带间断系数的弹性问题

2023 年 5 月 7 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弹性界面问题进行求解

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

Abstractname

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弹性界面问题进行求解
关键字：**平面弹性问题**; **间断系数**； **非协调有限元**; **locking-free**

目录

1 引言

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 [1]、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者大量的界面问题。

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 [2]。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[3]。

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[4]、混合有限元法[5]、增强有限元法[6] 和不连续 Galerkin 法[7] 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[2]。

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，伴随着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 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[8]。

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

$$(f,g) = \int_G fg dxdy$$
$$||f|| = \sqrt{(f,f)} = (\int_G |f|^2 dxdy)^{\frac{1}{2}}$$

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

$$\int_G g\varphi dxdy = -\int_g f\frac{\partial\varphi}{\partial x} dxdy$$
$$\int_G h\varphi dxdy = -\int_G f\frac{\partial\varphi}{\partial y} dxdy$$

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

$$f_x = \frac{\partial f}{\partial x} = g$$
$$f_y = \frac{\partial f}{\partial y} = y$$

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

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

和范数

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

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

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

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

$$-div\sigma(u)=f \quad \in\Omega \tag{1}$$

应力张量 $\sigma(u)$ 定义为

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

其中 $\Omega\in\mathbb{R}^2$，正常数 λ，μ 为 Lam$\acute{e}$ 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

$$u|_{\Gamma_1}=g \tag{3}$$

并且 Γ_2 上的牵引力边值条件为

$$(\sigma(u)\nu)|_{\Gamma_2}=t \tag{4}$$

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

$$u|_{\partial\Omega}=0 \tag{5}$$

设 $\nu=(\nu_1,\nu_2)^t,\quad \nu_1,\nu_2\in C_0^\infty(\Omega)$，方程 (1) 两边同乘 ν 并积分得

$$-\int_\Omega div\sigma(u)\nu dxdy=\int_\Omega f\nu dxdy \tag{6}$$

参考文献知[9]

$$fdiva=div(fa)-a:gradf \tag{7}$$

$$\int_\Omega divadV=\int_{\partial\Omega}adS \tag{8}$$

将边界条件 (5)，方程 (7)，(8) 带入方程 (6) 得

$-\int_\Omega div\sigma(u)\nu dxdy$

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

所以

$$\mu\int_\Omega gradu:grad\nu dxdy+(\mu+\lambda)\int_\Omega divudiv\nu dxdy=\int_\Omega f\nu dxdy$$

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

$$a(u,\nu)=\int_\Omega f\cdot\nu dxdy\quad \forall\nu\in V \tag{9}$$

其中

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

Lax-Milgram 定理[10]：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

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

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

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

当 Lam$\acute{e}$ 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 $\lambda=\lambda_2$，$\mu=\mu_2$，通过计算得到与此问题对应的双线性形式为

$$
\begin{aligned}
a(u,\nu) = \mu_1\int_{\Omega_1} gradu : grad\nu dxdy + (\mu_1+\lambda_1)\int_{\Omega_1} divudiv\nu dxdy \\
+ \mu_2\int_{\Omega_2} gradu : grad\nu dxdy + (\mu_2+\lambda_2)\int_{\Omega_2} divudiv\nu dxdy
\end{aligned} \tag{10}
$$

2.3 离散

2.3.1 Galerkin 法

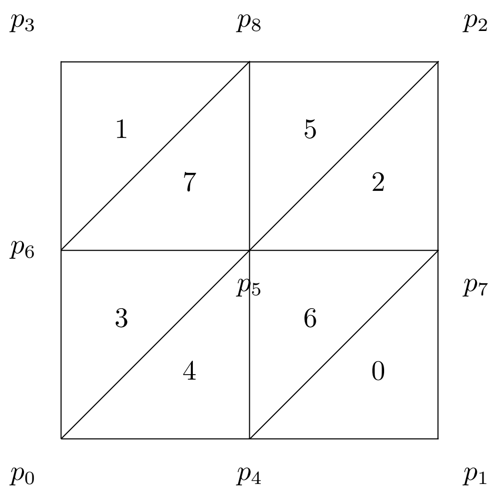


图 1

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 $U_h \times U_h$ 为试图函数空间。

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

$$u_h = \sum_{i=0}^{2n+1} c_i \phi_i \tag{11}$$

将式 (11) 带入方程 (9) 中得到 Galerkin 方程

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

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 方程的矩阵形式为

$$Ac = F$$

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]

