

微分方程数值解计算实习 Lecture 6

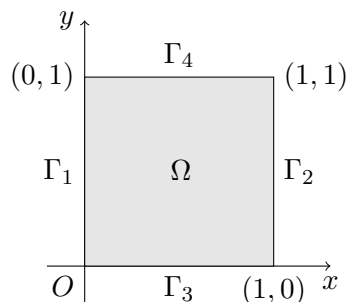
朱荃凡

(吉林大学数学系计算唐班)

2023 年 4 月 13 日

1 问题重述

如图所示, Ω 表示 $[0, 1]^2$ 的区域, $\Gamma_1, \Gamma_2, \Gamma_3, \Gamma_4$ 是它的四条边:



利用 Lagrange 双线性元求解区域 Ω 区域上的偏微分问题:

$$\begin{cases} -\Delta u - 2\pi^2 u = -2\pi^2 xy, & \text{in } \Omega, \\ u(x, y) = 0, & \text{in } \Gamma_1, \Gamma_3, \\ \partial_x u(x, y) = y - \pi \sin(\pi y), & \text{in } \Gamma_2, \\ \partial_y u(x, y) = x - \pi \sin(\pi x), & \text{in } \Gamma_4. \end{cases} \quad (1.1)$$

其相应的的真解为

$$u^* = xy + \sin(\pi x) \sin(\pi y). \quad (1.2)$$

2 算法设计

二维区域上的有限元问题相对一维会复杂很多, 包括但不限于网格剖分, 边值条件处理, 积分公式. 我们分成几个部分来讨论.

2.1 转化为变分形式

记函数

$$\begin{aligned} g_2(x, y) &= y - \pi \sin(\pi y), \text{ in } \Gamma_2, \\ g_4(x, y) &= x - \pi \sin(\pi x), \text{ in } \Gamma_4. \end{aligned} \quad (2.1)$$

再记函数空间

$$V(\Omega) = \{v \in H^1(\Omega), v|_{\Gamma_1 \cup \Gamma_3} = 0\}. \quad (2.2)$$

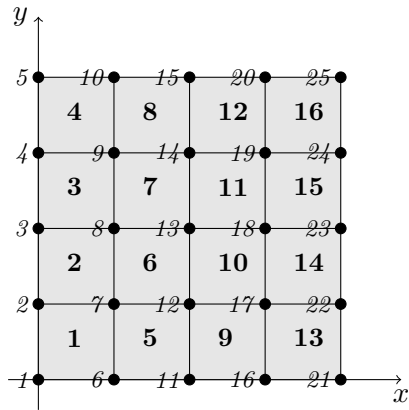
那么初边值问题 (1.1) 的变分问题为: 找到 $u \in V(\Omega)$, 使得

$$\int_{\Omega} \Delta u \cdot \Delta v - 2\pi^2 uv \, dx = \int_{\Omega} f v \, dx + \int_{\Gamma_2} g_2 v \, ds + \int_{\Gamma_4} g_4 v \, ds, \quad \forall v \in V(\Omega).$$

2.2 网格剖分

在一维有限元中, 无论网格剖分的均匀与否, 区间上的节点位置和其解向量中的次序是完全对应的. 这使我们索引刚度矩阵和右端项元素时, 十分容易. 但是在二维有限元中, 这种对应关系不存在了, 因此我们需要额外的矩阵去单独保存剖分后点, 边和单元的信息.

在此问题中, 我们仍使用均匀剖分. 设剖分数为 N , 此时 x 轴和 y 轴均被 N 等分, 区域 Ω 上有 N^2 个小正方形单元和 $(N+1)^2$ 个剖分节点. 下面以 $N=4$ 为例, 展示我们如何储存剖分信息:



首先, 将节点从下往上, 从左往右依次编号, 序号从 1-25. 对小单元也是类似的编号顺序. 节点矩阵 p 是一个 2×25 的矩阵, 其中第 i 列表示序号为 i 的节点的横纵坐标, 如下所示:

$$p = \begin{pmatrix} 0 & 0 & 0 & \cdots & 0.5 & 0.5 & 0.5 & \cdots & 1 & 1 \\ 0 & 0.25 & 0.5 & \cdots & 0.25 & 0.5 & 0.75 & \cdots & 0.75 & 1 \end{pmatrix}. \quad (2.3)$$

单元矩阵 t 是一个 4×16 的矩阵, 其中第 i 列表示序号为 i 的小单元, 四行表示其四个节点在 p 中的索引, 如下所示:

$$t = \begin{pmatrix} 1 & 2 & 3 & \cdots & 7 & 8 & \cdots & 18 & 19 \\ 2 & 3 & 4 & \cdots & 8 & 9 & \cdots & 19 & 20 \\ 7 & 8 & 9 & \cdots & 13 & 14 & \cdots & 24 & 25 \\ 6 & 7 & 8 & \cdots & 12 & 13 & \cdots & 23 & 24 \end{pmatrix}. \quad (2.4)$$

