有限元方法 大作业

郑滕飞 2401110060

目录

1 准备工作

1.1 问题描述与边界处理

考虑如下的二维 Dirichlet 问题：

$$-\nabla\cdot(\alpha(x,y)\nabla u)\big|_{\Omega}=f,\quad u\big|_{\partial\Omega}=g$$

其中 $\alpha(x,y)$ 满足一致椭圆条件，即有正上下界。

通过分部积分，考虑 $V=\{v\in W^1_\infty(\Omega)\mid v\big|_{\partial\Omega}=0\}$ 设

$$a_0(u,v)=\iint_\Omega\alpha(x,y)\nabla u\cdot\nabla v\mathrm{d}x\mathrm{d}y$$

由一致椭圆条件，存在唯一 $u-g\in V$ 使得

$$a_0(u-g,v)=(f,v)-a_0(g,v)$$

对一切 $v\in V$ 成立，

选取有限元空间 V_h，满足分片为 P_3-Hermite 单元 (中间点取在中心)，且边界为 0。设三角形数为 M，结点数 (含边界) 为 N，则结点基为 $v_1,\dots,v_{3N+M}$，并记 u 在其上的插值为 $u_1,\dots,u_{3N+M}$、g 为 $g_1,\dots,g_{3N+M}$。进一步设其中固定的下标集合为 B，其余为 $\tilde{B}$，记 $a_0(v_i,v_j)=A_{ij}$、$(f,v_i)=f_i$，方程可化为

$$\forall i\in B,\quad u_i=g_i$$

$$\forall i\in\tilde{B},\quad\sum_j A_{ij}(u_j-g_j)=f_i-\sum_j A_{ij}g_j$$

由于 B 中 $u=g$ 抵消，且 $j\in\tilde{B}$ 时的左右抵消，可不妨令 $g_j=0$，第二个方程可最终化为 (下标表示分块)

$$\forall i\in\tilde{B},\quad A_{\tilde{B}}u_{\tilde{B}}=f_{\tilde{B}}-(Ag)_{\tilde{B}}$$

由此，固定网格的有限元算法基本流程为 (优化 g 的存储)：

- 计算刚度矩阵 A；
- 计算向量 f；
- 初始化 $u=0$，将边界点处的 u_i 设置为 g_i，计算 $r=f-Au$；
- 在 $\tilde{B}$ 对应行列中求解 $Au=r$，最终得到 u。

接下来的计算中，我们约定前 N 个分量代表结点处的值，$N+1$ 到 $2N$ 代表结点处的 x 方向导数，$2N+1$ 到 $3N$ 表示结点处的 y 方向导数，$3N+1$ 到 $3N+M$ 表示单元中心处的值。

值得注意的是，固定的 B 并不只在 1 到 N 中出现：由于边界处的 g 固定，且单元会涉及顶点的两方向导数，切向导数事实上是固定的。此外，我们考虑的所有情况中边界均为 x 或 y 方向，因此不会出现线性组合固定的情况，可以直接控制其为对应的 g_x 或 g_y。

1.2 单元与数值积分

进行完上述分析后，我们需要估算积分已确定矩阵 A 与向量 f。根据 iFEM 包中的存储方式，网格以两个数组进行存储，node 为一组坐标，elem 则代表每个三角形三个顶点 (逆时针排序) 在 node 中的下标。

首先，需要给出 Hermite 单元的结点基。设三个顶点为 (x_1,y_1)、(x_2,y_2)、(x_3,y_3)，由于平移不影响结果，不妨设 $x_1=y_1=0$ (最终所有估算将 x,y 减去对应的 x_1,y_1 即可)，剩下两顶点记为 (a,b)、(c,d)，

图 1: 一组结点基

$$\frac{1}{(bc-ad)^3}(b(c-x)+dx-cy+a(-d+y))\left(b^2(c-x)(c+2x)+a^2(d-y)(d+2y)+a(d-11y)(dx-cy)-2(dx-cy)^2+b(c-11x)(-dx+cy)-ab(2cd+dx+cy-4xy)\right)$$

$$\frac{(dx-cy)\left(7b^2(c-x)x+7a^2(d-y)y+b(3c+7x)(dx-cy)-a(3d+7y)(dx-cy)+2(dx-cy)^2-7ab(dx+cy-2xy)\right)}{(bc-ad)^3}$$

$$\frac{(bx-ay)\left(b^2(3c-2x)x+a^2(3d-2y)y+7b(c-x)(dx-cy)-7a(d-y)(dx-cy)+7(dx-cy)^2+ab(-3dx-3cy+4xy)\right)}{(bc-ad)^3}$$

$$\frac{\left(x\left((bc-ad)^2-\left(b^2c-2b(a+c)d+ad^2\right)x\right)-2\left(bc^2+a^2d\right)xy+ac(a+c)y^2\right)(b(c-x)+dx-cy+a(-d+y))}{(bc-ad)^3}$$

$$-\frac{1}{(bc-ad)^3}(-dx+cy)\left(2a^3y(-d+y)+bcx(b(c-x)+2dx-2cy)+a^2\left(d(2b+d)x+2(b(c-2x)+dx)y-3cy^2\right)+a\left(x\left(-d^2x+2b^2(-c+x)-2bd(c+x)\right)+4bcxy+c^2y^2\right)\right)$$

$$\frac{(bx-ay)\left(b^2cx(-c+x)+a(2cd+dx-3cy)(dx-cy)-2c(dx-cy)^2+2b(c-x)\left((a-c)dx+c^2y\right)-a^2\left(d^2x-2dxy+cy^2\right)\right)}{(bc-ad)^3}$$

$$\frac{\left(bd(b+d)x^2+\left((bc-ad)^2-2\left(b^2c+ad^2\right)x\right)y-\left(bc^2+a^2d-2ac(b+d)\right)y^2\right)(b(c-x)+dx-cy+a(-d+y))}{(bc-ad)^3}$$

$$\frac{1}{(bc-ad)^3}(dx-cy)\left(2b^3x(-c+x)+ady(-2dx+a(d-y)+2cy)+b\left(d^2x^2-2ad(a+c-2x)y+\left(2a^2-2ac-c^2\right)y^2\right)+b^2\left(-3dx^2+c(c+2x)y+2a(dx+cy-2xy)\right)\right)$$

$$-\frac{(bx-ay)\left(b^2\left(dx^2+c(c-2x)y\right)+b\left(d^2(2c-3x)x-2cd(a+c-2x)y-c(-2a+c)y^2\right)+d\left(a^2(d-y)y-2a(d-y)(dx-cy)+2(dx-cy)^2\right)\right)}{(bc-ad)^3}$$

$$\frac{27(bx-ay)(-dx+cy)(b(c-x)+dx-cy+a(-d+y))}{(bc-ad)^3}$$

则利用 Mathematica 计算得到一组结点基 $p_{1,\dots,10}(x,y)$ 如图 1 。可以发现，其中具有不少共同部分，例如三边方程 $b(c-x)+dx-cy+a(y-d)$、$dx-cy$、$bx-ay$ 与行列式 $\triangle=ad-bc$，可将它们先行算出以简化计算。

在已知结点基后，我们需要先在在单元中对 $p_ip_j\alpha$ 与 p_if 进行数值积分，再将结果适当累加生成最终的 A 与 f。利用 github.com/bempp/bempp-cl/blob/main/bempp/api/integration/triangle_gauss.py，我们选取 16 个点进行精度为 7 阶的数值积分，并设置对应权重 (按照代码中所给的权重，最终每个三角形应再乘三角形面积的两倍)。