显然 $S_0 + S_1 + S_2 = S$，令

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

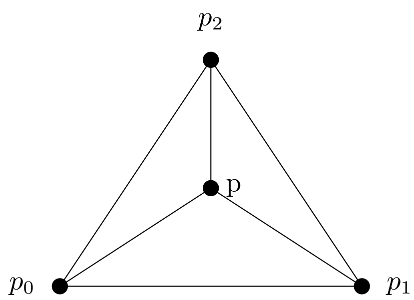


图 2

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

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

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

2.3.3 C-R 元

如图 3，设三角形 $\triangle(q_0, q_1, q_2)$ 是以 $q0, q1, q2$ 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 $(x_0.y_0), (x_1, y_1), (x_2, y_2)$。

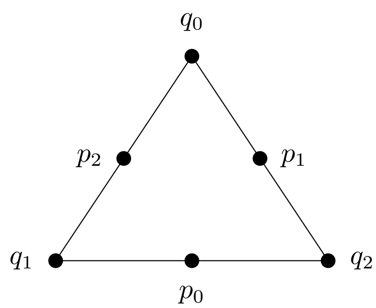


图 3

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

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

且其在 p_0，p_1，p_2 点上满足以下关系式

$$\varphi_i(p_j) = \begin{cases} 1, & i=j \\ 0, & i\neq j \end{cases}, \quad i,j=0,1,2 \tag{12}$$

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 (12) 的矩阵形式为

$$Ac_i = e_i, \quad i=0,1,2$$

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

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

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 ($\Gamma_2=\emptyset$), 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

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

Theorem[10]**.** 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

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

则存在一个正常数 $C_{(\mu,\lambda)}$ 使得

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

Theorem[10]**.** 令 $u \in H^2(\Omega)$ 满足纯牵引力问题。令 $u_h \in V_h$ 满足

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

则存在一个正常数 $C_{(\mu,\lambda)}$ 使得[10]

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

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[10]。

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

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

注意给定的 f ，当 $\lambda \to \infty$，$\|divu^\lambda\|_{H^1(\Omega)} \to 0$. 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

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

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

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

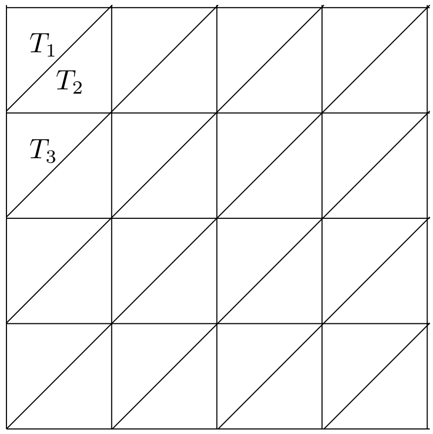


图 4

定义 $L_{\lambda,h}$ 为

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

则存在一个与 h 无关的正常数 C 使得[10]

$$\lim_{\lambda\to\infty} \inf L_{\lambda,h} \geq C. \tag{13}$$

式 (13) 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。

3 算例

3.1 弹性问题

3.1.1 算例一

考察以下边值问题

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

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

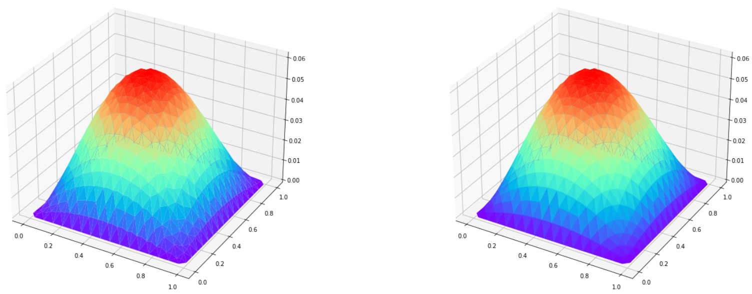
$$u_1 = (x-1)(y-1)ysin(x)$$
$$u_2 = (x-1)(y-1)xsin(y)$$

表 1: $|u-u_h|_{H_1}$

h / λ	1.0	0.5	0.25	0.125	0.0625	误差阶
1	4.102804E-1	1.424371E-1	8.382384E-2	4.737414-2	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

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
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，$\Omega = [0,1] \times [0,1]$

