

# 带间断系数的弹性问题

2023 年 5 月 8 日

## 摘要

客观世界下的很多能接触到的材料都不仅仅是单一的材料构成,往往由不同种材料复合而来。其中不同材料复合而来的物理现象所构成的数学模型即为界面问题。材料之间的间断面称之为界面,且在上述模型中,模型的解通常受间断系数的影响需要满足某种跳跃条件,即服从某种意义上的守恒律。界面问题在足够光滑的界面条件下,其解也会在此区域上光滑。但是由于解在界面上的跳跃,导致使用一般的数值解法很难在解的整体光滑性较差的情况下还能有比较理想的逼近效果和逼近精度。本文采用非协调有限元对二维情况下带间断系数的弹性问题进行求解

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

## Abstract

Many materials in the objective world that can be touched are not composed of a single material, but often composed of different materials. The mathematical model composed of the physical phenomena of different materials is called the interface problem. The discontinuous surface between materials is called the interface, and in the above model, the solution of the model usually needs to satisfy a certain jump condition under the influence of the discontinuous coefficient, that is, it obeys a certain sense of conservation law. The solution of the interface problem will also be smooth in this area under sufficiently smooth interface conditions. But because of the jump of the solution on the interface, it is difficult to use general numerical methods to achieve ideal approximation effect and approximation accuracy when the overall smoothness of the solution is poor. In this paper, non-conforming finite element method is used to solve the elastic problem with discontinuous coefficient in two-dimensional case.

**Key words:** linear elasticity problem; discontinuous coefficient; non-conforming finite element; locking-free

# 目录

<b>1 绪论</b>	<b>2</b>
1.1 研究背景 . . . . .	2
1.2 国内外研究现状 . . . . .	2
<b>2 有限元理论</b>	<b>3</b>
2.1 Sobolev 空间 . . . . .	3
2.2 弹性问题 . . . . .	4
2.2.1 边值问题 . . . . .	4
2.2.2 变分 . . . . .	5
2.2.3 引入间断系数 . . . . .	6
2.3 离散 . . . . .	6
2.3.1 Galerkin 法 . . . . .	6
2.3.2 线性元 . . . . .	8
2.3.3 C-R 元 . . . . .	9
2.4 误差估计 . . . . .	10
2.5 闭锁现象 . . . . .	10
<b>3 算例</b>	<b>12</b>
3.1 弹性问题 . . . . .	12
3.1.1 算例一 . . . . .	12
3.1.2 算例二 . . . . .	13
3.1.3 算例三 . . . . .	14
3.2 带间断系数的弹性问题 . . . . .	15
3.2.1 算例 . . . . .	15
<b>4 总结</b>	<b>18</b>
<b>参考文献</b>	<b>19</b>
<b>致谢</b>	<b>20</b>
<b>A 附录数值程序</b>	<b>21</b>

# 1 绪论

## 1.1 研究背景

具有间断系数的方程成为交界面问题。这类问题起源于许多应用领域,例如 2 种不同材料或者相同材料在不同状态下物理机制的研究<sup>[1]</sup>、声音在水中的传递、海市蜃楼现象等,同时在数值模拟等场景中也有大量的界面问题。

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<sup>[2]</sup>。对于各向同性均匀介质的平面弹性问题,当材料的 Lamé 常数  $\lambda \rightarrow \infty$  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<sup>[3]</sup>。

为了消除(近)不可压缩弹性问题中遇到的锁定现象,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<sup>[4]</sup>、混合有限元法<sup>[5]</sup>、增强有限元法<sup>[6]</sup>和不连续 Galerkin 法<sup>[7]</sup>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<sup>[2]</sup>。

本文将通过数值实验的方法,考察对于具有间断系数的平面弹性问题,使用 C-R 元是否仍可以解除闭锁现象。

## 1.2 国内外研究现状

1943 年, R.W.Courant 首次提出有限元法的核心思想。1956 年, R.W.Clough 等四位教授与工程师在科技期刊上发表一篇计算飞机机翼强度的论文,且把这种解法称之为刚性法 (Stiffness), 是有限元法在工程学界上的开端。1960 年, R.W.Clough 教授发表的平面弹性论文中,“有限元法”这个名称被首次使用,同时也将有限元法扩展到土木工程上。1963 年, Richard MacNeal 博士与 Robert Schwendler 联手创办了 MSC 公司,并开发了第一个软件程序,名为 SADSAM,即数字仿真模拟结构分析,标志着有限元方法 (FEA) 由理论向程序的转变,1964-1965 年, O.C.Zienkiewicz 等人发表关于利用极小位能原理和虚功原理,以新的思路推导出有限元法。

我国有限元发展之路中较为著名的有:冯康(有限单元法理论),钱令希(余能定理),钱伟长(广义变分原理)等等。但是受限与当时的国际和国内环境,我国的学者在有限元方法上的进一步研究困难重重,很难跟上国际潮流,遗憾的被国外拉开距离。在之后的十年,伴随者国内有限元软件和有限元方法理论的诞生和发展,大型工程也逐渐使用有限元方法来计算,如