对结果生成，我们先计算 p_i、$(p_i)_{x,y}$、α 与 f 在每个积分点的值，随后生成每个单元中所有 $((p_i)_x(p_j)_x+(p_i)_y+(p_j)_y)\alpha$ 与 p_if 的积分结果，并在每次生成后进行适当的累加——利用 A 的对称性，计算 A 时事实上只需要生成 16 个部分中的 10 个，也即对应 $9\times 6+3\times 3+1=64$ 个积分结果，而计算 r 时则需要另外 10 个。

1.3 误差估算

最后，我们需要进行 H^1、L^2、W^1_∞ 与 L^∞ 范数的估算。由于我们有 $M+N$ 个值、N 个对 x 的偏导与 N 个对 y 的偏导，假设每个分量的数值解与真解误差为 e_i，按之前的下标约定，我们将误差范数估算为

$$\|e\|^2_{L^2}\approx\frac{|\Omega|}{M+N}\left(\sum_{i=1}^{N}e_i^2+\sum_{i=3N+1}^{3N+M}e_i^2\right)$$

$$\|e\|^2_{H^1}\approx\|e\|^2_{L^2}+\frac{|\Omega|}{N}\sum_{i=N+1}^{2N}e_i^2+\frac{|\Omega|}{N}\sum_{i=2N+1}^{3N}e_i^2$$

$$\|e\|_{L^\infty}\approx\max_{i\notin(N,3N]}|e_i|$$

$$\|e\|_{W_1^\infty}\approx\max_{i}|e_i|$$

这里 $|\Omega|$ 表示区域面积。

值得注意的是，这样估算的前提是网格是拟一致的，因此可以近似认为所有三角形面积相等，对 L^2 范数的估算无需考虑系数。之后使用自适应方法进行部分加密时不能再以此进行简单估算，我们将在对应节中叙述估算方式。

另一个可以考虑的误差形式是能量范数误差 $a(u-u_I,u-u_I)$，可以直接通过 e^TAe 计算。不过，由于已经计算了 H^1 误差，能量范数误差可被控制，这里不进行计算。

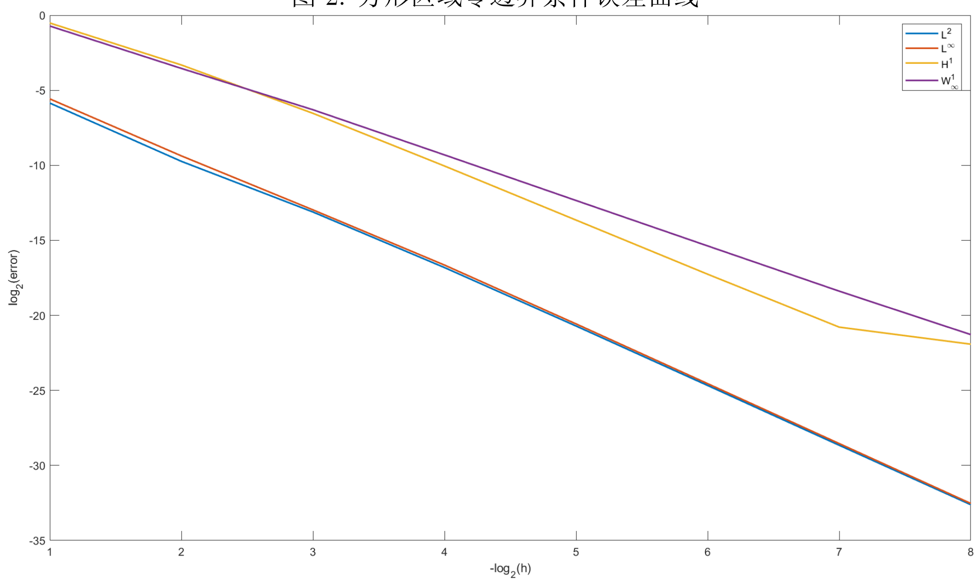
2 一致网格

2.1 方形区域

考虑方形区域 $[-1,1]^2$，并固定 $\alpha(x,y)=1+0.5\sin(\pi x)$，我们以不同函数为例测试算法的收敛性。假设初始网格为 iFEN 包以 0.5 为尺度自动生成的网格，每次进行一致加密（即将每个小三角形按三边中点连线剖分为四个，尺度对应除以 2）。

先考虑边界处恒为 0 的例子，$u(x,y)=\sin(\pi x)\sin(\pi y)$，计算出对应的 f、g 与导数后，在不同尺度的网格进行拟合的误差如图 2（作图横轴为尺度 2 为底对数的相反数，纵轴为误差的 2 为底对数）。

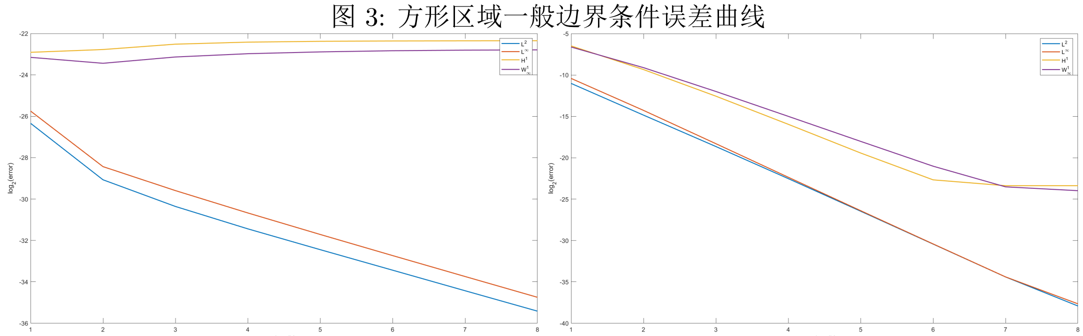
图 2: 方形区域零边界条件误差曲线



直接拟合可以发现，L^2 与 L^∞ 的数值误差在 3.8 阶左右，而 H^1 与 W_1^∞ 的数值误差在 3 阶左右，符合三次多项式逼近的理论结果。

对边界不恒为 0 的例子，考虑以多项式为真解的 $u(x,y)=x^2+y^2$ 与 $u(x,y)=\sin(x+y)$，得到的结果如图 3 。

图 3: 方形区域一般边界条件误差曲线



对于多项式真解，由于误差基本全部由数值积分与数值计算过程引起，其初始就达到了 1e-7 级别的极低误差，且网格加密时并不能保证误差的显著降低。而在 $\sin(x+y)$ 的模拟过程中可以发现，在误差下降到一定量级后，数值产生的误差将更加明显，从而影响收敛阶数。不过，总体还是能直接拟合出 L^2、L^∞ 的四阶误差与 H^1、W_∞^1 的三阶误差。最终的 H^1 与 W_1^∞ 在三个例子中均收敛到了 3e-8 左右，而 L^2 与 L^∞ 在可见范围内并未收敛，一般至少达到了 1e-10，可接近数值精度的极限。

值得一提的是，在三个例子最终的尺度 2^{-8} 的网格中，进行求解的时间都在 30 秒左右，其中核心时间为构造 A 与求解，它们分别占用 14 秒、11 秒。

2.2 L 形区域

下方讨论的 L 形区域均指区域 $[-1,1]^2\backslash[0,1]\times[-1,0]$，也即方形区域去除第四象限的部分。

考虑极坐标下的真解

$$u(r,\theta)=(1-r^2)r^{2/3}\sin\frac{2\theta}{3}$$

并取定 $\alpha=1$。直接计算可知

$$u_x(r,\theta)=-\frac{2}{3}r^{-1/3}\sin\frac{\theta}{3}\left(3r^2\cos\frac{2\theta}{3}+3r^2\cos\frac{4\theta}{3}-r^2+1\right)$$

