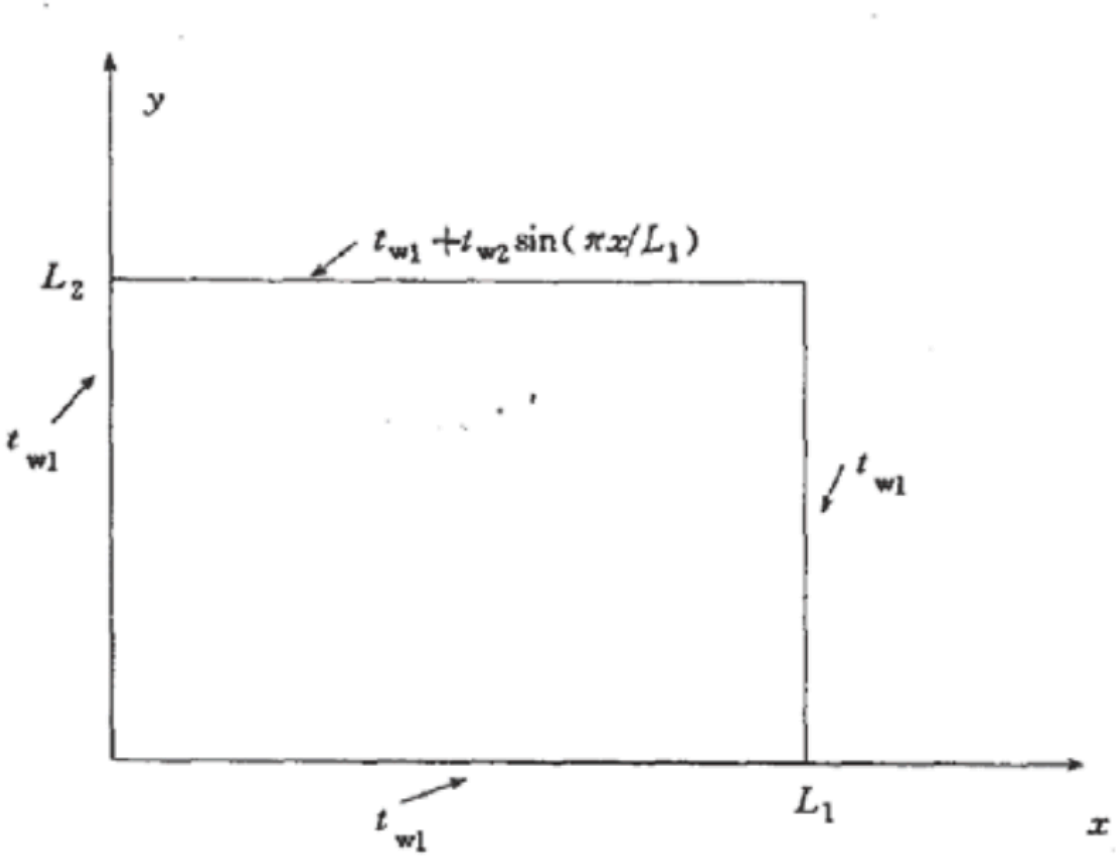
大作业内容见PDF，BIGHOMEWORK数字编号对应不同题目。COMSOL文件可以进行验证，其它细节如下文所述。

一、问题描述

1、题目描述

如图所示，一个无限长矩形柱体，其横截面的边长分别为 *L*1 和 *L*2，常物性。该问题可视为二维稳态导热问题，边界条件如图中所示，其中$L_1 = 0.6m$，$L_2 = 0.4m$，$T_{w1} = 60^\circ C$，$T_{w2} = 20^\circ C$，$\lambda = 200W/(m \cdot K)$



第一问：

(1)编写程序求解二维导热方程。

(2)绘制$x = L_1/2$和$y = L_2/2$处的温度场，并与解析解进行比较。已知矩形内的温度场的解析解为
$t(x,y) = t_{w1} + t_{w2}\sin(\pi x/L_1)\frac{\sinh(\pi y/L_1)}{\sinh(\pi L_2/L_1)}$