水利、机械等多个领域，并且也都取得了不错的效果。上世纪 90 年代，国外的有限元软件大批量地涌入国内市场，涉及到各个领域。国外的学者专家也都进入各大学、工厂与企业进行宣传他们所掌握的技术和使用技巧，导致国内有限元发展更加困难。管理部门对有限元软件的认知上产生了偏差，对此失去了必要的支持，核心技术掌握在国外，所以至上世纪最后十几年里，国内自主技术创新的进度十分缓慢。但是进入 21 世纪后，国内自主知识产权的软件逐渐市场化，获得了一定的发展，同时也获得了国家对有限元技术的关注，逐渐走出低迷状态，有限元技术也不再仅仅停留在高校和企业中。

## 2 有限元理论

### 2.1 Sobolev 空间

假定  $G$  是有界平面区域, 其边界  $\Gamma$  是按段光滑的简单闭曲线,  $\bar{G} = G \cup \Gamma$  是  $G$  的闭包。对于  $\bar{G}$  上的任一函数  $u(x,y)$ , 称集合  $\{(x,y) \mid u(x,y) \neq 0, (x,y) \in \bar{G}\}$  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  $\Gamma$  的某一邻域内恒等于零<sup>[8]</sup>。

用  $C_0^\infty$  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

$$(f, g) = \int_G f g dx dy \quad (1)$$

$$\|f\| = \sqrt{(f, f)} = \left( \int_G |f|^2 dx dy \right)^{\frac{1}{2}} \quad (2)$$

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

$$\int_G g \varphi dx dy = - \int_G f \frac{\partial \varphi}{\partial x} dx dy \quad (3)$$

$$\int_G h \varphi dx dy = - \int_G f \frac{\partial \varphi}{\partial y} dx dy \quad (4)$$

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

$$f_x = \frac{\partial f}{\partial x} = g \quad (5)$$

$$f_y = \frac{\partial f}{\partial y} = h \quad (6)$$

定义

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

其中  $f_x, f_y$  是  $f$  的广义导数。与  $H^1(G)$  引入内积

$$(f, g)_1 = \int_G [fg + f_x g_x + f_y g_y] dx dy \quad (7)$$

和范数

$$\|f\|_1 = \sqrt{(f, f)} = \left( \int_G [|f|^2 + |f_x|^2 + |f_y|^2] dx dy \right)^{\frac{1}{2}} \quad (8)$$

则  $H^1$  是 Hilbert 空间，称之为 Sobolev 空间。

## 2.2 弹性问题

### 2.2.1 边值问题

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

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

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

$$-\text{div} \sigma(u) = f \quad \in \Omega \quad (9)$$

应力张量  $\sigma(u)$  定义为

$$\sigma(u) = 2\mu \epsilon(u) + \lambda \text{tr}(\epsilon(u)) \delta \quad (10)$$

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

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

$$u|_{\Gamma_1} = g \quad (11)$$

并且  $\Gamma_2$  上的牵引力边值条件为

$$(\sigma(u)\nu)|_{\Gamma_2} = t \quad (12)$$

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

$$u|_{\partial\Omega} = 0 \quad (13)$$

设  $\nu = (\nu_1, \nu_2)^t$ ,  $\nu_1, \nu_2 \in C_0^\infty(\Omega)$ , 方程 (9) 两边同乘  $\nu$  并积分得

$$-\int_{\Omega} \operatorname{div} \sigma(u) \nu dx dy = \int_{\Omega} f \nu dx dy \quad (14)$$

参考文献知<sup>[9]</sup>

$$f \operatorname{div} a = \operatorname{div}(fa) - a : \operatorname{grad} f \quad (15)$$

$$\int_{\Omega} \operatorname{div} a dV = \int_{\partial\Omega} a dS \quad (16)$$

将边界条件 (13), 方程 (15), (16) 带入方程 (14) 得

$$\begin{aligned} & -\int_{\Omega} \operatorname{div} \sigma(u) \nu dx dy \\ &= -\int_{\Omega} \operatorname{div}(\sigma(u)\nu) dx dy - \int_{\Omega} \sigma(u) : \operatorname{grad} \nu dx dy \\ &= -\int_{\Gamma} \sigma(u) \nu dx dy + \int_{\Omega} \sigma(u) : \operatorname{grad} \nu dx dy \\ &= \int_{\Omega} \sigma(u) : \operatorname{grad} \nu dx dy \\ &= \int_{\Omega} 2\mu \epsilon(u) : \operatorname{grad} \nu + \lambda \operatorname{div} u \operatorname{div} \nu dx dy \\ &= \mu \int_{\Omega} \operatorname{grad} u : \operatorname{grad} \nu dx dy + (\mu + \lambda) \int_{\Omega} \operatorname{div} u \operatorname{div} \nu dx dy \end{aligned}$$

所以

$$\mu \int_{\Omega} \operatorname{grad} u : \operatorname{grad} \nu dx dy + (\mu + \lambda) \int_{\Omega} \operatorname{div} u \operatorname{div} \nu dx dy = \int_{\Omega} f \nu dx dy \quad (17)$$