边矩阵 e 较为特殊, 我们在处理边值条件的时候只会用到最外侧的边的信息, 内部边是不需要储存的, e 是一个 3×16 的矩阵, 每一列代表区间边界上的一条小边, 第一行和第二行是这条小边的起点和终点在 p 中的索引. 第三行表示小边属于 $\Gamma_i (1 \leq i \leq 4)$ 中的哪条大边, 属于 Γ_1 就即成 1, 属于 Γ_2 就即成 2, 属于 Γ_3 就即成 3, 属于 Γ_4 就即成 4. 如下所示:

$$e = \begin{pmatrix} 1 & 2 & 3 & \cdots & 23 & 24 & 1 & \cdots & 15 & 20 \\ 2 & 3 & 4 & \cdots & 24 & 25 & 6 & \cdots & 20 & 25 \\ 1 & 1 & 1 & \cdots & 2 & 2 & 3 & \cdots & 4 & 4 \end{pmatrix}. \quad (2.5)$$

有了剖分信息 p, t, e 以后, 可以极大地简化剩余部分地编程难度.

2.3 数值积分公式

在此算法中, 我选择了正方形区域上的九点积分公式, 它是由一维高斯积分公式直接推导出来的, 以标准区间 $[-1, 1] \times [-1, 1]$ 为例:

$$\int_{-1}^1 \int_{-1}^1 f(x, y) dx dy \approx \int_{-1}^1 \sum_{i=1}^3 \omega_i f(x_i, y) dy \approx \sum_{i,j=1}^3 \omega_i \omega_j f(x_i, y_j). \quad (2.6)$$

其中 x_i, y_j 为一维三点高斯积分节点 $-\frac{\sqrt{15}}{5}, 0, \frac{\sqrt{15}}{5}$. w_i 为其权重 $\frac{5}{9}, \frac{8}{9}, \frac{5}{9}$. 易见此公式对于所有形如 $x^m y^n (m, n \leq 5)$ 的多项式都是严格取等的. 本题而言中, 四点高斯积分公式精度也够, 不过为了代码的通用性, 还是采用了九点.

2.4 其余部分

2.4.1 生成刚度矩阵和右端项

这部分和一维有限元是比较相似的. 对小单元数进行循环, 计算单元刚度矩阵和局部右端项, 找到对应节点位置并更新刚度矩阵和右端项. 由于每个小单元上有四个基函数, 所以单元刚度矩阵的大小是 4×4 . 由于是等距剖分, 所以每个单元刚度均一致, 我们可以事先算出来.

此外, 这一部分我在代码的调试过程还发现了有意思的内容, 但是和本次主题关系不大, 因此放在报告最后加以说明.

2.4.2 处理边值条件

边值条件的处理在原理上不难理解, 和一维区别不大, 但是在编程上难度较大.

处理 Neumann 边值条件时, 边矩阵 e 中提供了 Γ_2 和 Γ_4 上小边的位置信息, 在此基础上我们需要像一维有限元那样去更新右端向量的数值. 二维矩形区域的拉格朗日型元在边界上仍然是线性的, 计算基函数的函数值只需要做一个线性变换即可.

Dirichlet 边值条件的处理过程不再赘述.

2.4.3 画出函数图像

在 Matlab 中想画出三维图像需要专门的函数 `surf()` 或者 `mesh()`. 这两个函数均需要输入矩形区域的网格节点坐标和节点处的函数值. 观察画出后图像的棱角可以发现有意思的一点: Matlab 在绘制此类图形时会对网格在进行一次剖分, 把每个矩形区域分成两个三角形, 然后每个三角形区域上绘制一个平面即可.

3 程序结果

3.1 数值解图像

这里展示了剖分数 $N = 5$ 和 $N = 10$ 时的数值解图像, 以及 $N = 20$ 时的真解图像 (显然, 画真解图像也是要剖分的).

