计算物理第八次作业

吴远清 2018300001031

第一题

理论

$$\nabla^2 = (\hat{i}\frac{\partial}{\partial x} + \hat{j}\frac{\partial}{\partial y}) = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

二阶偏微分差分格式为:

$$\frac{\partial^2 V}{\partial x^2} = \frac{V(i+1,j) + V(i-1,j) - 2V(i,j)}{\Delta x^2}$$

由此差分Laplace方程得:

$$V(i,j) = \frac{V(i+1,j) + V(i-1,j) + V(i,j+1) + V(i,j-1)}{4}$$

代码实现

首先导入库:

```
import numpy as np
import numba
from numba import jit
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

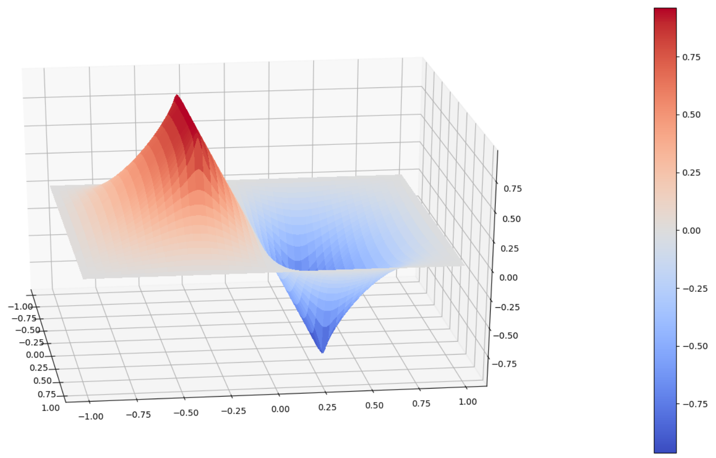
然后实现`CapcitorSolver`类:

```
class CapcitorSolver():
    def __init__(self, geometry = [(-1,-1),(1,1)], meshLeng = 0.001, relaxError
= 0.00001):
        self.geo = geometry
        self.size = (int((self.geo[1][0] - self.geo[0][0])/meshLeng),
int((self.geo[1][1] - self.geo[0][1])/meshLeng))
        self.dl = meshLeng
        self.v = np.zeros(self.size)
        self.r = relaxError
    @jit()
    def __boundaryCondition(self):
        self.v[0,:] = 0
        self.v[-1,:] = 0
        self.v[:,0] = 0
        self.v[:,-1] = 0
        self.v[int(self.size[0]/3), int(self.size[1]/4):int(self.size[1]*3/4)] =
1
        self.v[int(self.size[0]*2/3), int(self.size[1]/4):int(self.size[1]*3/4)]
= -1
```