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

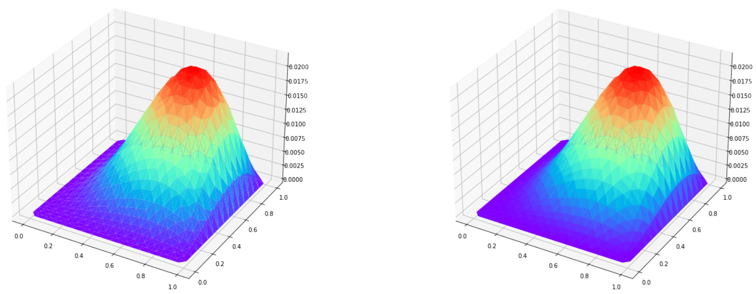
通过数值实验得到，当 $\mu = \lambda = 1, 1E4, 1E8$ 时的误差及误差阶如下表

表 2: $|u - u_h|_{H_1}$

h \ λ	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

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

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

$$u_1 = -(x-1)(e^x-1)(y-1)(e^y-1)$$
$$u_2 = -(x-1)(e^x-1)(y-1)(e^y-1)$$

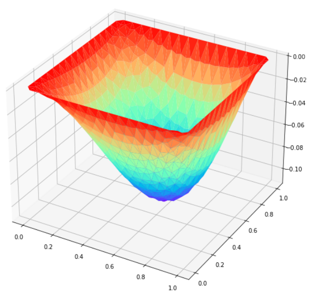
通过数值实验得到，当 $\mu = \lambda = 1, 1E4, 1E8$ 时的误差及误差阶如下表

表 3: $|u - u_h|_{H_1}$

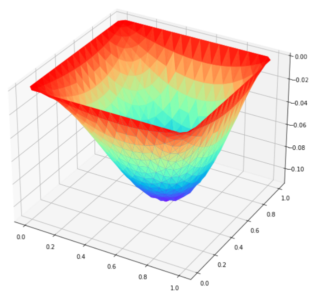
h / λ	1.0	0.5	0.25	0.125	0.0625	误差阶
1	8.170963E-1	3.487612E-1	1.669124E-1	8.921846E-2	4.718501E-2	0.95611946
1E4	8.170963E-1	3.487612E-1	1.669124E-1	8.921846E-2	4.718501E-2	0.95611946
1E8	8.170963E-1	3.487612E-1	1.669124E-1	8.921846E-2	4.718501E-2	0.95611946

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

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

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，$\mu=\lambda=\lambda_4$

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

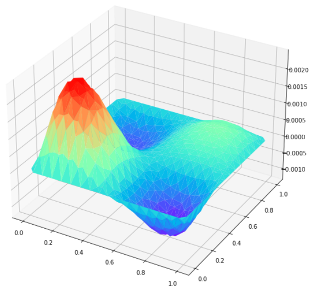
令 $\lambda=[\lambda_1,\lambda_2,\lambda_3,\lambda_4]$，$\mu=[\mu_1,\mu_2,\mu_3,\mu_4]$

1. 当 $\lambda=\mu=[1,2,3,4]$ 时误差如下

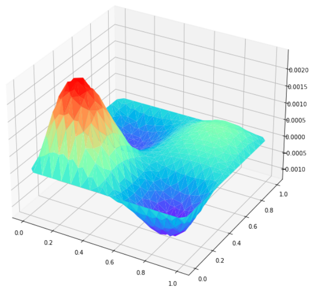
表 4: $|u-u_h|_{H_1}$

h	0.5	0.25	0.125	0.0625	误差阶
λ	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: $|u - u_h|_{H_1}$

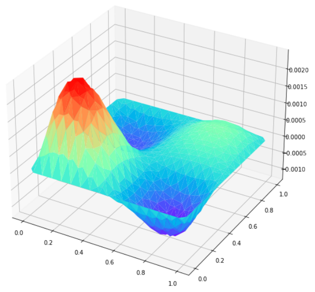
h	0.5	0.25	0.125	0.0625	误差阶
λ	1.984010E-1	8.447614E-1	4.446847E-1	2.373949E-2	0.9156274

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

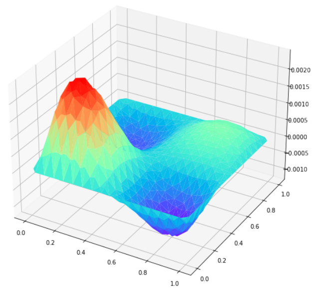
表 6: $|u - u_h|_{H_1}$

h	0.5	0.25	0.125	0.0625	误差阶
λ	1.984010E-1	8.447614E-1	4.446847E-1	2.373949E-2	0.9156274

