湘潭大学毕业论文

题　　目：带间断系数的弹性问题的有限元法

学　　院：数学与计算科学学院

班　　级：2019 信息与计算科学 2 班

学　　号：201905755601

姓　　名：唐小康

指导教师：王华

完成日期：2023 年 5 月

摘要

本文以带间断系数的弹性问题为研究对象。首先介绍了研究背景和国内外研究现状，然后回顾了有限元理论的基本知识，包括 Sobolev 空间、弹性问题的边值问题和变分、带间断系数的方程的引入、离散方法和误差估计等。接着给出了几个算例，分别展示了弹性问题和带间断系数的弹性问题在有限元方法下的数值解，比较了不同的参数对结果的影响。最后总结了本文的主要结论，指出了存在的不足和未来的研究方向。

关键字： 平面弹性问题; 间断系数； 非协调有限元; locking-free

Abstract

This paper studies the elastic problem with discontinuous coefficients. Firstly, the research background and the domestic and foreign research status are introduced. Then, the basic knowledge of finite element theory is reviewed, including Sobolev space, boundary value problem and variational form of elastic problem, introduction of equation with discontinuous coefficients, discrete method and error estimation. Next, several examples are given to show the numerical solutions of elastic problem and elasticity problem with discontinuous coefficients under finite element method, and the influence of different parameters on the results is compared. Finally, the main conclusions of this paper are summarized, and the existing shortcomings and future research directions are pointed out.

Key words: linear elasticity problem; discontinuous coefficient; nonconforming finite element; locking-free

目录

1 绪论

1.1 研究背景

带间断系数的方程是指一种用于模拟复合材料层合板的力学性能的数学模型。复合材料层合板是由两层或多层单层板粘合在一起的结构单元，具有不同的铺层顺序和铺层角度，可以表现出不同的力学特性。带间断系数的方程是一种考虑了层间应力和层间变形协调的方程，可以描述层合板在面内和面外的应力-位移关系。带间断系数的方程也可以用于模拟材料的分离和剥落，例如复合材料剥离、金属焊接材料损伤、混凝土材料开裂等。

平面弹性力学方程组是弹性力学中最基础、最常见的模型。当研究的弹性体形状和受力具有一定特点时，通过适当的简化处理，就可以归结为平面弹性问题 [1]。对于各向同性均匀介质的平面弹性问题，当材料的 Lamé 常数 $\lambda \to \infty$ 时，即对于几乎不可压介质，通常低阶的协调有限元解，往往不再收敛到原问题的解，或者达不到最优收敛阶，这就是闭锁现象[2]。

为了消除（近）不可压缩弹性问题中遇到的闭锁现象，国内外研究学者提出了多种有效的数值分析方法。根据型函数建立过程中是否需要网格剖分，这些数值方法可以分为两类：一类是有网格方法，这其中包括高阶有限元法[3]、混合有限元法[4]、增强有限元法[5] 和不连续 Galerkin 法[6] 等；另一类是无网格方法，无网格方法又可分为弱形式无网格法和强形式的无网格方法[1]。

1.2 国内外研究现状

有限元法是一种的数值分析方法，它可以用来求解各种复杂的工程问题。有限元法的核心思想最早可以追溯到 1943 年，当时 R.W.Courant 提出了一种基于变分原理的离散化方法。1956 年，R.W.Clough 等四位教授与工程师在一篇发表在科技期刊上的论文中，首次将这种方法应用到飞机机翼强度的计算中，并将其命名为刚性法 (Stiffness)。这篇论文标志着有限元法在工程学界上的正式诞生。在 1960 年，R.W.Clough 教授在一篇关于平面弹性问题的论文中，首次提出了“有限元法”这个术语，并将这种方法应用到了土木工程领域。三年后，Richard MacNeal 博士与 Robert Schwendler 合作创立了 MSC 公司，并开发出了一款名为 SADSAM 的软件程序，实现了数字仿真模拟结构分析的功能，这标志着有限元方法 (FEA) 从理论走向了实践。1964-1965 年期间，O.C.Zienkiewicz 等人在多篇论文中，采用极小位能原理和虚功原理，以一种新颖的思路推导出了有限元法。

在我国，有限元方法的发展历史上，涌现出了一批杰出的学者，他们为有限元方法的理论和应用做出了重要的贡献，如冯康（有限单元法理论），陈伯屏（结构矩阵方法），钱令希（余能定理），钱伟长（广义变分原理）等。但是，受到当时的国际和国内环境的制约，我国的学者在有限元方法的深层次研究上遇到

了很多困难，很难跟上国际的发展步伐，导致与国外的技术水平之间的差距逐步扩大。20 世纪 60 年代初期，我国的老一辈计算科学家较早地将计算机应用于土木、建筑和机械工程领域。当时黄玉珊教授就提出了“小展弦比机翼薄壁结构的直接设计法”和“力法 - 应力设计法”；而在 70 年代初期，钱令希教授提出了“结构力学中的最优化设计理论与方法的近代发展”。这些理论和方法都为国内的有限元技术指明了方向。1964 年初崔俊芝院士研制出国内第一个平面问题通用有限元程序，解决了刘家峡大坝的复杂应力分析问题。20 世纪 60 年代到 70 年代，国内的有限元方法及有限元软件诞生之后，曾计算过数十个大型工程，应用于水利、电力、机械、航空、建筑等多个领域。20 世纪 70 年代中期，大连理工大学研制出了 JEFIX 有限元软件，航空工业部研制了 HAJIF 系列程序。80 年代中期，北京大学的袁明武教授通过对国外 SAP 软件的移植和重大改造，研制出了 SAP-84；北京农业大学的李明瑞教授研发了 FEM 软件；建筑科学研究院在国家“六五”攻关项目支持下，研制完成了“BDP-建筑工程设计软件包”；中国科学院开发了 FEPS、SEFEM；航空工业总公司飞机结构多约束优化设计系统 YIDOYU 等一批自主程序。

然而，在上世纪 90 年代，国外的有限元软件大规模地进入国内市场，涵盖了各个领域。国外的学者专家也经常到各大学、工厂和企业进行技术推广和使用指导，使得国内有限元方法的发展面临着更大的挑战。管理部门对有限元软件的认识也出现了偏差，对此缺少必要的支持，核心技术控制在国外，所以一直到上世纪末期，国内自主技术创新的速度十分缓慢。但是，在 21 世纪初期以来，国内拥有自主知识产权的软件逐渐实现了市场化，取得了一定的发展空间，同时也引起了国家对有限元技术的高度重视，使得有限元方法逐渐走出低迷状态，不再仅仅停留在高校和企业之中。

2 有限元理论

2.1 Sobolev 空间

假定 G 是有界平面区域, 其边界 Γ 是按段光滑的简单闭曲线, $\overline{G} = G \cup \Gamma$ 是 G 的闭包。对于 $\overline{G}$ 上的任一函数 $u(x,y)$, 称集合 $\{(x,y) \,|\, u(x,y) \neq 0, (x,y) \in \overline{G}\}$ 的闭包为 u 的支集。如果 u 的支集 $\in G$ 内, 则说 u 于 G 具有紧致支集。具有紧致支集的函数必在边界 Γ 的某一邻域内恒等于零[7]。

用 C_0^∞ 表示 G 上无穷次可微并具有紧致支集的函数类, $L^2(G)$ 是定义在 G 上平方可积的可测函数空间, 其内积和范数分别为

$$(f,g) = \int_G fg dxdy, \tag{2.1.1}$$

$$||f|| = \sqrt{(f,f)} = (\int_G |f|^2 dxdy)^{\frac{1}{2}}. \tag{2.1.2}$$

对 $f \in L^2(G)$, 如果存在 $g, h \in L^2(G)$, 使等式

$$\int_G g\varphi dxdy = -\int_g f\frac{\partial \varphi}{\partial x} dxdy, \tag{2.1.3}$$

$$\int_G h\varphi dxdy = -\int_G f\frac{\partial \varphi}{\partial y} dxdy, \tag{2.1.4}$$

对任意的 $\varphi \in C_0^\infty$ 成立, 则说 f 对 x 的一阶广义导数 g 和对 y 的一阶导数 h, 记作

$$f_x = \frac{\partial f}{\partial x} = g, \tag{2.1.5}$$

$$f_y = \frac{\partial f}{\partial y} = y. \tag{2.1.6}$$

定义

$$H^1(G) = \{f(x,y) | f, f_x, f_y \in L^2(G)\},$$

其中 f_x, f_y 是 f 的广义导数。与 $H^1(G)$ 引入内积

$$(f,g)_1 = \int_G [fg + f_x g_x + f_y g_y] dxdy, \tag{2.1.7}$$

和范数

$$||f||_1 = \sqrt{(f,f)} = (\int_G [|f|^2 + |f_x|^2 + |f_y|^2] dxdy)^{\frac{1}{2}}. \tag{2.1.8}$$

则 $H^1(\Omega)$ 是 Hilbert 空间, 称之为 Sobolev 空间。

2.2 弹性问题

2.2.1 边值问题

令 u, g, t, $\sigma=(\sigma_{ij})_{1\le i,j\le 2}$, $\tau=(\tau_{ij})_{1\le i,j\le 2}$ 是双变量函数，定义以下符号

$$\begin{aligned}
\epsilon(u) &= \tfrac{1}{2}(grad\, u + (grad\, u)^t),\\
tr(\tau) &= \tau_{11}+\tau_{22},\\
grad(u) &= \begin{pmatrix} \frac{\partial u_1}{\partial x} & \frac{\partial u_1}{\partial y} \\ \frac{\partial u_2}{\partial x} & \frac{\partial u_2}{\partial y} \end{pmatrix},\\
\delta &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},\\
div\, u &= \frac{\partial u_1}{\partial x}+\frac{\partial u_2}{\partial y},\\
div\, \tau &= \begin{pmatrix} \frac{\partial \tau_{11}}{\partial x}+\frac{\partial \tau_{12}}{\partial y} \\ \frac{\partial \tau_{12}}{\partial x}+\frac{\partial \tau_{22}}{\partial y} \end{pmatrix},\\
\sigma:\tau &= \sum_{i=1}^{2}\sum_{j=1}^{2}\sigma_{ij}\tau_{ij}.
\end{aligned}$$

考虑各项同性弹性材料，令 $u(x,y), f(x,y)$ 是其位移和体力，由线弹性问题的静态理论，u, f 满足以下方程

$$-div\, \sigma(u) = f \quad \in \Omega, \tag{2.2.1}$$

应力张量 $\sigma(u)$ 定义为

$$\sigma(u) = 2\mu\epsilon(u) + \lambda tr(\epsilon(u))\delta. \tag{2.2.2}$$

其中 $\Omega\in\mathbb{R}^2$, 正常数 λ, μ 为 Lamé 常数。假定 $(\mu,\lambda)\in[\mu_1,\mu_2]\times(0,+\infty)$.