第二问：将第一题中$y = L_2$处的边界条件变为$t = t_{w2}$，其它条件不变。

(1)编写程序求解二维导热方程并计算从$y = 0$处导入的热量Φ_2

(2)当$L_2 \ll L_1$时，改二维导热问题可简化为一维导热问题。在一维近似下，试计算从$y = 0$处倒入的热量Φ_1，并比较不同L_2/L_1下Φ_2/Φ_1的比值。由该问题的解析解可知：

$L2/L1$	**0.007**	**0.01**	**0.05**	**0.08**	**0.1**
Φ_2/Φ_1	0.9987	0.9912	0.956	0.93	0.912

2、题目分析

对物理问题进行数值求解的基本思想可以概括为：把原来在时间、空间坐标系中连续的物理量的场，用有限个离散点上的值的集合来答题，通过求解按一定方法建立起来的关于这些值的代数方程，获得离散点上被求物理量的值。这些值称为该物理量的数值解。

本题的物理背景为第一类边界条件的有限长平面导热问题。无论是第一题还是第二题，对待此类问题，将采用数值解法的一般步骤流程：(1)建立控制方程及定解条件；(2)区域离散化；(3)建立节点物理量的代数方程；(4)设立迭代初场；

对于第一问，将通过泰勒级数展开法或热平衡法进内节点的离散方程。并通过高斯-赛德尔方法对设定的初值进行迭代。将通过建立多个全同矩阵的方法储存内节点并映射向不同的物理量与精度量。这种方法将快速且建议的完成任意精度、网格下的数值求解。若要求解该平面内任意区域的温度（点、线、面），若该区域位于内节点上，则内节点笛卡尔坐标表述与设立的矩阵脚标一一对应；若该区域不位于内节点上，例如两个内节点之间，在网格划分足够密集的情况下可以用插值法近似表示。

对于第二问，虽然改变了边界温度表述，但其本质上仍然为第一类边界条件问题；第一问在计算得出数值解的基础上进行简单修改初值即可完成计算。边界热流量即可通过温度与划分的网格进行计算。第二问改变矩阵对应的几何标度即可。

二、问题的完整数学描述

1、内节点离散方程的建立

本题的核心是对内节点进行离散，通常我们有两种内节点离散方程建立的方法，它们分别是泰勒(Taylor)级数展开法与热平衡法。

（1）泰勒级数展开法

对节点$(m+1,n)$及$(m-1,n)$分别写出函数t对(m,n)点的泰勒级数展开式：

$$t_{m+1,n}=t_{m,n}+\Delta x\frac{\partial t}{\partial x}\bigg|_{m,n}+\frac{\Delta x^2}{2}\frac{\partial^2 t}{\partial x^2}\bigg|_{m,n}+\frac{\Delta x^3}{6}\frac{\partial^3 t}{\partial x^3}\bigg|_{m,n}+\frac{\Delta x^4}{24}\frac{\partial^4 t}{\partial x^4}\bigg|_{m,n}+\cdots$$

$$t_{m-1,n}=t_{m,n}-\Delta x\frac{\partial t}{\partial x}\bigg|_{m,n}+\frac{\Delta x^2}{2}\frac{\partial^2 t}{\partial x^2}\bigg|_{m,n}-\frac{\Delta x^3}{6}\frac{\partial^3 t}{\partial x^3}\bigg|_{m,n}+\frac{\Delta x^4}{24}\frac{\partial^4 t}{\partial x^4}\bigg|_{m,n}+\cdots$$

两式相加可得：

$$t_{m+1,n}+t_{m-1,n}=2t_{m,n}+\Delta x^2\frac{\partial^2 t}{\partial x^2}\bigg|_{m,n}+\frac{\Delta x^4}{12}\frac{\partial^4 t}{\partial x^4}\bigg|_{m,n}+\cdots$$

改写为$\frac{\partial^2 t}{\partial x^2}\Big|_{m,n}$的表达式，有：

$$\frac{\partial^2 t}{\partial x^2}\bigg|_{m,n}=\frac{t_{m+1,n}-2t_{m,n}+t_{m-1,n}}{\Delta x^2}+O\left(\Delta x^2\right)$$