图 9



(a) 数值解图像

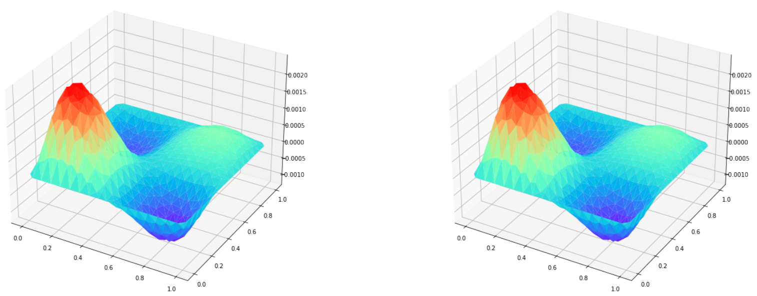


(b) 精确解图像

4 总结

弹性问题在使用非协调元求解时可以解除闭锁现象，本文对带间断系数的平面弹性问题使用 C-R 有限元求解，考察是否仍然可以解除闭锁现象。通过编写程序得到的近似解收敛阶，与理论上的收敛阶一致，可以得知当 Lamé 常数间断且相等的情况下使用 C-R 元仍然可以解除闭锁现象。

图 10



(a) 数值解图像　　(b) 精确解图像

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

Listing 1: **elasticityCR.py**

```
## PDE1
## 模型与剖分

import math
import numpy as np

from tool import PDE, 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

n   = 4  #剖分次数
n   = n + 1
Lam = [1,1e1,1e2,1e3,1e4,1e5]
Mu  = Lam

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

for i in  range( len(Lam)):
    pde = PDE(Mu[i], Lam[i])# / 2 + 10, Lam[i])
    node = np.array([
            (0,0),
            (1,0),
            (1,1),
            (0,1)], dtype=np.float64)
    cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)
    for j in  range(n):
        NC = cell.shape[0]
        nn  = math.log(NC/2, 4)
        NN =  int(3 * 2 * 4**nn - (3 * 2 * 4**nn - 4 * 2**nn) / 2)
        cm = np.ones(NC, dtype=np.float64) / NC

        cr_node, cr_cell = get_cr_node_cell(node,cell)

        # 单元刚度矩阵和单元载荷向量
        A1, A2 = get_stiff_and_div_matrix(cr_node, cr_cell, cm)
        bb = get_bb(pde, node, cell, cm)
```

```

        A1, A2, F = get_A1_A2_F(A1, A2, bb, cr_node, cr_cell)
        A = pde.mu * A1 + (pde.lam + pde.mu) * A2

        uh = my_solve(A, F, cr_node, getIsBdNode)
        u = pde.solution(cr_node)
        H[j] = np.sqrt(2 * cm[0])
        # 计算误差
        E[i][j] = H1Error(u, uh)
        if j < n-1 :
            node, cell = uniform_refine(node, cell)
    uh_dir = "../../image/tmp/elaticity_uh_u/PDE1/uh_lam={}.png".format(Lam[i])
    u_dir = "../../image/tmp/elaticity_uh_u/PDE1/u_lam={}.png".format(Lam[i])
    drawer_uh_u(cr_node, uh, u, uh_dir, u_dir)

if n-1 > 1:
    for i in  range( len(Lam)):
        f = np.polyfit(np.log(H[1:]), np.log(E[i][1:]) ,1)
        P[i] = f[0]

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

Listing 2: **tool.py**

```
import math
import numpy as np
from numpy.linalg import solve
from scipy.sparse import csr_matrix
from scipy.sparse.linalg import spsolve

# PDE1
# u1 = y(x-1)(y-1)sin(x)
# u2 = x(x-1)(y-1)sin(y)
# uh_dir = "../../image/tmp/elaticity_uh_u/PDE1/uh_lam={}.png".format(Lam[i])
# u_dir = "../../image/tmp/elaticity_uh_u/PDE1/u_lam={}.png".format(Lam[i])
class PDE():
    def __init__(self, mu=1, lam=1):
        self.mu  = mu
```

```
        self.lam = lam
        self.node = np.array([
            (0,0),
            (1,0),
            (1,1),
            (0,1)], dtype=np.float64)
        self.cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)

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

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

        frac_u1_x   = y * (y-1) * (2*cos(x) - (x-1) * sin(x))
        frac_u1_y   = 2 * (x-1) * sin(x)
        frac_u1_x_y = (2*y-1) * (sin(x) + (x-1) * cos(x))
        frac_u2_x   = 2 * (y-1) * sin(y)
        frac_u2_y   = x * (x-1) * (2*cos(y) - (y-1) * sin(y))
        frac_u2_x_y = (2*x-1) * (sin(y) + (y-1) * cos(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):
        x = p[..., 0]
        y = p[..., 1]