令 Γ_1、Γ_2 为 $\partial\Omega$ 的两个开子集，使得 $\partial\Omega=\overline{\Gamma_1}\bigcup\overline{\Gamma_2}$ 并且 $\overline{\Gamma_1}\bigcap\overline{\Gamma_2}=\emptyset$, 令 Γ_1 上的位移边界条件为

$$u|_{\Gamma_1} = g. \tag{2.2.3}$$

并且 Γ_2 上的牵引力边值条件为

$$(\sigma(u)\nu)|_{\Gamma_2} = t. \tag{2.2.4}$$

如果 $\Gamma_1=\emptyset$ (或$\Gamma_2=\emptyset$), 则边值问题为纯牵引力 (或纯位移) 问题。

2.2.2 变分

对于齐次纯位移问题，令 u 在边界上满足

$$u|_{\partial\Omega} = 0. \tag{2.2.5}$$

设 $\nu=(\nu_1,\nu_2)^t$, $\nu_1,\nu_2\in C_0^\infty(\Omega)$, 方程 (2.2.1) 两边同乘 ν 并积分得

$$-\int_{\Omega}(div\,\sigma(u))\,\nu\,dxdy=\int_{\Omega}f\nu\,dxdy. \tag{2.2.6}$$

参考文献知[8]

$$f div\,a=div\,(fa)-a:grad\,f, \tag{2.2.7}$$

$$\int_{\Omega}div\,a\,dV=\int_{\partial\Omega}a\,dS. \tag{2.2.8}$$

将边界条件 (2.2.5)，方程 (2.2.7)，(2.2.8) 带入方程 (2.2.6) 得

$$\begin{aligned}
&-\textstyle\int_{\Omega}(div\,\sigma(u))\,\nu\,dxdy\\
&=-\int_{\Omega}div\,(\sigma(u)\nu)\,dxdy-\int_{\Omega}\sigma(u):grad\,\nu\,dxdy\\
&=-\int_{\Gamma}\sigma(u)\nu\,dxdy+\int_{\Omega}\sigma(u):grad\,\nu\,dxdy\\
&=\int_{\Omega}\sigma(u):grad\,\nu\,dxdy\\
&=\int_{\Omega}2\mu\epsilon(u):grad\,\nu+\lambda\,div\,u\,div\,\nu\,dxdy\\
&=\mu\int_{\Omega}grad\,u:grad\,\nu\,dxdy+(\mu+\lambda)\int_{\Omega}div\,u\,div\,\nu\,dxdy.
\end{aligned}$$

所以

$$\mu\int_{\Omega}grad\,u:grad\,\nu\,dxdy+(\mu+\lambda)\int_{\Omega}div\,u\,div\,\nu\,dxdy=\int_{\Omega}f\nu\,dxdy. \tag{2.2.9}$$

该问题的变分问题为，求 $u\in H^1(\Omega)$ 使得 $u|_{\Gamma_1}=0$, 并且

$$a(u,\nu)=\int_{\Omega}f\cdot\nu\,dxdy\quad\forall\nu\in V, \tag{2.2.10}$$

其中

$$\begin{aligned}
&a(u,\nu):=\mu\int_{\Omega}grad\,u:grad\,\nu\,dxdy+(\mu+\lambda)\int_{\Omega}div\,u\,div\,\nu\,dxdy,\\
&V:=\{\nu\in H^1(\Omega)\quad|\quad\nu|_{\Gamma}=0\}.
\end{aligned} \tag{2.2.11}$$

Lax-Milgram 定理[9]：设 H 是 Hilbert 空间，$a(\cdot,\cdot)$ 是 $H\times H$ 上的有界的强制的双线性泛函。则对任意的 $F\in H$, 存在唯一的 $u\in H$ 满足

$$a(u,\nu)=(f,\nu),\quad\forall\nu\in H. \tag{2.2.12}$$

由 Lax-Milgram 定理知，此变分问题的解存在且唯一。

2.2.3 引入间断系数

设 Ω_1，Ω_2 是 Ω 的两个子集，使得 $\Omega_1 \bigcup \Omega_2 = \Omega$ 并且 $\Omega_1 \bigcap \Omega_2 = \emptyset$,
考虑以下边值问题

$$\begin{aligned} -div\,\sigma(u) &= f \quad \in \Omega, \\ u|_{\partial\Omega} &= 0. \end{aligned} \tag{2.2.13}$$

当 Lamé 常数 λ, μ 在 Ω_1, Ω_2 上取不同值，即 $(x,y) \in \Omega_1$ 时 $\lambda = \lambda_1, \mu = \mu_1$, $(x,y) \in \Omega_2$ 时 $\lambda = \lambda_2, \mu = \mu_2$, 并且 $\lambda_1 \neq \lambda_2, \mu_1 \neq \mu_2$, 通过计算得到与此问题对应的双线性形式为

$$\begin{aligned} a(u,\nu) = \mu_1 \int_{\Omega_1} grad\,u : grad\,\nu\, dxdy + (\mu_1 + \lambda_1) \int_{\Omega_1} div\,u\, div\,\nu\, dxdy \\ + \mu_2 \int_{\Omega_2} grad\,u : grad\,\nu\, dxdy + (\mu_2 + \lambda_2) \int_{\Omega_2} div\,u\, div\,\nu\, dxdy. \end{aligned} \tag{2.2.14}$$

2.3 离散

2.3.1 Galerkin 法

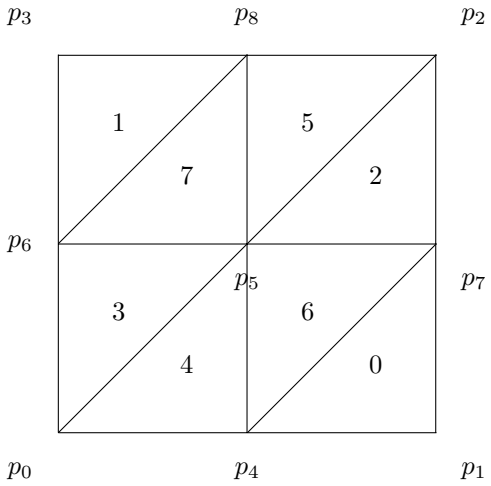


图 1

设求解区间 $\Omega = [0,1] \times [0,1]$, 首先对其按照图 1 进行网格剖分，节点为

$$p_0, p_1, ..., p_n.$$

图中的三角形区域称为单元。

其次，在 Sobolev 空间 H^1 内取子空间 U_h, 它的元素在每一单元是次数不超过某一正整数 m 的多项式，在全区域 Ω 上属于函数空间 H^1. 则 $U_h \times U_h$ 为试探函数空间。

设

$$U_h = span(\varphi_0, \varphi_1, ..., \varphi_n),$$
$$\phi_{2i} = (\varphi_i, 0), \quad \phi_{2i+1} = (0, \varphi_i), \quad i = 0, ..., n.$$

则 $\forall u_h \in U_h \times U_h$, 可表成

$$u_h = \sum_{i=0}^{2n+1} c_i \phi_i. \tag{2.3.1}$$

将式 (2.3.1) 带入方程 (2.2.10) 中得到 Galerkin 方程

$$\sum_{i=0}^{2n+1} a(\phi_i, \phi_j) c_i = (f, \phi_j), \quad j = 0, 1, ..., 2n+1. \tag{2.3.2}$$

令

$$A = (a(\phi_j, \phi_i))_{0 \le i,j \le 2n+1},$$
$$F = ((f, \phi_i))_{0 \le i \le 2n+1},$$
$$c = (c_i)_{0 \le i \le 2n+1}.$$

则 Galerkin 方程 (2.3.2) 的矩阵形式为

$$Ac = F. \tag{2.3.3}$$

考虑齐次边界条件, 若 (x_i, y_i) 为边界点, 则 A 第 $2i$ 行第 $2i$ 列, 第 $2i+1$ 行第 $2i+1$ 列元素为 1, 第 $2i$ 和 $2i+1$ 行的其他元素及 $F(2i), F(2i+1)$ 都为 0.

2.3.2 线性元

如图 2, 设 $\triangle(p_0, p_1, p_2)$ 是以 p_0, p_1, p_2 为顶点的任意三角型元, 面积为 S. 在 $\triangle(p_0, p_1, p_2)$ 内任取一点 p, 坐标为 (x, y). 过 p 点作与三个顶点的连线, 将 $\triangle(p_0, p_1, p_2)$ 分成三个三角形: $\triangle(p_1, p_2, p)$, $\triangle(p_0, p, p_2)$, $\triangle(p_0, p_1, p)$, 其面积分别为 S_0, S_1, S_2. [7]

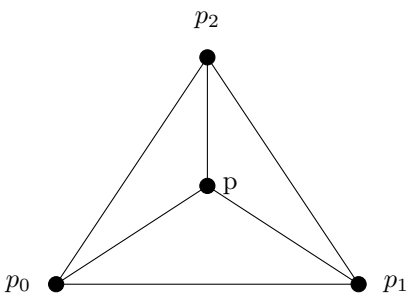


图 2

显然 $S_0+S_1+S_2=S$, 令

$$L_0=\frac{S_0}{S},\quad L_1=\frac{S_1}{S},\quad L_2=\frac{S_2}{S}, \tag{2.3.4}$$

$$\begin{cases} L_0=\frac{1}{2S}[(x_2y_3-x_3y_2)+(y_2-y_3)x+(x_3-x_2)y], \\ L_1=\frac{1}{2S}[(x_3y_0-x_0y_3)+(y_3-y_0)x+(x_0-x_3)y], \\ L_2=\frac{1}{2S}[(x_0y_1-x_1y_0)+(y_0-y_1)x+(x_1-x_0)y]. \end{cases}$$

因为

$$\begin{cases} L_0=\begin{cases} 1, & x=x_0,y=y_0, \\ 0, & x=x_1,y=y_1, \\ 0, & x=x_2,y=y_2, \end{cases} \\ L_1=\begin{cases} 0, & x=x_0,y=y_0, \\ 1, & x=x_1,y=y_1, \\ 0, & x=x_2,y=y_2, \end{cases} \\ L_2=\begin{cases} 0, & x=x_0,y=y_0, \\ 0, & x=x_1,y=y_1, \\ 1, & x=x_2,y=y_2, \end{cases} \end{cases}$$

所以在此区间上 $\varphi_i=L_i$，即

$$\begin{cases} \varphi_0=\frac{1}{2S}[(x_2y_3-x_3y_2)+(y_2-y_3)x+(x_3-x_2)y], \\ \varphi_1=\frac{1}{2S}[(x_3y_0-x_0y_3)+(y_3-y_0)x+(x_0-x_3)y], \\ \varphi_2=\frac{1}{2S}[(x_0y_1-x_1y_0)+(y_0-y_1)x+(x_1-x_0)y]. \end{cases} \tag{2.3.5}$$

2.3.3 C-R 元

如图 3,设三角形 $\triangle(q_0,q_1,q_2)$ 是以 q_0,q_1,q_2 为顶点的任意三角形元，p_0,p_1,p_2 为其三条边的中点，其坐标分别为 $(x_0.y_0),(x_1,y_1),(x_2,y_2)$.