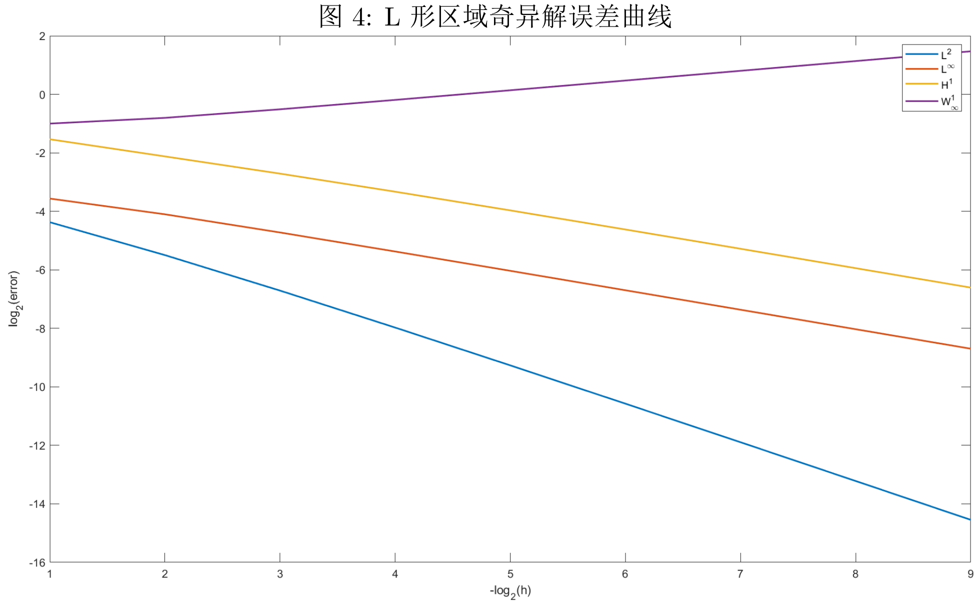
$$u_y(r,\theta)=-\frac{2}{3}r^{-1/3}\cos\frac{\theta}{3}\left(3r^2\cos\frac{2\theta}{3}-3r^2\cos\frac{4\theta}{3}+r^2-1\right)$$

$$-\triangle u(r,\theta)=\frac{20}{3}r^{2/3}\sin\frac{2\theta}{3}$$

注意到，在 $\theta=0$ 的边界上，$r\to0$ 时 $u_x\to0$，而 $\theta=\frac{3\pi}{2}$ 的边界上，$r\to0$ 时 $u_y\to0$，由此，所有的边界条件仍然是可以良好定义的。不过，事实上可发现 $\theta=0$ 和 $\theta=\pi$ 时、$\theta=\pi/2$ 和 $\theta=3\pi/2$ 时导数并不相同，因此其可微性无法延拓到原点。之后计算误差时，我们**规定** 0 处的 x,y 方向导数均为 0，也即边界方向逼近时的导数，避免出现无穷。这样做事实上相当于规定了 0 点误差为 0，以其他导数可以连续延拓的点刻画逼近性。

考虑初始网格为尺度 0.5 的方形网格去掉第四象限部分，并不断进行一致加密。在不同尺度的网格对此例子进行拟合的误差如图 4 。

图 4: L 形区域奇异解误差曲线



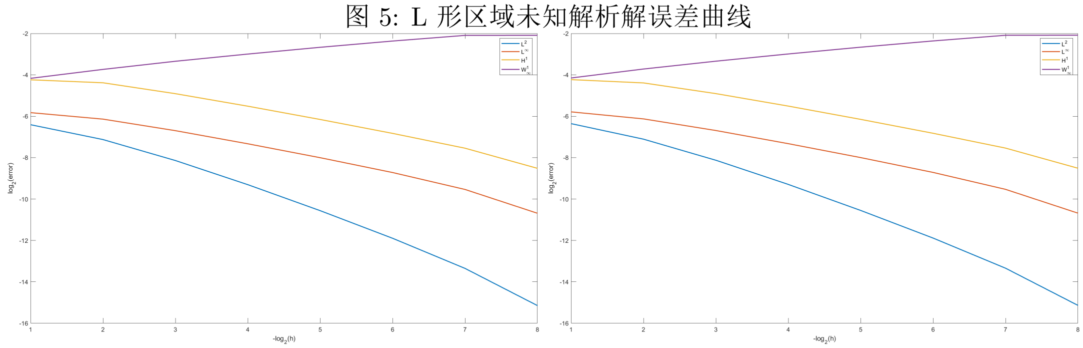
这里最终网格尺度为 2^{-9}，求解时间约 68 秒。拟合可发现，L^2 范数的误差只有 1.25 阶的下降，H^1 与 L^∞ 均不足一阶，而 W^1_∞ 范数则反而在上升。这足以看出，对于性态并不好的真解与并不好的区域，此模拟方法的效果可能大幅度下降。对 W^1_∞ 范数上升的讨论将在下一节进行。

2.3 未知解析解情况

最后，我们考虑取定 $f=1$、$g=0$，对给定的 a 进行数值求解的情况。为了在此情况中估算误差，我们利用了 iFEM 包提供的加密功能的特性：原有的结点与编号并不会在加密过程中改变。由此，我们只需要先进行若干次加密，在最终网格中得到数值解作为真解估计，再重复之前的加密过程，每次将对应编号顶点的 u、u_x 与 u_y 作差并估计范数即可。

此方法对 L^2、L^∞ 误差的估计比之前更为粗略，因为完全忽略了单元中心点处对 u 的值的估算。不过，这样的简单估算也足以看出收敛性了。对 $\alpha=1$ 与 $\alpha=\sin(x+y)$，进行拟合的误差作图如图 5 。

图 5: L 形区域未知解析解误差曲线



由于解析解未知，这里对误差的估算未必准确，不过有三件事仍然成立：L^2 误差以略高于一阶的速度下降、L^∞ 与 H^1 误差以低于一阶的速度下降、W_1^∞ 误差上升。

这里，前几个误差的下降速率低是因为区域的不光滑性。而对 W_1^∞ 误差，我们固定一点处观察其 u_x 值的变化，例如考虑 $(-1,0)$ 处，其值在不同层次网格的变化为：

```
0.4884
0.4996
0.5057
0.5083
0.5093
0.5097
0.5098
0.5099
0.5099
```

其他固定点处事实上也类似。由此，u_x 在任何一点处事实上是可以**收敛**的，但其会在区域的角点附近产生**振荡**，且振荡幅度会随网格精细反而增大，这就导致了 W_1^∞ 范数的上升。

3 自适应网格

3.1 算法实现

为在之前的估算算法上进一步实现自适应网格，我们采用课上介绍的误差估计子

$$\mathcal{E}^2(u_h,T)=h_T^2\|f+\nabla\cdot(\alpha\nabla u_h)\|^2_{L^2(T)}+\sum_{e\in\partial T}|e|\|[\alpha\nabla u_h]\|^2_{L^2(e)}$$

不过，我们需要进行一定的改进：由于真解并未保证边界的法向导数为 0，要求边界边 $\nabla u_h=0$ 是不合理的，因此，我们假定边界边的 $[\alpha\nabla u_h]$ 恒为 0。

为了估算此误差估计子，我们分为内部 R_A 与边界 R_J 的两个部分。

对 R_A，直接将其展开得到

$$(f+\nabla\cdot(\alpha\nabla u_h))^2=(f+\alpha_x(u_h)_x+\alpha_y(u_h)_y+\alpha((u_h)_{xx}+(u_h)_{yy}))^2$$

将 α_x 与 α_y 作为输入，由于 u_h 各个基的坐标已知，算出每个单元每个基的在数值积分点的结果后进行适当线性组合即可得到每个单元此积分的估算。虽然过程较为复杂，但总体思路是清晰的，与之前计算 A 与 r 时非常类似。在计算完数值积分后乘以单元面积即得到了第一部分的估算。