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

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

        return val
```

```

# PDE2
# u1 = u2 = x^2 * sin(x-1) * y^2 * sin(y-1)
# uh_dir = "../../image/tmp/elaticity_uh_u/PDE2/uh_lam={}.png".format(Lam[i])
# u_dir = "../../image/tmp/elaticity_uh_u/PDE2/u_lam={}.png".format(Lam[i])
class PDE2():
    def __init__(self, mu=1, lam=1):
        self.mu  = mu
        self.lam = lam
        self.node = np.array([
            (0,0),
            (1,0),
            (1,1),
            (0,1)], dtype=np.float64)
        self.cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)

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

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

        ux         = x**2 * sin(x-1)
        uy         = y**2 * sin(y-1)
        frac_ux_x  = 2 * x * sin(x-1) + x**2 * cos(x-1)
        frac_ux_xx = 2 * sin(x-1) + 2 * x *cos(x-1) + 2 * x * cos(x-1) - x**2 * sin(x-1)
        frac_uy_y  = 2 * y * sin(y-1) + y**2 * cos(y-1)
        frac_uy_yy = 2 * sin(y-1) + 2 * y *cos(y-1) + 2 * y * cos(y-1) - y**2 * sin(y-1)

        frac_u1_x   = frac_ux_xx * uy
        frac_u1_y   = ux * frac_uy_yy
        frac_u1_x_y = frac_ux_x * frac_uy_y
        frac_u2_x   = frac_ux_xx * uy
        frac_u2_y   = ux * frac_uy_yy
```

```
        frac_u2_x_y = frac_ux_x * frac_uy_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):
        x = p[..., 0]
        y = p[..., 1]

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

        ux = x**2 * sin(x-1)
        uy = y**2 * sin(y-1)

        val[..., 0] = ux * uy
        val[..., 1] = ux * uy

        return val

# PDE3
# u1 = u2 = -(x-1) * (e^x-1) * (y-1) * (e^y-1)
# uh_dir = "../../image/tmp/elaticity_uh_u/PDE3/uh_lam={}.png".format(Lam[i])
# u_dir = "../../image/tmp/elaticity_uh_u/PDE3/u_lam={}.png".format(Lam[i])
class PDE3():
    def __init__(self, mu=1, lam=1):
        self.mu  = mu
        self.lam = lam
        self.node = np.array([
            (0,0),
            (1,0),
            (1,1),
            (0,1)], dtype=np.float64)
        self.cell = np.array([(1,2,0), (3,0,2)], dtype=np.int64)

    def source(self, p):
```

```python
        x   = p[..., 0]
        y   = p[..., 1]
        mu  = self.mu
        lam = self.lam

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

        ux = (x-1) * (exp(x) - 1)
        uy = (y-1) * (exp(y) - 1)
        frac_ux_x = x * exp(x) - 1
        frac_ux_xx = exp(x) * (x+1)
        frac_uy_y = y * exp(y) - 1
        frac_uy_yy = exp(y) * (y+1)

        frac_u1_x   = frac_ux_xx * uy
        frac_u1_y   = ux * frac_uy_yy
        frac_u1_x_y = frac_ux_x * frac_uy_y
        frac_u2_x   = frac_ux_xx * uy
        frac_u2_y   = ux * frac_uy_yy
        frac_u2_x_y = frac_ux_x * frac_uy_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):
        x = p[..., 0]
        y = p[..., 1]

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

        ux = (x-1) * (np.exp(x) - 1)
        uy = (y-1) * (np.exp(y) - 1)

        val[..., 0] = ux * uy
        val[..., 1] = ux * uy
```

```
        return -val

# interfaceData1
# u1 = u2 = x(x-0.5)(x-1) * y(y-0.5)(y-1)
# uh_dir = "../../image/tmp/interface_uh_u/PDE1/uh_lam={}.png".format(
    Lam[i])
# u_dir = "../../image/tmp/interface_uh_u/PDE1/u_lam={}.png".format(Lam[
    i])
class interfaceData():
    def __init__(self, mu=np.array([1,2,3,4]), lam=np.array([1,2,3,4])):
        self.mu = mu
        self.lam = lam
        self.node = np.array([(0,0),
                        (0.5,0),
                        (1,0),
                        (0,0.5),
                        (0.5,0.5),
                        (1,0.5),
                        (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)

        crCell = getWhichCell(cr_node)
        for i in  range(4):
            val[crCell[i], :] /= self.lam[i]

        return val

def H1Error(u, uh):
    tmp = u - uh
    e = np.einsum("ni, ni -> n", tmp, tmp)
    sum = e. sum()
    return np.sqrt( sum)

def print_error(Lam, H, E):
    for i in  range( len(Lam)):
        print("---------------------Lam= {}---------------------".format(Lam[i]))
        n = H.shape[0]
        print()
        for j in  range(n):
            print("h= ", H[j])
            print("e=", E[i][j])
            print()
        print()
```