设三角形 $\triangle(q_0,q_1,q_2)$ 上的 C-R 元为 $\varphi_0,\varphi_1,\varphi_2$,

$$\varphi_i=a_ix+b_iy+c_i,\quad i=0,1,2, \tag{2.3.6}$$

且其在 p_0，p_1，p_2 点上满足以下关系式

$$\varphi_i(p_j)=\begin{cases} 1, & i=j \\ 0, & i\neq j \end{cases},\quad i,j=0,1,2. \tag{2.3.7}$$

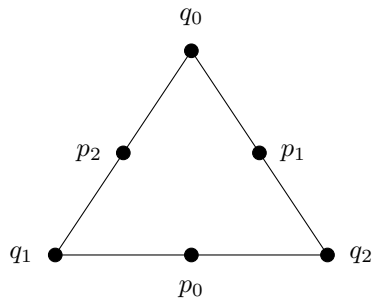


图 3

设

$$A = \begin{bmatrix} x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix}, \quad c_i = (a_i, b_i, c_i)^t, \quad f = (x, y, 1)^t.$$

则方程组 (2.3.7) 的矩阵形式为

$$Ac_i = e_i, \quad i = 0, 1, 2. \tag{2.3.8}$$

通过计算可以得到单元 $\triangle(q_0, q_1, q_2)$ 上的 C-R 元为

$$\varphi_i = A^{-1} e_i f, \quad i = 0, 1, 2. \tag{2.3.9}$$

2.4 误差估计

假设 Ω 是一个凸多边形区域，并且 Γ_1 or Γ_2 中任意一个为空。对于纯位移问题 ($\Gamma_2 = \emptyset$), 只考虑齐次边界条件。

令 T^h 是 Ω 三角划分的一个非退化族。对于纯位移问题 ($\Gamma_2 = \emptyset$), 使用有限元空间

$$V_h := \{\nu \in H^1(\Omega) : \nu|_T, \forall T \in T^h\},$$

并且对于纯牵引力问题 ($\Gamma_1 = \emptyset$), 使用

$$V_h := \{\nu \in H^1(\Omega) : \nu|_T, \forall T \in T^h\},$$

令 $u \in H^2(\Omega) \cap H^1(\Omega)$ 满足纯位移问题，并且 $u_h \in V_h$ 满足

$$a(u_h, \nu) = \int_\Omega f \cdot \nu \, dx \quad \forall \nu \in V_h. \tag{2.4.1}$$

则存在一个正常数 $C_{(\mu,\lambda)}$ 使得[9]

$$\|u - u_h\|_{H^1(\Omega)} \leq C_{(\mu,\lambda)} h \|u\|_{H^2(\Omega)}. \tag{2.4.2}$$

令 $u \in H^2(\Omega)$ 满足纯牵引力问题。令 $u_h \in V_h$ 满足

$$a(u_h, \nu) = \int_\Omega f \cdot \nu \, dx + \int_{\Gamma_2} t \cdot \nu \, ds \quad \forall \nu \in V_h. \tag{2.4.3}$$

则存在一个正常数 $C_{(\mu,\lambda)}$ 使得[9]

$$\|u - u_h\|_{H^1(\Omega)} \leq C_{(\mu,\lambda)} h \|u\|_{H^2(\Omega)}. \tag{2.4.4}$$

2.5 闭锁现象

对于固定的 μ 和 λ，以上定理给出了弹性问题令人满意近似的有限元近似。但是这些有限元方法的性能随着 λ 趋向于 ∞ 而变差。这就是所谓的锁定现象[9]。

令 $\Omega = (0,1) \times (0,1)$. 考虑 $\mu = 1$ 时的纯位移边值问题:

$$\begin{aligned} div\{2\epsilon(u^\lambda) + \lambda tr(\epsilon(u^\lambda))\delta\} = f \quad in \quad \Omega \\ u^\lambda|_{\partial\Omega} = 0. \end{aligned} \tag{2.5.1}$$

注意给定的 f, 当 $\lambda \to \infty$, $\|div u^\lambda\|_{H^1(\Omega)} \to 0$. 换句话说，我们正在处理一种几乎不可能压缩的弹性材料。为了强调对 λ 的依赖，将应力张量 $\sigma_\lambda(\nu)$ 和变分形式 $a_\lambda(\nu, \omega)$ 表示为

$$\begin{aligned} \sigma_\lambda(\nu) &= 2\epsilon(\nu) + \lambda tr(\epsilon(\nu))\delta, \\ a_\lambda(\nu, \omega) &= \int_\Omega \{2\epsilon(\nu) : \epsilon(\omega) + \lambda \, div \, \nu \, div \, \omega\} \, dx. \end{aligned} \tag{2.5.2}$$

令 T^h 为 Ω (图 4) 的一个规则三角剖分。对于每一个 $u \in H^2(\Omega) \cap H_0^1(\Omega)$, 定义 $u_h^\lambda \in V_h$ 为以下方程组的特解

$$\begin{aligned} a_\lambda(u_h^\lambda, \nu) = \int_\Omega [-div \, \sigma_\lambda(u)] \cdot \nu \, dx \quad \forall \nu \in V_h, \\ V_h := \{\nu \in H^1(\Omega) : \nu|_T, \forall T \in T^h\}. \end{aligned} \tag{2.5.3}$$

定义 $L_{\lambda,h}$ 为

$$L_{\lambda,h} := sup\{\frac{|u - u_h^\lambda|_{H^1(\Omega)}}{\|div \sigma_\lambda(u)\|_{L^2(\Omega)}} : 0 \neq u \in H^2(\Omega) \cap H^1(\Omega)\}. \tag{2.5.4}$$

则存在一个与 h 无关的正常数 C 使得[9]

$$\lim_{\lambda \to \infty} \inf L_{\lambda,h} \geq C. \tag{2.5.5}$$

式 (2.5.5) 意味着：无论 h 取多小，只要 λ 足够大，我们都能找到 $u \in H^2(\Omega) \cap H^1(\Omega)$ 使得相对误差 $|u - u_h|_{H^1(\Omega)} / \|div \sigma_\lambda(u)\|_{L^2(\Omega)}$ 以一个与 h 无关的常数为下界。换句话说，有限元方法的性能将会随着 λ 变大而变坏。

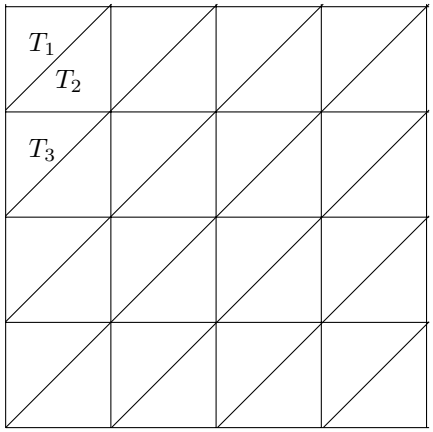


图 4

令 $\Omega = (0,1) \times (0,1)$. 考虑以下纯位移问题

$$\begin{aligned} -\mu \triangle u - (\mu + \lambda)\, grad\,(div\, u) = f \quad & \in \Omega, \\ u = 0 \quad & on\ \partial\Omega, \end{aligned} \tag{2.5.6}$$

其中，$f \in L^2(\Omega)$, Ω_1, Ω_2 是 Ω 的两个子集，使得 $\Omega_1 \bigcup \Omega_2 = \Omega$ 并且 $\Omega_1 \bigcap \Omega_2 = \emptyset$, $(x,y) \in \Omega_1$ 时 $\lambda = \lambda_1$, $\mu = \mu_1$, $(x,y) \in \Omega_2$ 时 $\lambda = \lambda_2$, $\mu = \mu_2$, $\lambda_1 \neq \lambda_2$, $\mu_1 \neq \mu_2$.

它的变分形式为，求 $u \in H^1(\Omega)$ 使得 $u|_\Gamma = 0$, 并且

$$a^s(u,v) = \int_\Omega f \cdot v\, dxdy \quad \forall v \in H^1(\Omega), \tag{2.5.7}$$

其中

$$a^s(u,v) = \mu \int_\Omega grad\, u : grad\, v\, dxdy + (\mu + \lambda) \int_\Omega (div\, u)\,(div\, v)\, dxdy. \tag{2.5.8}$$

令 T^h 是 Ω 三角划分的一个非退化族。定义

$$\begin{aligned} V_h^\star = \{ & v : v \in L^2(\Omega),\ v|_T \text{是线性的} \forall T \in T^h, \\ & v \text{在单元边界的中点上是连续的并且} v = 0\}. \end{aligned} \tag{2.5.9}$$

对 $v \in V_h^\star$, 定义

$$(grad_h\, v)|_T = grad\,(v|_T), \quad (div_h\, v)|_T = div\,(v|_T), \quad \forall T \in T^h. \tag{2.5.10}$$

则问题的离散形式为，求 $u_h \in V_h^\star$ 使得

$$a_h^\star(u_h, v) = \int_\Omega f \cdot v\, dxdy, \quad \forall v \in V_h^\star, \tag{2.5.11}$$

其中双线性形式 $a_h^\star(\cdot,\cdot)$ 在 $V_h^\star + H^1(\Omega)$ 上的定义为

$$\begin{aligned} a_h^\star(u,v) = \mu \int_\Omega grad_h\, u : grad_h\, v\, dxdy \\ + (\mu+\lambda) \int_\Omega (div_h\, u)\,(div_h\, v)\, dxdy. \end{aligned} \tag{2.5.12}$$

定义 $V_h^\star + H^1(\Omega)$ 上的非协调能量泛函

$$||v||_h = a_h^\star(v,v)^{1/2}. \tag{2.5.13}$$

显然

$$||grad_h\, v||_{L^2(\Omega)} \le \mu^{-1/2}||v||_h. \tag{2.5.14}$$

定义

$$(\Pi_h u)(m_e) = \frac{1}{|e|}\int_e u\, ds, \tag{2.5.15}$$

其中 m_e 为边缘 e 的中点。则有

$$div(\Pi_h u)|_T = \frac{1}{|T|}\int_T div\, u\, dxdy \quad \forall T \in T^h, \tag{2.5.16}$$

并且存在独立于 h 的正常数 C 使得

$$||u - \Pi_h u||_{L^2(\Omega)} + h||grad_h\,(u - \Pi_h u)||_{L^2(\Omega)} \le Ch^2|u|_H^2(\Omega). \tag{2.5.17}$$

参考文献得

$$||u||_{H^2(\Omega)} + \lambda||div\, u||_{H^1(\Omega)} \le C||f||_{L^2(\Omega)}, \tag{2.5.18}$$

$$||u - u_h||_h \le \inf_{v\in V_h}||u-v||_h + \sup_{v\in V_h\setminus\{0\}} \frac{|a_h^s(u,v) - \int_\Omega f\cdot v\, dxdy|}{||v||_h}, \tag{2.5.19}$$

$$\begin{aligned} |\int_\Omega grad_h\, u : grad_h\, v\, dxdy + \int_\Omega \triangle u \cdot v\, dxdy| \\ \le C\, h\, |u|_{H^2(\Omega)}||grad_h\, v||_{L^2(\Omega)}, \end{aligned} \tag{2.5.20}$$

$$\begin{aligned} |\int_\Omega div_h\, u\, div_h\, v\, dxdy + \int_\Omega grad\,(div\, u)\cdot v\, dxdy| \\ \le C\, h\, |div\, u|_{H^1(\Omega)}||grad_h\, v||_{L^2(\Omega)}. \end{aligned} \tag{2.5.21}$$

设

$$\begin{aligned} a^s(u,v)|_{\Omega_1} = \mu \int_{\Omega_1} grad\, u : grad\, v\, dxdy \\ + (\mu+\lambda)\int_{\Omega_1}(div\, u)\,(div\, v)\, dxdy. \end{aligned} \tag{2.5.22}$$