略去阶段误差$O\left(\Delta x^2\right)$可以的得到中心差分表达式：

$$\frac{\partial^2 t}{\partial x^2}\bigg|_{m,n}=\frac{t_{m+1,n}-2t_{m,n}+t_{m-1,n}}{\Delta x^2}$$

同理可以得到：

$$\frac{\partial^2 t}{\partial y^2}\bigg|_{m,n}=\frac{t_{m,n+1}-2t_{m,n}+t_{m,n-1}}{\Delta y^2}$$

可以的到最终的离散表达式：

$$\frac{t_{m+1,n}-2t_{m,n}+t_{m-1,n}}{\Delta x^2}+\frac{t_{m,n+1}-2t_{m,n}+t_{m,n-1}}{\Delta y^2}=0$$

（2）热平衡法

从节点$(m-1,n)$通过界面w传导到节点(m,n)的热流量可表示为：

$$\Phi_w=\lambda\Delta y\frac{t_{m-1,n}-t_{m,n}}{\Delta x}$$

其他三个界面e、n、s的也可以通过同样的方式给出，又由于元体(m,n)的热量符合能量守恒方程：

$$\Phi_e+\Phi_w+\Phi_n+\Phi_s=0$$

将上述表达式代入得：

$$\lambda\frac{t_{m-1,n}-t_{m,n}}{\Delta x}\Delta y+\lambda\frac{t_{m+1,n}-t_{m,n}}{\Delta x}\Delta y+\lambda\frac{t_{m,n+1}-t_{m,n}}{\Delta y}\Delta x+\lambda\frac{t_{m,n-1}-t_{m,n}}{\Delta y}\Delta x=0$$

若我们取$\Delta x=\Delta y$，则热平衡法与泰勒展开法表达形式一致。同时，由于第一类边界条件的导热问题所有内节点的离散方程组成了一个封闭的代数方程组可以立即求解，不需要对边界进行其它的处理。

2、Gauss-Seidel迭代法

设有一个三元方程组，记为：

$$\begin{cases}a_{11}t_1+a_{12}t_2+a_{13}t_3=b_1\\a_{21}t_1+a_{22}t_2+a_{23}t_3=b_2\\a_{31}t_1+a_{32}t_2+a_{33}t_3=b_3\end{cases}$$

采用高斯-赛德尔迭代法求解的步骤如下：

（1）将方程组改写成关于t_1、t_2、t_3的显式形式（迭代方程）：

$$\begin{cases}t_1=\frac{1}{a_{11}}(b_1-a_{12}t_2-a_{13}t_3)\\t_2=\frac{1}{a_{22}}(b_2-a_{21}t_1-a_{23}t_3)\\t_3=\frac{1}{a_{33}}(b_3-a_{31}t_1-a_{32}t_2)\end{cases}$$

（2）假设一组解（即迭代初场），通过上式逐一计算出改进值。每次计算均用最新的计算代入。

（3）以计算所得到的值作为初场，重复上述计算直到相邻两次迭代值之差小于允许值。此时称为已达到迭代收敛。迭代计算终止。通常由三种常用判据进行收敛判断：

$$\max\left|t_i^{(k)}-t_i^{(k+1)}\right|\leqslant\varepsilon$$

$$\max\left|\frac{t_i^{(k)}-t_i^{(k+1)}}{t_i^{(k)}}\right|\leqslant\varepsilon$$

$$\max\left|\frac{t_i^{(k)}-t_i^{(k+1)}}{t_{\max}^{(k)}}\right|\leqslant\varepsilon$$

通过热平衡法导出差分方程时，若每一个方程都选用该方程的中心节点温度作为迭代变量则一定满足对角占优$(\text{diagonal predominant})$，迭代一定收敛。

