j \to i,j') = \frac{\mathrm{E}_{ij,ij}{}^{\eta}}{\sum \mathrm{E}_{ij,ij}{}^{\eta}} \quad (10)$$

算式4中体现了任意两点之间的击穿概率的计算，在我们的 DBM 网格结构中，我们只考虑与对象点相连的 8 个点，其中 4 个点步长为 1 (上下左右)，4 个点步长为 $\sqrt{2}$ (左上左下右上右下)。其中的 η 为分形参数，根据生活经验我们取 $\eta = 3$。

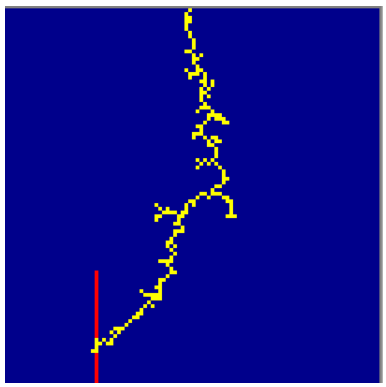

图 5: DBM 模型计算模拟图（避雷针引雷成功）

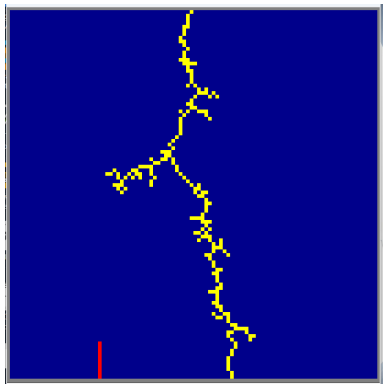

图 6: DBM 模型计算模拟图（避雷针引雷失败）

在后续的研究中，Mansell 于 2002 年将 DBM 模型与前文中提到的逃逸击穿理论结合。[11] 于雷暴起电、放电过程的数值模拟中。闪电的初始击穿采用了逃逸击穿理论，阈值随高度而变化；闪电一旦触发，将同时产生正. 负两种极性的先导；闪电传输由通道尖端与周围环境之间的电位差所决定，只要超过选定的阈值电场 $\mathrm{E}_{\mathrm{crit}}$，通道就可以延伸，对闪电每一步扩展、正负流光各自只扩展一个后继通道点，其从所有满足条件的后继通道点中随机选取一个，直到两个流光都无后继点时，一次闪电过程结束；通道每衍生一次，需计算通道自身对周围🄴境电场的影响，将通道看成是固定边界条件，并认为满足静电学理论，通过求解泊松方程重解周围环境电场；通道所携带的电荷由周围环境电位感应产生。在本实验中为进行适度的简化，还是采用传统的 DBM 模型进行模拟。

4.2 DBM 模拟具体实现

4.2.1 DBM_Data 与 DBMTkVisual 类设定

DBM 模拟程序的实现示意图已经全部展现在了附录，下面都是根据附录中的具体代码来进行说明。

首先在程序的最开始定义了 DBM_Data 类，该类的初始化主要是建立起要进行计算的各个二维矩阵：self.map（用于表示是否击穿，是否是避雷针）、self.U_cur（用于描述当前电势分布）、self.U_upd（用于描述下一次迭代时的电势分布）。并且设计了初始边界条件：1. 把云层与地面想象成两块大平面，通过相对电压分析，将地面电压设为 100MV，云层设为 0；2. 两块平行平面间的电势均匀分布，即从云层开始均匀递增；3. 将避雷针处的电压值与地面设为一致。

接着浅谈一下交互界面的设计，这里定义了 DBMTkVisual 类。在初始化环节主要是初始化了各项参量，并且指定了窗口大小位置，设定了颜色分配：背景为蓝，闪电路径为黄，避雷针为宏。int_canvas() 函数主要是对画布进行了设置；plot_pixel() 函数则是对小像素点和大像素图进行了设置，并且使其可以显示。

4.2.2 DBM 击穿机制的代码实现

在 DBM_Data 类中定义了三个函数，这三个函数便是实现 DBM 机制的核心。

首先是 U_matrix_upd() 函数，这个函数便是对 Laplace 方程的求解，这里我们采用超松弛迭代法进行计算。迭代因子根据经验选取 $w = \frac{2}{1+\sin\left(\frac{\pi}{p}\right)}$。这里没有把稀疏矩阵表示出来再去求解，因为该稀疏矩阵很大，不好创立。这里采用的是从上至下一行行的根据四周格点 self.U_cur 数值来修改该点 self.U_upd 值，从而不断逼近于数值解。因为上一行的数值是已经计算过一边的 self.U_upd 值，故可以直接采用。最后把误差定为小于 10^{-5}，定为得到数值模拟结果。