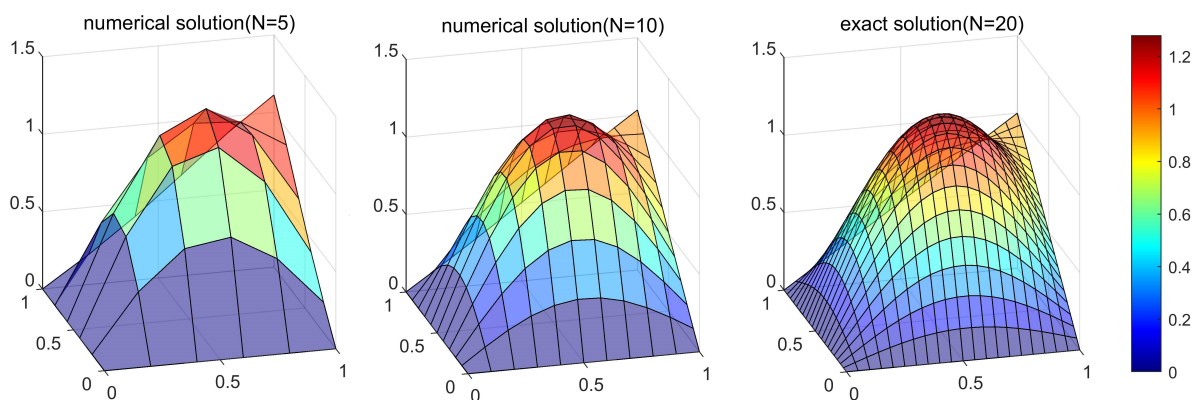


图 1: 真解和数值解

3.2 误差和收敛阶

取剖分数 $N = 5 \times 2^{n-1} (1 \leq n \leq 7)$, 分别计算 L^2 误差和 H^1 误差. 这里只展示了 L^2 模的误差及其收敛速度, 需要注意的是横轴代表的含义是”自由度”, 可以近似的认为是节点数量 $(N+1)^2$. 所以有些图像看起来并不均匀:

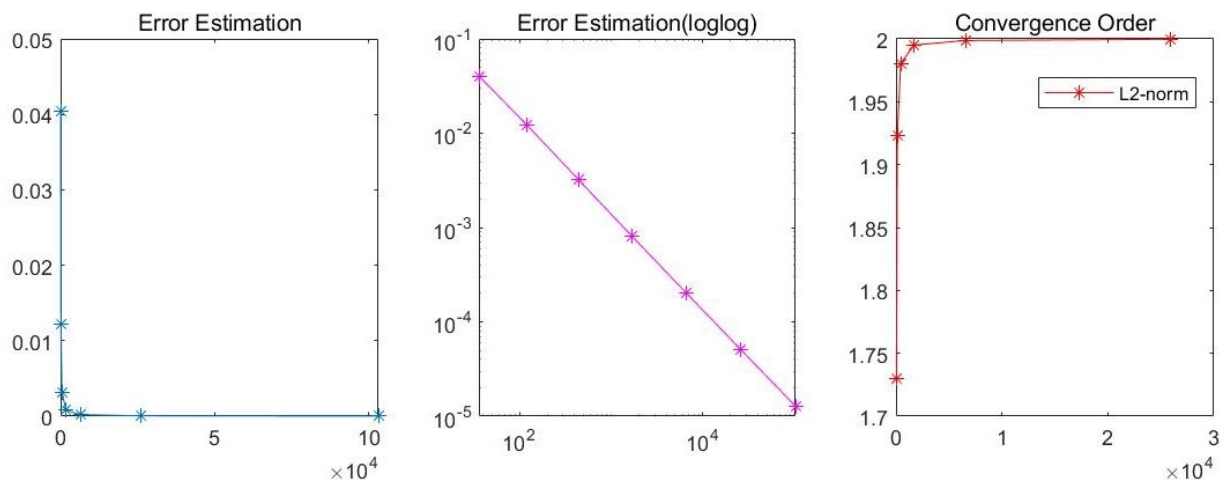


图 2: L^2 误差和收敛阶

3.3 刚度矩阵条件数和发散阶

剖分数如上所示, 这里展示了矩阵条件数和自由度的双对数图和发散阶: 有点出乎意料, 发散阶

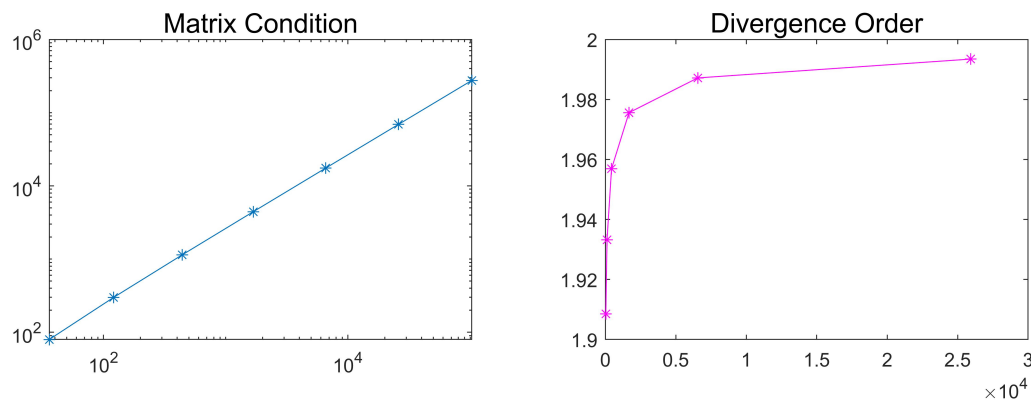









图 3: 矩阵条件数和发散阶

仍然为二阶, 这说明二阶矩形剖分后得到的线性方程组在数值计算上依旧是相对稳定的.