三、网格的建立与方程的离散

1.网格的划分与建立

通过给出的完整模型，可以得到适用于本题条件的网格建立方法。首先需要设定本题要求的几何尺寸L_1,L_2，再确定对于横、纵方向上需要划分的点的数量。最后通过几何尺寸与点尺寸的比值确定Δx、Δy的值。例如对于$L_1=0.6$,$L_2=0.4$的矩形平面，设定长方向分割数量为50，纵方向分割数量为50。这样构建出50×50的网格平面，对应的$\Delta x=\frac{0.6}{50}$、$\Delta y=\frac{0.4}{50}$。如此这般并可以完成网格的划分与建立。

2.方程的离散

依据之前提及的方程离散方法，通过定义$k=\dfrac{\Delta y}{\Delta x}$，进一步简化

$$\frac{t_{m+1,n}-2t_{m,n}+t_{m-1,n}}{\Delta x^2}+\frac{t_{m,n+1}-2t_{m,n}+t_{m,n-1}}{\Delta y^2}=0$$

或

$$\lambda\frac{t_{m-1,n}-t_{m,n}}{\Delta x}\Delta y+\lambda\frac{t_{m+1,n}-t_{m,n}}{\Delta x}\Delta y+\lambda\frac{t_{m,n+1}-t_{m,n}}{\Delta y}\Delta x+\lambda\frac{t_{m,n-1}-t_{m,n}}{\Delta y}\Delta x=0$$

可以得到适用于高斯-赛德尔迭代法的简化离散化表达形式：

$$t_{m,n}=\frac{k^2t_{m-1,n}+k^2t_{m+1,n}+t_{m,n+1}+t_{m,n-1}}{2k^2+2}$$

在第二问中还需要对导入的热量进行求解，热量满足表达式：

$$\Phi=\lambda\Delta y\frac{t_{m-1,n}-t_{m,n}}{\Delta x}=\lambda k(t_{m-1,n}-t_{m,n})$$

则在$y=0$处导入的热量为：

$$\Phi=\lambda\Delta y\frac{t_{m,n-1}-t_{m,n}}{\Delta x}=\lambda k(t_{m,n-1}-t_{m,n})$$

即只考虑沿一个方向的延伸。

四、求解的基本流程和程序框架

1、总模型的建立

1、核心流程图

2、模型框架展示

两问可以用同一框架描述：

1.确立初始条件

```
#确定初始条件
L1,L2 = 0.6,0.4
x_squard,y_squard = 60,40 #确定单边的分割数量
dx,dy = L1/x_squard,L2/y_squard
k=dx/dy
#确定精确度
e = 0.000000001
```

2.构建温度平面并赋予初值

```
#构建二维温度平面，此时为0矩阵
t = np.zeros([y_squard,x_squard])
#确定边界条件
tw1,tw2 = 60. ,20.
delta = np.linspace(0,L1,x_squard)
tw3 = tw1+tw2*np.sin(np.pi*delta/L1)
t[0]= tw3
t[1:,0]=tw1
t[y_squard-1]= tw1
t[1:,x_squard-1]=tw1
#将待求解内部区域添加初始值
t[1:y_squard-2,1:x_squard-2]=70
```

3.构建全同判断矩阵f

```
#构建与t完全一致待收敛判断序列
#当序列f全为1时，即每一点位置处符合精确度要求
f = np.zeros_like(t)
f[0]=1
f[1:,0]=1
f[y_squard-1]=1
f[1:,x_squard-1]=1
```

4.高斯-赛德尔法迭代求解

```
start = time.time()#计算迭代时间
while (f==1).all() == False:
    for x in range(1,x_squard-1):
        for y in range(1,y_squard-1):
            flag=t[y,x]
            t[y,x]=(k*k*t[y-1,x]+k*k*t[y+1,x]+t[y,x-1]+t[y,x+1])/(2*k*k+2)
            #判断是否符合精确度条件
```

```
            if abs(t[y,x]-flag) < e:
                f[y,x] = 1
            else:
                f[y,x] = 0
    count=count+1
    print('\r',count,end='',flush=True)#输出迭代次数
end = time.time()#计算迭代时间
```