第二个是 Breakdown_Point() 函数，主要用于在已知电势分布后计算下一点的击穿概率。第一步是计算整个网格的概率分母，这里的数值是每个击穿点周围 8 个击穿点的概率之和，遍历整个矩阵来求和。之后就是对每一个点进行概率计算，我们还是以每个击穿点作为对象，即便有些其周围的点会被重复计算几遍，但不影响结果。在随机选择击穿点时采用了之前提到的 Monte Carlo 方法，将 0 到 1 分划为各个概率点分出的区域，然后用 numpy 的 searchsorted () 函数和 random.choice() 函数随机投掷点，其在的范围便对应了击穿点。

第三个是 is_to_boundary() 函数，主要用于对边界条件的判定。每一次循坏要判定一次，闪电路径是否已经到达了地面或者避雷针，如果是的话就停止循环。

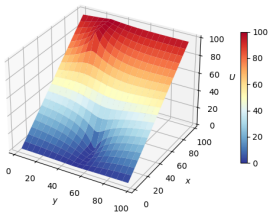

图 7: DBM 模拟过程中的电势图

4.2.3 其他函数

为了便于用户操作，这里设计了 int_input_consider_default 函数用于更好的读取输入信息，以及 player_guide_interface 函数来进行初始化数据的设定与录入。

为了更好的展现出闪电的物理特征，创建的 U_show 函数可以展现出电压在 x—y 平面上的立体示意图，从而能更好的看出闪电路径受 Laplace 方程的影响，同时利用 FileSave() 函数，将不同时间的闪电电压分布图存储进文件，可以更好的看出整个过程中，电压分布的变化。

最后是主函数部分。主函数主要是先录入读取的数据，并初始化 DBM_Data 与 DBMTkVisual 类，之后不断的对放电机制中的三个函数进行循环，直到循环结束，显示电压图。

4.2.4 模拟结果分析

图5与图6展现了闪电路径模拟结果。首先说明一下闪电下降的趋势，由于边界条件的设定，电势向下不断增大，因此和被击穿点的电势差越大，因此更容易被击穿，并且分型参数 η 越大，其电势差更大，分支更小，更像一条直线蔓延向下。在说一下避雷针的问题，可以看到避雷针失败的情况是因为避雷针高度太矮，实际上由于避雷针本身电势大，由 Laplace 方程它会增大自己周围的电势，从而起到吸引闪电的作用，而这种作用随着避雷针的高度增加而越显著。这也解释了为什么避雷针大多建在高楼之上。

5 神经网络训练

5.1 PINN 网络模型

基于物理信息的神经网络（Physics-informed Neural Network，简称 PINN），是一类用于解决有监督学习任务的神经网络，同时尊重由一般非线性偏微分方程描述的任何给定的物理规律。PINN 的优势是在于泛化预测，其不需要很多的数据集，便可以基于物理信息

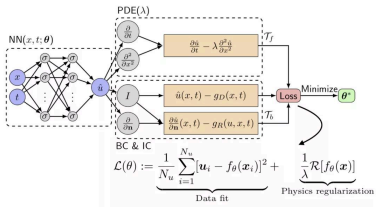

图 8: PINN 示意图

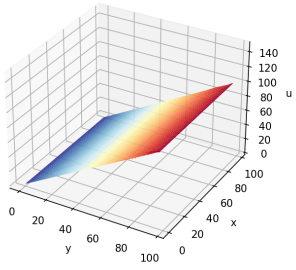

图 9: PINN 模拟 Laplace 方程的解

给出偏微分方程的近似解，但该近似解的精确度并不是很高。这里不去细讲神经网络的构成原理了，原理图如图**??**所示。其比较特别的点是利用对于边界条件以及函数内点取值的均分根差（MSE）来进行反向传播，从而实现梯度下降。

5.2 PINN 代码实现

主要讲一下模型里面的几个重要设置。首先对于网络搭建，这里设立了两个隐藏层来提高精度，在激励函数方面则选择了 tanh() 函数，这些都是 PINN 模型这种比较常见的设定了。下一步在 Laplace() 函数中表示了 Laplace 方程的求二阶偏导算式。接着设定了误差函数为均方根误差函数 torch.nn.MSELoss()，并且设定优化器为 torch.optim.Adam，该优化器可以自动调参比较便捷，然后这里选取的学习率为 10^{-3}，这个值的取定是在不断的尝试中选择了一个下降较快且浮动不大的适当学习率。再下一步便是边界条件的设定以及误差的计算，这个过程便是深度学习网络不断改进拟合的过程了。