对 R_J，采用 iFEM 自带的单元结构函数 auxstructure，即可得到其所有边界与对应的信息，只要在每个边界计算结果后对每个单元进行累加即可。为了简单起见，我们只计算中点处

$$\alpha^2((u_x^+-u_x^-)n_x+(u_y^+-u_y^-)n_y)^2$$

的值 (由于这里相差负号不影响，可以任意决定何侧为正、任取法向量)，并将它乘边长作为 R_J^2 积分的估算，再乘一次边长得到第二部分的估算。

两部分求和后，我们得到了每个单元上的误差估计子。取定 $\theta=0.7$，我们将误差估计子按从大到小排序，并逐个标记，直到被标记单元的误差估计子平方和大于所有误差估计子平方和的 θ 倍。最后，利用 iFEM 自带的 bisect 功能，我们将每个标记的单元一分为二，得到新的网格。

3.2 L 形区域

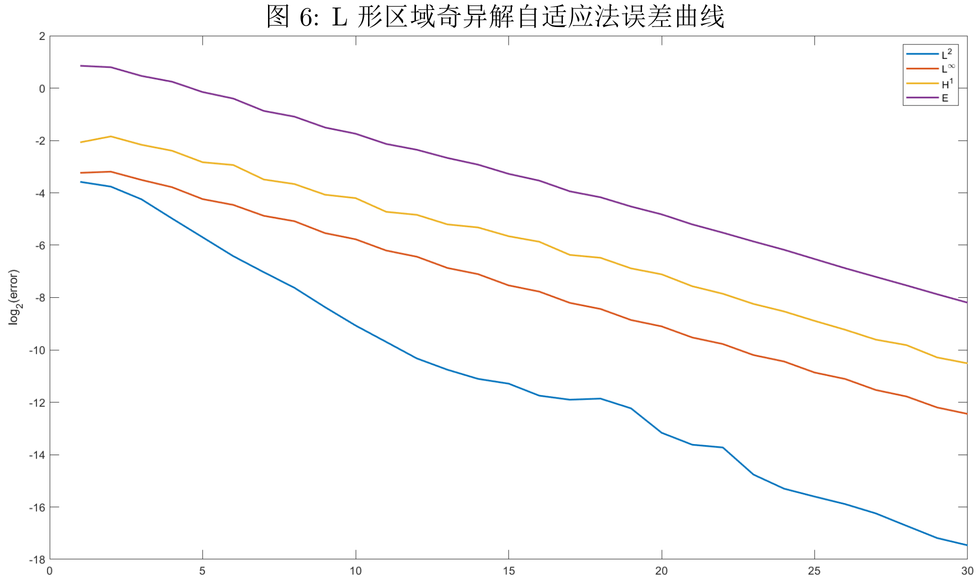
仍对之前的 L 形区域考虑。由于我们已经知道了 W_1^∞ 是无意义的，不再让它参与作图，而是作出

$$\mathcal{E}(u_h,\mathcal{T})=\sqrt{\sum_{T\in\mathcal{T}}\mathcal{E}(u_h,T)}$$

随自适应网格的变化。

对 2.2 节的例子进行 30 次求解并迭代网格，误差曲线如图 6 ，最终网格如图 7 。

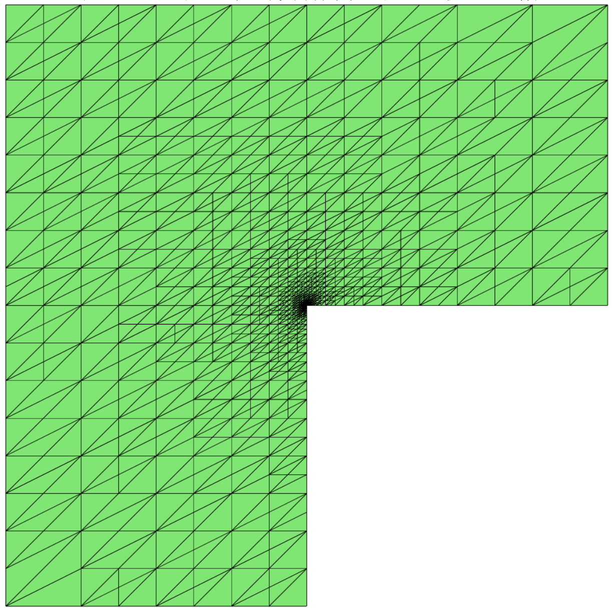
图 6: L 形区域奇异解自适应法误差曲线



可以发现，这里 30 次迭代就达到了 1e-5 量级的 L^2 误差，效果与一致网格的最终迭代后相仿，而 L^∞ 与 H^1 的误差更低。此外，最后一次迭代的用时仅为 0.04 秒，全程用时是一致网格最后一次迭代的千分之一量级，这足以体现自适应方法的优势。

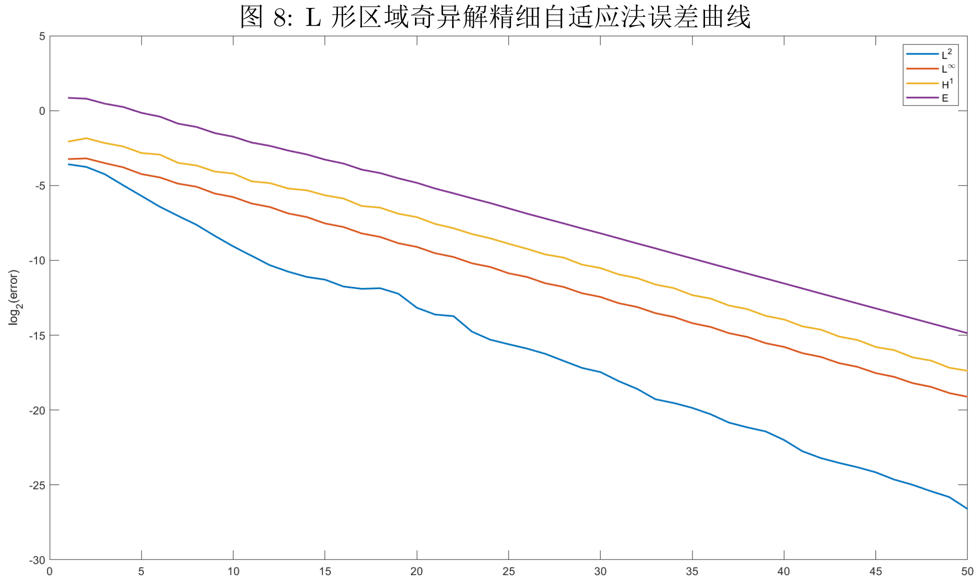
此外，观察最后网格可以发现，主要的加密确实落在了边界处，尤其是最奇异的原点，这也代表了自适应网格在相对奇异情况中的优势。

图 7: L 形区域奇异解自适应法最终网格



将迭代次数增加至 50 次，这时全程所耗时间与一致网格量级一致，最后一次迭代耗时约 1.5 秒，误差曲线如图 8 。

图 8: L 形区域奇异解精细自适应法误差曲线



此处的 L^2 误差最终达到了 1e-6 量级，而 L^∞ 与 H^1 误差也在 1e-5 量级，远好于一致网格的效果。

值得注意的是，这里我们的误差仍然是通过顶点误差计算的，但算法为，设 T 的三个顶点 $T_{1,2,3}$，考虑

$$\|e\|_{L^2}^2 \approx \sum_T \frac{|T|}{3} \sum_{i=1}^{3} e^2(T_i)$$

此算法相当于用三个顶点的值进行数值积分，并且加入了对网格大小的考虑，对 $(u_h)_x$ 与 $(u_h)_y$ 的二范数误差估算完全类似。由于事实上过程中涉及的网格三角形都是等腰直角三角形，形状正则参数一致，这样的估算是准确的。