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

$$a(u, \nu) = \int_{\Omega} f \cdot \nu dx dy \quad \forall \nu \in V \quad (18)$$



其中

$$a(u, \nu) := \mu \int_{\Omega} \text{grad} u : \text{grad} \nu dx dy + (\mu + \lambda) \int_{\Omega} \text{div} u \text{div} \nu dx dy \quad (19)$$

$$V := \{\nu \in H^1(\Omega) \mid \nu|_{\Gamma} = 0\}$$

**Lax-Milgram 定理<sup>[10]</sup>**: 设  $H$  是 Hilbert 空间,  $a(\bullet, \bullet)$  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

$$a(u, \nu) = (f, \nu), \quad \forall \nu \in H \quad (20)$$

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  $\Omega_1, \Omega_2$  是  $\Omega$  的两个子集, 使得  $\Omega_1 \cup \Omega_2 = \Omega$  并且  $\Omega_1 \cap \Omega_2 = \emptyset$ , 考虑以下边值问题

$$\begin{aligned} -\text{div} \sigma(u) &= f \quad \text{in } \Omega \\ u|_{\partial\Omega} &= 0 \end{aligned} \quad (21)$$

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

$$\begin{aligned} a(u, \nu) &= \mu_1 \int_{\Omega_1} \text{grad} u : \text{grad} \nu dx dy + (\mu_1 + \lambda_1) \int_{\Omega_1} \text{div} u \text{div} \nu dx dy \\ &\quad + \mu_2 \int_{\Omega_2} \text{grad} u : \text{grad} \nu dx dy + (\mu_2 + \lambda_2) \int_{\Omega_2} \text{div} u \text{div} \nu dx dy \end{aligned} \quad (22)$$

## 2.3 离散

### 2.3.1 Galerkin 法

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

$$p_0, p_1, \dots, p_n$$

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  $\Omega$  上属于函数空间  $H^1$ 。则  $U_h \times U_h$  为试探函数空间。

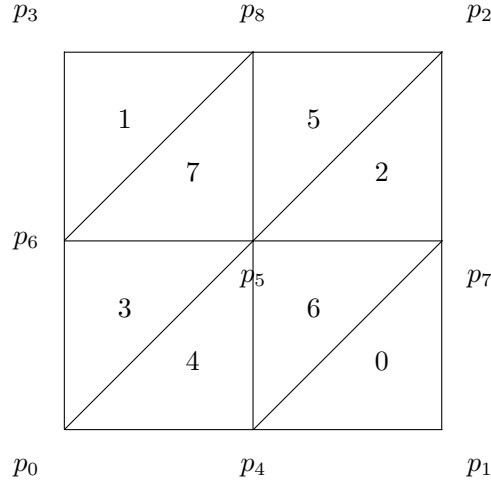


图 1

设

$$U_h = \text{span}(\varphi_0, \varphi_1, \dots, \varphi_n)$$

$$\phi_{2i} = (\varphi_i, 0), \quad \phi_{2i+1} = (0, \varphi_i), \quad i = 0, \dots, n$$

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

$$u_h = \sum_{i=0}^{2n+1} c_i \phi_i \quad (23)$$

将式 (23) 带入方程 (18) 中得到 Galerkin 方程

$$\sum_{i=0}^{2n+1} a(\phi_i, \phi_j) c_i = (f, \phi_j), \quad j = 0, 1, \dots, 2n+1 \quad (24)$$

令

$$A = (a(\phi_j, \phi_i))_{0 \leq i, j \leq 2n+1}$$

$$F = ((f, \phi_i))_{0 \leq i \leq 2n+1}$$

$$c = (c_i)_{0 \leq i \leq 2n+1}$$

则 Galerkin 方程 (24) 的矩阵形式为

$$Ac = F \quad (25)$$

模型为齐次边界条件, 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  $S_0, S_1, S_2$  [8]

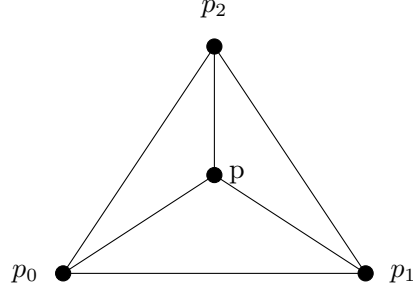


图 2

显然  $S_0 + S_1 + S_2 = S$ , 令

$$L_0 = \frac{S_0}{S}, \quad L_1 = \frac{S_1}{S}, \quad L_2 = \frac{S_2}{S} \quad (26)$$

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

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

$$\begin{cases} \varphi_0 = \frac{1}{2S}[(x_2y_3 - x_3y_2) + (y_2 - y_3)x + (x_3 - x_2)y] \\ \varphi_1 = \frac{1}{2S}[(x_3y_0 - x_0y_3) + (y_3 - y_0)x + (x_0 - x_3)y] \\ \varphi_2 = \frac{1}{2S}[(x_0y_1 - x_1y_0) + (y_0 - y_1)x + (x_1 - x_0)y] \end{cases} \quad (27)$$