由公式 (2.5.6), (2.5.12), (2.5.20), (2.5.21), (2.5.18) 和 (2.5.14) 得

$$
\begin{aligned}
&|a_h^s(u,v) - \int_\Omega f\cdot v\,dxdy| \\
&= |(a_h^s(u,v)|_{\Omega_1} - \int_\Omega f\cdot v\,dxdy) \\
&+ (a_h^s(u,v)|_{\Omega_2} - \int_\Omega f\cdot v\,dxdy)| \\
&\le C\,h\,||grad_h\,v||_{L^2(\Omega_1)}(\mu_1|u|_{H^2(\Omega_1)} + (\mu_1+\lambda_1)|div\,u|_{H^1(\Omega_1)}) \\
&+ C\,h\,||grad_h\,v||_{L^2(\Omega_2)}(\mu_2|u|_{H^2(\Omega_2)} + (\mu_2+\lambda_2)|div\,u|_{H^1(\Omega_2)}) \\
&\le C\,h\,||v||_h||f||_{L^2(\Omega)}.
\end{aligned}
\tag{2.5.23}
$$

参考文献得到，存在 $u_1 \in H^2(\Omega)\cup H^1(\Omega)$, 使得

$$
div\,u_1 = div\,u, \tag{2.5.24}
$$

$$
||u_1||_{H^2(\Omega)} \le C\,||div\,u||_{H^1(\Omega)}, \tag{2.5.25}
$$

$$
||u_1||_{H^2(\Omega)} \le \frac{C}{1+\lambda}\,||f||_{L^2(\Omega)}, \tag{2.5.26}
$$

$$
div_h\,\Pi_h\,u_1 = div_h\,\Pi_h\,u. \tag{2.5.27}
$$

由公式 (2.5.18), (2.5.17), (2.5.24), (2.5.26) 和 (2.5.27) 得

$$
\begin{aligned}
&\inf_{v\in V_h}||u-v||_h \\
&\le ||u-\Pi_h u||_h \\
&= (\mu_1||grad_h\,(u-\Pi_h u)||^2_{L^2(\Omega_1)} + (\mu_1+\lambda_1)||div_h\,(u-\Pi_h u)||^2_{L^2(\Omega_1)})^{1/2} \\
&+ (\mu_2||grad_h\,(u-\Pi_h u)||^2_{L^2(\Omega_2)} + (\mu_2+\lambda_2)||div_h\,(u-\Pi_h u)||^2_{L^2(\Omega_2)})^{1/2} \\
&= (\mu_1||grad_h\,(u-\Pi_h u)||^2_{L^2(\Omega_1)} + (\mu_1+\lambda_1)||div_h\,(u_1-\Pi_h u_1)||^2_{L^2(\Omega_1)})^{1/2} \\
&+ (\mu_2||grad_h\,(u-\Pi_h u)||^2_{L^2(\Omega_2)} + (\mu_2+\lambda_2)||div_h\,(u_1-\Pi_h u_1)||^2_{L^2(\Omega_2)})^{1/2} \\
&\le C\,h\,||f||_{L^2(\Omega)}.
\end{aligned}
\tag{2.5.28}
$$

由公式 (2.5.19), (2.5.23), (2.5.28) 得

$$
||u-u_h||_h \le C\,h\,||f||_{L^2(\Omega)}. \tag{2.5.29}
$$

3 算例

3.1 弹性问题

3.1.1 算例一

考察以下边值问题

$$-div\,\sigma(u) = f \quad \in \Omega,$$
$$u|_{\partial\Omega} = 0.$$

其中 $u = (u_1, u_2)^t$ 为求解向量，$f = (f_1, f_2)^t$ 为右端向量，$\Omega = [0,1] \times [0,1]$,

$$u_1 = (x-1)(y-1)ysin(x),$$
$$u_2 = (x-1)(y-1)xsin(y).$$

通过数值实验得到，

1. 当 Lam$\acute{e}$ 常数 $\mu = 1$, $\lambda = 1$ 时的误差及误差阶如下表

表 1

h	1.0	0.5	0.25	0.125
$\|u-u_h\|_{H^1(\Omega)}$	4.102804E-1	1.424371E-1	8.382384E-2	4.737414-2
H1 误差阶	1.526284	0.764893	0.823260	0.905945
$\|u-u_h\|_{L^2(\Omega)}$	4.102632E-1	7.121859E-2	2.095596E-2	5.921768E-3
L2 误差阶	2.526284	1.764893	1.823260	1.905945

2. 当 Lam$\acute{e}$ 常数 $\mu = 1$, $\lambda = 1E4$ 时的误差及误差阶如下表

表 2

h	1.0	0.5	0.25	0.125
$\|u-u_h\|_{H^1(\Omega)}$	3.954836E-1	1.369558E-1	8.048234E-2	4.543017E-2
H1 误差阶	1.529907	0.7669663	0.8250215	0.9072268
$\|u-u_h\|_{L^2(\Omega)}$	3.945746E-1	6.847791E-2	2.012058E-2	5.678771E-3
L2 误差阶	2.529607	1.766966	1.825021	1.907226

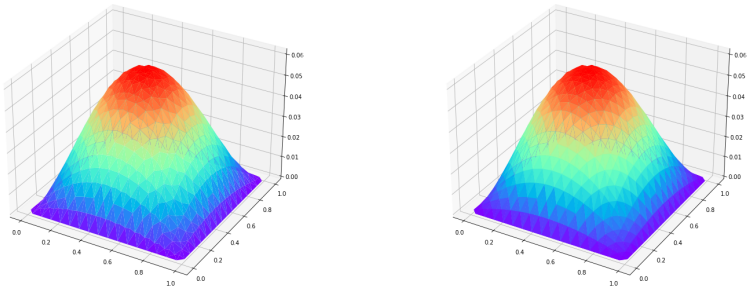
3. 当 Lam$\acute{e}$ 常数 $\mu = 1$, $\lambda = 1E8$ 时的误差及误差阶如下表

表 3

h / λ	1.0	0.5	0.25	0.125
$\|u-u_h\|_{H^1(\Omega)}$	4.102771E-1	1.424359E-1	8.382310E-2	4.737371E-2
H1 误差阶	1.526285	0.7648937	0.823261	0.9059459
$\|u-u_h\|_{L^2(\Omega)}$	4.104650E-1	7.121799E-2	2.095577E-2	5.921714E-3
L2 误差阶	2.526285	1.764893	1.823261	1.905945

数值解和精确解图像如下

图 5



(a) 数值解图像　　(b) 精确解图像

3.1.2 算例二

考察以下边值问题

$$-div\,\sigma(u) = f \quad \in \Omega,$$
$$u|_{\partial\Omega} = 0.$$

其中 $u = (u_1, u_2)^t$ 为求解向量，$f = (f_1, f_2)^t$ 为右端向量，$\Omega = [0,1] \times [0,1]$,

$$u_1 = x^2 sin(x-1) y^2 sin(y-1),$$
$$u_2 = x^2 sin(x-1) y^2 sin(y-1).$$

通过数值实验得到,

1. 当 Lam$\acute{e}$ 常数 $\mu = 1$, $\lambda = 1$ 时的误差及误差阶如下表

表 4

h	1.0	0.5	0.25	0.125
$\|u-u_h\|_{H^1(\Omega)}$	1.381670E-1	1.038442E-1	4.190134E-2	2.135923E-2
H1 误差阶	0.4119931	1.309352	0.972136	0.9399994
$\|u-u_h\|_{L^2(\Omega)}$	1.435610E-1	5.192211E-2	1.047533E-2	2.669904E-3
L2 误差阶	1.411993	2.309352	1.972136	1.939999

2. 当 Lam$\acute{e}$ 常数 $\mu = 1$, $\lambda = 1E4$ 时的误差及误差阶如下表

表 5

h	1.0	0.5	0.25	0.125
$\|u-u_h\|_{H^1(\Omega)}$	1.328102E-1	1.001068E-1	4.029172E-2	2.049834E-2
H1 误差阶	0.4078262	1.312984	0.9749761	0.9414672
$\|u-u_h\|_{L^2(\Omega)}$	1.647602E-1	5.005340E-2	1.007293E-2	2.562293E-3
L2 误差阶	1.407826	2.312984	1.974976	1.941467

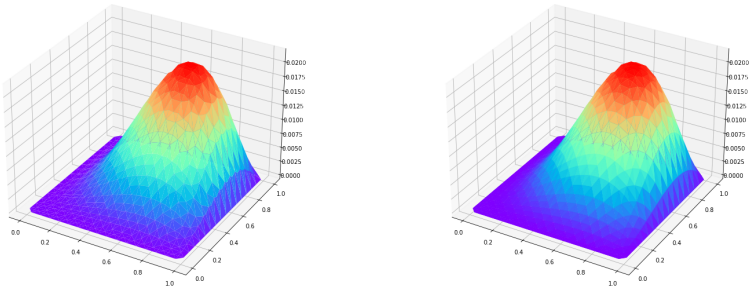
3. 当 Lam$\acute{e}$ 常数 $\mu = 1$, $\lambda = 1E8$ 时的误差及误差阶如下表

表 6

h / λ	1.0	0.5	0.25	0.125
$\|u-u_h\|_{H^1(\Omega)}$	1.381659E-1	1.038433E-1	4.190099E-2	2.135904E-2
H1 误差阶	0.4119922	1.309353	0.9721372	0.9399997
$\|u-u_h\|_{L^2(\Omega)}$	1.473169E-1	5.192169E-2	1.047524E-2	2.669880E-3
L2 误差阶	1.411992	2.309353	1.972137	1.939999

数值解和精确解图像如下

图 6



(a) 数值解图像　　(b) 精确解图像

3.2 带间断系数的弹性问题

3.2.1 算例一

考察以下边值问题

$$\begin{aligned} -div\,\sigma(u) &= f \quad \in \Omega, \\ u|_{\Gamma} &= 0. \end{aligned}$$

其中 $u = (u_1, u_2)^t$ 为求解向量，$f = (f_1, f_2)^t$ 为右端向量，$\Omega = [0,1] \times [0,1]$，$\Omega_1 = [0, 0.5] \times [0, 0.5]$，$\Omega_2 = [0.5, 1] \times [0, 0.5]$，$\Omega_3 = [0, 0.5] \times [0.5, 1]$，$\Omega_4 = [0.5, 1] \times [0.5, 1]$.

当 $(x, y) \in \Omega_1$ 时，$\mu = \lambda = \lambda_1$，

$$\begin{aligned} u_1 &= x(x-0.5)(x-1)y(y-0.5)(y-1) \,/\, \lambda_1, \\ u_2 &= x(x-0.5)(x-1)y(y-0.5)(y-1) \,/\, \lambda_1. \end{aligned}$$

当 $(x, y) \in \Omega_2$ 时，$\mu = \lambda = \lambda_2$，

$$\begin{aligned} u_1 &= x(x-0.5)(x-1)y(y-0.5)(y-1) \,/\, \lambda_2, \\ u_2 &= x(x-0.5)(x-1)y(y-0.5)(y-1) \,/\, \lambda_2. \end{aligned}$$

当 $(x, y) \in \Omega_3$ 时，$\mu = \lambda = \lambda_3$，

$$\begin{aligned} u_1 &= x(x-0.5)(x-1)y(y-0.5)(y-1) \,/\, \lambda_3, \\ u_2 &= x(x-0.5)(x-1)y(y-0.5)(y-1) \,/\, \lambda_3. \end{aligned}$$

当 $(x, y) \in \Omega_4$ 时，$\mu = \lambda = \lambda_4$，

$$\begin{aligned} u_1 &= x(x-0.5)(x-1)y(y-0.5)(y-1) \,/\, \lambda_4, \\ u_2 &= x(x-0.5)(x-1)y(y-0.5)(y-1) \,/\, \lambda_4. \end{aligned}$$

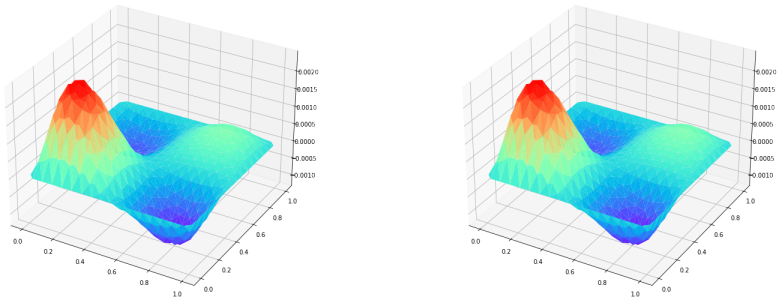
令 $\lambda = [\lambda_1, \lambda_2, \lambda_3, \lambda_4]$，$\mu = [\mu_1, \mu_2, \mu_3, \mu_4]$.