```python

def print_P(Lam, P):
    print("--------------------误差阶--------------------")
    for i in range(len(Lam)):
        print("lam= ", Lam[i])
        print("p= ", P[i])
        print()

#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
def is_in_cr_node(p, cr_node):
    #p 不会为[0,0]
    index = np.where((cr_node == p).all(axis=1))[0]
    if len(index):
        return index[0]
    else:
        in_index = np.where((cr_node == np.array([0,0])).all(axis=1))[0]
        if len(in_index) == 0:
            print("cr_node= ", cr_node)
            raise Exception("数组cr_node已满")
        cr_node[in_index[0]] = p
        return in_index[0]

#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
def is_in_node(p, node):
    #p 不会为[0,0]
    index = np.where((node == p).all(axis=1))[0]
    if len(index):
        return index[0]
    else:
        in_index = np.where((node == np.array([0,0])).all(axis=1))[0]
        if len(in_index) == 1:
            print("node= ", node)
            raise Exception("数组node已满")
        node[in_index[1]] = p
        return in_index[1]

# a_cell 是否属于 cell，是则返回下标， 否则返回 -1
def is_in_cell(a_cell, cell):
    i = np.where((cell == a_cell).all(axis=1))[0]
    if len(i):
```

```python
        return i[0]
    else:
        return -1

#将 a_cell（维数[3]）放入new_cr_cell
def push_cr_cell(a_cell, new_cr_cell):
    in_index = np.where((new_cr_cell == np.array([0,0,0])).all(axis=1))[0]
    if  len(in_index) == 0:
        raise Exception("数组cr_cell已满")
    new_cr_cell[in_index[0]] = a_cell
    return new_cr_cell


# 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
def a_creat(a_cell_node, new_cr_node, new_cr_cell):
    p1 = (a_cell_node[0] + a_cell_node[1]) / 2
    p2 = (a_cell_node[0] + a_cell_node[2]) / 2
    p3 = (a_cell_node[1] + a_cell_node[2]) / 2

    p1_i = is_in_cr_node(p1, new_cr_node)
    p2_i = is_in_cr_node(p2, new_cr_node)
    p3_i = is_in_cr_node(p3, new_cr_node)

    push_cr_cell([p1_i, p2_i, p3_i], new_cr_cell)

    return new_cr_node, new_cr_cell

def refine_a_cell(a_cell, new_node, new_cell):
    p1 = (new_node[a_cell][0] + new_node[a_cell][1]) / 2
    p2 = (new_node[a_cell][0] + new_node[a_cell][2]) / 2
    p3 = (new_node[a_cell][1] + new_node[a_cell][2]) / 2

    p1_i = is_in_node(p1, new_node)
    p2_i = is_in_node(p2, new_node)
    p3_i = is_in_node(p3, new_node)

    push_cr_cell([p1_i, p2_i, p3_i], new_cell)
    push_cr_cell([a_cell[0], p1_i, p2_i], new_cell)
    push_cr_cell([p1_i, a_cell[1], p3_i], new_cell)
    push_cr_cell([p2_i, p3_i, a_cell[2]], new_cell)

```

```python
    return new_node, new_cell

# 从剖分node, cell得到 cr_node, cr_cell
# 单元数 NC
# 剖分次数 n : log_4(NC / 2)
# 外边 out_edge : 4 * 2**n
# 总边 all_edge :  3 * NC - (3 * NC - out_edge) / 2
def get_cr_node_cell(node, cell):
    NC = cell.shape[0]
    # n 特定情况下剖分次数
    n  = math.log(NC/2, 4)
    NN =  int(3 * 2 * 4**n - (3 * 2 * 4**n - 4 * 2**n) / 2)
    cr_node = np.zeros((NN, 2), dtype=np.float64)
    cr_cell = np.zeros_like(cell)