2、第一问求解

第一小问的求解依据主框架进行程序编写，并通过matplotlib库绘制出三位图即可。

```
X=np.linspace(0,L1,x_squard)
Y=np.linspace(0,L2,y_squard)
X,Y = np.meshgrid(X,Y)
ax= plt.axes(projection = '3d')
ax.plot_surface(X,Y,t,rstride=1, cstride=1,cmap=cm.coolwarm,
                      linewidth=0, antialiased=False)
plt.show()
```

第二小问的求解在主框架的基础上取数组中对应的某一行与某一列绘制图片即可

```
X = np.arange(0,y_squard,1)
Y = t[0:,30]
plt.plot(X*0.4/40,Y,label='Numerical solution')
N = np.arange(0,L2,0.01)
M = tw1+tw2*np.sin(0.3*np.pi/L1)*(np.sinh(np.pi*(0.4-N)/L1)/np.sinh(np.pi*L2/L1))
plt.plot(N,M,'--',color = "red",label='Analytical solution')
plt.xlim(0,0.4)
plt.legend( )
以及
X = np.arange(0,x_squard,1)
Y = t[0,0:]
plt.plot(X*0.6/60,Y,label='Numerical solution')
N = np.arange(0,L1,0.001)
M = tw1+tw2*np.sin(N*np.pi/L1)*(np.sinh(np.pi*L2/L1)/np.sinh(np.pi*L2/L1))
plt.plot(N,M,'--',color = "red",label='Analytical solution')
plt.xlim(0,0.6)
plt.legend( )
```

3、第二问求解

第二问的求解需要在框架的基础上修改初始条件。在第一小问中仅需修改tw3的温度值即可，Φ计算的方式可由$\Phi = \lambda\Delta y\frac{t_{m,n-1}-t_{m,n}}{\Delta x} = \lambda k(t_{m,n-1}-t_{m,n})$的变式给出：

```
phi1 = 0
for i in range(0,x_squard):
    phi1 = lamb*(1/dy)*(t[y_squard-1,i]-t[y_squard-2,i]) + phi1
```

第二小问改变初始条件的赋值策略并增加一位稳态的计算公式，其余的程序与第一小问一直。

```
#赋值策略
p=0.007
L1=0.6
L2 = p*L1
#一位稳态计算公式：
phi1 = L1*lamb*(tw1-tw2)/L2
```

五、数值计算结果与比较分析

本程序使用Python 3.8编写，使用miniforge3创建虚拟环境。程序运行中使用的外部依赖库及版本为为numpy 1.21.2，Matplotlib 3.4.2。使用UTF-8编码。IDE环境为社区版pycharm。操作系统环境为基于M1-Arm芯片架构的macOS Monterey 12.0.1

1、第一问计算结果与比较分析

（1）第一小问结果与比较：

第一小问结果如下图所示：

为了验证该图的正确性，使用Comsol5.6进行了绘制：

由此可见与商用软件的符合程度较好。

（2）第二小问结果与分析

$x = L1/2$处的温度场的解析解与数值解如下图所示：

$y = L2/2$处的解析解与数值解比较如下图所示：

分析：可以看出，该题结果与解析解结果在初始范围内符合的较好，但在边界处产生了一定的误差。可能的原因是该程序使用的语言特性会在赋值不一致的时候向下或者向上取值，导致整体沿着某一方向进行偏差。但整体的符合性较高，误差在可接受的范围内。

2、第二问计算结果与比较分析

（1）第一小问结果与分析

第一小问可以得到如下图所示的温度场：

与Comsol绘制的图片相比较（开尔文表示）：

具备较好的相似程度，此时：$\Phi = 5217.4585W$

（2） 第二小问结果与分析

第二小问通过计算得到如下数值：

$L2/L1$	**0.007**	**0.01**	**0.05**	**0.08**	**0.1**
Φ_1	1142857.14	800000.0	160000.0	100000.0	80000.0
Φ_2	1122386.519	784907.597	152559.310	92795.705	72848.406
Φ_2/Φ_1	0.982	0.981	0.954	0.928	0.911
实际Φ_2/Φ_1	0.9987	0.9912	0.956	0.93	0.912