4 关于提升运行速度的一点思考

计算收敛阶的时候, 剖分数 $N = 5 \times 2^{n-1}$ 是指数级增长的, 节点数的增长速度就更快了. 当 $n = 6$ 时, 我的程序需要算 9 秒, $n = 7$ 时要算 3 分钟, $n = 8$ 时, 要算 50 分钟!

那么究竟是哪一步花了过长的时间多呢, 如下是程序运行时的分析表:

代码	调用次数	总时间(秒)	% 时间	时间图
A=stiffnessMatrix(p,e,t,h);	1	7.652	85.7%	
[L2_error,H1_error]=errorEstimate(u,p,t,h);	1	0.631	7.1%	
F=rightHands(p,e,t,h);	1	0.416	4.7%	
Mat_con=cond(A,2);	1	0.134	1.5%	
u=solveAF(A,F,N,pde,boundary_method);	1	0.073	0.8%	
		0.021	0.2%	
A(nd,nd)=A(nd,nd)+K;	25600	7.608	99.4%	
nd=t(1:4,iel);	25600	0.043	0.6%	
end	25600	0.002	0.0%	
a1=2/3-2*pi^2*h^2/9;	1	0.000	0.0%	
A=sparse(sDof,sDof);	1	0.000	0.0%	
		0.000	0.0%	

可以看到在生成刚度矩阵, 尤其是组装刚度矩阵的时候, 占了格外长的时间. 根据结果不难猜测是因为生成刚度矩阵使用到了 `sparse()` 函数. 好处是可以节约空间, 坏处是像这样, 在索引元素位置是会花费大量的时间.

如果我们把 `sparse()` 方法改回 `full()` 方法, 那么就能在 3 秒内跑完程序. 不过需要注意, 在解方程组之前仍然需要改回稀疏矩阵, 并且不能计算条件数: 从图中看到, 生成刚度矩阵的时间被大大减

函数名称	调用次数	总时间(秒)	自时间*(秒)	总时间图 (深色条带 = 自时间)
FEM_2d_1p_sq_ND	1	2.781	0.002	
FEM_2d_1p_sq_ND>run_main	1	2.771	1.568	
FEM_2d_1p_sq_ND>errorEstimate	1	0.596	0.242	
FEM_2d_1p_sq_ND>rightHands	1	0.417	0.197	
FEM_2d_1p_sq_ND>square_quadrature	179200	0.279	0.279	
FEM_2d_1p_sq_ND>stiffnessMatrix	1	0.097	0.097	
FEM_2d_1p_sq_ND>get_integration_points	51202	0.074	0.074	
FEM_2d_1p_sq_ND>solveAF	1	0.073	0.073	
FEM_2d_1p_sq_ND>tune_sol_gradient	25600	0.043	0.043	

少了, 当时我感到很兴奋, 直接就令 $n = 7$ 又算一次.

错误使用 **zeros**

请求的 103041x103041 (79.1GB)数组超过预设的最大数组大小。创建大于此限制的数组可能需要较长时间，并且会导致 MATLAB 无响应。

出错 **FEM_2d_1p_sq_ND>stiffnessMatrix** (第 107 行)

A=zeros(sDof,sDof);

出错 **FEM_2d_1p_sq_ND>run_main** (第 48 行)

A=stiffnessMatrix(p,e,t,h);

出错 **FEM_2d_1p_sq_ND** (第 24 行)

[sDof, L2_error,H1_error, Mat_con]=run_main(N,pde)

结果就得到了如上的报错-内存容量不够. 所以我们可以得到结论, 使用 `sparse()` 方法就是用时间换空间, 使用 `full()` 方法就是用空间换时间, 在超大内存的情况可以很好的节约时间. 那有没有既节约时间, 又节约空间的方法呢?

答案是有的, Matlab 对矩阵运算的优化很好, 如果我们能把生成刚度的循环操作改成矩阵运算, 那么就能大大节约时间. 我曾经在张凯老师的有限元程序里见过, 40 万个节点, 只需要不到一分钟就能跑完 (在我程序里, $n=7$ 也才 10 万个节点左右). 这两天我花了很多时间尝试在二维拉格朗日型元中实现这种方法, 但是做不到...