1. 当 Lamé 常数 $\lambda = \mu = [1, 2, 3, 4]$ 时误差如下

表 7

h	0.5	0.25	0.125	0.0625
$\|u - u_h\|_{H^1(\Omega)}$	1.599642E-2	7.130159E-3	3.516517E-3	1.900384E-3
H1 误差阶	1.165743	1.019786	0.8878559	0.9186291
$\|u - u_h\|_{L^2(\Omega)}$	3.999106E-3	8.912698E-4	2.197823E-4	5.938701E-5
L2 误差阶	2.165743	2.019786	1.887855	1.918629

图 7



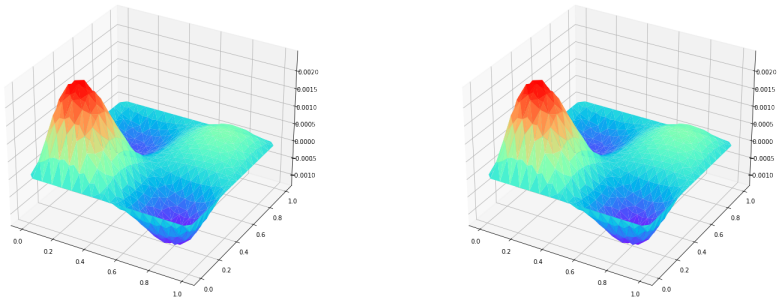
(a) 数值解图像　　(b) 精确解图像

2. 当 Lamé 常数 $\lambda = \mu = [1E4, 2E4, 3E4, 4E4]$ 时误差如下

表 8

h	0.5	0.25	0.125	0.0625
$\|u - u_h\|_{H^1(\Omega)}$	1.599735E-2	7.130099E-3	3.516522E-3	1.900428E-3
H1 误差阶	1.165839	1.019772	0.8878243	0.918610
$\|u - u_h\|_{L^2(\Omega)}$	3.999338E-3	8.912623E-4	2.197826E-4	5.938839E-5
L2 误差阶	2.165839	2.019772	1.887824	1.91861

图 8



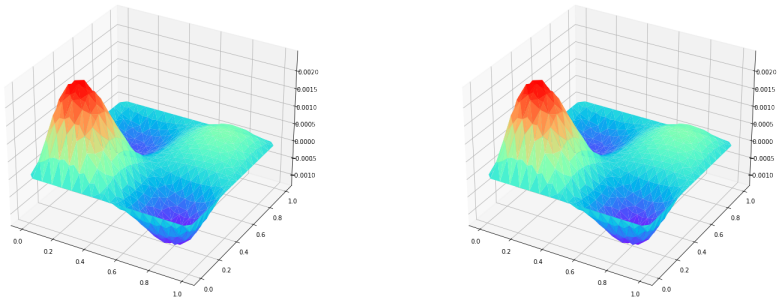
(a) 数值解图像　　(b) 精确解图像

3. 当 Lamé 常数 $\lambda = \mu = [1E8, 2E8, 3E8, 4E8]$ 时误差如下

表 9

h	0.5	0.25	0.125	0.0625
$\|u - u_h\|_{H^1(\Omega)}$	3.015333E-3	1.139928E-3	6.718610E-4	3.793589E-4
H1 误差阶	1.403373	0.7627090	0.8245995	0.9054011
$\|u - u_h\|_{L^2(\Omega)}$	7.538332E-4	1.424911E-5	4.199131E-5	1.185496E-5
L2 误差阶	2.403373	1.7627090	1.824599	1.905401

图 9



(a) 数值解图像　　(b) 精确解图像

3.2.2 算例二

考察以下边值问题

$$\begin{aligned} -div\,\sigma(u) &= f \quad \in \Omega, \\ u|_{\Gamma} &= 0. \end{aligned}$$

其中 $u = (u_1, u_2)^t$ 为求解向量，$f = (f_1, f_2)^t$ 为右端向量，$\Omega = [0,1] \times [0,1]$，$\Omega_1 = [0, 0.5] \times [0, 0.5]$，$\Omega_2 = [0.5, 1] \times [0, 0.5]$，$\Omega_3 = [0, 0.5] \times [0.5, 1]$，$\Omega_4 = [0.5, 1] \times [0.5, 1]$.

当 $(x, y) \in \Omega_1$ 时，$\mu = \lambda = \lambda_1$，

$$\begin{aligned} u_1 &= \sin(\pi x)\,\cos(\pi x)\,\sin(\pi y)\,\cos(\pi y) \,/\, \lambda_1, \\ u_2 &= \sin(\pi x)\,\cos(\pi x)\,\sin(\pi y)\,\cos(\pi y) \,/\, \lambda_1. \end{aligned}$$

当 $(x, y) \in \Omega_2$ 时，$\mu = \lambda = \lambda_2$，

$$\begin{aligned} u_1 &= \sin(\pi x)\,\cos(\pi x)\,\sin(\pi y)\,\cos(\pi y) \,/\, \lambda_2, \\ u_2 &= \sin(\pi x)\,\cos(\pi x)\,\sin(\pi y)\,\cos(\pi y) \,/\, \lambda_2. \end{aligned}$$

当 $(x, y) \in \Omega_3$ 时，$\mu = \lambda = \lambda_3$，

$$\begin{aligned} u_1 &= \sin(\pi x)\,\cos(\pi x)\,\sin(\pi y)\,\cos(\pi y) \,/\, \lambda_3, \\ u_2 &= \sin(\pi x)\,\cos(\pi x)\,\sin(\pi y)\,\cos(\pi y) \,/\, \lambda_3. \end{aligned}$$

当 $(x, y) \in \Omega_4$ 时，$\mu = \lambda = \lambda_4$，

$$\begin{aligned} u_1 &= \sin(\pi x)\,\cos(\pi x)\,\sin(\pi y)\,\cos(\pi y) \,/\, \lambda_4, \\ u_2 &= \sin(\pi x)\,\cos(\pi x)\,\sin(\pi y)\,\cos(\pi y) \,/\, \lambda_4. \end{aligned}$$

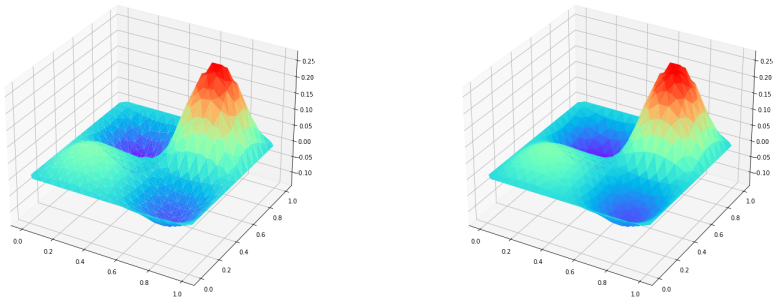
令 $\lambda = [\lambda_1, \lambda_2, \lambda_3, \lambda_4]$，$\mu = [\mu_1, \mu_2, \mu_3, \mu_4]$.

1. 当 Lamé 常数 $\lambda = \mu = [4, 3, 2, 1]$ 时误差如下

表 10: $|u - u_h|_{H^1(\Omega)}$

h	0.5	0.25	0.125	0.0625
$\|u - u_h\|_{H^1(\Omega)}$	1.462698	6.166073E-1	4.261624E-1	2.415138E-1
H1 误差阶	1.246208	0.532948	0.819297	0.939899
$\|u - u_h\|_{L^2(\Omega)}$	7.313491E-1	1.541518E-1	5.327030E-2	1.509461E-2
L2 误差阶	2.246208	1.532948	1.819297	1.939899

图 10



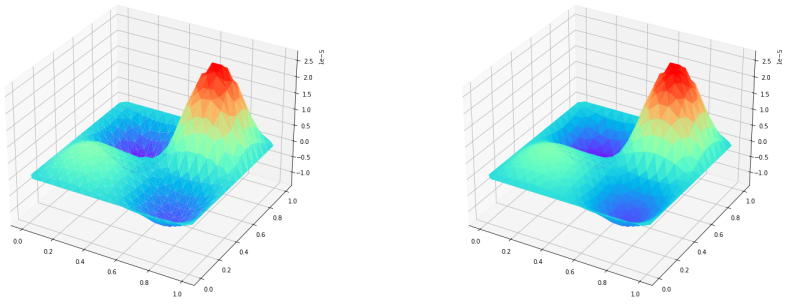
(a) 数值解图像　　(b) 精确解图像

2. 当 Lamé 常数 $\lambda = \mu = [4E4, 3E4, 2E4, 1E4]$ 时误差如下

表 11

h	0.5	0.25	0.125	0.0625
$\|u - u_h\|_{H^1(\Omega)}$	1.462698E-4	6.166073E-5	4.261624E-5	2.415138E-5
H1 误差阶	1.246208	0.532948	0.819297	0.939899
$\|u - u_h\|_{L^2(\Omega)}$	7.313491E-5	1.541518E-5	5.327030E-6	1.509461E-6
L2 误差阶	2.246208	1.532948	1.819297	1.939899

图 11



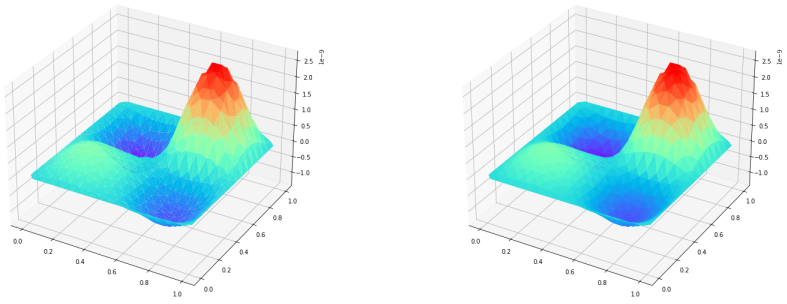
(a) 数值解图像　　(b) 精确解图像

3. 当 Lamé 常数 $\lambda = \mu = [4E8, 3E8, 2E8, 1E8]$ 时误差如下

表 12: $|u - u_h|_{H^1(\Omega)}$

h	0.5	0.25	0.125	0.0625
$\|u - u_h\|_{H^1(\Omega)}$	1.462698E-8	6.166073E-9	2.415138E-9	2.373949E-9
H1 误差阶	1.246208	0.532948	0.819297	0.939899
$\|u - u_h\|_{L^2(\Omega)}$	7.313491E-9	1.541518E-9	5.327030E-10	1.509461E-10
L2 误差阶	2.246208	1.532948	1.819297	1.939899

图 12



(a) 数值解图像　　(b) 精确解图像

4 总结

本文使用 C-R 有限元方法求解了带间断系数的平面弹性问题，分析了非协调元对闭锁现象的影响。数值结果表明，当 Lamé 常数间断且相等时，C-R 元可以有效地解除闭锁现象，并且具有预期的收敛阶。

为了完善本文的研究，未来可以考虑对纯牵引力问题进行数值实验，以检验 C-R 元在不同的边界条件下的表现。同时，也可以通过改变间断系数的大小和形式，以及使用不同的网格划分方式，来进一步探究 C-R 元的有效性和稳定性，以及对间断系数的敏感性。

参考文献

[1] 王兆清, 徐子康, 李金. 不可压缩平面问题的位移-压力混合重心插值配点法. *应用力学学报*, 35(3):631–636, 2018.