3.3 未知解析解情况

对于未知解析解的情况，与之前的讨论相似，我们将进行**两次**自适应估算，第一次用于生成最终的解，第二次在对应位置估算误差。

对 2.3 节的两个例子进行 40 次求解并迭代网格，最终网格上求解一次的时间约为 0.5 秒，误差曲线如图 9，最终网格如图 10 。

图 9: L 形区域未知解析解自适应法误差曲线

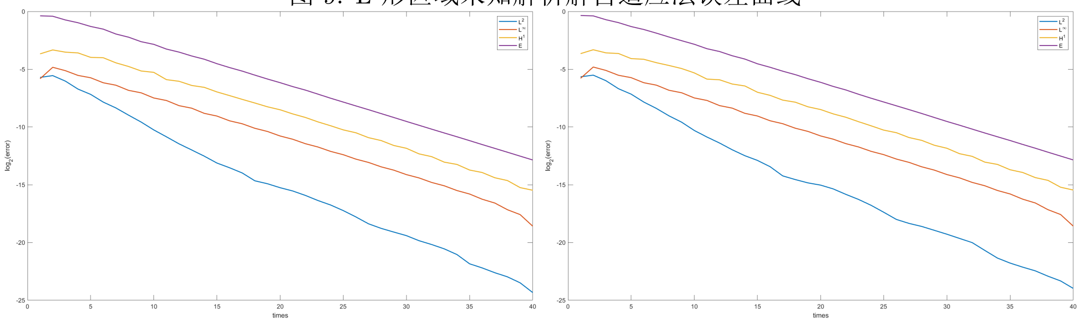
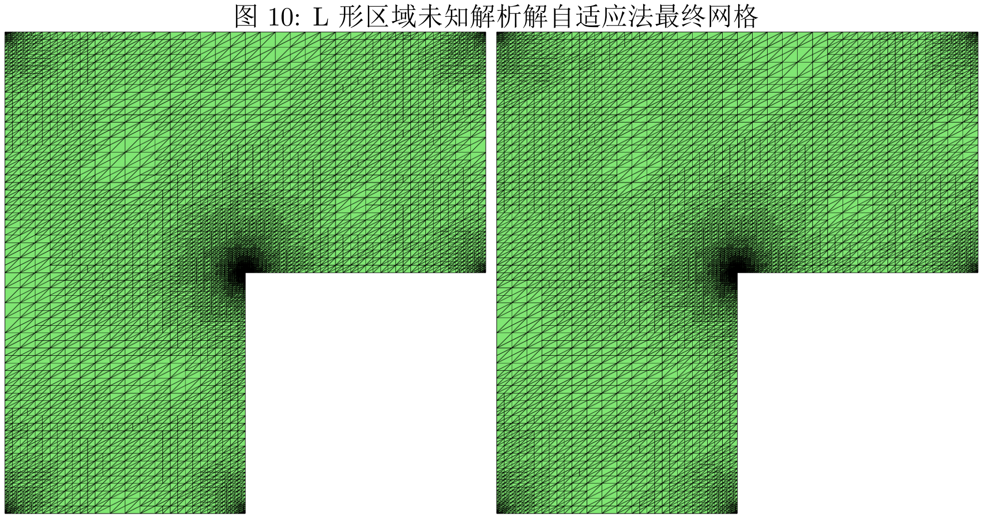


图 10: L 形区域未知解析解自适应法最终网格



可以看出，对于更一般的例子，除了在原点处产生误差，事实上各个角点的误差估计子都偏大。而两个例子中四十次迭代都得到了 1e-7 量级的 L^2 范数误差，效果与耗时同样远好于一致网格时。

由于大部分的计算细节与实现细节都已在代码中标出，这里不再赘述。总体来说，采用自适应方法一般能达成相对较好的收敛性与收敛速度，更适合实际应用，不过自变量即成为迭代次数，与尺度的关系不够明晰，从而理论分析会遇到更多的困难。

有限元方法 大作业 II

郑滕飞 2401110060

* 感谢沈城峰同学提供的在 Colab 上使用 Firedrake 的参考教程，在尝试配了几个环境后发现还是这个最简单清晰。下方的结果均**利用了 Firedrake 包，在 Colab 上在线执行得到**，详见文件 fem.ipynb，其可以直接在 Colab 中运行，文件中 problem1 到 5 对应前两题与第三题的三个小问。

* 一个值得注明的事情是，Firedrake 中的单元 RT_{k+1} 等价于我们上课定义的 RT_k，这可以通过构造单元后计算维数得到。于是，**下方以课堂上的记号为准**，RT_k 实际实现时代码中参数对应 $k+1$。

目录

1 基本情况

1.1 弱形式

如无特殊说明，我们以希腊字母 σ, τ 与 n 表示向量，英文字母 f, g 表示标量。考虑方程组

$$\begin{cases} \sigma - \nabla u = 0 & \forall x \in \Omega \\ -\nabla \cdot \sigma = f & \forall x \in \Omega \\ u = g & \forall x \in \partial\Omega \end{cases}$$

为进行测试，给定真解 $\tilde{u}$，并取 $g = \tilde{u}$、$f = -\triangle \tilde{u}$，则原方程以 $\tilde{u}$ 为唯一解。

考虑有限元空间 $W = (V, Q)$，其中 V 为 RT_k 或 BDM_{k+1}，Q 为 $\mathcal{P}_k^{-1}$ (代码中对应 k 阶 DG 单元)。直接计算可发现，目标函数 $(\sigma, u) \in W$ 满足，对任何函数 $(\tau, v) \in W$，有

$$\int_\Omega (\sigma \cdot \tau + \nabla \cdot \tau u) \mathrm{d}x = \int_\Gamma \tau \cdot n g \mathrm{d}s$$

$$\int_\Omega \nabla \cdot \sigma v \mathrm{d}x = -\int_\Omega f v \mathrm{d}x$$

于是构造

$$L((\sigma, u), (\tau, v)) = \int_\Omega (\sigma \cdot \tau + \nabla \cdot \tau u + \nabla \cdot \sigma v) \mathrm{d}x$$

只需求解方程组

$$L((\sigma, u), (\tau, v)) = -\int_\Omega f v \mathrm{d}x + \int_\Gamma \tau \cdot n g \mathrm{d}s$$

而对应的边界条件为 u 在边界上等于 g。

利用 Firedrake 包的特性，上述描述已经可以通过代码实现，即 (代码中 p, q 对应 σ, τ)

```
a = dot(p, q) * dx + div(p) * v * dx + div(q) * u * dx
L = -f * v * dx + g * dot(q, n) * ds
bcs = DirichletBC(W.sub(1), g, "on_boundary")
w = Function(W)
solve(a == L, w, bcs)
```

此外，包中也提供了直接的误差估算工具，作差后利用 norm 函数即可得到各阶误差的估算，如

```
pHdiv = norm(p_ex - ph, "Hdiv")
pL2 = norm(p_ex - ph)
uL2 = norm(u_ex - uh)
```