分析:可以发现随着精度的不断趋紧，其误差越来越大。该误差一方面由之前所述的原因所导致的，另一方面，该误差还可能来自于考虑到$python$语言速度较慢而降低了网格数量和精度误差，而这些采用保守型策略的方式会对该在高精度环境下产生较大的偏差。但整体来看，此方法还是具备一定的精度，可以用来进行简单的计算和示意。

六、结论

本次大作业完成了有关第一类边界问题的数值计算问题，使用泰勒展开与热平衡法两类方法对传热方程进行了合理的离散化处理，并使用高斯赛德尔方法对离散后的传热方程进行迭代，由此获得了该类问题的数值解。在第一问中我们通过温度场分布绘制出所需要的温度分布图像与特殊位置处的温度分布曲线。在第二问中我们得到计算热量传递的方法以及与一维传热问题之间的比较分析。通过将数值解与解析解比较，我们可以发现该方法具备一定的精度，可以满足在合理误差范围内的应用。但为了获得更高的精度与更高的运行速度，除了采用更为精确的离散方程与更为高效的迭代方法外，还应该具备较好的程序处理意识与能力。本次大作业通过计算第一类边界问题的两种模型，验证了传热定律的有效性，解决了温度分布问题与热量计算问题。不足的是仍然存在一定的误差，而且在高精度条件下运行速度过慢。需要继续努力学习。