[2] 陈绍春, 肖留超. 平面弹性的一个新的 locking-free 非协调有限元. *应用数学*, 20(4):739–747, 2007.

[3] YT Peet and PF Fischer. Legendre spectral element method with nearly incompressible materials. *European Journal of Mechanics-A/Solids*, 44:91–103, 2014.

[4] Arif Masud, Timothy J Truster, and Lawrence A Bergman. A variational multiscale a posteriori error estimation method for mixed form of nearly incompressible elasticity. *Computer Methods in Applied Mechanics and Engineering*, 200(47-48):3453–3481, 2011.

[5] Ferdinando Auricchio, L Beirao Da Veiga, Carlo Lovadina, and Alessandro Reali. An analysis of some mixed-enhanced finite element for plane linear elasticity. *Computer Methods in Applied Mechanics and Engineering*, 194(27-29):2947–2968, 2005.

[6] Peter Hansbo and Mats G Larson. Discontinuous galerkin and the crouzeix–raviart element: application to elasticity. *ESAIM: Mathematical Modelling and Numerical Analysis*, 37(1):63–72, 2003.

[7] 李荣华, 刘播. 偏微分方程数值解, 2007.

[8] 陈纪修, 於崇华, 金路. 数学分析. 高等教育出版社, 2004.

[9] Susanne C Brenner, L Ridgway Scott, and L Ridgway Scott. *The mathematical theory of finite element methods*, volume 3. Springer, 2008.

致谢

时光荏苒, 岁月如梭, 转眼间大学生活来到了最后阶段. 当我写完这篇毕业论文的时候, 有一种如释重负的感觉, 感慨颇多. 回首大学四年, 得到过太多人的帮助了. 首先诚挚的感谢我的论文指导老师王华老师. 本文的研究工作都是在王华老师的悉心指导下完成的. 王老师平易近人, 严谨务实, 由于我知识储备不足, 在论文撰写过程中遇到了许多困难和疑惑, 王华老师都及时给予指点, 耐心解释所犯的错误, 投入了大量的心血和精力, 更是不厌其烦地帮我察看论文中的小漏洞. 王华老师对我的帮忙和关怀实在无法用言语表明. 还要感谢所有的老师们, 正是因为有了他们的督促和教导才能让我在这四年的学习生活里受益匪浅, 快速汲取专业知识, 提升专业能力. 同时也要感谢组内的同学们, 是他们以极大的热情来解答我在理论和程序上的疑问, 帮忙收集资料, 让平淡的日子不再那么枯燥乏味. 最后还要感谢我的家人, 是他们的支持与付出才给了我学习的机会, 感谢一直对我的理解, 这是我不断前进的动力.

附录 A　附录数值程序

Listing 1: **elasticityCR.py**

```
## 模型与剖分

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

from tool import interfaceData, getIsBdNode, uniform_refine,
    get_cr_node_cell
from tool import get_A1_A2_F, my_solve, H1Error, print_error, print_P
from tool import get_stiff_and_div_matrix, get_bb, drawer_uh_u
from tool import getCellInOmega

n   = 3  #剖分次数
n   = n + 1
Lam = np.array([[1e0, 2e0, 3e0, 4e0],
                [1e1, 2e1, 3e1, 4e1],
                [1e2, 2e2, 3e2, 4e2],
                [1e3, 2e3, 3e3, 4e3],
                [1e4, 2e4, 3e4, 4e4]])

#Lam = np.array([[1e0, 2e0, 3e0, 4e0]])

Mu  = Lam

H     = np.zeros(n)                                  #步长
P     = np.zeros(Lam.shape[0])                       #误差阶
E     = np.zeros((Lam.shape[0],n), dtype=np.float64) #每个lambda(行)对应
    的误差(列)

for i in  range(Lam.shape[0]):
    pde = interfaceData(Lam[i], Mu[i])
    node = pde.node
    cell = pde.cell
    for j in  range(n):
        NC = cell.shape[0]
        #print("cr_NC= ", NC)
        # nn 特定情况下剖分次数
        nn  = math.log(NC/2, 4)
        NN =  int(3 * 2 * 4**nn - (3 * 2 * 4**nn - 4 * 2**nn) / 2)
```

```
38          cm = np.ones(NC, dtype=np.float64) / NC
39
40          cr_node, cr_cell = get_cr_node_cell(node,cell)
41
42          # 单元刚度矩阵和单元载荷向量
43          A1, A2 = get_stiff_and_div_matrix(cr_node, cr_cell, cm)
44          bb = get_bb(pde, node, cell, cm)
45          cellInOmega = getCellInOmega(cr_node, cr_cell)
46          kk = 1
47          for k in  range(4):
48              A1[cellInOmega[k]] *= pde.mu[k]
49              A2[cellInOmega[k]] *= pde.mu[k] + pde.lam[k]
50          A1, A2, F = get_A1_A2_F(A1, A2, bb, cr_node, cr_cell)
51          A = A1 + A2
52
53          uh = my_solve(A, F, cr_node, getIsBdNode)
54          u = pde.solution(cr_node, cr_node)
55          H[j] = np.sqrt(2 * cm[0]) / 2
56          E[i][j] = H1Error(u, uh)
57          if j < n-1:
58              node, cell = uniform_refine(node, cell)
59      drawer_uh_u(cr_node, uh, u, "../../image/tmp/interface_uh_u/uh_lam
            ={}.png".
            format(Lam[i]), "../../image/tmp/interface_uh_u/u_lam={}.png".
            format(Lam[i]))
60
61  # 画图 得到误差阶
62  if n-1 > 1:
63      # 画图
64      for i in  range( len(Lam)):
65          fig = plt.figure()
66          plt.plot(np.log(H[1:]), np.log(E[i][1:]))
67          plt.title("lam={}". format(Lam[i]))
68          plt.xlabel("log(h)")
69          plt.ylabel("log(e)")
70          plt.savefig(fname="interfaceCRFem/elasticityCRFemLam_{}.png"
71                      . format(Lam[i]))
72          plt.close(fig)
73      # 求误差阶
74      # 得到 P
75      for i in  range( len(Lam)):
76          f = np.polyfit(np.log(H[1:]), np.log(E[i][1:]) ,1)
77          P[i] = f[0]
```

```
78
79 print_error(Lam, H, E)
80 if n-1 > 1:
81     print_P(Lam, P)
```