这里 ex 代表真解，h 代表数值解，这样即可以直接得到 σ 的 $H(\mathrm{div})$ 误差、L^2 误差与 u 的 L^2 误差。

1.2 实验结果

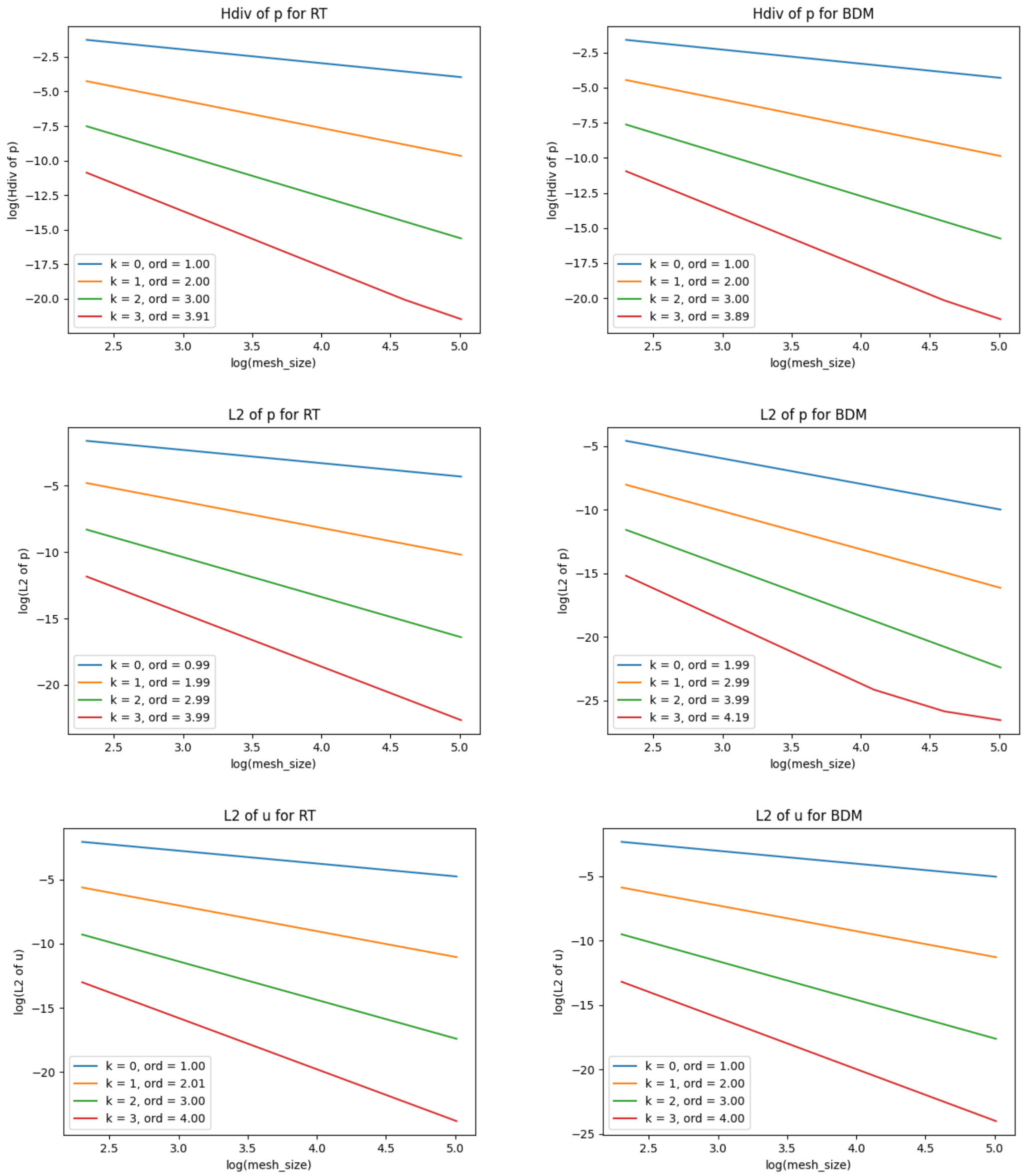
对于方形区域 $[-1, 1]^2$ 的情况，我们考虑函数为

$$u(x, y) = \exp(\sin(x + y) - \sin(y))$$

并对应计算真解 $\sigma = \nabla u$，且取 $g = u$、$f = -\triangle u$。不同单元、不同层次的各范数误差与误差阶如图 1，这里 mesh size 为 $2h^{-1}$，此后也遵循此式。

与课上学到的理论结果对比，可发现两者的 u 的 L^2 范数都符合理论结果，有着 $k+1$ 阶精度；对 RT 单元来说，σ 的 L^2 与 $H(\mathrm{div})$ 范数均为 $k+1$ 阶；与之相对，BDM 单元具有 $k+1$ 阶的 σ 的 $H(\mathrm{div})$ 范数与 $k+2$ 阶的 σ 的 L^2 范数，也符合理论。不过，在 k 与网格都偏大时，两种单元均存在一定数值精度导致的掉阶。

图 1: 方形区域 RT 与 BDM 单元误差阶



2 后处理

* 这部分没有参考除课堂笔记外的资料 (但讲义里有个地方漏了个 v，当时看得疑惑了一会儿)，是自己独立研究出的做法。

2.1 实现方式

记 $\Pi_h^0 u$ 为每个单元上等于 u 积分平均的分片常值函数，并记

$$V^* = \{v \in \mathcal{P}_{k+1}^{-1} \mid \Pi_h^0 v = 0\}$$

假设已有解 u_h、σ_h，则后处理的目标为，找到 $u^* \in \mathcal{P}_{k+1}^{-1}$ 使得

$$\Pi_h^0 u^* = \Pi_h^0 u_h$$

$$\forall v \in V^*, \quad \forall T, \quad \int_T \nabla u^* \cdot \nabla v \mathrm{d}x = \int_T f v \mathrm{d}x + \int_{\partial T} v \sigma_h \cdot n \mathrm{d}s$$

在翻阅 Firedrake 的文档时，名为 R-space 的文档 (www.firedrakeproject.org/r-space.html) 给了我灵感。此文档处理了一个 Neumman 边值问题，为了找到其在 $L^2/\mathbb{R}=L_0^2$ 中的解，它考虑了 $L^2\times\mathbb{R}$ 中的问题，并通过引入附加变量来实现等价。

在我们的例子中，记 $W_N=(\mathcal{P}_{k+1}^{-1},\mathcal{P}_0^{-1})$。则考虑目标函数 $(u^*,r)\in W_N$，其应满足对任何 $(v,s)\in W_N$ 有 (第二个式子能成立是由于，v 为 $k+1$ 次均值为 0 的多项式时其即等价于条件要求，而其他情况通过多加的一个 r 自由度可控制)

$$\int_T su^*\mathrm{d}x=\int_T s u\mathrm{d}x$$

$$\int_T\nabla u^*\cdot\nabla v\mathrm{d}x+\int_T rv\mathrm{d}x=\int_T fv\mathrm{d}x+\int_{\partial T}v\sigma_h\cdot n\mathrm{d}s$$

由此可进一步写为

$$\int_T\nabla u^*\cdot\nabla v\mathrm{d}x+\int_T su^*\mathrm{d}x+\int_T rv\mathrm{d}x=\int_T su\mathrm{d}x+\int_T fv\mathrm{d}x+\int_{\partial T}v\sigma_h\cdot n\mathrm{d}s$$

至此，我一开始写出的代码是 (vN 表示 v，uN 表示 u^*)

```
QN = FunctionSpace(mesh, "DG", k + 1)
R = FunctionSpace(mesh, "DG", 0)
WN = QN * R
uN, r = TrialFunctions(WN)
vN, s = TestFunctions(WN)
aN = dot(grad(uN), grad(vN)) * dx + r * vN * dx + s * uN * dx
LN = f * vN * dx + s * uh * dx + dot(ph, n) * vN * ds
wF = Function(WN)
solve(aN == LN, wF)
uF, _ = split(wF)
```

不过，这个版本始终无法收敛到合理的结果，在研究了许久之后，我终于意识到，问题在于最后一项相加时。前几项对 T 的积分都是化为了整体的积分，但 ∂T 上的积分由不连续性无法相互抵消，必须考虑**非连续的内部积分**，也即对应 Firedrake 中实现 DG 的写法，最终 LN 更改为

```
LN = f * vN * dx + s * uh * dx + \
    2 * avg(dot(ph, n) * vN) * dS + dot(ph, n) * vN * ds
```