### 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  $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ 。

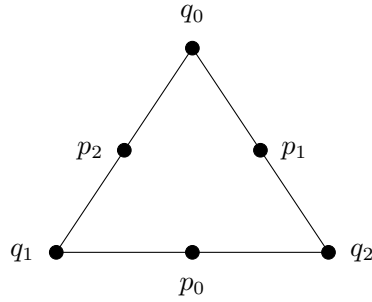


图 3

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

$$\varphi_i = a_i x + b_i y + c_i, \quad i = 0, 1, 2 \quad (28)$$

且其在  $p_0, p_1, p_2$  点上满足以下关系式

$$\varphi_i(p_j) = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}, \quad i, j = 0, 1, 2 \quad (29)$$

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 (29) 的矩阵形式为

$$Ac_i = e_i, \quad i = 0, 1, 2 \quad (30)$$

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

$$\varphi_i = A^{-1}e_i f, \quad i = 0, 1, 2 \quad (31)$$

## 2.4 误差估计

假设  $\Omega$  是一个凸多边形区域, 并且  $\Gamma_1$  or  $\Gamma_2$  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  $\Omega$  三角划分的一个非退化族。对于纯位移问题 ( $\Gamma_2 = \emptyset$ ), 我们使用有限元空间

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

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

$$a(u_h, \nu) = \int_{\Omega} f \cdot \nu dx \quad \forall \nu \in V_h. \quad (32)$$

则存在一个正常数  $C_{(\mu, \lambda)}$  使得<sup>[10]</sup>

$$\|u - u_h\|_{H^1(\Omega)} \leq C_{(\mu, \lambda)} h \|u\|_{H^2(\Omega)}. \quad (33)$$

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

$$a(u_h, \nu) = \int_{\Omega} f \cdot \nu dx + \int_{\Gamma_2} t \cdot \nu ds \quad \forall \nu \in V_h. \quad (34)$$

则存在一个正常数  $C_{(\mu, \lambda)}$  使得<sup>[10]</sup>

$$\|u - u_h\|_{H^1(\Omega)} \leq C_{(\mu, \lambda)} h \|u\|_{H^2(\Omega)}. \quad (35)$$

## 2.5 闭锁现象

对于固定的  $\mu$  和  $\lambda$ , 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  $\lambda$  趋向于  $\infty$  而变差。这就是所谓的锁定现象<sup>[10]</sup>。

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

$$\begin{aligned} \operatorname{div}\{2\epsilon(u^\lambda) + \lambda \operatorname{tr}(\epsilon(u^\lambda))\delta\} &= f \quad \text{in } \Omega \\ u^\lambda|_{\partial\Omega} &= 0. \end{aligned} \quad (36)$$

注意给定的  $f$ ，当  $\lambda \rightarrow \infty$ ， $\|div u^\lambda\|_{H^1(\Omega)} \rightarrow 0$ 。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  $\lambda$  的依赖，将应力张量  $\sigma_\lambda(\nu)$  和变分形式  $a_\lambda(\nu, \omega)$  表示为

$$\begin{aligned}\sigma_\lambda(\nu) &= 2\epsilon(\nu) + \lambda tr(\epsilon(\nu))\delta \\ a_\lambda(\nu, \omega) &= \int_{\Omega} \{2\epsilon(\nu) : \epsilon(\omega) + \lambda div \nu div \omega\} dx.\end{aligned}\tag{37}$$

令  $T^h$  为  $\Omega$  (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

$$\begin{aligned}a_\lambda(u_h^\lambda, \nu) &= \int_{\Omega} [-div \sigma_\lambda(u)] \cdot \nu dx \quad \forall \nu \in V_h. \\ V_h &:= \{\nu \in H^1(\Omega) : \nu|_T, \forall T \in T^h\}\end{aligned}\tag{38}$$

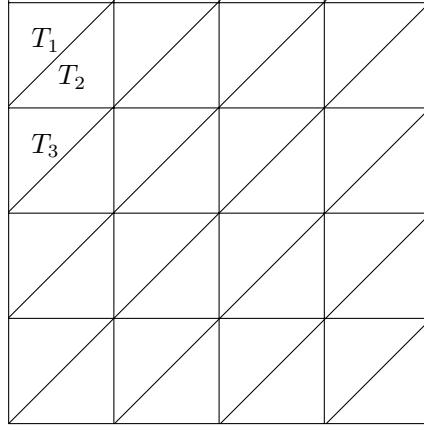


图 4

定义  $L_{\lambda, h}$  为

$$L_{\lambda, h} := \sup \left\{ \frac{|u - u_h^\lambda|_{H^1(\Omega)}}{\|div \sigma_\lambda(u)\|_{L^2(\Omega)}} : 0 \neq u \in H^2(\Omega) \cap H^1(\Omega) \right\}.\tag{39}$$

则存在一个与  $h$  无关的正常数  $C$  使得<sup>[10]</sup>

$$\lim_{\lambda \rightarrow \infty} \inf L_{\lambda, h} \geq C.\tag{40}$$

式 (40) 意味着：无论  $h$  取多小，只要  $\lambda$  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  $\lambda$  变大而变坏。