代码中的运算基本都是在 GPU 上进行的，通过 cuda 进行并行计算，算力的提升进步显著。然而相比于传统的数值模拟 SOR 算法，其的精度与速度皆显逊色，因此在整个模型的实现中还是采用 SOR 算法。

6 总结

本次论文的写作涉及的面很广，从物理机制到数学分型，从代码实现到神经网络，这些种种领域的交叉也展现了处理我们生活中一些常见问题的新的思路。这一次的实验也有不少遗憾，比如没有用根据逃逸理论优化后的 DBM 模型在三维空间中进行模拟，比如没能搭建一个性能更强的深度学习网络来求解偏微分方程。但是这次论文写作过程中我获得的知识、能力、兴趣也是不少的，希望这科研路上的第一步能指引我继续前进的方向。

参考文献

[1] L. Niemeyer, L. Pietronero, and H. J. Wiesmann, “Fractal dimension of dielectric breakdown,” *Physical Review Letters*, vol. 52, no. 12, p. 1033, 1984.

[2] Wikipedia contributors, “Lightning — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Lightning&oldid=1157698527, 2023. [Online; accessed 8-June-2023].

[3] A. Gurevich, G. Milikh, and R. Roussel-Dupre, “Runaway electron mechanism of air breakdown and preconditioning during a thunderstorm,” *Physics Letters A*, vol. 165, no. 5-6, pp. 463–468, 1992.

[4] R. Y. Rubinstein and D. P. Kroese, *Simulation and the Monte Carlo method*. John Wiley & Sons, 2016.

[5] 丁泽军, *计算物理第四版*. Hefei,China: 中国科学技术大学物理系.

[6] T. A. Witten Jr and L. M. Sander, “Diffusion-limited aggregation, a kinetic critical phenomenon,” *Physical review letters*, vol. 47, no. 19, p. 1400, 1981.

[7] S. Havlin and A. Bunde, “Fractals and disordered systems,” 1991.

[8] M. B. Hastings, *A renormalization approach to diffusion-limited aggregation*. PhD thesis, Massachusetts Institute of Technology, 1997.

[9] 维基百科, “有限差分法 — 维基百科, 自由的百科全书,” 2023.

[10] Wikipedia contributors, “Successive over-relaxation — Wikipedia, the free encyclopedia.” https://en.wikipedia.org/w/index.php?title=Successive_over-relaxation&oldid=1144407691, 2023. [Online; accessed 8-June-2023].

[11] E. R. Mansell, D. R. MacGorman, C. L. Ziegler, and J. M. Straka, “Simulated three-dimensional

branched lightning in a numerical thunderstorm model," *Journal of Geophysical Research: Atmospheres*, vol. 107, no. D9, pp. ACL–2, 2002.