这里 avg 表示两侧边界处的值平均，其乘 2 自然对应求和，这就覆盖了所有边界边与内部边。对此式应用求解器后，得到的 uF 即为后处理的结果。

2.2 实验结果

仍然考虑两种单元，作出后处理前与后处理后的 L^2 范数，对比如图 2 。可以看到，结果完全符合理论中的后处理后误差阶从 $k+1$ 提升至 $k+2$，且 RT 单元的稳定性相对较好，数值精度导致的掉阶比 BDM 单元更晚发生。

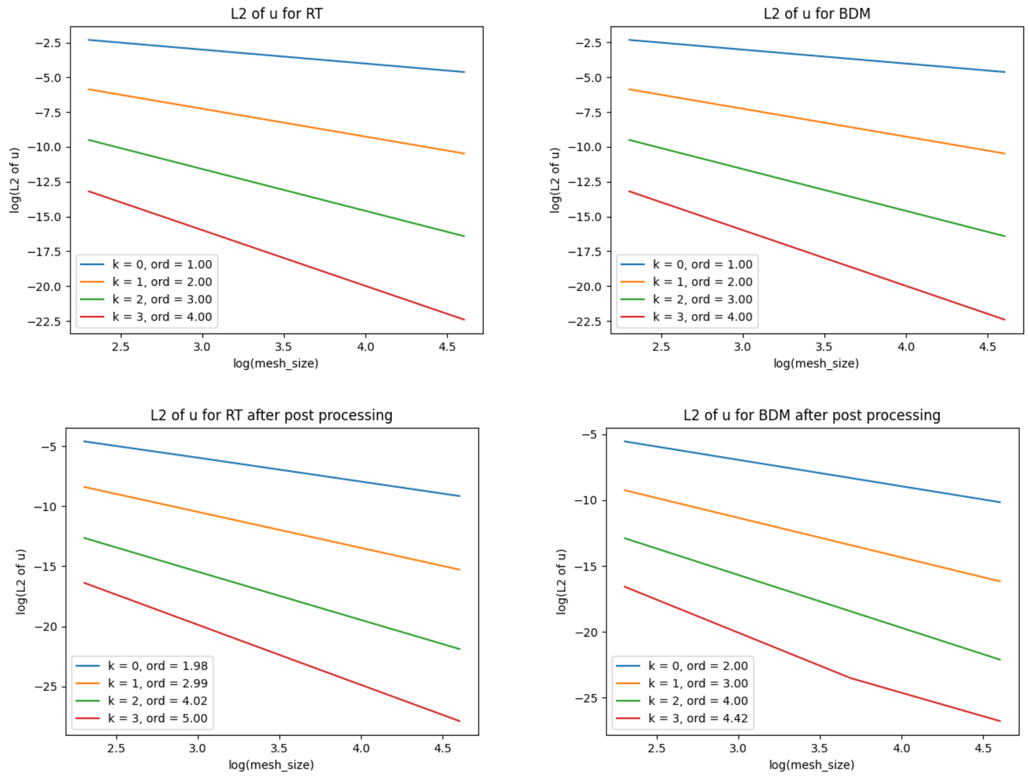
3 非凸区域

3.1 基本结果与后处理

本节中考虑的区域为

$$\Omega=\left\{(x,y)\in(-1,1)^2\mid 0<\theta<\frac{\pi}{\beta}\right\}$$

图 2: 方形区域 RT 与 BDM 单元后处理效果



其中 $\beta \in [0.5, 1)$ 给定。函数对应为

$$u = r^\beta \sin(\beta\theta)$$

利用 atan2 函数，u 可以方便地写为

```
u_ex = (x * x + y * y) ** (beta / 2) * sin(beta * atan2(y, x))
```

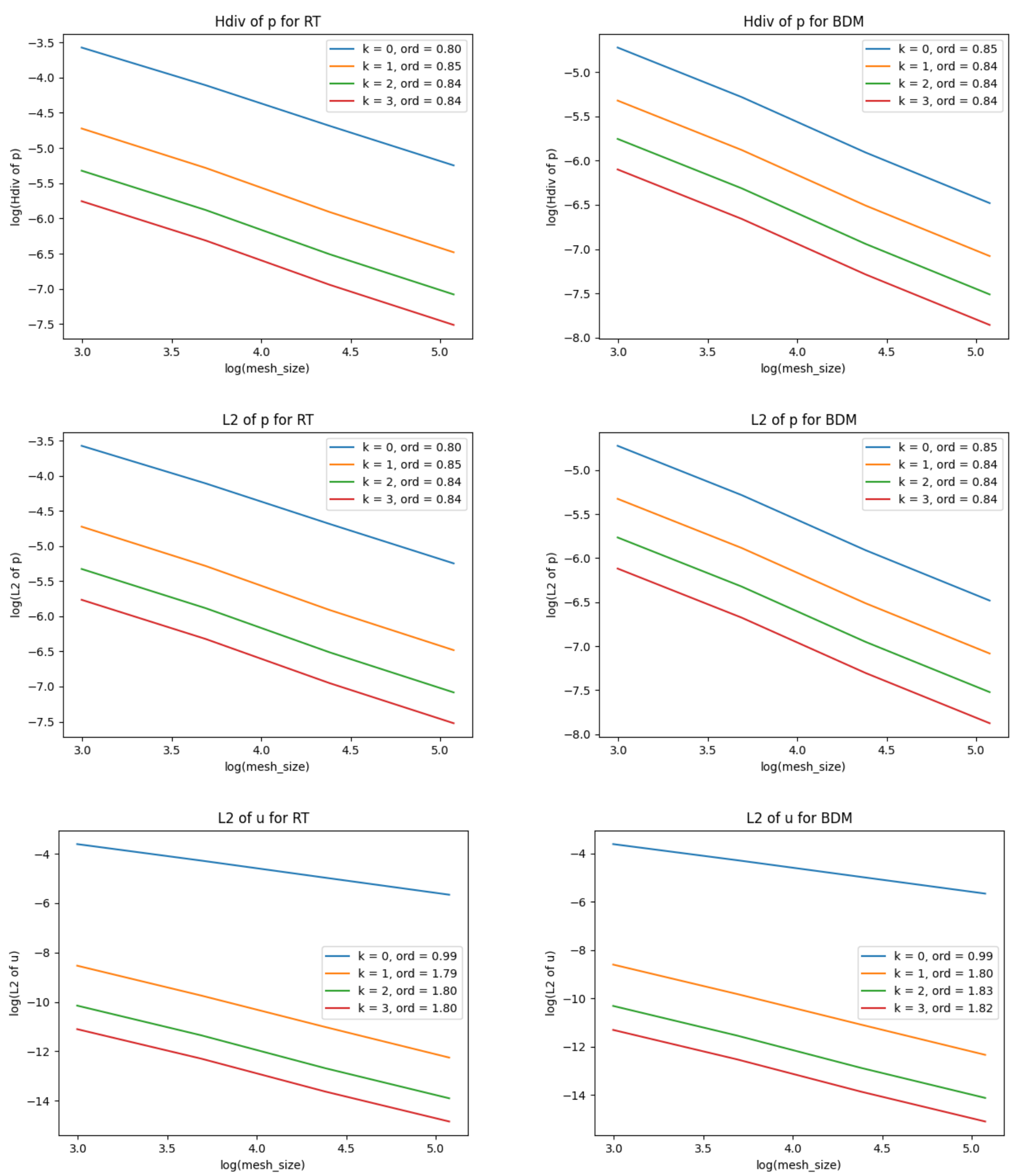
由此仍然可以自动构造出对应的 σ 与 f, g。而此时对区域的描述较复杂，我们选择通过 gmsh 进行构建。具体来说，我们只需要描述其全部边界点，即可通过 gmsh 建立网格，导出到文件后再由 FireDrake 进行导入即可 (事实上更简单的方法是利用 netgen，但我跑相关内容的时候出现了匪夷所思的错误，被迫更换成这个相对麻烦的方式)。由于 β 不同时对应区域形状不同，需要分类讨论进行建立。我们以 $\beta < 4/7$ 时为例，此时网格共有 7 条边，建立为

```
gmsh.model.geo.addPoint(0, 0, 0, h, 1)
gmsh.model.geo.addPoint(1, 0, 0, h, 2)
gmsh.model.geo.addPoint(1, 1, 0, h, 3)
gmsh.model.geo.addPoint(-1, 1, 0, h, 4)
gmsh.model.geo.addPoint(-1, -1, 0, h, 5)
gmsh.model.geo.addPoint(1, -1, 0, h, 6)
gmsh.model.geo.addPoint(1, np.tan((1 / beta - 2) * np.pi), 0, h, 7)
gmsh.model.geo.addLine(1, 2, 1)
gmsh.model.geo.addLine(2, 3, 2)
gmsh.model.geo.addLine(3, 4, 3)
gmsh.model.geo.addLine(4, 5, 4)
gmsh.model.geo.addLine(5, 6, 5)
```

```
gmsh.model.geo.addLine(6, 7, 6)
gmsh.model.geo.addLine(7, 1, 7)
gmsh.model.geo.addCurveLoop([1,2,3,4,5,6,7], 1)
gmsh.model.geo.addPlaneSurface([1], 1)
gmsh.model.geo.synchronize()
gmsh.model.mesh.generate(2)
gmsh.write("t1.msh")
```