### 3 算例

#### 3.1 弹性问题

##### 3.1.1 算例一

考察以下边值问题

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

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

$$\begin{aligned} u_1 &= (x-1)(y-1)y \sin(x) \\ u_2 &= (x-1)(y-1)x \sin(y) \end{aligned}$$

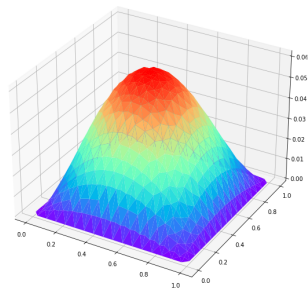
通过数值实验得到, 当  $\mu = \lambda = 1, 1E4, 1E8$  时的误差及误差阶如下表

表 1:  $|u - u_h|_{H^1(\Omega)}$

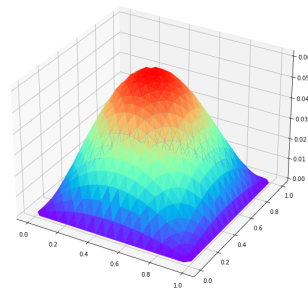
$\lambda \backslash h$	1.0	0.5	0.25	0.125	0.0625	误差阶
1	4.102804E-1	1.424371E-1	8.382384E-2	4.737414E-2	2.528276E-2	0.8305559
1E4	4.102804E-1	1.424371E-1	8.382384E-2	4.737414E-2	2.528276E-2	0.8305559
1E8	4.102804E-1	1.424371E-1	8.382384E-2	4.737414E-2	2.528276E-2	0.8305559

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

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

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

$$\begin{aligned} u_1 &= x^2 \sin(x-1) y^2 \sin(y-1) \\ u_2 &= x^2 \sin(x-1) y^2 \sin(y-1) \end{aligned}$$

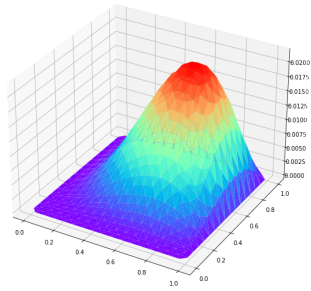
通过数值实验得到, 当  $\mu = \lambda = 1, 1E4, 1E8$  时的误差及误差阶如下表

表 2:  $|u - u_h|_{H^1(\Omega)}$

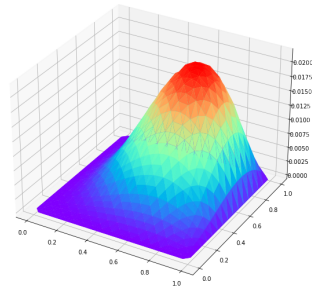
$\lambda \backslash h$	1.0	0.5	0.25	0.125	0.0625	误差阶
1	1.381670E-1	1.038442E-1	4.190134E-2	2.135923E-2	1.113314E-2	1.063660
1E4	1.381670E-1	1.038442E-1	4.190134E-2	2.135923E-2	1.113314E-2	1.063660
1E8	1.381670E-1	1.038442E-1	4.190134E-2	2.135923E-2	1.113314E-2	1.063660

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



### 3.1.3 算例三

考察以下边值问题

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

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

$$\begin{aligned} u_1 &= -(x-1)(e^x-1)(y-1)(e^y-1) \\ u_2 &= -(x-1)(e^x-1)(y-1)(e^y-1) \end{aligned}$$

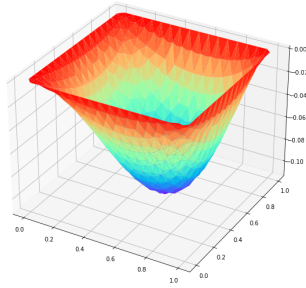
通过数值实验得到, 当  $\mu = \lambda = 1, 1E4, 1E8$  时的误差及误差阶如下表

表 3:  $|u - u_h|_{H^1(\Omega)}$

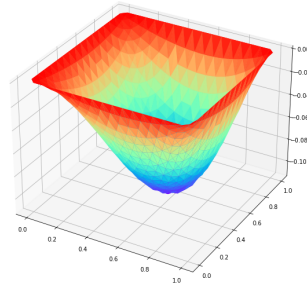
$\lambda \backslash h$	1.0	0.5	0.25	0.125	0.0625	误差阶
1	8.170963E-1	3.487612E-1	1.669124E-1	8.921846E-2	4.718501E-2	0.95611946
1E4	8.170963E-1	3.487612E-1	1.669124E-1	8.921846E-2	4.718501E-2	0.95611946
1E8	8.170963E-1	3.487612E-1	1.669124E-1	8.921846E-2	4.718501E-2	0.95611946

数值解和精确解图像如下

图 7



(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

考察以下边值问题

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

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,  $\mu = \lambda = \lambda_4$

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

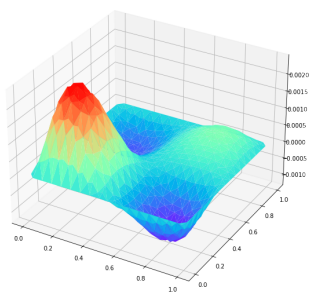
令  $\lambda = [\lambda_1, \lambda_2, \lambda_3, \lambda_4]$ ,  $\mu = [\mu_1, \mu_2, \mu_3, \mu_4]$