附录

下面是 DBM 实现代码:

```
import random
import tkinter
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import sys
import os
import torch
import torch.nn as nn
import time
from numpy.linalg import pinv
from matplotlib import cm

class DBM_Data:
    is_BD = 1
    not_BD = 0
    rod = 2

    def __init__(self, length, width, root_pos,use_lightning_rod,
        lightning_rod_pos,lightning_rod_height):
        # Setting two more line will make it easy to enable the periodic map
        self.column=length+2
        self.row=width+2
        self.map = np.zeros((self.row, self.column), dtype=int)
        self.map[1,root_pos] =DBM_Data.is_BD
        self.use_lightning_rod=use_lightning_rod
        self.lightning_rod_pos=lightning_rod_pos
        self.lightning_rod_height=lightning_rod_height
        self.U_cur=np.zeros((self.row, self.column),dtype=float)
        self.U_upd=np.zeros((self.row, self.column),dtype=float)
        for i in range(1,self.row-1):
            for j in range(1,self.column-1):
                self.U_cur[i,j]=100*((i-1)/(self.row-3))
        if self.use_lightning_rod:
            for i in range(lightning_rod_height):
                self.U_cur[self.row-2-i,lightning_rod_pos]=100
                self.map[self.row-2-i,lightning_rod_pos]=DBM_Data.rod

    def U_matrix_upd(self):
        #Calculate the best relaxation factor
        SOR_w=2/(1+np.sin(2*np.pi/(self.row+self.column-2)))
        delta=1
        times=0
        self.U_upd=self.U_cur.copy()
```

```
        while delta>1e-5:
        delta=0
        times=times+1
        for i in range(2,self.row-2):
            for j in range(2,self.column-2):
                if self.map[i,j]==DBM_Data.not_BD:
                    self.U_upd[i,j]=self.U_cur[i,j]+SOR_w*\
                        (self.U_cur[i,j+1]+self.U_cur[i+1,j]+self.U_upd[i-1,j]\
                         +self.U_upd[i,j-1]-4*self.U_cur[i,j])/4
                    if abs(self.U_cur[i,j]-self.U_upd[i,j])>delta:
                        delta=abs(self.U_cur[i,j]-self.U_upd[i,j])
        self.U_cur=self.U_upd.copy()

def Breakdown_Point(self):
    frac_para=3
    point_probability=np.zeros((self.row, self.column),dtype=float)
    #Calculate the whole map's potential probability
    whole_p=0
    for i in range(1,self.row-1):
        for j in range(1,self.column-1):
            if self.map[i,j]==DBM_Data.is_BD:
                if self.map[i-1,j]==DBM_Data.is_BD:
                    p_1=0
                else:
                    p_1=(self.U_cur[i-1,j])**(frac_para)
                if self.map[i,j-1]==DBM_Data.is_BD:
                    p_2=0
                    else:
                    p_2=(self.U_cur[i,j-1])**(frac_para)
                if self.map[i-1,j-1]==DBM_Data.is_BD:
                    p_3=0
                else:
                    p_3=(self.U_cur[i-1,j-1]/np.sqrt(2))**(frac_para)
                if self.map[i+1,j]==DBM_Data.is_BD:
                    p_4=0
                else:
                    p_4=(self.U_cur[i+1,j])**(frac_para)
                if self.map[i,j+1]==DBM_Data.is_BD:
                    p_5=0
                else:
                    p_5=(self.U_cur[i,j+1])**(frac_para)
                if self.map[i+1,j+1]==DBM_Data.is_BD:
                    p_6=0
                else:
                    p_6=(self.U_cur[i+1,j+1]/np.sqrt(2))**(frac_para)
                if self.map[i-1,j+1]==DBM_Data.is_BD:
```

```
                    p_7=0
                else:
                    p_7=(self.U_cur[i-1,j+1]/np.sqrt(2))**(frac_para)
                if self.map[i+1,j-1]==DBM_Data.is_BD:
                    p_8=0
                else:
                    p_8=(self.U_cur[i+1,j-1]/np.sqrt(2))**(frac_para)
                whole_p=whole_p+p_1+p_2+p_3+p_4+p_5+p_6+p_7+p_8
#Calculate each point's breakdown probability
s_1=0;s_2=0;s_3=0;s_4=0;s_5=0;s_6=0;s_7=0;s_8=0
ran_choice=random.uniform(0,1)
for i in range(1,self.row-1):
    for j in range(1,self.column-1):
        if self.map[i,j]==DBM_Data.is_BD:
            if self.map[i-1,j]==DBM_Data.is_BD:
                point_probability[i-1,j]=0
            else:
                point_probability[i-1,j]=(self.U_cur[i-1,j])**(frac_para)/whole_p
            if self.map[i,j-1]==DBM_Data.is_BD:
                point_probability[i,j-1]=0
            else:
