1. 方程形式：

一维burgers方程如上式所示，其中方程中值为未知待求解，初始条件为，边界条件设置为0。取值范围：

1. 精确解：该方程有精确解的数学公式，由式子可得到精确解文件：\burger反一维\Data\burgers\_shock.mat，该文件可用于作图，验证最终结果，于反问题中当作精确的实验数据。

文件中数据：

#有三个double数据，分别是

    #x :256\*1 是-1到1的均分

    #t :100\*1 0-1

    #usol精确解 256\*100

1. 文件说明：
   1. \Data 文件是用于存储burgers\_shock.mat文件
   2. \result\_data 文件是用于存放结果文件
   3. \result\_plot 文件是用于存放得到的结果图
   4. net.py 是网络的一些基础构建设置
   5. Parser\_PINN,py 是该问题所用到的所有参数，整合成解析器可以方便修改
   6. solve\_burger\_inverse.py 是求解一维burger方程反问题的主函数
   7. .ipynb 是Jupyter Notebook 来查看数据文件和写代码时留下的一些思路代码，属于调试文件，对结果无直接影响
2. 求解思路说明：
   1. 定义方程：上面已给出
   2. 构建神经网络：在此我们设计了一个全连接的神经网络

网络结构为：[2, 20, 20, 20, 20, 20, 20, 1]

激活函数为：tanh

学习率为:0.001

优化器为:adam法

残差计算：MSE 均方误差

Loss函数：loss\_PDE+loss\_BC 将PDE方程和边界条件加入到Loss函数以达到控制神经网络拟合方程的效果

的初始值：0 学习率0.00001

* 1. 具体学习：（思路）

optimizer.add\_param\_group({'params': nu, 'lr': 0.00001})

    #增加一个nu变量，此参数可以影响方程，以至于影响最终loss，我们要得到loss最小时对应的nu，此时为最佳参数

    #因为我们要求nu精度要求比较高，所以用小的学习率

for epoch in range(epochs):

optimizer.zero\_grad()

首先设置PDE方程内部的点：目的是求loss\_pde

随机取点 取10000（不取边界上的）

U\_f = PINN(t,x) uf是通过网络pinn预测出的解

PDE条件即U\_f ，x,t三个尽量满足方程：方程尽量趋近于0

Loss\_pde即求出

其次是边界条件：分三个，左边界，有边界，初始条件

以左边界为例：

随机分布,x=-1

U\_b\_1 = PINN(t,x)

边界函数值应该趋于0

三个条件1：1：1相加

Loss\_bc即求出

最后根据精确解（优秀实验数据）：选取多个点

求出预测解与真实解求均方误差

Loss\_data求出

Loss函数综合PDE、边界条件与真实解的权重比例为1:1:1

Loss = 1 \* loss\_PDE + 1 \* loss\_BC+ 1\*loss\_data

记录损失函数及画图

Loss.backward()

optimizer.step()

* 1. 验证网络及作图：
     1. 预测解的热力图：\result\_plot\ PINN\_pred(60000).png
     2. Loss值历史图 nu的值
     3. 三幅子图，分别是t=0.25,t=0.5,t=0.75时的x图

以上即为实验二的基本思路