作业 2：数值求解泊松方程

谢文进

2021 年 4 月 26 日

1 数值求解泊松方程

1.1 理论推导

现求解二维泊松方程：

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y), \quad (x,y) \in \Omega \\ u = u_0 \quad \text{on } \partial\Omega \end{cases} \tag{1}$$

其中区域 Ω 为矩形区域，即 $\Omega = \{(x,y) : 0 < x < 1, 0 < y < 1\}$. 首先对该矩形区域进行划分，划分成若干个小矩形，取 $x_i = i\Delta x, \quad i = 0, 1, \cdots, N+1, \Delta x = \frac{1}{N+1}$, $y_j = j\Delta y, \quad j = 0, 1, \cdots, M+1, \Delta y = \frac{1}{M+1}$。由有限差分方法可推得：

$$\frac{1}{\Delta x^2}(u_{i-1,j} - 2u_{i\,j} + u_{i+1,j}) + \frac{1}{\Delta y^2}(u_{i,j-1} - 2u_{i\,j} + u_{i,j+1}) = f_{i,j}$$

$$\frac{\Delta y^2}{\Delta x^2}(u_{i-1,j} - 2u_{i\,j} + u_{i+1,j}) + (u_{i,j-1} - 2u_{i\,j} + u_{i,j+1}) = \Delta y^2 f_{i,j}$$

令 $\lambda = \frac{\Delta y}{\Delta x}$，化简可得：

$$\lambda^2 u_{i-1,j} + \lambda^2 u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 2(1+\lambda^2)u_{i\,j} = \Delta y^2 f_{i,j} \tag{2}$$

(2) 式中有 $M \times N$ 个未知的 u_{ij}，将之写成矩阵形式 $AU = F$，其中

$$U = [u_{1,1}, \ldots, u_{N,1}, u_{1,2}, \ldots, u_{N,2}, \ldots, u_{1,M}, \ldots, u_{N,M}]^T.$$

$$A = \begin{bmatrix} T & D & & & & \\ D & T & D & & & \\ & & \ddots & \ddots & \ddots & \\ & & & D & T & D \\ & & & & D & T \end{bmatrix},$$

$$T = \begin{bmatrix} -2(1+\lambda^2) & \lambda^2 & & & & \\ \lambda^2 & -2(1+\lambda^2) & \lambda^2 & & & \\ & & \ddots & \ddots & \ddots & \\ & & & \lambda^2 & -2(1+\lambda^2) & \lambda^2 \\ & & & & \lambda^2 & -2(1+\lambda^2) \end{bmatrix}.$$

1.2 实例

求解如下方程：

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 5e^{x+2y}, & 0 < x < 1, 0 < y < 1 \\ u = e^{2y} & x = 0, 0 < y < 1 \\ u = e^{x} & y = 0, 0 < x < 1 \\ u = e^{1+2y} & x = 1, 0 < y < 1 \\ u = e^{x+2} & y = 1, 0 < x < 1 \end{cases} \tag{3}$$

其精确解为 $u(x,y) = e^{x+2y}$。

1.3 编程实现

根据上述推导，用 python 编写程序，代码如下：

```python
import numpy as np
import matplotlib.pyplot as plt
import math

# 生成矩阵T、D，为生成矩阵A做准备
def generate_TD(N, dx, dy):
    T = np.zeros([N,N])
    D = np.zeros([N,N])
    a = (dy/dx)**2
```

```
    for i in range(N):
        T[i,i] = -2*(1+a)
        D[i,i] = 1
        if (i < N-1):
            T[i,i+1] = a
        if (i > 0):
            T[i,i-1] = a
    return T, D

# 生成矩阵A
def assemble_A(N, M, dx, dy):
    T, D = generate_TD(N, dx, dy)
    A = np.zeros([N*M, N*M])
    for j in range(M):
        A[j*N:(j+1)*N, j*N:(j+1)*N] = T
        if (j < M-1):
            A[j*N:(j+1)*N, (j+1)*N:(j+2)*N] = D
        if (j > 0):
            A[j*N:(j+1)*N, (j-1)*N:(j)*N] = D
    return A


def f(x, y):
    return 5 * math.exp(x + 2 * y)

# 精确解
def exact_f(x, y):
    return math.exp(x + 2 * y)

def gL(y):
    return math.exp(2 * y)

def gR(y):
    return math.exp(1 + 2 * y)

def gB(x):
    return math.exp(x)

def gT(x):
```

```python
    return math.exp(x + 2)

def assemble_F(x, y, dx, dy, N, M, gL, gR, gB, gT):
    F = np.zeros(N*M)

    a = (dy/dx)**2

    # dy^2 * f(i,j)
    for j in range(M):
        for i in range(N):
            F[j * N + i] += ((dy) ** 2) * f(x[i + 1], y[j + 1])

    # left BCs
    for j in range(M):
        F[j*N] += -a*gL(y[j+1])

    # right BCs
    for j in range(M):
        F[(j+1)*N - 1] += -a*gR(y[j+1])

    # top BCs
    for i in range(N):
        F[N * (M - 1) + i] += -gT(x[i+1])

    # bottom BCs
    for i in range(N):
        F[i] += -gB(x[i + 1])

    return F

def exact_solution(N, M, x, y):
    U_exact = np.zeros(N * M)
    for j in range(M):
        for i in range(N):
            U_exact[j * N + i] = exact_f(x[i + 1], y[j + 1])
    return U_exact

def Possion_solver(N, M, gL, gR, gB, gT):
    dx = 1./(N+1)
```

```
        dy = 1./(M+1)
        x = np.linspace(0, 1, N+2)
        y = np.linspace(0, 1, M+2)

        A = assemble_A(N, M, dx, dy)

        F = assemble_F(x, y, dx, dy, N, M, gL, gR, gB, gT)

        U = np.linalg.solve(A, F)
        U_exact = exact_solution(N, M, x, y)
        error = max(abs(U-U_exact))

        u = np.reshape(U, (N,M))
        u_exact = np.reshape(U_exact, (N,M))


        X, Y = np.meshgrid(x[1:N+1], y[1:M+1])
        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1, projection='3d')

        ax.plot_surface(X, Y, u, cmap='rainbow')

        # print (u)
        print(error)
        plt.show()

    Possion_solver(19, 19, gL, gR, gB, gT)
```

1.4 结果分析

当取 $h = 0.05$ 时，此时误差为 0.0013997692884775148，结果如下图所示：

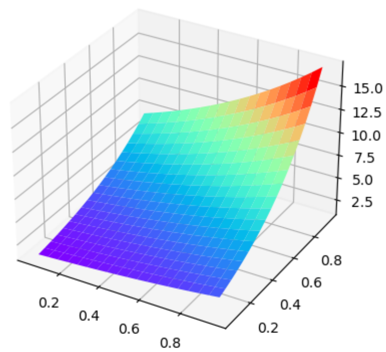

图 1: $h = 0.05$ 结果图

当取不同 h，得到的误差如下表所示：

表 1: 不同 h 的误差表

h	误差
$\frac{1}{10}$	0.005519939625335368
$\frac{1}{20}$	0.0013997692884775148
$\frac{1}{40}$	0.0003511099859592193
$\frac{1}{80}$	8.787626974093854e-05