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

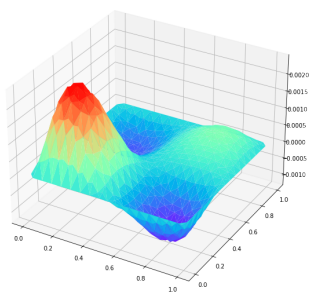
表 4:  $|u - u_h|_{H^1(\Omega)}$

h	0.5	0.25	0.125	0.0625	误差阶
$ u - u_h _{H^1(\Omega)}$	1.984010E-1	8.447614E-1	4.446847E-1	2.373949E-2	0.9156274

图 8



(a) 数值解图像



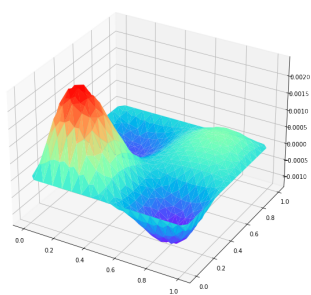
(b) 精确解图像

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

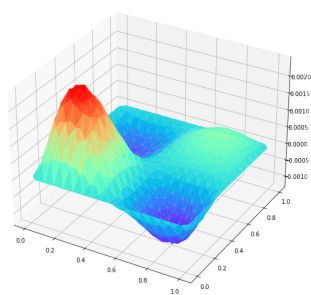
表 5:  $|u - u_h|_{H^1(\Omega)}$

h	0.5	0.25	0.125	0.0625	误差阶
$ u - u_h _{H^1(\Omega)}$	1.984010E-1	8.447614E-1	4.446847E-1	2.373949E-2	0.9156274

图 9



(a) 数值解图像



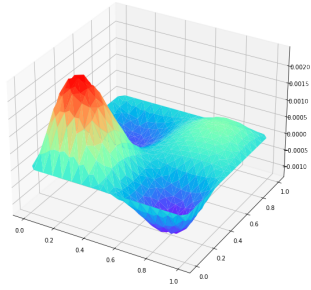
(b) 精确解图像

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

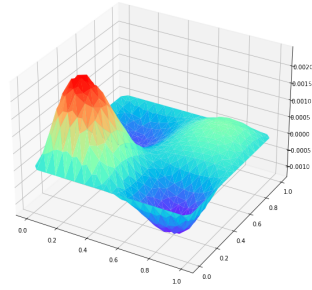
表 6:  $|u - u_h|_{H^1(\Omega)}$

h	0.5	0.25	0.125	0.0625	误差阶
$ u - u_h _{H^1(\Omega)}$	1.984010E-1	8.447614E-1	4.446847E-1	2.373949E-2	0.9156274

图 10



(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] 邵文婷. 求解一类交界面问题的模态基函数谱元法数值实验. 上海第二工业大学学报, 34(4):283–290, 2017.
- [2] 王兆清, 徐子康, and 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.
- [3] 陈绍春 and 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.
- [4] YT Peet and PF Fischer. Legendre spectral element method with nearly incompressible materials. *European Journal of Mechanics-A/Solids*, 44:91–103, 2014.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 李荣华. 偏微分方程数值解, 2007.
- [9] 陈纪修, 於崇华, and 金路. 数学分析. 高等教育出版社, 2004.
- [10] 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.