                point_probability[i,j-1]=(self.U_cur[i,j-1])**(frac_para)/whole_p
            if self.map[i-1,j-1]==DBM_Data.is_BD:
                point_probability[i-1,j-1]=0
            else:
                point_probability[i-1,j-1]=\
                  (self.U_cur[i-1,j-1]/np.sqrt(2))**(frac_para)/whole_p
            if self.map[i+1,j]==DBM_Data.is_BD:
                point_probability[i+1,j]=0
            else:
                point_probability[i+1,j]=(self.U_cur[i+1,j])**(frac_para)/whole_p
            if self.map[i,j+1]==DBM_Data.is_BD:
                point_probability[i,j+1]=0
            else:
                point_probability[i,j+1]=(self.U_cur[i,j+1])**(frac_para)/whole_p
            if self.map[i+1,j+1]==DBM_Data.is_BD:
                point_probability[i+1,j+1]=0
             else:
                point_probability[i+1,j+1]=\
                  (self.U_cur[i+1,j+1]/np.sqrt(2))**(frac_para)/whole_p
            if self.map[i-1,j+1]==DBM_Data.is_BD:
                point_probability[i-1,j+1]=0
            else:
                point_probability[i-1,j+1]=\
                  (self.U_cur[i-1,j+1]/np.sqrt(2))**(frac_para)/whole_p
            if self.map[i+1,j-1]==DBM_Data.is_BD:
                point_probability[i+1,j-1]=0
```

```
            else:
                point_probability[i+1,j-1]=\
                  (self.U_cur[i+1,j-1]/np.sqrt(2))**(frac_para)/whole_p
            s_1=s_8+point_probability[i-1,j]
            s_2=s_1+point_probability[i,j-1]
            s_3=s_2+point_probability[i-1,j-1]
            s_4=s_3+point_probability[i+1,j]
            s_5=s_4+point_probability[i,j+1]
            s_6=s_5+point_probability[i+1,j+1]
            s_7=s_6+point_probability[i-1,j+1]
            s_8=s_7+point_probability[i+1,j-1]
            #Decide which point would be broken down
            intervals=[0,s_1,s_2,s_3,s_4,s_5,s_6,s_7,s_8,1]
            interval_index=\
              np.searchsorted(intervals,ran_choice,side='left')-1
            if interval_index!=8:
              if interval_index==0:
                self.map[i-1,j]=DBM_Data.is_BD
                self.U_cur[i-1,j]=0
              if interval_index==1:
                self.map[i,j-1]=DBM_Data.is_BD
                self.U_cur[i,j-1]=0
              if interval_index==2:
                self.map[i-1,j-1]=DBM_Data.is_BD
                self.U_cur[i-1,j-1]=0
              if interval_index==3:
                self.map[i+1,j]=DBM_Data.is_BD
                self.U_cur[i+1,j]=0
              if interval_index==4:
                self.map[i,j+1]=DBM_Data.is_BD
                self.U_cur[i,j+1]=0
              if interval_index==5:
                self.map[i+1,j+1]=DBM_Data.is_BD
                self.U_cur[i+1,j+1]=0
              if interval_index==6:
                self.map[i-1,j+1]=DBM_Data.is_BD
                self.U_cur[i-1,j+1]=0
              if interval_index==7:
                self.map[i+1,j-1]=DBM_Data.is_BD
                self.U_cur[i+1,j-1]=0
              break
        else:
            continue
        break
```

```
        #If the flash reach the boundary,the system is over
    def is_to_boundary(self):
        is_to_boundary=False
        for i in range(1,self.column-1):
            if self.map[self.row-2,i]==1 :
                is_to_boundary=True
        for i in range(1,self.column-1):
            if self.map[i,1]==1 or self.map[i,self.column-2]==1:
                is_to_boundary=True
        if self.use_lightning_rod:
            for i in range(self.lightning_rod_height):
                if self.map[self.row-2-i,self.lightning_rod_pos]==DBM_Data.is_BD:
                    is_to_boundary=True
        return is_to_boundary

# Settings for the visual interface
class DBMTkVisual:
    kWindowZoomRatio = 0.9
    def __init__(self, array_data, DBM_length, DBM_width):
        self.data_row_size = len(array_data)
        self.data_column_size = len(array_data[0])
        self.array_data = array_data
        self.color_dict = {DBM_Data.not_BD:'darkblue',
          DBM_Data.is_BD:'yellow',DBM_Data.rod:"red"}
        self.root = tkinter.Tk()
        self.root.title('DBM for Flash')
        size_window_length = \
          self.root.winfo_screenwidth() * DBMTkVisual.kWindowZoomRatio
        size_window_width = \
          self.root.winfo_screenheight() * DBMTkVisual.kWindowZoomRatio
        # Calculate the single pixel size
