第一次大作业：第三题 泊松方程

姓名：赵瑞博
学号：1800011355

2020 年 4 月 19 日

编程语言与轮子的搭建、调用

编程语言选择为 $Python$
调用的已有模块如下

- $matplotlib.pyplot$: 数据的可视化处理
- $math$: 基础函数 sin 与常数 π 的调用
- $copy$: 由于 $python$ 语言特性需要使用

调用自建轮子如下

- 矩阵类 $Matrix$: 包含基本的矩阵运算
- $zero\_mat$: 零矩阵生成函数，用于初始化

1 利用二阶中心差分格式求解泊松方程

原泊松方程与边界条件如下

$$\begin{cases} -u_{xx}-u_{yy}=2\pi^2\sin(\pi x)\sin(\pi y), 0<x,y<1 \\ u|_{\partial\Omega}=0, \Omega=(0,1)\times(0,1) \end{cases} \tag{1}$$

将 x, y 轴进行剖分，剖分数为 N，步长为 h，得到离散的网格点 $(x_i, y_j) = (ih, jh)$。用二阶微分的中心差分格式对二阶偏导数进行数值计算，离散后方程为：

$$\begin{cases} -\dfrac{u_{i-1,j}-2u_{i,j}+u_{i+1,j}}{h^2}-\dfrac{u_{i,j-1}-2u_{i,j}+u_{i,j+1}}{h^2}=f_{ij}=f(x_i,y_j) \\ 1\le i,j\le N-1,\ u_{ij}=0,\ i=0,N\ or\ j=0,N \end{cases} \tag{2}$$

问题转化为线性方程组 $Au = f$ 求解。令方程两侧同乘 h^2，且 $u_i = (u_{i,1}, \cdots, u_{i,N-1})^T, f_i = (f_{i,1}, \cdots, f_{i,N-1})^T, 1 \le i \le N-1$，方程组写为

$$\begin{pmatrix} A & -I & & \\ -I & A & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & A \end{pmatrix} \begin{pmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} h^2 f_0 \\ h^2 f_1 \\ \vdots \\ h^2 f_{N-1} \end{pmatrix} \tag{3}$$

其中

$$
A = \begin{pmatrix} 4 & -1 & & \\ -1 & 4 & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix} \tag{4}
$$

泊松方程数值求解第一步为系数矩阵与右侧列向量的构建，二者算法如下

```python
def coefficientmatrix(n):
    """
    系数矩阵生成，已包含边界条件‘u=0’
    input:  剖分数 n
    output: 系数矩阵A
    """
    n0 = (n - 1) * (n - 1)          # 系数矩阵的维数
    A = Matrix(zero_mat(n0, n0))
    for i in range(0, n0):          # 设置中央对角线
        A[i][i] = 4
    for j in range(0, n - 1):       # 设置里面的两条斜对角线
        for k in range(0, n - 2):
            kk = j * (n - 1) + k
            A[kk][kk + 1] = -1
            A[kk + 1][kk] = -1
    for k in range(0, n0 - n + 1): # 设置外面的两条斜对角线
        A[k][k + n - 1] = -1
        A[k + n - 1][k] = -1
    return A
```

```python
def fij(n, h, x0, y0):
    """
    线性方程组右侧列向量
    input: 剖分数n, 步长h, x0, y0
    output: Matrix f
    """
    n0 = (n - 1) * (n - 1)
    f = Matrix(zero_mat(n0, 1))
    for i in range(1, n):
        for j in range(1, n):
            k = (i - 1) * (n - 1) + j - 1
            f[k][0] = fxy(x0 + i * h, y0 + j * h) * h * h
    return f
```

接下来的任务是线性方程组求解，采用共轭梯度法求解，方法要求每次迭代依下方算法计算出相

应的搜索方向 $\mathbf{p}$、搜索步长 α、残差向量 $\mathbf{r}$、迭代值 $\mathbf{x}$:

$$
\begin{aligned}
\mathbf{p_k} = \mathbf{r_k} + \beta_{\mathbf{k-1}}\mathbf{p_{k-1}}, \beta_{\mathbf{k-1}} = -\frac{\mathbf{r_k^T A p_{k-1}}}{\mathbf{p_{k-1}^T A p_{k-1}}} \\
\alpha_k = \frac{\mathbf{r_k^T p_k}}{\mathbf{p_k^T A p_k}} \\
\mathbf{r_k} = \mathbf{b} - \mathbf{A x_k} \\
\mathbf{x_k} = \mathbf{x_{k-1}} + \alpha \mathbf{p_{k-1}}
\end{aligned}
\tag{5}
$$