Listing 2: **tool.py**

```
1  import math
2  import numpy as np
3  from numpy.linalg import solve
4  from scipy.sparse import csr_matrix
5  from scipy.sparse.linalg import spsolve
6
7  # PDE1
8  # u1 = y(x-1)(y-1)sin(x)
9  # u2 = x(x-1)(y-1)sin(y)
10 # uh_dir = "../../image/tmp/elaticity_uh_u/PDE1/uh_lam={}.png".format(
       Lam[i])
11 # u_dir = "../../image/tmp/elaticity_uh_u/PDE1/u_lam={}.png".format(Lam[
       i])
12 class PDE():
13     def __init__(self, mu=1, lam=1):
14         self.mu  = mu
15         self.lam = lam
16         self.node = np.array([
17             (0,0),
18             (1,0),
19             (1,1),
20             (0,1)], dtype=np.float64)
21         self.cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)
22
23     def source(self, p):
24         x   = p[..., 0]
25         y   = p[..., 1]
26         mu  = self.mu
27         lam = self.lam
28
29         sin = np.sin
30         cos = np.cos
31         val = np.zeros(p.shape, dtype=np.float64)
32
33         frac_u1_x   = y * (y-1) * (2*cos(x) - (x-1) * sin(x))
34         frac_u1_y   = 2 * (x-1) * sin(x)
```

```
35          frac_u1_x_y = (2*y-1) * (sin(x) + (x-1) * cos(x))
36          frac_u2_x   = 2 * (y-1) * sin(y)
37          frac_u2_y   = x * (x-1) * (2*cos(y) - (y-1) * sin(y))
38          frac_u2_x_y = (2*x-1) * (sin(y) + (y-1) * cos(y))
39
40          val[..., 0] = -((2*mu+lam) * frac_u1_x + (mu+lam) * frac_u2_x_y
                + mu*frac_u1_y)
41          val[..., 1] = -((2*mu+lam) * frac_u2_y + (mu+lam) * frac_u1_x_y
                + mu*frac_u2_x)
42
43          return val
44
45      def solution(self, p):
46          x = p[..., 0]
47          y = p[..., 1]
48
49          val = np.zeros(p.shape, dtype=np.float64)
50
51          val[..., 0] = y * (x - 1) * (y - 1) * np.sin(x)
52          val[..., 1] = x * (x - 1) * (y - 1) * np.sin(y)
53
54          return val
55
56  # PDE2
57  # u1 = u2 = x^2 * sin(x-1) * y^2 * sin(y-1)
58  # uh_dir = "../../image/tmp/elaticity_uh_u/PDE2/uh_lam={}.png".format(
        Lam[i])
59  # u_dir = "../../image/tmp/elaticity_uh_u/PDE2/u_lam={}.png".format(Lam[
        i])
60  class PDE2():
61      def __init__(self, mu=1, lam=1):
62          self.mu  = mu
63          self.lam = lam
64          self.node = np.array([
65              (0,0),
66              (1,0),
67              (1,1),
68              (0,1)], dtype=np.float64)
69          self.cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)
70
71      def source(self, p):
72          x   = p[..., 0]
```

```
73          y   = p[..., 1]
74          mu  = self.mu
75          lam = self.lam
76
77          sin = np.sin
78          cos = np.cos
79          val = np.zeros(p.shape, dtype=np.float64)
80
81          ux         = x**2 * sin(x-1)
82          uy         = y**2 * sin(y-1)
83          frac_ux_x  = 2 * x * sin(x-1) + x**2 * cos(x-1)
84          frac_ux_xx = 2 * sin(x-1) + 2 * x *cos(x-1) + 2 * x * cos(x-1) -
                 x**2 * sin(x-1)
85          frac_uy_y  = 2 * y * sin(y-1) + y**2 * cos(y-1)
86          frac_uy_yy = 2 * sin(y-1) + 2 * y *cos(y-1) + 2 * y * cos(y-1) -
                 y**2 * sin(y-1)
87
88          frac_u1_x   = frac_ux_xx * uy
89          frac_u1_y   = ux * frac_uy_yy
90          frac_u1_x_y = frac_ux_x * frac_uy_y
91          frac_u2_x   = frac_ux_xx * uy
92          frac_u2_y   = ux * frac_uy_yy
93          frac_u2_x_y = frac_ux_x * frac_uy_y
94
95          val[..., 0] = -((2*mu+lam) * frac_u1_x + (mu+lam) * frac_u2_x_y
                + mu*frac_u1_y)
96          val[..., 1] = -((2*mu+lam) * frac_u2_y + (mu+lam) * frac_u1_x_y
                + mu*frac_u2_x)
97
98          return val
99
100     def solution(self, p):
101         x = p[..., 0]
102         y = p[..., 1]
103
104         sin = np.sin
105         val = np.zeros(p.shape, dtype=np.float64)
106
107         ux = x**2 * sin(x-1)
108         uy = y**2 * sin(y-1)
109
110         val[..., 0] = ux * uy
```

```
111         val[..., 1] = ux * uy
112 
113         return val
114 
115 # PDE3
116 # u1 = u2 = -(x-1) * (e^x-1) * (y-1) * (e^y-1)
117 # uh_dir = "../../image/tmp/elaticity_uh_u/PDE3/uh_lam={}.png".format(
        Lam[i])
118 # u_dir = "../../image/tmp/elaticity_uh_u/PDE3/u_lam={}.png".format(Lam[
        i])
119 class PDE3():
120     def __init__(self, mu=1, lam=1):
121         self.mu  = mu
122         self.lam = lam
123         self.node = np.array([
124             (0,0),
125             (1,0),
126             (1,1),
127             (0,1)], dtype=np.float64)
128         self.cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)
129 
130     def source(self, p):
131         x   = p[..., 0]
132         y   = p[..., 1]
133         mu  = self.mu
134         lam = self.lam
135 
136         sin = np.sin
137         cos = np.cos
138         exp = np.exp
139         val = np.zeros(p.shape, dtype=np.float64)
140 
141         ux = (x-1) * (exp(x) - 1)
142         uy = (y-1) * (exp(y) - 1)
143         frac_ux_x = x * exp(x) - 1
144         frac_ux_xx = exp(x) * (x+1)
145         frac_uy_y = y * exp(y) - 1
146         frac_uy_yy = exp(y) * (y+1)
147 
148         frac_u1_x   = frac_ux_xx * uy
149         frac_u1_y   = ux * frac_uy_yy
150         frac_u1_x_y = frac_ux_x * frac_uy_y
```

```
151         frac_u2_x   = frac_ux_xx * uy
152         frac_u2_y   = ux * frac_uy_yy
153         frac_u2_x_y = frac_ux_x * frac_uy_y
154 
155         val[..., 0] = -((2*mu+lam) * frac_u1_x + (mu+lam) * frac_u2_x_y
                + mu*frac_u1_y)
156         val[..., 1] = -((2*mu+lam) * frac_u2_y + (mu+lam) * frac_u1_x_y
                + mu*frac_u2_x)
157 
158         return -val
159 
160     def solution(self, p):
161         x = p[..., 0]
162         y = p[..., 1]
163 
164         val = np.zeros(p.shape, dtype=np.float64)
165 
166         ux = (x-1) * (np.exp(x) - 1)
167         uy = (y-1) * (np.exp(y) - 1)
168 
169         val[..., 0] = ux * uy
170         val[..., 1] = ux * uy
171 
172         return -val
173 
174 # interfaceData1
175 # u1 = u2 = x(x-0.5)(x-1) * y(y-0.5)(y-1)
176 # uh_dir = "../../image/tmp/interface_uh_u/PDE1/uh_lam={}.png".format(
        Lam[i])
177 # u_dir = "../../image/tmp/interface_uh_u/PDE1/u_lam={}.png".format(Lam[
        i])
178 class interfaceData():
179     def __init__(self, mu=np.array([1,2,3,4]), lam=np.array([1,2,3,4])):
180         self.mu = mu
181         self.lam = lam
182         self.node = np.array([(0,0),
183                               (0.5,0),
184                               (1,0),
185                               (0,0.5),
186                               (0.5,0.5),
187                               (1,0.5),
188                               (0,1),
```

```
                        (0.5,1),
                        (1,1)], dtype=np.float64)
        self.cell = np.array([[0,1,4],
                [0,4,3],
                [1,2,5],
                [1,5,4],
                [3,4,7],
                [3,7,6],
                [4,5,8],
                [4,8,7]], dtype=np.int64)

    def source(self, p):
        x   = p[..., 0]
        y   = p[..., 1]

        mu  = 1
        lam = 1
        val = np.zeros(p.shape, dtype=np.float64)

        frac_u1_x   = (6*x-3) * (y**3-(3/2)*y**2+(1/2)*y)
        frac_u1_y   = (x**3-(3/2)*x**2+(1/2)*x) * (6*y-3)
        frac_u1_x_y = (3*x**2-3*x+1/2) * (3*y**2-3*y+1/2)
        frac_u2_x   = frac_u1_x
        frac_u2_y   = frac_u1_y
        frac_u2_x_y = frac_u1_x_y

        val[..., 0] = -((2*mu+lam) * frac_u1_x + (mu+lam) * frac_u2_x_y
            + mu*frac_u1_y)
        val[..., 1] = -((2*mu+lam) * frac_u2_y + (mu+lam) * frac_u1_x_y
            + mu*frac_u2_x)

        return val

    def solution(self, p, cr_node):
        x = p[..., 0]
        y = p[..., 1]

        val = np.zeros(p.shape, dtype=np.float64)

        val[..., 0] = x * (x - 0.5) * (x - 1) * y * (y - 0.5) * (y - 1)
        val[..., 1] = x * (x - 0.5) * (x - 1) * y * (y - 0.5) * (y - 1)