附录：源代码

```
#BIGHOMEWOEK11
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time
L1,L2 = 0.6,0.4
x_squard,y_squard = 60,40 #确定单边的分割数量
```

```
dx,dy = L1/x_squard,L2/y_squard
k=dx/dy
print(k)
##确定精确度区间
e = 0.000000001
#构建二维温度平面，此时为0矩阵
t = np.zeros([y_squard,x_squard])
#确定边界条件
tw1,tw2 = 60. ,20.
delta = np.linspace(0,L1,x_squard)
tw3 = tw1+tw2*np.sin(np.pi*delta/L1)
t[0]= tw3
t[1:,0]=tw1
t[y_squard-1]= tw1
t[1:,x_squard-1]=tw1
#将待求解内部区域添加初始值
t[1:y_squard-2,1:x_squard-2]=70
#构建与t完全一致待收敛判断序列
#当序列f全为1时，即每一点位置处符合精确度要求
f = np.zeros_like(t)
f[0]=1
f[1:,0]=1
f[y_squard-1]=1
f[1:,x_squard-1]=1
#通过高斯-赛德尔法进行迭代
x,y=1,1
count=0
start = time.time()
while (f==1).all() == False:
    for x in range(1,x_squard-1):
        for y in range(1,y_squard-1):
            flag=t[y,x]
            t[y,x]=(k*k*t[y-1,x]+k*k*t[y+1,x]+t[y,x-1]+t[y,x+1])/(2*k*k+2)
            #判断是否符合精确度条件
            if abs(t[y,x]-flag) < e:
                f[y,x] = 1
            else:
                f[y,x] = 0
    count=count+1
    print('\r',count,end='',flush=True)
end = time.time()
print("\r高斯-赛德尔法迭代时间：%.2f秒"%(end-start))
#偷懒方法做出图像
#plt.matshow(t,cmap=plt.cm.Reds)
#不给偷懒!
X=np.linspace(0,L1,x_squard)
Y=np.linspace(0,L2,y_squard)
X,Y = np.meshgrid(X,Y)
ax= plt.axes(projection = '3d')
```

```
ax.plot_surface(X,Y,t,rstride=1, cstride=1,cmap=cm.coolwarm,
                        linewidth=0, antialiased=False)
"""
X = np.arange(0,51,1)
Y = t[25,0:]
plt.plot(X*dx,Y)
"""
plt.show()
#输出矩阵，取三位有效值
print(np.around(t,3))
```

```
#BIGHOMEWOEK12
import numpy as np
import matplotlib.pyplot as plt
import time
L1,L2 = 0.6,0.4
x_squard,y_squard = 60,40 #确定单边的分割数量
dx,dy = L1/x_squard,L2/y_squard
k=dx/dy
print(k)
##确定精确度区间
e = 0.000000001
#构建二维温度平面，此时为0矩阵
t = np.zeros([y_squard,x_squard])
#确定边界条件
tw1,tw2 = 60. ,20.
delta = np.linspace(0,L1,x_squard)
tw3 = tw1+tw2*np.sin(np.pi*delta/L1)
t[0]= tw3
t[1:,0]=tw1
t[y_squard-1]= tw1
t[1:,x_squard-1]=tw1
#将待求解内部区域添加初始值
t[1:y_squard-2,1:x_squard-2]=70
#构建与t完全一致待收敛判断序列
#当序列f全为1时，即每一点位置处符合精确度要求
f = np.zeros_like(t)
f[0]=1
f[1:,0]=1
f[y_squard-1]=1
f[1:,x_squard-1]=1
#通过高斯-赛德尔法进行迭代
x,y=1,1
count=0
start = time.time()
while (f==1).all() == False:
```

```
    for x in range(1,x_squard-1):
        for y in range(1,y_squard-1):
            flag=t[y,x]
            t[y,x]=(k*k*t[y-1,x]+k*k*t[y+1,x]+t[y,x-1]+t[y,x+1])/(2*k*k+2)
            #判断是否符合精确度条件
            if abs(t[y,x]-flag) < e:
                f[y,x] = 1
            else:
                f[y,x] = 0
    count=count+1
    print('\r',count,end='',flush=True)
end = time.time()
print("\r高斯-赛德尔法迭代时间：%.2f秒"%(end-start))
#偷懒方法做出图像
#plt.matshow(t,cmap=plt.cm.Reds)
#不给偷懒!
'''
X = np.arange(0,y_squard,1)
Y = t[0:,30]
plt.plot(X*0.4/40,Y,label='Numerical solution')
N = np.arange(0,L2,0.01)
M = tw1+tw2*np.sin(0.3*np.pi/L1)*(np.sinh(np.pi*(0.4-N)/L1)/np.sinh(np.pi*L2/L1))
plt.plot(N,M,'--',color = "red",label='Analytical solution')
plt.xlim(0,0.4)
plt.legend( )
'''
X = np.arange(0,x_squard,1)
Y = t[0,0:]
plt.plot(X*0.6/60,Y,label='Numerical solution')
N = np.arange(0,L1,0.001)
M = tw1+tw2*np.sin(N*np.pi/L1)*(np.sinh(np.pi*L2/L1)/np.sinh(np.pi*L2/L1))
plt.plot(N,M,'--',color = "red",label='Analytical solution')
plt.xlim(0,0.6)
plt.legend( )

plt.show()
#输出矩阵，取三位有效值
print(np.around(t,3))
```