    for i in  range(NC):
        cr_node, cr_cell = a_creat(node[cell[i]], cr_node, cr_cell)

    return cr_node, cr_cell

# 返回 node 中是否为边界点的信息
# isBdNode [NN] bool
def getIsBdNode(cr_node):
    is_BdNode = np.zeros(cr_node.shape[0], dtype= bool)
    for i in  range(cr_node.shape[0]):
        a = np. min(np. abs(cr_node[i] - np.array([0,0])))
        b = np. min(np. abs(cr_node[i] - np.array([1,1])))
        if a < 1e-13 or b < 1e-13:
            is_BdNode[i] = True
    return is_BdNode

def getIsBdLineNode(cr_node):
    NN = cr_node.shape[0]
    isBdLineNode = getIsBdNode(cr_node)
    for i in  range(NN):
        a = cr_node[i,0]
        b = cr_node[i,1]
        if a == 0.5 or b == 0.5:
            isBdLineNode[i] = True
    return isBdLineNode

def uniform_refine(node, cell):
```

```python
    old_NN = node.shape[0]
    old_NC = cell.shape[0]
    n  = math.log(old_NC/2, 4)
    NC = 4 * old_NC
    num_edge =  int(3 * 2 * 4**n - (3 * 2 * 4**n - 4 * 2**n) / 2)
    NN = old_NN + num_edge

    new_node = np.zeros((NN, 2), dtype=np.float64)
    new_cell = np.zeros((NC, 3), dtype=np.int64)
    new_node[:old_NN] = node

    for i in  range(old_NC):
        new_node, new_cell = refine_a_cell(cell[i], new_node, new_cell)

    return new_node, new_cell

def get_cr_glam_and_pre(cr_node, cr_cell):
    NC = cr_cell.shape[0]
    NN = cr_node.shape[0]
    cr_node_cell = cr_node[cr_cell]
    ##求解CR元导数
    cr_node_cell_A = np.ones((NC, 3, 3), dtype=np.float64)
    #求解CR元导数的系数矩阵
    cr_node_cell_A[:, :, 0:2] = cr_node_cell
    #用于求解CR元的值
    # cr_glam_x_y_pre [NC, 3, 3]
    cr_glam_x_y_pre = np.zeros((NC, 3, 3), dtype=np.float64)
    for k in  range(NC):
        cr_glam_x_y_pre[k, :, :] = solve(cr_node_cell_A[k, :, :], np.diag(np.ones(3)))
    #[NC,3,3]
    cr_glam_x_y = np.copy(cr_glam_x_y_pre)
    cr_glam_x_y = cr_glam_x_y[:, 0:2, :]
    cr_glam_x_y = cr_glam_x_y.transpose((0,2,1))
    return cr_glam_x_y, cr_glam_x_y_pre

# phi_val [NC,3(点),6(6个基函数),2(两个分量)]
def get_phi_val(node, cell, cr_glam_pre):
    NC = cell.shape[0]
    # cr_node_val [NC,3(点),3(三个cr元的值)] CR元在各顶点的值
    node_cell_A = np.ones((NC,3,3), dtype=np.float64)
    node_cell_A[:,:,0:2] = node[cell]
```

```
    cr_node_val = np.einsum("cij, cjk -> cik", node_cell_A, cr_glam_pre)

    # phi_node_val [NC,3(点),6(6个基函数),2(两个分量)]
    phi_node_val = np.zeros((NC,3,6,2), dtype=np.float64)
    phi_node_val[:,:,0:5:2,0] = cr_node_val
    phi_node_val[:,:,1:6:2,1] = cr_node_val
    return phi_node_val

def get_phi_grad_and_div(cr_node, cr_cell):
    NC = cr_cell.shape[0]
    cr_glam_x_y, cr_glam_x_y_pre = get_cr_glam_and_pre(cr_node, cr_cell)
    #求 cr_phi_grad [NC,6(基函数),2(分量 x , y),2(导数)]
    cr_phi_grad = np.zeros((NC,6,2,2), dtype=np.float64)
    cr_phi_grad[:, 0:5:2, 0, :] = cr_glam_x_y
    cr_phi_grad[:, 1:6:2, 1, :] = cr_glam_x_y

    # cr_phi_div [NC, 6]
    #cr_phi_div = np.einsum("cmij -> cm", cr_phi_grad)
    cr_phi_div = cr_glam_x_y.copy()
    cr_phi_div = cr_phi_div.reshape(NC, 6)
    return cr_phi_grad, cr_phi_div