```

```
229         crCell = getWhichCell(cr_node)
230         for i in  range(4):
231             val[crCell[i], :] /= self.lam[i]
232 
233         return val
234 
235 def H1Error(u, uh):
236     tmp = u - uh
237     e = np.einsum("ni, ni -> n", tmp, tmp)
238      sum = e. sum()
239     return np.sqrt( sum)
240 
241 def print_error(Lam, H, E):
242     for i in  range( len(Lam)):
243         print("---------------------Lam= {}----------------------".
                format(Lam[i]))
244         n = H.shape[0]
245         print()
246         for j in  range(n):
247             print("h= ", H[j])
248             print("e=", E[i][j])
249             print()
250         print()
251 
252 def print_P(Lam, P):
253     print("---------------------误差阶----------------------")
254     for i in  range( len(Lam)):
255         print("lam= ", Lam[i])
256         print("p= ", P[i])
257         print()
258 
259 #判断 P (维度[2]) 是否在 cr_node, 是则放回其下标, 否则 (val = 1 时) 将 P
      加入 cr_node 并返回下标
260 def is_in_cr_node(p, cr_node):
261     #p 不会为[0,0]
262     index = np.where((cr_node == p). all(axis=1))[0]
263     if  len(index):
264         return index[0]
265     else:
266         in_index = np.where((cr_node == np.array([0,0])).
                all(axis=1))[0]
267         if  len(in_index) == 0:
268             print("cr_node= ", cr_node)
```

```
269             raise Exception("数组cr_node已满")
270         cr_node[in_index[0]] = p
271         return in_index[0]
272 
273 #判断 P（维度[2]）是否在 node，是则放回其下标，否则将 P 加入 node 并返回下标
274 def is_in_node(p, node):
275     #p 不会为[0,0]
276     index = np.where((node == p). all(axis=1))[0]
277     if  len(index):
278         return index[0]
279     else:
280         in_index = np.where((node == np.array([0,0])). all(axis=1))[0]
281         if  len(in_index) == 1:
282             print("node= ", node)
283             raise Exception("数组node已满")
284         node[in_index[1]] = p
285         return in_index[1]
286 
287 # a_cell 是否属于 cell，是则返回下标， 否则返回 -1
288 def is_in_cell(a_cell, cell):
289     i = np.where((cell == a_cell). all(axis=1))[0]
290     if  len(i):
291         return i[0]
292     else:
293         return -1
294 
295 #将 a_cell（维数[3]）放入new_cr_cell
296 def push_cr_cell(a_cell, new_cr_cell):
297     in_index = np.where((new_cr_cell == np.array([0,0,0])).
        all(axis=1))[0]
298     if  len(in_index) == 0:
299         raise Exception("数组cr_cell已满")
300     new_cr_cell[in_index[0]] = a_cell
301     return new_cr_cell
302 
303 
304 # 对单个三角形 a_cell_node（维度 [3, 2]）求三条边中点 p1, p2, p3 并将其放入 new_cr_node 、 new_cr_cell
305 def a_creat(a_cell_node, new_cr_node, new_cr_cell):
306     p1 = (a_cell_node[0] + a_cell_node[1]) / 2
307     p2 = (a_cell_node[0] + a_cell_node[2]) / 2
308     p3 = (a_cell_node[1] + a_cell_node[2]) / 2
```

```
309
310     p1_i = is_in_cr_node(p1, new_cr_node)
311     p2_i = is_in_cr_node(p2, new_cr_node)
312     p3_i = is_in_cr_node(p3, new_cr_node)
313
314     push_cr_cell([p1_i, p2_i, p3_i], new_cr_cell)
315
316     return new_cr_node, new_cr_cell
317
318 def refine_a_cell(a_cell, new_node, new_cell):
319     p1 = (new_node[a_cell][0] + new_node[a_cell][1]) / 2
320     p2 = (new_node[a_cell][0] + new_node[a_cell][2]) / 2
321     p3 = (new_node[a_cell][1] + new_node[a_cell][2]) / 2
322
323     p1_i = is_in_node(p1, new_node)
324     p2_i = is_in_node(p2, new_node)
325     p3_i = is_in_node(p3, new_node)
326
327     push_cr_cell([p1_i, p2_i, p3_i], new_cell)
328     push_cr_cell([a_cell[0], p1_i, p2_i], new_cell)
329     push_cr_cell([p1_i, a_cell[1], p3_i], new_cell)
330     push_cr_cell([p2_i, p3_i, a_cell[2]], new_cell)
331
332     return new_node, new_cell
333
334 # 从剖分node, cell得到 cr_node, cr_cell
335 # 单元数 NC
336 # 剖分次数 n : log_4(NC / 2)
337 # 外边 out_edge : 4 * 2**n
338 # 总边 all_edge :  3 * NC - (3 * NC - out_edge) / 2
339 def get_cr_node_cell(node, cell):
340     NC = cell.shape[0]
341     # n 特定情况下剖分次数
342     n  = math.log(NC/2, 4)
343     NN =  int(3 * 2 * 4**n - (3 * 2 * 4**n - 4 * 2**n) / 2)
344     cr_node = np.zeros((NN, 2), dtype=np.float64)
345     cr_cell = np.zeros_like(cell)
346
347     for i in  range(NC):
348         cr_node, cr_cell = a_creat(node[cell[i]], cr_node, cr_cell)
349
350     return cr_node, cr_cell
```

```
351
352 # 返回 node 中是否为边界点的信息
353 # isBdNode [NN] bool
354 def getIsBdNode(cr_node):
355     is_BdNode = np.zeros(cr_node.shape[0], dtype= bool)
356     for i in  range(cr_node.shape[0]):
357         a = np. min(np. abs(cr_node[i] - np.array([0,0])))
358         b = np. min(np. abs(cr_node[i] - np.array([1,1])))
359         if a < 1e-13 or b < 1e-13:
360             is_BdNode[i] = True
361     return is_BdNode
362
363 def getIsBdLineNode(cr_node):
364     NN = cr_node.shape[0]
365     isBdLineNode = getIsBdNode(cr_node)
366     for i in  range(NN):
367         a = cr_node[i,0]
368         b = cr_node[i,1]
369         if a == 0.5 or b == 0.5:
370             isBdLineNode[i] = True
371     return isBdLineNode
372
373 def uniform_refine(node, cell):
374     old_NN = node.shape[0]
375     old_NC = cell.shape[0]
376     n  = math.log(old_NC/2, 4)
377     NC = 4 * old_NC
378     num_edge =  int(3 * 2 * 4**n - (3 * 2 * 4**n - 4 * 2**n) / 2)
379     NN = old_NN + num_edge
380
381     new_node = np.zeros((NN, 2), dtype=np.float64)
382     new_cell = np.zeros((NC, 3), dtype=np.int64)
383     new_node[:old_NN] = node
384
385     for i in  range(old_NC):
386         new_node, new_cell = refine_a_cell(cell[i], new_node, new_cell)
387
388     return new_node, new_cell
389
390 def get_cr_glam_and_pre(cr_node, cr_cell):
391     NC = cr_cell.shape[0]
392     NN = cr_node.shape[0]
```

```
393     cr_node_cell = cr_node[cr_cell]
394     ##求解CR元导数
395     cr_node_cell_A = np.ones((NC, 3, 3), dtype=np.float64)
396     #求解CR元导数的系数矩阵
397     cr_node_cell_A[:, :, 0:2] = cr_node_cell
398     #用于求解CR元的值
399     # cr_glam_x_y_pre [NC, 3, 3]
400     cr_glam_x_y_pre = np.zeros((NC, 3, 3), dtype=np.float64)
401     for k in  range(NC):
402         cr_glam_x_y_pre[k, :, :] = solve(cr_node_cell_A[k, :, :], np.
            diag(np.ones(3)))
403     #[NC,3,3]
404     cr_glam_x_y = np.copy(cr_glam_x_y_pre)
405     cr_glam_x_y = cr_glam_x_y[:, 0:2, :]
406     cr_glam_x_y = cr_glam_x_y.transpose((0,2,1))
407     return cr_glam_x_y, cr_glam_x_y_pre
408
409 # phi_val [NC,3(点),6(6个基函数),2(两个分量)]
410 def get_phi_val(node, cell, cr_glam_pre):
411     NC = cell.shape[0]
412     # cr_node_val [NC,3(点),3(三个cr元的值)] CR元在各顶点的值
413     node_cell_A = np.ones((NC,3,3), dtype=np.float64)
414     node_cell_A[:,:,0:2] = node[cell]
415     cr_node_val = np.einsum("cij, cjk -> cik", node_cell_A, cr_glam_pre)
416
417     # phi_node_val [NC,3(点),6(6个基函数),2(两个分量)]
418     phi_node_val = np.zeros((NC,3,6,2), dtype=np.float64)
419     phi_node_val[:,:,0:5:2,0] = cr_node_val
420     phi_node_val[:,:,1:6:2,1] = cr_node_val
421     return phi_node_val
422
423 def get_phi_grad_and_div(cr_node, cr_cell):
424     NC = cr_cell.shape[0]
425     cr_glam_x_y, cr_glam_x_y_pre = get_cr_glam_and_pre(cr_node, cr_cell)
426     #求 cr_phi_grad [NC,6(基函数),2(分量 x , y),2(导数)]
427     cr_phi_grad = np.zeros((NC,6,2,2), dtype=np.float64)
428     cr_phi_grad[:, 0:5:2, 0, :] = cr_glam_x_y
429     cr_phi_grad[:, 1:6:2, 1, :] = cr_glam_x_y
430
431     # cr_phi_div [NC, 6]
432     #cr_phi_div = np.einsum("cmij -> cm", cr_phi_grad)
433     cr_phi_div = cr_glam_x_y.copy()
```

```
434     cr_phi_div = cr_phi_div.reshape(NC, 6)
435     return cr_phi_grad, cr_phi_div
436 
437 ## 单元刚度矩阵, 单元质量矩阵 stiff, div [NC, 6, 6]
438 def get_stiff_and_div_matrix(cr_node, cr_cell, cm):
439     cr_phi_grad, cr_phi_div = get_phi_grad_and_div(cr_node, cr_cell)
440 
441     ## 单元刚度矩阵
442     # A1 A2 [NC, 6, 6]
443     A1 = np.einsum("cnij, cmij, c -> cnm", cr_phi_grad, cr_phi_grad, cm)
444     A2 = np.einsum("cn, cm, c -> cnm", cr_phi_div, cr_phi_div, cm)
445     return A1, A2
446 
447 ## 单元载荷向量 bb [NC, 6]
448 def get_bb(pde, node, cell, cm):
449     cr_node, cr_cell = get_cr_node_cell(node,cell)
450     cr_glam_x_y, cr_glam_x_y_pre = get_cr_glam_and_pre(cr_node, cr_cell)
451     # phi_val [NC,3(点),6(6个基函数),2(两个分量)]
452     phi_node_val = get_phi_val(node, cell, cr_glam_x_y_pre)
453 
454     # val [NC,3(点),2(分量)] 右端项在各顶点的值
455     val = pde.source(node[cell])
456 
457     # phi_val [NC,3,6] 基函数和右端项的点乘
458     phi_val = np.einsum("cijk, cik -> cij", phi_node_val, val)
459     # bb [NC,6]
460     bb = phi_val. sum(axis=1) * cm[0] / 3
461     return bb
462 
463 #input
464 #单元刚度矩阵 A1, A2 [NC, 6, 6], bb [NC,6]
465 # output
466 # 总刚度矩阵 A1, A2 [2*NN,2*NN], F [2*NN]
467 def get_A1_A2_F(A1, A2, bb, cr_node, cr_cell):
468     NN = cr_node.shape[0]
469     NC = cr_cell.shape[0]
470     # cell_x_y [NC, 3(三个点), 2(x y 方向上基函数的编号)]
471     cell_x_y = np.broadcast_to(cr_cell[:,:,None], shape=(NC, 3, 2)).copy
            ()
472     cell_x_y[:,:,0] = 2 * cell_x_y[:,:,0]       #[NC,3] 三个节点x方向上
            基函数在总刚度矩阵的位置
473     cell_x_y[:,:,1] = 2 * cell_x_y[:,:,1] + 1   #[NC,3] 三个节点y方向上
```

```
        基函数在总刚度矩阵的位置
    cell_x_y = cell_x_y.reshape(NC, 6)
    I = np.broadcast_to(cell_x_y[:, :, None], shape=A1.shape)
    J = np.broadcast_to(cell_x_y[:, None, :], shape=A2.shape)

    A1 = csr_matrix((A1.flat, (I.flat, J.flat)), shape=(2 * NN,2 * NN))
    A2 = csr_matrix((A2.flat, (I.flat, J.flat)), shape=(2 * NN,2 * NN))
    F = np.zeros(2 * NN)
    np.add.at(F, cell_x_y, bb)
    return A1, A2, F

def my_solve(A, F, cr_node, getIsBdNode):
    NN = cr_node.shape[0]
    isBdNode    = getIsBdNode(cr_node)
    isInterNode = ~isBdNode
    #print("isInterNode= ", isInterNode)
    isInterNodeA = np.broadcast_to(isInterNode[:, None], shape=(NN, 2))
    isInterNodeA = isInterNodeA.reshape(2 * NN)
    #print("isInterNodeA= ", isInterNodeA)

    uh = np.zeros((2 * NN), dtype=np.float64)
    uh[isInterNodeA] = spsolve(A[:, isInterNodeA][isInterNodeA], F[
        isInterNodeA])
    #uh = spsolve(A, F)
    #print("uh= ", uh)
    uh = uh.reshape(NN, 2)
    return uh

## [4,NN] 返回各点属于哪个区间
## \Omega_1 [0,0.5]   \times [0,0.5)
## \Omega_2 (0.5, 1]  \times [0,0.5]
## \Omega_3 [0,0.5)   \times [0.5,1]
## \Omega_4 [0.5,1] \times (0.5,1]
def getWhichCell(node):
    isWhichCellNode = np.zeros((4,node.shape[0]), dtype= bool)
    for i in  range(node.shape[0]):
        a = node[i, 0] - 0
        b = node[i, 1] - 0
        if a <= 0.5 and b < 0.5:
            isWhichCellNode[0,i] = True
        if a > 0.5 and b <= 0.5:
            isWhichCellNode[1,i] = True
```

```
514         if a < 0.5 and b >= 0.5:
515             isWhichCellNode[2,i] = True
516         if a >= 0.5 and b > 0.5:
517             isWhichCellNode[3,i] = True
518     return isWhichCellNode
519 
520 ## [4, NC] 返回各单元属于哪个区间
521 def getCellInOmega(node, cell):
522     cellInOmega = np.zeros(cell.shape[0], dtype= bool)
523     #print("node[cell]= ", node[cell])
524     mid_p = node[cell]. sum(axis=1) / 3
525     #print("mid_p= ", mid_p)
526     cellInOmega = getWhichCell(mid_p)
527     return cellInOmega
528 
529 ## [4,NN]
530 ## \line_1 x=0.5, y<0.5
531 ## \line_2 x=0.5, y>0.5
532 ## \line_3 x<0.5, y=0.5
533 ## \line_4 x>0.5, y=0.5
534 def getInterfaceCell(node):
535     interfaceCell = np.zeros((4,node.shape[0]), dtype= bool)
536     for i in  range(node.shape[0]):
537         a = node[i,0]
538         b = node[i,1]
539         if a == 0.5 and b < 0.5:
540             interfaceCell[0,i] = True
541         if a == 0.5 and b > 0.5:
542             interfaceCell[1,i] = True
543         if a < 0.5 and b == 0.5:
544             interfaceCell[2,i] = True
545         if a > 0.5 and b == 0.5:
546             interfaceCell[3,i] = True
547     return interfaceCell
548 
549 def getInterLineNode(node):
550     NN = node.shape[0]
551     lineNode = np.zeros(NN, dtype= bool)
552     for i in  range(NN):
553         a = node[i,0]
554         b = node[i,1]
555         if a == 0.5 or b == 0.5:
```

```
556             lineNode[i] = True
557     return lineNode
558
559 def phiInWhichCell(whichCell):
560     phiCell = np.broadcast_to(whichCell[:, :, None], shape=(4, whichCell
            .shape[1], 2))
561     phiCell = phiCell.reshape(4, 2 * whichCell.shape[1])
562     return phiCell
563
564 # uh_dir = "../../image/tmp/elaticity_uh_u/uh_lam={}.png".format(Lam[i])
        "
565 # u_dir = "../../image/tmp/elaticity_uh_u/u_lam={}.png".format(Lam[i])
566 def drawer_uh_u(cr_node, uh, u, uh_dir, u_dir):
567     import matplotlib.pyplot as plt
568     fig = plt.figure(figsize=(10,10))
569     ax = fig.add_subplot(111, projection='3d')
570
571     x = cr_node[:,0]
572     y = cr_node[:,1]
573
574     ax.plot_trisurf(x, y, uh[:,0], cmap='rainbow')
575     plt.savefig(fname=uh_dir)
576
577     ax.plot_trisurf(x, y, u[:,0], cmap='rainbow')
578     plt.savefig(fname=u_dir)
579     plt.close(fig)
```