        pixel_length = size_window_length / float(DBM_length + 2)
        pixel_width = size_window_width / float(DBM_width + 2)
        pixel_size = min(pixel_length, pixel_width)
        self.pixel_size = int(pixel_size)
        # Update the window size
        size_window_length = size_window_length * self.pixel_size / pixel_length
        size_window_width = size_window_width * self.pixel_size / pixel_width
        windows_position_info = \
          '%dx%d+%d+%d' %(size_window_length, size_window_width,
                          (self.root.winfo_screenwidth() - size_window_length)/2,
                          (self.root.winfo_screenheight() - size_window_width)/2)
        self.root.geometry(windows_position_info)
        self.canvas = tkinter.Canvas(self.root, bg = 'black')
        self.init_canvas()
        self.canvas.pack()
        self.canvas.config(width=size_window_length, height=size_window_width)
```

```
    def init_canvas(self):
        # A 2d list for saving every handle of 'canvas rectangle'
        self.grid_handle = list()
        self.canvas.create_rectangle(0, 0, self.pixel_size*(self.data_row_size),
                                     self.pixel_size*(self.data_column_size),
                                     fill='gray',
                                     outline='gray')
        # Plot every pixel in the canvas by 'for' cycle
        for row_num in range(1, self.data_row_size-1):
            self.grid_handle.insert(row_num-1,list())
            for column_num in range(1, self.data_column_size-1):
                self.grid_handle[row_num-1].insert(column_num-1,
                self.canvas.create_rectangle(
                  self.pixel_size*column_num,
                  self.pixel_size*row_num,
                  self.pixel_size*(column_num+1),
                  self.pixel_size*(row_num+1),
                  fill=self.color_dict[self.array_data[row_num][column_num]],
                  outline=self.color_dict[self.array_data[row_num][column_num]]))

    def plot_pixel(self):
        # Use itemconfig to change the color of every canvas rectangle
        # Plot every pixel in the canvas by 'for' cycle and update
        for row_num in range(1, self.data_row_size-1):
            for column_num in range(1, self.data_column_size-1):
                self.canvas.itemconfig(
                  self.grid_handle[row_num-1][column_num-1],
                  fill=self.color_dict[self.array_data[row_num][column_num]],
                  outline=self.color_dict[self.array_data[row_num][column_num]])
        self.canvas.update()

def int_input_consider_default(show_str, defalut_value):
    # integer input reads considering the default value
    input_str = str(input(show_str))
    if '' == input_str:
      is_int_num = False
    else:
      is_int_num = True
    for str_element in range(len(input_str)):
        is_int_num = is_int_num and (ord(input_str[str_element])>=ord('0') \
                                     and ord(input_str[str_element])<=ord('9'))
        if not is_int_num:
           break
        if is_int_num:
        get_value = int(input_str)
    else:
        get_value = defalut_value
```

```
        lightning_rod_height=\
                  int_input_consider_default('>>> ',lightning_rod_height)
    return get_value
#user commands to create the model
def player_guide_interface():
    while True:
        print("Do you want to show each step of the evolution?")
        print("Y:show the every step on the output window")
        print("N:just save the result into the data_file ")
        input_str_1 = str(input('>>> '))
        data_file_mode = False
        if ('N' == input_str_1) or ('n' == input_str_1):
            data_file_mode = True
        #defalut value set
        dbm_data_length=100
        dbm_data_width=100
        root_pos=50
        use_lightning_rod=True
        lightning_rod_pos=28
        lightning_rod_height=30
        print('   ')
        print(' DEFAULT PARAMETER LIST:')
        print('| 2D Data Array Width:      %d' % dbm_data_width)
        print('| 2D Data Array Length:     %d' % dbm_data_length)
        print('| Init Root Position:       %d' % root_pos)
        print('| Init Lightning rod position:      %d' % lightning_rod_pos)
        print('| Init Lightning rod height:        %d' % lightning_rod_height)