```
#BIGHOMEWOEK21
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time
L1,L2 = 0.6,0.4
x_squard,y_squard = 60,40 #确定单边的分割数量
dx,dy = L1/x_squard,L2/y_squard
k=dx/dy
```

```
lamb = 200.
print(k)
##确定精确度区间
e = 0.00000001
#构建二维温度平面，此时为0矩阵
t = np.zeros([y_squard,x_squard])
#确定边界条件
tw1,tw2 = 60. ,20.
delta = np.linspace(0,L1,x_squard)
tw3 = tw2
t[0]= tw3
t[1:,0]=tw1
t[y_squard-1]= tw1
t[1:,x_squard-1]=tw1
#将待求解内部区域添加初始值
t[1:y_squard-2,1:x_squard-2]=60
#构建与t完全一致待收敛判断序列
#当序列f全为1时，即每一点位置处符合精确度要求
f = np.zeros_like(t)
f[0]=1
f[1:,0]=1
f[y_squard-1]=1
f[1:,x_squard-1]=1
#通过高斯-赛德尔法进行迭代
x,y=1,1
count=0
start = time.time()
while (f==1).all() == False:
    for x in range(1,x_squard-1):
        for y in range(1,y_squard-1):
            flag=t[y,x]
            t[y,x]=(k*k*t[y-1,x]+k*k*t[y+1,x]+t[y,x-1]+t[y,x+1])/(2*k*k+2)
            #判断是否符合精确度条件
            if abs(t[y,x]-flag) < e:
                f[y,x] = 1
            else:
                f[y,x] = 0
    count=count+1
    print('\r',count,end='',flush=True)
end = time.time()
print("\r高斯-赛德尔法迭代时间：%.2f秒"%(end-start))
phi1 = 0
for i in range(0,x_squard):
    phi1 = lamb*(1/dy)*(t[y_squard-1,i]-t[y_squard-2,i]) + phi1

print(round(phi1/100,10))

#偷懒方法做出图像
#plt.matshow(t,cmap=plt.cm.Reds)
```

```
#不给偷懒!
X=np.linspace(0,L1,x_squard)
Y=np.linspace(0,L2,y_squard)
X,Y = np.meshgrid(X,Y)
ax= plt.axes(projection = '3d')


ax.plot_surface(X,Y,t,rstride=1, cstride=1,cmap=cm.coolwarm,
                        linewidth=0, antialiased=False)
"""
X = np.arange(0,51,1)
Y = t[25,0:]
plt.plot(X*dx,Y)
"""
plt.show()
#输出矩阵，取三位有效值
print(np.around(t,3))
```

```
#BIGHOMEWOEK22
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time
p=0.007
L1=0.6
L2 = p*L1
x_squard,y_squard = 60,60 #确定单边的分割数量
dx,dy = L1/x_squard,L2/y_squard
k=dx/dy
lamb = 200.
print(k)
##确定精确度区间
e = 0.000001
#构建二维温度平面，此时为0矩阵
t = np.zeros([y_squard,x_squard])
#确定边界条件
tw1,tw2 = 60. ,20.
delta = np.linspace(0,L1,x_squard)
tw3 = tw2
t[0]= tw3
t[1:,0]=tw1
t[y_squard-1]= tw1
t[1:,x_squard-1]=tw1
#将待求解内部区域添加初始值
t[1:y_squard-2,1:x_squard-2]=60
#构建与t完全一致待收敛判断序列
#当序列f全为1时，即每一点位置处符合精确度要求
f = np.zeros_like(t)
```

```
f[0]=1
f[1:,0]=1
f[y_squard-1]=1
f[1:,x_squard-1]=1
#通过高斯-赛德尔法进行迭代
x,y=1,1
count=0
start = time.time()
while (f==1).all() == False:
    for x in range(1,x_squard-1):
        for y in range(1,y_squard-1):
            flag=t[y,x]
            t[y,x]=(k*k*t[y-1,x]+k*k*t[y+1,x]+t[y,x-1]+t[y,x+1])/(2*k*k+2)
            #判断是否符合精确度条件
            if abs(t[y,x]-flag) < e:
                f[y,x] = 1
            else:
                f[y,x] = 0
    count=count+1
    print('\r',count,end='',flush=True)
end = time.time()
print("\r高斯-赛德尔法迭代时间：%.2f秒"%(end-start))
phi2 = 0
for i in range(0,x_squard):
    phi2 = lamb*(1/dy)*(t[y_squard-1,i]-t[y_squard-2,i]) + phi2
print(round(phi2/100,10))

phi1 = L1*lamb*(tw1-tw2)/L2
print(phi1)
print(phi2/(100*phi1))
#偷懒方法做出图像
#plt.matshow(t,cmap=plt.cm.Reds)
#不给偷懒!
X=np.linspace(0,L1,x_squard)
Y=np.linspace(0,L2,y_squard)
X,Y = np.meshgrid(X,Y)
ax= plt.axes(projection = '3d')


ax.plot_surface(X,Y,t,rstride=1, cstride=1,cmap=cm.coolwarm,
                        linewidth=0, antialiased=False)
"""
X = np.arange(0,51,1)
Y = t[25,0:]
plt.plot(X*dx,Y)
"""
plt.show()
#输出矩阵，取三位有效值
print(np.around(t,3))
```