完整共轭梯度求解方程组算法如下:

```python
def cg(A, b, x):
    """
    共轭梯度法求解线性方程组
    input: 系数矩阵A, 列向量b, 初始值x
    output: 解向量x
    """
    r = b - A * x          # r=b-Ax, r是梯度方向
    p = copy.deepcopy(r) # 搜索方向
    i = 0                # 记录迭代步数
    while ((r.T() * r)**0.5 > 1.e-10 and i < 100):
        a = (r.T() * p) / (r.T() * A * p) # 搜索步长
        x = x + a * p                     # 更新解向量
        r = r - a * A * p                 # 更新残差向量/梯度向量
        b = -(r.T() * A * p) / (p.T() * A * p)
        p = r + b * p                     # 更新搜索方向
        i = i + 1
    return x
```

最后一步，将解得的列向量 $\mathbf{u}$ 重新化为散点矩阵 $[u_{ij}]$。算法如下

```python
uij = []                          # 转换成二维矩阵
fT = f_1d.T().matrix[0]
zero = [0 for i in range(n + 1)]
uij.append(zero)
for i in range(n - 1):
    uij.append([0] + fT[i * (n - 1):(i + 1) * (n - 1):] + [0])
uij.append(zero)
```

如此计算得 $N=16,32,64,128$ 不同剖分数的数值结果将 $u(x,y)$ 作为深度后可视化后如下：

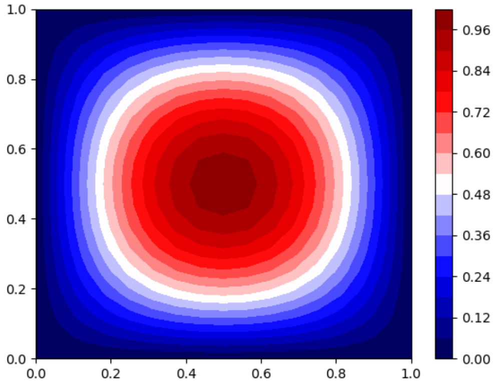

图 1: $N=16$

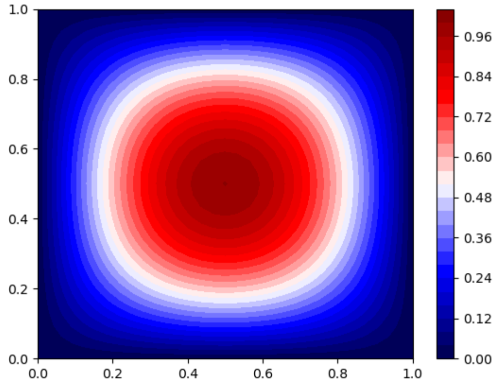

图 2: $N=32$

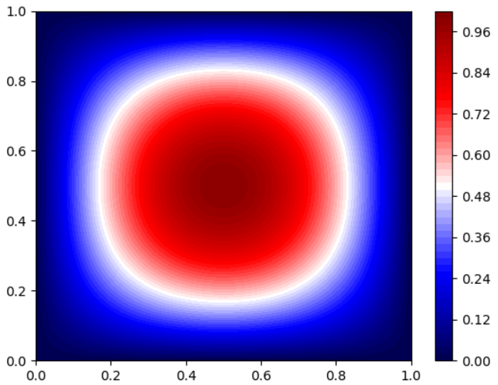

图 3: $N=64$

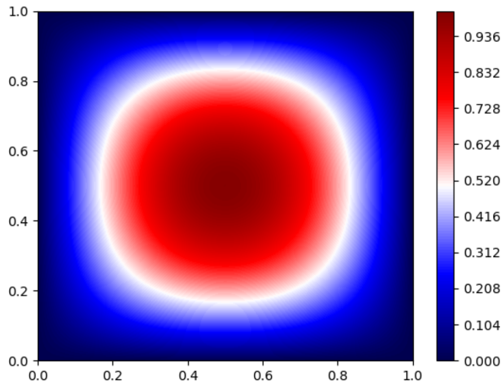

图 4: $N=128$

2 利用高斯积分计算每个单元格的 $\int_{\Omega_{i,j}}(\hat{u}-u)^2dxdy$

2.1 对单元格插值计算高斯点值

将每个 $h\times h$ 的单元格映射到标准单元格 $(-1,1)\times(-1,1)$ 上

$$\hat{u}_h(x,y)=\hat{u}(\xi,\eta) \tag{6}$$

在参考标准单元格上构造一个线性函数，需要以下四个基函数：

$$\phi_1=\frac{(1-\xi)(1-\eta)}{4},\phi_2=\frac{(1+\xi)(1-\eta)}{4},\phi_3=\frac{(1+\xi)(1+\eta)}{4},\phi_4=\frac{(1-\xi)(1+\eta)}{4} \tag{7}$$