        print('Change the DEFAULT parameter?(Y/[N])')
        change_defult_parameter = str(input('>>> '))
        if ('Y' == change_defult_parameter) or
('y' == change_defult_parameter):
            print("Set the Canvas:")
            print("Please input the 2D data array width:")
            dbm_data_width = int_input_consider_default('>>> ', dbm_data_width)
            print("Please input the 2D data array length:")
            dbm_data_length = int_input_consider_default('>>> ', dbm_data_length)
            print("Set the root of the flash:")
            root_pos=int_input_consider_default('>>>',root_pos)
            if root_pos>dbm_data_width-1:
                print("out of area!")
            print("Do you want to set a lightning rod?[Y/N]")
            input_str_2=str(input('>>>'))
            if ('N' == input_str_2) or ('n' == input_str_2):
                use_lightning_rod = False
            else:
                use_lightning_rod = True
                print("Please input the position of the rod")
```

```
            lightning_rod_pos=int_input_consider_default('>>>',lightning_rod_pos)
            print("Please input the height of the rod")
        print("Start to calculate...")
        return data_file_mode,dbm_data_length,dbm_data_width,root_pos,use_lightning_rod,\
          lightning_rod_pos,lightning_rod_height

def U_show(U_array,len,wid):
    def title_and_labels(ax,title):
        ax.set_title(title);ax.set_xlabel("$y$")
        ax.set_ylabel("$x$");ax.set_zlabel("$U$")
    fig, axes=plt.subplots(2, 2, figsize=(6, 6),
                           subplot_kw={'projection': '3d'})
    y=np.linspace(2,wid-3,num=wid-4).astype('int32')
    x=np.linspace(2,len-3,num=len-4).astype('int32')
    X,Y=np.meshgrid(x,y)
    z=[]
    for i in range(0,len-4):
        for j in range(0,wid-4):
            z.append(U_array[x[i],y[j]])
    Z=np.array(z).reshape(len-4,wid-4)
    norm = mpl.colors.Normalize(vmin = Z.min(), vmax = Z.max())
    p=axes[0, 0].plot_surface(X,Y,Z,linewidth=0,rcount=20,
                              ccount=20,norm=norm,
                              cmap=cm.RdYlBu_r, edgecolor='blue')
    cb=fig.colorbar(p,ax=axes[0, 0],pad=0.1,shrink=0.6)
    title_and_labels(axes[0, 0], "surface plot")
    p=axes[0, 1].plot_wireframe(X,Y,Z,rcount=20,ccount=20,
                                color="green")
    title_and_labels(axes[0, 1],"wireframe plot")
    cset=axes[1, 0].contour(X,Y,Z,zdir='x',levels=20,
                            norm=norm,cmap=cm.RdYlBu_r, edgecolor='blue')
    title_and_labels(axes[1, 0],"contour x")

    cset=axes[1, 1].contour(X,Y,Z,zdir='y',levels = 20,
                            norm=norm,cmap=cm.RdYlBu_r, edgecolor='blue')
    title_and_labels(axes[1, 1],"contour y")
    fig.tight_layout(); plt.show()

def FileSave(U_array,len,wid,num):
    y=np.linspace(2,wid-3,num=wid-4).astype('int32')
    x=np.linspace(2,len-3,num=len-4).astype('int32')
    X,Y=np.meshgrid(x,y)
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    ax.set_xlabel("$y$")
    ax.set_ylabel("$x$")
    ax.set_zlabel("$U$")
```

```
    z=[]
    norm = mpl.colors.Normalize(vmin = Z.min(), vmax = Z.max())
    p=ax.plot_surface(X,Y,Z,linewidth=0,rcount=20,
                                ccount=20,norm=norm,
                                cmap=cm.RdYlBu_r, edgecolor='blue')
    cb=fig.colorbar(p,ax=ax,pad=0.1,shrink=0.6)
    plt.savefig('U_pic_%d.png' % num)
    plt.close(fig)


def main():
    file_mode,length, width, root_pos,use_lightning_rod,\
        lightning_rod_pos,lightning_rod_height=player_guide_interface()
    data=DBM_Data(length, width, root_pos,use_lightning_rod,\
        lightning_rod_pos,lightning_rod_height)
    plot=DBMTkVisual(data.map,length, width)
    flag=0
    runtime=time.time()
    while not data.is_to_boundary():
        sys.stdout.flush()
        sys.stdout.write("\b"*len("Iteration times:"+str(flag)))
        data.Breakdown_Point()
        data.U_matrix_upd()
        if not file_mode:
            plot.plot_pixel()
        flag=flag+1
        sys.stdout.write("Iteration times:"+str(flag))
        if file_mode:
            if flag%50==0:
                FileSave(data.U_cur,length,width,flag)

    print("\n run time:%f" % (time.time()-runtime))
    U_show(data.U_cur,length,width)

if __name__ == '__main__':
    main()
```