## A 附录数值程序

Listing 1: elasticityCR.py

```
1  ## PDE1
2  ## 模型与剖分
3
4  import math
5  import numpy as np
6
7  from tool import PDE, getIsBdNode, uniform_refine, get_cr_node_cell
8  from tool import get_A1_A2_F, my_solve, H1Error, print_error, print_P
9  from tool import get_stiff_and_div_matrix, get_bb, drawer_uh_u
10
11  n = 4 #剖分次数
12  n = n + 1
13  Lam = [1,1e1,1e2,1e3,1e4,1e5]
14  Mu = Lam
15
16  H = np.zeros(n) #步长
17  P = np.zeros(len(Lam)) #误差阶
18  E = np.zeros((len(Lam),n), dtype=np.float64) #每个lambda(行)对应的误差(列)
19
20  for i in range(len(Lam)):
21      pde = PDE(Mu[i], Lam[i])# / 2 + 10, Lam[i])
22      node = np.array([
23          (0,0),
24          (1,0),
25          (1,1),
26          (0,1)], dtype=np.float64)
27      cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)
28      for j in range(n):
29          NC = cell.shape[0]
30          nn = math.log(NC/2, 4)
31          NN = int(3 * 2 * 4**nn - (3 * 2 * 4**nn - 4 * 2**nn) / 2)
32          cm = np.ones(NC, dtype=np.float64) / NC
33
34          cr_node, cr_cell = get_cr_node_cell(node, cell)
35
36          # 单元刚度矩阵和单元载荷向量
37          A1, A2 = get_stiff_and_div_matrix(cr_node, cr_cell, cm)
38          bb = get_bb(pde, node, cell, cm)
```



```

39
40     A1, A2, F = get_A1_A2_F(A1, A2, bb, cr_node, cr_cell)
41     A = pde.mu * A1 + (pde.lam + pde.mu) * A2
42
43     uh = my_solve(A, F, cr_node, getIsBdNode)
44     u = pde.solution(cr_node)
45     H[j] = np.sqrt(2 * cm[0])
46     # 计算误差
47     E[i][j] = H1Error(u, uh)
48     if j < n-1 :
49         node, cell = uniform_refine(node, cell)
50     uh_dir = "../image/tmp/elasticity_uh_u/PDE1/uh_lam={}.png".
41         format(Lam[i])
51     u_dir = "../image/tmp/elasticity_uh_u/PDE1/u_lam={}.png".
41         format(Lam[i])
52     drawer_uh_u(cr_node, uh, u, uh_dir, u_dir)
53
54 if n-1 > 1:
55     for i in range( len(Lam)):
56         f = np.polyfit(np.log(H[1:]), np.log(E[i][1:]), 1)
57         P[i] = f[0]
58
59 print_error(Lam, H, E)
60 if n-1 > 1:
61     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/elasticity_uh_u/PDE1/uh_lam={}.png".format(
41     Lam[i])
11 # u_dir = "../image/tmp/elasticity_uh_u/PDE1/u_lam={}.png".format(Lam[
41     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
41             + mu*frac_u1_y)
42         val[..., 1] = -((2*mu+lam) * frac_u2_y + (mu+lam) * frac_u1_x_y
43             + mu*frac_u2_x)
44
45         return val
46
47     def solution(self, p):
48         x = p[..., 0]
49         y = p[..., 1]
50
51         val = np.zeros(p.shape, dtype=np.float64)
52
53         val[..., 0] = y * (x - 1) * (y - 1) * np.sin(x)
54         val[..., 1] = x * (x - 1) * (y - 1) * np.sin(y)
55
56         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
96             + mu*frac_u1_y)
97         val[..., 1] = -((2*mu+lam) * frac_u2_y + (mu+lam) * frac_u1_x_y
98             + mu*frac_u2_x)
99
100         return val
101
102     def solution(self, p):
103         x = p[..., 0]
104         y = p[..., 1]
105
106         sin = np.sin
107         val = np.zeros(p.shape, dtype=np.float64)
108
109         ux = x**2 * sin(x-1)
110         uy = y**2 * sin(y-1)
111
112         val[..., 0] = ux * uy
113         val[..., 1] = ux * uy
114
115         return val
116
117     # PDE3
118     # u1 = u2 = -(x-1) * (e^x-1) * (y-1) * (e^y-1)
119     # uh_dir = "../image/tmp/elasticity_uh_u/PDE3/uh_lam={}.png".format(
120     #     Lam[i])
121     # u_dir = "../image/tmp/elasticity_uh_u/PDE3/u_lam={}.png".format(Lam[
122     #     i])
123
124     class PDE3():
125         def __init__(self, mu=1, lam=1):
126             self.mu = mu
127             self.lam = lam
128             self.node = np.array([
129                 (0,0),
130                 (1,0),
131                 (1,1),
132                 (0,1)], dtype=np.float64)
133             self.cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)
134
135         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
156                   + mu*frac_u1_y)
157     val[..., 1] = -((2*mu+lam) * frac_u2_y + (mu+lam) * frac_u1_x_y
158                   + mu*frac_u2_x)
159
160     return -val
161
162 def solution(self, p):
163     x = p[..., 0]
164     y = p[..., 1]
165
166     val = np.zeros(p.shape, dtype=np.float64)
167
168     ux = (x-1) * (np.exp(x) - 1)
169     uy = (y-1) * (np.exp(y) - 1)
170
171     val[..., 0] = ux * uy
172     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(
177         Lam[i])
178     # u_dir = "../image/tmp/interface_uh_u/PDE1/u_lam={}.png".format(Lam[
179         i])
180
181     class interfaceData():
182         def __init__(self, mu=np.array([1,2,3,4]), lam=np.array([1,2,3,4])):
183             self.mu = mu
184             self.lam = lam
185             self.node = np.array([(0,0),
186                 (0.5,0),
187                 (1,0),
188                 (0,0.5),
189                 (0.5,0.5),
190                 (1,0.5),
191                 (0,1),
192                 (0.5,1),
193                 (1,1)], dtype=np.float64)
194             self.cell = np.array([[0,1,4],
195                 [0,4,3],
196                 [1,2,5],
197                 [1,5,4],
198                 [3,4,7],
199                 [3,7,6],
200                 [4,5,8],
201                 [4,8,7]], dtype=np.int64)
202
203         def source(self, p):
204             x = p[... , 0]
205             y = p[... , 1]
206
207             mu = 1
208             lam = 1
209             val = np.zeros(p.shape, dtype=np.float64)
210
211             frac_u1_x = (6*x-3) * (y**3-(3/2)*y**2+(1/2)*y)
212             frac_u1_y = (x**3-(3/2)*x**2+(1/2)*x) * (6*y-3)
213             frac_u1_x_y = (3*x**2-3*x+1/2) * (3*y**2-3*y+1/2)