构造的线性函数为

$$\hat{u}(\xi,\eta)=a_1\phi_1+a_2\phi_2+a_3\phi_3+a_4\phi_4,a_i\text{为待定系数} \tag{8}$$

代入单元 (i,j) 的四个顶点值 $u_{i,j}, u_{i+1,j}, u_{i+1,j+1}, u_{i,j+i}$，可解的线性插值后的标准单元下的函数：

$$\hat{u}(\xi,\eta) = u_{i,j}\phi_1 + u_{i+1,j}\phi_2 + u_{i+1,j+1}\phi_3 + u_{i,j+i}\phi_4 \tag{9}$$

由此可以很简单的求得高斯点 $(\pm\frac{1}{\sqrt{3}}, \pm\frac{1}{\sqrt{3}})$ 处的近似值。

2.2 高斯积分在单元格内计算 $\int_{\Omega_{i,j}}(\hat{u}-u)^2dxdy$

依上步可以计算的每个单元格内四个高斯点的值 $s_i(i=1,2,3,4)$. 依据高斯积分

$$\int_{\Omega_{i,j}}(\hat{u}-u)^2dxdy = \sum_{i=1}^{4} s_i \times \left(\frac{h}{2}\right)^2 = h^2 \times \frac{s_1+s_2+s_3+s_4}{4} \tag{10}$$

如是可以计算出一个单元格内的方差。具体实现算法如下

```python
def singlecell(u1, u2, u3, u4, xi, yj):
    """
    一个单元格对 (uij-u)^2 进行高斯积分
    input: 四结点值u, i,j 单元格坐标，亦是单元格左下结点指标
    output: s:(uij-uxy)^2
    """
    p1 = (2 + 3 ** 0.5) / 6
    p2 = (2 - 3 ** 0.5) / 6
    p3 = 1. / 6.                    # 高斯点处四个基函数可能取值
    d1 = h * (1 - 1 / (3 ** 0.5)) / 2
    d2 = h * (1 + 1 / (3 ** 0.5)) / 2 # 高斯点在原单元的映射相对左下差值
    s1 = (u1 * p1 + u2 * p3 + u3 * p2 + u4 * p3 - uxy(xi + d1, yj + d1)) ** 2
    s2 = (u1 * p3 + u2 * p2 + u3 * p3 + u4 * p1 - uxy(xi + d2, yj + d1)) ** 2
    s3 = (u1 * p3 + u2 * p1 + u3 * p3 + u4 * p2 - uxy(xi + d2, yj + d2)) ** 2
    s4 = (u1 * p2 + u2 * p3 + u3 * p1 + u4 * p3 - uxy(xi + d1, yj + d2)) ** 2
    s = (s1 + s2 + s3 + s4) / 4
    return s
```

3 计算误差

依据 2 中算法可以计算得每个单元格内的方差，如是只需对每个单元格进行求和再开方即可得到误差 E_{L_2}，具体算法如下

```python
def e_l2(uij, n, h, x0, y0):
    """
    L2范数下的误差估计(高斯积分)
    input: 解矩阵 uij,  切分数 n,  步长 h,  初值 x0,y0
    output: 误差值 el2
    """
    el2 = 0
    for i in range(n):
        for j in range(n):
            el2 += singlecell(uij[i][j], uij[i + 1][j], uij[i + 1][j + 1], uij[i][j +
                1], x0 + i * h, y0 + j * h)
```

```python
    el2 = el2 ** 0.5 * h
    return el2
```

可以计算的不同剖分数下的误差如下表所示 可见随着数值求解网格剖分愈加精细，L_2 范数下的

表 1: 不同剖分数对应误差

n	4	5	6	7
剖分数 $N=2^n$	16	32	64	128
E_{L_2}	0.0565978	0.0283301	0.0141690	0.0070850

误差越小，这是显而易见的。另外可以看到，4 个误差与剖分数均程等比数列，且公比互为倒数，这不失为一个规律。