```
    @jit()
    def __update(self):
        new = (self.v[0:-2, 1:-1] + self.v[2:, 1:-1] + self.v[1:-1,2:] +
self.v[1:-1,0:-2])/4
        deltaV = np.mean(np.abs(new - self.v[1:-1,1:-1]))
        print(deltaV)
        self.v[1:-1,1:-1] = new
        if deltaV > self.r:
            return False
        else:
            return True
    def calc(self):
        flag = False
        while not flag:
            self.__boundaryCondition()
            flag = self.__update()
    def show(self):
        X = np.arange(self.geo[0][0], self.geo[1][0], self.dl)
        Y = np.arange(self.geo[0][1], self.geo[1][1], self.dl)
        X, Y = np.meshgrid(X, Y)
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        surf = ax.plot_surface(X, Y, self.v,cmap=cm.coolwarm,linewidth=0,
antialiased=False)
        fig.colorbar(surf)
        plt.show()
```

对于`__boundaryCondition`和`__update`方法,我们使用`numba`库进行`jit`编译,以提升运行速度.

结果展示



可见,计算结果还算准确.

第二题

问题分析

首先,在我们的算法中, 较多的用到矩阵的切片, 而不需要用到矩阵之间的加法和乘法, 因此, 我们选用 `lil_matrix` (List of lists format)

> 关于 `scipy` 中各种不同稀疏矩阵储存方式的区别,请参见:https://blog.csdn.net/jeffery0207/article/details/100064602

我们可以直接继承自第一题中的 `CapcitorSolver` 类, 但是我们需要重写 `__init__` 方法和 `show` 方法:

- 在 `__init__` 方法中,我们需要重写, 以将电势转化为稀疏矩阵储存
- 在 `show` 方法中, 我们需要重写, 因为 `matplotlib` 不支持稀疏矩阵形式的输入, 我们需要调用稀疏矩阵的 `toarray` 方法,来实现成功的绘图.

代码实现

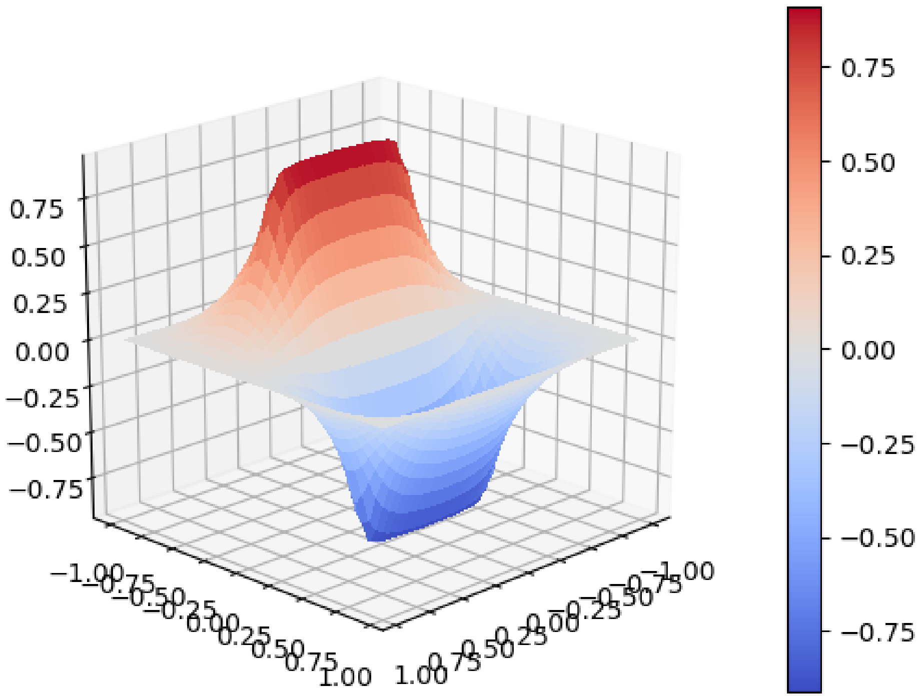
引入相关库

```
import numpy as np
from Q1 import CapcitorSolver
from scipy.sparse import lil_matrix
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
```

继承并重写类:

```
class sparseMatrixCapcitorSolver(CapcitorSolver):
    def __init__(self, geometry = [(-1,-1),(1,1)], meshLeng = 0.05, relaxError =
0.003):
        super().__init__(geometry, meshLeng, relaxError)
        print("Common matrix occupied {} bytes of
memory.".format(self.v.__sizeof__()))
        self.v[0,:] = 0
        self.v[-1,:] = 0
        self.v[:,0] = 0
        self.v[:,-1] = 0
        self.v[int(self.size[0]/3), int(self.size[1]/4):int(self.size[1]*3/4)] =
1
        self.v[int(self.size[0]*2/3), int(self.size[1]/4):int(self.size[1]*3/4)]
= -1
        self.v = lil_matrix(self.v)
        print("Sparse matrix occupied {} bytes of
memory.".format(self.v.__sizeof__()))
    def show(self):
        X = np.arange(self.geo[0][0], self.geo[1][0], self.dl)
        Y = np.arange(self.geo[0][1], self.geo[1][1], self.dl)
        X, Y = np.meshgrid(X, Y)
        fig = plt.figure()
        ax = fig.gca(projection='3d')
        surf = ax.plot_surface(X, Y,
self.v.toarray(),cmap=cm.coolwarm,linewidth=0, antialiased=False)
        fig.colorbar(surf)
        plt.show()
```

结果展示



计算出来的结果自然是相同的.但是值得注意的是空间占用情况. 在代码中, 我们调用了`python`内置的`__sizeof__`方法来获取对象的内存占用, 可以看出, 对于网格长度为0.05时,

```
Common matrix occupied 12912 bytes of memory.
Sparse matrix occupied 32 bytes of memory.
```

二者占用空间已存在较大差距.

如果进一步,将网格长度缩小到0.0001时:

```
Common matrix occupied 3200000112 bytes of memory.
Sparse matrix occupied 32 bytes of memory.
```

可见, 此时非稀疏矩阵的内存占用远远大于稀疏矩阵.