下面是神经网络的搭建程序:

```
import torch
import torch.nn as nn
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
import torch.optim as optim
from torch.autograd import Variable
import numpy as np
from matplotlib import cm


torch.backends.cudnn.benchmark = True

# bulid net
class Net(nn.Module):
    def __init__(self, NN):
        super(Net, self).__init__()

        self.input_layer = nn.Linear(2, NN)
        self.hidden_layer1 = nn.Linear(NN,NN)
        self.hidden_layer2 = nn.Linear(NN, NN)
        self.output_layer = nn.Linear(NN, 1)

    def forward(self, x):
        out = torch.tanh(self.input_layer(x))
        out = torch.tanh(self.hidden_layer1(out))
        out = torch.tanh(self.hidden_layer2(out))
        out_final = self.output_layer(out)
        return out_final

def laplace(x, net):
    #laplace equation
    u = net(x)
    u_tx = torch.autograd.grad(u, x, grad_outputs=torch.ones_like(net(x)),
                               create_graph=True, allow_unused=True)[0]
    d_y = u_tx[:, 0].unsqueeze(-1)
    d_x = u_tx[:, 1].unsqueeze(-1)
    u_xx = torch.autograd.grad(d_x, x, grad_outputs=torch.ones_like(d_x),
                          create_graph=True, allow_unused=True)[0][:,1].unsqueeze(-1)
    u_yy = torch.autograd.grad(d_y, x, grad_outputs=torch.ones_like(d_y),
                          create_graph=True, allow_unused=True)[0][:,0].unsqueeze(-1)
    return torch.from_numpy(np.zeros((150,1))).float()
```

```
net = Net(64).to('cuda')
#loss function and optimizer
mse_cost_function = torch.nn.MSELoss().to('cuda')
optimizer = torch.optim.Adam(net.parameters(), lr=1e-3)


x_right=torch.from_numpy(np.zeros((150,1))).float().to('cuda')
x_left=torch.from_numpy(np.full((150,1),97)).float().to('cuda')
y_up=torch.from_numpy(np.zeros((150,1))).float().to('cuda')
y_down=torch.from_numpy(np.full((150,1),97)).float().to('cuda')
bc_zero=torch.from_numpy(np.zeros((150,1))).float().to('cuda')
bc_max=torch.from_numpy(np.full((150,1),100)).float().to('cuda')
iterations = 5000
for epoch in range(iterations):
    optimizer.zero_grad()
# set grad to 0

    x_var = torch.from_numpy(
        np.random.uniform(low=0, high=97, size=(150, 1))).float().to('cuda')
    y_var = torch.from_numpy(
        np.random.uniform(low=0, high=97, size=(150, 1))).float().to('cuda')
    u_bc=y_var*100/97

    # boundary
    net_right = net(torch.cat([x_right,y_var], 1))
    net_left = net(torch.cat([x_var, y_var], 1))
    net_up = net(torch.cat([x_var, y_up], 1))
    net_down = net(torch.cat([x_var,y_down],1))
    mse_right = mse_cost_function(net_right, u_bc).to('cuda')
    mse_left = mse_cost_function(net_left, u_bc).to('cuda')
    mse_up = mse_cost_function(net_up, bc_zero).to('cuda')
    mse_down = mse_cost_function(net_down, bc_max).to('cuda')

    # interior point
    x_in = torch.from_numpy(
        np.random.uniform(low=0, high=97, size=(150, 1))).float().to('cuda')
    y_in = torch.from_numpy(
        np.random.uniform(low=0, high=97, size=(150, 1))).float().to('cuda')
    y_in.requires_grad=True
    x_in.requires_grad=True
    f_out = laplace(torch.cat([x_in, y_in], 1), net).to('cuda')
    mse_f = mse_cost_function(f_out, bc_zero).to('cuda')

    # counting loss
    loss = 100*mse_f+mse_up+mse_down+mse_left+mse_right
    loss=loss.to('cuda')
    loss.backward()
```

```
    optimizer.step()
    #equivalent to : theta_new = theta_old - alpha * derivative of J w.r.t theta
    with torch.autograd.no_grad():
        if epoch % 100 == 0:
            print(f"Epoch {epoch}/{4000}, Loss: {loss.item():.4f}")

#Picture
x = np.linspace(0, 97,150)
y = np.linspace(0, 97,150)
ms_x, ms_y = np.meshgrid(x, y)
x = np.ravel(ms_x).reshape(-1, 1)
y = np.ravel(ms_y).reshape(-1, 1)
pt_x = torch.from_numpy(x).float()
pt_y = torch.from_numpy(y).float()
pt_x.requires_grad=True
pt_y.requires_grad=True
pt_x = pt_x.to('cuda')
pt_y = pt_y.to('cuda')
pt_u0 = net(torch.cat([pt_x, pt_y], 1))
print(pt_u0.reshape(150,150))
u = pt_u0.detach().cpu().numpy()
pt_u0 = u.reshape(150,150)
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.set_zlim([0, 150])
ax.plot_surface(ms_y, ms_x, pt_u0, cmap=cm.RdYlBu_r,
    edgecolor='blue', linewidth=0.0003, antialiased=True)
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('u')
plt.show()
```