有了真解与网格后，求解代码与之前完全一致，得到结果如图 3 (这里取定了 $\beta = 0.85$，实验可发现不同 β 的情况无本质区别)。

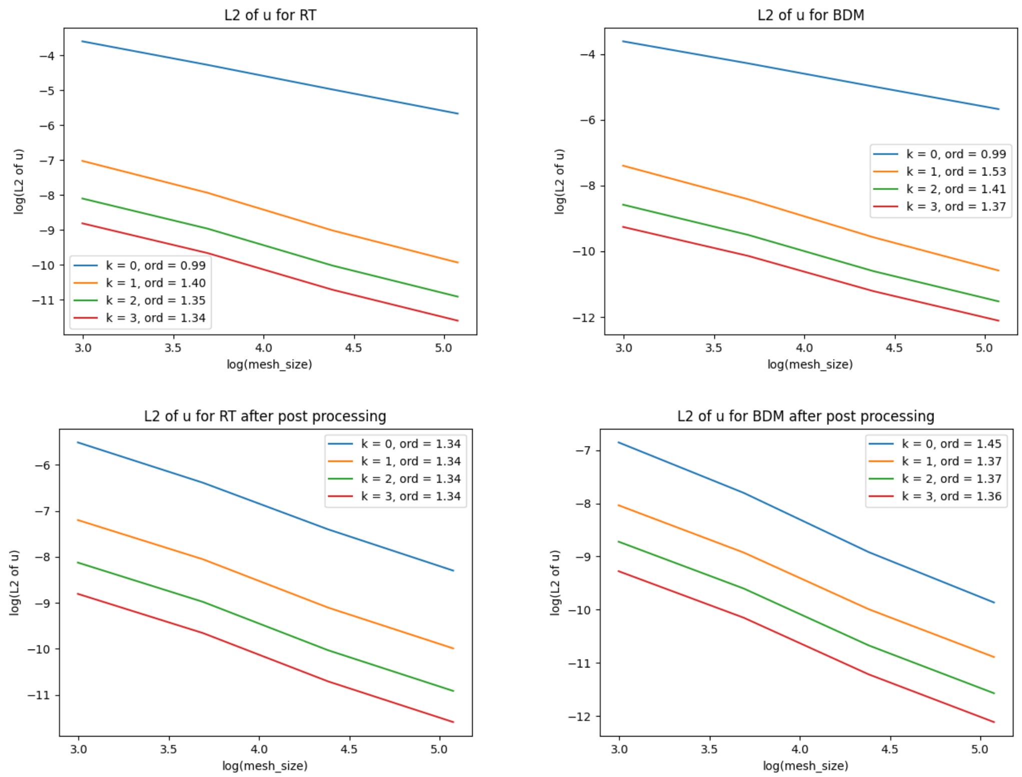
图 3: 奇异情况 RT 与 BDM 单元误差阶



可以发现，即使 0 处存在一定的奇异，结果仍然是可以收敛的，不过收敛阶数要相应降低。对两种单元来说，σ 的 L^2 与 $H(\mathrm{div})$ 范数收敛阶都在一阶附近，而 u 的 L^2 范数当 $k = 0$ 时在一阶附近，更大时在二阶附近。

由于后处理也依赖 σ_h，可以预期，后处理几乎无法使 u_h 更好收敛，具体效果如图 4，这里取 $\theta = 0.67$，可以发现 $k > 0$ 时的收敛阶相较 $\theta = 0.85$ 时有较明显的下降，而后处理也只在 $k = 0$ 时能一定程度提升收敛阶，否则反而会使收敛阶下降。事实上，进一步观察可发现 θ **越小**，收敛阶**越低**。

图 4: 奇异情况 RT 与 BDM 单元后处理效果



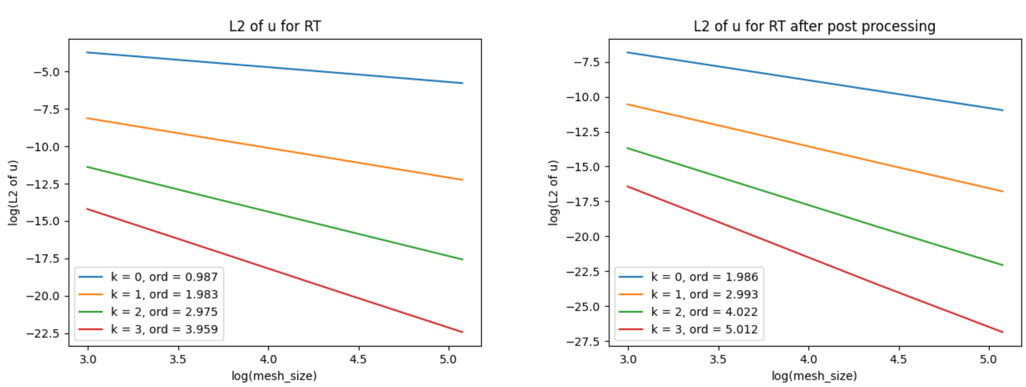
3.2 挖去奇异点

为使对此方程能进行有效的模拟，我们需要考虑挖去奇异点的区域

$$\Omega' = \Omega \backslash [-c_0, c_0]^2$$

事实上，在网格尺度一定的情况下，这几乎相当于强行指定 0 处满足。为了方便后续研究解的收敛阶与 β 的关系，我们固定 $c_0 = 0.2$。

图 5: RT 单元挖去奇异点后效果



这时，网格的构造将更加复杂，不过终归是可以通过连接点得到的多边形，此处不再赘述细节。真解与求解方式、后处理方式均与之前完全相同，因此只需要代入观察新的结果。此时，代入可以发现，结果已经接近正常情况的理论收敛阶了，如图 5 。在后处理前，比起理论阶来说的掉阶相对较小，而后处理之后甚至能超出理论的收敛阶。这足以说明，挖去奇异点后有限元方法可以得到远好于之前的收敛效果。