## 单元刚度矩阵, 单元质量矩阵 stiff, div [NC, 6, 6]
def get_stiff_and_div_matrix(cr_node, cr_cell, cm):
    cr_phi_grad, cr_phi_div = get_phi_grad_and_div(cr_node, cr_cell)

    ## 单元刚度矩阵
    # A1 A2 [NC, 6, 6]
    A1 = np.einsum("cnij, cmij, c -> cnm", cr_phi_grad, cr_phi_grad, cm)
    A2 = np.einsum("cn, cm, c -> cnm", cr_phi_div, cr_phi_div, cm)
    return A1, A2

## 单元载荷向量 bb [NC, 6]
def get_bb(pde, node, cell, cm):
    cr_node, cr_cell = get_cr_node_cell(node,cell)
    cr_glam_x_y, cr_glam_x_y_pre = get_cr_glam_and_pre(cr_node, cr_cell)
    # phi_val [NC,3(点),6(6个基函数),2(两个分量)]
    phi_node_val = get_phi_val(node, cell, cr_glam_x_y_pre)

    # val [NC,3(点),2(分量)] 右端项在各顶点的值
    val = pde.source(node[cell])

```

```python
    # phi_val [NC,3,6] 基函数和右端项的点乘
    phi_val = np.einsum("cijk, cik -> cij", phi_node_val, val)
    # bb [NC,6]
    bb = phi_val. sum(axis=1) * cm[0] / 3
    return bb

#input
#单元刚度矩阵 A1, A2 [NC, 6, 6], bb [NC,6]
# output
# 总刚度矩阵 A1, A2 [2*NN,2*NN], F [2*NN]
def get_A1_A2_F(A1, A2, bb, cr_node, cr_cell):
    NN = cr_node.shape[0]
    NC = cr_cell.shape[0]
    # cell_x_y [NC, 3(三个点), 2(x y 方向上基函数的编号)]
    cell_x_y = np.broadcast_to(cr_cell[:,:,None], shape=(NC, 3, 2)).copy()
    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
    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
        if a < 0.5 and b >= 0.5:
            isWhichCellNode[2,i] = True
        if a >= 0.5 and b > 0.5:
            isWhichCellNode[3,i] = True
    return isWhichCellNode

## [4, NC] 返回各单元属于哪个区间
def getCellInOmega(node, cell):
    cellInOmega = np.zeros(cell.shape[0], dtype= bool)
    #print("node[cell]= ", node[cell])
    mid_p = node[cell]. sum(axis=1) / 3
    #print("mid_p= ", mid_p)
    cellInOmega = getWhichCell(mid_p)
    return cellInOmega

## [4,NN]
## \line_1 x=0.5, y<0.5
## \line_2 x=0.5, y>0.5
## \line_3 x<0.5, y=0.5
## \line_4 x>0.5, y=0.5
def getInterfaceCell(node):
    interfaceCell = np.zeros((4,node.shape[0]), dtype= bool)
    for i in  range(node.shape[0]):
```

```python
        a = node[i,0]
        b = node[i,1]
        if a == 0.5 and b < 0.5:
            interfaceCell[0,i] = True
        if a == 0.5 and b > 0.5:
            interfaceCell[1,i] = True
        if a < 0.5 and b == 0.5:
            interfaceCell[2,i] = True
        if a > 0.5 and b == 0.5:
            interfaceCell[3,i] = True
    return interfaceCell

def getInterLineNode(node):
    NN = node.shape[0]
    lineNode = np.zeros(NN, dtype= bool)
    for i in  range(NN):
        a = node[i,0]
        b = node[i,1]
        if a == 0.5 or b == 0.5:
            lineNode[i] = True
    return lineNode

def phiInWhichCell(whichCell):
    phiCell = np.broadcast_to(whichCell[:, :, None], shape=(4, whichCell
        .shape[1], 2))
    phiCell = phiCell.reshape(4, 2 * whichCell.shape[1])
    return phiCell

# uh_dir = "../../image/tmp/elaticity_uh_u/uh_lam={}.png".format(Lam[i])
    "
# u_dir = "../../image/tmp/elaticity_uh_u/u_lam={}.png".format(Lam[i])
def drawer_uh_u(cr_node, uh, u, uh_dir, u_dir):
    import matplotlib.pyplot as plt
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(111, projection='3d')

    x = cr_node[:,0]
    y = cr_node[:,1]

    ax.plot_trisurf(x, y, uh[:,0], cmap='rainbow')
    plt.savefig(fname=uh_dir)

```

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