```

```

211         frac_u2_x    = frac_u1_x
212         frac_u2_y    = frac_u1_y
213         frac_u2_x_y  = frac_u1_x_y
214
215         val[..., 0] = -((2*mu+lam) * frac_u1_x + (mu+lam) * frac_u2_x_y
216             + mu*frac_u1_y)
217         val[..., 1] = -((2*mu+lam) * frac_u2_y + (mu+lam) * frac_u1_x_y
218             + mu*frac_u2_x)
219
220     return val
221
222 def solution(self, p, cr_node):
223     x = p[..., 0]
224     y = p[..., 1]
225
226     val = np.zeros(p.shape, dtype=np.float64)
227
228     val[..., 0] = x * (x - 0.5) * (x - 1) * y * (y - 0.5) * (y - 1)
229     val[..., 1] = x * (x - 0.5) * (x - 1) * y * (y - 0.5) * (y - 1)
230
231     crCell = getWhichCell(cr_node)
232     for i in range(4):
233         val[crCell[i], :] /= self.lam[i]
234
235     return val
236
237 def H1Error(u, uh):
238     tmp = u - uh
239     e = np.einsum("ni, ni -> n", tmp, tmp)
240     sum = e.sum()
241     return np.sqrt(sum)
242
243 def print_error(Lam, H, E):
244     for i in range(len(Lam)):
245         print("-----Lam= {}-----".
246             format(Lam[i]))
247         n = H.shape[0]
248         print()
249         for j in range(n):
250             print("h= ", H[j])
251             print("e=", E[i][j])
252             print()
253         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])).
298                         all(axis=1))[0]
299     if len(in_index) == 0:
300         raise Exception("数组cr_cell已满")
301     new_cr_cell[in_index[0]] = a_cell
302     return new_cr_cell
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
473     cell_x_y[:, :, 1] = 2 * cell_x_y[:, :, 1] + 1  #[NC,3] 三个节点y方向上
474     cell_x_y = cell_x_y.reshape(NC, 6)
475     I = np.broadcast_to(cell_x_y[:, :, None], shape=A1.shape)
476     J = np.broadcast_to(cell_x_y[:, None, :], shape=A2.shape)
477
478     A1 = csr_matrix((A1.flat, (I.flat, J.flat)), shape=(2 * NN, 2 * NN))
479     A2 = csr_matrix((A2.flat, (I.flat, J.flat)), shape=(2 * NN, 2 * NN))
480     F = np.zeros(2 * NN)
481     np.add.at(F, cell_x_y, bb)
482     return A1, A2, F
483
484 def my_solve(A, F, cr_node, getIsBdNode):
485     NN = cr_node.shape[0]
486     isBdNode = getIsBdNode(cr_node)
487     isInterNode = ~isBdNode
488     #print("isInterNode= ", isInterNode)
489     isInterNodeA = np.broadcast_to(isInterNode[:, None], shape=(NN, 2))
490     isInterNodeA = isInterNodeA.reshape(2 * NN)
491     #print("isInterNodeA= ", isInterNodeA)
492
493     uh = np.zeros((2 * NN), dtype=np.float64)
494     uh[isInterNodeA] = spsolve(A[:, isInterNodeA][isInterNodeA], F[
        isInterNodeA])

```

```

495     #uh = spsolve(A, F)
496     #print("uh= ", uh)
497     uh = uh.reshape(NN, 2)
498     return uh
499
500 ## [4,NN] 返回各点属于哪个区间
501 ## \Omega_1 [0,0.5] \times [0,0.5)
502 ## \Omega_2 (0.5, 1] \times [0,0.5]
503 ## \Omega_3 [0,0.5] \times [0.5,1]
504 ## \Omega_4 [0.5,1] \times (0.5,1]
505 def getWhichCell(node):
506     isWhichCellNode = np.zeros((4,node.shape[0]), dtype= bool)
507     for i in range(node.shape[0]):
508         a = node[i, 0] - 0
509         b = node[i, 1] - 0
510         if a <= 0.5 and b < 0.5:
511             isWhichCellNode[0,i] = True
512         if a > 0.5 and b <= 0.5:
513             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
561         .shape[1], 2))
562     phiCell = phiCell.reshape(4, 2 * whichCell.shape[1])
563     return phiCell
564
565 # uh_dir = "../../image/tmp/elasticity_uh_u/uh_lam={}.png".format(Lam[i])
566 # u_dir = "../../image/tmp/elasticity_uh_u/u_lam={}.png".format(Lam[i])
567 def drawer_uh_u(cr_node, uh, u, uh_dir, u_dir):
568     import matplotlib.pyplot as plt
569     fig = plt.figure(figsize=(10,10))
570     ax = fig.add_subplot(111, projection='3d')
571
572     x = cr_node[:,0]
573     y = cr_node[:,1]
574
575     ax.plot_trisurf(x, y, uh[:,0], cmap='rainbow')
576     plt.savefig(fname=uh_dir)

```

```
577     ax.plot_trisurf(x, y, u[:,0], cmap='rainbow')
578     plt.savefig(fname=u_dir)
579     plt.close(fig)
```